



DIPARTIMENTO
DI INFORMATICA
SAPIENZA
UNIVERSITÀ DI ROMA

RACER

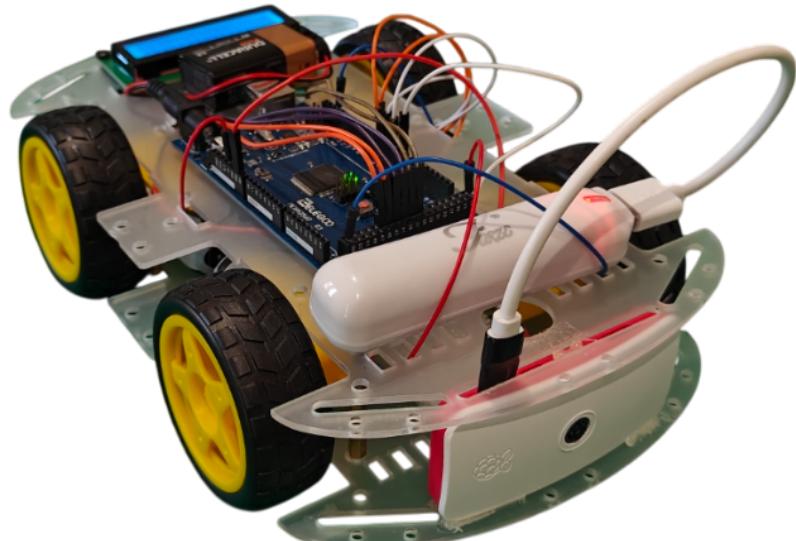
Raspberry & Arduino Car for Environmental Recognition

Progetto per il corso di AiLab Computer Vision - A.A. 2024/2025

Docente: Daniele Pannone

1 luglio 2025

Cola Valerio cola.2060062@studenti.uniroma1.it
Cerboni Federico cerboni.2006100@studenti.uniroma1.it



Indice

1	Introduzione	3
2	Progettazione del Veicolo	3
2.1	Arduino - Il cuore	3
2.2	Raspberry Pi Z2W + PiCamera V2 - Gli occhi della macchina	4
2.3	Comunicazione tra i moduli	4
2.4	Alimentazione	5
3	Sviluppo Software	5
3.1	Introduzione	5
3.2	YoloV5s - Il Cervello	5
3.2.1	Cosa è YoloV5	5
3.2.2	Architettura	5
3.2.3	Perchè lo abbiamo scelto	6
3.2.4	Training	7
3.2.5	Object Detection	9
3.3	Lane Detection	11
4	Sviluppi futuri del progetto	12

1 Introduzione

Il progetto RACER si incentra sullo sviluppo da zero di una macchina a guida autonoma grazie alla combinazione del Microcontrollore [Ele] Arduino Mega 2560 R3 (prodotto dalla Elegoo), il Micro-computer [Rasb] Raspberry Pi Zero 2W e il Modello Convoluzionale [ulta] YoloV5s. Sebbene non si tratti di una classica RC-Car, l'idea principale è sostituire il controllo umano e il Radiocomando con un sistema basato su una rete neurale e sulla computer vision tramite OpenCV. Questo sistema è in grado di osservare l'ambiente, rilevare enti e segnali stradali e analizzare le corsie per correggere automaticamente la direzione del veicolo.

Un primo approccio ha portato il team ad utilizzare un ESP32-S3-N16R8 in combinazione con una camera OV5560. Purtroppo la scarsa potenza di calcolo e la qualità costruttiva ci hanno convinti a sostituirlo con un Raspberry: l'antenna Wi-Fi era troppo debole, i contatti della camera talmente sensibili da compromettere il funzionamento al minimo movimento, e il corpo macchina della OV5560 si surriscaldava rapidamente, peggiorando ulteriormente le prestazioni.

Il progetto completo è disponibile su GitHub al seguente [link](#).

2 Progettazione del Veicolo

2.1 Arduino - Il cuore

Alla base di ogni autovettura c'è sempre un motore, per RACER sono state utilizzati 4 piccoli motori DC a bassa tensione (9v) che possono essere controllati con i pin PWM di Arduino. In particolare la struttura è 4WD (Four Wheel Drive), ovvero a trazione integrale dove ogni ruota è motrice e può essere impostata autonomamente.

A disposizione avevamo un solo L293D (dual H-Bridge motor driver) per il controllo di direzione e velocità, sono quindi stati collegati in serie i 2 motori di destra e di sinistra. Ciò semplifica anche la gestione della svolta poiché per sterzare è necessario che i due motori di ambi i lati abbiano velocità equivalenti. I comandi che verranno gestiti sono Start, Stop, Sterzata a destra e sinistra e variazione di velocità a due livelli. È stato inserito anche uno schermo LCD 16x2 che funge da debugger per comprendere in tempo reale cosa sta facendo l'Arduino.

Il "telaio" è in resina con ruote in gomma e plastica.

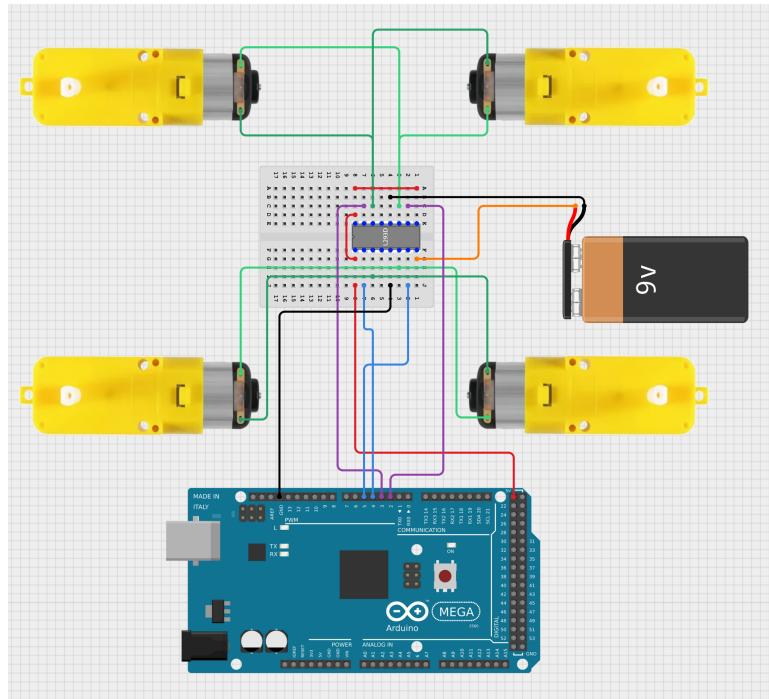


Figura 1: Circuito completo dei motori.

2.2 Raspberry Pi Z2W + PiCamera V2 - Gli occhi della macchina

Per catturare i dati ambientali abbiamo utilizzato un Raspberry Pi Z2W, dotato di processore Quad-core Cortex-A53 a 1.0 GHz a 64bit e 512MB LPDDR2 di RAM, in combinazione con una [Rasa] PiCamera V2 che monta un sensore Sony IMX219 da 8 Megapixel. Entrambi i componenti sono stati montati in un case apposito della Raspberry mostrato nella figura in prima pagina e montato nella parte anteriore del veicolo.

Questo modulo si occuperà di:

- Catturare un flusso video 720x1280 e streammarlo su una rete locale (Hotspot Cellulare). Lo streaming sarà in contemporanea catturato mediante OpenCV su un PC che sta eseguendo il programma principale di analisi ambientale.
- Gestire connessioni TCP con un terminale per lo scambio di messaggi.
- Inoltro dei comandi ad Arduino.

2.3 Comunicazione tra i moduli

1. PC collegato alla stessa rete del Raspberry cattura con OpenCV lo streaming video.
2. Con la libreria Socket invierà i comandi/messaggi utilizzando il protocollo TCP su porta 5005 al Raspberry.
3. Una volta ricevuti, il Raspberry li inoltrerà all'Arduino tramite il collegamento UART0 (Universal Asynchronous Receiver-Transmitter) che permette lo scambio su porta seriale in modo asincrono. Abbiamo deciso di rendere la comunicazione unidirezionale, solo il Pi invia Pin TX Pi - Pin Rx Arduino.

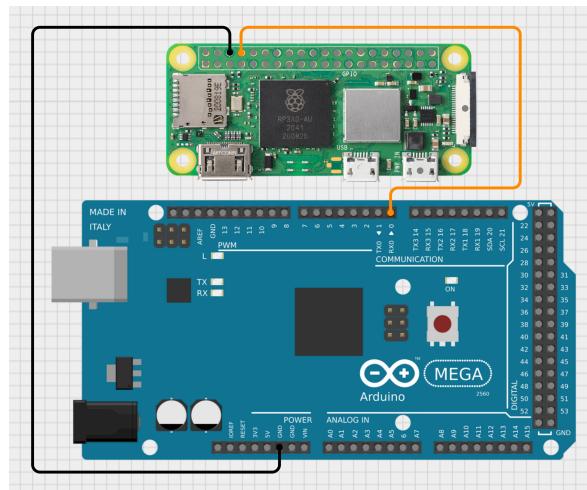


Figura 2: Collegamento UART

4. [Gri] Streaming Video Raspberry

Abbiamo creato un'app Web Camera Server utilizzando il framework Flask e le librerie PiCamera2 e OpenCV per catturare i frame dalla PiCamera. L'app si occupa di inviare il flusso di frame catturato dalla camera al dispositivo connesso su porta 8080. Successivamente abbiamo reso lo script un servizio linux, in modo che venga eseguito in automatico all'accensione. Per la realizzazione dell'app ci siamo affidati alla seunete repository.

2.4 Alimentazione

- Arduino: Batteria 9v.
- Motori: Batteria 9v.
- Raspberry: Powerbank 5V/1A.

3 Sviluppo Software

3.1 Introduzione

Il programma di elaborazione dei frame sfrutta il [Gee] multithreading per ottenere prestazioni ottimali e fluide e si suddivide come segue:

- Thread Principale: Esegue la Lane Detection e invia i comandi relativi a direzione e ciò che Yolo ha individuato solo se il Thread1 ha acquisito il frame.
- Thread1: Si occupa di acquisire i frame.
- Thread2: Esegue l'inferenza di Yolo e salva le classi rilevate.

Di seguito è rappresentato il flusso di computazione semplificato: *Thread1 Frame Acquisito → Thread2 Oggetti individuati nel frame → Thread Principale Lane Detection + Invio Comandi*

3.2 YoloV5s - Il Cervello

3.2.1 Cosa è YoloV5

[Red+16] YOLO (You Only Look Once) è una serie di modelli di riconoscimento degli oggetti in tempo reale basato sulla rete neurale convoluzionale. Il suo nome fa riferimento alla caratteristica che lo contraddistingue da modelli simili precedenti, ovvero che il modello richiede un solo passaggio di propagazione in avanti per poter fare delle previsioni. Il modello è stato creato da Joseph Redmon, che ha sviluppato le prime tre versioni e pubblicate sul suo sito web. Successive versioni sono state sviluppate da altri enti, la versione utilizzata da noi (v5) ad esempio è stata sviluppata dalla Ultralytics.

3.2.2 Architettura

[Ultb] [Kun] Modelli di riconoscimento di oggetti classici richiedono due passaggi dell'immagine in input, la prima per generare un insieme di potenziali posizioni di oggetti (proposte), la seconda serve per raffinare tali proposte e ottenere le previsioni finali. Alcuni modelli, come il Region Proposal Networks, eseguono diverse istanze del modello per singola immagine, una per ogni regione separatamente. Tutto ciò richiede una quantità di risorse considerevole. Il modello YOLO esegue un singolo passaggio dell'immagine in input sia per le proposte e sia per la previsione finale, inoltre esegue le previsioni sull'immagine in input nella sua interezza nello stesso momento, grazie all'uso di un singolo layer completamente connesso. Infine, la struttura stessa di YOLO è relativamente semplice: una singola rete convoluzionale esegue le previsioni sia delle classi sia dei bounding box. Questo è possibile grazie ad un approccio differente ed innovativo rispetto ad algoritmi precedenti, che sono essenzialmente dei modelli di classificazione modificati per la rilevazione di oggetti. La rete convoluzionale è suddivisa in tre parti:

- Backbone: che è responsabile dell'estrazione delle caratteristiche dall'immagine di input.
- Neck: che esegue trasformazioni sulle caratteristiche estratte dalla Backbone
- Head: che si occupa delle previsioni finali. Versioni successive di YOLO sono aggiornamenti e modificazioni di questi tre moduli.

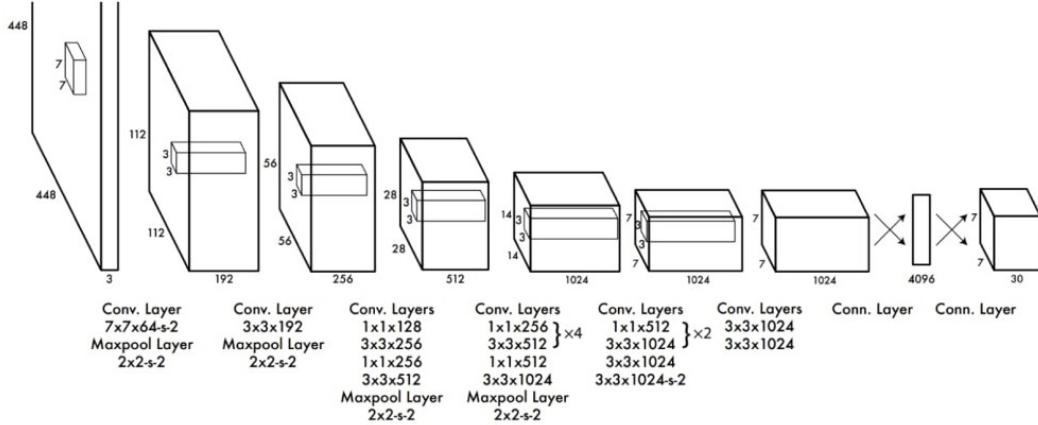


Figura 3: Architettura YoloV5

YOLO divide l'immagine in input in una griglia SxS. Se il centro di un oggetto ricade in una cella, è quella cella che si incarica di rilevarlo. Ogni cella prevede B bounding box e un punteggio di confidenza per ognuno di quei box. La confidenza quantifica la probabilità di quel box di contenere un oggetto e con quale precisione. Inoltre ad ogni cella prevede C probabilità di classe, ovvero che tipo di oggetto si trova dentro la cella. Queste due informazioni vengono unite per produrre le previsioni finali.

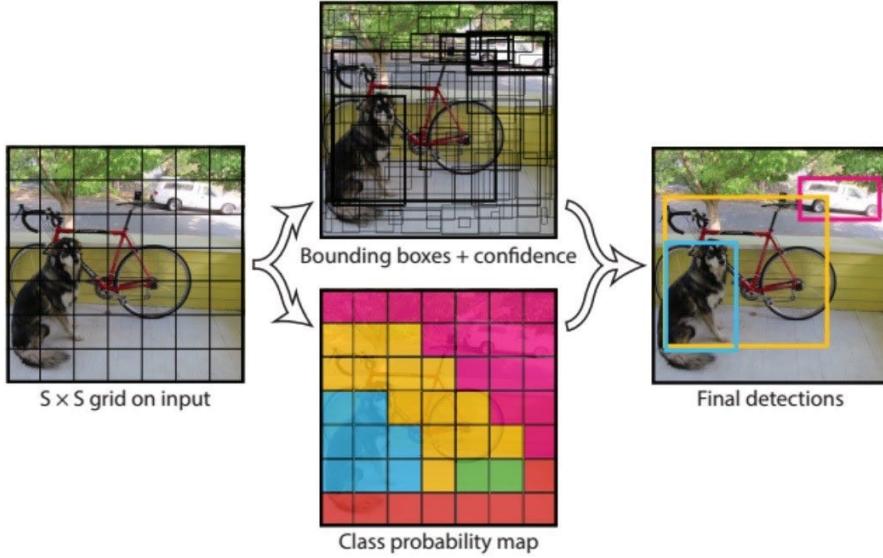


Figura 4: Tiling Input

3.2.3 Perchè lo abbiamo scelto

Grazie a questo, YOLO è estremamente veloce, in grado di elaborare un flusso di immagini in tempo reale con bassissima latenza, pur mantenendo un alta precisione, e ciò è proprio il nostro caso di utilizzo: l'automobile deve riconoscere rapidamente i vari ostacoli e situazioni che si presentano lungo la strada, ed attuare strategie di correzione tempestivamente.

3.2.4 Training

Il modello è stato addestrato su Google Colab. Abbiamo utilizzato un dataset creato sulla piattaforma Roboflow [Rob], composto da circa 3000 immagini con risoluzione 640x640, ed è stato aumentato ruotando immagini e aggiungendo rumore.

Le classi con le relative annotazioni sono le seguenti:

- Pedoni: 1173
- Segnale di Stop: 261
- Segnale di limite a 50 km/h: 254
- Segnale di limite a 20 km/h: 423
- Semaforo rosso: 149
- Semaforo verde: 167

Iperparametri:

- 150 epoche
- Dimensione batch 64
- Learning rate 0.01

Questi iperparametri si sono rivelati ottimali per massimizzare le prestazioni del modello senza incorrere in overfitting. Ciò è stato compreso andando ad effettuare diverse sessioni di training con configurazioni differenti. Inoltre, abbiamo integrato il dataset con un maggior numero di immagini di segnali da 20 km/h, dopo aver riscontrato che il modello faticava a distinguerli da quelli da 50 km/h. I dati sono riferiti all'ultima istanza di allenamento del modello.

Durante la fase di training viene generato un file CSV che contiene le metriche delle performance della rete neurale nella fase di validazione per ogni epoca del training. Nel caso di questo modello, e in generale per i modelli di object detection, ci sono diversi valori da osservare per comprendere il comportamento e l'efficacia della rete neurale. Precision P: Rappresenta il numero di previsioni corrette rispetto al numero di previsioni effettuate. Recall R: Rappresenta il numero di previsioni effettuate dal modello rispetto al numero effettivo di previsioni corrette presenti nell'immagine. Mean Average Precision mAP: Quantifica la performance generale del modello in rapporto tra i valori di P e R. Tutti i valori sono rappresentati come numeri decimali da 0 a 1.

Intersection over Union (IoU): Rispetto alla classificazione, bisogna tenere conto anche di un'altra capacità dei modelli di riconoscimento oggetti: quella di riconoscere la posizione di tali oggetti e delimitarli con una bounding box. Per valutare con precisione le prestazioni del modello si utilizza la metrica Intersection over Union (IoU), ovvero il rapporto tra l'intersezione dell'area della bounding box predetta rispetto all'area di quella effettiva, e l'unione di queste due aree. Più questo rapporto è vicino ad 1, più il bounding box predetto combacia con il bounding box corretto del dataset.

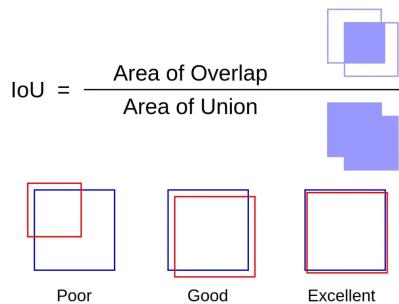


Figura 5: Intersection over Union

Nel Mean Average Precision (mAP), viene impostato un threshold in cui una previsione con valore IoU superiore ad esso viene considerata corretta, e viene fatta una media di quante previsioni corrette esistono per tutte le immagini del dataset, per tutte le classi. Nel nostro caso abbiamo due metriche di mAP: una con threshold impostato a 0.5, e l'altra che rappresenta una media tra i valori di Average Precision con threshold a 0.5 e a 0.95, in modo da avere una metrica più severa. mAP@0.5 mostra un valore del 94,2%, indicando un'ottima precisione del modello. Normalmente il valore scende al 65% per mAP@0.5:0.95, e si tratta comunque di un valore alto considerando che il metro di giudizio di una previsione corretta è più stringente.

Un valore alto di P significa che la maggior parte delle previsioni del modello sono corrette, mentre un valore basso significa un numero alto di falsi positivi. Un alto valore di R significa che il numero di previsioni corrette effettuate dal modello corrispondono al numero effettivo di positivi del dataset, un valore basso invece indica un alto numero di falsi negativi. Nel nostro caso, entrambi i valori superano il 90% (P=93,1%, R=90,9%) mostrando che il modello si comporta bene nel riconoscere gli oggetti.

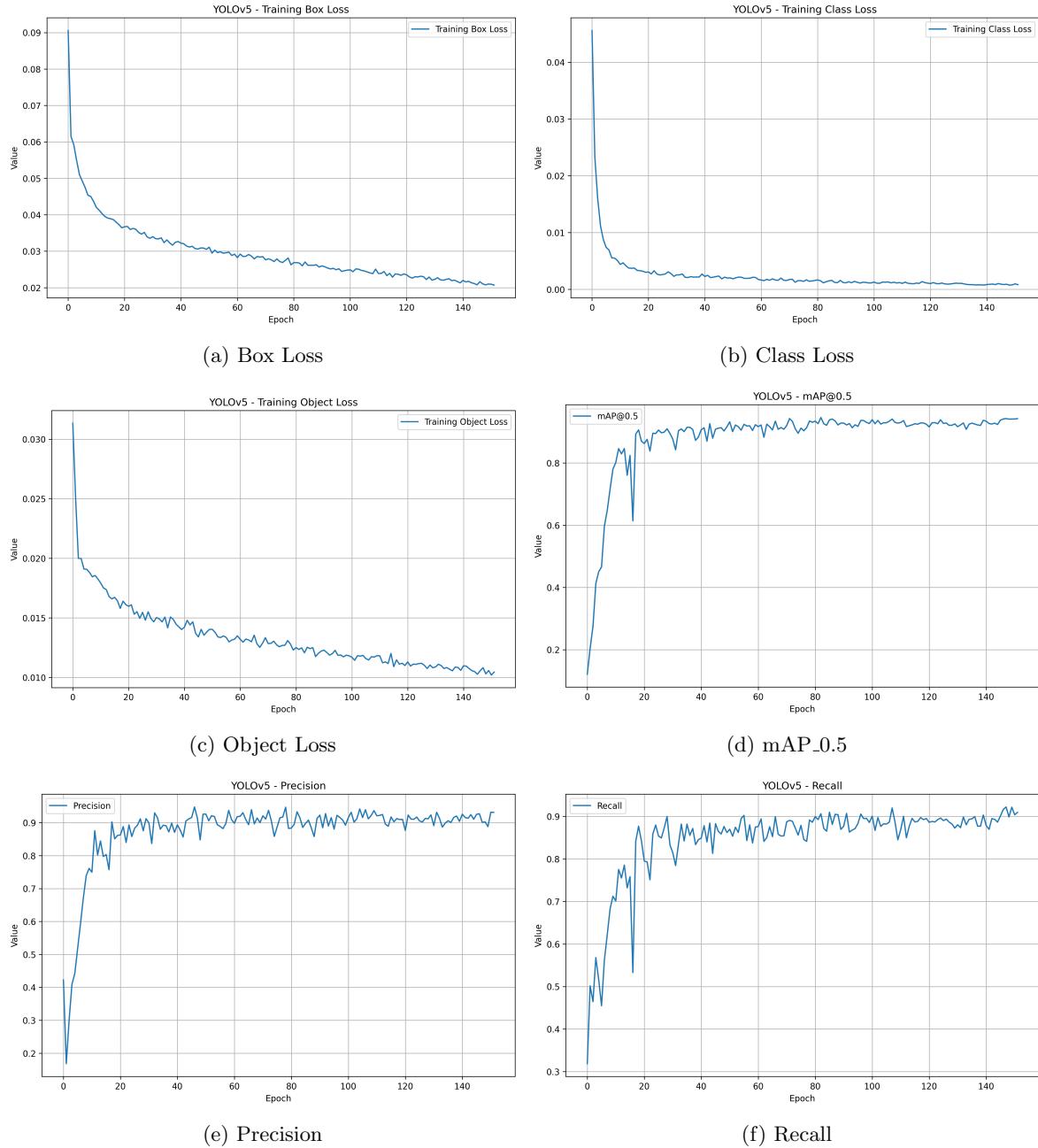


Figura 6: Andamento delle metriche durante l'addestramento del modello: perdite e performance.

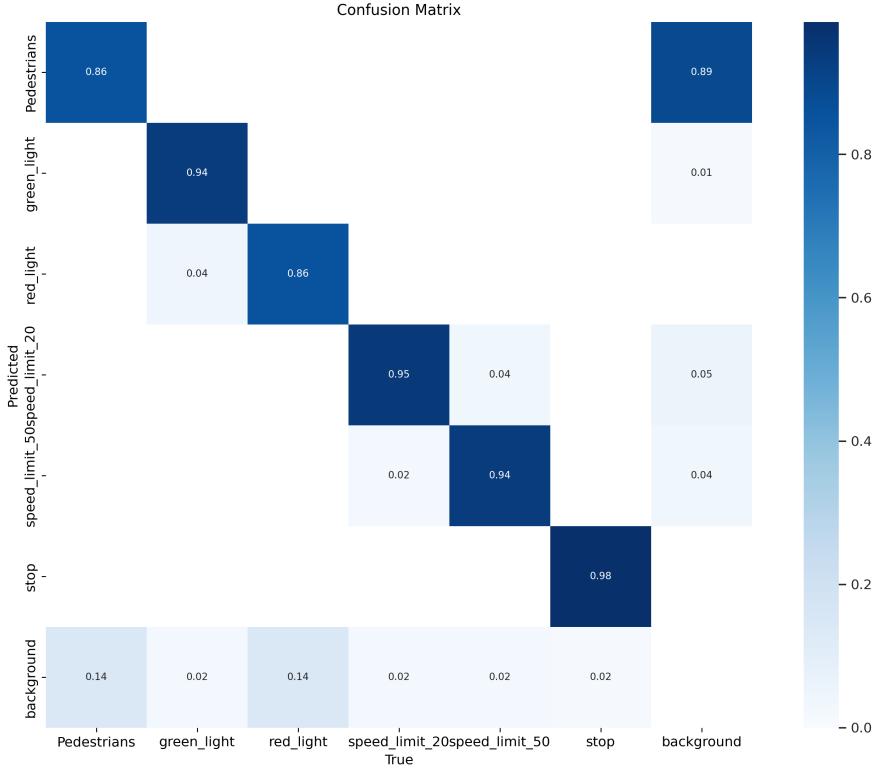


Figura 7: Confusion Matrix.

3.2.5 Object Detection

Come descritto in precedenza, il modello identifica un elemento stradale per volta. In particolare per evitare confusioni con falsi positivi abbiamo deciso di salvare solo l'oggetto con maggiore confidenza. In futuro puntiamo a migliorare questo aspetto, in modo da poter salvare più oggetti contemporaneamente e prendere decisioni più complesse. Come ad esempio se il veicolo si trova davanti a un semaforo verde e un pedone, il veicolo dovrà fermarsi e dare la precedenza al pedone. Abbiamo utilizzato una formula chiamata [Ros] Triangle Similarity per calcolare una stima della distanza tra il veicolo e l'oggetto individuato. Questa formula si basa sulla dimensione originale dell'oggetto in cm, la dimensione del bounding box in pixel e una distanza nota tra il veicolo e l'oggetto. In questo modo è possibile calcolare la distanza focale (La distanza tra il centro della lente (o l'obiettivo della fotocamera) e il punto focale, dove i raggi di luce paralleli convergono dopo essere passati attraverso la lente) e successivamente la distanza tra il veicolo e l'oggetto individuato.

$$Distanza_{focale} = \frac{\text{Larghezza}/\text{Altezza (px)} \times \text{Distanza nota (cm)}}{\text{Larghezza}/\text{Altezza reale (cm)}}$$

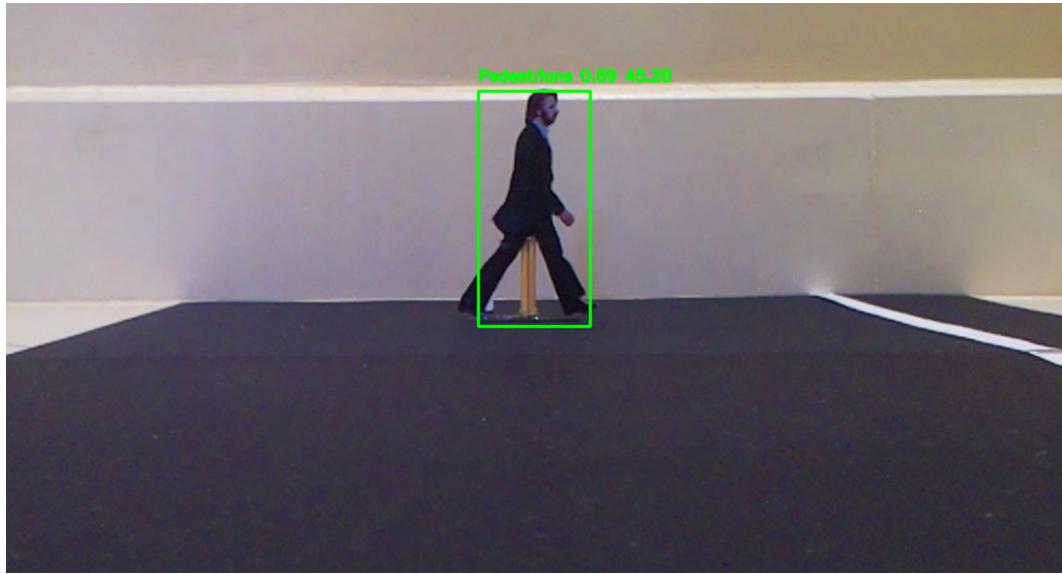
$$Distanza(cm) = \frac{\text{Larghezza}/\text{Altezza reale (cm)} \times \text{Distanza focale}}{\text{Larghezza}/\text{Altezza (px)}}$$

Grazie a questo calcolo siamo in grado di stabilire una distanza ottimale per inviare il comando. Volendo essere più precisi conoscendo FOV e la larghezza del sensore in pixel abbiamo sostituito il calcolo della distanza focale con la seguente formula [Bou]:

$$Distanza_{focale} = \frac{\text{Larghezza Sensore}}{2 \cdot \tan(\frac{\text{FOV in radianti}}{2})}$$

La lente della PiCamera ha un FOV (Field of View) di 62.2 gradi, non è quindi in grado di individuare oggetti a distanze molto ravvicinate. Un cartello al lato della strada già a circa 15 cm di distanza non è più nell'inquadratura. Per questo motivo è stato necessario inviare il comando

preventivamente ed introdurre un delay in Arduino in modo da far fermare il veicolo (o cambiare la velocità) ad una distanza ottimale e realistica. è quindi stato necessario inserire nel calcolo della distanza le dimensioni dei vari elementi della strada.



(a) Pedone



(b) Segnale Stop



(c) Segnale 20 km/h



(d) Segnale 50 km/h

Figura 8: Object Detection: esempi di classi individuate con accuratezza e stima della distanza.

Infine abbiamo notato che essendo un modello molto piccolo YoloV5s non è estremamente preciso, molto spesso per pochi frame la classe scompare per poi essere di nuovo individuata. Per questo motivo abbiamo introdotto un timer di tolleranza. Ad esempio se per un numero di frame consecutivi il modello non individua nulla o individua un'altra classe, non invierà alcun comando. Allo scadere del timer, verrà preso in considerazione ogni nuovo comando.

3.3 Lane Detection

Il controllo della direzione viene calcolato prima dell'object detection nel thread principale. Abbiamo convertito in scala di grigi il frame e tracciato una linea orizzontale. Questa riga parte dal centro e si estende a tutto il lato destro (le Figure 9, 10, 11 sono croppate), per ogni pixel andiamo a rintracciare la striscia bianca in base all'intensità. Basandosi sulla posizione rilevata verrà stabilito se si dovrà effettuare la svolta o continuare dritto. Abbiamo impostato un range di pixel (rappresentato dalla linea bianca) che indica che il veicolo è posizionato correttamente e deve quindi procedere dritto.

Nota: I comandi di direzione non sovrascrivono quelli dell'object detection. In questo modo una volta ferma allo stop l'analisi della corsia riprenderà solo se verrà dato il comando di ripartenza.

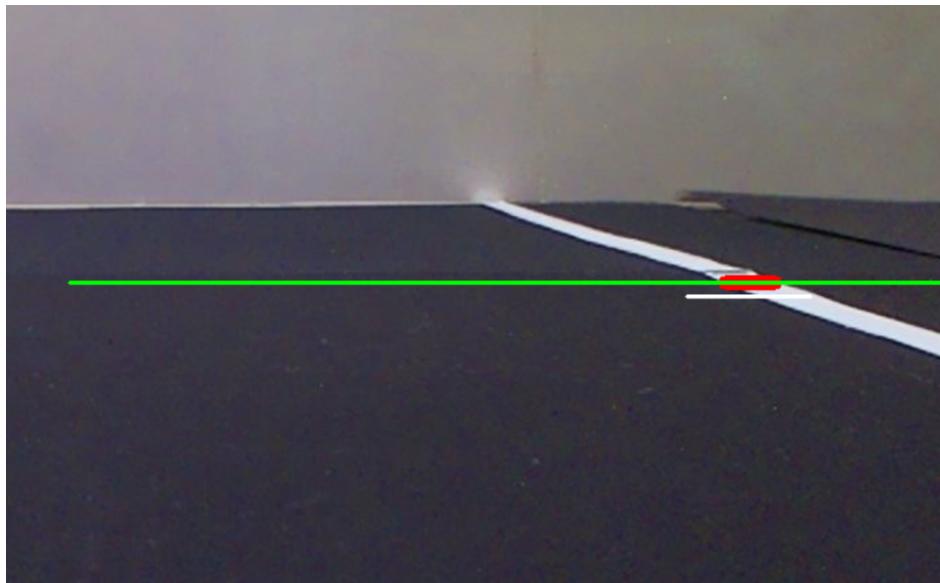


Figura 9: Veicolo Centrato, prosegue dritto.

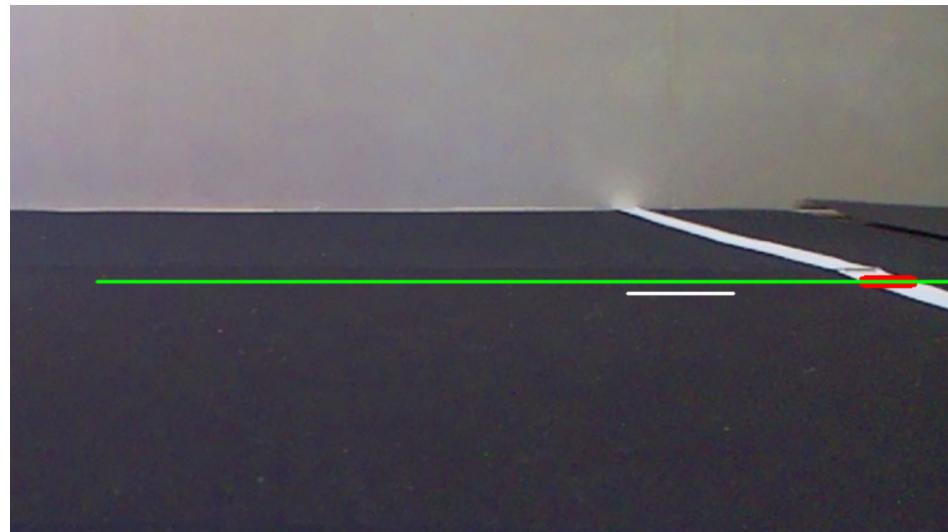


Figura 10: Svolta a Destra

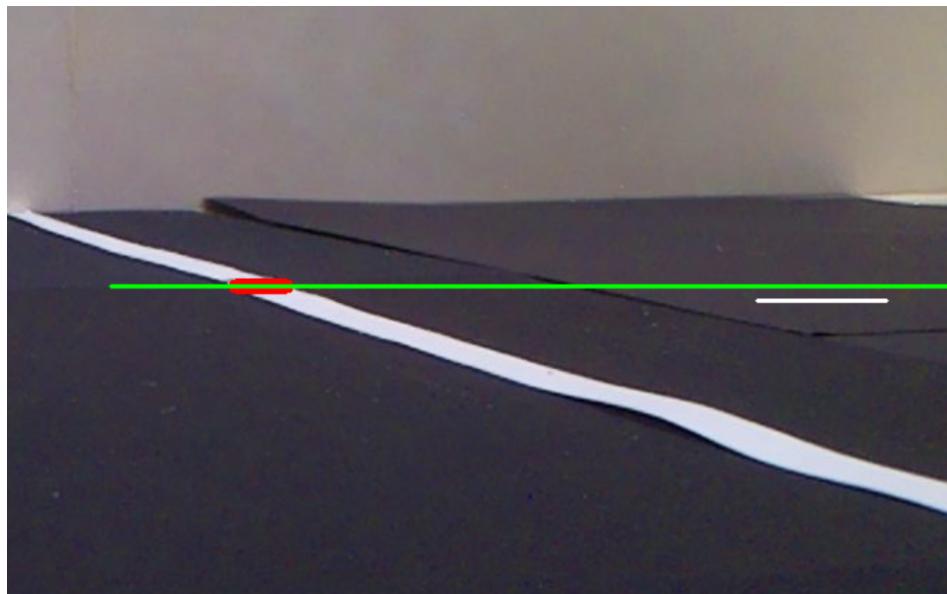


Figura 11: Svolta a Sinistra

4 Sviluppi futuri del progetto

1. Migliorare il comparto Hardware sostituendo il Pi Z2W con un Pi 4/5 in combinazione con una camera con FOV maggiore. In questo modo la macchina sarà del tutto autonoma poichè il modello girerà direttamente sul Raspberry.
2. Implementare una lane detection più complessa:
 - Prenda in considerazione entrambe le strisce di corsia.
 - Calcoli un valore di intesità di svolta.
3. Ottimizzare l'alimentazione in un'unica soluzione.

Riferimenti bibliografici

- [Red+16] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. <https://arxiv.org/pdf/1506.02640.pdf>. 2016.
- [Bou] Paul Bourke. *Field of view and focal length*. URL: <https://paulbourke.net/miscellaneous/lens/index.html>.
- [Ele] Elegoo. *Arduino Mega 2560 R3*. URL: <https://eu.elegoo.com/it-it/products/elegoo-mega-2560-r3-board?srsltid=AfmB0oqlCYHo64N17LxwVv7XPuF1ZlfI-sFE9XTx8FvvoUylTrOLYtfg>.
- [Gee] GeeksForGeeks. *Multithreading Tutorial*. URL: <https://www.geeksforgeeks.org/multithreading-python-set-1/>.
- [Gri] Miguel Grinberg. *Video Streaming with Flask*. URL: <https://blog.miguelgrinberg.com/post/video-streaming-with-flask>.
- [Kun] Rohit Kundu. *YOLO: Algorithm for Object Detection Explained [+Examples]*. URL: <https://www.v7labs.com/blog/yolo-object-detection>.
- [Rasa] Raspberry. *Raspberry Pi Camera V2*. URL: <https://www.raspberrypi.com/products/camera-module-v2/>.
- [Rasb] Raspberry. *Raspberry Pi Zero 2W*. URL: <https://www.raspberrypi.com/products/raspberry-pi-zero-2-w/>.
- [Rob] RoboFlow. *Dataset*. URL: <https://app.roboflow.com/yolorc-rrbm0/road-object-detection-v081a/8>.
- [Ros] Adrian Rosebrock. *Find distance from camera to object/marker using Python and OpenCV*. URL: <https://pyimagesearch.com/2015/01/19/find-distance-camera-objectmarker-using-python-opencv>.
- [Ulta] Ultralytics. *YoloV5*. URL: <https://github.com/ultralytics/yolov5>.
- [Ultb] Ultralytics. *YoloV5 Tutorials*. URL: https://docs.ultralytics.com/yolov5/tutorials/architecture_description/.