



4 LUGLIO 2023

RELAZIONE PROGETTO IOT

REALIZZAZIONE DI UNA VEICOLO A GUIDA AUTONOMA

CHIARCOSSI ENRICO (152017)

PIZZINATO MATTEO (152216)

RUSSOLO DAVIDE (152134)

TAM GABRIELE (153262)



Sommario

Introduzione	2
Design del robottino	3
Struttura principale	3
Cervello	3
Sensori	4
Ambiente di Test	7
Segnaletica	9
Implementazione software	11
Setup iniziale	11
Calibrazione	11
PicarX	12
Single Line Tracking	12
Collision avoidance	13
Two lines detection	14
Riconoscimento della segnaletica	19
Raccolta dati	19
Preparazione dei dati	20
Addestramento e Test del modello	20
Risultati e conclusioni	23

Introduzione

Nell'era moderna dell'intelligenza artificiale e dell'automazione, i robot autonomi stanno diventando sempre più presenti nelle nostre vite quotidiane.

L'obiettivo principale di questi automi è quello di svolgere compiti in modo indipendente, senza la necessità di un controllo umano costante. In quest'ottica, la presente relazione descrive la realizzazione di un robottino capace di guidare autonomamente, utilizzando un Raspberry Pi, sensori di profondità, un sensore di grigi e una webcam. L'intero sistema è stato sviluppato utilizzando il linguaggio di programmazione Python. Durante il processo di sviluppo, sono state implementate diverse fasi progressive per migliorare le capacità del robottino.

Inizialmente, ci si è concentrati sulla creazione di un robottino in grado di svolgere un compito molto semplice, seguire una singola linea. Il sensore di grigi è stato impiegato per mantenere il robottino allineato rispetto alla traiettoria desiderata e attraverso algoritmi di controllo appropriati, il robottino è stato in grado di seguire con successo la linea.

Successivamente, è stata implementata la funzionalità di rilevamento degli ostacoli e di arresto in caso di collisione. L'utilizzo dei sensori di profondità ha permesso al robottino di rilevare la presenza di ostacoli nella sua traiettoria e di fermarsi in modo tempestivo per evitare collisioni potenzialmente dannose.

Per rendere il sistema più avanzato, è stata introdotta la computer vision attraverso l'uso della webcam. Questo ha consentito al robottino di identificare due linee distintive sulla carreggiata, aprendo la strada a percorsi più complessi. Utilizzando algoritmi di elaborazione delle immagini, il robottino è stato in grado di riconoscere e seguire le due linee con precisione.

Successivamente, sono state introdotte le capacità di riconoscimento della segnaletica stradale utilizzando una rete neurale addestrata con l'algoritmo YOLO (You Only Look Once). La webcam acquisiva le immagini dell'ambiente circostante e, attraverso l'analisi delle immagini tramite la rete neurale, permetteva al robottino di identificare e interpretare segnali stradali, adattando di conseguenza il suo comportamento.

Questa relazione fornisce una panoramica dettagliata delle diverse fasi di sviluppo del robottino autonomo e spiega le tecniche utilizzate, i risultati ottenuti e le potenziali applicazioni future di tale sistema.

Design del robottino

I sensori e l'hardware utilizzati sono fondamentali per acquisire informazioni sull'ambiente circostante e consentire al sistema di prendere decisioni di guida appropriate.

Di seguito sono elencati i principali elementi costitutivi del progetto divisi in base al loro ruolo.

Struttura principale

La struttura di base è stata realizzata utilizzando il kit "Sound Founder Picar X". Esso fornisce una serie di componenti essenziali per il movimento e il controllo del veicolo, nonché un telaio su cui assemblare tutte le altre parti.

Le principali componenti fornite dal kit includono:

- a) Telaio: una serie di componenti strutturali che, una volta assemblati, forniscono una base solida e stabile su cui montare gli altri componenti.
- b) Ruote: Il kit comprende quattro ruote che consentono al robottino di muoversi lungo la superficie di guida. Esse sono progettate per offrire una presa salda e stabile sul terreno, garantendo una trazione adeguata per il movimento.
- c) Sterzi: l'equipaggiamento include un meccanismo di sterzo che consente alla macchina di cambiare direzione. Attraverso il controllo dei servomotori, il robottino può orientarsi verso sinistra o verso destra per seguire le linee della carreggiata e muoversi nello spazio.
- d) Motori: I motori inclusi forniscono la potenza di movimento. Sono collegati alle ruote e controllati dal Raspberry Pi per determinare la velocità del robottino.

Per l'assemblaggio della struttura, sono stati utilizzati rivetti e viti forniti dal kit stesso, seguendo attentamente le istruzioni di montaggio allegate: questo ha consentito di realizzare una struttura solida e ben organizzata per il robottino autonomo.

Grazie al kit "Sound Founder Picar X", è stato possibile creare una struttura robusta e funzionale per il robottino, fornendo una base solida per l'installazione dei componenti e per il suo movimento su diversi percorsi.

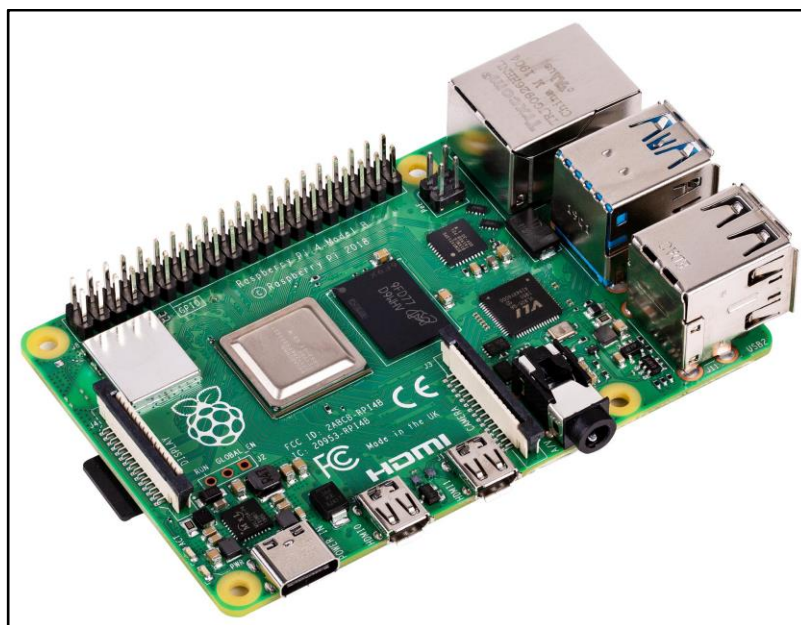
Cervello

La mente del robottino autonomo è rappresentata dal potente Raspberry Pi 4, una scheda single-board computer che offre un'ampia capacità di elaborazione e una vasta gamma di funzionalità. Grazie alla sua CPU quad-core e alle porte di connettività, il Raspberry Pi consente il controllo e l'elaborazione dei dati provenienti dai sensori. È su questa piattaforma che vengono implementati gli algoritmi di controllo, la computer vision e le reti neurali necessari per rendere il robottino autonomo.

Per garantire l'alimentazione del sistema, sono state utilizzate due batterie 18650 da 3000mAh a 3.7V. Queste batterie sono alloggiare in un "portatile" posizionato strategicamente sotto la struttura principale del robottino. Questo sistema di alimentazione fornisce l'energia necessaria per il funzionamento del Raspberry Pi e delle sue funzionalità. Al fine di posizionare correttamente il "cervello" all'interno del robottino, sono stati utilizzati degli "standoff". Questi distanziali sollevano la scheda sopra la struttura principale, consentendo una migliore ventilazione e proteggendo la scheda da eventuali cortocircuiti o danni fisici.

Inoltre, per semplificare la connessione delle varie componenti e ridurre la complessità del cablaggio, è stato utilizzato un "robot HAT". Questa scheda di espansione progettata per il Raspberry Pi si collega direttamente sopra i pin GPIO della scheda, facilitando l'interfacciamento con le diverse componenti del robottino. Il robot HAT ha consentito un'installazione più rapida e un cablaggio meno complicato, semplificando l'assemblaggio del robottino autonomo.

L'utilizzo del Raspberry Pi 4, delle batterie, degli standoffs e del robot HAT costituiscono una solida base hardware per il robottino autonomo, garantendo una potente capacità di elaborazione, un'adeguata alimentazione e una connessione semplificata delle componenti.



Sensori

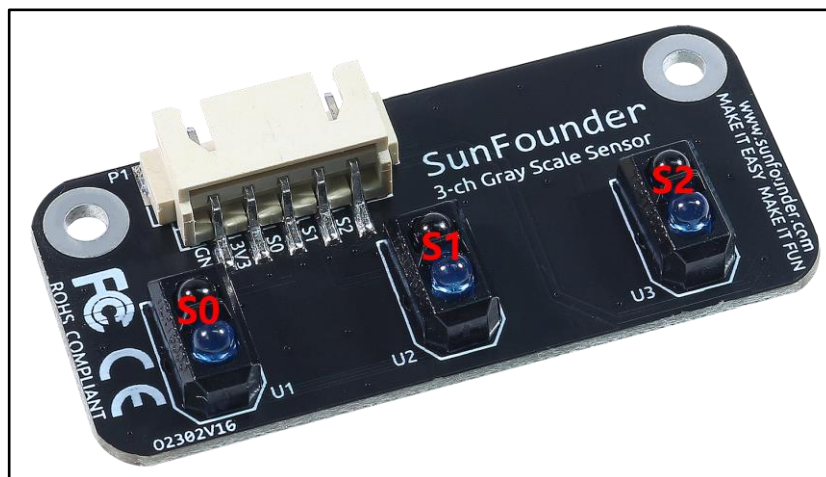
I sensori utilizzati nel robottino autonomo svolgono un ruolo fondamentale nell'acquisizione di informazioni sull'ambiente circostante e nel consentire al sistema di prendere decisioni di guida adeguate. Essi sono: una webcam, un sensore "Gray Scale Module" e un sensore "Ultrasonic Module".

- a) La webcam acquisisce le immagini dell'ambiente circostante e consente al robottino di rilevare le linee della carreggiata, identificare i segnali stradali e ottenere informazioni visive

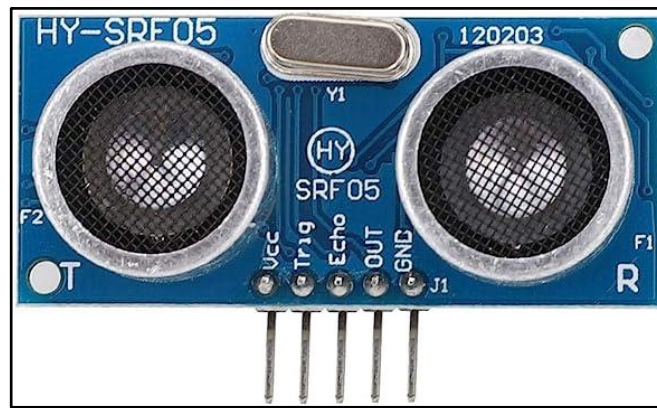
per la guida autonoma. Le immagini acquisite sono elaborate attraverso algoritmi di computer vision per riconoscere e analizzare gli elementi chiave della scena stradale.



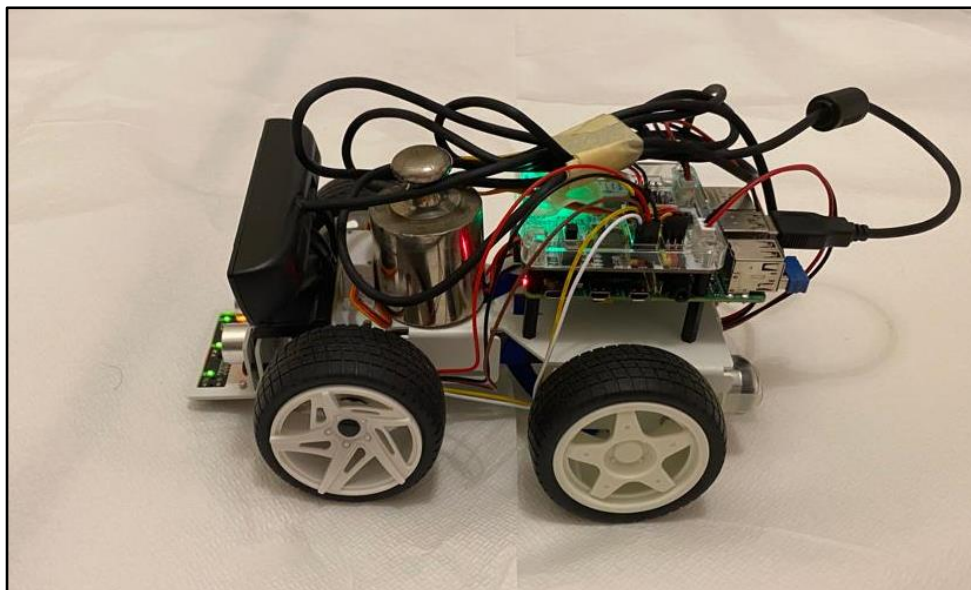
- b) Il sensore "Gray Scale Module" è utilizzato per mantenere il robottino allineato rispetto alla linea della carreggiata. Questo sensore rileva le variazioni di colore sulla superficie stradale e fornisce segnali al sistema per correggere la traiettoria. Utilizzando una matrice di fotodiodi, il sensore è in grado di distinguere tra diverse tonalità di grigio sulla strada e determinare la posizione del robottino rispetto alla linea di guida desiderata.



- c) Il sensore "Ultrasonic Module" viene impiegato per rilevare la distanza dagli ostacoli lungo la traiettoria di guida. Questo modulo utilizza onde sonore ad alta frequenza per misurare il tempo impiegato dalle stesse per tornare dopo essere state riflesse dagli ostacoli circostanti. In base al tempo di ritorno, il sensore calcola la distanza tra il robottino e gli oggetti circostanti. Queste informazioni consentono al sistema di attivare la funzionalità di "collision detection" e di arresto in caso di ostacoli imminenti lungo il percorso.



Grazie alle informazioni acquisite da questi sensori ed elaborate dal Raspberry Pi, il robottino è in grado di prendere decisioni di guida autonoma, mantenere l'allineamento sulla linea della carreggiata e reagire in modo adeguato agli ostacoli lungo il percorso.



Ambiente di Test

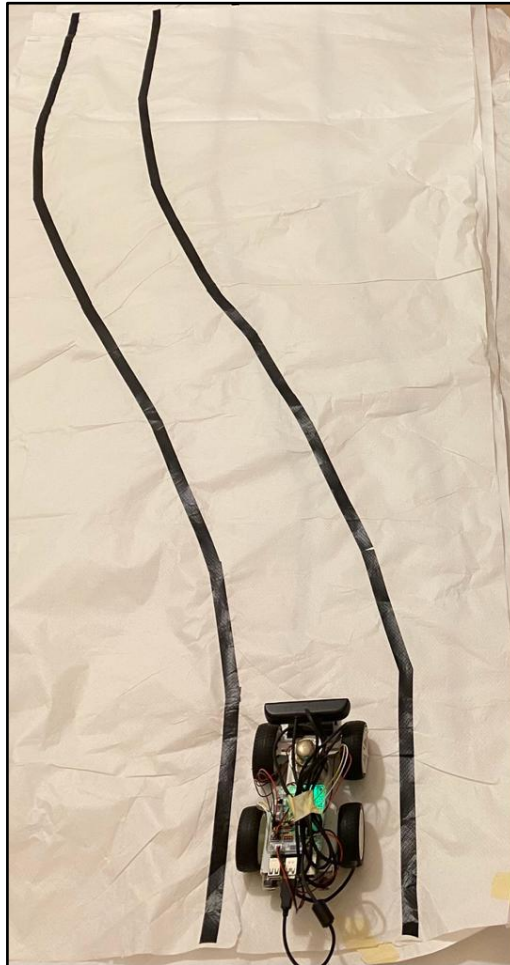
Abbiamo scelto di testare il robottino in 3 diversi scenari al fine di metterlo alla prova in situazioni diverse. L'idea iniziale era quella di realizzare un unico circuito, ma a causa delle limitazioni di spazio disponibile abbiamo optato per la costruzione di 3 scenari distinti, che rappresentano le tre situazioni principali con cui un dispositivo deve confrontarsi: un rettilineo, una doppia curva leggera (chicane) e una curva.

Di seguito viene illustrata una rappresentazione schematica delle 3 configurazioni:

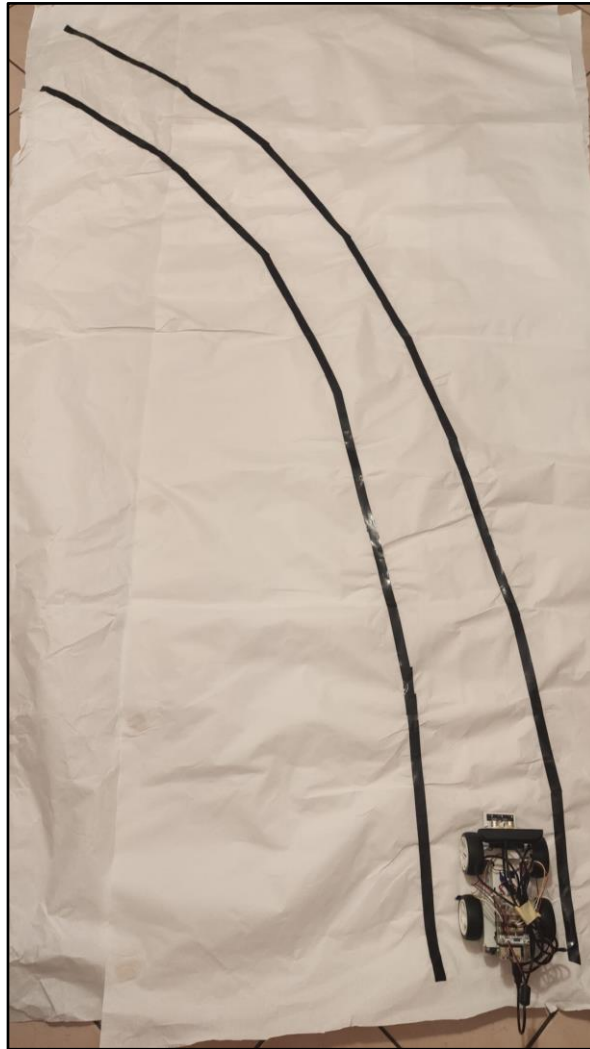
- Scenario rettilineo: esso consiste in un tratto rettilineo senza curve. È stato realizzato applicando del nastro adesivo nero su un terreno bianco, creando un contrasto elevato tra le due componenti. Ciò consente al robottino di rilevare la linea nera e mantenere l'allineamento durante la guida autonoma.



Scenario doppia curva leggera (chicane): Questo scenario simula una doppia curva leggera che richiede al robottino di adattarsi al cambiamento di direzione. Anche in questo caso, è stato utilizzato del nastro adesivo nero su un terreno bianco per creare la linea di guida e garantire una buona visibilità per i sensori del robottino.



- Scenario curva: consiste in una curva più pronunciata rispetto alla chicane. È stato realizzato nello stesso modo degli altri scenari, con del nastro adesivo nero su un terreno bianco. La curva rappresenta una sfida aggiuntiva per il robottino autonomo, richiedendo una maggiore capacità di sterzata e adattamento alla traiettoria curva.

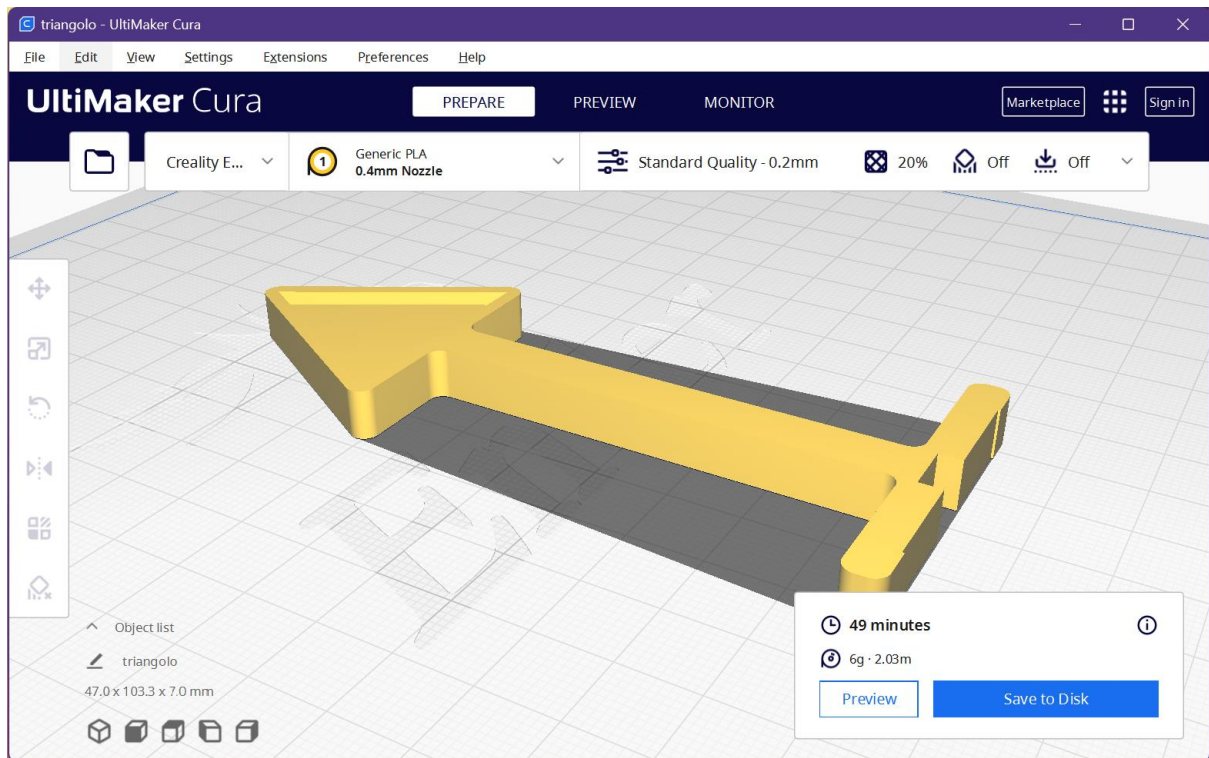


L'utilizzo del nastro adesivo nero su un terreno bianco offre un ambiente di test solido e riconoscibile per i sensori del robottino. Il contrasto elevato, dato dai due colori, consente ai sensori di rilevare con precisione la linea di guida e prendere decisioni di controllo in base alle informazioni fornite.

La scelta di creare scenari distinti ci ha permesso di valutare le prestazioni del robottino autonomo in varie condizioni realistiche, consentendo un'analisi più completa del suo comportamento e delle sue capacità di guida autonoma.

Segnaletica

Come detto nel capitolo precedente si vuole affrontare anche la sfida del riconoscimento della segnaletica stradale e dei pedoni utilizzando le immagini acquisite dalla webcam. Per rendere il tutto più reale possibile è stato scelto di utilizzare degli omini Lego per rappresentare i pedoni e realizzare, mediante una stampante 3D, la segnaletica in “miniatura”.



Implementazione software

Setup iniziale

La prima cosa da fare è installare il giusto OS sul raspberry: come sistema operativo è stato scelto "Raspbian GNU/Linux 10 (buster)". E' una distribuzione linux debian ottimizzata per raspberry.

Successivamente, è stata scaricata la repository ufficiale di "Sun founder" e installate tutte le dipendenze necessarie con il comando `"sudo python3 setup.py install"`.

Infine, è stato necessario abilitare l'interfaccia I2C per collegare i vari sensori al mini computer: essa normalmente è disattivata e per cambiare questa opzione si può agire come segue:

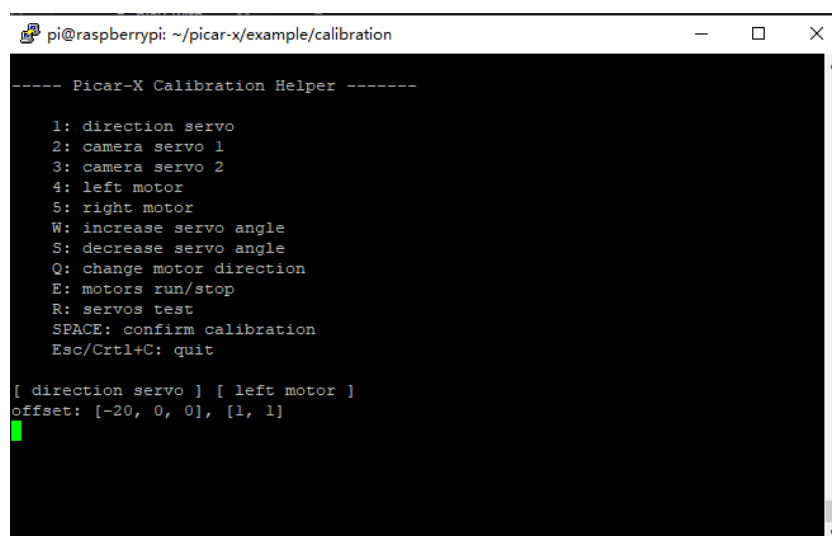
- da linea di comando digitare `"sudo raspi-config"`
- Selezionare "3 Interfacing Options"
- Selezionare "P5 I2C"
- Selezionare "Yes".

Per accedere al meglio al robottino è stata adottata una soluzione da remoto: per farlo è stato utilizzato il software "VNC Viewer" e abilitato il server VNC sul raspberry. Ciò è stato fatto collegando il raspberry e il pc che è stato usato come dispositivo di accesso alla stessa rete Wi-Fi e impostando un indirizzo IP statico sul raspberry disabilitando quindi il DHCP.

Calibrazione

Dopo aver installato tutto il necessario è stata eseguita una calibrazione per impostare al meglio i motori e l'angolo dello sterzo.

Per fare ciò è stato eseguito lo script `"calibration.py"` presente nella repository ufficiale.



```
pi@raspberrypi: ~/picar-x/example/calibration
----- Picar-X Calibration Helper -----

1: direction servo
2: camera servo 1
3: camera servo 2
4: left motor
5: right motor
W: increase servo angle
S: decrease servo angle
Q: change motor direction
E: motors run/stop
R: servos test
SPACE: confirm calibration
Esc/Ctrl+C: quit

[ direction servo ] [ left motor ]
offset: [-20, 0, 0], [1, 1]
```

La calibrazione è un passaggio fondamentale in quanto durante l'assemblaggio le varie parti possono disallinearsi (come è successo a noi) e risulta quindi necessario sistemare "a

posteriori” questo inconveniente, tramite una calibrazione degli angoli iniziali che le varie componenti devono avere.

PicarX

La libreria "PicarX" svolge un ruolo fondamentale nel controllo e nel movimento del robottino. Essa offre una serie di metodi in Python per comandare i motori e i servomotori collegati al Raspberry Pi.

Questi metodi consentono di controllare la velocità e la direzione del movimento, permettendo al robot di muoversi e di sterzare.

Oltre al controllo dei motori e dei servomotori, la libreria "PicarX" ci permette di interagire con i sensori presenti sul robottino. Tra questi sensori, vi sono il sensore di prossimità e quello di grigi già citati precedentemente.

Di seguito un elenco dei principali comandi utilizzati:

- `from picarx import Picarx` → importa la libreria necessaria nell'ambiente
- `px = Picarx()` → inizializza un'istanza di tipo Picarx
- `px.forward(speed)` → permette al robottino di muoversi in avanti con la velocità "speed" ("speed" == 0 ferma il movimento)
- `px.set_dir_servo_angle(angle)` → imposta lo sterzo (quindi le ruote anteriori) ad un angolo "angle"
- `px.stop()` → stop

Single Line Tracking

Il primo task è stato far seguire al robottino una singola linea tracciata sul percorso. Per fare ciò è stato usato il sensore che rileva la scala di grigi ("grey sensor") e le funzioni fornite dalla libreria di "PicarX".

Nello specifico il codice sfrutta le seguenti funzioni:

- `"get_grayscale_data()"` → fornisce in output 3 valori che rappresentano il valore di "chiarezza" rilevata dal sensore. Il valore ritornato sarà un qualcosa del tipo `[out_sx, out_centro, out_dx]` e minore sarà il valore rilevato, maggiore sarà la tonalità scura.
- `"get_line_status(gm_val_list)"` → in base ai valori rilevati da `get_grayscale_data()` ritorna i comandi "forward", "left" o "right" in base alla posizione della zona scura rispetto al sensore.

Analizzando il lato implementativo queste due funzioni si possono riassumere nel seguente modo:

- `get_grayscale_data()` → sfrutta una libreria "adc" che serve per convertire la corrente generata dal fotodiodo in una tensione proporzionale utilizzando un circuito di conversione analogico-digitale (ADC). Questo circuito converte la tensione in un

valore digitale che rappresenta l'intensità della luce rilevata. Maggiore sarà il valore ottenuto e maggiore sarà la luminosità che è stata rilevata.

- `get_line_status(list)` → esegue un controllo sui valori dell'array passato in ingresso comparandoli con un valore soglia impostato a 1000:
 - Se tutti e tre i valori sono maggiori del valore di riferimento allora ritorna "stop". Significa che il terreno sotto al sensore è tutto bianco e quindi non c'è nessuna linea da seguire.
 - Controlla se il fotosensore centrale (`array[1]`) è minore del valore soglia. Viene impartito il comando "vai dritto" (ritorna "forward").
 - Se il valore di `array[0]` è minore del valore soglia allora ritorna "right". Significa che è stata rilevata una luminosità elevata a destra.
 - Se il valore di `array[2]` è minore del valore soglia allora ritorna "left". Significa che è stata rilevata una luminosità elevata a sinistra

Infine, la decisione del robottino su come girare le ruote per continuare a seguire la linea scura viene presa leggendo prima il valore dei 3 fotodiodi (con `get_grayscale_data()`), poi capendo dov'è la parte più scura (`get_line_status()`) e infine a seconda del valore riscontrato adattando l'inclinazione delle ruote.

Collision avoidance

Questo task è stato realizzato sfruttando l' "Ultrasonic Module".

Attraverso la libreria PicarX possiamo accedervi in modo veloce ed efficace: usando il metodo "`ultrasonic.read()`" otteniamo il valore che legge il modulo. Questo valore indica la distanza dal sensore al primo oggetto rilevato, il tutto in centimetri.

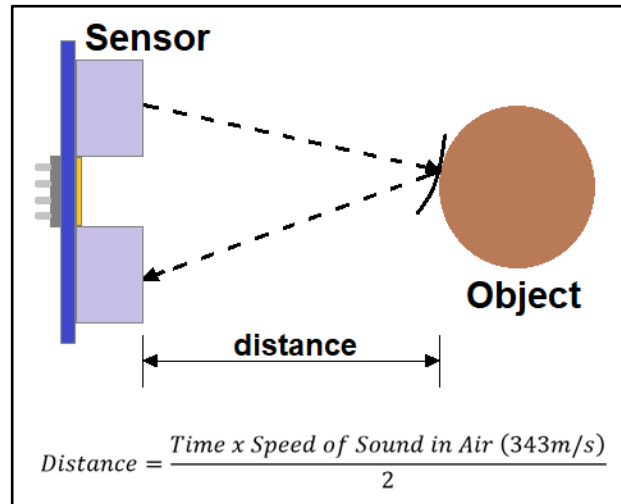
```
distance = px.ultrasonic.read()
```

Questa misura viene stimata effettuando questo processo:

1. Generazione dell'onda sonora → Il sensore emette un segnale sonoro ad alta frequenza, di solito nell'intervallo degli ultrasuoni, non udibile dall'orecchio umano.
2. Riflessione dell'onda sonora → L'onda sonora emessa colpisce un oggetto o una superficie circostante e venendo riflessa, ritorna verso il sensore.
3. Rilevamento del tempo di ritorno e calcolo della distanza → Il sensore misura il tempo impiegato dall'onda sonora per viaggiare, dalla sua emissione all'oggetto e tornare indietro al sensore stesso. Poiché la velocità del suono è nota (circa 343 metri al secondo a 20 gradi Celsius), è possibile calcolare la distanza percorsa dall'onda sonora. Il calcolo eseguito è il seguente:

```
during = pulse_end - pulse_start  
cm = round(during * 340 / 2 * 100 , 2)
```

During è l'intervallo di tempo tra quando si ha iniziato ad inviare il segnale (`pulse_start`) e quando si è ricevuto (`pulse_end`), 340 è la velocità dell'onda in m/s. Moltiplichiamo per 100 per avere il risultato in centimetri e dividiamo per due poiché consideriamo che l'onda ha percorso lo stesso spazio due volte (sia all'andata che al ritorno).



Il tutto viene implementato da dalla funzione “`ultrasonic.read()`”.

Poiché a noi interessa evitare collisioni impostiamo come valore soglia “5” (cinque centimetri) in modo tale che se la parte anteriore dell’automobile rileva un ostacolo ad una distanza inferiore a questo valore, si ferma:

```
distance = px.ultrasonic.read() #          leggi          distanza
if distance < 5:                 #          controlla         distanza
px.forward(0)                   # fermati
```

Two lines detection

L’idea ora è quella di rendere l’ambiente esterno più simile possibile ad un ambiente reale: per fare ciò abbiamo deciso di realizzare non più solo una linea centrale ma due. Esse possono essere ricondotte alle linee laterali che delimitano gli estremi della carreggiata. Il nostro obiettivo era quello di realizzare un algoritmo che permettesse al robottino di stare in mezzo a queste due linee e proseguire in avanti seguendo le linee poste lateralmente.

Gli strumenti utilizzati nei punti precedenti non sono stati sufficienti per adempiere a questo scopo perciò si è reso necessario utilizzare un dispositivo di acquisizione di input ulteriore: è stato scelto di utilizzare una webcam capace di catturare le immagini dell’ambiente esterno.

La webcam è stata posizionata nella parte frontale della struttura e collegata al raspberry tramite USB: le immagini catturate rappresentano ciò che si trova di fronte al robottino.

L’angolazione della webcam è fondamentale: essa infatti non deve essere troppo inclinata verso il basso in quanto questo limiterebbe molto lo spazio di analisi e si perderebbero eventuali curve. Di contro se l’angolo è troppo alto si catturerebbe solo la carreggiata lontana: questo potrebbe andare a confondere il robottino facendogli pensare che debba affrontare nell’immediato una curva o un rettilineo quando invece è ad una distanza considerevole.

Dovendo lavorare con le immagini acquisite dalla webcam il problema muta in un problema di computer vision: come facciamo a far riconoscere al raspberry quali sono le linee della carreggiata, la loro posizione e fargli decidere che tipo di azione intraprendere (proseguire dritto o sterzare)? Per risolvere questi problemi è stato diviso il processo in vari sottoproblemi affrontandoli uno per volta.

Il primo problema da affrontare è capire come acquisire un frame usando la webcam da Python e impostare la sua risoluzione: a questo scopo si è scelto di utilizzare la libreria OpenCV. Questa è la libreria maggiormente utilizzata nell'ambito della computer vision con linguaggio di programmazione Python. È molto flessibile e presenta una serie di funzionalità e procedure facili ed intuitive pronte all'uso e al contempo molto potenti ed efficienti.

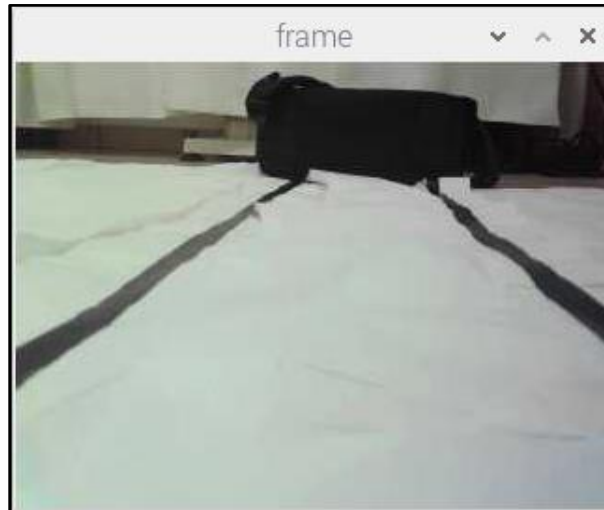
Le seguenti linee di codice rappresentano la nostra implementazione per catturare il flusso video della webcam.

```
# acquisizione video dalla webcam (0)
cap = cv2.VideoCapture(0)

cap.set(cv2.CAP_PROP_FRAME_WIDTH, 480)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 360)

while True:
    # acquisizione di un singolo frame dalla cattura video
    ret, frame = cap.read()
```

Con “`cap = cv2.VideoCapture(0)`” viene creato un oggetto `VideoCapture` che rappresenta la connessione alla webcam del computer. Il numero 0 come argomento indica che la prima webcam disponibile sarà utilizzata. Con i seguenti comandi “`cap.set(cv2.CAP_PROP_FRAME_WIDTH, 480)`” e “`cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 360)`” viene imposta la larghezza del frame video acquisito a 480 pixel e 360 pixel di altezza. La funzione `cap.read()` legge il frame successivo dalla webcam e restituisce due valori: `ret`, un valore booleano che indica se l'acquisizione è avvenuta correttamente, e `frame`, che rappresenta il frame acquisito come un'immagine.



Ottenuta l'immagine è necessario trovare un modo per trovare le linee della carreggiata e identificare la loro posizione.

Il primo passo è quindi trovare i bordi (edges) dei vari elementi all'interno dell'immagine:

```
edges = cv2.Canny(frame, 180, 280)
```

La funzione `cv2.Canny()` applica l'algoritmo di Canny per individuare i bordi nell'immagine `frame`.

Il risultato ottenuto è simile al seguente:



Successivamente viene selezionata solo una regione d'interesse applicando una maschera sull'immagine dei bordi: di fatto l'obiettivo attuale rimane individuare le linee della carreggiata e sapendo che esse saranno solo sulla parte inferiore dell'immagine, la parte superiore contiene informazioni non utili e che potenzialmente potrebbero confondere i prossimi passi.

```
mask = np.zeros_like(edges)
polygon = np.array([[
```

```
..]])  
cv2.fillPoly(mask, polygon, 255)  
cropped_edges = cv2.bitwise_and(edges, mask)
```

Viene dapprima creata una nuova immagine "maschera" ("mask") delle stesse dimensioni dell'immagine dei bordi ("edge"): essa avrà tutti i pixel a 0 (ovvero sarà tutta nera).

Successivamente viene definito un "poygon": esso specifica l'area di interesse all'interno dell'immagine. Viene creato come array di punti e rappresenta la metà dell'immagine con un piccolo margine aggiunto.

Attraverso "`fillPoly(mask, polygon, 255)`" viene creata un'area bianca (valore 255) all'interno del poligono, mentre il resto della maschera rimane nera.

Infine con "`cv2.bitwise_and(edges, mask)`" viene applicata un'operazione di "bitwise and" tra l'immagine edges e la maschera mask. Ciò significa che i pixel corrispondenti tra edges e mask vengono mantenuti, mentre i pixel non corrispondenti vengono impostati a zero. Il risultato è l'immagine cropped_edges, che contiene solo i bordi rilevati nella regione di interesse definita dal poligono.



Poi usando la trasformata di Hough probabilistica vengono rilevati tutti i segmenti di linee nell'immagine "cropped_edges"; ciò viene fatto usando la funzione "`cv2.HoughLinesP()`" a cui vengono passati tra gli altri i seguenti parametri:

- `rho = 1`: Rappresenta la precisione della distanza in pixel per la trasformata di Hough probabilistica. Un valore di 1 significa che si considera una precisione di un pixel nella rappresentazione dei segmenti di linea.
- `angle = np.pi / 180`: Rappresenta la precisione angolare in radianti per la trasformata di Hough probabilistica. Un valore di `np.pi / 180` indica che si considera una precisione di 1 grado nella rappresentazione dei segmenti di linea.
- `min_threshold = 10`: Rappresenta la soglia minima dei voti richiesta per considerare un segmento di linea rilevato come valido. I segmenti di linea con un numero di voti superiore a questa soglia verranno considerati come linee rilevate.

Viene restituita una lista di segmenti di linea rilevati nell'immagine `cropped_edges`, rappresentati dalle coordinate dei punti di inizio e fine di ciascun segmento:

```
[[[x1, y1, x2, y2]],  
 [[x1, y1, x2, y2]],  
 ...  
 [[x1, y1, x2, y2]]]
```

Con `x1` e `y1` il punto d'inizio del segmento e `x2` e `y2` il punto di fine del segmento.

I vari segmenti di linea rilevati e contenuti nell'array vengono poi combinati per formare una o due linee di corsia. Per farlo viene implementata una funzione chiamata `"average_slope_intercept()"`. Questa funzione utilizza le pendenze dei vari segmenti passati come input per determinare se appartengono alla corsia sinistra o destra, in base alla loro posizione rispetto ai confini dell'immagine. Calcola quindi le medie delle pendenze e degli intercetti dei segmenti per ciascuna corsia e infine restituisce le linee di corsia combinate come output.

Ciò che abbiamo ottenuto a questo punto è un array così formato:

```
[[[x1, y1, x2, y2]], # prima linea  
 [[x1, y1, x2, y2]] # seconda linea (se esiste)]
```

Ciò che vede la fotocamera è riportato di seguito:



Ora il problema è identificare come dovrà muoversi il robottino: idealmente vorremmo trovare la linea di mezzo tra le due linee laterali in modo tale che riesca a proseguire al centro della carreggiata. Non sempre però l'immagine acquisita è composta da due segmenti: può capitare infatti, che a causa di una curva il robottino "veda" solo la "linea più esterna". Dobbiamo quindi capire ciò che viene rilevato, se due linee o solamente una e successivamente trovare la linea mediana che rappresenta la direzione che le ruote del robottino dovranno seguire.

Nel caso in cui ci siano 2 linee rilevate l'angolo di curvatura per mantenere il veicolo nella corsia viene calcolato nel seguente modo:

- vengono calcolati x_offset e y_offset ovvero l'offset orizzontale e verticale del punto medio tra le linee di corsia rispetto al punto medio ideale;
- successivamente si calcola l'angolo di curvatura usando la funzione arcotangente dell'offset orizzontale diviso per l'offset verticale;
- l'angolo viene poi convertito in gradi e verificato che l'angolo di sterzata sia un valore tra -10 e 10 gradi in modo tale che si evitino sterzate troppo brusche.

Nel caso in cui ci sia solo una linea rilevata viene eseguito lo stesso procedimento ma x_offset viene calcolato tenendo in considerazione solo i valori $x1$ e $x2$ della singola linea. Inoltre, l'angolo di sterzata viene adattato e limitato nell'intervallo che intercorre tra -15 e 15 gradi per una gestione più stabile del veicolo.

Infine si noti che la velocità impostata è bassa (`px.forward(0.1)`) poiché si vuole garantire un tempo adeguato affinché il robottino analizzi il frame.

Riconoscimento della segnaletica

Successivamente, si è voluto sfruttare le immagini catturate dalla webcam per il riconoscimento degli oggetti presenti di fronte al robottino. Questa meccanica è nota come "object detection" ed è stata risolta tramite l'uso di reti neurali addestrate da un algoritmo chiamato YOLO.

Idea di fondo è quella di creare un modello capace di rilevare la presenza, all'interno dell'immagine passata come input, degli oggetti su cui è stato addestrato.

Per addestrare un modello con YOLO sono stati effettuati diversi passaggi:

- In primo luogo, sono stati raccolti i dati per l'addestramento: è necessario raccogliere un ampio set di immagini, le quali devono rappresentare le condizioni reali in cui il modello sarà utilizzato.
- In secondo luogo, i dati devono essere preparati con un formato adatto: è necessario suddividere le immagini in celle e associare a ciascuna cella le relative etichette di oggetti e bounding box.
- Successivamente avviene la fase dell'effettivo addestramento dove il modello impara a generare le bounding box corrette e a classificare correttamente gli oggetti presenti nelle immagini. Questo processo coinvolge l'ottimizzazione dei pesi del modello tramite l'algoritmo di discesa del gradiente.
- Infine, vengono eseguiti dei test e delle valutazioni per verificare le prestazioni del modello

Raccolta dati

Inizialmente, quindi, è stato acquisito un dataset di immagini contenenti gli oggetti che vogliamo che il nostro robottino riesca a riconoscere: in particolare le immagini contenevano il segnale di divieto, il segnale del semaforo e due tipi di pedoni diversi. I pedoni sono

rappresentati da due omini lego mentre la struttura dei segnali è stata realizzata attraverso una stampante 3D.

E' stato fatto in modo che il numero di immagini che ritraevano ciascun oggetto sia eguale per le varie categorie così da non creare discrepanze in fase di addestramento e/o verifica. Inoltre sono state realizzate immagini che contengono diverse tipologie di oggetti al loro interno: questo per realizzare situazioni più reali possibili (ad esempio pedone + semafo).

Preparazione dei dati

Successivamente, su ciascuna immagine sono state aggiunte le "etichette" e le "bounding box" per identificare i vari soggetti presenti e le loro posizioni all'interno delle immagini. Fatto questo, il dataset è stato diviso in training set (70%), validation set (20%) e testing set (10%): questa suddivisione aiuta a misurare e ottimizzare l'efficacia del modello. In particolare il training set è utilizzato per addestrare il modello di machine learning, il validation set viene utilizzato per ottimizzare i parametri del modello e selezionare la migliore configurazione del modello. Dopo ogni iterazione dell'addestramento, il modello viene valutato utilizzando il validation set per misurare la sua performance su dati non visti durante l'addestramento. Questo aiuta ad evitare il sovradattamento (overfitting) del modello ai dati di addestramento e a scegliere l'architettura e gli iperparametri migliori. Una volta che il modello è stato addestrato e ottimizzato, il testing set viene utilizzato per valutare le prestazioni del modello su dati completamente nuovi, simili a quelli che il modello incontrerà nella pratica. Questo fornisce una stima imparziale dell'accuratezza e delle capacità di generalizzazione del modello.

Per svolgere questi due compiti è stata utilizzata la piattaforma online chiamata "roboflow": questa fornisce una serie di strumenti e servizi che aiutano gli utenti a semplificare il processo della preparazione dei dati nei campi della visione artificiale e apprendimento automatico. In particolare i dati sono stati caricati sulla piattaforma, poi su ciascuna immagine sono stati aggiunti i vari box con le label adatte e infine si è deciso la proporzione dei vari set.

Nella pagina finale è stato scaricato il dataset per la versione YOLOv5: esistono diversi algoritmi, noi abbiamo scelto questo poiché il più veloce ed efficiente.

Addestramento e Test del modello

Per addestrare il modello è stata scaricata la repository ufficiale di YOLOv5 (<https://github.com/ultralytics/yolov5>), e successivamente sono state installate tutte le dipendenze necessarie: tra di esse troviamo torch.

Su un pc diverso dal raspberry, dopo aver sistemato i dati come suggerito dallo schema riportato dalla figura posta sopra, è stato addestrato un nuovo modello con il seguente comando:

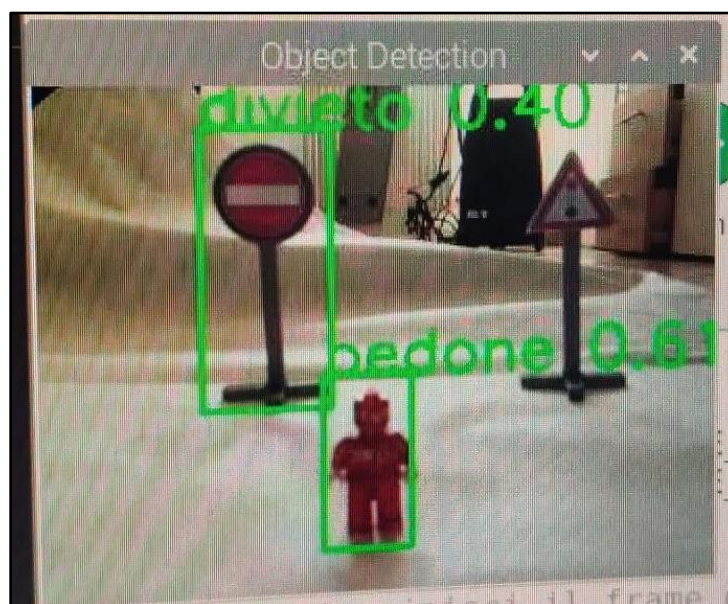
```
python train.py --img 240 --batch 16 --epochs 100 --data data.yaml -  
-weights yolov5.pt
```

Il comando crea dei pesi (un file in formato ".pt") che possono essere utilizzati per valutazioni future. Il modello personalizzato può essere caricato con il seguente comando:

```
model = torch.hub.load('ultralytics/yolov5', 'custom',  
path='best2.pt')
```

Questo sfrutta la funzione load della libreria di torch e prende come modello di riferimento quello posizionato nella repository presente in “ultralytics/yolov5”.

Il nostro scopo era quello di realizzare un'infrastruttura capace in tempo reale di riconoscere la segnaletica e l'ambiente esterno ed eventualmente prendere decisioni sulla base degli elementi che venivano rilevati. Per questo motivo la nostra idea iniziale è stata quella di elaborare le immagini attraverso una rete neurale che sfrutta i pesi creati nel punto precedente. Purtroppo dopo svariati test si è notato che il tempo di elaborazione per analizzare una singola immagine da parte del raspberry è eccessivamente elevato: nell'ordine dei 10 secondi. Abbiamo cercato di migliorare la situazione creando e alleggerendo il più possibile la struttura della rete neurale usando versioni di Yolo v5 come Yolo v5s e Yolo v5n ma purtroppo non abbiamo ottenuto i risultati sperati.

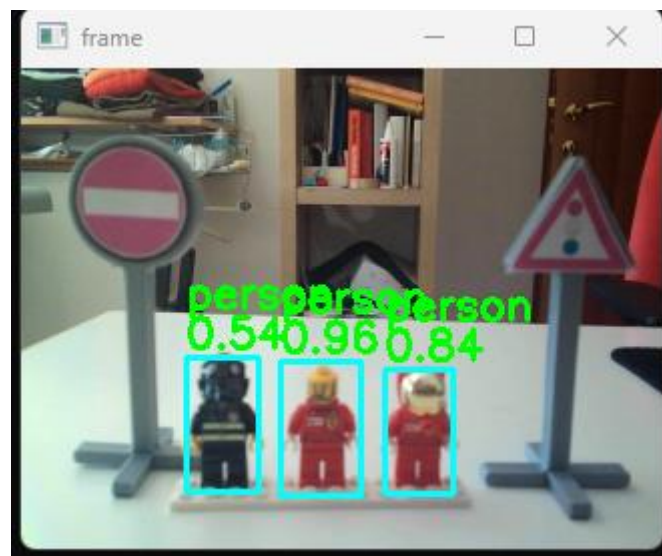


Successivamente abbiamo testato anche un secondo approccio: usare il raspberry come semplice acquirente dell'input video e usare un pc più potente per elaborare le immagini. Questo approccio va a ridurre considerevolmente il tempo per rilevare gli oggetti presenti in un singolo frame: il “pc remoto” utilizzato monta un processore i7-12700K, 32 GB di RAM e una scheda video RTX 3070 Ti. Con questa configurazione i frame vengono analizzati in tempo reale risolvendo il problema riscontrato nella situazione precedente. Per fare ciò è stata creata una struttura client-server tra il robottino e il “pc remoto”: il raspberry cattura i frame mediante Opencv, questi vengono inviati ad una socket verso il pc remoto, il quale li scarica e li analizza utilizzando la rete addestrata allo scopo. Elaborato l'oggetto rilevato viene inviata una richiesta GET, contenente un campo in cui inseriamo una stringa identificativa dell'oggetto rilevato, al server posto sul raspberry. A tal fine, abbiamo utilizzato una libreria chiamata “Flask” che permette di creare un server web e ricevere richieste a un dato indirizzo esponendo una API. Il robottino, quindi, capta l'informazione ottenuta e potrà decidere come usarla al meglio. Purtroppo, anche questa struttura non è sufficiente a soddisfare i nostri obiettivi: tra la cattura del frame e la ricezione dell'eventuale oggetto rilevato si è stimato che

passino circa 6 secondi. Questo tempo è troppo elevato per una struttura che dovrebbe funzionare in real-time.

Questi risultati potrebbero essere condizionati dal fatto che abbiamo utilizzato una rete wireless per collegare i due dispositivi. Solitamente questo tipo di rete è molto più lenta e possiede una latenza maggiore. I risultati sarebbero stati migliori se fosse stata utilizzata una rete con una latenza minore come un WIFI-6 a 5Ghz o una rete cablata. Si potevano collegare, ad esempio, i due dispositivi con un cavo ethernet ad uno switch. Quest'ultimo approccio però non è stato realizzato per mancanza di spazio e di materiale.

Infine, è stato testato un terzo approccio: al posto di Yolo si è utilizzato una sua versione modificata e adattata a dispositivi low end (come ad esempio il raspberry). Questa versione è reperibile al link: <https://github.com/dog-qiugu/Yolo-FastestV2>. E' stata utilizzata questa infrastruttura unita a un modello già pre addestrato fornito dalla repository: questo permette di riconoscere in maniera molto veloce le eventuali "persone" presenti nelle immagini. Questo modello è stato testato sul raspberry e si è notato che per singolo frame il raspberry ci metta circa 3 secondi per analizzarlo e verificare se è presente oppure no un omino nell'immagine: questo tempo purtroppo è ancora troppo elevato ma risulta un miglioramento rispetto a prima. Altra nota dolente per questa specifica implementazione è che usando un modello già pre addestrato su un dataset noto come COCO (<https://cocodataset.org/#home>), la segnaletica non viene rilevata.



Risultati e conclusioni

Concludendo, durante questo progetto è stato realizzato un robottino capace di muoversi autonomamente in uno spazio fisico reale ricreando il più possibile un ambiente esterno simile alla carreggiata di una strada.

Inizialmente, il primo obiettivo del robot è stato quello di seguire una singola linea tracciata sulla strada, task completato tramite l'utilizzo di un sensore a scala di grigi. I risultati ottenuti rispecchiano l'obiettivo in quanto il robottino è stato capace di muoversi autonomamente seguendo linee anche di diversa angolazione in tempo reale.

Successivamente, si è voluto implementare la funzionalità di "collision avoidance": questa rappresenta la capacità del robottino di fermarsi nel caso in cui rilevi un oggetto solido posto di fronte a lui. Questo serve ad evitare collisioni tra la macchinina ed eventuali ostacoli del mondo esterno. Il tutto è stato implementato sfruttando un modulo ad ultrasuoni. Il risultato finale è un automa capace di fermarsi in tutte le occasioni in cui un oggetto si pone nella sua traiettoria e di ripartire quando questo viene spostato.

Dopodiché, si è voluto implementare un modo per cui il robottino possa seguire una carreggiata con due linee laterali che la delimitano. Questa rappresenta la struttura più simile possibile a quella che ritroviamo quando guidiamo un'automobile nella quotidianità. Il nostro obiettivo è quello di rendere capace il robottino di rilevare le linee e seguire l'andamento della carreggiata rimanendo al centro della stessa. Per fare ciò è stata utilizzata la computer vision adattata alla nostra situazione. Il risultato finale è molto soddisfacente in quanto la macchina riesce a completare tutti e tre i percorsi che gli sono stati proposti: essi rappresentano tre situazioni tipiche che si possono incontrare nella realtà di tutti i giorni ovvero un rettilineo, una chicane e una curva.

Infine, si è voluto rendere il più possibile il robottino "intelligente" dandogli la capacità di riconoscere la segnaletica ed eventuali pedoni presenti all'interno del suo campo visivo. L'obiettivo era quindi quello di eseguire una "object detection" sui frame acquisiti dalla webcam. I risultati purtroppo non sono stati molto soddisfacenti poiché il tempo di computazione per analizzare l'immagine da parte del raspberry e rendere utilizzabili i dati estratti in real-time è molto più elevato rispetto a quanto previsto. Anche il tentativo di far svolgere l'analisi ad un "pc remoto", nonostante riesca a migliorare il risultato ottenuto, non riesce a permetterci di realizzare un sistema che funzioni in real-time. Infine, cambiando struttura al modello, usando un modello già addestrato e adatto a dispositivi low end si è riscontrato un miglioramento in tempi di computazione (circa 3 secondi per analizzare ciascun frame). Purtroppo però gli unici oggetti che riescono ad essere rilevati sono gli omini e il tempo di computazione risulta ancora troppo elevato per un utilizzo in real-time.

Queste problematiche sono sicuramente causate dalla difficoltà computazionale che il modello richiede per essere eseguito: il raspberry non possiede una potenza sufficiente per affrontare un calcolo così complesso in real-time. Per migliorare le cose si potrebbe utilizzare un dispositivo più performante adatto ad essere usato per il machine learning come il NVIDIA Jetson Orin Nano oppure di aggiungere una scheda video esterna al raspberry aggiungerebbe maggiore potenza e ridurrebbe il tempo di esecuzione come il Google Coral USB.

Come sviluppi futuri si potrebbe continuare a lavorare sul filone della computer vision migliorando il riconoscimento dell'ambiente esterno: algoritmi di computer vision e reti neurali create ad hoc per dispositivi low-end potrebbero essere sviluppati per migliorare le prestazioni.