



**Istituto Tecnico Industriale Statale Pietro Paleocapa**

Indirizzo Informatica e Telecomunicazioni

ELABORATO MULTIDISCIPLINARE PER ESAME DI STATO 2020/2021

# **Self Driving Car**

Tutor

**Prof.ssa Vita Anna Rosa Antonicelli**

Studente

**Niccolò Salvi**  
**salvi.16159**

**Anno Scolastico 2020-2021**



# Indice

<b>1</b>	<b>Introduzione</b>	<b>5</b>
1.1	Descrizione del progetto . . . . .	5
1.2	Nascita del progetto . . . . .	5
<b>2</b>	<b>Componenti</b>	<b>7</b>
2.1	Veicolo . . . . .	7
2.1.1	Raspberry Pi . . . . .	7
2.1.2	Pololu Romi . . . . .	8
2.1.3	Servo Motor . . . . .	8
2.1.4	Sensori ad ultrasuoni . . . . .	9
2.1.5	Raspberry Pi Camera . . . . .	10
2.2	Segnali stradali . . . . .	10
2.2.1	Stop . . . . .	10
2.2.2	Semaforo . . . . .	11
2.2.3	Limite di Velocità . . . . .	11
2.3	Pista . . . . .	11
<b>3</b>	<b>Guida Autonoma</b>	<b>13</b>
3.1	Librerie . . . . .	13
3.1.1	OpenCV . . . . .	13
3.1.2	PyTorch . . . . .	13
3.2	Rilevamento Corsie . . . . .	15
3.2.1	Descrizione Algoritmo . . . . .	15
3.2.2	Dimostrazione . . . . .	21
3.3	Riconoscimento Segnale di Stop . . . . .	22
3.3.1	Comportamento . . . . .	22
3.3.2	Descrizione Algoritmo . . . . .	22
3.3.3	Classificatore a cascata . . . . .	23
3.3.4	Dimostrazione . . . . .	24
3.4	Riconoscimento Semaforo . . . . .	25
3.4.1	Comportamento . . . . .	25
3.4.2	Rilevamento Struttura . . . . .	25
3.4.3	Rilevamento distanza . . . . .	25
3.4.4	Rilevamento colore . . . . .	26
3.4.5	Dimostrazione . . . . .	26
3.5	Riconoscimento Limite di Velocità . . . . .	27
3.5.1	Comportamento . . . . .	27
3.5.2	Rilevamento ROI . . . . .	27
3.5.3	Rilevamento distanza . . . . .	27
3.5.4	Rilevamento cifre . . . . .	27
3.5.5	OCR . . . . .	28
3.5.6	Dimostrazione . . . . .	30
3.6	Evitare Ostacoli . . . . .	31
3.7	Programmazione concorrente . . . . .	31
3.7.1	Dimostrazione . . . . .	31

<b>4 Alexa Skill</b>	<b>33</b>
4.1 Introduzione . . . . .	33
4.2 Skill . . . . .	33
4.2.1 Concetti di progettazione . . . . .	34
4.2.2 Lambda . . . . .	34
4.3 Raspberry Pi . . . . .	35
4.4 AWS IoT Core . . . . .	35
4.4.1 AWS IoT Device Shadow . . . . .	35
4.4.2 Broker di messaggi . . . . .	36
4.4.3 Sicurezza . . . . .	36
<b>5 Collegamenti Interdisciplinari</b>	<b>39</b>
5.1 Matematica . . . . .	39
5.2 PCTO . . . . .	39

# Capitolo 1

## Introduzione

Il campo dell'automazione sta riscuotendo un interesse sempre maggiore da parte della ricerca e grazie ad esso sono stati raggiunti numerosi traguardi in vari campi applicativi; uno di questi è stato quello delle auto a guida autonoma.

Essa rivestirà un ruolo fondamentale nella riduzione dei rischi connessi al fattore umano e, grazie alla possibile interconnessione tra veicoli e infrastrutture, nella ricerca di un ottimo globale in termini di consumi, inquinamento e tempi di percorrenza.

Il presente documento tratta dettagliatamente l'analisi, gli aspetti tecnici, l'algoritmo, l'implementazione e tutto ciò che riguarda il progetto *Self Driving Car*.

Il progetto è stato realizzato come elaborato multidisciplinare per l'Esame di Stato dell'anno scolastico 2020/2021.

### 1.1 Descrizione del progetto

*Self Driving Car* è l'implementazione di un veicolo in grado di muoversi in maniera autonoma e di prendere decisioni al riconoscimento di specifici segnali stradali, tramite l'utilizzo di tecniche di intelligenza e visione artificiale.

### 1.2 Nascita del progetto

Il progetto nasce dalla passione per la robotica e l'automazione, dalla curiosità di comprendere, in piccola parte, come le macchine reali siano in grado di muoversi autonomamente e dalla volontà di avvicinarsi e di approfondire l'ambito riguardante intelligenza e visione artificiale.

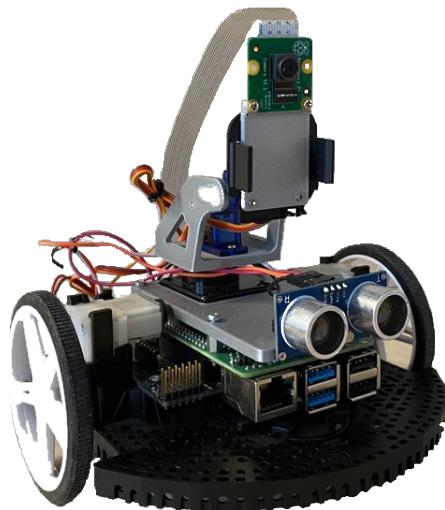


# Capitolo 2

## Componenti

### 2.1 Veicolo

Il robot impiegato durante lo sviluppo del progetto è quello mostrato in figura e descritto dettagliatamente nelle prossime sezioni.



**Figura 2.1:** Veicolo impiegato

#### 2.1.1 Raspberry Pi 4

Esso rappresenta il cervello del veicolo, al quale potenzialmente potrebbero giungere numerosi flussi di dati in input (stream video delle camere montate, segnali da sensori, ...) e stabilire, mediante la definizione di altrettanti algoritmi, l'operazione da svolgere (rallentare, accelerare, ruotare, fermarsi, ...).

Nella versione implementata, due sole tipologie di dati giungono al *Raspberry Pi*: video stream catturato dalla camera montata ed eventuale distanza che lo separa da un ostacolo di fronte a sé.

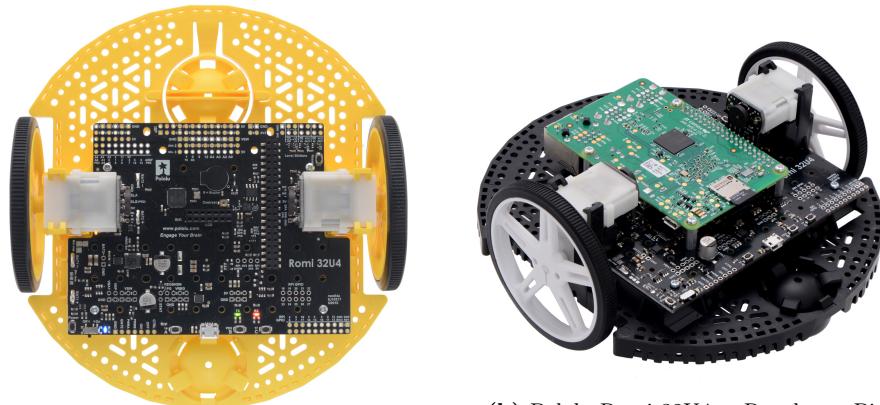


**Figura 2.2:** Raspberry Pi 4

### 2.1.2 Pololu Romi 32U4

La scheda di controllo *Romi 32U4* trasforma lo chassis Romi in un robot programmabile basato su MCU ATmega32U4 compatibile con Arduino. Le sue caratteristiche includono driver a doppio motore integrati, un circuito di alimentazione versatile e sensori inerziali, connessioni per encoder ed un display LCD opzionale.

La scheda ha la capacità di interfacciarsi con un *Raspberry Pi* aggiuntivo: i traslatori di livello integrati semplificano l'impostazione della comunicazione  $I^2C$  e l'interfaccia di altri segnali tra i due controller. In questa configurazione, il *Raspberry Pi* può gestire il controllo del robot di alto livello facendo affidamento sulla scheda di controllo *Romi 32U4* per attività di basso livello, come l'azionamento di motori, la lettura di encoder e l'interfaccia con altri dispositivi analogici o sensibili alla temporizzazione.



**(b)** Pololu Romi 32U4 + Raspberry Pi

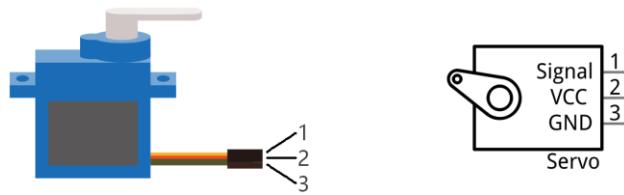
**(a)** Pololu Romi 32U4

**Figura 2.3:** Pololu Romi Setup

### 2.1.3 Servo Motor

Il servo motore è un sistema composto da motore DC, riduttore, sensore e circuito di controllo, in grado di ruotare nell'intervallo di 180° e 360°.

Dispone di tre linee: quella di segnale (1 → *Signal*), quella di alimentazione elettrica positiva (2 → *VCC*) e negativa (3 → *GND*).



**Figura 2.4:** Servo motore

È stato utilizzato il segnale PWM a 50Hz con un duty cycle in un certo intervallo per muovere il servo. Il tempo di durata 0,5 ms - 2,5 ms del livello alto di ciclo singolo PWM corrisponde all'angolo del servo 0° - 180° linearmente. Alcuni dei valori sono riportati nella tabella seguente:

High level time	Servo angle
0.5 ms	0°
1 ms	40°
1.5 ms	90°
2 ms	135°
2.5 ms	180°

I servo montati sul veicolo sono due: quello alla base del supporto della camera, che consente al robot di inquadrare ciò che si trova al suo fianco, e quello montato nella parte superiore che permette di regolare l'angolo di inclinazione della camera.

#### 2.1.4 Sensori ad ultrasuoni

Il modulo ad ultrasuoni integra un trasmettitore e un ricevitore: il primo viene utilizzato per convertire i segnali elettrici (energia elettrica) in onde sonore (energia meccanica) e la funzione del ricevitore è opposta.

Esso utilizza il principio secondo il quale gli ultrasuoni vengono riflessi quando incontrano degli ostacoli, ritornando verso la sorgente.

È necessario memorizzare il tempo in cui l'ultrasuono viene trasmesso ed il tempo in cui esso viene ricevuto, in modo tale da ottenere il delta.

Poiché la velocità del suono nell'aria è costante,  $v = 340 \text{ m/s}$ , è possibile calcolare la distanza tra il sensore e l'ostacolo nel seguente modo:

$$s = v * t / 2$$



**Figura 2.5:** Sensori ad ultrasuoni

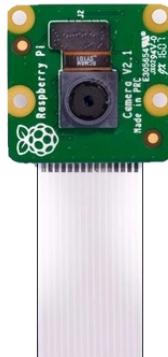
Tale sensore è in grado di rilevare una distanza minima di 2 cm ed una massima di 200 cm.

### 2.1.5 Raspberry Pi Camera

Il *Camera Module V2* ha un sensore Sony IMX219 da 8 mpx.

Supporta le modalità video 1080p30, 720p60 e VGA90, oltre alla cattura di immagini fisse.

Si collega tramite un cavo a nastro alla porta *CSI* del *Raspberry Pi*; è possibile accedervi tramite le API *MMAL* e *V4L* e sono disponibili numerose librerie di terze parti, inclusa la libreria *Picamera Python*.



**Figura 2.6:** Raspberry Pi Camera

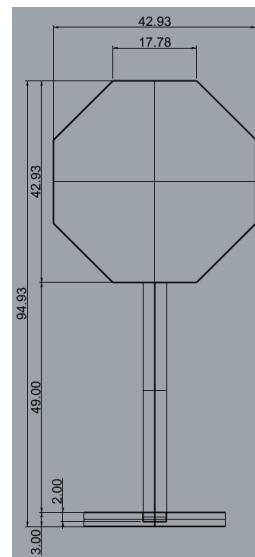
## 2.2 Segnali stradali

I segnali stradali impiegati sono stati prima disegnati, tramite il software di modellazione 3D *Rhinoceros 6* e successivamente stampati, utilizzando la stampante 3D *Alfawise U30*.

### 2.2.1 Stop



(a) Modello 3D



(b) Quotatura

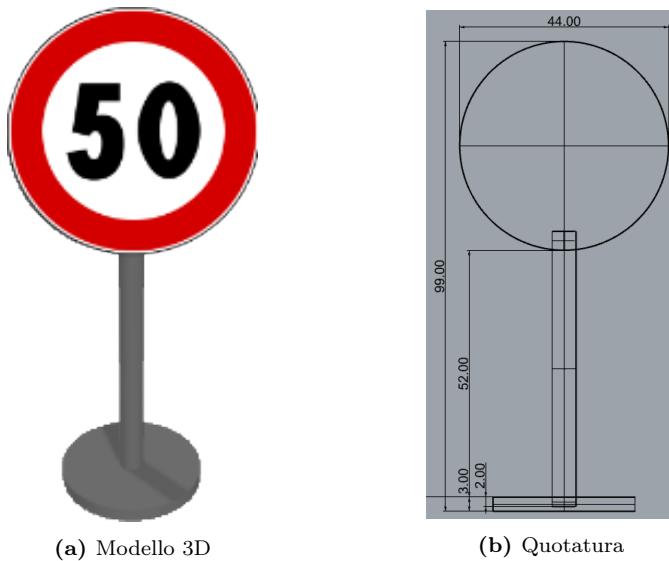
**Figura 2.7:** Segnale stop 3D

## 2.2.2 Semaforo



**Figura 2.8:** Modello 3D

## 2.2.3 Limite di Velocità



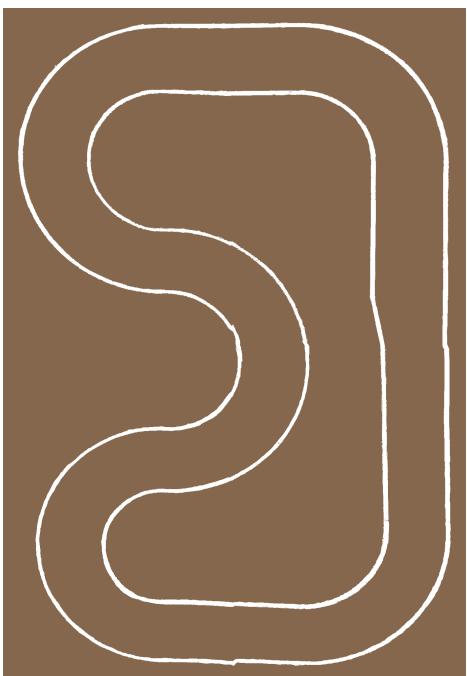
**Figura 2.9:** Limite velocità 3D

## 2.3 Pista

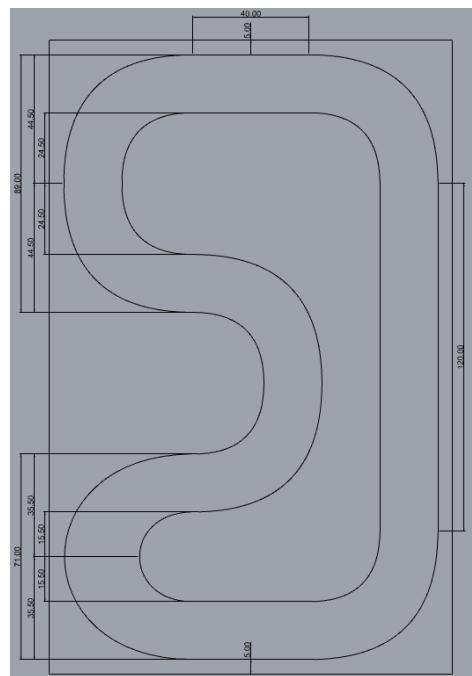
Nella versione iniziale della pista, le corsie sono state disegnate su dei cartoni color beige e colorate con della pittura bianca acrilica. È stato verificato, in seguito a numerose prove, che in condizione di luce non ottimali, l'algoritmo di threshold per distinguere le corsie dal resto della pista non funzionasse correttamente e non consentiva al robot di seguire correttamente il tracciato.

Per risolvere ciò, la base della pista è stata sostituita con dei cartoni neri lucidi e le corsie pitturate nello stesso modo della versione precedente. Il contrasto tra le linee bianche ed il resto della pista è così evidente,

che l'algoritmo di threshold funziona anche in condizione di luce scarse e non ottimali.

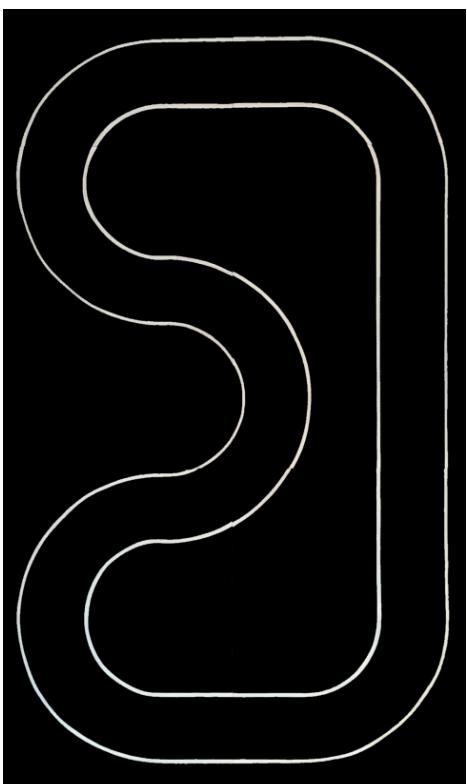


(a) Disegno

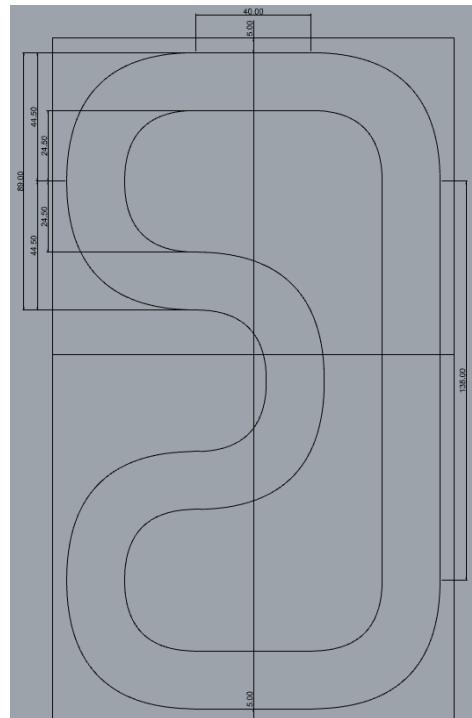


(b) Quotatura

**Figura 2.10:** Versione di test



(a) Disegno



(b) Quotatura

**Figura 2.11:** Versione finale

# Capitolo 3

## Guida Autonoma

In questo capitolo affronteremo ed approfondiremo i vari algoritmi che permettono al veicolo di muoversi in maniera autonoma.

Esso è in grado di riconoscere le corsie disegnate sul percorso, di determinare la presenza di ostacoli dinnanzi a sé, di rilevare i seguenti segnali stradali: stop, limite di velocità e semaforo.

### 3.1 Librerie

#### 3.1.1 OpenCV

OpenCV (Open Source Computer Vision Library) è una libreria open source che include numerosi algoritmi di computer vision; essa supporta un'ampia varietà di linguaggi di programmazione come C++, Python, Java, JavaScript, ecc... ed è disponibile su diverse piattaforme tra cui Windows, Linux, Android e iOS.

Sito web ufficiale accessibile al seguente link: <http://opencv.org>

#### OpenCV-Python

OpenCV-Python è l'API Python per OpenCV, che combina le API OpenCV C++ ed il linguaggio Python.

Si tratta di un wrapper Python per l'implementazione originale di OpenCV C++.

Documentazione ufficiale accessibile al seguente link: [https://docs.opencv.org/master/d6/d00/tutorial\\_py\\_root.html](https://docs.opencv.org/master/d6/d00/tutorial_py_root.html)

#### NumPy

OpenCV-Python utilizza Numpy, una libreria altamente ottimizzata per operazioni numeriche, con la quale tutte le strutture di array OpenCV vengono convertite in Numpy array.

Sito web ufficiale accessibile al seguente link: <https://numpy.org/>

#### 3.1.2 PyTorch

PyTorch è una libreria di apprendimento automatico open source basata sulla libreria Torch, utilizzata per applicazioni come la visione artificiale e l'elaborazione del linguaggio naturale.

PyTorch ha anche un'interfaccia C++, sebbene l'interfaccia Python sia più evoluta e sia l'obiettivo principale dello sviluppo.

Essa offre due funzionalità di alto livello:

- Tensor computing (come NumPy) con forte accelerazione tramite unità di elaborazione grafica (GPU)
- Possibilità di sviluppare accurate reti neurali

PyTorch definisce una classe *Tensor* per memorizzare e operare su matrici rettangolari multidimensionali omogenee di numeri.

I tensori PyTorch sono simili agli array NumPy, ma possono essere utilizzati anche su GPU Nvidia compatibili con CUDA.

Sito web ufficiale accessibile al seguente link: <https://pytorch.org/>

## Torchvision

TorchVision è la libreria di visione artificiale di PyTorch che contiene numerosi dataset di dati, oggetti, nonché modelli e operazioni di trasformazione che vengono spesso impiegati nell'area della visione artificiale.

Sito web ufficiale accessibile al seguente link: <https://pytorch.org/vision/stable/index.html>

## 3.2 Rilevamento Corsie

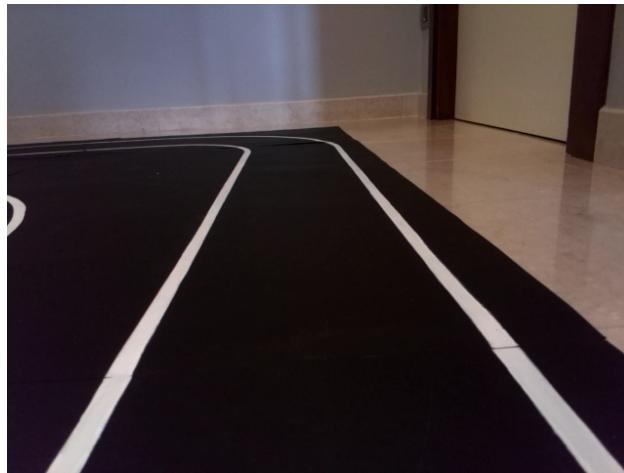
### 3.2.1 Descrizione Algoritmo

Le caratteristiche dell'immagine catturata dalla *Raspberry Pi Camera* sono le seguenti:

**larghezza:** 640 px

**altezza:** 480 px

**formato:** BGR



**Figura 3.1:** Frame

### Trasformazione immagine

Partendo dall'immagine catturata dalla camera, l'operazione di riconoscimento delle corsie viene eseguita su una regione d'interesse (*ROI*, mostrata in Figura: 3.2), la quale consente al veicolo di focalizzare la propria attenzione su una porzione ridotta del frame originale. Tale scelta è motivata dal fatto che al veicolo non serve calcolare l'angolo di curvatura delle regioni distanti, bensì della porzione di strada prossima ad esso.



**Figura 3.2:** ROI

Successivamente è necessario distinguere le corsie rispetto al resto dell'ambiente e viene prima effettuata una trasformazione dell'immagine in scala di grigi (Figura: 3.3a) e successivamente viene applicato l'algoritmo di threshold.

Per ogni pixel del frame, se il suo valore è inferiore ad una determinata soglia, viene settato a 0, altrimenti ad un valore massimo stabilito (Figura: 3.3b).



(a) ROI - Scala di grigi



(b) ROI - Threshold

**Figura 3.3:** Trasformazione immagine

Le operazioni che verranno trattate nelle prossime sezioni, verranno applicate all’immagine in bianco e nero, mostrata in Figura: 3.3b.

Per rendere più efficace il rilevamento delle corsie, si procede con la trasformazione prospettica dell’immagine, generando una vista dall’alto che consente di visualizzare delle linee quasi dritte, facilitando l’applicazione dell’algoritmo.

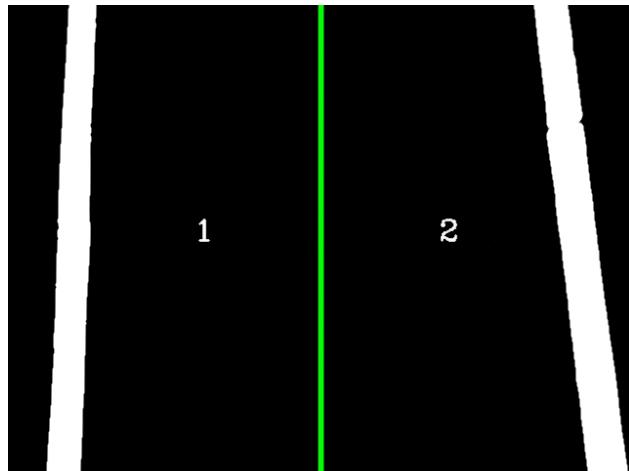


**Figura 3.4:** Trasformazione prospettica

### Rilevamento linee

Giunti a tal punto, sono terminate le trasformazioni sull’immagine e l’algoritmo prevede, partendo dal punto medio del frame, l’individuazione dell’area di sinistra (1) e di destra (2 ).

Il veicolo ruotando su sé stesso, si pone l’obiettivo di mantenere le due superfici le più simili possibile, rimanendo in tal modo al centro della corsia.



**Figura 3.5:** Divisione dell'Immagine

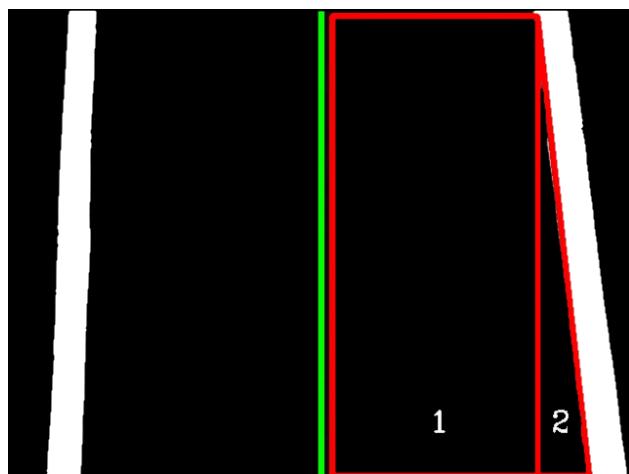
**Area di destra** Per facilitare la comprensione dell'algoritmo eseguito, vengono impiegati due colori differenti, per distinguere le superficie da sommare e da sottrarre:

■ porzione di area da sommare

■ porzione di area da sottrarre

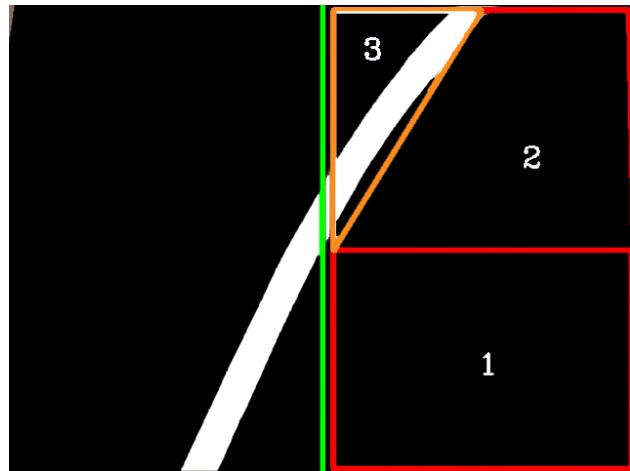
Le situazioni gestite sono due:

**Rettilineo**  $Area = Area_1 + Area_2$



**Figura 3.6:** Condizione di rettilineo

**Curva stretta**  $Area = Area_1 + Area_2 - Area_3$

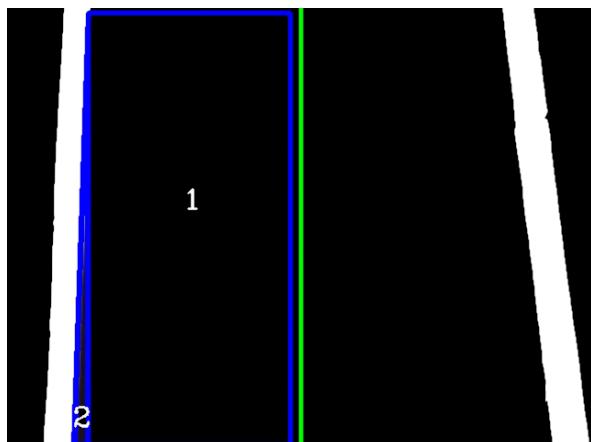


**Figura 3.7:** Condizione di curva

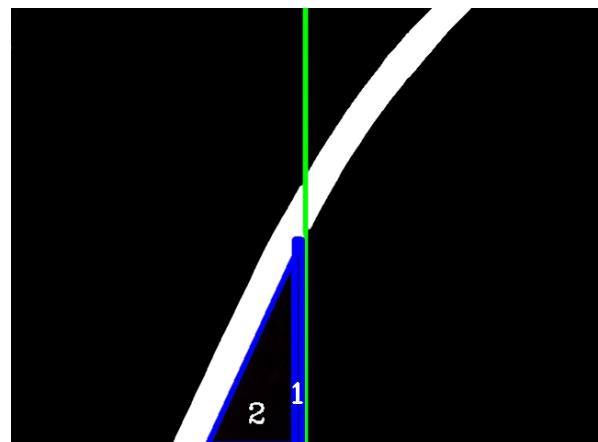
**Area di sinistra** Esattamente allo stesso modo di quanto viene svolto con l'area di destra, sono stati impiegati due colori differenti per facilitarne la comprensione:

■ porzione di area da sommare

■ porzione di area da sottrarre



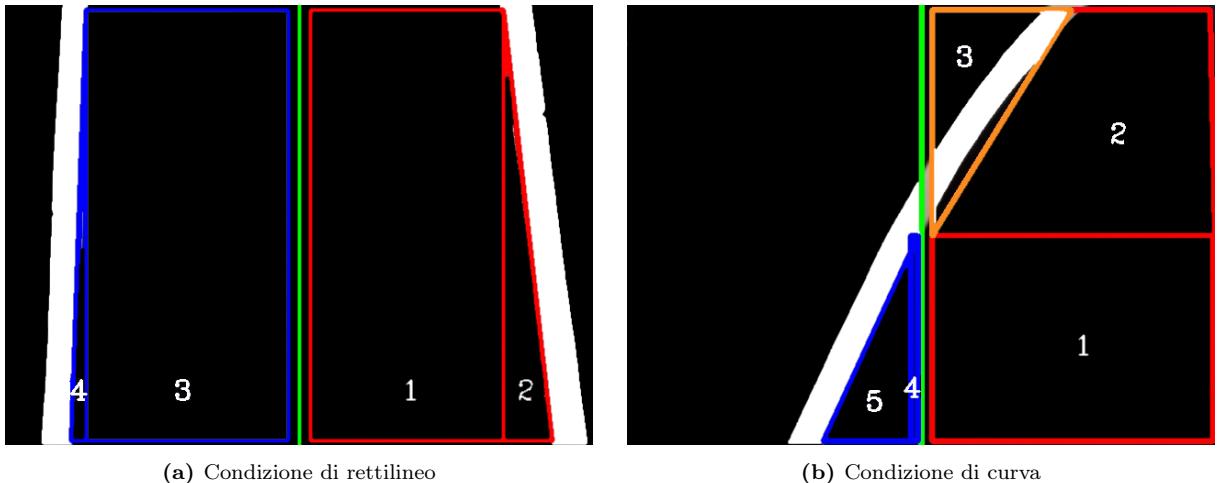
(a) Condizione di rettilineo



(b) Condizione di curva

**Figura 3.8:** Area di destra

**Arearie combinate** Naturalmente, durante il movimento del robot serve che vengano calcolate entrambe le aree, come mostra la figura che segue.



**Figura 3.9:** Area di destra

### Angolo di curvatura

Il senso di rotazione dipende dal parametro ricevuto in input dalla funzione *rotate*: se positivo ruota in senso orario, altrimenti in senso antiorario.

Il veicolo, per rimanere al centro della corsia, deve effettuare una rotazione di un preciso angolo, definito in base alla seguente modalità: viene calcolato il quoziente tra una costante (dipendente dalla velocità) ed il risultato della sottrazione tra un'area e l'altra.

$$\text{angoloDiCurvatura} = (\text{areaDestra} - \text{areaSinistra}) / \text{const}$$

È facilmente intuibile che se l'area di destra è maggiore di quella sinistra, serve ruotare verso destra e di conseguenza l'angolo di curvatura ottenuto sarà positivo e quindi coerente con quanto detto precedentemente riguardo i parametri in ingresso della funzione di rotazione.

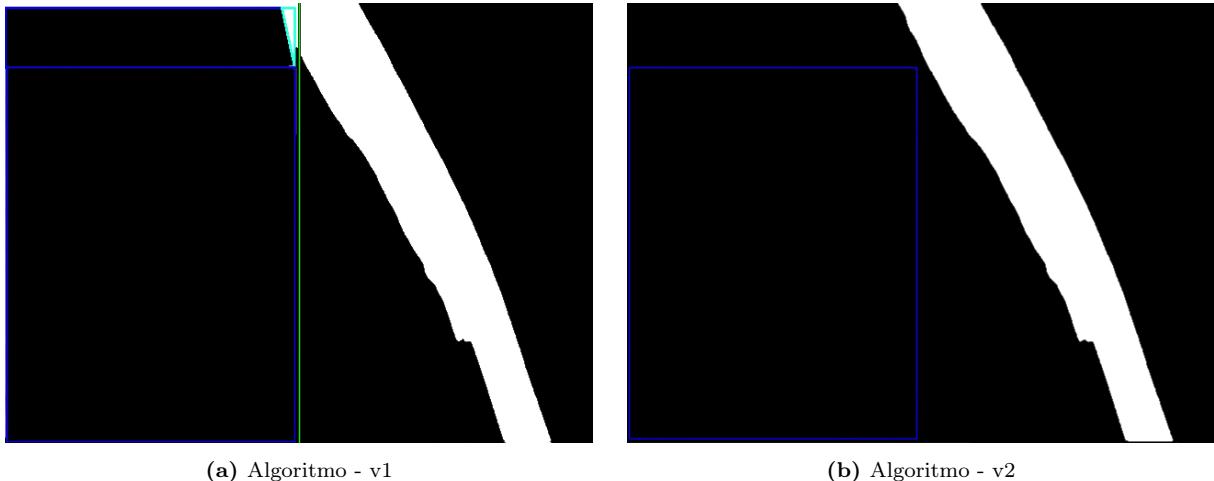
**Test** Durante le prime corse sulla pista indicata in Figura 2.10a, il veicolo seguiva una traiettoria fin troppo interna, che tagliava in maniera evidente le corsie, specialmente la curva in basso a sinistra, di curvatura minore rispetto alle altre.

Tale comportamento era dovuto ad una calcolo errato della superficie, dovuto ad una mal visione delle linee, a causa del loro raggio di curvatura eccessivamente stretto. Per migliorare l'esecuzione, dopo ulteriori prove e approfondimenti, ho individuato due strategie: o aumentare il raggio di curvatura di tale curva oppure applicare l'algoritmo ad una porzione di area ristretta.

Concludendo, possiamo affermare la correttezza dell'algoritmo sviluppato, ma non ottimale nell'esecuzione di curve troppo strette.

### Nuova implementazione

Avendo compreso che il problema era nell'ottenimento delle porzioni d'area da aggiungere nelle situazioni di curva stretta (Figura 3.10b), decido di sviluppare una nuova versione, basata sulla medesima logica, ma con un approccio leggermente differente al calcolo delle aree.



**Figura 3.10:** Differenza algoritmi

Essendo le porzioni di aree minori, era necessario l'applicazione di un'operazione matematica differente: la divisione tra le aree, mantenendo al numeratore l'area di superficie minore e successivamente moltiplicando per una costante..

```

if areaSinistra > areaDestra:
    angoloDiCurvatura = ((areaDestra/areaSinistra)-1) * const
    if angoloDiCurvatura > maxValore:
        angoloDiCurvatura = maxValore
    else:
        angoloDiCurvatura = -((areaSinistra/areaDestra)-1) * const
    if angoloDiCurvatura < -maxValore:
        angoloDiCurvatura = -maxValore

```

Tale implementazione, consente al veicolo di seguire correttamente le corsie, persino nella curva stretta che precedentemente dava qualche problema.

Per il disegno della pista definitiva (Figura 2.11a), ho preferito ugualmente mantenere tutte le curve dello stesso raggio, motivo per il quale, come si può vedere anche nei video, la macchina segue correttamente le corsie con entrambi gli algoritmi.

## Conclusioni

Per decidere quale delle due soluzioni sia effettivamente la migliore è necessario introdurre qualche considerazione, dopo che entrambe le versioni sono state testate diverse volte ed a differenti velocità.

Valutando l'aspetto algoritmico è più corretto quello che prevede l'operazione di sottrazione, perché calcola in maniera decisamente precisa e robusta ciascun area; considerando però il lato pratico, ovvero l'effettiva efficienza, come si può anche notare dai video linkati alla Sezione 3.2.2, il veicolo taglia eccessivamente le corsie nelle situazioni di curva.

Per quanto riguarda la seconda implementazione, quella che prevede l'utilizzo della divisione tra le aree, esegue in maniera molto precisa le curve, ma d'altra parte, impiegando la divisione su delle porzioni d'area minori, presenta un movimento più brusco e non lineare come quello precedente.

L'algoritmo di divisione tra le aree è il più indicato, in vista anche di future implementazioni, che sfrutteranno il riconoscimento delle corsie.

### 3.2.2 Dimostrazione

#### Youtube Video

[Sottrazione aree](#) video applicativo che mostra il funzionamento dell'algoritmo che implementa la sottrazione

[Divisione aree](#) video applicativo che mostra il funzionamento dell'algoritmo che implementa la divisione

#### Codice Github

[Sottrazione aree](#) codice sorgente eseguito su Raspberry per mettere in funzionamento e testare algoritmo che implementa la sottrazione

[Divisione aree](#) codice sorgente eseguito su Raspberry per mettere in funzionamento e testare algoritmo che implementa la divisione

### 3.3 Riconoscimento Segnale di Stop

#### 3.3.1 Comportamento

Il veicolo quando incontra il segnale di stop deve decelerare, a partire da una distanza adeguata, fino ad arrestarsi in prossima del segnale riconosciuto.

#### 3.3.2 Descrizione Algoritmo

L'algoritmo, definito per riconoscere la presenza del segnale di stop, prevede l'utilizzo di un classificatore a cascata.

Come mostrato in Figura 3.11a, il veicolo tramite tale tecnica è in grado di rilevare la regione di interesse (ROI), che racchiude al suo interno il segnale ricercato.

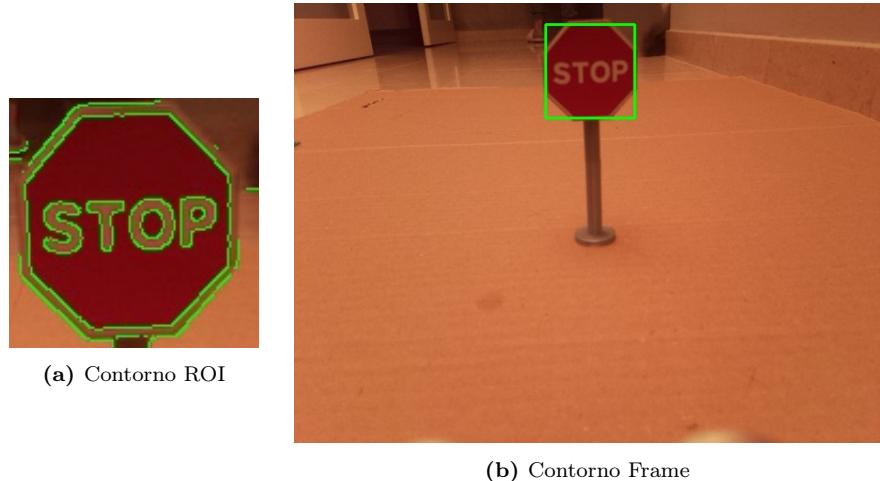


#### Rilevamento distanza

Come è stato definito nella Sezione 3.3.1, è previsto che il veicolo si fermi in prossima dello stop e quindi necessario sapere a quale distanza esso si trovi.

Come vedremo in seguito, nell'applicazione della formula per trovare la distanza, è fondamentale avere la corretta larghezza e/o altezza della regione d'interesse.

Dal momento che il classificatore, come si nota in Figura 3.11b, non riconosce perfettamente la sagoma del segnale, vengono ricercati, all'interno del ROI, i contorni (vedi Figura 3.12a).



**Figura 3.12:** Rilevamento Distanza

Il rettangolo mostrato nella Figura 3.12b è il contorno, fra quelli nella lista mostrati in Figura 3.12a, avente area maggiore, il quale racchiude perfettamente al suo interno il cartello e dal quale è possibile applicare la formula per ricavare la distanza.

**Distanza Focale** Per ottenere l'effettiva distanza alla quale il veicolo si trova, è stata prima calcolata la lunghezza focale della lente, nel modo che segue:

$$\text{lunghezzaFocale} = \text{distanzaReale} * \text{lunghezzaROI} / \text{lunghezzaSegnale}$$

I cui parametri sono:

- *lunghezzaFocale* rappresenta la distanza tra il centro ottico dell'obiettivo ed il piano della messa a fuoco
- *distanzaReale* indica la distanza reale al momento delle misurazioni [cm]
- *lunghezzaROI* rappresenta la lunghezza della regione di interesse [px]
- *lunghezzaSegnale* indica la lunghezza reale del segnale di stop [cm]

**Calcolo Distanza** Calcolata la lunghezza focale della lente, per ottenere la distanza viene applicata la seguente operazione:

$$\text{distanza} = \text{lunghezzaSegnale} * \text{lunghezzaFocale} / \text{lunghezzaRettangolo}$$

I cui parametri sono:

- *distanza* indica la distanza reale, entro la quale la macchina si deve fermare [cm]
- *lunghezzaSegnale* rappresenta la lunghezza reale del segnale di stop [cm]
- *lunghezzaFocale* indica la lunghezza focale della lente (calcolata precedentemente)
- *lunghezzaRettangolo* rappresenta la lunghezza del rettangolo che contiene al suo interno il segnale di stop [px]

### 3.3.3 Classificatore a cascata

Il rilevamento di oggetti che sfrutta i classificatori a cascata basati su funzionalità di Haar è un approccio basato sull'apprendimento automatico, in cui la funzione è addestrata da molte immagini positive e negative.

### **3.3.4 Dimostrazione**

#### **Youtube Video**

Video applicativo che mostra il funzionamento di quanto descritto in precedenza

#### **Codice Github**

Codice sorgente eseguito su Raspberry

## 3.4 Riconoscimento Semaforo

### 3.4.1 Comportamento

Il veicolo si arresterà in prossimità del semaforo rosso o giallo, preceduto da una breve fase di decelerazione lineare; mentre in prossimità del verde potrà riprendere e/o continuare la propria marcia.

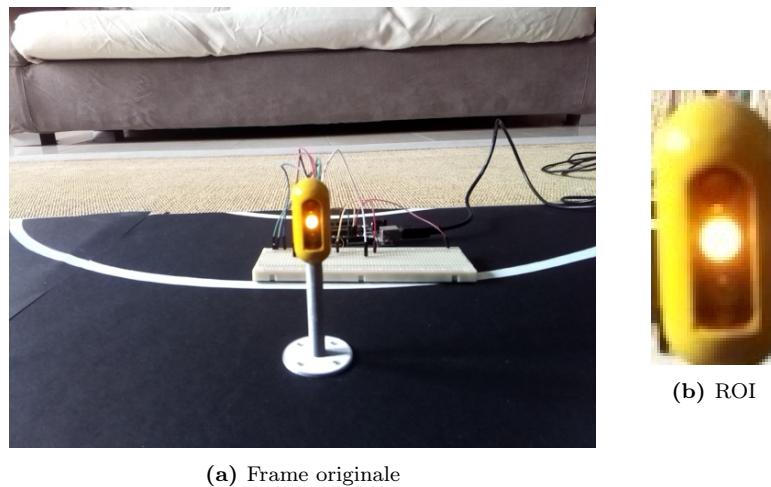
Una volta fermatosi, per poter inquadrare nuovamente il semaforo, viene ruotato il servo che muove la camera e successivamente, quando viene riconosciuto il colore **verde**, prima di riprendere la marcia, la camera verrà riportata nella sua posizione iniziale, per poter consentire al veicolo di riconoscere correttamente le corsie e gli altri segnali presenti.

Gli stati implementati sono 3:

- rimane acceso per 10 sec
- rimane acceso per 4 sec
- rimane acceso per 6 sec

### 3.4.2 Rilevamento Struttura

Per facilitare il riconoscimento del semaforo, esso è stato dipinto di giallo, in modo tale da rendere più efficace la rilevazione; viene applicata al frame catturato dalla camera una maschera, che ha la funzione di isolare le zone di colore giallo.



### 3.4.3 Rilevamento distanza

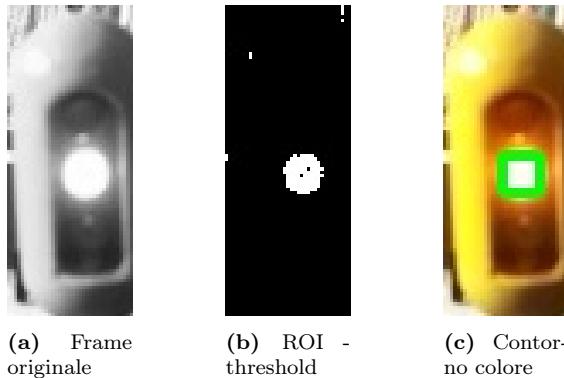
Una volta ottenuta la regione d'interesse, mostrata in la Figura 3.14b prima di riconoscere quale colore sia attivo al momento, serve verificare che la distanza alla quale si trova il veicolo dal semaforo sia minore di una soglia, determinata durante la fase di sviluppo e test.

Questa procedura è necessaria, in primis perché a grandi distanze, l'algoritmo di rilevamento dei colori non è in grado di ottenere correttamente i colori (le zone di luce non sono particolarmente chiare e non idonee con l'algoritmo definito) ed in secondo luogo perché nel caso di colore rosso o giallo attivo, la macchina deve fermarsi in prossimità.

La modalità di rilevamento della distanza è la medesima descritta nella Sezione 3.3.2.

### 3.4.4 Rilevamento colore

Quando viene riconosciuta la figura del semaforo entro la soglia corretta, vengono applicate le seguenti trasformazioni.



Ottenuta l'immagine in bianco e nero, vengono ricercati i contorni (il cui centro si trova a metà dell'immagine esaminata) presenti e quello di area maggiore rappresenterà la luce in quel momento accesa, come in Figura 3.14c.

Per ogni contorno, la libreria OpenCV ci fornisce anche le coordinate di esso e partendo da queste, è possibile stabilire quale dei tre colori sia acceso, effettuando un semplice controllo sulle coordinate Y.

Sia  $Y$  il la coordinata Y del angolo in alto a sinistra del contorno:

- $25 < Y < 35\%$  indica il colore verde
- $45\% < Y < 55\%$  indica il colore giallo
- $60\% < Y < 70\%$  indica il colore rosso

### 3.4.5 Dimostrazione

#### Youtube Video

Video applicativo che mostra il funzionamento di quanto descritto in precedenza

#### Codice Github

Codice sorgente eseguito su Raspberry

## 3.5 Riconoscimento Limite di Velocità

### 3.5.1 Comportamento

Il veicolo nel momento in cui riconosce il limite di velocità entro una determinata distanza da esso, avvia una fase di decelerazione o accelerazione (in base alla velocità precedente al riconoscimento e quella imposta dal limite riconosciuto) che si conclude in prossimità del segnale.

### 3.5.2 Rilevamento ROI

All'immagine catturata dalla camera, per localizzare il segnale di limite di velocità, viene applicata una maschera che consente di isolare le zone di colore rosso ed ottenere così la regione d'interesse, contenente al suo interno il segnale ricercato.



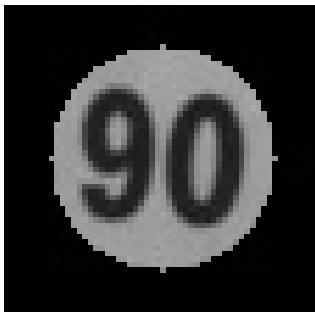
### 3.5.3 Rilevamento distanza

Similmente a quanto descritto nella Sezione 3.4.3, prima di procedere al riconoscimento dei valori numerici, viene calcolata la distanza a cui il segnale si trova per due motivi principali: primo, da un'elevata ed inadeguata distanza, il riconoscimento dei numeri non funzionava correttamente ed in secondo luogo ho preferito effettuare una fase di accelerazione o decelerazione piuttosto breve.

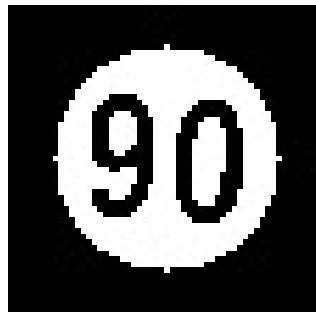
La fase di rilevamento della distanza è la medesima descritta nella Sezione 3.3.2.

### 3.5.4 Rilevamento cifre

Ottenuta la regione d'interesse, mostrata nella Figura 3.20c, entro la soglia prestabilita della distanza, vengono effettuate le seguenti trasformazioni, che consentono di individuare ogni cifra all'interno del segnale riconosciuto.



(a) ROI - Scala di gridi



(b) ROI - threshold



(c) Cifre identificate

### 3.5.5 OCR

Giunti a tal punto, è stata implementata, utilizzando la libreria PyTorch, una rete neurale in grado di identificare e riconoscere le cifre, in maniera tale da ottenere il valore del limite di velocità.

Una volta effettuato il threshold, viene restituito un vettore di oggetti (ciascuno di essi rappresenta le coordinate del contorno rettangolare) ordinato in base alla coordinata X del punto in alto-sinistra dello stesso.

Seguendo l'ordine di tale vettore, ciascun elemento viene analizzato e riconosciuto dalla rete neurale ed infine concatenato con gli elementi precedenti del vettore. Quando sono stati processati tutti gli elementi dell'array, verrà restituito un numero intero rappresentante la velocità massima raggiungibile in quel tratto di strada.

### Dataset MNIST

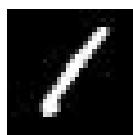
Inizialmente la rete neurale è stata addestrata utilizzando il dataset *MNIST*, una vasta base di dati di cifre scritte a mano comunemente impiegata come insieme di addestramento in vari sistemi per l'elaborazione delle immagini.

I risultati non erano quelli previsti, molto raramente la rete riconosceva correttamente le cifre presenti e nonostante abbia cercato di lavorare sulla trasformazione dell'immagine (applicando algoritmi di super-risoluzione, threshold, dilatazione, contorno, ...) non c'è stato modo di ottenere risultati ottimali.

Di seguito sono mostrati alcuni delle cifre presenti nel dataset.



(a) 0



(b) 1



(c) 2



(d) 3



(e) 9

### Dataset

Terminate le possibili trasformazioni che potevano essere applicate all'immagine in input alla rete neurale ed in seguito a ricerche più approfondite, ho trovato in rete un dataset di cifre stampate e scritte in formato digitale, molto simili a quelle che erano disegnate sui miei limiti di velocità.

Di seguito sono riportate alcune delle immagini provenienti dal dataset.

**0**

(a) 0

**1**

(b) 1

**2**

(c) 2

**3**

(d) 3

**9**

(e) 9

Il dataset utilizzato è composto da 1016 immagini (risoluzione  $32 \times 32 \text{ px}$ ) per ogni cifra (da 0 a 9), suddiviso in due sottocartelle: *train* (formato da 813 elementi, corrisponde al 80%) e *test* (composto da 203 immagini, cioè il rimanente 20%).

## Trasformazioni

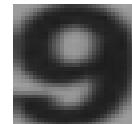
Prima di poter consegnare il dato in input alla rete neurale, è necessario svolgere una serie di trasformazioni sulla porzione di immagine identificata dal contorno rettangolare:



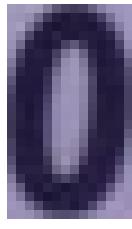
(a) Cifra originale



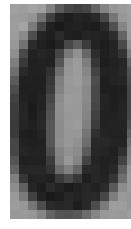
(b) Scala di grigi



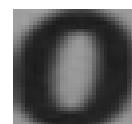
(c) Resize



(a) Cifra originale



(b) Scala di grigi



(c) Resize

Terminate le trasformazioni necessarie, l'immagine viene trasformata in un vettore di tensori e successivamente passata alla rete neurale.

## Addestramento

Le caratteristiche della rete neurale utilizzata sono le seguenti:

**input size = 1024** valore che indica la lunghezza del vettore in input, definito dalle dimensioni delle immagini che formano il dataset ( $32 \times 32 \text{ px}$ )

**hidden layers = 100** le informazioni fornite da una rete neurale sono propagate layer-by-layer dallo strato di input a quello di output attraverso nessuno, uno o più strati nascosti

**number classes = 10** numero di classi in output, che corrispondono dall cifre da 0 a 9

Terminata la fase di addestramento della rete, essa viene salvata tramite una funzione di *PyTorch*, in modo tale da poterla facilmente importare ed utilizzare.

## Output

La rete neurale restituisce come output un vettore tensori di lunghezza 10 ed per ogni posizione di esso viene indicata l'accuratezza della previsione e viene preso il considerazione l'elemento che l'ha maggiore.

Di seguito vengono mostrati gli output della rete durante la fase di riconoscimento dei numeri in Figura 3.16c.

Cifra	Predizione
0	-0.5700780749320984
1	-1.1365952491760254
2	-3.464038848876953
3	-1.893628716468811
4	1.1854283809661865
5	-0.8575473427772522
6	-1.7844761610031128
7	-2.928117275238037
8	-0.09916530549526215
<b>9</b>	<b>1.4027529954910278</b>

**Tabella 3.1:** Predizioni di Figura 3.19a

Cifra	Predizione
0	<b>4.327493190765381</b>
1	-8.269551277160645
2	-0.01426425576210022
3	-3.854306221008301
4	-7.25352668762207
5	-3.59706044197082
6	-2.2099931240081787
7	-2.458707094192505
8	-3.4987590312957764
9	-1.5457154512405396

**Tabella 3.2:** Predizioni di Figura 3.20a

## 3.5.6 Dimostrazione

### Youtube Video

Video applicativo che mostra il funzionamento di quanto descritto in precedenza

### Codice Github

Codice sorgente eseguito su Raspberry per mettere in funzione quanto spiegato fino nella seguente sezione

## 3.6 Evitare Ostacoli

Durante il movimento della macchina, per evitare di urtare contro ostacoli lungo suo il percorso, sono stati montati nella parte anteriore del veicolo dei sensori ad ultrasuoni.

Quando la distanza dagli ostacoli di fronte ad esso è minore di una determinata soglia, il robot inizierà una fase di decelerazione e successivamente lo farà arrestare poco prima dell'oggetto dinnanzi a sé.

## 3.7 Programmazione concorrente

Per implementare i moduli descritti in precedenza, sono stati utilizzati i threads, i quali condividono il frame proveniente dalla camera, e ciascuno di esse svolge le operazioni che gli consentono di raggiungere il proprio obiettivo:

- ottenimento dell'angolo di curvatura
- verifica della presenza del segnale di stop ed eventuale distanza a cui esso si trova
- verifica della presenza del semaforo ed eventuale colore attivo, al suo passaggio
- verifica della presenza di limiti di velocità ed eventuale valore

### 3.7.1 Dimostrazione

#### Youtube Video

Video applicativo che mostra il funzionamento completo

#### Codice Github

Codice sorgente eseguito su Raspberry



# Capitolo 4

## Alexa Skill

### 4.1 Introduzione

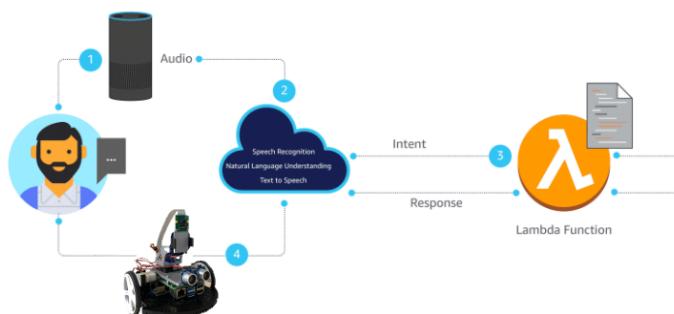
Vista la crescita esponenziale dell'utilizzo di assistenti vocali e virtuali (per citarne alcuni: Siri, Amazon Alexa, Google Assistant, Cortana, ...) nella vita di tutti i giorni, anche le più moderne ed avanzate case automobilistiche stanno implementando le proprie versioni, per supportare e facilitare il guidatore durante la guida.

L'implementazione delle Alexa skills nel progetto *Self Driving Car* cerca di riprodurre, in maniera più semplificata, quanto le grandi aziende automobilistiche stanno implementando attualmente, migliorando l'esperienza utente di chi si trova alla guida.

### 4.2 Skill

L'implementazione del controllo vocale permette all'utente di chiedere le seguenti informazioni:

- la distanza percorsa
- l'ultimo segnale riconosciuto
- la velocità attuale
- il limite di velocità attualmente valido
- la velocità media
- la quantità di batteria rimanente

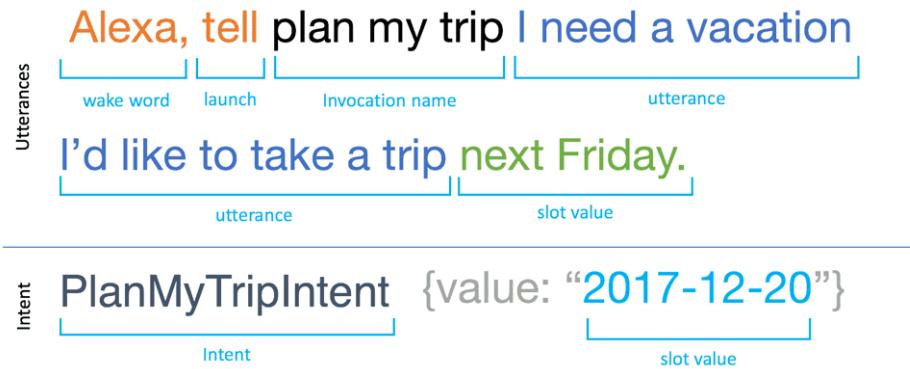


**Figura 4.1:** Workflow skill

#### 4.2.1 Concetti di progettazione

Per creare un'interfaccia utente vocale è necessario comprendere il ruolo dei seguenti:

- **Wake word:** richiede l'attenzione e l'attivazione di Alexa, che da questo momento rimane in ascolto di possibili comandi
- **Launch word:** una parola che indica un'azione, che segnala ad Alexa che probabilmente seguirà un'invocazione di una determinata skill (alcuni esempi sono: chiedi, avvia, inizia, apri, usa, ...).
- **Invocation name:** il nome della skill con la quale utente vuole interagire
- **Utterance:** la richiesta vocale di un utente, che potrebbe richiedere una skill, fornire input, confermare un'azione, ...
- **Prompt:** una stringa di testo che dovrebbe essere pronunciata al cliente per chiedere informazioni
- **Intent:** l'azione che soddisfa la richiesta vocale di un utente; possono facoltativamente avere argomenti chiamati *slot*
- **Slot value:** valori di input forniti nella richiesta vocale di un utente; essi aiutano Alexa a comprendere al meglio l'intento dell'utente



**Figura 4.2:** Concetti

#### 4.2.2 Lambda

Il backend della skill è hostato e gestito da *AWS Lambda*, un'offerta di Amazon Web Services, che esegue il codice solo quando è necessario; l'utente che usufruisce di tale risorsa non è vincolato ad eseguire il provisioning o eseguire continuamente i server.

L'utilizzo di una funzione *Lambda* per il servizio elimina parte della complessità relativa alla configurazione e alla gestione dell'endpoint:

- non è necessario amministrare o gestire nessuna risorsa di elaborazione per il servizio
- non necessita di un certificato SSL
- non è necessario verificare personalmente che le richieste provengano dal servizio Alexa, esso è invece controllato dalle autorizzazioni all'interno di AWS
- Alexa crittografa le sue comunicazioni con *Lambda* utilizzando TLS

## 4.3 Raspberry Pi

Rappresenta il device che effettuerà periodicamente l'upload di dati ed informazioni al shadow.

Per connettersi ad essa, viene utilizzato l'endpoint che rappresenta univocamente l'oggetto stabilito e le chiavi univoche precedentemente scaricate.

## 4.4 AWS IoT Core

### 4.4.1 AWS IoT Device Shadow

Con *AWS IoT Core* è possibile creare versioni virtuali e persistenti dei dispositivi, chiamate *shadow* dei dispositivi (versioni ombra), che includono l'ultimo stato noto del dispositivo, consentendo ad applicazioni e altri dispositivi di leggerne i messaggi e interagire con esso.

La shadow dei dispositivi conserva l'ultimo stato noto e lo stato futuro impostato per ciascun dispositivo anche quando è offline.

La shadow dei dispositivi semplifica la creazione di applicazioni che interagiscono con i dispositivi perché fornisce sempre API REST disponibili. Inoltre, le applicazioni possono impostare uno stato futuro per un dispositivo indipendentemente dal suo stato corrente. *AWS IoT Core* confronterà lo stato futuro con l'ultimo stato noto e imporrà al dispositivo di aggiornarsi.

Dispositivi, altri client Web e servizi possono creare, aggiornare ed eliminare copie shadow utilizzando MQTT e la copia shadow.

#### Shadow

Ogni copia shadow dispone di un argomento MQTT riservato e di un URL HTTP che supporta le operazioni *get*, *update* e *delete* sulla copia shadow.

Le copie shadow utilizzano i documenti *shadow JSON* per archiviare e recuperare i dati. Il documento di una copia shadow contiene una proprietà di stato che descrive questi aspetti dello stato del dispositivo:

**desired** le app specificano gli stati desiderati delle proprietà del dispositivo aggiornando l'oggetto desired

**reported** i dispositivi segnalano il loro stato corrente nell'oggetto reported

**delta** AWS IoT segnala le differenze tra lo stato desiderato e quello riportato nella copia delta

Un esempio è riportato di seguito:

```
{  
  "desired": {  
    "welcome": "aws-iot"  
    "speed": 8  
  },  
  "reported": {  
    "welcome": "aws-iot",  
    "speed": 10  
  }  
}
```

#### 4.4.2 Broker di messaggi

Il broker di messaggi è un broker pub/sub a throughput elevato che trasmette in modo sicuro i messaggi a e da tutti i dispositivi IoT e le applicazioni a bassa latenza. La natura flessibile della struttura topica del broker di messaggi consente di inviare messaggi a, o riceverli da, quanti dispositivi si desidera.

Il broker di messaggi è un servizio interamente gestito, quindi indipendentemente da come si decide di utilizzarlo, dimensiona automaticamente in base al volume di messaggi, senza richiedere la gestione di alcuna infrastruttura.

#### MQTT

*Message Queuing Telemetry Transport*, si tratta di un protocollo pensato per le comunicazioni leggere, ed è di tipo publish - subscribe: per ricevere un messaggio è quindi necessario essere iscritti al topic sul quale esso è stato inviato.

Questo protocollo opera su TCP/IP, garantendo quindi che le comunicazioni arrivino al destinatario esattamente come sono state mandate. Esso prevede due tipi di entità:

- **broker** server a cui i client si connettono che riceve tutti i messaggi ed effettua il routing di questi verso le destinazioni appropriate
- **client** qualsiasi dispositivo che utilizza un software di connessione MQTT e si connette ad un broker

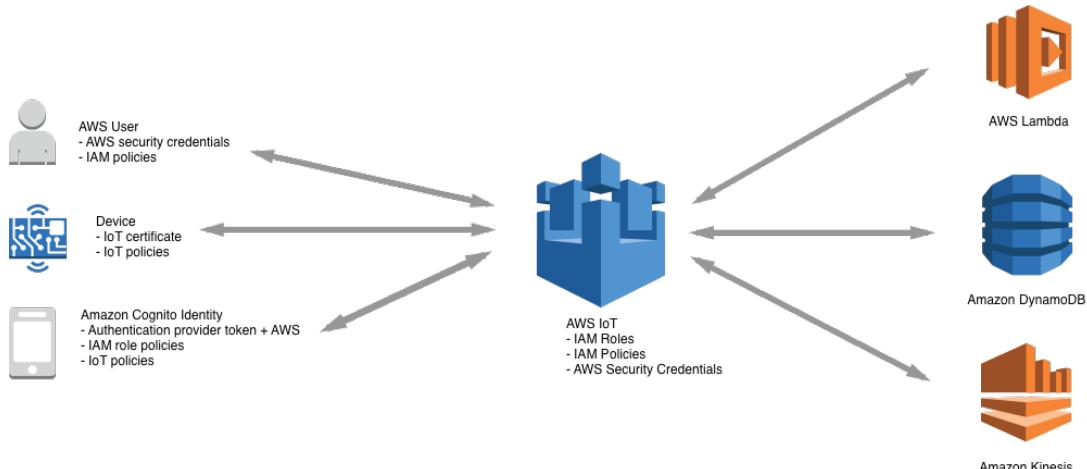
La comunicazione, come citato prima, è organizzata in topic. Quando un client invia delle informazioni su un topic a sua scelta e tutti i client iscritti a questo possono visualizzare il messaggio appena inviato; nel caso su quel topic non vi sia alcun client iscritto, il broker scatterà il messaggio.

I client non interagiscono mai tra di loro ma esclusivamente con il broker.

MQTT, nella maggior parte dei casi, transita le informazioni in chiaro e non include nessuna misura di sicurezza. Nel caso fosse necessario, è possibile utilizzare TLS per criptare e proteggere informazioni trasmesse da eventuali intercettazioni o modifiche.

#### 4.4.3 Sicurezza

Ogni dispositivo o client connesso deve disporre di una credenziale per interagire con AWS IoT. Tutto il traffico da e verso AWS IoT viene inviato in modo sicuro tramite Transport Layer Security (TLS). I sistemi di sicurezza del cloud AWS proteggono i dati durante il trasferimento da AWS IoT ad altri servizi AWS.



**Figura 4.3:** Sicurezza

L'utente è responsabile della gestione delle credenziali del dispositivo (certificati X.509, credenziali AWS, identità federate o token di autenticazione personalizzati) e delle policy in AWS IoT. Egli ha anche la responsabilità dell'assegnazione di identità univoche a ogni dispositivo e della gestione delle autorizzazioni per ogni dispositivo o gruppo di dispositivi.

## TLS

Il broker di messaggi AWS IoT e il servizio Device Shadow crittografano tutta la comunicazione durante il transito utilizzando TLS versione 1.2.

Esso viene usato per garantire la riservatezza dei protocolli applicativi (MQTT, HTTP e WebSocket) supportati da AWS IoT.

- per MQTT, TLS crittografa la connessione tra il dispositivo e il broker
- per HTTP, TLS crittografa la connessione tra il dispositivo e il broker
- l'autenticazione client TLS viene usata da AWS IoT per identificare i dispositivi
- L'autenticazione viene delegata ad AWS Signature Version 4

AWS IoT supporta le suite di crittografia seguenti:

- AES128-SHA256
- AES128-SHA
- AES256-SHA256
- AES256-SHA



# Capitolo 5

## Collegamenti Interdisciplinari

### 5.1 Matematica

Sappiamo che  $s(t)$  è lo spazio percorso all'istante  $t$  da un punto materiale in un momento rettilineo, allora la velocità del punto in quell'istante è la derivata di  $s(t)$ :

$$v(t) = s'(t)$$

Ne segue che:

$$\int_{t_1}^{t_2} v(t) \, dt = \int_{t_1}^{t_2} s'(t) \, dt = s(t_2) - s(t_1) \quad (5.1)$$

Per ciò lo spostamento subito da un punto che si muove su una retta con velocità  $v(t)$  nell'intervallo  $[t_1, t_2]$  è dato dall'integrale:

$$\int_{t_1}^{t_2} v(t) \, dt \quad (5.2)$$

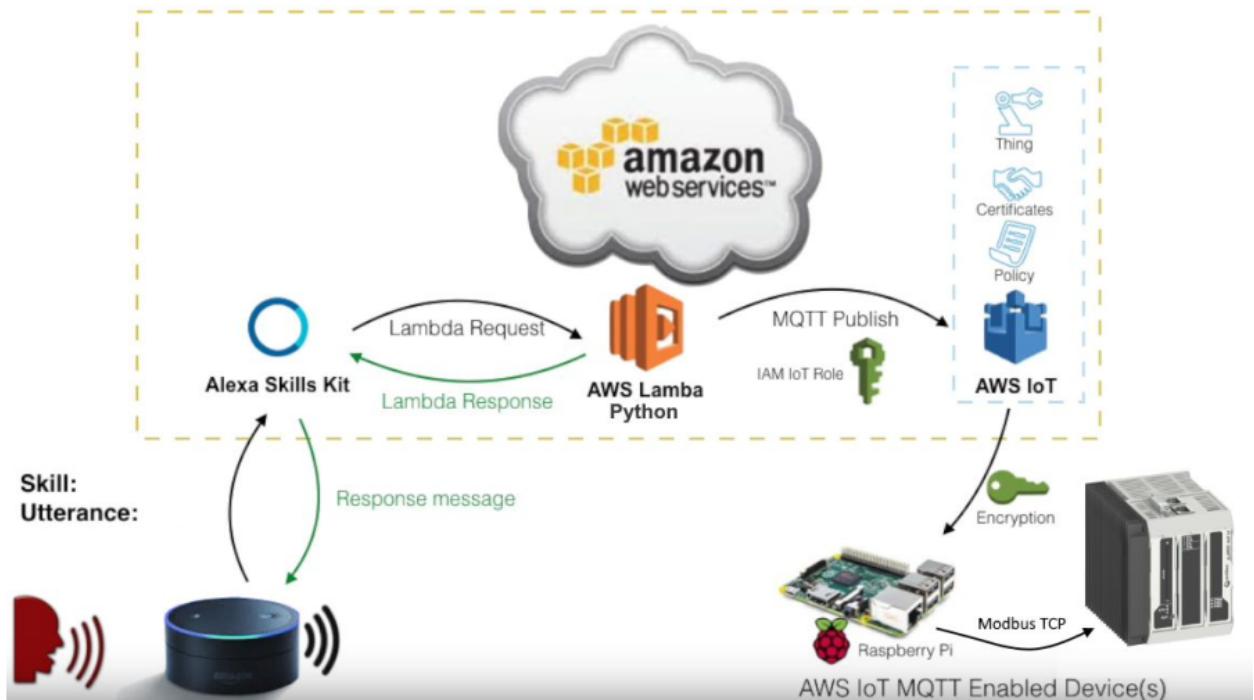
Se  $v(t) \geq 0$ , l'integrale 5.2 esprime lo spazio effettivamente percorso dal punto; in caso contrario, per determinare quest'ultimo bisogna tenere conto delle inversioni di moto, perciò occorre integrare il valore assoluto della funzione velocità, cioè calcolare:

$$\int_{t_1}^{t_2} |v(t)| \, dt \quad (5.3)$$

### 5.2 PCTO

L'implementazione della skill di Alexa (Capitolo: 4) nel progetto *Self Driving Car* è la rivisitazione di quanto svolto durante l'esperienza di alternanza scuola-lavoro svolta nell'estate tra la 3° e la 4° superiore.

In poche parole, durante quell'esperienza lavorativa avevo ideato e sviluppato un sistema che permetteva di controllare le macchine industriali dell'azienda (ordinando di partire, arrestarsi, chiedere la quantità di materiale prodotto, . . .) tramite comandi vocali, implementando le skill di Alexa, come raffigurato di seguito.



**Figura 5.1:** Workflow PCTO

Il diario di bordo è consultabile cliccando il seguente link: <https://github.com/NiccoloSalvi/alexa-pymodbus/blob/master/diarioBordo.pdf>.