Gap Follower Project

Enrico Mannocci - enrico.mannocci3@unibo.it

Dicembre 2023

1 Introduzione

In questo progetto avrai la possibilità di implementare un algoritmo di guida autonoma chiamato **Gap Follower**. Il linguaggio scelto per l'implementazione è Python. Ho preparato già un ambiente in cui potrai testare il tuo algoritmo così da non farti perdere tempo nella grafica e nel modello fisico dell'auto. Il codice che ho scritto è reperibile su GitHub. Supporrò per tutte le mie istruzioni che tu stia usando Linux. Se dovessi avere problemi scrivimi una e-mail a **enrico.mannocci3@unibo.it**.

2 Set up dell'Environment

2.1 Cloniamo la mia Repositori

Per prima cosa dobbiamo installare git (se non lo possiedi ancora), qui troverai il comando per un sacco di distribuzioni linux diverse. Se hai Ubuntu il comando è:

```
sudo apt-get install git
```

Dopo esserti spostato in una posizione che ti aggrada dell'albero dei file esegui:

```
git clone https://github.com/Noce99/gap_follower_project.git
```

Nella posizione attuale ora dovresti avere una cartella chiamata **gap_follower_project**. Controllalo eseguendo un **ls**.

```
cd gap_follower_project
```

Creiamo un virtual environment di python con:

```
python3 -m venv ./
```

Attiviamo il virtual environement con:

```
source bin/activate
```

N.B. Dovrai eseguire questo comando tutte le volte che vorrai lanciare il tuo programma da un nuovo terminale!

Installiamo le dipendenze con:

```
pip install -r requirements.txt
```

E' tutto pronto, prova a lanciare il programma con:

```
python main.py
```

Dovrebbe comparirti la finestra mostrata in Figura 1:

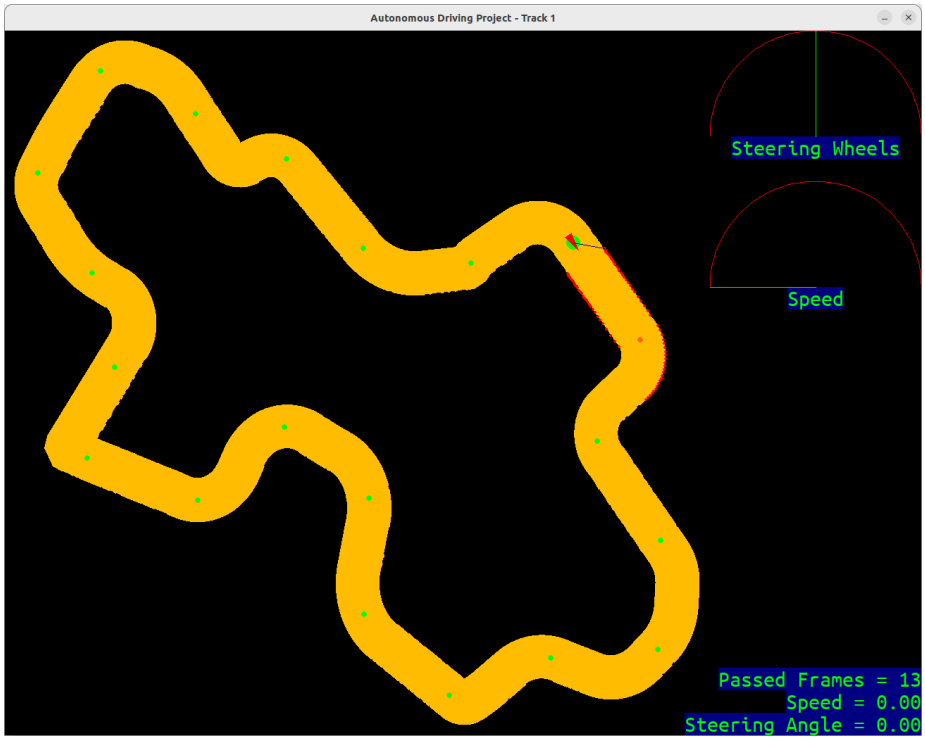


Figure 1: Quando esegui il programma per la prima volta questo è quello che ottieni: una macchina ferma su un tracciato

3 Esploriamo l'algoritmo Gap Follower

3.1 Altre Fonti

Oltre a questo pdf puoi utilizzare come fonte alternativa di informazioni questo video e questa presentazione.

3.2 Che cosa percepisce il veicolo?

Questo algoritmo necessita di un unico sensore: un lidar. Questo sensore restituisce un valore di distanza per ogni direzione attorno al veicolo, una animazione chiarificatrice è disponibile qui. I principali parametri di un lidar che interessano il nostro caso sono tre: la **risoluzione**, la **massima distanza percepibile** e l'**area analizzata**. La **risoluzione** semplicemente è il numero di punti che avremo in output a parità di area osservata, la **massima distanza percepibile** è quella distanza dopo la quale tutti i punti ci appaiono infinitamente distanti e l'**area analizzata** identifica quale e quanta porzione dell'area tutta attorno a noi stiamo analizzando.

Come puoi facilmente intuire un lidar con alta risoluzione, grande distanza massima percepibile e capace di analizzare tutta l'area che ci circonda, è molto costoso. Un lidar con piccola risoluzione, distanza massima e area analizzata risulta invece molto più economico. Analizzare correttamente quanto questi parametri debbano essere elevati per un particolare task ci permette di risparmiare riuscendo a risolvere il nostro problema. In particolare più avanti ti mostrerò come terremo conto di ciò nella nostra simulazione.

Il nostro Gap Follower prenderà dunque in input **solo** le distanze ottenute dal lidar e si baserà **solo** su queste per decidere quale sia la direzione verso la quale è preferibile muoversi!

3.3 Visualizziamo l'input

Per chiarirci le idee visualizziamo che cosa otterrà il nostro algoritmo in input. Supponiamo che il nostro lidar abbia una risoluzione di 20° e come massima distanza 5 m; se l'area analizzata fosse tutta l'area che ci circonda (360°) avremmo come input una lista composta da 18 numeri compresi tra 0 e 5 dove con 5 indichiamo anche tutte le distanze maggiori di 5 m.

$$[2.0 \;\; 1.5 \;\; 3.2 \;\; 4.5 \;\; 0.5 \;\; 3.0 \;\; 4.0 \;\; 4.5 \;\; 5.0 \;\; 4.7 \;\; 4.3 \;\; 4.0 \;\; 3.0 \;\; 0.5 \;\; 0.3 \;\; 1.5 \;\; 5.0 \;\; 4.0] \tag{1}$$

3.4 Che cos'è un Gap?

Definiamo un gap di n_g elementi e valore minimo t_g come una serie di almeno n_g numeri maggiori o uguali a t_g. Quindi prendendo come esempio l'input mostrato in 1 evidenzio di seguito in rosso i gap trovati considerando $n_g = 3$ e $t_g = 3.5$.

$$[3.8 \;\; 1.5 \;\; 3.2 \;\; 4.5 \;\; 4.0 \;\; 3.0 \;\; 4.0 \;\; 4.5 \;\; 5.0 \;\; 4.7 \;\; 4.3 \;\; 4.0 \;\; 3.0 \;\; 0.5 \;\; 0.3 \;\; 1.5 \;\; 5.0 \;\; 4.0] \tag{2}$$

Da notare come, essendo che tutto lo spettro attorno all'auto viene analizzato, è necessario considerare gli ultimi due valori e il primo come un gruppo di valori adiacenti.

In blu ho invece riportato un gruppo di valori adiacenti maggiori di $t_g = 3.5$ ma non sufficientemente numeroso: 2 elementi a discapito dei 3 minimi richiesti ($n_g = 3$).

Ci tengo a sottolineare come questi due parametri modifichino completamente i gap trovati e di come sia possibile raggiungere i due casi limite di nessun gap trovato o di solo un unico gap grande come tutta la lista semplicemente avendo nel primo caso $n_g = 7$ e nel secondo $t_g = 0.3$.

3.5 Come usare i Gap?

Data una lista di valori e due parametri n_g e t_g siamo ora in grado di identificare i gap ma come questo ci avvicini all'obiettivo di un agente a guida autonoma è una domanda più che lecita a cui proveremo con quanto segue a rispondere.

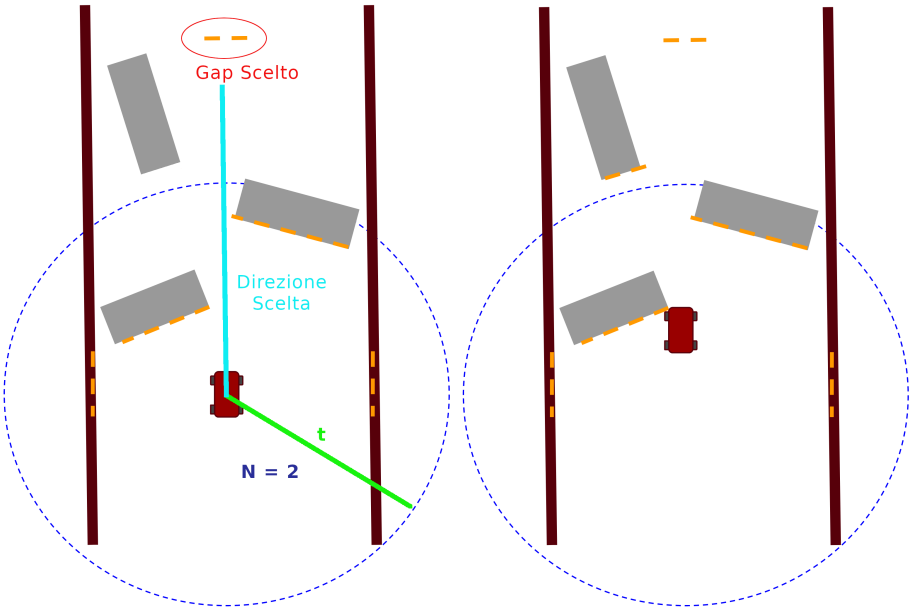
L'idea alla base è quella che vogliamo muoverci verso la direzione con il minor numero di ostacoli e quindi in cui il nostro lidar riesce a "vedere" lontano. La soluzione più facile a questo punto è quindi scegliere il gap più grande (composto da più elementi) e all'interno di quest'ultimo scegliere la direzione che corrisponde alla distanza maggiore. In poche parole nell'esempio visto prima sceglieremmo il seguente gap e la seguente direzione.

$$[3.8 \;\; 1.5 \;\; 3.2 \;\; 4.5 \;\; 4.0 \;\; 3.0 \;\; 4.0 \;\; 4.5 \;\; 5.0 \;\; 4.7 \;\; 4.3 \;\; 4.0 \;\; 3.0 \;\; 0.5 \;\; 0.3 \;\; 1.5 \;\; 5.0 \;\; 4.0] \tag{3}$$

Abbiamo quindi trovato un modo per identificare una direzione verso la quale muoverci.

3.6 Qualche problema?

Se provi ad implementare l'algoritmo fino a questo punto vedrai che funziona ma con qualche limite e che capita, in prossimità di curve strette o di un ostacolo, che si esca dalla pista o si abbia un incidente. Per visualizzare meglio il problema riporto Figura 2.



(a) Selezione dei Gap (b) La macchina urta un ostacolo

Figure 2: Le due figure mostrano perché l'algoritmo sviluppato fino ad ora risulta debole e può causare incidenti

3.7 Risolviamo con delle Bolle

La soluzione al problema mostrato in Figura 2 è creare delle cosiddette **Bolle di Sicurezza** attorno ai punti critici. Definendo come punto critico un punto ad una distanza inferiore ad un certo valore t_b possiamo identificarli nel consueto esempio come segue supponendo $t_b = 1.0$.

$$[2.0 \;\; 1.5 \;\; 3.2 \;\; 4.5 \;\; 0.5 \;\; 3.0 \;\; 4.0 \;\; 4.5 \;\; 5.0 \;\; 4.7 \;\; 4.3 \;\; 4.0 \;\; 3.0 \;\; 0.5 \;\; 0.3 \;\; 1.5 \;\; 5.0 \;\; 4.0] \tag{4}$$

Ora possiamo creare le nostre bolle di sicurezza settando a zero tutti i valori adiacenti ai valori critici. Anche qui definiamo un parametro r_b come il "raggio" della bolla. Visualizziamo quanto detto con $r_b = 2$.

$$[2.0 \;\; 1.5 \;\; 0.0 \;\; 0.0 \;\; 0.5 \;\; 0.0 \;\; 0.0 \;\; 4.5 \;\; 5.0 \;\; 4.7 \;\; 4.3 \;\; 0.0 \;\; 0.0 \;\; 0.5 \;\; 0.3 \;\; 0.0 \;\; 0.0 \;\; 4.0] \tag{5}$$

Praticamente stiamo escludendo da una possibile selezione tutti i punti (magari anche con valori elevati di distanza) adiacenti a punti con distanze critiche.

3.8 Conclusioni

L'algoritmo così descritto ci permette, dati i valori di un lidar, di selezionare una direzione verso la quale si vuole andare. Questioni dinamiche quali come muoversi verso una direzione molto lontana da quella corrente o a che velocità procedere rimangono ancora insolute.

4 Implementazione dell'Algoritmo

Per implementare l'algoritmo l'unico file python da modificare è: **gap_follower.py**. Appena scaricata la repositori il file si presenta come segue.

```python
import math
import utils
import numpy as np
from typing import Callable

# MODIFY FROM HERE ...
"""
You can modify the following parameters for having a better or a worst Lidar!
Remember that you need to minimze the cost!
"""
utils.LIDAR_MAX_DISTANCE = 300 # A float between 0.0 and 500.0
utils.LIDAR_NUMBER_OF_RAY = 100 # An interge between 0 and 500
utils.LIDAR_ANGLE = math.pi/2 # A float (angle in !RADIANTS!) between 0.0 and PI
utils.QUANTITY_OF_NOISED_RAY = 0 # Index of [0 %, 5 %, 10 %, 20 %] so an integer
                                    between 0 and 3
utils.ERROR_SIZE = 1 # Index of [0, 10, 20, 30] so an integer between 0 and 3
# TO HERE!

utils.check_lidar_parameters()


def gap_follower(distances: list[np.float64]) -> tuple[tuple[int, int], int, Callable]:
    """
    This function gets the lidar's information and return the

    Parameters:
        distances (list[np.float64]): List of floats that are the distances from the
                                      car asked to the lidar
                                      len(distances) = utils.LIDAR_NUMBER_OF_RAY


    Returns:
        choosen_gap (tuple[int, int]): The starting index in the 'distances' list that
                                       identify the gap that you have choosen and its
                                       lenght.
                                       N.B. This is used just for visualizatio!
        choosen_direction (int): The index of the 'distances' list that identify
                                 the wanted direction!
        control_the_vehicle (function): The function called for control the acelleration
                                        and steering angle of the vehicle.
    """
    choosen_gap = None
    choosen_direction = 0
    # MODIFY FROM HERE ...
```

```
    # TO HERE!
    def control_the_vehicle(orientation: float, steering_angle: float, speed: float,
                            wanted_direction: float) -> (bool, bool):
        """
        This function gets the vehicle's informations (orientation, steering_angle
        and speed) and the wanted direction calculated in 'gap_follower'
        function and returns the basic commands to control the
        car (acelleration/decelleration and steering).

        Parameters:
            orientation (float): The orientation respect to Nord of the vehicle in radiants
            steering_angle (float): The actual steering angle of the vehicle in radiants
            speed (float): The actual speed of the vehicle in m/s
            wanted_direction (float): The wanted orietation respect to Nord in radiants

        Returns:
            acelleration (bool):
                                    True -> Increase a little bit Speed
                                    False -> Decrease a little bit Speed
                                    None -> Do not modify Speed
            steering (bochoosen_gapol):
                                    True -> Turn steering a little bit to the Left
                                    False -> Turn steering a little bit to the Right
                                    None -> Do not modify Steering
        """
        acceleration = None
        steering = None
        # MODIFY FROM HERE ...


        # TO HERE!
        return acceleration, steering
    return choosen_gap, choosen_direction, control_the_vehicle
```

Per fare il progetto devi completare le due funzioni, affinché il veicolo (la freccia rossa visibile in Figura 1) riesca a completare un giro del circuito in completa autonomia (in tutti i 4 tracciati). In ciò che segue proverò a darti una mano.

4.1 La funzione gap_follower

In questa funzione dovrai implementare l'algoritmo descritto nella sezione precedente (Sezione 3). Come ci aspettiamo l'unico input è una lista di distanze. Gli output che si aspetta la funzione sono 2:

1. gap

2. choosen_direction
3. control_the_vehicle

Il primo ha solo un fine estetico e di debug in quanto ti permetterà di visualizzare il gap scelto durante l'esecuzione del tuo programma mentre il secondo è l'indice dell'elemento di **distances** verso il quale vuoi muoverti. In particolare **gap** è una tupla di interi contenente l'indice del primo elemento del gap e la sua lunghezza. Infine **control_the_vehicle** è la seconda funzione che devi implementare. Ho già scritto io il return nel modo corretto e non devi preoccupartene. Vediamo subito un esempio per chiarire, supponendo di avere la seguente lista di distanze (*distances*), i seguenti gap e la seguente direzione scelta. Riporto di seguito per comodità gli indici.

$$[3.8 \quad 1.5 \quad 3.2 \quad 4.5 \quad 4.0 \quad 3.0 \quad 4.0 \quad 4.5 \quad 5.0 \quad 4.7 \quad 4.3 \quad 4.0 \quad 3.0 \quad 0.5 \quad 0.3 \quad 1.5 \quad 5.0 \quad 4.0] \tag{6}$$

$$[00 \quad 01 \quad 02 \quad 03 \quad 04 \quad 05 \quad 06 \quad 07 \quad 08 \quad 09 \quad 10 \quad 11 \quad 12 \quad 13 \quad 14 \quad 15 \quad 16 \quad 17] \tag{7}$$

Avremo che:

$$\text{gap} = (6; 6) \tag{8}$$

$$\text{choosen\_direction} = 8 \tag{9}$$

N.B. La variabile **choosen_direction** deve sempre essere un valore diverso da **None** altrimenti la simulazione si ferma.

4.2 La funzione control_the_vehicle

Questa funzione prende in input quattro variabili:

1. orientation
2. steering_angle
3. speed
4. wanted_direction

Le variabili **orientation** e **wanted_direction** sono gli angoli **!IN RADIANTI!** rispetto a Nord che identificano l'orientamento attuale della macchina e quello che si vorrebbe raggiungere. Entrambi gli angoli si possono visualizzare in Figura 3 rispettivamente in magenta e verde. Per entrambe queste variabili ci si aspetta valori compresi tra 0 e $2\pi \approx 6.28$.

La variabile **speed** contiene la velocità attuale in $\frac{m}{s}$.

La variabile **steering_angle** contiene l'angolo **!IN RADIANTI!** di cui sono attualmente inclinate le ruote anteriori. Questo angolo può essere visualizzato in Figura 3 in azzurro e ci si aspettano valori compresi tra $-\frac{\pi}{4} \approx -0.79$ e $\frac{\pi}{4} \approx 0.79$.

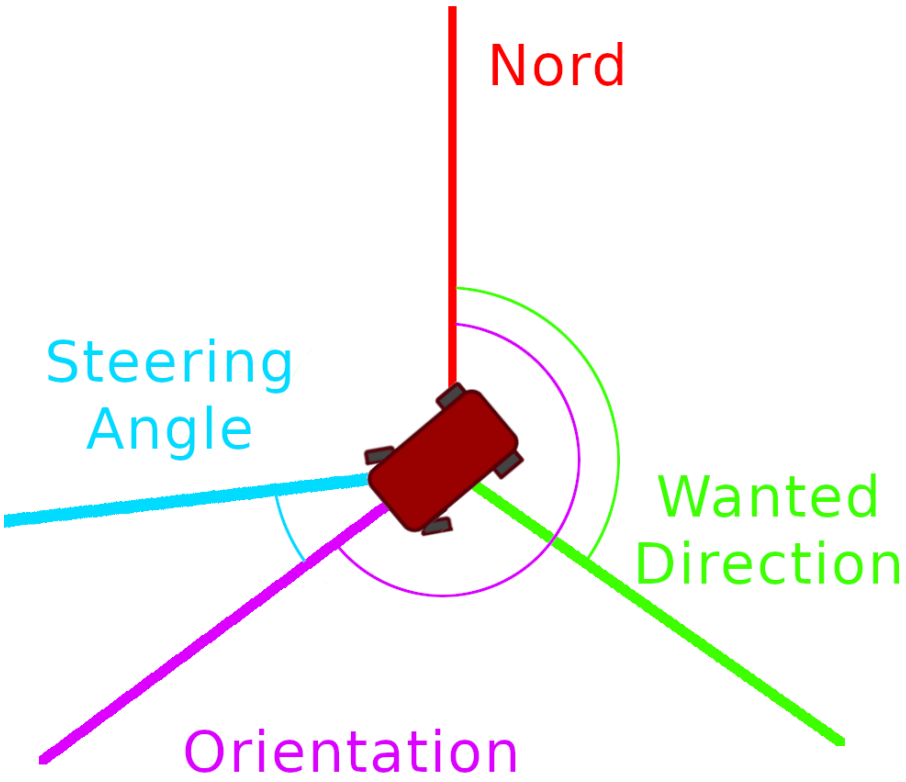


Figure 3: Variabili ottenute come input della funzione **control_the_vehicle**

Ad ogni iterazione puoi modificare contemporaneamente sia lo sterzo che l'acceleratore/freno. Per entrambi devi quindi scegliere una possibile azione assegnando una valore alle due variabili **acceleration** e **steering**; le azioni e i valori corrispondenti sono riportati rispettivamente in Tabella 1 e 2.

True	Aumenta leggermente la Velocità
False	Diminuisce leggermente la Velocità
None	Lascia la velocità attuale Invariata

Table 1: Tabella che rappresenta le possibili azioni riguardanti l'acceleratore/freno

True	Ruota lo sterzo leggermente a Sinistra
False	Ruota lo sterzo leggermente a Destra
None	Lascia lo sterzo Invariato

Table 2: Tabella che rappresenta le possibili azioni riguardanti lo sterzo

!ATTENZIONE! Assegnare None alla variabile **steering** non significa andare dritto ma, non modificare lo **steering_angle**. Per andare dritti bisogna che lo **steering_angle** sia 0.

4.2.1 Qualche Consiglio sulla Velocità

Per i primi test ti consiglio di mantenere una velocità costante, potrai ottimizzarla in un secondo momento. Potresti ad esempio accelerare fino a che non hai raggiunto la velocità voluta e poi semplicemente non toccare più acceleratore e freno per tutto il resto della simulazione.

Per finalizzare il modello invece ti consiglio di prendere in considerazione lo spettro di distanze che hai di fronte: se hai molto spazio significa che non ci sono ostacoli e puoi andare più veloce altrimenti è meglio rallentare.

4.2.2 Qualche Consiglio sullo Sterzo

Nel caso in cui, come in Figura 3, la direzione cercata è molto lontana da quella attuale non pretendere di poterla correggere in un'unica interazione. Semplicemente avvicinati ad essa.

4.3 Parametri del Lidar

Oltre all'efficacia del tuo algoritmo considera anche quanto di qualità (e quindi costoso) deve essere il lidar per farlo funzionare correttamente. I parametri che puoi modificare sono riportati all'inizio di **gap_follower.py** e sono:

Nome nel programma	Descrizione	Intervallo Accettato	Tipo	Nome
LIDAR_MAX_DISTANCE	La massima distanza percepibile dal sensore	$[0; 500]$	float	A
LIDAR_NUMBER_OF_RAY	Il numero di raggi emessi dal sensore (infulisce sulla sua risoluzione)	$[0; 500]$	int	B
LIDAR_ANGLE	La grandezza dell'angolo (centrato verso il fronte della macchina) entro il quale vengono sparati i raggi	$[0; \pi]$	float	C
QUANTITY_OF_NOISED_RAY	Quantità in percentuale di raggi con rumore	$[0; 3]$	int	D
ERROR_SIZE	Grandezza del rumore	$[0; 3]$	int	E

Table 3: Tabella che rappresenta i parametri del lidar modificabili

N.B. Gli ultimi due sono gli indici a due liste che riporto sotto.

$$[0\%, 5\%, 10\%, 20\%] \tag{10}$$

$$[0\ m, 10\ m, 20\ m, 30\ m] \tag{11}$$

L'equazione utilizzata per calcolare il costo finale del sensore (*cost*) è:

$$cost = \frac{1}{5}\left(\frac{A}{500} + \frac{B}{500} + \frac{C}{\pi} + \frac{3-D}{3} + \frac{3-E}{3}\right) \tag{12}$$

5 Tracciati

Sono presenti vari tracciati nella cartella chiamata 'tracks'. Quelli con nome inferiore a 100 sono tracciati senza ostacoli quelli con nome superiore a 100 li possiedono. Per selezionare su quali valutare il proprio algoritmo modificare il dizionario 'TRACKS' presente nelle prime righe di **gap_follower.py**.

6 Valutazione

Per valutare il tuo risultato puoi utilizzare la variabile EVALUATION che viene scritto sul terminale a seguito del completamento di un giro di tracciato. Per completezza ecco come viene calcolata:

$$\text{EVALUATION} = \frac{\text{NUM\_OF\_FRAMES}}{10000} \cdot cost \tag{13}$$

Di seguito riporto un esempio di output:

```
Starting Track 1
Race finished in 2804 frames!
Lidar Cost = 0.67
Evaluation = 0.19
Closing Track 1
Starting Track 101
Race finished in 3204 frames!
Lidar Cost = 0.67
Evaluation = 0.21
Closing Track 101
Bye Bye!
```