Global Planner

Cristiano Parenti
340338@studenti.unimore.it

Giacomo Salici
270385@studenti.unimore.it

Indice

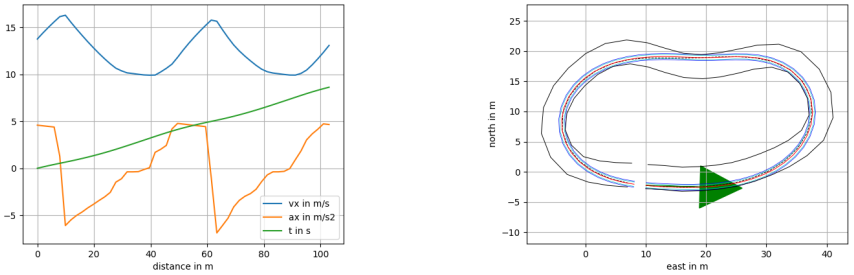


Figura 1: Profili di velocità e traiettoria ottimale del classico circuito “a fagiolo”.

1 Introduzione

L’obiettivo del *global planner* è generare un percorso e un profilo di velocità (e successivamente di curvatura), che ottimizzi la traiettoria della macchina.

L’input dell’algoritmo prevede di avere un array di dimensione $(n, 4)$, contenente $[x_i, y_i, w_i^r, w_i^l]$ per ognuno degli n valori dell’array.

In particolare,

- n il numero dei valori campionati.
- x_i, y_i le coordinate di ciascun punto della centerline.
- w_i^r, w_i^l la distanza fra la centerline e i boundaries rispettivamente di destra e sinistra.

Tutti questi valori sono ereditati dalla fase di local planning e ascoltati attraverso i rispettivi topic ROS. Per avere un algoritmo efficace la centerline deve essere il più smooth possibile. Per questo potrebbe essere necessaria una fase di preprocessing per ricampionare e ottimizzare la centerline.

2 Minimizzazione della curvatura

La ricerca del minimo della curvatura della traiettoria non è esattamente equivalente alla ricerca della traiettoria ottima; spesso tuttavia il percorso ottimale è un compromesso tra la curvatura minima e il percorso minimo. Tuttavia per quello che ci riguarda va bene anche partire da un problema di questo tipo, principalmente perché in questo modo è massimizzabile la velocità di entrata in curva (e inoltre è anche computazionalmente più semplice). Il tutto risulterà in un problema di ottimizzazione quadratica (QP) risolvibile in maniera semplice e robusta. Il lavoro è tratto da [1].

2.1 Formulazione del problema QP

Si definiscono i punti della centerline come $p_i = (x_i, y_i)$, con $i = 1, \ldots, n$ e i punti della raceline variati rispetto a quelli della centerline come $r_i = p_i + \alpha_i n_i$ (n vettore normale alla centerline). Allora l'ottimizzazione agisce esattamente sui parametri α_i, che ovviamente sono vincolati ad esprimere un punto interno alla pista.

Useremo spline cubiche, per avere a disposizione derivate continue fino alla seconda.

Vogliamo dunque minimizzare la norma L2 della curvatura: $\min \sum_i \kappa_i^2$. Attraverso alcune manipolazioni arriviamo all'espressione $\min x''^T P_{xx} + y''^T P_{xy} x'' + y''^T P_{yy} y''$, dove x'' e y'' sono i vettori delle seconde derivate di x e y rispettivamente, e P_{xx}, P_{xy} e P_{yy} sono matrici definite positive che dipendono dai vettori x' e y'.

Si assume che le matrici P_{xx}, P_{xy} e P_{yy} siano costanti lungo la pista. Questo è vero se la pista è sufficientemente smooth, e in ogni caso è un'assunzione che si può fare senza perdere troppa generalità. Il problema è quando ci sono delle curve: per questo l'algoritmo prevede delle iterazioni (vedi dopo).

Insomma alla fin dei conti utilizzando delle altre proprietà delle spline si definiscono le matrici $T_{n,x}$ e con questa le matrici $H_x, H_y, H_{xy}, f_x, f_{xy}, fy$ e dunque il problema di minimizzazione diventa

$$\min \alpha^T H \alpha + f^T \alpha$$
$$\text{soggetto a: } \alpha_i \text{ deve stare dentro la raceline}$$

Questo è uno standard QP problem.

2.2 Regolarizzazione della centerline

La centerline deve essere smooth, altrimenti il problema QP risulta troppo rumorosa. Vengono quindi effettuati 4 passaggi:

- interpolazione lineare della centerline con uno small step size;
- approssimazione con spline;
- correzione della track width dovuta alla spline;
- interpolazione con spline con uno step size più grande (vengono suggeriti 3 metri, bisogna vedere quanto è lungo il circuito).

2.3 Curvature costraint

Se la macchina ha un raggio di curvatura massimo che non deve essere superato si deve fare in modo che la somma tra la curvatura della centerline (presente per forza) e quella introdotta dall'ottimizzazione non superi il raggio di curvatura massimo. Dopo aver definito un altro paio di matrici e aver completato qualche passaggio algebrico si ottiene un ulteriore vincolo del tipo $E\alpha \leq k$.

2.4 Loops

Abbiamo detto di come per semplificare il problema si considerano le matrici P, che contengono le derivate dei punti della centerline, costanti lungo il percorso. Questo ovviamente non vale quando ci sono le curve; questo porta a soluzioni non ottimali. Viene allora implementato un loop, utilizzando ad ogni iterazione la soluzione trovata al passo precedente. Tuttavia, per mitigare l'inaccuratezza delle prime soluzione calcolate, si moltiplica il vettore α per $\frac{1}{3}$ alla prima iterazione e per $\frac{2}{3}$ alla seconda. Naturalmente all'inizio di ogni iterazione è necessario effettuare i 4 passaggi descritti precedentemente per mettere a posto la nuova centerline e per ricalcolare i punti distanti lo step size selezionato. Viene poi definita una tolleranza legata alla curvatura come criterio di arresto del ciclo (si propone ad esempio $\triangle\kappa = 0.005$ rad/m).

3 Generazione del profilo di velocità

Il profilo della velocità è generato tramite un *diagramma* `ggV`, cioè un grafico che mette in relazione l'accelerazione longitudinale e laterale massima in corrispondenza di certi valori della velocità. Il profilo della velocità è poi calcolato in tre passaggi:

1. Calcolo della velocità massima in ogni punto considerando l'accelerazione tutta laterale: $V_1(s) = \sqrt{\frac{\mu g}{k(s}}$, dove μ è il coefficiente di resistenza aerodinamica e $k(s)$ è la curvatura in ogni punto.

2. Forward solver: calcolo della velocità in ogni punto considerato il potenziale di *accelerazione* della macchina e la posizione precedente:

   $$V_2(s + \delta s) = \sqrt{V_2(s)^2 + 2a_{x,max}\delta s}.$$

   Viene poi scelto il minimo fra il risultato e quanto ottenuto in precedenza.

3. Backward solver: calcolo della velocità in ogni punto considerato il potenziale di *frenata* della macchina e la posizione successiva:

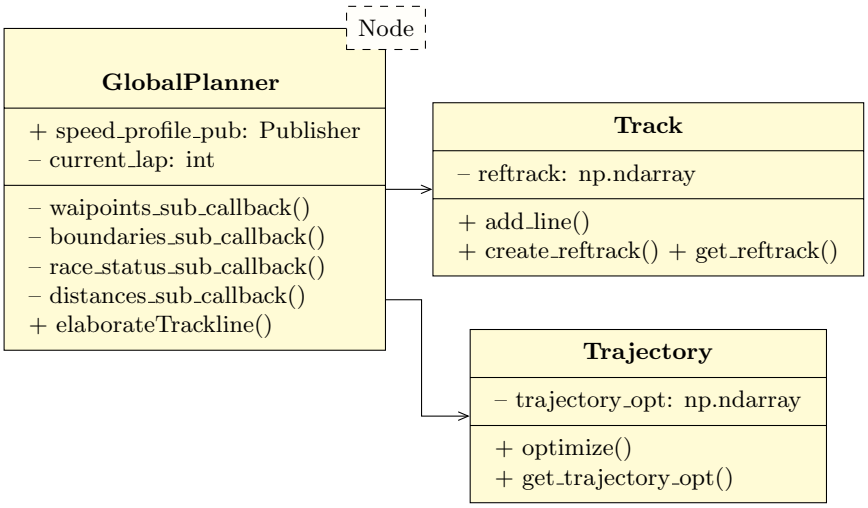
   $$V_2(s - \delta s) = \sqrt{V_2(s)^2 - 2a_{x,max}\delta s}.$$

   Viene poi scelto il minimo fra il risultato e quanto ottenuto in precedenza.

In questo modo i due profili di velocità vengono come intersecati, considerando quindi al meglio il potenziale delle ruote.

4 Codice

Il codice è composto da un nodo ROS2 Foxy che parte insieme al resto della architettura. Il diagramma seguente mostra i principali metodi e attributi delle tre classi coinvolte.



4.1 Architettura

Al momento dell'inizializzazione del nodo/classe `GlobalPlanner`, vengono inizializzati anche tutti i vari *subscribers* che si mettono in ascolto dei topic coinvolti. Dopo la conclusione del primo giro, il *Local Planner*[1] ha terminato di campionare la posizione dei coni a bordo della pista nonché la posizione della macchina durante il primo giro (la cosiddetta *centerline* che non per forza è al centro della corsia).

A questo punto verranno publicati sui topic `/planning/waypoints_all`, `/planning/boundaries_all` e `/planning/distances_all` rispettivamente i punti della *centerline*, i punti dei due bordi e le distanze fra ciascun bordo e la *centerline* per ogni punto di quest'ultima. Tutti e tre sono rappresentati come array e vengono passati alla classe `Track` che li elabora e crea l'array da qui in poi chiamato *reftrack* per distiguerlo dalla *centerline*. Il formato $(n, 4)$, contenente $[x_i, y_i, w_i^r, w_i^l]$ è già stato descritto nella sezione (1).

Successivamente all'interno della classe `GlobalPlanner` viene invocato il metodo `optimize()` della classe `Trajectory` che incomincia ad eseguire il procedimento iterattivo di ottimizzazione della traiettoria. Una volta che questo è concluso, il nodo può pubblicare i risultati.

Il codice della parte di ottimizzazione è stato preso direttamente dalla repository ufficiale del TUM[2]. L'output, che viene anche salvato in un file a parte, è anch'esso un array ndimensionale di formato $(n, 7)$, dove le 7 colonne rappresentano:

0. `s_m`: `float32`, metri. Distanza curvi-lineare lungo la linea di percorrenza.

1. `x_m`: `float32`, metri. Coordinata X del punto della linea di percorrenza.

[1] Il local planner rappresenta il punto precedente della nostra architettura
[2] https://github.com/TUMFTM/global_racetrajectory_optimization

2. `y_m`: `float32`, metri. Coordinata Y del punto della linea di percorrenza.

3. `psi_rad`: `float32`, rad. Direzione della linea di percorrenza nel punto corrente da $-\pi$ a $+\pi$ radianti. Zero è nord (lungo l'asse y).

4. `kappa_radpm`: `float32`, rad/metro. Curvatura della linea di percorrenza nel punto corrente.

5. `vx_mps`: `float32`, metro/secondo. Velocità target nel punto corrente.

6. `ax_mps2`: `float32`, metro/secondo2. Accelerazione target nel punto corrente. Assumiamo che questa accelerazione sia costante dal punto corrente fino al punto successivo.

4.2 Output

L'array della *reftrack* viene pubblicato dal nodo Global Planner sullo specifico topic `/planning/speedProfilePoints` usando il formato custom *SpeedProfilePoints.msg* definito come array di punti *SpeedProfilePoint.msg* anch'essi custom.

```
SpeedProfilePoint[] points
```

Ciascun *SpeedProfilePoint.msg* è definito come segue e contiene, per ogni riga della *reftrack*, le coordinate della traiettoria ottimizzata come geometry point e il valore della velocità e dell'accelerazione come ackermann point.

```
geometry_msgs/Point point
ackermann_msgs/AckermannDrive ackerman_point
```

4.3 Parametri

Nella cartella `input` si può modificare il file `racecar.ini` per modificare i parametri dell'automobile.

- File diagrammi di accelerazione (anch'essi nella cartella `input`)
  - `ggv_file`: Nome del file del diagramma GGV
  - `ax_max_machines_file`: Nome del file `ax_max_machines` da utilizzare
- Opzioni per la dimensione del passo:
  - `stepsize_prep`: [m] utilizzato per l'interpolazione lineare prima dell'approssimazione dello spline
  - `stepsize_reg`: [m] utilizzato per l'interpolazione dello spline dopo l'approssimazione dello spline (passo durante l'ottimizzazione)
  - `stepsize_interp_after_opt`: [m] utilizzato per l'interpolazione dello spline dopo l'ottimizzazione

- Opzioni per la smooth regression dello spline:
  - `k_reg`: [-] ordine dei B-Splines (standard: 3)
  - `s_reg`: [-] fattore di smoothing, intervallo [1.0, 100.0]
- Parametri generali del veicolo richiesti in diverse funzioni:
  - `v_max`: [m/s] velocità massima del veicolo
  - `length`: [m] lunghezza del veicolo
  - `width`: [m] larghezza del veicolo
  - `mass`: [kg] massa del veicolo
  - `dragcoeff`: [kg*m2/m3] coefficiente di resistenza calcolato come $0.5 * \rho_{\text{air}} * c_w * A_{\text{front}}$
  - `curvlim`: [rad/m] limite di curvatura del veicolo
  - `g`: [N/kg] accelerazione di gravità
- Opzioni per il calcolo del profilo di velocità:
  - `dyn_model_exp`: [-] esponente utilizzato nel modello dinamico del veicolo (intervallo [1.0, 2.0])
  - `vel_profile_conv_filt_window`: [-] dimensione della finestra del filtro di media mobile per il profilo di velocità (imposta null se non utilizzato)
- Opzioni del problema di ottimizzazione (ottimizzazione della curvatura minima):
  - `width_opt`: [m] larghezza del veicolo per l'ottimizzazione includendo la distanza di sicurezza
  - `iqp_iters_min`: [-] numero minimo di iterazioni per l'IQP
  - `iqp_curverror_allowed`: [rad/m] massima curvatura consentita per l'IQP

Infine, all'interno del file `global_trajectory.py` si possono modificare manualmente diverse opzioni, fra cui quali grafici devono essere visualizzati in fase di debug (esempi in figura 1).

Riferimenti bibliografici

[1] Alexander Heilmeier et al. "Minimum curvature trajectory planning and control for an autonomous race car". In: *Vehicle System Dynamics* 58.10 (giu. 2019), pp. 1497–1527. ISSN: 1744-5159. DOI: 10.1080/00423114.2019.1631455. URL: http://dx.doi.org/10.1080/00423114.2019.1631455.