Relazione Progetto: Controllore per Veicolo Driverless

Filippo Cattazzo

14 novembre 2025

Indice

1 Introduzione e Setup del Progetto

L'obiettivo di questo progetto è l'implementazione di un controllore a retroazione (cioè che guarda, ragiona e corregge costantemente) per un veicolo Formula Student Driverless. Il progetto è stato sviluppato in C++ e ha richiesto la configurazione di un ambiente di sviluppo basato su CMake.

1.1 Configurazione Ambiente

Trattandosi della mia prima esperienza con molti di questi strumenti, la fase di setup iniziale è stata fondamentale.

Apprendimento di CMake Non avendo mai usato CMake prima, il primo passo è stato comprendere la sua funzione. Grazie a documentazione online e tutorial video [1], ho capito che CMake non è un compilatore, ma un **gestore di build**: si scrive un unico piano di costruzione (il file `CMakeLists.txt`) che definisce sorgenti e dipendenze (come `raylib`). CMake usa questo piano per generare i file di progetto corretti per qualsiasi ambiente, rendendo il progetto portabile.

Version Control con Git Parallelamente, ho configurato una repository Git su GitHub. Anche questa è stata una mia prima esperienza, approfondita tramite guide video [2]. Ho quindi utilizzato Git per documentare lo sviluppo passo dopo passo tramite commit.

Struttura del Progetto È stata adottata una struttura di cartelle standard, separando gli *header file* (`include/`) dai file sorgente (`src/`). Questa separazione è cruciale per la leggibilità, la gestione della compilazione e l'astrazione (separando interfaccia `.h` e implementazione `.cpp`).

2 Livello 1: Modello Cinematico e Visualizzazione

Il primo livello è consistito nell'implementazione del modello matematico del veicolo e di un'interfaccia grafica per la simulazione.

2.1 Modello Cinematico (Bonus 1 Incluso)

È stata creata una classe `KinematicModel` che implementa il modello a bicicletta, ovvero la rappresentazione di una macchina a quattro ruote con solo due, in modo da rendere l'implementazione e i controlli più semplici. La funzione chiave è `update()`, che calcola la nuova posizione del veicolo a ogni intervallo di tempo dt. Seguendo il bonus del Livello 1, il modello è stato reso più realistico: riceve un comando di **accelerazione** e la velocità è diventata uno stato interno del veicolo ($v_{t+1} = v_t + a \cdot dt$). Le equazioni di moto complete

implementate sono:

$$
\begin{aligned}
x_{t+1} &= x_t + v_t \cos(\theta_t) dt \\
y_{t+1} &= y_t + v_t \sin(\theta_t) dt \\
\theta_{t+1} &= \theta_t + \frac{v_t}{L} \tan(\delta_t) dt \\
v_{t+1} &= v_t + a_t dt
\end{aligned}
$$

2.2 Visualizzazione e Studio Autonomo

Per la visualizzazione 2D è stata scelta la libreria `raylib`, integrata tramite `FetchContent` in CMake. Il `main.cpp` è stato strutturato come un "game loop" che usa `GetFrameTime()` per un Δt reale.

A questo punto ho capito che mi sarebbe servita almeno un'idea di come funziona il mondo della guida autonoma, quindi mi sono documentato sui tipi di controller che avrei incontrato lungo la strada:

- PID controller [3]
- Pure Pursuit controller [4]
- Stanley controller [5]

3 Livello 2: Controllo dell'Orientamento (P-Controller)

Per il Livello 2, è stato implementato un P-controller sull'angolo di orientamento (heading). Dando un disturbo ($\theta_{start} \neq 0$), la legge di controllo $\delta = K_{p,h} \cdot e_\theta$ (dove $e_\theta = \theta_{target} - \theta_{current}$) si è dimostrata efficace nel raddrizzare il veicolo.

4 Livello 3: Limiti del P-Controller su Traiettoria

Questo livello ha richiesto l'inseguimento di una sinusoide ($y = f(x)$) e ha rivelato i limiti fondamentali di un P-controller "miope" ($\delta = K_{p,y} \cdot e_y$).

4.1 Scoperta 1: Fallimento Catastrofico (Bonus 2)

Testando il controller con un errore e_y molto grande, ho scoperto due fallimenti critici.

Problema 1: Comando di Sterzo Impossibile L'errore e_y elevato generava un comando δ fisicamente impossibile (es. 800°). È stato necessario introdurre un **limitatore** (clamp) a $\pm\delta_{max}$ per simulare il fine corsa del volante.

Problema 2: Stallo Circolare (Analisi Fisica) Anche con il limitatore, l'auto *non riusciva fisicamente* a raggiungere il percorso, bloccandosi in un cerchio. Ho capito perché: il raggio di sterzata minimo R dipende dal passo L e dallo sterzo massimo δ_{max}.

$$R = \frac{L}{\tan(\delta_{\max})} \quad \text{e} \quad D = 2R$$

Con $L = 2.5$ pixel e $\delta_{max} = 0.5$ radianti ($\approx 28{,}65°$):

$$D = 2 \cdot \frac{2.5}{\tan(0.5)} \approx 2 \cdot \frac{2.5}{0.546} \approx 9.16 \text{ pixel}$$

Se l'errore laterale (e_y) è molto maggiore di questo diametro, il controller "saturerà" e il cerchio descritto non sarà mai abbastanza largo da intersecare la traiettoria, o quantomeno da ridurre l'errore per permettere alla macchina di addolcire l'angolo di sterzata e riconnettesi al percorso.

4.2 Scoperta 2: Instabilità ad Alta Velocità

Anche partendo sul percorso, ho notato che l'auto diventava violentemente instabile accelerando. L'analisi del modello ($\theta_{t+1} = \cdots + \frac{v}{L}\tan(\delta)\dots$) rivela che l'effetto dello sterzo δ è **amplificato dalla velocità** v. Il P-controller, essendo "cieco" alla velocità, diventa eccessivamente reattivo e instabile.

4.3 La Scelta Progettuale: Saltare al Livello 5

È stato quindi inevitabile comprendere che un P-controller puro (Livello 3) è intrinsecamente limitato. A questo punto, le strade erano due:

1. **Rimanere nel Livello 4 (PID):** "Rattoppare" il P-controller con un termine Derivativo (D) per smorzare le oscillazioni.
2. **Cambiare al Livello 5 (Pure Pursuit/Stanley):** Riconoscere la logica "reattiva" come causa del fallimento e sostituirla con un controller "proattivo".

Ho determinato che 'rattoppare' con un PID (Livello 4) sarebbe stato solo un curare il sintomo, non la malattia. Ho quindi deciso di passare direttamente a una soluzione superiore (Livello 5) per risolvere questi problemi alla radice.

5 Livello 5: Implementazione di Controller Avanzati

Il Livello 5 richiedeva un controller "proattivo". Il mio processo di sviluppo è avvenuto in due fasi, portandomi prima al Pure Pursuit e infine allo Stanley.

5.1 Path Planning (Bonus 3)

Per prima cosa, ho abbandonato la logica $y = f(x)$ e ho implementato il Bonus 3. Ho ridefinito il percorso come `std::vector<Vector2>` (o `Waypoint`) e ho scritto un "path planner" che genera un cerchio usando **equazioni parametriche** ($x = f(t), y = g(t)$).

5.2 Fase 1: Tentativo con Pure Pursuit (PP)

Come primo approccio, ho scelto il PP perché mi è sembrato l'evoluzione logica della mia intuizione ("guardare avanti"). L'algoritmo implementato cercava un punto a distanza Ld e calcolava l'errore angolare $\alpha = \theta_{target} - \theta_{current}$ (usando `atan2`).

Questo ha richiesto un tuning iterativo che ha svelato ulteriori limiti:

- **Il "Bug di π":** L'auto si "perdeva" dopo un giro. Ho capito che `current_theta` cresceva all'infinito, mentre `atan2` no. Ho risolto **normalizzando** l'errore α nel range $[-\pi, +\pi]$.
- **Errore a Regime Stazionario (Offset):** L'auto manteneva un offset costante. Per risolverlo, ho "rubato" il termine Integrale (I) dal Livello 4, creando un controller **PI-proattivo**.
- **Instabilità Finale:** Anche con l'Anti-Windup, il controller PI-PP era ancora **instabile ad alta velocità**. Proprio come il P-controller, è "cieco" alla v, che rimane il problema fondamentale.

5.3 Fase 2: Soluzione Finale con Stanley Controller (L5, Opzione 1)

Avendo diagnosticato che il problema era l'adattamento alla velocità, ho implementato la soluzione ingegneristica corretta: lo **Stanley Controller**.

Ho sostituito la logica del PP con la formula dello Stanley, come descritto in letteratura [6]. Questo controller è un ibrido che risolve tutti i problemi identificati:

$$\delta = \theta_e + \arctan\left(\frac{K \cdot e_y}{v + \epsilon}\right)$$

- θ_e **(Errore di Angolo):** È l'errore tra l'angolo dell'auto e la *tangente* del percorso (il termine P-controller del Livello 2).
- e_y **(Errore Laterale):** È la distanza dal percorso (il termine P-controller del Livello 3).
- v **(Velocità):** Includendo la velocità v al denominatore, il controller diventa **adattivo**: si "calma" automaticamente all'aumentare della velocità, risolvendo il problema di instabilità.

6 Risultati e Conclusioni

6.1 Risultati della Simulazione

L'implementazione finale dello **Stanley Controller (Livello 5)** si è dimostrata un successo. Il controller è in grado di:

- Acquisire correttamente il percorso circolare (grazie all'angolo di partenza tangente).
- Seguire la traiettoria **senza offset** (grazie al termine e_y).
- Rimanere **stabile anche accelerando** (grazie al termine adattivo $/v$).

Il risultato finale del tracking è visibile qui:

[Link al video della simulazione finale]

6.2 Conclusione Finale

Questo progetto è stato un percorso iterativo attraverso i problemi fondamentali del controllo laterale. Partendo da un P-controller (Livello 3) palesemente fallimentare, ho analizzato e dimostrato i suoi limiti (stallo circolare, instabilità a v). Questo mi ha portato a implementare prima un Pure Pursuit (L5, Opzione 2), che ha richiesto soluzioni avanzate (normalizzazione, PI, anti-windup), e infine allo Stanley Controller (L5, Opzione 1) come soluzione definitiva e robusta.

Il processo ha confermato l'importanza di analizzare i limiti fisici (v/L) e di scegliere architetture di controllo (come lo Stanley) che li affrontino alla radice.

Riferimenti bibliografici

[1] Tutorial video sull'uso di CMake. *(https://youtu.be/_BWU5mWqVA4)*

[2] Tutorial video sull'uso di Git e GitHub. *(https://youtu.be/JB7YD7OKm5g)*

[3] Video esplicativo sul funzionamento dei controllori PID. *(https://youtu.be/4Y7zG48uHRo)*

[4] Video esplicativo sul funzionamento del Pure Pursuit Controller. *(https://youtu.be/zMdoLO4kRKg)*

[5] Video esplicativo sul funzionamento dello Stanley Controller. *(https://youtu.be/9Y7kVIJs_JI)*

[6] Thrun, S., et al. (2006). *Stanley: The Robot that Won the DARPA Grand Challenge.* Journal of Field Robotics.