Immagine che contiene scatola

Descrizione generata automaticamente

Università degli Studi di Salerno

Immagine che contiene testo, clipart

Descrizione generata automaticamente

Dipartimento di Ingegneria dell’Informazione ed Elettrica e Matematica Applicata

Corso di Laurea in Ingegneria Informatica

**Intelligenza Artificiale: Metodi ed Applicazioni**

**2024/2025**

Project Work

Gruppo n. **29**

## Partecipanti

Cocco Luigi 0612708320 - [l.cocco1@studenti.unisa.it](mailto:l.cocco1@studenti.unisa.it)

D’onofrio Matteo 0612707979 - [m.donofrio33@studenti.unisa.it](mailto:m.donofrio33@studenti.unisa.it)

Di Lieto Gabriele 0612708727 - g.dilieto5@studenti.unisa.it

Ciniello Lorenzo 0612708956 - l.ciniello@studenti.unisa.it

Sommario

[Partecipanti 1](#_Toc1036759918)

[0. Introduzione 2](#_Toc316411373)

[1. Scelta feature 2](#_Toc1619237343)

[1.1Scelta classificatore 3](#_Toc1485652423)

[1.2 Normalizzazione 4](#_Toc1898773548)

[2. Descrizione classi 4](#_Toc498698808)

[2.1Funzionamento di CharReader 5](#_Toc864224171)

[2.2 Funzionamento di NearestNeighbor 5](#_Toc1649761101)

[2.3Funzionamento di SimpleDriver 5](#_Toc1577724257)

[2.3.1 Modalità di funzionamento 5](#_Toc263235311)

[2.3.2 Attributi principali 5](#_Toc750121839)

[2.3.3 Metodi principali 6](#_Toc365952920)

[2.4 Funzionamento Sample 7](#_Toc1348601115)

[3. Creazione dataset 7](#_Toc618509027)

[3.1 Funzionamento 7](#_Toc970065424)

# Introduzione

L’obiettivo del progetto è lo sviluppo di un sistema di guida autonoma per un’auto virtuale all’interno dell’ambiente simulato TORCS.

Il nostro gruppo ha realizzato una soluzione basata sull’acquisizione dati in modalità manuale, l’addestramento su tali dati, e l’utilizzo di un classificatore Nearest Neighbor (NN) per controllare l’auto in modalità autonoma.

L’addestramento e la gara si svolgono sul tracciato \*\*FORZA\*\*, appartenente alla categoria \*\*Road Tracks\*\*.

Durante la fase di guida manuale, l’utente controlla il veicolo tramite tastiera, mentre i dati dei sensori e le azioni vengono salvati in un file CSV (`dataset.csv`).

Successivamente, in fase di guida autonoma, il classificatore NN utilizza questi dati per predire le azioni da eseguire in tempo reale.

# Scelta feature

Nel nostro progetto abbiamo selezionato 8 feature, scelte in base alla loro rilevanza per descrivere in maniera efficace lo stato dell’auto all’interno del tracciato e consentire una classificazione affidabile delle azioni.

Le feature sono gestite nella classe SimpleDriver.java e sono:

* **Speed**: rappresenta la velocità dell’auto. È ottenuta dal sensore speedX e viene utilizzata per distinguere situazioni di stallo, rallentamento o accelerazione.
* **DistanzaLineaCentrale**: rappresenta la distanza dell’auto dalla linea centrale della pista, ottenuta dal sensore trackPos. Utile per capire quanto l’auto si discosta dal centro.
* **SensoreSX1**: valore track[3], usato per percepire ostacoli o curve a sinistra.
* **SensoreSX2**: valore track[6], ulteriore misura dell’ambiente laterale sinistro.
* **SensoreCentrale**: valore track[9], rappresenta ciò che è davanti all’auto.
* **SensoreDX1**: valore track[12], per rilevare curve e ostacoli sul lato destro.
* **SensoreDX2**: valore track[15], supporto aggiuntivo alla percezione laterale destra.
* **Angolo**: angolo tra la direzione dell’auto e la tangente alla pista, ottenuto dal sensore angle. Fondamentale per la gestione della sterzata.

Questa combinazione di feature ci consente di:

* Capire se l’auto è centrata o fuori pista.
* Identificare la presenza e il tipo di curva.
* Discriminare le azioni da intraprendere (accelerare, frenare, sterzare).

Le features **DistanzaLineaCentrale** e **Angolo** sono state scelte per discriminare le varie situazioni in cui si può trovare l’auto all’interno della pista e determinare gli opportuni comportamenti da seguire.

Le features relative al sensore di “track” sono state scelte per discriminare le varie tipologie di curve e identificare accuratamente i “fuori-pista”. In particolare, sono state selezionate le features relative ai sensori di “track”:

* track[3] e track[6] per analizzare l’ambiente al lato sinistro dell’auto;
* track[9] per analizzare l’ambiente di fronte l’auto;
* track[12] e track[15] per analizzare l’ambiente al lato destro dell’auto. Sono opportunamente distanziati per avere un range di analisi adeguato e non creare sovrapposizioni inutili.

## 1.1 Scelta classificatore

Per la guida autonoma dell’auto abbiamo scelto un classificatore di tipo **Nearest Neighbor (NN)**.

Il funzionamento è il seguente:

* L’algoritmo confronta l’input corrente (costituito dalle 8 feature) con i dati presenti nel dataset (dataset.csv).
* Viene calcolata la distanza tra l’input e tutti i campioni salvati durante l’addestramento.
* L’azione associata al campione più vicino viene predetta e applicata all’auto.

Nel nostro caso, ogni classe (cioè ogni azione che l’auto può compiere) è rappresentata da un intero tra 0 e 9, dove ogni numero rappresenta una combinazione unica di accelerazione, frenata e sterzata (es. svolta a sinistra leggera, frenata secca, ecc.):

0 -> accelera

1 -> giraSXMolto;

2 -> giraSX;

3 -> giraSXPoco;

4 -> giraDXMolto;

5 -> giraDX;

6 -> giraDXPoco;

7 -> frena;

8 -> retromarcia;

9 -> decelera;

## 1.2 Normalizzazione

Nel nostro sistema, le feature raccolte dai sensori dell’auto simulata hanno scale numeriche molto differenti tra loro. Per evitare che questo squilibrio influenzi negativamente le decisioni del classificatore Nearest Neighbor, abbiamo applicato una normalizzazione.

La tecnica scelta è la **Min-Max Normalization**, che trasforma ogni valore in un numero compreso tra 0 e 1, mantenendo le proporzioni relative tra i dati originali.

In dettaglio, per ciascuna feature:

* **Speed**: il valore della velocità è ottenuto dal sensore getSpeed() e normalizzato nell’intervallo da 0 a 280. Questo intervallo è coerente con le massime prestazioni osservate del veicolo nel contesto della pista utilizzata.
* **Distanza dalla linea centrale**: derivata dal sensore getTrackPosition(), viene normalizzata tra -10 e 10. In TORCS, questo sensore restituisce valori da -1 (auto totalmente a sinistra della pista) a 1 (auto completamente a destra), con 0 corrispondente al centro della carreggiata.
* **Sensori di distanza dal bordo pista**: i sei sensori track[3], track[6], track[9], tracl[12], track[15]  forniscono la distanza dal bordo pista in metri. Il loro range naturale va da 0 (ostacolo immediatamente vicino) a 200 metri (nessun ostacolo rilevato), ma possono anche restituire **-1** in caso di valori non validi o auto fuori pista. Per questo abbiamo scelto come intervallo di normalizzazione **[-1, 200]**.
* **Angolo di orientamento**: il sensore getAngle() fornisce l’angolo tra la direzione dell’auto e l’asse della pista. Il range considerato è quello naturale di [-π, π], e viene anch’esso normalizzato per rientrare nel range [0,1].

Questa normalizzazione permette:

* Di garantire un contributo uniforme di tutte le feature nel calcolo della distanza in modo da avere delle scale comparabili.
* Di prevenire bias dovuti a range numerici squilibrati in modo tale da non avere feature più importanti delle altre.
* Di rendere il comportamento del sistema più stabile e interpretabile.

# Descrizione classi

### 

## 2.1 Funzionamento di CharReader

Questa classe gestisce l’interfaccia per acquisire input da tastiera durante l’addestramento. Estende JFrame e:

* mostra un campo testuale all’utente,
* registra ogni carattere premuto (es. a, d, w...),
* comunica l’input alla classe SimpleDriver, che lo usa per associare la situazione dell’auto a un’azione (scrivendola nel dataset).

È quindi l’interfaccia di controllo nella fase manuale.

## 2.2 Funzionamento di NearestNeighbor

Questa classe implementa il classificatore **Nearest Neighbor**. È semplice ed efficace per il tipo di problema trattato.

Contiene:

* una lista di Sample, che rappresentano i dati raccolti durante l’addestramento;
* un metodo readPointFromCSV, che carica i dati dal file dataset.csv ignorando l’intestazione;
* un metodo classify, che confronta un nuovo vettore di feature con tutti i campioni salvati e restituisce la **classe del più vicino**, calcolando la distanza Euclidea.

È un componente fondamentale per la guida autonoma: riceve in input la situazione dell’auto (le feature) e decide quale azione eseguire.

## 2.3 Funzionamento di SimpleDriver

La classe  SimpleDriver è il cuore del nostro progetto: si occupa della gestione della guida (manuale o autonoma), della normalizzazione delle feature, della costruzione del dataset e dell’esecuzione delle azioni sul veicolo.

Estende la classe Controller e gestisce sia la fase di addestramento (in cui viene generato il dataset) sia la guida autonoma (in cui le decisioni vengono prese tramite classificazione).

### **2.3.1 Modalità di funzionamento**

* In **modalità manuale** (addestramento), l’utente controlla l’auto tramite tastiera. A ogni input vengono letti i dati dei sensori e salvati, insieme all’azione corrispondente, nel file dataset.csv.
* In **modalità autonoma**, l’auto utilizza un classificatore Nearest Neighbor per predire quale azione eseguire in base allo stato corrente dei sensori.

### **2.3.2 Attributi principali**

La classe è composta da numerosi attributi, tra cui:

* action: oggetto di tipo Action, rappresenta l’azione attuale da compiere.
* features: array di 8 double contenente le feature normalizzate in tempo reale.
* angolo: valore ottenuto dal sensore angle.
* classe: intero tra 0 e 9 che rappresenta l’azione classificata (sterzata, accelerazione, frenata...).
* ch: carattere immesso dall’utente (usato solo in modalità manuale).
* lettura: booleano che abilita o meno la scrittura nel dataset.
* file: puntatore al file CSV del dataset.
* firstLineOfTheFile: intestazione da scrivere nel file CSV.
* nn: istanza della classe NearestNeighbor, attiva solo in guida autonoma.
* prototypes\_filename: path al file dataset da usare in fase di predizione.
* guidaAutonoma: flag booleano che determina la modalità (manuale o autonoma).

Sono inoltre presenti numerose costanti per:

* la gestione della trasmissione (gearUp, gearDown),
* il rilevamento dello stato di "stuck" dell’auto,
* i parametri di accelerazione, frenata, ABS e frizione (clutch),
* la gestione della sterzata (steerLock, steerSensitivityOffset, ecc.).

### **2.3.3 Metodi principali**

* control(SensorModel sensors):  
  Metodo principale, chiamato a ogni frame di simulazione.
  + In entrambe le modalità, legge i dati dai sensori (speed, angle, trackPos, track[]).
  + Normalizza le feature con il metodo normalizzatoreMinMax.
  + In modalità manuale, scrive il dato nel CSV.
  + In modalità autonoma, invoca predictControl per ottenere la classe, e autoControl per eseguire l’azione associata.
* normalizzatoreMinMax(double dato, double min, double max):  
  Trasforma il valore nel range [0,1] in modo proporzionale.
* writeCSV(SensorModel sensors):  
  Scrive nel dataset una riga composta dalle 8 feature normalizzate e dalla classe associata al tasto premuto.
* predictControl():  
  Utilizza l’istanza nn per ottenere, tramite nearest neighbor, la classe predetta sulla base del vettore di feature corrente.
* autoControl():  
  Associa a ogni classe una specifica funzione di guida, ad esempio:
  + classe == 0 → accelera()
  + classe == 3 → giraSXPoco()
  + classe == 6 → giraDXPoco()
* accelera(), frena(), giraSXMolto(), giraDXPoco(), retromarcia(), decelera(), ecc.:  
  Funzioni che impostano l’oggetto action con valori specifici di accelerate, brake, steering, e gear.

Alcuni esempi concreti:

* + Nelle sterzate più strette (giraSXMolto, giraDXMolto): accelerate = 0, brake = 1, steering = ±0.5.
  + In quelle leggere (giraSXPoco, giraDXPoco): accelerate = 1, brake = 0, steering = ±0.1.
  + In retromarcia(): imposta gear = -1, accelerate = 0.15 e calcola steering come -angolo / steerLock.
* filterABS():  
  Applica una logica semplificata di ABS (Anti-lock Braking System), attivato solo sopra una certa velocità e in base al valore di slittamento.
* clutching():  
  Regola il valore della frizione nei primi istanti della gara per favorire la partenza.
* reset() e shutdown():  
  Gestione dell’inizio e fine gara.
* Getter/setter come getCh, setCh, isLettura, setLettura:  
  Servono per scambiare input tra l’interfaccia grafica (CharReader) e il driver.

## 2.4 Funzionamento Sample

La classe Sample rappresenta un singolo punto del dataset. Ogni oggetto ha:

* un array di double (features) che rappresenta le feature normalizzate,
* un intero (cls) che rappresenta la classe (cioè l’azione presa).

Questa classe fornisce anche un metodo distance(Sample other) che calcola la **distanza Euclidea** tra due vettori di feature, utilizzato nel classificatore NN.

# Creazione dataset

La fase di creazione del dataset avviene durante la **guida manuale** dell’auto. In questa modalità, l’utente controlla il veicolo tramite tastiera e ogni azione compiuta viene registrata insieme allo stato dei sensori al momento dell’input.

Questa funzionalità è implementata interamente nella classe SimpleDriver.java, in particolare nei metodi control() e writeCSV().

## ****Funzionamento****

1. **Avvio in modalità manuale**:  
   Quando il parametro guidaAutonoma è impostato a false, l’auto non esegue azioni autonome, ma si limita a ricevere input da tastiera tramite la classe CharReader.
2. **Raccolta delle feature**:  
   Ad ogni frame di simulazione, il metodo control() legge i dati dei sensori (velocità, posizione rispetto al centro della pista, sensori di distanza, angolo dell’auto) e li normalizza tramite la funzione normalizzatoreMinMax().
3. **Registrazione della classe (azione)**:  
   L’input da tastiera inserito dall’utente (ad esempio a, d, w, s, ecc.) viene convertito in un intero compreso tra 0 e 9. Ogni numero rappresenta una classe, ovvero un comportamento specifico dell’auto (es. gira a sinistra molto, accelera, frena, ecc.).
4. **Scrittura del dato**:  
   I valori normalizzati delle 8 feature, insieme alla classe numerica, vengono salvati in una riga del file CSV dataset.csv.  
   Questo file si trova nella directory Torc/classes, e viene scritto con la modalità **append**, in modo da accumulare i dati sessione dopo sessione.
5. **Intestazione del file**:  
   L’intestazione del file viene scritta una sola volta, al momento della prima scrittura, e contiene i nomi delle feature:  
   Speed; DistanzaLineaCentrale; SensoreSX1; SensoreSX2; SensoreCentrale; SensoreDX1; SensoreDX2; Angolo; Classe