Università degli Studi di Salerno

Dipartimento di Ingegneria dell'Informazione ed Elettrica e Matematica Applicata

Corso di Laurea in Ingegneria Informatica

INTELLIGENZA ARTIFICIALE: METODI ED APPLICAZIONI 2023/2024

Project Work

Relazione progetto simulazione auto guida autonoma

Gruppo n. 03

Cognome e Nome	Matricola	e-mail
Cipriano Ivan Luigi	0612705301	i.cipriano1@studenti.unisa.it
Faraulo Simone	0612706493	s.faraulo@studenti.unisa.it
Graziosi Antonio	0612705823	a.graziosi4@studenti.unisa.it
Fortunato Sara	0612706737	s.fortunato25@studenti.unisa.it

1. OBIETTIVO DEL PROGETTO

L'obiettivo del progetto è quello di realizzare un client TORCS che, tramite un approccio behavioral cloning o rule-based, consenta all'auto di poter completare un giro in una pista nel minor tempo possibile.
Il behavioral cloning prevede l'addestramento del modello basato su dati di guida raccolti dal giocatore umano durante le sessioni di gara. Questa tecnica è la più semplice forma di "imitation learning", in cui il modello imita le azioni del conducente umano, permettendo al veicolo di apprendere e replicare comportamenti complessi nelle diverse situazioni.
L'approccio rule-based, invece, integra un insieme di regole predefinite e algoritmi specifici per gestire tutti i possibili scenari.

2. APPROCCIO UTILIZZATO

L'approccio da noi utilizzato è stato quello del behavioral cloning, mediante l'uso di un classificatore K-Nearest Neighbors (KNN). Questo algoritmo, privo di parametri, non prevede una vera e propria fase di addestramento, ma memorizza semplicemente il dataset fornito. L'algoritmo effettua:

- la classificazione di un nuovo sample, calcolando la distanza euclidea tra il nuovo sample fornito e tutti i sample presenti nel dataset di addestramento;
- la selezione dei *k* sample più vicini;
- la classificazione del nuovo sample utilizzando un algoritmo basato sulla mediana.

I vantaggi nell'utilizzo di questo algoritmo sono la semplicità di implementazione e comprensione, oltre alla mancanza di necessità di una fase di addestramento, inoltre rispetto ai sistemi basati su regole, permette di costruire le regioni di decisione delle classi non lineari, divenendo così più flessibile.
Gli svantaggi principali riguardano il costo computazionale, poiché calcolare per ogni sample la distanza può essere molto costoso in termini di tempo e risorse, specialmente in presenza di un dataset molto vasto, inoltre questi algoritmi sono molto sensibili alla presenza di attributi irrilevanti.

2.1. MODIFICA DEL CODICE E RACCOLTA DEI DATI

La prima fase del progetto ha riguardato la modifica del codice e la raccolta dei dati. Siamo partiti dal client fornitoci inizialmente, che abbiamo modificato per adattarlo al nostro scopo, consentendoci di pilotare manualmente l'auto e al contempo di memorizzare su un file CSV i dati dei sensori e degli attuatori.
Per perseguire il nostro scopo è stato necessario introdurre una nuova classe, ContinuousCharReaderUI.java (già fornita), che rappresenta un'interfaccia utente e

consente all'utente di inserire caratteri tramite una casella di testo. Abbiamo opportunamente modificato la classe aggiungendo un attributo keys di tipo Set, utilizzato per memorizzare i tasti attualmente premuti dall'utente. I metodi keyTyped(KeyEvent e) e keyReleased(KeyEvent e) permettono rispettivamente di aggiungere alla collezione il tasto digitato dall'utente e di rimuoverlo.
Un'istanza di questa classe è poi utilizzata nella classe SimpleDriver.java, anche essa modificata, andando completamente ad eliminare e riscrivere il metodo control() presente. Questo metodo è responsabile della gestione di input da tastiera e del loro utilizzo per controllare il veicolo. I tasti premuti sono recuperati utilizzando il metodo getKeys() implementato nella classe ContinuousCharReaderUI.java. In base ai tasti premuti, abbiamo previsto azioni di: accelerazione, frenata, sterzata e retromarcia, impostando i vari valori in base al nostro stile di guida. I valori impostati degli attuatori vengono inviati al server in input per controllare il veicolo e, al contempo, scritti su file per le fasi successive. Abbiamo deciso già in questa fase di non considerare tutti i sensori messi a disposizione, ma di scrivere su file solo quelli che ci sembravano più adatti, poiché alcuni risultavano essere inutili per i nostri scopi, come ad esempio: quelli relativi al tempo trascorso durante il giro corrente, danno dell'auto, distanza della vettura dalla linea di partenza ecc. Il file è organizzato in colonne, ogni campo è separato da un carattere ";" e la prima riga descrive il contenuto informativo delle varie colonne presenti nel file. Abbiamo deciso di eseguire il task relativo alla scrittura su file periodicamente ad intervalli di 50ms, in modo tale da non prelevare tutti i dati, rischiando di creare file molto grandi. È stata implementata anche un'ulteriore funzionalità che ci permette di decidere quando iniziare a prelevare i dati e quando fermare la scrittura su file, in modo da poter gestire più facilmente tutte le varie situazioni e poter prelevare dati solo quando lo si vuole.
Dopo aver modificato le classi, abbiamo avviato la fase di raccolta effettuando diversi giri sulle piste e registrando numerosi dati. In particolare, le piste su cui sono stati raccolti i dati sono: CG Speedway number 1, Forza, Alpine 1. La scelta delle piste è stata mirata, poiché tra le tante a disposizione, abbiamo selezionato queste per vari motivi: ogni pista presenta un insieme di curve e rettilinei variegato, che ci ha consentito di poter registrare dati sulle varie casistiche, costruendo così un dataset più robusto.
In particolare la maggior parte dei dati sono stati reperiti dalla pista CG Speedway number 1; per raggiungere velocità più alte, abbiamo acquisito dati sulla pista Forza solo per i rettilinei, dove le velocità risultano essere maggiori; infine sono stati raccolti dati sulla pista Alpine 1, solo relativamente ai tornanti, in modo da rendere l'automobile capace di poter effettuare anche curve che siano più strette e più lente. Sono stati registrati anche dati relativi al caso in cui l'auto dovesse andare fuori pista e sbattere sul guardrail, consentendo di poter effettuare la retromarcia per tornare in carreggiata.

2.2. TEST

La fase successiva ha previsto l'utilizzo dei dati raccolti per effettuare i primi test di guida autonoma. È stato nuovamente necessario modificare il codice, aggiungendo altre classi per implementare l'algoritmo del classificatore KNN. Anche per il KNN ci erano stati forniti degli esempi, che abbiamo modificato adattandoli alla nostra situazione.

La classe Sample.java è stata modificata in base al nostro vettore di features. Dopo un'attenta analisi dei vari sensori registrati e i primi test effettuati, abbiamo deciso di mantenere solo alcuni dati tra quelli raccolti: angle, speedX, speedY, track[] e track_pos. L'utilizzo di questi sensori come features ha consentito prestazioni dell'auto molto elevate e simili alle nostre.

In questa classe è presente un'istanza di Classe.java, che è stata da noi implementata per impostare i valori degli attuatori da fornire a TORCS al fine di eseguire l'azione corretta. La classe KDTree.java è stata adattata per organizzare e gestire un insieme di punti nello spazio multidimensionale. La distanza euclidea è stata calcolata utilizzando tutte le features, ma il numero di assi è stato ridotto rispetto al totale delle features. Questa scelta è derivata dall'analisi dei dati raccolti in precedenza dai sensori track[]. Abbiamo notato che i valori di alcuni sensori non variano significativamente rispetto a quelli selezionati, quindi abbiamo deciso di utilizzare solo i sensori nelle posizioni:

track[0], track[3], track[7], track[8], track[10], track[12], track[15], track[18].

Questo approccio ha permesso di ridurre il tempo di ricerca dell'algoritmo mantenendo un alto livello di precisione. La classe KNearestNeighbor.java implementa un algoritmo di classificazione che utilizza il metodo dei KNN in combinazione con un albero KDTree per ottimizzare le operazioni di ricerca dei vicini più prossimi nei dati di addestramento. In particolare il metodo classify() permette di classificare un sample utilizzando innanzitutto l'algoritmo dei KNN, che restituisce la lista dei *k* sample più vicini al sample in input, e poi applicando a questi l'algoritmo della mediana per determinare la classe più affidabile tra i suoi *k* vicini più prossimi.

Infine, abbiamo modificato la classe Client.java aggiungendo inizialmente la possibilità di scegliere la modalità di esecuzione del codice, 'automatica' o 'manuale', leggendo dal terminale il carattere inserito dall'utente. Successivamente, nel metodo main, è stato istanziato e aggiunto il codice relativo alla guida autonoma. A partire dal messaggio ricevuto dal server contenente i dati dei sensori utilizzati come features in ogni istante di tempo, si istanzia un nuovo sample che viene utilizzato come parametro di input del metodo classify. Questo metodo restituirà l'istanza della classe contenente i valori degli attuatori da inviare al server.

La scelta del parametro *k* è stata fondamentale per garantire ottime prestazioni, la scelta è stata effettuata sperimentalmente, valutando le performance del classificatore sul dataset costruito. Alla fine il valore 5 si è dimostrato ottimale, consentendo di ottenere sia alte prestazioni che precisione molto elevata.

2.2.1. RISULTATI DEI TEST

I test sono stati condotti sulle seguenti piste: CG Speedway number 1, CG track 2, Forza, Alpine 1, Ruudskogen, Wheel 1. I risultati mostrano che il modello KNN è in grado di completare più giri di cui:

- CG Speedway number 1: di cui il tempo migliore è 0:43:71 in 5 giri.
- CG track 2: di cui il tempo migliore è 1:02:34 in 3 giri.
- Forza: di cui il tempo migliore è 1:44:084 in 4 giri.
- Alpine 1: di cui il tempo migliore è 2:55:014 in 5 giri, con qualche difficoltà nelle curve strette, però riesce a completare tutti i giri.
- Ruudskogen: di cui il tempo migliore è 1:20:086, dopodiché effettua dei testacoda e non è più controllabile.
- Wheel 1: di cui il tempo migliore è 1:44:013 in 2 giri, dopodiché effettua un testacoda e non è più controllabile.

2.2.1.1. NOTE AGGIUNTIVE SUI TEST

Ci siamo resi conto che la retromarcia funziona correttamente solo in determinate circostanze:

- se l'auto sbatte sulla destra o sulla sinistra spesso è in grado di tornare indietro e ripartire, ciò accade per lo più sulla pista Alpine 1 la quale è più stretta rispetto alle piste su cui sono stati prelevati i dati del dataset.
- Se l'auto effettua dei testacoda non è in grado di tornare indietro.

Inoltre, durante i test della modalità guida automatica su vari PC, ci siamo resi conto che per ottenere le massime prestazioni è consigliabile evitare l'esecuzione di altre applicazioni in background, poiché potrebbero influire nei tempi di completamento dei giri su alcune piste, rallentando l'esecuzione dell'algoritmo. I tempi sopra citati e i test finali sono stati effettuati con PC alle massime prestazioni.

3. DISCUSSIONE

Durante lo sviluppo del progetto, abbiamo incontrato diverse sfide, la prima delle quali ha riguardato la fase di modifica del codice per poter pilotare manualmente l'auto nel modo più efficace possibile e facendo in modo che l'esperienza del giocatore sia il più vicino possibile a quella del simulatore di default. La scelta dei sensori da utilizzare nel vettore di features è stata cruciale, per ottenere delle prestazioni elevate anche su piste che non ricadono nell'insieme di quelle da cui sono stati prelevati i dati. Abbiamo dovuto poi bilanciare la quantità di dati raccolti per evitare file di grandi dimensioni cercando però di non perdere informazioni fondamentali.

4. CONCLUSIONI

In conclusione possiamo dire che il nostro client funziona in maniera ottimale su piste simili a quelle in cui sono stati prelevati i dati. Su altre, invece, l'auto si comporta bene solo in alcuni tratti, in altri, non avendo a disposizioni sufficienti dati effettua delle azioni errate.

Il sistema di guida autonoma sviluppato potrebbe essere migliorato utilizzando ad esempio algoritmi di machine learning avanzati. Inoltre ulteriori analisi dei dati raccolti potrebbero portare alla creazione di nuove feature derivate, migliorando le capacità del modello di prendere decisioni accurate. Potremmo migliorare la capacità dell'auto di navigare in scenari più complessi, come ad esempio in presenza di altre auto o condizioni metereologiche differenti.