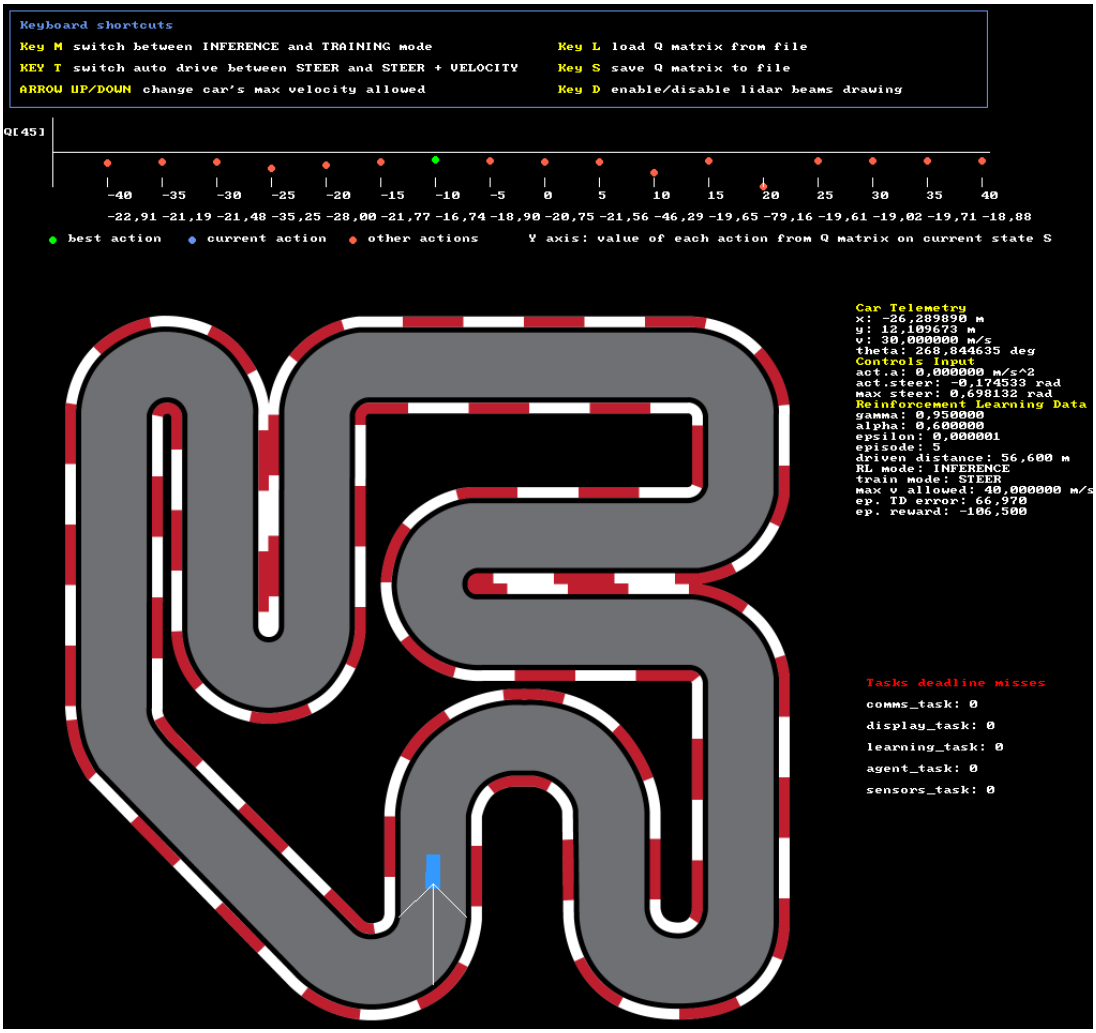
Autonomous Driving

Report

Laurea Magistrale in Ingegneria Robotica e dell' Automazione
Sistemi in Tempo Reale 2021/2022



Popitanu Silviu Roberto
s.popitanu@studenti.unipi.it
matricola: 508924

September 10, 2022

Abstract

Simulazione di una macchina autonoma su pista, controllata da un algoritmo basato su Reinforcement Learning. La percezione del movimento della macchina su pista è realizzata mediante utilizzo di sensore lidar. La lettura dei sensori, l'interprete dei comandi, interfaccia grafica e l'apprendimento artificiale sono gestiti mediante l'utilizzo di diversi task interagenti secondo una logica di tipo *Priority Scheduling*.

Indice

Panoramica

1.1 Introduzione

Lo scopo di questo progetto consiste nel simulare un'applicazione di tipo *real time*, costituita da più task, in un ambiente *Unix*, in linguaggio C, usando le librerie *pthread* e *allegro*. Il tema scelto per tale scopo è la simulazione di guida autonoma all'interno di una pista 2D. La percezione dell'ambiente avviene utilizzando una versione semplifica di un sensore di tipo lidar. Esso è costituito da 3 raggi, di cui uno frontale e due laterali inclinati di 45° rispetto a quello centrale, come visibile nella figura 1.1.

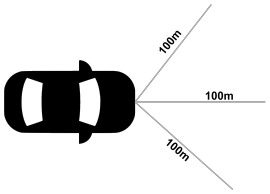


Figure 1.1: Disposizione raggi lidar

Il sensore misura la distanza rispetto al bordi della pista, fino a un valore massimo pari a 100m, preso come riferimento di portata massima del sensore.

La guida autonoma è stata simulata attraverso il controllo da parte di un agente intelligente sullo sterzo e accelerazione della macchina, in seguito a un processo di apprendimento neurale basato su un algoritmo di reinforcement learning intitolato *Q-Learning* che sfrutta le informazioni sulla posizione della macchina sulla pista date dal sensore lidar in relazione ai comandi di guida mandati dall'agente. In questo progetto sono state simulate due modalità di guida:

Solo sterzo: in questa modalità viene fissata la velocità della macchina e tenuta constante, mentre l'agente deve mandare i comandi di sterzo per garantire la guida della macchina lungo la pista senza collisioni.

Sterzo + Accelerazione: in questa modalità la velocità iniziale viene impostata a zero e l'agente deve mandare sia i comandi di sterzo che accelerazione per guidare la macchina lungo la pista.

1.2 Collisione macchina-bordo pista

Ai fini dell'apprendimento è necessario individuare la situazione critica in cui i comandi dell'agente portano alla collisione della macchina con i bordi della pista ed agire di conseguenza. In questo progetto, per individuare tale condizione, è stato deciso di sfruttare un test grafico analizzando l'intersezione dei vertici del rettangolo rappresentante la macchina con i pixel neri presenti nei bordi interni della pista. Se tale test ha esito positivo significa che una collisione è avvenuta e di conseguenza, la variabile globale *rl_agent* che tiene traccia dello stato dell'agente viene modificata, settando il parametro *alive=0*, per segnalare tale comportamento. La verifica del crash viene gestita dal task *agent_task*.

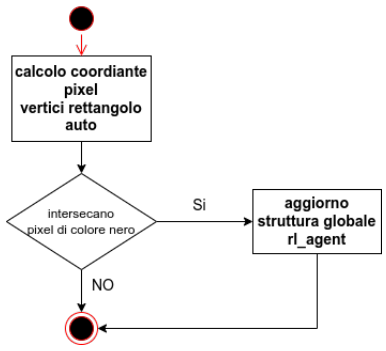


Figure 1.2: logica funzione crash_check()

1.3 Considerazioni sensori lidar e gestione pixel

La percezione dei sensori delle distanze rispetto ai bordi della pista viene effettuata analizzando il colore dei pixel nelle direzioni dei raggi. Partendo dalla macchina, con una risoluzione di 1m, la distanza percepita viene incrementata fino a quando il pixel di tali coordinate non ha più il colore dell'asfalto, fino a un valore massimo di 100m (come evidenziato in figura 1.3)

Figure 1.3: Pista con rappresentazione macchina e sensore lidar

Come scelta progettuale è stato deciso di adottare diverse scale di conversione pixel/m in modo da avere un'interfaccia più vicina a un simulatore. In particolare per quanto riguarda la valutazione delle distanze dei lidar è stata adottata una scala di 1m/1px mentre per quanto riguarda lo spostamento della macchina in pista è stata adottata una scala di 1px/0.16m per rendere realistica la posizione in pista di una macchina lunga 5m e larga 2m.

Interfaccia utente

L'applicazione è costituita da 4 sezioni principali.

2.1 Istruzioni comandi tastiera

Permette di offrire all'utente una panoramica sui possibili comandi per interagire con l'applicazione (figura 2.1).

Figure 2.1: Possibili comandi da tastiera

- Tasto *M*: passa da modalità apprendimento agente a modalità *inferenza*, dove l'agente utilizza i dati appresi per guidare la macchina in pista
- Tasto *T*: permette di cambiare la modalità di guida da controllo *sterzo* a controllo *sterzo-velocità*
- Tasto *L*: permette di caricare da file la matrice Q utilizzata dall'algoritmo Q-Learning.
- Tasto *S*: permette di salvare su file la matrice Q modificata dall'algoritmo Q-learning durante la fase di apprendimento
- Tasto *D*: abilita e disabilita la visualizzazione grafica dei raggi del sensore lidar presente sulla macchina
- Frecce *Up/Down*: incrementano e decrementano la massima velocità a cui la macchina può andare

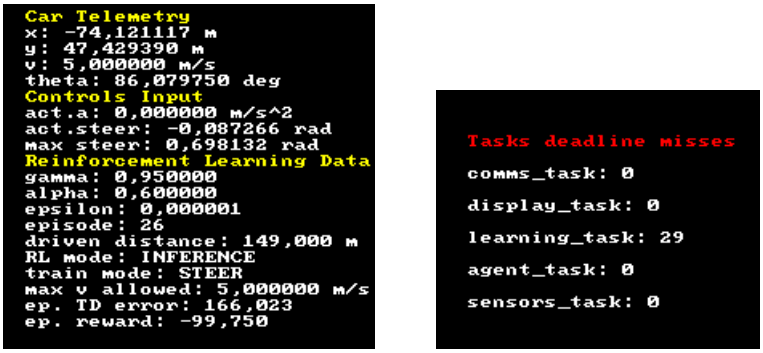
2.2 Dati telemetria

In questa sezione è possibile visualizzare la telemetria della macchina e i dati sull'evoluzione dell'algoritmo di apprendimento (figura 2.2a).

- sottosezione **Car Telemetry**: viene mostrata la posizione *(x,y)* della macchina in pista, in coordinate locali, la velocità attuale della macchina e l'orientazione θ della macchina.

- sottosezione **Controls Input**: viene mostrato il comando di input mandato dall'agente in termini di accelerazione e angolo di sterzo δ da utilizzare nel modello della macchina per calcolare la posizione futura in pista, in termini di (x, y, θ). Inoltre viene mostrato anche il massimo angolo di sterzo ammissibile in radianti.

- sottosezione **Reinforcement learning Data**: vengono mostrati i valori dei parametri utilizzati dall'algoritmo Q-Learning, la distanza in metri, percorsa dalla macchina lungo la pista e il numero di episodi trascorsi. Inoltre vengono visualizzati la modalità di guida e stato algoritmo di apprendimento (come spiegato in 2.1), il reward e l'errore accumulato durante l'episodio corrente.



(a) Dati di telemetria (b) Deadline misses dei task

Figure 2.2: Sezione laterale interfaccia applicazione

2.3 Deadline miss dei task

In questa sezione dell'applicazione (figura 2.2b) vengono mostrati le eventuali deadline miss dei task agenti nel programma:

- **comms_task**: processo che legge i comandi da tastiera inviati ed esegue le azioni opportune
- **display_task**: processo responsabile del rendering grafico
- **learning_task**: processo agente in modalità *single task* che permette di eseguire tutta l'applicazione ed effettuare l'apprendimento utilizzando un solo task. Raggruppa tutti gli altri task in uno solo e principalmente viene usato in fase di debug del codice e test di corretta esecuzione della fase di apprendimento.
- **agent_task**: tale processo viene usato in modalità *multi task* e si occupa di aggiornare lo stato della macchina in pista ed eseguire l'algoritmo di apprendimento.
- **sensors_task**: processo responsabile dell'aggiornamento del sensore lidar in relazione alla posizione della macchina in pista.

2.4 Grafico algoritmo Q-Learning

In questa sezione dell'applicazione viene mostrato lo stato corrente della matrice Q utilizzata dall'algoritmo Q-Learning sia in fase di apprendimento che di inferenza.

Sull'asse orizzontale sono indicate le possibili azioni che l'agente può mandare come comandi di sterzo δ, in gradi, alla macchina. Sull'asse verticale invece viene mostrato il relativo valore della matrice Q (come spiegato nel capitolo 4) per ognuna delle possibili azioni, scalato in relazione al massimo valore presente su tale riga. In figura 2.3, ad esempio, è possibile notare che lo stato attuale della matrice Q era $s = 45$ e di conseguenza i possibili valori indicati sull'asse verticale sono Q[45][i] con i che varia da -40 a +40.

Tale grafico, inoltre, mostra anche la migliore azione che l'agente può scegliere (dato lo stato attuale della matrice Q) usando un pallino verde e la vera azione scelta dall'agente, usando un pallino blu. Nella figura 2.3, per esempio, l'agente ha scelto l'azione migliore e di conseguenza è presente soltanto il pallino verde dato che l'azione scelta coincide con quella migliore.

Poiché la parte più critica della fase di apprendimento è data dalla scelta dell'angolo di sterzo in relazione allo stato del sensore lidar, come scelta progettuale è stato deciso di non mostrare, a livello grafico, anche lo stato della matrice *Q_vel*, responsabile dell'apprendimento delle accelerazione da usare in fase di guida nella modalità *sterzo+velocità*.

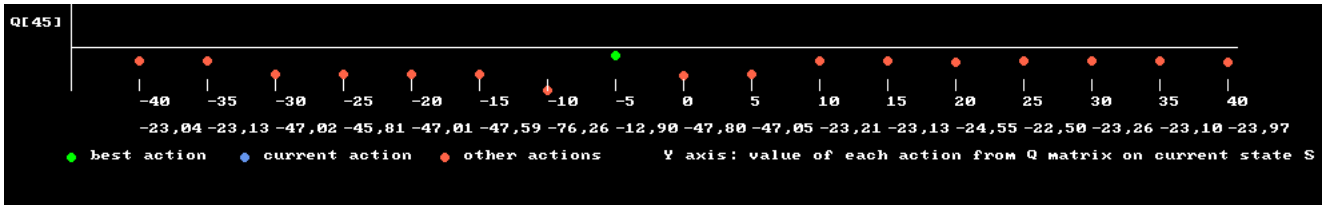


Figure 2.3: Rappresentazione grafica della matrice Q

Modello fisico della macchina

Per aggiornare la posizione della macchina durante l'evoluzione in pista è stato scelto come modello matematico il modello cinematico monotraccia rappresentato in figura 3.1.

Per facilitare i calcoli, il centro del sistema di riferimento rispetto al quale sono stati ricavati i vari parametri è stato posto sull'asse posteriore del modello.

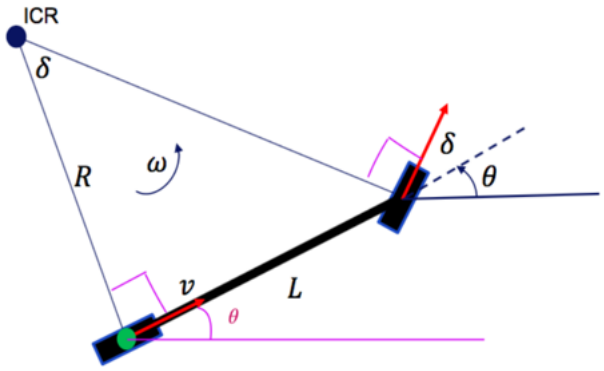


Figure 3.1: Modello cinematico monotraccia asse posteriore

L'aggiornamento di velocità della macchina avviene secondo la formula del moto uniformemente accelerato. È importante osservare, come evidenziato nella figura 3.2, che nella fase di aggiornamento del modello matematico della macchina viene utilizzata la velocità dello stato precedente (old_v) e non quella aggiornata (new_v), poiché quest'ultima rappresenta lo step successivo a cui si troverà la macchina in seguito all'input dato dall'agente. Data la velocità attuale della macchina, aggiorno stato (x, y, θ) e calcolo la nuova velocità che la macchina avrà. Questi nuovi parametri saranno il nuovo stato del modello. Allo step successivo, sfrutterò v_new come nuova velocità corrente e i dati di input dati dall'agente per ripetere l'operazione.

$$\begin{aligned} new\_v &= old\_v + (\Delta t * action.a) \\ x'(t) &= old\_v * cos(\theta) \\ y'(t) &= old\_v * sin(\theta) \end{aligned} \tag{3.1}$$

Figure 3.2: Equazioni aggiornamento velocità

Per il calcolo di θ' è necessario sfruttare alcuni dati geometrici del modello come ad esempio la lunghezza della macchina L e il raggio R (figura 3.3).

$$\begin{aligned} R &= L/tan(\delta) \\ \omega &= v * R \\ \theta' &= \omega \end{aligned} \tag{3.2}$$

Figure 3.3: Equazioni aggiornamento θ'

Poiché l'aggiornamento della macchina sulla pista avviene in maniera discreta, in relazione al periodo del task che esegue tale operazione (*agent_task* oppure *learning_task* come evidenziato nella sezione 2.2b), il calcolo dei parametri aggiornati del modello di figura 3.1, deve essere effettuato in relazione a tale periodo come evidenziato in 3.4.

$$\begin{aligned} x(t+1) &= x(t) + x'(t) * \Delta t \\ y(t+1) &= y(t) + y'(t) * \Delta t \\ \theta(t+1) &= \theta(t) + \theta'(t) * \Delta t \end{aligned} \tag{3.3}$$

Figure 3.4: Equazioni aggiornamento stato della macchina

Come scelta progettuale è stata adottata una normalizzazione di θ nel range $(0, 360°)$. Inoltre sia per quanto riguarda l'accelerazione che la velocità massima è stata introdotta una saturazione dei valori quando le equazioni matematiche del modello superano i massimi definiti per via parametrica.

Per evitare movimenti della macchina in retromarcia, nel caso in cui la velocità della macchina diventi negativa in seguito a comandi di decelerazione è stato deciso di tenere la velocità pari a zero, rendendo il comportamento meno realistico però più facile ai fini dell'apprendimento.

Reinforcement Learning

4.1 Q-Learning

L'apprendimento è stato gestito usando un algoritmo di Reinforcement Learning intitolato *Q-Learning* in modalità ϵ-*greedy*. Tale approccio consiste nel ottimizzare una funzione di qualità $Q(s, a)$ mediante l'interazione dell'agente con l'ambiente in cui si trova ad operare, ovvero la pista.

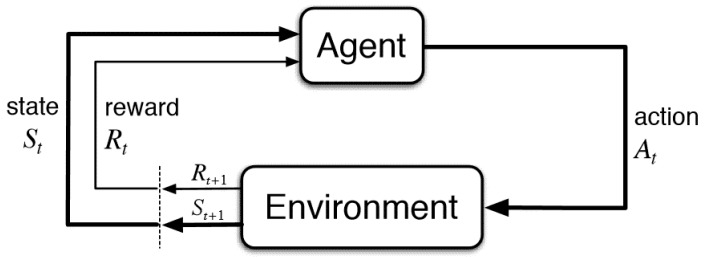


Figure 4.1: Approccio reinforcement learning

Come evidenziato in 4.1, l'algoritmo Q-Learning, internamente tiene traccia della matrice Q di dimensioni $N \times M$, con N pari al numero di stati S e M pari al numero di azioni A che l'agente può eseguire. Ad ogni passo l'agente, dato uno stato S, intraprende un'azione A e osserva l'esito di tale azione, mediante la ricezione di una ricompensa R in seguito all'interazione con l'ambiente e il nuovo stato in cui si troverà l'agente. Successivamente la matrice Q viene aggiornata in maniera opportuna allo scopo di massimizzare la ricompensa cumulativa che l'agente può ottenere in futuro.

$$\begin{aligned} Q_{target} &= r + \gamma(\max_{a'} Q(s', a')) \\ Q_{new} &= Q_{old} + \alpha(Q_{target} - Q_{old}) \end{aligned} \tag{4.1}$$

Figure 4.2: Equazioni aggiornamento funzione di qualità Q

Il parametro $\gamma \in [0, 1]$, intitolato **discount factor** rappresenta il bilanciamento tra la scelta di ricompense nell'immediato e ricompense future.

Il parametro $\alpha \in [0, 1]$, intitolato **learning rate**, permette di controllare la differenza tra valori attuali e precedenti di Q.

La scelta della prossima azione da parte dell'agente segue una logica definita *ϵ-greedy* in quanto, agendo su un parametro $\epsilon \in [0, 1]$ definito **fattore di esplorazione**, con una probabilità $1 - \epsilon$, viene scelta l'azione migliore dato lo stato corrente S e con probabilità ϵ viene scelta un azione random a_i in modo da favorire l'esplorazione dell'ambiente da parte dell'agente e scoprire possibili scelte che permettono di avere la ricompensa cumulativa futura più alta. Inizialmente è opportuno avere un fattore ϵ più elevato per favorire l'esplorazione e con l'aumentare degli episodi, cercare di ridurre questo parametro per favorire la scelta delle azioni che massimizzano la funzione di qualità.

4.2 Scelte progettuali

In questo progetto, gli stati della matrice Q sono ricavati dai sensori lidar, quantizzando la distanza di misurazione, definita dal parametro *SMAX*, su un numero di livelli definito dal parametro *MAX_STATES_LIDAR*. Invece di prendere i 3 raggi separatamente, per ridurre il numero di stati è stato decidere di combinare i due raggi laterali considerando la loro differenza. È stato deciso di usare 7 livelli di quantizzazione per un totale di 49 stati.

$$
\begin{aligned}
lat &= \frac{1}{2}(\frac{d_{left} - d_{right}}{SMAX} + 1) \in [0, 1) \\
f &= \frac{d_{front}}{SMAX + 1} \in [0, 1) \\
s_1 &= floor(lat * MAX\_STATES\_LIDAR) \\
s_2 &= floor(f * MAX\_STATES\_LIDAR) \\
s &= (MAX\_STATES\_LIDAR * s_2) + s_1
\end{aligned}
\tag{4.2}
$$

Figure 4.3: Decodifica distanza lidar in stati matrice Q

Per quanto riguarda invece in numero di azioni possibili nella matrice Q relativa al controllo dello sterzo, esso viene ricavato partendo dal massimo angolo di sterzo **MAX_THETA** e la risoluzione di quantizzazione definita da **ACTION_STEP** secondo la formula in figura 4.4. Nel progetto viene utilizzata una risoluzione di 5° e un massimo valore di sterzo pari a 40°.

$$
num\_actions = \frac{MAX\_THETA * 2}{ACTIONS\_STEP} + 1 \tag{4.3}
$$

Figure 4.4: Calcolo numero di azioni matrice Q

Approccio simile è stato seguito anche per il calcolo del numero di azioni della matrice Q relativa al controllo dell'accelerazione dove è stato scelto come massimo valore di accelerazione 0.2g definito dal parametro **MAX_A** e una risoluzione pari a 0.1, definita dal parametro **ACC_STEP**.

Quando l'applicazione viene eseguita nella modalità *inferenza*, il valore di ϵ viene messo quasi vicino a zero in modo da permettere all'agente di scegliere le azioni ottimali dalla matrice Q. Inoltre, prima di passare a tale modalità, viene letta da file la matrice Q e inizializzato l'algoritmo Q-Learning.

Per quanto riguarda la gestione della ricompensa che l'agente riceve in seguito all'interazione con l'ambiente sono state adottate 5 criteri:

- **crash / in pista**: se la macchina si trova in pista, ad ogni step viene sommato al reward totale un piccolo valore negativo *RWD_ALIVE*, mentre se la macchina si scontra contro i bordi della pista viene restituito il valore negativo molto alto *RWD_CRASH*.

- **distanza percorsa**: ogni qual volta la macchina supera una distanza minima definita dalla variabile *DIST_THRESHOLD_RWD*, alla ricompensa totale si aggiunge un reward positivo pari a *RWD_DISTANCE*

- **variazioni sterzo**: per ridurre le oscillazioni durante la guida, viene calcolata la differenza tra il valore di sterzo al passo precedente e quello al passo odierno, normalizzato rispetto al comando di sterzo massimo consentito e tale valore viene moltiplicato per un fattore *RWD_STEER_DELTA_GAP* e sommato al reward totale

- **comandi accelerazioni sbagliate**: se il valore di accelerazione scelto continua ad essere positivo una volta raggiunta la saturazione di velocità oppure negativo con macchina ferma, viene resituito come reward un contributo pari a *RWD_BAD_ACC*

- **mantenimento centro pista**: data la distanza dai lati della macchina rispetto alla pista, supponendo di avere i raggi lidar inclinati di 90°, se la differenza tra il valore destro e sinistro è inferiore a una certa soglia, l'agente viene premiato con un contributo pari a *RWD_ON_CENTRE* sommato a quello totale.

RWD_ALIVE	-1
RWD_CRASH	-100
RWD_DISTANCE	5
RWD_STEER_DELTA_GAP	-2
RWD_BAD_ACC	-50
RWD_ON_CENTRE	1

Table 4.1: Valori delle ricompense usate durante l'apprendimento

4.3 Schema apprendimento

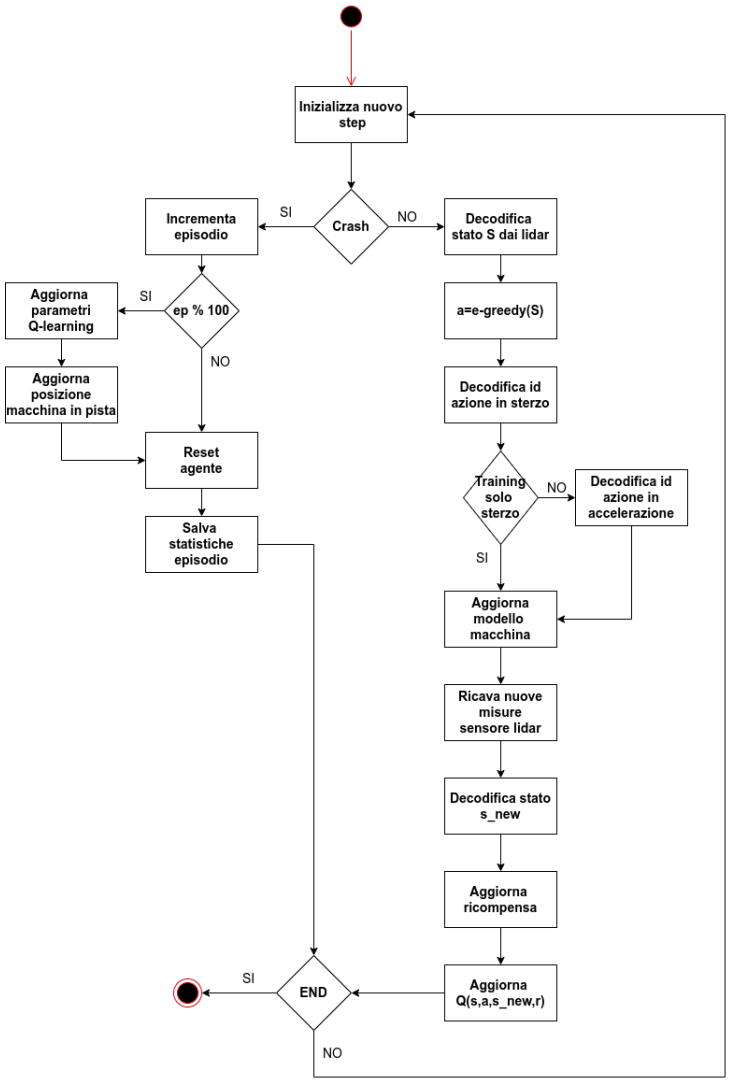
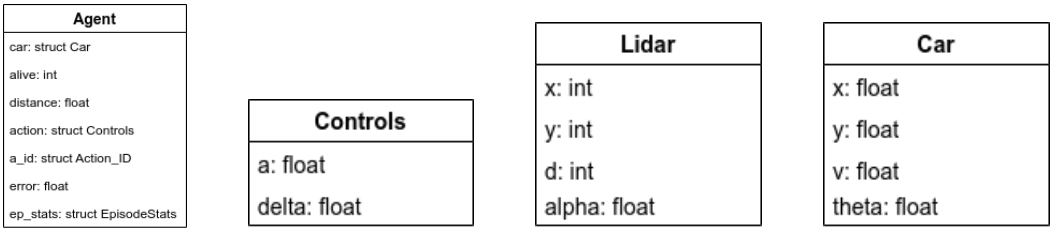


Figure 4.5: Schema apprendimento matrice Q

Accesso a risorse condivise

Alcune strutture globali condivise fra i task sono protette da *semafori* con protocollo *Priority Inheritance* per garantire consistenza.



(a) Agente (b) Controlli auto (c) Lidar (d) Macchina

Figure 5.1: Strutture dati protette da mutex

In particolare gli accessi alle strutture dati *Agent*, *Controls* e *Car* hanno l'accesso protetto mediante il semaforo **mux_agent** (figura 5.2a). In questo caso il task *agent_task* e *sensors_task* hanno accesso in scrittura mentre i task *display_task* e *comms_task* hanno accesso in lettura.

Nell'applicazione, è presente anche il semaforo **mux_sensors** per proteggere una variabile globale di tipo *Lidar* che si occupa di tenere traccia delle misurazioni dei sensori lidar durante il movimento della macchina (figura 5.2c). In questo caso il task *sensors_task* ha diritto di modifica di tale variabile globale mentre i task *agent_task* e *display_task* hanno accesso in lettura.

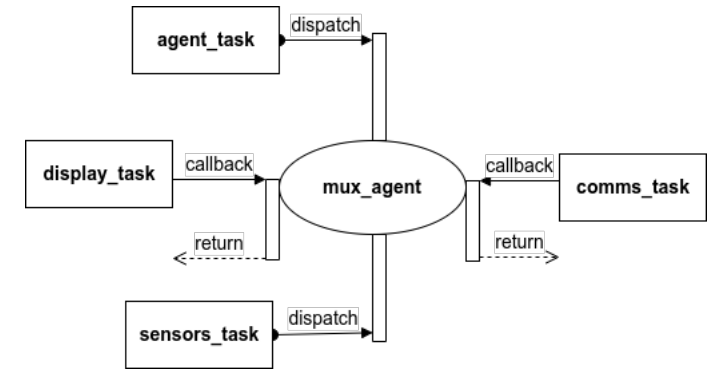
Un altro semaforo presente è **mux_velocity** che viene usato per proteggere la variabile globale *MAX_V_ALLOWED*. L'utente ha la possibilità di modificare la massima velocità da tastiera e poiché tale dato è accessibile sia da *agent_task* che *display_task* è necessario proteggerla.

L'ultimo semaforo presente è **mux_q_matrix** il cui ruolo consiste nel proteggere gli accessi alla libreria *qlearn*, usata per il reinforcement learning (figura 5.2d). Questo semaforo si comporta in maniera diversa rispetto ai semafori indicati in precedenza, poiché non protegge soltanto una variabile globale ma un'insieme di variabili. Alla libreria *qlearn* hanno accesso in scrittura il task *comms_task* ,per gestire il cambiamento delle modalità di apprendimento, passaggio training $\leftrightarrow$ inferenza e salvataggio su file cosi come caricamento da file della matrice Q.

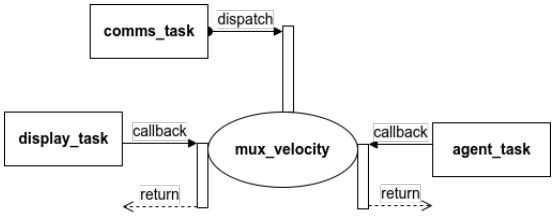
agent_task ha anch'esso accesso in scrittura alla libreria per la gestione della fase di apprendimento mentre in lettura l'accesso è consentito soltanto al task *display_task*.

Nella modalità single task, tali semafori sono superflui però nel funzionamento multi task sono molto importanti.

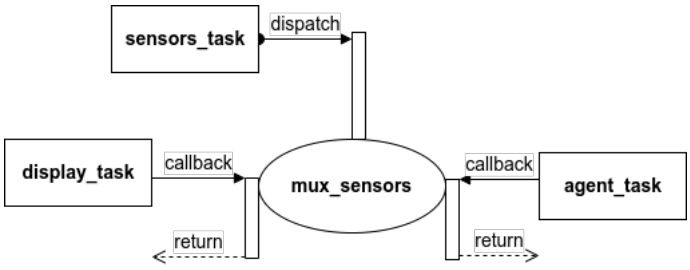
5.1 Schema task - risorse



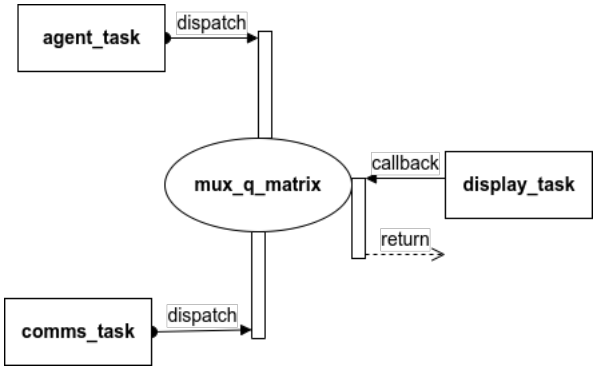
(a) Struttura Agent



(b) Massima velocità



(c) Struttura Lidar



(d) Libreria Q-Learning

Figure 5.2: Accesso protetto dei task

5.2 Parametri dei task

Task	Priorità	Periodo[ms]	Deadline rel.[ms]
display_task	5	25	25
sensors_task	10	20	20
comms_task	40	20	20
agent_task	10	20	20
learning_task	10	20	20

Table 5.1: Parametri dei task usati nell'applicazione

In fase di training era essenziale eseguire più simulazioni possibili in modo da ridurre i tempi di attesa della fase di apprendimento. A questo riguardo una scelta importante consiste nel fissare un periodo basso per *agent_task* e *learning_task* con lo scopo di aumentare la frequenza di aggiornamento del modello matematico della macchina. Ciò permette di evitare dei salti della posizione in pista troppo elevati e compromettere di conseguenza la simulazione stessa anche quando si accelerano i tempi di rendering.

Sono state riscontrate, a livello progettuale, alcune criticità per quanto riguarda la scelta delle priorità e periodi dei task *agent_task* e *sensors_task* poiché in modalità multi task possono creare dei rallentamenti che portano ad avere delle deadline miss. In particolare, per garantire la consistenza dell'algoritmo di apprendimento, in uno scenario multi task, prima di iniziare ogni step di apprendimento, tale task deve accedere alla struttura dati che salva i dati dei sensori e che viene aggiornata da *sensors_task*, per capire quale stato corrente della matrice Q usare. Come conseguenza logica, ha senso dare la priorità a quest'ultimo task, aumentando possibilmente anche il periodo stesso per garantire un aggiornamento dei dati più elevato. Tuttavia, tale task a sua volta ha bisogno di accedere all'orientazione della macchina, modificata da *agent_task* in fase di aggiornamento del modello matematico della macchina, il che comporta ad avere comunque delle deadline miss. Per questo motivo è stato deciso di assegnare ad entrambi i task la stessa priorità e periodo in modo da permettere a entrambi di avere tempi di esecuzione sulla CPU con politica *Round Robin*, con possibilità sporadiche di deadline miss. È stata valutata anche la possibilità di separare l'aggiornamento del modello matematico da quello di apprendimento, usando un nuovo task, però, per come è organizzato l'algoritmo di reinforcement learning, ci sarebbero stati lo stesso dei bloccaggi o possibili inconsistenze tra i dati. Inoltre assegnare un periodo più elevato a *agent_task* non è possibile poiché esso rischierebbe di eseguire l'apprendimento su dati dei sensori non consistenti.

Risultati

In seguito ai test effettuati è stato notato che l'apprendimento avviene in maniera quasi completa nella modalità di guida che prevede il controllo dello sterzo soltanto da parte dell'agente. Mentre nella modalità sterzo+accelerazione è stato notato che l'apprendimento non avviene in maniera corretta e completa. Un modo non completamente giusto che è stato trovato per migliorare l'apprendimento è stato quello di allenare l'agente prima in modalità *solo sterzo* e successivamente in modalità *sterzo + accelerazione*, facilitando l'apprendimento.

Inoltre in modalità *solo sterzo*, con una velocità massima di allenamento pari a 5 m/s, è stato notato che aumentando tale velocità l'agente riesce a gestire la macchina fino a un valore pari a circa 15 m/s, segno che ci potrebbe essere una sorta di overfitting dei reward rispetto alla pista e velocità di training.

Per quanto riguarda i parametri dell'algoritmo Q-Learning è stato notato che un ϵ elevato in combinazione con un fattore α elevato, rallenta il training in maniera sostanziale. I risultati migliori sono stati ottenuti con i valori $\epsilon \simeq 0.1$, $\alpha = 0.6, \gamma = 0.95, discount = 0.95$ e un apprendimento soddisfacente in circa 8000 episodi.

Elenco figure