PROGETTO DI LABORATORIO DI SISTEMI OPERATIVI

Progetto A.A. 2022/2023 – ADAS made trivial: rappresentazione ispirata alle interazioni in un sistema di guida autonoma

Informazioni autori e data di consegna:

- Jonathan Colombo, Matricola: 7011579, Email: jonathan.colombo@stud.unifi.it
- Athos Macaluso, Matricola: 7006636, Email: athos.macaluso@stud.unifi.it

- Data di consegna: 25/06/2023

Istruzioni dettagliate per compilazione ed esecuzione:

All'interno della cartella src è presente un makefile per cui è possibile compilarlo da terminale. Quindi, posizionarsi all'interno della cartella src tramite il terminale e lanciare il comando:

- $ make

Una volta compilato il makefile, è possibile lanciare l'esecuzione del programma indicando come entry point il file ./hmi e inserendo come parametro d'ingresso NORMALE o ARTIFICIALE.
Di seguito sono riportati gli esempi:

- $ ./hmi NORMALE
- $ ./hmi ARTIFICIALE

Una volta avviata l'esecuzione, si dovrà digitare uno dei seguenti comandi per far partire il sistema:

- INIZIO: inizio della navigazione.
- PARCHEGGIO: viene avviata la procedura di parcheggio del veicolo.
- ARRESTO: viene avviata la procedura di arresto del veicolo.

Sistema obiettivo e caratteristiche SW e HW:

Il progetto è stato realizzato tramite l'editor di testo Visual Studio Code, il compilatore gcc versione 11.3.0, il make versione 4.3 e la distribuzione Ubuntu Jammy Jelfish versione 22.04.
Il progetto è stato testato su un portatile MSI aventi le seguenti caratteristiche:

- Processore: 11th Gen Intel© Core™ i5-1135G7 @ 2.40GHz × 4
- Memoria principale: 15.3 GiB
- Memoria secondaria: 518.4 GB
- Scheda video: Intel Corporation TigerLake-LP GT2 [Iris Xe Graphics]

Indicazione degli elementi facoltativi realizzati:

#	**Elemento facoltativo**	**Realizzato (Si/No)**	**Descrizione dell'implementazione con indicazione del metodo/i principale/i**
1	Ad ogni accelerazione, c'è una probabilità di 10^(-5) che l'acceleratore fallisca. In tal caso, il componente throttle control invia un segnale alla Central ECU per evidenziare tale evento, e la Central ECU avvia la procedura di ARRESTO	Si	void handleIncrement(FILE *randomFile, FILE *throttleLog, int ecuFileDescriptor): legge dalla pipe se è presente la stringa INCREMENTO e nel caso converte la stringa in numero e scrive nel file di log throttle.log. Successivamente legge un numero casuale da 0 a 9999 (da dev/random o da randomArtificiale.binary) e se il numero casuale è zero invia un segnale al processo padre e termina con un errore.
2	Componente "forward facing radar"	Si	File forwardingfacingradar.c
3	Quando si attiva l'interazione con park assist, la Central ECU sospende (o rimuove) tutti I sensori e attuatori, tranne park assist e surround view cameras.	Si	File: ecu.c. Funzione: int main(int argc, char *argv[]) riga 158: se viene rilevata dalla socket la stringa PARCHEGGIO e il comando di input 3, si frena fino a quando la velocità non raggiunge il valore 0. Dopodichè,

			si fermano tutti i processi designati dal testo dell'esercizio con segnale SIGSTOP e e si genera il processo figlio Park Assist e si esegue.
4	Il componente Park assist non è generato all'avvio del Sistema, ma creato dalla Central ECU al bisogno.	Si	File: ecu.c. Funzione: int main(int argc, char *argv[]) riga 158: il processo figlio Park Assist viene generato solo in questo punto dal processo padre Ecu.
5	Se il componente surround view cameras è implementato, park assist trasmette a Central ECU anche i byte ricevuti da surround view cameras.	No	Questa operazione è effettuata direttamente da parkAssist.c
6	Componente "surround view cameras"	Si	File surroundviewcameras.c
7	Il comando di PARCHEGGIO potrebbe arrivare mentre i vari attuatori stanno eseguendo ulteriori comandi (accelerare o sterzare). I vari attuatori interrompono le loro azioni, per avviare le procedure di parcheggio.	Si	File hmiInput.c. Funzione: int main(int argc, char *argv[]) riga 43. Tramite il terminale si può inserire come comando INIZIO, PARCHEGGIO O ARRESTO. Nel caso venga intercettata la stringa PARCHEGGIO, il comando viene inviato alla ECU tramite pipe.
8	Se la Central ECU riceve il segnale di fallimento accelerazione da "throttle control", imposta la velocità a 0 e invia all'output della HMI un messaggio di	Si	File: ecu.c. Funzione: void throttleFailure(). Quando l'acceleratore rileva un fallimento, manda un segnale SIGUSR1 a ECU. Quest'ultima imposta la velocità a 0, ferma brake by wire con un segnale SIGSTOP,

	totale terminazione dell’esecuzione		manda un segnale al processo hmi e dopodichè richiama la funzione endProgram(int __sig) che provvede a fermare tutti quanti i processi figli e ad eliminare tutte quante le pipe e le socket attive. Infine termina con un errore.

Progettazione ed implementazione:

Il processo **HMI** è “l’entry point” del sistema e all’avvio genera un terminale di input da cui l’utente può interagire con il sistema inviando i comandi INIZIO, PARCHEGGIO o ARRESTO e il processo della Central ECU. Il processo HMI input comunica con la Central ECU attraverso una pipe con la quale invia I comandi digitati dall’utente.

La **Central ECU** viene generata dal processo padre HMI e provvede a diverse funzioni: imposta la velocità a zero all’inizio delle navigazione, crea una pipe per interagire con l’HMI output, riceve i comandi da HMI input attraverso una pipe in lettura, genera i processi figli ovvero attuatori e sensori, crea le pipe per comunicare agli attuatori gli input dei vari comandi ricevuti dai sensori, crea una socket iterativa con cui leggere/scrivere i dati inviati dai sensori, si preoccupa di gestire tutti i controlli e gli errori relativi ai dati ricevuti dai sensori ed eventualmente poi li filtra e li rispedisce verso gli attuatori e si preoccupa della gestione di terminazione del sistema.

L’attuatore **Steer By Wire** viene generato dalla ECU all’avvio del programma e riceve tramite una pipe in lettura i comandi di sterzata da parte del processo ECU. Il processo steer by wire implementa le funzionalità di: impostare un gestore dei segnali di fallimento, creare e scrivere sul file di log **steer.log** quando riceve correttamente la stringa **“SINISTRA”** o **“DESTRA”** altrimenti scrive una volta al secondo **“NO ACTION”,** apre la pipe in lettura non bloccante per ricevere i dati dalla Central ECU, gestisce correttamente la chiusura del file di log e la terminazione corretta del processo.

L’attuatore **Throttle By Control** viene generato dalla Central ECU all’avvio del programma e riceve tramite una pipe in lettura i comandi di accelerazione/decelerazione da parte del processo ECU. Il processo throttle by control si preoccupa di: controllare gli argomenti passati dall’utente all’avvio del sistema (**NORMALE** o **ARTIFICIALE**) per poter utilizzare la sorgente di numeri casuali **dev/random** o il file di numeri casuali **randomArtificiale.binary** per poter calcolare la probabilità di 10^(-5) di fallimento dell’acceleratore, crea e scrive sul file di log **throttle.log** quando riceve correttamente la stringa di **“INCREMENTO”**, apre la pipe in lettura per ricevere i dati dalla

Central ECU, gestisce correttamente la chiusura del file di log e la terminazione corretta del processo inviando un segnale al processo padre.

L'attuatore **Brake By Wire** viene generato dalla ECU all'avvio del programma e riceve tramite una pipe in lettura i comandi di frenata da parte del processo ECU. Il processo brake by wire si preoccupa di: aprire la pipe di comunicazione in lettura con la Central ECU, creare e scrivere sul file di log **brake.log** quando riceve correttamente la stringa **"FRENO"**, scrivere sul file di log **"ARRESTA AUTO"** quando il componente riceve il segnale di arresto da parte di ECU, gestire tramite dei gestori dei segnali il segnale eventuali errori e chiudere il file di log e terminare correttamente.

Il sensore **Front windshield camera** viene generato dalla Central ECU all'avvio del programma e legge da frontCamera.data una riga ogni secondo fino al raggiungimento della fine del file e invia alla Central ECU tramite socket la stringa letta. Inoltre si preoccupa di: di creare e scrivere sul file di log **camera.log** i comandi letti, impostare un gestore dei segnali in caso di errore o fallimento delle operazioni, chiudere correttamente il file di log e gestire correttamente la terminazione inviando un segnale al processo padre.

Il sensore **Forward Facing Radar** viene generato dalla Central ECU all'avvio del programma ed implementa le seguenti funzionalità: crea un file di log denominato **radar.log**, scrive sul file di log l'avvio di lettura e imposta un gestore dei segnali in caso di fallimento o di errore durante l'esecuzione.

Il sensore **Park Assist** viene generato dalla Central ECU come processo nel momento in cui viene letta la stringa dal file frontCamera.data **"PARCHEGGIO"** o viene digitata e inserita da tastiera dall'utente attraverso l'HMI Input. Prima della generazione del processo figlio, viene eseguito brake by wire fino a quando non si raggiunge la velocità zero e vengono fermati tutti gli altri attuatori e sensori. Il processo park assist si preoccupa di: creare il file di log **assist.log** su cui scrivere le coordinate lette dalla sorgente /dev/urandom o urandomArtificiale.binary (in base alla modalità specificata dall'utente all'avvio del sistema), gestisce la procedura di parcheggio in un intervallo di tempo di 30 secondi controllando se non sono presenti le coordinate di fallimento e in tal caso riavvia la procedura di parcheggio, invia i dati letti alla Central ECU attraverso una comunicazione socket e gestisce la chiusura del file e la terminazione del processo inviando correttamente il segnale di arresto al processo padre.

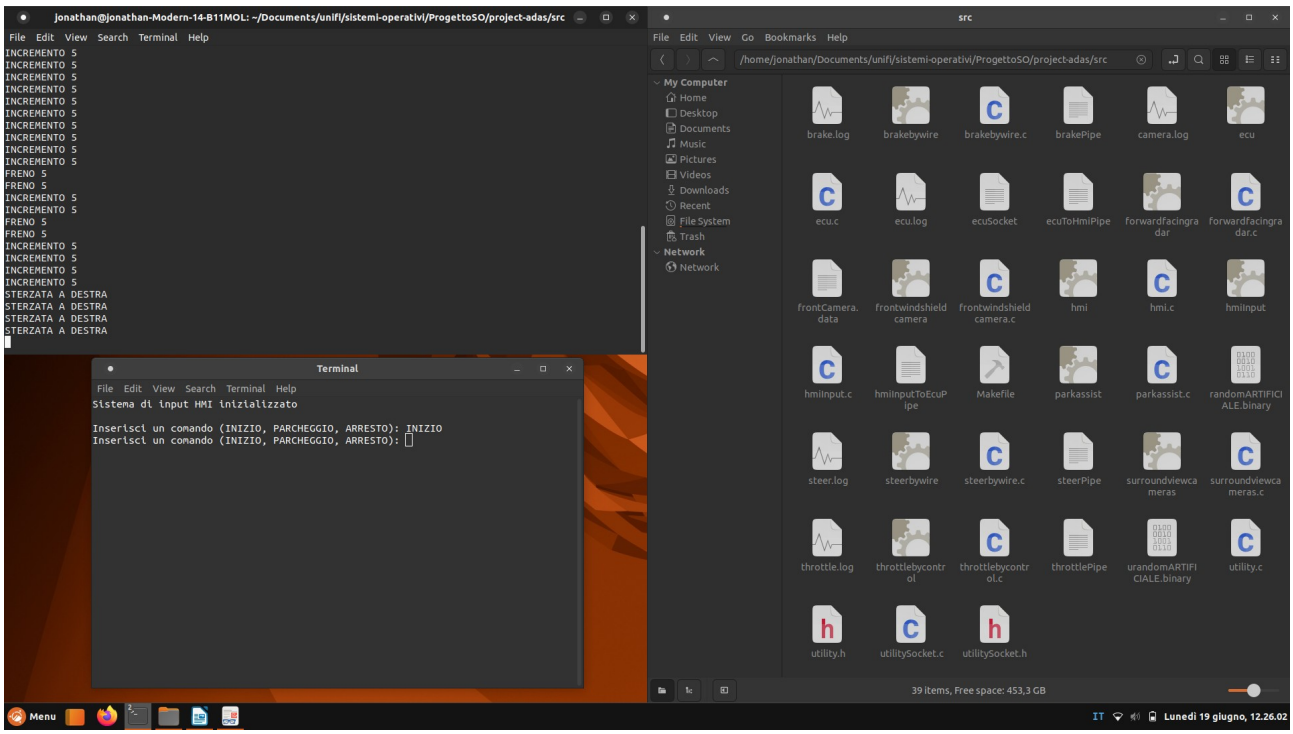
Il sensore **Surround View Cameras** è stato implementato ma non è stato portato a termine in quanto le funzionalità per la procedura di parcheggio sono state tutte implementate in park assist.

Note implementative: ogni componente è stato rappresentato con almeno un processo e non è necessaria la modalità di esecuzione da root.

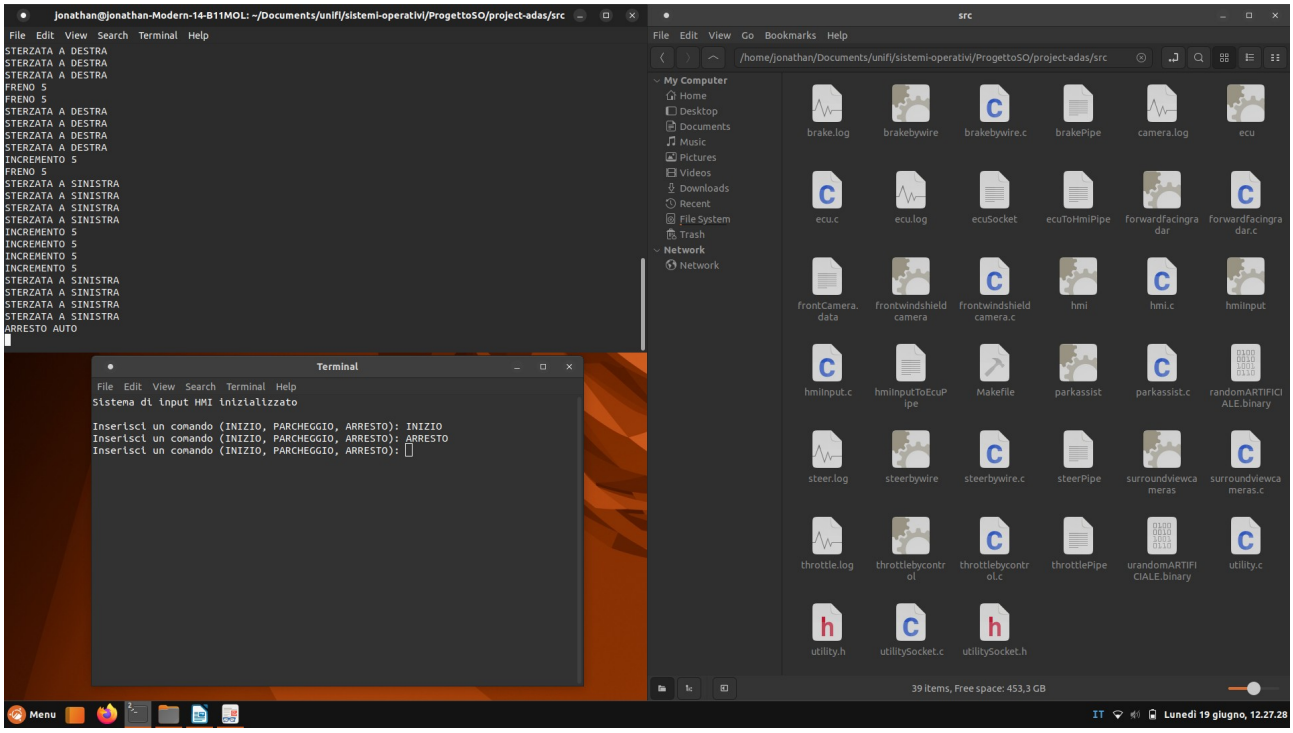
Esecuzione:

Di seguito sono riportati alcune schermate di cattura di esempio di esecuzione del programma:

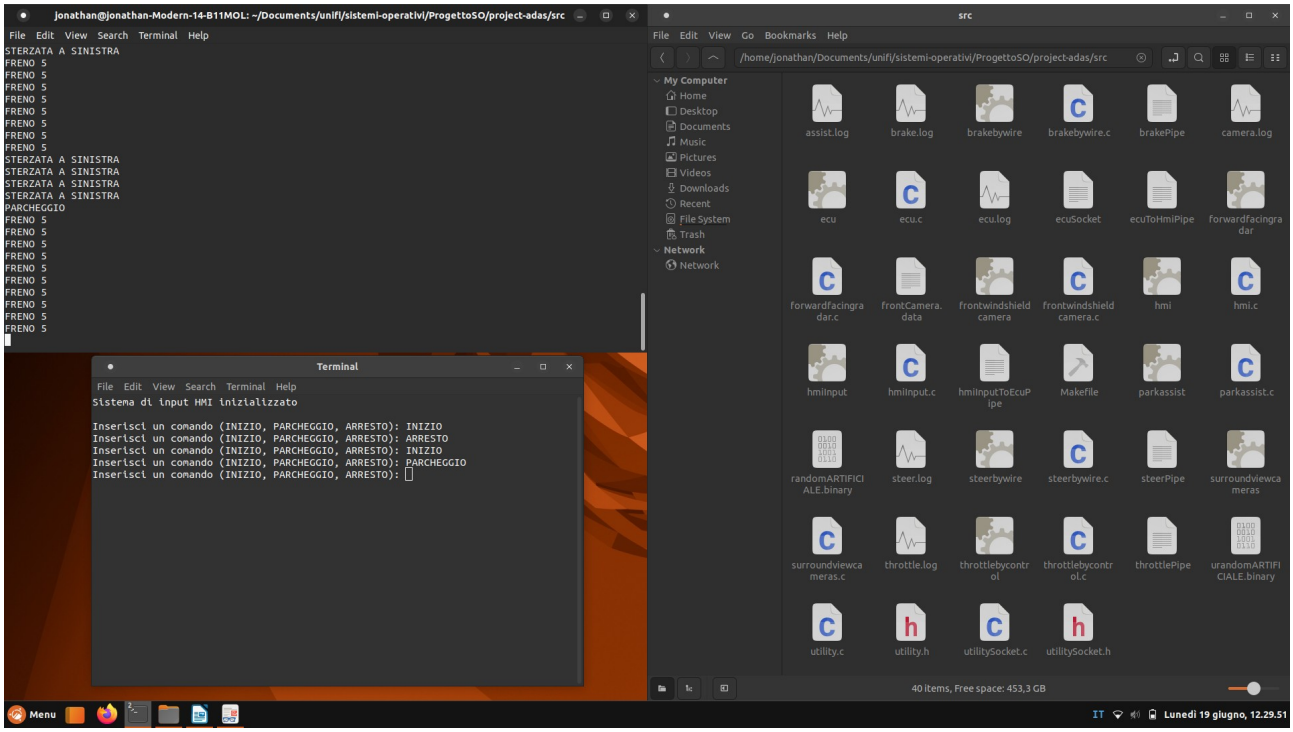
- Modalità di esecuzione di avvio ./hmi NORMALE e comando INIZIO



- Inserimento input ARRESTO che sospende tutti i processi (per far ripartire l’esecuzione digitare di nuovo INIZIO, nel caso il comando ARRESTO venga letto da un sensore il programma non sospende tutti i processi ma imposta la velocità a zero e prosegue l’esecuzione):



- Inserimento input PARCHEGGIO che avvia la procedura di parcheggio:



Note di esecuzione: la modalità esecuzione ARTIFICIALE è analoga visivamente in quanto cambiano solo le sorgenti da cui vengono letti o dati ma il funzionamento è equivalente.
I file di log non vengono eliminati a fine esecuzione ed è possibile consultarli aprendoli con un editor di testo.
Nel caso si volesse rimuovere i file di log e gli eseguibili è possibile farlo lanciando il comando da terminale:

- $ make clean