

RELAZIONE

PROGETTO SISTEMI OPERATIVI

MATRICOLA 7029446 Wu Jinkang
E-Mail: jinkang.wu@stud.unifi.it
MATRICOLA 7030209 Giulio Morandini
E-Mail: giulio.morandini1@stud.unifi.it
MATRICOLA 7072914 Lucia Huang
E-Mail: lucia.huang1@stud.unifi.it
Data di consegna: 15/07/2023

RICHIESTA:

L'obiettivo del progetto è costruire un'architettura, estremamente stilizzata e rivisitata, per sistemi ADAS, descrivendo possibili interazioni e alcuni comportamenti tra componenti in scenari specifici. Dovrà comprendere varie componenti, suddivise tra sensori (front windshield camera, forward facing radar, park assist, surround views cameras), attuatori (Steer By Wire, throttle control, brake by wire), il tutto controllato da una componente Central ECU e dovrà interfacciarsi con l'utente attraverso un'interfaccia Human-Machine interface. I dettagli delle varie componenti sono espressi nel testo. Il sistema potrà essere eseguito in due modalità: NORMALE ed ARTIFICIALE. Nel caso della modalità di avvio NORMALE: i componenti forward facing radar, park assist, surround view cameras, leggono i dati da /dev/urandom, mentre nella modalità ARTIFICIALE componenti forward facing radar, park assist, surround view cameras, leggono i dati invece che da /dev/urandom, dal file urandomARTIFICIALE.binary.

ISTRUZIONI PER COMPILAZIONE ED ESECUZIONE:

Per la fase di compilazione, ed esecuzione, è stato creato un file apposito chiamato Makefile, il quale contiene tutte le istruzioni riguardanti la fase di compilazione, e quindi della creazione dei file oggetto ed il loro assemblaggio. Ogni file eseguibile corrisponde a una componente del sistema, il main corrisponde alla componente Central ECU.

Passi da svolgere per la compilazione ed esecuzione:

1. Aprire due shell e posizionarsi all'interno della cartella principale del programma.
2. Digitare make all'interno della shell (basta in una sola)
3. Per la modalità:
 - ARTIFICIALE: digitare 'bin/main ARTIFICIALE'
 - NORMALE: digitare 'bin/main NORMALE'in una delle due shell.
4. Digitare 'bin/hmiOutput' nell'altra shell.

Il punto 2 fa sì che vengano creati tutti i file oggetto di cui il programma necessita e tutti i file eseguibile.

Inoltre nel file Makefile è presente il comando 'clean' (digitare 'make clean' all'interno della cartella principale), il quale serve ad eliminare tutti i file oggetto,

eseguibile, log, oltre ad eliminare i file log. Questo per poter riprovare ad eseguire in maniera rapida il programma, senza andare ad eliminare i file a mano.

CARATTERISTICHE SW E HW:

Il progetto è stato scritto usando visual studio code, e testato su tre macchine diverse: la prima è una macchina virtuale gestita da Oracle Virtual Box con sistema operativo Ubuntu (64 bit) versione 23.04 (versione kernel: Linux 6.2.0-24-generic), con 5 gb di memoria di base, processore AMD ryzen 5 1400x3; La seconda è un macbook air con sistema operativo macOS versione Ventura 13.2.1, con processore m1 e 16 gb di ram; La terza è una macchina virtuale gestita da Oracle Virtual Box con sistema operativo Debian (amd64) versione 11.6.0 (versione kernel: Linux 5.10.0-21-amd64) con 4 gb di ram, processore Intel Celeron N4000.

ELEMENTI FACOLTATIVI:

#	Elemento Facoltativo	Realizzato (SI/NO)		Descrizione dell'implementazione con indicazione del metodo/i 5 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		Nel file src/throttleControl.c è presente un metodo <code>throttleBreaks(int fd)</code> che legge un intero casuale dal file <code>/dev/random</code> o <code>/res/randomARTIFICIALE..binary</code> per poi simulare un fallimento con probabilità di 10^{-5} . Se l'acceleratore fallisce manda un segnale SIGUSR1 al Central ECU che verrà catturato dal signal handler <code>throttleControlBrokeHandler</code> nel file <code>src/main.c</code> , per poi terminare tutti i processi.
2	Componente "forward facing radar"	SI		Si trova nel file <code>src/forwardFacingRadar.c</code> , è avviato dal main con la modalità data dall'utente e inizialmente cerca di connettersi al Central ECU. Successivamente, a seconda della modalità, sceglie da quale file leggere i dati. La logica della lettura è nella funzione <code>read8(int fd)</code> che si trova nel file <code>src/util.c</code> . Infine trasmette gli 8 byte letti a Central ECU con la funzione

				sendC(int fd, char *buffer), che si trova nel file src/conn.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		Sfrutta la funzione slowingDownToParking(unsigned int brakeByWireReady, int logFd, char *mode, int centralFd) che si trova nel file src/main.c Manda il segnale SIGTERM a tutti i processi, sensori e attuatori, tranne park assist e surround view cameras.
4	Il componente Park assist non è generato all'avvio del Sistema, ma creato dalla Central ECU al bisogno.	SI		La funzione sfruttata è slowingDownToParking(unsigned int brakeByWireReady, int logFd, char *mode, int CentralFd) che si trova nel file src/main.c. Esegue execParkAssist(char *mode), che mette in esecuzione il file eseguibile di park assist in un processo figlio generato tramite fork(), che poi attenderà una richiesta di connessione.
5	Se il componente surround view cameras è implementato, park assist trasmette a Central ECU anche i byte ricevuti da surround view cameras.	SI		Si trova nel file src/surroundViewCameras.c. Il componente surround view cameras e il lancio di tale processo avviene nel file src/parkAssist.c, dove viene messo in esecuzione da un suo processo figlio creato tramite fork(). La comunicazione tra di loro è tramite socket AF_UNIX, una volta che si è stabilita la connessione ogni secondo, oltre a leggere i 8 byte del park assist, prova a leggere anche i 8 byte del surround view cameras per poi mandare i byte letti al Central ECU.
6	Componente "surround view cameras"	SI		Si trova nel file src/surroundViewCameras.c, all'avvio cerca di connettersi al park assist per inviare i byte letti.
7	Il comando di PARCHEGGIO potrebbe arrivare mentre i vari attuatori stanno	SI		Per effettuare il parcheggio è stata aggiunta una variabile parking, che viene posta a 1 quando l'utente digita PARCHEGGIO o quando viene letto dalla front camera. Quindi punto la funzione

	e eseguendo ulteriori comandi (accelerare o sterzare). I vari attuatori interrompono le loro azioni, per avviare le procedure di parcheggio.			slowingDownToParking(unsigned int brakeByWireReady, int logFd, char *mode, int centralFd) manda il segnale SIGTERM a tutti gli altri processi, sensori e attuatori, in caso la variabile diventasse 1.
	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 totale terminazione dell'esecuzione	SI		La funzione throttleControlBrokeHandler(int sig) Si trova nel file src/main.c. Si occupa, appena riceve il segnale SIGUSR1, di impostare la velocità a zero attraverso la variabile globale, e manda il segnale SIGTERM a tutti i processi componenti. Infine, termina l'esecuzione di Central ECU.

SCELTE PROGETTUALI:

Per risolvere il problema posto nella richiesta, è stato innanzitutto analizzato il problema per capire cosa venisse richiesto. Successivamente abbiamo analizzato come strutturare il codice nelle varie cartelle e file (dove andarlo a scrivere e dove salvare i file) ed in seguito si è passati alla scrittura effettiva del codice.

Il processo HMI è il punto di ingresso 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. Per la parte di output è necessario un secondo terminale, che comunica direttamente con la Central ECU e ci fa vedere a video i vari cambiamenti alla velocità e le sterzate a destra o a sinistra.

La nostra Central ECU inizialmente cerca di connettersi a tutti i componenti (sensori, attuatori e HMI Input) tramite il socket.

La Central ECU è stata progettata come un server iterativo, non crea nessun processo figlio per gestire le richieste, tutto è nel processo main.

Da evidenziare il fatto che la connessione con i processi una volta stabilita è salvata fino alla fine del programma; quindi, non verrà mai toccata, a parte per il riavvio del park assist.

Il socket nella parte di Central ECU è non bloccante, per verificare l'arrivo di un nuovo messaggio controlla la lunghezza del buffer letto, se è presente un messaggio passa all'elaborazione dei dati ricevuti.

Per far rispettare i timer la ECU è sempre attiva, c'è solo un sleep di 0.1 secondo per non esaurire la CPU.

Quando cerchiamo di inviare un messaggio alla Central ECU il componente si mette in attesa di una conferma, finché non viene mandata il componente rimane in attesa.

Per varie situazioni particolari ci sono dei flag, ad esempio per il parcheggio avremo flag parking, per il pericolo letto dal Front Windshield Camera invece suspend.

Le variabili dei file descriptor sono definite globali per poi rilasciare nei gestori di segnali SIGTERM.

L'attuatore Steer By Wire viene generato dalla Central ECU all'avvio del programma e riceve tramite una socket in lettura non bloccante i comandi di sterzata da parte del processo Central 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", riceve i dati dalla Central ECU attraverso una socket, 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 socket i comandi di accelerazione 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 calcolare la probabilità di 10^{-5} di fallimento dell'acceleratore, crea e scrive sul file di log throttle.log quando riceve correttamente la stringa "INCREMENTO 5", 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 Central ECU all'avvio del programma e riceve tramite una socket i comandi di frenata da parte del processo ECU. Il processo brake by wire si preoccupa di: creare la socket di comunicazione con la Central ECU, creare e scrivere sul file di log brake.log quando riceve correttamente la stringa "FRENO 5", scrivere sul file di log "ARRESTO 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.

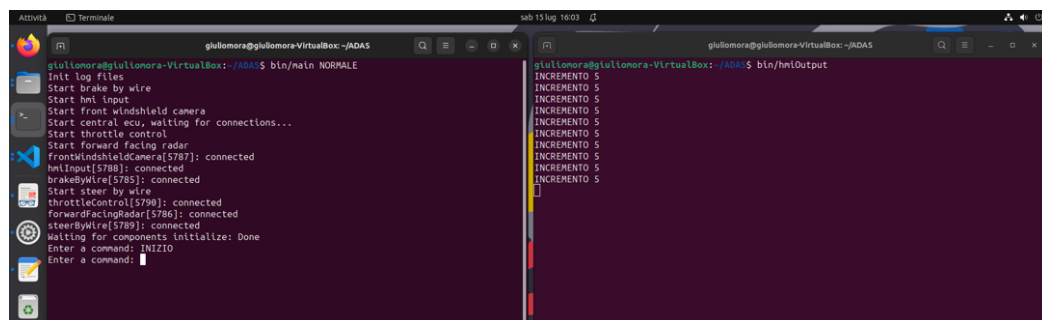
Il sensore Park Assist viene generato dalla Central ECU come processo nel momento in cui viene letta la stringa dal file frontCamera.data "PARCHEGGIO" o viene digitata e inserita da tastiera dall'utente attraverso l'HMI Input. Prima della generazione del processo figlio, viene eseguito Brake By Wire 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) che sono convertiti in numeri esadecimali in stringa, 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 viene generato dalla Park Assist all'avvio del programma ed implementa le seguenti funzionalità: crea un file di log denominato cameras.log, comunica con Park Assist tramite socket e 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.

ESECUZIONE:

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

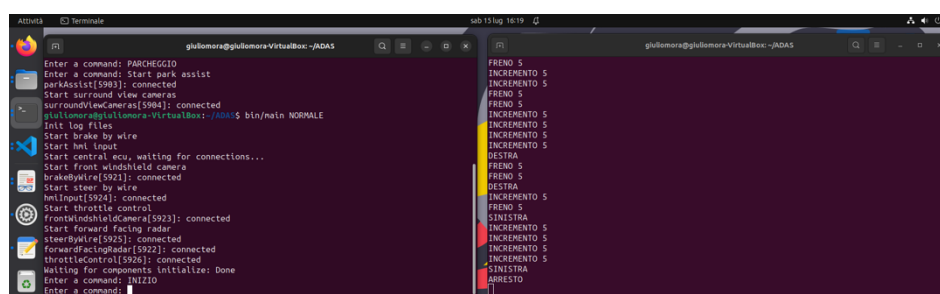
Modalità di esecuzione di avvio./hmi NORMALE e comando INIZIO:



```
giuliomora@giuliomora-VirtualBox: ~/ADAS$ bin/main NORMALE
Init log files
Start brake by wire
Start hmi input
Start front windshield camera
Start central ecu, waiting for connections...
Start throttle control
Start forward facing radar
frontWindshieldCamera[5787]: connected
hmiInput[5788]: connected
brakeByWire[5785]: connected
Start steer by wire
throttleControl[5790]: connected
forwardFacingRadar[5786]: connected
steerByWire[5789]: connected
Waiting for components initialize: Done
Enter a command: INIZIO
Enter a command:

giuliomora@giuliomora-VirtualBox: ~/ADAS$ bin/hmiOutput
INCREMENTO 5
INCREMENTO 5
INCREMENTO 5
INCREMENTO 5
INCREMENTO 5
INCREMENTO 5
INCREMENTO 5
INCREMENTO 5
INCREMENTO 5
INCREMENTO 5
```

Quando viene letto un comando ARRESTO la velocità diventa zero, e l'esecuzione viene sospesa, inoltre il sistema attende che venga inserito in input nuovamente INIZIO:



```
giuliomora@giuliomora-VirtualBox: ~/ADAS$ bin/main NORMALE
Enter a command: PARCHEGGIO
Enter a command: Start park assist
parkAssist[5983]: connected
Start surround view cameras
surroundViewCameras[5984]: connected
giuliomora@giuliomora-VirtualBox: ~/ADAS$ bin/main NORMALE
Init log files
Start brake by wire
Start hmi input
Start front windshield camera
brakeByWire[5923]: connected
Start steer by wire
hmiInput[5924]: connected
Start throttle control
frontWindshieldCamera[5923]: connected
steerByWire[5923]: connected
forwardFacingRadar[5922]: connected
throttleControl[5926]: connected
Waiting for components initialize: Done
Enter a command: INIZIO
Enter a command:

giuliomora@giuliomora-VirtualBox: ~/ADAS$ bin/hmiOutput
FRENO 5
INCREMENTO 5
INCREMENTO 5
FRENO 5
FRENO 5
INCREMENTO 5
INCREMENTO 5
INCREMENTO 5
FRENO 5
FRENO 5
DESTRA
INCREMENTO 5
FRENO 5
SINISTRA
INCREMENTO 5
INCREMENTO 5
INCREMENTO 5
INCREMENTO 5
SINISTRA
ARRESTO
```

The screenshot shows two side-by-side terminal windows from the Kali Linux desktop environment. The left window displays the output of running './bin/main NORMAL' at the prompt 'gslu@morag:gslu/morora-VirtualBox: ~/ADAS'. It lists various components being initialized or connected, such as 'ent log files', 'Start brake by wire', 'Start hmi input', 'Start central ecu, waiting for connections...', 'Start front windshield camera', 'brakeByWire[5921]: connected', 'Start steer by wire', 'hmInput[5924]: connected', 'Start throttle control', 'frontWindshieldCamera[5923]: connected', 'Start forward facing radar', 'steerByWire[5925]: connected', 'forwardFacingRadar[5922]: connected', 'throttleControl[5926]: connected', and 'Waiting for components initialize: Done'. It also shows commands like 'Enter a command: INIZIO' and 'parkAssist[5960]: connected'. The right window shows a series of status messages: 'FRENO S', 'FRENO S', 'FRENO S', 'FRENTO S', 'SINISTRA', 'FRENO S', 'FRENO S', 'SINISTRA DESTRA', 'FRENO S', 'FRENO S', 'FRENO S', 'PAROLEGGIO FRENO S', 'FRENO S', 'FRENO S', 'FRENO S', 'FRENO S', and 'FRENO S'.

The screenshot shows two side-by-side terminal windows from a Linux desktop environment. The left window's title bar reads "gslumora@gslumora-VirtualBox: /ADAS bin/main NORMALE". It displays a series of status messages for various sensors and systems, such as "Start log files", "Start brake by wire", "Start forward facing radar", "Start central ecu, waiting for connections...", "Start steer by wire", "brakeByWire[5874]: connected", "Start front windshield camera", "steerByWire[5873]: connected", "forwardFacingRadar[5875]: connected", "Start hml input", "frontWindshieldCamera[5876]: connected", "Start throttle control", "hmlInput[5877]: connected", "throttleControl[5879]: connected", and "Waiting for components initialize: Done". At the bottom, it prompts "Enter a command: INIZIO" and "Enter a command: ARRESTO". The right window's title bar reads "gslumora@gslumora-VirtualBox: /ADAS bin/hwOutput". It displays a repeating sequence of sensor status updates: "INCREMENTO 5" followed by "INCREMANTO 5" for each of the five sensors listed in the left window. At the bottom, it prompts "Enter a command: INIZIO" and "Enter a command: ARRESTO". Both terminals have a dark background with light-colored text.

[illegible]

Nota: la modalità esecuzione ARTIFICIALE è analoga visivamente in quanto cambiano solo le sorgenti da cui vengono letti i dati ma il funzionamento è equivalente.