Informazione

Autore: **Daniil Radchanka**

Email: **daniil.radchanka@unifi.it**

Matricola: **7079901**

Autore: **Anastasia Moskalenko**

Email: **anastasia.moskalenko@unifi.it**

Matricola: **7015595**

Data di consegna: **25.06.2023**

Compilazione ed esecuzione

Per compilare il programma in cartello del progetto c'è il *Makefile.* Utilizando il MakeFile si puo:

1. Compilare in modalita normale: *make*
2. Compilare per il Debug: *make DEBUG=1*
3. Pulire il risultato di compilazione: *make clean*
4. Pulire i file log: *make cleanLogs*

Dopo di aver compilato il programma dobbiamo eseguire *Human machine interface (HMI)*. Ci serviranno due terminali per la scrittura e lettura dei commandi. Eseguendo *HMI* per la modalita di scrittura *HMI* lancia anche la *Central ECU* (*ECU*).

Commandi per esecuzione di HMI:

1. HMI per la lettura con la modalita Normale:
./hmi NORMALE
2. HMI per la lettura con la modalita Artificiale:
./hmi ARTIFICIALE
3. HMI per la scrittura: *./hmi -w*

Implementazione dell programma

Per sviluppare il programma avevo utilizzato il Windows 11 con Windows Subsystem for Linux (WSL) installato. Avevo utilizzato Ubuntu con la versione di WSL 2. Avendo scelto tal sistema ho trovate dei problemi mentre implementavo la comunicazione tra i componenti, perché WSL 2 non supporta il *UNIX socket*, cioè dovevo usare invece il *INET socket.*

L'idea principale che avevo in mente per implementare la comunicazione di comandi tra i vari componenti e stata la generalità e affidabilità. Perciò i componenti sono suddivisi in server e in client. Il componente server, quello che riceve i comandi/messaggi tiene il suo *socket* sempre aperto fino alla sua terminazione. Il client invece per ogni richiesta che lui manda crea un nuovo *socket* per la connessione al server e dopo di aver scritto la richiesta chiude subito il *socket*.

I componenti server sono seguenti:

- *ECU*
- *HMI writer*
- *Brake By Wire (BBW)*

- *Steer By Wire (SBW)*
- *Throttle Control (TC)*
- *Park Assist (PA)*

I component client sono seguenti (con la freccia al suo server):

- *HMI reader → ECU*
- *Front Windshield Camera (FWC) → ECU*
- *Forward Facing Radar (FFR) → ECU*
- *PA → ECU*
- *ECU → HMI writer*
- *ECU → BBW*
- *ECU → SBW*
- *ECU → TC*
- *Surround View Cameras (SVC) → PA*

Se il client non riesce a mandare la richiesta allora la puoi riprovare a fare in qualche secondo più tardi (come fa l'*ECU*) oppure pretendere come se non avessi l'errore di mandata di richiesta e continuare il suo procedimento (come fanno *SBW, BBW* e etc.).

Se qualche componente vuole loggare qualche messaggio oppure anche l'errore avevo implementato dei metodi per gestire tal funzionamento. Il file di log viene salvato nella *directory log* e gli errori vengono salvati nei file corrispondenti al nome del componente nella *directory eLog*.

Per migliorare la leggibilità dell'codice avevo cercato di suddividere il codice in file diversi in modo di ottenere meno possibile le ridondanze del codice. Piu precisamente:

- la gestione del *socket* e stata messa in *Ipc.c/.h*
- la gestione del *logging* e stata messa in *Logger.c/.h*
- definizione di tutti costanti in *Consts.h*
- *FilePathProvider* per ricavare la directory sorgente con il file name aggiunto
- DateProvider per ricavare la data sorgente
- I vari componenti sono stati messi in suoi .h/.c file più se avevano bisogno definivano anche .h per la gestione di tipi di comandi che possono ricevere con l'informazione su i client che la possono utilizzare.
- Central ECU era separata anche nel altri file .h/.c per la gestione di comunicazione con gli altri componenti tramite i *socket* oppure tramite i segnali.

Inizialmente vengono lanciati solo *HMI reader*, *HMI writer* e *Central ECU*. Gli altri componenti crea Central ECU al bisogno. Per esempio per quando

auto parte Central ECU crea tutti attuatori e sensori tranne il *PA* e *SVC*. Quando viene fatta la richiesta del parcheggio viene creato il *PA* (senza attivazione) e finché l'auto non si ferma completamente Central ECU lascia andare i suoi sensori ed attuatori, ma riceve solo l'informazione necessaria per la procedura di parcheggio. Dopo di aver fermato l'auto Central ECU termina l'esecuzione di sensori e attuatori attivando il *PA.* Il *PA* al suo tempo crea *SVC* finche non finisce la procedura di parcheggio. Se il parcheggio non ha avuto il successo Central ECU riattiverà il *PA,* altrimenti termina il PA e torna allo stato iniziale.

Per migliorare la leggibilità di bytes loggati da *FFR*, *SVC*, *PA* loro vengono trasformati in tabella dove ogni i-esima colonna corrisponde al byte i-esimo scritto in forma esadecimale.

Implementazione di elementi facoltativi

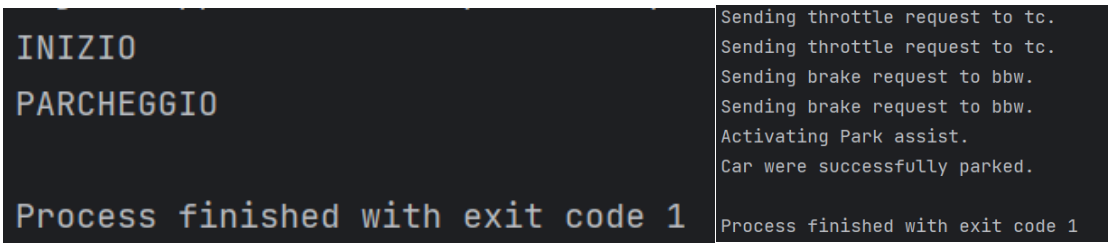
#	Elemento Facoltativo	Realizzato (SI/NO)	Descrizione dell'implementazione con indicazione del metodo principale
1	Ad ogni accelerazione, c'è una probabilità di 10-5 che l'acceleratore fallisca. In tal caso, il componente throttle control invia un segnale alla Central ECU per evidenziare tale evento, e la Central ECU avvia la procedura di ARRESTO	SI	Ogni richiesta di accelerazione da *ECU* a *TC* viene chiamato il metodo *isThrottleFailed*() che utilizza il random per generare il valore double in [0,1] e ritorna *true*, se il accelerazione e fallita. In caso di fallimento di accelerazione *TC* manda il segnale *SIG_THROTTLE_FAIL* che è definito in *Consts.h* come *SIGUSR2*. *Central ECU* gestisce tal segnale avviando l'arresto di auto e impostando l'auto nello stato iniziale.
2	Componente "forward facing radar"	SI	*FFR* viene creato all'inizio di guida dell'auto e manda se riesce leggere 8 bytes ogni secondo alla *Central ECU*. Alla richiesta di parcheggio il processo di *FFR* viene terminato.
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	Tutti i componenti (tranne *HMI*) vengono creati e eseguiti come i processi figli di *ECU* al bisogno di essi da *ECU*. Esecuzione di *SVC* e controllato da *PA*, cioè viene lanciato come il figlio di PA solo quando il *PA* è stato attivato dall'*ECU*. Quando il PA ha finito di trasmettere tutti i suoi dati all'ECU lui termina il SVC.
4	Il componente Park assist non è generato all'avvio del Sistema, ma creato dalla	SI	

	Central ECU al bisogno.		
5	Se il componente surround view cameras è implementato, park assist trasmette a Central ECU anche i byte ricevuti da surround view cameras.	SI	Per il ciclo di 30 secondi il *PA* cerca di leggere 8 bytes per mandare a ECU, contemporaneamente cercando di riprendere i dati ricevuti da SVC (usa il *socket* non bloccante) e trasmette anche questi 8 byte all'ECU.
6	Componente "surround view cameras"	SI	Viene creato da *PA*. Per il ciclo di 30 secondi trasmette i dati a *PA*.
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	Appena è arrivato il commando di parcheggio l'unico attuatore che non termina finche la velocita non diventa uguale a 0 è il componente *BBW*. Allo stesso momento viene lanciato il *PA* (ma non viene attivato). Puoi dopo di aver raggiunto tale velocita terminano tutti i attuatori e sensori tranne quelle necessarie per il parcheggio.
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 totale terminazione dell'esecuzione	SI	Funziona così, ma con l'unico cambiamento, che invece di terminare completamente la *CentralECU*, lei ritorna allo stato iniziale. Questo serve per non rieseguire la *CentralECU* da capo.

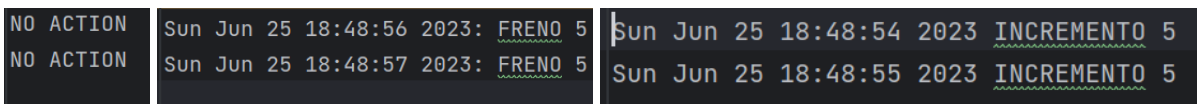
Esecuzione del programma

Parcheggio avviato da HMI

HMI reader e *HMI writer* console output in ordine corrispondente

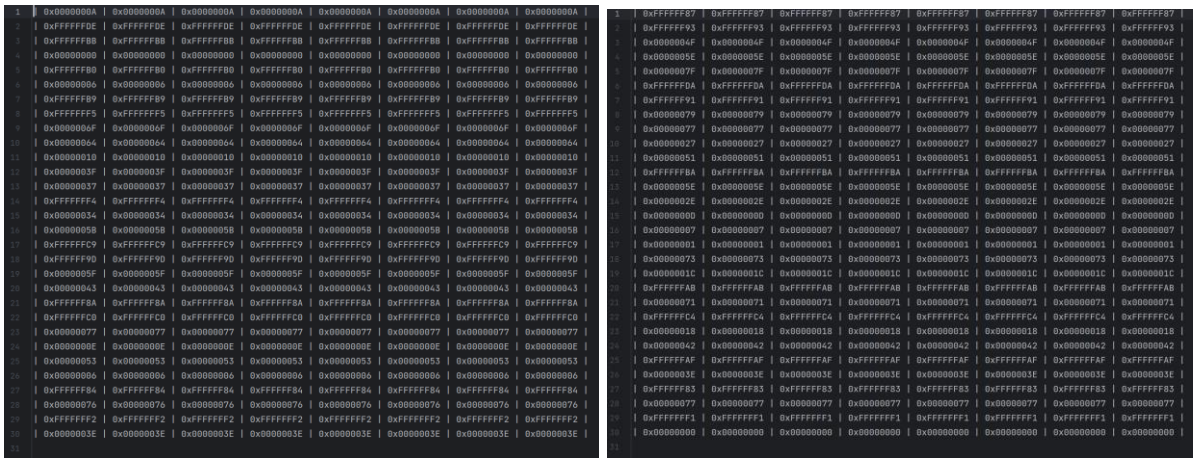


SBW, *TC* e *BBW* log file in ordine corrispondente



Come si vede da queste immagini SBW smette di funzionare appena riceve il segnale di parcheggio e la durata di freno e uguale alla durata di accelerazione, cioè parcheggio viene fatto dopo di raggiungere la velocita uguale a 0.

PA e *SVC* log file in ordine corrispondente



Qui pero si vede che PA e SVC trasmettano esattamente 30 volte 8 bytes per ogni uno, perché sempre riuscivano di leggerli.

Parcheggio avviato da *FWC* e il segnale di Pericolo

FWC log file

```
10
20
20
20
PERICOLO
10
10
PARCHEGGIO

```

SBW, TC log files in ordine corrispondente

```
ARRESTO AUTO
Sun Jun 25 19:15:30 2023: FRENO 5
Sun Jun 25 19:15:31 2023: FRENO 5

```

```
Sun Jun 25 19:15:23 2023 INCREMENTO 5
Sun Jun 25 19:15:24 2023 INCREMENTO 5
Sun Jun 25 19:15:25 2023 INCREMENTO 5
Sun Jun 25 19:15:26 2023 INCREMENTO 5
Sun Jun 25 19:15:28 2023 INCREMENTO 5
Sun Jun 25 19:15:29 2023 INCREMENTO 5

```

Come si vede qui il segnale di pericolo imposta la velocita a zero, perché dopo di pericolo la velocita desiderata è 10, cioè servono proprio 2 accelerazioni. Puoi avendo la richiesta di parcheggio dobbiamo frenare due volti.

Avvio della sterzata

SBW e *FWC* log files in ordine corrispondente

```
NO ACTION
NO ACTION
STO GIRANDO A SINISTRA
STO GIRANDO A SINISTRA
STO GIRANDO A SINISTRA
STO GIRANDO A SINISTRA
NO ACTION
STO GIRANDO A DESTRA
STO GIRANDO A DESTRA
STO GIRANDO A DESTRA
STO GIRANDO A DESTRA
NO ACTION
NO ACTION
NO ACTION
NO ACTION
NO ACTION
```

```
10
10
SINISTRA
20
20
20
20
DESTRA
20
20
20
20
20
20
20
PARCHEGGIO
```

TC avvia il segnale di fallimento

Prima dobbiamo impostare la probabilità di fallimento a 1.

HMI read e *HMI write* console outputs

```
INIZIO
```

```
Sending throttle request to tc.
Throttle failed. Terminating current execution.
```

Fallimento di parcheggio per esecuzione artificiale

Cameras log file

```
| 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 | 0x0000004E | 0xFFFFFFA3 | 0xFFFFFFCE | 0xFFFFFFE4 |
```

ECU log file

```
Sending brake request to bbw.
Activating Park assist.
Car failed to park.
Reactivating Park assist.
```

Si vede che primi 4 byte letti da SVC portano a l'errore del parcheggio e riavvio della procedura del parcheggio perché corrispondono a 0x0000.