Pietro Fanti, Gianmarco Ferruzzi,

Mohamed Salah Jebali

6297405, 6358071, 5968114
pietro.fanti1@stud.unifi.it, gianmarco.ferruzzi@stud.unifi.it,
mohamed.jebali@stud.unifi.it

Progetto di Sistemi Operativi A.A. 2018/2019 - ADAS made trivial

Istruzioni per compilazione ed esecuzione:

Il progetto include un makefile. Per cui è possibile compilarlo dal terminale lanciando:

```
$ make allMake
```

A questo punto è possibile iniziare l'esecuzione normale o artificiale rispettivamente con:

```
$ ./hmi NORMALE
```

o

```
$ ./hmi ARTIFICIALE
```

Sistema obiettivo

Il programma è stato programmato e testato su un sistema operativo **Ubuntu 18.04.2 LTS** tramite macchina virtuale.

Componenti e funzionalità implementate

Sono state implementate tutte le componenti obbligatorie e tutte quelle facoltative, insieme a tutte le loro funzionalità base:

- **Human-Machine Interface**
- **Attuatori:**
  - steer-by-wire
  - throttle control
  - brake-by-wire
- **Sensori:**
  - front windshield camera
  - forward facing radar
  - park assist
  - surround view cameras
  - blind spot
- **Central ECU**

Inoltre sono state implementate anche le seguenti funzionalità facoltative

- Terminazione di tutti i sensori e di tutti gli attuatori (ad eccezione di *park assist* e di *surround view cameras*) durante l'operazione di parcheggio.
- *Blind spot* è costantemente in attesa di un segnale direttamente da *steer-by-wire*, piuttosto che di una richiesta dalla *Central ECU*.

Progettazione ed Implementazione

Schema Architetturale

Lo schema architetturale adottato nel progetto è il seguente: l'***HMI*** è il programma principale che genera la **ECU** al suo avvio. La ECU appena generata rimane in attesa di una *signal* di avvio da parte dell'*HMI*, che viene mandata quando si digita "INIZIO". Quando la ECU si avvia, vengono generati due processi: uno funge da ***server*** (ECU *server*) e l'altro da ***client*** *(*ECU *client*). I **sensori** sono *client* della ECU *server,* e inviano ad essa dati letti dalle varie fonti tramite ***socket***. Gli **attuatori** sono *server* della ECU *client*, e ricevono i vari input tramite *socket*. Abbiamo utilizzato le *socket* perché abbiamo ritenuto che fossero lo strumento più consono per attuare lo scambio di messaggi bidirenzionale tra i vari programmi *client-server*. Lasciamo quindi la responsabilità di gestire il contenuto del messaggio, e di attuare eventuali procedure, al programma destinatario. Per far sì che alcuni programmi attivassero e/o disattivassero le specifiche di un altro programma, indipendente da eventuali messaggi scambiati, abbiamo utilizzato le ***signal***. Ad esempio, *Steer By Wire* attiva e disattiva l'esecuzione del processo figlio, *Blind Spot*, tramite *signal*. Infine, poiché molti programmi generano processi interni che hanno bisogno di conoscere informazioni in comune, utilizziamo le ***pipe*** come strumento di comunicazione unidirezionale tra processo padre e processo figlio.

Sensori

I sensori sono programmi gestiti come ***client*** della ECU *server* e scambiano messaggi con quest'ultima tramite ***socket*.**

Tutti i sensori vengono creati dalla ECU non appena questa riceve il comando di "INIZIO", ad eccezione dei sensori facoltativi *Surround View Cameras* e *Blind Spot*, che vengono creati dal sensore *Park Assist* e dall'attuatore *Steer By Wire,* rispettivamente.

Quando ***Park Assist*** viene creato, questo genera il sensore *Surround View Cameras*, si connette alla *socket* per scambiare messaggi con ECU *server*, e si mette in ascolto della *signal* di parcheggio. Il sensore ***Surround View Cameras****,* quando viene creato, si connette alla *socket* per

scambiare messaggi con ECU *server* e attende la *signal* di attivazione da parte di *Park Assist*. Dunque, quando *Park Assist* riceve la *signal* di parcheggio da parte della ECU, inizia ad inviare, una volta al secondo, per 30 secondi, alla ECU ciò che legge da */dev/urandom* o */urandomARTIFICIALE.binary* (a seconda della modalità scelta all'inizio dell'esecuzione) e invia una *signal* a *Surround View Cameras*, per metterlo in esecuzione. Quando termina con successo l'esecuzione del parcheggio, disattiva *Surround View Cameras*, sempre tramite *signal,* e termina se stesso e il processo figlio.

Il sensore ***Blind Spot***, non appena creato si connette alla *socket* per scambiare messaggi con la ECU *server*. Inoltre, come da richiesta facoltativa, *Blind Spot* è in costante attesa di una *signal* di attivazione da parte di *Steer By Wire*. Infatti, quando *Steer By Wire* inizia la procedura di sterzata, invia una *signal* di attivazione a *Blind Spot* e appena conclude la procedura, invia una *signal* di disattivazione.

Il sensore ***Front Windshield Camera*** quando viene creato dalla ECU si connette alla *socket* per scambiare i messaggi con ECU *server* e iterativamente, ogni 10 secondi, legge dati da una sorgente e li invia alla ECU.

Il sensore ***Forward Racing Radar*** quando viene creato dalla ECU si connette alla *socket*, e ogni 2 secondi prova a leggere 24 byte da */dev/random*. Se riesce a leggere correttamente 24 byte, li trasmette alla ECU e li scrive nel file di log, altrimenti non invia niente.

Attuatori

I tre attuatori sono programmi che si comportano come ***server*** per la ECU *client,* e utilizzano le ***socket*** per scambiare messaggi con la ECU, mentre utilizzano le ***pipe*** per scambiarsi le informazioni ricevute dalla ECU tra i processi interni ai programmi.

Tutti e tre gli attuatori vengono generati dalla ECU centrale quando questa riceve il comando "INIZIO".

Quando l'attuatore ***Throttle Control*** viene creato, genera due processi interni: Il processo padre, si occupa della generazione della *socket* e di rimanervi in ascolto; il processo figlio, invece, si occupa di aprire e scrivere sul *file di log* "NO ACTION" se non riceve comandi di accelerazione,

oppure “AUMENTO 5” se riceve il comando di accelerazione. Il processo padre riceve continuamente messaggi dalla ECU client e ne scrive il contenuto nella *pipe*. Il processo figlio legge dalla *pipe* il valore della accelerazione da apportare all’auto: se è maggiore di zero la esegue, altrimenti stampa “NO ACTION”.

Anche l’attuatore ***Brake By Wire***, non appena creato, genera due processi interni: il processo padre ha il compito di generare la *socket*, vi rimane in ascolto e scrive la differenza di velocità che riceve dalla ECU *client* nella *pipe*; il processo figlio, invece, si occupa di aprire e scrivere sul file di log “NO ACTION” se non riceve messaggi di frenata, altrimenti scrive “DECREMENTO 5”. Il processo figlio legge dalla *pipe* il valore della decelerazione da apportare all’auto: se è maggiore di zero, la esegue, altrimenti stampa “NO ACTION”. Il processo padre, inoltre, si occupa di gestire l’operazione di **parcheggio.** Infatti, quando la ECU riceve il comando “PARCHEGGIO”, invia, tramite *socket,* a *Brake By Wire,* il messaggio “PARCHEGGIO X” (X è la velocità corrente). La stringa di messaggio viene *splittata* in “PARCHEGGIO” e “X”. Il processo padre, quando legge “PARCHEGGIO” attua l’apposita procedura di arresto normale dell’auto. A questo punto, il processo figlio decrementa la velocità normalmente fino a zero, e quando l’auto è ferma, il processo padre comunica alla ECU, tramite *signal*, che l’auto si è fermata e avvia la procedura di terminazione del processo figlio e di se stesso. Infine, il processo padre gestisce l’operazione di **pericolo**. Quando la ECU rileva un pericolo invia una *signal Brake By Wire*. Il processo padre cattura la *signal* e scrive “ARRESTO AUTO” sul file di log. Dopodiché comunica, tramite *signal*, alla ECU di aver completato l’esecuzione, e questa inizierà la procedura di terminazione dei sensori e attuatori.

Infine, l’attuatore ***Steer By Wire,*** alla creazione, genera tre processi interni, uno dei quali è ***Blind Spot***. In questo caso, il processo padre si occupa dell’apertura e della scrittura del file di log e di svegliare, tramite *signal,* il processo figlio *Blind Spot* quando esegue la curva, e di disattivarlo quando smette di curvare (come da richiesta facoltativa). Invece, il processo figlio si occupa di generare la *socket* e di rimanervi in ascolto. I messaggi che riceve, tramite socket, da ECU *client* vengono scritti nella *pipe*. Il processo padre, legge dalla *pipe* l’eventuale direzione ricevuta, e se legge “DESTRA” o “SINISTRA”, avvia la procedura di sterzata che consiste nell’attivazione del

processo figlio *Blind Spot* e nella scrittura nel file di log di “sto girando a [DIREZIONE]”, per 4 secondi. Quando la procedura è terminata, *Blind Spot* viene disattivato.

ECU

L’unità di controllo gestisce la creazione dei processi legati ai sensori, attuatori e l’elaborazione dei dati ed è costituita da: una interfaccia per i sensori (ECU server), una per gli attuatori (ECU client) e una unità di elaborazione.
Dopo che questa viene creata dalla HMI e avviata da un segnale ricevuto dalla HMI stessa, genera un processo per ogni sensore e per ogni attuatore. Per i primi vengono anche predisposti dei processi addetti alla elaborazione dei dati generati da essi e all’invio di eventuali comandi indirizzati agli attuatori (unità di elaborazione).

La comunicazione tra ECU e sensori è gestita da processi che svolgono la funzione di server, i quali ricevono tramite socket i dati letti dai sensori. I processi server dialogano con gli altri processi interni tramite pipe anonime. Infatti essi inviano i dati ricevuti ai processi che si occupano di elaborare i dati, i quali, tramite socket o tramite segnali, inviano la risposta della elaborazione agli attuatori interessati.
Tutti i comandi che vengono generati dalla unità di elaborazione, vengono inviati ad un processo, il padre di tutti gli altri processi sopra descritti, il quale ha il compito di scrivere tali comandi su un file di log.

Alla ricezione del segnale relativo al parcheggio, la ECU si occupa di inviare il relativo comando alla componente Brake By Wire in modo da fermare il veicolo e, dopo aver terminato sensori e attuatori non necessari tramite segnali di terminazione, iniziare la procedura di parcheggio inviando un segnale alla componente Park Assist. Quando poi quest’ultimo segnala che la procedura è andata a buon fine, la ECU invia un segnale a HMI per indicare che il parcheggio è avvenuto.

Nel caso in cui l’unità di elaborazione riceva dai sensori valori indicanti un pericolo, viene inviato un segnale a Break By Wire per arrestare il veicolo e, dopo l’arresto, invia un segnale alla HMI relativo l’avvenimento di una situazione di pericolo.

HMI

L'HMI è divisa in tre processi: un processo padre che costituisce **l'interfaccia di input** con l'utente , e due processi figli: uno esegue semplicemente una `execv()` e avvia la ECU, l'altro, **l'interfaccia di output**, stampa su un secondo terminale il contenuto di *ECU.log* tramite una `tail`.

Il processo padre, che comunica gli input alla ECU tramite socket, si occupa inoltre, quando riceve gli appositi segnali, della terminazione dell'intero albero dei processi.

Esecuzione

I seguenti test sono stati eseguiti dopo aver lanciato la modalità "NORMALE" e "ARTIFICIALE". Verranno descritti gli INPUT inseriti in *frontcamera.data* e gli OUTPUT scritti sul terminale.

CASO ZERO

INPUT: 15 50 10 DESTRA SINISTRA PERICOLO

OUTPUT: INCREMENTO 15 | INCREMENTO 35 | FRENA 40 | DESTRA | SINISTRA

Dopo l'input PERICOLO il veicolo si arresta correttamente e non viene più stampato niente sul terminale. Occorre digitare nuovamente "INIZIO" per far ripartire l'esecuzione del programma. La lettura da *frontcamera.data* riparte dalla riga successiva al segnale di pericolo.

CASO UNO

INPUT: 15 15 65 15 DESTRA DESTRA SINISTRA PERICOLO SINISTRA

OUTPUT: INCREMENTO 15 | INCREMENTO 50 | FRENA 50 | DESTRA | DESTRA | SINISTRA

Dopo l'input PERICOLO il veicolo si arresta correttamente e non viene più stampato niente sul terminale. Occorre digitare nuovamente "INIZIO" per far ripartire l'esecuzione del programma. La lettura da *frontcamera.data* riparte dalla riga successiva al segnale di pericolo.

CASO DUE

INPUT: 15 15 75 90 15 DESTRA DESTRA SINISTRA PERICOLO

OUTPUT: INCREMENTO 15 | INCREMENTO 60 | DESTRA | FRENA 60 | SINISTRA

Lo scopo di questo test è di verificare che con una variazione di velocità maggiore di 50 e un altro incremento di velocità consecutivo, il file di log *throttle.log* viene compromesso, poiché, come da specifiche, i dati da *frontcamera.data* vengono letti ogni 10 secondi, e il comando "AUMENTA 5" sul log di accelerazione viene scritto ogni secondo per 12 secondi (60/5), e successivamente per 3 secondi (15/5), aspettandosi un totale di 15 "AUMENTO 5" nel file di log per questa serie di rilevamenti (15 75 90). Invece, il numero di "AUMENTO 5" ottenuto in tale serie è di 13, e poiché dopo la durata massima di 10 secondi, passa a scrivere gli incrementi di velocità del rilevamento successivo.

Dopo l'input PERICOLO il veicolo si arresta correttamente e non viene più stampato niente sul terminale. Occorre digitare nuovamente "INIZIO" per far ripartire l'esecuzione del programma. La lettura da *frontcamera.data* riparte dalla riga successiva al segnale di pericolo.