**Consegna #3 - *Smart Car***

***Componenti gruppo: Chiana Lorenzo e Minardi Matteo***

Descrizione del funzionamento e breve introduzione alla logica.

Il sistema si suddivide essenzialmente in due parti:

1. Lo Sketch Arduino (*smart\_car.ino)*
2. E il progetto Android (*SmartCar*)

Il programma della parte *Embedded* è diviso in 4 Task e dispone di un oggetto globale (*env*) della classe che rappresenta il contenitore delle variabili che gestiscono lo stato e la situazione attuale.

I quattro task sono:

* **ControllerTask**

Questo task utilizza la variabile *env* soltanto per ottenere il canale di comunicazione e impostare il buffer di scrittura con relativa variabile booleana che indica se è disponibile. Quando ha ottenuto il canale, controlla se è disponibile un messaggio e nel caso controlla se è un cambiamento di stato. Nel caso sia qualcos’altro (un numero che serve per regolare un motorino servo) viene salvato nel buffer in modo da poter essere prelevato in base alla necessità dell’altro task (*pressionTask*)

* **DistanceTask**

Il task della distanza è semplice e conciso però incapsula tutto ciò che riguarda il sensore ultrasuoni e le sue rilevazioni durante il ciclo di produzione. Ogni tick prende la distanza, la aggiorna nell’oggetto globale e nel caso ci sia necessità prende possesso del canale in invio e manda l’allarme richiesto in consegna.

* **PressionTask**

Il task che incapsula ciò che riguarda la pressione del pulsante tattile è quello di più difficile comprensione e funzionamento più articolato.  
Innanzitutto rileva solo se è necessario ed in ogni caso se avviene un contatto viene segnalato tramite il canale in scrittura. Il messaggio viene inviato soltanto una volta per pressione e non di continuo ad ogni ciclo di clock, perché potrebbe inutilmente saturare il canale con un’informazione ridondante.  
Il caso interessante è quello in cui si trovi nello stato di movimento. In questo caso arduino si mette in ascolto e aspetta un messaggio numerico (*waitMsg*) che indichi l’angolo del servo motore. Una volta segnalata l’informazione e confermata con un messaggio smette di aspettare un messaggio informativo che concerne l’implementazione fisica della difesa.

* **LedTask**

Questo stato controlla il funzionamento dei dispositivi di output che vengono incapsulati in questo task che in base alle variabili di stato e le variabili che indicano la situazione generale ne modificano i valori.

Il comportamento è descritto da un insieme di macchine a stati finiti che cooperano in modo sincrono all’interno di uno scheduler con un certo clock arbitrario.  
Riassumendo la parte che concerne arduino si occupa dello stato del sistema mentre l’applicazione android pilota con degli input concordati in un protocollo embrionale di comunicazione.

I principali problemi riscontrati riguardano la parte hardware. Si sono presentati scenari di malfunzionamenti dovuti alla gestione del timer e ai segnali bluetooth. A causa dell’eccessivo utilizzo di timer è stato necessario utilizzare una libreria esterna per il servo motore (*ServoTimer2.h*).

Per quanto riguarda la parte *android* è stata implementata soltanto un’activity che si divide essenzialmente in quattro punti logici:

* La presa e il rilascio delle risorse in modo da utilizzare correttamente i sensori hardware quali: bluetooth e gps.
* La gestione del protocollo di comunicazione concordato tra noi componenti del gruppo.
* L’implementazione personalizzata dei listener collegati ai componenti.
* Gestione dell’invio delle email.

Per la gestione dei sensori e delle email sono state utilizzate le classi ed in metodi datoci a lezione con alcune modifiche per adattarle meglio al funionamento del sistema.

La comunicazione tra i due sottosistemi viene fatta mediante scambio di sette messaggi univoci e concordati inizialmente tra i componenti del gruppo e sono:

* “spenta non in parcheggio”, inviato da C a T per segnalare il cambio di stato;
* “spenta in parcheggio”, inviato da C a T per segnalare il cambio di stato;
* “accesa in movimento”, inviato da C a T per segnalare il cambio di stato;
* “contatto”, inviato da T a C per segnalare l’avvenuto contatto;
* “Presenza veicolo – distanza: \*distanza in metri\*”, inviato da T a C per segnalare la presenza troppo vicina di un altro veicolo;
* invio da C a T dei valori selezionati dall’utente per impostare il servo;
* “fine”, inviato da C a T per segnalare la fine dell’invio dei valori al servo.

I messaggi provenienti dal sistema embedded sono visualizzati in una opportuna TextView.  
Vi sono poi quattro listener legati a quattro componenti per gestire l’invio dei vari messaggi a T:

* **MySpinnerOnItemSelectedListener**

Attraverso un componente di tipo Spinner l’utente sceglie lo stato del sistema e l’applicazione invia lo stato scelto al sottosistema T.

* **MySeekBarListener**

Una volta avvenuto il contatto nello stato “accesa in movimento” l’utente potrà modificare la posizione di M1 mediante una SeekBar. Inizialmente si era optato per un approccio “real time” dove l’applicazione inviava ad Arduino i messaggi con la posizione alla modifica della progressione. Questo approccio è risultato inefficace dato che andava a saturare la comunicazione con l’invio di troppi messaggi in un lasso di tempo breve. Si è risolto mandando l’invio del messaggio solo quando l’utente toglie il dito dal componente evitando così la saturazione del canale di comunicazione.

* **MyButtonEndSeekListener**

La fine dell’invio dei messaggi della posizione di M1 è determinato dal messaggio “fine” inviato alla pressione del bottone a cui è legato questo listener.

* **MySwitchOnClickListener**

Listener che attiva o disattiva la modalità notifica.

Infine le coordinate della posizione corrente vengono fatte visualizzare tramite TextView alla ricezione del messaggio “contatto” e quando il sistema è in “spenta in parcheggio”.