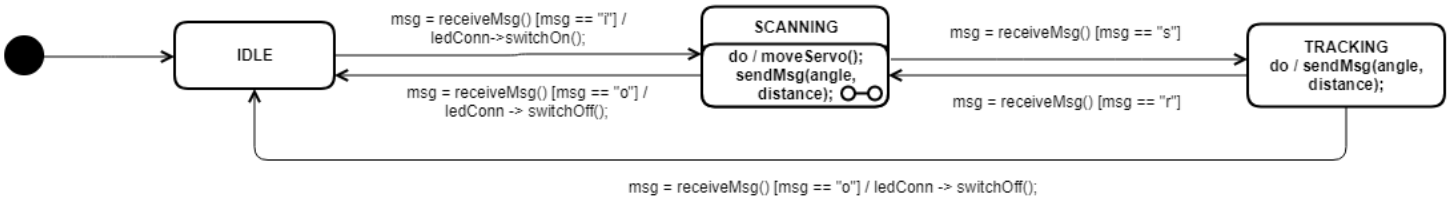
CONSEGNA 3 – “SMART RADAR”

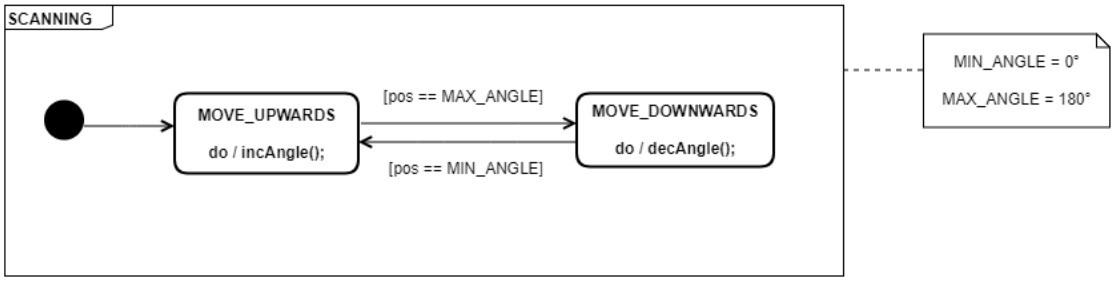
RELAZIONE

di Lorenzo Campanelli, Luca Ragazzi, Mattia Leonessi

E’ richiesto di realizzare un radar intelligente mediante l’interazione tra un Raspberry Pi e un Arduino che in questo contesto fungono da Radar (sensore) il primo e da Centralina il secondo. La logica dell’applicazione e il controllo di essa, vengono gestiti interamente tramite il RasPi mentre il microcontrollore si occupa solamente di analizzare l’ambiente circostante (attraverso la rotazione del radar) e di inviare i dati rilevati al sistema di controllo. La comunicazione avviene mediante seriale.

FSM – Radar (Arduino)





Come già detto sopra il microcontrollore in questo caso rappresenta un radar “non intelligente” che si limita a rilevare la situazione dell’ambiente attorno a sé e ad inviare tali informazioni alla centralina.

L’approccio inizialmente adottato non era basato sull’utilizzo di uno scheduler, in quanto non lo ritenevamo necessario, data la presenza di un solo task. Il tutto veniva governato dal ciclo di clock del micro e dalla presenza di un delay dopo la write sul servo per attendere che questo giungesse in posizione.

Ci siamo però accorti che il sensore di prossimità (prox) non misurava oltre i 70 cm nel vuoto ed abbiamo compreso che questa anomalia di misurazione deriva da un’interferenza hardware tra servo e prox. Abbiamo quindi deciso (dopo esserci consultati con altri compagni) di passare ad uno scheduler e ad un sistema che consta di 2 task: uno che implementa la FSM del Radar (riportato in figura) e l’altro che si occupa di gestire la seriale (leggere i messaggi in arrivo e tradurli in opportune operazioni compiute su un contesto condiviso col task del Radar).

L’adozione di uno scheduler (o alternativamente di un delay maggiore dopo la write del servo) ha permesso di raggiungere misurazioni oltre i 70 cm, fermo restando il fatto che risultano comunque essere misurazioni alquanto imprecise sopra al metro ed estremamente fluttuanti. A questo problema si è cercato di porre rimedio settando valori opportuni per la distanza di detection e di tracking su RasPi.

L’adozione dello scheduler ci ha “costretti” ad utilizzare la libreria ServoTimer2 per la gestione del servo in quanto la libreria Servo si basa sul Timer1 utilizzato anche dalla classe Timer necessaria allo scheduler per funzionare.

In fase di setup del sistema Arduino legge il valore di un potenziometro che, opportunamente mappato, funge da moltiplicatore per il calcolo della velocità angolare del servo (espressa come periodo di scheduling del task del Radar).

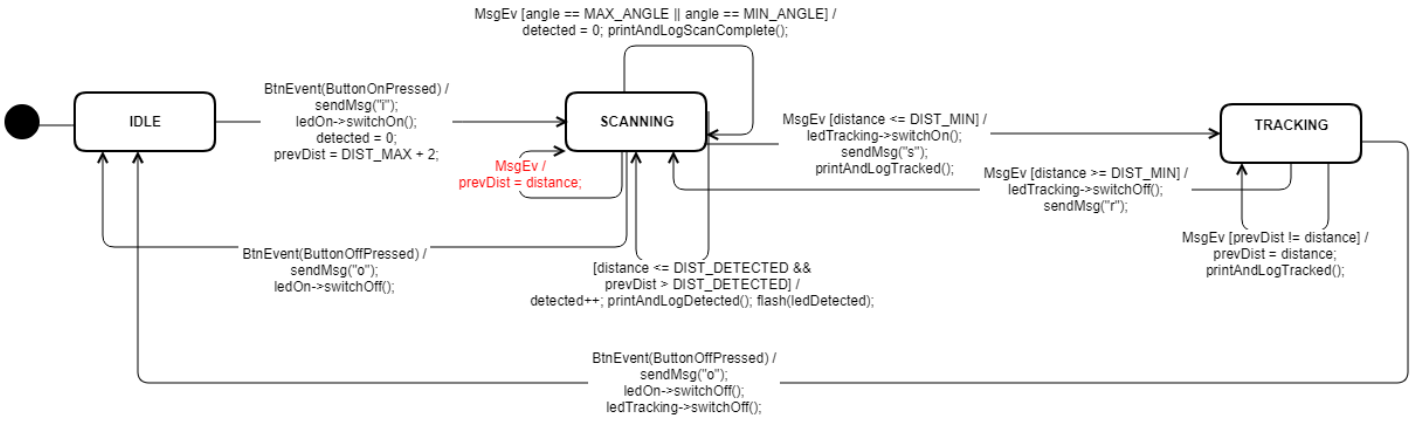
Una volta ricevuto il messaggio di start dalla centralina il radar passa allo stato SCANNING in cui compie uno scanning di 180°. Ad ogni step di un grado viene inviato un messaggio alla centralina (sotto forma di stringa) contenente l’angolo attuale del servo e la distanza rilevata.

Se in stato SCANNING giunge il comando di tracking, il micro transita in stato TRACKING nel quale continua a rilevare la distanza e l’angolo ma il servo non viene più mosso.

Qualora in stato TRACKING giunga il comando di reset poiché l’oggetto tracciato si è allontanato ad una distanza maggiore di quella di tracking, il sistema torna in stato di SCANNING.

In qualunque stato si trovi il micro, se giunge un comando di stop il sistema si ferma e si riporta allo stato iniziale (IDLE) in attesa di un nuovo comando di scanning.

FSM – Centralina (Raspberry Pi)



MsgEv := MsgEvent(PositionInfoMsg(angle,distance))

The action in red in SCANNING state is executed whenever a MsgEv occurs

MIN_ANGLE = 0°
MAX_ANGLE = 180°
DIST_MIN = 0.5m
DIST_MAX = 2m

L’applicazione Java che implementa la centralina è basata su una FSM event-based. In particolare vengono presi in considerazione 2 tipi di eventi: bottone premuto (ButtonPressed) generato dai pulsanti ON/OFF e evento di messaggio in arrivo dal radar (MsgEvent) su seriale.

La struttura generale sulla quale si basa il Task che implementa la centralina (RadarController) è un event-loop. Per ogni messaggio in arrivo (evento) sulla seriale viene scatenato un interrupt e nella funzione serialEvent() che funge da handler per gli eventi da seriale viene creato un nuovo PositionInfoMsg che contiene l’angolo del radar e la distanza rilevata.

La seriale (SerialCommChannel) e i pulsanti sono Observable cosicché il RadarController (che implementa Observer) possa registrarsi come loro listener ed essere notificato ogniqualvolta c’è un evento generato da queste. La notifica comporta l’aggiunta dell’evento nella BlockingQueue di eventi mantenuta dal RadarController.

L’implementazione dell’event-loop mediante una BlockingQueue fa si che il thread che esegue il RadarController possa sospendersi in attesa dell’arrivo di un evento da gestire, evitando quindi inutili attese attive.

All’arrivo dell’evento generato dal pulsante ON la centralina provoca il risveglio del Radar, accende il led ON e transita nello stato SCANNING dove rimane in attesa di messaggi dal radar contenenti le informazioni su distanza e angolo.

A questo punto se l’oggetto rilevato è ad una distanza minore o uguale a DIST_DETECTED viene aumentato il conteggio degli oggetti individuati nella scansione e viene fatto flashare il led Detected.

Per determinare se l’oggetto rilevato è uno solo o sono più oggetti diversi si guarda se prima che l’oggetto venisse rilevato c’è stato almeno un angolo in cui non è stato rilevato nessun oggetto (distanza misurata > DIST_DETECTED).

Se viene rilevato un oggetto a distanza minore o uguale a MIN_DIST (distanza di tracking) la centralina comanda al radar di passare in stato TRACKING, accende il led Tracking e resta in attesa dei messaggi contenenti le informazioni di posizione per decidere quale azione intraprendere.

Nello specifico se la distanza non varia non accade nulla, se invece varia rimanendo dentro al range di tracking vengono soltanto stampati a video e loggati l’angolo di tracking e la distanza mentre se l’oggetto esce dal range di tracking viene spento il led Tracking e lo scanning riprende normalmente (ritorno allo stato SCANNING).

In qualunque stato diverso da IDLE si trovi la centralina, la pressione del pulsante OFF provoca lo spegnimento del radar, lo spegnimento del led ON e il ritorno allo stato IDLE.

NOTE E PROBLEMI RISCONTRATI

1) Accade spesso (in momenti casuali) che nel programma Java (la centralina) venga generata l'eccezione di tipo *IOException: underlying input stream returned zero bytes*.
   Questo, in base a varie prove effettuate, parrebbe provocare poi l'eccezione di tipo *NumberFormatException* a causa del tentativo di parsare una stringa vuota al momento della creazione del PositionInfoMsg.
   Purtroppo i vari tentativi di risoluzione del problema si sono conclusi senza successo e quando questa eccezione avviene il sistema non si "riprende" più costringendo a killare l'applicazione o in certi casi a riavviare il sistema (su RasPi), in quanto a volte anche ad un riavvio dell'applicazione il problema continua a mostrarsi (come se il buffer rimanesse pieno di stringhe vuote).
   Abbiamo ipotizzato che il problema possa dipendere dalla libreria RXTX (poiché l'eccezione di tipo IOException avviene spesso, ed accade anche in altri programmi che fanno uso della seriale gestita mediante RXTX, mentre al contrario non avviene nell'interazione con Arduino attraverso il Serial Monitor di Arduino IDE o mediante un altro terminale). Sarebbe interessante (ma per questioni di tempo non siamo riusciti a farlo) provare a sostituire RXTX con un'altra libreria per la gestione della seriale come jSSC.

2) Abbiamo notato che i momenti di maggiore sballamento della distanza misurata dal prox coincidevano coi momenti di maggior sforzo del servo, il che ci ha portato a pensare che potesse essere una mancanza di alimentazione sufficiente al servo.
   Abbiamo quindi cercato di alimentare separatamente il servo dal resto mantenendo il segnale PWM sull'Arduino ma così facendo il servo non si muoveva più.