9/12/2017 Consegna n.3 Simone Magnani – 754082

Smart Radar

Premessa

Per la terza consegna, ho deciso di strutturare il progetto nel seguente modo:

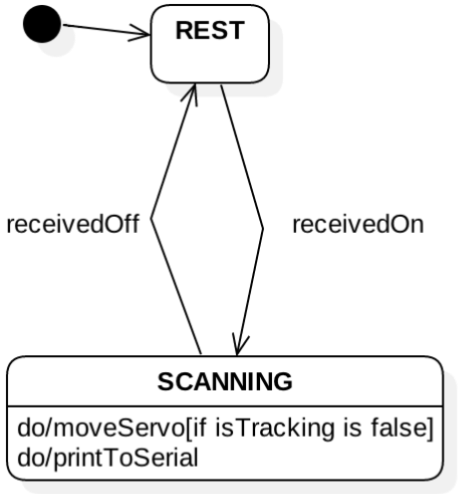
- *Arduino* → n.2 Task
- *Raspberry* → n.2 Agenti (Thread nel mio caso)

Riflettendo sulla consegna, ho pensato che questa suddivisione potesse essere una buona scelta. Analizziamo ora parte per parte le varie componenti.

Arduino

Radar

Task per il rilevamento. È composto da due stati: *REST* e *SCANNING*. La scelta di utilizzare semplicemente due stati e non tre (qualcuno avrà utilizzato anche lo stato *TRACKING*), è motivata dal fatto che analizzando il funzionamento del Radar, l'unica differenza tra lo stato di *SCANNING* e lo stato di *TRACKING* risiede nel motore servo, il quale non deve muoversi durante quest'ultimo stato. A questo punto ho pensato di aggiungere una variabile booleana privata all'interno del task la quale verrà settata a *true* solamente quando viene ricevuto il messaggio "*Tracking*" da *Raspberry*. Nello stato di *SCANNING*, viene continuamente aggiornato l'angolo del motorino a seconda del valore letto dal potenziometro (il valore è mappato in un range che va da 1 a 10 onde evitare bruschi scatti) e viene inviata una stringa al *Raspberry* la quale contiene l'angolo attuale e la distanza rilevata.



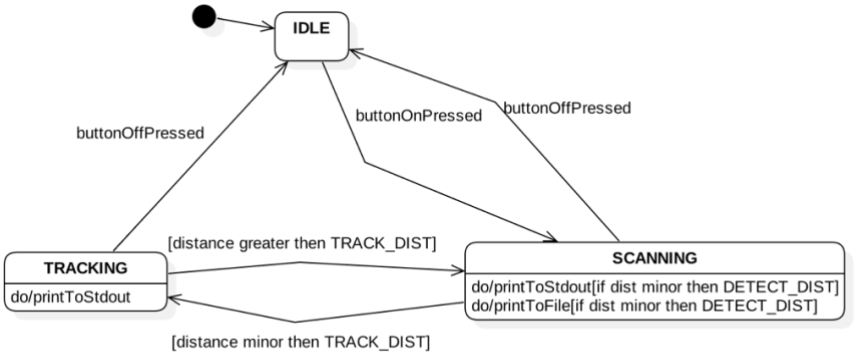
RadarComm

Task per la comunicazione via seriale. È composto da un unico stato che, tramite il sevizio *MsgService*, controlla se vi sono messaggi in arrivo nella seriale e, nel caso vi siano, legge e copia il messaggio nella stringa globale (usata per comunicare con l'altro task). Ho scelto di non aggiungere altri stati/funzionalità, in quanto è presente un'altra classe *Logger* la quale può essere invocata ovunque per inviare messaggi. In questo modo *Arduino* è sempre aggiornato all'ultimo messaggio ricevuto e, ogniqualvolta vi sia la necessità di inviare messaggi verso *Raspberry*, non basterà altro che richiamare staticamente la classe *Logger*.

Raspberry

CentralUnit

È l'agente responsabile della logica di tutto il programma. Al suo interno vi sono tutte le componenti richieste per la centralina, un'istanza di calendario (per stampare il tempo), variabili di controllo e la *Seriale* attraverso la quale vengono inviati i messaggi ad *Arduino*. Si poteva benissimo creare un agente apposito per l'invio di messaggi (o utilizzarne uno solo per invio + ricezione), ma per semplificare il tutto ho deciso di implementarli in questo modo. Passando al costruttore di entrambi gli agenti l'oggetto seriale istanziato nel *Main* (Pattern *Observer*) verrà utilizzata la medesima istanza per l'invio e ricezione (in maniera coordinata). La *CentralUnit* estende un BasicEventLoopController, ovvero un loop di controllo di eventi continuo: quando si verifica un nuovo evento, esso viene aggiunto alla coda di eventi la quale viene di volta in volta svuotata tramite il metodo *processEvent*, il quale processa l'evento più "vecchio" nella coda. In questo modo la *CentralUnit* entra in funzione quando vi sono eventi da processare, evitando di fare altro quando non è necessario.
All'interno della *processEvent* ho creato 3 stati: *IDLE*, *SCANNING* e *TRACKING*. A differenza del task su *Arduino*, il *Raspberry* per quanto riguarda la mia personale interpretazione ha bisogno di esplicitare la differenza tra i due stati *SCANNING* e *TRACKING*, in quanto vengono compiute operazioni differenti. Innanzitutto ad ogni evento viene controllato se è stato premuto il bottone *OFF* e, in caso positivo, viene comunicato ad *Arduino* lo spegnimento; nello stato di *IDLE* il programma attende l'evento *"Bottone ON premuto"* per poi accendere il radar e iniziare le scansioni nello stato *SCANNING*. L'agente riceve continuamente messaggi dal radar i quali contengono angolo e distanza dell'oggetto. Ogni volta che la distanza è al di sotto di *DETECT_DIST*, viene stampata a video e salvata su file la stringa di informazione. Se invece la distanza è inferiore a quella di *TRACKING*, il sistema entra nello stato di *TRACKING* e viene comunicato al microcontrollore di fermarsi per tracciare l'oggetto. Ogni 180 gradi viene stampata a video il risultato della scansione (n° oggetti rilevati).
Ad ogni stato sono associati solamente gli eventi interessati, per esempio se ci troviamo nello stato *IDLE* è inutile che vi sia il controllo dell'evento" Bottone *OFF premuto"*, in quanto il sistema è già a riposo. Mentre l'evento *"Bottone ON premuto"* sarà tenuto in considerazione solamente nello stato di *IDLE*, in quanto negli altri stati il sistema è già acceso.



InputMsgReceiver

È l'agente responsabile della ricezione dei messaggi attraverso il canale *Seriale*. Sostanzialmente sfrutta le tecnologie presentate a lezione (es. la *BlockinQueue, serialEvent*, ecc..): tramite il metodo *waitForMsg* viene letto (non appena disponibile) il messaggio in arrivo, per poi notificare la *CentralUnit* (tramite il pattern Observer) dell'arrivo di un nuovo messaggio grazie all'evento appositamente creato *ScanInfoReceived*.