Assignment #2: Relazione

CORSO DI SISTEMI EMBEDDED E IOT

LOMBARDINI ALESSANDRO
GIACHIN MARCO
BAIARDI MARTINA

Sommario

1. Introduzione: scopo del progetto

Il sistema da realizzare consiste in uno smart radar, composto dai seguenti componenti embedded:

- N. 1 Sensore di movimento (PIR)
- N. 1 Sensore di distanza ad ultrasuoni
- N. 2 Led
- N. 3 Bottoni tattili
- N. 1 Potenziometro
- N. 1 Servo motore

Per la realizzazione del sistema è stato utilizzato un approccio di programmazione a Task mediante linguaggio C++. A questo è stato affiancato un programma, realizzato in Java mediante JavaFX come libreria grafica, in grado di controllare il sistema e visualizzarne l'output durante il funzionamento.

2. Progettazione

La prima parte del processo di progettazione è stata rivolta alla suddivisione del comportamento complessivo del sistema in singoli Task, intesi come unità di lavoro ben definito. Si sono per prima cosa individuati alcuni macro-compiti, che sono emersi essere:

1. Rilevamento del movimento
2. Scrittura e lettura dei messaggi tramite applicativo Java
3. Campionamento della distanza
4. Rotazione del radar
5. Cambio della velocità

Il comportamento complessivo del sistema è stato poi decomposto in più Task:

- Gestione del led di stato di allarme
- Gestione del led di stato di tracking
- Gestione dei pulsanti per la scelta della modalità
- Gestione del servo motore
- Gestione delle letture del potenziometro
- Gestione del risparmio energetico e del sensore di movimento
- Gestione del campionamento ad ultrasuoni
- Gestione delle letture da seriale
- Gestione dei messaggi ricevuti

Per realizzare questa suddivisione sono state prese delle specifiche decisioni progettuali, poiché il sistema doveva essere realizzato in maniera da lavorare in tre condizioni di funzionamento differenti, ognuna delle quali caratterizzata da specifiche funzionalità.
Una prima possibilità di implementazione di questo meccanismo prevederebbe per ciascuna funzionalità, e per ogni singola modalità, la realizzazione di uno specifico Task; in questo modo avremmo, a partire da una implementazione comune (per esempio quello della gestione del motore), tre diverse specializzazioni, una per ogni modalità (in questo caso, per esempio, avremmo avuto motore in auto, motore in single e motore in Manual). Questa scelta ci è sembrata inadeguata in quanto avrebbe appesantito enormemente il codice, pertanto si è scelto un approccio nel quale ogni funzionalità è contenuta in un singolo Task in grado di espletare correttamente i compiti di tutte le modalità.

È previsto dunque un oggetto chiamato State, comune a tutti i Task, che denota lo stato generale del sistema. Il comportamento complessivo si basa quindi sui dati tenuti costantemente aggiornati dell'oggetto State, che tiene traccia sia della modalità di funzionamento che di ogni altra informazione utile riguardante il sistema, tra cui:

- Modalità attuale
- Posizione del servo motore
- Valore della velocità del sistema
- Stato di rilevamento del sensore di movimento (stato di allarme e di tracking)
- Buffer dei messaggi ricevuti

Il sistema sfrutta uno scheduler per eseguire sequenzialmente i vari Task. I Tick sono scanditi con un tempo pari a *75 ms*, che corrisponde al massimo comun divisore dei periodi delle diverse machine a stati rappresentanti le funzionalità. Ad ogni tick dello scheduler questo controlla ogni Task che possiede e, per decidere se evocare il metodo tick(), si controlla il tempo intercorso dall'ultima chiamata dello stesso con il tempo di periodo del Task; nel caso si sia raggiunto il periodo, viene richiamato il tick. Abbiamo adottato come periodo per tutti i nostri Task *75 ms*, in quanto empiricamente dimostratosi rimanere superiore al Worst-Case-Execution-Time. Il sistema è stato modellato in questo modo in modo da garantire maggiore reattività possibile al sistema e, al tempo stesso, per far sì che non si generi overrun.

Non sono state previste sospensioni all'esecuzione dei task in quanto avrebbe aggravato la complessità del sistema, si è quindi optato per una gestione interna ai task; in questo modo ogni task, accedendo alle informazioni contenute nell'oggetto State, è a conoscenza dello stato generale del sistema e, unendo le informazioni globali a quelle locali, questi risultano autonomi dal punto di vista decisionale.

Si è trovato opportuno centralizzare l'interpretazione dei messaggi provenienti da seriale, in modo che la logica interpretativa non venga dispersa all'interno dei numerosi task.

Qualora la modalità venga modificata, il sistema è programmato per eseguire un reset di alcuni suoi parametri; i casi distinti sono due:

1. Se si esce dalla modalità Manual o Auto il servo viene lasciato nel punto in cui si trovava (non ne viene resettata la posizione); vengono inoltre resettati i valori relativi allo stato di tracking e alarm.
2. Se si esce dalla modalità Single, mentre il sistema è in risparmio energetico, oltre alle operazioni di cui sopra, si riporta anche il motore in uno stato iniziale con angolo pari a 0; se invece si esce da Single ma non in modalità risparmio energetico il comportamento è analogo alle modalità Manual e Auto

Inoltre, sono state prese diverse scelte autonome su argomenti non ben definiti nel testo del problema:

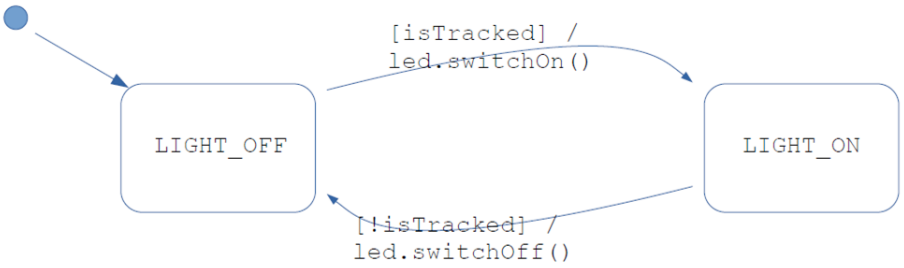
- la velocità viene decisa dall'utente in tutte le modalità
- il campionamento delle distanze non eccede i 40 centimetri (contrassegnati come D_{far})
- la funzionalità di tracking viene realizzata a prescindere dalla modalità di funzionamento
- la notifica di un oggetto fra D_{near} e D_{far} è stata mantenuta in tutte le modalità, ma solo in Auto ha semantica di alert.

3. Macchine a stati

Per la progettazione accurata del sistema sono state prodotte delle macchine a stati che mostrano il funzionamento di ogni Task. Il comportamento complessivo è dato dall'insieme delle macchine. Come strumento abbiamo trovato opportuno adottare i diagrammi di stato, facenti parte del linguaggio di modellazione UML.

3.1 Gestione del led di tracking

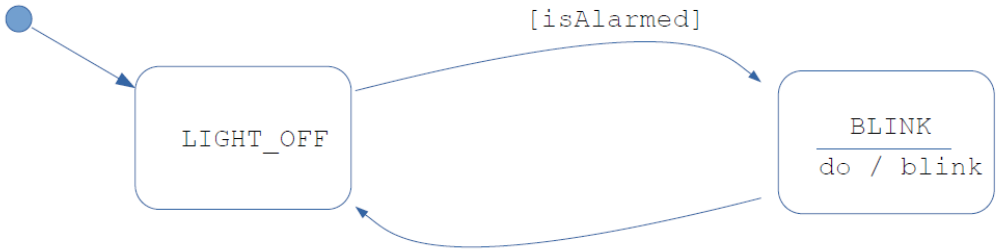
Questo led ha il compito di notificare lo stato di tracking del sistema e cioè se un oggetto è stato rilevato a distanza inferiore D_{near}.



Il funzionamento è molto intuitivo, qualora allo scattare del periodo del Task l'oggetto State abbia la variabile $isTracked$ con valore True allora il led si accenderà, e viceversa.

3.2 Gestione del led di alert

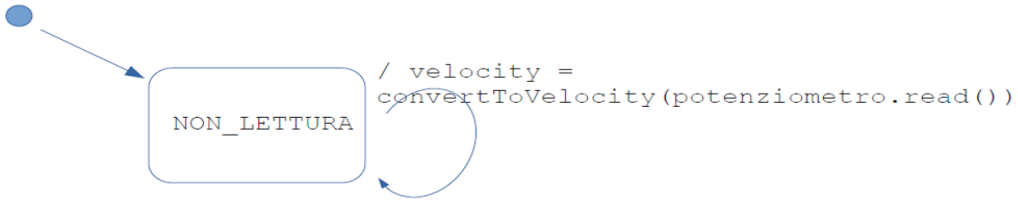
Lo stato di allarme viene notificato dal blink di un led.



Anche in questo caso il task controlla lo State del sistema e, nel caso $isAlarmed$ sia True, procede ad attivare il blinking del led, che viene acceso e spento con delay di *5ms*.

3.3 Gestione delle letture del potenziometro

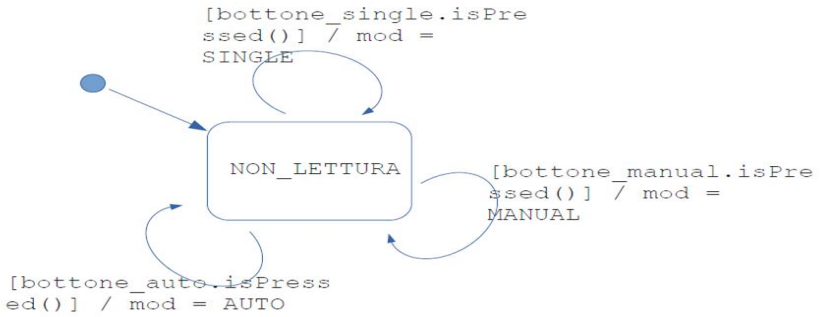
Per la gestione delle letture del potenziometro, il cui scopo è quello di selezionare la velocità di movimento del servo tra 8 differenti valori, è sufficiente controllare ad ogni periodo il valore che viene letto in una scala *0-1023* e riportarlo in un range *1-8* per selezionare la velocità del sistema in maniera crescente.



Ad ogni periodo viene quindi letto il valore analogico del potenziometro e questo viene convertito nel valore di velocità.

3.4 Gestione pulsanti per la scelta della modalità

Il task dei pulsanti, che gestisce tutti quelli disponibili, fornisce le funzioni per il cambio della modalità. Ad ogni periodo viene controllato se un pulsante è stato premuto. Oltre al cambio della modalità, si procede anche all'azzeramento del sistema secondo le regole illustrate sopra.

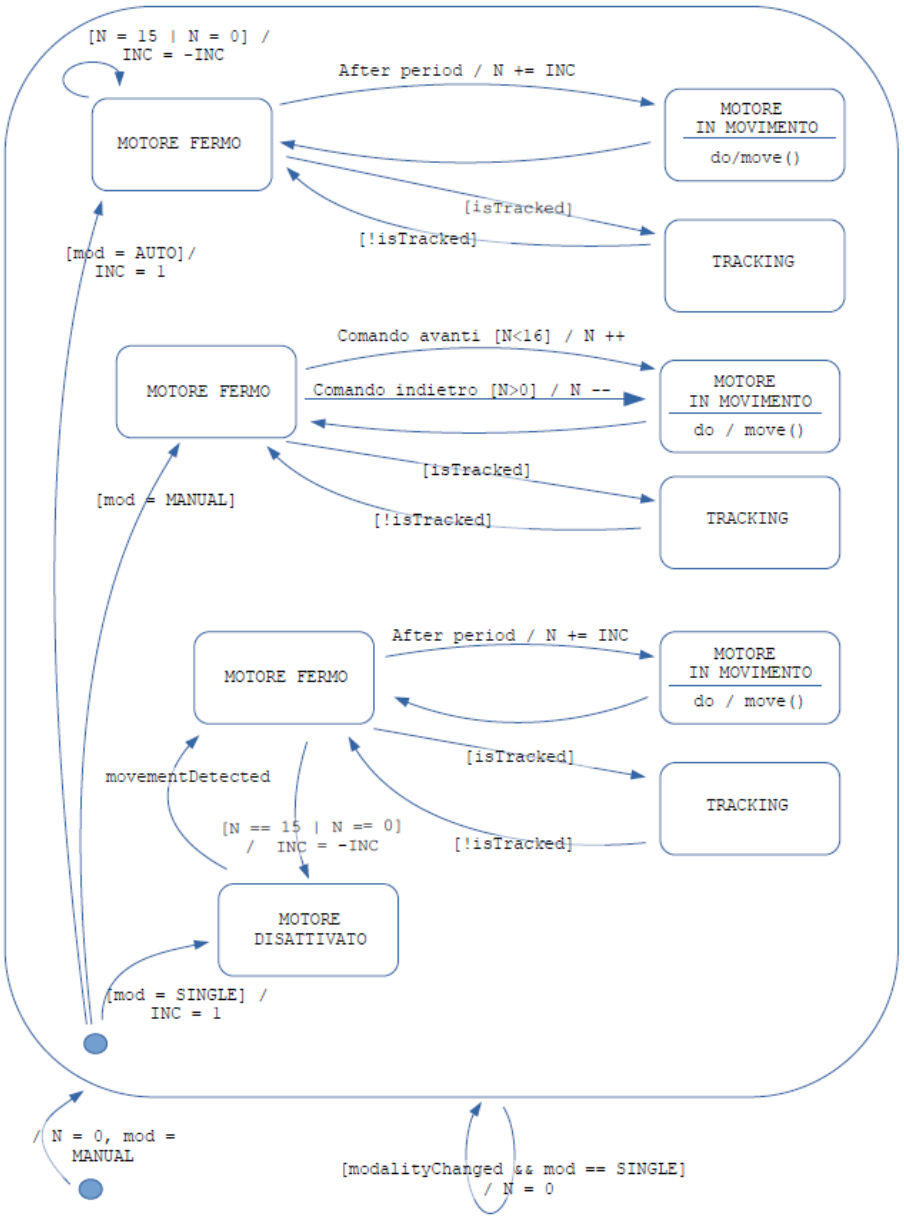


3.5 Gestione del servo motore

Il Task del motore è stato rappresentato mediante tre macchine a stati per esprimere al meglio il diverso funzionamento nelle singole modalità, seppure, a livello di implementazione, l'oggetto riguardante il Task del motore sia uno solo, come spiegato in precedenza.
All'accensione del sistema la modalità selezionata in automatico è Manual quindi l'ingresso nella macchina a stati avviene con l'angolo del servo inizializzato a 0.
Ad ogni tick il motore esegue operazioni diverse in base alla modalità in cui si trova:

1. Modalità Auto: ad ogni periodo il motore esegue un incremento dell'angolo di uno spicchio
2. Modalità Single: si attende che il sensore di movimento reagisca ad uno spostamento. In tal caso il motore avvia una scansione, salvo poi ritornare, alla fine della scansione, in posizione $0°$ o $180°$ in attesa del prossimo movimento rilevato
3. Modalità Manual: in questo caso il motore rimane fermo finché non riceve un comando apposito dalla seriale, in qual caso di muove di uno spicchio in una direzione o nell'altra (in base al comando) per poi tornare nello stato iniziale

In tutti e tre gli stati è stato implementato un controllo sulla condizione di tracking proposta come feature facoltativa. La modalità Single se non interrotta, prevede scansioni prima in un senso, e poi nell'altro; quando la modalità viene inserita la partenza è in posizione $0°$.

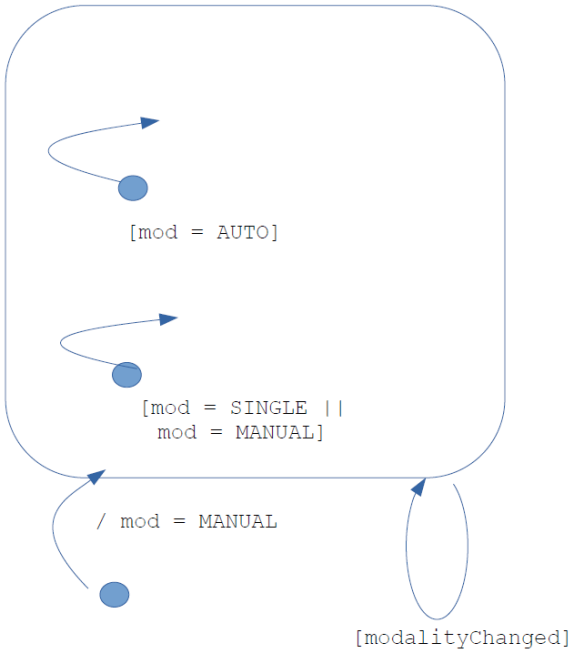


3.6 Gestione del campionamento ad ultrasuoni

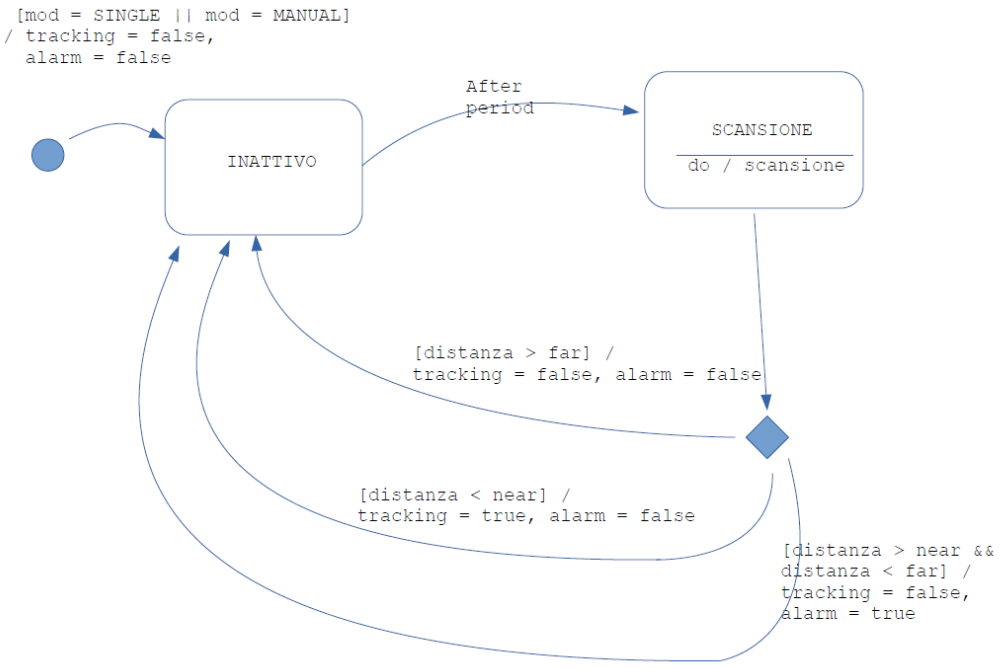
Anche il sensore di distanza è stato modellato con una forte dipendenza dalle modalità. In questo caso è stato necessario realizzare soltanto due macchine a stati, in quanto il funzionamento in Single e Manual è esattamente analogo. Tutte le modalità prevedono una scansione ad ogni periodo e il conseguente aggiornamento dell'oggetto State. Il funzionamento si distingue quindi in due:

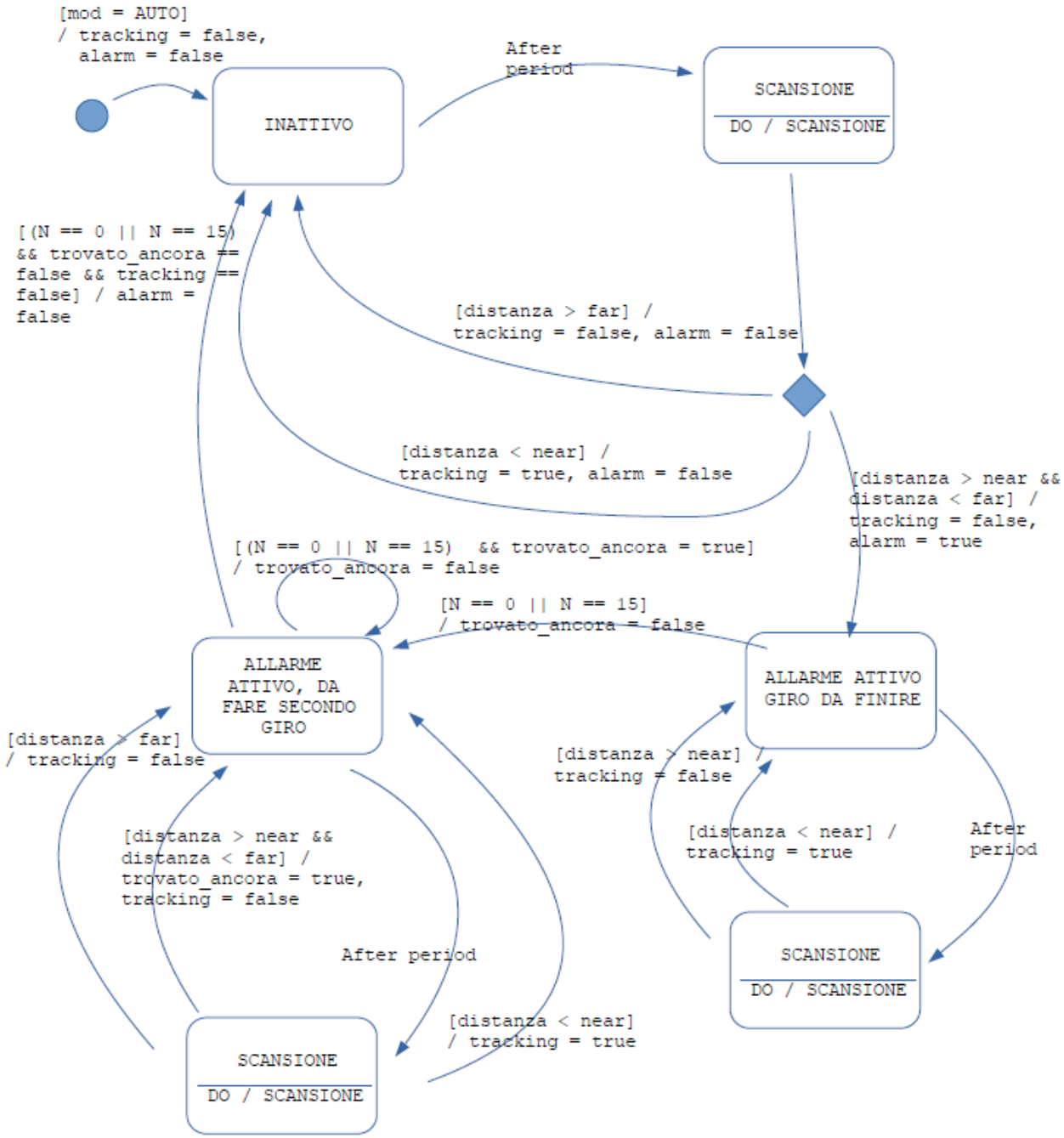
1. Modalità Single e Manual: non ha funzionalità di rilievo, ottenuta la distanza si occupa di valutare lo stato in cui il sistema si trova, distinguendo tra Alert, Tracking, oppure in cui non è stato rilevato nulla entro D_{far}.
2. Modalità Auto: modalità più complessa in quanto deve, ogni volta che si rileva un oggetto in regione di allarme, essere eseguito in controllo sulla presenza o meno dello stesso alla scansione successiva. Il sensore quindi ad ogni periodo rileva l'effettiva presenza di oggetti e, qualora si rilevi un oggetto a distanza compresa tra la minima e la massima, si entra in stato di allarme. In questa situazione il sistema deve controllare mediante due scansioni successive (inizialmente completa quella attualmente in corso arrivando a $180°$ o $0°$ e ne esegue poi un'altra completa in verso opposto) se l'oggetto individuato è ancora presente e, in caso contrario, spegne il led di allarme; diversamente prosegue con una ulteriore scansione.

È opportuno fare caso alla differenza semantica che lo stato di Allarme presenta nelle diverse modalità. In Single implica un iter di scannerizzazione da seguire, nelle altre due modalità implica l'attivazione momentanea di un flag, al solo fine di indicare che in quel specifico campionamento è stato rilevato un oggetto all'interno del range specificato.



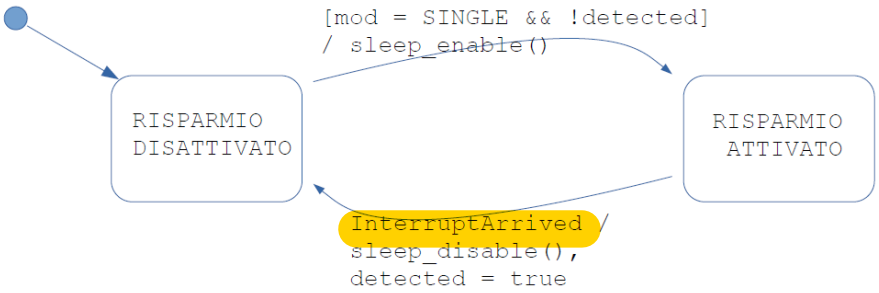
Per maggiore chiarezza visiva le macchine a stati sono mostrate separatamente, e non in unico schema. La figura sopra schematizza l'unione delle due macchine a stati, una per la modalità Single e Manual, e l'altra per Auto, che vengono invece mostrate in dettaglio di seguito.




[mod = AUTO]
/ tracking = false,
alarm = false
INATTIVO
After
period
SCANSIONE
DO / SCANSIONE
[(N == 0 || N == 15)
&& trovato_ancora ==
false && tracking ==
false] / alarm =
false
[distanza > far] /
tracking = false, alarm = false
[distanza < near] /
tracking = true, alarm = false
[distanza > near &&
distanza < far] /
tracking = false,
alarm = true
[(N == 0 || N == 15) && trovato_ancora = true]
/ trovato_ancora = false
[N == 0 || N == 15]
/ trovato_ancora = false
ALLARME
ATTIVO, DA
FARE SECONDO
GIRO
ALLARME ATTIVO
GIRO DA FINIRE
[distanza > far]
/ tracking = false
[distanza > near] /
tracking = false
[distanza > near &&
distanza < far] /
trovato_ancora = true,
tracking = false
[distanza < near] /
tracking = true
After
period
After period
SCANSIONE
DO / SCANSIONE
[distanza < near]
/ tracking = true
SCANSIONE
DO / SCANSIONE

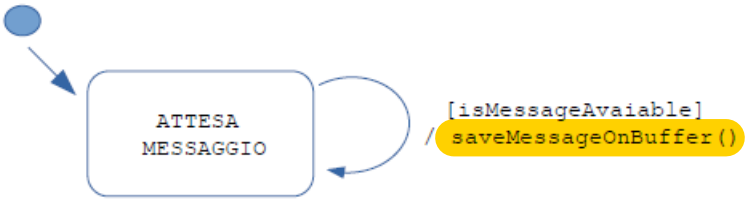
3.7 Gestione del risparmio energetico e del sensore di movimento

Il sistema rimane in stato di risparmio disattivo fintanto che non viene rilevato movimento o la modalità non è Single (quando si entra in questa modalità si procede automaticamente all'attivazione del risparmio energetico). Prima di entrare in risparmio vengono abilitate le interruzioni provenienti dai pulsanti, dalla seriale e dal PIR. Successivamente si attende in questo stato fintanto che non viene generata un' interruzione; una volta attivata ci si occupa di riattivare correttamente il sistema disattivando anche le interruzioni precedentemente attivate.



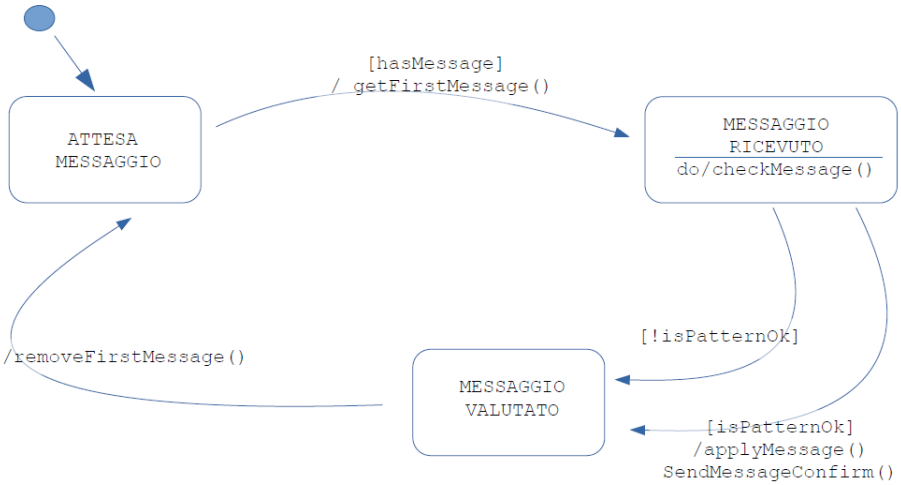
3.8 Gestione delle letture da seriale

Questo task, logicamente collegato a quello seguente, si occupa di effettuare la lettura dei messaggi provenienti da seriale e salvarli in un buffer presente all'interno dell'oggetto State.



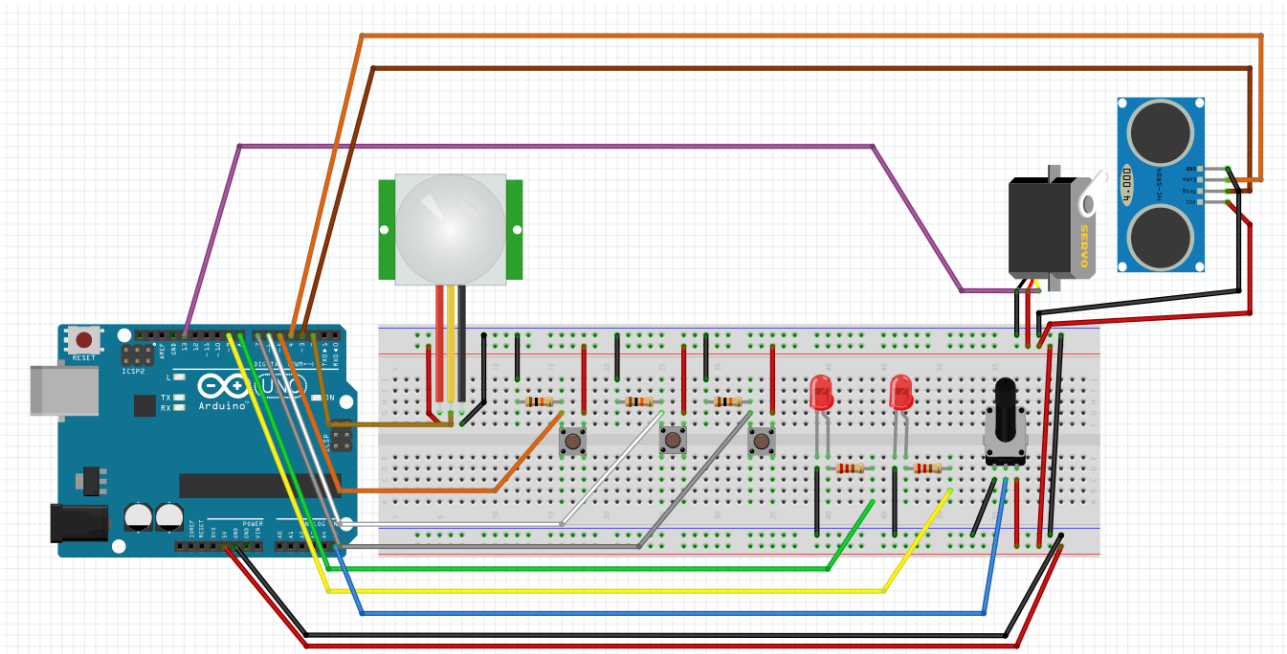
3.9 Gestione dei messaggi ricevuti

Il Task interpreta i messaggi presenti nel buffer; se validi, applica le modifiche necessarie all'oggetto State. Una volta che il messaggio è stato elaborato, viene mandato su seriale una conferma per notificare che il comando è stato ricevuto e validato con successo. Dopo aver letto il messaggio, questo viene rimosso dal buffer.



4. Schema circuitale

Schema del circuito realizzato.



5. Note

La libreria JSSC pare non funzionare con le nuove versioni di Java, viene dunque fornita, oltre al codice sorgente Java, anche una diversa versione della libreria. Nelle nuove versioni di Java, inoltre, non è presente JavaFX. Data la grande dimensione della libreria non è stato possibile fornirla assieme al resto.