Smart Radar

Il progetto in questione ha l'obiettivo di creare un sistema capace di monitorare l'area circostante con l'ausilio di un servo-motore (addetto alla rotazione di 180°), un sensore ad infrarossi passivo (addetto al rilevamento di un movimento) e un sensore ad ultrasuoni (addetto al calcolo della distanza).
Il microcontrollore, in cui risiede l'hardware e il software di controllo, comunica via seriale con un programma grafico capace di scambiare (inviare e ricevere) eventi e cambiamento di stati real-time (in un determinato istante).
Il software di controllo è stato realizzato in linguaggio C/C++ e l'interfaccia grafica in linguaggio Java utilizzando in framework grafico Java Swing.

Software C++ e diagramma macchina a stati sincrona

La logica d'implementazione di questo progetto segue il principio delle macchine a stati finite sincrone;
Nello specifico, il programma si compone di 3 macchine a stati finiti asincrone (modalità manual, single e auto), che si alternano all'arrivo di input inviati dall'applicazione java o dai bottoni cablati sull'Arduino stesso. Questo comportamento viene eseguito dall'oggetto "modeSetter".
Le modalità sono composte da Task sincroni, che si alternano quindi senza il bisogno di input esterni, abilitandosi e disabilitandosi reciprocamente per motivi di risparmio energetico; infatti due di essi (engineServo e readDistance) hanno lo stesso periodo di tick() poiché uno solo di essi è attivo in un dato istante.
Analizzando il setup è possibile notare l'inizializzazione della serial ad un valore maggiore di 9600 (solitamente utilizzato), poiché il programma è stato progettato per essere diviso in task, implementati quindi in modo che sia impossibile dividerli in sotto-task e che il loro tempo di esecuzione sia ipoteticamente nullo o tendente a 0; aumentare perciò il baudrate a 115200 contribuisce a questo scopo: pur essendo la comunicazione più soggetta ad errori, l'alternarsi dei task risulterà più veloce poiché non rallentati dallo scambio di dati.

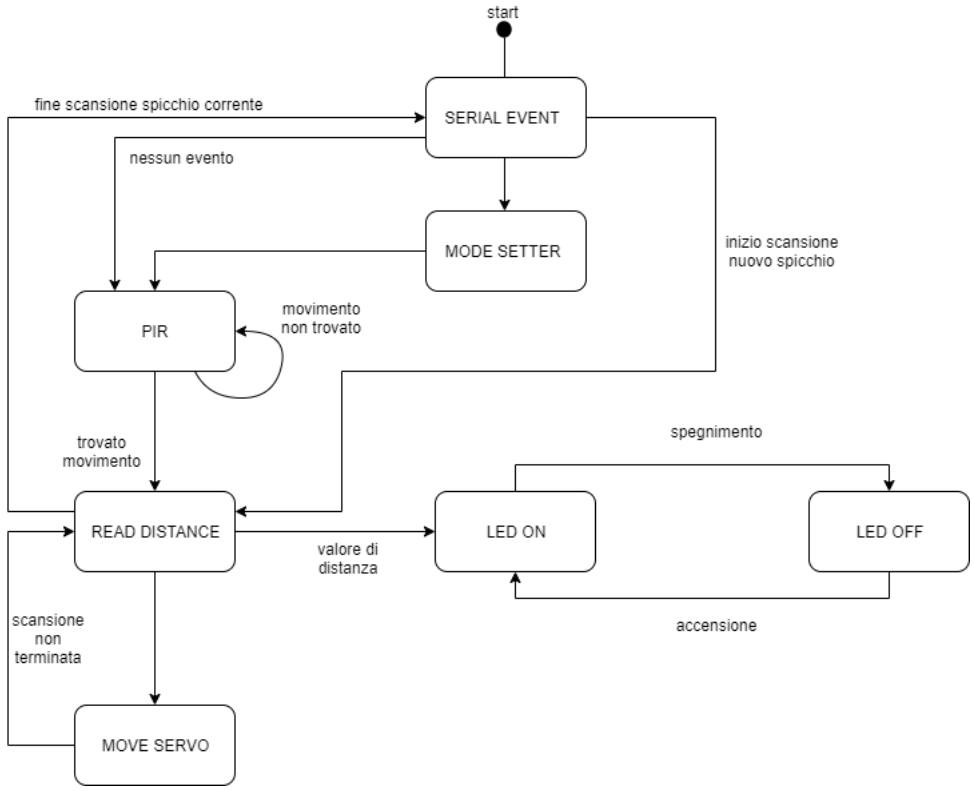
Breve descrizione dei Task

- ***BLINKTASK:*** blinking dei led in caso in cui il radar percepisca un oggetto ad una distamza variabile (di default tra 20 e 40 centimetri).
- ***BUTTONLISTENER:*** task dedicato all'ascolto dei tre tasti, è inizializzato a 40 millisecondi perché la nostra scelta implementativa prevede che l'utente non prema il tasto per una durata minore. Nel caso in cui rilevi un evento, manda sulla seriale (a java) il comando corrispondente al relativo cambio di modalità.
- ***ENGINESERVO:*** viene diviso in due casi, in base alla modalità attuale:
    - *Single e auto:* nel passaggio dalla modalità manual, il servo si sposta nello spicchio più vicino alla sua posizione attuale, successivamente il servo si sposterà di uno spicchio alla volta, alternandosi ad ogni passo con il task readDistance (Ultrasuono).
    - *Manual:* nel caso in cui serialEvent riceva un angolo, il servo andrà a posizionarsi nel suddetto angolo, abilitando la lettura dell'ultrasuono, che invierà alla GUI il valore della distanza dell'oggetto più vicino.

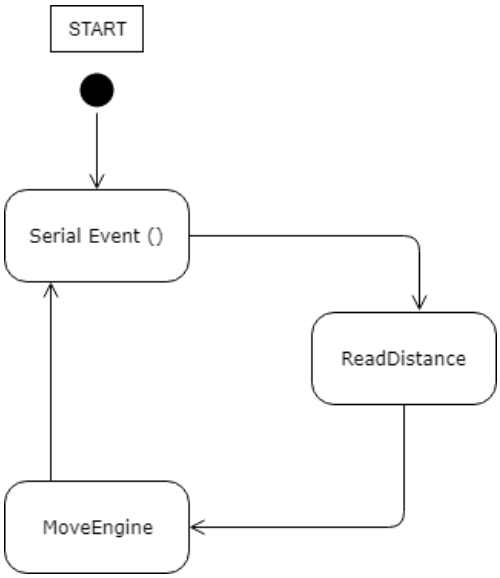
- ***PIRSLEEP:*** pensato per un risparmio energetico: infatti alla sua accensione disattiva tutti i task tranne se stesso e buttonListener. Abbiamo posto il caso in cui il pir faccia delle letture ogni 40ms; nel momento in cui rileva un movimento riabilita blinkTask e moveServo, il tick successivo del suo task viene posticipato ad 1 secondo dopo il completamento della scansione. Nel caso invece in cui non rilevi nulla rimane nel suo stato di 40ms continuando a ricercare il suo event trigger.
- ***READDISTANCE:*** il sonar effettua una scansione: viene inviato dall'ultraS un'onda PWM che, passato un certo periodo, verrà letto dall'echo pin usando la funzione pulseIn; questo è uno dei punti critici del programma, poiché, seppur molto breve, il tempo in cui il suono viaggia non è ovviamente nullo. È stata trovata come soluzione un timout inserito come 3' parametro della funzione pulseIn.
  PRO: il tempo d'esecuzione del task non viene influenzato da questo timeOut.
  CONTRO: per le distanze molto lunghe non è possibile effettuare scansioni che occuperebbero più tempo del task.
  Al termine di una scansione disabilita readTask e riabilita il servoTask).
- ***SERIALEVENT:*** abbiamo deciso di trattarla come un task (pur non essendo una vera classe che implementa l'interfaccia "Task") posizionato alla fine dell'array dello scheduler; esso è il listener della seriale, che viene eseguito solo se nel buffer seriale ci sono delle informazioni: legge quindi il primo valore sulla seriale e da questo definisce che tipo di informazione sta arrivando, successivamente imposta i valori di alcune variabili relativamente ad essa.

<u>Rappresentazione macchine a stati</u>

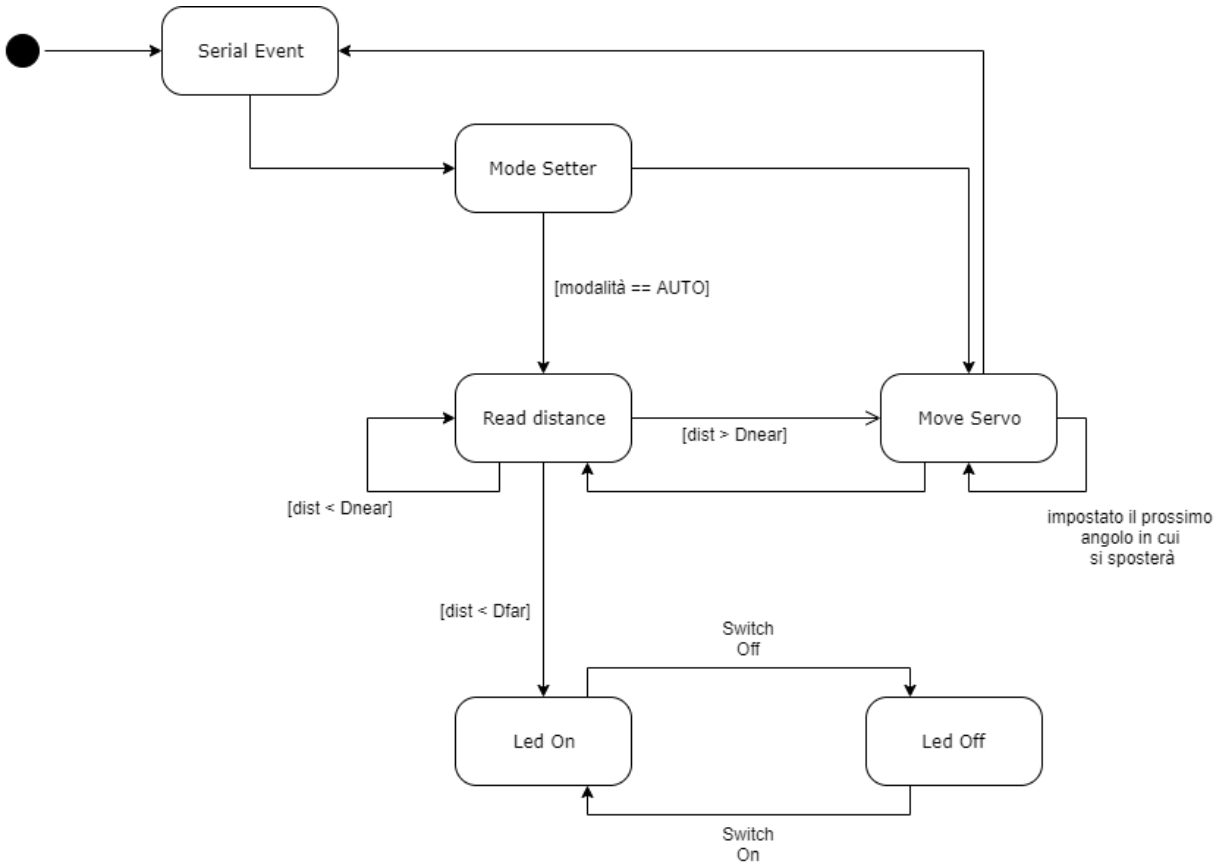
Single Mode



Manual Mode



Auto Mode



Interfaccia grafica Java

L'applicazione grafica è suddivisa in due parti, home e pannello di controllo:

- Home: è la schermata iniziale dalla quale è possibile selezionare la COM-port interessata, inizializzarla per la connessione e avviare il pannello di controllo, oltre che decidere il numero di spicchi per le modalità auto e single.
- Pannello di controllo: da questa schermata è possibile cambiare modalità, inviare nuovi valori di angolazione (solo in modalità manuale) e velocità di scansione (solo in modalità single e auto), oltre che interpretare le varie informazioni inviate da Arduino.

Schema Fritzing