Report Smart Car Washing

Annibalini Lorenzo, Bacchini Lorenzo, Sanchi Emanuele

Architettura

L'architettura del nostro progetto prevede un modulo chiamato GlobalVariables che ha lo scopo di mantenere lo stato attuale del sistema. Al suo interno sono presenti delle variabili booleane che regolano l'avanzamento e il passaggio da uno stato all'altro. In particolare, ogni Task ha accesso al modulo delle variabili globali e, al termine della propria esecuzione, va a settare su false la propria variabile di accesso e contestualmente a true quella del Task successivo . Per esempio al termine del task Welcome, troveremo due righe di codice come segue:

```
welcome = false;
moving = true;
```

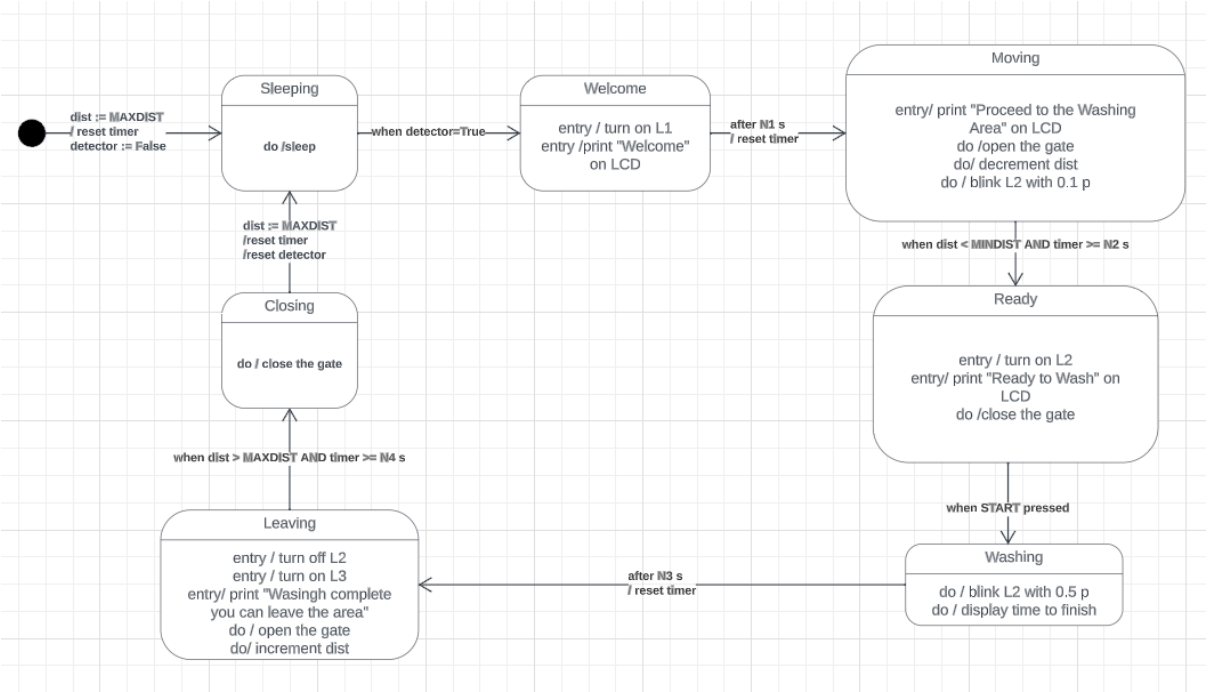
In questo modo è come se ogni task comandasse quello successivo e quindi ogni task contiene le guardie per attivare il task successivo. All'interno del metodo tick di ogni task troveremo quindi il controllo sulla propria variabile booleana che ne permetterà o vieterà l'esecuzione. L'unico task che è sempre in esecuzione e che non ha questo tipo di controllo è il Communication Task, ovvero quello adibito alla comunicazione con la GUI; questo infatti deve sempre essere in esecuzione per permettere un aggiornamento continuo della GUI.
Per regolarci coi tempi abbiamo preso spunto da quanto visto a lezione e abbiamo quindi deciso di utilizzare una variabile counter che viene aumentata a ogni tick di una quantità pari al period del task stesso; quando poi il counter sarà pari all'N che regola quel determinato task, allora si scatenerà l'evento di guardia e accadrà quanto descritto nella prima parte. Da notare che ciò non accade sempre, ma solo nei task che necessitano di questo controllo.

Scelte implementative

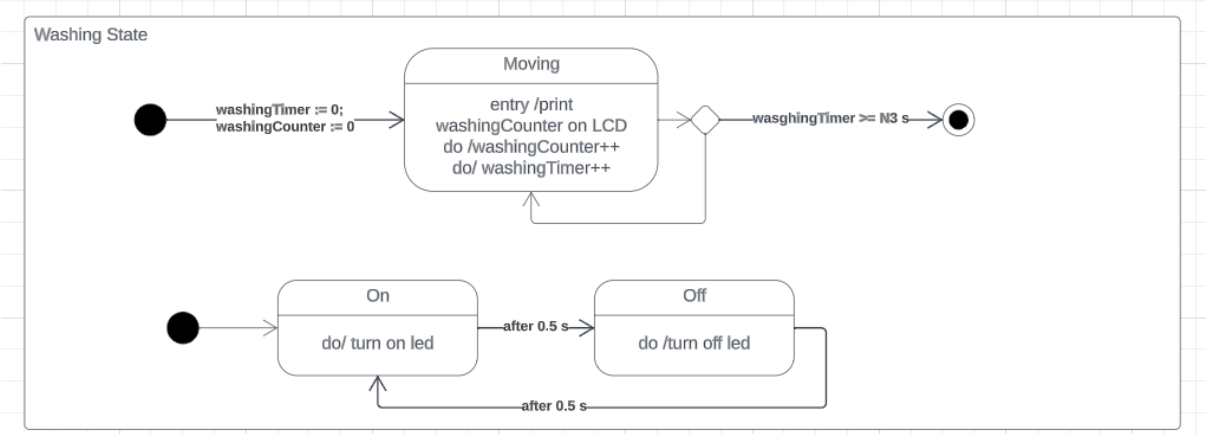
Durante lo sviluppo ci siamo imbattuti in qualche difficoltà riguardo la gestione dei tempi.
Ci siamo accorti che il problema principale sorgeva quando il sistema transitava per il ready task e attendeva quindi la pressione del bottone per cambiare di stato. Probabilmente per un errato utilizzo degli interrupt o per problemi della libreria TimerOne.h, il sistema smetteva di reagire ed entrava in un loop infinito in cui la funzione timerHandler all'interno dello Scheduler non veniva mai richiamata. Abbiamo quindi optato per la rimozione di TimerOne.h e l'utilizzo del calcolo della differenza tra millis() e previousMillis per cambiare lo stato della flag.
Altra scelta implementativa che ci siamo ritrovati a dover fare è stata riguardo alla gestione del rimbalzo del bottone. Infatti, nonostante all'interno della classe Button.cpp tramite il metodo resolveBouncing effettuassimo già un controllo del bouncing in alcuni casi il bottone rimbalzava per più tempo e questo faceva sì che il Ready task non aspettasse la pressione del tasto per proseguire allo stato successivo. Abbiamo quindi inserito alla fine del thick del Ready task una assegnazione alla variabile buttonPressed = false in modo da evitare tale problema.

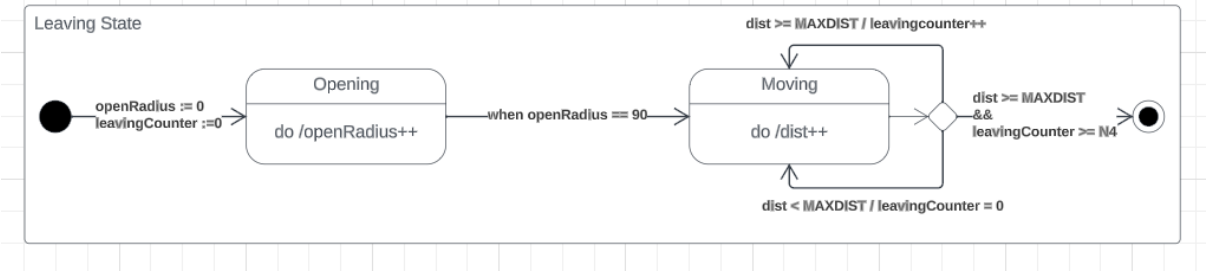
Diagramma stati finiti

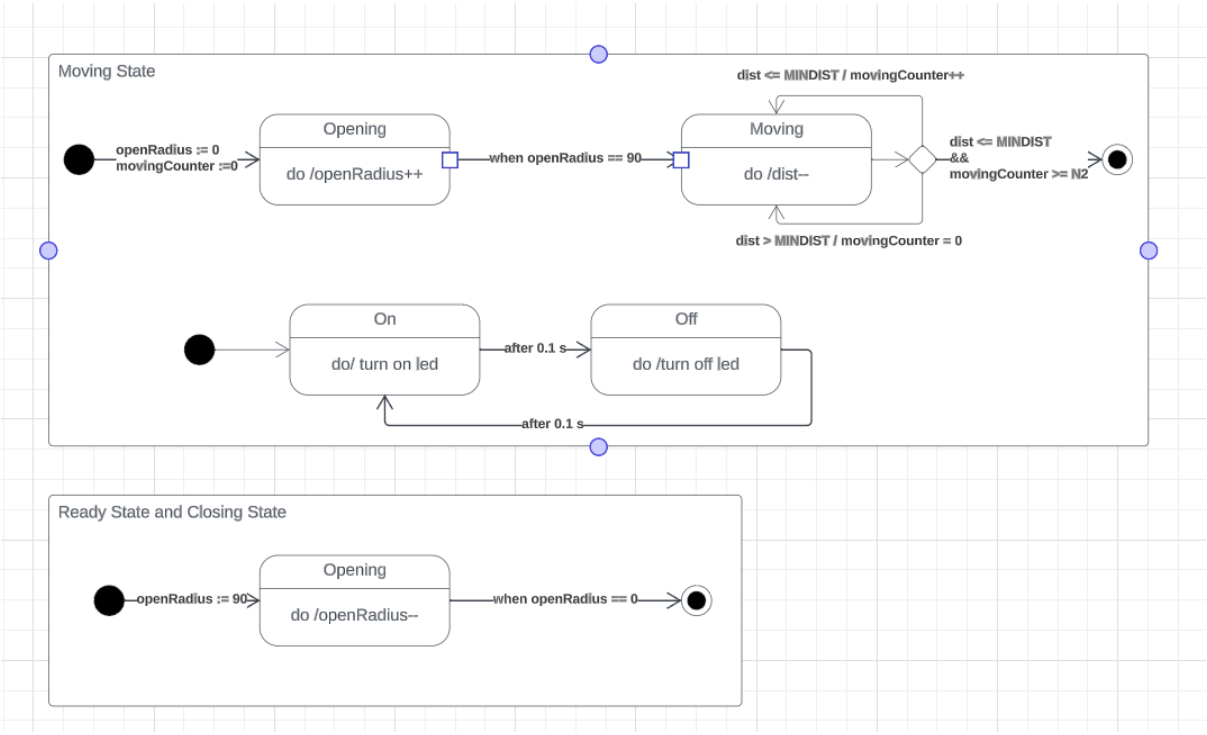
Schema completo



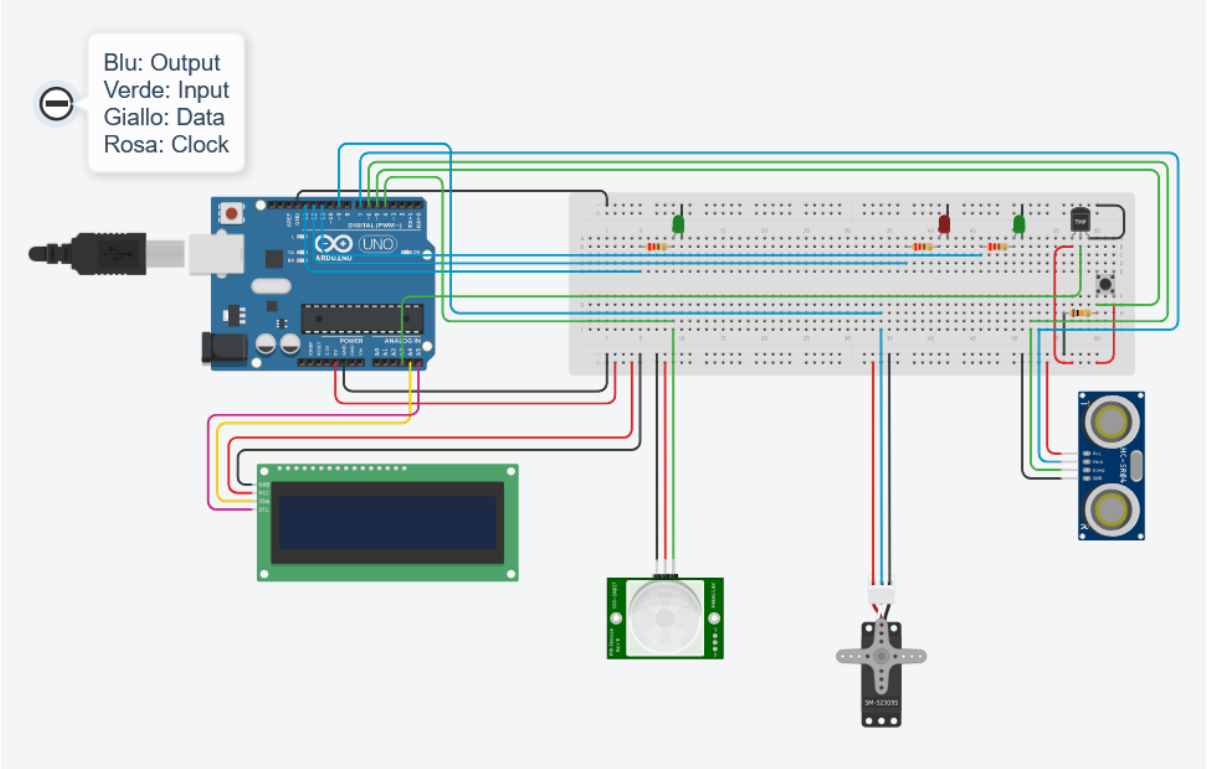
Esplosioni del diagramma a stati:







Schema tinkercad



*i pin utilizzati nello schema ed i pin nel video esempio potrebbero non coincidere