Relazione Smart GreenHouse

Introduzione

Lo scopo di questo progetto mira ad implementare un simulatore di un sistema intelligente di una serra che analizzando il valore di umidità sia in grado di reagire, attraverso la scheda programmabile Arduino, l'utilizzo del componente ESP8266 per la comunicazione via IP/http, lo sviluppo di un applicazione Android e il supporto di un modulo Bluetooth HC-06, di un server in Java e un Front End realizzato tramite i linguaggi HTML, CSS, JavaScript e PHP. Il sistema deve fornire all'utente la possibilità di operare in due diverse modalità, di norma essere in grado di agire autonomamente, invece su richiesta dell'utilizzatore cedergli il controllo dell'intero funzionamento.

Progettazione

Nella fase di design del progetto, sono state ideate le modalità di comunicazione tra le varie componenti, in quanto sia per il sistema che per l'utente è necessario conoscere in tempo reale lo stato della serra per poterla curare al meglio. Inizialmente viene utilizzata la scheda ESP8266 per prelevare il valore dell'umidità, tramite l'apposito sensore, e comunicarlo all'applicazione server di Java, alla quale è assegnato il compito di far conoscere, tramite seriale, il valore ad Arduino che si occuperà di gestirlo con appositi task e di renderlo visibile tramite connessione Bluetooth all'applicazione (non emulata) di Android. Inoltre il server memorizza i dati ottenuti, in un Database in modo tale da rendere reperibili le informazione anche a Front End. L'applicazione implementata per Arduino, è stata organizzata in task, ognuno dei quali presenta un differente compito per il corretto funzionamento dell'intero sistema. Nel dettaglio abbiamo strutturato due task principali dai quali viene effettuata la gestione delle due modalità di utilizzo (AutoModeTask e ManualModeTask), le quali vengono però gestite da un "precedente" task (ModeTask) che segnala al sistema quale dei due deve essere il dominante. Questi hanno il compito di gestire gli stati del sistema, controllare il corretto funzionamento e il settaggio delle variabili condivise in modo da eseguire gli altri task nel momento opportuno, questi ultimi sono divisi in due tipologie:

- Task di rilevazione : Sono task che assegnano il valore letto dai sensori alle corrispettive variabili condivise, in modo da permettere al task principale di avere i dati dei sensori sempre aggiornati, ma senza doversene occupare lui direttamente.

- Task di esecuzione : In modo differente, questi task vengono "avviati" dal main task ed eseguono delle routine come il task di irrigazione e la comunicazione con la piattaforma in Java. In modo più accurato, questi task vengono eseguiti dallo scheduler in continuazione come gli altri task e con stessa priorità, ma al loro interno hanno un semplicissimo controllo sulla variabile condivisa rispettiva e nel caso in cui si a impostata a false, il task non esegue niente. Altrimenti viene eseguita la routine specificata.

Architettura

Siamo partiti con una progettazione orientata agli oggetti e che al suo interno includeva la creazione di più task, ognuno con un determinato compito e la possibilità di comunicare con gli altri attraverso delle variabili condivise (dal file SharedState.h). Per la realizzazione del sistema abbiamo implementato i seguenti task, che rispecchiano il funzionamento delle FSM mostrato nelle immagini rispettive.

ARDUINO

- **AutoModeTask**: Il task AutoModeTask è uno dei due principali, usato per eseguire il sistema in modalità automatica, a lui appartiene la responsabilità di far rispettare la giusta sequenza di sviluppo e di attivazione dei task, ad esso viene assegnato un periodo di 50ms. Lo stato da cui il task comincia è di **IDLE** in cui aspetta che il valore di umidità scenda sotto un certo limite per passare ad un stato di scelta (**CHOOSE**), a questo punto il sistema deve capire quale portata deve essere erogata, terminato il procedimento si avvia verso lo stato di **WAIT** il quale rappresenta la durata di erogazione dell'acqua, in questo passo si procede anche all'invio su seriale di tale valore in modo da poter essere poi, tramite Java, inserito nel DB di supporto per il Front End, nel caso in cui l'erogazione fallisca e quindi si presenti l'errore di un tempo maggiore o uguale di 5sec, il sistema verte allo stato di **ERROR** nel quale viene comunicato l'errore, ed effettuato un ritorno allo stato di **IDLE**, anche nel caso tutto proceda senza errori in questa fase si ha un ritorno allo stato di **IDLE]**.

UNREGISTERED UNREGISTERED UNREGISTERED UNREGISTER

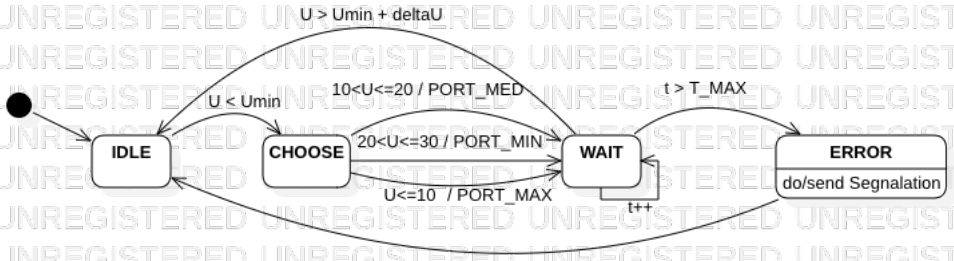


Figura 1: Diagramma a stati AutoMode Task

- **DistanceTask**:

  Il DistanceTask incapsula la logica per rilevare la distanza che intercorre tra la persona e il sistema, grazie ad esso è possibile decidere se rendere possibile il passaggio in altri task. Si avvale della classe Sonar che astrae il funzionamento del sensore per la rilevazione della distanza (Sonar). Necessita di un periodo per gestire le rilevazioni sulla distanza, inizializzato a 50ms. Ad ogni tick del periodo il task viene eseguito per poter confrontare sempre il valore quando richiesto. Per gestire il rilevamento corretto da parte del Sonar si è deciso di utilizzare una media pesata dei valori presi in INPUT, cioè i valori precedentemente rilevati avranno una priorità pari ad α invece il valore rilevato correntemente avrà priorità 1-α così da gestire gli sbalzi che il sensore ha in certi rilevamenti.

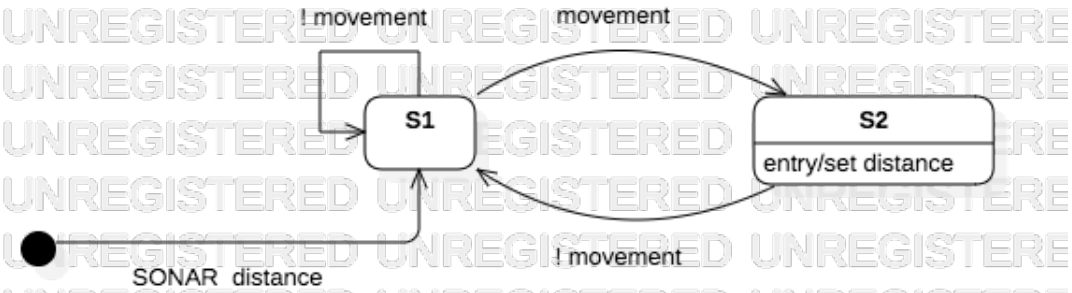
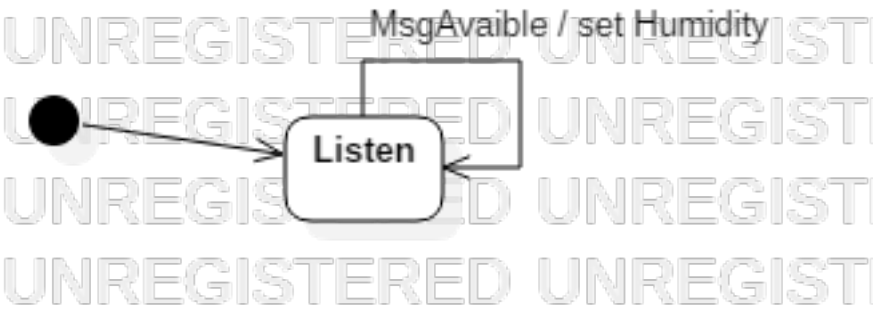


Figura 2: Diagramma a stati Distance Task

- **HumidityTask**: Il task HumidityTask si occupa della ricezione del valore dell'umidità presente, inviatogli tramite seriale dall'implementazione Java, grazie al dato ottenuto è possibile sapere quando erogare acqua se si è in modalità AUTO. Questa funzionalità necessita di un un periodo pari a 50ms, cosi facendo ad ogni tick del periodo il procedimento viene eseguito.



UNREGISTERED UNREGIST
UNREGISTERED UNREGIST
UNREGISTERED UNREGIST
UNREGISTERED UNREGIST

Figura 3: Diagramma a stati Humidity Task

- **IrrigationTask**: Quando l'umidità diminuisce rispetto ad una certa soglia minima, o su richiesta dell'utente, la pompa dell'acqua deve essere aperta per poter aumentare l'umidità; tale comportamento è rappresentato dal task IrrigationTask che sfruttando la classe ServoMotor è in grado di simulare il procedimento di apertura e chiusura della pompa, attraverso il servo motore e quindi di procedere all'innaffiamento. Questo task presenta un periodo di 100ms in cui ad ogni tick viene controllata la possibilità di essere eseguito.

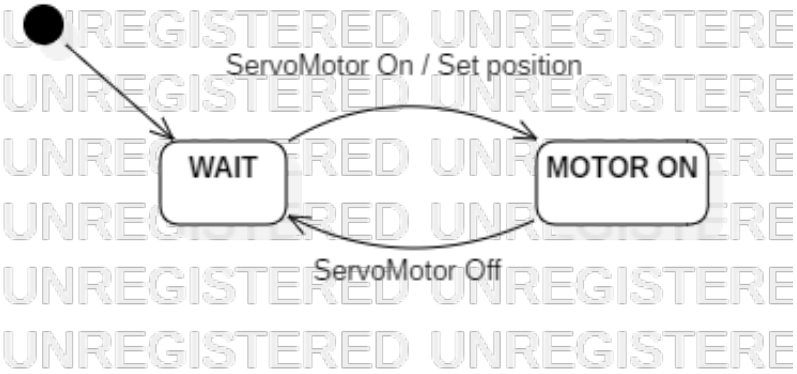


Figura 4: Diagramma a stati Irrigation Task

- **ManualModeTask**: ManualModeTask è l'altro task principale, viene attivato quando l'utente è ad una giusta distanza dal sistema (rilevata dal sonar) e quando richiede, tramite app Android, di essere connesso tramite modulo Bluetooth al sistema, diventando cosi in grado di poter gestire la propria serra da telefono. Il suo funzionamento è dettato da un approccio di scambio messaggi, inoltre in questo task bisogna costantemente inviare ad Android il valore di umidità corrente, e in base a ciò che viene ricevuto agire nel corretto modo. Questo task presenta un periodo di 50ms, nel quale ad ogni tick controlla se ha ottenuto il comando del sistema, se ha ricevuto messaggi da gestire e invia il corrente valore di umidità.

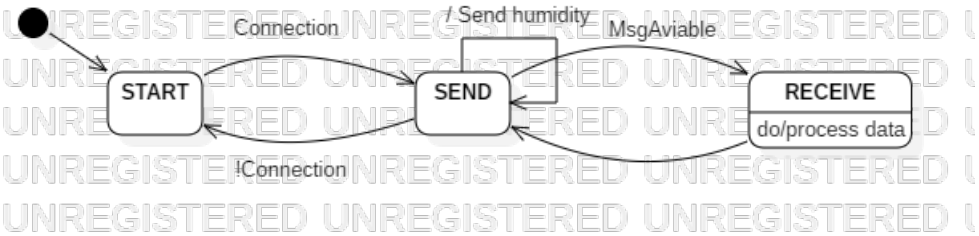


Figura 5: Diagramma a stati ManualMode Task

- **ModeTask**: Come già anticipato il task ModeTask ha una funzione molto importante, indirizzare la modalità in cui il sistema dovrà operare, se quindi in modalità AUTO o MANUAL, ciò avviene analizzando la distanza

ottenuta dal DistanceTask e se l'utente ha richiesto di essere connesso, controllando quindi le variabili condivise siamo in grado di agire nel giusto modo. Al task è stato associato un periodo di 100ms, il cambio di modalità può avvenire in qualsiasi momento per questo motivo ad ogni tick viene effettuato il controllo. Quando viene selezionata una delle due modalità viene anche fatto accendere il corrispettivo led e spento il suo opposto.

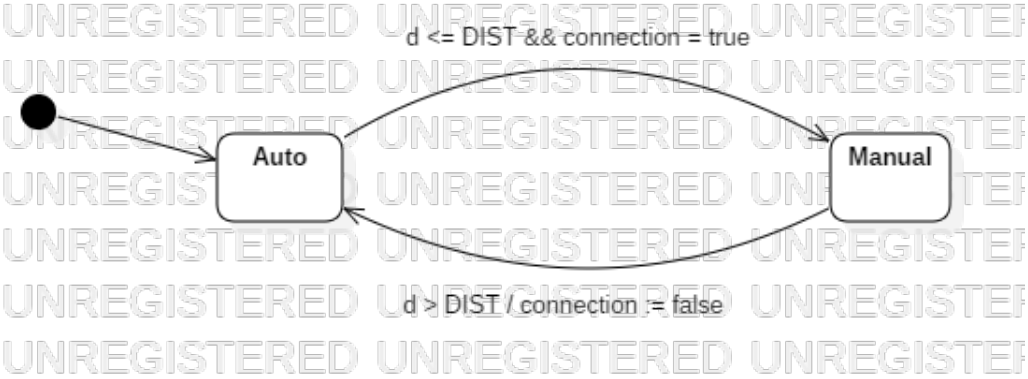


Figura 6: Diagramma a stati Mode Task

JAVA

Per quanto riguarda la parte server Java, si ha un Data service che ascolta i messaggi in arrivo da un dispositivo ESP tramite il servizio Ngrok che crea un tunnel tra i due dispositivi. C'è un controller che permette di connettersi ad Arduino e di avviare un Agent di monitoraggio che rimane in ascolto di tutti i messaggi provenienti da Arduino e carica tutte le informazione necessarie al Front End su DB.

ANDROID

La parte di sviluppo android sfrutta un asyncTask per poter stabilire una connessione Bluetooth con la scheda Arduino, tramite questa connessione avviene la ricezione di messaggi, in questo specifico caso verrà costantemente ricevuto il valore dell'umidità, informazione utile all'utente per poter prendere decisioni sul sistema. I messaggi inviati necessari per poter far operare il sistema come voluto da utente, sono invece attaccati ad un handler che gestisce la comunicazione tramite un looper.