Abbiamo deciso di decomporre il comportamento complessivo del programma in 3 task, ognuno di essi è descritto da una macchina a stati sincrona. I tre task presentano una forma di dipendenza relativa ai dati che permette loro di comunicare, per fare ciò abbiamo inserito variabili globali condivise in lettura e scrittura. Abbiamo modellato ogni task come classe separata, tutti implementano l’interfaccia Task, con i metodi init() e tick(). Inoltre abbiamo creato uno scheduler per gestire il multi-tasking cooperativo con scheduler di tipo round-robin, esso tiene traccia della lista dei task in esecuzione mediante un array e dà la parola ad un task alla volta, non dovremo quindi preoccuparci di gestire eventuali corse critiche. Ogni task viene eseguito in modo atomico e quindi non ulteriormente decomponibile. Ogni task ha un proprio specifico periodo, lo scheduler funziona con periodo che corrisponde al massimo comune divisore dei tre periodi, ogni task capirà se deve o no eseguire le proprie istruzioni.

Spiegare le variabili globali

DETECT MOTION

Questo task gestisce le informazioni fornitegli dal pir e dal sonar, esegue i vari controlli e setta i valori

Questo task interroga il pir e il sonar e, a seconda dei valori che essi assumono, decide come far variare i valori delle variabili booleane globali. Questi due dispositivi rilevano la presenza di un individuo e la relativa distanza, a seconda di tali valori deduco il comportamento dell’utente e decido se attivare il sistema e quali operazioni saranno disponibili. Scriviamo qualcosa sulle azioni generali che deve gestire??

* presenza
* vicino per chiedere caffè
* vicino per prendere caffè
* tornare indietro

Abbiamo deciso di articolare questa macchina in sei stati:

* DM0: il task rimane in questo stato finché non rileva alcuna presenza, quando il pir restituisce un valore positivo, quindi ha rilevato un movimento, imposta la variabile isPresent = true. Grazie a questa variabile l’intero sistema può osservare la presenza di qualcuno e prendere le eventuali decisioni in merito a questo evento.
* DM1: questo stato interroga il sonar per capire se, e per quanto tempo, è presente qualcuno ad una distanza inferiore a DIST1. Se rilevo questa distanza per un tempo >DT1 allora passo allo stato successivo e setto la variabile correctDistance = true, altrimenti attendo. Per effettuare il controllo sul tempo che è trascorso utilizziamo una variabile t che viene incrementata ogni periodo. È necessario poter tornare allo stato DM0 nel caso in cui l’individuo si sia avvicinato per poco tempo o nel caso in cui il pir abbia restituito un valore sbagliato per cui non c’è più nessuno nelle vicinanze.
* DM2: rimango in questo stato finché l’utente risulta essere ad una distanza inferiore a DIST1, ma egli non svolge altre azioni (es. non richiede un caffè). Se però l’individuo si allontana per un certo tempo allora dovrò impostare correctDistance = false e passare allo stato successivo, DM3. Nel caso in cui l’utente chieda un caffè allora setterà il valore della variabile newCoffe a true e ciò scatenerà l’evento che mi permette di cambiare di stato e passare a DM4.
* DM4: In questo stato il task sa che il sistema sta facendo il caffè quindi non effettua rilevazioni, si preoccupa di cambiare stato solo quando il caffè è pronto, possibile uscire da questo stato solo dal momento in cui il caffe risulti essere pronto per essere prelevato, quindi quando la variabile coffeReady sarà impostata a true passeremo allo stato DM5.
* DM5: questo stato controlla il valore restituito dal sonar, se esso risulta essere inferiore a DIST2 o se sono in questo stato per almeno DT4 secondi allora passo allo stato DM3 ed imposto coffeTaked = true e coffeReady = false.
* DM3: in questo stato controllo se il pir rileva o no un movimento, se per un tempo maggiore di DT2b non rilevo nessuno allora passo allo stato DM0, se invece rilevo qualcuno passo allo stato DM1.

Nel risolvere questo task abbiamo riscontrato alcuni problemi con il pir e il sonar. Il pir risulta essere uno strumento poco preciso, in molti casi può non rilevare in modo corretto la presenza di un movimento o la osserva dopo un tempo abbastanza lungo. Il sistema sembra essere poco reattivo, ma tale problema è dato dall’accuratezza dello strumento. Il sonar in alcuni casi esegue misurazioni sbagliate con valori molto elevati, per evitare di considerarli come valori validi, abbiamo deciso di effettuare una media ponderata tra la media dei valori precedentemente misurati(70%) e l’ultimo valore misurato(30%).Questa soluzione risulta essere, a nostro avviso, efficiente nell’offuscare i valori errati, allo stesso tempo siamo consapevoli che può causare un ritardo nella notifica.

MAKE COFFE

Questo task simula l’azione di fare il caffè gestendo l’accensione progressiva dei tre led.

Questa macchina a stati si articola su quattro stati:

* L0: è lo stato di inizio, il sistema rimane in questo stato finché non viene richiesto un nuovo caffè, cioè finché la variabile newCoffe == false. Quando tale variabile assume valore positivo, allora effettuerò il cambio di stato e passo a L1.
* L1: questo stato accende il primo led per DT3/3 s per poi passare allo stato L2.
* L2: questo stato si preoccupa di spegnere il primo led e tenere acceso il secondo per DT3/3 s, successivamente passa allo stato L3.
* L3: è l’ultimo stato, esso rappresenta l’accensione del terzo led e lo spegnimento del secondo. Infine ritorna allo stato iniziale L0 e setta coffeReady = true, newCoffe = false e azzera count.

Gli stati L1, L2, L3 hanno bisogno di ricordarsi quanto tempo è passato, per fare ciò utilizziamo una variabile count che incrementiamo ogni periodo di tempo, una volta che essa assume valore maggiore o uguale di DT3 allora dovremo cambiare di stato e resettare tale variabile.

USER HANDLER

Questo task è responsabile dell’interazione con l’utente, sia per scambiare messaggi con esso attraverso l’interfaccia grafica, che per accettare richieste. Tra le vaire azioni: zucchero, bottone tattile per richiesta coffe e refill.

Il suo comportamento si organizza in sei stati esplicitati nell’enumerazione MainState:

* STANDBY: rappresenta lo stato di partenza, finché la macchina rimane in questo stato viene attivata la modalità di risparmio energetico. Eseguirà il passaggio di stato verso ON solo nel momento in cui isPresent risulterà true.
* ON: il sistema rimarrà in questo stato per tutto il tempo in cui viene rilevata un movimento, ma esso non è sufficientemente vicino. Quando la variabile correctDistance == true, allora effettuerò il passaggio di stato a READY. Devo poter anche tornare indietro da questa situazione nel caso in cui l’utente si allontani a tal punto che il pir non rilevi più la sua presenza e che quindi setti il valore di isPresent a false. In questo caso tornerò in modalità STANDBY.
* READY: questo stato indica ch
* MAKECOFFE:
* TAKECOFFE:
* MAINTENANCE: