Relazione Progetto #2

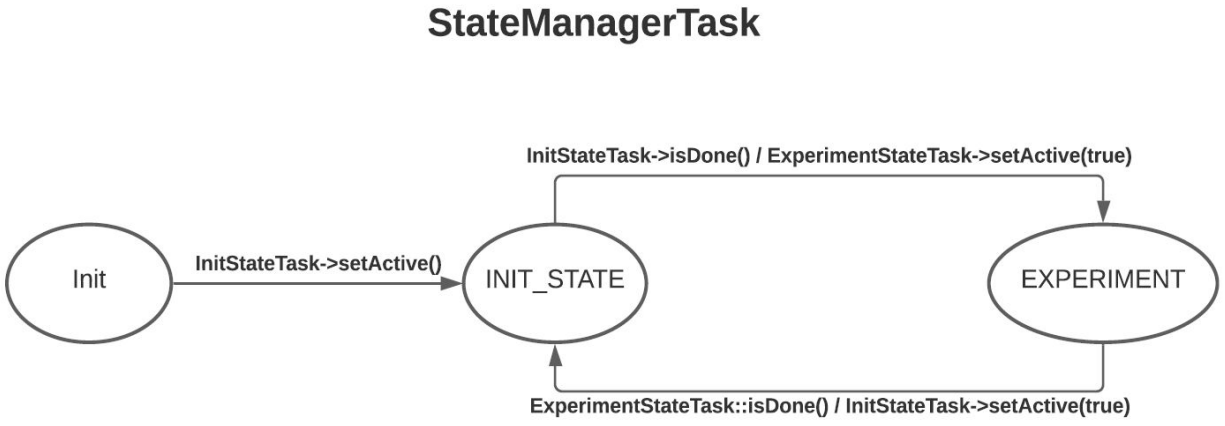
Smart Exp

Studenti: Colotti Manuel Enrique, Lisotta Marco.

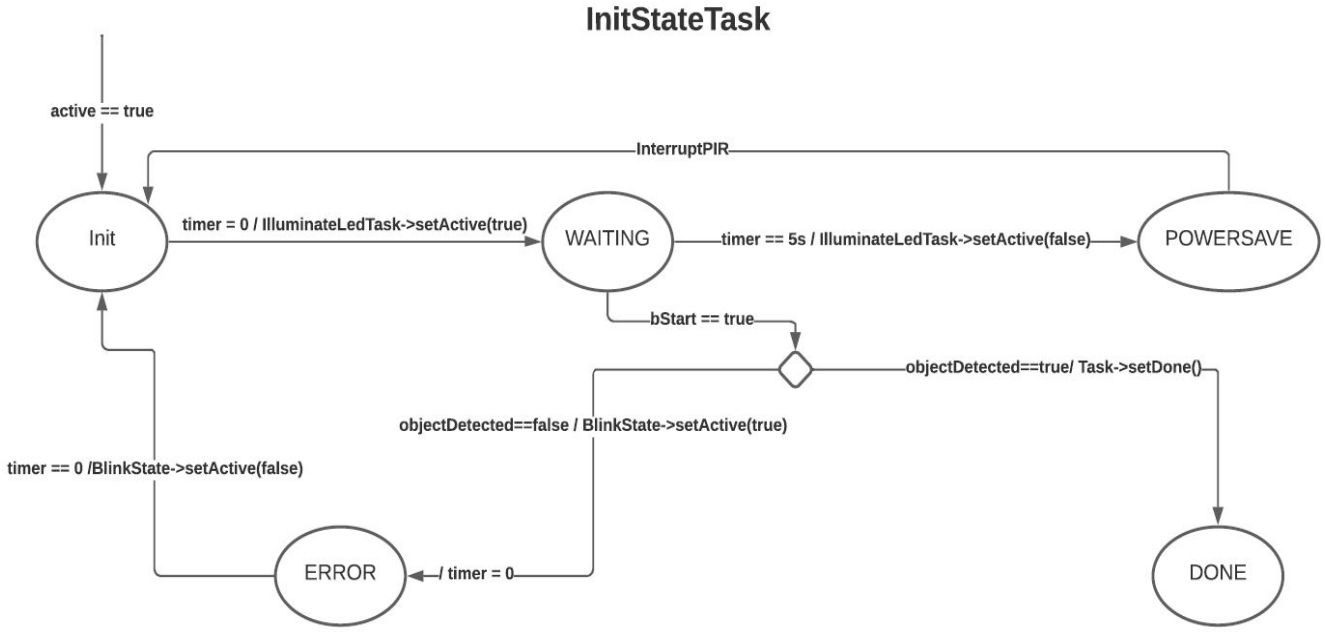
Modellazione a Task

Per iniziare la realizzazione del sistema è stato necessario capire come suddividere in Task i vari compiti di cui ci si sarebbe dovuti occupare. A tal fine abbiamo sviluppato delle macchine a stati in grado di descrivere il funzionamento del sistema in linea di massima (*Vedi in fondo al file lo schema completo*).
La suddivisione a cui siamo giunti è la seguente:

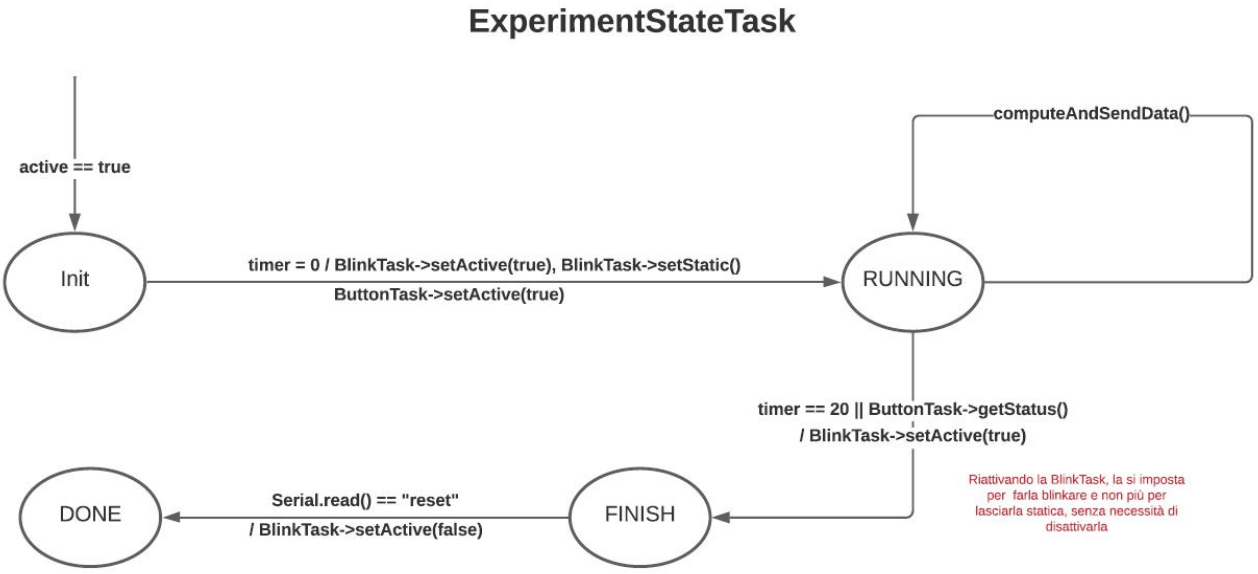
- **StateManagerTask**: Questo Task si occupa di gestire tutti gli altri sotto-task, attivandoli e disattivandoli quando necessario. Il discriminante per decidere se un Task debba attualmente essere eseguito, è la fase dell'esperimento in cui ci si trova attualmente. Comportandosi in questo modo sarà così possibile risparmiare le limitate risorse di cui disponiamo al fine di aumentare l'efficienza.



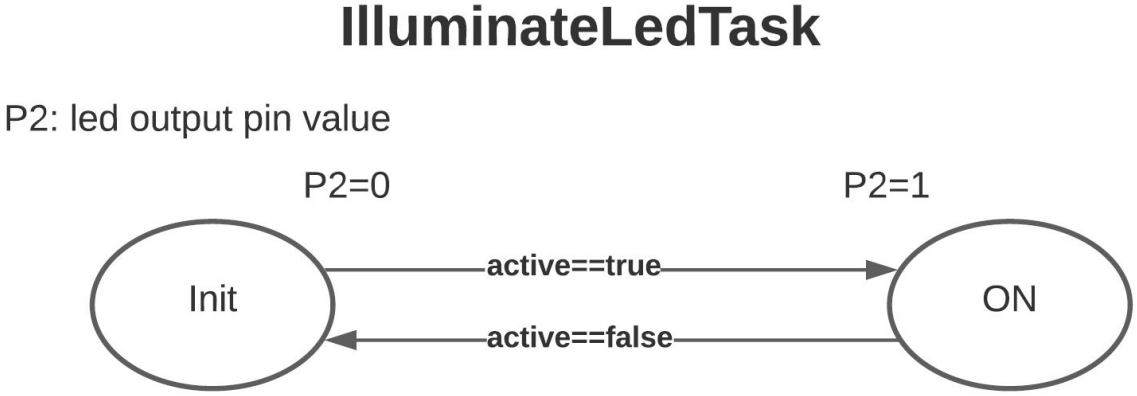
- **InitStateTask:** InitStateTask gestisce la fase iniziale e terminale dell'esperimento, nonchè i momenti in cui il sistema si trova in errore o in modalità risparmio energetico. Al completamento dei propri compiti, questo task e ExperimentStateTask descritto di seguito, mettono in pausa la loro esecuzione fino ad una nuovo cambio nel loro stato comandato da parte dello StateManagerTask.



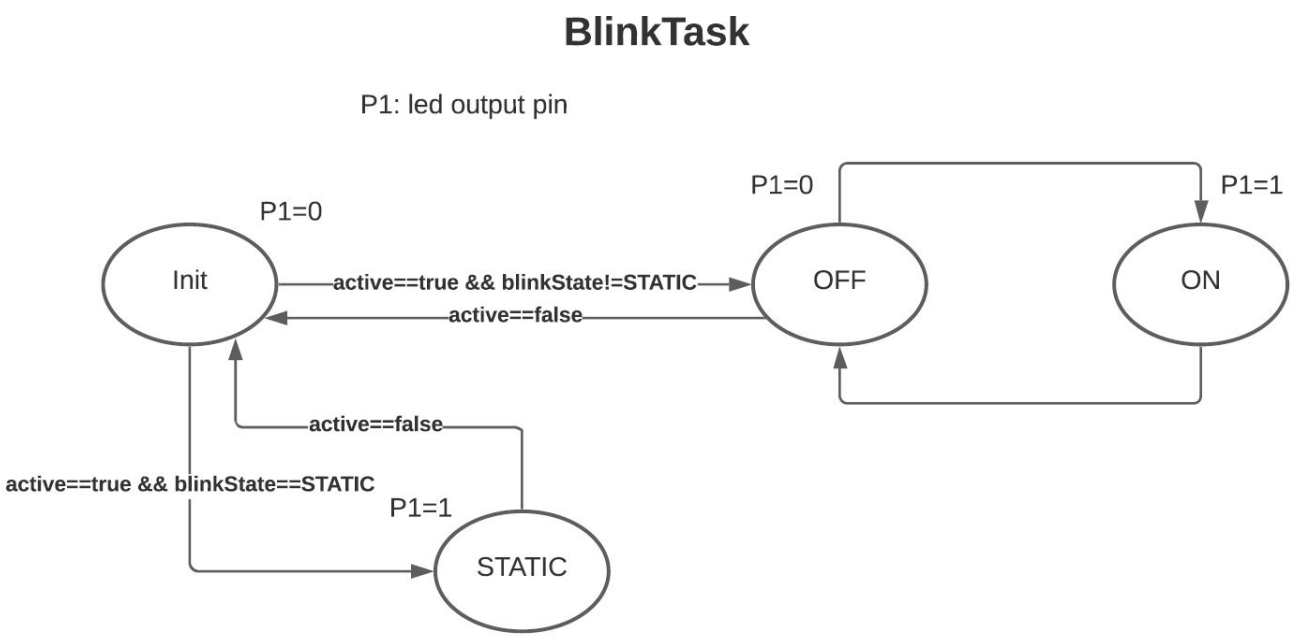
- **ExperimentStateTask:** ExperimentStateTask gestisce la fase più importante di tutte, ossia il campionamento dei dati da parte dei vari sensori e il loro invio al client su PC attraverso una comunicazione su seriale. Il suo periodo di esecuzione (e quindi di campionamento dei dati) può essere impostato dinamicamente ad ogni esperimento grazie allo StateManagerTask.



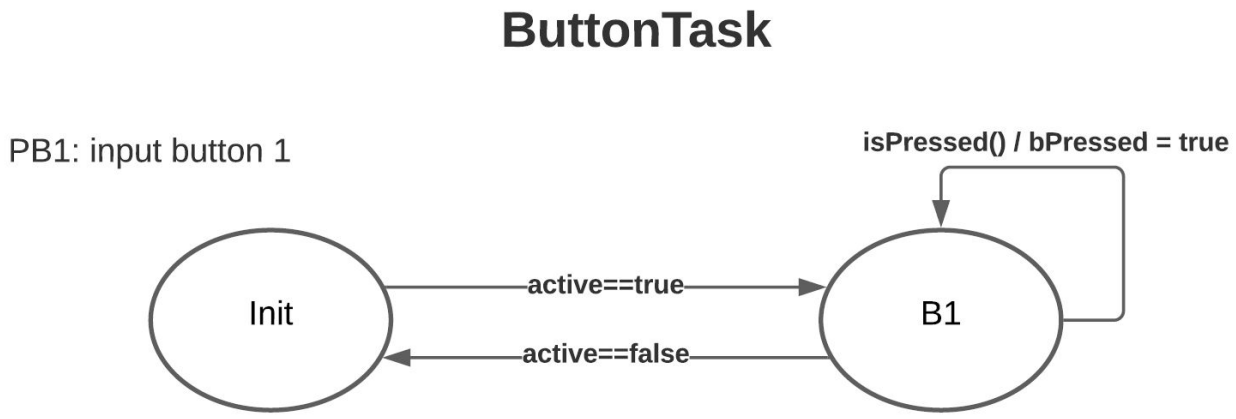
- **IlluminateLedTask:** Semplice Task che si occupa di accendere o spegnere un led nelle varie fasi dell'esperimento.



- **BlinkTask:** Altro semplice Task che si occupa di far lampeggiare un Led quando richiesto dalle specifiche, con la possibilità di utilizzarlo come un semplice IlluminateLed.



- **ButtonTask:** Un semplice Task che si occupa di controllare lo stato del bottone durante la fase di esperimento indipendentemente dalla frequenza utilizzata per quest'ultimo.



Comportamento Sincrono di Scheduler e Task

Un'altra specifica richiesta era quella di modellare il sistema tramite un modello di comportamento basato su macchine a stati finiti sincrone. Per questi motivi abbiamo implementato uno Scheduler che ogni 20ms richiama i vari Task. Quest'ultimi sono inizializzati con un periodo base che indica ogni quanto debbano essere eseguiti, e con una flag che indica se sono attivi o disattivi. Nel caso in cui un Task non sia attivo o non è il suo momento di eseguire, lo Scheduler lo ignora e passa al seguente.

Client su PC

Come client abbiamo sviluppato un semplice programma basato su pattern architetturale Model-View-Controller, in più ci siamo anche serviti della libreria JavaFX per quanto riguarda gli aspetti di View.

Interfacciamento al Client

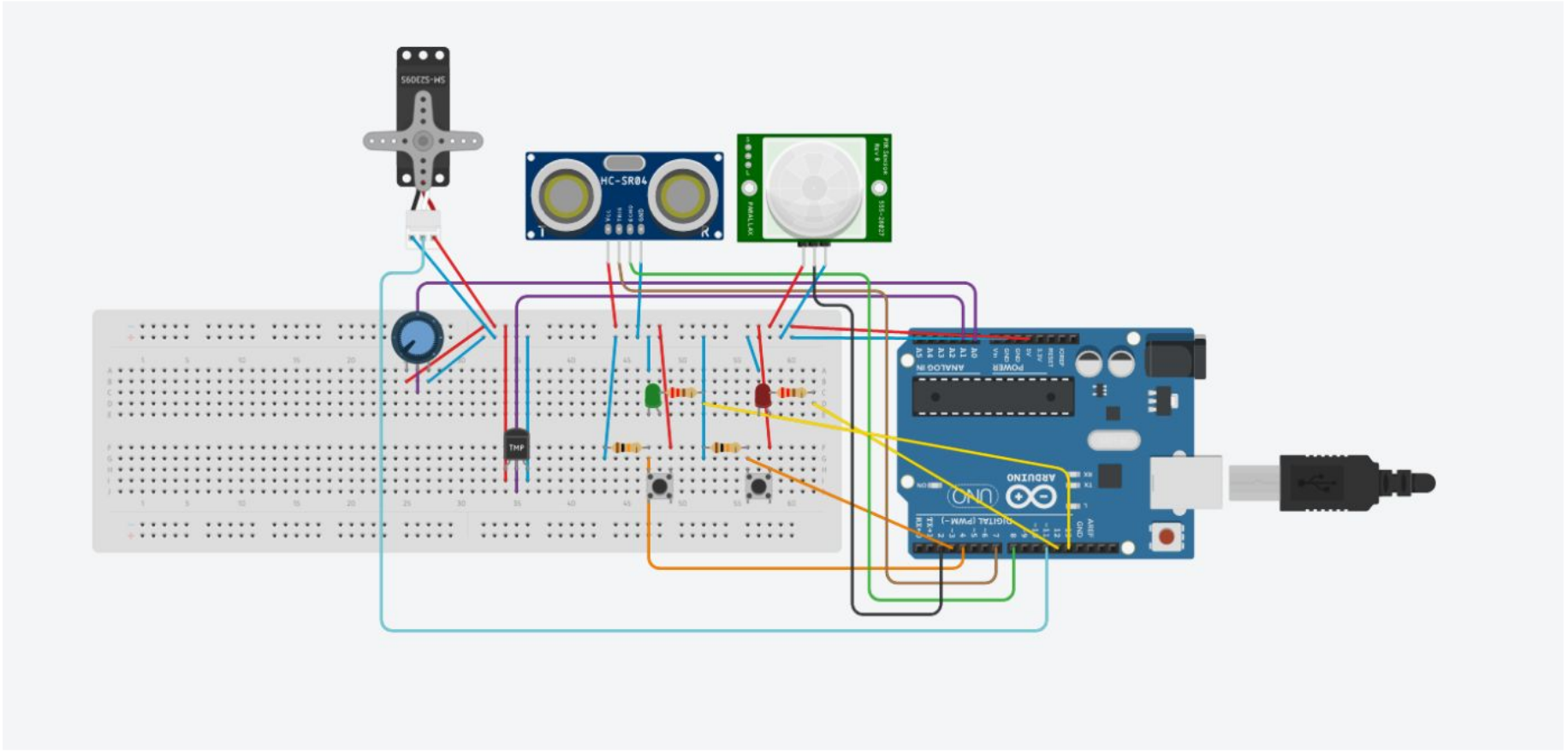
Una volta sviluppato il comportamento del sistema lato microcontrollore abbiamo iniziato a progettare un Client in grado di ricevere ed inviare dati sulla seriale di arduino, la ricezione di un messaggio su PC è possibile grazie ad un semplice Thread che, tramite apposita classe di interfacciamento (**CommunicationInterface**), prende tutti i messaggi inviati sulla seriale e li invia ad un Controllore. L'invio di un messaggio ad arduino invece avviene semplicemente grazie all'invocazione di un metodo dell'oggetto di comunicazione il quale si occuperà di scrivere sulla seriale.

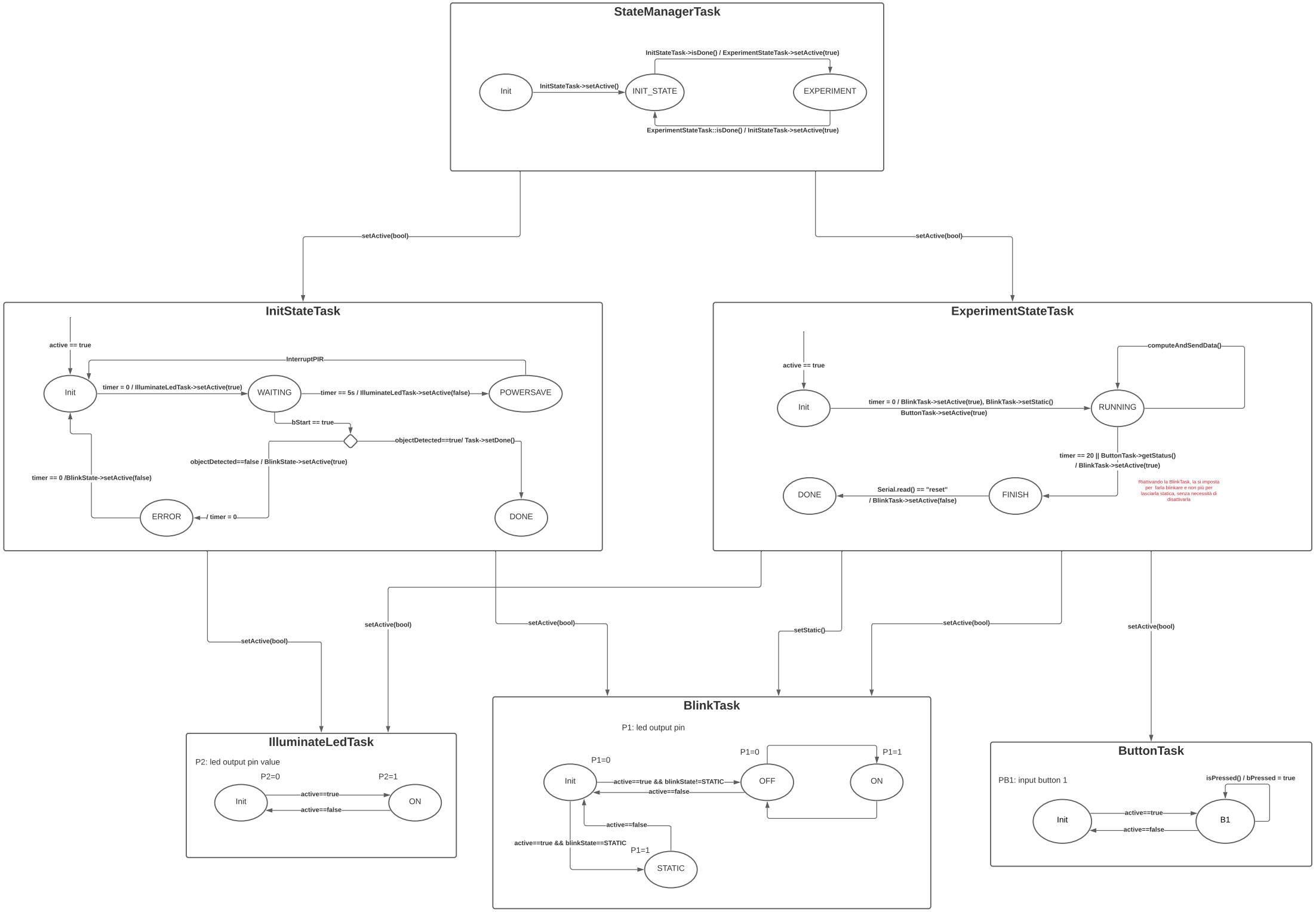
Misurazioni tramite Sonar

La parte fondamentale dell'esperimento è quella di ricevere dal sonar dei dati sulla posizione di un oggetto in movimento a frequenze di campionamento comprese tra gli 1 e i 50hz (scelte arbitrariamente dall'utente attraverso un potenziometro). Per fare ciò abbiamo sviluppato una classe (**PhysicsComputation**) che si occupa di ricevere i dati dal sonar e calcolare velocità e accelerazione. Ciò che abbiamo notato in seguito a numerose valutazioni empiriche è che l'HC-SR04 presenta degli errori nella valutazione della posizione di un oggetto, variabili nell'ordine dei 5mm. Questa caratteristica del sensore porta inevitabilmente a dei risultati imprecisi nel caso in cui si decida di campionare i dati con una frequenza molto alta.

Specificazioni sulla fase di esperimento

I valori di velocità ed accelerazione vengono computati in modo da ottenere sempre dei risultati positivi, sia che l'oggetto si allontani dal sonar, sia che si avvicini. Abbiamo anche deciso di gestire l'eventualità in cui il corpo venga rimosso da davanti al sonar in fase di esperimento impostando la velocità a zero m/s quando questo si trova a più di un metro.


SM-S2309S
HC-SR04
TMP
ARDUINO
UNO


StateManagerTask
InitStateTask->isDone() / ExperimentStateTask->setActive(true)
Init
InitStateTask->setActive()
INIT_STATE
EXPERIMENT
ExperimentStateTask::isDone() / InitStateTask->setActive(true)
setActive(bool)
setActive(bool)
InitStateTask
active == true
InterruptPIR
Init
timer = 0 / IlluminateLedTask->setActive(true)
WAITING
timer == 5s / IlluminateLedTask->setActive(false)
POWERSAVE
bStart == true
objectDetected==true/ Task->setDone()
objectDetected==false / BlinkState->setActive(true)
timer == 0 /BlinkState->setActive(false)
ERROR
/ timer = 0
DONE
ExperimentStateTask
active == true
computeAndSendData()
Init
timer = 0 / BlinkTask->setActive(true), BlinkTask->setStatic()
ButtonTask->setActive(true)
RUNNING
timer == 20 || ButtonTask->getStatus()
/ BlinkTask->setActive(true)
Riattivando la BlinkTask, la si imposta per farla blinkare e non più per lasciarla statica, senza necessità di disattivarla
DONE
Serial.read() == "reset"
/ BlinkTask->setActive(false)
FINISH
setActive(bool)
setActive(bool)
setActive(bool)
setStatic()
setActive(bool)
setActive(bool)
IlluminateLedTask
P2: led output pin value
P2=0
P2=1
Init
active==true
active==false
ON
BlinkTask
P1: led output pin
P1=0
P1=0
P1=1
Init
active==true && blinkState!=STATIC
active==false
OFF
ON
active==false
active==true && blinkState==STATIC
P1=1
STATIC
ButtonTask
PB1: input button 1
isPressed() / bPressed = true
Init
active==true
active==false
B1