Smart Experiment

Relazione secondo Assignment SEIOT

Luca Fabri 0000892878
Luigi Olivieri 0000880401
6 dicembre 2020

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Indice

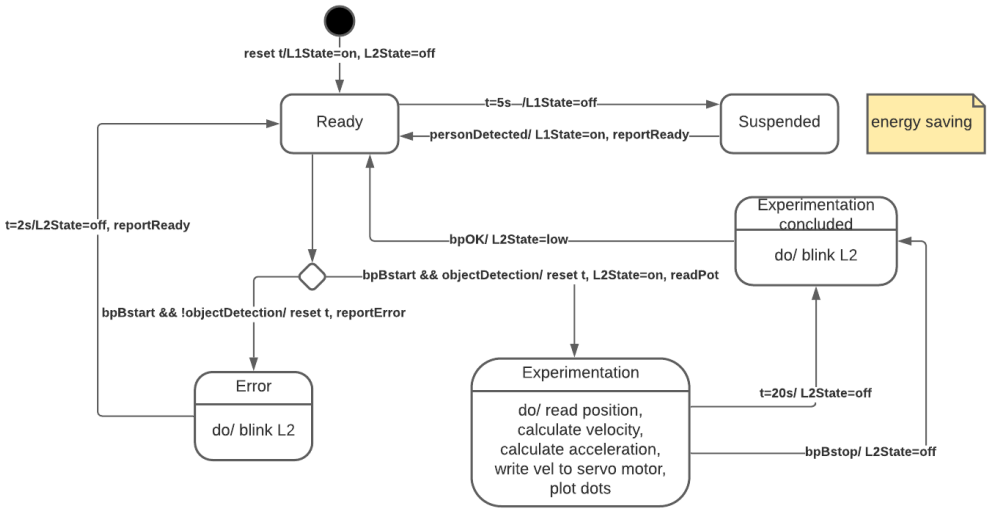
Capitolo 1

Arduino

1.1 Analisi e modellazione

Per utilizzare un approccio a task basato su macchine a stati finite sincrone, abbiamo cercato di scorporare le funzionalità che deve gestire il sistema. Partendo da un diagramma a stati senza task (immagine successiva: 1.1), abbiamo individuato: funzionalità di blink del led L2 negli stati di errore e sperimentazione conclusa, calcoli di cinematica, funzionalità di interfacciamento all'applicazione Viewer, accensione e spegnimento dei led al passaggio tra due stati, gestione del risparmio energetico e degli eventi per le transazioni tra gli stati.

Quindi, al fine di costruire più macchine a stati finiti (una per ogni task), gli stati dell'esperimento (Ready, Suspended, Error, Experimentation, Experimentation Concluded) verranno trattati come gli "Step" dell'esperimento, che verranno modificati dai vari task.



Pertanto, partendo dalle funzionalità descritte in precedenza, abbiamo modellato 5 task:

- **ChangeStepTask** il cui compito è quello di eseguire le guardie per i vari Step dell'esperimento (es. pressione pulsante, eventi temporali);
- **ActionTask** che si occupa dell'accensione/spegnimento dei led al passaggio tra due Step e del risparmio energetico;
- **BlinkLedTask** per gli Step di Error ed Experimentation Concluded;
- **KinematicsTask** il cui obiettivo è leggere la distanza dal sonar e calcolarne velocità ed accelerazione;
- **ViewerCommunicatorTask** il cui compito è quello di gestire la seriale e quindi la comunicazione con l'applicazione, tramite letture e scritture.

Questi task, necessiteranno di comunicare tra di loro, perciò la classe `ExperimentationStep` consentirà loro di modificare od ottenere lo Step corrente dell'esperimento attraverso l'enumerazione `Step`. Inoltre, la classe `KinematicsData` permette al KinematicsTask di settare i dati calcolati che verranno letti e inviati a Viewer dal ViewerCommunicatorTask.

Per quanto riguarda lo Scheduler il suo periodo di esecuzione è 50ms (MCD del periodo di tutti i task).

1.2 Sviluppo

Sotto sono riportati i diagrammi FSM visti nel dettaglio per ogni Task.

1.2.1 Change Step Task

Questo task si occupa di eseguire le guardie per il passaggio tra Step ad eccezione della pressione di buttonOK, Button dell'applicazione, che necessita di letture dalla Seriale, pertanto compito di ViewerCommunicatorTask.

La transizione dallo stato **ES4** a **ES0** verrà luogo quando ViewerCommunicatorTask modificherà lo Step a READY nell'oggetto Experimentation-Step. In tutti gli altri casi, quando una guardia è verificata, verranno eseguite delle set dello Step corrente.

Il periodo del task in alcuni casi viene re-inizializzato al tempo di timeout, poichè quest'ultimo varia di Step in Step. In particolare in **ES0**: SLEEP_TIME, in **ES2**: ERROR_TIME, in **ES3**: MAX_TIME.

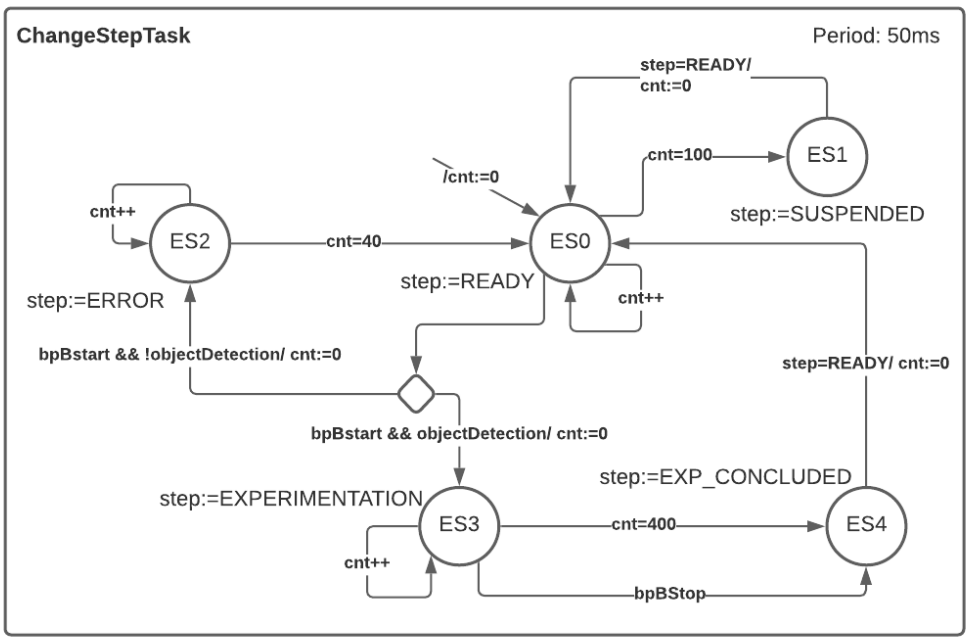


Figura 1.1: ChangeStepTask

1.2.2 Action Task

Come precedentemente descritto, questo task si occupa dell'esecuzione delle azioni al passaggio da uno Step ad un altro. In **A0, A2, A3, A4** corrispondono all'accensione e/o spegnimento dei led, mentre in **A1** corrispondono all'attivazione dello sleep mode, che in particolare è stato gestito con `SLEEP_MODE_PWR_DOWN` che è il massimo livello di risparmio energetico. In questa modalità le interruzioni sono abilitate quindi il Pir al rilevamento di un movimento risveglierà il sistema.

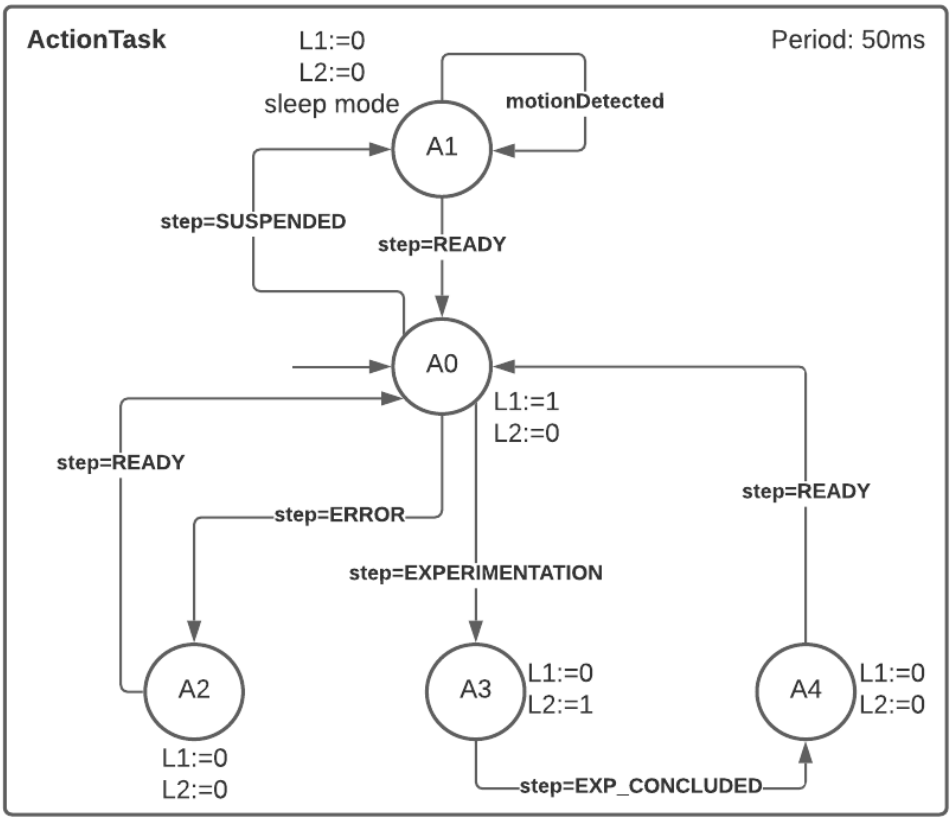


Figura 1.2: ActionTask

1.2.3 Blink Led Task

Il **BlinkLedTask** si occupa di far lampeggiare il il led assegnato (*nel nostro caso il led di colore rosso*) nel momento opportuno.

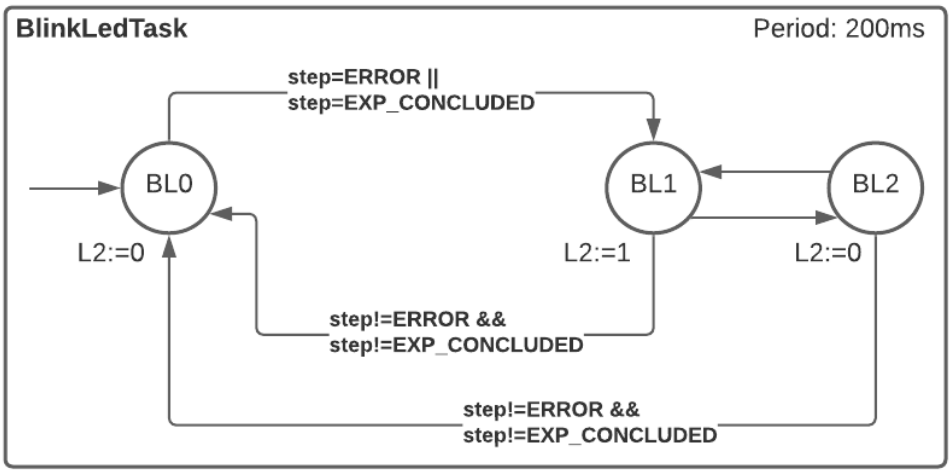


Figura 1.3: BlinkLedTask

- **BL0**: Durante questo stato il led non lampeggia. Quando il sistema va in stato di errore o la fase di sperimentazione è conclusa si passa allo stato **BL1**.
- **BL1**: In questo stato il led viene acceso per poi passare a **BL2** a meno che stato del sistema non cambi.
- **BL2**: In questo stato il led viene spento per poi passare a **BL2** a meno che stato del sistema non cambi.

Gli stati **BL1** e **BL2** si alternano creando l'effetto del lampeggiamento fino a quando il sistema è in stato di errore o di sperimentazione conclusa, altrimenti il Task torna allo stato **BL0** e il lampeggiamento termina.

1.2.4 Kinematic Task

Il **KinematicTask** si occupa di ricavare i valori di **distanza**, **velocità** e **accelerazione** dell'oggetto in movimento durante l'esperimento dai dati empirici del sistema.

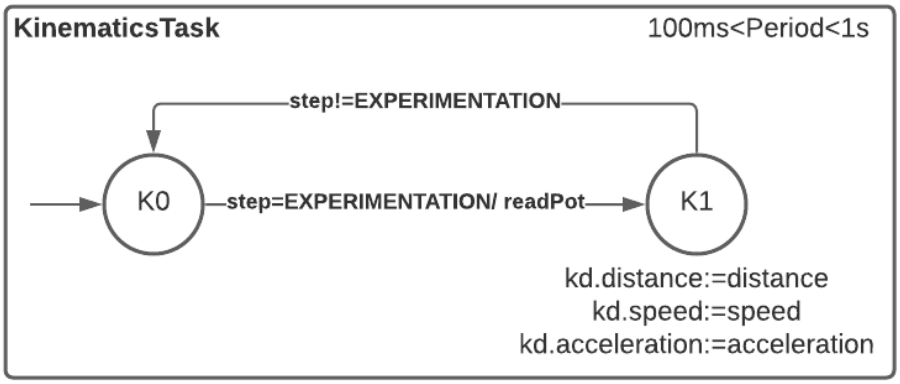


Figura 1.4: KinematicsTask

- **K0**: In questo stato il sistema non sta svolgendo esperimenti. Quando il sistema inizia la sperimentazione si passa allo stato **K1** inizializzando le utilità per i calcoli.

- **K1**: Al passaggio a questo stato vengono calcolati velocità del suono per il Sonar attraverso il sensore di temperatura DHT, frequenza di campionamento regolabile dal potenziometro (10Hz - 1Hz, sempre multiplo della frequenza dello Scheduler) e velocità massima con cui l'oggetto può muoversi nello spazio antistante il Sonar (MAX_VEL, variabile in base alla frequenza).

  Una volta che siamo in questo stato, viene presa tramite il Sonar la distanza dell'oggetto e vengono calcolate **distanza**, **velocità** e **accelerazione**.

  Successivamente le misure vengono passate al **Viewer Communicator Task** (1.5) per essere inviate sulla **Seriale** e viene mostrato il valore della velocità sul **Servo Motore**. Quando il sistema termina la sperimentazione si passa allo stato **K0**. .

1.2.5 Viewer Communicator Task

Il **ViewerCommunicatorTask** si occupa dello scambio di dati tra PC e Arduino attraverso la seriale.

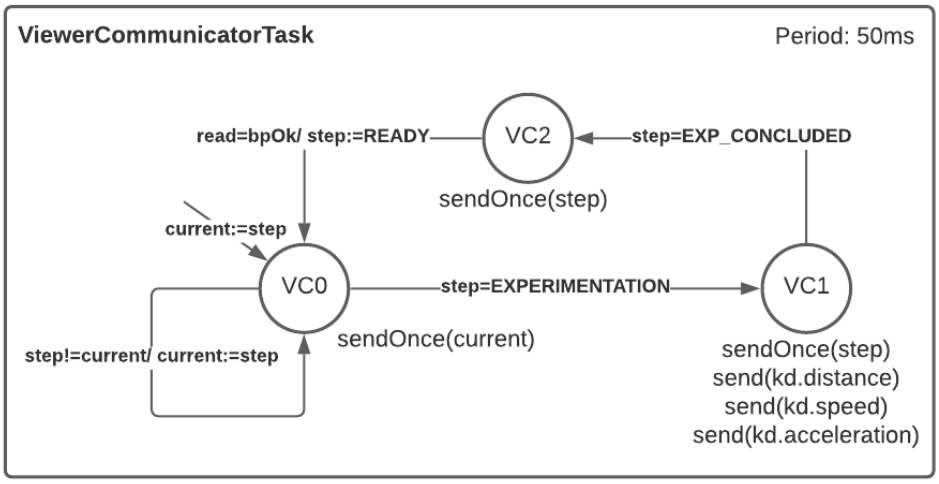


Figura 1.5: ViewerCommunicatorTask

- **VC0**: Durante questo stato viene mandato un messaggio contenente lo stato in cui si trova al momento il sistema sulla Seriale. Il messaggio viene inviato solo una volta e se viene rilevato un cambio nello stato del sistema. Quado il sistema entra in fase di sperimentazione si passa allo stato **VC1**.

- **VC1**: Durante questo stato, oltre a notificare che si è entrati nello stato di sperimentazione, viene mandato un messaggio contente i dati catturati dal sistema da mostrare graficamente sull'applicazione Viewer con ritmo dettato dal task **Kinematic Task** (1.4). Quado il sistema esce dalla fase di sperimentazione si passa allo stato **VC2**.

- **VC2**: Durante questo stato, oltre a notificare che si è entrati nello stato di sperimentazione, si controlla l'arrivo del messaggio di ok da parte dell'applicazione Viewer una volta che l'esperimento è concluso. Quando il messaggio di ok viene rilevato si torna allo stato **VC0**.

Per modellare la figura dei task si è scelto di creare un interfaccia **Task** che accorpasse i metodi comuni essenziali e poi una sua specializzazione per ogni Task da implementare (1.6).

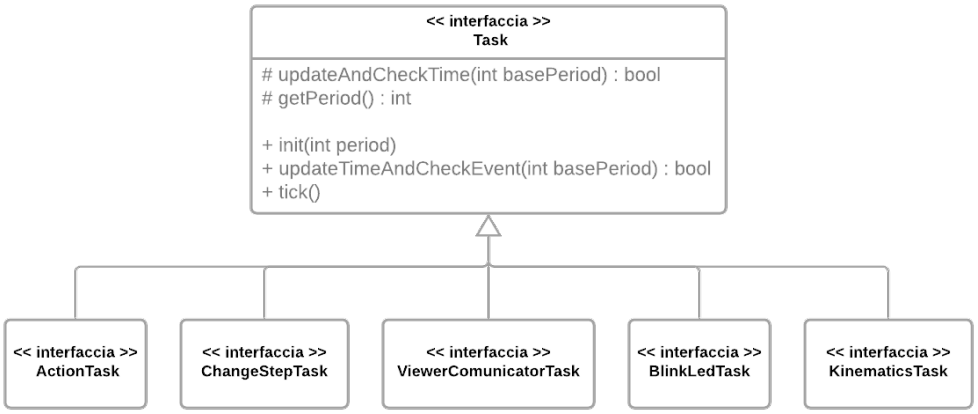
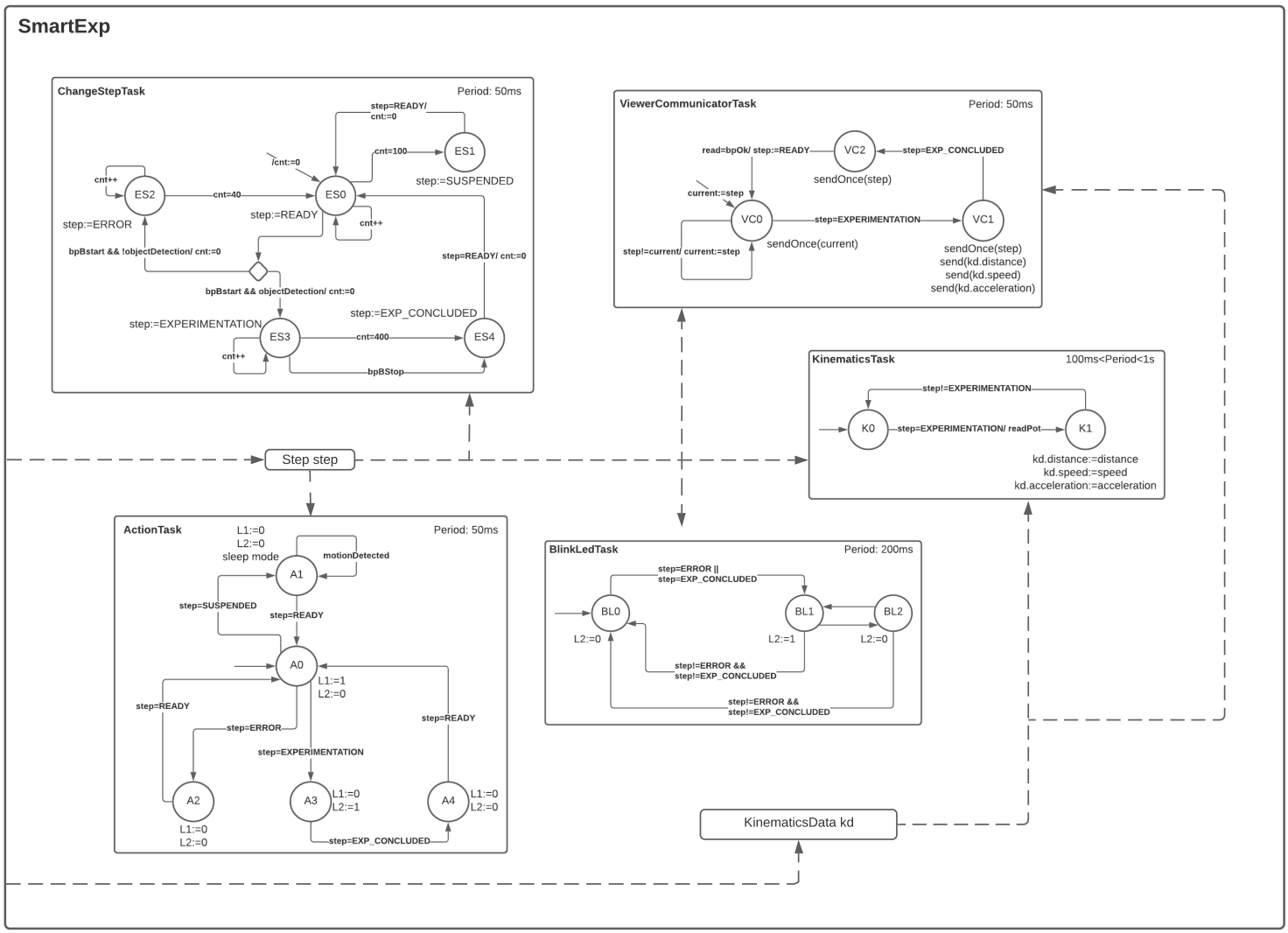


Figura 1.6: Lo schema UML che rappresenta la gerarchia dei Task

Inoltre per sfruttare i vantaggi della programmazione a oggetti in C++ sono state progettate delle classi per rappresentare i componenti hardware del sistema.


SmartExp
ChangeStepTask
Period: 50ms
step=READY/
cnt:=0
cnt=100
ES1
step:=SUSPENDED
/cnt:=0
cnt++
ES2
cnt=40
ES0
step:=ERROR
step:=READY
cnt++
bpBstart && !objectDetection/ cnt:=0
step=READY/ cnt:=0
bpBstart && objectDetection/ cnt:=0
step:=EXP_CONCLUDED
step:=EXPERIMENTATION
ES3
cnt=400
ES4
cnt++
bpBStop
ViewerCommunicatorTask
Period: 50ms
read=bpOk/ step:=READY
VC2
step=EXP_CONCLUDED
sendOnce(step)
current:=step
VC0
step=EXPERIMENTATION
VC1
step!=current/ current:=step
sendOnce(current)
sendOnce(step)
send(kd.distance)
send(kd.speed)
send(kd.acceleration)
KinematicsTask
100ms<Period<1s
step!=EXPERIMENTATION
K0
step=EXPERIMENTATION/ readPot
K1
kd.distance:=distance
kd.speed:=speed
kd.acceleration:=acceleration
Step step
ActionTask
Period: 50ms
L1:=0
L2:=0
sleep mode
motionDetected
A1
step=SUSPENDED
step=READY
A0
L1:=1
L2:=0
step=READY
step=ERROR
step=READY
step=EXPERIMENTATION
A2
L1:=0
L2:=0
A3
L1:=0
L2:=1
A4
L1:=0
L2:=0
step=EXP_CONCLUDED
BlinkLedTask
Period: 200ms
step=ERROR ||
step=EXP_CONCLUDED
BL0
L2:=0
BL1
L2:=1
BL2
L2:=0
step!=ERROR &&
step!=EXP_CONCLUDED
step!=ERROR &&
step!=EXP_CONCLUDED
KinematicsData kd

1.3 Circuito Scheda Arduino

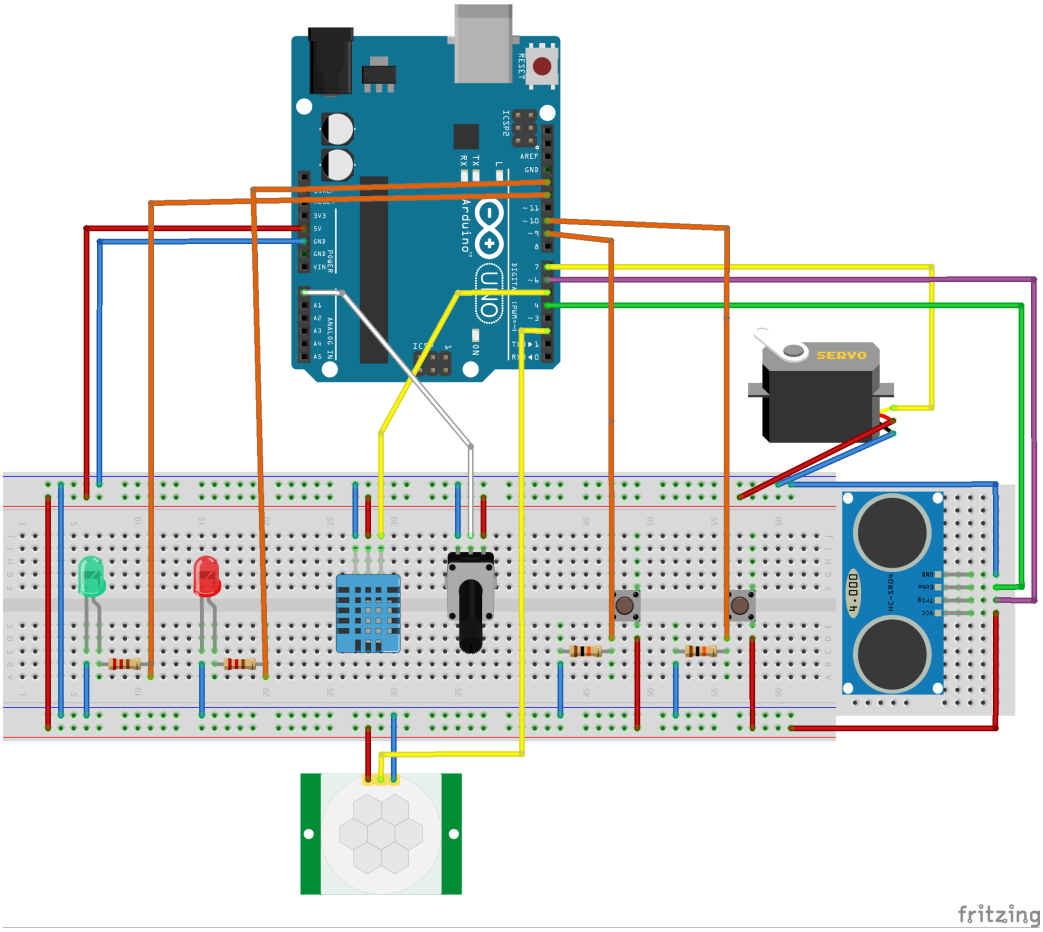


Figura 1.7: Schema del circuito realizzato con Fritizing

Capitolo 2

Viewer

Per lo sviluppo dell'applicazione **Viewer** è stato scelto il linguaggio ad oggetti **Java** con il supporto delle librerie esterne **JavaFx** e **JavaSimpleSerialConnector** (*nella sua versione 2.9.2*).

2.1 Serial Communication

Per gestire lo spostamento di dati tra PC e scheda Arduino viene creata la classe **SerialController** che implementa l'interfaccia **SerialEventListner** fornita da *jssc*. Nel suo metodo *serialEvent* i caratteri mandati dalla scheda Arduino vengono salvati come *String* fino al rilevamento del carattere di fine messaggio. Una volta che il messaggio è completo viene processato attraverso gli altri metodi privati della classe per aggiornare la parte grafica.

2.2 Data Visualization

Per ogni grandezza da rappresentare su Viewer (*Distanza*, *Velocità* e *Accelerazione*) viene creato un grafico attraverso i metodi statici di **GraphFactory**. I **Graph** creati utilizzano il metodo *updatePlot* per aggiornare i dati rappresentati sul corrispondente *LineChart* (*oggetto di javaFx*) e il metodo *reset* per cancellare tutti i punti segnati per ripulire il piano del grafico. Per utilizzare l'implementazione **Graph** la classe **SerialController** utilizza il design pattern **Strategy**.