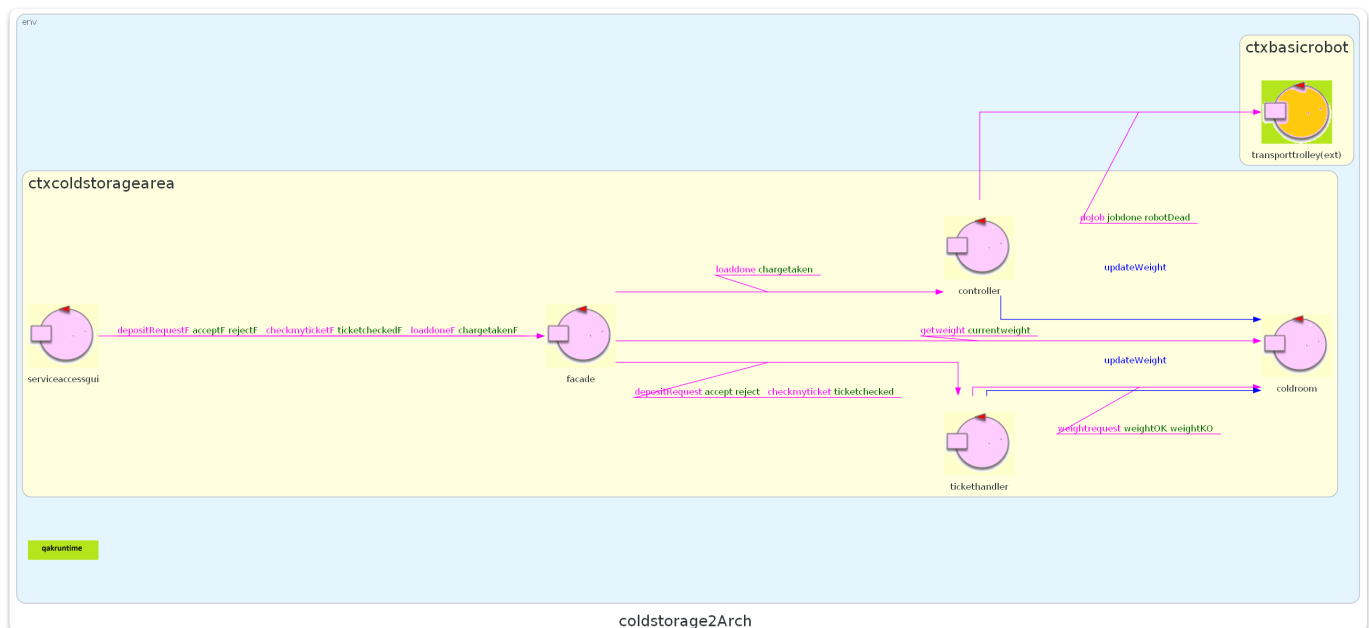


## Goal dello Sprint 2

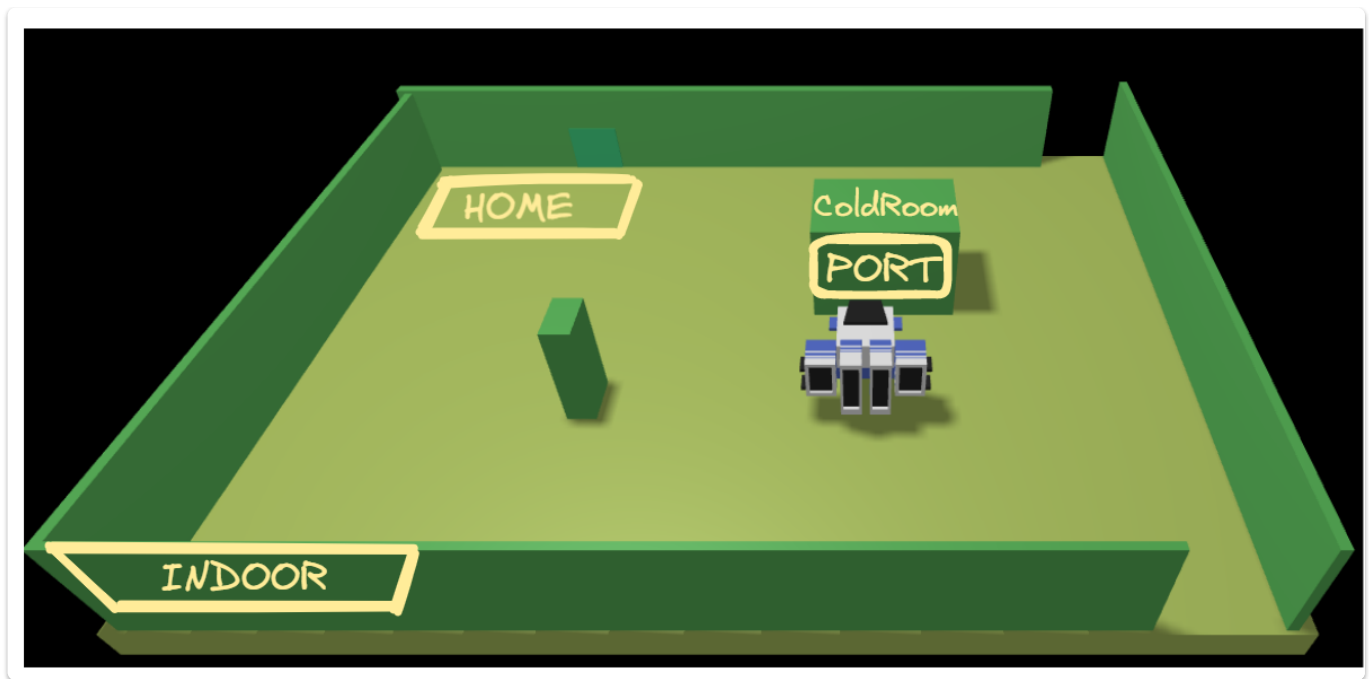
Implementazione di Led e Sonar su RaspberryPi.

### Descrizione >

Nel secondo sprint verranno implementati il sistema di led e sonar con la logica ad essi associata. I due componenti si troveranno su un dispositivo esterno e dovranno interagire con il sistema remotamente.



## Requisiti



## [Requisiti](#)

### Alarm requirements

The system includes a Sonar and a Led connected to a RaspberryPi.

The Sonar is used as an 'alarm device': when it measures a distance less than a prefixed value **DLIMIT**, the transport trolley must be stopped; it will be resumed when Sonar detects again a distance higher than **DLIMIT**.

The Led is used as a *warning devices*, according to the following scheme:

- the Led is **off** when the transport trolley is at HOME
- the Led **blinks** while the transport trolley is moving
- the Led is **on** when transport trolley is stopped

## Analisi dei Requisiti

### [requisiti sprint 0](#)

### Domande al Committente

Il committente fornisce software relativo al Led e al Sonar? NO

Il LED può/deve essere connesso allo stesso RaspberryPi del sonar? SI

Il valore **DLIMIT** deve essere cablato nel sistema o è bene sia definibile in modo configurabile dall'utente finale?

## Analisi del Problema

### Sistema distribuito - Come implementiamo la comunicazione?

```
Context ctxalarm ip [host="localhost" port=8300]
Context ctxcoldstoragearea ip [host="127.0.0.1" port=8040]

ExternalQActor controller context ctxcoldstoragearea
```

Rispetto al resto del sistema Sonar e Led si trovano da requisiti su un RaspberryPi esterno. I due nodi di elaborazione devono potersi scambiare informazione via rete. Incapsuliamo in due attori i componenti che si occuperanno di gestire Led e Sonar e sfruttiamo questi per scambiare i messaggi.

## Segnali

```
#segnali per il led
Dispatch arrivedhome : arrivedhome(NO_PARAM)
Dispatch moving : moving(NO_PARAM)
Dispatch stopped : stopped(NO_PARAM)

#segnali per il sonar
Dispatch stop : stop(NO_PARAM)
Dispatch continue : continue(NO_PARAM)
```

### Perchè Dispatch? >

In entrambi i casi i segnali sono destinati ad un attore specifico conosciuto. Nel caso del sonar, anche trattandosi di uno stop d'emergenza non è stato usato Req/Resp poiché in un caso reale, se anche malauguratamente il segnale di stop dovesse non arrivare correttamente sarebbe estremamente facile mandarne un secondo immediatamente

## Business Logic

**Led:** La logica di accensione e spegnimento del led verrà gestita dall'attore associato in base allo stato comunicato dal controller. Il componente di basso livello deve solo essere in grado di accendere/spegnere il led.

**Sonar:** Sfruttiamo l'attore associato al sonar per elaborare i dati emessi dal sonar e decidere se lanciare al controller il segnale di allarme.

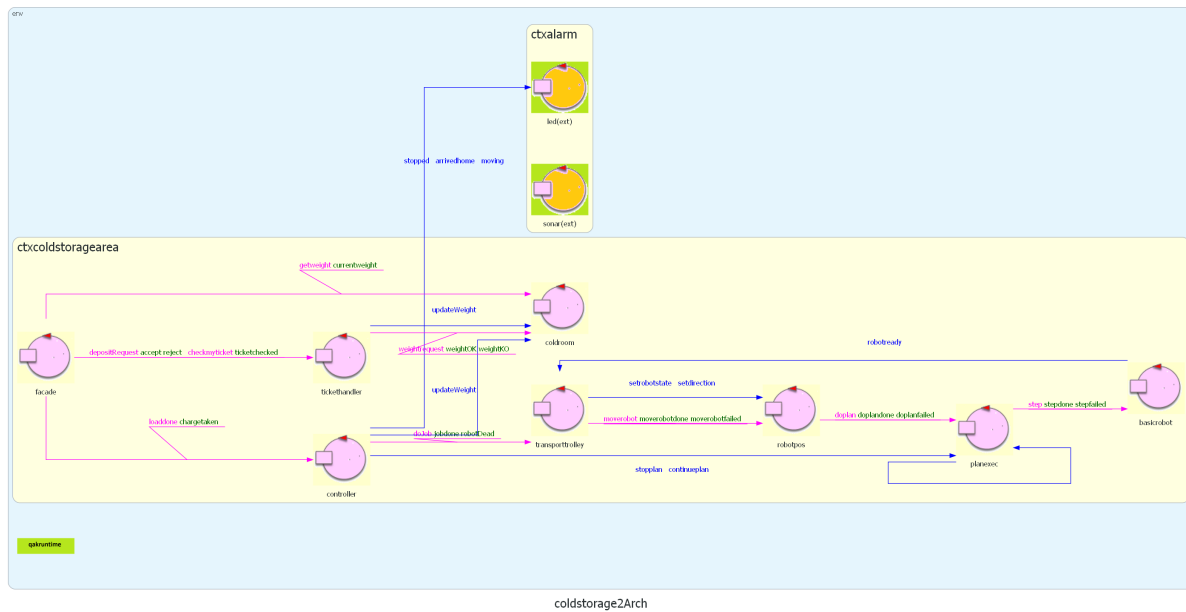
- PRO: alleggerisco il carico di dati nella rete, semplifico il controller, principio di singola responsabilità.

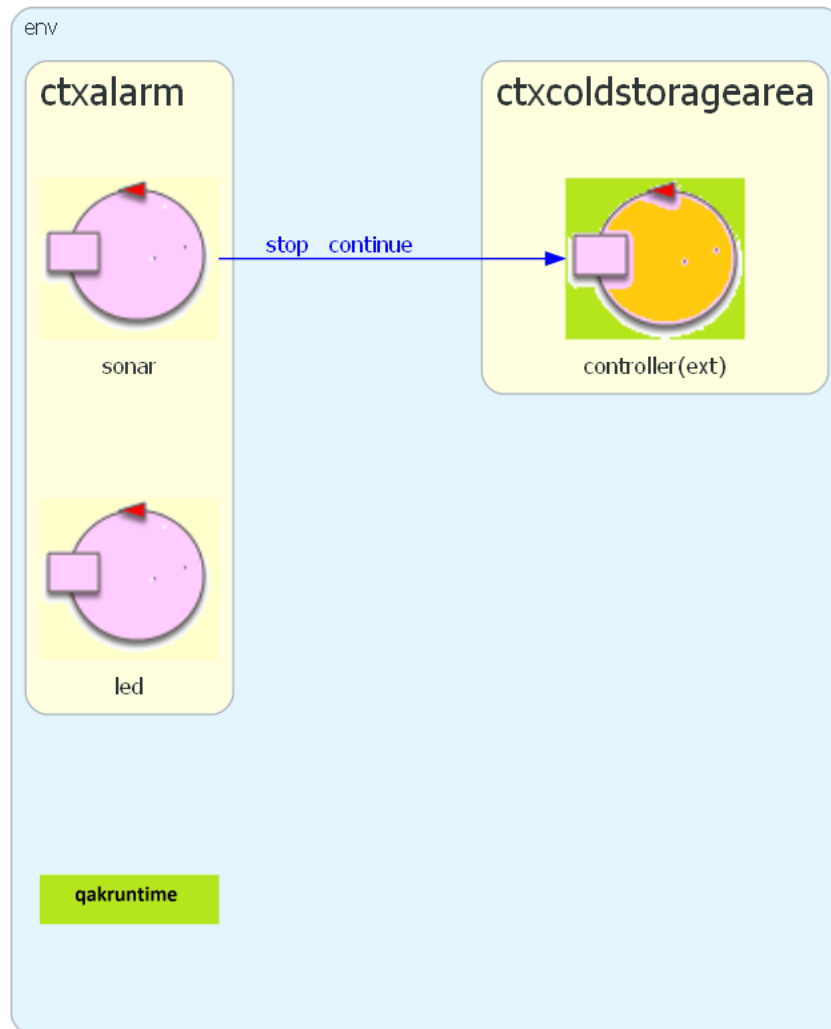
- CONTRO: difficoltà maggiore nel caso di DLIMIT variabile, il file di config sarebbe diverso dal precedente e posto sul raspberry.

### Problema del messaggio duplicato

Dobbiamo gestire il caso in cui arrivano più volte gli stessi messaggi (ottengo 2 stop di fila ad esempio perdendo il "continue" intermedio)

### Architettura logica dopo l'analisi del problema





alarmArch

## Test Plan

Stato del led che si aggiorna correttamente.

Controller che modifica correttamente lo stato dopo aver ricevuto un segnale dal sonar.

Il testing di un sonar riguarda due aspetti distinti:

1. il test sul corretto funzionamento del dispositivo in quanto tale. Supponendo di porre di fronte al Sonar un ostacolo a distanza  $D$ , il Sonar deve emettere dati di valore  $D \pm \epsilon$ .
2. il test sul corretto funzionamento del componente software responsabile della trasformazione del dispositivo in un produttore di dati consumabili da un altro componente.

Test implementato:

Supponendo di porre di fronte al Sonar un ostacolo a distanza  $D$ , il BasicRobot deve fermarsi e riparte solo dopo che l'ostacolo è rimosso.

```
@Test
public void mainUseCaseTest(){
    //connect to port
    try{
        Socket client= new Socket("localhost", 8040);
        BufferedWriter out = new BufferedWriter(new
OutputStreamWriter(client.getOutputStream()));
        BufferedReader in = new BufferedReader(new
InputStreamReader(client.getInputStream()));

        out.write("msg(depositRequestF,request,test2,facade,depositRequestF(1),1)\n");
        out.flush();
        //wait for response
        String response= in.readLine();
        response = response.split(",")[4];
        String ticket = response.replace("acceptF(", "");
        ticket = ticket.replace(")", "");

        out.write("msg(checkmyticketF,request,test2,facade,checkmyticketF(" +
ticket + "),1)\n");
        out.flush();
        String responseC= in.readLine();

        out.write("msg(loaddoneF,request,test2,facade,loaddoneF(1),1)\n");
        out.flush();
        String responseL= in.readLine();

        System.out.println("sleep 2 seconds");
        TimeUnit.SECONDS.sleep(4);

        out.write("msg(stop,dispatch,test2,controller,stop(),1)\n");
```

```

        System.out.println("sleep 1 seconds");
        TimeUnit.SECONDS.sleep(1);

    out.write("msg(getrobotstate,request,test2,robotpos,getrobotstate(ARG),1)\n");
    out.flush();
    String responsePos1= in.readLine();
    responsePos1 = responsePos1.split(",")[4];
    System.out.println("pos1: "+responsePos1); //robotstate(pos(0,4),DOWN)

    System.out.println("sleep 1 seconds for pos");
    TimeUnit.SECONDS.sleep(1);

    out.write("msg(getrobotstate,request,test2,robotpos,getrobotstate(ARG),1)\n");
    out.flush();
    String responsePos2= in.readLine();
    responsePos2 = responsePos2.split(",")[4];
    System.out.println("pos2: "+responsePos2); //robotstate(pos(0,4),DOWN)

    assertTrue(responsePos1.equalsIgnoreCase(responsePos2));

    out.write("msg(continue,dispatch,test2,controller,continue(),1)\n");

    System.out.println("sleep 1 seconds for pos");
    TimeUnit.SECONDS.sleep(1);

    out.write("msg(getrobotstate,request,test2,robotpos,getrobotstate(ARG),1)\n");
    out.flush();
    String responsePos3= in.readLine();

    System.out.println("pos3: "+responsePos3);

    assertFalse(responsePos1.equalsIgnoreCase(responsePos3));

    }catch(Exception e){
        fail();
        System.out.println(e.getStackTrace());
    }
}

```

## Progettazione



## SonarActor

```
QActor sonar context ctxalarm{

    [#      var Distanza = 20.0; #]

    State s0 initial {
        println("alarm - sonar started") color green
    } Goto work

    State work{
        [#
            while(SonarService.getDistance() > Distanza){}
        #]

        forward controller -m stop : stop(1)
        println("alarm - sent stop") color green
    }Goto stopped

    State stopped {
        [#
            while(SonarService.getDistance() < Distanza){}
        #]
        forward controller -m continue : continue(1)
        println("alarm - sent continue") color green
    } Goto work
}
```

## SonarService

Legge i dati rilevato dal sonar da stdin

```
import java.io.BufferedReader
import java.io.InputStreamReader

object SonarService {
    var reader: BufferedReader? = null

    init {
        try {
            var p = Runtime.getRuntime().exec("python3 -u sonar.py")
        }
    }
}
```

```

        reader = BufferedReader( InputStreamReader(p.inputStream))
    } catch (e : Exception) {
        println(e.message)
    }
}

fun getDistance() : Double {
    var distance = reader!!.readLine().toDouble()
    println(distance)
    return distance
}
}

```

## Sonar

sonar in python: dopo l'avvio scrive la distanza calcolata su stdout 4 volte al secondo

```

import RPi.GPIO as GPIO
import time
import sys

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
TRIG = 17
ECHO = 27

GPIO.setup(TRIG,GPIO.OUT)
GPIO.setup(ECHO,GPIO.IN)

GPIO.output(TRIG, False)    #TRIG parte LOW
print ('Waiting a few seconds for the sensor to settle')
time.sleep(2)

while True:
    GPIO.output(TRIG, True)    #invia impulsoTRIG
    time.sleep(0.00001)
    GPIO.output(TRIG, False)

    #attendi che ECHO parta e memorizza tempo
    while GPIO.input(ECHO)==0:
        pulse_start = time.time()

    # register the last timestamp at which the receiver detects the signal.
    while GPIO.input(ECHO)==1:
        pulse_end = time.time()

```

```

pulse_duration = pulse_end - pulse_start
distance = pulse_duration * 17165    #distance = vt/2
distance = round(distance, 1)
print ( distance )
sys.stdout.flush()    #Importante!
time.sleep(0.25)

```

## LedActor

```

QActor led context ctxalarm{
    State s0 initial {

    }Goto athome

    State athome {
        [#
            try{
                val p = Runtime.getRuntime().exec("python3 ledOFF.py")
            }catch( e : Exception){
                println(e.message)
            }
        #]
        println("alarm - atHome -led off ") color yellow

    } Transition t1 whenMsg arrivedhome -> athome
                                whenMsg moving -> currmoving
                                whenMsg stopped -> arrested

    State currmoving {
        [#
            try{
                val p = Runtime.getRuntime().exec("python3 ledON.py")
            }catch( e : Exception){
                println(e.message)
            }
        #]

        println("alarm - moving - ledOn") color yellow

        [# Thread.sleep(1000);
            try{

```

```

        val p = Runtime.getRuntime().exec("python3 ledOFF.py")
    }catch( e : Exception){
        println(e.message)
    }
    #]

    println("alarm - moving - LedOff") color yellow
} Transition t2 whenTime 1000 -> currmoving
    whenMsg moving -> currmoving
    whenMsg arrivedhome -> athome
    whenMsg stopped -> arrested

State arrested {
    [#
        try{
            val p = Runtime.getRuntime().exec("python3 ledON.py")
        }catch( e : Exception){
            println(e.message)
        }
    #]
    println("alarm - arrested - led on") color yellow
} Transition t3 whenMsg stopped -> arrested
    whenMsg arrivedhome -> athome
    whenMsg moving -> currmoving
}

```

## Led

facciamo solo uno script che accende e uno script che spegne e ci pensa l'attore ad invocare lo script secondo bisogno per mostrare lo stato corrente, la logica di lampeggiamento è lasciata all'attore led da gestire e non allo script.

ledOn in python

```

import RPi.GPIO as GPIO

LED_PIN = 21

GPIO.setmode(GPIO.BCM)
GPIO.setup(LED_PIN,GPIO.OUT)
GPIO.setwarnings(False)

```

```
GPIO.output(LED_PIN, GPIO.HIGH)
```

ledOff in python

```
import RPi.GPIO as GPIO

LED_PIN = 21

GPIO.setmode(GPIO.BCM)
GPIO.setup(LED_PIN,GPIO.OUT)
GPIO.setwarnings(False)

GPIO.output(LED_PIN, GPIO.LOW)
```

## Controller

```
QActor controller context ctxcoldstoragearea {

    [# var PESO = 0 #]

    State s0 initial {
        printCurrentMessage
    } Goto work

    State work{ //inHome
        println("controller - work") color green
    } Transition t0 whenMsg stop -> stopped
                                   whenMsg continue -> work
                                   whenRequest loaddone -> startjob

    State stopped{
        println("controller - stopped") color green
        forward led -m stopped : stopped(1)
        forward planexec -m stopplan : stopplan(1)
    }Transition t0 whenMsg stop -> stopped
                                   whenMsg continue -> contineworking

    State contineworking{
        println("controller - continue") color green
        forward led -m arrivedhome : arrivedhome(1)
        forward planexec -m continueplan : continueplan(1)
    }
```

```

}Goto work

State startjob {
    forward led -m moving : moving(1)
    onMsg(loaddone : loaddone(PESO) ){
        [# PESO = payloadArg(0).toInt() #]
        println("controller - startjob dichiarato: $PESO") color
green
    }
    replyTo loaddone with chargetaken : chargetaken( NO_PARAM )
    request transporttrolley -m doJob : doJob($PESO)
} Transition endjob whenMsg stop -> stoppedwhileworking
                                whenReply robotDead ->
handlerobotdead

                                whenReply jobdone -> jobdone

State stoppedwhileworking {
    forward planexec -m stopplan : stopplan(1)
    println("stopped while working") color magenta
    forward led -m stopped : stopped(1)
}Transition t0 whenMsg continue -> waitingforreply

State waitingforreply{
    forward planexec -m continueplan : continueplan(1)
    println("continued") color green
    forward led -m moving : moving(1)
}Transition endjob whenMsg stop -> stoppedwhileworking
                                whenReply robotDead ->
handlerobotdead

                                whenReply jobdone -> jobdone

State jobdone{
    println("jobdone") color green
    forward coldroom -m updateWeight : updateWeight($PESO, $PESO)
    forward led -m arrivedhome : arrivedhome(1)
} Goto work

State handlerobotdead{
    println("robotdead") color red
    printCurrentMessage

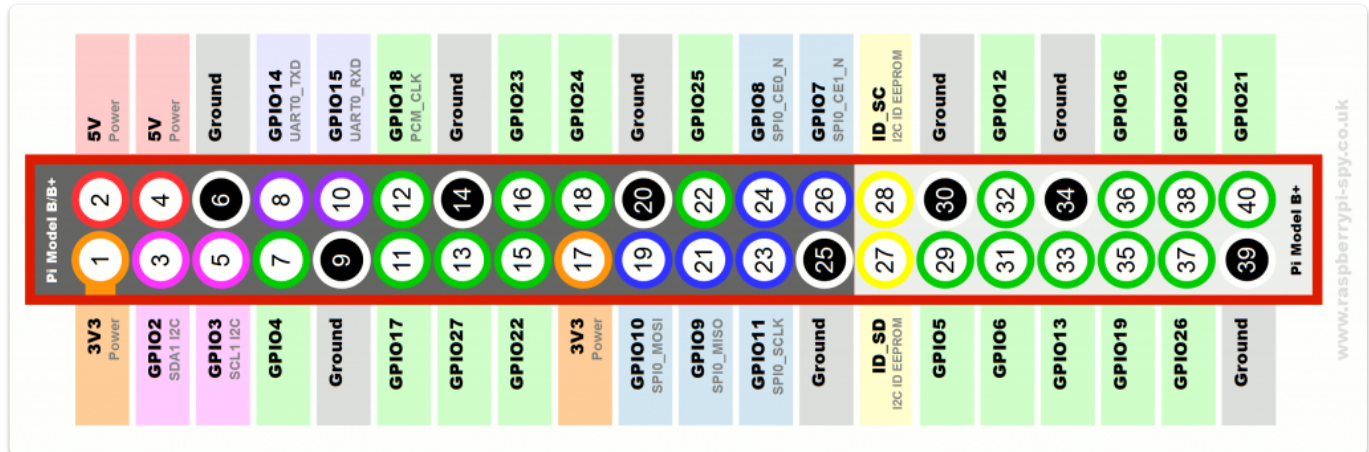
```

}

}

## Deployment

### Deployment on RaspberryPi 3B/3B+






### Led

- braccino corto: pin fisico 39 (GND)
- braccino lungo: pin fisico 40 (GPIO21)

### Sonar

- VCC : pin fisico 4 (+5v)
- GND : pin fisico 6 (GND)
- TRIG: pin fisico 11 (GPIO 17)
- ECHO: pin fisico 13 (GPIO 27)

<b>Lica Uccini</b>	<b>Luca Lombardi</b>	<b>Giacomo Romanini</b>
		
<a href="#">github: LisalU00</a>	<a href="#">github: Lombax99</a>	<a href="#">github: RedDuality</a>