Relazione Assignment 2

In questo progetto e' richiesto di realizzare una macchina del caffe utilizzando la scheda arduino e la programmazione di task asincroni.

File con specifiche

La prima cosa che e' stata fatta e' un diagramma con il quale si sono chiarite le idee su come i vari task saranno stati divisi e progettati, l'intenzione e' stata quella di inserire nel loop solo ed esclusivamente la funzione schedule della Classe Scheduler, alla quale si sono aggiunti mano a mano i vari task.

```
/* Scheduler.cpp*/

void Scheduler::init(int basePeriod){
  this->basePeriod = basePeriod;
  timerFlag = false;
  long period = 1000l*basePeriod;
  Timer1.initialize(period);
  Timer1.attachInterrupt(timerHandler);
  nTasks = 0;
}

bool Scheduler::addTask(Task* task){
  if (nTasks < MAX_TASKS-1){
    taskList[nTasks] = task;
    nTasks++;
    return true;
  } else {
    return false;
  }
}

void Scheduler::schedule(){
  while (!timerFlag){}
  timerFlag = false;

  for (int i = 0; i < nTasks; i++){
    if (taskList[i]->updateAndCheckTime(basePeriod)){
      taskList[i]->tick();
    }
  }
}
```

Machine

Per comodita' si e' deciso di creare una classe che chiudesse al proprio interno tutte i sensori e le funzionalita' della macchina, instanziandola solo una volta e' come se si avesse una macchina del caffe e si passasse il riferimento a questo oggetto ai vari task.

```
/* Machine.h */

#ifndef __MACHINE__
#define __MACHINE__

#include "Display.h"
#include "Product.h"
#include "ButtonImpl.h"
#include "ServoMotorImpl.h"
#include "Temp.h"
#include "Pir.h"


#define N_MAX_QUANTITY 50
#define T_OUT 5000L
#define T_MAKING 55

#define B_UP 2
#define B_DOWN 3
#define B_MAKE 4
#define PIR_PIN 5
#define TRIG_PIN 7
#define ECHO_PIN 8
#define SONAR_DISTANCE 40
#define SERVO_PIN 10
#define TEMP_PIN A0

#define PROD_NUM 3

enum stato{
  WELCOME,
  READY,
  SELECT,
  MAKING,
  WAITING_REMOVING,
  ASSISTANCE,
  SLEEP
};
```

```
class Machine {
  public:
    Machine();
    NewPing* sonar;
    Pir* pir;
    ServoMotor* servo ;
    Display* display_lcd ;
    Temp* temp;
    Product* coffee ;
    Product* tea ;
    Product* chocolate ;
    Button* bUp;
    Button* bDown ;
    Button* bMake ;
    stato state;
    int checkDone;
    bool lastMove;
};

#endif
```

Di conseguenza si e' realizzato lo schema del circuito nel seguente modo.

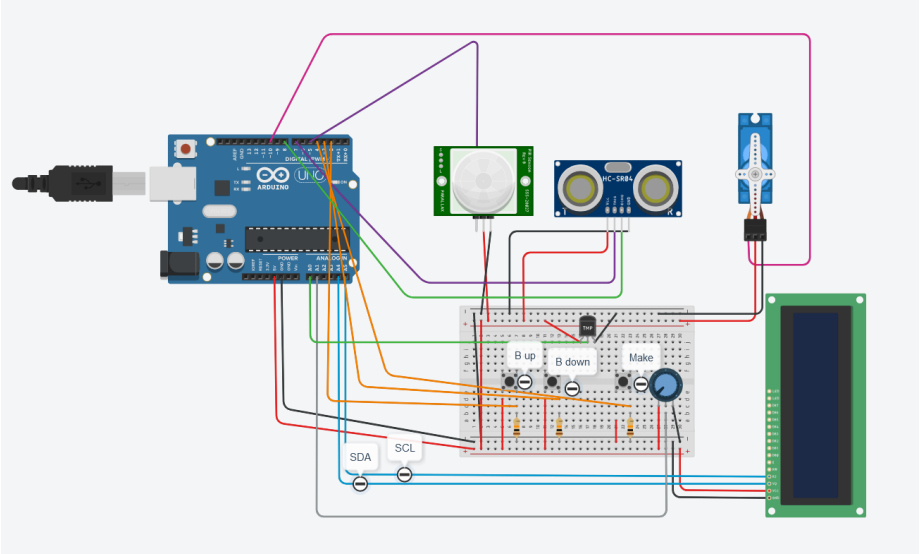

Figure 1: Circuit schema

Selection Task

Il task Selection racchiude in se buona parte della logica della macchinetta, dalla selezione dei prodotti alla loro produzione e quindi ai vari controlli dei sensori di prossimita' e presenza.

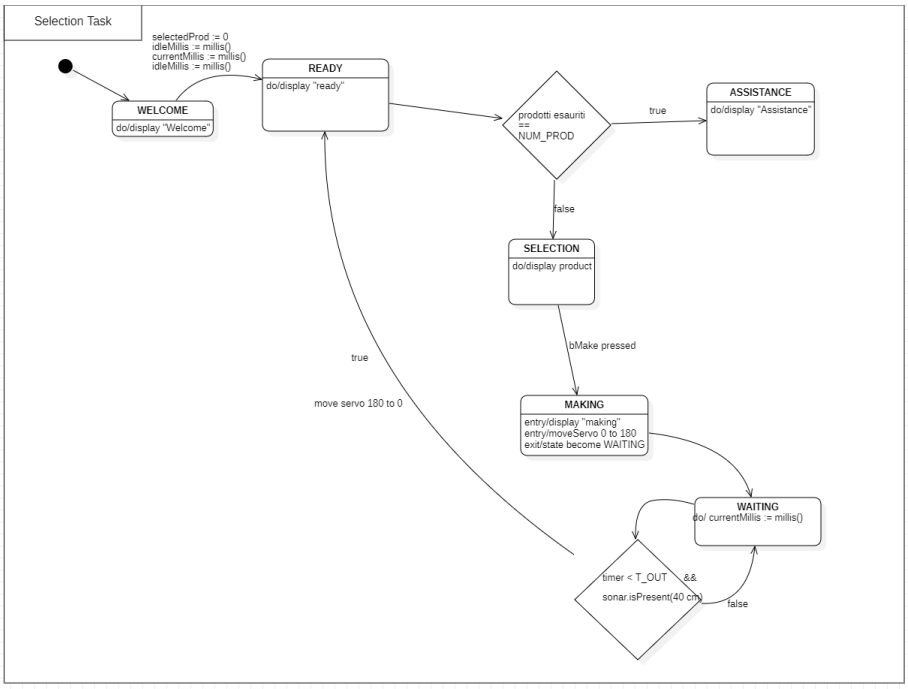

Figure 2: Selection Task Diagram

Check Task

Il Task di check effettua una routine per la quale la macchinetta controlla periodicamente che i suoi componenti funzionino correttamente, l'utlizzo di un contatore all'interno rende comodo effettuare il task ogni 180 secondi, tempo che altrimenti sarebbe stato difficilmente assegnabile come period nella dichiarazione della task.

E' stato riscontrato un problema con il sensore di temperatura, pur usando lo stesso codice a nostra a disposizione e 3 sensori diversi di temperatura ottenevamo temperature altissime quindi abbiamo diminuito di 25 il nostro valore di temperatura.

```
/* CheckTask.cpp */
```

```
void CheckTask::checkTemp(){
    int temp = analogRead(TEMP_PIN);
    int value_in_mV = temp * 4.8876;
    double value_in_C = value_in_mV * 0.1;
    value_in_C -= 25;
    if (value_in_C < TEMP_MIN || value_in_C > TEMP_MAX)
    {
        this->machine->state = ASSISTANCE;
        this->machine->coffee->setQuantity(0);
        this->machine->tea->setQuantity(0);
        this->machine->chocolate->setQuantity(0);
    } else {
      Serial.println(value_in_C);
      this->machine->state = READY;
    }
}
```

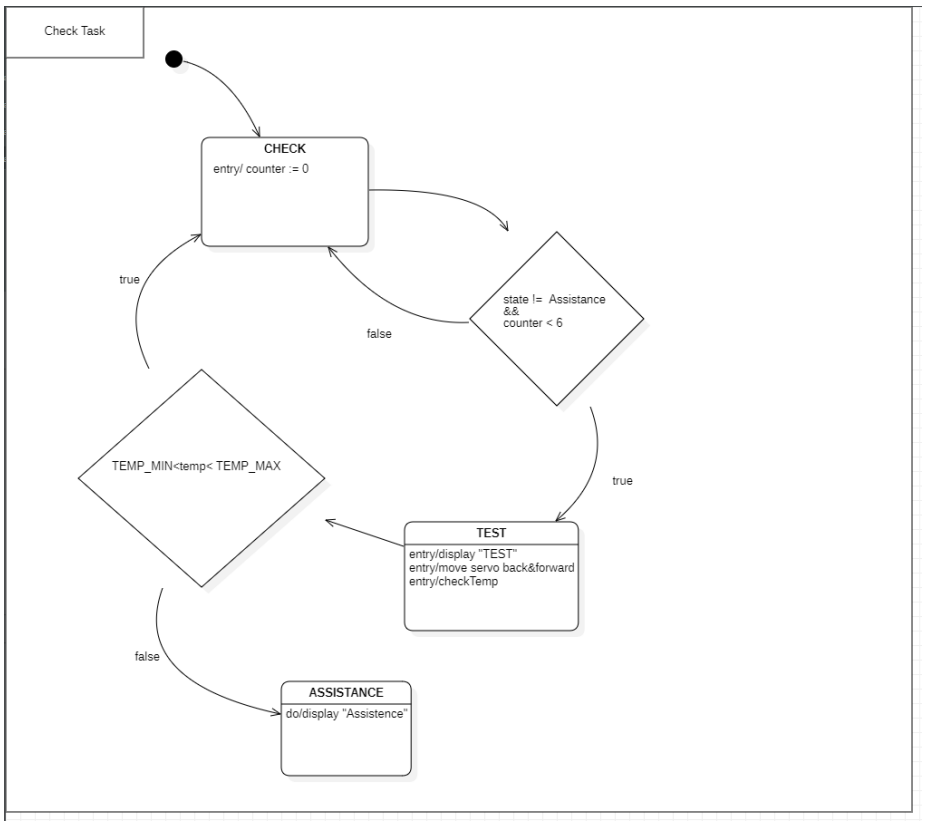

Figure 3: Check Task Diagram

Sugar Task

Terzo task, ad ogni tick questo task controlla che il selettore di zucchero (il potenziometro) non sia stato mosso, altrimenti se mosso oltre una certa soglia modifica e stampa a display la quantita' di zucchero che la nostra bevanda selezionata dovra' contenere.

GUI

Abbiamo creato la GUI in javaFX, in cui sono stati inseriti diversi elementi tra cui i bottoni per il refill/recover e label per mostrare lo stato e le quantità dei prodotti.
Si è considerato lo stato di idle come l'attesa di un input da parte dell'utente, lo stato working invece come quello della selezione e l'erogazione del prodotto.

A livello di codice invece abbiamo usato la funzione.

```
new Timer().scheduleAtFixedRate(new TimerTask(){

            @Override
            public void run(){
              // code to run
            }
        },0, PERIOD);
```

Per poter eseguire del codice ogni PERIOD (in questo caso 300ms), all'interno del quale e' stato messo il codice con il quale la GUI si mette in attesa di un messaggio. Tramite questo approccio si evita completamente la perdita di messaggi e la GUI puo' aggiornarsi in modo autonomo e in tempo reale. Questo codice eseguito all'interno del controller della GUI pero' viene eseguito inun thread diverso da quello della GUI e si ottiene un errore che blocca tutto, per poter risolvere questo problema e' stato necessario richiamare una funzione che va ad operare sulla GUI quale:

```
Platform.runLater(() -> {
                updateGui(coffee, tea, chocolate, check, stato);
            });
```

Per il formato del messaggio per comodita' il JSON semplifica di molto lo scambio di informazioni.