



CONTENUTI	<ul style="list-style-type: none">• Aggiungi qui i titoli delle sezioni. Sono utili dei brevi riepiloghi!• Consiglio: compila questa sezione dopo aver terminato tutte le altre.
------------------	---

Progetto Industriale: *Tango*

Documentazione tecnica del progetto industriale del corso 12.1 di ITS - Accademia Digitale Liguria, svolto in collaborazione con Scuola di Robotica.

Introduzione

Tango è un robot in grado di muoversi, registrare dati dall'ambiente circostante e trasportare carichi. È controllato da remoto tramite una mobile app sviluppata dal nostro team. Questo documento descrive lo scopo, gli obiettivi e le finalità del software di gestione, analizza il codice, fornisce una panoramica riguardo la progettazione dei componenti fisici in ambiente Blender (con l'ausilio di Shapr3D e XTools), descrive il montaggio dei componenti a bordo macchina e offre tavole illustrate sullo schema elettrico e i collegamenti hardware, fungendo anche un manuale d'uso.

Obiettivo del progetto

L'obiettivo del progetto Tango è l'assemblaggio di un robot e del software necessario per gestirlo. Il robot viene controllato tramite una applicazione mobile sviluppata appositamente.

Background

Il corso 12.1 di *ITS - Accademia Digitale Liguria* deve portare a termine, secondo curriculum, la progettazione e realizzazione di una macchina IoT nello spazio di 84 ore. Abbiamo deciso, insieme al team di *Scuola di Robotica*, di procedere alla personalizzazione del robot *iRobot ATRV jr*, dotandolo di sensori ambientali e di un software di controllo remoto.

Dopo una prima analisi, sono stati identificati quattro obiettivi principali:

- Controllo del robot: per poter portare a termine i propri compiti, il robot necessita di un'unità di controllo opportunamente programmata.
- Registrazione dei dati dall'ambiente circostante: attraverso i sensori montati a bordo macchina, il robot deve essere in grado di registrare dati dall'ambiente circostante.
- Movimento del robot: il robot deve essere in grado di muoversi, controllato da remoto, in ogni direzione senza limiti.
- Il robot deve essere in grado di fermarsi immediatamente in caso di emergenza.

Panoramica del documento

Indice

Requisiti

Overview delle funzioni

- Caratteristiche delle funzioni

Analisi del codice

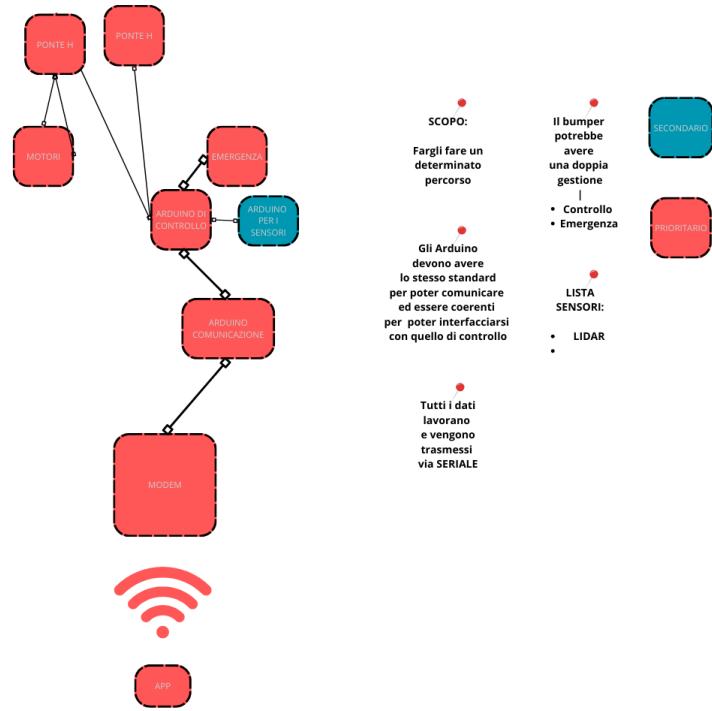
Testing del codice

Appendice - Manuale di utilizzo

Requisiti

Specifiche tecniche

- Robot ATRV-JR
- Arduino Mega 2560 (3 schede)
- App per dispositivi mobile sviluppata in Java





Overview delle funzioni

Il software svolge tre macro-funzioni:

- comunicazione;
- controllo;
- gestione dei sensori.

Comunicazione

Il sistema di comunicazione bidirezionale collega l'applicazione Android e il sistema di controllo del robot, che è basato su Arduino MEGA.

Applicazione Android

L'interfaccia utente per il controllo di Tango è un'app mobile sviluppata utilizzando Android Studio. Offre un'interfaccia intuitiva per gli utenti finali, consentendo loro di inviare comandi di controllo al robot tramite degli input touchscreen. Tutti gli input generati dall'applicazione vengono inviati a un server Node.js tramite richieste HTTP POST.

Server Node.js

Il server Node.js funge da intermediario tra l'applicazione Android e Arduino MEGA. È responsabile della ricezione degli input inviati dall'applicazione e dell'inoltro di tali dati al dispositivo Arduino MEGA tramite una connessione Ethernet. Il server è stato implementato per gestire le richieste HTTP POST provenienti dall'applicazione Android e instradare correttamente i dati al dispositivo Arduino MEGA.

Arduino MEGA

Arduino MEGA è il componente centrale del sistema di controllo del robot. È stato equipaggiato con uno shield Ethernet per consentire la comunicazione con il server Node.js ed eventualmente altri dispositivi di rete. Una parte di codice è stata sviluppata per configurare Arduino MEGA come un server utilizzando la libreria Ethernet.h. Questa parte di codice permette la ricezione dei comandi di controllo inviati dal server Node.js per poi eseguirli, controllando così i motori di Tango.

Flusso di comunicazione

Il flusso di comunicazione inizia con l'utente che interagisce con l'applicazione Android tramite input touchscreen. Gli input vengono quindi inviati al server Node.js tramite richieste HTTP POST. Il server Node.js riceve gli input, li elabora e li inoltra alla board Arduino MEGA tramite la connessione Ethernet. Infine, Arduino MEGA interpreta i comandi ricevuti e attua le azioni corrispondenti per controllare i motori del robot.

Controllo

Il sistema di controllo è basato su una scheda Arduino MEGA che gestisce la comunicazione e il controllo del veicolo, garantendo la sicurezza e la coerenza del movimento in base ai comandi ricevuti e allo stato dei sensori.

In aggiunta, alla scheda Arduino MEGA di controllo è demandato anche il compito di gestire il sensore a ultrasuoni per la rilevazione degli ostacoli.

Scheda Arduino - Controllo e Comunicazione

Si tratta del firmware per il controllo di Tango tramite un Arduino MEGA, che permette la gestione della comunicazione con altri dispositivi e sensori. Di seguito, una breve panoramica delle sue funzionalità:

Configurazione e inizializzazione: in questa sezione vengono dichiarate le variabili, configurati i pin di input/output e inizializzate le comunicazioni seriali con altri dispositivi.

Loop principale: il programma esegue continuamente un ciclo in cui controlla la comunicazione seriale, mappa i messaggi ricevuti, gestisce lo stato di emergenza e il movimento del veicolo.

Gestione dell'emergenza: viene definita una funzione per gestire lo stato di emergenza del veicolo, che ferma il veicolo in caso di emergenza e invia segnali agli altri dispositivi.

Comunicazione seriale: una funzione per gestire la comunicazione seriale, sia in entrata che in uscita, con altri dispositivi e sensori.

Gestione della distanza: utilizzando sensori ad ultrasuoni, viene misurata la distanza dagli oggetti circostanti e si gestiscono situazioni di emergenza se la distanza è inferiore a una certa soglia.

Gestione del movimento: la funzione movement() gestisce il movimento del veicolo in base ai comandi ricevuti. Utilizza un'interruzione per rilevare lo stato di emergenza e reagisce di

conseguenza.

Controllo della velocità: ci sono funzioni per controllare la velocità del veicolo e garantire che rimanga entro limiti sicuri.

Gestione della Rotazione: viene controllata la rotazione del veicolo e vengono prese misure per evitare movimenti indesiderati o pericolosi.

Funzioni Ausiliarie: le funzioni ausiliarie gestiscono operazioni specifiche, come l'arresto dei motori o la decelerazione graduale.

Sensori

Scheda Arduino - Sensori

Sulla scheda dedicata è stato caricato un software che si occupa di misurare la temperatura e l'umidità ambientale con un sensore DHT11 e di monitorare lo stato della batteria tramite un partitore DFR0051.

Le informazioni raccolte vengono quindi trasmesse tramite la comunicazione seriale per essere lette da un dispositivo esterno, come ad esempio un computer, e possono anche essere visualizzate su un display LCD. Nel nostro caso, verranno poi visualizzate tramite l'app dedicata.

Di seguito, una breve panoramica delle sue funzionalità.

Gestione di temperatura e umidità: Viene utilizzato un sensore DHT11 per misurare la temperatura e l'umidità ambiente e questi fornisce le informazioni tramite la comunicazione seriale.

Monitoraggio dello stato della batteria: Lo stato della batteria viene monitorato misurando la tensione tramite un partitore DFR0051. Il dato viene trasmesso tramite la comunicazione seriale.

Loop principale: Nel loop principale del programma, vengono gestite le operazioni di monitoraggio della distanza, aggiornamento del display LCD e monitoraggio periodico della temperatura, umidità e tensione della batteria.

Gestione del tempo: Vengono eseguite determinate azioni ogni tot minuti utilizzando il modulo millis() per calcolare il tempo trascorso.

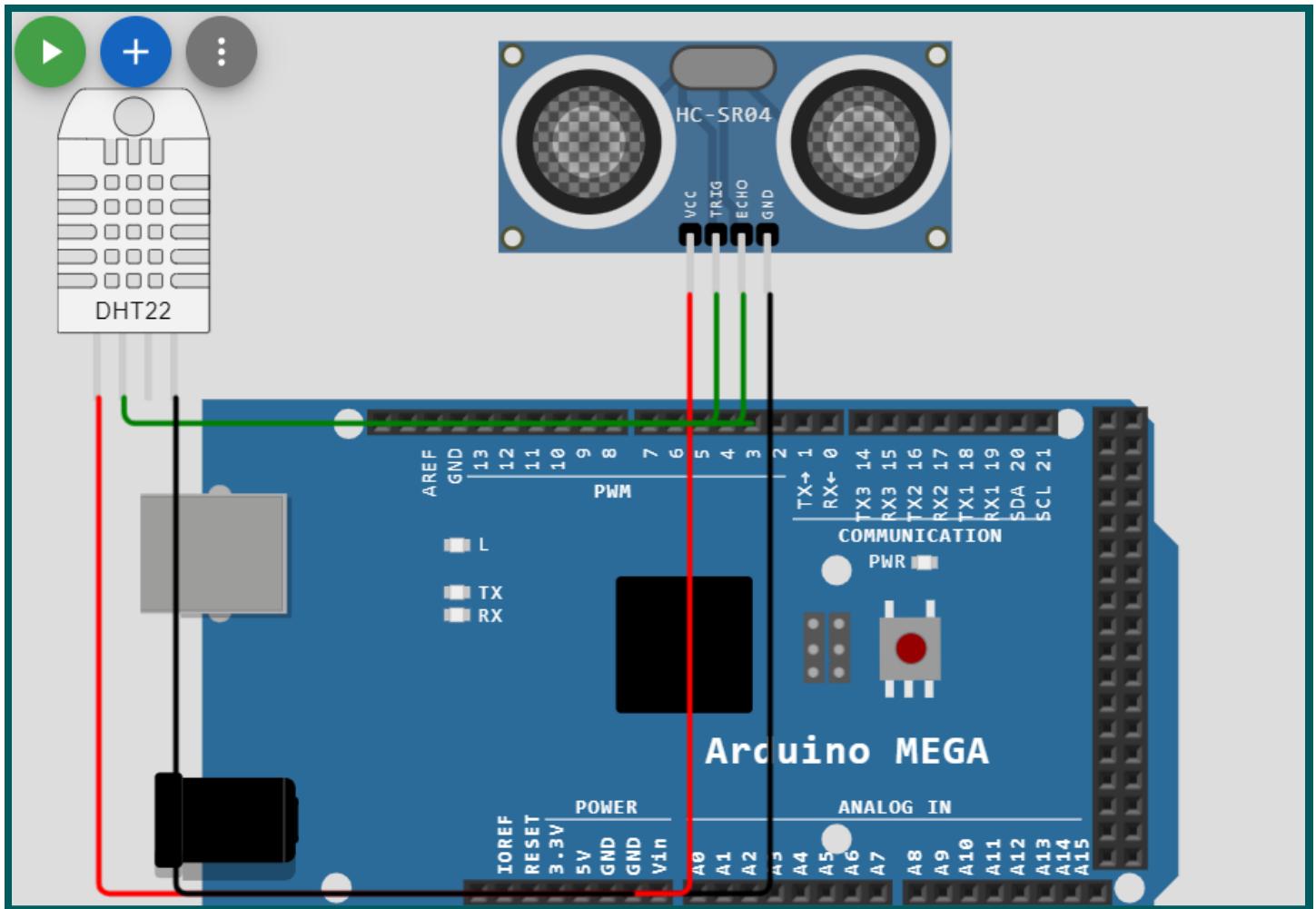


Fig. 1 - simulazione di collegamento sensori su board Arduino MEGA

Sviluppo - CODICE

L'analisi del codice permette una migliore comprensione dei passi fatti nello sviluppo e delle buone pratiche impiegate.

ArduinoServer.ino:

Questo framework è stato progettato in modo da gestire la connettività di rete Ethernet, il controllo di emergenza e la comunicazione con un altro dispositivo Arduino.

Dipendenze

Il codice utilizza le seguenti librerie:

- **SPI**: Comunicazione SPI con la scheda Ethernet.
- **Ethernet**: Gestione della connessione Ethernet.
- **String**: Manipolazione delle stringhe.
- **ArduinoJson**: Analisi dei dati JSON ricevuti tramite le richieste HTTP.

```
#include <SPI.h>
#include <Ethernet.h>
#include <string.h>
#include <ArduinoJson.h>
```

Configurazione iniziale

Di seguito, vengono definiti i pin utilizzati e le variabili per la gestione della connettività e dello stato di comunicazione. Vengono quindi impostate le informazioni di rete, tra cui l'indirizzo MAC, l'indirizzo IP del dispositivo Arduino e l'indirizzo IP del server remoto.

```

//PINS

int emergency = 13;

// Connection checks variables

unsigned long startTime = millis();

unsigned long currentTime;

unsigned long duration = 4500;

bool checked = false;

bool failed = false;

bool comExtableshed = false;

//Connectivity Test IP (Node Server)

int currIPNode = 2;

//Arduino Server

int arduinolP = 4;

int arduinoPort = 80;

// Network Settings

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED }; // MAC address

IPAddress ip(192, 168, 1, arduinolP); // IP address

IPAddress nodeServer(192, 168, 1, currIPNode); // IP address of the node server

EthernetClient client;

EthernetServer server(80);

```

Parametri di configurazione di rete:

Per quanto riguarda i parametri di configurazione di rete per la scheda Arduino Mega, è stata ipotizzata una connessione Ethernet per la comunicazione.

```
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED }; // MAC address  
IPAddress ip(192, 168, 1, arduinolP); // IP address  
IPAddress nodeServer(192, 168, 1, currIPNode); // IP address of the node server
```

Viene definito l'indirizzo MAC del dispositivo Arduino, univoco e perciò non customizzabile.

Per impostare l'indirizzo IP del dispositivo, viene utilizzato l'oggetto IPAddress e i parametri 192, 168, 1 e arduinolP. Quest'ultimo deve essere sostituito dall'ultimo ottetto dell'indirizzo IP del dispositivo Arduino in rete.

Per definire l'indirizzo IP del server nodo, con cui l'Arduino comunicherà, viene utilizzato anche in questo caso l'oggetto IPAddress, con 192, 168, 1 come parte fissa dell'indirizzo IP e currIPNode come variabile che rappresenta l'ultimo ottetto dell'indirizzo IP del server nodo all'interno della rete.

Setup()

Nella funzione di setup, vengono inizializzati i pin, la comunicazione seriale e la connessione Ethernet.

```
void setup() {  
    // emergency PIN  
    pinMode(13, OUTPUT);  
  
    unsigned long startTime = millis();  
  
    Serial.begin(9600);  
  
    // -- Start the Ethernet connection --  
  
    Ethernet.begin(mac);  
  
    Ethernet.setLocalIP(ip);  
    // -- Testing connectivity -- Serial.println(">> Testing connectivity (using Node server connection on  
    port 3000)");  
  
    if (Ethernet.linkStatus() == LinkOFF){  
  
        Serial.println("Ethernet cable is not connected!");  
    }  
}
```

Viene quindi eseguito un test di connettività con il server remoto usando i retry e viene avviato il server Arduino.

```
//  
  
// -- HTTP GET request to test connectivity (using retries) --  
  
//  
  
for (int retryCount = 0; retryCount < 3; retryCount++) {  
  
    if (client.connect(nodeServer, 3000)) {  
  
        /*DEBUG: */Serial.println("Connected to server");  
  
        client.println("GET /");  
  
        String connectionString = String("Host: 192.168.0." + String(currIPNode));  
  
        client.println(connectionString); // Replace with the actual IP address or domain of Node server  
  
        client.println("Connection: keep-alive");  
  
        client.println();  
  
        break;  
  
    } else{  
  
        if (retryCount == 2){  
  
            /*DEBUG: */Serial.println("Connection failed!");  
  
        } else{  
  
            /*DEBUG: */Serial.println("Connection failed. Retrying...");  
  
            delay(3000);  
  
        }  
  
    }  
  
}  
  
client.stop();
```

Funzioni ausiliarie

- **startEmergencyStop()**: Invia un segnale di emergenza per interrompere il funzionamento del sistema.

```
void startEmergencyStop(){  
  
    digitalWrite(emergency, HIGH);  
  
    delay(1000);  
  
    digitalWrite(emergency, LOW);  
  
}
```

Viene impostato il pin emergency su HIGH (o 1), attivando il dispositivo di arresto di emergenza, mentre la successiva funzione delay() permette l'attesa di 1000 millisecondi prima di ripristinare il pin emergency a LOW (o 0) per disattivare il dispositivo di arresto.

- **forwardToControlArduino()**: Invia una stringa di direzione all'altro Arduino.emerge

```
void forwardToControlArduino(String value){  
  
    Serial.println(value);  
  
}
```

Loop()

La funzione loop() gestisce continuamente la connessione con il client, riceve e analizza i dati inviati tramite richieste HTTP POST e gestisce il controllo di emergenza. Viene anche letta la comunicazione seriale dall'altro Arduino per acquisire i valori dei sensori disponibili.

```
void loop() {  
  
currentTime = millis();  
  
EthernetClient client = server.available();  
  
if (client) {  
  
boolean currentLineIsBlank = true;  
  
while (client.connected()) {  
  
while(client.available()) {  
  
char c = client.read();  
  
if (c == '\n' && currentLineIsBlank) {  
  
while(client.available())  
  
{  
  
String body = client.readString();  
  
String key = body.substring(body.indexOf("{\"") + 2, body.indexOf("\":\"")); //get key from json  
  
String value = body.substring(body.indexOf(":\"") + 2, body.length() - 2); // get value from json  
  
if (key == "arduinoCheck"){  
  
if (value == "stop"){  
  
comExtableshed = false;  
  
} else if (value == "ok"){  
  
checked = true;  
  
}  
  
}  
  
if (key == "direction"){  
  
if (value != "emergencyStop"){  
  
forwardToControlArduino(value);  
  
comExtableshed = true;  
  
}
```

```
        }

        else {

            comExtableshed = false;

            Serial.println("EMERGENCY!");

            startEmergencyStop();

        }

    }

    Serial.print(key);

}

Serial.println();

client.println("HTTP/1.0 200 OK");

client.println("Content-Type: text/html");

client.println();

client.println("ok");

client.stop();

}

else if (c == '\n') {

    currentLineIsBlank = true;

}

else if (c != '\r') {

    currentLineIsBlank = false;

}

}

if (currentTime - startTime > duration) {

    startTime = currentTime;

}
```

```
if (checked) {  
  
    Serial.println("> Check passed!");  
  
    checked = false; //check passed  
  
    failed = false; } else if (comExtablished){  
  
    if (!failed){  
  
        failed = true;  
  
    }  
  
    else {  
  
        Serial.println("> Check failed!");  
  
        comExtablished = false;  
  
        checked = false;  
  
        failed = false;  
  
        startEmergencyStop();  
  
    }  
  
}  
  
}  
  
}  
  
readSensorValues();
```

App:

fragment_home.xml

-ConstraintLayout: È il layout principale e contiene tutti gli altri elementi. Utilizza il sistema di vincoli (app:layout_constraint*) per posizionare gli elementi in relazione l'uno all'altro.

```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/rootLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".HomeFragment"
    android:background="@color/lightgray">
```

-FrameLayout (topFrame): È un contenitore che funge da sfondo principale. Contiene un TextView dove si potrà inserire testo a piacere. Ha un'immagine di sfondo (@drawable/frame_border) e una altezza di 310dp.

```
<FrameLayout
    android:id="@+id/topFrame"
    android:layout_width="match_parent"
    android:layout_height="310dp"
    android:layout_marginTop="20dp"
    android:background="@drawable/frame_border"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent">

    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:text="Your Text Here"
        android:textColor="@android:color/white"
        android:textSize="18sp" />

</FrameLayout>
```

-FrameLayout (bottomLeftFrame): È un contenitore posizionato sotto topFrame sulla sinistra e al suo interno verranno mostrati i valori di temperatura. Ha un'immagine di sfondo (@drawable/frame_border) e contiene l'elemento personalizzato TemperatureView che riporterà i valori catturati dal sensore a bordo macchina.

```

<!-- Frame a Sinistra del TopFrame -->
<FrameLayout
    android:id="@+id/bottomLeftFrame"
    android:layout_width="140dp"
    android:layout_height="170dp"
    android:layout_gravity="start"
    android:layout_marginStart="30dp"
    android:layout_marginTop="-200dp"
    android:background="@drawable/frame_border"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/topFrame">

    <com.example.irobotapplication.TemperatureView
        android:id="@+id/temperatureView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</FrameLayout>

```

A destra e al centro rispetto a questo contenitore sono stati inseriti altri due contenitori di dimensioni e caratteristiche identiche, allo scopo di mostrare rispettivamente i dati sull'umidità rilevati dal sensore e sul livello di batteria disponibile per Tango.

-TextView: All'interno di topFrame, visualizza il testo definito in origine al centro con uno stile specificato.

```

<TextView
    android:id="@+id/textView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:text="Your Text Here"
    android:textColor="@android:color/white"
    android:textSize="18sp" />

```

-TemperatureView: È un componente personalizzato (com.example.irobotapplication.TemperatureView) posizionato dentro bottomLeftFrame. Il suo aspetto e comportamento specifici dipenderanno dall'implementazione all'interno della classe TemperatureView.

```
<com.example.irobotapplication.TemperatureView  
    android:id="@+id/temperatureView"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"/>
```

-BatteryView: È un componente personalizzato (com.example.irobotapplication.BatteryView) che visualizza un'immagine (@drawable/ic_chargeing), un'icona di ricarica, e un indicatore di percentuale di batteria (app:bv_percent="80", app:bv_charging="false").

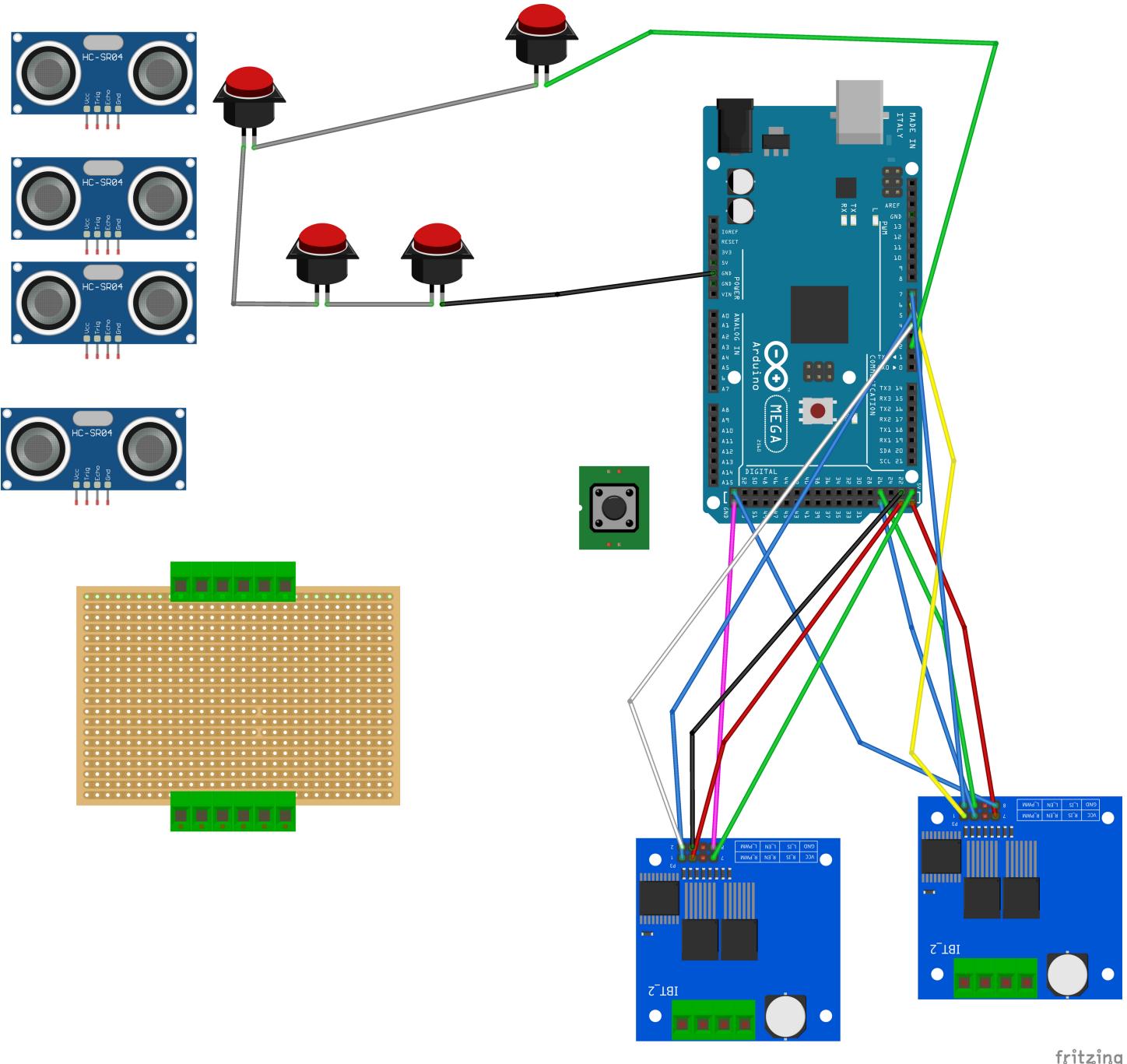
```
<com.example.irobotapplication.BatteryView  
    android:id="@+id/battery_view"  
    android:layout_width="63dp"  
    android:layout_height="130dp"  
    android:src="@drawable/ic_chargeing"  
    app:bv_charging="false"  
    app:bv_percent="80"  
    app:layout_constraintBottom_toBottomOf="parent"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toTopOf="parent" />
```

Sviluppo - STAMPA E GRAFICA

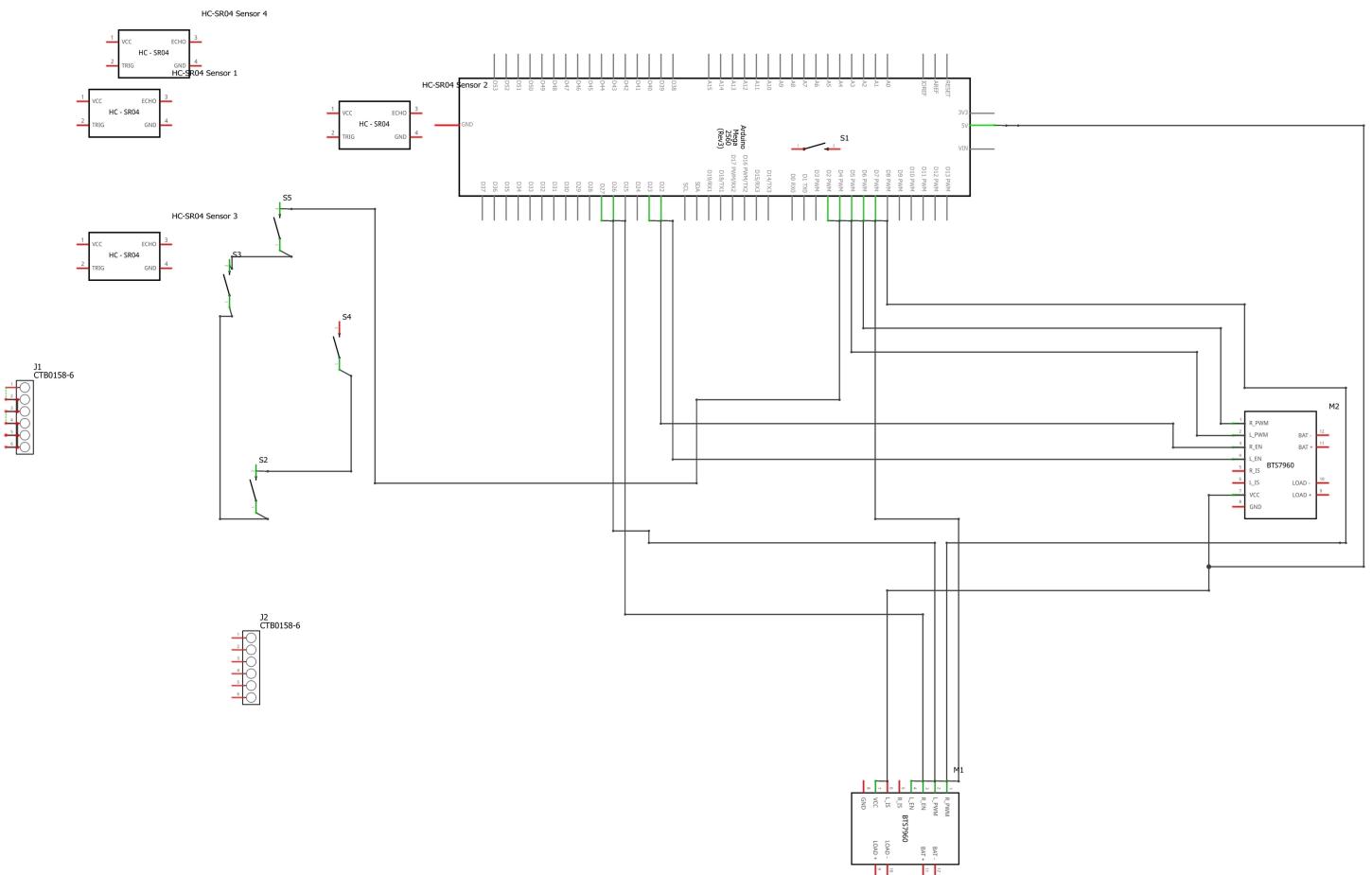
1. Creazione di render per la stampa 3D e con lasercut di componenti
2. Uso di programmi come Xtool Creative Space (XCS) e Shapr3D

Sviluppo - HARDWARE

1. Montaggio dei componenti hardware a bordo macchina
2. Schema elettrico



fritzing



fritzing

TESTING

1. Modalità di testing del software
2. Modalità di testing dell'hardware