Swarm Scribble Bot - Prototipo

Descrizione

Il primo prototipo che è stato pensato è quello visualizzato nelle seguenti foto (figura 1, 2, 3, 4)

Fig. 1

Fig. 2

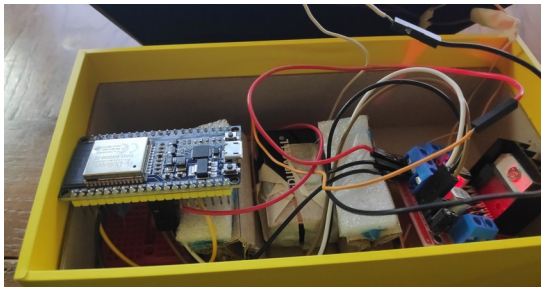

Fig. 3

Versione beta– 02.02.2025

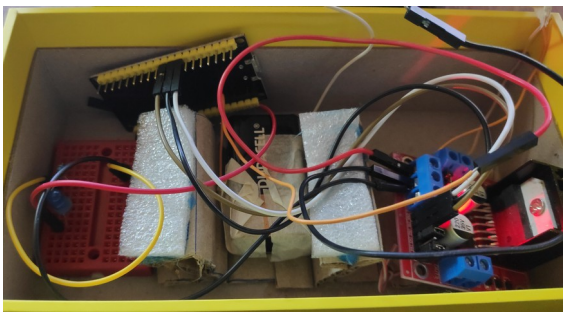

Fig. 4

Sfortunatamente, tenuto conto che le batterie sono dei componenti pesanti, la struttura, considerata nella sua interezza, risulta poco dinamica e di conseguenza quando viene attivato il volano mediante l'accensione del motore elettrico i pennarelli si muovo poco e il loro tracciato è minimo. Per risolvere questo problema si è pensato di dividere le due parti. Tutte le componenti elettroniche comprese le batterie vengono inserite nel box e il "castello" con il motore elettrico e i pennarelli viene staccato dal corpo principale collegato con due cavi elettrici molto sottili (figura 5)

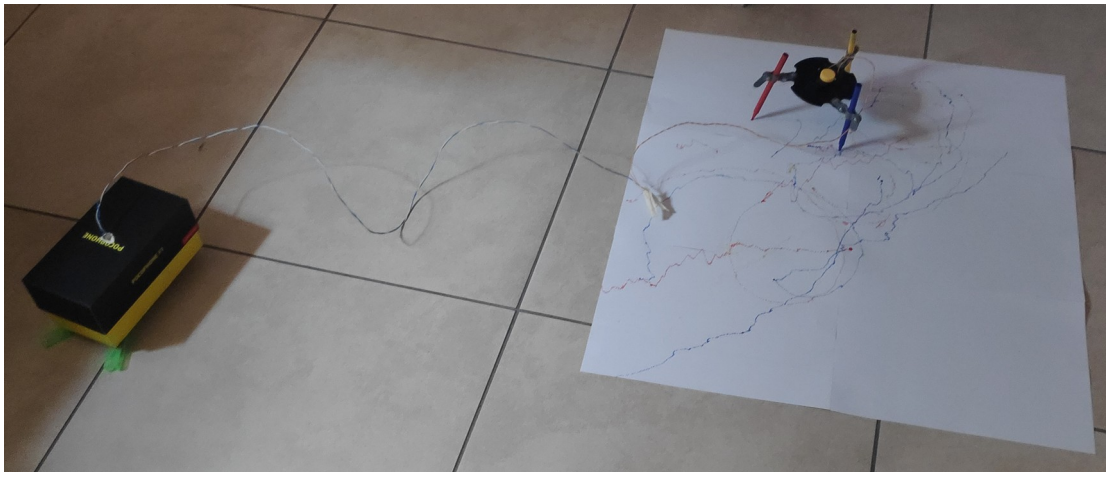

Fig. 5

All'interno del box troviamo il circuito principale con tutte le componenti già descritte nel precedente documento: *ESP32 with DC Motor and L298N Motor Driver* e principalmente la SoC Esp32, la scheda motor driver L298N e la batteria da 9V, eventualmente il power bank. Nel prototipo proposto è stato inserito anche il circuito per l'accensione o lo spegnimento del led.
In sintesi nel corpo del SSB sono racchiusi due circuiti (figura 6 - Led e figura 7 - Motor)

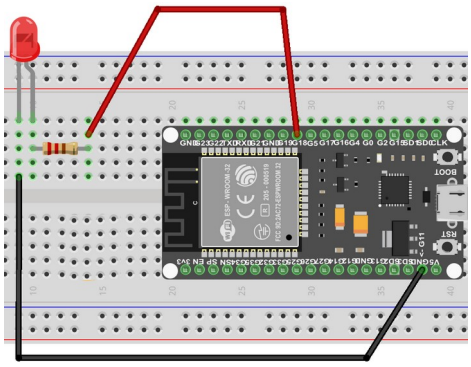

Fig. 6 - Led

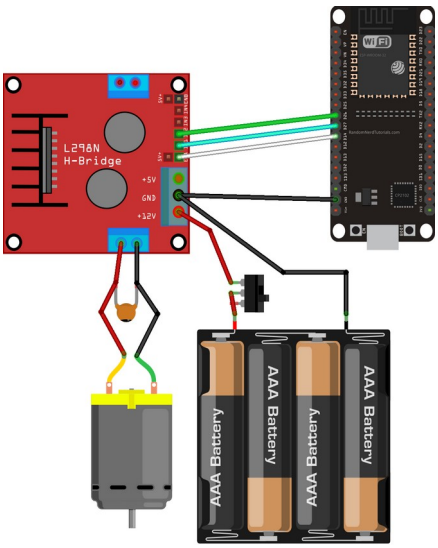

Fig. 7 – Motor

Programmazione della scheda Esp32

Tramite la connessione della scheda con il software Arduino IDE si procede alla configurazione delle funzionalità del Bot. Di seguito descriviamo il codice utilizzato (listato 1)

```
//#############################################################################
//# Filename    : Server_Bluetooth_Motor_Led.c
//# Project IoTA Swarm Scribble Bot
//# Description : Turn on/off Led and motor from the web
//# Author      : Marco Ercolani in data 21.01.2025
//# A.A. 2024/2025
//# Corso di Interaction Design
//# Scuola Nuove Tecnologie dell'Arte
//# Accademia di Belle Arti di Urbino
//# modification: 2025-01-21
//#############################################################################
//
#include "BluetoothSerial.h"
#include "esp_system.h"
#include "esp_log.h"
#include "esp_mac.h"

// Check if Bluetooth is available
#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to and enable it
#endif

// Check Serial Port Profile
#if !defined(CONFIG_BT_SPP_ENABLED)
#error Serial Port Profile for Bluetooth is not available or not enabled. It is only available for the ESP32 chip.
#endif

// Check Simple Secure Pairing
#if !defined(CONFIG_BT_SSP_ENABLED)
#error Simple Secure Pairing for Bluetooth is not available or not enabled.
#endif
// Nome del Bluetooth con cui individuare il Bot all'interno dello Sciame
const char *deviceName = "IoTAGrigio";
```

```
// The following lines defines the method of pairing
// When both Input and Output are false only the other device authenticates pairing without any pin.
// When Output is true and Input is false only the other device authenticates pairing without any pin.
// When both Input and Output are true both devices display randomly generated code and if they match authenticate pairing on both devices
//   - This must be implemented by registering callback via onConfirmRequest() and in this callback request user input and call confirmReply(true); if the authenticated
//      otherwise call `confirmReply(false)` to reject the pairing.
// When Input is true and Output is false User will be required to input the passkey to the ESP32 device to authenticate.
//   - This must be implemented by registering callback via onKeyRequest() and in this callback the entered passkey will be responded via respondPasskey(passkey);
const bool INPUT_CAPABILITY = true;  // Defines if ESP32 device has input method (Serial terminal, keyboard or similar)
const bool OUTPUT_CAPABILITY = true;  // Defines if ESP32 device has output method (Serial terminal, display or similar)

// PIN Led
int ledPin = 32;
//
// Engine management pin
//
int motor1Pin1 = 27;
int motor1Pin2 = 26;
int enable1Pin = 14;

// Setting PWM properties
int freq = 30000;
int pwmChannel = 0;
int resolution = 8;
int dutyCycle = 200;
int dutCycleTop = 260;
//
//To view the Bluetooth MAC
//
uint8_t baseMac[6];
//
BluetoothSerial SerialBT;
bool confirmRequestDone = false;
//
// Functions used when using authentication in Bluetooth
//
void BTConfirmRequestCallback(uint32_t numVal) {
  confirmRequestDone = false;
#ifndef AUTO_PAIR
  Serial.printf(
    "The PIN is: %06lu. If it matches number displayed on the other device write \'Y\' or \'y\':\n", numVal
  );  // Note the formatting "%06lu" - PIN can start with zero(s) which would be ignored with simple "%lu"
  while (!Serial.available()) {
    delay(1);  // Feed the watchdog
    // Wait until data is available on the Serial port.
  }
  Serial.printf("Oh you sent %d Bytes, lets see...", Serial.available());
  int dat = Serial.read();
  if (dat == 'Y' || dat == 'y') {
    SerialBT.confirmReply(true);
  } else {
    SerialBT.confirmReply(false);
  }
#else
```

```
  SerialBT.confirmReply(true);
#endif
}
//
// Functions used when using authentication in Bluetooth
//
void BTKeyRequestCallback() {
  Serial.println("BTKeyRequestCallback");  // debug
  char buffer[7] = {0};                   // 6 bytes for number, one for termination '0'
  while (1) {
    Serial.print("Enter the passkey displayed on the other device: ");
    while (!Serial.available()) {
      delay(1);  // Feed the watchdog
      // Wait until data is available on the Serial port.
    }
    size_t len = Serial.readBytesUntil('\n', buffer, sizeof(buffer) - 1);
    buffer[len] = '\0';  // Null-terminate the string.
    try {
      uint32_t passkey = std::stoi(buffer);
      Serial.printf("Entered PIN: %lu\n", passkey);
      SerialBT.respondPasskey(passkey);
      return;
    } catch (...) {
      Serial.print("Wrong PIN! Try again.");
    }  // try
  }  // while(1)
}
//
//Function that ends the Bluetooth pairing (connection) process
//
void BTAuthCompleteCallback(boolean success) {
  if (success) {
    confirmRequestDone = true;
    Serial.println("Pairing success!!!!!");
  } else {
    Serial.println("Pairing failed, rejected by user!!");
  }
}
//
// Function that activates the motor in a clockwise rotation
//
void MotorForward(void) {
  // Forward the DC motor
  Serial.println("Moving Forward");
  ledcWrite(enable1Pin, dutyCycle);
  digitalWrite(motor1Pin1, HIGH);
  digitalWrite(motor1Pin2, LOW);

}
//
// Function that stops the rotating engine
//
void MotorStop(void){
  // Stop the DC motor
  Serial.println("Motor stopped");
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, LOW);
}
//
// Function that activates the motor in a counterclockwise rotation
```

```
//
void MotorBackwards(void){
  // Move DC motor backwards at maximum speed
  Serial.println("Moving Backwards");
  ledcWrite(enable1Pin, dutyCycle);
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, HIGH);

}
//
// Function that turns on the LED
//
void LedOn(void){
  digitalWrite(ledPin,HIGH);
  Serial.print("Led On...");
}
//
// Function that turns off the LED
//
void LedOff(void){
  digitalWrite(ledPin,LOW);
  Serial.print("Led Off...");
}
//
// Bluetooth Server Main Function
//
void serial_response() {
  if (Serial.available()) {
    SerialBT.write(Serial.read());
  }
  if (SerialBT.available()) {
    // A new command is received from the SSB system
    String cmdIoTA = SerialBT.readString();
    Serial.println("Command Received:");
    Serial.println(cmdIoTA);
    Serial.println("");
    Serial.println("-----------------");
    // The new command is analyzed and the related operation is performed
    if ( cmdIoTA == "IoTALedOn" ) {
      Serial.println("Bisogna accendere il Led");
      LedOn();
    } else if ( cmdIoTA == "IoTALedOff" ){
      Serial.println("Bisogna spegnere il Led");
      LedOff();
    } else if ( cmdIoTA == "IoTAMotorForward" ){
      Serial.println("Bisogna accendere il Motore in avanti");
      MotorForward();
    } else if ( cmdIoTA == "IoTAMotorBackwards" ){
      Serial.println("Bisogna accendere il Motore indietro");
      MotorBackwards();
    }else if ( cmdIoTA == "IoTAMotorStop" ){
      Serial.println("Bisogna spegnere il Motore");
      MotorStop();
    } else {
      Serial.println("Comando non riconosciuto");
    }
  }
  delay(20);
}
//
```

```
// Setup of the Esp32, Bluetooth and the two circuits
//
void setup() {
                // Esp32 Console Setup
        Serial.begin(115200);
        // Pin setup for the LED
        pinMode(ledPin,OUTPUT);
        // Get the MAC address of the Bluetooth interface
        esp_read_mac(baseMac, ESP_MAC_BT);
        Serial.print("Bluetooth MAC: ");
        for (int i = 0; i < 5; i++) {
           Serial.printf("%02X:", baseMac[i]);
        }
        Serial.printf("%02X\n", baseMac[5]);
        // Pin Setup for Motor
        Serial.print("------> Begin Setup DC Motor...");
        pinMode(motor1Pin1, OUTPUT);
        pinMode(motor1Pin2, OUTPUT);
        pinMode(enable1Pin, OUTPUT);
        // Setting PWM properties
        freq = 30000;
        pwmChannel = 0;
        resolution = 8;
        dutyCycle = 200;
             // If the engine circuit is well built then the engine setup process is complete.
        if ( ledcAttachChannel(enable1Pin, freq, resolution, pwmChannel) ) {
          Serial.println("------> Attivazione canale pwd...");
          ledcWrite(enable1Pin, dutyCycle);
        } else {
          Serial.println("------> Attenzione! Attivazione canale pwd non riuscita...");
        }
        Serial.print("------> End Setup DC Motor...");
        //
        // Part relating to the Bluetooth setup
        //
        SerialBT.enableSSP(INPUT_CAPABILITY, OUTPUT_CAPABILITY);  // Must be called before begin
        SerialBT.onConfirmRequest(BTConfirmRequestCallback);
        SerialBT.onKeyRequest(BTKeyRequestCallback);
        SerialBT.onAuthComplete(BTAuthCompleteCallback);
        SerialBT.begin(deviceName);  // Initiate Bluetooth device with name in parameter
        //SerialBT.deleteAllBondedDevices(); // Uncomment this to delete paired devices; Must be called after begin
        Serial.printf("The device started with name \"%s\", now you can pair it with Bluetooth!\n", deviceName);
        Serial.println("Begin operation");
        if (INPUT_CAPABILITY and OUTPUT_CAPABILITY) {
                Serial.println("Both devices will display randomly generated code and if they match authenticate
pairing on both devices");
        } else if (not INPUT_CAPABILITY and not OUTPUT_CAPABILITY) {
                Serial.println("Authenticate pairing on the other device. No PIN is used");
        } else if (not INPUT_CAPABILITY and OUTPUT_CAPABILITY) {
                Serial.println("Authenticate pairing on the other device. No PIN is used");
        } else if (INPUT_CAPABILITY and not OUTPUT_CAPABILITY) {
                Serial.println("After pairing is initiated you will be required to enter the passkey to the ESP32 device
to authenticate\n > The Passkey will displayed on "
     "the other device");
        }
}
//
// Swarm Scribble Bot command reception cycle
//
void loop() {
```

```
        if (confirmRequestDone) {
                serial_response(); // Analyze the command received
        } else {
                delay(1);  // Feed the watchdog ( Wait a time )
        }
}
```

Lst. 1

Si segnala che è molto importante durante le prime esecuzioni dello software di programmazione dell'Esp di leggere il MAC Address Bluetooth del SoC. Durante l'esecuzione della funzione di setup del listato 1 viene letta e visualizzata tale in informazione. In particolare le seguenti istruzioni implementano tale passaggio:

```
// Get the MAC address of the Bluetooth interface
esp_read_mac(baseMac, ESP_MAC_BT);
Serial.print("Bluetooth MAC: ");
for (int i = 0; i < 5; i++) {
   Serial.printf("%02X:", baseMac[i]);
}
Serial.printf("%02X\n", baseMac[5]);
```

Un esempio di output a tale proposito potrebbe assomigliare:
00:E0:4C:F4:55:07
Quest'ultima informazione sarà fondamentale per le comunicazioni Bluetooth.
Un altro personalizzazione che dovrà essere svolta nel listato 1 è quella di rinominare la propria definizione Bluetooth e precisamente:

```
// Nome del Bluetooth con cui individuare il Bot all'interno dello Sciame
const char *deviceName = "IoTAGrigio";
```

In tale caso è bene sostituire la voce *IoTAGrigio* con *IoTA<colore gruppo>*.

Test del sistema Esp32

Una vota costruito il circuito per verificare che il circuito dello Scribble Bot funzioni in modo corretto è possibile utilizzare un script Python di test. Il test deve essere svolto da una postazione PC munito di porta Bluetooth attiva. Il file dell script di test è:
Test_Server_Bluetooth_IoTA_Led_Motor.py
Di seguito il codice dello script (listato 2)

```
import bluetooth
import socket
import sys

target_name = "IoTA"
### MAC Address Esp32
target_address = "1C:69:20:CD:7B:DA"
port = 1
### It reads the command test for the Esp32
cmdIoTA = sys.argv[1]
### It checks the connection with the Esp32
if target_address is not None:
   print ("found target bluetooth device with address ", target_address)
   s = socket.socket(socket.AF_BLUETOOTH, socket.SOCK_STREAM, socket.BTPROTO_RFCOMM)
   s.connect((target_address,port))
```

```
    data = cmdIoTA
    enddata=""
    print("--> Send data to Esp32")
    s.send(bytes(data, 'UTF-8'))
    print(data)
    s.send(bytes(enddata, 'UTF-8'))
    s.close()
else:
    print ("Warnings! Could not find target bluetooth device nearby")
```

Lst. 2

Bisogna notare che la variabile *target_address* deve essere aggiornata con il reale MAC Address dell'Esp32 come già segnalato nel listato 1. Lo script si utilizza con le modalità usuali di un programma Python da una linea di comando si esegue il comando:

python3 Test_Server_Bluetooth_IoTA_Led_Motor.py <comando IoTASsb>

esempio:

python3 Test_Server_Bluetooth_IoTA_Led_Motor.py IoTALedOn.

Sistema Swarm Scribble Bot

Le altre componenti importanti del sistema Swarm Scribble Bot sono rappresentate dai seguenti moduli:

- Client_IoTASsb;
- IoTASsb;
- Proxy_IoTA_Aba.

Analogamente ai progetti già analizzati durante il corso (prj_IoTAPing e prj_IoTAPingRay) i componenti elencati presentano funzioni e caratteristiche simili. IoTASssb rappresenta il sito Web che ha il compito di esporre il progetto del gruppo di studio e risulta anche interfaccia di comandi verso il Bot.
Proxy_IoTA_Aba è il server che pilota gli eventi da e verso i due componenti client Python e Web.
Il Client_IoTASsb svolge la funzione di raccogliere gli eventi provenienti dal Proxy e di connettersi con il server Bluetooth dell'Esp32 per poi inviare il comando per attivare/fermare le parti elettroniche.

Client_IoTASsb

Il package Client_IoTASsb come detto riceve gli eventi di input dal Proxy e si connette con il Bluetooth dell'Esp32.
A tal proposito si evidenziano le parti rilevanti di tali funzioni descritte (listato 3).

```
if slc.category in listCommand:
###
### Enable incoming command SSB  from internet
###
### Here you need to enter the actions to be performed when the event is received
###
### Run BOT
###
```

```
success = convertIoTAtoBot(slc.category)
if success == 0:
   necolor = "#FF0000"
   print("Warning! Command failed :-((")
else:
   necolor = "#00FF00"
   print("Eureka! Command success! :-))")
###
### Send the response to client for the return command
###
ne = makeEvent()
ne.color = necolor
ne.value = success
ne.category = slc.category
                    nesend = command_proxy('client_socket','client_web',False, ne.eventIoTA_to_json_string() )
 # convert into JSON string:
sendmsg = nesend.command_proxy_to_json_string()
if sendmsg != None:
   try:
      print(" Si spedisce il msg ---->", sendmsg)
      client.sendall(bytes(sendmsg,'UTF-8'))
   except:
      print(" La spedizione non è andata bene ---->", sendmsg)
            else:
print("Il comado non riconosciuto: ", slc.category)


....
....
....

listCommand = ['IoTALedOn','IoTALedOff','IoTAMotorForward','IoTAMotorBackward','IoTAMotorStop']
###
### Convert Events IoTA in Command Bot
###
def convertIoTAtoBot(cmdIoTA):
   ### Bluetooth Mac Address
   ### Warning! Every Esp32 has your Mac
   ### Example "D8:BC:38:E4:5E:2A"
   ###
   print("Comando verso il BOT ---->", cmdIoTA)
   target_address = "1C:69:20:CD:7B:DA"
   portBluetooth = 1
   success = 0
   if target_address is not None:
      print ("found target bluetooth device with address ", target_address)
      try:
         s = socket.socket(socket.AF_BLUETOOTH, socket.SOCK_STREAM, socket.BTPROTO_RFCOMM)
         s.connect((target_address,portBluetooth))
         data = cmdIoTA
         enddata=""
         print("Start send data")
         s.send(bytes(data, 'UTF-8'))
         print(data)
         s.send(bytes(enddata, 'UTF-8'))
         success = 1
      except:
         success = 0
         print("Server Bluetooth non disponible")
      s.close()
```

```
else:
   print ("could not find target bluetooth device nearby")
return success
```

Lst. 3

Anche per questo listato bisogna notare che la variabile *target_address* deve essere aggiornata con il reale MAC Address dell'Esp32 (vedi listato 1). Per la configurazione e l'uso di questo componente si rimanda alla documentazione di *IoTAPing* e *IoTAPingRay.*

Proxy_IoTA_Aba

Anche il componente Proxy non subisce modifiche rispetto i progetti precedenti già citati. L'unica avvertenza che bisogna osservare è quella che conviene non attivare gli *archive web/client*. Di conseguenza le due opzioni possono essere in entrambe le circostanze uguali a No. Esempio:

```
python3 proxy_iota_aba.py --address <server> --port_client_socket  <port>  --port_client_websocket <port> --logging proxy_iota_aba.log --archive_client No --archive_web No
```

IoTASsb

Il sito web per pilotare lo Scribble Bot ricalca nella struttura e nelle scelte d'implementazione i precedenti esempi di IoTAPing e IoTAPingRay (figura 8).

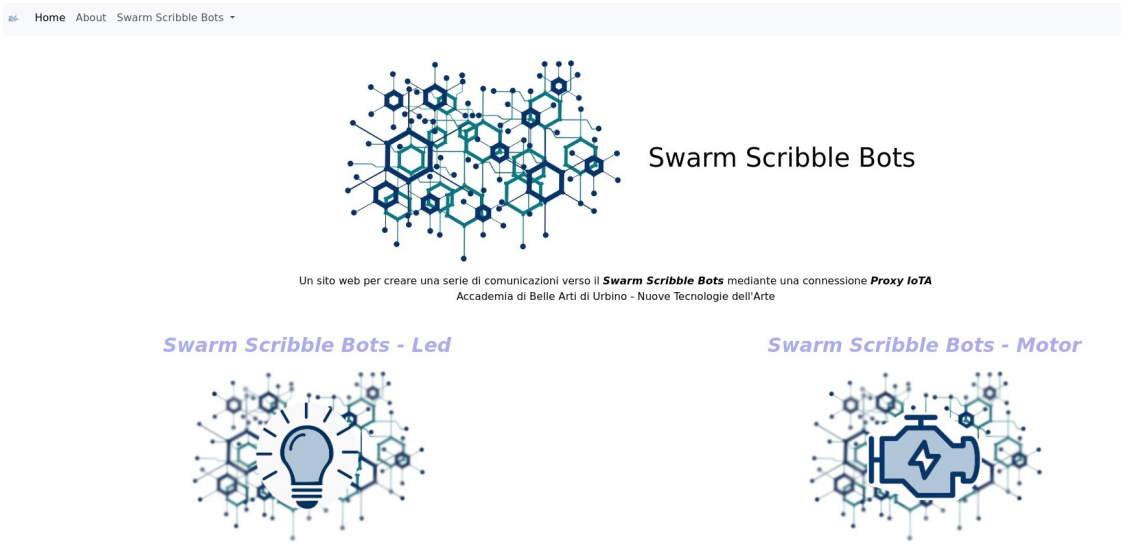

Fig. 8

Le due pagine principali sono *ssb_led.html* e *ssb_motor.html* che pilotano rispettivamente l'accensione e lo spegnimento del led e la rotazione "oraria" , "antioraria" e lo stop del motore.
Quando si attiva l'icona del led o del motore la funzione javascript oltre a disegnare delle forme geometriche nella lavagna invia al Proxy IoTA che contiene il relativo comando (listato 4). I comandi che vengono inviati trameti l'evento sono:

IoTALedOn, IoTALedOff, IoTAMotorForward, IoTAMotorBackward, IoTAMotorStop.

```
$( document ).ready(function() {
console.log( "ready!" );
// Calculation of the fingerprint of device
sessionUid=uid;
console.log( "Fingerprint : " +uid);
$("#ImgIoTASSBScreen").click(function(){
saveAsPng();
```

```
});
// When there is a click on the green icon, you need to draw a green circle on the board with the addRectCloud
function with the color "Green"
$("#ImgIoTASSBLedOn").click(function(){
addSSBCloud("IoTALedOn");
});
$("#ImgIoTASSBLedOff").click(function(){
addSSBCloud("IoTALedOff");
});
$("#ImgIoTASSBMotorForward").click(function(){
addSSBCloud("IoTAMotorForward");
});
$("#ImgIoTASSBMotorBackward").click(function(){
addSSBCloud("IoTAMotorBackward");
});
$("#ImgIoTASSBMotorStop").click(function(){
addSSBCloud("IoTAMotorStop");
});
/*
Attention!!
you will need to insert the instructions for the other actions
*/
});


/*
Function used to add action to canvas in the page web and send this to server
in the Ping Project
*/
function addSSBCloud(ssbCommand) {
var np;
// Preparing the Event to be sent to the IoTA Proxy
np = makeEventIota(sessionUid, ssbCommand);
// You draw the circle on the board
drawFormCloud("client_web",np);
// You send the event with the same color to IoTA Proxy
contactServer(np.eventIoTAToJson());
}
```

Lst. 4

In questa versione del sito web è stata inserita una pagina per inviare al database le preferenze per il sito IoTASsb.
La pagina *like.html* prevede cinque Emoji: very bad, bad, neutral, good e very good (figura 9) . Attivando con un click l'icona preferita tra quelle citate s'invia la relativa preferenza: 0, 1, 2, 3 e 4 punti.

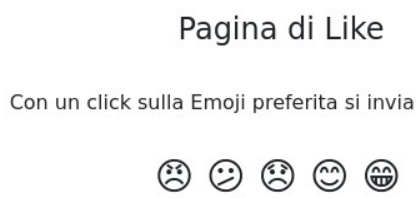

Fig. 9

In quest'ultima pagina la parte di HTML relativa ai cinque pulsanti è la seguente (listato 5):

```
<p class="leftAbout"> <span class="text-important"><h4>Pagina di Like</h4></span><br>
Con un click sulla Emoji preferita si invia la preferenza
<br><br>
<span onclick="like(11,0)" id="likeVeryBad" >&#128544</span> 
<span onclick="like(11,1)" id="likeBad" >&#128533</span> 
<span onclick="like(11,2)" id="likeNeutral" >&#128542</span> 
<span onclick="like(11,3)" id="likeGood" >&#128522</span> 
<span onclick="like(11,4)" id="likeVeryGood" >&#128513</span> 
<br>
<div id="totLike"></div>
</p>
```

Lst. 5

Nel listato si può notare che al click sull'immagine viene chiamata la funzione *like*, esempio *like(11, 1)*. I due parametri sono rispettivamente **l'Id** di gruppo (11) e il valore associato all'immagine (in questo esempio *1* è relativo al *"bad"*).
La funzione like() viene descritta di seguito con il suo codice (listato 6).

```
/*
Function that send the value of like to DataBase
*/
function like(idprogetto, totprj) {
	likeurl= "https://teseo.abaurbino.it/sisteminterattivi/preferenze/preferenze.php";
	dataurl = "?idsession="+sessionUid+"&totprj="+totprj+"&idprogetto="+idprogetto;
	if (sessionUid == null) {
		document.getElementById("totLike").innerHTML = "Warning! Like system is not online";
		return;
	} else {
		var xmlhttp = new XMLHttpRequest();
		completeurl = likeurl+dataurl;
		xmlhttp.open("GET", completeurl , true);
		xmlhttp.send();
		xmlhttp.onreadystatechange = function() {
			if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
				document.getElementById("totLike").innerHTML = xmlhttp.responseText;
			}
		};
	}
}
```

Lst. 6

Gli Id dei gruppi sono definiti nella seguente tabella:

Id	Gruppo
1	Rosso
2	Rosa
3	Viola
4	Azzurro
5	Arancione
6	Marrone
7	Nero
8	Giallo
9	Blu
10	Celeste
11	Grigio
12	Bordeaux