

Sommario

Sommario	1
Introduzione	3
Dispositivi utilizzati	5
GY-521	5
Servo	5
Sensore EMG	6
Elettrodi	7
Pulse Sensor	8
Test dei sensori	9
Divisione dei dispositivi	10
Arduino nano per il rilevamento	11
Arduino uno per l'esecuzione	13
Bluetooth HC-06	13
Servomotori	14
GUI	15
Processing and computer app	15
Sliders	15
Grafico	15
Switch	16
Bpm	17
Roll	17
Termometro	17
Decodifica bluetooth	18
Codice	19
App per smartphone	19
Registrazione e Archiviazione dei dati	21
ESP 8266	21
DHT 11	22
MySQL Database	22
Richieste POST	22
Visualizzazione dei dati	23
Come rendere tutto più compatto	24
Cavo con i sensori	24

Stazione di lettura	24
Alimentazione	25
Conclusioni e sviluppi futuri.....	26
Contenuti aggiuntivi	26
Riferimenti.....	27
Indice delle figure	28
Appendice	29
Appendice A.....	29
Appendice B.....	30
Appendice C.....	31
Appendice D	32
Appendice E	33
Appendice F	34
Appendice G	35
Appendice H	36

Introduzione

Lo scopo del progetto è mostrare come sia possibile controllare una pinza e la sua rotazione attraverso un segnale di alcuni sensori come un sensore inerziale e un sensore per l'elettromiografia. La struttura è molto semplice ed è stata progettata nel pieno della filosofia Maker cercando di sfruttare oggetti di recupero con il solo scopo di mostrare l'idea che c'è dietro. Un esempio in Fig. 1.

La pinza è stata realizzata in tre modi diversi, ma lo stesso progetto può essere realizzato acquistando direttamente una pinza per robot. Una possibilità è quella di realizzarla con due ingranaggi, connettendone uno al servomotore e collegare le due braccia della pinza direttamente agli ingranaggi. Come mostrato in Fig. 2.

L'architettura complessiva del progetto prevede diversi livelli. Si parte da un Arduino Nano che deve leggere i sensori e fa una prima elaborazione del segnale ottenendo alcuni parametri sintetici che invia, attraverso una comunicazione radio a 2,4 GHz ad un Arduino Uno a cui sono collegati i due servo che muovo la pinza.

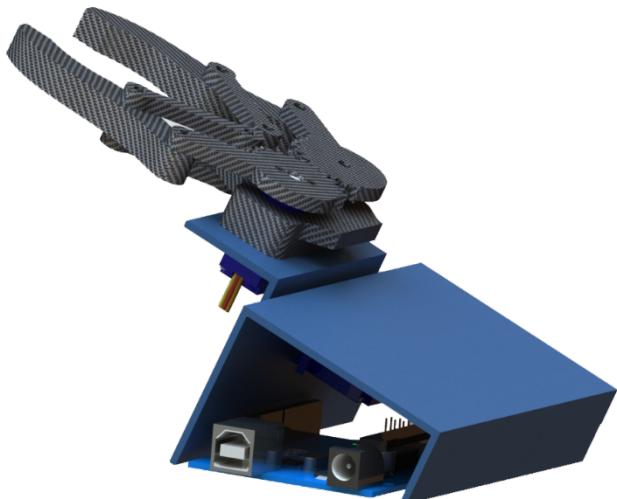


Fig. 1 – Struttura di esempio della pinza per il controllo di apertura e chiusura e della rotazione

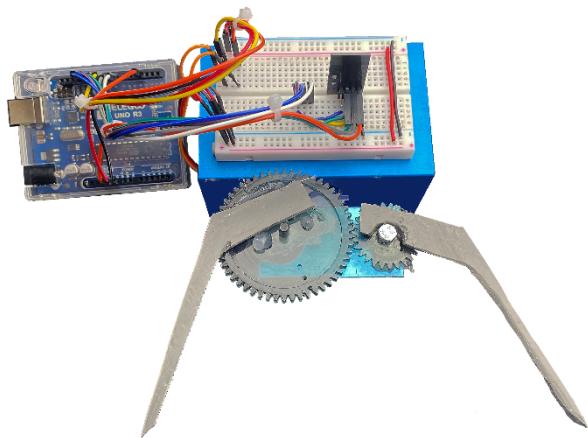
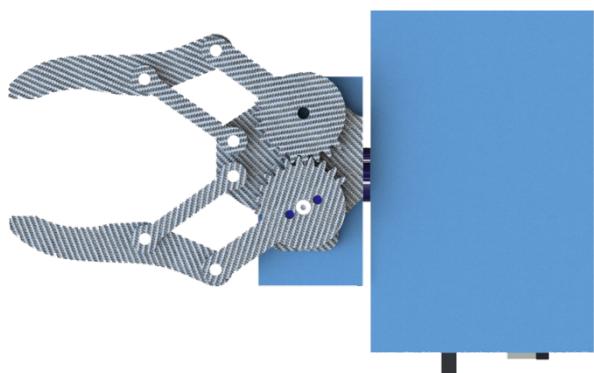


Fig. 2 – Esempio di struttura realizzata con gli ingranaggi, plastica sagomata e una base di alluminio anodizzato.

L'Arduino Uno provvede anche a comunicare via Bluetooth sia con dispositivi mobile che con il computer.

Inoltre, è stato aggiunto un modulo, un ESP 8266 che permette di raccogliere i dati ed inviarli su un server sotto un dominio di proprietà attraverso il quale è possibile salvare i dati e visualizzarli da remoto.

Sono stati utilizzati un sensore inerziale per leggere l'inclinazione del polso e controllare la rotazione della pinza, tale sensore include anche un sensore di temperatura che è stato sfruttato con l'idea di raccogliere la temperatura dell'utente. Inoltre, è stato utilizzato anche un sensore EMG per misurare l'attività



muscolare in modo da regolare la chiusura o l'apertura della pinza.

Proponiamo di seguito le altre due modalità di costruzione della pinza. Una possibile realizzazione della pinza può essere effettuata utilizzando il compensato Fig. 3. Per la progettazione si parte dal disegno dei vari componenti sul compensato, si procede poi con il taglio e infine l'assemblamento. Sopra la base che supporta la pinza è stato montato l'Arduino Uno con il ricetrasmettitore, il modulo bluetooth e i due servo che controllano la rotazione e l'apertura- chiusura della pinza.

Anche in questo caso uno degli ingranaggi è stato collegato con un servo mentre la base che contiene la pinza e ne permette la rotazione è stata collegata all'altro servo.

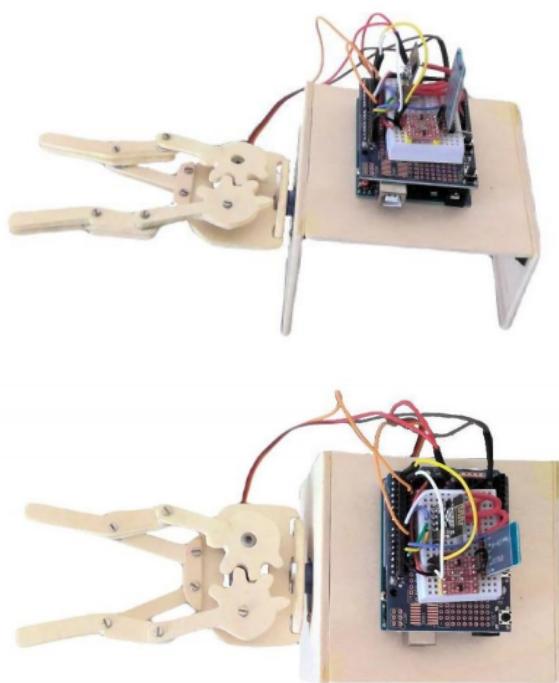


Fig. 3 – Modello della pinza in compensato

Dispositivi utilizzati

GY-521

Il GY-521 è un modulo che permette di includere un sensore MPU-6050 e un sensore di temperatura. MPU-6050 è un chip prodotto da Invesense [1], è un'unità di misura inerziale (IMU) basata sulla tecnologia MEMS che contiene un accelerometro a tre assi e un giroscopio a tre assi all'interno di un singolo chip. Questo sistema a 6 gradi di libertà (6 DOF) fornisce quindi 6 valori in uscita.

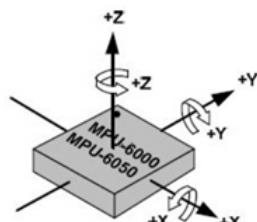


Fig. 4 – Orientazione degli assi per il sensore inerziale.

L'accelerometro è in grado di rilevare l'accelerazione di un corpo lungo il proprio asse, mentre il giroscopio è in grado di rilevare la velocità angolare di un corpo su di un proprio asse. Il modulo GY-521 ha piccole dimensioni ma è molto preciso, contiene un chip con un convertitore analogico-digitale a 16 bit integrato per ogni canale e questo permette di catturare i canali X, Y e Z contemporaneamente. Il range di misura dell'accelerometro è +2, +4, +8, +16g, mentre il range di misura del giroscopio è ±250, 500, 1000 e 2000 °/s. Necessita inoltre, di una alimentazione da 3V a 5V.

Questo sensore utilizza il protocollo di comunicazione standard I²C, in particolare per la comunicazione è necessario importare due librerie una per l'I²C e una per la comunicazione con l'MPU-5060.

```
#include <MPU6050_tockn.h>
MPU6050 mpu6050(Wire);
#include <Wire.h>
```

Servo

Un Servomotore è un attuatore rotativo o lineare che consente un controllo preciso della posizione angolare o lineare, della velocità e dell'accelerazione.

È costituito da un opportuno motore accoppiato ad un sensore per il feedback di posizione. Richiede anche un controller relativamente sofisticato che spesso consiste in un modulo dedicato, progettato specificatamente per l'uso con i servomotori.

In questo progetto è stato usato il Microservo SG90. Può ruotare approssimativamente di 180° (90° per entrambe le direzioni) e funziona esattamente come i servomotori standard ha solo delle dimensioni ridotte.

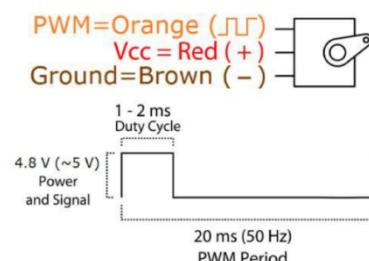


Fig. 5 – Collegamento dei pin del servo.

La posizione "0" (impulso di 1,5 ms) è al centro, "90" (impulso di ~ 2 ms) è tutto a destra, "-90" (impulso di ~ 1 ms) è tutto a sinistra.

Per gestire facilmente il controllo PWM viene utilizzata la libreria Servo.

```
#include <Servo.h>
Servo servo-object-name;
```

Invece, per controllare il servo è stato utilizzato il seguente comando:

```
servo-object-name.write(degrees);
```

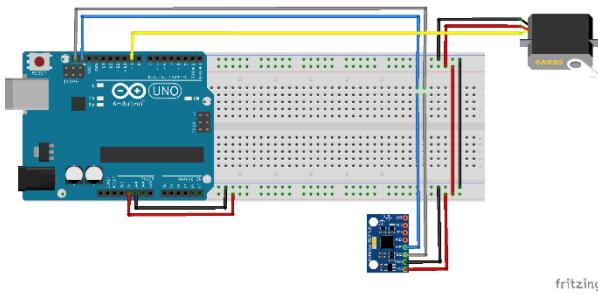


Fig. 6 – Schema di collegamento del servo e dell'IMU (sensore inerziale).

Sensore EMG

Il sensore per l'EMG, dove EMG sta per Elettromiografia [2], è un sensore in grado di misurare l'attività muscolare monitorando il potenziale elettrico generato dalle cellule muscolari. L'elettromiografia nasce per la ricerca medica e per la diagnosi dei disturbi neuromuscolari. Tuttavia, i sensori per l'EMG oggi vengono utilizzati nella robotica e in altri sistemi di controllo.

L'EMG valuta lo stato di salute dei muscoli e delle cellule (i motoneuroni) che li controllano. Queste cellule trasmettono segnali elettrici che provocano la contrazione e il rilassamento muscolare. Un EMG traduce questi segnali in grafici o dati e aiuta i medici ad eseguire una diagnosi.

Questo sensore amplifica ed elabora l'attività elettrica di un muscolo e lo converte in un segnale analogico che può essere letto da qualsiasi microcontrollore con un ADC, come Arduino.

Quando il muscolo sottoposto a misurazione si contrae, la tensione di uscita del sensore aumenta.

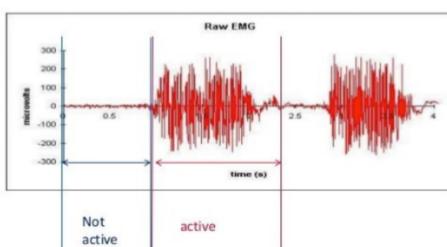


Fig. 7 – Segnale grezzo EMG prima di essere filtrato, rettificato e smoothed [3].

Collegato al sensore, tramite un connettore jack da 3,5 mm, c'è un cavo che da un lato è collegato ad Arduino, mentre dall'altro lato termina con tre connettori a clip che si attaccano a tre elettrodi.

In questo progetto è stato usato il Muscle Sensor V3 [4]. Questo sensore necessita di una tensione di riferimento positiva ed una negativa, quindi sono richieste due batterie da 9V.

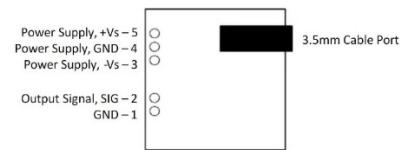


Fig. 8 – Schema di collegamento dei pin del sensore.

L'Output Signal richiede un pin analogico, in particolare è stato utilizzato il pin A0.

L'EMG di superficie può essere registrato da una coppia di elettrodi o da una matrice più complessa di elettrodi. È necessario più di un elettrodo perché le registrazioni dell'EMG mostrano la differenza di potenziale tra due elettrodi separati.

Come si può leggere nel datasheet o nello schema elettrico, il segnale dei due elettrodi entra in un amplificatore per strumentazione:

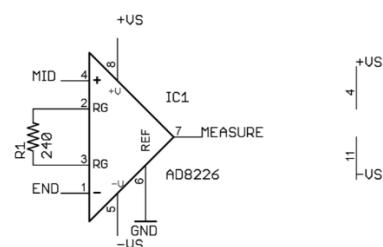


Fig. 9 – Schema dell'amplificatore per strumentazione.

Successivamente il segnale viene rettificato:

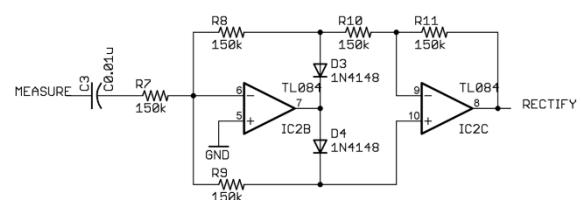


Fig. 10 – Raddrizzatore a doppia semionda che permette di rettificare il segnale amplificato.

Alla fine, il segnale viene ripulito attenuando rumori generati da artefatti ambientali, elettrici, elettronici, fisiologici ecc.

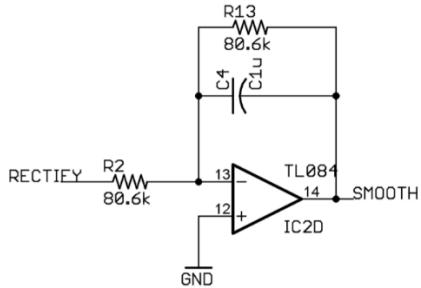


Fig. 11 – Smoothing, filtro attivo passa basso.

In sintesi, si ottiene un segnale proporzionale alla contrazione muscolare che verrà inviato all'ADC di Arduino.

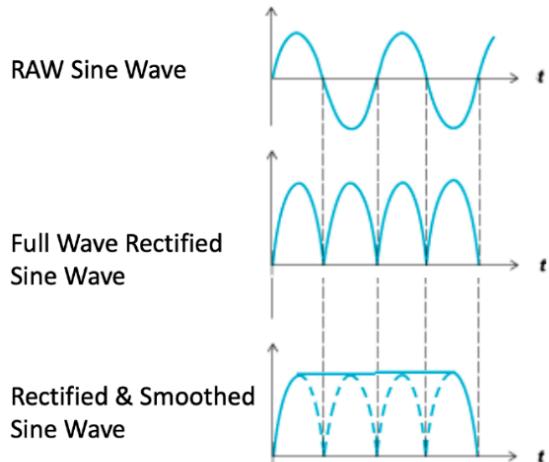


Fig. 12 – Raw EMG vs EMG dopo il raddrizzatore a doppia semionda e lo smoothing.

Il Muscle Sensor V3 è fuori produzione, quindi per il progetto è stata acquistata una copia del sensore. Questa copia risulta più economica tuttavia presenta alcuni difetti. Il sensore, infatti emette un segnale senza livellamento e pieno di disturbi.



Fig. 13 – Copia del Muscle Sensor V3 utilizzata nel progetto.

Elettrodi

L'attività elettrica di un muscolo viene rilevata con l'ausilio degli elettrodi per l'EMG. Ci sono differenti tipi di elettrodi, in questo progetto sono stati utilizzati gli elettrodi di superficie, in particolare quelli in gel [5, 6]. Questo tipo di elettrodi contiene un gel elettrolitico come interfaccia tra gli elettrodi e la pelle.

Gli elettrodi gelificati forniscono una tecnica non invasiva per la misurazione e il rilevamento del segnale EMG, ma sono generalmente utilizzati per i muscoli di superficie poiché vengono applicati sulla pelle.

Gli elettrodi devono essere applicati in modo appropriato, in particolare, due elettrodi devono essere posizionati tra l'unità motoria e l'inserzione tendinea del muscolo, lungo la linea longitudinale del muscolo. La distanza tra il centro dei due elettrodi dovrebbe essere di 1-2 cm.

Il connettore a clip rosso va posizionato sull'elettrodo al centro del corpo muscolare, quello verde ad un'estremità del corpo muscolare e infine, il terzo connettore (giallo) deve essere posizionato su una sezione inattiva del corpo.

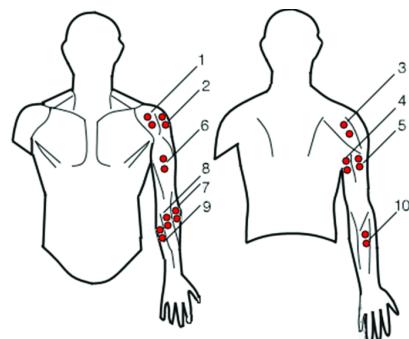


Fig. 14 – Posizionamento degli elettrodi.

È importante fare alcune considerazioni sui segnali EMG.

La fonte elettrica è il potenziale di membrana muscolare di circa -90 mV. La frequenza di ripetizione tipica dell'impulso dell'unità muscolo-motoria è di circa 7-20 Hz, a seconda

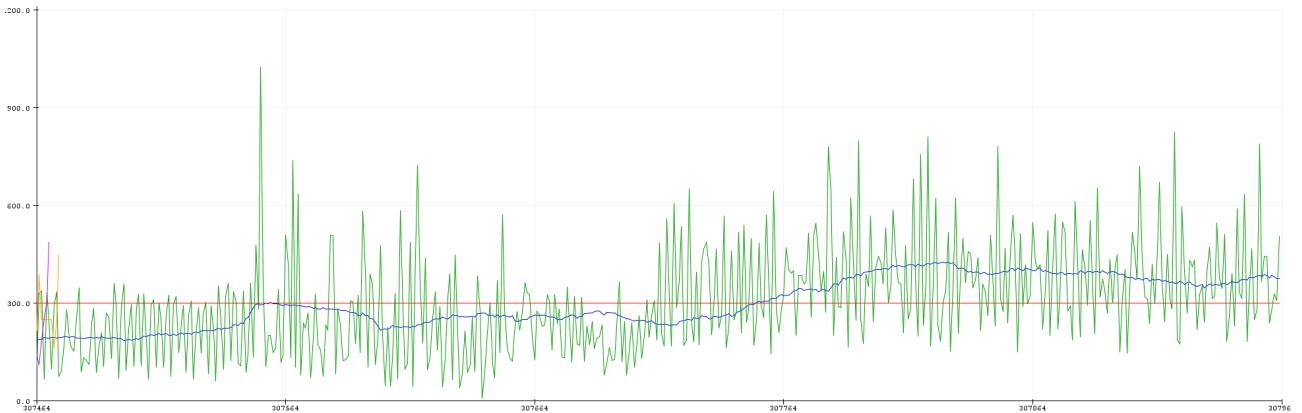


Fig. 15 – Segnale smoothed (blu), segnale originale (verde), soglia (rosso). Quando il segnale è al di sotto del valore soglia l'arto è rilassato, mentre, quando il segnale è al di sopra della soglia il muscolo è contratto.

della dimensione del muscolo i potenziali misurati variano tra meno di 50 μV fino a 30 mV. Questo significa che il segnale deve essere amplificato come è stato mostrato nel paragrafo precedente.

A causa della bassa qualità del circuito EMG, il segnale analogico acquisito è risultato pieno di rumore. La tensione è stata solo rettificata ma non molto livellata, quindi è stato utilizzato un semplice algoritmo di livellamento riportato in Appendice A.

Quanto riportato in Arduino's reference smoothing script [7] è lo script di smoothing di riferimento di Arduino che è stato utilizzato apportando piccole modifiche dopo un lungo periodo di test. In Fig. 15 si può osservare il segnale nel Plotter seriale di Arduino, in cui sono stati riportati il segnale grezzo, un valore soglia e il segnale ripulito.

Questa media livellata consente di personalizzare il codice riportato in Appendice B per controllare il servo che permette l'apertura e la chiusura della pinza.

Pulse Sensor

Il principio di questo sensore è quello di puntare una luce tra le dita, la quale viene assorbita o riflessa e una parte passerà attraverso i tessuti più sottili. Quando il sangue viene pompato attraverso il corpo, viene schiacciato nei tessuti capillari e il volume di quei tessuti

aumenta leggermente, quindi tra i battiti del cuore il volume diminuisce. La variazione del volume influisce sulla quantità di luce che verrà trasmessa. Questa fluttuazione è molto piccola ma può essere percepita con l'elettronica.

Si inizia con una sorgente e un rilevatore di luce. Si utilizzano un LED a infrarossi e un fotodiodo. La luce infrarossa viene assorbita dal sangue e riflessa dai tessuti circostanti, tale assorbimento è dovuto alla presenza dell'emoglobina nel sangue [8]. È importante che i due dispositivi siano accoppiati correttamente, così che la lunghezza d'onda della luce emessa dal LED venga rilevata dal fotodiodo. Il fotodiodo consiste di una giunzione che genera una piccola corrente quando viene bombardato con i fotonni.

A questo punto è necessario amplificare il segnale che esce dal fotodiodo e questo è possibile grazie ad un circuito che è il Convertitore corrente tensione. In Fig. 16 è riportato lo schema circuitale completo del sensore.

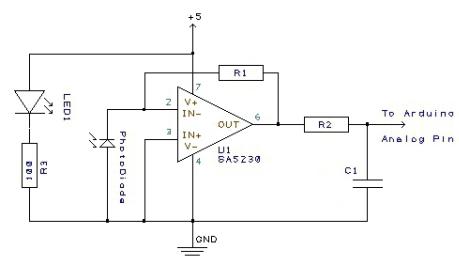


Fig. 16 – Schema circuitale del Pulse Sensor [9].

Il fotodiodo deve essere schermato dalla luce ambientale che genera molto rumore nel segnale. Infatti, come si può vedere in Fig. 18 il segnale del sensore è distorto anche da piccoli movimenti. Inoltre, qualsiasi disallineamento o movimento del LED a infrarossi confonderà il segnale.



Fig. 17 – Immagine del sensore. Il cavo rosso corrisponde ai 5V, il cavo nero al GND e il cavo viola è il segnale.

Pulsesensor.com produce un sensore [10] che è abbastanza piccolo da poter essere indosso comodamente in diverse configurazioni ed è immune al rumore. È costituito da un circuito integrato di dimensioni ridotte che ha combinati al suo interno un fotodiodo e un circuito op-amp. Le dimensioni ridotte hanno come risvolto il fatto che è possibile mantenere un contatto stretto e costante con la pelle. Questo di fatto protegge il sensore dai cambiamenti della luce ambientale e riduce al minimo i rumori dovuti al movimento.

Il segnale grezzo è quello in Fig. 18.

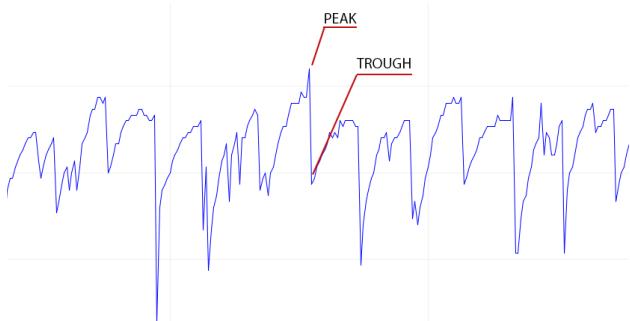


Fig. 18 – Picco e depressione del segnale grezzo.

Per le prime prove con il sensore su Arduino è stata utilizzata la libreria *PulseSensor Playground*, tuttavia questa non è risultata

completamente compatibile con le altre funzionalità che sono state implementate successivamente. Quindi è stata eliminata la libreria ed è stato scritto un codice apposito.

La lettura è richiamata dalla funzione *battiti()* descritta nella libreria *funzioniBPM.h*.

Sono state messe insieme idee prese da Instructables [11] e da Pulsesensor [10] per ottenere un codice che si occupi di leggere il valore analogico e identificare il battito al di sopra di una certa soglia. Il codice è relativamente complesso. Oltre ad una parte deputata a stampare i simboli o una frase affermativa quando un battito viene rilevato, il Timer 2 viene utilizzato per impostare un interrupt ogni 2 ms. Una parte di codice più complessa si occupa di misurare il tempo trascorso dall'ultimo battito e di fare campioni appropriati per evitare il rumore. In particolare, si cerca di evitare il rumore dicrotico e di rilevare i picchi. Quando viene rilevato un battito (un picco conforme alle condizioni imposte) il valore del tempo viene aggiornato e vengono calcolati i BPM e il numero di battiti che possono verificarsi in 1 minuto (60000 ms).

Le funzioni primarie sono svolte da *ISR(TIMER2_COMPA_vect)* e *battiti()* e *serialOutputWhenBeatHappens()* che si occupa di restituire il valore del battito.

Test dei sensori

Nella Fig. 19 è riportato il circuito che mostra la configurazione del test per lo sviluppo del codice iniziale.

Questa configurazione in Fig. 19, con due batterie da 9 V consente di avere +9 e -9 V sui terminali Vs del sensore per l'EMG. È anche possibile utilizzare un alimentatore da 18 V DC e dividerlo in +9 e -9.

Questa configurazione è stata utilizzata per eseguire tutti i test dei dispositivi e del codice. È stato, in seguito, rielaborato parte del codice

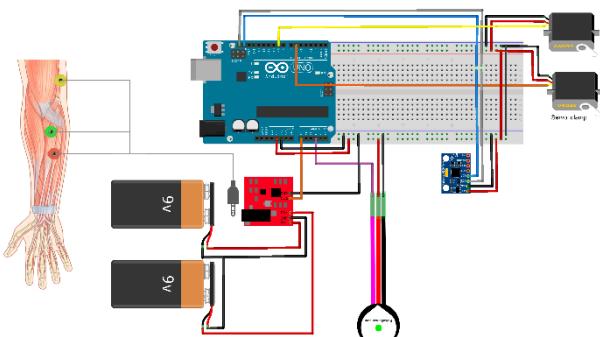


Fig. 19 – Circuito con i diversi sensori utilizzati.

suddividendolo nei vari moduli in modo da avere un sistema più modulare che permettesse di aggiungere e rimuovere i dispositivi a seconda delle necessità. Da questa configurazione di partenza, sono stati suddivisi i diversi dispositivi in una struttura determinata per il monitoraggio e una struttura con gli attuatori.

Alla fine, i sensori che devono venire a contatto con il braccio e il polso sono stati uniti in un unico cavo dotato di connettore per collegarlo alla struttura di monitoraggio.

Tutto è stato quindi riorganizzato in modo più compatto.

Divisione dei dispositivi

Come riportato nel paragrafo precedente dopo una serie di prove con i diversi sensori i dispositivi utilizzati sono stati separati e i codici sono stati riorganizzati. In particolare, è stato sfruttato un Arduino Nano per la lettura dei sensori e un Arduino Uno per controllare i servomotori.

La comunicazione tra i due Arduino avviene tramite radio utilizzando due ricestrasmittitori NRF24L01 [12] da 2.4 GHz.



Fig. 20 – Modulo NRF24L01 e schema di collegamento dei pin.

Questo dispositivo ha una tensione operativa di 1.9V ÷ 3.6V, un'antenna di 2,4 GHz incorporata e una velocità operativa (massima) di 2Mbps. Il chip è gestito dall'interfaccia SPI. Tramite questa interfaccia è possibile accedere ai registri del chip e modificarne tutti i parametri.

Questa operazione complessa, però, è facilitata dall'utilizzo delle librerie.

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
```

Il collegamento del modulo richiede l'utilizzo di diversi pin:

- GND
- Vcc, il modulo utilizza 3.3V
- CE, Chip Enable, utilizzato per abilitare la comunicazione SPI
- CSN, Chip Select Not, un pin che deve essere sempre al livello alto altrimenti disabilita l'SPI
- SCK, Serial Clock, fornisce l'impulso di clock con il quale funziona la comunicazione SPI
- MOSI, Master Out Slave In, collegato al pin MOSI di Arduino, serve per il modulo, per ricevere dati da Arduino
- MISO, Master In Slave Out, collegato al pin MISO di Arduino, serve per il modulo, per inviare dati da Arduino.
- IRQ, è un pin basso attivo, viene utilizzato solo se è richiesto l'interrupt.

Per utilizzare questo modulo è necessario impostare la lettura o la scrittura nel *setup()* come segue:

```

radio.begin();
radio.openWritingPipe(address); //for receiving
//radio.openWritingPipe(address); // for trasmitting
radio.setPALevel(RF24_PA_MIN);
radio.stopListening(); //for receiving
//radio.stopListening(); // for trasmitting

```

È anche importante definire i pin CE e CSN.

Per la trasmissione è stata utilizzata una funzione *charfortransmission()* creata ad hoc che permette di concatenare una lettera che indica quali dati vengono trasmessi e il valore assunto dalla variabile corrispondente. Questa funzione viene ripetuta nel *loop()* del codice principale e viene continuamente chiamata per inviarle parametri diversi a seconda della variabile. All'interno della funzione i parametri da inviare devono essere prima convertiti in *char* per consentire la corretta trasmissione con le librerie utilizzate.

Il ricevitore ha una funzione simile, *radianuovo()*, che per prima cosa analizza il primo carattere identificando quali informazioni vengono ricevute e inserisce i dati nelle variabili corrispondenti dopo aver effettuato le opportune conversioni sul tipo di variabile.

Arduino nano per il rilevamento

Si può osservare in Fig. 21 la configurazione dell'Arduino Nano con i sensori sulla breadboard per la fase di test.

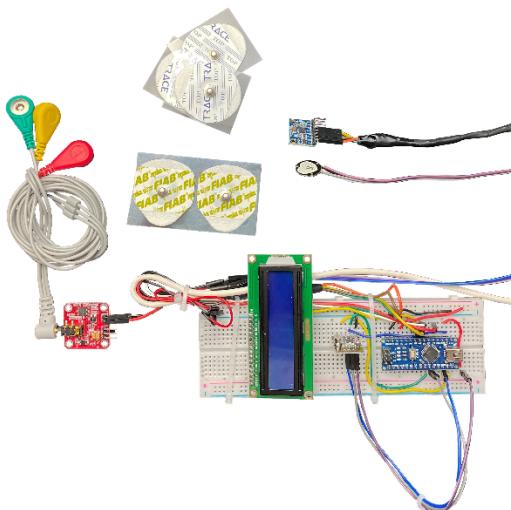


Fig. 21 – Disposizione dei sensori e dell'Arduino nano sulla breadboard.

Per l'Arduino Nano dedicato ai sensori, il circuito è stato modificato aggiungendo un display LCD per monitorare i parametri oltre al modulo NRF24L01 per la trasmissione radio. Il display LCD è collegato ad un modulo di interfaccia I²C per semplificare la comunicazione e la gestione dei cavi. Si può anche fare a meno del modulo di interfaccia I²C andando a collegare il display ai rimanenti pin dell'Arduino Nano, in tal caso saranno necessari 6 pin digitali, in particolare in questa configurazione possono essere utilizzati i pin (9, 10, 2, 3, 4, 5).

Per il resto la connessione dei sensori è rimasta invariata.

Come già riportato nei paragrafi precedenti l'utilizzo dell'Arduino Nano permette di rendere il tutto più compatto. La disposizione dei pin dei diversi sensori rimane la stessa, l'unica accortezza riguarda il fatto che non ci sono i pin dedicati SDA e SCL ma sono sui pin A4 e A5 come è possibile osservare in Fig. 22.

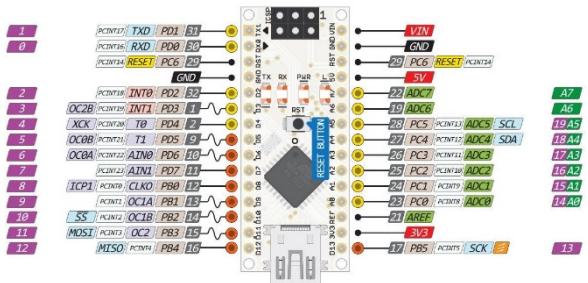


Fig. 22 – Disposizione e definizione dei pin su Arduino Nano [13].

Nel codice relativo a questa parte che coinvolge i diversi sensori la sezione relativa ai servomotori è stata rimossa e il resto è stato riorganizzato.

Il codice principale è riportato in Appendice C.

In particolare, il codice principale è stato semplificato aggiungendo diverse funzioni che chiamano definizioni di librerie esterne:

averagecalc() è stata definita in *EM-Gsmooth.h* e viene utilizzata per calcolare la media del segnale distorto.

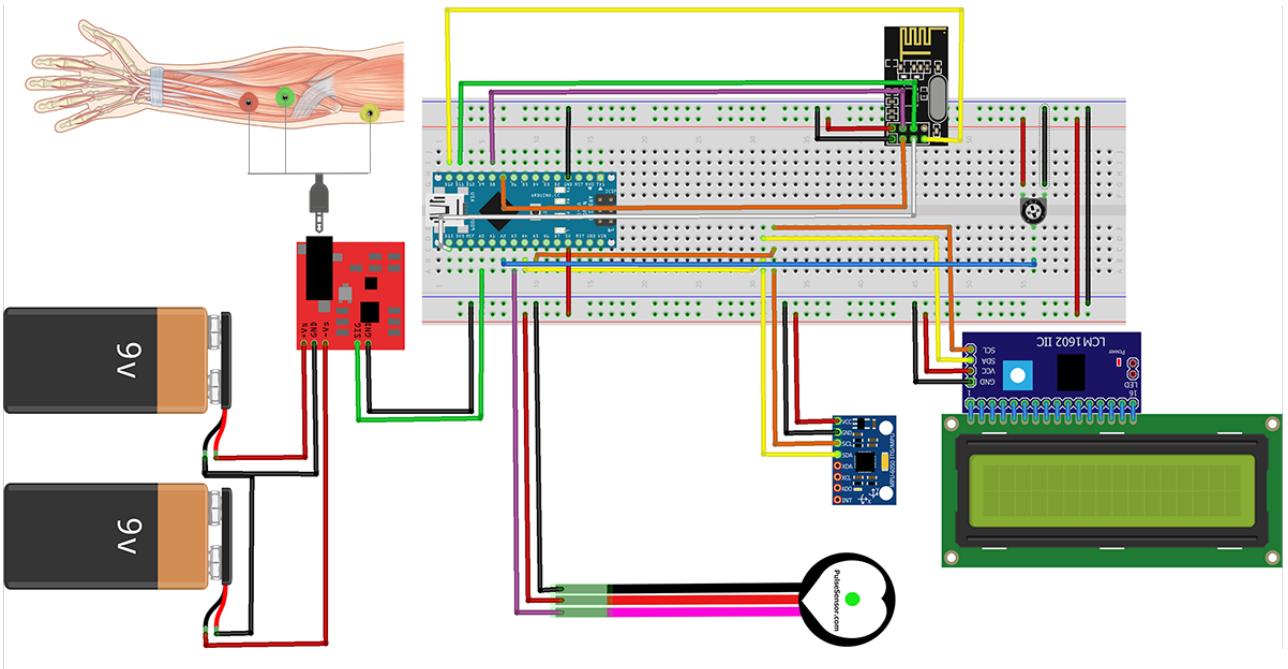


Fig. 23 – Schema completo dei sensori.

printatore(LOW) è una funzione speciale che può essere usata per stampare il segnale EMG per il plotter seriale in modo da eseguire il debug di eventuali problemi di rilevamento. Di solito viene utilizzato per stampare tutti i valori sul monitor seriale. È definito in *printerfuzioni.h*

lcdprint() viene utilizzata per stampare i diversi valori e i relativi simboli sul display LCD.

battiti() è usato per rilevare i battiti cardiaci. È stato definito nelle funzioni *funzioniBPM.h*

charfortransmission() per la trasmissione dei dati con le relative informazioni.

Per mantenere il codice chiaro queste funzioni sono state definite in librerie esterne. La libreria *EMGsmooth.h* è responsabile del livellamento del segnale EMG al fine di ottenere precisione nella determinazione della soglia. La libreria *funzioniBPM.h* è già stata descritta nel paragrafo del sensore di battito. Il *printerfuzioni.h* contiene la funzione per stampare nel monitor seriale e nel display lcd, contiene anche la definizione del simbolo freccia e cuore sul display LCD.

Per creare caratteri speciali per LCD viene utilizzato LCD Custom Generator [14] che permette di generare un byte personalizzato con la definizione dei pixel ON ed OFF. Questo consente di usare simboli speciali oltre ai caratteri ASCII definendo individualmente i pixel di attivazione e disattivazione in una matrice 5x8 (*byte_matrix*). È possibile creare fino a 8 simboli diversi con la funzione *createChar()* della libreria LiquidCrystal e stamparli con *lcd.Write()*. Ad esempio, per il simbolo del cuore Fig. 24.

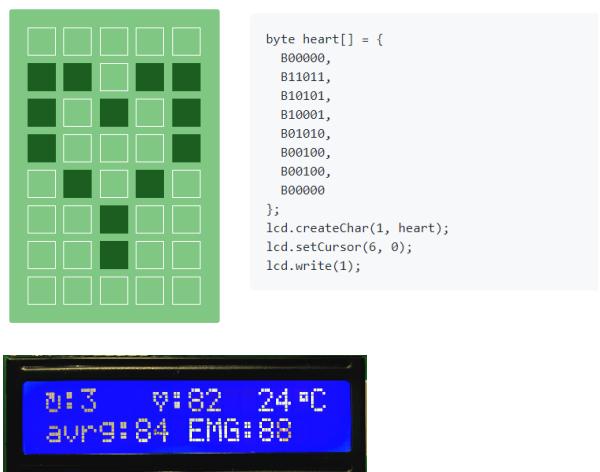


Fig. 24 - Creazione di caratteri speciali e disposizione dei valori sul display.

Ogni libreria contiene tutte le variabili e tutte le altre librerie necessarie al suo funzionamento.

Sono stati, quindi, ottenuti i seguenti valori che devono essere inviati all'Arduino con gli attuatori:

- tempmpu
- angolo
- media
- BPM
- soglia

Infine, è stato aggiunto anche un trimmer per controllare manualmente la soglia sul pin analogico A2.

Arduino uno per l'esecuzione

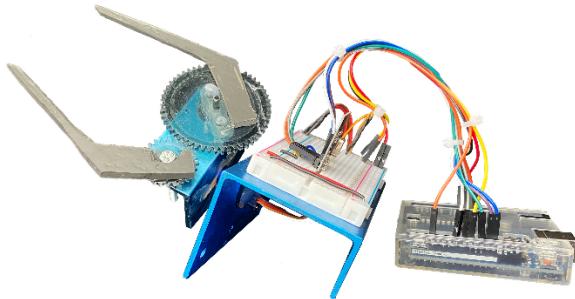


Fig. 25 – Configurazione dell'Arduino uno con i servo, il ricevitore e la pinza.

Questa seconda struttura si occupa di ricevere i segnali radio e ritrasmetterli via bluetooth (o via seriale) e di far muovere i due servomotori. È importante osservare che per aggiungere il modulo bluetooth, che verrà discusso nel paragrafo successivo, è stato utilizzato un convertitore di livello logico (Fig. 26) 3,3V – 5V TTL.

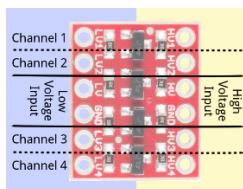


Fig. 26 – Convertitore di livello logico.

Bluetooth HC-06

Il modulo hc-06 riportato in Fig. 27 è un dispositivo che permette di trasformare un segnale sul bus seriale in un segnale bluetooth.



Fig. 27 – Modulo Bluetooth.

Tipicamente viene utilizzato un protocollo seriale software per comunicare, tuttavia in questo caso è stato utilizzato direttamente il bus UART sui pin TX e RX. Quindi, sono state create due nuove funzioni *radioperbt()* e *decodserial()*. La prima analizza il carattere ricevuto dall'NRF24L01 e a seconda della lettera ricevuta aggiorna la rispettiva variabile e contemporaneamente stampa il valore codificato nella seriale (quindi anche nel bluetooth). La seconda funzione, invece si occupa di analizzare i comandi ricevuti. Questa seconda parte verrà discussa meglio nella sezione GUI.

radioperbt() è stata originariamente scritta come una funzione per decodificare il messaggio radio e aggiornare le variabili (*radianuovo()*). Successivamente è stata aggiornata per consentire la comunicazione bluetooth inviando il messaggio codificato con la prima lettera che indica quale variabile corrisponde al dato.

```
if (text[0] == 'T') {
    Serial.print("T");
    valore = "";
    for (int i = 1; i < 32; i++) {
        valore = valore + String(text[i]);
    }
    tempmpu = valore.toFloat();
    Serial.println(tempmpu);
}
```

La logica è quella di ricevere un *array chart*, analizzare il primo carattere e convertire tutti gli altri nelle variabili di interesse (*float* o *int*) e stampare nuovamente il messaggio come una Stringa codificata.

Servomotori

Per controllare i servomotori sono state create due funzioni.

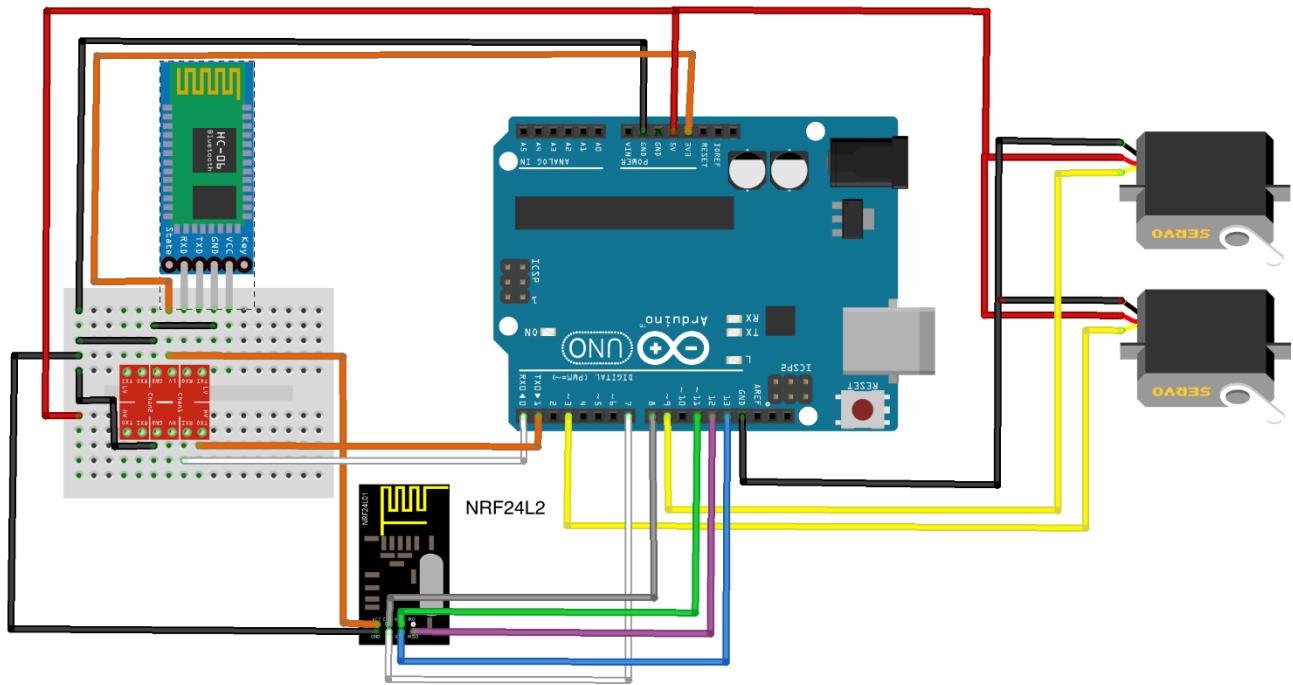
pos_servo() è la funzione a cui viene passata l'inclinazione e si occupa di rimapparla correttamente per ruotare la struttura.

pinzacontrol() si occupa di far chiudere o meno la pinza in base al valore di soglia. C'è un ulteriore controllo prima di riaprire la pinza per evitare problemi dovuti alla fluttuazione del segnale.

```

if (average > soglia) {
    servopinza.write(pinzamax);// chiudo la pinza
}
if (millis() - t3 > t4) {
    if (average < soglia) {
        controllo[i] = LOW;
    }
    else {
        controllo[i] = HIGH;
    }
    if (i < 10) {
        i++;
    }
    t3 = millis();
}
for (int j = 0; j < 10; j++) {
    if (controllo[j] == LOW) {
        stato = LOW;
    }
    else {
        stato = HIGH;
    }
}
if (stato == LOW) {
    servopinza.write(pinzamin);// apriamo la pinza
}

```



fritzing

Fig. 28 – Arduino Uno con i servo, il convertitore di livello logico, il modulo NRF24 e il modulo bluetooth.

GUI

Nel progetto è stata sviluppata anche un'interfaccia grafica che consente di visualizzare i dati trasmessi e di prendere il controllo della pinza, sia dal computer che dallo smartphone.

Processing and computer app

Per sviluppare l'interfaccia grafica sul computer è stato utilizzato Processing, compatibile con MacOs, Windows e Linux.

È riportata di seguito una breve descrizione dei vari elementi che costituiscono la finestra grafica.

Sliders

Per creare gli slider non sono state importate librerie, infatti gli elementi che creano gli slider sono stati disegnati con un programma di grafica e poi importate come immagini nella cartella del programma.

```
barra1 = loadImage("barra1.png");
indicatore1 = loadImage("indicatore1.png");
barra2 = loadImage("barra2.png");
indicatore2 = loadImage("indicatore2.png");
```

Una volta caricate le immagini nel codice, nella sezione *draw* sono state incollate nella finestra grafica.

```
image(barra1, x1, y1);
image(indicatore1, x2, y2);
image(barra2, x1, y3);
image(indicatore2, x4, y4);
```

Quando il mouse è premuto e il cursore si muove lungo la coordinata x per la lunghezza dello slider, l'indicatore seguirà il mouse simulando la funzione dello slider. Lo stesso per il secondo slider.

Le prime prove sono state effettuate utilizzando la libreria Firmata, alla fine questa libreria non è più stata utilizzata ma è stato inviato il comando con l'angolo di apertura o chiusura della pinza mappando i valori nello slider.

Grafico

Per poter visualizzare i dati provenienti dall'EMG è stato creato un grafico in cui è possibile visualizzare il trend di valori registrati.

La costruzione del grafico parte dal disegno degli assi, questo avviene tramite la funzione *line(x1, y1, x2, y2)* i cui valori indicano la posizione in cui è stato collocato il grafico e

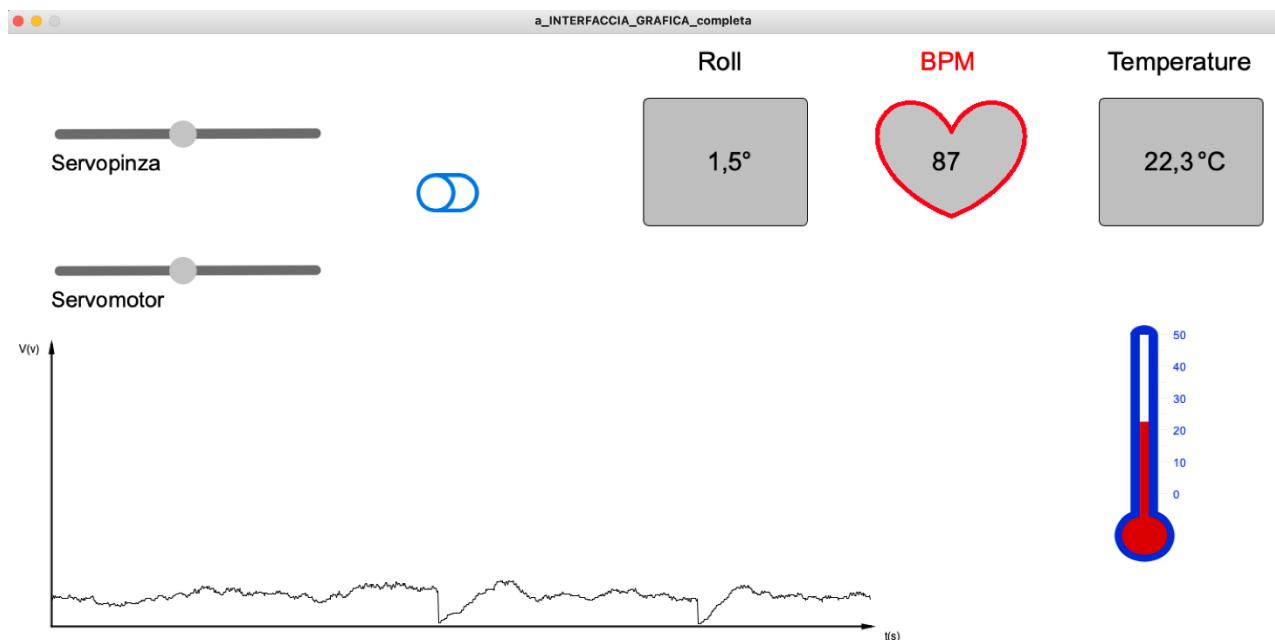


Fig. 29 – Interfaccia grafica Processing

anche la lunghezza degli assi; l'asse x è il tempo mentre l'asse y è l'ampiezza V delle oscillazioni dell'EMG.

All'interno di *setdati()* è stato definito un vettore di 1000 elementi e attraverso un ciclo for si accede agli elementi del vettore che sono i valori dell'EMG mappati tra 0 e 300 (l'altezza dell'asse y). È stato definito poi un altro vettore che contiene lo stesso numero di elementi del precedente e di nuovo si accede agli elementi tramite un ciclo for. È stata definita una variabile *float* chiamata *passo* che corrisponde alla lunghezza dell'asse x diviso il numero di campioni (*data.length*). Il valore assunto dall'elemento i-esimo del vettore *datax[i]* corrisponde a $50 + i \cdot \text{passo}$ e questo vettore permette di individuare la coordinata x per costruire la linea che rappresenta l'andamento dell'EMG.

```
public class Grafico {
    public float[] datax;
    public Grafico(){}
    public void setdati(){
        data = new float[1000];
        for (int i=0; i < data.length; i++) {
            data[i] = 100;
        }
        datax = new float[data.length];
        for (int i=0; i < datax.length; i++) {
            float passo=900/float(data.length);
            datax[i] = 50+i*passo;
        }
    }
}
```

Una volta che è stata definita la relazione tra i dati ricevuti e l'asse x è possibile disegnare la linea tra due punti consecutivi. Questo è stato realizzato tramite la funzione *line(datax[i], 650-data[i], datax[i+1], 650-data[i+1])*.

Una volta che la curva raggiunge la fine dell'asse x, ricomincia dall'origine. La direzione degli assi è indicata dalla punta della freccia che è stata creata tramite la funzione triangolo composta da 6 valori che rappresentano le coordinate dei tre vertici del triangolo.

```
public void assi(){
    for (int i=0; i<data.length-1; i++) {
        fill(55, 200, 50);
        line(datax[i], 650-data[i], datax[i+1], 650-data[i+1]);
    }

    stroke(0, 0, 250);
    strokeWeight(2);
    //line(50,800-V,950,800-V);
    stroke(0, 0, 0);
    line(50, 350, 50, 650);
    line(50, 650, 950, 650);
    fill(0, 0, 0);
    triangle(48, 350, 52, 350, 50, 340);
    triangle(940, 648, 940, 652, 950, 650);
    textSize(12);
    text("t(s)", 965, 665);
    text("V(v)", 15, 350);
}
```

Switch

Lo switch è stato aggiunto per commutare i comandi, così da rendere possibile il controllo della pinza tramite gli slider presenti nell'interfaccia grafica.

Lo switch è stato creato prendendo spunto dagli esempi della libreria Cp5 che è stata importata nel codice.

```
import controlP5.*;
ControlP5 cp5;
```

Nella classe *Serialcomunication* è stata creata la funzione *serialsend()* in cui se il controllo manuale risulta attivo (*if (pulsante == true)*), se il mouse è premuto e si trova in corrispondenza di uno dei due slider, in base a quale slider verrà attivato la pinza potrà aprirsi/chiudersi oppure ruotare e questo risulta possibile mappando la posizione sullo slider in un range compreso tra 0° e 180° per la pinza e tra -90° e 90° per la rotazione. Questo valore viene in seguito scritto, come *Stringa* insieme ad un carattere identificativo, sulla porta seriale.

```

...
void serialsend() {
    if (mousePressed) {
        if (millis()-ti>500) {
            if (pulsante==true) {
                if (mousePressed && mouseY < y2+20) {
                    float pinzamap=map(target1, 50, 350, 0, 180);
                    String pinza="P"+str(pinzamap);
                    MyPort.write(pinza);
                    println(pinza);
                }
            }
            ti=millis();
        }
        if (millis()-ti2>500) {
            if (pulsante==true) {
                if (mousePressed && mouseY > y4-20) {
                    float rotazmap=map(target2, 50, 350, -90, 90);
                    String rotaz="R"+ str(rotazmap);
                    MyPort.write(rotaz);
                    println(rotaz);
                }
            }
            ti2=millis();
        }
    }
}
...

```

Cambiando lo stato dello switch, che di base è stato impostato con il valore *false*, si troverà quindi nello stato *true* e contemporaneamente verrà inviata la stringa "C1" sulla seriale, viceversa in corrispondenza dello stato *false* verrà inviata sulla seriale la stringa "C0".

Quando viene letta la stringa "C1" dalla seriale la variabile booleana *controlloapp* viene portata ad 1 e questo corrisponde ad attivare il controllo manuale, ovvero permette di gestire la pinza attraverso gli slider. Quando, invece, viene letta dalla seriale la stringa "C0" la variabile *controlloapp* è portata a 0 e la pinza è controllata dai valori provenienti dal sensore di inerzia e dal sensore per l'EMG.

```

void icon(boolean theValue) {

    println("got an event for icon", theValue);
    if (theValue == true) {
        MyPort.write("C1");
        pulsante=true;
    }
    if (theValue == false) {
        MyPort.write("C0");
        pulsante=false;
    }
}

```

Sono stati creati, inoltre, dei riquadri in cui è possibile visualizzare i valori provenienti dai sensori come la temperatura, l'inclinazione e i battiti. Per la temperatura è stato costruito anche un termometro.

Bpm

Per visualizzare il valore dei battiti è stata importata l'immagine di un cuore all'interno del quale viene scritto il valore. L'immagine è stata prima copiata nella cartella del programma, poi è stata caricata nel codice e infine incollata nella finestra grafica con la funzione *image()*.

L'aggiornamento continuo dei valori dei battiti è reso possibile dalla funzione *text()* in cui viene mandata una stringa *nfn(Batt, 0)* che trasforma la variabile float *Batt* che contiene i valori dei battiti trasmessi via bluetooth dall'Arduino Uno in una stringa con zero cifre a destra del punto decimale.

```

public void cuore(){
    battitook = loadImage("battitook.png");
}

public void battito (float Batt){
    textAlign(CENTER);
    fill(250, 0, 0);
    text("BPM", x1+xshift+950, y1-10);
    fill(0, 0, 0);
    image(battitook, x1+900, y1+20);
    text(nfc(Batt, 0), 1017, 150);
}

```

Roll

Nel riquadro chiamato Roll, vengono visualizzati i valori dell'inclinazione del sensore di inerzia, che corrispondono ai valori di rotazione della pinza.

È stato creato semplicemente un rettangolo con la funzione *rect()* all'interno della quale sono stati riportati i valori aggiornati della variabile con la stessa modalità del battito, questa volta utilizzando la variabile *Incl*.

Termometro

Per creare il termometro è stato prima disegnato il bulbo con la funzione *ellipse()* che

disegna una ellisse centrata in x, y e i semiassi corrispondono alla lunghezza e all'altezza. Il bulbo è stato poi riempito con il mercurio e questo è reso possibile dalla funzione `fill(200, 0, 0)` che restituisce una colorazione sulla scala del rosso.

Successivamente è stato costruito il termometro che è dato dall'unione di un arco di circonferenza e un rettangolo, quindi sono state utilizzate le funzioni `arc()` e `rect()`. La funzione `arc()` contiene diversi valori come la posizione x e y del centro, la larghezza e l'altezza ed infine l'angolo di partenza e di arrivo.

Una volta che sono stati disegnati il bulbo e il termometro la parte interna al termometro viene riempita con il mercurio fino ad una certa altezza che è determinata da una funzione che permette di variare l'altezza del mercurio in accordo al valore di temperatura ricevuto.

Di seguito sono riportate le spiegazioni relative al meccanismo del mercurio e della scala centigrada.

Mercurio

Per costruire la colonna di mercurio che segnala la temperatura rilevata è stata utilizzata la funzione `map(tempC, 0, 50, 0, 175)` che mappa la temperatura tra 0 e 50 gradi nell'altezza della scala graduata. Poi $h = -y - 66$ corrisponde all'offset tra lo zero e il bulbo.

Scala centigrada

In questa parte del codice è stata costruita la scala centigrada formata da una tacca più grande ogni 10 gradi e una più piccola a metà di ogni due tacche grandi.

Inizialmente sono state disegnate le tacche più grandi, quelle ogni 10 °C. È stato utilizzato un ciclo `for` da 0 a 5. Il disegno delle tacche è stato fatto tramite il comando `line()`, le due x del comando rappresentano la lunghezza della tacca, mentre le due y del segmento sono uguali e dipendono dal valore dell'indice.

Successivamente sono state disegnate le tacche più piccole con la stessa logica. La larghezza delle tacche diminuisce, in particolare è la metà delle precedenti, mentre per le y il ragionamento è analogo al precedente.

`line(x inizio tacca, y, x fine tacca, y)`

Il codice completo è riportato nell'Appendice E.

Decodifica bluetooth

Il Bluetooth svolge due importanti funzioni, la prima riguarda il passaggio al controllo manuale.

All'attivazione dello switch viene inviato un comando codificato Cx, dove x rappresenta lo stato del sistema. Se è 1 allora il controllo sarà manuale altrimenti sarà lasciato ai sensori.

Oltre ad inviare questo comando vengono inviati anche i valori angolari dei servomotori che verranno ricevuti, decodificati e implementati dalle funzioni `pos_servo2(angolo)` e `pinzacontrol2(angolo)` nella struttura in cui sono presenti i servomotori. Per inviare questi comandi è stato creato un doppio controllo (come riportato nella sezione Sliders) controllando il mouse premuto sulle coordinate dello slider e da un timer non bloccante per evitare errori nel buffer seriale di ricezione.

```
if (millis()-ti>500) {
    if (pulsante==true) {
        if (mousePressed && mouseY < y2+20) {
            float pinzamap=map(target1, 50, 350, 0, 180);
            String pinza="P"+str(pinzamap);
            MyPort.write(pinza);
            println(pinza);
        }
    }
    ti=millis();
}
```

La rotazione è mappata e codificata con P per la pinza e R per la rotazione angolare.

L'altra funzione importante è ricevere i dati di temperatura, rotazione ed elettromiografia.

Per fare questo è stata implementata la funzione *serialreceive()*.

```
if (MyPort.available() > 0) {
    dati = MyPort.readStringUntil('\n');
    dati=dati.trim();
}
dativec=dati.charAt(0);
if (dativec == 'T') {
    valore = "";
    for (int i = 1; i < dati.length(); i++) {
        valore = valore +dati.charAt(i) ;
    }
    tempC = float(valore);
}
// ecc.
```

Questa funzione controlla i dati in ingresso e li converte in *char array* (un array di caratteri). Quindi controlla il primo elemento per decodificare i dati e converte il valore in un numero e aggiorna la variabile.

Codice

Per rendere il codice più scorrevole e funzionale, sono state create delle classi contenenti i vari elementi grafici e funzionali. Ci sono differenti classi per:

- Termometro
- Schermi virtuali per visualizzare i dati
- Slider
- Comunicazione bluetooth e seriale
- Controllo dello switch
- Grafico EMG

Le varie classi sono collegate in modo da scambiarsi le variabili più importanti.

All'interno di ogni classe sono stati creati vari oggetti per implementare le diverse funzioni. È molto interessante, ad esempio, come questi tre diversi oggetti vengono richiamati nella classe di visualizzazione dei dati.

```
schermo.temperaturadigitale(tempC);
schermo.inclinazione(Incl);
schermo.battito(Batt);
```

Questo per disegnare e aggiornare i tre display virtuali.

In questo modo è stato ottenuto un codice finale molto più organizzato. Il codice è riportato in Appendice D.

App per smartphone

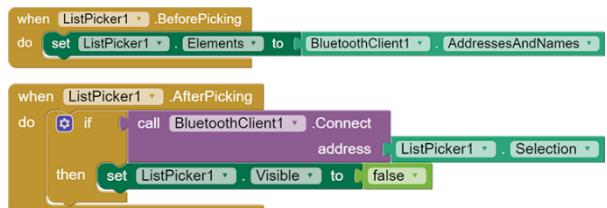
Dopo aver sviluppato un'interfaccia grafica sul computer è stata sviluppata anche un'applicazione per smartphone tramite MIT App Inventor [15]. Questa applicazione ha permesso di sviluppare una app completa e funzionale ma allo stesso tempo semplice. Un limite di questa applicazione consiste nel fatto che può utilizzata solo su dispositivi Android.

Le funzionalità sono le stesse dell'app per computer ovviamente integrate con la portabilità e l'ubiquità dello smartphone.

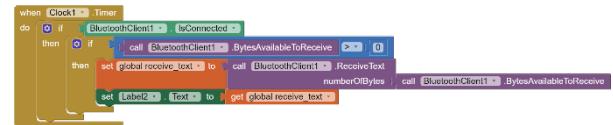
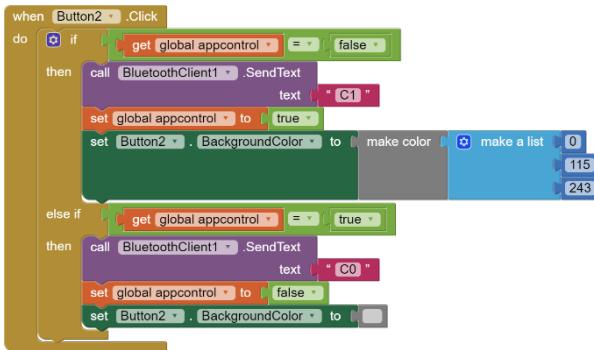
Per sviluppare l'applicazione per prima cosa si devono aggiungere tutti i componenti necessari come il List Picker, i Label, gli slider, il Bluetooth Client, i Clock e i Bottoni.

Una volta selezionati i componenti, si creano i blocchi. I diversi blocchi sono stati creati separatamente, facendo di volta in volta test per verificarne la funzionalità. Solo alla fine la grafica è stata ultimata e perfezionata.

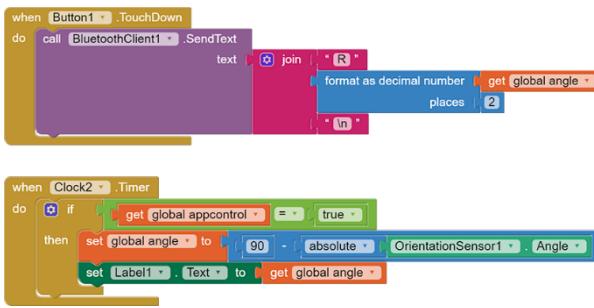
I primi blocchi necessari sono quelli che permettono di selezionare il Bluetooth MAC Address del modulo bluetooth (HC-05) e che quindi permettono l'accoppiamento tra lo smartphone e il bluetooth su Arduino.



Successivamente, per passare dal controllo automatico dei servo a quello manuale è stato creato un blocco con funzioni simili a quelle di uno switch che invia "C0" e "C1" con le stesse regole di codifica viste per Processing.



Quando il controllo manuale risulta attivato (*appcontrol == true*) è possibile inviare i relativi dati. In particolare, vengono inviati sulla seriale i valori angolari dell'apertura o chiusura della pinza e della sua rotazione. Per quanto riguarda la rotazione per simulare questa funzione è stato utilizzato il giroscopio integrato nello smartphone e il relativo valore è stato riportato in *Button 1*. L'apertura o chiusura della pinza è invece controllata da uno slider.



Per ricevere i dati, invece, è necessario verificare la disponibilità e salvare quanto ricevuto sul buffer della seriale in una variabile.

In base ai tipi di variabile proposti da MIT App Inventor è stato necessario dividere la stringa contenente i dati ricevuti in una *lista*. Questa lista, *decode_text*, contiene diversi elementi ciascuno dei quali rappresenta un nuovo dato aggiornato codificato. I diversi elementi sono stati, quindi, decodificati controllando il primo carattere e salvando il valore corrispondente nella rispettiva variabile. Per il codice consultare Appendice G.

A questo punto è stata utilizzata un'interfaccia grafica di prova Fig. 31 per verificare il funzionamento generale.

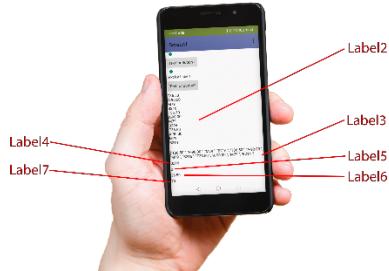


Fig. 31 – Interfaccia grafica di prova.

Dopo una serie di prove è stato ultimato il layout grafico e i blocchi sono stati riorganizzati. La struttura finale dei blocchi è riportata in Appendice H.

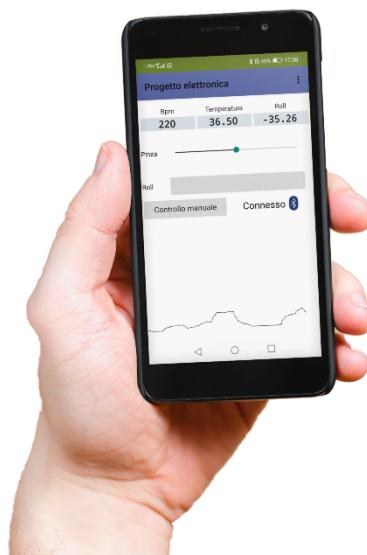


Fig. 30 – Layout finale dell'app.

Registrazione e Archiviazione dei dati

Visto il recente interesse per la telemedicina è sembrato opportuno memorizzare i dati dell'esperimento direttamente in un server remoto, dando così la possibilità di visionarli da remoto e anche in tempi successivi.

Il circuito originale è stato, inoltre, integrato con un sensore di temperatura e umidità ambientale.

I dati vengono salvati su MySQL database raggiungibile da un dominio di proprietà di uno degli autori [16].

Per la visualizzazione dei dati sono state usate le librerie Highcharts [17], si tratta di una libreria software per la creazione di grafici dinamici scritta in puro JavaScript.

Di seguito sono riportati i dispositivi che sono stati utilizzati.

ESP 8266

Il modulo Wi-Fi è stato aggiunto per permettere il salvataggio dei dati da remoto.

NodeMCU è una piattaforma open source sviluppata appositamente per l'IoT, include un firmware che funziona tramite il modulo Wi-Fi SoC ESP8266 e hardware con la base del modulo ESP-12E. Per comunicare con il computer utilizza un chip CH340.

La scheda è molto potente e avrebbe potuto sostituire l'Arduino Nano, tuttavia, è stato scelto di utilizzare l'Arduino per lasciare il sistema modulare con la possibilità di aggiungere o meno questa funzione.

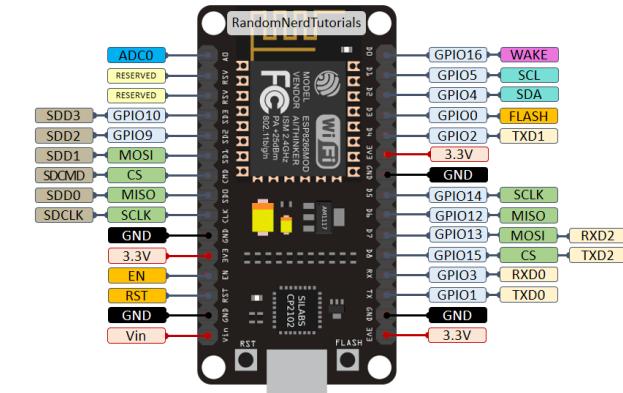


Fig. 32 – Pinout NodeMCU - Modulo Wi-Fi [18]

Per leggere i dati provenienti dall'Arduino nano, è stata implementata una software serial sui pin 3 e 4, trasmettendo i dati con la stessa logica di codifica utilizzata nei passaggi precedenti.

Dal lato dell'Arduino Nano si ha:

```
void trasmitt (String param, String val) {  
    invio = param + String(val);  
    mySerial.println(invio);  
}
```

Passandogli i diversi parametri come *transmit("T", String (tempmpu))*.

Dal lato dell'ESP, invece, è stata inizialmente ripulita la stringa di caratteri di formattazione estrema (*decodserial()*) per poi passarla ad una seconda funzione che si occupa di trasformarla in un char array (*process()*) e infine la decodifica vera e propria aggiornando le variabili (*analizestring()*).

Inoltre, il dispositivo viene alimentato dai 5V di Arduino Nano questo per poterli accendere insieme.

È stato inoltre, implementato un software seriale su NodeMCU con una libreria dedicata.

```
#include <SoftwareSerial.h>  
SoftwareSerial Esp_serial(D3, D4);
```

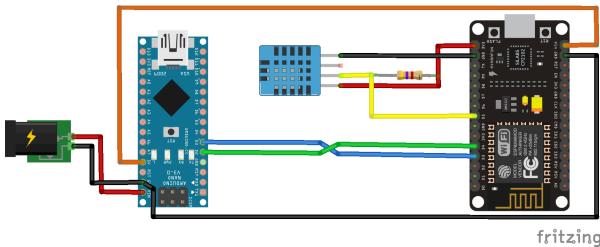


Fig. 33 – Circuito con lo schema di collegamento del modulo Wi-Fi e del sensore di temperatura.

DHT 11

Come riportato nel paragrafo precedente, è stato aggiunto anche un sensore di temperatura e umidità.

Il sensore DHT11 legge i valori utilizzando la libreria Adafruit per i sensori DHT.

È stato implementato in una libreria dedicata:

```
#include "DHT.h"
#define DHTTYPE DHT11 // DHT 11
const int DHTPin = 5;
DHT dht(DHTPin, DHTTYPE);
float h;
float t;
unsigned long timea=0;

void letturadht() {
    if (millis() - timea > 500) {
        h = dht.readHumidity();
        t = dht.readTemperature();
        Serial.println(h);
        Serial.println(t);
        timea=millis();
    }
}
```

Di seguito è stato approfondito il discorso relativo alla memorizzazione dei dati in un server remoto.

MySQL Database

Dal lato del server è stato creato un database dedicato con un utente personalizzato. Sono stati necessari i seguenti dati (*nome_db* e *username*) per implementare il codice nelle fasi successive.

Utilizzando *phpMyAdmin* è stata creata una tabella pronta a ricevere tutte le variabili di interesse:

```
CREATE TABLE Sensor (
    id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    tempmpu VARCHAR(10),
    angle VARCHAR(10),
    average VARCHAR(10),
    bpm VARCHAR(10),
    soglia VARCHAR(10),
    temp VARCHAR(10),
    hum VARCHAR(10),
    reading_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)
```

Richieste POST

Per inviare dati è stato utilizzato il metodo di richiesta HTTP POST [19]. Per impostazione predefinita, il metodo di richiesta POST richiede che un server Web accetti i dati racchiusi nel corpo del messaggio di richiesta, tipicamente per l'archiviazione.

È stato, quindi, collegato l'ESP 8266 ad Internet tramite la libreria Wi-Fi. È stato inoltre necessario includere una libreria per HTTP.

È stata, quindi implementata una funzione che invia ripetutamente un messaggio di testo contenente le variabili concatenate in una stringa che il server decodificherà in modo da posizionare correttamente i dati.

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
//timer
unsigned long t2 = 0;
unsigned long dt = 0;
unsigned long t3 = 200;//update database every

void servercall () {
    if (WiFi.status() == WL_CONNECTED) {
        HTTPClient http;
        http.begin(serverName);
        http.addHeader("Content-Type", "application/x-www-form-urlencoded");
        dt = millis() - t2;
        if (dt >= t3) {
            String httpRequestData = "api_key=" + apiKeyValue + "&tempmpu=" + String(tempmpu)
                + "&angle=" + String(angle) + "&average=" + String(average)
                + "&bpm=" + String(BPM) + "&soglia=" + String(soglia)
                + "&temp=" + String(t) + "&hume=" + String(h) + "";
            int httpResponseCode = http.POST(httpRequestData);
            t2 = millis();
        }
        http.end();
    }
}
```

I dati sensibili come, per esempio, le password Wi-Fi e i nomi utente del database vengono salvate nel file *sensdata.h*.

Il codice principale è il seguente:

```
#include "serialreceive.h"
#include "dhtread.h"
#include "wifidata.h"
void setup() {
  Serial.begin(9600);
  Esp_serial.begin(9600);
  dht.begin();
  WiFi.begin(ssid, password);
  Serial.println("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println(".");
  }
  Serial.println(WiFi.localIP());
}
void loop() {
  letturadht();
  servercall();
  decodeserial();
  delay(1);
}
```

Una pagina php viene richiamata sul server e si occupa di decodificare i dati. La funzione si occupa di ricevere la stringa completa e di identificare le diverse variabili concatenate nell'ordine prescritto, collocandole poi nella colonna di riferimento all'interno della tabella. La funzione segna anche l'ora dell'inserimento.

Per il codice fare riferimento ad Appendice F.

Visualizzazione dei dati

Per leggere i dati viene chiamata un'altra pagina php. Il codice è diviso in tre sezioni. La prima sezione si occupa di recuperare i dati dal database; la seconda si occupa delle strutture HTML e la terza si occupa dei grafici con l'aiuto delle librerie Highchart.

La pagina web con i dati è accessibile al seguente link <http://tempcontrol.it/progelettronica/chart.php> o con questo short link: <http://bit.ly/datiprogettoelettronica>

La pagina web è protetta da una password con un algoritmo php [20], inserire *elettronica*.

Una pagina di struttura php è stata creata in diverse sezioni.

Una prima sezione di occupa della chiamata del database e della lettura degli ultimi valori. Anche questo è contenuto nel file *call.php*. Quindi salva i dati in variabili che verranno richiamate dai grafici. Ovviamente si tratta di vettori.

```
$readings_time = array_column($sensor_data, 'reading_time');
$tempmpu = json_encode(array_reverse(array_column($sensor_data, 'tempmpu')), JSON_NUMERIC_CHECK);
$angle = json_encode(array_reverse(array_column($sensor_data, 'angle')), JSON_NUMERIC_CHECK);
$average = json_encode(array_reverse(array_column($sensor_data, 'average')), JSON_NUMERIC_CHECK);
$bpm = json_encode(array_reverse(array_column($sensor_data, 'bpm')), JSON_NUMERIC_CHECK);
$sgoglia = json_encode(array_reverse(array_column($sensor_data, 'sgoglia')), JSON_NUMERIC_CHECK);
$temp = json_encode(array_reverse(array_column($sensor_data, 'temp')), JSON_NUMERIC_CHECK);
$hum = json_encode(array_reverse(array_column($sensor_data, 'hum')), JSON_NUMERIC_CHECK);
$reading_time = json_encode(array_reverse($readings_time), JSON_NUMERIC_CHECK);
```

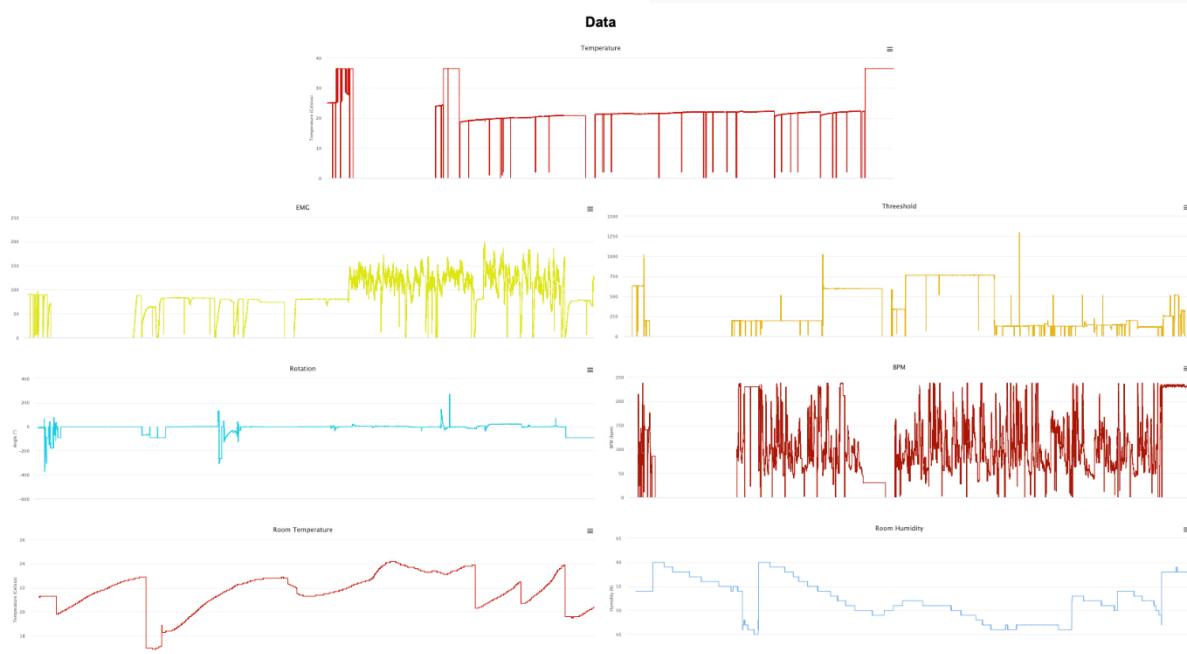


Fig. 34 – Grafici.

Poi c'è la struttura HTML in cui sono inseriti i diversi contenitori ognuno con un proprio stile per organizzare il layout.

```
body {
    min-width: 310px;
    max-width: 90%;
    height: 500px;
    margin: 0 auto;
}
.container {
    position: center;
    column-count: 2;
    column-width: 50%;
    padding-top: 50px;
}
```

Richiamando le variabili contenenti i dati, i grafici vengono costruiti utilizzando, come già scritto precedentemente, le librerie Highchart e quindi con un frammento di codice Java del tipo:

```
var tempmpu = <?php echo $tempmpu; ?>;
// ...
var reading_time = <?php echo $reading_time; ?>;
// ...
var chartT = new Highcharts.Chart({
    chart: { renderTo: 'chart-tempmpu', zoomType: 'x', panning: true, panKey: 'shift' },
    title: { text: 'Temperature' },
    series: [ { showInLegend: false, data: tempmpu } ],
    plotOptions: { line: { animation: false, dataLabels: { enabled: false } } },
    series: { color: "#C80303" },
    xAxis: { type: 'datetime', labels: { enabled: false }, categories: reading_time },
    yAxis: { title: { text: 'Temperature (Celsius)' } },
    credits: { enabled: false }
});
```

A questo si aggiunge un ulteriore algoritmo per proteggere la visualizzazione dei grafici con una password come mostrato in Appendice F.

I grafici sono ampiamente personalizzabili, in particolare, è stata attivata la possibilità di zoomare in un certo intervallo di tempo e di scorrire lungo l'asse del tempo. È anche possibile selezionare un singolo valore e leggere altre informazioni.

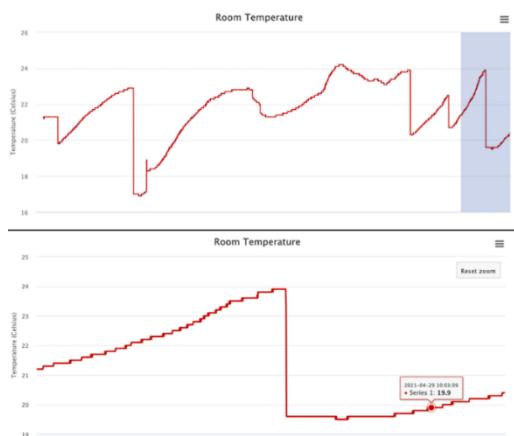


Fig. 35 – Esempio di un grafico con una regione zoomata riportata nella seconda figura.

Come rendere tutto più compatto

Per rendere tutto più compatto ed evitare problemi dovuti alla mancanza di stabilità della breadboard è stato deciso di saldare il tutto su più schede millefori. Tuttavia, per consentire i test e la sostituzione dei componenti sono stati utilizzati anche connettori multi-pin.

Cavo con i sensori

Per consentire un facile trasporto e smontaggio sono stati uniti tutti i sensori in un unico cavo.

L'EMG ha il suo cavo con jack da 3,5 mm.

È stato utilizzato un cavo Cat 5e con schema di cablaggio T568B. VCC e GND del pulsesensor e del sensore di inerzia sono stati uniti insieme e i dati del pulsesensor sono stati mandati sul pin A3.

I sensori sono stati collegati come segue:

Color	Wire
white/orange	
orange	+5V
white/green	SCL
blue	A3
white/blue	SDA
green	
white/brown	
brown	GND

Fig. 36 – Tabella codice colore per l'assegnazione dei pin.

Stazione di lettura

Per avere una struttura compatta ma che consentisse di effettuare diversi test, sono state impilate diverse schede millefori. Sono inoltre presenti un connettore jack femmina da 3,5 mm per il cavo dell'elettromiografia e un connettore RJ45 per il cavo del sensore. Al piano superiore sono presenti un display LCD e un trimmer per la regolazione della soglia.



Fig. 37 – Stazione di lettura.

Alimentazione

Per l'alimentazione vengono proposte due modalità diverse.

Nella prima sono stati utilizzati due diversi alimentatori da 9V, unendo un morsetto (+) con un (-) si ottiene un riferimento di massa in modo da avere +9V e -9V sugli altri due cavi.



Fig. 39 – Modifica degli alimentatori

Questo serve per alimentare il circuito dell'elettromiografia. È stato successivamente aggiunto un alimentatore da 12V per alimentare l'Arduino Nano da Vin che a sua volta alimenta l'ESP8266 e l'intero sistema. È stato quindi collegato il tutto alla base della stazione di lettura in modo da alimentare tutto con un unico cavo con connettore C13. Sono stati aggiunti, inoltre, due interruttori per accedere selettivamente al sistema e all'alimentazione per l'EMG.

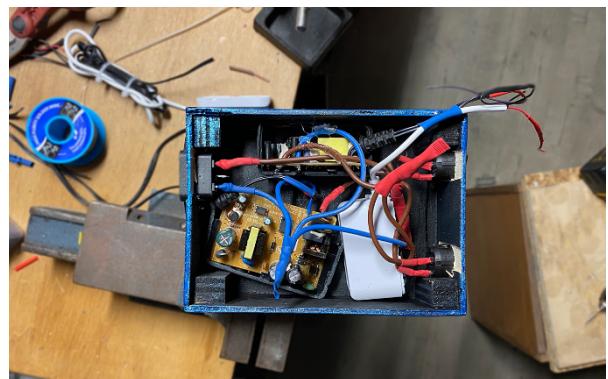


Fig. 40 – Configurazione dello schema di alimentazione

Nella seconda è previsto l'utilizzo di due batterie da 9V per il sensore dell'EMG e due power bank per alimentare i due Arduino, senza dover ricorrere ad un computer con cavo USB.

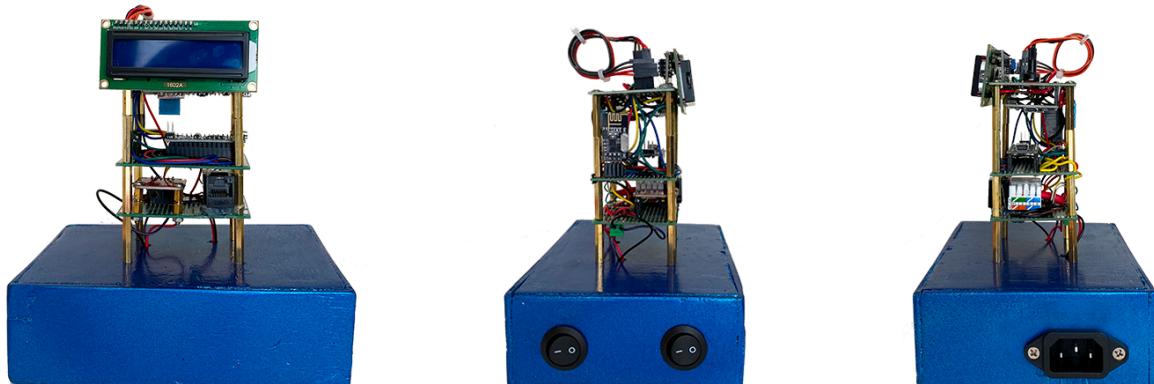


Fig. 38 – Struttura finale della stazione di lettura dati.

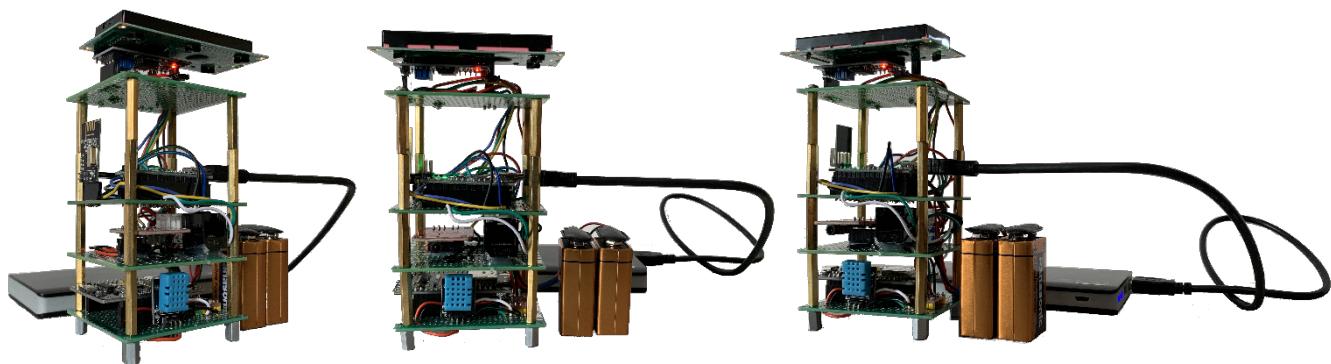


Fig. 41 – Alternativa della stazione di lettura dati.

Conclusioni e sviluppi futuri

Il sensore di temperatura non riferisce perfettamente la temperatura del soggetto e potrebbe essere sostituito con un sensore a contatto con la pelle ed eventualmente studiato un algoritmo per riportare la temperatura periferica a quella centrale.

L'utilizzo di un sensore EMG più avanzato come il MyoWare [21] permette di ridurre i disturbi e avere una struttura più compatta.

Per sviluppare un'applicazione per iOS è necessario sostituire il modulo bluetooth con uno con Bluetooth Low Energy come il modulo SH-HC-08. Una prima bozza è stata discussa nella repository online di questo progetto.

Per aumentare la sicurezza dei dati in ottica di avere un rispetto della privacy e delle normative GDPR per il trattamento dei dati personali online, è sicuramente necessario provvedere ad un algoritmo di sicurezza molto più serio del semplice nascondere il contenuto della pagina con lo script php. Inoltre, potrebbe essere opportuno provvedere a una criptazione dei dati anche prima di inviarli al server e attivare il protocollo HTTPS nella pagina web.

Per la temperatura e umidità ambientali può essere opportuno sostituire il DHT11 con un

DHT22 ottenendo così una sensibilità di 0.5°C e 2% RH.

Per l'alimentazione, invece di tre alimentatori si potrebbe utilizzare un unico alimentatore da 18V con un voltage divider per ottenere $\pm 9\text{V}$.

Infine, la stazione di lettura potrebbe essere ri-progettata su PCB in modo da avere un unico piano più compatto.

Contenuti aggiuntivi

Per il video dimostrativo:

youtu.be/k1hNBbgfAyE

A seguire un'Appendice in cui sono riportate le parti più interessanti di codice.

È possibile trovare un resoconto fatto passo passo con ulteriori dettagli, i codici completi e gli schemi di collegamento fritzing sulla repository: github.com/mastroalex/progelettronica

Riferimenti

- [1] Invesense, «MPU6050,» [Online]. Available: <https://invensense.tdk.com/products/motion-tracking/6-axis/mpu-6050/>.
- [2] Wikipedia, «Elettromiografia ed elettroneurografia,» [Online]. Available: https://it.wikipedia.org/wiki/Elettromiografia_ed_elettroneurografia.
- [3] A. Newton, «Electromyography(EMG) with MyoWare Muscle Sensor & Arduino,» [Online]. Available: <https://how2electronics.com/electromyography-emg-with-myoware-muscle-sensor-arduino/>.
- [4] Advancer Technologies Shop, «Muscle Sensor v3,» [Online]. Available: <http://www.advancertechnologies.com/p/muscle-sensor-v3.html>.
- [5] FIAB, «Elettrodi,» [Online]. Available: <https://www.fiab.it/prodotti.php>.
- [6] Ram Apparecchi Medicali, «Elettrodi pregellati in FOAM per ECG e Stress Test 36x42mm,» TopTrace, [Online]. Available: <https://www.elettromedicali.it/diagnostica/elettrocardiografi/elettrodi-monouso-per-ecg/prodotto-elettrodi-pregellati-in-foam-per-ecg-e-stress-test-36x42-mm-solid-gel-confez-da-50pz/>.
- [7] Arduino, «Smoothing,» [Online]. Available: <https://www.arduino.cc/en/Tutorial/BuiltInExamples/Smoothing>.
- [8] ROHM, «Pulse sensor,» [Online]. Available: <https://www.rohm.com/electronics-basics/sensor/pulse-sensor>.
- [9] pulsesensor.com, «Anatomy of The DIY Heart Rate Monitor,» [Online]. Available: <https://pulsesensor.com/blogs/news/6326816-anatomy-of-the-diy-heart-rate-monitor>.
- [10] «Pulsesensor,» [Online]. Available: <https://pulsesensor.com/pages/code-and-guide>.
- [11] «Instructables - Pulse sensor,» [Online]. Available: <https://www.instructables.com/circuits/howto/pulse+sensor/>.
- [12] «NRF24L01,» [Online]. Available: <https://components101.com/wireless/nrf24l01-pinout-features-datasheet>.
- [13] Makerguides.com, «Arduino Nano pinout,» [Online]. Available: <https://www.makerguides.com/arduino-nano/>.
- [14] «LCD Custom Generator,» [Online]. Available: <https://maxpromer.github.io/LCD-Character-Creator/>.
- [15] «MIT App Inventor,» [Online]. Available: <https://appinventor.mit.edu/>.
- [16] M. Alessandro, «Tempcontrol,» [Online]. Available: <https://github.com/mastroalex/tempcontrol>.
- [17] «Highcharts,» [Online]. Available: <https://www.highcharts.com/blog/products/highcharts/>.
- [18] randomnerdtutorials.com, «ESP8266 Pinout Reference: Which GPIO pins should you use?,» [Online]. Available: <https://randomnerdtutorials.com/esp8266-pinout-reference-gpios/>.
- [19] Wikipedia, «HTTP POST,» [Online]. Available: [https://en.wikipedia.org/wiki/POST_\(HTTP\)](https://en.wikipedia.org/wiki/POST_(HTTP)).
- [20] mrw.it, «Pagina protetta da password,» [Online]. Available: https://www.mrw.it/php/pagina-protetta-password_11675.html.
- [21] Sparkfun, «MyoWare Muscle Sensor,» [Online]. Available: <https://www.sparkfun.com/products/13723>.

Indice delle figure

Fig. 1 – Struttura di esempio della pinza per il controllo di apertura e chiusura e della rotazione.....	3
Fig. 2 – Esempio di struttura realizzata con gli ingranaggi, plastica sagomata e una base di alluminio anodizzato.....	3
Fig. 3 – Modello della pinza in compensato.....	4
Fig. 4 – Orientazione degli assi per il sensore inerziale.....	5
Fig. 5 – Collegamento dei pin del servo.....	5
Fig. 6 – Schema di collegamento del servo e dell'IMU (sensore inerziale).....	6
Fig. 7 – Segnale grezzo EMG prima di essere filtrato, rettificato e smoothed [3].....	6
Fig. 8 – Schema di collegamento dei pin del sensore.....	6
Fig. 9 – Schema dell'amplificatore per strumentazione.....	6
Fig. 10 – Raddrizzatore a doppia semionda che permette di rettificare il segnale amplificato.....	6
Fig. 11 – Smoothing, filtro attivo passa basso.....	7
Fig. 12 – Raw EMG vs EMG dopo il raddrizzatore a doppia semionda e lo smoothing.....	7
Fig. 13 – Copia del Muscle Sensor V3 utilizzata nel progetto.....	7
Fig. 14 – Posizionamento degli elettrodi.....	7
Fig. 15 – Segnale smoothed (blu), segnale originale (verde), soglia (rosso). Quando il segnale è al di sotto del valore soglia l'arto è rilassato, mentre, quando il segnale è al di sopra della soglia il muscolo è contratto.....	8
Fig. 16 – Schema circuitale del Pulse Sensor [9].....	8
Fig. 17 – Immagine del sensore. Il cavo rosso corrisponde ai 5V, il cavo nero al GND e il cavo viola è il segnale.....	9
Fig. 18 – Picco e depressione del segnale grezzo.....	9
Fig. 19 – Circuito con i diversi sensori utilizzati.....	10
Fig. 20 – Modulo NRF24L01 e schema di collegamento dei pin.....	10
Fig. 21 – Disposizione dei sensori e dell'Arduino nano sulla breadboard.....	11
Fig. 22 – Disposizione e definizione dei pin su Arduino Nano [13].....	11
Fig. 23 – Schema completo dei sensori.....	12
Fig. 24 - Creazione di caratteri speciali e disposizione dei valori sul display.....	12
Fig. 25 – Configurazione dell'Arduino uno con i servo, il ricetrasmettitore e la pinza.....	13
Fig. 26 – Convertitore di livello logico.....	13
Fig. 27 – Modulo Bluetooth.....	13
Fig. 28 – Arduino Uno con i servo, il convertitore di livello logico, il modulo NRF24 e il modulo bluetooth... 14	14
Fig. 29 – Interfaccia grafica Processing.....	15
Fig. 30 – Layout finale dell'app.....	20
Fig. 31 – Interfaccia grafica di prova.....	20
Fig. 32 – Pinout NodeMCU - Modulo Wi-Fi [18]	21
Fig. 33 – Circuito con lo schema di collegamento del modulo Wi-Fi e del sensore di temperatura.....	22
Fig. 34 – Grafici.....	23
Fig. 35 – Esempio di un grafico con una regione zoomata riportata nella seconda figura.....	24
Fig. 36 – Tabella codice colore per l'assegnazione dei pin.....	24
Fig. 37 – Stazione di lettura.....	25
Fig. 39 – Modifica degli alimentatori.....	25
Fig. 38 – Struttura finale della stazione di lettura dati.....	25
Fig. 40 – Configurazione dello schema di alimentazione.....	25
Fig. 41 – Alternativa della stazione di lettura dati.....	26

Appendice

Appendice A

```
int readings[numReadings];           // the readings from the analog input
int readIndex = 0;                   // the index of the current reading
int total = 0;                      // the running total
int average = 0;                    // the average
int inputPin = A0;                  // EMG analog pin

void setup() {
    Serial.begin(9600);
    // initialize all the readings to 0:
    for (int thisReading = 0; thisReading < numReadings; thisReading++) {
        readings[thisReading] = 0;
    }
}

void loop() {

    total = total - readings[readIndex];           // subtract the last reading:
    readings[readIndex] = analogRead(inputPin);     // read from the sensor
    total = total + readings[readIndex];            // add the reading to the total:
    readIndex = readIndex + 1;                      // advance to the next position in the array:

    if (readIndex >= numReadings) {                // if we're at the end of the array
        readIndex = 0;                            // wrap around to the beginning
    }
    average = total / numReadings;                 // calculate the average:
    // print different data for plotting

    Serial.print(average);
    Serial.print("\t");
    Serial.print(soglia);
    Serial.print("\t");
    Serial.println(analogRead(inputPin));
    delay(1);                                     // delay in between reads for stability
}
```

Appendice A

Appendice B

```
#include <Servo.h>
Servo servoclamp;
long t3 = 0;
long t4 = 20;
boolean controllo[10];
boolean stat0;
int i = 0;
int soglia = 350; // Threeshold
// This value derives from a long period of testing.
// You will need to do several tests
void setup() {
    servoclamp.attach(5);
    // other code...
}
void loop() {
// other code..
    if (average > soglia) {
        servoclamp.write(max_servo_angle); // gripper closing
    }
    if (millis() - t3 < t4) {
        if (average < soglia) {
            controllo[i] = LOW;
        }
        else {
            controllo[i] = HIGH;
        }
        if (i < 10) {
            i++;
        }
        t3 = millis();
    }
    // only if the average is below the threshold
    // for a long time is it possible to close the clamp
    // this allows to avoid problems due to sudden oscillations of the signal
    for (int j = 0; j < 10; j++) {
        if (controllo[j] == LOW) {
            stat0 = LOW;
        }
        else {
            stat0 = HIGH;
        }
    }
    if (stat0 == LOW) {
        servoclamp.write(min_servo_angle); // gripper opening
    }
}
```

Appendice B

Appendice C

```
#include <MPU6050_tockn.h>
#include <Wire.h>
#include "funzioniBPM.h" // battiti
#include "printerfunzioni.h" // stampante
#include "EMGsmooth.h" // lettura emg
MPU6050 mpu6050(Wire);
void setup() {
    Serial.begin(9600);
    Wire.begin(); // avvio e inizializzo il gyro
    mpu6050.begin();
    mpu6050.calcGyroOffsets(true);
    // mpu6050.setGyroOffsets(0, 0, 0);
    for (int thisReading = 0; thisReading < numReadings; thisReading++) { // initialize all the readings to 0:
        readings[thisReading] = 0;
    }
    t1 = millis();
    interruptSetup(); // sets up to read Pulse Sensor signal every 2mS
    lcd.begin(); // initialize the LCD
    lcd.backlight();
}
void loop() {
    average = averagecalc(); // calcolo media EMG
    mpu6050.update();
    tempmpu = mpu6050.getTemp(); // prende la temperatura
    angle = mpu6050.getAngleX(); // prende l'angolo lungo x
    printatore(LOW); // set HIGH to debug EMG with serial plotter
    lcdprint();
    battiti();
    charfortransmission("T", String(tempmpu));
    charfortransmission("A", String(angle));
    charfortransmission("M", String(average));
    charfortransmission("B", String(BPM));
    charfortransmission("S", String(soglia));
    delay(1);
}
```

Appendice C

Appendice D

```
public class Termom {  
    float y, h;  
  
    public Termom() {  
    }  
    //Disegno termometro  
    public void bulbo() {  
        // Bulbo  
        fill(200, 0, 0);  
        smooth();  
        stroke(0, 46, 200);  
        strokeWeight(8);  
        ellipse(250+1000, 250+300, 58, 50);  
    }  
  
    public void termometro() {  
        //Termometro  
        noStroke();  
        fill(0, 46, 200);  
        arc(250+1000, 30+300, 30, 20, PI, PI+PI);  
        rect(235.2+1000, 30+300, 30, 200);  
    }  
  
    public void mercurio() {  
        // Disegno solco mercurio  
        fill(250, 250, 250);  
        rect(245+1000, 30+300, 10, 200);  
  
        // Mercurio  
        fill(200, 0, 0);  
        smooth();  
        strokeWeight(0);  
        y= -2.0*tempC + 170;  
        h = 240-y;  
        rect(245.2+1000, y+300, 10, h);  
    }  
}
```

Appendice D

Appendice E

```
import controlP5.*;
ControlP5 cp5;
import processing.serial.*;
Serial MyPort;
float target1;
float target2;
PFont font24;
PFont font12;
boolean pulsante=false;
float tempC = 0;
float Incl = 0;
float Batt=0;

Termom termom= new Termom();
Schermo schermo=new Schermo();
Slider slider=new Slider();
Serialcommunication serialcommunication=new Serialcommunication();
Switchcommand switchcommand = new Switchcommand();
Grafico grafico = new Grafico();

void setup()
{
    size(1400, 700);
    background(255);
    MyPort = new Serial(this, "/dev/tty.HC-05-SerialPort", 9600);
    grafico.setdati();
    cp5 = new ControlP5(this);
    switchcommand.setup();
    schermo.cuore();
    slider.sliderset();
}

void draw()
{
    background(255);
    serialcommunication.serialreceive();
    grafico.assi();
    termom.disegna();
    slider.slider1();
    slider.slider2();
    schermo.temperaturadigitale(tempC);
    schermo.inclinazione(Incl);
    schermo.battito(Batt);
    serialcommunication.serialsend();
}

void icon(boolean theValue) {
    println("got an event for icon", theValue);
    if (theValue == true) {
        MyPort.write("C1");
        pulsante=true;
    }
    if (theValue == false) {
        MyPort.write("C0");
        pulsante=false;
    }
}
```

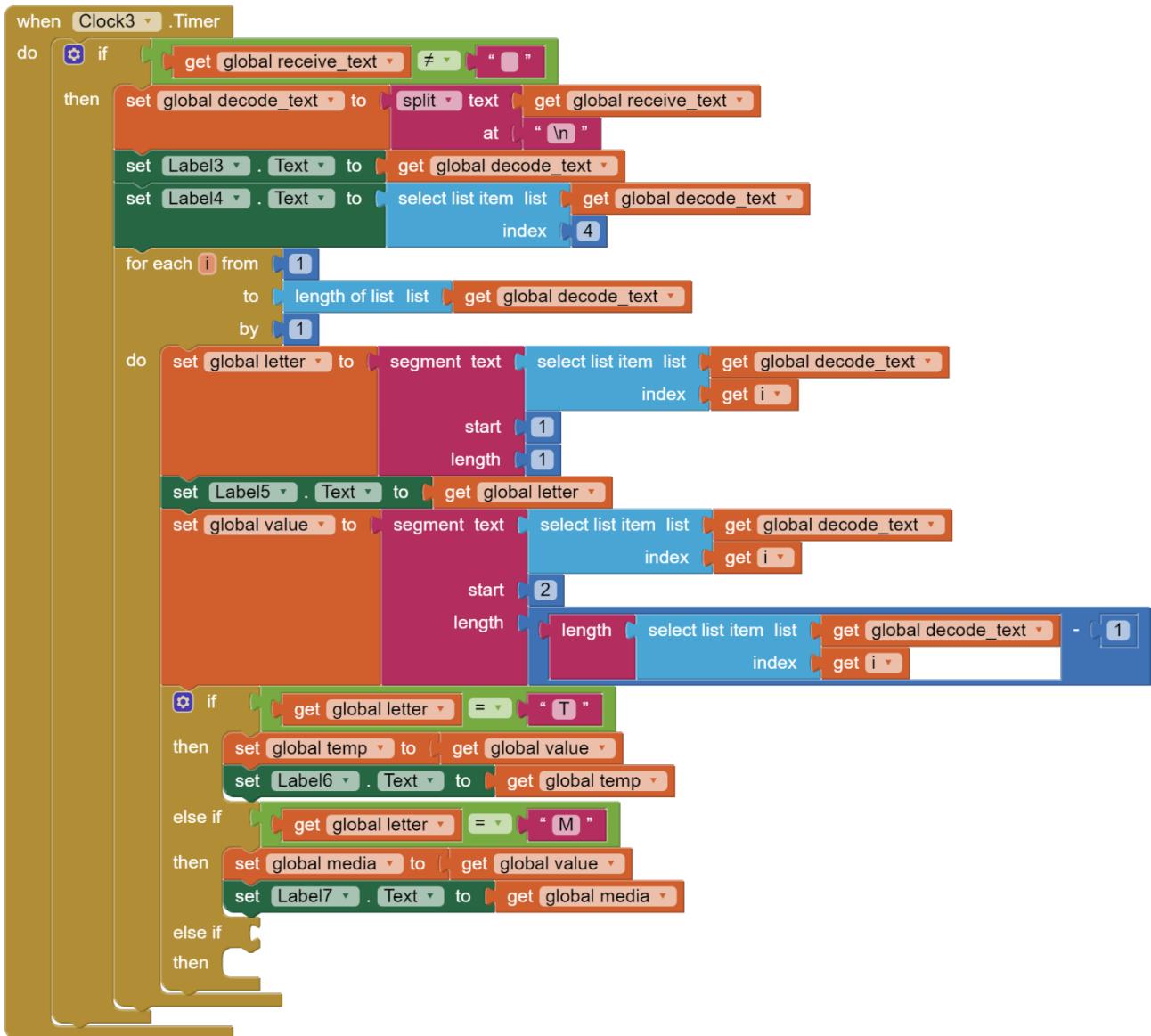
Appendice F

```
<?php

$servername = "localhost";
$dbname = "dbname";
$username = "username";
$password = "user_pw";
$api_key_value = "APIKEY";
$api_key = $tempmpu = $angle = $average = $bpm = $soglia = $temp = $hum = "";
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $api_key = test_input($_POST["api_key"]);
    if($api_key == $api_key_value) {
        $tempmpu = test_input($_POST["tempmpu"]);
        $angle = test_input($_POST["angle"]);
        $average = test_input($_POST["average"]);
        $bpm = test_input($_POST["bpm"]);
        $soglia = test_input($_POST["soglia"]);
        $temp = test_input($_POST["temp"]);
        $hum = test_input($_POST["hum"]);
        $conn = new mysqli($servername, $username, $password, $dbname);
        if ($conn->connect_error) {
            die("Connection failed: " . $conn->connect_error);
        }
        $sql = "INSERT INTO Sensor (tempmpu, angle, average, bpm, soglia, temp, hum)
VALUES ('" . $tempmpu . "', '" . $angle . "', '" . $average . ',
'" . $bpm . "','" . $soglia . "','" . $temp . "','" . $hum . "')";
        if ($conn->query($sql) === TRUE) {
            echo "New record created successfully";
        } else {
            echo "Error: " . $sql . "<br>" . $conn->error;
        }
        $conn->close();
    }
} else {
    echo "No data posted with HTTP POST.";
}
function test_input($data) {
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    return $data;
}
```

Appendice F

Appendice G



Appendice G

Appendice H



Appendice H