

ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA
Corso di Laurea in Ingegneria Informatica

SVILUPPO DI UN SISTEMA DI GESTURE RECOGNITION

Relatore:
Prof.
STEFANO MATTOCCIA

Candidato:
FEDERICO TERZI

Sessione I
Anno Accademico 2017-2018

Ringraziamenti

Prima di iniziare la tesi, tengo molto a ringraziare tutte le persone che, tra i vari alti e bassi, mi sono state accanto nel corso degli anni.

Un'enorme grazie va alla mia famiglia, che mi ha sempre dato la possibilità di inseguire i miei sogni e le mie passioni, anche nei momenti difficili, senza mai smettere di credere in me. Grazie a loro ho sempre avuto, sin dalla tenera età, la possibilità di utilizzare un computer e questo mi ha permesso di coltivare la mia passione per l'informatica. Se ho la fortuna di sapere cosa voglio fare nella vita, lo devo a loro.

Un grandissimo grazie va a Elena, la mia ragazza, che, oltre a rendere ogni giorno la mia vita più bella, mi ha anche aiutato a correggere gli errori grammaticali della tesi, dato che l'italiano non è mai stato il mio forte.

Un grande grazie va ai miei amici che, durante questi ultimi anni di università, mi hanno permesso di superare infinite giornate di studio col sorriso. Perché anche una semplice risata può fare la differenza.

Tengo molto anche a ringraziare il Prof. Mattoccia, che si è sempre dimostrato molto disponibile nei confronti dei suoi studenti, e mi ha permesso di portare questo progetto come tesi.

Lo scopo della tesi è creare un sistema di riconoscimento gestuale flessibile e in grado di adattarsi ad una grande varietà di casi. Dopo una breve analisi delle tecnologie hardware utilizzate, è stata trattata la creazione del dispositivo utilizzato per registrare i movimenti. L'attenzione si è poi spostata sulla realizzazione della libreria di riconoscimento gestuale, in cui sono stati descritti tutti i componenti con i rispettivi compiti. Infine sono stati trattati tre diversi casi di studio per analizzare le capacità e l'accuratezza del sistema.

Contenuti

1	Introduzione	1
2	Studio delle tecnologie hardware utilizzate	4
2.1	Arduino	4
2.2	Switch Meccanici	5
2.3	L'integrato MPU-6050	7
2.4	Modulo Bluetooth: HC-06	9
2.5	Sensore Piezoelettrico	10
2.6	Amplificatore LM386	11
2.7	Alimentazione	12
3	Progetto del prototipo hardware	13
3.1	Progettazione del Circuito	13
3.2	Realizzazione del Prototipo	14
3.3	Software del microcontrollore	14
3.3.1	Formato di Trasmissione	15
3.4	Considerazioni	17
4	Sviluppo della libreria di Gesture Recognition	18
4.1	Linguaggi e librerie utilizzate	18
4.2	Design Pattern e scelte strutturali	18
4.3	Flusso di gestione dei gesti	19
4.4	Pipeline di elaborazione dei dati	19
4.4.1	DataReader	21
4.4.2	SampleManager	21
4.4.3	Middleware	21
4.4.4	GesturePredictor	22
4.4.5	Callback Manager	22
4.5	Interfacce e Classi astratte	22
4.5.1	AbstractDataReader	22
4.5.2	Sender e AbstractSampleManager	23
4.5.3	Receiver e AbstractMiddleware	24
4.5.4	AbstractGesturePredictor	24
4.5.5	Classifier e AbstractClassifier	25
4.6	Principali implementazioni dei componenti	26
4.6.1	SerialDataReader	26

4.6.2	DiscreteSampleManager	27
4.6.3	StreamSampleManager	28
4.6.4	FileGestureRecorder	29
4.6.5	SVMClassifier	29
4.6.6	MLPClassifier	30
4.6.7	ClassifierPredictor	30
5	Caso di studio: Tastiera gestuale	31
5.1	Creazione del Dataset	32
5.2	Training del classificatore e analisi della Confusion Matrix	32
5.2.1	Training del SVMClassifier	33
5.2.2	Training del MLPClassifier	35
5.2.3	Scelta del classificatore	36
5.3	Pipeline di elaborazione del segnale	37
5.4	Risultati ottenuti e possibili miglioramenti	38
6	Caso di studio: Sensore di tocchi	40
6.1	Realizzazione del dispositivo	40
6.2	Analisi del segnale	42
6.2.1	GradientThresholdMiddleware	43
6.2.2	AbsoluteScaleMiddleware	47
6.3	Definizione dei gesti e creazione del Dataset	48
6.4	Training del classificatore e analisi della Confusion Matrix	49
6.5	Pipeline di elaborazione del segnale	50
6.6	Risultati ottenuti e possibili miglioramenti	52
7	Caso di studio: Anello d'ausilio a persone con difficoltà di movimento	54
7.1	Creazione dell'anello	54
7.2	Analisi del segnale	55
7.2.1	FFTMiddleware	56
7.3	Definizione dei gesti e creazione del Dataset	56
7.4	Training del classificatore e analisi della Confusion Matrix	58
7.5	Pipeline di elaborazione del segnale	59
7.6	Creazione dell'interfaccia utente	61
7.7	Risultati ottenuti e possibili miglioramenti	62
8	Conclusioni	63

Capitolo 1

Introduzione

Cercare nuovi modi per far interagire l'essere umano con la tecnologia è da sempre una priorità per i ricercatori che si occupano di informatica. Ogni decennio infatti assistiamo ad un sostanziale cambiamento delle tecnologie di input, che si rivelano sempre più comode ed efficienti. Basti pensare ai dispositivi di telefonia mobile di ultima generazione, che dal classico tastierino numerico sono passati a sistemi touchscreen capaci di adattarsi a ogni necessità. Tra la vastità dei possibili mezzi di input, questo progetto si sofferma su uno in particolare: i gesti.

L'obiettivo principale di questa ricerca è creare un sistema di riconoscimento gestuale flessibile e personalizzabile, in grado di adattarsi alle varie necessità riducendo al minimo i componenti da implementare e di conseguenza i tempi di realizzazione.

Per poter catturare e analizzare i gesti di un utente è necessario un dispositivo dotato di sensori e capace di registrare i movimenti. Per farlo, sono stati utilizzati due sensori: l'accelerometro e il giroscopio, in grado di rilevare rispettivamente l'accelerazione e la velocità angolare di un corpo in movimento [1][2]. Entrambi i sensori sono integrati all'interno del modulo MCU-6050, di cui parleremo nel dettaglio successivamente, e insieme forniscono un'accurata stima del movimento effettuato.

Un altro modo per catturare i gesti effettuati da un utente è quello di registrare le vibrazioni di una superficie. Infatti le oscillazioni saranno diverse in base al tipo di tocco e grazie a questa differenza sarà possibile discriminare il gesto effettuato.

Una volta registrato il movimento è necessario analizzarlo per determinare il tipo di gesto eseguito. Questo compito è affidato ad un algoritmo di machine learning che, a seguito di un "allenamento" iniziale, è in grado di riconoscere il gesto con un'accuratezza elevata.

Uno dei compiti del sistema in esame è prendersi carico delle modalità in cui i dati del sensore vengono trasferiti, raggruppati e analizzati in modo da permettere allo sviluppatore di concentrarsi sulla logica applicativa. Tuttavia è data la possibilità di personalizzare ogni strato dell'elaborazione per adattarsi a esigenze specifiche quando necessario.

Il primo caso di studio verterà sulla realizzazione di una tastiera gestuale, ovvero un dispositivo in grado di convertire i gesti di un utente in caratteri.

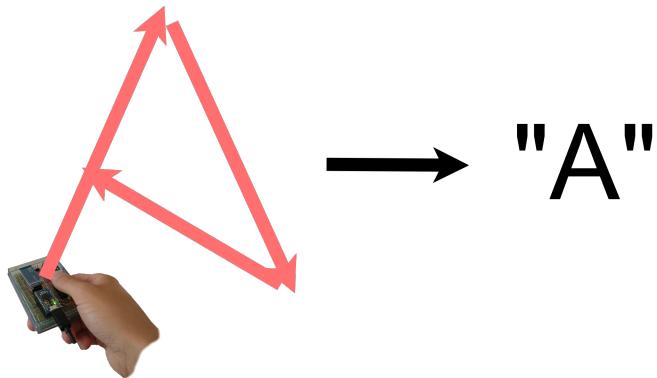


Figura 1.1: Obiettivo della tastiera gestuale.

La seconda parte del progetto tratterà la creazione di un dispositivo in grado di determinare il tipo di gesto effettuato a partire dalle vibrazioni della superficie toccata.

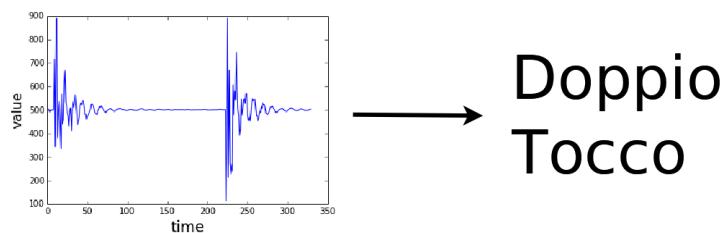


Figura 1.2: Obiettivo del sensore di tocco.

Infine, il terzo caso di studio verterà sulla realizzazione di un anello dotato di sensori ed in grado di aiutare le persone con difficoltà di movimento ad interagire con il mondo esterno.



Figura 1.3: Obiettivo del terzo caso di studio.

Capitolo 2

Studio delle tecnologie hardware utilizzate

Seppure in commercio esistano numerosi apparecchi in grado di registrare i movimenti, si è deciso di costruire un dispositivo ad-hoc che permettesse un completo controllo sull'intera elaborazione del segnale. Questa sezione, pur non pretendendo di essere esaustiva in ogni aspetto, si occupa di introdurre gli strumenti utilizzati per progettare il prototipo hardware.

2.1 Arduino

Arduino è una piattaforma open-source per la prototipazione hardware [3] che negli ultimi anni sta avendo un grande successo. Ciò è dovuto alla sua estrema facilità di utilizzo che permette, con risorse molto limitate, di implementare soluzioni personalizzate e adatte ad un grande spettro di utilizzi.

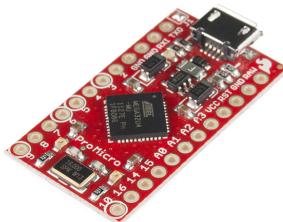


Figura 2.1: Arduino Pro Micro, prodotto dalla SparkFun Electronics.

Per queste ragioni, si è scelto di utilizzare Arduino per la realizzazione del dispositivo di registrazione dei movimenti. In particolare è stata utilizzata una sua versione più compatta, la *Pro Micro*, progettata dalla *SparkFun Electronics*.

Questa versione, essendo più compatta, si presta molto alla creazione di un dispositivo hand-held. È dotata di un micro-processore Atmega32U4 da 16MHz, programmabile tramite la IDE ufficiale di Arduino, connettendo il dispositivo via USB al computer.

Il Pro Micro presenta molte possibilità per quanto riguarda l'I/O: fornisce 12 porte utilizzabili come I/O digitali, 9 porte come input analogico e 5 porte capaci di PWM (Pulse Width Modulation), che permettono di connettere il modulo alla quasi totalità dei sensori in commercio. È inoltre possibile connettere l'Arduino ad altri moduli, utilizzando protocolli comuni come I2C e SPI.

Per funzionare, il modulo ha bisogno di una tensione di alimentazione pari a 5 volt, che può essere fornita secondo varie modalità, la più comune è tramite USB. Lo schema del circuito e delle porte utilizzate in questo progetto, è analizzato nel Capitolo 3.

2.2 Switch Meccanici

Lo *switch* meccanico, più comunemente noto come “bottone” o “pulsante”, è la tipologia più semplice di sensore. La sua funzione è quella di permettere, o bloccare, il passaggio di corrente attraverso un circuito. Consiste normalmente di un insieme di uno o più contatti elettrici che, a seguito di una pressione, entrano in contatto chiudendo il circuito [4].



Figura 2.2: Vari tipi di Switch meccanici.

A livello digitale, uno switch può essere codificato come un bit, dove il valore 0 rappresenta il circuito aperto e 1 rappresenta il circuito chiuso. È importante notare come questa convenzione non sia rigida, ma si adatti alla specifica applicazione. Non è quindi insolito utilizzare una codifica invertita rispetto a quella presentata.

Collegando uno switch ad un circuito logico, al fine di poter leggere correttamente il suo stato, è importante introdurre un altro componente: la *resistenza di pull-up*. Essa permette di mantenere il segnale ad un livello logico valido, evitando che vada in tri-state nel caso in cui lo switch non sia premuto. In quest'ultimo caso, infatti, il voltaggio fornito dalla resistenza di pull-up prevale sullo switch e porta l'ingresso ad un livello logico alto [5].

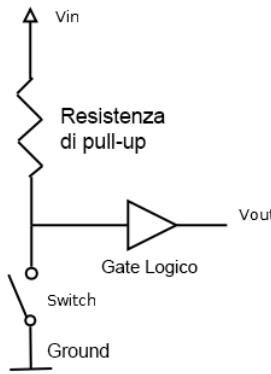


Figura 2.3: Esempio di circuito che utilizza una resistenza di pull-up.

Nel caso di Arduino, lo stato di uno switch può essere letto collegandolo ad una porta preventivamente impostata in lettura. Ad esso sarà quindi associato un valore booleano che, in base allo stato dello switch al momento della lettura, potrà essere HIGH oppure LOW.

Facendo riferimento allo schema riportato in Figura 2.3 e collegando lo switch al pin 2 dell'Arduino, possiamo impostare la porta in input ed effettuare la lettura nel modo descritto in Figura 2.4.

Bisogna inoltre specificare che il processore utilizzato dall'Arduino possiede dei resistori di pull-up integrati che permettono, tramite opportuni comandi via software, di ottenere risultati analoghi a quelli dell'esempio sopra riportato, senza la necessità di inserire un resistore esterno. Basterà sostituire INPUT con INPUT_PULLUP nella direttiva che imposta il pin in modalità di lettura [6].

```

// Esempio Switch Arduino

void setup() {
    // Inizializza il pin 2 come INPUT
    pinMode(2, INPUT);
}

void loop() {
    // Legge lo stato del pin 2 che rappresenta lo stato dello switch.
    // Se lo switch è premuto restituirà il valore LOW, se non premuto HIGH.
    int switchState = digitalRead(2);
}

```

Figura 2.4: Esempio di utilizzo di uno switch in ambiente Arduino.

2.3 L'integrato MPU-6050

Il primo passo per creare un dispositivo di riconoscimento gestuale è renderlo capace di registrare i movimenti. Per questo motivo è stato incluso l'integrato *MPU-6050* prodotto dalla InvenSense, e in particolare una sua *breakout board*, la *GY-521*, in modo da permetterne il collegamento ad Arduino.

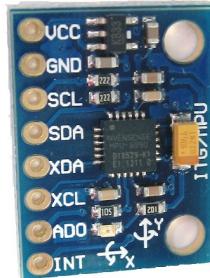


Figura 2.5: La breakout board GY-521. Al centro si può notare l'integrato MPU-6050.

Questo modulo comprende al suo interno alcuni sensori, tra cui un giroscopio ed un accelerometro, e permette di rilevare con precisione il movimento del dispositivo nelle tre dimensioni. È progettato per applicazioni a basso consumo energetico ed è in grado di comunicare con l'esterno tramite il protocollo *I₂C* [7]. Bisogna inoltre specificare che l'integrato presenta al suo interno un *DMP (Digital Motion Processor)* che permette di svolgere complessi calcoli con i dati raccolti, anche se il suo utilizzo risulta particolarmente difficoltoso [8] e, nel caso del progetto trattato, non necessario.

Il settaggio del modulo e la lettura dei sensori da parte di Arduino sono resi possibili da una serie di registri interni, accessibili tramite il protocollo I2C. Prima di poter effettuare le letture è necessario “svegliare” preventivamente il modulo, facendolo uscire dallo stato di *sleep*, tramite il settaggio del registro PWR_MGMT_1 a 0.

Le registrazioni dei sensori sono rappresentate da interi a 16 bit. Per questo motivo ogni valore è diviso in due registri da 8 bit, che rappresentano rispettivamente gli 8 bit più significativi e gli 8 bit meno significativi. Per ogni valore sarà quindi necessario effettuare due letture, a cui seguiranno uno *shift* dei bit più significativi ed una concatenazione di quelli meno significativi.

```
// Includo la libreria che permette la comunicazione I2C
#include<Wire.h>

const int MPU_addr=0x68; // Indirizzo I2C dell'integrato MPU-6050

// Interi a 16 bit che conterranno i valori letti dai sensori
int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;

// Impostazione iniziale
void setup(){
    Wire.begin(); // Inizializzo l'I2C.
    Wire.beginTransmission(MPU_addr); // Inizio la trasmissione con MPU-6050.

    Wire.write(0x6B); // Seleziono il registro PWR_MGMT_1
    // che determina lo stato del MPU-6050.
    Wire.write(0); // Imposto a 0, svegliando MPU-6050.

    Wire.endTransmission(true); // Fine della trasmissione con MPU-6050.
}

// Loop continuo in cui leggo i valori dei sensori
void loop(){
    Wire.beginTransmission(MPU_addr); // Inizio la trasmissione con MPU-6050.
    Wire.write(0x3B); // Imposto l'indirizzo a 0x3B (ACCEL_XOUT_H)
    Wire.endTransmission(false); // Mantengo attiva la trasmissione

    Wire.requestFrom(MPU_addr,14,true); // Richiedo un totale di 14 registri
    AcX=Wire.read()<<8|Wire.read(); // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
    AcY=Wire.read()<<8|Wire.read(); // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
    AcZ=Wire.read()<<8|Wire.read(); // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
    Tmp=Wire.read()<<8|Wire.read(); // 0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
    GyX=Wire.read()<<8|Wire.read(); // 0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
    GyY=Wire.read()<<8|Wire.read(); // 0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
    GyZ=Wire.read()<<8|Wire.read(); // 0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)
}
```

Figura 2.6: Codice utilizzato per la lettura dei valori dal modulo MPU-6050.

Al fine di velocizzare il processo di lettura è possibile effettuare un trasferimento *burst*, segnalando l'indirizzo del registro di partenza e il numero di

registri totali. In Figura 2.6 è riportato un esempio di lettura dei sensori interni al MPU-6050 da parte di Arduino:

Nelle ultime righe di codice si può notare come ogni valore sia caratterizzato da due letture, di cui la prima viene sottoposta ad uno shift di 8 bit a sinistra e la seconda viene concatenata a destra.

Le modalità di collegamento dell'integrato MPU-6050 all'Arduino verranno discusse in dettaglio nel Capitolo 3.

2.4 Modulo Bluetooth: HC-06

Per ottenere un dispositivo più pratico, il primo passo è stato renderlo senza fili. Eliminando infatti la necessità di dover essere costantemente collegato ad un cavo USB, aumenta la libertà di esecuzione dei gesti. Le strade per realizzare un dispositivo *wireless* sono molteplici, ma considerando il tipo di dato trasmesso, si è ritenuta come scelta più appropriata la tecnologia *Bluetooth*.

Il Bluetooth è uno standard di trasmissione senza fili, molto efficace per inviare modeste quantità di dati a brevi distanze [9]. Ha inoltre il vantaggio di essere integrato nella maggior parte dei dispositivi mobili recenti.

Al fine di permettere una comunicazione Bluetooth all'Arduino, si è deciso di utilizzare il modulo *HC-06*. Esso permette di stabilire una connessione di tipo seriale con qualsiasi dispositivo dotato di Bluetooth e ciò rende la sua implementazione software quasi immediata, non richiedendo particolari modifiche rispetto alla versione utilizzante una connessione USB.

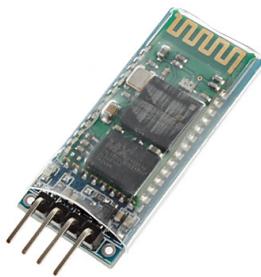


Figura 2.7: Il modulo HC-06.

Il modulo presenta 4 porte di collegamento: due utilizzate per l'alimentazione

e due per la comunicazione. Queste ultime due porte, la RX e la TX, vengono utilizzate nella trasmissione seriale e rappresentano rispettivamente la porta di entrata e di uscita. Il metodo più semplice per utilizzare il modulo con Arduino è quello di simulare una connessione seriale, tramite la libreria *SoftwareSerial* [10]. Segue un esempio di utilizzo in ambiente Arduino:

```
// Includo la libreria necessaria a simulare un canale di comunicazione seriale
#include <SoftwareSerial.h>

// Inizializzo il canale di comunicazione seriale Bluetooth
// Collegando il Pin TX del HC-06 al pin 2 dell'Arduino
// ed il Pin RX del HC-06 al pin 3 dell'Arduino
SoftwareSerial bluetooth(3,2);

// Impostazione iniziale
void setup()
{
    // Avvio il canale di comunicazione Bluetooth con una baud rate di 9600
    bluetooth.begin(9600);
}

void loop()
{
    // Invio ripetutamente la stringa "message" tramite Bluetooth
    bluetooth.print("message");
}
```

In questo esempio, l'Arduino invia ripetutamente la stringa “message” tramite Bluetooth. Bisogna inoltre specificare il significato di *baud rate*, che rappresenta il numero di simboli trasmessi in un secondo [11], e che può più semplicemente essere vista come velocità di trasmissione.

2.5 Sensore Piezoelettrico

Durante la trattazione di un caso di studio, è stato necessario rilevare con precisione le vibrazioni di una determinata superficie e, per farlo, si è scelto di utilizzare un *sensore piezoelettrico*: questo è un dispositivo che, tramite l'effetto piezoelettrico, permette di misurare cambiamenti di pressione, accelerazione, temperatura o forza convertendoli in una carica elettrica [12].

Grazie a queste proprietà, il sensore è in grado di misurare con grande precisione le vibrazioni propagate attraverso una superficie, generando ai terminali una tensione proporzionale all'energia meccanica rilevata. Questa differenza di potenziale, essendo continua nei valori, è un segnale analogico e per leggerlo tramite un microcontrollore è necessario un convertitore analogico-

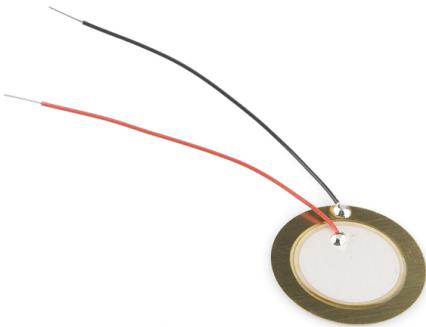


Figura 2.8: Sensore Piezoelettrico.

digitale. Questo componente è convenientemente incluso nella maggior parte delle schede Arduino e ciò rende la lettura del sensore molto semplice. Tuttavia il segnale in uscita è troppo debole per poter essere agevolmente letto dal microcontrollore. Per questo motivo, come sarà meglio spiegato nel capitolo 6, è stato necessario aggiungere un amplificatore in modo da potenziare la tensione in ingresso all'Arduino.

2.6 Amplificatore LM386

Data la necessità di un amplificatore e la sua disponibilità al momento del progetto, si è scelto di utilizzare un *LM386* per potenziare il segnale del piezoelettrico. Questo integrato permette di amplificare il segnale in ingresso da 20 a 200 volte e richiede una bassa tensione di alimentazione [13], ciò lo ha reso il perfetto candidato per il progetto in esame.

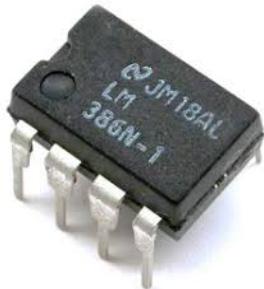


Figura 2.9: Amplificatore LM386.

2.7 Alimentazione

Per funzionare correttamente, l'Arduino richiede una tensione di alimentazione pari a 5 volt. È possibile fornire questa corrente in vari modi, il più semplice dei quali è tramite cavo USB. Quest'ultimo infatti permette sia di alimentare il microcontrollore che di scambiare dati con esso.

Uno dei requisiti del dispositivo era essere senza fili, è stato quindi necessario trovare una soluzione di alimentazione portabile ed efficiente. Dopo un'attenta valutazione delle possibilità, si è optato per l'utilizzo di una batteria portatile, o *power bank*, dati i suoi evidenti vantaggi dal punto di vista pratico e implementativo. Il power bank fornisce infatti una connessione a 5 volt, è inoltre portatile, ricaricabile e compatto. Date queste caratteristiche, si è ritenuto fosse la scelta più indicata per il progetto.

Si può notare come i power bank forniscono una connessione USB e questo li rende particolarmente pratici in quanto è possibile utilizzarli con un Arduino per mezzo dello stesso cavo usato per programmarli.



Figura 2.10: Esempi di Power Bank.

Capitolo 3

Progetto del prototipo hardware

Come già detto in precedenza, si è deciso di realizzare un dispositivo che permettesse di registrare i movimenti e inviasse in tempo reale i dati raccolti ad un computer, per una successiva elaborazione. Questo capitolo verterà sulla progettazione del dispositivo e sulla sua successiva realizzazione.

3.1 Progettazione del Circuito

Per realizzare il dispositivo è stata necessaria una fase di progettazione del circuito. In primo luogo sono state studiate le tecniche e i protocolli utilizzati per connettere i vari moduli all'Arduino e successivamente sono stati tracciati i collegamenti tra i pin, facendo riferimento ai *datasheet* dei vari componenti. Utilizzando poi il software open-source *Fritzing* è stato realizzato lo schema del circuito, riportato in Figura 3.1:

Dal circuito si può notare la presenza di diversi switch e led, utilizzati per interagire con il dispositivo, nonché l'accelerometro MCU-6050, collegato all'Arduino tramite i pin D2 e D3 che permettono di utilizzare il protocollo I2C. È inoltre presente il modulo HC-06, collegato tramite i pin D10 e D16, che permette al dispositivo di comunicare via Bluetooth.

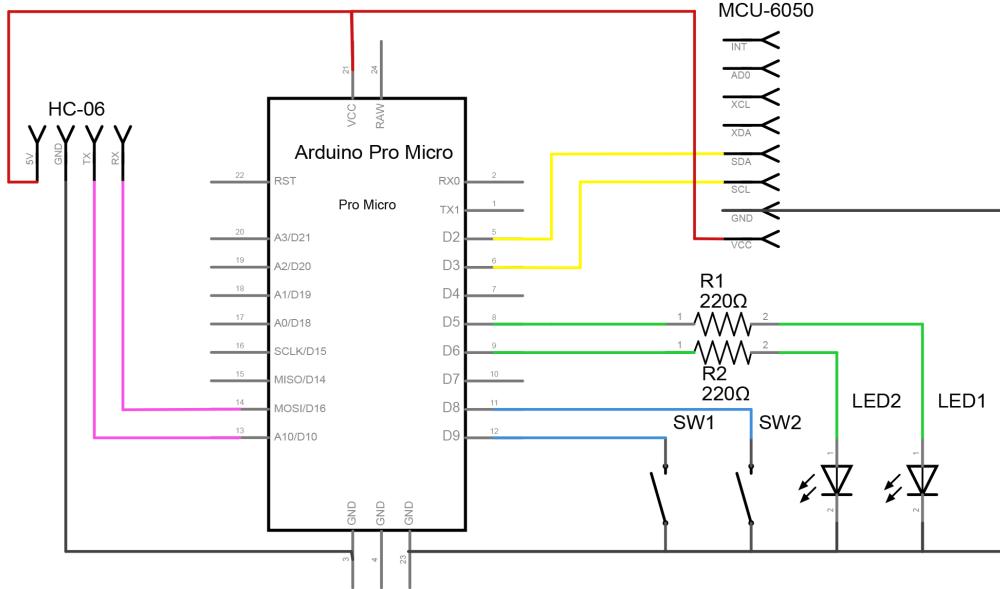


Figura 3.1: Schema circuitale del dispositivo.

3.2 Realizzazione del Prototipo

Dopo aver progettato il circuito, si è studiata la disposizione dei componenti su una *basetta millefori*, in modo tale da riuscire a saldare i vari moduli e ottenere un prototipo funzionante, mostrato in Figura 3.2.

I componenti sono stati collegati tramite degli *header*, un particolare tipo di connettore che permette di sostituire facilmente gli elementi circuituali in caso di guasto. Si può inoltre notare il powerbank utilizzato per alimentare il dispositivo che, tramite collegamento usb, fornisce i 5 volt necessari al funzionamento dello stesso.

3.3 Software del microcontrollore

Per terminare lo sviluppo del prototipo, è stato necessario realizzare il software per il microcontrollore: come già detto in precedenza, lo scopo del dispositivo è quello di leggere i dati dal modulo MCU-6050 e inviarli ad un computer per una successiva elaborazione. Gli aspetti legati all'utilizzo dell'accelerometro e del modulo Bluetooth sono già stati trattati nel capitolo precedente, resta da valutare il formato di trasmissione.

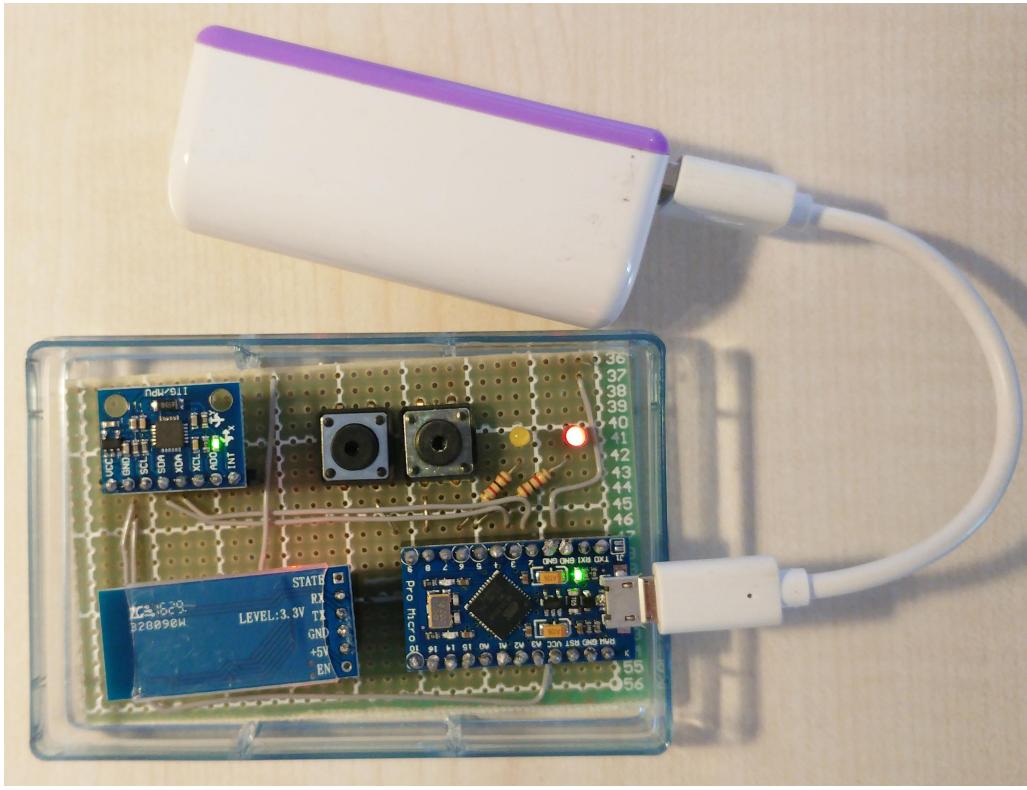
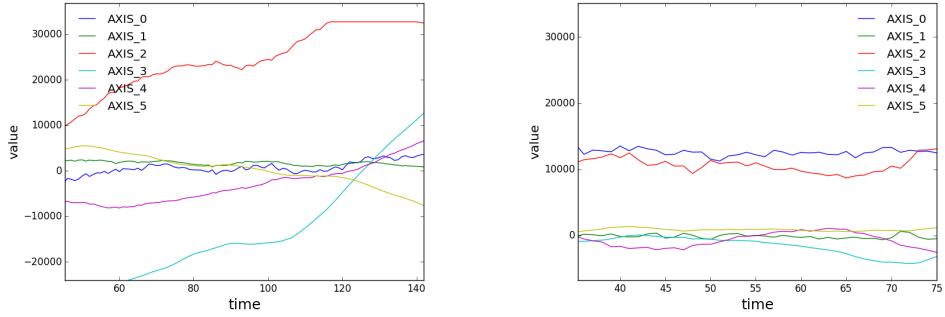


Figura 3.2: Prototipo del dispositivo.

3.3.1 Formato di Trasmissione

Per trasmettere i dati si è scelto di utilizzare un formato che permettesse una certa flessibilità riguardo ai tipi e al numero di sensori utilizzati. Ognuno di essi è caratterizzato da una serie di valori: ad esempio, il modulo MPU-6050, è determinato da 6 valori che rappresentano l'accelerazione e la rotazione nei 3 assi cartesiani.

Lo scopo dell'intero sistema è quello di riconoscere i gesti, è quindi necessario definire cosa ne caratterizza uno: un gesto è una sequenza di *frame*, dove un frame è l'insieme dei valori, raccolto dai vari sensori, in uno specifico istante di tempo. Possiamo quindi definire un gesto come “l’immagine” di uno specifico movimento, caratterizzato dai valori di accelerazione e rotazione in un determinato intervallo di tempo. A titolo d’esempio, sono riportati i grafici di due differenti gesti.



Assumendo quindi di avere una serie di sensori tali per cui il numero totale di assi è N e considerando un intervallo di tempo che contiene M frame, i dati raccolti vengono codificati nel formato indicato in Figura 3.3:

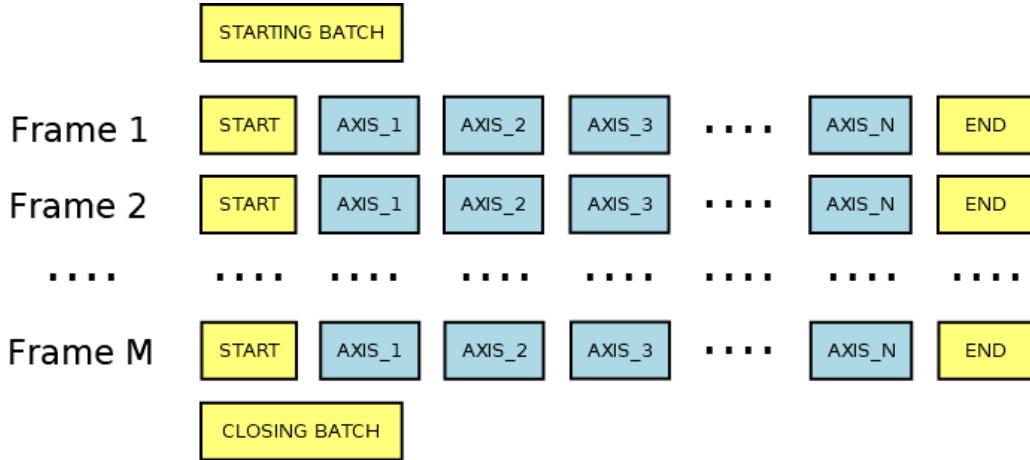


Figura 3.3: Formato di trasmissione.

In particolare, un gesto è caratterizzato da un prologo ed un epilogo che ne determinano l'inizio e la fine. Nel caso sopra riportato, essi sono rappresentati dai segnali di **STARTING_BATCH** e **CLOSING_BATCH**. All'interno di questi sono compresi i *data frame*, contenenti i valori degli assi in un determinato istante di tempo. Ognuno di questi frame è delimitato dai segnali di **START** ed **END**, utilizzati per verificare che la comunicazione sia avvenuta correttamente.

I dati, indipendentemente dal fatto che siano trasferiti via USB o Bluetooth, vengono inviati al computer con una connessione seriale. Questo implica che il formato di trasmissione indicato precedentemente debba es-

sere formattato come una stringa, utilizzando dei separatori per distinguere le varie parti. Un esempio di gesto trasmesso rispettando il formato sopra descritto è riportato in Figura 3.4.

```
STARTING BATCH
START 2544 6760 3996 5131 1277 -5211 END
START -9348 956 8628 5959 -3307 -3566 END
START -13768 1504 10692 2903 -2576 -1539 END
START -13408 600 12560 319 200 160 END
START -7764 716 9048 -1231 2793 2122 END
START -7248 940 10236 -1681 665 2681 END
CLOSING BATCH
```

Figura 3.4: Esempio di gesto trasmesso tramite connessione seriale.

Si può notare come i valori degli assi siano separati da dei caratteri spazio, mentre i differenti frame sono separati da un terminatore di riga. Questo permette al computer in ricezione di analizzare ed eseguire il *parsing* dei valori e dei segnali di controllo in maniera semplice.

3.4 Considerazioni

Il prototipo realizzato, per quanto funzionante, non è perfetto: la dimensione e l’usabilità possono sicuramente essere migliorate. In particolare, utilizzando attrezzature industriali, sarebbe possibile realizzare dispositivi compatti al punto tale da essere comodamente indossati o inglobati all’interno di altri apparecchi. Questo comporterebbe un nuovo insieme di possibilità e aprirebbe la strada alla creazione di dispositivi sempre più “smart”.

Capitolo 4

Sviluppo della libreria di Gesture Recognition

L'obiettivo di questo studio è stata la creazione di un sistema di riconoscimento gestuale completo. Una volta realizzato il dispositivo, il passo successivo è stato quello di sviluppare una libreria che permettesse di studiare e realizzare in maniera veloce ed efficiente progetti riguardanti la gesture recognition.

4.1 Linguaggi e librerie utilizzate

Per lo sviluppo della libreria è stato scelto *Python* come linguaggio di programmazione: esso è un linguaggio ad alto livello, orientato ad oggetti e dinamico, estremamente utilizzato in ambito scientifico [14]. Ciò è dovuto alla presenza di numerose librerie, come *Numpy* e *Scipy*, che permettono di effettuare calcoli in maniera estremamente efficiente e veloce [15]. Inoltre, è stata utilizzata anche *Scikit Learn*, una libreria per il *Machine Learning* particolarmente versatile e semplice da usare [16].

4.2 Design Pattern e scelte strutturali

Dopo una serie di esperimenti iniziali finalizzati a valutare la fattibilità del progetto, si è subito capito quanto un design modulare fosse importante. È stata quindi riscritta l'intera libreria, allo scopo di renderla completamente modulare. Questo ha portato numerosi vantaggi:

- **Personalizzazione:** la modularità dei componenti permette di scambiare e sostituirli con estrema facilità in modo da adattarli alle esigenze più specifiche.
- **Testabilità:** la separazione dei vari moduli ha permesso di testare estensivamente le varie parti, rendendo il progetto più stabile.
- **Manutenibilità:** essendo i compiti di ogni componente ben definiti, la realizzazione degli stessi e la correzione di eventuali errori si è rivelata molto più semplice.

L’obiettivo del sistema è quello di rispondere in maniera appropriata ad uno specifico gesto a seguito di una sua ricezione. Questa situazione viene comunemente definita a “*gestione di eventi*” ed è caratterizzata da un flusso di programma determinato da specifici eventi [17], in questo caso i gesti. In particolare, nell’applicativo è presente un’entità che realizza l’*event loop*, ossia un insieme di istruzioni che attende dati dai sensori e successivamente notifica eventuali eventi alle entità in ascolto [18].

Per realizzare questo sistema, si è ritenuta essere la scelta più appropriata il pattern *Observer*. Esso si basa su un’entità, il *soggetto*, che mantiene una lista dei suoi dipendenti, gli *osservatori*, a cui notifica automaticamente gli eventi, normalmente chiamando uno dei loro metodi [19].

4.3 Flusso di gestione dei gesti

In maniera astratta, il flusso di gestione dei gesti può essere schematizzato come in Figura 4.1:

L’event loop rappresenta il soggetto del pattern Observer, un’entità che rimane in attesa di dati dai sensori e, quando questi risultano disponibili, si occupa di propagare l’evento, e quindi i dati ricevuti, agli strati inferiori. Saranno quindi questi ultimi, gli osservatori, a processare i vari dati e a determinare a quale gesto essi corrispondano. Una volta determinata la gestione, l’osservatore notifica il gestore degli eventi, che si occupa di eseguire il codice necessario a realizzare le specifiche funzionalità richieste.

4.4 Pipeline di elaborazione dei dati

Per quanto lo schema presentato nel paragrafo precedente risulti concettualmente valido, l’elaborazione dei dati che porta alla determinazione di

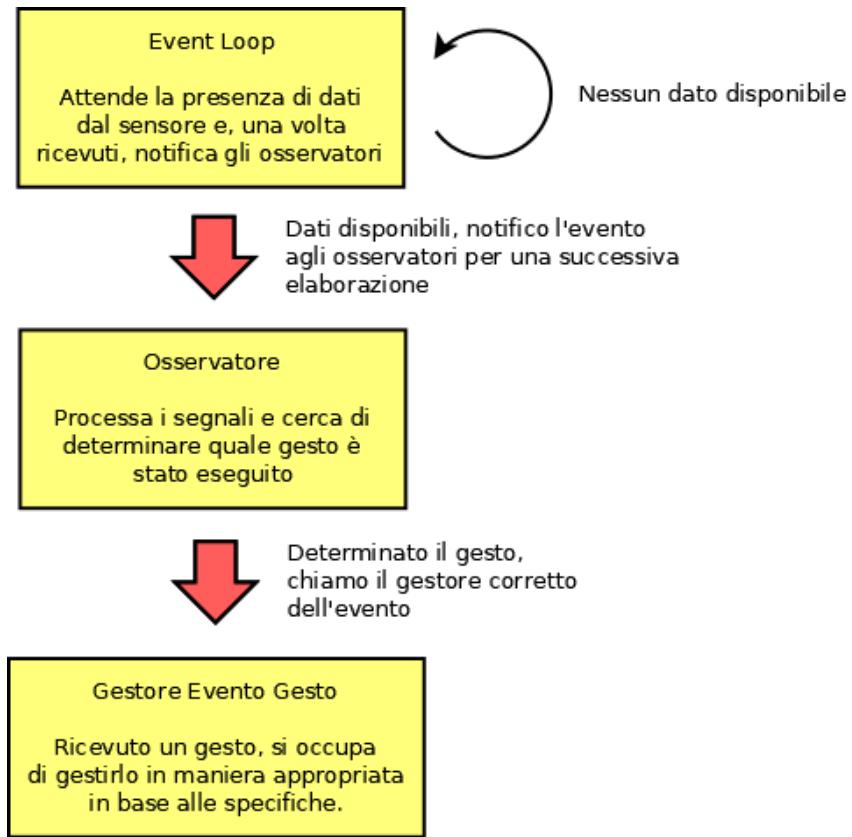


Figura 4.1: Flusso di gestione dei gesti.

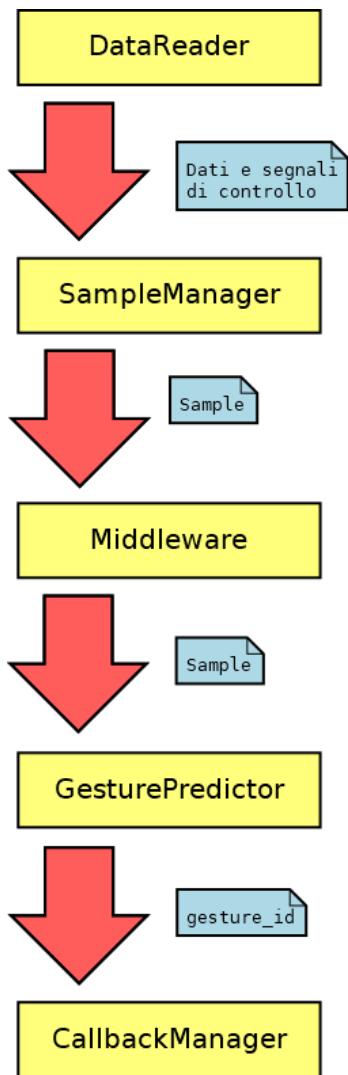
un gesto è complessa e difficilmente esprimibile come un'unica entità.

In realtà, proprio per rispettare la premessa di un design modulare, si è decisa una struttura fortemente gerarchizzata, con una divisione dei compiti marcata.

La *pipeline* di elaborazione dei dati, schematizzata in Figura 4.2, è formata da una serie di componenti che si occupano di processare progressivamente i dati ricevuti, fornendo una visione sempre più astratta agli strati inferiori. Ogni modulo implementa il pattern Observer, in modo da propagare in maniera efficiente l'evento a tutti gli strati successivi. Nei seguenti sotto-paragrafi verranno esposte le motivazioni e gli scopi di ciascun componente.

4.4.1 DataReader

Il **DataReader** è il modulo principale dell'intera pipeline: si interfaccia direttamente con i sensori da cui legge i dati.



Si occupa inoltre di interpretare questi valori e di intercettare eventuali errori, per poi inviare allo strato successivo dei frame contenenti dati di movimento e segnali di controllo. È il componente che implementa l'event loop, rimanendo in attesa di dati disponibili dal sensore e, qualora ce ne fossero, propagandoli allo strato successivo.

4.4.2 SampleManager

Il **SampleManager** è il modulo che, ricevuti i frame contenenti dati e segnali di controllo dal **DataReader**, si occupa di impacchettarli in appositi contenitori, detti *Sample*. Questi Sample realizzano il concetto di “Campione”, ovvero un insieme di frame che individuano un movimento, ma che non sono ancora associati a nessun gesto. Una volta impacchettato, il Sample viene inviato allo strato successivo che si occuperà di effettuare ulteriori manipolazioni sui dati ricevuti.

4.4.3 Middleware

Il **Middleware** è un componente che si occupa di processare e manipolare i Sample. Data la sua natura, il numero di middleware in una pipeline è estremamente variabile. Esso infatti prende in ingresso e restituisce un Sample, applicando una serie di trasformazioni nel processo. Ciò implica che se il Sample non richiede alcuna manipolazione, questo strato può essere completamente omesso. Allo stesso modo, se invece richiede diverse trasformazioni, è possibile utilizzare multipli middleware in cascata.

4.4.4 GesturePredictor

Il `GesturePredictor` è il componente che si occupa di associare un gesto al Sample ricevuto. È importante notare come la logica utilizzata per stimare la gestione dipenda dalla specifica implementazione. Una volta ottenuto il Sample dallo strato precedente, invia a quello successivo una stringa, chiamata `GESTURE_ID`, che identifica univocamente il gesto.

4.4.5 Callback Manager

Il `CallbackManager` si occupa di chiamare il gestore dell'evento, qualora sia stato ricevuto il gesto ad esso associato. In fase iniziale, si configura il *binding* tra i `GESTURE_ID` e le funzioni callback, ovvero funzioni chiamate nel caso venga ricevuto l'identificativo del gesto corrispondente, che si occupano di gestire l'evento.

4.5 Interfacce e Classi astratte

Una volta definite le entità fondamentali e i loro compiti, si è passati alla modellizzazione delle interfacce e delle classi astratte. In particolare, si è cercato di individuare i metodi e le procedure propri di ciascuna entità, indipendenti dalla specifica implementazione.

4.5.1 AbstractDataReader

L'`AbstractDataReader` implementa i metodi necessari alla realizzazione del pattern Observer, in particolare quelli per registrare un `SampleManager` e per propagare dati e segnali di controllo. Inoltre specifica i metodi astratti necessari ad aprire e chiudere una connessione, indicando anche il metodo `mainloop` che rappresenta l'event loop del sistema.

Nello schema è presente anche l'enumerativo `ControlSignal` che contiene i possibili segnali di controllo che un qualunque `DataReader` è in grado di inviare.

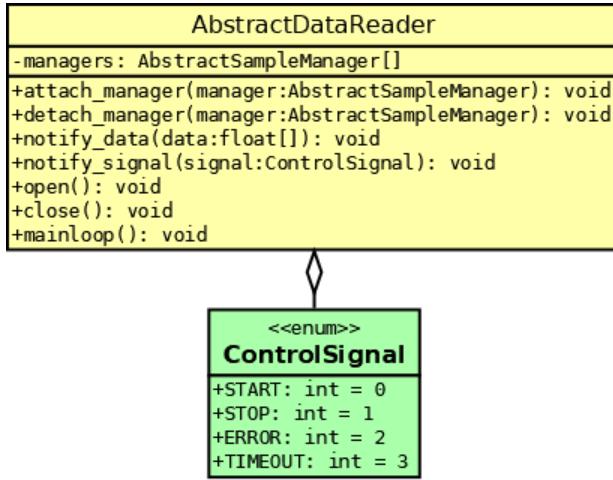


Figura 4.3: UML dell'AbstractDataReader.

4.5.2 Sender e AbstractSampleManager

Le entità in grado di inviare un Sample allo strato successivo sono molteplici. Si è quindi ritenuto appropriato definire il concetto di **Sender**, ovvero una classe che implementa i metodi necessari a propagare un Sample ad un opportuno ricevitore, secondo le modalità descritte dal pattern Observer.

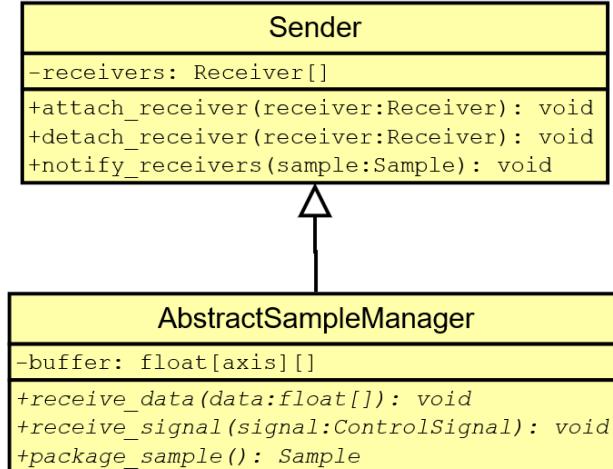


Figura 4.4: UML del Sender e dell'AbstractSampleManager.

A questo punto è stato possibile definire l'**AbstractSampleManager**, un'entità in grado di ricevere dati e segnali di controllo da un **DataReader** e impac-

chettarli in un Sample. I criteri con cui avviene questo processo dipendono dalla specifica implementazione e non sono specificati nella classe astratta. Inoltre, l'`AbstractSampleManager` estende la classe `Sender`, questo permette di propagare il Sample appena ottenuto agli strati successivi.

4.5.3 Receiver e AbstractMiddleware

Con un ragionamento analogo a quello del sotto-paragrafo precedente, si è definita l'interfaccia `Receiver`, che permette di ricevere un Sample, opportunamente inviato da un `Sender`. A differenza di quest'ultimo però, non è definito come classe astratta, a causa del fatto che le azioni da eseguire, una volta ricevuti i dati, variano in base alla specifica applicazione.

Si è quindi definito l'`AbstractMiddleware`, un'entità che, estendendo il `Sender` e implementando il concetto di `Receiver`, è in grado di ottenere un Sample dallo strato superiore, processarlo, e inviarlo allo strato successivo. Le manipolazioni effettuate dipendono dalla specifica implementazione, che definisce la logica sovrascrivendo il metodo `process_sample`.

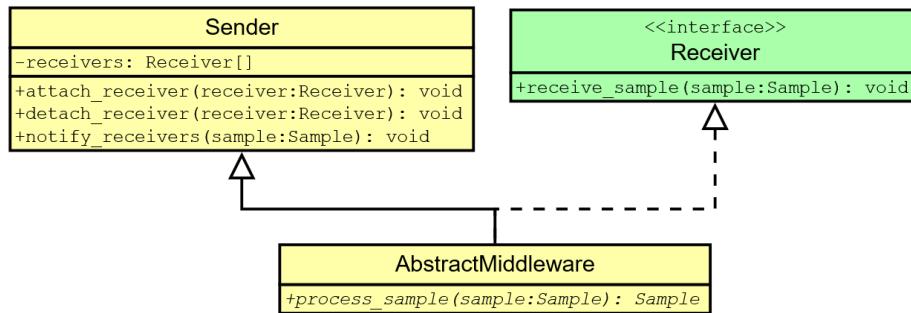


Figura 4.5: UML del Receiver e dell'AbstractMiddleware.

4.5.4 AbstractGesturePredictor

L'`AbstractGesturePredictor` implementa i meccanismi necessari alla propagazione dei `GESTURE_ID` agli strati successivi e definisce la firma del metodo `predict`. Sarà infatti questo metodo, opportunamente implementato nelle classi derivate, a fornire il gesto associato al Sample ricevuto. Il meccanismo di ricezione, come nell'`AbstractMiddleware`, si ottiene implementando l'interfaccia `Receiver`.

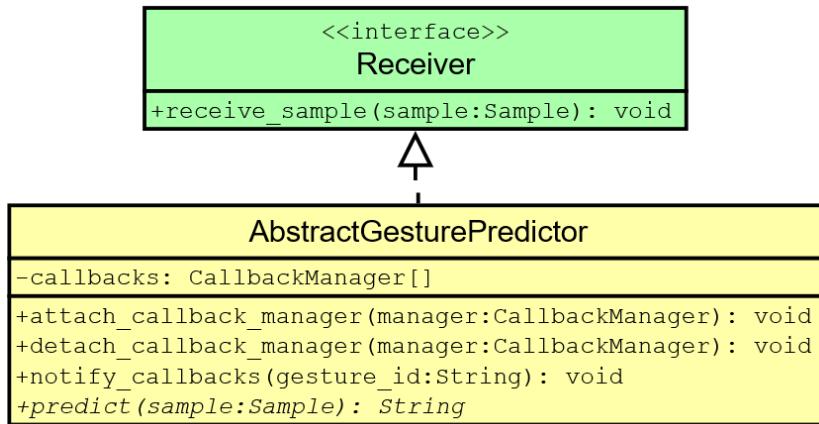


Figura 4.6: UML dell'AbstractGesturePredictor.

4.5.5 Classifier e AbstractClassifier

A differenza dei precedenti, questo componente non è direttamente incluso nella pipeline di elaborazione, ma rappresenta comunque una parte importante del sistema.

Il **Classifier** è un’entità che, dato un insieme di Sample a cui sono associati dei gesti, realizza un modello contenente le “regole” che li mettono in relazione. In altre parole, si può dire che “impara” come i Sample sono legati ai gesti e successivamente è in grado di determinare il gesto associato ad un Sample mai visto prima. Per questo motivo, i **GesturePredictor** si avvalgono frequentemente di **Classifier** al fine di stimare il gesto associato ad un Sample ricevuto.

In generale, il processo di allenamento di un **Classifier** inizia con la creazione di un *Dataset*, ovvero un insieme di Sample, ognuno associato ad un gesto. In seguito avviene il *training* del modello, che dipende dalla specifica implementazione del classificatore. A questo punto si procede al salvataggio del modello, in maniera tale da poterlo riutilizzare velocemente in futuro, senza doverlo allenare nuovamente.

Per fornire all’utilizzatore una visione astratta del processo, i metodi di basso livello, come il caricamento del dataset e il salvataggio del modello, sono implementati dall’**AbstractClassifier**. Per realizzare un classificatore è quindi sufficiente estendere la classe astratta e sovrascrivere i metodi

legati al modello, implementando la logica del `train_model` e del `predict`.

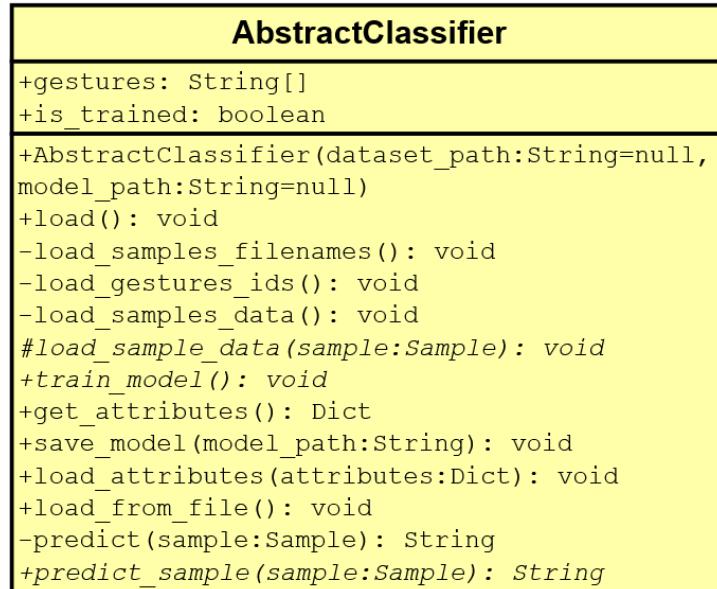


Figura 4.7: UML dell'AbstractClassifier.

4.6 Principali implementazioni dei componenti

Una volta definite le entità astratte, il passo successivo è stato implementare le classi derivate. In seguito verranno elencate solo le principali mentre quelle più specifiche verranno riprese nei capitoli dedicati ai singoli casi di studio in modo da fornire anche il contesto e le motivazioni alla base di esse.

4.6.1 SerialDataReader

Il dispositivo realizzato nel capitolo precedente si occupa di leggere e inviare i dati del sensore ad un computer. Questo scambio avviene tramite trasmissione seriale ed è indipendente dal mezzo scelto. Il primo passo è stato quindi realizzare un'entità che permettesse di ricevere i dati da una porta seriale.

Il `SerialDataReader` si occupa di leggere i dati inviati dal dispositivo, di interpretarli e di propagarli allo strato successivo. Inoltre accetta come

parametro il numero di assi previsto per ogni frame: se infatti arriva un numero di valori diverso rispetto a quello atteso, significa che si è verificato un errore, e il **SerialDataReader** lo propaga allo strato successivo. Per maggiore chiarezza, viene riportato in Figura 4.8 un esempio di traduzione dei dati, ricevuti dalla porta seriale sotto forma di stringhe, da parte del **SerialDataReader**.

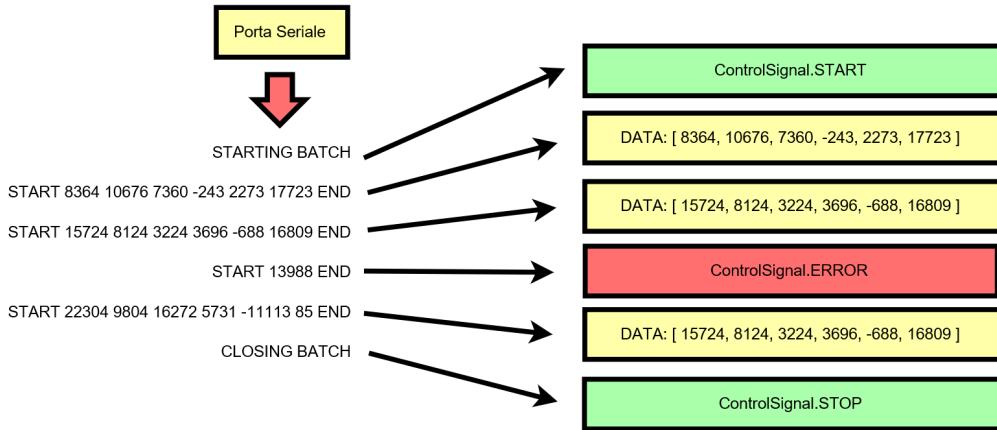


Figura 4.8: Esempio di traduzione dei dati da parte del **SerialDataReader**.

Si può notare come vengano propagati i segnali di controllo di inizio e fine campione e inoltre, nel caso in cui il numero di valori sia sbagliato, venga inviato un segnale di errore. Infine, nei frame contenenti dati, viene effettuato il parsing e la conversione dei valori.

4.6.2 DiscreteSampleManager

Il **DiscreteSampleManager** rappresenta il tipo più semplice di **SampleManager** e, come tale, si occupa di impacchettare i dati ricevuti da un **DataReader**. La registrazione del Sample inizia nel momento in cui un segnale di **START** viene ricevuto e procede fino all'arrivo di un segnale di **STOP**. Di conseguenza, il numero di frame contenuti non è fisso, ma è in generale diverso per ogni campione. In caso venga utilizzato con il dispositivo realizzato nel capitolo precedente, l'inizio e la fine della registrazione coincideranno con gli istanti in cui il pulsante viene rispettivamente premuto e rilasciato. Il pulsante è quindi l'elemento che discrimina i campioni e li separa l'uno dall'altro.

Inoltre, per evitare la creazione di Sample troppo corti, è previsto un

parametro che permette di specificare il numero minimo di frame. In caso questa condizione non sia verificata, il campione viene scartato. Ciò è molto utile al fine di filtrare eventuali oscillazioni presenti nei dati ricevuti.

4.6.3 StreamSampleManager

Successivi esperimenti hanno creato la necessità di un **SampleManager** che, a differenza del **DiscreteSampleManager**, permettesse di impacchettare i campioni in maniera continua, senza la necessità dei segnali di START e STOP per delimitarne l'inizio e la fine. Lo **StreamSampleManager** si occupa esattamente di questo.

Richiede come parametri aggiuntivi la dimensione di una finestra e l'offset tra di esse. Questo permette di esprimere come i Sample vengano impacchettati, dato in ingresso un flusso continuo di valori. In Figura 4.9 viene riportato un esempio di funzionamento.

Esempio con finestra=5 e offset=3

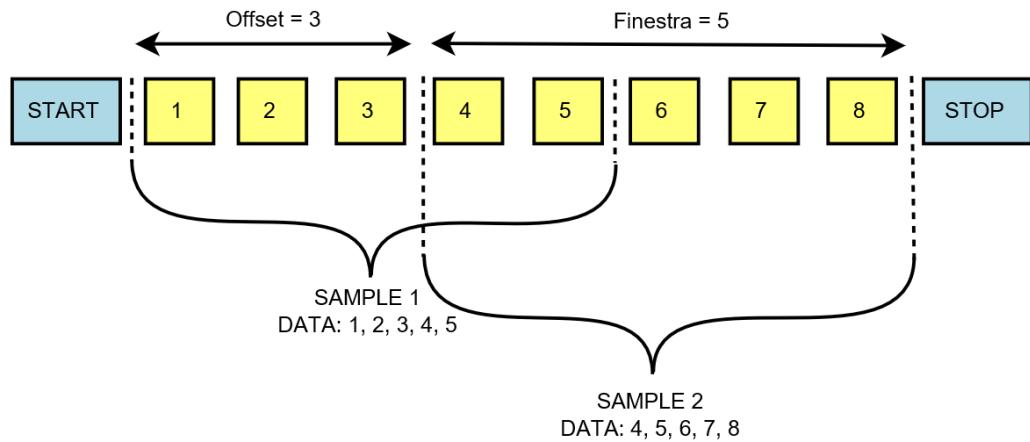


Figura 4.9: Meccanismo di raggruppamento dello StreamSampleManager.

Se l'offset è minore della dimensione della finestra, ci sarà una sovrapposizione dei campioni, che quindi condivideranno alcuni valori tra di loro.

4.6.4 FileGestureRecorder

Il `FileGestureRecorder` nasce dalla necessità di salvare i campioni su un supporto persistente, in modo da poterli successivamente ricaricare e analizzare. Implementando l’interfaccia `Receiver`, può essere agganciato alla pipeline principale e ricevere dei `Sample` direttamente da essa.

Prende in ingresso come parametro il direttorio di destinazione in cui il campione verrà salvato e si occupa di generare automaticamente un nome univoco per ogni file.

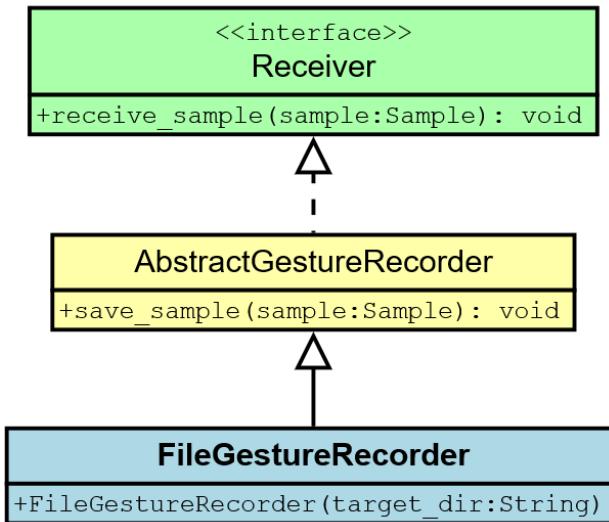


Figura 4.10: UML del FileGestureRecorder.

È importante specificare che i `Sample` in ingresso al `FileGestureRecorder` devono essere associati ad un `GESTURE_ID` per essere salvati. In fase di caricamento è infatti necessario sapere a quale gesto corrisponde un determinato movimento, soprattutto quando utilizzato da un classificatore in fase di allenamento.

4.6.5 SVMClassifier

Il `SVMClassifier` è un classificatore che utilizza l’algoritmo *Support Vector Machines*, o *SVM*, per determinare il gesto associato ad un determinato campione. Questi algoritmi, utilizzati tramite la libreria Scikit-learn, hanno il vantaggio di essere molto versatili. Ciò è dovuto alla possibilità di poter

cambiare la funzione di decisione, ovvero il meccanismo alla base della classificazione, per adattarsi in maniera ottimale ad una grande varietà di casi [20].

Uno dei fattori più importanti da considerare quando si utilizzano questi algoritmi è la scelta dei parametri, la cui corretta calibrazione è fondamentale per ottenere dei buoni risultati. Spesso, tuttavia, scegliere i giusti valori non è semplice. Per questo motivo, nel **SVMClassifier**, viene utilizzata una tecnica di tipo combinatorio: viene creata una griglia contenente i valori possibili per ogni parametro e successivamente viene testata ogni combinazione per valutare quale ha il risultato migliore. Ne deriva anche come il numero di valori debba essere limitato al minimo necessario per evitare l'aumento esponenziale del tempo computazionale.

4.6.6 MLPClassifier

Il **MLPClassifier** è un classificatore che come algoritmo utilizza un *Multi-Layer Perceptron*, un particolare tipo di rete neurale a più livelli che permette di identificare i campioni anche in situazioni non lineari [21].

Come nel caso del **SVMClassifier**, la scelta dei parametri è fondamentale per la buona riuscita del modello, viene perciò utilizzata la medesima tecnica combinatoria per scegliere i valori ottimali.

4.6.7 ClassifierPredictor

Il **ClassifierPredictor** si occupa di stimare il gesto associato ad un campione utilizzando un **Classifier**. È quindi il componente che permette di inserire i classificatori nella pipeline di elaborazione.

Capitolo 5

Caso di studio: Tastiera gestuale

Il primo caso in esame è stata la realizzazione di una tastiera “gestuale”, ovvero un dispositivo in grado di tradurre determinati movimenti in caratteri, permettendo quindi di scrivere eseguendo specifici gesti in aria. I movimenti sono scanditi dalla pressione di un pulsante che ne determina l'inizio e la fine, consentendo di isolare i singoli gesti. Il processo è sinteticamente descritto nella Figura 5.1.

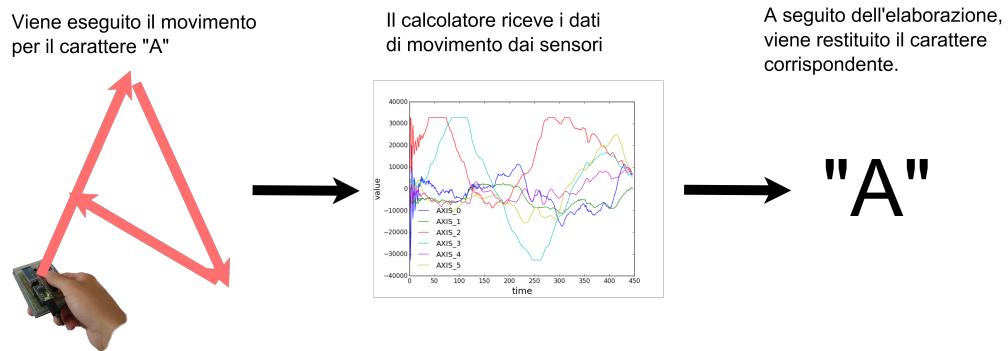


Figura 5.1: Processo di elaborazione della tastiera gestuale.

5.1 Creazione del Dataset

Il primo passo per realizzare il progetto è stata la creazione di un dataset contenente i campioni di ogni gesto previsto. Oltre ai 26 caratteri presenti nell’alfabeto inglese si è deciso di aggiungere anche due caratteri di controllo, corrispondenti allo spazio e al tasto cancella, per un totale di 28 gesti distinti. Per ottenere un dataset valido, è importante avere un grande numero di campioni per ogni gesto, in maniera tale da fornire ai classificatori più materiale su cui basarsi per creare un modello.

Per semplificare il lavoro è stato appositamente creato uno strumento per registrare, etichettare e salvare i movimenti su disco rigido in maniera automatica. Questo, combinando moduli già esistenti, permette di ricevere direttamente i dati dal sensore e di impacchettarli in file contenenti tutte le informazioni necessarie per le successive fasi di allenamento. La pipeline di registrazione è esposta in Figura 5.2.

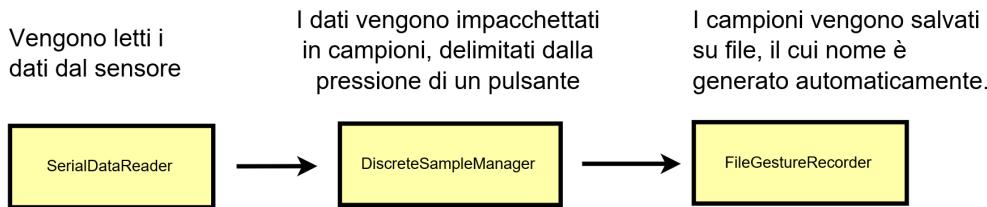


Figura 5.2: Pipeline di registrazione dei gesti.

Si è ritenuto che almeno 50 campioni fossero necessari per ogni carattere e, alla fine del processo di registrazione, il numero totale è stato di 1934 sample.

5.2 Training del classificatore e analisi della Confusion Matrix

Una volta creato il dataset, il passo successivo è stata la creazione di un modello in grado di determinare il gesto associato ad un nuovo campione. È stato quindi necessario realizzare un classificatore adatto allo scopo e, per trovare il classificatore ottimale, si è deciso di mettere a confronto i vari moduli implementati in precedenza, confrontandone i punteggi e i risultati.

5.2.1 Training del SVMClassifier

Il primo classificatore preso in esame è stato il `SVMClassifier`. Per trovare i coefficienti ottimali è stata utilizzata la tecnica combinatoria introdotta nel capitolo precedente. È stata predisposta una griglia di possibili valori e, successivamente, sono state provate tutte le combinazioni al fine di trovare la migliore.

Griglia dei possibili valori:

I parametri considerati sono stati la funzione di decisione, o *kernel*, e il parametro di penalità dell'errore. I possibili valori sono esposti nella Tabella 5.1.

Funzione di decisione (kernel):			
Lineare	Polinomiale	Sigmoide	RBF
Parametro di penalità dell'errore (C):			
10^0	10^{-1}	10^{-2}	10^{-3}

Tabella 5.1: Possibili valori dei parametri del `SVMClassifier`.

Questi valori hanno portato alla creazione di 16 possibili combinazioni, tutte testate singolarmente per valutarne la migliore. I risultati del training del classificatore sul dataset in esame sono riportati in Tabella 5.2.

Kernel	C	Risultato
lineare	0,001	0,994
rbf	0,001	0,078
polinomiale	0,001	0,987
sigmoide	0,001	0,078
lineare	0,01	0,993
rbf	0,01	0,078
polinomiale	0,01	0,987
sigmoide	0,01	0,078
Kernel	C	Risultato
lineare	0,1	0,993
rbf	0,1	0,077
polinomiale	0,1	0,992
sigmoide	0,1	0,124
lineare	1	0,993
rbf	1	0,078
polinomiale	1	0,981
sigmoide	1	0,141

Tabella 5.2: Risultati del training del `SVMClassifier` in base ai parametri kernel e C.

In verde è evidenziata la combinazione dei parametri che ha portato al risultato migliore, ovvero quella la cui funzione di decisione è lineare e ha il parametro di penalità dell'errore pari a 0,001.

Trovata la combinazione ottimale, si è passati ad analizzare la *Confusion Matrix*, un grafico che permette di valutare quali gesti vengano confusi con altri e in quale misura. In un caso ideale, ci aspetteremmo una diagonale

ben definita nel grafico, e pochi elementi al di fuori. Ciò significa che, se tutto fosse ideale, nessuna categoria verrebbe confusa con un'altra, perciò gli unici elementi presenti sarebbero quelli confusi con loro stessi, ovvero sulla diagonale. La Confusion Matrix del classificatore appena trovato è riportata in Figura 5.3.

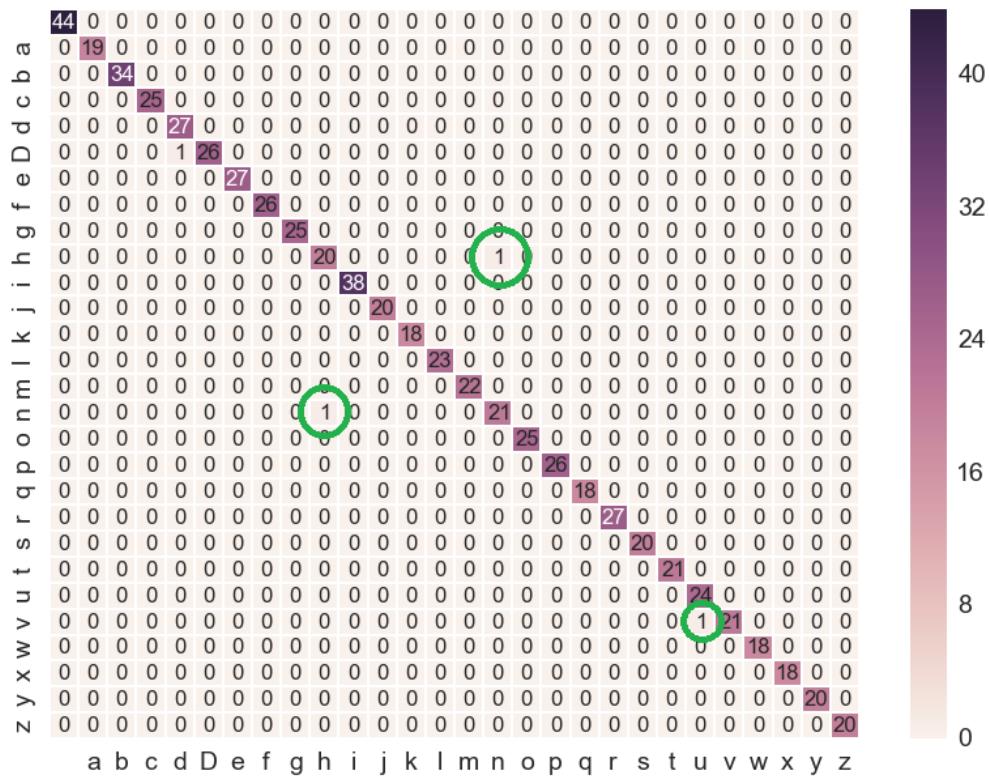


Figura 5.3: Confusion Matrix del SVMClassifier.

Si può subito vedere come complessivamente il risultato sia ottimo: la distribuzione è quasi totalmente diagonale e gli errori del classificatore sono minimi. In particolare, sono evidenziati con un cerchio verde gli errori più interessanti, dove possiamo notare che:

- La “n” può essere confusa con la “h” e viceversa.
- La “u” può essere confusa con la “v”.

Ciò non dovrebbe minimamente sorprendere vista la grande somiglianza

tra le forme dei gesti confusi. Anche per la mente umana risulta talvolta difficile distinguere caratteri scritti a mano se eseguiti male.

5.2.2 Training del MLPClassifier

Il secondo classificatore preso in esame è stato il **MLPClassifier**, basato sul concetto di Multi-layer Perceptron, ovvero una rete neurale. Come nel caso precedente, nella ricerca dei coefficienti ottimali, è stata utilizzata una tecnica combinatoria per trovare la combinazione migliore. A differenza del **SVMClassifier** però, la necessità di trovare il numero giusto di livelli nascosti nella rete neurale ha fatto sì che il numero di valori da analizzare fosse molto più elevato.

Griglia dei possibili valori:

I parametri considerati sono stati il parametro di regolarizzazione (α) e la configurazione dei livelli nascosti della rete neurale. I possibili valori sono esposti in Tabella 5.3.

Parametro di regolarizzazione (α):						
10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}
<hr/>						
Configurazione dei livelli nascosti (hidden layers): <hr/>						
Numero di livelli nascosti:						
1	2	3				
Numero di nodi per livello: *						
5	25	45				

* Si è scelto di utilizzare solo 3 possibili valori per il numero di nodi in modo da limitare il tempo di calcolo.

Tabella 5.3: Possibili valori dei parametri del MLPClassifier.

Il numero finale di combinazioni è stato 273, e i risultati migliori dell’allenamento sono riportati in Tabella 5.4.

In verde è evidenziata la combinazione che ha portato al risultato migliore, ovvero quella con un solo livello nascosto da 45 nodi e il parametro alpha pari a 0,1. In Figura 5.4 è riportata la confusion matrix del classificatore, dove sono stati evidenziati gli errori in verde.

α	Livelli	Risultato
0.1	(45)	0.976133651551
0.1	(25)	0.975338106603
0.1	(45, 45)	0.97215592681
0.1	(45, 25)	0.971360381862
0.1	(25, 45, 45)	0.971360381862
0.1	(45, 45, 45)	0.971360381862
0.1	(45, 25, 25)	0.970564836913
0.1	(45, 25, 45)	0.970564836913

Tabella 5.4: Risultato del training del MLPClassifier al variare dei parametri.

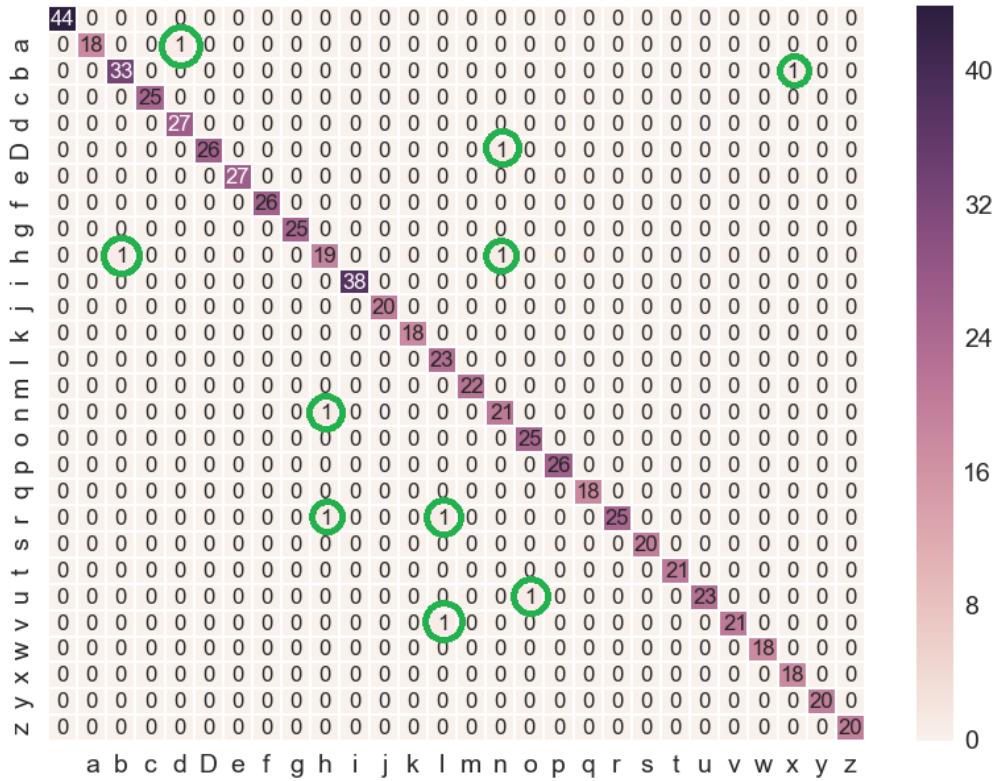


Figura 5.4: Confusion Matrix del training con MLPClassifier.

5.2.3 Scelta del classificatore

Per quanto valido, il risultato ottenuto con il **MLPClassifier** non si è rivelato positivo quanto il precedente. Questo può essere dipeso dal numero di

campioni utilizzati per il training o dal numero di livelli della rete neurale. Tuttavia, visti gli ottimi risultati ottenuti con il **SVMClassifier**, si è scelto di utilizzare quest'ultimo come classificatore per le parti successive e di non approfondire ulteriormente.

5.3 Pipeline di elaborazione del segnale

Una volta realizzato il classificatore, il passo successivo è stata la realizzazione della pipeline di elaborazione del segnale. L'obiettivo era quello di simulare la pressione di un tasto a seguito di uno specifico movimento. Per farlo sono stati combinati i componenti precedentemente introdotti e il risultato è la sequenza di operazioni proposta in seguito:

1. Viene eseguito il gesto di un determinato carattere utilizzando il dispositivo realizzato nel capitolo 3, avendo cura di premere il pulsante all'inizio del movimento e di rilasciarlo al termine. Questo permette di discriminare un gesto dall'altro, aumentando notevolmente la precisione nelle fasi successive.
2. I valori del sensore vengono trasferiti tramite una connessione seriale, indipendentemente dal fatto che avvenga tramite Bluetooth o USB. È stato quindi utilizzato un **SerialDataReader** in grado leggere i dati dalla porta seriale, effettuare il parsing e inviarli allo strato successivo.
3. I dati vengono ricevuti da un **DiscreteSampleManager** che li impacchetta in campioni e li invia allo strato successivo.
4. I campioni vengono ricevuti da un **ClassifierPredictor** che ha il compito di determinare il gesto associato a ognuno di essi. Per farlo, invia ogni sample al **SVMClassifier** precedentemente allenato che restituisce la **GESTURE_ID** associata. Ricevuta una **GESTURE_ID**, il **ClassifierPredictor** la invia allo strato successivo.
5. Il **CallbackManager**, ricevuta una **GESTURE_ID**, si occupa di eseguire il gestore corrispondente, ovvero una funzione precedentemente associata al gesto. In questo caso, viene invocato il metodo **press_key** per ognuno dei caratteri, passando come parametro il gesto ricevuto.
6. Il metodo **press_key** si occupa di simulare la pressione del carattere ricevuto. Il risultato è un comportamento analogo a quello che si avrebbe premendo il tasto corrispondente di una tastiera.

Questa sequenza di operazioni è sinteticamente esposta in Figura 5.5.

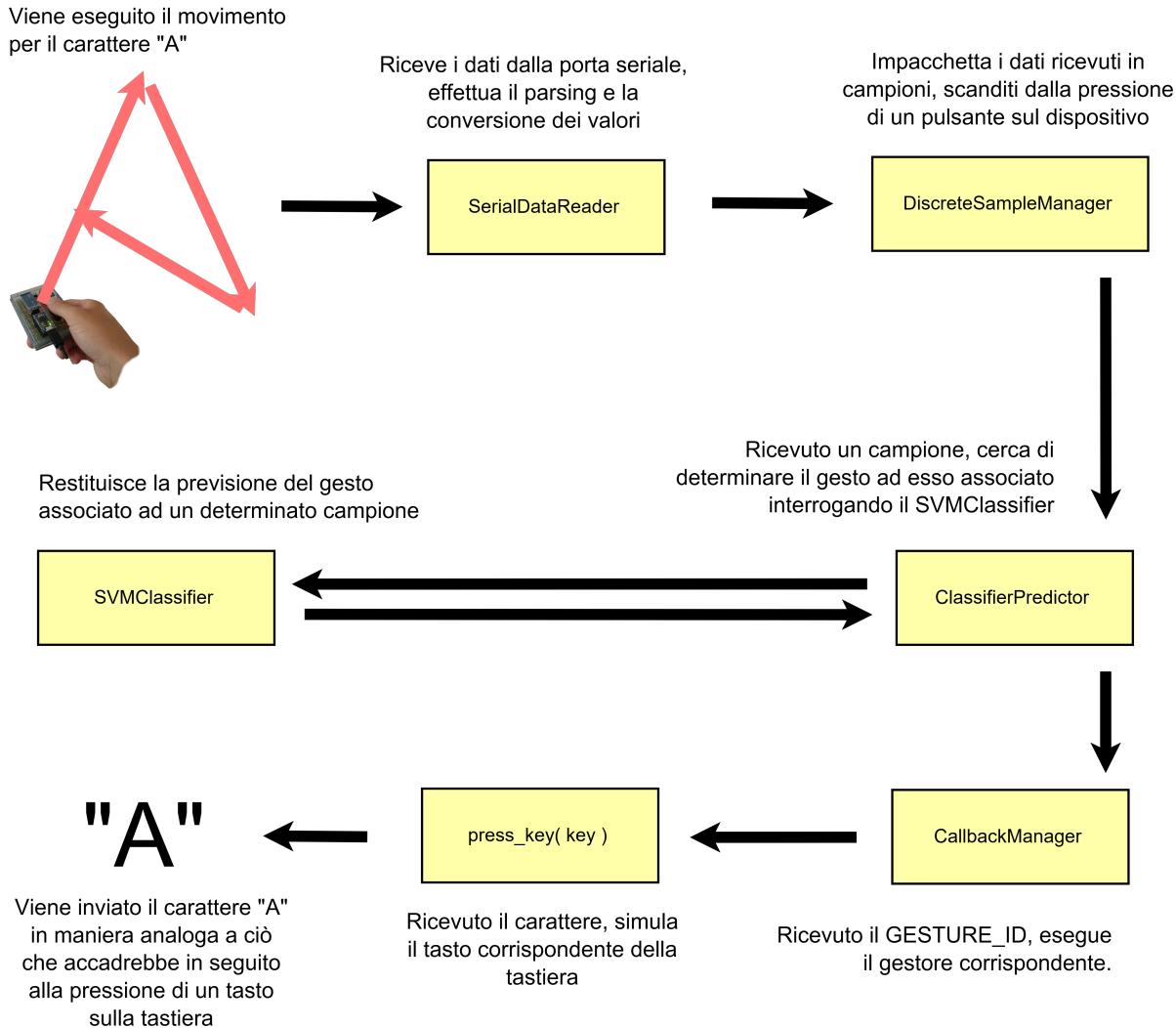


Figura 5.5: Pipeline di elaborazione della tastiera gestuale.

5.4 Risultati ottenuti e possibili miglioramenti

La pipeline di elaborazione appena proposta, combinata con il dispositivo realizzato nel capitolo 3, ha permesso di realizzare un apparecchio in grado di riconoscere gesti e convertirli in caratteri. A seguito di un'attenta analisi, l'accuratezza del dispositivo è risultata valida, sopra al 96%.

Una serie di sviluppi sono possibili, sia per migliorare l’usabilità che la precisione. In seguito ne vengono proposti alcuni:

- Per migliorare l’accuratezza, si è studiata la possibilità di aggiungere un dizionario in grado di correggere eventuali errori di conversione. Ad esempio, tenendo presente che la “n” è stata spesso confusa con la “h”, la parola “notel” verrebbe corretta in “hotel” e questo migliora notevolmente l’accuratezza. Tuttavia lo svantaggio è la possibile mancanza della parola nel dizionario, che comporta una previsione sbagliata.
- Attualmente il sistema necessita di essere connesso ad un computer al fine di eseguire l’elaborazione. Per migliorare l’usabilità si potrebbe studiare una soluzione embedded, in cui i calcoli vengano eseguiti sul dispositivo stesso.

Capitolo 6

Caso di studio: Sensore di tocchi

Il caso di studio successivo ha riguardato la realizzazione di un sensore di tocchi, ovvero un dispositivo in grado di rilevare specifici colpi su una superficie e reagire di conseguenza. In particolare, nel caso proposto, l'obiettivo era quello di creare una “scrivania intelligente” capace di riconoscere determinati tocchi ed eseguire azioni in base ad essi.

Inizialmente si è provato a utilizzare il dispositivo costruito nel capitolo 3, ma si è rapidamente capito come non fosse sufficientemente preciso e quindi non adatto alla situazione. Si è perciò deciso di costruire un altro dispositivo e il prossimo paragrafo ne espone i punti salienti.

6.1 Realizzazione del dispositivo

Per realizzare l'apparecchio in grado di rilevare le vibrazioni di una superficie è stato utilizzato un sensore piezoelettrico, ovvero un componente in grado di trasformare l'energia meccanica di movimento in corrente, i cui valori vengono letti da un microcontrollore, anche in questo caso un Arduino. Il voltaggio generato è tuttavia troppo basso per poter essere direttamente letto, perciò è stato utilizzato un modulo LM386 per amplificare il segnale elettrico in uscita dal sensore. A questo è stato inoltre aggiunto un potenziometro per poter regolare i livelli di guadagno in uscita, ciò permette di impostare la sensibilità del sensore stesso.

In caso di forti vibrazioni, il piezoelettrico può generare elevate tensioni, cosa molto dannosa per il microcontrollore. Per questo motivo sono stati aggiunti in parallelo al sensore un diodo zener ed una resistenza da $1 M\Omega$ che fungono da protezione. Sono inoltre stati inseriti due pulsanti per interagire con l'utente. Il circuito finale è mostrato in Figura 6.1.

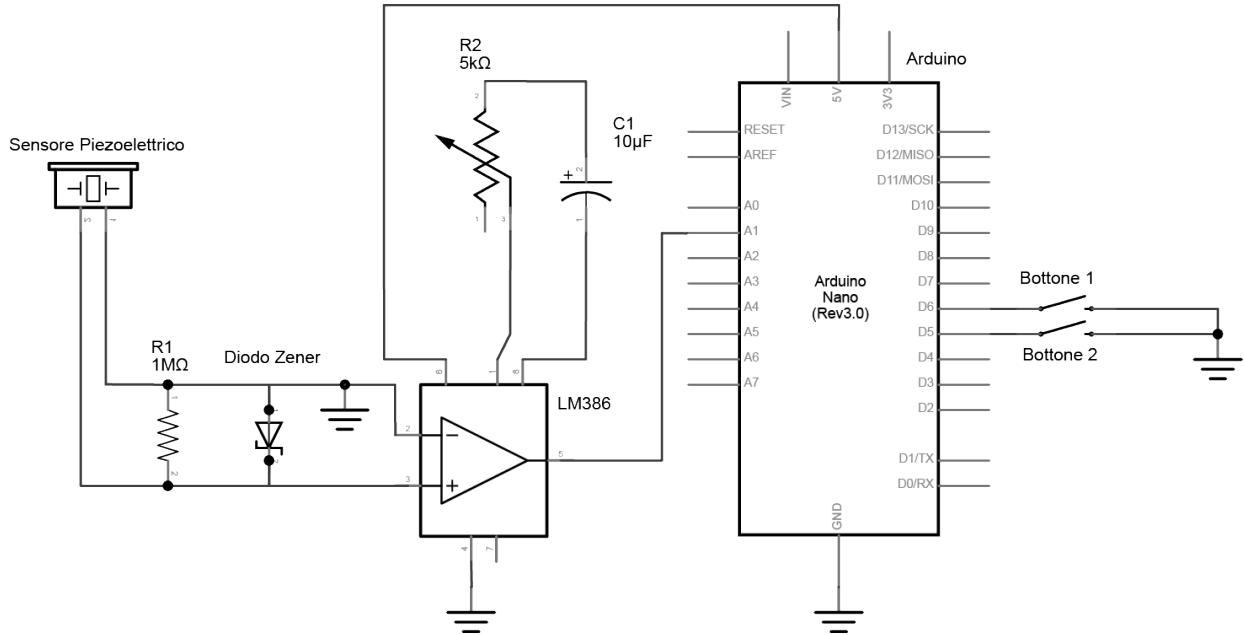


Figura 6.1: Schema circuitale del dispositivo di rilevazione delle vibrazioni..

Il passo successivo è stato realizzare il software per l'Arduino che permettesse di leggere il valore del sensore e di comunicarlo al computer. Per fare ciò è stato sufficiente utilizzare la funzione di lettura di un valore analogico del microcontrollore per poi impacchettare il risultato nel medesimo formato di trasmissione descritto nel capitolo 3. Ciò ha permesso di ottenere una piena compatibilità con i moduli software realizzati per il precedente dispositivo. Bisogna inoltre far notare come nel prototipo finale, mostrato in Figura 6.2 siano presenti una serie di monete disposte al di sopra del sensore che permettono di aumentarne notevolmente la sensibilità. Questo perché grazie al loro peso, incrementano l'entità delle vibrazioni da esso percepite.

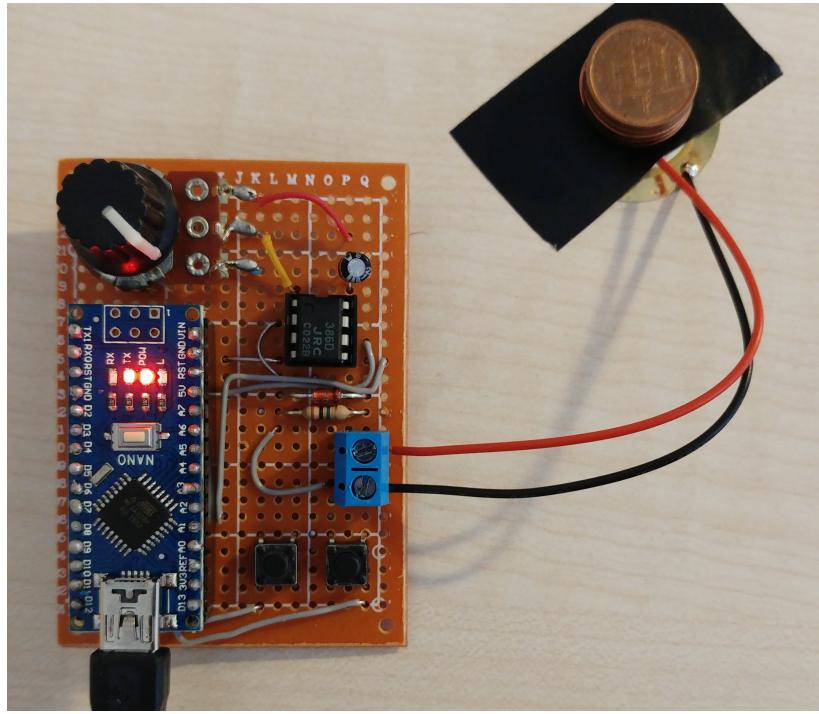


Figura 6.2: Prototipo finale del dispositivo di rilevazione delle vibrazioni.

6.2 Analisi del segnale

Realizzato il dispositivo, il passo successivo è stata la creazione di una pipeline in grado di elaborare il segnale ricevuto e di determinare il gesto corrispondente. A differenza del precedente caso di studio però si è presentato un ulteriore ostacolo, la separazione dei vari tocchi. In questo dispositivo infatti non era disponibile un pulsante che permetesse di segnalare l'inizio e la fine di un gesto, cosa che ha reso necessaria la creazione di un sistema in grado di distinguere tra loro i diversi tocchi. In Figura 6.3 è riportato un esempio di questa situazione, ovvero la separazione dei tre gesti evidenziati in rosso a partire dalla serie dei valori in ingresso.

Il punto di partenza dell'analisi è quindi un flusso continuo di dati inviati dal sensore. Per leggerli è stato utilizzato come nel caso precedente un `SerialDataReader` che però, a differenza della tastiera gestuale, cede i valori ad uno `StreamSampleManager`. Questo permette di ottenere una serie di campioni in rapida successione che tuttavia hanno bisogno di un'ulteriore elaborazione per essere utilizzati. Qui entrano in gioco i `Middleware`, ovvero

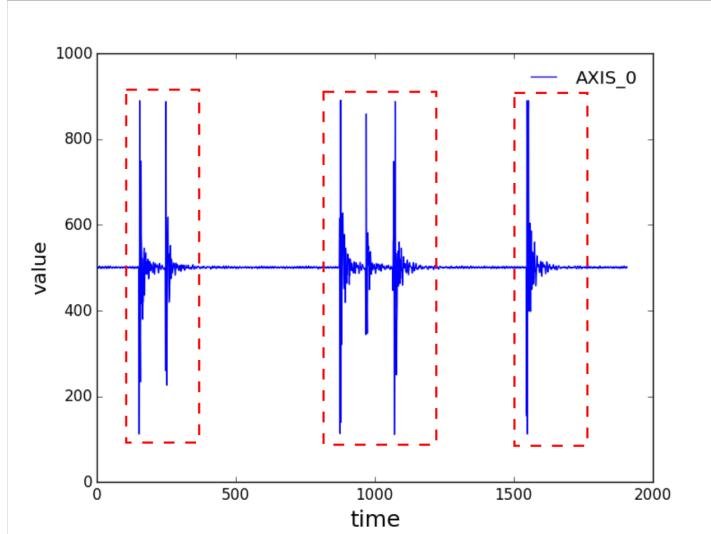


Figura 6.3: Flusso continuo di valori in uscita dal sensore, in cui sono evidenziati i gesti da separare.

dei componenti in grado di intercettare e modificare i Sample prima che arrivino alla fase di riconoscimento.

6.2.1 GradientThresholdMiddleware

Il **GradientThresholdMiddleware** è il componente che si occupa della separazione dei vari tocchi a partire da un flusso continuo di valori. Il concetto su cui si basa è l'applicazione di una soglia, o *threshold*, al *gradiente* del segnale per determinare se il campione corrente contiene o meno un gesto.

Il motivo per cui si è deciso di calcolare il gradiente prima di applicare la soglia è stata la necessità di rendere il sistema più stabile a eventuali cambi del sistema di riferimento. In particolare utilizzando il gradiente, il segnale viene portato sull'origine e ciò semplifica notevolmente il lavoro di filtraggio. Questa considerazione si rivelerà particolarmente utile nel caso di studio affrontato nel capitolo 7. Per svolgere il suo compito, il componente ha bisogno di eseguire una serie di elaborazioni, esposte con un esempio in seguito:

Il primo passo è quello di ottenere il flusso di valori dal sensore. Nel caso in esame è stata presa in considerazione l'esecuzione di due tocchi in rapida

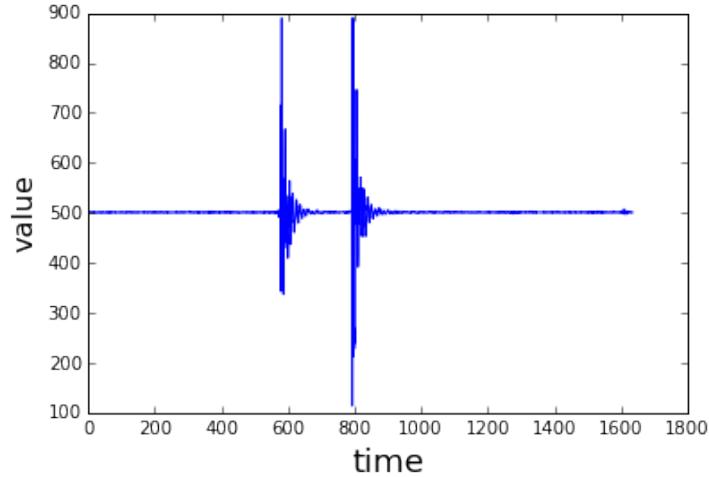


Figura 6.4: Flusso di dati dal sensore.

successione e la Figura 6.4 riporta una rappresentazione grafica di questo movimento. Si può notare come nei periodi di quiete, ovvero all'inizio e alla fine, il segnale oscilli intorno al valore 500. L'obiettivo è quindi quello di isolare i due tocchi principali e di raggrupparli in un unico gesto.

Il secondo passo è stato quello di calcolare il gradiente. Come si può notare in Figura 6.5, a seguito dell'operazione il segnale, in assenza di movimenti, oscilla intorno all'origine. Questo ha semplificato notevolmente il filtraggio delle fasi successive.

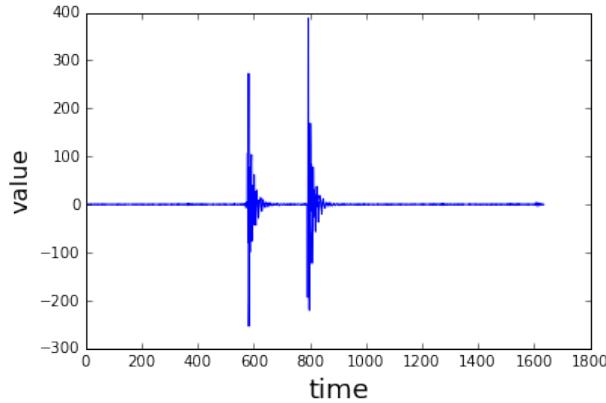


Figura 6.5: Gradiente applicato al segnale.

Il terzo passo è stata l'applicazione di una soglia, ovvero un limite che, se

superato, denota un movimento all'interno del campione corrente. In Figura 6.6 è stata applicata una soglia di valore 100 e sono stati evidenziati i due punti in cui questa è superata.

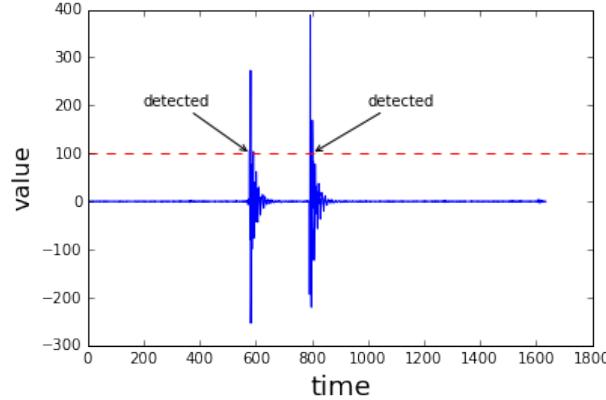


Figura 6.6: Soglia applicata al gradiente.

In Figura 6.7 sono evidenziati due campioni che presentano un picco di movimento, determinato dal superamento della soglia. In questo stato i due campioni non possono essere utilizzati perché, pur facendo parte di un unico gesto, sono separati.

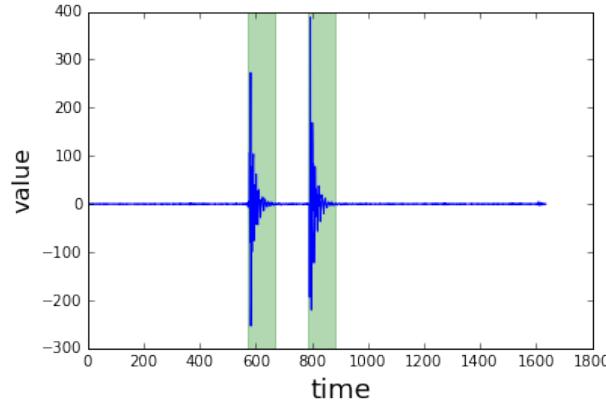


Figura 6.7: Campioni che presentano un picco di movimento.

Per ottenere il gesto originario sono stati uniti i campioni della fase precedente. In particolare è stato considerato un intervallo in cui i campioni successivi rimanessero validi, pur non superando la soglia. Questo ha permesso di ottenere un unico grande campione, ma con l'inconveniente di inserire

una “coda” al segnale in cui non si verifica alcun movimento.

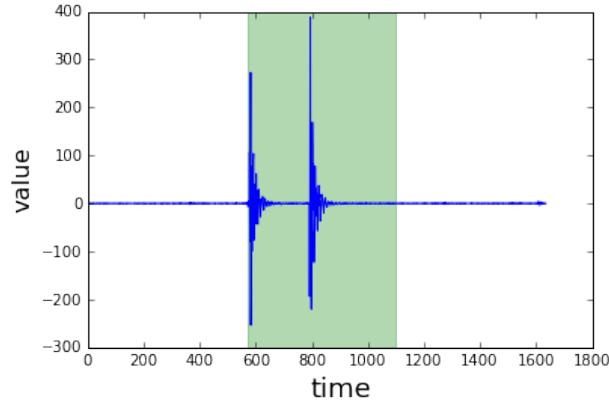


Figura 6.8: Raggruppamento dei campioni.

Il passaggio finale è stato quello di rifinire il campione ottenuto dal passaggio precedente, rimuovendo gli estremi in eccesso. A questo punto non rimaneva che tagliare il segnale originale nell’intervallo ottenuto a seguito dell’elaborazione, ottenendo quindi il campione rappresentato in Figura 6.9.

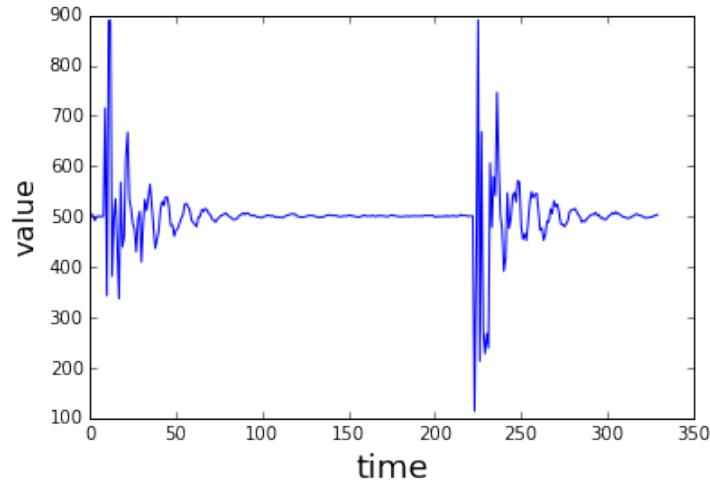


Figura 6.9: Campione finale del gesto.

6.2.2 AbsoluteScaleMiddleware

Una volta isolato il campione, il passo successivo è stato preparare il segnale al riconoscimento da parte di un classificatore. Per farlo è necessario estrarre il profilo della forma d'onda e di questo si occupa l'**AbsoluteScaleMiddleware**. I passaggi dell'elaborazione sono esposti in seguito.

Partendo dal campione estratto in precedenza, viene sottratto il valore 500 ad ogni istante, in modo da portare l'oscillazione in assenza di movimento sull'asse dell'origine. A quel punto viene calcolato il valore assoluto in maniera tale da rendere la forma d'onda positiva. In Figura 6.10 è mostrato il risultato di questa elaborazione partendo dal campione estratto nel paragrafo precedente, raffigurato in Figura 6.9.

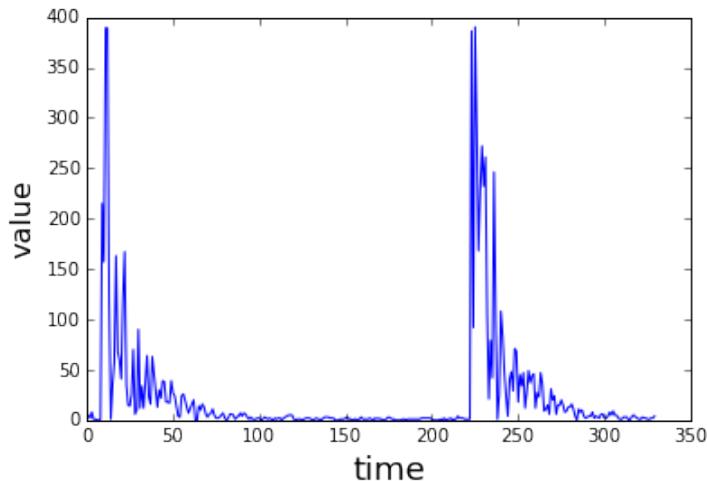


Figura 6.10: Campione precedente a cui è stato sottratto 500 e calcolato il valore assoluto.

Successivamente, per estrarre il profilo della forma d'onda, si è proceduto a calcolarne la *media mobile*, uno strumento molto utilizzato in statistica per smussare eventuali fluttuazioni in una serie di valori [22]. Questa consiste nel sostituire ad ogni valore la media dei valori ad esso vicini, eliminando quindi i picchi dovuti alle vibrazioni improvvise. È interessante notare come la media mobile sia un caso particolare di filtro passa-basso e proprio grazie a questa proprietà ci permette di filtrare le fluttuazioni ad alta frequenza composte principalmente da rumore.

Il passaggio finale è stato scalare il campione ad una lunghezza fissa, in maniera da rendere comparabili tra loro campioni diversi. Il risultato di

questa elaborazione è mostrato in Figura 6.11.

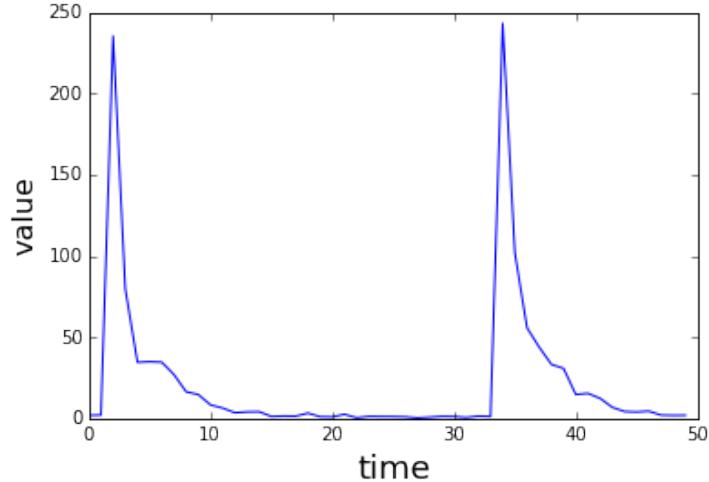


Figura 6.11: Risultato finale dell'elaborazione da parte dell'AbsoluteScaleMiddleware.

6.3 Definizione dei gesti e creazione del Dataset

Dopo aver realizzato i componenti incaricati di preprocessare il segnale, il passo successivo è stato definire i gesti da riconoscere. Ne sono stati scelti 3 e sono esposti in Tabella 6.1.

Gesti da riconoscere		
KNOCK	DOUBLE-KNOCK	HAND
Rappresenta un colpo singolo.	Rappresenta due colpi in rapida successione.	Movimento di scorimento con le dita.

Tabella 6.1: Definizione dei 3 gesti da riconoscere.

Una volta definiti i gesti, si è proceduto a creare un dataset di campioni per poter poi allenare un classificatore. Il procedimento utilizzato è sim-

ile a quello del capitolo 5 ma con qualche modifica per adattarsi al nuovo caso. In particolare, trattandosi di un flusso continuo di valori, è stato necessario inserire un **GradientThresholdMiddleware** alla pipeline per separare i campioni tra loro e il risultato è mostrato in Figura 6.12.

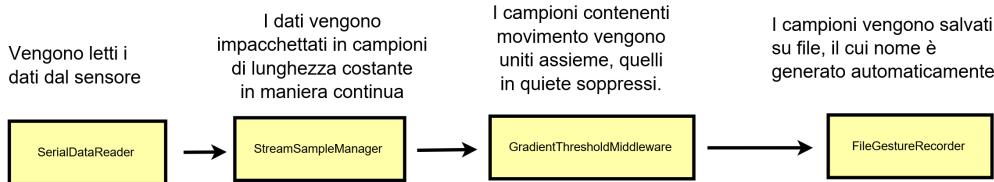


Figura 6.12: Pipeline di registrazione dei gesti.

Si può notare come nella pipeline, pur essendo necessario al riconoscimento, non sia presente l'**AbsoluteScaleMiddleware**. Questo è dovuto al fatto di volere registrare i campioni con un basso numero di alterazioni, in modo da poterli modificare in caso di necessità.

6.4 Training del classificatore e analisi della Confusion Matrix

Una volta creato il dataset, si è proceduto a creare il modello in grado di riconoscere i gesti. Per farlo sono stati seguiti gli stessi passaggi spiegati nel capitolo 5, confrontando i vari tipi di classificatori. Per brevità tuttavia viene riportato solamente il risultato del **SVMClassifier** che anche in questo caso ha superato il **MLPClassifier** in termini di accuratezza.

Per trovare i coefficienti ottimali è stata utilizzata la tecnica combinatoria esposta in precedenza, utilizzando i valori esposti in Tabella 6.2.

Funzione di decisione (kernel):			
Lineare	Polinomiale	Sigmoide	RBF
Parametro di penalità dell'errore (C):			
10^0	10^{-1}	10^{-2}	10^{-3}

Tabella 6.2: Possibili valori dei parametri del SVMClassifier.

Terminata l'elaborazione, il risultato ha evidenziato come anche in questo caso la miglior combinazione fosse quella caratterizzata dal kernel lineare. I risultati sono esposti in Tabella 6.3.

Kernel	C	Risultato
lineare	1	0,97531
lineare	0.01	0,96296
lineare	0.1	0,96296
polinomiale	1	0,90741
polinomiale	0.01	0,90741
polinomiale	0.1	0,90741

Tabella 6.3: Risultati dell’allenamento del SVMClassifier.

Viene inoltre riportata in Figura 6.13 la confusion matrix relativa al miglior classificatore trovato. L’unico errore è relativo ad uno scambio del DOUBLE-KNOCK con HAND che, come si può vedere nelle forme d’onda riportate in precedenza, possono essere simili in alcune circostanze.

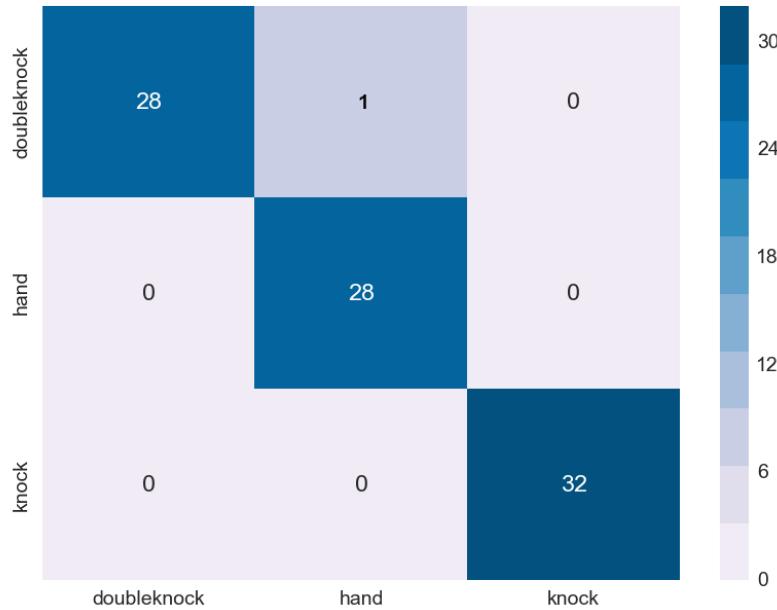


Figura 6.13: Confusion Matrix relativa al miglior classificatore trovato.

6.5 Pipeline di elaborazione del segnale

Trovato il classificatore, si è proceduto ad assemblare i vari componenti per creare una pipeline di elaborazione in grado di discriminare un gesto a

partire dal flusso continuo di valori provenienti dal sensore. In Figura 6.14 è descritto lo schema del processo.

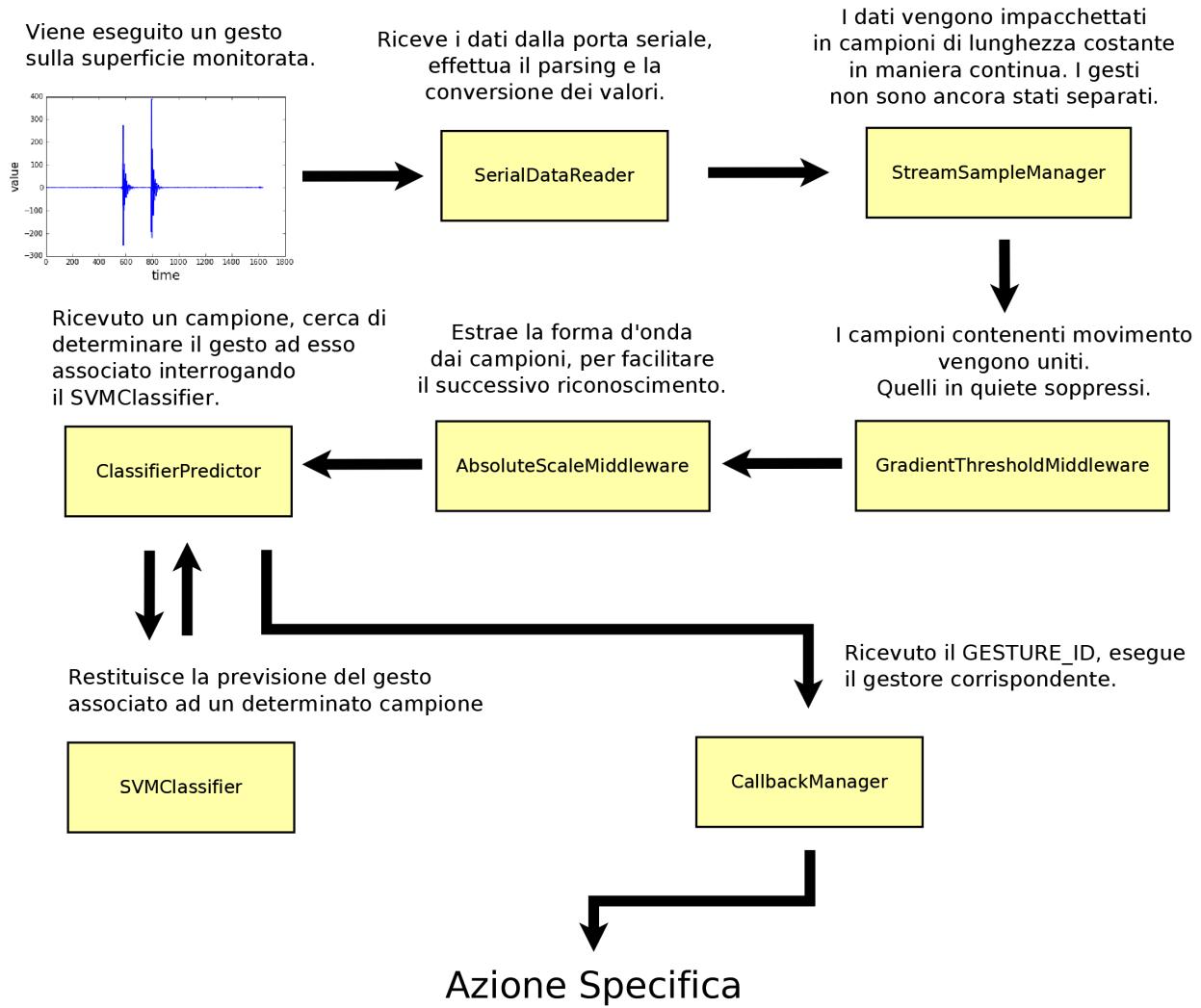


Figura 6.14: Pipeline di elaborazione del segnale.

La pipeline può quindi essere riassunta nei seguenti passaggi:

1. Viene eseguito un gesto sulla superficie monitorata e la vibrazione viene catturata dal dispositivo.
2. Il **SerialDataReader** legge i valori dal sensore ed effettua il parsing, per la successiva elaborazione nella catena.

3. Lo **StreamSampleManager** accorda i valori ricevuti dallo stadio precedente in campioni di lunghezza costante, senza effettuare alcuna elaborazione in merito alla separazione dei movimenti. I campioni in uscita non sono ancora utilizzabili per il riconoscimento.
4. Il **GradientThresholdMiddleware** si occupa di individuare ed isolare i gesti a partire dal flusso continuo di campioni, unendo quelli appartenenti allo stesso movimento e scartando quelli in quiete.
5. L'**AbstractScaleMiddleware** estrae la forma d'onda del campione in ingresso.
6. Il **ClassifierPredictor**, in maniera analoga al caso di studio precedente, interroga il classificatore e fornisce in uscita il gesto associato al campione in ingresso.
7. Il **CallbackManager**, ricevuto l'id del gesto, esegue il gestore corrispondente e attua l'azione specifica associata.

6.6 Risultati ottenuti e possibili miglioramenti

Il dispositivo realizzato, in combinazione con i componenti software esposti in questo capitolo, si è rivelato capace di discriminare un piccolo insieme di gesti e di reagire conseguentemente con un'accuratezza notevole. In particolare, eseguendo dei movimenti sulla superficie monitorata è possibile interagire con essa, anche in maniera complessa. Si potrebbe dire che è capace di rendere “smart” oggetti che normalmente non lo sono, ad un prezzo molto contenuto.

Le azioni eseguibili in reazione ai gesti sono personalizzabili in base ai requisiti di progetto, e possono spaziare in un numero molto elevato di ambiti. Ad esempio, collegando il sensore ad una parete sarebbe possibile impartire comandi ad un sistema di domotica rendendo la casa stessa il dispositivo di input.

Tuttavia, per quanto in generale il risultato sia positivo, il dispositivo non è privo di difetti. Il principale problema è il basso numero di gesti distinguibili. Aumentandone il numero infatti l'accuratezza diminuisce ed è perciò importante scegliere dei gesti ragionevolmente diversi tra loro, compito non semplice utilizzando le vibrazioni di un solo sensore. Per mitigare il problema quindi una possibile soluzione sarebbe aumentare il numero di rilevatori piezo-elettrici. Un numero maggiore di sensori comporterebbe infatti

un aumento dei dati disponibili per il riconoscimento e di conseguenza un aumento del numero di gesti distinguibili. Inoltre, distanziandoli in maniera appropriata, sarebbe possibile rilevare vibrazioni in zone diverse e quindi studiare anche l'interazione tra esse.

Capitolo 7

Caso di studio: Anello d'ausilio a persone con difficoltà di movimento

Questo caso di studio tratta la creazione di un anello dotato di sensori capace di aiutare le persone con particolari difficoltà motorie ad interagire con il mondo esterno. Consiste di un accelerometro che, attaccato alla punta di un dito tramite un anello, permette di registrare e di interpretare i movimenti, anche minimi, effettuati dall'utente e reagire di conseguenza.

L'interazione avviene tramite una tabella composta da una serie di caratteri, attraverso la quale l'utente si muove per comporre delle frasi. Questa navigazione è resa possibile dal sensore tramite specifici gesti configurabili in base alle esigenze del caso. La realizzazione di questa interfaccia sarà commentata nella sezione 7.6.

7.1 Creazione dell'anello

Anche in questo caso si è scelto di utilizzare un MPU-6050 come accelerometro, in modo da poter riutilizzare il dispositivo descritto nel capitolo 3 effettuando un numero minimo di modifiche. Date le nuove esigenze, il sensore è stato attaccato ad un anello composto da due bande di velcro che permettono di adattarsi alle dita di utenti diversi. È stata inoltre aggiunta una prolunga al sensore che permette di posizionare l'Arduino lontano dal

dito e quindi un movimento più agevole. Un prototipo di quanto esposto è riportato in Figura 7.1.



Figura 7.1: Prototipo dell'anello con sensori e prolunga.

È importante precisare come la prolunga sia stata creata in modo da avere una configurazione dei pin di collegamento analoga a quella del dispositivo realizzato nel Capitolo 3. Questo ne ha permesso l'interfacciamento senza ulteriori modifiche. Il collegamento alla scheda e il risultato finale sono esposti in Figura 7.2.

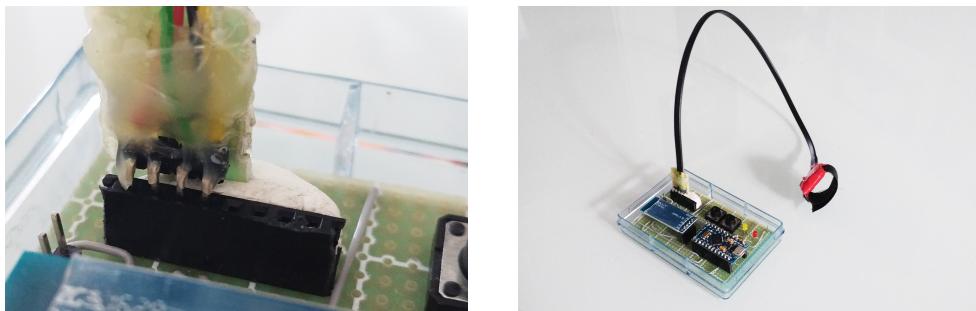


Figura 7.2: Collegamento dell'anello alla scheda e configurazione finale.

7.2 Analisi del segnale

Di tutti i casi di studio analizzati finora, questo rappresenta sicuramente il più complesso per quanto riguarda l'analisi del segnale. Per realizzare una pipeline di riconoscimento efficace sono state combinate una serie di tecniche elaborate nei precedenti casi di studio.

Come nel progetto della tastiera gestuale, esposto nel Capitolo 5, è stato necessario analizzare i valori in uscita da un accelerometro. A differenza di esso però, i gesti non sono stati scanditi da pressioni di un tasto, ma è stato necessario estrarlarli da un flusso continuo di valori, come nel Capitolo 6.

Anche in questo caso, i valori del sensore sono forniti da un `SerialDataReader` tramite la porta seriale e successivamente inviati ad un `StreamSampleManager` per raggrupparli. A questo punto, un `GradientThresholdMiddleware` è stato utilizzato per separare i campioni in base al movimento rilevato. In aggiunta, sono stati introdotti altri componenti in grado di aumentare la precisione del riconoscimento.

7.2.1 FFTMiddleware

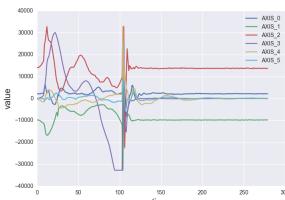
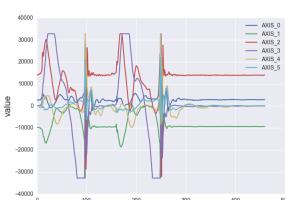
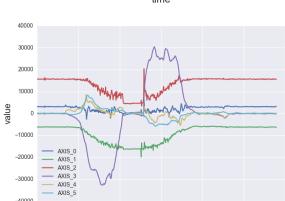
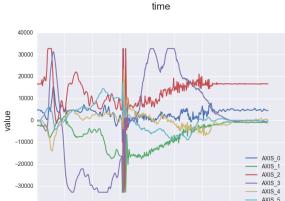
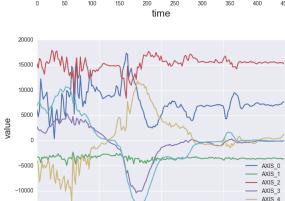
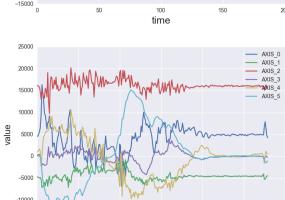
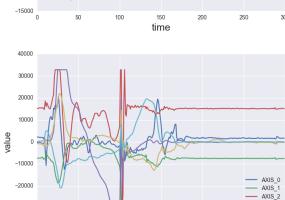
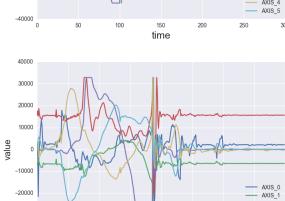
Data la natura dei segnali trattati, si è valutata la possibilità di utilizzare una *trasformata di Fourier* per fornire un ulteriore punto di vista all'algoritmo di riconoscimento, migliorandone le performance. Questa permette di analizzare un segnale nel dominio delle frequenze a partire dalla sua rappresentazione temporale [23].

Ciò si rivela particolarmente efficace nel caso di segnali in cui è presente una ripetizione, come ad esempio un doppio tocco, in quanto si manifesterà un picco nella rappresentazione in frequenza. È importante notare che, nel caso di un segnale non ripetuto, il picco non sarà presente.

Questo componente permette quindi di rendere l'algoritmo capace di riconoscere anche altri aspetti di un movimento, come le ripetizioni. La miglior performance è offerta quando i dati derivanti dal `FFTMiddleware` non sono sostituiti al segnale nel dominio nel tempo, ma aggiunti. Singolarmente, infatti, i due domini hanno una visione limitata di un gesto, mentre combinati riescono ad ottenere ottimi risultati. Nella fase di training del modello, descritta nella sezione 7.4, si è potuto stimare come l'aggiunta di questo componente abbia portato ad un aumento dell'accuratezza pari al 6%.

7.3 Definizione dei gesti e creazione del Dataset

Definiti quindi i componenti necessari a pre-processare i campioni, il passo successivo è stato definire l'insieme dei gesti da riconoscere. Partendo dal presupposto che i movimenti fossero effettuati da un singolo dito, si è scelto di definire 8 gesti. Questi sono stati appositamente concepiti con il requisito di essere semplici da effettuare e sufficientemente differenti nel movimento in modo da semplificare il più possibile il riconoscimento. In Tabella 7.1 sono riportati i gesti con una breve descrizione del movimento ed un esempio di andamento nel tempo.

Andamento	ID	Descrizione
	TAP	Rappresenta un singolo tocco.
	DOUBLETAP	Rappresenta un doppio tocco.
	PULL	Rappresenta un trascinamento del dito verso l'interno.
	PUSH	Rappresenta una spinta del dito verso l'esterno.
	LEFT	Rappresenta un movimento verso sinistra.
	RIGHT	Rappresenta un movimento verso destra.
	TAPCLOCKWISE	Rappresenta una rotazione del dito in senso orario.
	TAPANTICLOCKWISE	Rappresenta una rotazione del dito in senso antiorario.

Una volta definiti i gesti, si è passati alla creazione del dataset per poter successivamente allenare l’algoritmo. Per aumentare l’accuratezza si è deciso di cambiare il metodo di registrazione dei campioni rispetto ai precedenti casi di studio. Al soggetto incaricato di eseguire i movimenti sono stati proposti i vari gesti in ordine casuale. Prima infatti la registrazione avveniva in gruppi, ovvero, deciso un movimento da registrare, ne venivano effettuati un gran numero in sequenza. Questo nuovo approccio ha permesso di differenziare ulteriormente il dataset rendendolo capace di distinguere meglio i cambi di movimento durante i vari gesti.

7.4 Training del classificatore e analisi della Confusion Matrix

Creato il dataset, il passo successivo è stato creare il modello di riconoscimento dei gesti. Per farlo sono stati seguiti gli stessi passaggi del capitolo 5, confrontando i vari tipi di classificatori. Per brevità viene riportato solamente il risultato del **SVMClassifier** che, anche in questo caso, ha superato il **MLPClassifier** in termini di accuratezza.

Per trovare i coefficienti ottimali è stata utilizzata la tecnica combinatoria esposta in precedenza, utilizzando i valori esposti in Tabella 7.2.

Funzione di decisione (kernel):			
Lineare	Polinomiale	Sigmoide	RBF
Parametro di penalità dell’errore (C):			
10^0	10^{-1}	10^{-2}	10^{-3}

Tabella 7.2: Possibili valori dei parametri del SVMClassifier.

Analizzando i risultati dell’elaborazione, riportati in Tabella 7.3, possiamo notare come il kernel lineare si sia rivelato anche in questo caso il più accurato, con un risultato vicino al 95%. L’aggiunta del **FFTMiddleware**, come già detto in precedenza, ha permesso un incremento del 6% in termini di precisione.

Guardando la confusion matrix, riportata in Figura 7.3, possiamo affermare che il risultato è positivo. La separazione tra i vari gesti è netta anche se, osservando attentamente, l’algoritmo tende a confondere il doppio tocco con la rotazione antioraria ed il singolo tocco.

Kernel	C	Risultato
lineare	1	0,9468
lineare	0.01	0,9468
lineare	0.1	0,9468
polinomiale	1	0,9193
polinomiale	0.01	0,9193
polinomiale	0.1	0,9193

Tabella 7.3: Risultati dell’allenamento del SVMClassifier.



Figura 7.3: Confusion Matrix relativa al miglior classificatore trovato.

7.5 Pipeline di elaborazione del segnale

A questo punto non resta che definire la pipeline di elaborazione del segnale, ovvero i passaggi necessari a portare il flusso continuo di valori provenienti dall’accelerometro a comandare un’interfaccia utente. Uno schema riassuntivo del processo è esposto in Figura 7.4.

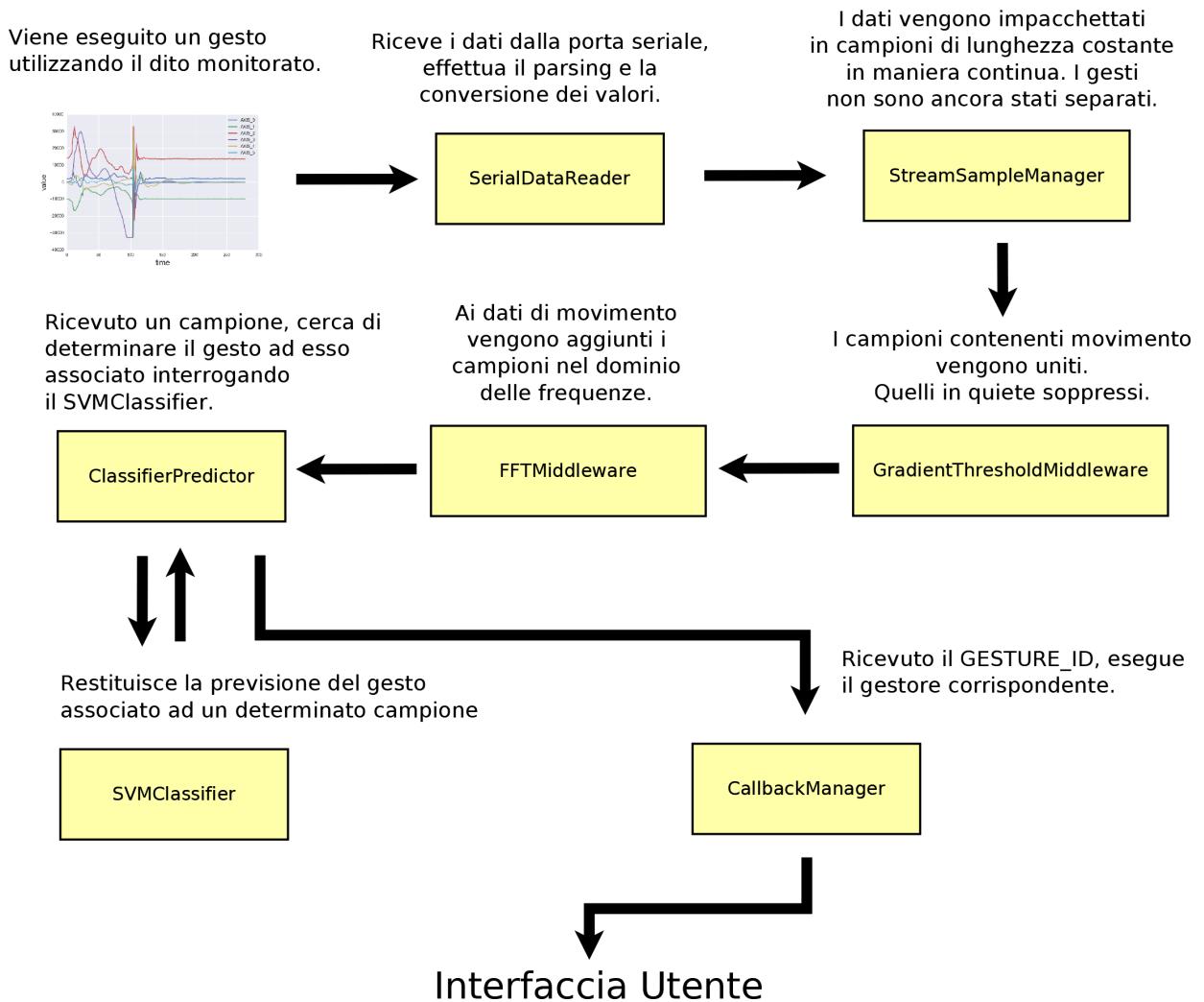


Figura 7.4: Pipeline di elaborazione del segnale.

La pipeline può quindi essere riassunta nei seguenti passaggi:

1. Tramite il dito a cui è attaccato l'anello viene eseguito un gesto.
2. Il **SerialDataReader** legge i valori dal sensore ed effettua il parsing.
3. Lo **StreamSampleManager** accorda i valori ricevuti dallo stadio precedente in campioni di lunghezza costante, senza effettuare alcuna elaborazione in merito alla separazione dei movimenti. I campioni in uscita non sono ancora utilizzabili per il riconoscimento.

4. Il `GradientThresholdMiddleware` si occupa di individuare e isolare i gesti a partire dal flusso continuo di campioni, unendo quelli appartenenti allo stesso movimento e scartando quelli in quiete.
5. Il `FFTMiddleware` calcola la trasformata di Fourier dei campioni ricevuti e la concatena ad essi.
6. Il `ClassifierPredictor`, in maniera analoga al caso di studio precedente, interroga il classificatore e fornisce in uscita il gesto associato al campione in ingresso.
7. Il `CallbackManager` invia il gesto ricevuto all’interfaccia utente, che permetterà l’effettiva interazione.

7.6 Creazione dell’interfaccia utente

L’ultima componente realizzata è stata l’interfaccia grafica che permette all’utente di interagire con il sistema. Si è scelto di realizzare una tabella contenente tutti i caratteri dell’alfabeto nella quale un utente può navigare per comporre il messaggio desiderato. In Figura 7.5 è riportata una delle schermate dell’interfaccia.

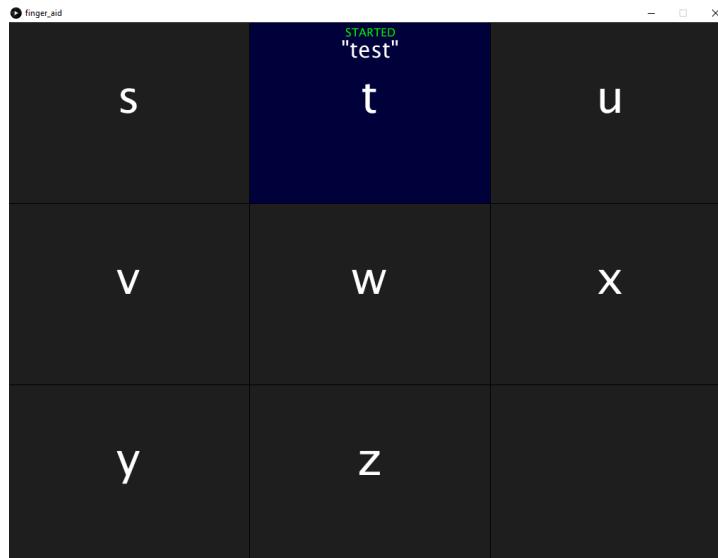


Figura 7.5: Pipeline di elaborazione della tastiera gestuale.

Questa è stata realizzata utilizzando *Processing*, una libreria basata su *Java* che permette in maniera semplice di realizzare delle grafiche interattive.

tive [24]. Il sistema di riconoscimento comunica con l’interfaccia tramite lo standard output, inviando i gesti rilevati.

Per muoversi all’interno della griglia, l’utente utilizza i gesti definiti in precedenza e ognuno di essi è associato ad una azione specifica. La combinazione di questi movimenti permette all’utilizzatore di costruire delle frasi. In Tabella 7.4 sono esposte le azioni corrispondenti ad ogni gesto.

Gestore ID	Azione corrispondente
TAP	Seleziona un carattere e lo aggiunge alla fine della frase corrente.
DOUBLETAP	Scorre la tabella alla pagina successiva, mostrando il prossimo insieme di caratteri.
PULL	Muove il cursore di selezione in basso.
PUSH	Muove il cursore di selezione in alto.
LEFT	Muove il cursore di selezione a sinistra.
RIGHT	Muove il cursore di selezione a destra.
TAPCLOCKWISE	Inserisce uno spazio al termine della frase corrente.
TAPANTICLOCKWISE	Elimina l’ultimo carattere dalla frase corrente.

Tabella 7.4: Azioni corrispondenti ai gesti.

7.7 Risultati ottenuti e possibili miglioramenti

Questo caso di studio ha permesso di creare un dispositivo in grado di aiutare le persone con difficoltà di movimento ad esprimersi. Indossando l’anello dotato del sensore è infatti possibile comandare un’interfaccia che permette di comporre parole e frasi.

Il vantaggio più grande di questo approccio è la capacità di adattarsi alle specifiche esigenze dell’utente. Infatti anche se si è scelto di utilizzare un anello per monitorare i movimenti di un dito, con modifiche minime è possibile monitorare qualsiasi parte del corpo umano.

I fronti su cui si può migliorare il sistema sono molteplici. Ad esempio, aumentare il numero di sensori permetterebbe di migliorare la capacità espressiva del dispositivo senza incorrere in modifiche sostanziali. Questo è possibile grazie alla dinamicità del sistema che richiede unicamente un insieme di campioni per allenare il classificatore.

Capitolo 8

Conclusioni

Il fine di questo progetto è stata la creazione di un sistema di riconoscimento gestuale flessibile e in grado di adattarsi a vari scenari d'uso. Nel corso dei diversi casi di studio, infatti, sono emerse delle problematiche che hanno richiesto la realizzazione di componenti specifici.

Una volta realizzato il dispositivo in grado di registrare i movimenti, si è passati alla progettazione della libreria di riconoscimento gestuale. I requisiti fondamentali sono stati la modularità e la capacità di personalizzazione. Strutturare la libreria in componenti si è rivelato particolarmente utile durante i vari casi di studio, in cui sono state realizzate solo poche parti specifiche, riutilizzandone la maggior parte.

Il primo caso di studio ha riguardato la realizzazione di una tastiera gestuale, ovvero un dispositivo in grado di tradurre gesti in caratteri, simulando una tastiera. La principale problematica è stata la scelta del classificatore più opportuno, il cui risultato è stato però positivo, con un'accuratezza al di sopra del 96%.

Il secondo caso trattato è stato un sensore di tocchi, un dispositivo in grado di determinare il tipo di gesto a partire dalle vibrazioni della superficie colpita. La principale problematica è stata la separazione dei vari gesti. In questo caso infatti i valori si presentavano come un flusso continuo e ciò ha reso necessario realizzare una logica di separazione.

Il terzo caso di studio era inerente alla creazione di un anello d'ausilio a persone con difficoltà motorie, che permettesse tramite piccoli gesti di com-

porre delle frasi e quindi di esprimersi. Anche in questo caso la separazione dei gesti ha richiesto alcune accortezze, a cui si sono aggiunte le difficoltà date dal dover creare un insieme di gesti abbastanza vario da permettere un’interazione anche complessa.

Il sistema si è rivelato flessibile e capace di rispondere anche a necessità differenziate, con un buon grado di riutilizzo. Ci sono tuttavia molti aspetti in cui può essere migliorato. Al momento, il sistema ha bisogno di un computer per eseguire l’elaborazione e ciò può essere limitante in determinati contesti. Uno sviluppo potrebbe quindi riguardare un sistema portatile in grado di eseguire l’elaborazione su un dispositivo embedded. Un altro possibile miglioramento potrebbe essere quello di utilizzare algoritmi di riconoscimento più complessi e accurati nei classificatori.

Bibliografia

- [1] Wikipedia, *Accelerometer*, <https://en.wikipedia.org/wiki/Accelerometer>
- [2] Wikipedia, *Gyroscope*, <https://en.wikipedia.org/wiki/Gyroscope>
- [3] Sito ufficiale Arduino, *What is Arduino*, <https://www.arduino.cc/en/Guide/Introduction>
- [4] Wikipedia, *Switch*, <https://en.wikipedia.org/wiki/Switch>
- [5] Wikipedia, *Pull-up Resistor*, https://en.wikipedia.org/wiki/Pull-up_resistor
- [6] Sito ufficiale Arduino, *Digital Pins*, <https://www.arduino.cc/en/Tutorial/DigitalPins>
- [7] Sito ufficiale InvenSense, *MPU-6050 Six-Axis (Gyro + Accelerometer) MEMS MotionTrackingTM Devices*, <https://www.invensense.com/products/motion-tracking/6-axis/mpu-6050/>
- [8] Sito ufficiale Arduino, *MPU-6050 Accelerometer + Gyro*, <http://playground.arduino.cc/Main/MPU-6050>
- [9] Wikipedia, *Bluetooth*, <https://en.wikipedia.org/wiki/Bluetooth>
- [10] Sito ufficiale Arduino, *SoftwareSerial Library*, <https://www.arduino.cc/en/Reference/SoftwareSerial>
- [11] Wikipedia, *Baud*, <https://it.wikipedia.org/wiki/Baud>
- [12] Wikipedia, *Piezoelectric sensor*, https://en.wikipedia.org/wiki/Piezoelectric_sensor
- [13] Texas Instruments, *Datasheet LM386*, <http://www.ti.com/lit/ds/symlink/lm386.pdf>

- [14] Wikipedia, *Python (programming language)* , [https://en.wikipedia.org/wiki/Python_\(programming_language\)](https://en.wikipedia.org/wiki/Python_(programming_language))
- [15] Sito ufficiale Numpy, *Numpy*, <http://www.numpy.org/>
- [16] Sito ufficiale Scikit-learn, *Scikit-learn*, <http://www.numpy.org/>
- [17] Wikipedia, *Event-driven programming*, https://en.wikipedia.org/wiki/Event-driven_programming
- [18] Wikipedia, *Event Loop*, https://en.wikipedia.org/wiki/Event_loop
- [19] Wikipedia, *Observer Pattern*, https://en.wikipedia.org/wiki/Observer_pattern
- [20] Sito ufficiale Scikit Learn, *Support Vector Machines*, <http://scikit-learn.org/stable/modules/svm.html>
- [21] Wikipedia, *Multilayer perceptron*, https://en.wikipedia.org/wiki/Multilayer_perceptron
- [22] Wikipedia, *Moving Average*, https://en.wikipedia.org/wiki/Moving_average
- [23] Wikipedia, *Fourier transform*, https://en.wikipedia.org/wiki/Fourier_transform
- [24] Sito ufficiale Processing, *Processing*, <https://processing.org/>