Università di Pisa

SCUOLA DI INGEGNERIA

Corso di Laurea in Ingegneria Informatica

Sviluppo di una app per rilevare operazioni di sollevamento manuale di carichi pesanti in contesti industriali

Sviluppo modulo di detection dell'azione di abbassamento

Candidato
Marco Parola

Relatori
Prof.ssa Lazzerini Beatrice
Dr. Francesco Pistolesi

Anno Accademico 2019–2020

Dedicato a Rachele e Alessandro...
...avete reso tutto questo possibile

Indice

Capitolo 1

Introduzione

> *" Le operazioni di trasporto o di sostegno di un carico ad opera di uno o più lavoratori, comprese le azioni del sollevare, deporre, spingere, tirare, portare o spostare un carico, che, per le loro caratteristiche o in conseguenza delle condizioni ergonomiche sfavorevoli, comportano rischi di patologie da sovraccarico biomeccanico, in particolare dorso-lombari ".*

Questa è la definizione di Movimentazione Manuale dei Carichi MMC (o, in inglese, Manual Handlng of Loads, MHL), data dall'articolo 167, decreto legislativo 81/08.
Queste manovre, svolte quotidianamente da operai sul posto di lavoro, rappresentano una delle cause che favoriscono l'insorgere di disturbi e patologie alla colonna vertebrale. Infatti, se consideriamo i dati provenienti dagli enti di previdenza sociale relativi al tipo e ai numeri degli infortuni sul lavoro e delle malattie professionali, questi evidenziano come, a livello mondiale, a prevalere siano malattie professionali classificabili come disturbi Muscolo-Scheletrici.
Dalle statistiche emerge quanto i rischi derivanti da MMC siano un problema su molteplici fronti, nonostante l'introduzione di normative, che negli anni hanno posto sempre più attenzione alla tutela dei lavoratori, oltre a sensibilizzare i datori di lavoro e standardizzare il modo in cui si svolge la mansione.
Il D.Lgs. 81/08, nell'allegato XXIII, introduce metodi di valutazione del rischio, presentati nella norma ISO-11228, redatta a seguito di studi effettuati dal NIOSH (National Institute for Occupational Safety and Health Administration), l'agenzia federale statunitense che si occupa di ricerca e formulazione di raccomandazioni per prevenire infortuni e malattie sul lavoro.
Tale norma è divisa in tre parti, una per ogni tipologia di movimentazione dei carichi, nello specifico:

- Sollevamento e trasporto;
- Traino e spinta;
- Maneggiare carichi leggeri ad alta frequenza;

La movimentazione analizzata in questa tesi è quella relativa al sollevamento dei carichi.

In generale, si rende necessario quindi, procedere ad una corretta valutazione del rischio da movimentazione manuale di carichi, al fine di attuare idonei interventi di prevenzione e protezione che vadano a mitigare, se non annullare, eventuali danni a carico degli operatori. Considerando poi che quasi un terzo dei lavoratori dichiara di svolgere quotidianamente questo tipo di operazione, è lecito pensare che, avere uno strumento capace di analizzare il modo in cui si effettua questo task, potrebbe permettere di comprendere il problema più a fondo e cercare una soluzione mirata.

L'obiettivo finale è, quindi, lo sviluppo di uno strumento, che monitori costantemente i movimenti di un lavoratore e valuti il rischio a cui si espone, compiendo determinati mansioni; al fine di poter correggere gli errori compiuti durante l'esecuzione e trovare soluzioni ergonomiche atte a migliorare la qualità e l'ambiente lavorativo.

Lo scopo di questa tesi è analizzare i movimenti compiuti durante il **sollevamento di un carico elevato** mediante dispositivi wearable, che contengono al loro interno sensori da cui raccogliere dati, che verranno analizzati in un secondo momento, per capire se i dispositivi hardware, utilizzati nella prima fase di storage dei dati, siano sufficienti a distinguere un' operazione eseguita in modo corretto, da una eseguita scorrettamente e che potrebbe causare lesioni. Questa analisi deve gettare le basi per poter realizzare uno strumento che, in tempo reale, possa avvisare un utente, ogni qual volta compia una manovra potenzialmente dannosa; in particolare il modulo, che viene analizzato con più attenzione e su cui la tesi si concentra maggiormente, è la rilevazione dell'abbassamento.

Capitolo 2

Movimentazione

In prima battuta è fondamentale comprendere come debba essere compiuto il task del sollevamento; con lo scopo di formare il personale, che durante la giornata lavorativa si trova a dover affrontare il movimento considerato.

2.1 Movimento corretto

Il modo corretto, al fine di evitare infortuni e minimizzare l'indice di rischio durante il compimento del task critico, è piegare le gambe, mantenendole leggermente divaricate, all'altezza delle spalle e, mantenendo un'inclinazione lieve e costante del busto verso il carico, il quale deve essere posizionato alla minore distanza possibile dai piedi (circa 10 cm), tornare in posizione eretta facendo forza principalmente sulle gambe.
La figura 2.1 illustra i passaggi da seguire per un corretto sollevamento.

Figura 2.1: Sollevamento corretto.

2.2 Movimento scorretto

Un modo scorretto di sollevare un carico è invece illustrato nella seguente figura, dove non si utilizzano le gambe, che rimangono stese durante tutta l'esecuzione della manovra, per aiutarsi nella distribuzione del peso, ma ci si piega con la sola schiena, esponendola di fatto a un alto rischio di disturbi dovuti a sovraccarico biomeccanico delle vertebre e dei dischi intervertebrali.
La figura 2.2 illustra un esempio di movimenti assolutamente da evitare.

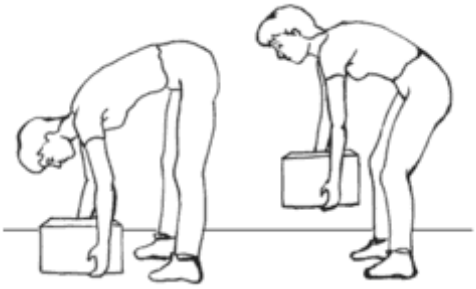

Figura 2.2: Sollevamento scorretto.

Capitolo 3

Raccolta dati

Per poter eseguire uno studio del rischio che corre una persona, sollevando un carico elevato, si affronta una prima fase in cui si raccolgono i dati relativi alla movimentazione di quest'ultima.
Dunque si rende necessario uno strumento che permetta di raccogliere e catalogare informazioni, con cui poter ricostruire il movimento e, in uno step successivo, eseguire un' analisi per poter quantificare il rischio.

3.1 Sistema per la raccolta ed il salvataggio di dati

Il sistema, di cui occore munirsi, dovrà contenere al suo interno sensori di vario genere, che gli permettano di registrare i movimenti e le variazioni delle condizioni ambientali esterne, a cui è sottoposto.
I sensori che prenderemo in considerazione, di cui l'hardware dovrà essere fornito, sono i seguenti:

- **Accelerometro triassiale**, misura l'accelerazione che subisce il dispositivo sui tre assi cardinali.
- **Giroscopio triassiale**, misura la velocità angolare del dispositivo attorno ai tre assi del sistema di riferimento.
- **Magnetometro triassiale**, misura l'intensità del flusso del campo magnetico terrestre relativamente ai tre assi del sistema di riferimento. Tramite questo tipo di sensore possiamo ricavare informazioni sull'angolo che questo dispositivo forma con il campo magnetico terrestre.
- **Barometro**, misura la pressione atmosferica alla quale il dispositivo è sottoposto; grazie alla elevata sensibilità di questo sensore è possibile individuare variazioni di pressione anche molto piccole, dell'ordine dei mbar.

Un'altra proprietà che il nostro sistema deve possedere è essere **mini-invasivo**, infatti la persona che utilizza uno strumento di questo genere, non dovrebbe quasi accorgersi della sua presenza, affinché non si senta eccessivamente controllato oppure disturbato nei movimenti che compie; questo ci permetterà di effettuare delle misurazioni naturali e non falsate.

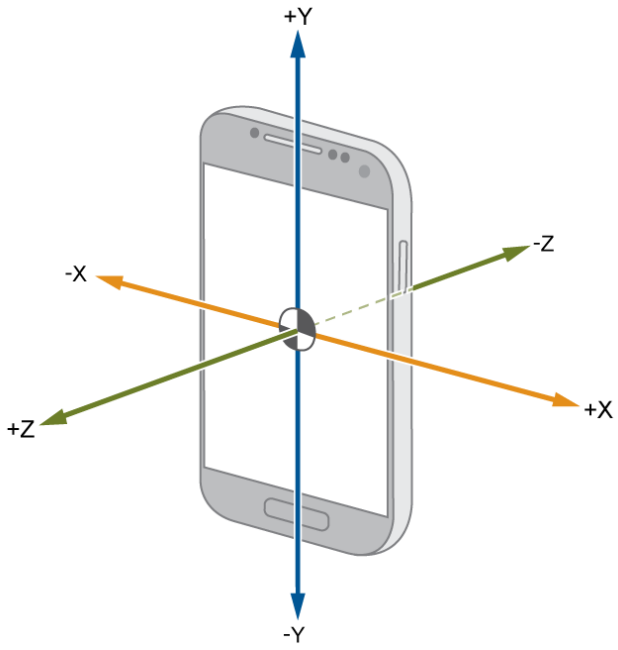

Figura 3.1: Sistema di coordinate utilizzato dall'API Sensor.

La figura 3.1 mostra il sistema di riferimetno cartesiano, di cui i sensori fanno uso, per descrivere la posizione e il moviemento nello spazio del dispositivo che li contiene.

3.2 Applicazione Android per la raccolta

La soluzione che abbiamo trovato, che rispecchia i requisiti sopra riportati è un'applicazione Android, che gira su due dispositivi hardaware distinti: smartphone e smartwatch.
La scelta del sistema operativo, su cui si basa il sistema, è dovuta a vari fattori; in primo luogo la vasta gamma di prodotti, su cui è installato Android, molto differenti tra loro sia per caratteristiche, che per fasce di prezzi. Inoltre l'applicazione è stata sviluppata sull'IDE Android Studio, che integra interessanti funzionalità sia per la creazione di interfacce grafiche, sia per la parte relativa all'accesso al file system dei dispositivi compatibili.
In particolare durante lo svolgimento della tesi è stato adoperato uno smartphone Samsung Galaxy S6, sistema operativo Android 7.0 e uno smartwatch Huawei Watch 2, sistema operativo Wear OS 2.0 .
Di seguito i 3 moduli principali, nei quali è organizzato il progetto:

- wear
- shared_mobile_watch
- mobile

Il fine di questo sistema è produrre nel file system del telefono, in particolare nella directory "Documents", *Environment.DIRECTORY_DOCUMENTS*, otto file CSV, quattro relativi allo smartphone: *magnetometr_phone, gyroscope_phone, accelerometr_phone, pressure_phone* e quattro relativi allo smartwatch: *magnetometr_watch, gyroscope_watch, accelerometr_watch, pressure_watch*. Ognuno di questi 4 file descrive il segnale proveniente da uno dei sensori precedentemente citati.

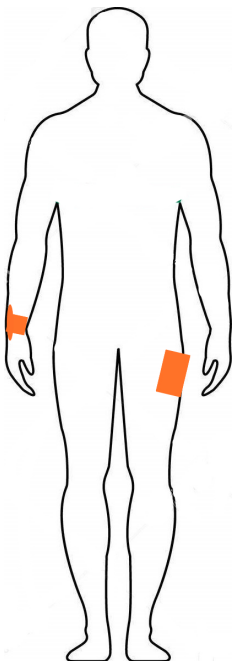

Figura 3.2: Posizionamento dei dispositivi indossabili sull'utente.

3.2.1 Modulo wear

Il modulo wear implementa la parte dell'applicazione relativa allo smartwatch, essa ha la funzionalità di raccogliere i dati provenienti dai suoi sensori e contemporaneamente inviarli tramite bluetooth al telefono.
Di seguito non verrà riportato il codice di tutto il modulo, ma solamente la parte relativa alla raccolta dei dati e all'invio. Nella seguente porzione di codice viene mostrato il callback che viene chiamato ogni volta che il valore di uno dei sensori subisce un cambiamento: ad ogni chiamata viene inserito nel buffer (che può contenere 50 elementi) un oggetto che contiene informazioni relative al cambiamento appena avvenuto.

```
public class MainActivity extends WearableActivity implements SensorEventListener {

    private TextView textView;
    private SensorManager sensorManager;
    ArrayList<Sensor> sensorList;
    DataSensor buffer[];
    int quanti;
    int iterazione = 0;
    ArrayList<Integer> type;
    boolean running;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        textView = findViewById(R.id.text);

        // register sensor
        sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        sensorList = new ArrayList<Sensor>();
        sensorList.add(sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD));
        sensorList.add(sensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE));
        sensorList.add(sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER));
        sensorList.add(sensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE));

        quanti = 0;
        buffer = new DataSensor[50];
        for(int i=0; i<50; i++)
            buffer[i] = new DataSensor();

        running = false;
        setAmbientEnabled();
    }

    public void startButton(View v) {
        if(!running) {
            Log.i("start", "start");
            Toast.makeText(getBaseContext(), "Start recording...", Toast.LENGTH_LONG).show();
            for (Sensor s: sensorList) {
                sensorManager.registerListener(this, s, SensorManager.SENSOR_DELAY_FASTEST);
            }
            running = true;
        }
    }

    public void stopButton(View v)
    {
        if(running) {
            Log.i("stop", "stop");
            Toast.makeText(getBaseContext(), "Stop recording...", Toast.LENGTH_LONG).show();
            sensorManager.unregisterListener(this);
            running = false;
        }
    }

    public void onSensorChanged(SensorEvent event) {
```

```
            buffer[quanti].setSensorType(event.sensor.getType());
            buffer[quanti].setTimestamp(String.valueOf(event.timestamp));
            buffer[quanti].setValue0(String.valueOf(event.values[0]));
            if(event.sensor.getType() != Sensor.TYPE_PRESSURE){
                buffer[quanti].setValue1(String.valueOf(event.values[1]));
                buffer[quanti].setValue2(String.valueOf(event.values[2]));
            }

            quanti++;

            if (quanti == 50) {
                new SendMessage("/data", buffer).start();
                for(int i=0; i<50; i++){
                    Log.i("data" + iterazione + "_" + i, buffer[i].getSensorType()+"_");
                }
                quanti = 0;
                iterazione++;
            }
        }

        public void onAccuracyChanged(Sensor sensor, int accuracy) {}

        class SendMessage extends Thread {
            String path;
            DataSensor message[];

            //Constructor for sending information to the Data Layer//
            SendMessage(String p, DataSensor m[]) {
                path = p;
                message = m;
            }

            public void run() {

                //Retrieve the connected devices//
                Task<List<Node>> nodeListTask =
                        Wearable.getNodeClient(getApplicationContext()).getConnectedNodes();
                try {

                    //Block on a task and get the result synchronously//
                    List<Node> nodes = Tasks.await(nodeListTask);
                    for (Node node : nodes) {
                        try {
                            ByteArrayOutputStream bos = new ByteArrayOutputStream();
                            ObjectOutputStream oos = new ObjectOutputStream(bos);
                            oos.writeObject(message);

                            //Send the message//
                            Wearable.getMessageClient(MainActivity.this).sendMessage(node.getId(),
                                                                    path, bos.toByteArray());
                            oos.close();
                            bos.close();
                        }
                        catch(IOException e){e.printStackTrace();}
                    }
                }
                catch (ExecutionException e) {e.printStackTrace();}
                catch (InterruptedException e) {e.printStackTrace();}
            }
        }
}
```

3.2.2 Modulo shared_mobile_watch

Questo modulo molto semplice è una libreria android richiamata dai moduli mobile e wear e contenente un'unica classe DataSensor che implementa l'interfaccia Serializable, poichè le istanze di tale classe dovranno essere inviate dall'orologio al telefono. Tale classe memorizza le informazioni principali relative ad un cambiamento di valore rilevato da uno dei sensori, di cui abbiamo discusso precedentemente: il tipo di sensore sensorType, tre valori value0, value1, value2 e l'istante in cui è avvenuto il cambiamento timestamp. E' buona pratica aggiungere in tutti i progetti, che contengono più moduli che lavorano con gli stessi tipi di classi, una libreria android, che contenga le classi necessarie ad entrambi; nonostante questo possa creare alcuni problemi durante la compilazione se ci sono aggiornamenti della versione.

```
public class DataSensor implements Serializable {
    private int sensorType;
    private String value0;
    private String value1;
    private String value2;
    private String timestamp;

    public int getSensorType() {
        return sensorType; }
    public void setSensorType(int sensorType) {
        this.sensorType = sensorType; }

    public String getValue0() {
        return value0; }
    public void setValue0(String value0) {
        this.value0 = value0; }

    public String getValue1() {
        return value1; }
    public void setValue1(String value1) {
        this.value1 = value1; }

    public String getValue2() {
        return value2; }
    public void setValue2(String value2) {
        this.value2 = value2; }

    public String getTimestamp() {
        return timestamp; }
    public void setTimestamp(String timestamp) {
        this.timestamp = timestamp; }

    public byte[] getBytes() {
        ByteArrayOutputStream bos = new ByteArrayOutputStream();
        ObjectOutput out = null;
        try {
            out = new ObjectOutputStream(bos);
            out.writeObject(this);
            out.flush();
            return bos.toByteArray();
        } catch (IOException e) {
            Log.e("DataSensor", e.getLocalizedMessage(), e);
        } finally {
            try {
                bos.close();
            } catch (IOException ex) {
                // ignore close exception
            }
        }
        return new byte[]{};
    }
}
```

3.2.3 Modulo mobile

In questo modulo vengono ricevuti i dati provenienti dall'orologio grazie al service MessageService, dichiarato nel manifesto.

```
public class MessageService extends WearableListenerService {

    public void onMessageReceived(MessageEvent messageEvent) {

        if (messageEvent.getPath().equals("/data")) {
            final DataSensor[] message = getDataSensor(messageEvent.getData());
            Intent messageIntent = new Intent();
            messageIntent.setAction(Intent.ACTION_SEND);
            messageIntent.putExtra("message", message);
            LocalBroadcastManager.getInstance(this).sendBroadcast(messageIntent); }
    }

    public DataSensor[] getDataSensor(byte[] b) {
        ByteArrayInputStream bis = new ByteArrayInputStream(b);
        ObjectInput in = null;
        try {
            in = new ObjectInputStream(bis);
            return (DataSensor[]) in.readObject();
        }
        catch (IOException e) { Log.e("MainMobile", e.getLocalizedMessage(), e);}
        catch (ClassNotFoundException e) { Log.e("MainMobile", e.getLocalizedMessage(), e); }
        finally {
            try {
                if (in != null) {
                    in.close();
                } } catch (IOException ex) {ex.printStackTrace();} }
        return null; }
}
```

Tale classe, non scrive direttamente i dati sul file, ma li invia in broadcast ad altri thread. Tali dati vengono ricevuti e scritti da un un thread Receiver, dichiarato nel metodo onCreate e richiamato all'avvio dell'applicazione. Ogni volta che viene inviato un buffer di 50 elementi dal service precedentemente descritto, viene invocato questo metodo del thread che riceve il pacchetto e ne trascrive i dati contenuti, chiamando la funzione writeFile(DataSensor buf[]), di cui non è riportato il codice, in quanto molto semplice (chiama la open su un file, scrive su di esso con il metodo write ed infine chiude il file con il metodo close).

```
    public void onReceive(Context context, Intent intent) {
        // ottengo un array di oggetti DataSensor da uno stream di byte
        DataSensor buffer[];
        buffer = (DataSensor[]) intent.getSerializableExtra("message");

        for(int i=0; i<50; i++){
            Log.i("data" + iterazione + "_" + i, buffer[i].getSensorType() + "_" );
        }
        iterazione++;
        writeFile(buffer);
    }
```

Sempre in questo modulo, inoltre, viene fatta la registrazione dei sensori contenuti nel telefono, similmente a come viene fatta nel modulo wear. Per evitare ripetizioni del codice nella documentazione, è omessa anche questa parte.

3.3 Esperimenti di registrazione

Dopo essersi muniti di uno strumento che rispecchi i requisiti precedentemente elencati, si affronta una fase in cui si eseguono uno o più esperimenti di registrazione: ogni esperimento è definito da un set di azioni, che dovranno essere svolte da un candidato, mentre il sistema di registrazione e salvataggio dei dati memorizza tutte le informazioni neccessarie a ricostruire la movimentazione compiuta.

3.3.1 Esperimento 1

L'esperimento svolto comprende le seguenti tre azioni:

- camminata
- sollevamento del carico
- rilascio del carico.

La ultime due azioni possono essere fatte in maniera sicura oppure dannosa per la schiena.
Di seguito è riportato uno schema del modulo:

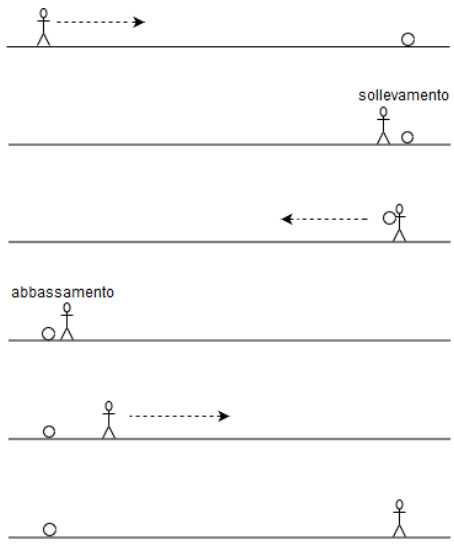

Figura 3.3: Schema dei task dell'esperimento 1.

La figura 3.3 descrive un ciclo del percorso totale compiuto. L'esperimento completo prevederà 2 cicli, per un totale di 2 sollevamenti e due rilasci del carico

In generale gli esperimenti da svolgere, dovrebbero essere molteplici e dovrebbero prevedere una gamma molto più ampia di azioni, che un lavoratore potrebbe compiere: permanenza alla scrivania, utilizzo di dispositivi elettronici, utilizzo di attrezzatura da operaio. Inoltre dovrebbero essere effettuati in condizioni ambientali differenti, per poter realizzare un sistema il più universale possibile.

Capitolo 4

Analisi

La fase che segue la raccolta è l'analisi dei dati ottenuti dall'esperimento eseguito. Questo passaggio prevede l'importazione dei dati sull'ambiente di calcolo MATLAB, l'analisi e la processazione dei segnali, mediante l'utilizzo del pacchetto Signal Processing Toolbox 8.1.
Questa fase di analisi si divide in due sottofasi:

- Individuazione dell'istante in cui avviene il task del sollevamento. L'obiettivo di questa prima fase dell'analisi è individuare gli istanti, in cui viene eseguito il movimento, in modo da non dover passare alla fase della classificazione tutti i segnali dei sensori, ma solamente alcune porzioni, e rendere il sistema più efficiente.
- Classificazione della manovra. Questa fase viene fatta da una rete neurale, che riceve in input i parametri più significativi, calcolati dalle porzioni di segnale individuate nella fase di detection, e classifica la manovra come *corretta*, *scorretta* o *carico assente*.

4.1 Visualizzazione grafici

La prima fase di analisi ha previsto la creazione dei grafici, risultanti dalle rilevazioni degli otto sensori (relativi alla stessa registrazione), per poter avere una panoramica dell'andamento dei segnali collezionati, in modo da poter compiere le prime considerazioni ed inquadrare eventuali problemi che potrebbero insorgere.
MATLAB mette a disposizione la funzione *plot* per poter visualizzare grafici di forme diverse, in relazione ai tipi che vengono passati come argomento, quando la fuzione viene chiamata: Se x è un vettore, *plot(x)* produce un grafico lineare degli elementi di x contro l'indice degli elementi di x. Se venogno specificati due vettori x e y come argomento, *plot(x, y)* produce un grafico, in cui i valori della x vengono posizionati sull'ascissa e quelli di y sull'ordinata.
Inoltre il comando può essere chiamato specificando alcune opzioni con la seguente sintassi: *plot(x, y, NomeOpzione1, ValoreOpzione1, NomeOpzione2, ValoreOpzione2, ..., NomeOpzioneN, ValoreOpzioneN)*. Con tale comando si possono realizzare grafici personalizzati, come lo spessore della linea *plot(x, y, 'LineWidth', 3)* o il colore *plot(x, y, 'red')*.

4.1.1 Barometro smartphone

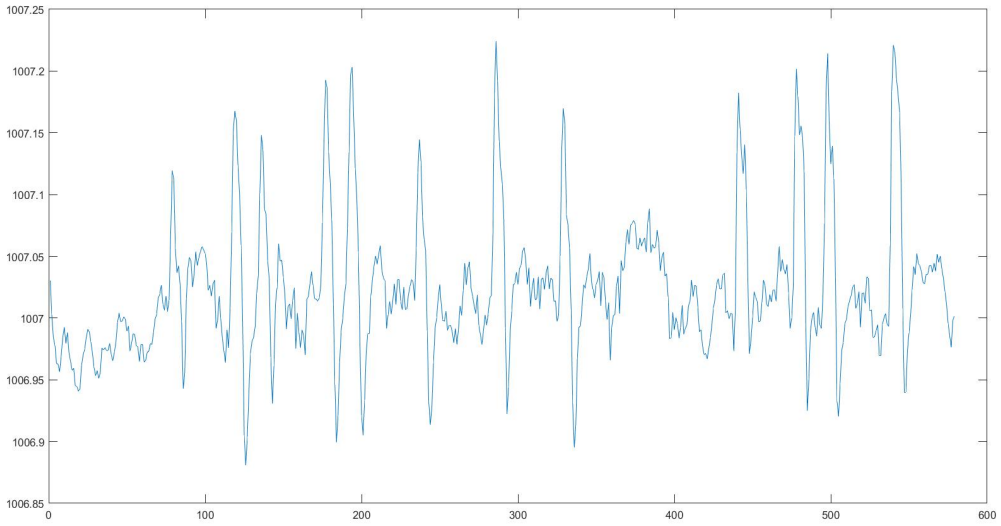

Figura 4.1: Segnale barometrico dello smartphone.

4.1.2 Barometro smartwatch

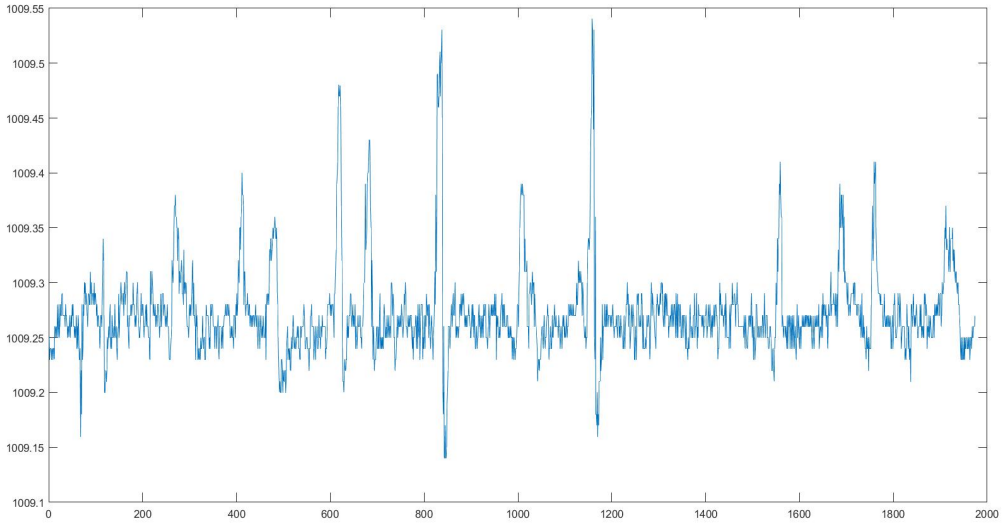

Figura 4.2: Segnale barometrico dello smartwatch.

4.1.3 Accelerometro smartwatch

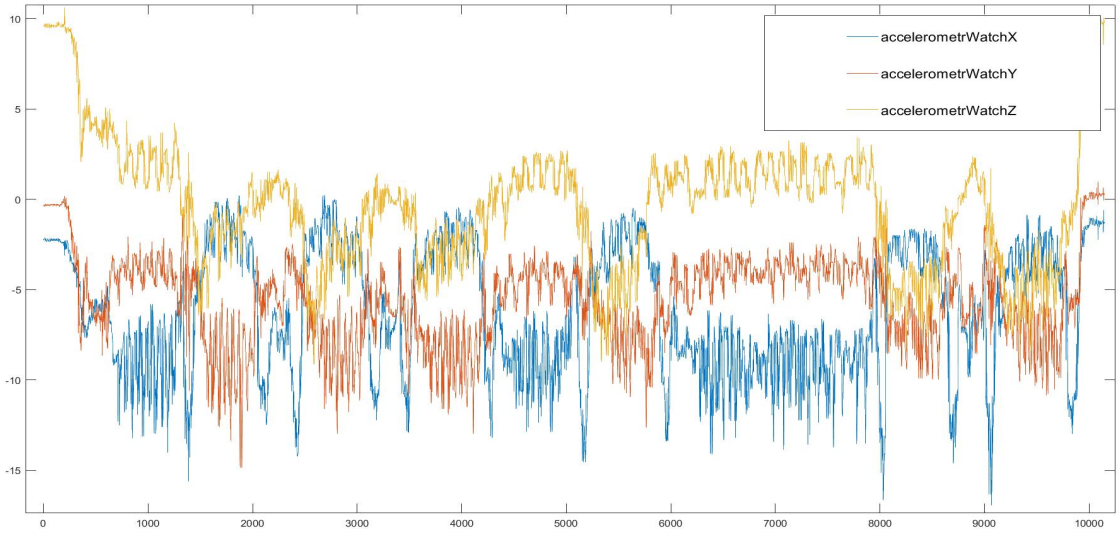

Figura 4.3: Grafico accelerometro triassiale dello smartwatch.

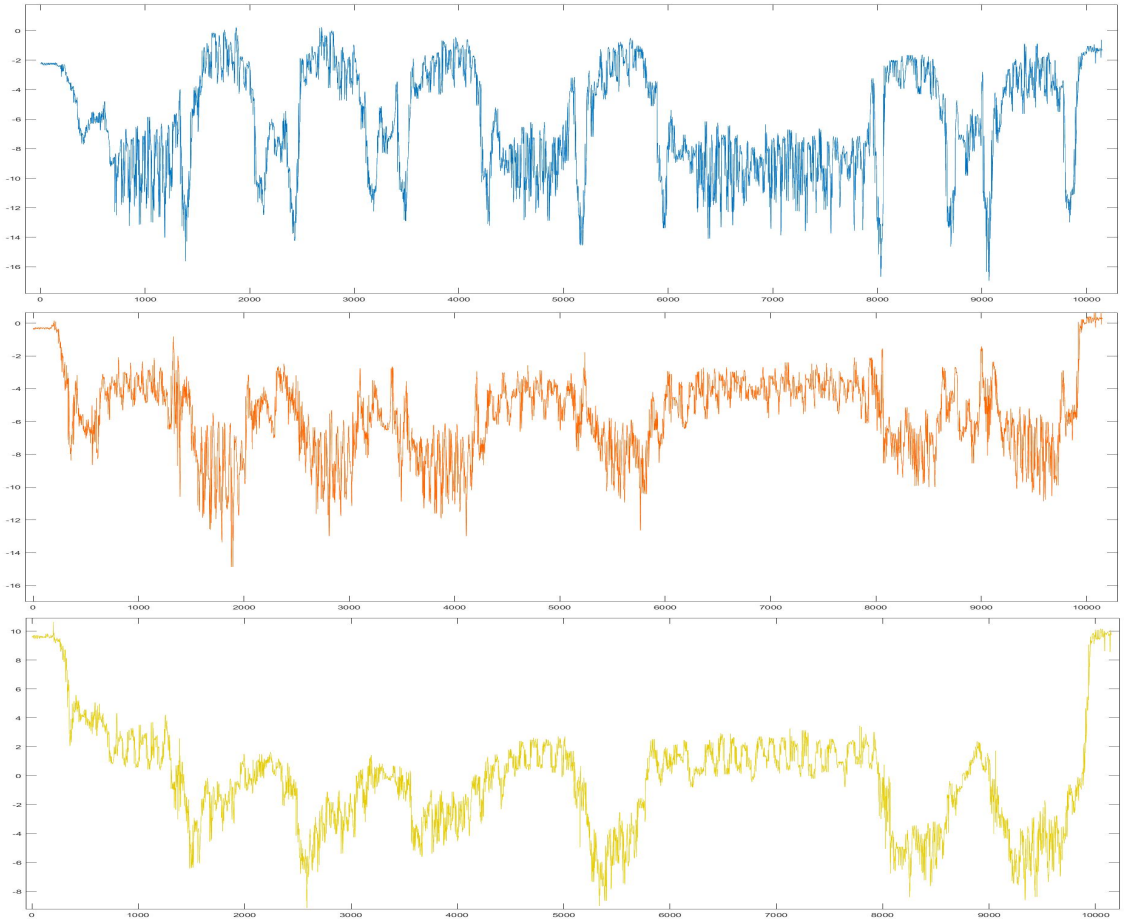

Figura 4.4: Scomposizione del segnale accelerometrico triassiale in tre grafici.

4.1.4 Accelerometro smartphone

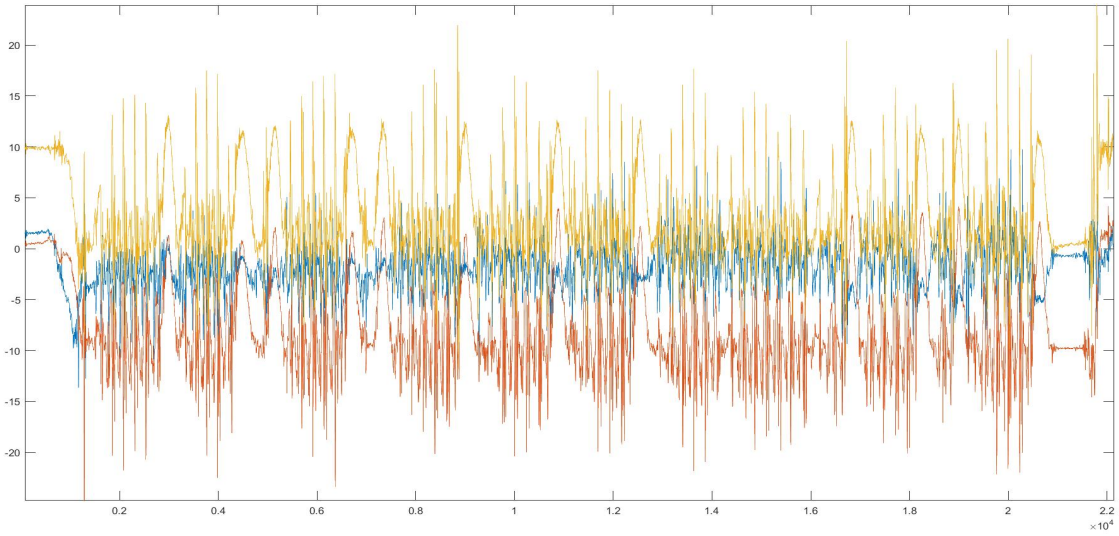

Figura 4.5: Grafico accelerometro triassiale dello smartphone.

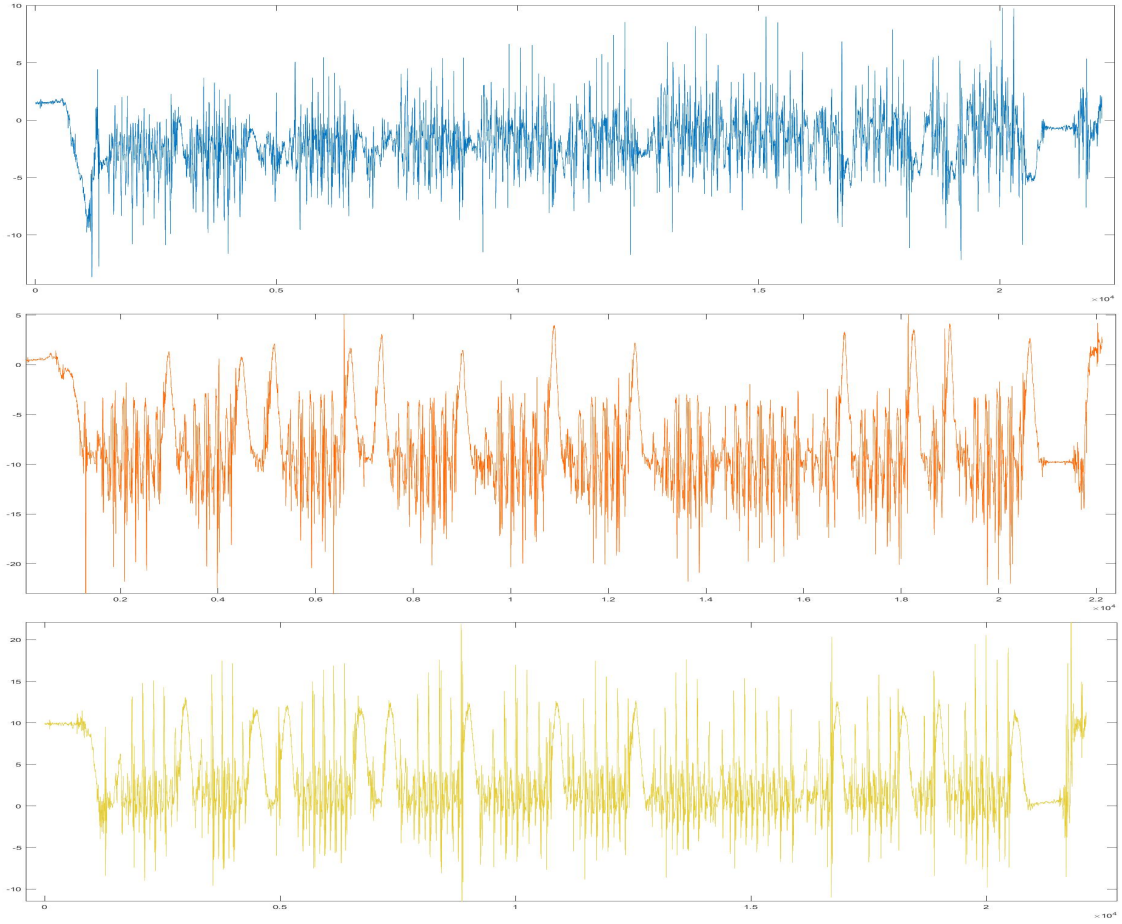

Figura 4.6: Scomposizione del segnale accelerometrico triassiale in tre grafici.

4.1.5 Magnetometro smartphone

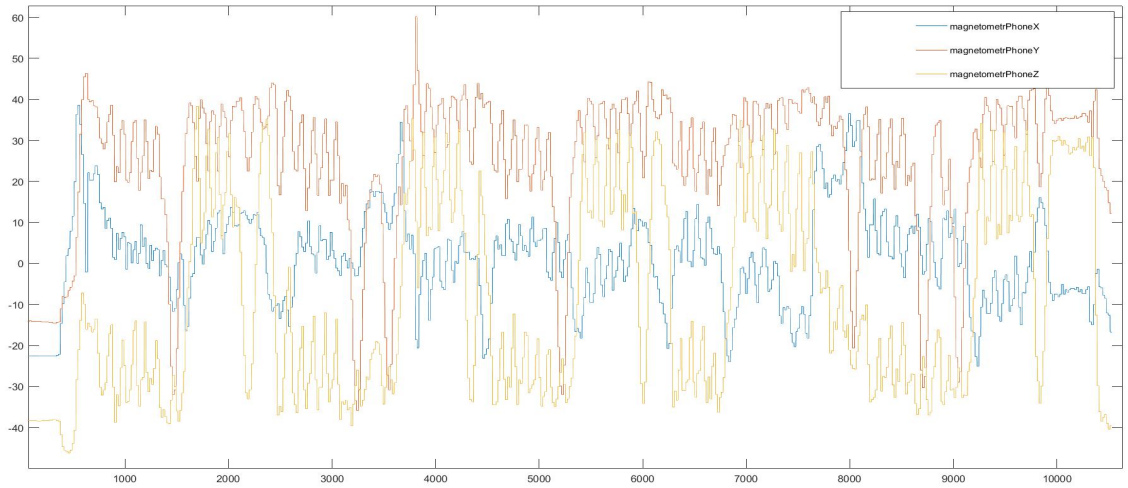

Figura 4.7: Grafico magnetometro triassiale dello smartphone.

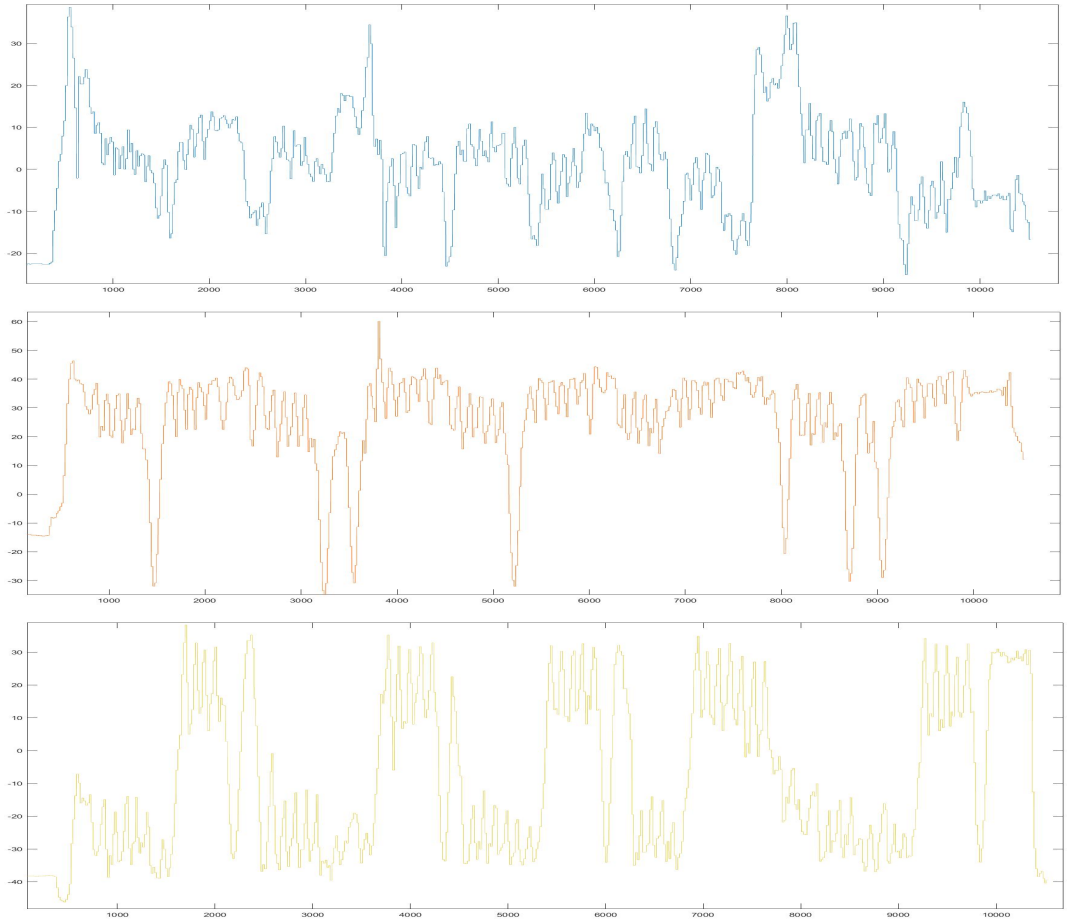

Figura 4.8: Scomposizione del segnale magnetometrico triassiale in tre grafici.

4.1.6 Giroscopio smartphone

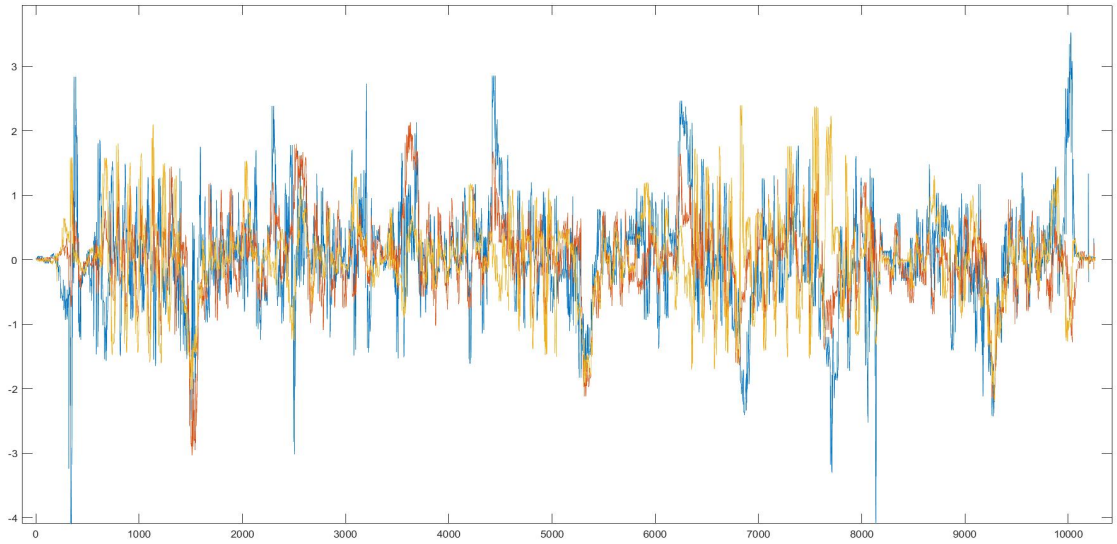

Figura 4.9: Grafico accelerometro triassiale dello smartphone.

4.2 Analisi di una finestra temporale

La figura 4.10 mostra il grafico di una finestra dei segnali relativi ad un abbassamento, in cui il segnale posto per primo, che detta gli istanti da analizzare, è quello barometrico dello smartwatch, seguito dal segnale barometrico dello smartphone, accelerometrico lungo l'asse X dello smartwatch ed infine accelerometrico lungo l'asse Y dello smartphone.

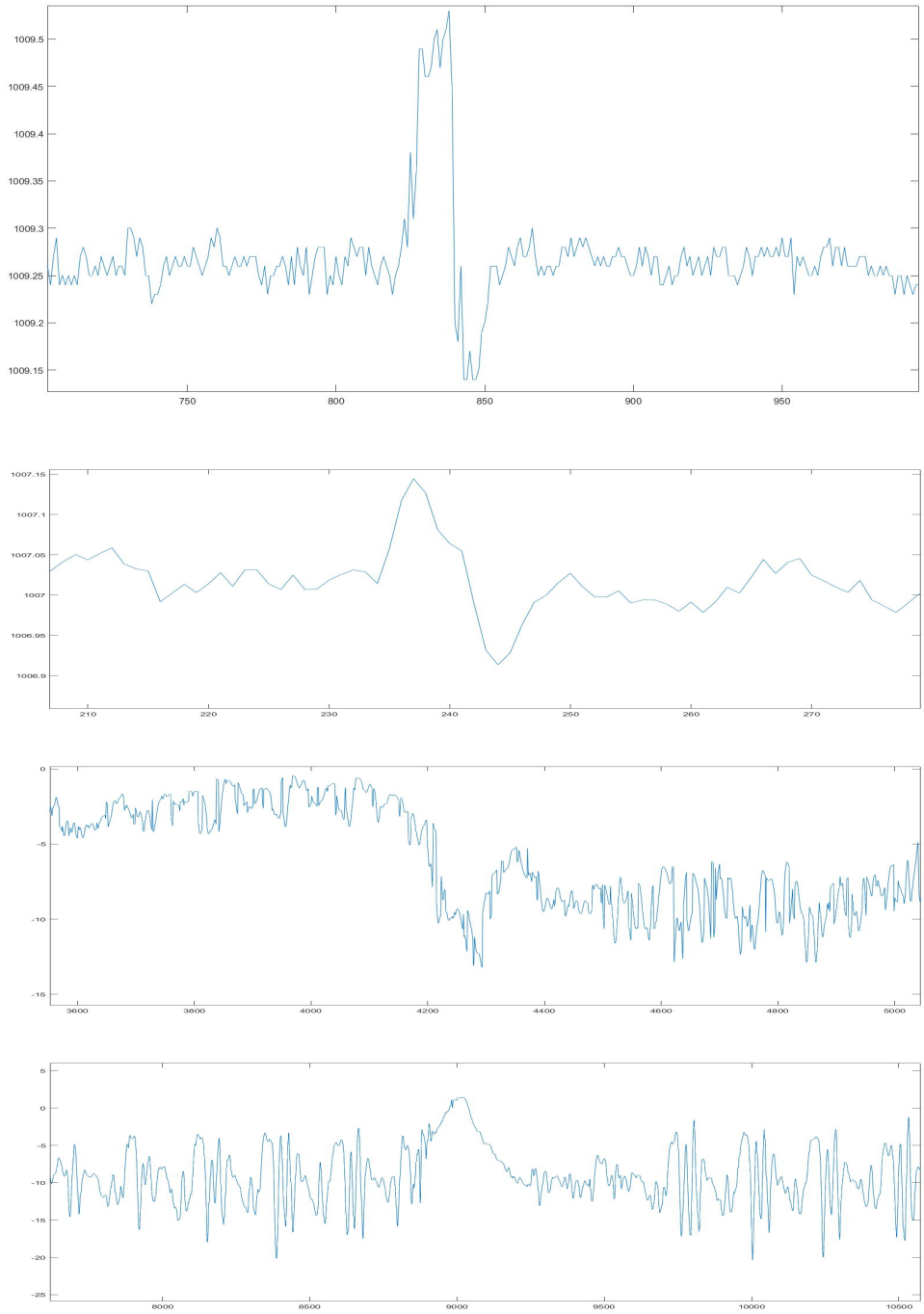

Figura 4.10: Finestra relativa ad una abbasamento del barometro dello smartwatch, barometro dello smartphone, accelerometroX smartwatch e accelerometriY smartphone.

4.3 Il problema degli outlier

Un problema che si riscontra dalla fase precedente, è la presenza di outlier: sample caratterizzati da valori non appartenenti ad un intervallo di valori attesi e numericamente distanti dal resto dei dati raccolti.
I valori anomali possono avere molte cause, essendo dovuti a improvvisi glitch del sistema: un apparato hardware per eseguire misurazioni potrebbe aver subito un malfunzionamento transitorio oppure potrebbe essersi verificato un errore nella trasmissione o trascrizione dei dati.

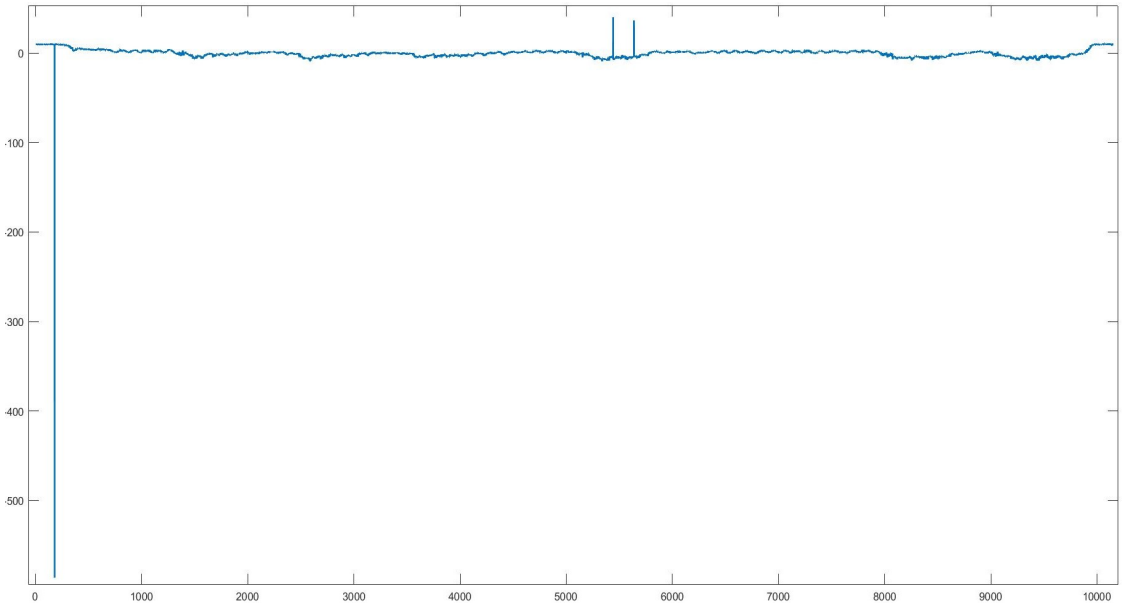

Figura 4.11: Grafico accelerometro asse X dello smartwatch.

Costruendo il grafico dei segnali (figura 4.11), è evidente la presenza di questi valori inattesi dai picchi che compaiono lungo il segnale, tali sample non permettono una scala ragionevolmente visualizzabile, per cui è neccessaria una fase di signal preprocessing, nella quale vengono eliminati questi campioni fuoriscala.
Essendo gli outlier un numero irrisorio (3 sample per il grafico in figura 4.11), possiamo procedere ad una pulizia manuale del segnale, cercando dal grafico generato da MATLAB, gli indici corrispondenti ai campioni presi in esame ed in seguito eliminarli.

4.4 Il segnale barometrico per rilevare la manovra

Il sensore che dà maggiori informazioni sul momento in cui avviene l'abbassamento è il barometro dell'orologio; nel grafico, infatti, si nota che in corrispondenza di un abbassamento si ha un innalzamento della pressione atmosferica del barometro, con un conseguente abbassamento quando ci si alza nuovamente.
L'elevato livello di rumore bianco di questo segnale rende difficile il compito di trovare delle regole generali per individuare algoritmicamente il momento in cui la persona si abbassa.
Inoltre, il sensore è affetto da disturbi che possono influenzare l'andamento del segnale (es. correnti d'aria o le variazioni di pressione generate dall'aprire o chiudere una finestra).

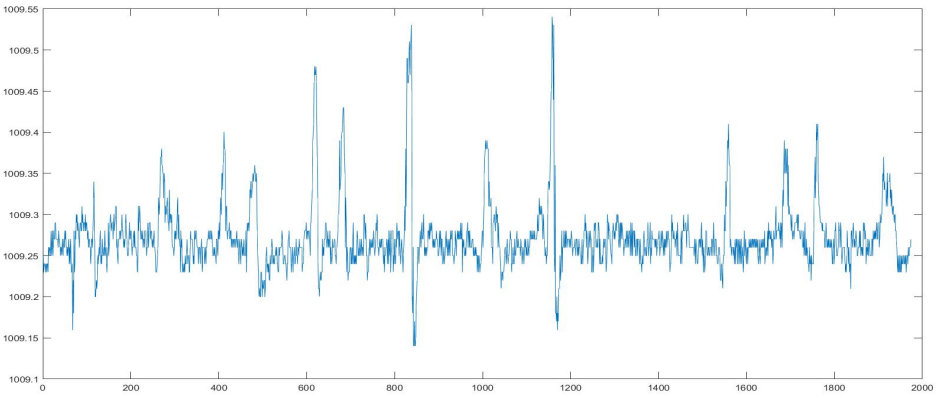

Figura 4.12: Segnale barometrico dello smartwatch.

Per rendere più agevole la visualizzazione è opportuno applicare al segnale di partenza opportuni filtri. In particolare facendo vari tentativi, si giunge alla conclusione che, per ottenere una buonissima replica del segnale pulita e più lineare, si possono applicare un filtro a media mobile, seguito da un filtro gaussiano.

- Filtro a media mobile. Questo filtro prevede di ricostruire il segnale sostituendo al valore di ogni campione la media di campioni vicini. E' una tecnica matematica utilizzata per smussare le fluttuazioni nel segnale. Si dice "mobile" perché il numero degli elementi considerati è fisso (finestra), ma l'intervallo di tempo avanza.
- Filtro Gaussiano. Questo filtro viene applicato in cascata al precedente; anche in questo caso viene effettuata una media dei vicini, non aritmetica ma ponderata, in particolare ogni elemento verrà normalizzato, usando i coefficienti di una funzione gaussiana.

Il seguente codice MATLAB applica al segnale di partenza i due filtri, producendo i grafici riportati in seguito:

```
watch_mov = movmean(pressurewatch(:,2),50);
watchgaussian = smoothdata(watch_mov,'gaussian');
plot(watch_mov);
plot(watchgaussian)
```

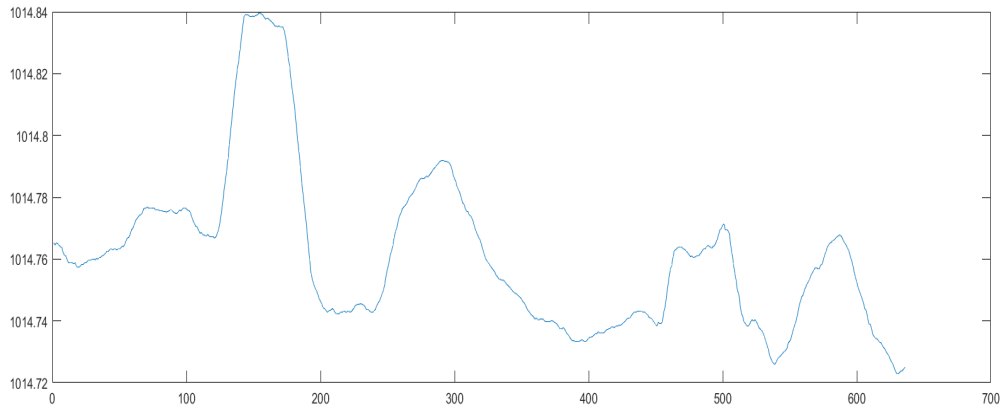

Figura 4.13: Segnale barometrico a cui è stato applicato il filtro a media mobile.

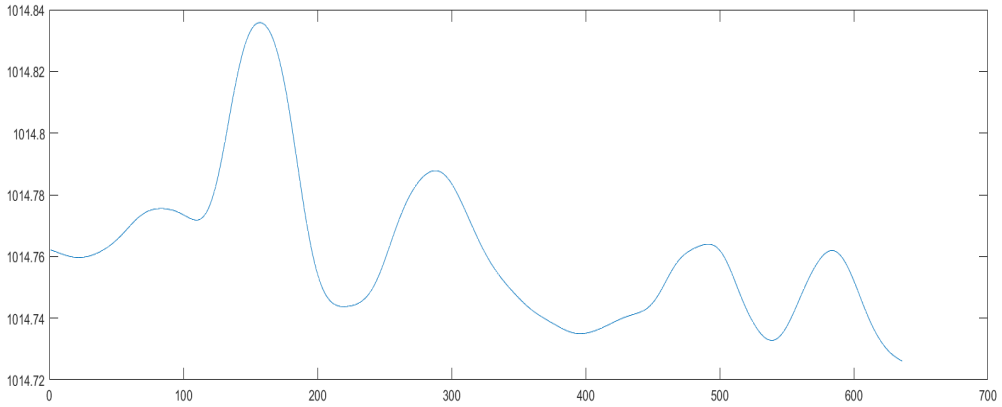

Figura 4.14: Segnale barometrico a cui sono stati applicati i filtri a media mobile e gaussiano in cascata.

Capitolo 5

Sviluppo Applicazione Realtime

Durante la fase finale è stato realizzato un sistema per il monitoraggio della movimentazione in tempo reale, seguendo i medesimi passaggi compiuti nella fase di analisi (vedi paragrafo 4). Infatti, anche questa fase, è divisa in due parti: l'individuazione degli istanti in cui sono state compiute le manovre, seguita dalla classificazione di queste ultime.
In realtà non è propriamente corretto definire questa applicazione realtime, in quanto la rilevazione non è immediata, ma si presenta un tempo di latenza dovuto principalmente all'aplicazione di filtri al segnale grezzo. Come spiegato nei paragrafi successivi, al segnale registrato, vengono applicati due filtri a media mobile (il filtro gaussiano è un filtro a media mobile, in cui i sample hanno pesi diversi): in questo tipo di filtro si considera un certo numero di campioni N, detto FINESTRA, per calcolare il valore del segnale ad un determinato passo; dunque è neccessario aver collezionato gli N/2 campioni precedenti e gli N/2 successivi. Se considerando che vengono applicati due filtri di questa tipologia in cascata, sono neccessari un numero "elevato" di campioni futuri, per calcolare il valore del segnale ad un istante precedente.
Il sistema è implementato da un'applicazione Android, divisa in tre moduli, come l'applicazione utilizzata durante la fase della raccolta dei dati (vedi paragrafo 3.2):

- **wear**, per la fase di detection
- **mobile**, per la fase di classificazione.
- **shared_mobile_watch**, una semplice libreria per la condivisioni di classi comuni ad entrambi i moduli precedenti

5.1 Detection

La fase di detection degli istanti in cui è avvenuto il task del sollevamento è sviluppata nel modulo *wear*, in quanto è il segnale che, se pulito correttamente dai rumori, è quello che evidenzia meglio il compimento di un abbassamento.

5.1.1 Realizzazione dei filtri per la pulizia del segnale

Come primo step è necessario compiere una scelta progettuale: come implementare i filtri per la pulizia del segnale, che realizzino il medesimo comportamento delle funzioni MATLAB *movmean(signal, window_size)* e *smoothdata(signal, 'gaussian')*, citate precedentemente (vedi paragrafo 4.1).
Alcune opzioni possibili sono le seguienti:

- scrivere uno script matlab (.m) contenente una funzoine ed esportare un un file .cpp (o .c) generato da MATLAB, contenente del codice scritto nel linguaggio di programmazione C++

(oppure C), da includere nel progetto Android ed utilizzare Android Native Kit (NDK), per sviluppare questa porzione del modulo.
Questa opzione è molto valida, e dunque da prendere in considerazione per una versione più ottimizzata del sistema; in particolare perchè l'utilizzo del linguaggio C/C++ su Andoroid, viene fatto nei casi in cui si presentino probemi di calcolo computazionale, che appunto sono risolti con questa tecnica. Tuttavia non è stata presa come scelta, a causa delle complicazioni che introduce nello sviluppo del progetto.
Il seguente script MATLAB mostra la funzione da scrivere, che prende come argomento un array di valori (il segnale grezzo) e come tipo di ritorno restituisce un altro array di valori (il segnale filtrato).

```
function gaussian_signal = signal_filter(x)
    watch_mov = movmean(x,50);
    gaussian_signal = smoothdata(watch_mov,'gaussian');
end
```

- facendo riferimento allo script MATLAB del punto precedente, è possibile creare un Package java, grazie al tool **MATLAB Builder JA**; includendo il file .jar prodotto all'interno del progetto android, è possibile effettuare la pulizia del segnale.
  Questa procedura ha come inconveniente la generazione di codice ridondante, per cui la scelta presa è quella descritta nel seguente punto.

- implementare from scratch i due filtri necessari. L'implementazione sarà descritta nel seguente paragrafo.

5.1.2 Memorizzazione dei segnali e valutazione delle lunghezze dei buffer

Il secondo passo da compiere, per realizzare questo modulo, è lo storage dei campioni che descrivono un certo segnale. Non potendo mantere in memoria tutto il segnale registrato dal barometro, perchè potenzialmente infinito, è molto importante definire la quantità di campioni che possiamo memorizzare, per questioni legate principalmente alle risorse limitate dei dispositivi di cui ci siamo dotati.
Per questa parte di modulo è necessario mantenere in memoria tre buffer, uno per il segnale grezzo *noisy_signal*, uno per il segnale a cui viene applicato il primo filtro *movmean_signal* e l'ultimo per il segnale pulito finale *gaussian_signal*. In particolare la scelta che è stata fatta è di 50 campioni per il primo segnale, poichè la frequenza di campionamento è di 50 Hz, e di 100 campioni per quanto riguarda i secondi due segnali, poichè dalla fase di analisi si può notare che il compimento di un abbassamento, corrisponde a circa 100 campioni del segnale barometrico dello smartwatch. Questi variabili costanti possono essere soggette a modifiche, in quanto valutate euristicamente, in particolare la dimensione dei secondi due buffer; dunque compiendo uno studio in futuro più dettagliato sulle modalità e i tempi, con cui può essere compiuto questo task, si possono modificare queste dimensioni, in modo da migliorare i risultati.

```
        private static int WINDOW_SIZE = 50;
        private static int GAUSSIAN_WINDOW_SIZE = 100;
        private static int PORTION_OF_SIGNAL = 100;

        ArrayList<Float> noisy_signal;
        ArrayList<Float> movmean_signal;
        ArrayList<Float> gaussian_signal;
```

Inoltre per ognuno dei precedenti segnali, viene memorizzato un array della stessa dimensione, per i timestamp: stesso indice descrive valore e istante di tempo dello stesso campione: *ArrayList<Long> noisy_signal_timestamp; ArrayList<Long> movmean_signal_timestamp; ArrayList<Long> gaussian_signal_timestamp;*

5.1.3 Implementazione filtri

```
if(event.sensor.getType() == 6){ // PRESSURE

    noisy_signal.add(event.values[0]);
    noisy_signal_timestamp.add(event.timestamp);
    counter++;

    if(counter == WINDOW_SIZE){
        for(Float f : noisy_signal)
            media += f;
        media /= WINDOW_SIZE;
    }

    if(counter > WINDOW_SIZE){

        Float removed = noisy_signal.remove(0);
        noisy_signal_timestamp.remove(0);
        media = media + ((event.values[0] - removed) / WINDOW_SIZE);

        movmean_signal.add(media);
        movmean_signal_timestamp.add(noisy_signal_timestamp.get(WINDOW_SIZE/2));

        if(counter >  GAUSSIAN_WINDOW_SIZE + WINDOW_SIZE){
            movmean_signal.remove(0);
            movmean_signal_timestamp.remove(0);

            Float fl = gaussian_average(movmean_signal);


            gaussian_signal.add(fl);
            gaussian_signal_timestamp.add(movmean_signal_timestamp.get ...
                                        ...(GAUSSIAN_WINDOW_SIZE / 2));

            if(counter >  WINDOW_SIZE + GAUSSIAN_WINDOW_SIZE + PORTION_OF_SIGNAL){
                gaussian_signal.remove(0);
                gaussian_signal_timestamp.remove(0);
            }
        }
    }
}
```

5.1.4 Testing modulo detection

In questa fase è emersa l'impossibilità di poter realizzare il modulo di rilevazione della manovra, come descritto nel paragrafo 5.1.2, poichè l'applicazione di 3 filtri in cascata è un'oprazione di complessità, sia temporale che spaziale, non indifferente, per cui l'orologio non riesce a stare al passo del calcolo del segnale filtrato con l'arrivo dei campioni dai vari sensori. In particolare eseguendo l'applicazione in modalità debug e andando a leggere lo stacktrace, si può notare come l'esecuzione del Garbage-Collector (un thread Daemon che sta in esecuzione in background, che si

occupa della gestione della memoria) non permetta al Main-thread di mantere traccia dei campioni e di calcolarne il segnale pulito.

5.1.5 Seconda implementazione filtri

In questa parte dello sviluppo dell'applicazione si è reso necessario riprogettare il modulo della detection in modo che non sia computazionalmente pesante per smartwatch: lo smoothing del segnale viene fatto solamente con un filtro a media mobile.
La rilevazione del picco e dunque dell'abbassamento, invece, viene fatta eseguendo un'analisi dell'andamento del segnale filtrato: data una finestra di campioni, si divide quest'ultima in 10 sottofinestre e per ciascuna si calcola la media, dopo di che si verifica che i primi 5 valor medi abbiano un andamento crescente, mentre i secondi 5 un andamento decrescente; la verifica di questa condizione implica la presenza di un abbassamento. In generale questo approccio per la verifica dell'abbassamento, è sviluppato sulla falsa riga della media mobile, perchè stiamo descrivendo il segnale, mediante la media tra un sottinsime di campioni.

```
case 6:
    noisy_signal.addLast(event.values[0]);
    noisy_signal_timestamp.addLast(event.timestamp);
    counter++;


    if(counter == WINDOW_SIZE){
        for(Float f : noisy_signal)
            media += f;
        media /= WINDOW_SIZE;
    }

    if(counter > WINDOW_SIZE){

        Float removed = noisy_signal.removeFirst();
        noisy_signal_timestamp.removeFirst();
        media = media + ((event.values[0] - removed) / WINDOW_SIZE);

        movmean_signal.addLast(media);
        movmean_signal_timestamp.addLast(noisy_signal_timestamp.get
                                                    (WINDOW_SIZE/2));

        temp += media;
        if((counter \% FINDPEAK_WINDOW) == 0) {
            temp /= FINDPEAK_WINDOW;
            avg_array.addLast(temp);
            temp = 0.0f;
        }
        if(counter > MOVMEAN_WINDOW_SIZE + WINDOW_SIZE){
            movmean_signal.removeFirst();
            Long time = movmean_signal_timestamp.removeFirst();
            resize_windows(time);

            if((counter \% FINDPEAK_WINDOW) == 0)
                avg_array.removeFirst();

            boolean peak = findPeak(avg_array);

            if(peak){
                Log.i("PEAK", "TROOOOOOOOOOOOOOVATOOOOOOO");
            }
        }
    }
    break;
```

5.1.6 Memorizzazione dei segnali di sensori diversi dal barometro

Oltre a dover monitorare costantemente il segnale barometrico, è fondamentale tenere traccia anche dei segnali provenienti da accelerometro, giroscopio, magnetometro. In particolare viene tenuta in memoria una finestra per ognuno dei sensori citati in precedenza, che descriva la parte di segnale relativa agli istanti trovati nel paragrafo precedente (5.1.5), tali finestre devono essere aggiornate ogni qual volta arriva un nuovo samples, appartenente al segnale della pressione.
Il seguente codice realizza questa funzionalità.

```
private void resize_windows(Long time) {

        // ACCELEROMETR
        while(accelerometr_signal_timestamp.get(1) < time) {
                accelerometr_signal_timestamp.remove(0);
                accelerometr_signal.remove(0);
        }
        if((time - accelerometr_signal_timestamp.get(0)) >
    (time - accelerometr_signal_timestamp.get(1))) {
                accelerometr_signal_timestamp.remove(0);
                accelerometr_signal.remove(0);
        }

        // MAGNETOMETR
        while(magnetometr_signal_timestamp.get(1) < time) {
                magnetometr_signal_timestamp.remove(0);
                magnetometr_signal.remove(0);
        }
        if((time - magnetometr_signal_timestamp.get(0)) >
    (time - magnetometr_signal_timestamp.get(1))) {
                magnetometr_signal_timestamp.remove(0);
                magnetometr_signal.remove(0);
        }

        // GYROSCOPE
        while(gyroscope_signal_timestamp.get(1) < time) {
                gyroscope_signal_timestamp.remove(0);
                gyroscope_signal.remove(0);
        }
        if((time - gyroscope_signal_timestamp.get(0)) >
    (time - gyroscope_signal_timestamp.get(1))) {
                gyroscope_signal_timestamp.remove(0);
                gyroscope_signal.remove(0);
        }
    }
```

5.1.7 Testing seconda implementazione modulo detection

Durante la fase di testing della seconda implementazione si riscontra che il problema della scarsa potenza di calcolo dello smartwatch è risolto e che l'applicazione viene eseguita senza interruzioni o crash.
E' stato possibile, dunque, effettuare alcune simulazioni di abbassamento e raccolta di un oggetto da terra, per vedere se il sistema è in grado di comprendere se, chi indossa i dispositivi, ha effettivamente compiuto la manovra potenzialmente critica.
Dai risultati delle simulazioni, è emerso che il sistema realizzato è in grado di rilevare circa il 68% degli abbassamenti. Ponendo attenzione agli abbassamenti non rilevati e interrogandosi sul motivo di questa mancanza, si nota che le manovre, di cui l'applicazione non è riuscita a fare la detection, sono quelle in cui l'utilizzatore è rimasto abbassato per più tempo, facendo durare la manovra più del previsto: le manovre compiute in 2 sec / 2.5 sec. vengono indentificate, mentre gli abbassamenti compiuti in un tempo maggiore di 2.5 sec passano inosservati.
Per poter effettuare dei miglioramenti nei risultati ottenuti, sarebbe opportuno effettuare altre

simulazioni, coinvolgendo più persone con differenti caratteristiche fisiche (altezza, età, sesso, ecc.), il cui obbiettivo è trovare la dimensione della finestra, e dunque il numero di sample, che megliono descrivono la manovra e grazie al quale viene minimizzato il numero di task non rilevati, così da poter cambiare il parametro costante *static final int MOVMEAN_WINDOW_SIZE* ed ottenere un maggiore grado di precisione.

Capitolo 6

Conclusione

Una volta compiuta la detection del task in questione, l'applicazione wear invia un oggetto contenente le porzioni di segnale dei sensori, che descrivono l'abbassamento, alla parte di applicazione che viene seguita sullo smartphone.
Lo sviluppo del progetto termina a questo punto; la prosecuzione del lavoro dovrebbe prevedere la classificazione della manovra, avvalendosi di una rete neurale classificatrice, la quale prende in input le feature, dopo che sono state estratte dai segnali ricevuti, e classifica il task appena avvenuto in una delle seguenti classi: *corretta*, *scorretta* o *carico assente*, allertando l'utente, in caso sia stato rilevato un compimento scorretto e, dunque, dannoso.

La semplicità di questa soluzione è il suo punto di forza, infatti non è richiesta alcuna infrastruttura, che potrebbe innalzare notevolmente i costi di installazione e scoraggiarne la diffusione. È infatti sufficiente l'acquisto di dispositivi, che ormai fanno parte della vita quotidiana (smartphone e smartwatch), e l'installazione su di essi di applicativi che implementino gli algoritmi descritti. A fronte di un investimento di piccola entità (il prezzo medio di uno smartwatch che include tutta la sensoristica necessaria è di circa 300 Euro, mentre di uno smartphone circa 400), è possibile ridurre il numero di infortuni dovuti a questo compito, riducendo di conseguenza anche il numero di giorni di assenza e favorendo condizioni di lavoro più sicure per i dipendenti, che si rispecchia anche in una maggiore produttività.

Capitolo 7

Sitografia

- Documentazione Android, *https://developer.android.com/docs*
- Documentazione java Oracle, *https://docs.oracle.com/en/java/index.html*
- Documentazione MATLAB, *https://it.mathworks.com/*
- Wikipedia, *https://en.wikipedia.org/wiki/Main_Page*
- IEEE Xplore, *https://ieeexplore.ieee.org/Xplore/home.jsp*
- Science Direct, *https://www.sciencedirect.com/*