Politecnico di Torino

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Sviluppo e completamento dell'App IOS PulsEcg per la misurazione cuffless della pressione arteriosa tramite rete neurale

Relatori

Prof. Eros PASERO

Ing. Vincenzo RANDAZZO

Candidato

Terranova Davide

Febbraio 2022

Sommario

Oggigiorno la tecnologia e specialmente gli smartphone hanno pervaso la nostra quotidianità. In tutti gli ambiti vengono sviluppate app per agevolare i piccoli e grandi problemi presenti nella nostra vita quotidiana. Particolare attenzione stanno assumendo i dispositivi che controllano i parametri vitali del corpo umano, in special modo le funzioni cardiache. Una grande limitazione di questi dispositivi è l'impossibilità di acquisire la pressione arteriosa senza l'utilizzo dello sfigmomanometro. Il progetto descritto in questa tesi sviluppa un'applicazione da affiancare ad un dispositivo indossabile per acquisire i parametri vitali come elettrocardiogramma, pletismografia e pressione arteriosa.

In particolare, lo scopo della tesi verte sullo sviluppo dell'app PulsEcg, precedentemente creata dal dipartimento di elettronica e comunicazione (DET) del Politecnico di Torino, affinando le API e servizi già implementati e aggiungendone degli altri. Il dispositivo, chiamato anch'esso PulsEcg, ha le sembianze di un orologio in cui sono presenti due elettrodi, uno posto a contatto con il polso e l'altro posizionato nella parte superiore dove va collocato il dito dell'altra mano. Con questa configurazione si può misurare una differenza di potenziale che rappresenta il segnale dell'elettrocardiogramma. Inoltre, sul dispositivo, nella posizione adiacente all'elettrodo superiore, è presente un sensore per la misurazione della fotopletismografia. Questi segnali sono inviati all'app per mezzo di una connessione bluetooth 4.0 (Low Energy) e processati tramite appositi metodi sviluppati ad hoc. Dai segnali, opportunamente puliti dal rumore, si derivano i valori della frequenza cardiaca e della saturazione del sangue. In aggiunta, l'ECG e il PPG sono raffigurati su grafici millimetrati.

In conclusione, tramite uno studio accurato dell'andamento della curva, è possibile individuare una possibile fibrillazione atriale e, con l'utilizzo di una rete neurale, calcolare la pressione sistolica e diastolica. Il progetto è pensato per una diffusione più ampia possibile nella popolazione, per queste ragioni si è prestata particolare attenzione nel rendere il design dell'app il più possibile user-friendly.

La parte finale della tesi si è concentrata nella generazione e compilazione dei documenti utili per avviare una campagna di acquisizioni dati. L’obiettivo di quest’ultima è appurare la precisione e l'attendibilità dei segnali acquisiti su un campionario di volontari eterogeneo rispetto all'età, patologie e genere sessuale.

Ringraziamenti

Un grazie al Prof. Pasero, per la sua cordialità e disponibilità nel darmi supporto e consigli durante il lavoro di questa tesi.

Un grazie speciale all'Ing. Vincenzo Randazzo, sempre presente nei momenti difficili. Il suo aiuto è stato fondamentale per la buona riuscita del progetto.

Grazie ai miei amici. Con voi ho passato gli anni più belli della mia vita.

Il grazie più grande alla mia famiglia, non ce l'avrei fatta senza di voi. Il vostro sostegno mi ha dato la serenità per poter crescere come studente e come uomo. Sono stato fortunato ad avere una nonna, dei genitori e dei fratelli come voi. Grazie infinite.

Indice

Elenco delle tabelle

Elenco delle figure

Acronimi

ECG

Elettrocardiogramma

FA

Fibrillazione atriale

SpO_2

Saturazione del sangue

PPG

Fotopletismografia

BLE

Bluetooth Low Energy

ATT

Attribute Protocol

PDF

Portable Document Format

ANN

Artificial Neural Network

ML

Machine Learning

LSTM

Long Short-Term Memory

GATT

General Attribute Profile

VC

View Controller

UX

User Experience

SV

StackView

HB

Heart Beat

CER

Comitato Etico della Ricerca

GDPR

General Data Protection Regulation

UE

Unione Europea

DPO

Data Protection Officer

DMP

Data Management Plan

Capitolo 1

Introduzione

In Italia le malattie cardiovascolari sono la prima causa di morte con un'incidenza del 34,8% sul totale[1].
Negli ultimi anni per fronteggiare queste patologie, hanno assunto un ruolo centrale la prevenzione e la tempestività di intervento. Pertanto, oltre che adottare uno stile di vita sano e migliorare le proprie abitudini alimentari, si dovrebbe avere un costante monitoraggio dei propri parametri vitali.
Molto importante è il monitoraggio della pressione ai fini di prevenzione e di rivelazione delle malattie cardiovascolari. Lo strumento meno invasivo più diffuso per la misurazione della pressione avviene tramite lo sfigmomanometro. Ma alcuni pazienti, come neonati o anziani con patologie, non possono sopportare fisicamente la pressione esercitata dal manicotto. Inoltre, non è possibile avere un monitoraggio continuo per limiti intrinseci dell'oggetto di misurazione.
Per questi motivi è nata l'applicazione **PulsEcg** che permette un controllo periodico non invasivo del proprio corpo e alla portata di tutti.

1.1 Nozioni Di Medicina

Ai fini di comprendere pienamente le scelte tecniche adottate nell'applicazione daremo, in questo capitolo, una breve descrizione sui principali parametri misurati per prevenire e monitorare le malattie cardiovascolari. In particolare, descriveremo la misura dell'Elettrocardiogramma, della Pletismografia e della pressione arteriosa sistemica.

1.1.1 Elettrocardiogramma

L'elettrocardiogramma (ECG) consiste nella rappresentazione grafica del segnale elettrico prodotto dal cuore. La presenza dei campi elettrici di bassa intensità sono dovuti alla depolarizzazione e ripolarizzazione del cuore[2].

L'ECG ha 12 punti di osservazione, chiamate derivazioni, dove vengono posti degli elettrodi ai fini di registrare le variazioni dei campi elettrici tramite uno strumento chiamato elettrocardiografo, trasformando la differenza di potenziale in un movimento fisico[3]. Lo strumento andrà a tracciare l'onda complessa caratteristica PQRST (Fig. 1.1). Analizzando l'onda e misurando le differenze dall'andamento ideale, si potranno individuare delle anomalie/disfunzioni del cuore.

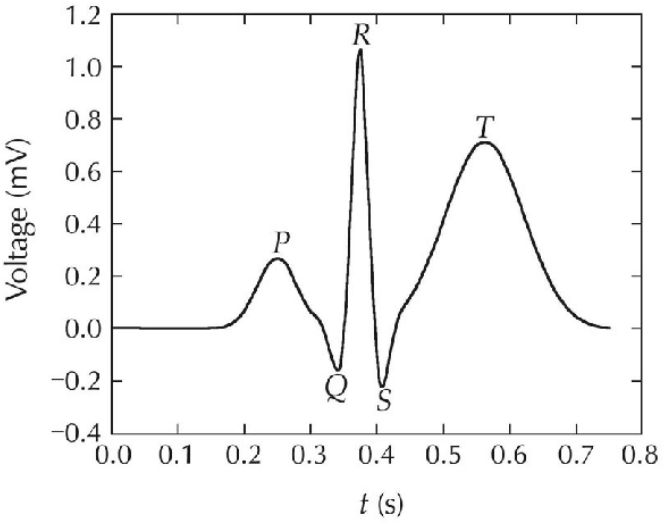


Figura 1.1: Onda PQRST

La derivazione più semplice è chiamata **derivazione bipolare**, consiste nel registrare la differenza di potenziale in soli due punti posti ai vertici del triangolo di **Einthoven** (Fig. 1.2). Naturalmente diminuendo i punti di osservazione diminuirà la precisione del segnale, quindi aumenterà la difficoltà nell'individuare possibili anomalie.

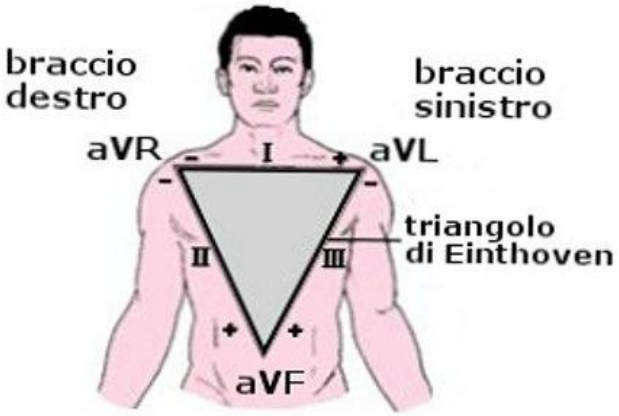


Figura 1.2: Triangolo di Einthoven

Il segnale registrato viene riportato in una carta millimetrata, dove in ascissa è riportato il tempo e nella ordinata l'ampiezza della curva (Fig. 1.3). Ogni quadratino ha un lato di 1mm, e i quadratini vengono raggruppati in quadrati 5x5mm. Il lato del quadratino parallelo alle ascisse corrisponde a 0,04 secondi e 5 quadrati grandi misurano 1 secondo. Il lato del quadratino parallelo alle ordinate misura 1mV[4].

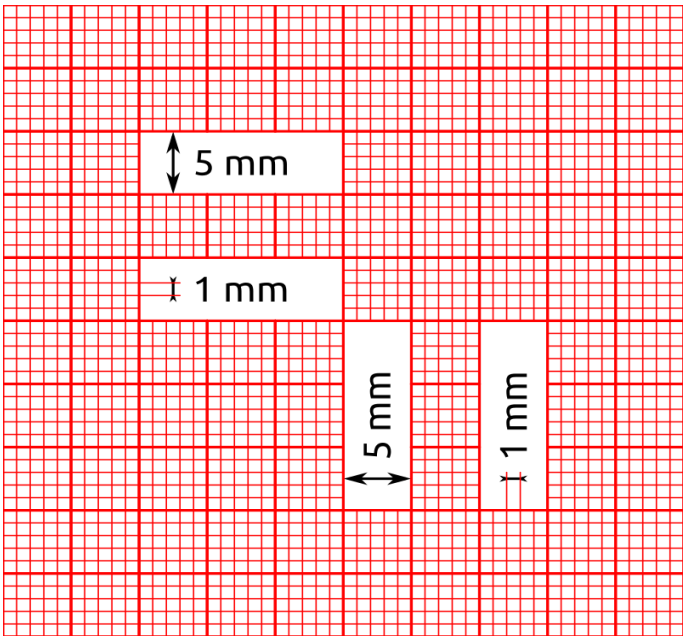


Figura 1.3: Misure della carta millimetrata

Fibrillazione atriale

La fibrillazione atriale (FA) è una patologia elettrica con due principali caratteristiche: l'attivazione elettrica rapida del tessuto atriale e l'aumento del rischio tromboembolico. Essa è generata da un insieme di disordini cardiaci come la cardiomiopatie, l'ipertensione arteriosa, malattie genetiche e tante altre[5].
Questa disfunzione non ha le componenti P ben distinte tra un battito e l'altro. Sono sostituite da onde F con morfologia e voltaggio variabile ma comunque inferiori a 200mv (Fig. 1.4).

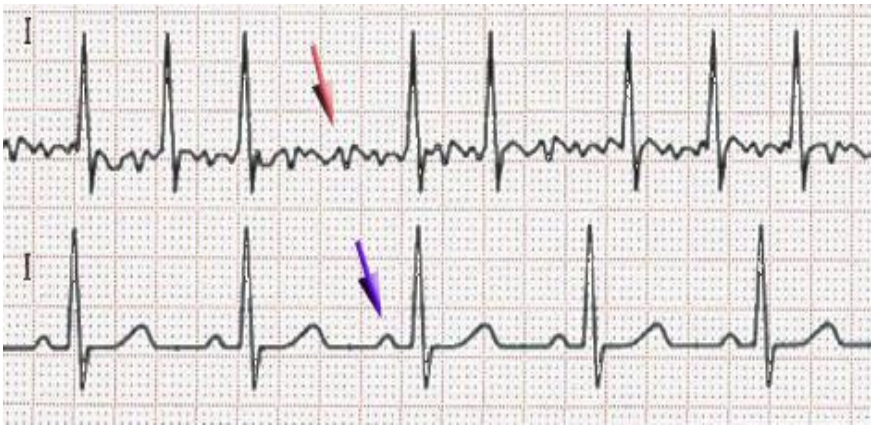


Figura 1.4: ECG con fibrillazione atriale (sopra) e normale (sotto)

La FA è una patologia molto difficile da individuare con l'1-2% della popolazione affetta da questa malattia. Se si guarda alle persone con più di 45 anni di età questa percentuale sale al 25%[6].
Pertanto è auspicabile un miglioramento delle strumentazione per la diagnostica della FA, punto centrale dell'applicazione PulsEcg.

1.1.2 Fotopletismografia e Saturazione

Il parametro della saturazione del sangue (SpO_2) è molto importante per monitorare lo stato di salute del cuore.
La **Pulsossimetria** è un metodo, non invasivo, per misurare i livelli di ossigeno nel sangue arterioso, specificatamente la quantità della proteina di emoglobina

responsabile del trasporto dell'ossigeno. Questa rilevazione avviene tramite l'erogazione di fasci di luce rivolti verso un fotodiodo (Fig. 1.5). Vengono utilizzati due tipi di fasci di luce: un fascio di luce rossa con lunghezza d'onda pari a 660nm e un fascio di luce infrarosso con lunghezza d'onda pari a 940nm. L'emoglobina carica di ossigeno assorbirà più luce infrarossa e lascerà passare più luce rossa[7].

La ricezione dei fasci di luce totali diminuisce durante la sistole per l'aumento del volume di sangue arterioso ed aumenta durante la diastole.

La **fotopletismografia** (**PPG**) consiste nel riportare in un pletismogramma la variazione del volume di sangue misurata in una delimitato tessuto microvascolare [8].

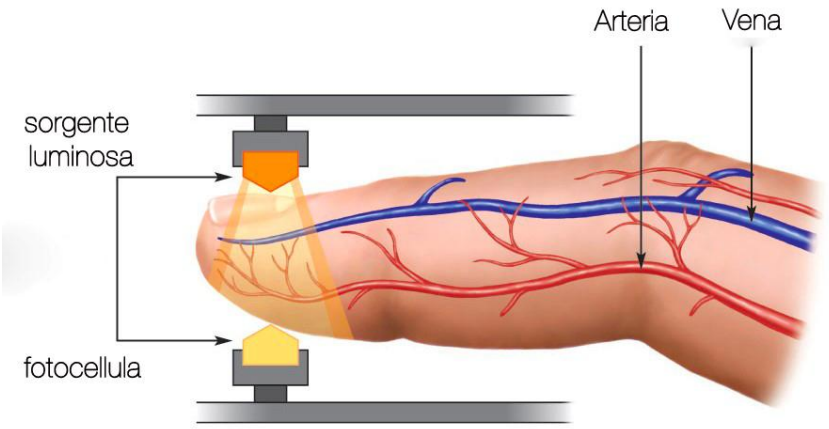


Figura 1.5: Principio di fotopletismografia

Si noti la figura di un'onda PPG (Fig. 1.6). Essa si suddivide in due parti: la parte ascendente rappresenta la variazione massima del volume del sangue nella fase sistolica, la parte "piatta" indica la chiusura della valvola aortica e infine nella parte discendente rappresenta la fase diastolica. La distanza tra due picchi identificata con t1 rappresenta un intero ciclo cardiaco[9].

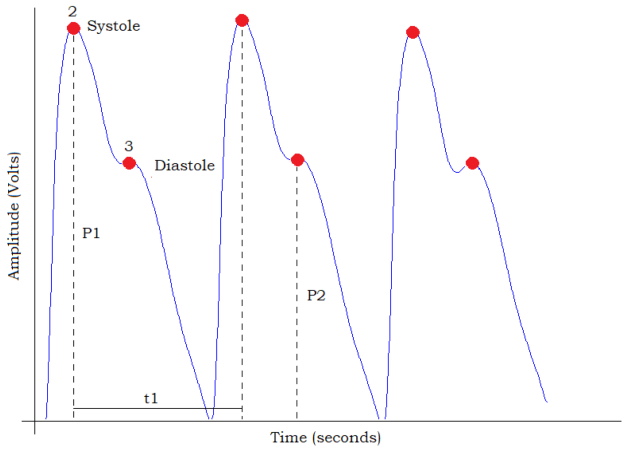


Figura 1.6: Onde del PPG

1.1.3 Pressione Arteriosa Sistemica

La **pressione arteriosa sistemica** è la pressione esercitata dal sangue sulle pareti arteriose e viene misurata in millimetri di mercurio (mmHg)[10].
La pressione ha due componenti:

- **Pressione sistolica** (o massima): è la pressione esercitata nella fase di contrazione del miocardio.
- **Pressione diastolica** (o minima): è la pressione esercitata nella fase del rilassamento del miocardio.

Lo strumento tradizionale per misurare la pressione è lo **sfigmomanometro,** composto da un manicotto gonfiabile e una pompetta per insufflare l'aria**.** Questo strumento ha diversi limiti di utilizzo a seconda del paziente. Infatti se il paziente è una persona con particolari patologie oppure si tratta di un neonato, questo tipo di misurazione non potrà avvenire[11].
Un altro metodo di misurazione consiste nell'inserire un catetere arterioso collegato a un trasduttore di pressione esterno. La misura è continua e molto precisa; tuttavia questo metodo invasivo, comporta numerosi rischi, come il verificarsi di embolia, di emorragia, di danno ai vasi o ai tessuti e di infezione[12].

Nella tabella 1.1 vengono riassunti i range di valori possibili della pressione:

classificazione	**sistolica**	**diastolica**
ipotensione grave	<80	<60
ipotensione moderata	80-99	60-64
normale bassa	100-109	65-74
ottimale	110-129	75-84
normale alta	130-139	85-89
Ipertensione di primo grado	140-159	90-99
Ipertensione di secondo grado	160-179	100-109
Ipertensione di terzo grado	>=180	>=110

Tabella 1.1: Valori pressione sistolica e diastolica

1.2 Dispositivi Wearable

Ai giorni d'oggi abbiamo diverse tecnologie per monitorare la salute del nostro cuore. In particolare si stanno diffondendo sempre più i dispositivi wearable (indossabili), ovvero dei dispositivi elettronici user friendly che si collegano al nostro smartphone tramite bluetooth e vengono indossati solitamente ai polsi[13]. Questi dispositivi sono in grado di monitorare continuamente lo stato di salute dell'utente offrendo dei servizi di documentazione e condivisione del problema riscontrato.

In questo modo l'utente potrà decidere se approfondire il suo stato di salute con esami più specifici.

1.2.1 Dispositivi in commercio

In questo paragrafo analizzeremo i principali dispositivi in commercio specializzati nel monitoraggio dell'attività del cuore.

Kardia

Kardia è un dispositivo sviluppato dall'azienda Alivecor composto da due elettrodi pensati per trasmettere i ritmi dell'ECG allo smartphone collegato tramite bluetooth. Le peculiarità distintive sono la presenza di un filtro avanzato che mostra la curva ECG regolare e il rilevamento della fibrillazione atriale. L'applicazione associata è anche in grado di esportare l'ECG in formato pdf[14].

Withings

Questa azienda ha sviluppato lo smartwatch **ECG Move**, un dispositivo in grado di acquisire l'ECG, e tramite l'analisi del segnale identifica la fibrillazione atriale. Utilizza la tecnologia Bluetooth Low Energy, effettua acquisizioni da 30 secondi e può esportare L'ECG in formato pdf(su quest'ultima cosa non sono sicuro)[15].

Apple Watch

Tramite il download dell'app **ECG,** l'Apple ha dato la possibilità di acquisire il segnale ECG tramite il suo apple watch. Il dispositivo è in grado di rilevare la fibrillazione atriale e di esportare l'ECG in formato pdf. [16]

1.2.2 Dispositivo PulsEcg

Come possiamo notare nel paragrafo 1.2.1 tutti i dispositivi in commercio acquisiscono il segnale ECG, individuano la fibrillazione atriale e riescono ad esportare il segnale in formato pdf.

Il dispositivo PulsEcg implementa tutte le funzionalità precedentemente descritte e aggiunge funzionalità come l'acquisizione del PPG, contatto diretto con il medico curante con invio dei segnali e della posizione del paziente, e funzionalità importante riesce ad acquisire la pressione arteriosa sistemica attraverso una rete neurale. Nel capitolo 1.1.3 abbiamo descritto tutte le problematiche intrinseche nell'utilizzo dello sfigmomanometro per la misurazione della pressione arteriosa, con PulsEcg il monitoraggio della pressione avviene in maniera rapida, comoda, accessibile a tutti e quasi costante.

Specifiche hardware

Lo stato del dispositivo PulsEcg è ancora in forma sperimentale come in figura 1.7. Questo prototipo è formato da due PCB(Process Control Block), uno dedicato all'ECG e alla batteria, e l'altro per il sensore infrarosso per l'acquisizione del PPG.

Figura 1.7: Prototipo PulsEcg

Il primo circuito è costituito da:

- modulo Bluetooth 4.0: usato per la comunicazione tra il dispositivo e l'applicazione.
- connettori: connettore USB, connettore elettrodi, connettore batteria.

- blocchi di alimentazione: Battery Charge e Battery Gauge servono per ricaricare la batteria e controllare lo stato di ricarica; Voltage Regulator e Analog Power Domain servono per dare una tensione di alimentazione stabile.
- filtri: utilizzati per preprocessare il segnale ECG.
- microcontrollore Texas Instruments CC2640R2F: componente che si interfaccia con tutti i moduli precedentemente presentati ed ha il compito di raccogliere i dati dell'ECG convertendoli da analogico a digitale (ADC) e tramite la comunicazione I2C acquisisce i valori di batterie e del segnale pletismografico.

Il secondo circuito è costituito da:

- connettore piatto: utilizzato per connettere il circuito stampato al circuito principale portando le linee di alimentazione e le linee I2C.
- sensore Maxim Integrated MAXM86161: dispositivo per acquisire i dati per la fotopletismografia, emette la luce infrarossa e rossa.

Il progetto di sviluppo hardware è in atto con lavori di tesi paralleli puntando al miglioramento dei filtri di acquisizione e maggior efficienza del firmware scritto in linguaggio C.

Capitolo 2

AMBIENTE IOS

2.1 Linguaggio di programmazione swift

Swift è un linguaggio di programmazione open source ed è stato creato da Apple per sostituire l'ormai obsoleto Objective-C per facilitare la creazione di app per IOS, Mac, Apple TV e Apple Watch. Una peculiarità molto importante è quella di essere completamente compatibile con il vecchio linguaggio Objective-C con il vantaggio di recuperare le risorse create nel passato[17].

2.2 Sviluppare un'app con swift

In questa sezione affronteremo i concetti e componenti base per lo sviluppo di un'app IOS con il linguaggio Swift. I componenti che vedremo di seguito sono lo UIViewController, il Segue e l'InterfaceBuilder.
Nello specifico, il ruolo di un ViewController consiste nel gestire la creazione e l'interazione degli elementi contenuti nella vista associata. Il Segue ha la funzione di passare da una vista all'altra e tenere traccia della successione nelle varie viste. Infine l'InterfaceBuilder è un tool di supporto per agevolare il programmatore a creare l'interfaccia grafica nello StoryBoard, quindi collegare le funzionalità dei vari elementi con le funzioni contenute nel ViewController associato.

2.2.1 UIViewController

La classe UIViewController definisce il comportamento comune di tutti i ViewController. Il ViewController viene associato ad una interfaccia grafica in cui gestisce la sua rappresentazione, l'aggiornamento delle viste, le risposte all'interazione con l'utente e il ridimensionamento delle viste. Quindi quando l'utente naviga attraverso le varie schermate dell'app passa da un ViewController all'altro.

Ogni qual volta il ViewController cambia il suo stato il sistema IOS chiamerà dei metodi in maniera automatica. Questo agevola il programmatore a gestire i cambiamenti dei dati e delle subView in maniera efficiente ed efficace. Il programmatore ha il compito di fare l'override dei suddetti metodi per personalizzare il comportamento di ogni ViewController [18].
I possibili stati di un ViewController sono raffigurati nella figura 2.1:

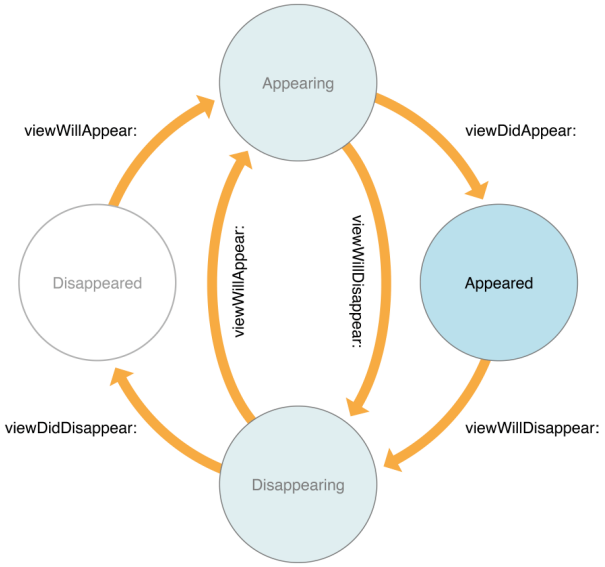


Figura 2.1: Stati del ciclo di vita di un VC

Di seguito l'elenco dei metodi:

- viewDidLoad(): Chiamato subito dopo l'inizializzazione del ViewController. Ha il compito di eseguire una inizializzazione aggiuntiva sull'interfaccia e verrà eseguito solo una volta.

- viewWillAppear(): Chiamato appena prima che la view sia stata aggiunta alla gerarchia di visualizzazione dell'app. Utilizzata per modificare la view a seconda di certe condizioni. Ad Esempio l'orientazione dello schermo.

- viewDidAppear(): Chiamato appena dopo che la view sia stata aggiunta alla gerarchia di visualizzazione dell'app.

2.2.2 Segue

Il Segue definisce il flusso delle interfacce nell'app, in sostanza permette all'app di passare da un ViewController all'altro memorizzando il ViewController di partenza, per un possibile ritorno, e passando eventuali dati utili nel ViewController di destinazione. Il Segue può essere attivato da un pulsante, da una riga di una tabella oppure dal riconoscimento di una gesture[19].

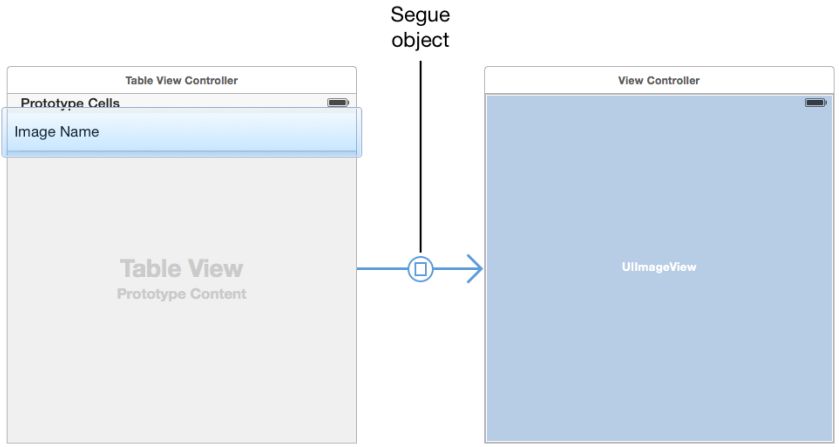


Figura 2.2: Rappresentazione di un segue tra due VC

2.2.3 Interface Builder

L'editor Interface Builder fornito da XCode (l'ambiente di sviluppo integrato fornito da Apple) è uno strumento interattivo utile per creare interfacce in maniera dinamica e senza la necessità di scrivere alcuna linea di codice. Infatti basta trascinare pulsanti, caselle di testo e tanti altri elementi presenti nell'aria di progettazione per creare un'interfaccia interattiva. Le interfacce create verranno salvate in un file chiamato Storyboard utile per avere una panoramica delle interfacce e sui collegamenti tra esse[20].

Come abbiamo visto nel paragrafo 2.2.1 ogni interfaccia, con i suoi elementi, è associata ad una ViewController che ne gestisce le sue funzionalità.

Gli elementi per poterli personalizzare oppure per far si che scatenino degli eventi al tocco, bisogna collegarli al codice tramite:

- Action: eseguono dei metodi associati a degli elementi interattivi come i bottoni.
- Outlet: è un riferimento dell'oggetto e serve principalmente per gestire il contenuto oppure l'aspetto di un elemento.

2.3 Libreria CoreBluetooth

Il framework CoreBluetooth gestisce tutte le API della connessione wireless Bluetooth Low Energy, tecnologia utilizzata dal nostro dispositivo PulsEcg.

2.3.1 Bluetooth Low Energy

La tecnologia Bluetooth Low Energy (BLE) è stata sviluppata come miglioramento della versione Bluetooth 4.0. Le peculiarità principali del BLE sono il basso consumo di energia e una ridotta banda di trasmissione, infatti è stata pensata per collegare tra di loro dispositivi come sensori, attuatori oppure tutti quegli oggetti che necessitano di un consumo molto basso e hanno la necessità di trasferire poche quantità di dati[21].

2.3.2 Servizio GATT

I dispositivi scambiano dati attraverso il protocollo GATT basato su servizi, caratteristiche e descrittori.
Ogni servizio, per esempio il servizio per ricevere le informazioni del dispositivo, contiene diverse caratteristiche con i rispettivi descrittori, attributi (valori) e permessi. Gli attributi delle caratteristiche possono essere scritti oppure letti. Inoltre una caratteristica può inviare delle notifiche asincrone al dispositivo centrale. Ogni caratteristica e servizio sono identificati da un codice univoco UUID (Universally Unique Identifier)[22].

Fig. : Architettura GATT

Di seguito la descrizione del protocollo:

1. Il dispositivo invia una notifica per comunicare la sua disponibilità a connettersi
2. Il dispositivo centrale esegue uno scan per identificare le periferiche disponibili.
3. Una volta scelta la periferica il dispositivo centrale interrompe lo scanning e avvia la procedura di connessione.
4. Il dispositivo centrale controlla tutti i servizi e caratteristiche della periferica.
5. I due dispositivi scambiano dati con il servizio scelto.

Tutte le procedure di lettura dei UUID dei servizi e delle caratteristiche con la possibile scrittura o lettura degli attributi vengono gestiti dal Attribute Protocol (ATT).
Questo protocollo segue una logica di client/server dove il server, in questo caso il dispositivo PulsEcg, mostra una serie di servizi e attributi che il client può leggere e scrivere[23].

2.3.3 API CoreBluetooth

Il framework CoreBluetooth raggruppa tutte classi utili per connettersi e scambiare dati con i dispositivi BLE[24].
Le principali classi utilizzate nell'applicazione sono:

- CBCentral: fornisce i metodi per lo scan e connessione alle periferiche.
- CBPeripheral: fornisce i metodi per scoprire, esplorare e interagire con i servizi disponibili.
- CBService: rappresenta i servizi di una periferica.
- CBCharacteristic: rappresentano le caratteristiche di un servizio con la loro descrizione e valore.
- CBUUID: contiene i metodi per gestire gli UUID dei servizi e delle caratteristiche

2.4 Libreria Charts

Nell’applicazione PulsEcg è stata data grande attenzione nella rappresentazione dei segnali di ECG e PPG. Per questa parte del progetto si è scelta la libreria Charts, un’ampia libreria scritta interamente utilizzando il linguaggio di programmazione Swift per diretta traduzione della corrispettiva libreria Java MPAndroid Charts. Questa libreria contiene 8 tipologie di grafici diversi. Per la rappresentazione dei segnali ECG e PPG è stata scelta la tipologia LineChart.

La difficoltà maggiore è stata la modifica del codice sorgente per adattare la libreria alle esigenze di rappresentazione con carta millimetrata: quadratini precisi e scala di valori fissa sulle ordinate per uno studio puntuale della curva di ECG, spessori diversi per agevolare la lettura dei valori e unità di misura.

2.5 NotificationCenter

Quando si presenta la necessità di aggiornare dei dati in un ViewController prodotti da altre parti dell'applicazione, come ad esempio la classe bluetooth che lavora in background, si utilizza la classe **NotificationCenter.**
Con questa classe si possono trasferire dati da una parte all'altra della applicazione con il pattern **Observer.** Il pattern **Observer** è un **meccanismo asincrono di notifica** ed è composta da 3 componenti:

- Gli ascoltatori che sono in attesa della notifica
- Un mittente che invia la notifica
- Un notification center che gestisce e organizza lo scambio dei dati

Nel NotificationCenter sono registrate le notifiche identificate da una stringa univoca. I ViewController, utilizzando il metodo *addObserver*, si registrano ad un determinato centro di notifiche specificando quali notifiche desidera ricevere. Inoltre la classe deve specificare un **handler della notifica**, una funzione che sarà attivata dall'arrivo di una determinata notifica. Infine una classe mediante l'utilizzo della funzione *post* potrà inviare una notifica[25].

2.6 Funzioni per Import-Export

In questa sezione presenteremo le classi utilizzate per importare-esportare file JSON ed esportare file PDF delle acquisizioni. Queste operazioni possono avvenire attraverso due canali: con i maggiori framework di messaggistica o tramite email. Inoltre verranno presentate le funzioni per salvare od importare le acquisizioni nell'app.

Le classi utilizzate per implementare queste funzioni sono la **VC Activity**, **VC DocumentPicker** e il framework **MessageUI.**

2.6.1 Activity View Controller e PDFKit

La classe **ActivityViewController** fornisce servizi per condividere dei contenuti sui social network, inviare elementi con email o con i framework di messaggistica e altro ancora. L'app è responsabile della configurazione, presentazione e della chiusura del view controller.
Nella configurazione del VC si specifica il tipo e il contenuto del file e l'app dovrà gestire la presentazione tramite pop-up[26].
Per la gestione della creazione dei PDF è stata utilizzata la classe **PDFKit.** Questa classe considera il file PDF come un Rectangle definendo un punto d'origine, un'altezza e una larghezza. Il Rectangle principale può contenere altri rectangle (le immagini e dati dell'app), in questo modo si ha una maggiore flessibilità nella composizione del PDF.

2.6.2 Document Picker View Controller

La classe **Document Picker View Controller** è stata utilizzata per poter gestire l'importazione, l'esportazione, l'apertura e la chiusura dei file. Particolare attenzione ha l'importazione di un file che può avvenire in due modi:

- Importazione del file creando una copia. Il file originale a fronte di modifiche da parte dell'app non verrà modificato.
- Importazione del file originale. Tutte le modifiche saranno apportate nel file originale.

Questa classe deve essere istanziata e presentata dal programmatore, specificando il tipo di file che potranno essere importabili (in questo caso JSON).

2.6.3 MessageUi

Il framework IOS **MessageUI** permettere di generare in maniera automatica dei

messaggi di testo o e-mail. Questo VC mette a disposizione delle interfacce dove l'utente può compilare dei documenti senza uscire dall'app; inoltre i campi possono essere precompilati risparmiando tempo e possibilità di errori da parte dell'utente. Il programmatore dovrà solamente presentare il VC e un delegate a seconda dell'azione dell'utente gestirà la sua chiusura[27].

2.7 Libreria CoreLocation

La libreria utilizzata per ricavare la posizione di un dispositivo IOS è il framework **Core Location.** Il framework utilizza varie tecniche per ricavare la posizione: A-GPS, triangolazione di torri cellulari, WI-FI, ecc. Di seguito i tre servizi offerti da Core Location:

1. *Standard Location Service*: il servizio standard offerto dai dispositivi IOS. Esso permette una localizzazione molto precisa ed è utile utilizzarlo quando non si ha il bisogno di avere un costante aggiornamento della posizione.
2. *Significant Change Location Service*: segnala all'utente la posizione corrente e lo informa dei cambiamenti della posizione. Questo servizio è utile nel caso servissero continui aggiornamenti della posizione.
3. *Region Monitoring Service*: tiene traccia solo dei confini tra regioni. Informa l'utente se vengono oltrepassati i suddetti confini[28].

Per la traduzione da coordinate a un indirizzo contenente la via, la città e lo stato, si è utilizzata la classe user-friendly **CLPlacemark**.

Capitolo 3

Rete Neurale

3.1 Nozioni sulle Reti Neurali

Le reti neurali artificiali (ANN, *Artificial Neural Network*) sono un modello di apprendimento che emula il funzionamento dei neuroni biologici. I modelli matematici utilizzati sono troppo semplici per descrivere fedelmente il comportamento realistico dei neuroni, ma stanno diventando sempre più utili nel risolvere problemi in campo ingegneristico [29].
Facendo un parallelismo con il cervello, tale modello è formato da nodi (neuroni), che sono le nostre unità computazionali, collegati tra loro tramite sinapsi. Nello specifico, la rete neurale artificiale la possiamo suddividere in *layer* (strati) di nodi, ciascuno dei quali è collegato ai nodi del layer successivo. Possiamo identificare tre gerarchie di layer:

- Input layer: ovvero lo strato di nodi che riceve i dati di input
- Uno o più Hidden layer: lo strato di nodi dove avviene la computazione vera e propria
- Output layer: lo strato finale che contiene i risultati

Ogni nodo riceve dei segnali di input provenienti dal layer precedente. Ogni segnale verrà moltiplicato per un fattore (peso), che corrisponde alla sua importanza nella rete. Il valore risultante verrà elaborato da una funzione di attivazione interna ad ogni nodo. Se il segnale supera la soglia di attivazione, il nodo sollecitato invierà un segnale ai nodi del layer successivo, altrimenti il valore inviato sarà pari a zero. Infine il segnale convoglierà maggiormente in un nodo dello strato di output che identifica il risultato.

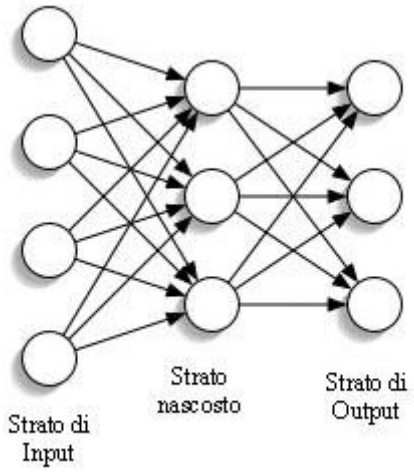


Figura 3.1 : Struttura rete neurale semplice

Quest'ultimo prende il nome di *predizione*; più essa è vicina alla realtà, più si dice che la rete è *accurata*. Con la fase di *training* (addestramento) si cambiano i pesi della rete fin quando la "precisione" si considera sufficiente. Per calcolare l'accuratezza della rete si utilizza una funzione di *Loss*, che permette di modificare i pesi nella maniera corretta. Per fase di *training* la rete ha bisogno di due set di dati: un *training set* e un *validation set*, il primo per addestrare la rete e il secondo per calcolare la precisione. L'addestramento viene ripetuto più volte ed ogni step è chiamato *epoca*.

3.2 Reti Neurali Convolutive

Le **reti neurali convolutive** sono delle particolari reti impiegate nel riconoscimento di immagini e nella bioinformatica. Ogni nodo ha un **campo recettivo** diverso dagli altri, ovvero essi analizzano diverse features. Gli output dei neuroni vengono combinati, in modo da passare all'analisi di caratteristiche più complesse. Gli input vengono gestiti in un formato detto **tensore** (matrice a N dimensioni). I dati vengono scansionati da **filtri scorrevoli**, in modo da estrapolare solo alcune features. Man mano che si avanza nella rete, le caratteristiche vengono combinate per eseguire un'analisi sempre più astratta[30].

Nell'analisi dei dati possiamo identificare due fasi:

- **Livello Convolutivo:** i filtri scorrevoli identificano le features. La funzione di attivazione modifica l'output.

- **Polling:** viene ridotta la grandezza del problema. Ad ogni livello convolutivo aumenta la dimensione che viene ridotta dal polling generando una versione differente da come era inizialmente.

L'n-esimo layer convoluzionale è formato da un livello convolutivo e da uno strato di polling. Con una complessità dei dati maggiori, la rete avrà un maggior numero di strati per analizzare dettagli di ancor più basso livello.
Nel layer finale la rete diventa una normale feed forward con in uscita una softmax che normalizzi i risultati.

3.3 Rete Neurale del PulsEcg

In questo capitolo descriveremo la rete neurale artificiale utilizzata per derivare la pressione arteriosa dai segnali di ECG e PPG. Nello specifico, nella prima parte parleremo dello stato dell'arte della rete neurale e in seguito di tutte le modifiche apportate per renderla compatibile con l'app PulsEcg.

3.3.1 Stato dell'arte della rete neurale

Il codice della rete neurale è stato scritto utilizzando un IDE online chiamato *Google Colab*, uno strumento molto potente in cui è possibile utilizzare delle risorse remote, come GPU e TPU, in maniera gratuita.
La rete è realizzata in Python utilizzando *TensorFlow*, una libreria open source end-to-end, utile nella realizzazione di algoritmi nel campo del *machine learning* (ML, apprendimento automatico).
In Tensorflow, gli input e gli output sono costituiti dai tensori: sono array multidimensionali con un tipo unificato (chiamato dtype). Tutti i tensori sono immutabili come i numeri e le stringhe Python: non si può mai aggiornare il contenuto di un tensore, solo crearne uno nuovo[31].

L'architettura della rete neurale è una versione modificata della ResNet, ovvero una rete pensata per la classificazione. Essa è composta da layer convoluzionali e layer fully connected (cioè tutti i nodi di due layer adiacenti sono collegati tra loro), a cui vengono aggiunti ulteriori tre layer *Long Short-Term Memory* (LSTM), adatte a classificare, elaborare e fare previsioni su dati basati su serie temporali[32].

Il codice Python riportato in seguito è la realizzazione della rete neurale appena descritta. I layer utilizzati sono forniti da Keras, una libreria open source per il machine learning che fornisce API semplici ed efficaci[33].

```
1   def resnet_model(input_shape, num_classes=10):
2
3   filters = [64, 128, 256, 512]
4   kernels = [3 , 3 , 3 , 3]
5   strides = [1, 2, 2, 2]
6
7   tf1 = tf.compat.v1
8
9   # Inizio della rete
10  sign = tf.keras.layers.Input(shape=input_shape, dtype='float32')
11  x = tf.keras.layers.Conv1D(64, 3, strides=1, padding='same')(sign)
12  #tf.keras.layers.Conv1D(filters=32, kernel_size=10,
   activation='relu', 13 padding='same')
14
15
16  for i in range (len (kernels)):
17      x =_resnet_block (
18         x,
19         filters[i],
20         kernels[i],
21         strides[i])
22
23
24  x = tf.keras.layers.BatchNormalization()(x)
25  x = tf.keras.layers.Activation('relu')(x)
26  x = tf.keras.layers.AveragePooling1D(4,1)(x)
27
28  x = tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(128,
   return_sequences=True))(x)
29  x = tf.keras.layers.LSTM(128 , return_sequences=True)(x)
30  x = tf.keras.layers.LSTM(128)(x)
31  x = tf.keras.layers.Dense(num_classes)(x)
32
33  model = tf.keras.Model(inputs=sign, outputs=x, name='resnet18')
```

```
34
35  return model
```

3.3.2 Aggiornamento versione TensorFlow

Inizialmente la rete neurale è stata sviluppata nella versione di TensorFlow 1.15. Per poter utilizzare la rete su uno smartphone, quindi con una potenza di calcolo molto minore rispetto a un computer, bisogna convertire il modello, con i relativi pesi, nella versione di TensorFlow Lite (tflite). Il problema nasce dal fatto che la versione tflite non supporta i layer convoluzionali dati dal TensowFlow 1.x, generando un'eccezione nella fase di istanziamento del modello.
Per queste ragioni è nata l'esigenza di migrare il codice della rete neurale nella versione di TensorFlow 2.x, che fornisce supporto nella conversione in TensorFlow Lite contenenti layer convoluzionali.
La migrazione nella nuova versione non ha portato a sostanziali cambiamenti; l'unico problema è stato riscontrato nei nuovi layer LSTM. I vecchi pesi, calcolati con l'addestramento precedente alla nuova versione, non erano più compatibili e non è stato possibile riutilizzarli. Per questo motivo è stato necessario eseguire l'addestramento nella nuova versione.
Di seguito il codice Python dell'addestramento:

```
1   loss_object = tf.keras.losses.Huber()
2   optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate,
   name='adam ' )
3
4   train_loss = tf.keras.metrics.Mean(name='train_loss')
5   train_accuracy = tf.keras.metrics.MeanAbsoluteError(name='
   train_accuracy ')
6
7   test_loss = tf.keras.metrics.Mean(name='test_loss')
8   test_accuracy = tf.keras.metrics.MeanAbsoluteError (name='
   test_accuracy')
9
10
11   def train_step(x,y):
12       with tf.GradientTape() as tape:
13       predictions = model(x, training=True)
```

```
14        loss = loss_object(y_true=y ,y_pred=predictions)
15        gradients = tape.gradient(loss, model.trainable_variables)
16        optimizer.apply_gradients(grads_and_vars=zip(gradients,
          model.trainable_variables))
17        train_loss(loss)
18        train_accuracy(y, predictions)
19
20    def test_step (x, y):
21        predictions = model (x, training=False)
22        v_loss = loss_object(y, predictions)
23
24        test_loss(v_loss)
25        test_accuracy(y, predictions)
26
27
28    #start training
29    for epoch in range(epochs):
30        train_loss.reset_states()
31        train_accuracy.reset_states()
32        test_loss.reset_states()
33        test_accuracy.reset_states()
34
35    for x, y in train_ds:
36        train_step(x, y)
37
38    for test_x, test_y in test_ds:
39        test_step(test_x , test_y)
```

Le metriche e l'optimizer coincidono con la prima versione della rete neurale, naturalmente aggiornate con la nuova versione di TensorFlow. Il valore delle epoche è stato fissato a 100, il numero tipicamente scelto. Per settare il valore ottimale di epoche bisogna esaminare l'andamento del parametro della loss ad ogni ciclo: meno grande è il valore di loss più la rete è accurata. Nella rete in esame, siamo partiti da un valore di loss pari a 66.45325469970703, per poi diminuire fino al valore di 0.5899175405502319. Dato che all'epoca 98-99 il valore si attesta intorno al 0.60, abbiamo concluso che con 100 epoche si raggiunge un livello di accuratezza soddisfacente.

3.3.3 Rete con TensorFlow Lite

Il ToolKit TensorFlow Lite è pensato per adattare le reti neurali, sviluppate con TensorFlow 2.x, destinate ad essere utilizzate su dispositivi mobile o IoT aventi una bassa capacità computazionale, pratica sempre più utilizzata chiamata **EdgeAI**. Il principale vantaggio è quello di elaborare i dati in locale, senza l'utilizzo di server esterni. Così facendo si eliminano problemi di banda, sicurezza e latenza[34].

Come abbiamo accennato nel capitolo 3.3.2, la rete neurale è stata convertita in un modello tflite.

Di seguito viene riportato il codice utilizzato per realizzare il modello tflite:

```
1   converter = tf.lite.TFLiteConverter.from_saved_model("/
   content/drive/My Drive/Checkpoint")#path to the SavedModel
   directory
2   converter.target_spec.supported_ops = [tf.lite.OpsSet.
   TFLITE_BUILTINS,tf.lite.OpsSet.SELECT_TF_OPS]
3
4   converter.allow_custom_ops=True
5   tflite_model = converter.conve rt()
```

Come possiamo vedere la libreria TensorFlow agevola l'operazione di conversione del modello; inoltre, è stato necessario specificare alcuni parametri al converter, perché alcune operazioni, presenti nel codice della versione antecedente a questa, non sono supportate di default da tflite.

Il modello TensorFlow Lite accetta in input tensori aventi dimensioni [1 625 2]; con 625 viene indicato il numero di campioni e con 2 il numero dei segnali (ECG e PPG). Con una frequenza di campionamento di 125 Hz, il tensore ha campioni corrispondenti a 5 secondi di acquisizione. Per testare 10 secondi di acquisizione, ad esempio, occorre spezzare a metà il segnale e passare due volte gli input alla rete. Di seguito è riportato il codice per testare 5 secondi di acquisizione:

```
1  interpreter = tf.lite.Interpreter(model_content=tflite_model)
2  interpreter.allocate_tensors()
3  # Get input and output tensors.
4  input_details = interpreter.get_input_details()
5  output_details = interpreter.get_output_details()
6
7  input_data=np.array(test, dtype=np.float32)
8  interpreter.set_tensor(input_details[0]['index'], input_data)
9
10 interpreter.invoke()
11 output_data =interpreter.get_tensor(output_details[0]['index'])
12 print('\n\n\n\n\n')
13 print(output_data)
```

Inizialmente viene creato un oggetto *interpreter* a cui viene passato il modello in formato TensorFlow Lite. Di seguito, tramite *np.array(test, dtype=np.float32)*, viene creato l'input dei dati da passare al tensore, settati con la funzione *set_tensor(input_details[0]['index'], input_data)*. Poi utilizzando il metodo *invoke()* si esegue la rete neurale. Infine, con *get_tensor(output_details[0]['index'])* si ottiene il tensore di output che avrà dimensione [1,2].

3.3.4 Training

Il modello in TensorFlow Lite eredita i pesi calcolati con la versione precedente. Una funzionalità importante delle reti neurali è quella di poter aggiornare i suddetti pesi con una fase di training periodica; così facendo è possibile migliorare i risultati di predizione della pressione. Questa operazione è molto onerosa, in termini di potenza computazionale, per questo motivo è preferibile eseguirla in una fase di riposo del dispositivo, in modalità background[35].

Capitolo 4

PulsEcgIOS

Nei capitoli precedenti abbiamo fornito delle nozioni sia teoriche che tecniche impiegate per la progettazione dell'app. In questo capitolo affronteremo, nel dettaglio, l'implementazione e le scelte tecniche adottate per la realizzazione dell'applicazione.

La prima versione dell'app PulsEcg, sviluppata nel lavoro di tesi antecedente a questo, si è concentrata sullo suo sviluppo ex novo dell'applicazione. I principali componenti sviluppati sono state le interfacce con un design user friendly, la classe designata per la gestione del protocollo bluetooth 4.0, le principali funzioni per l'elaborazione dei segnali ECG e PPG, rappresentazione su grafici dei segnali e la creazione di un PDF contenente i parametri vitali acquisiti. A seguito di un'attenta analisi del progetto precedente, si sono individuate delle criticità riguardanti i metodi già implementati e diversi limiti superati aggiungendo altre funzionalità.

Nelle sezioni successive descriveremo nei dettagli i problemi individuati e le soluzioni adottate per risolverli.

4.1 Applicazione IOS

Nel capitolo corrente descriveremo l'organizzazione generale dell'app PulsEcg. La sua navigazione, come le GUI di ogni schermata, è stata realizzata tramite l'utilizzo dell'Interface Builder di XCode ed è contenuta nel file **Storyboard**(fig. 4.1).

L'applicazione, concettualmente, è possibile suddividerla in due macro aree: il **front-end** e il **back-end**. Con front-end denotiamo la parte visibile all'utente (interfaccia utente), con cui egli interagisce e riceve feedback riguardante il suo stato nell'applicazione[36].

Il back-end, invece, indichiamo tutta la parte nascosta all'utente, ovvero in cui avviene l'elaborazione, l'organizzazione, la memorizzazione e l'acquisizione dei

dati, e dove vengono implementati i protocolli di collegamento bluetooth con il dispositivo hardware PulsEcg.
Il front-end è formato da 7 View Controllers:

1. HomePage VC: è l'interfaccia principale. Da qui è possibile raggiungere il GeneralOption VC, il ConnectBluetoothDevice VC, l' AcquireECG VC e Achivio VC;
2. GeneralOption VC: l'utente in questo VC può configurare le proprie preferenze;
3. ConnectBluetoothDevice VC: ovvero il VC predisposto per la gestione dei dispositivi bluetooth;
4. AcquireECG VC: con questo VC è possibile avviare l'acquisizione dei parametri vitali;
5. Achivio VC: è il VC che gestisce e organizza tutte le acquisizioni passate;
6. GeneralShowECG VC: si tratta del VC adibito per raffigurare in maniera riepilogativa i segnali ECG e PPG appena acquisiti, i valori di SpO_2, i battiti cardiaci e i valori di pressione arteriosa. Esso viene lanciato dall'Acquire VC automaticamente una volta finita l'acquisizione.
7. ShowECG VC: è il VC per mostrare dettagliatamente su grafici millimetrati i segnali di ECG e PPG. È possibile attivarlo dal GeneralShowECG VC.

La parte di back-end è formata dalle seguenti classi:

1. BleManager: la classe utile per gestire le connessioni bluetooth e la ricezione dei dati da parte del dispositivo hardware PulsEcg.
2. SignalProcessing: in questa classe sono contenute tutte le funzioni per l'elaborazione dei dati.
3. NeuralNetworkManager: la classe contenente la rete neurale per la derivazione della pressione arteriosa dai segnali ECG e PPG.

Nei paragrafi successivi affronteremo la parte tecnica del front-end e back-end in maniera dettagliata, con le relative problematiche riscontrate e soluzioni adottate.

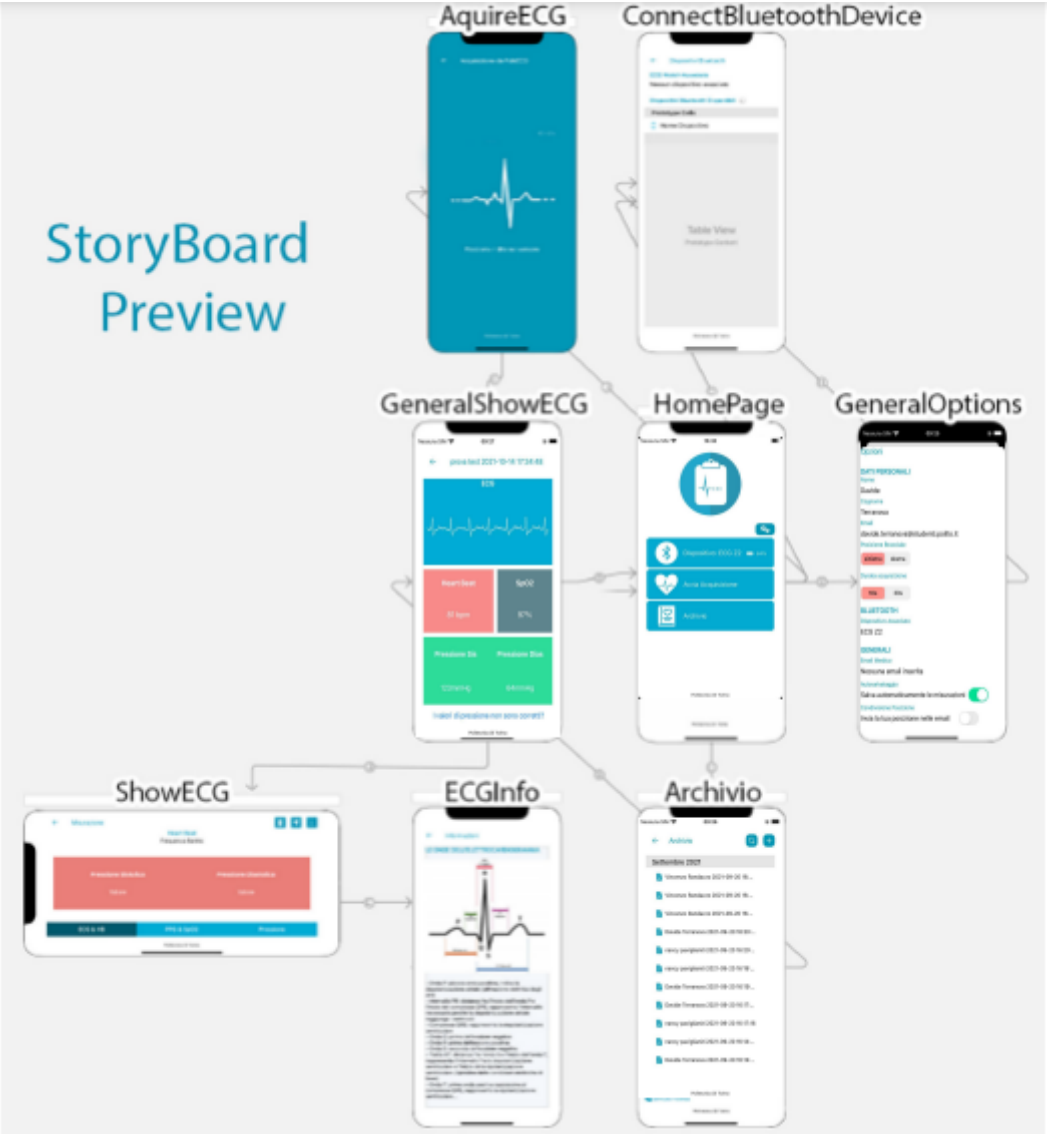


Figura 4.1: Raffigurazione dello Storyboard, con i View Controller e i collegamenti tra di essi

4.1.1 Permessi

Al primo avvio, l'app richiede una serie di permessi per il suo corretto funzionamento. Nella sua prima versione erano richiesti il permesso della geolocalizzazione e il permesso dell'attivazione del bluetooth.

In questa nuova versione è stato aggiunto il consenso della privacy (Fig. 4.2). La sua finalità è di informare l'utente quali sono gli scopi del progetto e come saranno gestiti i dati da lui forniti.

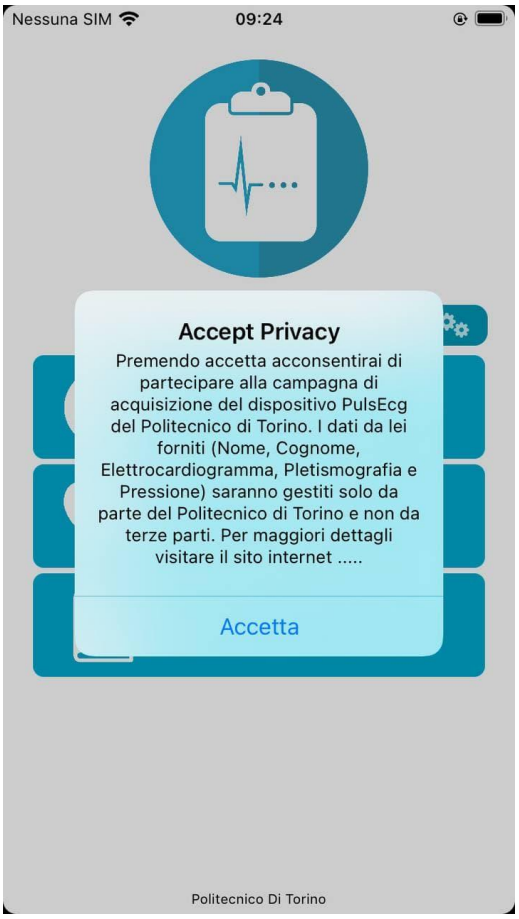


Figura 4.2: Pop-up privacy

Quando si avvia per la prima volta l'app, compare un pop-up con il seguente testo: "*Premendo accetta acconsentirà di partecipare alla campagna di acquisizione del dispositivo PulsEcg del Politecnico di Torino. I dati da Lei forniti (Nome, Cognome, Elettrocardiogramma, Pletismografia e Pressione) saranno gestiti solo da parte del Politecnico di Torino e non da terze parti. Per maggiori dettagli visitare il sito internet: http://neuronica.polito.it/wp-content/uploads/2021/11/Informativa-privacy_App-PulsECG.pdf* ". Esso è una finestra vincolante, si potrà continuare ad utilizzare l'app solo premendo il tasto *accetta*.

Il pop-up è stato implementato nel *HomePage VC*, in particolare nella funzione *viewDidAppear*. In questa funzione si controlla se è stata inizializzata la chiave

privacyIsAccepted tramite la funzione *UserDefaults.standard.integer(forKey:)*; se la chiave non è stata inizializzata, il metodo ritorna il valore intero 0, quindi siamo nel caso del primo avvio dell'app. Infine, per rendere il sito un link cliccabile è stata utilizzata la classe *NSMutableAttributedString* utile per aggiungere dei metodi aggiuntivi per cambiare il contenuto di una stringa [37] e la classe *URL* per renderlo effettivamente un link[38].
Di seguito viene riportato il codice per la generazione del pop-up:

```
1  var privacyIsAccepted : Int? = UserDefaults.standard.integer(forKey:
  "privacyIsAccepted")
2  if(privacyIsAccepted == 0){
3
4     let msg = "Premendo accetta acconsentirà di partecipare alla
     campagna di acquisizione del dispositivo PulsEcg del Politecnico di
     Torino. I dati da Lei forniti (Nome, Cognome, Elettrocardiogramma,
     Pletismografia e Pressione) saranno gestiti solo da parte del
     Politecnico di Torino e non da terze parti. Per maggiori dettagli
     visitare il sito internet "
5
6     let alertPrivacy = UIAlertController(title: "Accept Privacy",
     message: msg, preferredStyle: UIAlertController.Style.alert)
7     let textView = UITextView()
8     textView.autoresizingMask = [.flexibleWidth, .flexibleHeight]
9     let controller = UIViewController()
10    textView.frame = alertPrivacy.view.frame
11    controller.view.addSubview(textView)
12    textView.backgroundColor = UIColor.clear
13
14    alertPrivacy.setValue(controller, forKey: "contentViewController")
15
16    let attributedString = NSMutableAttributedString(string:
     "http://neuronica.polito.it/wp-content/uploads/2021/11/Informativa-pr
     ivacy_App-PulsECG.pdf")
17    let url = URL(string:
     "http://neuronica.polito.it/wp-content/uploads/2021/11/Informativa-pr
     ivacy_App-PulsECG.pdf")
18
19    attributedString.setAttributes([.link: url ?? ""], range:
     NSMakeRange(0, attributedString.length))
20
21    textView.attributedText = attributedString
22    textView.isUserInteractionEnabled = true
23    textView.isEditable = false
24
25    textView.linkTextAttributes = [
```

```
26            .foregroundColor: UIColor.blue,
27            .underlineStyle: NSUnderlineStyle.single.rawValue]
28
29
30    alertPrivacy.addAction(UIAlertAction(title: "Accetta", style:
    .default, handler: {[weak alertPrivacy] (_) in
31    UserDefaults.standard.set(1, forKey: "privacyIsAccepted")
32    }))
33
34
35    self.present(alertPrivacy, animated: true, completion: nil)
36 }
```

4.1.2 Interfacce ed user experience

In questa sezione parleremo dei componenti appartenenti al front-end. In particolare analizzeremo la realizzazione tecnica, le problematiche riscontrate e le soluzioni adottate.

HomePage VC

Esso è il primo view controller con cui l'utente viene a contatto una volta aperta l'applicazione (Fig. 4.3). In cima troviamo una **ImageView** rappresentante il logo dell'app. Inoltre, l'interfaccia contiene un oggetto **UITableView** per organizzare le proprie celle chiamate **TableViewCell.** Le celle contengono sia il layout che la logica di funzionamento di ogni elemento. La prima cella contiene il collegamento con il *ConnectBluetoothDevice VC*, la seconda permette il collegamento con l'*AcquireECG VC* e la terza è collegata all'*Archivio VC*. Infine, in alto a destra della prima cella troviamo un **UIButton** raffigurante un ingranaggio con cui è possibile collegarsi al *GeneralOption VC*.

L'unica problematica rilevata è stata nella prima cella in cui è raffigurata la connessione al dispositivo PulsEcg, ovvero l'assenza di qualche indicatore che raffigurasse lo stato della batteria del dispositivo. La presenza di questo elemento aumenta la **User Experience** (**UX**). Permetterebbe all'utente di avere la consapevolezza dello stato del dispositivo ed evitare spiacevoli inconvenienti.

Figura 4.3: Interfaccia HomePage

Quando l'app si connette ad un dispositivo riceve una notifica asincrona attivando il metodo *@objc private func justConnected(_ notification: Notification)*, aggiornando la prima cella che raffigura lo stato di connessione con il dispositivo. Di seguito il codice della funzione *justConnected*:

```
1   let pName = BleManager.sharedInstance.getConnectedDevice().name!
2   var valueBattery=UserDefaults.standard.string(forKey:"valueBattery")
3   if(valueBattery == nil){
4      valueBattery = "100"
5    }
6   menuOptions[0] = HomePageMenuOption(titolo: "Dispositivo:
   "+pName,im: #imageLiteral(resourceName: "bluetooth"),battery:
   valueBattery!)
7   tableView.reloadData()
```

La funzione appena presentata recupera attraverso *UserDefaults* l'ultimo valore aggiornato della batteria. Si è scelta questa soluzione perché il dispositivo manda

lo stato della batteria nei primi pacchetti di una acquisizione, di conseguenza non è possibile saperlo con la sola connessione al dispositivo. Infine, tramite il metodo di inizializzazione della classe *HomePageMenuOption* i parametri della cella vengono aggiornati. Di seguito il codice dell'aggiornamento dei dati:

```
1  nome.text = menuOption.nome
2  imag.image = menuOption.im
3  let received=menuOption.battery
4  if(received == "-1"){
5     battery.isHidden=true
6     imgBattery25.isHidden=true
7     imgBattery.isHidden=true
8   }
9   else{
10    battery.isHidden=false
11    let bat = " " + received + "%"
12    if Int(received)! >= 99 {
13       imgBattery25.isHidden=true
14       imgBattery.isHidden=false
15       battery.text=bat
16    }
17    else if Int(received)! >= 50 {
18       imgBattery25.isHidden=true
19       imgBattery.isHidden=false
20       battery.text=bat
21    }
22    else if Int(received)! <= 50 {
23       battery.text=bat
24       imgBattery25.isHidden=false
25       imgBattery.isHidden=true
26    }
```

Per rappresentare i vari livelli dello stato di batteria sono state utilizzate due immagini prontamente mostrate o nascoste a seconda del valore ricevuto. Inoltre, l'immagine è accompagnata da un valore in percentuale del livello di carica del dispositivo.

GeneralOption

Il view controller GeneralOption (Fig. 4.4) è composto da dei **StackView** (**SV**) verticali e orizzontali annidati. Un SV è un oggetto che semplifica la gestione di layout di altre view. Esso è composto da una serie di **UILabel**, che possono essere dei bottoni, dei segment control (per scegliere tra due opzioni) oppure degli switch (per abilitare o meno una funzionalità). In questa interfaccia l'utente può specificare diversi dati utili all'applicazione:

- Dati personali: nome, cognome e indirizzo email;
- Setup dispositivo: braccio utilizzato per l'acquisizione, tempo di acquisizione, condivisione automatica della posizione, salvataggio automatico delle acquisizioni, email del medico e collegamento al dispositivo;

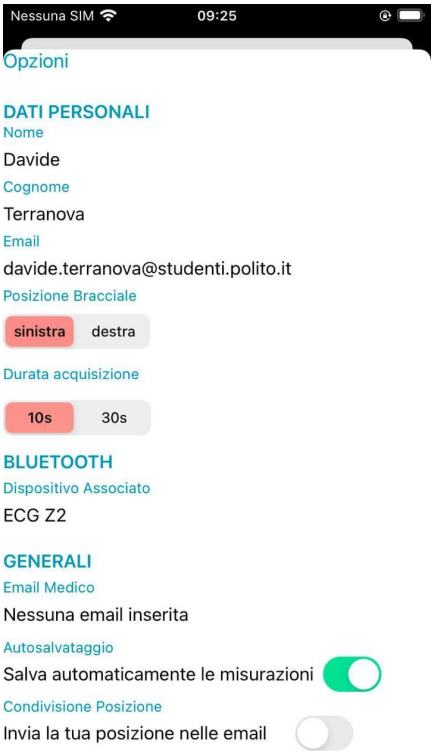


Figura 4.4: Interfaccia GeneralOption

In questa parte dell'app abbiamo aggiunto un *segment control* per settare la durata della acquisizione. In questa nuova versione l'utente può decidere se eseguire una acquisizione da 10 o 30 secondi.

La funzione handler *onChangeValueSecondsAcquisition* salverà negli *UserDefaults* (file system dell'applicazione) il valore scelto ed aggiornerà le specifiche da mandare al dispositivo. Di seguito il codice di *onChangeValueSecondsAcquisition*:

```
1  let index = sender.selectedSegmentIndex
2  if(index==0){
3     defaults.set(10, forKey: "AcquisitionSecondsInt")
4     defaults.set(0x0E, forKey: "AcquisitionSecondsHex")
5     let acquisitionSeconds = defaults.integer(forKey:
     "AcquisitionSecondsInt")
6     UserDefaults.standard.set(acquisitionSeconds * 1000, forKey:
     "Milliseconds_of_aquisition")
7     let n_Samples = 500 * acquisitionSeconds
8     UserDefaults.standard.set(n_Samples, forKey: "no_of_samples")
9  }
10 else{
11      defaults.set(30, forKey: "AcquisitionSecondsInt")
12      defaults.set(0x22, forKey: "AcquisitionSecondsHex")
13      let acquisitionSeconds = defaults.integer(forKey:
       "AcquisitionSecondsInt")
14      defaults.set(acquisitionSeconds * 1000, forKey:
       "Milliseconds_of_aquisition")
15      let n_Samples = 500 * acquisitionSeconds
16      defaults.set(n_Samples, forKey: "no_of_samples")
17 }
```

A seconda del tempo scelto verrà aggiornato il tempo globale di acquisizione, il valore esadecimale da mandare al dispositivo, il corrispettivo valore in millisecondi e il numero totale di pacchetti che verranno mandati dal dispositivo.
Avere a disposizione una acquisizione da 30 secondi comporta svariati vantaggi; l'applicazione avrà una maggiore possibilità di individuare una fibrillazione atriale e potrà calcolare un valore più preciso dei battiti cardiaci e della saturazione del sangue. Inoltre, il medico potrà avere una panoramica più completa dei segnali ECG e PPG.

ConnectBluetoothDevice

Il ConnectBluetoothDevice VC è l'interfaccia preposta per la connessione bluetooth con il dispositivo PulsEcg. Essa permette di scansionare, associare e disconnettere i dispositivi. Come possiamo vedere in fig. 4.5, l'interfaccia è composta da un bottone per tornare indietro, alcune labels per raffigurare il dispositivo attualmente associato e una **UITableView** per organizzare i dispositivi PulsEcg disponibili. All'avvio del viewcontroller si esegue una prima scansione. Se è presente l'ultimo dispositivo associato memorizzato in UserDefault, l'app provvederà a connettersi automaticamente, altrimenti basta un tap sulla riga di un dispositivo. Inoltre è possibile riattivare lo scanning con una tap-to-refresh. Questa funzionalità è possibile grazie ad un oggetto chiamato **UIRefreshControl** agganciato ad una ScrollView.

Il viewcontroller è associato ad una classe chiamata **BleManager**. Questa classe lavora in modalità background, ed implementa tutti i protocolli e funzionalità utili per una connessione bluetooth (ne parleremo nel capitolo 4.1.5).

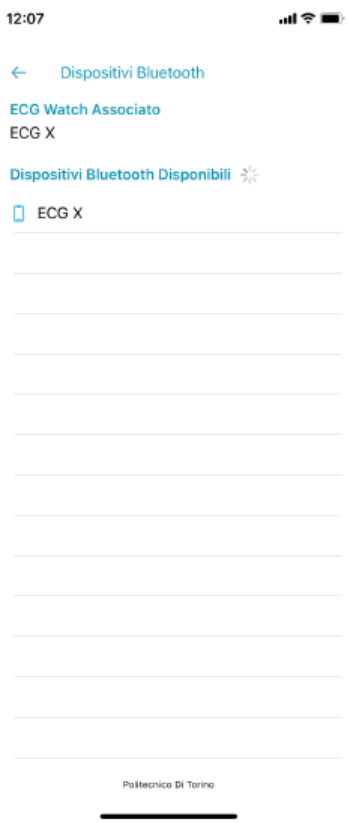


Figura 4.5: Interfaccia ConnectBluetoothDevice

AcquireECG

Dall'HomePage per avviare un'acquisizione utilizziamo il viewcontroller AcquireECG. L'interfaccia è mostrata solo se l'app è già stata associata ad un dispositivo. In caso contrario compare un pop-up permettendo all'utente di collegarsi immediatamente con il view controller ConnectBluetoothDevice (Fig. 4.6).

Figura 4.6: AcquireECG con dispositivo non associato

L'interfaccia AcquireECG è formata semplicemente da un bottone per tornare indietro, una gif animata che raffigura un segnale ecg, un'immagine dello stato della batteria e una label associata ad un count down della durata di un'acquisizione (fig. 4.7).

Il view controller inizialmente setta un timer per il count down, recuperando il tempo di acquisizione da UserDefaults. In seguito, chiama un metodo del BleManager per avviare l'acquisizione, ed eseguirà una attesa con una durata che varia da un minimo di 14 secondi fino ad un massimo di 34 secondi. Terminata l'acquisizione, il BleManager invia una notifica contenente i segnali ECG e PPG.

In questa fase finale, della durata di un secondo circa, compie le seguenti azioni:

1. Trasforma i dati dei segnali in double;
2. Amplifica e filtra i segnali;
3. Applica la media mobile e il detrending ai segnali;
4. Calcola i picchi e individua una possibile fibrillazione atriale;
5. Determina i massimi e i minimi locali del PPG per il calcolo della SpO_2;
6. Istanzia la rete neurale e ricava i valori di pressione sistolica;

Infine, se è attivo l'autosalvataggio, avviene la memorizzazione dell'acquisizione inizializzando un oggetto chiamato **Misurazione**.
La pressione arteriosa e l'elaborazione dei segnali vengono calcolati rispettivamente tramite l'utilizzo della classe NeuralNetworkManager (capitolo 4.1.7) e della classe SignalProcessing (capitolo 4.1.6).

Figura 4.7: Interfaccia AcquireECG

In questo VC è stato modificato il settaggio del timer. In questa nuova versione è possibile avviare una acquisizione da 10 o 30 secondi. Inoltre, sono stati aggiunti ulteriori 4 secondi per evitare di registrare imprecisioni dovute alla fase preliminare di posizionamento del dito. Come vedremo nel capitolo 4.1.5 i secondi aggiuntivi verranno effettivamente acquisiti ed infine tagliati, ottenendo un segnale pulito da imprecisioni motorie. Di seguito il codice per settare il timer:

```
1   let timeAcquisition = UserDefaults.standard.integer(forKey:
   "AcquisitionSecondsInt")
2   self.count += 4
3   timerAquire = Timer.scheduledTimer(withTimeInterval: 1.0,
   repeats:true) { timer in
4       if self.count > timeAcquisition{
5          self.info.text = "Fase di preparazione mantenere la
           posizione.."
6       }
7       else if self.count != 1 {
8          self.info.text = "Attendi ancora "+String(self.count)+"
           secondi.."
9       }
10      else if self.count==1 {
11         self.info.text = "Attendi ancora "+String(self.count)+"
           secondo.."
11      }

12      self.count -= 1
13      if self.count < 0 {
14         timer.invalidate()
15         return
16      }
17  }
```

Archivio

Il view controller Archivio gestisce tutte le acquisizioni passate. L'interfaccia associata, raffigurata in fig. 4.8, contiene i seguenti elementi:

- un bottone per tornare indietro
- un bottone per eseguire una ricerca di un file

- un bottone per importare un'acquisizione dal file system
- una UITableView per organizzare i vari file

I vari file sono organizzati per mese-anno e sono nominati con nome e cognome dell'utente, data e ora di acquisizione.

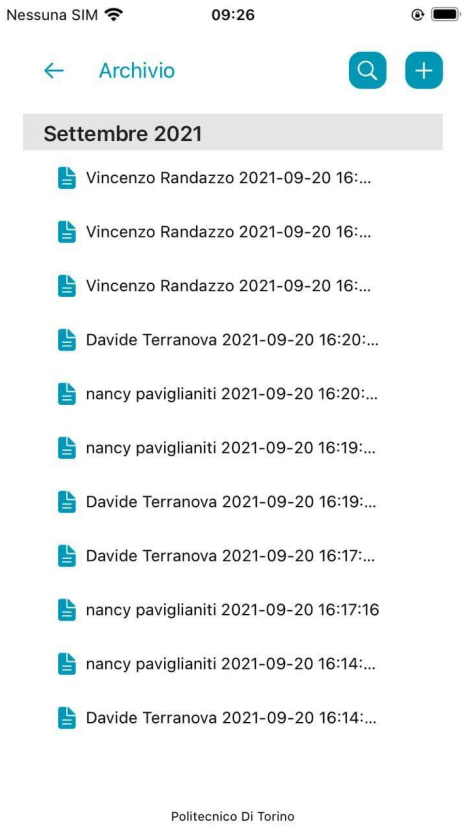
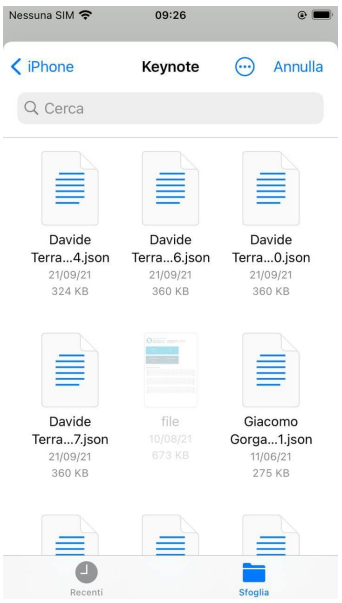
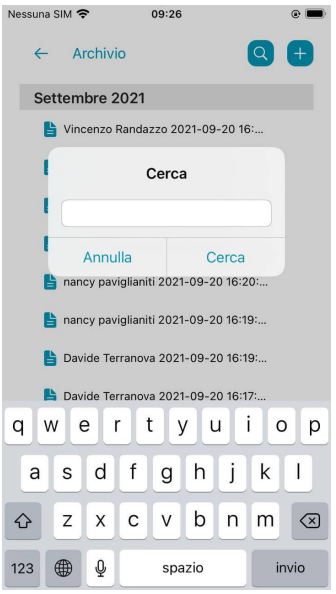


Figura 4.8: Interfaccia Archivio

Il **bottone importazione** è associato ad una azione che crea un **Document Picker VC** (Fig. 4.9-a).
Il **bottone ricerca** presenterà in pop-up con una barra di testo che utilizzerà per il filtraggio dei file (Fig. 4.9-b).



(a): Document Picker VC



(b): pop-up ricerca

Figura 4.9: Interfaccia Document Picker e pop-up ricerca

GeneralShowECG

L'interfaccia associata al viewcontroller GeneralShowECG, mostra una panoramica generale dei segnali e dei parametri vitali. Essa è presentata dopo la fine di una acquisizione, dall'AcquireECG VC, oppure è raggiungibile dall'Archivio VC, aprendo una misurazione. La view è composta da una serie di UILabel affiancate, a cui è stato assegnato un colore di background. Nella prima scheda vengono mostrati 5 secondi del segnale ecg tramite una LineCharView (capitolo 4.1.3) opportunamente configurata per non mostrare nessuna griglia. In figura 4.10 è mostrata un'anteprima dell'interfaccia appena descritta.

Il VC riceve il file misurazione, in cui sono presenti tutti i segnali e i parametri vitali derivati, preleva i dati dei vari attributi e li posiziona nelle giuste label. Inoltre, controlla se i valori appena prelevati contengono delle anomalie, sintomo di possibili problemi di salute:

1. Fibrillazione atriale;
2. battito cardiaco troppo basso (<=60 bpm) oppure troppo alto (>=140 bpm)
3. Saturazione del sangue troppo bassa (<=90%)

4. Ipertensione arteriosa (SBP >=180 oppure DBP >= 110)
5. Ipotensione arteriosa ((SBP <=90 oppure DBP <= 60)

Ogni label è interattiva, ovvero tramite un tocco è possibile mostrare i suoi dettagli passando al ShowECG VC. Questo è stato possibile aggiungendo un oggetto chiamato **UIGestureRecognizer**. Questo oggetto permette il riconoscimento dei gesti sui componenti attivando una funzione; in questo caso gli oggetti eseguono un performeSegue al ShowECG. Ad ogni elemento viene assegnata una gesture tramite il metodo *addGestureRecognizer*.

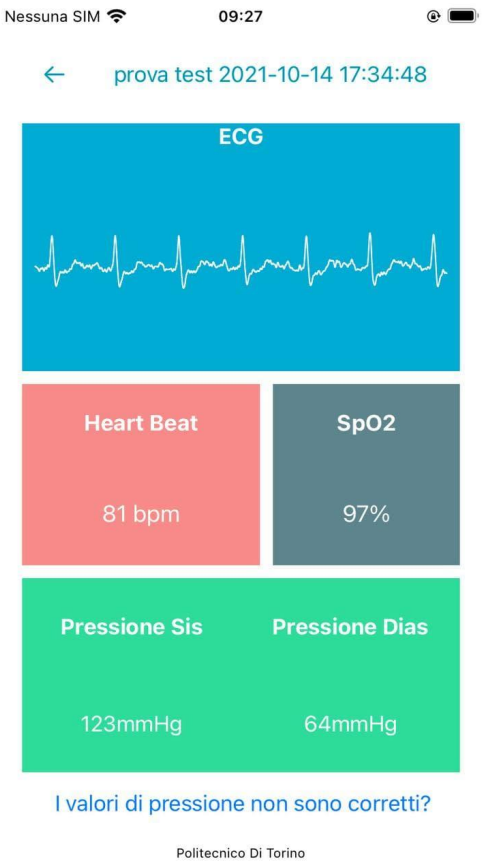


Figura 4.10: Interfaccia GeneralShowECG

Nella nuova versione del VC, sono state aggiunte due label per rappresentare i valori di pressione e per la sua calibrazione.

Per la label raffigurante i valori di pressione arteriosa è stata seguita la stessa tecnica delle label precedenti. Per l'ultima label "*I valori di pressione non sono corretti?*", posizionata in fondo alla schermata, è stata aggiunta una gesture chiamata UITapGestureRecognizer. Essa, riconoscendo un semplice tocco, avvia la funzione *chancheOffsetPression* che si farà carico di aprire un alert per la calibrazione della pressione (Fig. 4.11). Con la calibrazione modificheremo l'offset statico che si aggiunge ad ogni valore di pressione per adattarla meglio ad ogni tipologia di paziente. L'alert contiene una descrizione, due campi numerici per inserire i valori di pressione sistolica e diastolica, e due bottoni utili per confermare i dati oppure per tornare indietro.

Figura 4.11: Alert per la calibrazione della pressione

Di seguito è riportato il codice generato per la gestione della logica di funzionamento e della presentazione dell'alert:

```
1    let alert = UIAlertController(title: "Correzione Pressione",
    message: "Se i valori di pressione mostrati non sono conformi con la
    realtà, inserire i dati corretti misurati con uno sfigmmomanometro
    certificato", preferredStyle: .alert)
2    alert.addTextField { (textField) in
3        textField.placeholder = "Pressione sistolica"
4        textField.isSecureTextEntry = false
5        if (UIDevice.current.userInterfaceIdiom == .pad) {
6            textField.keyboardType = UIKeyboardType.numberPad // Only
            numerci input
```

```
7         } else {
8             textField.keyboardType = UIKeyboardType.numberPad  // Only
              numerci input
9         }
10    }
11    alert.addTextField { (textField) in
12        textField.placeholder = "Pressione diastolica"
13        textField.isSecureTextEntry = false
14        if (UIDevice.current.userInterfaceIdiom == .pad) {
15            textField.keyboardType = UIKeyboardType.numberPad // Only
             numerci input
16        } else {
17            textField.keyboardType = UIKeyboardType.numberPad  // Only
             numerci input
18        }
19    }
20    alert.addAction(UIAlertAction(title: "Invia", style: .default,
    handler: { [weak alert] (_) in
21        let textFieldSist = alert?.textFields?[0]
22        let textFieldDia = alert?.textFields?[1]
23        let intSistolica = Int(textFieldSist!.text!)
24        let intDiastolica = Int(textFieldDia!.text!)
25        if intSistolica == nil || intDiastolica == nil{
26            let alertError = UIAlertController(title: "Errore",
             message: "Sia la sistolica che la diastolica devono
             contenere dei valori", preferredStyle: .alert)
27            alertError.addAction(UIAlertAction(title: "Ok", style:
             .default, handler: nil))
28            self.present(alertError, animated: true, completion: nil)
29        }
30        else if intSistolica!-intDiastolica! < 15 ||
         intSistolica!-intDiastolica! > 60 || intSistolica! < 70 ||
         intSistolica! > 200 || intDiastolica! < 50 || intDiastolica! >
         120  {
31            let alertError = UIAlertController(title: "Errore",
             message: "I valori di sistolica devono essere compresi tra
             70-180, la diastolica compresa tra 40-100 e la differenza
             tra sistolica e diastolica deve essere maggiore di 15",
             preferredStyle: .alert)
32            alertError.addAction(UIAlertAction(title: "Ok", style:
             .default, handler: nil))
33            self.present(alertError, animated: true, completion: nil)
34        }
35        else{
36            let offsetDia = UserDefaults.standard.integer(forKey:
             "offsetDiastolica")
37            let offsetSi = UserDefaults.standard.integer(forKey:
             "offsetSistolica")
38            let pressSist = self.filteredMisurazione!.getpressSis()
39            let pressDia = self.filteredMisurazione!.getpressDia()
40            let differenceSist = intSistolica! - pressSist
41            let differenceDia = intDiastolica! - pressDia
```

```
42          let newOffsetSist = (offsetSi + (offsetSi +
           differenceSist)) / 2
43          let newOffsetDia = (offsetDia + (offsetDia +
           differenceDia)) / 2
44          UserDefaults.standard.set(newOffsetSist, forKey:
           "offsetSistolica")
45          UserDefaults.standard.set(newOffsetDia, forKey:
           "offsetDiastolica")
46          self.receivedMisurazione!.setpressSis(newValue:
           intSistolica!)
47          self.receivedMisurazione!.setpressDia(newValue:
           intDiastolica!)
48          self.receivedMisurazione!.setTruePressureCalibrated()
49          self.offsetPressione.text = "La pressione è già stata
           calibrata"
50          self.offsetPressione.isUserInteractionEnabled = false
51          self.pressSis.text = String(intSistolica!)+"mmHg"
52          self.pressDias.text = String(intDiastolica!)+"mmHg"
53          self.eliminaFile()
54          self.saveFile()
55      }
56  }))

57  alert.addAction(UIAlertAction(title: "Indietro", style: .default))
58
59  self.present(alert, animated: true, completion: nil)
```

Per evitare che l'utente possa inserire inavvertitamente dei dati intrinsecamente errati, come ad esempio del testo, i campi della pressione sistolica e diastolica accettano solo valori numerici. Inoltre, per agevolare l'utente, al tocco di un campo di inserimento si apre automaticamente il pad numerico.

Il tasto "invia" esegue un controllo sui valori di pressione appena inseriti. I valori di sistolica devono essere compresi tra 70-180, la diastolica compresa tra 40-100 e la differenza tra sistolica e diastolica deve essere maggiore di 15. Una volta superati tutti i controlli si calcolano i valori di offset separatamente per la pressione sistolica e diastolica. Il primo offset è calcolato come differenza tra i valori della rete neurale e i valori della strumentazione certificata. I successivi offset sono calcolati come (calibrazione attuale * 2 + nuovo offset) / 2.

Infine, terminato il calcolo degli offset, viene disattivata l'interattività con la label e il testo sostituito con "*La pressione è già stata calibrata*" e aggiornate le label della pressione arteriosa con i nuovi valori.

ShowECG

Il compito di questo VC è quello di mostrare nel dettaglio i segnali acquisiti tramite il bracciale PulsEcg ed i valori di pressione ricavati dalla rete neurale. Come mostrato in figura 4.12, l'interfaccia è composta dai seguenti elementi:

- Nella parte superiore sono presenti diversi bottoni: per tornare indietro, per eliminare la misurazione, per esportare il file e per accedere all'InfoEcg VC.
- Due labels per descrivere il tipo di segnale e il valore associato.
- Una LineChartView per mostrare il segnale su carta millimetrata.
- Uno Stack View orizzontale contenente dei bottoni. Da qui si può passare facilmente da un segnale all'altro oppure visualizzare i valori di pressione arteriosa.
- Uno Stack View orizzontale che a sua volta contiene due SV verticali per mostrare i valori di pressione arteriosa.

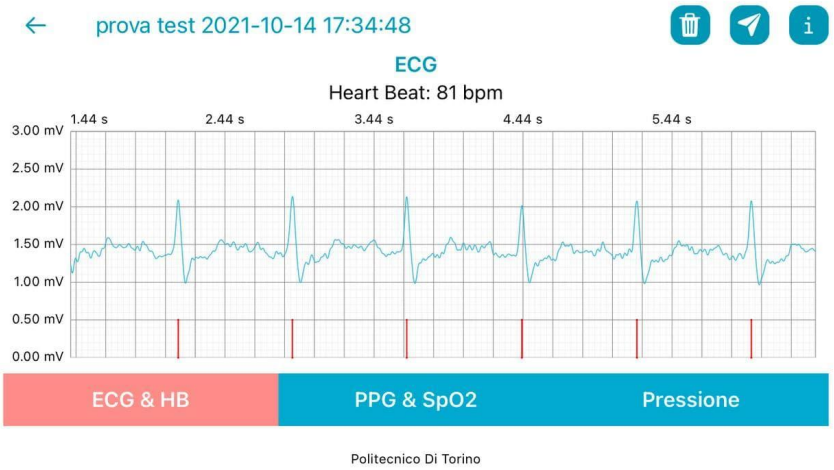


Figura 4.12: Interfaccia del ShowEcg con un segnale ECG

Le principali funzioni sono:

- **preparaGrafici()**: questo metodo inizializza i grafici da mostrare, in particolare il grafico ECG e il grafico PPG. Inoltre, per il segnale ecg, individua i picchi da visualizzare.

- **mostraEcg()**: incaricata ad aggiornare i dati del grafico e gli elementi di contorno come l'HeartBeat, il nome del grafico e il colore del pulsante "ECG & HB", mostrato in figura 4.12.
- **mostraPPG()**: ha la stessa funzione di *mostraEcg* ma gestisce il caso del segnale PPG, mostrato in 4.13.
- **mostraPressione()**: questa funzione nasconde il LineChartView e mostra i valori di pressione, mostrato in figura 4.14.

Figura 4.13: Interfaccia del ShowEcg con un segnale PPG

Figura 4.14: Interfaccia del ShowEcg con i valori di pressione

La nuova versione del view controller ShowEcg contiene delle modifiche sulla presentazione dei segnali ECG e PPG.

In particolare sono state cambiate delle caratteristiche dell'oggetto *LineChartView*. Con l'estensione del tempo di acquisizione a 30 secondi, come affrontato nei capitoli precedenti, si dovuta modificare il metodo *preparaGrafici()*. Il tempo di acquisizione non è più una costante ma una variabile salvata in *UserDefaults*, a seconda della durata del segnale i grafici verranno inizializzati in maniera differente. Inoltre, è stata modificata la posizione e il colore delle barre che denotano i battiti cardiaci (coincidenti con i picchi del segnale ECG). Con questa soluzione è possibile individuare i battiti cardiaci e conseguentemente i picchi del segnale ECG in maniera immediata.
Per quanto riguarda la presentazione del segnale ECG, è stato modificato il metodo *mostraECG()*; adesso l'asse delle ordinate ha una scala fissa che varia da 0.00 mV (millivolt) a 3.00 mV. Nella versione precedente, la scala delle ordinate era adattiva, ovvero dipendeva dal valore massimo e minimo del segnale, ma questa soluzione non permetteva un possibile confronto con altri segnali ECG.
Per il segnale PPG, modificando il metodo *mostraPPG()*, è stato implementato un grafico adattivo; inoltre, sono state eliminate le righe orizzontali della carta millimetrata digitale ma i dettagli di questa parte li vedremo nel capitolo 4.1.3.

4.1.3 Charts

Nel progetto PulsEcg grande attenzione si è data alla presentazione grafica dei segnali ECG e PPG. Per avere dei grafici personalizzabili, zoomabili ed esteticamente piacevoli, si è scelta la libreria **Charts**. La libreria è scritta interamente in Swift, diretta traduzione della libreria Android Java MPAndroidChart.
Il **grafico ECG** contiene il set di campioni del segnale e delle barre posizione sotto ogni picco del segnale, indicanti i battiti cardiaci. Inoltre, contiene una griglia per agevolare il medico nella lettura del segnale.
Il **grafico PPG** rappresenta sia il segnale infrarosso che il rosso e la griglia è formata solo da linee verticali, parallele all'asse y.
La sostanziale modifica fatta nella nuova versione dell'app è stata quella di rendere la griglia variabile a seconda se è mostrato il segnale ECG o PPG. Di seguito il codice generato per la funzionalità appena descritta:

```
1   let isEcg = UserDefaults.standard.bool(forKey: "gridEcg")
2   if(isEcg){
3      positions.forEach {
4         if i % 5 == 0 {
5            context.setStrokeColor(yAxis.gridColor.cgColor)
6            context.setLineWidth(0.5)
7            drawGridLine(context: context, position: $0)
8         }
9         else {
10           context.setStrokeColor(yAxis.gridColor.cgColor)
11           context.setLineWidth(yAxis.gridLineWidth)
12           drawGridLine(context: context, position: $0)
13        }
```

Se il booleano, con chiave “gridEcg”, è true l’algoritmo provvederà a disegnare le righe orizzontali, altrimenti avrà disegnato solo le righe verticali.

4.1.4 Import-Export

Nella nuova versione dell’app, nella parte di import-export è stata modificata la funzione per generare le immagini per la creazione del **PDF**. In particolare, è stata migliorata la funzione *getGraphImage*. Adesso la funzione supporta segnali da 10 o 30 secondi e la griglia differente a seconda se è un segnale ECG o PPG, come abbiamo visto nel capitolo 4.1.3. In seguito, abbiamo migliorato la risoluzione del grafico e la visibilità dei picchi. Il codice seguente mostra le modifiche apportate:

```
1   private func getGraphImage(graphName: String, tenSeconds: Int)
   -> LineChartView{
2       var ecgsets : [LineChartDataSet] = []
3       var ecgPeak : [Double] = []
4       let seconds_of_aquisition =
       UserDefaults.standard.integer(forKey: "AcquisitionSecondsInt")
5       var temporaryChart = LineChartView()
6       temporaryChart.frame = CGRect(x: 0 , y: 0, width: 800, height:
       145)
7       temporaryChart.extraTopOffset = 20
```

```
8       temporaryChart.extraLeftOffset = 30
9       temporaryChart.extraRightOffset = 30
10      temporaryChart.legend.enabled = false
11      if graphName == "ECG" {
12         var ecgData : [ChartDataEntry] = []
13         if tenSeconds == 0 {
14            for i in 0 ..<
              (filteredMisurazione!.getECGSamples().count) {
15                ecgData.append(ChartDataEntry(x: Double(i), y:
                  Double(filteredMisurazione!.getECGSamples()[i])*10))
16                ecgPeak.append(Double(filteredMisurazione!.
                  getECGSamples()[i]))
17            }
18         }
19         else {
20            for i in (tenSeconds*5000 - 1) ..<
              (filteredMisurazione!.getECGSamples().count-((2-tenSeconds)*
              5000)) {
21                ecgData.append(ChartDataEntry(x: Double(i), y:
                  Double(filteredMisurazione!.getECGSamples()[i])*10))
22                ecgPeak.append(Double(filteredMisurazione!.
                  getECGSamples()[i]))
23            }
24         }
25         let ecgset = LineChartDataSet(entries: ecgData, label: "ECG")
26         ecgset.highlightEnabled = false
27         ecgset.circleRadius=CGFloat.init(0.2)
28         ecgset.circleColors = [#colorLiteral(red: 0, green:
           0.6710312963, blue: 0.8332495093, alpha: 1)]
29         ecgset.colors = [#colorLiteral(red: 0, green:
           0.6710312963,blue: 0.8332495093, alpha: 1)]
30         ecgset.lineWidth = 0.5
31         ecgsets.append(ecgset)
32
33         var dataPointsPeaks : [ChartDataEntry] =
           [ChartDataEntry](repeating: ChartDataEntry(x:0,y:0), count: 2 *
           ((Constants.MAX_BPM / 60) * seconds_of_aquisition))
34         let absoluteMax = averaged_max(input : ecgPeak)
35         findPeak(input: ecgPeak, GVDataPeaks: &dataPointsPeaks,
           absolute_max: absoluteMax)
36
37         var GVDataPeak : [ChartDataEntry] =
           [ChartDataEntry](repeating:ChartDataEntry(x: 0, y: 0),count: 2)
38         for i in 0 ..< dataPointsPeaks.count {
39            if i != 0 &&
40               (dataPointsPeaks[i].x == 0 ||
41                dataPointsPeaks[i].x == dataPointsPeaks[i-1].x) {
42                break
43             }
44             let peakX : Double = dataPointsPeaks[i].x+Double(5000 *
               tenSeconds)
45             let peakY : Double = -5
```

```
46              GVDataPeak[0] = ChartDataEntry(x: peakX, y: peakY+5)
47              GVDataPeak[1] = ChartDataEntry(x: peakX, y: peakY+10)
48              let tempSet = LineChartDataSet(entries: GVDataPeak,
                label: "Picco")
49              tempSet.circleRadius=CGFloat.init(0.8)
50              tempSet.circleColors = [#colorLiteral(red: 1, green: 0,
                blue: 0, alpha: 1)]
51              tempSet.colors = [#colorLiteral(red: 1, green: 0, blue:
                0, alpha: 1)]
52              ecgsets.append(tempSet)
53          }
54          temporaryChart.data = LineChartData(dataSets: ecgsets)
55      }
56      else {
57          temporaryChart.frame = CGRect(x: 0 , y: 0, width: 800, height:
            175)
58          temporaryChart.leftAxis.drawLabelsEnabled = false
59          temporaryChart.rightAxis.drawLabelsEnabled = false
60          if tenSeconds == 0 {
61              temporaryChart.data = dataPPG
62          }
63          else {
64              var sets : [LineChartDataSet] = []
65              var ppgIRdata : [ChartDataEntry] = []
66              for i in (tenSeconds*5000 - 1) ..<
                filteredMisurazione!.getPPGSamplesIR().count {
67                  ppgIRdata.append(ChartDataEntry(x: Double(i), y:
                    Double(filteredMisurazione!.getPPGSamplesIR()[i])))
68              }
69              let ppgIRset = LineChartDataSet(entries: ppgIRdata, label:
                "PPG_IR")
70              ppgIRset.highlightEnabled = false
71              ppgIRset.circleRadius=CGFloat.init(0.5)
72              ppgIRset.circleColors = [#colorLiteral(red: 0, green:
                0.6710312963, blue: 0.8332495093, alpha: 1)]
73              ppgIRset.colors = [ #colorLiteral(red: 0, green:
                0.6710312963, blue: 0.8332495093, alpha: 1) ]
74              sets.append(ppgIRset)
75
76              var ppgREDdata : [ChartDataEntry] = []
77              for i in (tenSeconds*5000 - 1) ..<
                filteredMisurazione!.getPPGSamplesRED().count {
78                  ppgREDdata.append(ChartDataEntry(x: Double(i), y:
                    Double(filteredMisurazione!.getPPGSamplesRED()[i])))
79              }
80              let ppgREDset = LineChartDataSet(entries: ppgIRdata,
                label: "PPG_RED")
81              ppgREDset.highlightEnabled = false
82              ppgREDset.circleRadius=CGFloat.init(0.5)
83              ppgREDset.circleColors = [#colorLiteral(red: 0.9168450342,
                green: 0.4932718873, blue: 0.4739984274, alpha: 1)]
```

```
84            ppgREDset.colors = [ #colorLiteral(red: 0.9168450342,
              green: 0.4932718873, blue: 0.4739984274, alpha: 1) ]
85            sets.append(ppgREDset)
86            temporaryChart.data = LineChartData(dataSets: sets)
87         }
88      }
89
90      temporaryChart.viewPortHandler.setMinimumScaleX(CGFloat
       (seconds_of_aquisition / 10))
91      temporaryChart.viewPortHandler.setMaximumScaleX(CGFloat
       (seconds_of_aquisition / 10))
92      temporaryChart.xAxis.labelCount = ( seconds_of_aquisition * 1000
       / 40 )
93      temporaryChart.leftAxis.labelCount = 30
94      temporaryChart.rightAxis.labelCount = 30
95      temporaryChart.setVisibleXRangeMaximum(5000)
96      temporaryChart.setVisibleXRangeMinimum(5000)
97      if graphName == "ECG" {
98         UserDefaults.standard.setValue(true, forKey: "gridEcg")
99         temporaryChart.leftAxis.axisMaximum = 30
100        temporaryChart.leftAxis.axisMinimum = 0
101        temporaryChart.rightAxis.axisMaximum = 30
102        temporaryChart.rightAxis.axisMinimum = 0
103     }
104     else{
105        UserDefaults.standard.setValue(false, forKey: "gridEcg")
106        temporaryChart.leftAxis.resetCustomAxisMax()
107        temporaryChart.leftAxis.resetCustomAxisMin()
108     }
109     return temporaryChart
110 }
```

4.1.5 BleManager

La classe BleManager, priva di interfaccia, gestisce la connessione bluetooth con il bracciale PulsEcg. Essa comunica con tutti gli altri view controller tramite un meccanismo di notifica asincrona, il Notification Center. Le notifiche vengono inviate ai view controller, che si registrano come *observer*, quando accade un determinato evento. Essa è una classe **Singleton**, ovvero esiste una sola istanza dell'oggetto nella applicazione; quindi tutti i view controller hanno lo stesso riferimento. Per implementare e gestire tutte le funzionalità della classe BleManager è stata utilizzata la libreria CoreBluetooth (capitolo 2.3).

In particolare, è stata utilizzata la classe **CBCentralManager**, che si occupa dello stato del bluetooth di sistema, dello scan dei dispositivi, della connessione/disconnessione a una periferica, dell'esplorazione di servizi e caratteristiche, e dei protocolli per la comunicazione.
Principali funzioni della classe:

1. **writeCommandStartAcquisition()** - è il metodo che invia il numero di secondi di acquisizione ed avvia l'acquisizione. È una funzione pubblica chiamata dal view controller AcquireECG.

2. **centralManagerDidUpdateState** - è il metodo con cui avviene lo scan delle periferiche.

3. **didDiscoverPeripheral** - metodo chiamato ogni volta che viene trovato un nuovo dispositivo. Se la periferica è un PulsEcg, viene aggiunta ad un array interno alla classe.

4. **didDisconnectPeripheral** - metodo chiamato quando avviene una disconnessione dal dispositivo. Manda una notifica avvisando tutti i VC interessati che la periferica non è più disponibile.

5. **didConnect** - metodo chiamato quando si connette ad una periferica; a sua volta chiamerà il metodo didDiscoverServices, per scoprire i servizi della periferica, che a sua volta chiamerà la funzione didDiscoverCharacteristics, per esplorare le sue caratteristiche.

6. **didUpdateValueFor** characteristic - è il metodo chiamato quando si ricevono i pacchetti dal dispositivo PulsEcg. I pacchetti vengono smistati a seconda della caratteristica di appartenenza (Batteria, ECG e PPG).

In questa nuova versione dell'app oltre ad aggiungere un nuovo tempo di acquisizione da 30 secondi, si è pensato di estendere i tempi di acquisizione di ulteriori 4 secondi. Questi secondi "cuscinetto" eliminano le imprecisioni motorie dell'utente quando inizia e termina una acquisizione, migliorando sensibilmente la qualità del segnale.

In questo modo è possibile analizzare e presentare un segnale pulito, perfezionando la precisione dei dati e la presentabilità dell'app. I 4 secondi sono stati suddivisi in 3 secondi all'inizio e 1 secondo alla fine dell'acquisizione.
Nella nuova versione della classe BleManager sono state modificate le seguenti funzioni: **writeCommandStartAcquisition()**, **didUpdateValueFor**, **getECGSamples**, **getPPGIRSamples** e **getPPGREDSamples**.
Nel metodo *writeCommandStartAcquisition()* si recupera il numero esadecimale da inviare al dispositivo da *UserDefaults*, a seconda da come è stato settato nel *GeneralOption VC* invierà "0x0E" (14 secondi in decimale) oppure "0x22" (34 secondi in decimale).
In *didUpdateValueFor* è stata implementata la logica per gestire i due tempi di acquisizione, settando correttamente il numero di pacchetti da ricevere.
Infine, nei metodi *getECGSamples*, *getPPGIRSamples* e *getPPGREDSamples* vengono ritornati gli array dei segnali tagliati di 4 secondi. Di seguito il codice di quanto spiegato:

```
1   func getECGSamples()->[UInt16]{
2        ECGSamples = Array(ECGSamples[1500..<(ECGSamples.count-500)])
3        return ECGSamples
4   }
5
6   func getPPGIRSamples()->[UInt32]{
7        PPGSamplesIR =
        Array(PPGSamplesIR[1500..<(PPGSamplesIR.count-500)])
8        return PPGSamplesIR
9   }
10
11  func getPPGREDSamples()->[UInt32]{
12       PPGSamplesRED =
        Array(PPGSamplesRED[1500..<(PPGSamplesRED.count-500)])
13       return PPGSamplesRED
14  }
```

Data una frequenza di campionamento di 500 Hz vengono eliminati (ignorati) i primi 1500 pacchetti (3 secondi) e gli ultimi 500 pacchetti (1 secondo).

4.1.6 Funzioni per l'elaborazione dei segnali

Qualsiasi segnale acquisito ed elaborato da un sistema hardware sarà soggetto al rumore, alterazioni indesiderate casuali del segnale. Esistono due tipologie di rumore: rumore termico e rumore esterno (o interferenze). Il primo è generato dalle temperature dei componenti hardware diverse dallo zero assoluto. Il secondo è causato dall'interferenza del mondo esterno nella fase di trasmissione[39].

Il rumore, generato da queste due componenti, può degradare molto la qualità e l'informazione utile del segnale. Per queste ragioni, nell'app PulsEcg, sono stati creati dei **filtri** per ridurre il rumore.

Tutte le funzioni e costanti utilizzate per raggiungere tale scopo sono state raccolte nella classe **SignalProcessing**.

Filtri ECG

Inizialmente, il segnale ECG è soggetto ad un pre-processamento. Il segnale viene **deamplificato** con una traslazione verso il basso di 1650 (mV) e poi diviso per 750. In seguito vengono applicati due filtri:

- **notchFilter** (filtro elimina-banda): è un filtro che non permette il passaggio di determinate frequenze. Nell'app viene utilizzata una funzione biquadratica ripetuta quattro volte con parametri differenti;

```
1    func notchFilter_10(x: [Double])->[Double]{
2         var result : [Double] = []
3         result = bquad(x: x, b0: 0.9622, b1: 0, b2: -0.9622, a1: 0,
         a2:-0.9844)
4         result = bquad(x: result, b0: 1, b1: 1.618, b2: 1, a1: 1.6054,
         a2: 0.9844)
5         result = bquad(x: result, b0: 1, b1: -1.618, b2: 1, a1:
         -1.6054, a2: 0.9844)
6         result = bquad(x: result, b0: 1, b1: 0.618, b2: 1, a1: 0.6132,
         a2: 0.9844)
7         result = bquad(x: result, b0: 1, b1: -0.618, b2: 1, a1:
         -0.6132, a2: 0.9844)
8         return result
9    }
```

```
10
11   private func bquad(x: [Double], b0: Double, b1: Double, b2:Double,
    a1: Double, a2: Double)->[Double]{
12        var y : [Double] = []
13
14        var x_2 : Double = 0
15        var x_1 : Double = 0
16        var y_2 : Double = 0
17        var y_1 : Double = 0
18
19        for i in 0 ..< x.count {
20             let part1 = b0 * x[i] + b1 * x_1 + b2 * x_2
21             let part2 = a1 * y_1 + a2 * y_2
22             y.append(part1 - part2) // IIR difference equation
23             x_2 = x_1           // shift delayed x, y samples
24             x_1 = x[i]
25             y_2 = y_1
26             y_1 = y[i]
27        }
28        return y
29   }
```

- **media mobile**: è un filtro che ad ogni elemento sostituisce il valore della media nel suo intorno, la cui ampiezza è uguale alla finestra ricevuta come parametro. Nella nuova versione la funzione è stata completamente cambiata, rendendola più efficiente ed efficace.

```
1    func mediaMobile(segnale: [Double], finestra: Int, ppg:
    Bool)->[Double]{
2         if (finestra <= 0){
3            return segnale
4         }
5
6         var temp : Double = 0
7         let medium : Int = finestra / 2
8         var count = 0
9         var output : [Double] = []
10        var j = 0
11        for i in 0 ..< segnale.count {
12            j = (i-medium)
13            var sum = i+medium
14
15            while j<sum {
16                if j >= 0 && j < segnale.count {
```

```
17                  temp += segnale[j]
18                  count+=1
19              }
20           j+=1
21          }
22
23       output.append(temp / Double(count))
24       temp = 0
25       count = 0
26       }
27       return output
28   }
```

La funzione wrapper *filterEcg*, inizialmente applicherà il *notchFilter*, dopo eseguirà 20 volte la *mediaMobile*, con una finestra che varia da 10 fino a 2 elementi. Infine, il segnale viene amplificato moltiplicando ogni elemento per due e alzando il suo valore sommando 1,4. Di seguito viene riportato il codice di *filterEcg*:

```
1   func filterECG(input: [Double]) -> [Double]{
2        var output = notchFilter_10(x: input)
3        output = mediaMobile(segnale: output, finestra: 10, ppg: false)
4        output = mediaMobile(segnale: output, finestra: 5, ppg: false)
5        var i=4
6        var j=0
7        while (i>1) {
8           while (j<i*2) {
9              output = mediaMobile(segnale: output, finestra: i, ppg:
               false)
10             j+=1
11          }
12          i=i-1
13          j=0
14       }
15       for i in 0 ..< output.count {
16       output[i] = 2 * output[i] + 1.4
17       }
18       return output
19  }
```

Filtri PPG

Nel segnale PPG, nel rosso come nell'infrarosso, avviene un pre-processamento. Occorre eliminare l'influenza dell'hardware sul segnale. Per questa fase viene utilizzata la funzione *inversioneAsseY* che calcola il valore medio sull'intero segnale ed ricalcola ogni elemento come differenza tra il fattore 2*valoreMedio e il valore originale. In seguito verranno applicati i seguenti filtri:

- **mediaMobile**: viene applicato per tre volte la stessa funzione descritta nella sezione *filtri ECG*, con finestre rispettivamente di 40, 40 e 60.
- **detrending**: con questo filtro si portano tutti i valori ad un certo "livello": l'onda, con il passare del tempo, tende ad aumentare i propri valori. Con il detrending si posizionano i massimi e minimi locali circa sulla stessa linea. In sostanza, si calcola il vettore della media mobile con una finestra pari a 1200 ed il valore medio complessivo del segnale. Infine, ad ogni elemento, viene sottratto il corrispettivo nel vettore della media mobile e aggiunto il valore medio complessivo. Nella vecchia versione si presentava un problema sulla lentezza della media mobile con una finestra di 1200 elementi, risolto con la sua nuova versione efficiente.

```
1   func detrending(segnale: [Double])->[Double] {
2       let meanedSig : [Double] = mediaMobile(segnale: segnale,
       finestra: 1200, ppg: true)
3       let mean = segnale.avg()
4       var risultato : [Double] = []
5
6       for i in 0 ..< segnale.count {
7          risultato.append(segnale[i]-meanedSig[i]+mean)
8       }
9       return risultato
10  }
```

Calcolo della saturazione

Il valore della saturazione del sangue (SpO_2) viene derivato dalle componenti *Red* e *Infrared* del PPG. Dato che le due luci hanno frequenze diverse, penetreranno dentro al polpastrello in modo differente, generando due segnali diversi. Le tecniche adottate in questa parte sono state completamente rivoluzionate a confronto della vecchia versione, migliorando sensibilmente la precisione del valore di SpO_2.

La funzione *calcolaSpO2* ricevendo i due segnali già filtrati, calcola, tramite il metodo *calcolaR()*, i massimi e i minimi locali per ogni segnale, e li salva in due array separati. Inoltre, ciclando su tutti gli elementi, calcola il rapporto tra (massimo-minimo)/minimo per ogni indice del vettore, salvando il quoziente in un altra lista. I due array dei rapporti servono per calcolare il valore di SpO_2.

Questo procedimento viene ripetuto svariate volte modificando ad ogni iterazione la minima distanza che possono avere due massimi/minimi, tenendo traccia di tutti i vettori di SpO_2 con le relative deviazioni standard associate. Infine verrà scelto il vettore di SpO_2 con deviazione standard minore, calcolando la media dei suoi valori.

Di seguito viene riportato il codice delle funzioni *calcolaSpO2()* e *calcolaR()*:

```
1   func calcolaSpO2(filteredPPG_IR: [Double], filteredPPG_RED:
   [Double])->Int{
2         var allLocalsSpo2 : [Int:[Double]] = [:]
3         var deviazioneStandard : [Int:Double] = [:]
4         var minSeparation = Constants.MIN_SEPARATION_START
5         while (minSeparation<=Constants.MIN_SEPARATION_END){
6
7              let R_RED : [Double] = calcolaR(PPGIR_or_RED:
              filteredPPG_RED, minSeparation: minSeparation)
8              let R_IR : [Double] = calcolaR(PPGIR_or_RED:
              filteredPPG_IR, minSeparation: minSeparation)
9              var vetSpO2 : [Double] = []
10
11             var i = 0
12             while (i<R_RED.count) && (i<R_IR.count) {
13                     let Rtot_i = R_RED[i] / R_IR[i]
14                     vetSpO2.append( 104 - (17 * Rtot_i) )
15                     i = i + 1
16             }
```

```
17            let index =
              (minSeparation-Constants.MIN_SEPARATION_START)/10
18            allLocalsSpo2[index] = vetSpO2
19            deviazioneStandard[index] = deviationStandard(vett:
              vetSpO2)
20            minSeparation = minSeparation + 10
21
22        }
23
24        var minSd : Double = 100
25        var minIndex : Int = 0
26
27        //find the lower standard deviation
28        for i in 0 ..< deviazioneStandard.keys.count {
29            let sd : Double? = deviazioneStandard[i]
30            if (minSd > sd!) {
31                minSd = sd!;
32                minIndex = i;
33            }
34        }
35
36        //if the lower found is however high, the misuration is not
            good
37        if (minSd > 10){
38            return -1;
39        }
40
41        var spo2 : Int = Int(round(allLocalsSpo2[minIndex]!.avg()))
42
43        if (spo2 > 99){
44            spo2 = 99
45        }
46
47        return spo2;
48    }
49
50    private func calcolaR(PPGIR_or_RED: [Double],minSeparation:
      Int)->[Double]{
51            var checkMax = true
52            var checkMin = true
53            var firstMaxIndex = -1
54            var firstMinIndex = -1
55            var currentMaxIndex = 0
56            var currentMinIndex = 0
57            var max : [Double] = []
58            var min : [Double] = []
59            var R : [Double] = []
60
61            //calcolo vettore dei massimi e vettore dei minimi
              corrisponendi
62            for i in 0 ..< PPGIR_or_RED.count {
63               if (checkMax &&  i - currentMaxIndex >= minSeparation){
```

```
64                if controlMaxorMin(ppgR: PPGIR_or_RED, i:
                  i,maxMin:true){
65                    max.append(PPGIR_or_RED[i])
66                    checkMax = false
67                    checkMin = true
68                    currentMaxIndex = i
69                    if (firstMaxIndex == -1){
70                       firstMaxIndex = currentMaxIndex
71                    }
72                 }
73              }
74              if (checkMin && i - currentMaxIndex >= minSeparation) {
75                 if controlMaxorMin(ppgR: PPGIR_or_RED, i: i, maxMin:
                  false){
76                    min.append(PPGIR_or_RED[i])
77                    checkMax = true
78                    checkMin = false
79                    currentMinIndex = i
80                    if (firstMinIndex == -1){
81                       firstMinIndex = currentMinIndex
82                    }
83                 }
84              }
85           }
86           //calcolo vettore R = AC/DC = (Max - Min)/Min
87           var i = 0
88           let firstMax = firstMaxIndex < firstMinIndex
89           while (firstMax) && (i<min.count) && (i<max.count) {
90               let temp = (max[i] - min[i]) / min[i]
91               R.append(temp)
92               i = i + 1
93           }
94           while (!firstMax) && (i<min.count-1) && (i<max.count) {
95               let temp = (max[i] - min[i+1]) / min[i+1]
96               R.append(temp)
97               i = i + 1
98           }
99           return R
100 }
```

Determinazione dei picchi

La determinazione dei picchi del segnale ECG è determinante per il calcolo della frequenza cardiaca.

Nella nuova versione dell'app, la funzione *findPeak* è stata completamente rivoluzionata, aumentando la precisione nell'individuazione dei picchi e conseguentemente nel calcolo della frequenza cardiaca. La funzione riceve il segnale ECG filtrato ed esegue, per l'individuazione dei picchi, calcoli prettamente matematici.

```
1   func findPeak(input : [Double], GVDataPeaks : inout
   [ChartDataEntry], absolute_max : Double) {
2       var GVDataPeak : [ChartDataEntry] =
       [ChartDataEntry](repeating: ChartDataEntry(x: 0, y: 0), count:
       4)
3       let length : Int = input.count
4       let interval : Int = 150
5       var maxima : [Int] = [Int](repeating: 0, count: length/200)
6       var temp_max : Int
7       var k : Int = 0
8       var i : Int=1
9       var localMaxs : [Int] = []
10      var localMins : [Int] = []
11
12      while(i < length-1){
13           if (input[i] > input[i + 1] && input[i] > input[i - 1]){
14               localMaxs.append(i);
15           }
16           i+=1;
17      }
18
19      i=1;
20      while (i < length-1) {
21          if (input[i] < input[i + 1] && input[i] < input[i - 1]){
22              localMins.append(i);
23          }
24          i += 1
25      }
26
27      var local_min = -1
28      for local_max in localMaxs{
29          if (local_max < 35 || local_max > length - 35){
30              continue;
31          }
32
33          local_min = -1
34
35          for min in localMins {
36              if(min <= local_max){
37                  continue;
```

```
38              }
39              if(min > local_max + 80){
40                 break;
41              }
42              if(min <= local_max + 40 && input[local_max] -
                input[min] >= 0.3){
44                 local_min = min;
45                 break;
46              }
47              else if(min > local_max + 40){
48                  let minTmp = local_max + ((min - local_max) / 2);
49                  let slope1 : Double = input[local_max] -
                    input[minTmp];
50                  let slope2 : Double = input[minTmp] - input[min];
51                  if (slope1 > slope2 && slope1 >= 0.25) {
52                      local_min = min;
53                      break;
54                  }
55              }
56          }
57
58          if (local_min != -1) {
59             if (k > 0 && local_max - maxima[k - 1] < 200) {
60                //peaks cannot be too close each other
61                let slope_temp : Double = (input[local_max] -
                  input[local_max + 20]);
62                let slope_old : Double = (input[maxima[k - 1]] -
                  input[maxima[k - 1] + 20]);
63                if (slope_temp > slope_old){
64                    maxima[k-1] = local_max
65                    GVDataPeaks[k-1] = ChartDataEntry(x:
                      Double(local_max), y: input[local_max])
66                }
67             }
68             else {
69                maxima[k] = local_max
70                GVDataPeaks[k] = ChartDataEntry(x:
                  Double(local_max), y: input[local_max])
71                k += 1;
72             }
73          }
74       }
75  }
```

Inizialmente si trovano tutti i massimi locali, poi si itera su tutti i massimi per capire se possono essere dei picchi o meno. La tecnica utilizzata sta nel confrontare il massimo corrente con il minimo seguente, e viene fatto in due modi:

- La distanza R-S di norma è inferiore a 80 ms; si cerca nel vettore dei minimi se è presente un valore compreso negli 80 campioni (corrispondenti a 80 ms) successivi al picco e la differenza nell'asse delle ordinate è più grande di 0.3 mV.

- nel caso di distanze superiori agli 80 ms ma inferiori a 160 ms, si confrontano il tratto di discesa tra il massimo e il valore medio tra massimo e minimo, e il tratto tra il valor medio e minimo. Il primo tratto dovrà avere una pendenza maggiore e una ampiezza superiore a 0.25 mV.

Seguendo il procedimento sopra indicato avremo un potenziale picco. Per confermare il suddetto picco si seguono i seguenti criteri:

- Se localmente non ci sono altri picchi e se abbastanza distante dall'ultimo individuato, allora viene confermato come picco.

- Se è presente un altro picco localmente si confrontano le pendenze. Se il nuovo picco ha una pendenza maggiore allora sostituirà il vecchio, ritenuto falso.

Calcolo dell'Heart Beat e fibrillazione atriale

Per un calcolo approssimativo dell'heart beat (**HB**) basta contare i picchi R e moltiplicare per 6, oppure dividere 300 per il numero di quadratini presenti tra due picchi. Ma queste due tecniche non bastano per avere una frequenza cardiaca puntuale. Per avere un heart beat puntuale si è scritto il seguente codice:

```
1   func heartBeatRate(dataPoint : [ChartDataEntry]) -> [Int]{
2      var min : Int = 0
3      var max : Int = 0
4      var sum : Int = 0
5      var RR : Int
6      var i : Int
7      var heart_rate : [Int] = [0, 0, 0, 0]
8
9      //added for FA detection
```

```
10     var FAbpmCounter : Int = 0 //number of FA beats of the entire
      acquisition
11     var previousBpm : Int = 0 // end FA
12     i = 1
13     while dataPoint[i].x != 0 && dataPoint[i].x != dataPoint[i-1].x {
14        if i==1 {
15          sum = Int(dataPoint[i].x) - Int(dataPoint[i-1].x)
16          min = Int(dataPoint[i].x) - Int(dataPoint[i-1].x)
17          max = min;
18          //added for FA detection
19          previousBpm = Constants.HEARTBEAT/min
20          // end FA
21        }
22        else{
23          let sample_frequency = UserDefaults.standard.integer(forKey:
          "sample_frequency")
24          RR =
          Int(dataPoint[i].x-dataPoint[i-1].x)*(1000/sample_frequency)
25          sum = sum + RR
26          if RR < min {
27             min = RR
28          }
29          if RR > max {
30             max = RR
31          }
32          //added for FA detection
33          if (Constants.HEARTBEAT/RR) < previousBpm - 5 ||
          (Constants.HEARTBEAT/RR) > previousBpm + 5 {
34              FAbpmCounter = FAbpmCounter + 1
35          }
36          previousBpm = Constants.HEARTBEAT/RR
37          // end FA
38       }
39       i = i+1
40     }
41       i = i - 1 // restore the last increment (now i is the number of
42      R-R interval)
43       sum = sum / i //average RR time
44       let seconds_of_aquisition =
        UserDefaults.standard.integer(forKey: "AcquisitionSecondsInt")
45       if i<5 || i > seconds_of_aquisition * 4 {// heart rate not
        possible maxBPM = 240
46          heart_rate[0] = -1// signaling error
47          heart_rate[1] = -1// signaling error
48          heart_rate[2] = -1// signaling error
49       }
50       else {
51          heart_rate[0] = Int( Double(Constants.HEARTBEAT)) / sum
52          heart_rate[1] = Constants.HEARTBEAT/max //average beat
53          heart_rate[2] = Constants.HEARTBEAT/min  //average beat
54          //added for FA detection
55          if FAbpmCounter > (i - FAbpmCounter) {
```

```
56                heart_rate[3]=1;
57            }
58            // end FA
59        }
60        return heart_rate;
61    }
```

Il codice riportato si impegna sia nel calcolo dell'HB che nella rilevazione della fibrillazione atriale (**FA**). Per determinare se è presente una FA vengono analizzati sia il battito per minuto che il ritmo complessivo; se le loro variazioni nel tempo superano la soglia di 3 bpm, l'acquisizione viene classificata fibrillante. Infine, nel caso di fibrillazione atriale, vengono analizzate le onde P, che saranno assenti. Tuttavia, alcune persone con segnali fibrillanti possono avere dei tratti con andamento simil-sinusoidale che assomigliano molto alle onde P. Per questa ragione, nella parte finale dell'algoritmo si cercano le onde P per mezzo dei massimi antecedenti il picco R. Quando viene trovata una potenziale onda P, viene analizzata la sua ampiezza, durata e distanza dai segnali QRS per confermare se è una onda P o fibrillatoria.

4.1.7 Predizione della pressione

La classe **NeuralNetworkManager** è una classe **Singleton** designata per l'istanziazione della rete neurale e calcolo della pressione arteriosa sistemica. La classe importa il modello utilizzando la libreria tflite (capitolo 3.3.3) e la gestisce tramite un interprete.

I segnali, per darli come input alla rete neurale, devono essere ricampionati da 500 Hz a 125 Hz e normalizzati sottraendo a ciascun elemento dell'array la media degli elementi divisi per la deviazione standard.

I tensori in input sono vettori tridimensionali 1 x 625 x 2, in cui i vettori di PPG ed ECG si trovano nella terza dimensione rispettivamente nell'indice 0 e 1. Invece, il tensore di output a dimensione 1 x 2.

Infine, una volta calcolate le due componenti della pressione, vengono aggiunti degli offset lineari come descritto nel capitolo 4.1.2.

Di seguito il codice per il calcolo della pressione arteriosa:

```
1  func computePressioni(ECG : [Double], PPG_RED: [Double]){
2     do{
3       ECGBatches = [:]
4       PPG_REDBatches = [:]
5       dataPreprocessing(ECG: ECG, PPG_RED: PPG_RED)
6
7       for i in 0 ..< UserDefaults.standard.integer(forKey:"nBatches"){
8
9           let PPG_ECG_1Batch = PPG_REDBatches[i]! + ECGBatches[i]!
10          let PPG_ECG_1Batch_Float =
           fromDoubleToFloat(batch:PPG_ECG_1Batch)
11          let inputData: Data =
           PPG_ECG_1Batch_Float.withUnsafeBufferPointer {Data(buffer:
           $0)}
12
13          // Copy the input data to the input `Tensor`.
14          try self.interpreter!.copy(inputData, toInputAt: 0)
15          // Run inference by invoking the `Interpreter`.
16          try self.interpreter!.invoke()
17          // Get the output `Tensor`
18          let outputTensor = try self.interpreter!.output(at: 0)
19          // Copy output to `Data` to process the inference results.
20          let outputSize = outputTensor.shape.dimensions.reduce(1, {x,
           y in x * y})
21          let outputData =
           UnsafeMutableBufferPointer<Float32>.allocate(capacity:
           outputSize)
22          outputTensor.data.copyBytes(to: outputData)
23          //salva risultati in presssis e pressdia
24          pressioniSis.append(outputData[0])
25          pressioniDia.append(outputData[1])
26      }
27
28      if pressioniDia.count == 2 {
29          pressSis = Int(pressioniSis.avg())
30          pressDia = Int(pressioniDia.avg())
31      }
32      else if pressioniDia.count > 2 {
33          var IndiceStart = pressioniDia.count/2 - 1
34          var accSis : Float32 = 0
35          var accDia : Float32 = 0
36          for i in IndiceStart ..< (IndiceStart+2){
37                accSis = accSis + pressioniSis[i]
38                accDia = accDia + pressioniDia[i]
39          }
40          pressSis = Int(accSis/2)
41          pressDia = Int(accDia/2)
```

```
42      }
43      pressSis = calibrazioneSis(pressioni: pressSis!)
44      pressDia = calibrazioneDia(pressioni: pressDia!)
45
46    } catch {
47            var e : InterpreterError = error as! InterpreterError
48            print(e.errorDescription)
49    }
50 }
```

Capitolo 5

Privacy e Sperimentazione

Dopo un esito positivo della sperimentazione preliminare dell'app PulsEcg, come vedremo nel capitolo 6, il gruppo di ricerca ha deciso di avviare una sperimentazione legale e strutturata. La finalità della sperimentazione è condurre un'analisi dei dati acquisiti al fine di valutare la precisione e l'affidabilità dei segnali acquisiti, confrontandoli con i segnali provenienti da strumentazione certificata.
Per poter avviare una sperimentazione occorre produrre tutta la documentazione necessaria per garantire la **privacy** dei volontari e avere l'approvazione del **Comitato Etico della Ricerca** (CER) del politecnico di Torino.
La sezione privacy è stata realizzata seguendo il regolamento generale sulla protezione dei dati (o GDPR da *General Data Protection Regulation*), un regolamento dell'Unione Europea (UE) in materia del trattamento dei dati personali entrato in vigore il 24 maggio 2016.
Con questo regolamento si rafforza la protezione dei dati personali dei cittadini dell'Unione Europea, dando loro il controllo dei dati, e si rende omogenea la normativa privacy dentro l'UE[40].
Una delle novità più importanti è la responsabilizzazione dei titolari del trattamento (accountability), ovvero presentare tutti i comportamenti da adottare a dimostrazione dell'applicazione del regolamento del GDPR, e presentare tutti i rischi che comportano l'acquisizione di questi dati con le tecniche per mitigare tali pericoli. Un'altra novità è la nascita della figura del *Data Protection Officer* (DPO), incaricato nel controllare che tutte le disposizioni per la protezione dei dati personali siano applicate nella maniera corretta[41].

5.1 Informativa privacy PulsEcg

Ai fini di pubblicazione dell'app nell'AppleStore e per la sperimentazione, è stata necessaria la creazione dell'**Informativa privacy** per l'applicazione PulsEcg.

La finalità del testo è quella di informare il volontario della tipologia dei segnali che saranno acquisiti tramite il dispositivo, il responsabile del trattamento dei dati, quale scopo avrà la sperimentazione e dove saranno memorizzati i dati acquisiti.
Il documento si suddivide nelle seguenti sezioni:

- **Introduzione**: breve descrizione sulla tipologia dei dati acquisiti e enunciazione del regolamento seguito per il trattamento dei dati.
- **Dati di Contatto**: in questa sezione si riferiscono il titolare del trattamento dati e a quale indirizzo email contattare il DPO.
- **Principi, base giuridica e finalità del trattamento**: parte centrale del documento, si elencano tutti gli atti normativi del GDPR e si descrivono nel dettaglio i dati che saranno acquisiti e la finalità del trattamento.
- **Soggetti autorizzati al trattamento, eventuali destinatari o categorie di destinatari e responsabili del trattamento**: in questa sezione viene specificato che i dati saranno accessibili solo dal gruppo di ricerca del prof. Pasero. Inoltre, i dati potranno essere trasferiti e pubblicati solo in forma aggregata e/o anonima. Infine, i dati personali saranno gestiti e ubicati all'interno dell'Ateneo e/o su archivi esterni di fornitori di alcuni servizi necessari alla gestione tecnico che saranno debitamente nominati come responsabili del trattamento.
- **Trasferimento dei dati**: i dati non saranno trasferiti verso un paese al di fuori dell'UE a meno che l'utente abbia abilitato il backup automatico dei contenuti del proprio dispositivo.
- **Periodo di conservazione dei dati**: i dati personali inerenti il trattamento saranno conservati per tutta la durata della ricerca e al termine della stessa per ulteriori 5 anni.
- **Conferimento dei dati**: per partecipare allo studio è obbligatorio conferire i dati, l'eventuale rifiuto comporta l'impossibilità a partecipare.
- **Processo decisionale automatizzato**: non è previsto.

- **Diritti dell'interessato**: in questa sezione finale vengono elencati i diritti del volontario sui dati personali da lui consegnati conformemente agli art. 15 e ss. del GDPR.

5.2 Comitato Etico per la Ricerca

Il Politecnico di Torino si è dotato di un Comitato Etico per la Ricerca (CER) nel 26/11/2020, con la finalità di stimolare una riflessione etica nei progetti di ricerca scientifica. Esso si pone l'obiettivo di tutelare coloro che vengono coinvolti nei progetti di ricerca e di promuovere una certa qualità scientifica. Per questo motivo l'ente chiede ai ricercatori di dimostrare che la loro ricerca rispetti i valori etici e tuteli i diritti fondamentali[42].
Per queste ragioni, è stato compilato un documento da inviare al CER per la valutazione del rispetto dei principi etici della ricerca e conseguentemente avere il permesso a procedere con la sperimentazione del dispositivo PulsEcg.

5.2.1 Modulo Richiesta

Il **Modulo Richiesta Espressione CER** è un documento precompilato per presentare, in modo standardizzato, il progetto di ricerca. In particolare, verrà chiesto una descrizione dell'oggetto di studio, di come sarà condotta la sperimentazione e con quali tecnologie saranno gestiti e memorizzati i dati acquisiti.
Nella **prima parte** viene chiesto un breve riassunto del progetto di ricerca. In seguito, vengono presentate una serie di tabelle contenenti delle domande riguardanti:

- la partecipazione di essere umani.
- se la ricerca prevede una raccolta dei dati.
- se sono coinvolti degli animali.
- se saranno impiegate sostanze che possono mettere in pericolo la flora e la fauna.

- se la ricerca potrà avere in futuro un impiego militare.
- se la ricerca può essere oggetto di uso scorretto.
- se la ricerca fa uso di tessuti umani.

Questa parte si conclude con una descrizione completa del progetto indicando gli obiettivi primari e secondari, i vantaggi che può portare la buona riuscita della ricerca, una descrizione tecnica del dispositivo, come verrà eseguita la sperimentazione, ed infine le tecniche che saranno impiegate per dimostrare o rigettare le ipotesi enunciate. In particolare, la sperimentazione è impostata nel seguente modo: Il volontario sarà fatto sedere su di una sedia, gli si farà indossare l'oggetto dell'esperimento, un orologio, e si faranno 5 acquisizioni da 10 secondi e 5 acquisizioni da 30 secondi. Gli si faranno domande sul comfort dell'orologio e sulla usabilità dell'applicazione.
Nella **seconda parte** viene generato il **foglio informativo** da consegnare al volontario per informarlo dello scopo dello studio, su come si svolgerà la sperimentazione, il motivo della sperimentazione, i benefici derivanti dallo studio, come viene garantita la sicurezza dei dati acquisiti ed i possibili rischi della sperimentazione.
La **terza parte** è composta dal documento dell'informativa di consenso per il trattamento dei dati personali, come descritto nel capitolo 5.1.
Infine, la **quarta e ultima parte** contiene il documento del *Data Management Plan* (DMP). Nel DMP vengono elencate le motivazioni della raccolta dati, le tipologie di dati raccolti, lo spazio stimato di occupazione e la sicurezza dello storage.
In particolare, i dati acquisiti saranno memorizzati in un server interno al Politecnico accessibile solo tramite credenziali autenticate e per mitigare il rischio di perdita dei dati a seguito di eventuali malfunzionamenti dei server, si provvederà a fare dei backup periodici dei dati su dispositivi esterni crittografati. Inoltre, per una maggiore sicurezza sulla privacy i dati riguardanti un volontario verranno anonimizzati. Il responsabile di ricerca predispone, per ciascun membro del team di ricerca, opportuni permessi di lettura/scrittura alle cartelle del server relative alla ricerca in oggetto.

Capitolo 6

Test e analisi risultati

6.1 Confronto con i dispositivi certificati

Nell'ultima parte del lavoro di tesi abbiamo condotto una campagna acquisizioni formata da 4 volontari. Lo scopo del test è di verificare la qualità dei segnali ottenuti con la nuova versione dell'app. In particolare, vengono esaminate l'efficacia dei nuovi algoritmi di filtraggio e di derivazione della pressione, dell'HB e del valore di SpO_2.

Per verificare quanto appena detto, abbiamo utilizzato vari **strumenti certificati** che siano in grado di emettere le stesse informazioni in maniera affidabile.

Sono stati utilizzati i seguenti strumenti:

- **GE Healthcare B105** (fig. 6.1), uno strumento professionale con il quale è possibile acquisire l'elettrocardiogramma, la pulsossimetria e la pressione sistolica.

Figura 6.1: Il GE Healthcare B105

- **KardiaMobile 6L** (fig. 6.2), dispositivo composto da 3 elettrodi in grado di misurare l'elettrocardiogramma a 6 derivazioni connessa con l'apposita app.

Figura 6.2: Il KardiaMobile 6L

- **Withings ScanWatch** (figura 6.3), uno strumento dalle sembianze di un orologio con cui si può acquisire l'elettrocardiogramma e la pulsossimetria, consultabili con l'app associata.

Figura 6.3: Il Withings ScanWatch

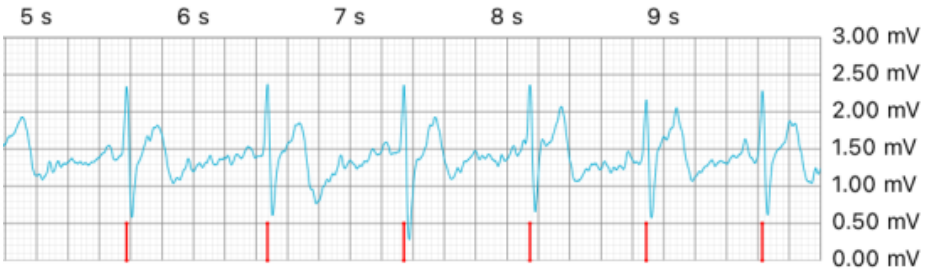
Nei prossimi capitoli confronteremo i risultati ottenuti con l'app PulsEcg con i risultati degli strumenti appena presentati.

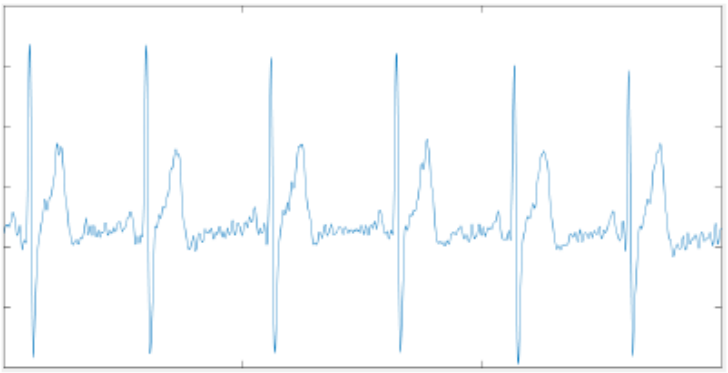
6.1.1 Confronto ECG e frequenza cardiaca

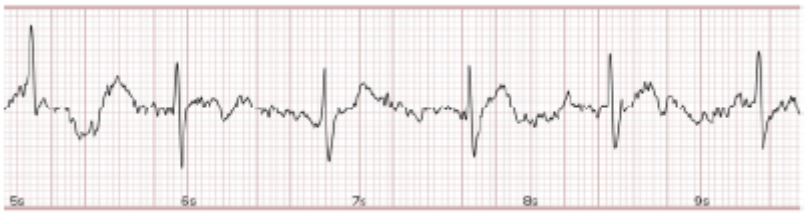
La parte di ECG è sicuramente il segnale più delicato da verificare, dato che gli elettrodi sono soggetti al rumore occorre progettare dei filtri efficaci.
Per ciascuna delle 4 persone, si riportano in ordine le misurazioni eseguite con PulsEcg, GE Healthcare B105, KardiaMobile 6L, Withings ScanWatch.

In particolare, l'acquisizione con PulsEcg e GE è stata eseguita contemporaneamente, mentre con gli strumenti successivi un istante immediatamente dopo.

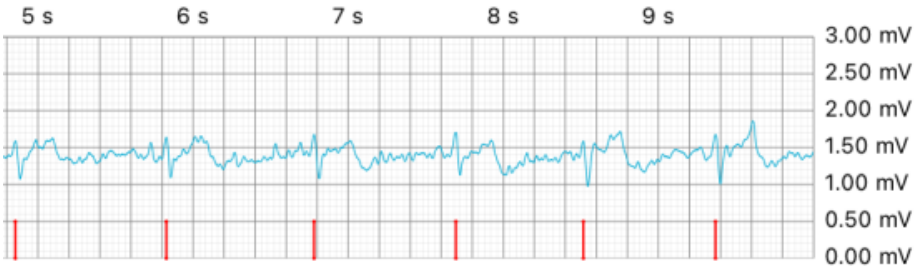
- Primo paziente, uomo di 24 anni:

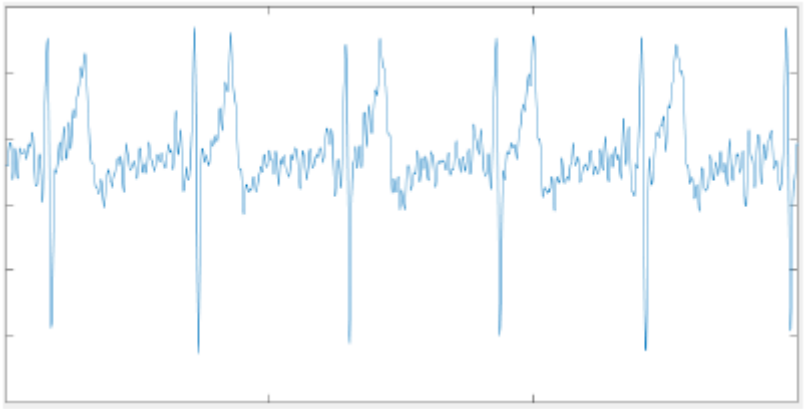


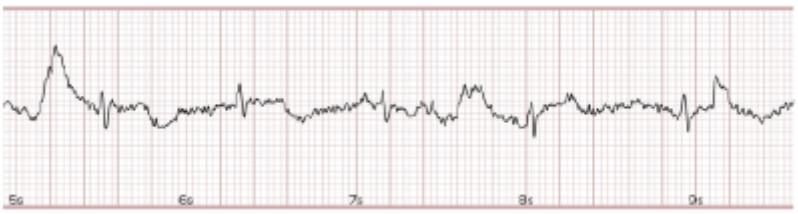




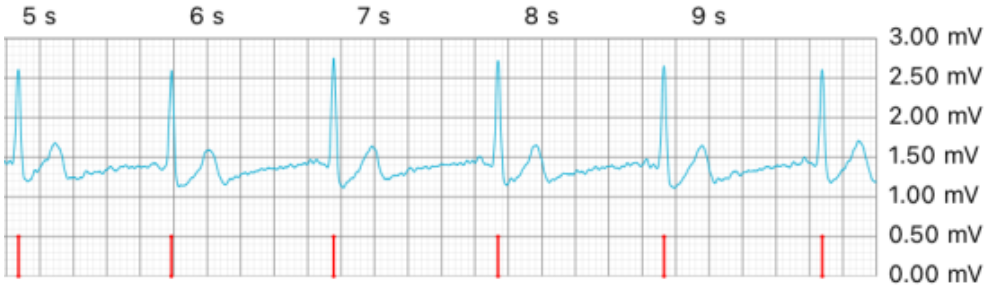
- Secondo paziente, uomo di 25 anni:

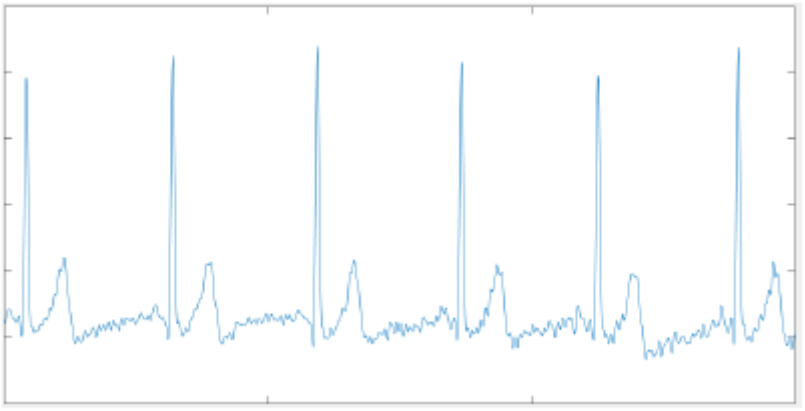


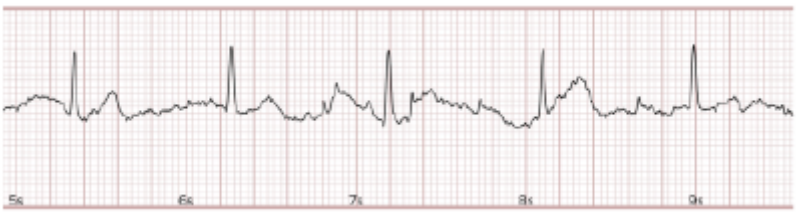




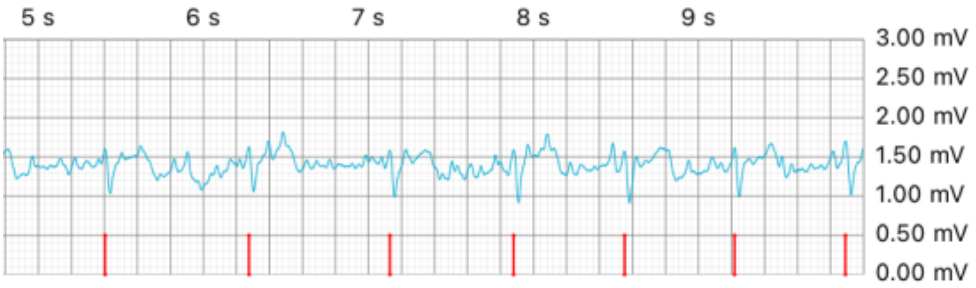
- Terzo paziente, uomo di 32 anni:

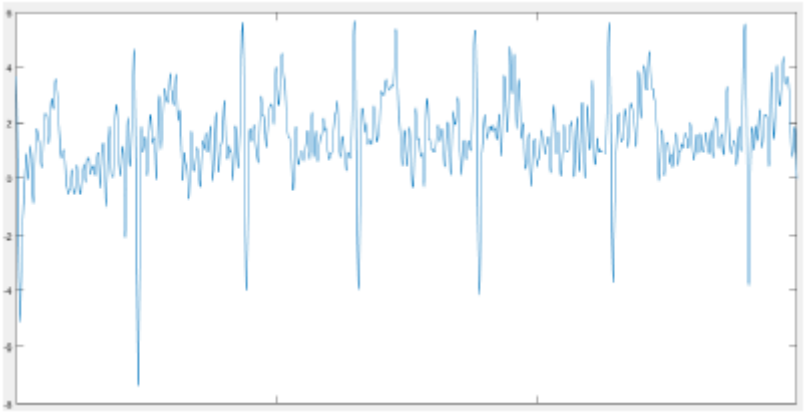


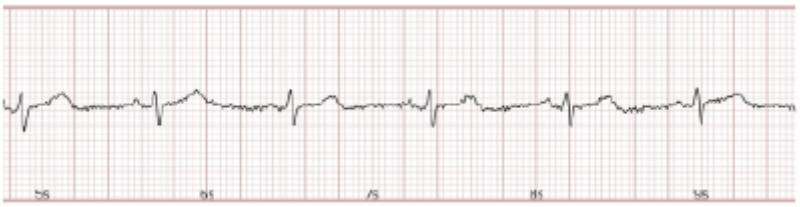




- Quarto paziente, donna 27 anni:







Come possiamo vedere dalle immagini appena presentate, i pazienti con un ECG ampio hanno una correlazione molto puntuale. Invece, per i pazienti con un ECG “piatto”, in particolare il paziente 1 e 4, il segnale del dispositivo ECG è sicuramente da migliorare.

In seguito, in tabella 6.1, vengono riportati i valori della frequenza cardiaca. Come possiamo vedere c’è una buona correlazione tra il valore calcolato con PulsEcg e GE Healthcare B105. Di conseguenza, possiamo affermare che il nuovo algoritmo per il calcolo dell’HB, descritto nel capitolo 4.1.6, produca un valore corretto. La tabella 6.2, riporta rispettivamente la media e la varianza della differenza tra i risultati dei due dispositivi.

Paziente	Sesso	Età	BPM PulsECG	BPM GE B105
I	Uomo	24	73 74 80	74 74 79
II	Uomo	25	65 65 70 65 68 68	65 69 72 68 69 71
III	Uomo	32	67 71 70 74	70 67 69 74
IV	Donna	27	68 67 73	70 70 72

Tabella 6.1: Confronto risultati BPM con GE B105

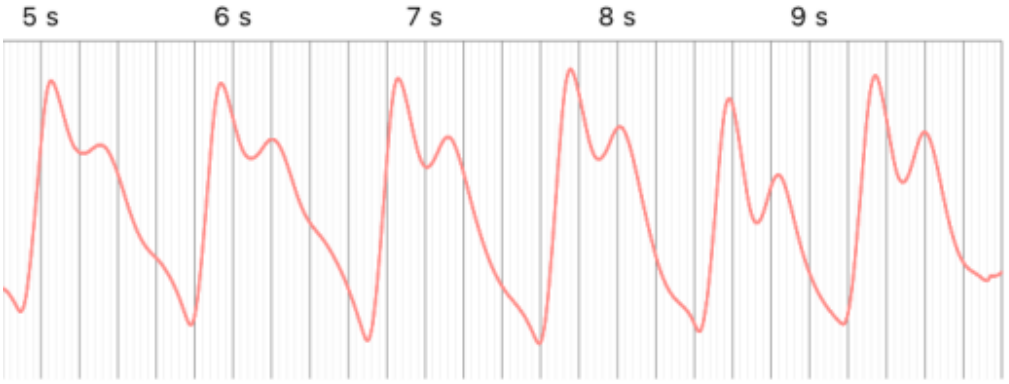
Paziente	Media	Varianza
I	0	0,67
II	2,17	1,81
III	0,5	6,25
IV	1,33	2,89
Totale	0,94	4,18

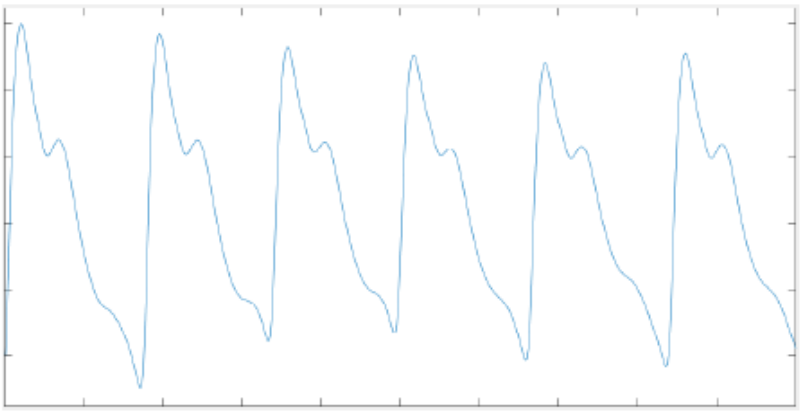
Tabella 6.2: Media e varianza della differenza tra i BPM misurati con PulsEcg e GE B105

6.1.2 Confronto PPG e saturazione

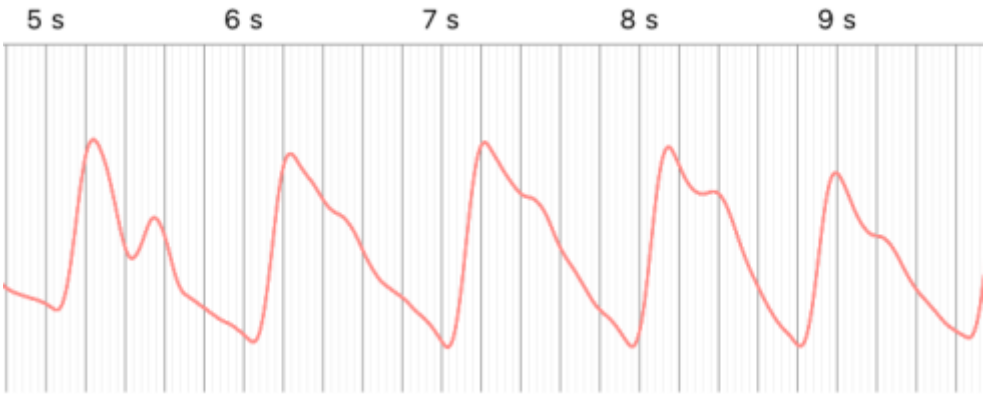
Il campione di volontari è uguale al paragrafo precedente, e per ciascuno vengono confrontati i segnali di PPG tra PulsEcg e GE B105:

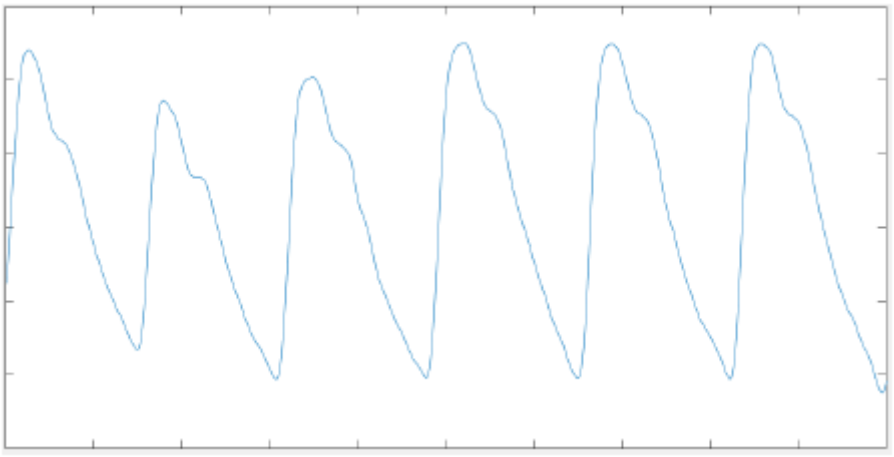
- Primo paziente: uomo di 24 anni:



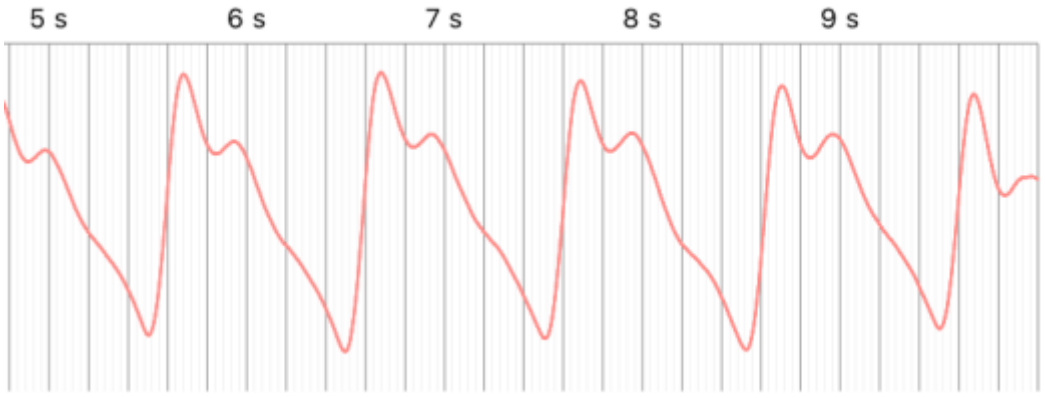


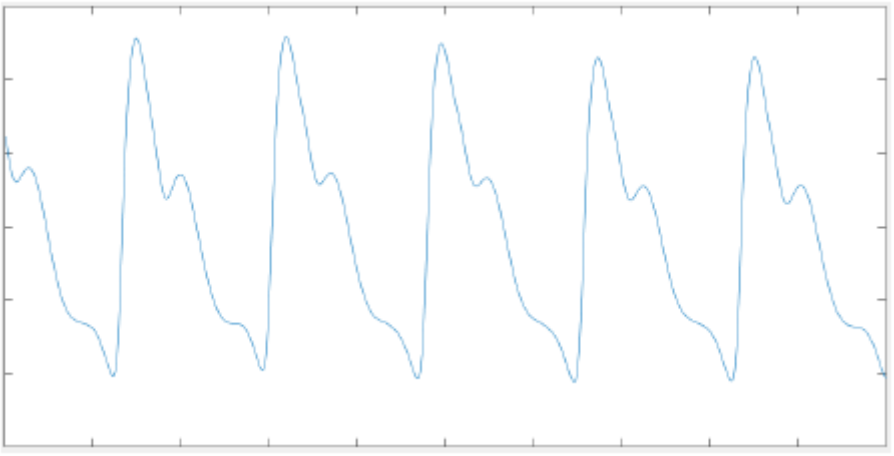
- Secondo paziente, uomo di 25 anni:



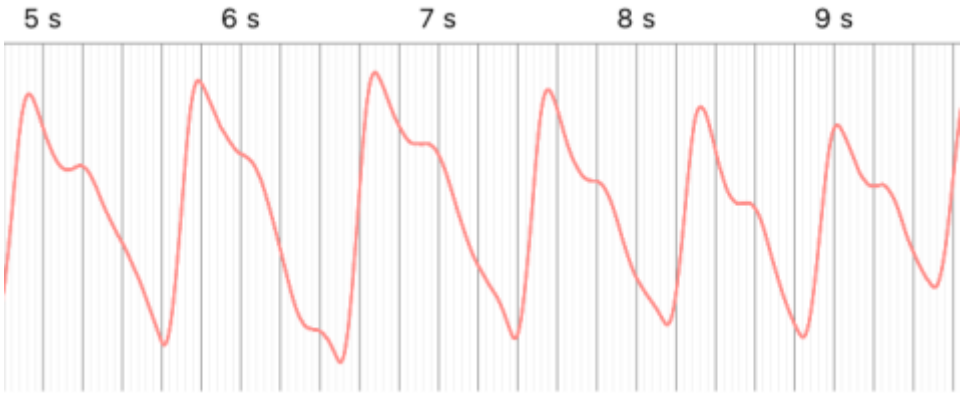


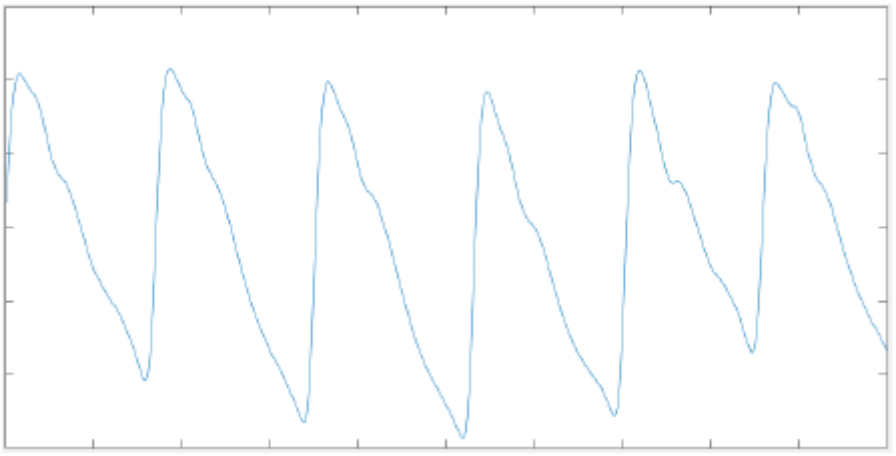
- Terzo paziente, uomo di 32 anni:





- Quarto paziente, donna di 27 anni:





I segnali tra di loro sono molto simili, in particolare possiamo vedere che ci sono pazienti con il piccolo diastolico poco accentuato e molto accentuato, caratteristiche rispettivamente concorde con i segnali del dispositivo GE B105. Invece, una differenza che possiamo notare è l'ampiezza del segnale PPG, il segnale PulsEcg è minore rispetto al GE. Questa differenza può influire la stima della saturazione del sangue, i cui risultati sono riportati in tabella 6.3, invece la media e la varianza della differenza in tabella 6.4.

Paziente	Sesso	Età	SpO_2 PulsECG	SpO_2 GE B105
I	Uomo	24	97% 97% 97%	97% 97% 98%
II	Uomo	25	95% 99% 93% 96% 97% 97%	97% 98% 98% 99% 98% 98%
III	Uomo	32	97% 96% 97% 98%	98% 97% 98% 98%
IV	Donna	27	98% 97% 96%	98% 98% 98%

Tabella 6.3: Confronto risultati SpO_2 con GE B105

Paziente	Media	Varianza
I	0,33	0,22
II	1,83	3,47
III	0,75	0,19
IV	1	0,67
Totale	1,13	1,86

Tabella 6.4: Media e varianza della differenza tra gli SpO_2 misurati con PulsEcg e GE B105

Come possiamo vedere dalle tabelle i risultati sono concordi con la strumentazione certificata. In genere la differenza è tra l'1%-2% tranne per il secondo paziente, su cui possiamo notare, per alcune acquisizioni, un discostamento del 3% fino al 5%. Questo è dovuto al fatto che il sensore su alcuni pazienti non riesce ad acquisire un buon segnale, a prescindere dall'età e dal sesso; le motivazioni non sono state affrontate nel lavoro di questa tesi e potrebbero essere oggetto per sviluppi futuri.

6.1.3 Confronto pressione

Di seguito vengono riportate le predizioni delle pressioni arteriose derivate dalla rete neurale confrontate con le acquisizioni dello strumento GE B105. Gli input della rete neurale sono gli ECG e PPG delle acquisizioni precedentemente presentate. Dai risultati della tabella 6.5 possiamo notare che le misurazioni che si discostano maggiormente dai valori acquisiti dal GE B105, hanno dei segnali ECG/PPG che non rispecchiano tanto la realtà. Inoltre, prima di registrare i valori di pressione arteriosa, è stata eseguita per ogni paziente la calibrazione personalizzata, affrontata nel capitolo 4.1.2 paragrafo GeneralShow, confermando l'utilità di questa procedura. Infine, in tabella 6.6 vengono riportate la media e la varianza della differenza dei risultati.

Paziente	Sesso	Età	Predizione	GE B105
I	Uomo	24	110 / 60 112 / 64	110 / 63 111 / 61
II	Uomo	25	113 / 64 115 / 77 122 / 73 114 / 72 110 / 76	117 / 74 112 / 72 119 / 74 120 / 74 111 / 69
III	Uomo	32	117 / 79 129 / 86 118 / 77 126 / 72	124 / 79 126 / 78 131 / 77 123 / 76
IV	Donna	27	103 / 61 107 / 57	92 / 58 91 / 59

Tabella 6.5: Risultati acquisizione pressione con GE B105

Paziente	Media	Varianza
I	0,5 / 0	0,25 / 9
II	1 / 0,2	13,2 / 35,76
III	3,5 / 1	46,75 / 19
IV	13,5 / 0,5	6,25 / 6,25
Totale	0,69 / 0,31	52,21 / 22,21

Tabella 6.6: Media e varianza della differenza tra le pressioni arteriose misurate con PulsEcg e GE B105

Capitolo 7

Conclusione

Possiamo riassumere le varie modifiche apportate nella nuova versione dell'app PulsEcg nei seguenti punti:

- Precisione ed efficienza delle funzioni per l'elaborazione dei segnali: migliore precisione del valore della saturazione di ossigeno periferica (Spo_2), incrementata l'efficienza della media mobile e la puntualità nell'individuazione dei picchi nel segnale ECG per un più accurato numero di battiti cardiaci.

- Tempo di acquisizione incrementato: è stato aumentato a 30 secondi per avere una più grande panoramica del segnale ECG. Questa estensione temporale aumenta la possibilità di individuare una fibrillazione atriale. Inoltre, sono stati aggiunti ulteriori 4 secondi extra, 3 secondi all'inizio e 1 secondo alla fine dell'acquisizione; questa scelta ha il vantaggio di tagliare eventuali errori o imprecisioni di posizionamento del dito da parte del paziente. Questa scelta permette un miglioramento di presentazione e analisi del segnale.

- Aggiunta di elementi grafici per migliorare la user experience: un'icona raffigurante lo stato di batteria del dispositivo, l'aggiunta di sezioni indicanti i valori di pressione sistolica e diastolica con pop-up per la calibrazione della rete neurale, un pop-up per la privacy nella schermata iniziale e l'aggiunta di uno switch per scegliere una acquisizione da 10 o 30 secondi.

- Integrazione di una rete neurale per la misurazione della pressione arteriosa.

- Miglioramenti sul PDF della misurazione: valori fissi nell'asse delle ordinate per l'ECG e adattivi per il segnale PPG, maggiore definizione della griglia del grafico e maggior risalto sul colore dei picchi nel grafico dell'ECG.

- Generazione documenti necessari per la sperimentazione: documento della privacy dell'app, il consenso informato e il documento per il comitato etico per la ricerca.

In conclusione, con l'affinamento degli algoritmi, l'aggiunta delle nuove funzionalità, gli elementi grafici e la rete neurale, possiamo considerare l'app PulsEcg un prodotto completo. Inoltre, confrontando le prestazioni del dispositivo con la strumentazione certificata, abbiamo dimostrato una buona precisione dei parametri vitali acquisiti. Rimane il proseguimento della sperimentazione, un'analisi dei risultati per individuare possibili sviluppi dell'app, e la conclusione della pubblicazione dell'applicazione nell'Apple Store.
L'interesse del progetto è il superamento dell'utilizzo dello sfigmomanometro che è stato raggiunto quasi pienamente. Un possibile miglioramento è l'aumento della precisione della predizione della rete neurale artificiale. Inoltre, l'aggiunta di sensori di temperatura e di movimento (per riconoscere una eventuale caduta) potrebbero rendere il sistema PulsEcg più funzionale.

Bibliografia

[1]Ministero della Salute. URL: *https://www.salute.gov.it/portale/donna/dettaglioContenutiDonna.jsp?area=Salut%20donna&id=4490&menu=patologie.* (cit. a p. 1)

[2] R.E. Phillips e M.K. Feeney, I ritmi cardiaci - Guida sistematica all'interpretazione, a cura di E. Papini, 2ª ed., Roma, Verduci Editore, 1983, ISBN 978-88-7620-007-6. (cit. a p. 1)

[3] S. Scheidt, Basi teoriche dell'Elettrocardiografia - ECG, traduzione di M.Carnovali, illustrazioni di Frank H. Netter, 5ª ed., Milano, CIBA-GEIGY Italia, 1992 [1991]. (cit. a p. 2)

[4] IPASVI Enna, L'Elettrocardiogramma (PDF), su ipasvienna.it, pp. 29-32.
(cit. a p. 3)

[5] January CT et al., 2014 AHA/ACC/HR, guideline for the management of patients with atrial fibrillation: executive summary: a report of the American College of Cardiology/American Heart Association Task Force on Practice Guidelines and the Heart Rhythm Society, in J Am Coll Cardiol, n. 64, 2014, pp.2246-2280. (cit. a p. 4)

[6] Stewart S et al., Population prevalence, incidence, and predictors of atrial fibrillation in the Renfrew/Paisley study, in Heart, n. 86, 2001, pp. 516–521, DOI:10.1136/heart.86.5.516, PMID 11602543. (cit. a p. 4)

[7] Principles of pulse oximetry, su Anaesthesia UK, 11 settembre 2004. URL consultato il 24 febbraio 2015 (archiviato dall'url originale il 24 febbraio 2015).
(cit. a p. 5)

[8] J. Allen, Photoplethysmography and its application in clinical physiological measurement, Physiological Measurement, vol. 28(3), pp. R1-39, 2007.(cit. a p. 5)

[9] Elgendi Mohamed. «On the analysis of fingertip photoplethysmogram signals». In: Current cardiology reviews vol. 8,1 (2012). doi: https://doi.org/10. 2174/157340312801215782 (cit. a p. 5).

[10] Diseases and conditions index - hypotension, su nhlbi.nih.gov, National Heart Lung and Blood Institute, settembre 2008. URL consultato il 16 settembre 2008. (cit. a p. 6)

[11] O'Brien E, Fitzgerald D. The history of indirect blood pressure measurement. In: O'Brien E, O'Malley K, eds.Blood pressure measurement. Handbook of hypertension. Amsterdam: Elsevier, 1991:1-54. (cit. a p. 6)

[12] Avanzolini G. Strumentazione Biomedica. Progetto e impiego dei sistemi di misura. Pàtron, 1998. (cit. a p. 6)

[13] DI TORE, STEFANO, TODINO, MICHELE DOMENICO, PLUTINO, ANTONINA (2019). Le wearable technologies e la metafora dei sei cappelli per pensare a supporto del seamless learning. PROFESSIONALITÀ, vol.Numero 4/II – 2019, p. 118-132, ISSN 0392-2790. (cit. a p. 7)

[14] A. Bansal and R. Joshi, "Portable out-of-hospital electrocardiography: A review of current technologies", Journal of Arrhythmia, vol. 34, no. 2, pp.129-138, December 2017. (cit. a p. 8)

[15] Withings. ECG Move. URL: https://www.withings.com/uk/en/move-ecg.
(cit. a p. 8)

[16] Apple. ECG Watch. URL: https://support.apple.com/it-it/HT208955.
(cit. a p. 8)

[17] Apple. Swift. URL: https://www.apple.com/it/swift/. (cit. a p. 11)

[18] Apple. Documentation. URL: https://developer.apple.com/documentation/uikit/uiviewcontroller. (cit. a pag. 12)

[19] Apple. Documentation. URL: https://developer.apple.com/library/archive/featuredarticles/ViewControllerPGforiPhoneOS/UsingSegues.html. (cit. a pag. 13)

[20] Apple. Xcode. https://developer.apple.com/xcode/interface-builder/ (cit. a pag. 13)

[21] N, Kajikawa Y. Minami E. Kohno e Y. Kakuda. «On Availability and Energy Consumption of the Fast Connection Establishment Method by Using Bluetooth Classic and Bluetooth Low Energy». In: (2016) (cit. a pag. 14)

[22] Slawomir Jasek. «Gattacking Bluetooth smart devices». In: SecuRing (2017) (cit. a p. 14).

[23] Bibliografia: Renesas. The Attribute Protocol (ATT). URL: http://lpccs-docs.dialog-semiconductor.com/tutorial-custom-profile-DA145xx/att.html. (cit. a p. 16)

[24] Bibliografia: Apple. Documentation. https://developer.apple.com/documentation/corebluetooth (cit. a pag. 16)

[25] Bibliografia: Aasif. How To: Using Notification Center In Swift. https://www.appypie.com/notification-center-how-to-swift. (cit. a p. 17)

[26] Apple. Documetation. UiKit. URL: https://developer.apple.com/documentation/uikit/uiactivityviewcontroller. (cit. a p. 18)

[27] Apple. MessageUI. URL: https://developer.apple.com/documentation/messageui. (cit. a p. 19)

[28] Giacomo Andreucci. Pro iOS Geo. Apress, 2013 (cit. a p. 19)

[29] F. Crick, The recent excitement about neural networks, in Nature, vol. 337, n. 6203, 12 gennaio 1989, DOI:10.1038/337129a0 (cit. a p. 20)

[30] Humayun Kabir. «Convolutional Neural Network». In: (2021). (cit. a p. 21)

[31] Google. Introduzione ai tensori. URL: https://www.tensorflow.org/guide/tensor. (cit. a p. 22).

[32] Yong Yu, Xiaosheng Si, Changhua Hu e Jianxun Zhang. «A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures». In: Neural Computation (2019) 31 (7) (Luglio 2019), pp. 1235–1270. doi: https://doi.org/10.1162/neco_a_01199. (cit. a p. 23).

[33] Keras. Documentation. URL: https://keras.io/api/utils/backend_utils/. (cit. a p. 23)

[34] Vittorio Mazzia. TensorFlow Lite. URL: https://vittoriomazzia.com/tensorflow-lite/. (cit. a p. 26)

[35] Google. Formazione sul dispositivo in TensorFlow Lite. URL: https://www.tensorflow.org/lite/examples/on_device_training/overview. (cit. a p. 27)

[36] Boolean Careers, Front-end, back-end e full-stack web developer: qual è la differenza?, 2020. (cit. a p. 28)

[37] Apple. Documentation. NSMutableAttributedString. URL: https://developer.apple.com/documentation/foundation/nsmutableattributedstring. (cit. a p. 32)

[38] Apple. Documentation. URL. URL: https://developer.apple.com/documentation/foundation/url. (cit. a p. 32)

[39] A. L. Osvalder Y. Liu e M. Karlsson. «Analysis of Signal Noise Reduction by Using Filters». In: (2018). (cit. a p. 57)

[40] Elio Errichiello, Cos'è il GDPR? | Regolamento generale sulla protezione dei dati - Data Protection Law | Privacy e protezione dati personali, in Data Protection Law | Privacy e protezione dati personali. (cit. a p. 71)

[41] Agenda Digitale. GDPR. URL: https://www.agendadigitale.eu/cittadinanza-digitale/gdpr-tutto-cio-che-ce-da-sapere-per-essere-preparati/. (cit. a p. 71)

[42] Politecnico di Torino. Comitato Etico. URL: https://www.polito.it/ricerca/comitato_etico/. (cit. a p. 73)