

ALMA MATER STUDIORUM
UNIVERSITA' DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA
Sede di Forlì

Corso di Laurea in
INGEGNERIA AEROSPAZIALE

Classe L-9

ELABORATO FINALE DI LAUREA

in Elaborazione dati per la navigazione

Progetto e implementazione di un algoritmo per la
navigazione a piedi su un dispositivo Windows Mobile

CANDIDATO

Alessandro Davario

RELATORE

Prof. Matteo Zanzi

CORRELATORE

Prof. Mirko Ravaioli

Anno Accademico 2013/2014

III Sessione di Laurea

Alla mia famiglia

Indice

Introduzione.....	1
1. Calcolo della distanza e principali risultati della tesi di tirocinio.....	3
1.1 Analisi delle applicazioni esistenti.....	4
1.2 Descrizione dell'algoritmo utilizzato.....	6
2. Il Contapassi.....	11
2.1 Stato dell'arte.....	12
2.2 Raccolta dati, relativa analisi e realizzazione del contapassi.....	13
2.3 Filtraggio supplementare dovuto alla frequenza.....	17
3. Introduzione a Windows Phone.....	21
3.1 Sistemi operativi a confronto.....	22
3.2 L'ambiente di sviluppo: Visual Studio for Windows Phone.....	23
4. Sviluppo dell'applicazione.....	31
4.1 Commento del code behind.....	32
4.2 Presentazione dell'applicazione, relativa grafica e funzionalità.....	42
5. Test effettuati e confronti con altre app.....	47
5.1 Principali risultati ottenuti durante i test.....	47
5.2 Applicazioni a confronto.....	50
Conclusioni	55
Bibliografia.....	57

Introduzione

In questi ultimi anni abbiamo assistito ad una vera e propria rivoluzione nel settore tecnologico. Con l'avvento dello smartphone il nostro modo di comunicare è cambiato profondamente.

Prima un cellulare veniva utilizzato solo per inviare messaggi e fare una chiamata. Oggi tutto ciò rappresenta solo la minima parte delle potenzialità messe a disposizione all'utente. Grazie infatti alla molteplicità di sensori presenti e al loro ormai basso costo, chiunque può disporre di un buon dispositivo in grado di fare le più svariate operazioni: una fra tutte la geolocalizzazione.

Infatti sfruttando il ricevitore GPS presente su ogni dispositivo possiamo, tramite un cellulare, determinare la nostra esatta posizione. Tuttavia tale segnale non è così forte da permettere anche la localizzazione all'interno degli edifici. Da qui prende spunto un progetto europeo che prevede la localizzazione indoor tramite un dispositivo elettronico. All'interno di tale progetto si inserisce il mio lavoro.

L'idea che è alla base del mio elaborato è la realizzazione di un'applicazione per smartphone in grado di stimare la distanza percorsa con una certa precisione semplicemente basandosi sulla lettura degli accelerometri presenti. Per far ciò è stato elaborato un algoritmo che calcoli la lunghezza di un passo, basato su dati sperimentali. Inoltre è stato necessario sviluppare anche un semplice contapassi in grado di rilevare i passi compiuti.

Nella prima parte di questo elaborato verrà descritto lo stato attuale della tecnologia a riguardo: verranno prese in esame le varie tipologie di contapassi, verrà richiamato l'algoritmo utilizzato dall'applicazione e commentato l'ambiente di sviluppo di applicazioni per Windows Phone. Nella seconda parte invece verrà analizzato il codice che implementa l'app realizzata e verranno mostrati i principali risultati dei test effettuati comparati con la tecnologia esistente.

Capitolo 1

Calcolo della distanza e principali risultati della tesi di tirocinio

In questi ultimi anni la vita quotidiana di ognuno di noi è profondamente cambiata. Ciò che ha inciso maggiormente su questo aspetto è stata una diffusione a largo spettro degli smartphones.

Essi infatti non sono entrati a far parte solo della vita dei più giovani, ma hanno abbracciato diverse fasce d'età. Data infatti la loro versatilità e la presenza di numerosissime app, ormai con un semplice cellulare è possibile fare ogni cosa si desidera. Ed in un mondo in cui la cura del proprio aspetto fisico per la maggior parte della popolazione è un elemento di primaria importanza, ecco che in brevissimo tempo si è assistiti ad un boom nell'utilizzo di applicazioni per il monitoraggio dell'attività fisica quotidiana. Se in passato era il proprio personal trainer che dettava modalità e tempi degli esercizi, ora basta scaricare un'app che faccia tutto al posto suo.

Uno degli aspetti che maggiormente richiama l'attenzione di uno sportivo è sapere quanta distanza ha percorso durante la propria attività fisica quotidiana. Tuttavia, a differenza di quanto accade per i veicoli dotati di ruote, avere la misura precisa della distanza percorsa non è così banale.

Obiettivo di questo capitolo sarà riassumere tutto il lavoro svolto durante la mia attività di tirocinio, fondamentale per il successivo sviluppo di questa tesi.

1.1 Analisi delle applicazioni esistenti

In questi ultimi anni ha avuto particolare successo l'utilizzo dei pedometri per cellulare, applicazioni in grado di calcolare il numero dei passi effettuati attraverso la lettura dei sensori inerziali (*microelectromechanical systems*), presenti sul proprio dispositivo mobile.

Il principio di funzionamento alla base di tali applicazioni è molto semplice: leggono i dati degli accelerometri, tramite specifici algoritmi rilevano un passo e successivamente incrementano la distanza percorsa. Ma tale distanza percorsa come viene calcolata?

Prendendo in esame una serie di applicazioni presenti negli store dei diversi sistemi operativi (iOS, Android, Windows Phone) si può osservare che:

- la lunghezza del passo figura come un input, o direttamente impostato dall'utente (e in questo caso sono le app stesse che indicano alcuni modi non proprio rigorosi di calcolare la propria lunghezza del passo, ad esempio facendo una media su un certo numero di passi), oppure calcolato dal dispositivo in base all'altezza dell'utente stesso;
- gli output sono il numero dei passi effettuati, la distanza percorsa (ottenuta facendo una semplice operazione di moltiplicazione tra il numero di passi totali e la lunghezza del passo) e velocità media (calcolata anch'essa come rapporto tra distanza totale percorsa e tempo impiegato).

E' ben chiaro quindi da questa analisi che l'applicazione non tiene conto di eventuali accelerazioni o decelerazioni, oppure di variazioni nel tipo di andatura (passo/corsa), che porterebbero a variazioni della lunghezza del passo. Tutto ciò si ripercuote in un errore nel calcolo finale della distanza percorsa. Tale errore può incidere notevolmente e compromettere così tale misurazione.

Per cercare di migliorare tale parametro, alcune applicazioni sono progettate in modo tale da avere come ulteriore informazione il tipo di attività fisica che l'utente decide di intraprendere: passo normale, corsa, jogging. In tal modo, per tutta la durata della sessione, la lunghezza del passo da utilizzare nel calcolo del percorso sarà diversa a seconda dei casi. Tuttavia tale soluzione presenta ovviamente dei limiti, dal momento in cui non viene superato il problema della variazione della frequenza di passo. L'applicazione infatti rimane poco duttile e limita l'utente nel suo effettivo utilizzo. Inoltre mantenere per tutta la durata del percorso la stessa andatura è alquanto improbabile, andando quindi incontro ad inevitabili errori.

Un escamotage che hanno escogitato i programmatori di applicazioni è stato quello di utilizzare l'informazione proveniente dal segnale GPS. Facendo un'integrazione tra la distanza stimata dal GPS e quella calcolata dall'applicazione tramite l'utilizzo del contapassi, si arriva a determinare una distanza percorsa di certo migliore rispetto alla precedente.

Naturalmente esistono dei limiti che rendono tali applicazioni imprecise:

- innanzitutto bisogna considerare la precisione del GPS, che nelle applicazioni civili non è inferiore ai 30 metri. Tutto ciò è dovuto all'accumularsi di una serie di errori tra cui in particolare:
 - imprecisione dell'orologio del satellite e imprecisioni delle effemeridi (1,5 metri);
 - ritardi dovuti alla propagazione del segnale attraverso l'atmosfera (tra i 5 e i 15 metri);
 - ritardo dovuto all'elaborazione del segnale nel satellite (1 metro);
 - cammini multipli, ossia il multipath (tra 1 e i 15 metri);
 - rumore e risoluzione del ricevitore (2 metri);
- nel momento in cui non siamo più all'aperto ma all'interno di un edificio, o il segnale GPS arriva molto degradato al ricevitore, o addirittura vi è la totale assenza di tale segnale. In questo caso l'applicazione lavorerà solo in funzione del contapassi e l'errore sulla distanza calcolata sarà notevole.

Per eliminare tale imprecisione è necessario esprimere la lunghezza del passo in funzione della frequenza del passo e dell'altezza della persona. A tal proposito si è provveduto all'elaborazione di un algoritmo in grado di fare tutto ciò.

1.2 Descrizione dell'algoritmo utilizzato

Il punto di partenza è stata la raccolta di dati sperimentali. Sono stati presi in esame individui di differente altezza e sesso. Ognuno ha percorso lo stesso tracciato prestando attenzione nel mantenere sempre la stessa andatura, evitando cioè improvvise accelerazioni o decelerazioni. In virtù di ciò la posizione da cui partire è stata fissata qualche metro prima dello start in modo tale da arrivare a regime nel punto di partenza e avere così un passo il più regolare possibile. Si è partiti da una camminata molto lenta e ad ogni passaggio la velocità è stata aumentata fino a che nelle ultime misurazioni non si è arrivati alla corsa. Il percorso utilizzato misura 45 metri. Tramite questa informazione e tramite la conoscenza del numero dei passi effettivamente realizzati, è stato possibile risalire alla lunghezza del passo. Inoltre tramite l'utilizzo di un cronometro, è stato calcolato anche il tempo impiegato per percorrere il percorso, da cui ricavare la frequenza di passo.

Sulla base di tali dati è stato possibile costruire l'algoritmo utilizzato nella progettazione della mia applicazione.

Facendo infatti un'interpolazione di secondo grado sui dati sperimentali ottenuti, è possibile scrivere che:

$$lp = af^2 + bf + c \quad (1.1)$$

dove con lp è indicata la lunghezza del passo risultante, con f la frequenza e con a , b e c i coefficienti derivanti dall'interpolazione di secondo grado.

La procedura di interpolazione è stata ripetuta per ogni individuo analizzato.

A questo punto si è proceduto ad una seconda interpolazione di secondo grado, questa volta utilizzando i dati relativi ai vari coefficienti a, b, c, trovati nella prima interpolazione, in funzione dell'altezza h, arrivando quindi a scrivere le seguenti espressioni:

$$a(h) = A_2h^2 + A_1h + A_0 \quad (1.2)$$

$$b(h) = B_2h^2 + B_1h + B_0 \quad (1.3)$$

$$c(h) = C_2h^2 + C_1h + C_0 \quad (1.4)$$

In conclusione è stato formulato l'algoritmo finale che prende in ingresso l'altezza (h) di una persona e la frequenza (f) del passo e in uscita restituisce la lunghezza del passo (lp).

Definendo infatti:

$$H = \begin{pmatrix} h^2 \\ h \\ 1 \end{pmatrix} \quad F = \begin{pmatrix} f^2 \\ f \\ 1 \end{pmatrix}$$

$$M = \begin{pmatrix} A_2 & A_1 & A_0 \\ B_2 & B_1 & B_0 \\ C_2 & C_1 & C_0 \end{pmatrix} \quad \text{che in termini numerici si può esprimere come:}$$

$$M = \begin{pmatrix} 3,009e - 05 & -0,011 & 1,139 \\ -0,00048 & 0,17 & -15,13 \\ 0,00051 & -0,18 & 16,117 \end{pmatrix}$$

dove M rappresenta la matrice dei coefficienti espressi nelle equazioni (1.2), (1.3) e (1.4). È quindi possibile scrivere la seguente espressione in grado di fornire la lunghezza del passo lp:

$$lp = H^T M^T F = (h^2 \quad h \quad 1) \begin{pmatrix} A_2 & B_2 & C_2 \\ A_1 & B_1 & C_1 \\ A_0 & B_0 & C_0 \end{pmatrix} \begin{pmatrix} f^2 \\ f \\ 1 \end{pmatrix}$$

$$lp = f^2 (h^2 A_2 + hA_1 + A_0) + f(h^2 B_2 + hB_1 + B_0) + h^2 C_2 + hC_1 + C_0 \quad (1.5)$$

Facendo un plot dei dati ottenuti, è possibile apprezzare nel seguente grafico l'andamento della lunghezza del passo in funzione della frequenza.

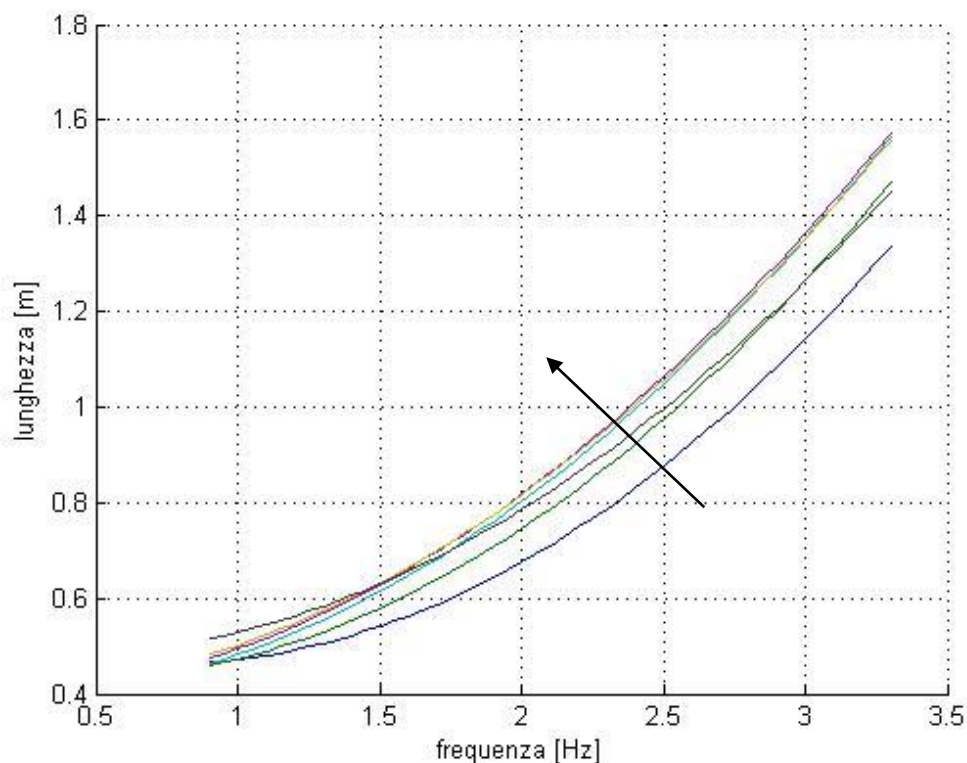


Figura 1.1: andamento della lunghezza del passo, espressa in metri, in funzione della frequenza in Hertz nei diversi soggetti. Il verso della freccia indica che l'altezza aumenta in tale direzione.

Naturalmente tale algoritmo può e deve essere migliorato, facendo altri test, impiegando di volta in volta persone di altezze diverse e di diverso sesso. In tal modo gli errori sulla misurazione del passo possono ridursi notevolmente.

Un obiettivo futuro consiste nel creare due diversi algoritmi in base al sesso dell'utente. Dalle prove sperimentali infatti si è rilevato che vi è una profonda differenza nell'incedere maschile e in quello femminile: in particolare si è notata una maggiore regolarità nell'andatura femminile.

Inoltre si ritiene necessario uno studio più approfondito in uno specifico intervallo di frequenze. Sono state rilevate infatti delle irregolarità nel passaggio dal passo veloce alla corsa. In particolare si è osservata una costanza se non anche una diminuzione della lunghezza del passo nel momento in cui si inizia a correre.

Capitolo 2

Il Contapassi

Da un'analisi svolta dall'OMS (Organizzazione Mondiale della Sanità) [1] si evince che per condurre uno stile di vita sano, rimanere in forma e controllare il proprio peso bisogna muoversi arrivando a fine giornata oltre la soglia di 10.000 passi.

Potrebbe sembrare una cifra non eccessivamente impegnativa; tuttavia è stato provato che dopo una normale giornata casa-lavoro-casa, il conteggio non va oltre i 4.000-5.000 passi.

Naturalmente è impensabile tenere a mente durante l'arco di una giornata il numero di passi effettuati. Ecco che nasce il bisogno di un dispositivo elettronico che si occupa di tutto ciò al posto nostro: il contapassi. Grazie all'utilizzo di tali apparecchi è possibile monitorare la propria attività fisica quotidiana; inoltre rappresentano un ottimo stimolo per raggiungere il traguardo prefissato se non addirittura superarlo.

I contapassi sono di semplice utilizzo: data la diversità di apparecchi, essi possono essere indossati al polso, alla cintura o messi semplicemente in una tasca. Una volta avviati, il conteggio parte automaticamente e non bisogna fare nient'altro se non muoversi.

Dato l'evolversi della tecnologia e il successo che hanno riscontrato nella popolazione, sono stati sviluppati dispositivi nella maggior parte dei casi profondamente differenti. Oltre infatti ai classici pedometri, con l'avvento degli smartphones sono state ideate

applicazioni in grado di calcolare il numero dei passi e non solo: tra le informazioni in grado di monitorare ci sono anche le calorie bruciate e altri parametri vitali come la frequenza cardiaca.

2.1 Stato dell'arte

Sul mercato sono presenti numerosissimi dispositivi in grado di contare i passi. Le tecnologie sfruttate sono diverse, così come le performance.

Prima della diffusione degli smartphones venivano impiegati dei classici contapassi o pedometri da indossare come braccialetti, da mettere in una tasca oppure da applicare alla vita. Inizialmente la rilevazione del passo era effettuata mediante il movimento di una sferetta d'acciaio. Con l'evolversi della tecnologia ormai la maggior parte dei dispositivi è dotata di accelerometri e la precisione nel conteggio è notevolmente aumentata.

Sotto (figura 2.1) è possibile vedere a sinistra un classico pedometro che va attaccato alla cintura, mentre sulla destra un contapassi da polso.



Figura 2.1: a sinistra un pedometro Digiwalker SW-200 (Yamax), mentre a destra il contapassi della Nike FuelBand.

Tuttavia, al giorno d'oggi la maggior parte della popolazione è dotata di uno smartphone. Si è quindi riscontrato negli ultimi anni un incremento nell'utilizzo di applicazioni per contare passi e non solo: tali app infatti sono in grado di monitorare costantemente diversi parametri vitali. Ma il loro vero punto di forza risiede nella stretta correlazione con i social network. Chi vuole può infatti condividere i propri risultati su facebook, ad esempio, e far sapere a tutti gli amici della propria attività fisica. Questo fatto viene visto come un incitamento nel far sempre meglio e l'individuo ne trae così dei benefici.

Tuttavia utilizzare il proprio smartphone per monitorare la propria attività fisica quotidiana ha anche degli aspetti negativi. In particolar modo uno dei problemi più evidenti di queste applicazioni consiste nell'utilizzo di più sensori (accelerometri, gps, ecc.) e nel funzionamento costante anche in background. Tutto ciò porta ad un notevole consumo della batteria. Quindi molti, dopo aver utilizzato il proprio cellulare come contapassi, fanno la scelta di preservare la batteria e tornano ad affidarsi ai più semplici pedometri.

2.2 Raccolta dati, relativa analisi e realizzazione del contapassi

Uno degli obiettivi di questa tesi è stata la realizzazione di un algoritmo che riuscisse a contare il numero di passi effettuati.

L'idea è stata quella di sfruttare la lettura degli accelerometri del cellulare e attraverso una serie di operazioni matematiche, arrivare a determinare il passo.

Per fare tutto ciò è stato quindi necessario capire cosa in realtà accadesse nel momento in cui veniva effettuato un passo. Fondamentale è stato il poter accedere ai dati registrati dagli accelerometri.

L'accelerometro è infatti l'unico sensore di movimento imposto dalle specifiche hardware minime fornite da Microsoft, il cui scopo è quello di restituire informazioni sulle forze che vengono applicate al device, in modo da determinare la direzione in cui esso viene spostato nello spazio.

Nei cellulari sono presenti tre accelerometri, uno per ogni asse, come è possibile vedere dalla figura 2.2.



Figura 2.2: posizione dei tre assi degli accelerometri in un dispositivo mobile.

Per meglio capire i valori restituiti, ecco i valori delle coordinate per gli orientamenti standard:

- Portrait standing (telefono verticale in piedi):
 - X: 0
 - Y: -1
 - Z: 0
- Landscape standing (telefono orizzontale in piedi):
 - X: -1
 - Y: 0
 - Z: 0
- Portrait flat (telefono in verticale appoggiato su una superficie):
 - X: 0
 - Y: 0
 - Z: -1
- Landscape flat (telefono in orizzontale appoggiato su una superficie):
 - X: 0

- Y: 0
- Z: -1

Il primo passo per la realizzazione del contapassi è stato quindi raccogliere i dati letti dagli accelerometri. Per far ciò è stata realizzata una applicazione che leggesse tali dati e successivamente li salvasse in un file da poter leggere al computer. L'intervallo di campionamento è stato impostato su 100 millisecondi.

Una volta realizzata l'applicazione, l'ho personalmente testata sul percorso realizzato nei laboratori didattici dell'Università. Il dispositivo mobile è stato inserito in una tasca e muovendomi ad un passo regolare ho percorso tutto il tracciato, tenendo a mente il numero di passi realmente effettuati.

A questo punto il device è stato collegato al pc e, tramite un apposito programma messo a disposizione dalla Microsoft, è stato possibile accedere al suo Isolated Storage, la parte di memoria dedicata al salvataggio di dati e file delle applicazioni installate. Infatti collegando semplicemente il cellulare al pc si ha accesso solo a file come musica, foto, video o download, ma non alla parte riservata ai file di sistema. Per accedere ai dati delle applicazioni a cui si sta lavorando e che sono installate sul proprio device, la Microsoft mette a disposizione degli sviluppatori un programma chiamato Windows Phone Power Tools.

Successivamente tali dati sono stati importati in Matlab per poter elaborare l'algoritmo che permettesse la determinazione del passo.

Innanzitutto è stato rappresentato, come è possibile vedere nel grafico 2.3, il modulo delle tre accelerazioni in funzione del tempo.

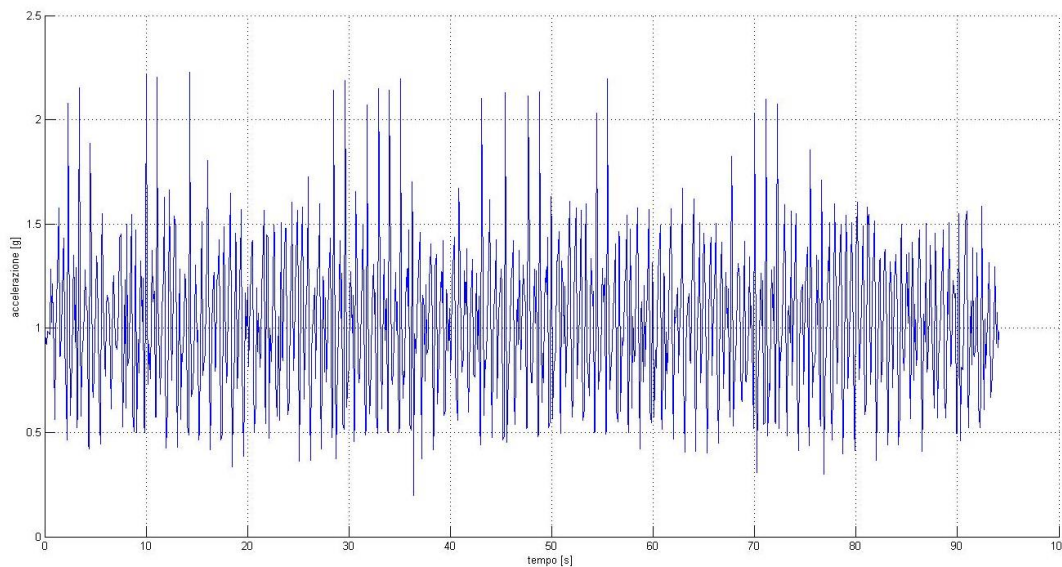


Figura 2.3: andamento del modulo del vettore accelerazione nel tempo. Il tempo viene espresso in secondi, mentre l'accelerazione in g.

Dal grafico è possibile apprezzare la presenza di numerosi picchi, in un numero superiore a quello dei passi effettivi. Ciò è dovuto alla presenza di numerosi falsi positivi, dovuti a movimenti bruschi ed improvvisi che tuttavia non corrispondono nella realtà a veri passi.

L'obiettivo che si pone ogni costruttore di contapassi è proprio quello di eliminare il più possibile il numero di falsi positivi. Per far ciò viene imposto un filtraggio ai dati ottenuti, impostando una determinata soglia.

Per la realizzazione di questo contapassi si è pensato di operare in questo modo:

- Innanzitutto ogni 100 millisecondi vi è una lettura dei tre accelerometri, che restituisce i dati dei tre assi, che per semplicità chiamerò qui x , y , z .
- Successivamente viene calcolato il modulo di questo vettore facendo:

$$\text{modulo} = \sqrt{x^2 + y^2 + z^2} \quad (2.1)$$

- A questo punto viene calcolato il seguente valore, che chiamerò *picco*:

$$\text{picco} = |\text{modulo} - 1| \quad (2.2)$$

- Avendo questo valore del picco ora è possibile fare la differenza tra un picco registrato all'istante di tempo $t+1$ e quello registrato all'istante di tempo t , dato

che questa sarà un'operazione iterativa, attuata ogni 100 millisecondi. Chiamiamo questo valore *diffPicchi*.

- L'ultimo passo è capire quale soglia impostare. Rappresentando in un grafico i valori *diffPicchi* ottenuti dai dati sperimentali si è notata una corrispondenza tra il numero dei passi realmente effettuati e quelli ottenuti impostando come valore di soglia il valore 0,22. Quindi l'ultima operazione di verifica da fare è:
$$passo = (diffPicchi - 0,22) > 0 \quad (2.3)$$

Solo se è verificata tale condizione, l'algoritmo registrerà il passo e procederà ad incrementare un contatore apposito.

Sarebbe a questo punto interessante provare di nuovo tutto il test effettuato su di me, su altri individui di differente altezza, sesso, massa corporea. Sarebbe un ottimo termine di paragone; inoltre sarà utile per capire quanto la nuova soglia trovata si discosti dalla precedente.

2.3 Filtraggio supplementare dovuto alla frequenza

Naturalmente in questo campo la perfezione non è stata ancora raggiunta. Contare il numero effettivo di passi realizzati è l'obiettivo che si pone ogni programmatore, l'interesse finale sarà poi avvicinarsi il più possibile alla realtà. Anche perché bisogna tener conto di un altro fattore: ogni individuo è diverso; diversi sono i movimenti, il tipo di andatura.

Tuttavia nel testare l'algoritmo che fornisce la lunghezza del passo combinato con il contapassi progettato precedentemente, si è riscontrata una singolarità: a causa infatti di alcuni falsi positivi molto ravvicinati (dell'ordine di 100/200 millisecondi) la relativa frequenza calcolata era talmente elevata da dare notevoli errori sulla lunghezza del passo.

Ecco un esempio di falso positivo, rappresentato nella seguente immagine:

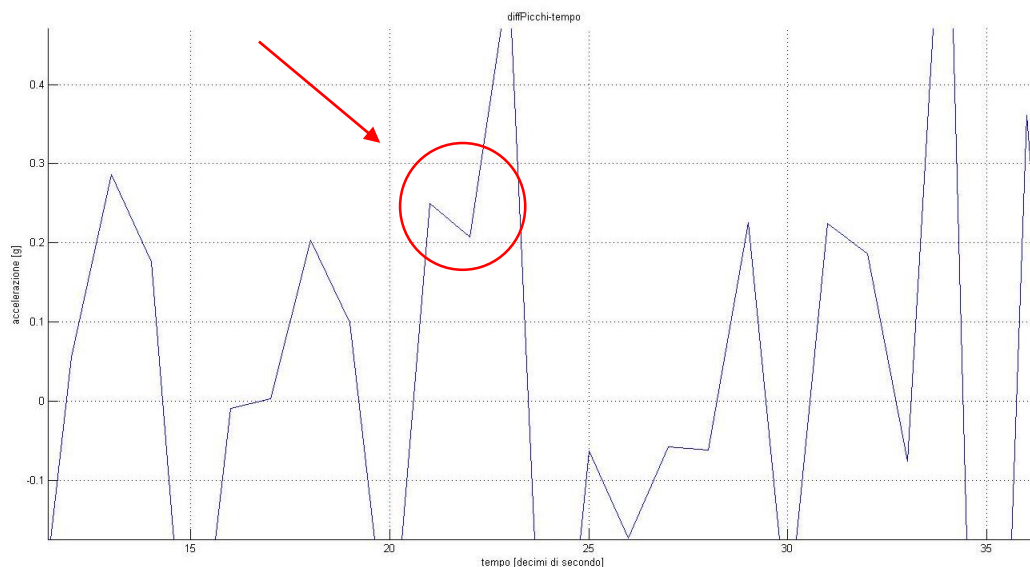


Figura 2.4: esempio di falso positivo.

All'interno della parte evidenziata della figura si può notare la presenza di due punti molto ravvicinati, corrispondenti a due picchi separati da un intervallo di tempo della durata di 100 millisecondi. Tale intervallo di tempo corrisponderebbe ad una frequenza troppo elevata che, applicata all'algoritmo precedentemente esposto, darebbe luogo ad un errore nel calcolo della lunghezza del passo intollerabile.

Da questa osservazione è nata la necessità di applicare un secondo filtro all'algoritmo, tenendo conto della frequenza di passo a cui si sta procedendo, dato da cui non si può prescindere nell'analisi di una determinata andatura.

Per capire quale limite imporre sulla frequenza, mi sono basato sui dati ottenuti sperimentalmente durante il mio tirocinio. Analizzando individui di varie altezze e tipi di marce totalmente differenti, ho ricavato importanti informazioni riguardo la frequenza. Tra i tanti dati ottenuti, ho individuato come limite massimo la frequenza di 3,317 Hz, corrispondente ad un soggetto di altezza pari a 189 cm, alla velocità di 4,665 m/s. Per la mia applicazione, ho impostato una soglia leggermente superiore, pari a 3,5 Hz, in modo tale da avere un certo margine di sicurezza che andrebbe verificato effettuando ulteriori test.

Grazie a questo filtraggio supplementare è stato possibile eliminare la presenza di numerosi falsi positivi, che avrebbero comportato errori sul numero di passi ma soprattutto avrebbero compromesso l'algoritmo relativo al calcolo della lunghezza di passo.

Capitolo 3

Introduzione a Windows Phone

In questi ultimi anni il mercato degli smartphone è stato, senza alcun dubbio, tra i più movimentati nel campo tecnologico.

I primi passi nel campo dei palmari sono stati mossi proprio dalla Microsoft, grazie ai dispositivi Windows Mobile. L'obiettivo era quello di riprodurre, il più fedelmente possibile, l'esperienza che ogni utente aveva con Windows, offrendo così la possibilità di utilizzare gli strumenti di cui più si poteva aver bisogno: Office, agenda, posta elettronica. Tali prodotti erano di ottima fattura, tuttavia suscitavano poco interesse nei confronti dell'utente medio: risultavano infatti ingombranti, poco estetici e scomodi da usare.

Con il passare del tempo però la Apple ha iniziato a sviluppare una nuova generazione di dispositivi mobili più facili da usare e dotati di un device più piacevole da vedere. Tuttavia è solo grazie alla diffusione degli store di applicazioni che si è avuto un boom nella diffusione di questi nuovi dispositivi. Infatti grazie ad essi gli sviluppatori hanno trasformato un semplice cellulare dotato di connessione internet in uno strumento in grado di coprire ogni singola esigenza da parte dell'utenza.

In questo nuovo scenario però la Microsoft non poteva competere con i nuovi sistemi operativi: iOS di Apple e Android di Google. Quindi è stato necessario provvedere ad una vera propria rivoluzione. Da Windows Mobile 7 si è passati a Windows Phone. Il

cambiamento non si è avuto solo nel nome: a cambiare è stata la filosofia con cui approcciarsi a questo nuovo progetto, la grafica, l'esperienza utente. Il principio alla base della nuova interfaccia è *l'International Typographic Style*, stile fondato su tre principi chiave: chiarezza, leggibilità e obiettività. Infatti prendendo in mano un dispositivo Windows Phone è immediato constatarne la notevole differenza rispetto ai concorrenti. Le applicazioni presentano un'interfaccia molto più semplice rispetto a quanto avviene su altre piattaforme; per questo motivo ciò che risalta agli occhi dell'utente è il contenuto.

3.1 Sistemi operativi a confronto

Anche nello sviluppo dei device vi è una notevole differenza tra la politica adottata dalla Microsoft e quella di iOS e di Android.

Infatti i produttori di smartphones hanno a disposizione due approcci profondamente diversi:

- sviluppare anche l'hardware, come ha fatto la Apple;
- vendere il proprio software ai produttori sul mercato che successivamente si faranno carico della produzione del device, caso questo di Android e Microsoft.

Analizziamo quindi i vari scenari e i vari vantaggi o svantaggi che possono derivare dall'una o dall'altra scelta.

Consideriamo innanzitutto il caso di Apple. Tale politica consente alla suddetta azienda di avere il pieno controllo dell'intero ecosistema e di avere minimi problemi di frammentazione, grazie alla limitata varietà di device presenti in commercio. Tuttavia tale scelta limita notevolmente la libertà degli utenti, che al momento dell'acquisto di un nuovo cellulare si trovano di fronte un gamma pressoché univoca. Unico il modello, dimensioni dello schermo fisse. Possibilità di scelta solo riguardo le

dotazioni hardware, relativamente parlando: capacità di memoria, risoluzione, capacità della batteria ecc.

Dall'altra parte invece vi è il caso di Android, che offre una gamma quasi illimitata di device con dimensioni, forme e caratteristiche tecniche del tutto indifferenti. Tuttavia questa scelta ha dei vantaggi per gli utenti ma anche dei notevoli svantaggi, non solo per chi acquista tali prodotti, ma anche per gli stessi sviluppatori. Chi deve acquistare infatti può scegliere tra un'infinità di dispositivi in base alle proprie esigenze. Tale diversità però risulta uno svantaggio: rende infatti difficoltosa la procedura di aggiornamento alle nuove versioni del sistema operativo, a cui devono direttamente provvedere i produttori del telefono. Dal punto di vista degli sviluppatori inoltre è penalizzante poiché le caratteristiche tecniche dei cellulari e, di conseguenza, le relative performance, non garantiscono la compatibilità universale delle applicazioni.

In questo scenario si inserisce la Microsoft, la cui idea è quella di fungere da punto d'incontro tra i due diversi approcci: da un lato offrire all'utente una molteplicità di dispositivi, dall'altro assicurare un puntuale aggiornamento del software, affidabilità e qualità delle applicazioni. Per raggiungere tale scopo, Microsoft ha deciso di imporre ai produttori di device una configurazione minima, che deve essere rispettata dai produttori di smartphone dotati di Windows Phone. In questo modo gli sviluppatori di applicazioni possono programmare liberamente senza preoccuparsi della presenza o meno di un determinato sensore. Tale configurazione minima può naturalmente essere potenziata dai produttori stessi, offrendo così un prodotto di miglior qualità. Inoltre la Microsoft ha deciso di non consentire ai produttori di personalizzare l'interfaccia, in modo tale da rendere omogenei i dispositivi presenti sul mercato e da offrire la stessa esperienza ad ogni utente Windows Phone.

3.2 L'ambiente di sviluppo: Visual Studio for Windows Phone

Per sviluppare applicazioni per Windows Phone è necessario conoscere due linguaggi: C# e VB.NET. Il primo è un linguaggio che deriva dal C come sintassi ed è il più diffuso; il secondo invece è l'evoluzione di Visual Basic per il framework .NET.

Avendo una buona padronanza di tali strumenti è possibile iniziare a scrivere i primi codici.

Il punto di partenza è il Windows Phone Dev Center, il portale di Microsoft dedicato agli sviluppatori Windows. In esso è possibile scaricare i tool di sviluppo, disponibili in diverse lingue. Essi sono composti da:

- *Visual Studio 2012 Express*, che costituisce l'ambiente di sviluppo Microsoft. Tramite tale programma è possibile lavorare sia sull'interfaccia grafica che sulla logica delle applicazioni;
- *Blend for Windows Phone*: include una serie di programmi rivolti soprattutto a chi si occupa di grafica e di design. Blend è un editor visuale di XAML, il linguaggio usato per la definizione dell'interfaccia grafica di un'applicazione;
- L'*SDK*, ovvero le librerie impiegate nella compilazione dei codici, indispensabili per interagire con le API (Application Programming Interface) del telefono;
- L'*emulatore*, con cui poter testare le applicazioni anche in assenza di un vero dispositivo.

Per poter installare tali tool sul proprio computer è necessario avere Windows 8. Con la versione precedente, Windows 7, è possibile comunque installare la versione 7.1 dell'*SDK* (Software development kit), che però permette esclusivamente lo sviluppo di applicazioni per Windows Phone 7.5.

Particolare importanza nel panorama dello sviluppo Microsoft viene riservata a NuGet: si tratta di un package manager che semplifica la procedura di installazione e di utilizzo di librerie di terze parti all'interno di un progetto. Se prima dell'avvento di NuGet era necessario compiere tutta una serie di operazioni a volte un po' tediose (collegarsi al sito dello sviluppatore, scaricare la libreria e copiarla all'interno di una cartella del proprio progetto e creare da Visual Studio un riferimento a tale libreria) ora NuGet fa tutto questo al posto nostro.

Inoltre un importante tool di sviluppo è rappresentato dall'emulatore, che ci dà la possibilità di testare le applicazioni senza possedere un device reale. Naturalmente tale

supporto non può sostituire del tutto un cellulare, tuttavia risulta uno strumento molto importante grazie alla facilità di utilizzo e al supporto per il testing di scenari di difficile realizzazione fisica.

Per quanto riguarda i test che ho compiuto sulla mia applicazione, ho avuto la possibilità di eseguirli direttamente sul mio device, un Nokia Lumia 520, un cellulare che non ha nulla da invidiare ai top di gamma.

Prima di poter installare applicazioni sul proprio device è necessario sbloccarlo. Per far ciò l'SDK installa un'applicazione chiamata Windows Phone Developer Registration. Collegando con il cavo USB il proprio cellulare, basta avviare l'applicazione e inserire le proprie credenziali dell'account Microsoft a cui è legata la propria sottoscrizione da sviluppatori; sarà poi l'applicazione che si collegherà ai server Microsoft e verificherà la validità dell'account. Fatto ciò è possibile eseguire il deploy di applicazioni sul proprio dispositivo. Naturalmente, esiste un limite massimo di 10 applicazioni installabili sul telefono utilizzando questo sistema: tutto ciò è stato fatto per impedire fenomeni di pirateria ed evitare che app disponibili sullo store venissero installate in maniera illegale.

Per poter pubblicare la propria applicazione sul MarketPlace di Windows Phone è necessario avere l'iscrizione da sviluppatori, che ha un costo di 79 euro all'anno e consente la pubblicazione di 100 applicazioni gratuite e di un numero illimitato di applicazioni a pagamento. L'applicazione prima di essere pubblicata deve superare una serie di controlli da parte della Microsoft, che si riserva la possibilità di rifiutare la certificazione di conformità.

Analizziamo ora come si presenta l'ambiente di sviluppo Visual Studio, il principale strumento per sviluppare applicazioni per Windows Phone.

All'apertura del programma la prima cosa che ci viene richiesta è la versione del sistema operativo per la quale sviluppare l'applicazione:

- Windows Phone OS 7.1, che corrisponde a Windows Phone 7.5;
- Windows Phone OS 8.0, che corrisponde a Windows Phone 8.

Una volta effettuata tale scelta, Visual Studio provvederà automaticamente a creare tutto il necessario.

Sulla pagina principale del programma avremo una sezione chiamata Solution Explorer, come è possibile vedere nella seguente immagine.

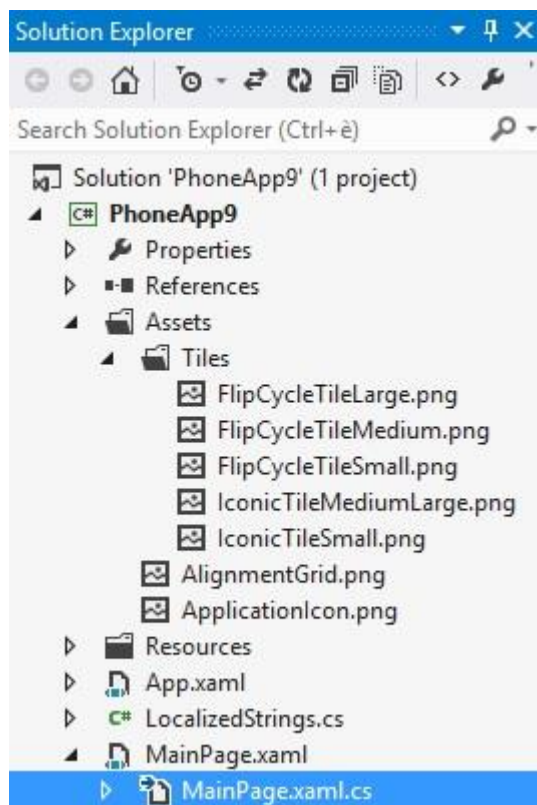


Figura 3.1: Struttura di un progetto Windows Phone.

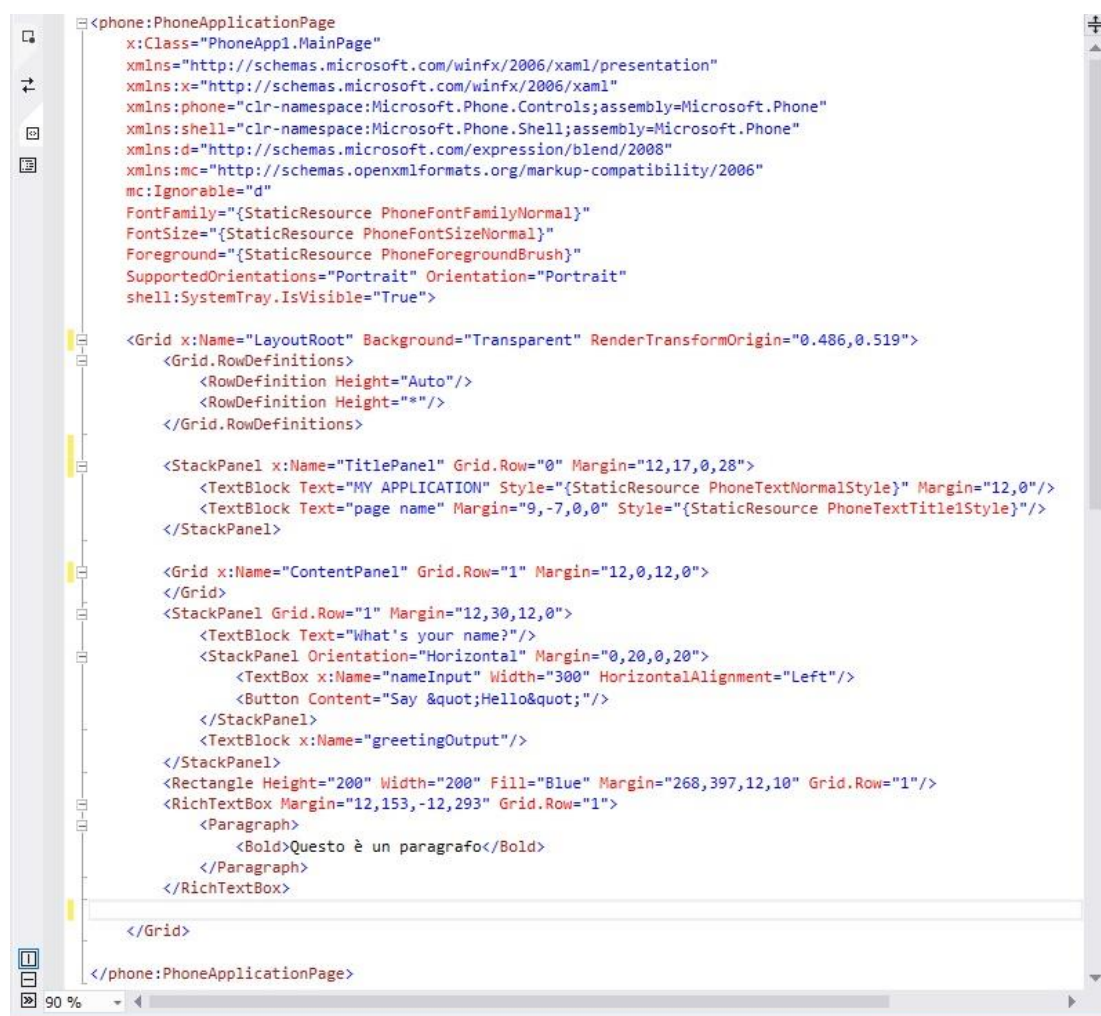
All'interno della cartella Properties vi è contenuto un file XML chiamato WMAppManifest.xml, che contiene tutta una serie di informazioni riguardo il nome, l'autore, il numero di versione. Tale file, che prende il nome di file "Manifest" contiene diverse sezioni tramite cui è possibile impostare il nome dell'applicazione, l'icona che la identifica, un'eventuale descrizione.

Inoltre sempre nella stessa cartella è contenuta una sezione molto importante chiamata Capabilities. In essa vanno dichiarate tutte le funzionalità utilizzate dall'applicazione.

Nella sezione Requirements invece vanno specificati i requisiti hardware indispensabili (come l'utilizzo di sensori, memoria necessaria, ecc.).

In fondo alla finestra Solution Explorer è inoltre possibile notare la presenza di altri due file, uno in formato .xaml e l'altro in formato .xaml.cs.

Lo XAML è il linguaggio con il quale si definiscono le interfacce grafiche di un'applicazione: controlli, animazioni ed effetti vengono tradotti in tag XML, ciascuno con vari attributi che ne definiscono le proprietà e gli eventi. Grazie all'Intellisense di Visual Studio, molte operazioni che prima bisognava fare manualmente, ora vengono applicate in modo automatico.



```
<phone:PhoneApplicationPage
  x:Class="PhoneApp1.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
  xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  mc:Ignorable="d"
  FontFamily="{StaticResource PhoneFontFamilyNormal}"
  FontSize="{StaticResource PhoneFontSizeNormal}"
  Foreground="{StaticResource PhoneForegroundBrush}"
  SupportedOrientations="Portrait" Orientation="Portrait"
  shell:SystemTray.IsVisible="True">
  <Grid x:Name="LayoutRoot" Background="Transparent" RenderTransformOrigin="0.486,0.519">
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto"/>
      <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
      <TextBlock Text="MY APPLICATION" Style="{StaticResource PhoneTextNormalStyle}" Margin="12,0"/>
      <TextBlock Text="page name" Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
    </StackPanel>
    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
      </Grid>
      <StackPanel Grid.Row="1" Margin="12,30,12,0">
        <TextBlock Text="What's your name?"/>
        <StackPanel Orientation="Horizontal" Margin="0,20,0,20">
          <TextBox x:Name="nameInput" Width="300" HorizontalAlignment="Left"/>
          <Button Content="Say &quot;Hello&quot;"/>
        </StackPanel>
        <TextBlock x:Name="greetingOutput"/>
      </StackPanel>
      <Rectangle Height="200" Width="200" Fill="Blue" Margin="268,397,12,10" Grid.Row="1"/>
      <RichTextBox Margin="12,153,-12,293" Grid.Row="1">
        <Paragraph>
          <Bold>Questo è un paragrafo</Bold>
        </Paragraph>
      </RichTextBox>
    </Grid>
  </phone:PhoneApplicationPage>
```

Figura 3.2: esempio di codice XAML.

In questo esempio di file XAML è possibile vedere nella prima parte una serie di istruzioni che vanno a richiamare le librerie necessarie per il funzionamento dell'applicazione. Dopo il primo blocco di informazioni, è presente la parte in cui viene definita la sua interfaccia grafica.

Lo XAML mette infatti a disposizione diversi controlli “contenitore” che permettono di posizionarne altri al loro interno per definirne il layout. I principali sono:

- Il controllo Grid: permette di realizzare delle griglie, fatte di righe e colonne. Al suo interno possiamo posizionare altri elementi. Tutto ciò è reso possibile da due proprietà, RowDefinitions e ColumnDefinitions, che danno la possibilità di specificare il numero e l'altezza delle righe e la larghezza delle colonne;
- Il controllo StackPanel: esso ha il semplice scopo di impilare i controlli. Gli elementi inseriti al suo interno infatti sono disposti uno sotto l'altro, occupando il massimo spazio concesso dall'elemento;
- Il controllo Canvas: ci consente il posizionamento degli elementi tramite l'utilizzo delle proprietà Top e Left, che definiscono la distanza in pixel, rispettivamente, dal margine superiore e da quello sinistro del canvas;
- Il controllo ScrollView: consente lo scrolling nel momento in cui vengono inseriti contenuti più lunghi della dimensione dello schermo.

Nella figura 3.3 è possibile apprezzare una finestra molto importante in fase di progettazione: l'editor visuale di Visual Studio.

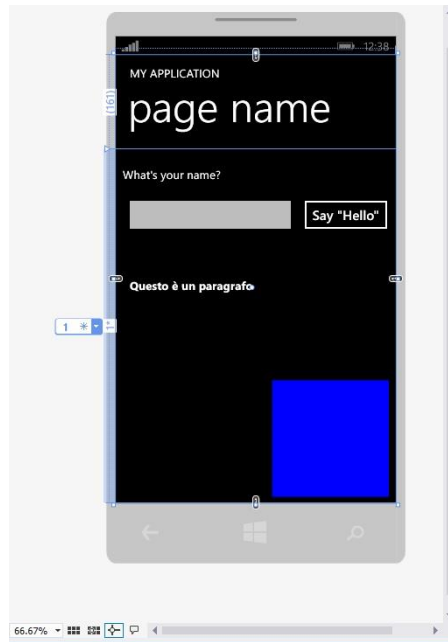


Figura 3.3: editor visuale di Visual Studio.

Grazie ad esso infatti possiamo vedere l'aspetto finale della nostra applicazione. Modificando il codice XAML, l'editor si modificherà in tempo reale. E' possibile inoltre lavorare direttamente nell'editor, interagendo con i vari componenti del nostro progetto. In questo caso sarà il codice XAML che andrà ad aggiornarsi automaticamente. Questo è un aspetto molto importante in fase di progettazione grafica dell'applicazione: infatti da questo punto di vista il compito di un programmatore è stato semplificato notevolmente.

Inoltre un altro aspetto molto importante dell'intellisense di Visual Studio è la presenza di una finestra chiamata "Toolbox". Tramite di essa infatti è possibile inserire in maniera molto rapida una serie di comandi che andranno a costituire la fisionomia di un'applicazione: pulsanti, caselle di testo, ecc. Basta semplicemente trascinarli nell'editor e automaticamente verrà generato il codice XAML che rappresenterà il comando in questione.

Il Code Behind invece è il codice che c'è dietro l'applicazione vera e propria. Esso è contenuto all'interno del file con estensione `.xaml.cs`, se stiamo utilizzando come linguaggio di programmazione `C#`, altrimenti `.xaml.vb`, se stiamo utilizzando

VB.NET. Al suo interno vanno inserite tutte le istruzioni che permettono il funzionamento della nostra app.

In conclusione in questo capitolo è stato analizzato lo sviluppo di questo nuovo sistema operativo e l'ambiente di sviluppo necessario per la creazione delle relative applicazioni.

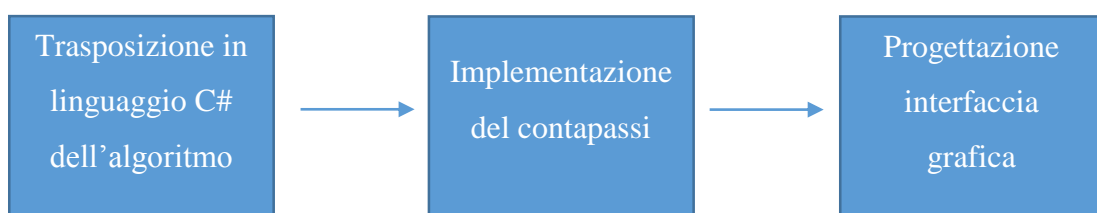
Capitolo 4

Sviluppo dell'applicazione

Questo capitolo rappresenta il punto chiave di questa tesi. Di seguito infatti verrà presentata l'applicazione finale da me realizzata. L'obiettivo di questa tesi infatti è il progetto e l'implementazione dell'algoritmo trovato durante il mio tirocinio. Tramite la realizzazione di un'applicazione è stato possibile mettere in pratica tutto quello a cui si era arrivati precedentemente. Soprattutto è stato fondamentale per i successivi test effettuati, altrimenti non realizzabili.

Come si può vedere dall'indice, tale capitolo è diviso in due sezioni: nella prima parte verrà esposto il codice che opera dietro l'applicazione. Nella seconda parte invece verrà chiarita la sua funzionalità, verrà mostrato il suo aspetto e commentata la sua grafica.

Per la progettazione ho seguito la seguente linea guida:



4.1 Commento del code behind

In questo paragrafo verrà esposto il codice impiegato per la realizzazione dell'applicazione e verrà commentato. Alle parti di codice esposte seguirà il relativo commento.

Innanzitutto le prime dichiarazioni che vanno fatte sono l'utilizzo delle librerie necessarie per il funzionamento di funzioni e sensori impiegati.

```
using System;  
  
using System.Collections.Generic;  
  
using System.Linq;  
  
using System.Net;  
  
using System.Windows;  
  
using System.Windows.Controls;  
  
using System.Windows.Navigation;  
  
using Microsoft.Phone.Controls;  
  
using Microsoft.Phone.Shell;  
  
using PhoneApp9.Resources;  
  
using Microsoft.Devices.Sensors;
```

Oltre a quelle standard, che vengono inserite automaticamente dall'ambiente di sviluppo ogni qualvolta venga creato un nuovo progetto, si può notare anche la presenza di un'ultima libreria.

Ogni sensore infatti è rappresentato da una delle classi disponibili all'interno del namespace `Microsoft.Devices.Sensors`: ognuna di esse espone il metodo

GetDefault(), che restituisce il riferimento al sensore. In particolare, nel mio progetto vengono utilizzati gli accelerometri.

Sono due le modalità con cui possiamo interagire con i sensori:

- Tramite il metodo *GetCurrentReading()*, che restituisce una singola rilevazione dei dati del sensore;
- Tramite l'evento *ReadingChanged()*, che viene scatenato ogni qualvolta la posizione è cambiata e quindi i dati rilevati dal sensore sono differenti.

Dopo la dichiarazione riguardo le librerie utilizzate, inizia la parte di codice relativa alla pagina dell'applicazione a cui stiamo facendo riferimento. Dato che la mia applicazione presenta un'unica pagina, sarà presente solo la dichiarazione della MainPage:

```
public partial class MainPage : PhoneApplicationPage
```

Al suo interno si procederà ad inserire le istruzioni che verranno eseguite successivamente dall'applicazione.

Le prime istruzioni che vengono eseguite al suo avvio sono quelle contenute all'interno del MainPage():

```
public MainPage()  
{  
    InitializeComponent();  
}
```

Come possiamo vedere dalla dichiarazione contenuta al suo interno, l'unica operazione richiesta è l'inizializzazione dei componenti. In un certo senso possiamo affermare che nel momento in cui apriamo l'app non viene scatenato nessun evento, viene semplicemente aperta la pagina iniziale, però il cellulare è pronto a ricevere gli input dall'utente.

A questo punto, aperta la main page, è presente un pulsante con la dicitura "Start". Come dice il nome stesso, tale pulsante ha lo scopo di avviare l'applicazione. Vediamo quindi cosa succede nel momento in cui viene premuto tale pulsante:

```
Accelerometer accelerometer = new Accelerometer();
private void start_Click(object sender, RoutedEventArgs e)
{
    if (altezza.Text == "")
    {
        MessageBox.Show("ERRORE! Non hai inserito la
tua altezza");
        altezza.Text = ("");
        altezza.Focus();
    }
    else
    {
        accelerometer.Start();
        accelerometer.CurrentValueChanged +=
accelerometer_CurrentValueChanged;
        accelerometer.TimeBetweenUpdates =
TimeSpan.FromMilliseconds(100);
    }
}
```

La prima cosa da fare è andare a definire un nuovo oggetto appartenente alla classe *Accelerometer*. In questo modo ogni qualvolta utilizzeremo l'oggetto *Accelerometer* andremo ad invocare i sensori del dispositivo. Come si può vedere nelle istruzioni successive infatti, tale oggetto verrà richiamato più volte.

Successivamente, come si può vedere dal codice sopra riportato, viene realizzata la funzione che viene scatenata ogni qualvolta venga premuto il pulsante Start. È una funzione *void* perché non dovrà restituirci nulla ma semplicemente dovrà eseguire le istruzioni contenute al suo interno.

La prima cosa che possiamo notare è la presenza di una biforcazione nell'esecuzione del codice: è presente infatti un'istruzione *if* che andrà ad eseguire determinati comandi in funzione della condizione espressa. In questo caso è

necessario che venga controllata la presenza o meno di testo all'interno della casella di testo *Altezza*. Infatti dato che l'algoritmo utilizzato dal programma riceve in input l'altezza di una persona, nel momento in cui l'utente dell'app non inserisce la propria statura, l'applicazione smette automaticamente di funzionare.

Per evitare questo evento, ho prontamente inserito la verifica di questa condizione. In particolare, si può osservare che, qualora la casella di testo rimanga vuota (`altezza.Text == ""`), l'applicazione provvederà a mostrare a video in una *MessageBox* un messaggio d'errore. Inoltre tramite il metodo *focus* andremo ad aprire la casella di testo in cui inserire la propria altezza e automaticamente apparirà anche il tastierino alfanumerico con cui inserire i dati.

Nel momento in cui l'utente avrà inserito la propria altezza e avrà premuto start, la condizione non sarà verificata e quindi il programma procederà con la seconda parte dell'istruzione *if*, ossia quella contenuta all'interno dell'*else*.

Innanzitutto tramite il metodo *Start* il programma inizia ad acquisire i dati provenienti dagli accelerometri. Successivamente viene dichiarato l'utilizzo dell'evento *CurrentValueChanged* che viene scatenato ogni qualvolta arrivino nuovi dati dagli accelerometri. Tale evento richiama una funzione che verrà commentata nel seguito. Inoltre, subito sotto all'evento, viene anche specificato l'intervallo di tempo che deve intercorrere tra due letture. In questo caso ho scelto di impostare il timer a 100 millisecondi.

A questo punto andiamo a vedere quali sono le istruzioni contenute all'interno della funzione *accelerometer_CurrentValueChanged*.

Innanzitutto, prima di iniziare a scrivere istruzioni, le variabili utilizzate all'interno della funzione devono essere dichiarate. Dato che tali dichiarazioni vengono effettuate all'esterno della funzione in questione, le variabili espresse verranno inizializzate a 0.

```
public double x_old { get; set; }  
public double y_old { get; set; }  
public double z_old { get; set; }
```

```
private int counter;
private double timeOld;
private double timeNew;
private double deltat;
private double distanzaPercorsa;
```

Le prime tre saranno utilizzate per le accelerazioni sui tre assi all'istante $t-1$. Possiamo notare inoltre la presenza di un contatore, la variabile *counter*, che andrà ad incrementarsi di un'unità ogni qualvolta venga registrato un nuovo passo. La variabile *timeOld* corrisponde all'istante di tempo $t-1$, mentre *timeNew* all'istante di tempo t . Come vedremo in seguito, il *deltat* sta ad indicare l'intervallo di tempo tra i due istanti. Infine con *distanzaPercorsa* indico, come già dice il nome, la lunghezza espressa in metri che abbiamo percorso.

A questo punto, dopo aver dichiarato le variabili che andremo ad utilizzare, è possibile procedere alla stesura del codice della funzione.

```
void accelerometer_CurrentValueChanged(object sender,
SensorReadingEventArgs<AccelerometerReading> e)
{
    Dispatcher.BeginInvoke(() =>
    {
        var position =
e.SensorReading.Acceleration;
        double x = position.X;
        double y = position.Y;
        double z = position.Z;
        double oldValue =
Math.Sqrt((double)(((x_old * x_old) + (y_old * y_old)) +
(z_old * z_old)));
        double newValue = Math.Sqrt((double)(((x
* x) + (y * y)) + (z * z)));
```

```
        double picchiOldValue =
picchi(oldValue);
        double picchiNewValue =
picchi(newValue);
        double k = diffPicchi(picchiOldValue,
picchiNewValue);
        double h =
Convert.ToDouble(altezza.Text);
        if ((k - 0.22) > 0)
        {
            deltat = timeNew - timeOld;
            double frequenza = 1.0 / deltat;
            if (frequenza < 3.5)
            {
                double lunghezza = passo(h,
frequenza);

                distanzaPercorsa += lunghezza;
                counter++;
                timeOld = timeNew;
            }
        }

        timeNew = timeNew + 0.1;
        x_old = x;
        y_old = y;
        z_old = z;
        passi.Text = counter.ToString();
        distanza.Text =
distanzaPercorsa.ToString();
        velocitàmedia.Text = (distanzaPercorsa /
timeNew).ToString();
```

```
    });  
}
```

Innanzitutto, va sottolineato l'utilizzo del *Dispatcher*. Infatti molte delle API di Windows Phone sono basate non solo su codice sincrono, ma anche sul concetto di multi thread: il thread principale di un'applicazione Windows Phone è quello che gestisce la UI e, per questo motivo, deve essere lasciato il più possibile libero, affinché l'interfaccia sia sempre reattiva e fluida. I thread secondari però non hanno accesso diretto alla UI. In questi casi, abbiamo a disposizione una classe chiamata *Dispatcher*, che permette di interagire con il thread principale della UI direttamente da un thread secondario. Nel caso in questione, se non ne avessi fatto uso, non avrei potuto mostrare nelle caselle di testo le variabili dichiarate.

A questo punto possiamo eseguire tutte le operazioni che vogliamo al suo interno e interagire con il thread principale.

La prima cosa che vado a dichiarare è l'utilizzo di un vettore costituito da tre elementi che vado a chiamare *position* e che rappresenterà le tre accelerazioni. Infatti successivamente vado ad assegnare a tre variabili (*x*, *y*, *z*) ognuna di esse. Esse costituiranno le accelerazioni percepite all'istante di tempo *t*.

Le due variabili dichiarate in seguito, *oldValue* e *NewValue* rappresentano rispettivamente il valore del modulo del vettore accelerazione all'istante *t-1* e all'istante *t*. Naturalmente, per poter utilizzare la funzione che mi calcola la radice quadrata dei due valori, devo dichiarare l'utilizzo della classe *Math* contenuta all'interno della libreria *System*.

Le istruzioni subito successive non fanno altro che tradurre in linguaggio C il contapassi elaborato precedentemente. Esse richiamano infatti due funzioni:

- La funzione *picchi* che riceve in ingresso una variabile *double* e restituisce un valore, sempre in formato *double*:

```
double picchi(double v)  
{  
    double z;
```

```
        z = Math.Abs((double)(v - 1));
        return z;
    }
```

Si può vedere che le istruzioni contenute all'interno della funzione non fanno altro che calcolare il valore assoluto della variabile inserita in ingresso a cui è stata sottratta un'unità.

- La funzione *diffPicchi* che invece prende in ingresso due variabili *double* e restituisce sempre un *double*:

```
double diffPicchi(double k, double j)
{
    double differenza;
    differenza = j - k;
    return differenza;
}
```

In questo caso la funzione procede ad eseguire la differenza tra le due variabili poste in ingresso.

A questo punto viene imposta la prima operazione di filtraggio: tramite infatti la verifica della condizione imposta dall'*if*, andremo a scartare tutti quei valori che sono al di sotto della soglia precedentemente trovata. Infatti se un valore è al di sotto della soglia, l'applicazione semplicemente provvederà ad incrementare la variabile *timeNew* di 0.1 secondi, dato che l'intervallo di campionamento è stato impostato sui 100 millisecondi, lasciando invariata tuttavia *timeOld*, poiché non si è verificato nessun passo. A questo punto gli attuali valori di accelerazione (*x*, *y*, *z*) diventeranno i valori passati della futura iterazione (*x_old*, *y_old*, *z_old*).

Qualora invece la condizione fosse verificata, ossia se il valore trovato supera la soglia imposta, ecco che in previsione del futuro filtraggio, si rende necessario il calcolo della frequenza. A tal proposito viene infatti calcolato l'intervallo di tempo occorso tra l'istante in cui è stato registrato il precedente passo e l'istante attuale:

```
deltat = timeNew - timeOld;
```

Avendo quindi questa informazione, possiamo direttamente calcolarci la frequenza, che ci servirà da input per i calcoli che andrà a svolgere l'algoritmo:

```
double frequenza = 1.0 / deltat;
```

Come è stato precedentemente esposto nel capitolo 2, la frequenza gioca un ruolo fondamentale per l'esatta verifica di un passo. È grazie a questa informazione infatti che è possibile effettuare il filtraggio supplementare basato sulla realtà effettiva dei fatti.

Di conseguenza si è reso necessario l'utilizzo di una nuova istruzione *if* che in questo caso va a controllare che la frequenza sia contenuta al di sotto di una determinata soglia. Se tale soglia viene superata, l'algoritmo contenuto all'interno dell'*if* non viene scatenato e quindi si torna alla situazione precedente, in cui non viene registrato nessun passo e viene predisposta una nuova iterazione. Nel momento in cui invece la frequenza è al di sotto della soglia, la condizione viene verificata e l'applicazione provvede ad eseguire le istruzioni contenute all'interno dell'*if*.

È solo in questa parte di codice infatti che viene invocato l'algoritmo studiato. In virtù di ciò viene richiamata la funzione che lo implementa:

```
double passo(double h, double f)
{
    int j;
    int k;
    double[] prod = { 0, 0, 0 };
    double prod1 = 0;
    double[,] mat1 = { { 3.00952407593057e-05, -
0.000480249748894518, 0.000515146458806515 },
                      { -0.0111268356865375,
0.170421017871976, -0.180027545833117 },
                      { 1.13882060664873, -
15.1303214612181, 16.1171570600353 } };
```

```
double[] H = { h * h, h, 1 };
double[,] F = { { f * f }, { f }, { 1 } };

for (j = 0; j < 3; j++)
    for (k = 0; k < 3; k++)
        prod[j] += H[k] * mat1[k, j];

for (j = 0; j < 1; j++)
    for (k = 0; k < 3; k++)
        prod1 += prod[k] * F[k, j];

return prod1;
}
```

Naturalmente, tale funzione prende in ingresso due variabili, *h* ed *f*, che stanno ad indicare l'altezza e la frequenza e in uscita dà un numero *double*, che costituirà la lunghezza di passo cercata.

Una volta ottenuta l'informazione riguardante la lunghezza del passo, si procede ad incrementare la distanza percorsa, semplicemente andando ad aggiungere alla lunghezza precedente la nuova trovata.

Fatto ciò è possibile incrementare il contatore di un'unità (in modo tale da aggiungere al conteggio il nuovo passo registrato). A questo punto, tale istante di tempo diventerà nella successiva iterazione l'istante di tempo passato in cui è avvenuta la registrazione di un passo.

Infine, con le ultime tre dichiarazioni, verranno mostrati nelle rispettive caselle di testo, il numero di passi, la distanza percorsa e la velocità media di percorrenza. È importante sottolineare che, nei calcoli effettuati nel code behind sono state utilizzate variabili *double*. Tuttavia nel mostrare a video nelle caselle di testo tali valori, bisogna convertirli nell'equivalente rappresentazione in forma di stringa. Per realizzare tale operazione basta applicare il metodo *ToString* all'oggetto in

questione ed esso verrà trasformato automaticamente nel formato leggibile dalla casella di testo.

Non resta quindi che analizzare l'ultimo degli eventi che possono essere invocati dall'applicazione. Nella pagina principale infatti è presente anche un pulsante di stop. Come è facilmente intuibile, tale pulsante ha lo scopo di interrompere l'esecuzione dell'evento scatenato dallo start.

```
private void stop_Click(object sender, RoutedEventArgs e)
{
    accelerometer.Stop();
}
```

Come si può vedere dal codice riportato, se prima per avviare la lettura degli accelerometri veniva utilizzato il metodo *Start*, ora viene utilizzato il metodo *Stop*, che letteralmente interrompe l'acquisizione di dati dall'accelerometro.

4.2 Presentazione dell'applicazione, relativa grafica e funzionalità

A questo punto vediamo come si presenta l'interfaccia dell'applicazione e analizziamo alcuni aspetti tecnici che rivestono un ruolo molto importante nella diffusione di tale prodotto nel mondo del mercato.

Innanzitutto, la scelta del nome per la propria applicazione. Essa infatti costituisce una decisione molto importante, dato che da questa potrà dipendere il destino dell'app.

L'obiettivo nella scelta del nome è l'essere riconoscibile e facilmente memorizzabile. Sono da evitare nomi simili a quelli già esistenti, comprendenti parole o troppo banali oppure di difficile pronuncia. Inoltre è buona norma non andare oltre un certo numero di caratteri.

Il nome che ho deciso di dare alla mia applicazione è *Space Meter*, scelta dovuta essenzialmente a due motivi, considerando la doppia accezione del termine inglese *Space*:

- Spazio inteso come distanza: poiché uno dei compiti che svolgerà l'app sarà quello di calcolare la distanza percorsa.
- Spazio inteso in termini astronomici: chiaro riferimento al mio percorso di studi, Ingegneria Aerospaziale.

La Microsoft permette agli sviluppatori di prenotare un nome per la propria app prima ancora di averla ideata. Il nome scelto in questo caso verrà momentaneamente bloccato e gli altri sviluppatori saranno impossibilitati nell'utilizzarlo. Tale prenotazione ha la durata di un anno. Trascorso l'anno, nel caso in cui l'applicazione non sia stata sviluppata, tale nome verrà reso di nuovo disponibile.

Naturalmente il nome scelto deve rispettare certi limiti: non è possibile utilizzare infatti nomi protetti da marchio, di cui non si dispone dei diritti necessari per l'utilizzo.

Un altro aspetto fondamentale nella diffusione della propria app è la scelta dell'icona che la rappresenterà. Nel mio caso, data la tipologia dell'applicazione la scelta è caduta sulla seguente immagine:



Figura 4.1: icona dell'applicazione.

Come si può vedere, è un'icona molto semplice, immediata e dal colore vivace: due impronte a simboleggiare il susseguirsi di passi.

Nel mondo Windows Phone, le icone vengono chiamate tile e possono essere posizionate nello Start Screen, in modo tale da rappresentare una scorciatoia rispetto al menù principale. Windows Phone 8 mette a disposizione tre formati differenti di tile:

- **Small:** la tile occupa $\frac{1}{4}$ del formato standard;
- **Medium:** è la dimensione quadrata standard;
- **Wide:** la tile è rettangolare e occupa lo spazio di due tile Medium affiancate.

Per impostare il nome, l'icona di un'applicazione e definirne alcune caratteristiche, dando una semplice descrizione, bisogna aprire il file di Manifest e specificarne alcune proprietà:

- *Display Name:* va a definire il nome che apparirà nel menù principale;
- *Description:* in cui è possibile dare una semplice descrizione dell'applicazione, descrivendo le sue funzioni principali;
- *App Icon:* che mostrerà nel menù principale l'icona impostata;
- *Tile Title:* ossia il nome che figurerà sulla tile presente nello Start Screen;
- *Tile Images:* le icone presenti nello Start Screen, una per ogni formato disponibile.

Per avere un'idea di come si presenterà l'applicazione nel menù principale, ho deciso di riportare qui sotto un'immagine esplicativa:

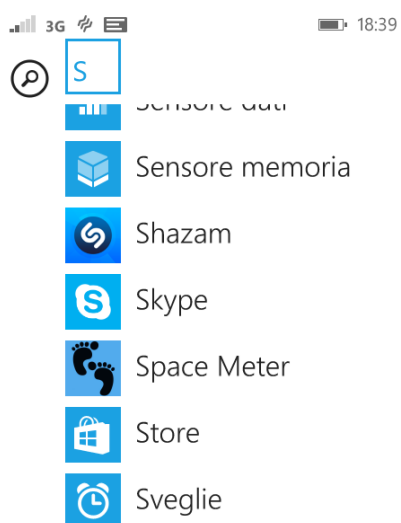


Figura 4.2: menù principale Windows Phone.

A questo punto passiamo ad analizzare come si presenta la pagina principale dell'applicazione.



Figura 4.3: main page dell'applicazione.

La prima cosa a cui dare risalto è il nome dell'applicazione, in alto con caratteri grandi.

Più in basso è possibile notare la presenza di due pulsanti, definiti nello XAML con due comandi *Button*: il primo a sinistra, con cui avviare la lettura dei dati degli accelerometri, il secondo a destra, con cui stoppare tale lettura. Gli eventi che vengono scatenati cliccando su questi due pulsanti vengono anch'essi riportati nel codice XAML contenuto nel Main Page. Tali istruzioni vengono create automaticamente nel momento in cui andiamo a valorizzare il campo click nella properties window di Visual Studio.

Successivamente ho inserito un *textblock* per indicare all'utente cosa inserire nella *textbox* sottostante: la propria altezza espressa in centimetri. È importante che l'utente inserisca la giusta unità di misura, dato che per come è stata progettata l'applicazione un errore di questo tipo comprometterebbe il giusto funzionamento dell'algoritmo.

Infine vi è la presenza di tre *textblock* e tre *textbox*, in cui verranno mostrati a video il numero dei passi effettuati, la distanza percorsa e la velocità media.

Capitolo 5

Test effettuati e confronti con altre app

In questo ultimo capitolo si procederà a riassumere i principali risultati ottenuti durante i test svolti nell'hangar in cui sono presenti i laboratori didattici dell'Università di Bologna con sede a Forlì.

In particolare nella prima parte del capitolo verranno spiegate le modalità con cui sono state effettuate le prove e i relativi risultati. Nella seconda parte invece l'applicazione realizzata da me verrà comparata con altre già esistenti.

5.1 Principali risultati ottenuti durante i test

Durante il mio tirocinio svolto nei laboratori didattici dell'Università, uno delle attività che ho svolto è stata la preparazione di un percorso per prove sperimentali, fondamentale per la raccolta delle informazioni necessarie alla realizzazione del mio algoritmo.

Per raggiungere tale scopo è stato realizzato un circuito chiuso, disponendo a terra un nastro adesivo per segnare il percorso scelto. Il punto di partenza del circuito è stato

uno degli ingressi dell'hangar e il punto di arrivo è stato scelto in maniera tale che il percorso si chiudesse su se stesso. Inoltre ad ogni metro è stato necessario indicare la misura della distanza percorsa. La lunghezza finale del tracciato è di circa 133 metri.



Figura 5.1: mappa del tracciato visto dall'alto.

Su questo stesso tracciato ho testato la mia app. Va sottolineato il fatto che tale percorso non è rettilineo ma sono presenti delle brusche variazioni di direzione che portano ad improvvisi cambiamenti di accelerazione che vanno ad influire notevolmente nella rilevazione di un passo. Il dispositivo mobile è stato inserito in una tasca dei pantaloni.

In linea di massima le mie prove possono essere racchiuse in due tipologie:

- Nella prima sessione ho realizzato i test mantenendo un'andatura quanto più regolare possibile, evitando accelerazioni o decelerazioni. Facendo ciò il parametro che in buona approssimazione ho mantenuto costante è stato la lunghezza del passo.

- Nella seconda sessione invece ho simulato un test che rispecchia maggiormente la vita reale: ho infatti alternato tratti ad andatura regolare ad improvvise accelerazioni o decelerazioni, sperimentando diversi livelli di velocità durante la stessa prova.

I risultati ottenuti nella prima sessione sono stati raccolti in un foglio di lavoro di Excel e sulla base di questi dati è stato possibile calcolare la percentuale di errore contenuta nel numero di passi e la percentuale di errore nella distanza percorsa. Lo stesso procedimento è stato effettuato per la seconda sessione. Nelle due tabelle seguenti è possibile apprezzare i risultati ottenuti nelle due sessioni di test.

	passiReali	passiMisurati	distanzaReale	distanzaMisurata	E_Passi%	E_Distanza%
prova1	140	142	119	129,59	1,43	8,90
prova2	151	149	119	123,18	1,32	3,51
prova3	161	157	119	118,73	2,48	0,23
prova4	99	101	119	110,62	2,02	7,04
prova5	152	145	119	109,42	4,61	8,05
prova6	152	146	119	107,67	3,95	9,52
prova7	95	97	119	109,75	2,11	7,77
prova8	148	146	119	130,37	1,35	9,55
prova9	149	155	119	145,62	4,03	22,37
prova10	160	148	119	116,99	7,50	1,69

Tabella 1: risultati I sessione di test con andatura regolare.

	passiReali	passiMisurati	distanzaReale	distanzaMisurata	E_Passi%	E_Distanza%
prova1	125	118	119	112,62	5,60	5,36
prova2	128	123	119	113,1254	3,91	4,94
prova3	138	142	119	125,6714	2,90	5,61
prova4	133	135	119	129,0924	1,50	8,48
prova5	123	120	119	119,184	2,44	0,15
prova6	126	122	119	118,2044	3,17	0,67

Tabella 2: risultati II sessione di test con andatura mista.

Alla fine è stata calcolata la media in percentuale degli errori in ogni singola sessione, ottenendo così i seguenti risultati:

- I SESSIONE:
 - Errore percentuale sul numero dei passi: 3,08%
 - Errore percentuale sulla distanza percorsa: 7,86%

- II SESSIONE:
 - Errore percentuale sul numero dei passi: 3,25%
 - Errore percentuale sulla distanza percorsa: 4,20%

Da come si può osservare dai dati, si hanno buone prestazioni anche in presenza di andatura mista: in questo senso possiamo dire che l'algoritmo che rileva i passi dà buoni risultati anche in presenza di brusche accelerazioni o decelerazioni.

Inoltre bisogna considerare un altro aspetto: se da un lato dall'errore sul numero dei passi si può constatare la bontà dello step counter elaborato, dall'altro lato, considerando l'errore sulla distanza, bisogna tener conto di due fattori: l'errore precedente sul numero dei passi va a sommarsi infatti con la bontà dell'algoritmo trovato durante il tirocinio.

Da questa analisi si evince quindi che migliorando l'algoritmo che rileva un passo possiamo migliorare entrambi i valori. Inoltre l'errore sulla distanza percorsa può essere ulteriormente migliorato andando a rendere più efficiente l'algoritmo precedentemente esposto.

5.2 Applicazioni a confronto

Dall'analisi fatta nel precedente paragrafo possiamo avere un'idea relativa della bontà dell'applicazione da me realizzata. Tuttavia per avere una visione d'insieme tali dati devono essere confrontati con quelli relativi ai dispositivi attualmente in commercio.

A tal proposito riporto un articolo pubblicato sul JAMA(The Journal of the American Medical Association) in data 10/02/2015 dal titolo "Accuracy of Smartphone Applications and Wearable Devices for Tracking Physical Activity Data"[2]. Tale articolo riassume i principali risultati trovati dai ricercatori dell'Università della Pennsylvania riguardo l'accuratezza delle applicazioni per smartphones e dispositivi indossabili per il monitoraggio dell'attività fisica.

Prima di vedere i risultati dei vari test analizziamo il metodo con cui sono stati effettuati. Sono stati scelti diversi individui di 18 o più anni che hanno corso su di un tapis roulant alla velocità costante di 3.0 mph. Nella prima sessione di test hanno effettuato 500 passi e nella seconda 1500 passi.

Sono stati selezionati 10 applicazioni e dispositivi contapassi tra i più conosciuti ed utilizzati negli Stati Uniti. Alla cintura, ogni partecipante indossava il pedometro Digi-Walker SW-200 (Yamax) e 2 accelerometri: il Zip e il One (Fitbit). Al polso 3 dispositivi indossabili: il Flex (Fitbit), l'UP24 (Jawbone) e il Fuelband (Nike). In una tasca dei pantaloni ognuno indossava un iPhone 5S (Apple) in cui erano in esecuzione tre applicazioni iOS: Fitbit (Fitbit), Health Mate (Withings) e Moves (ProtoGeo Oy). Nell'altra tasca invece era presente un Galaxy S4 (Samsung Electronics) in cui era in esecuzione un'applicazione Android: Moves (ProtoGeo Oy).

Tra tutti i dispositivi utilizzati, sono state registrate 552 prove, ottenute da 14 partecipanti in 56 sessioni. I partecipanti erano per il 71,4% donne, con un'età media di 28,1 anni.

Alla fine di ogni test i dati ottenuti sono stati registrati per essere successivamente elaborati. Tramite Excel è stato possibile calcolare l'errore percentuale tra il numero di passi realmente compiuti e quelli registrati.

Sono stati così ottenuti i seguenti risultati: la percentuale di errore oscilla tra -0,3% e 1,0% per i pedometri e gli accelerometri, tra -22,7% e -1,5% per i dispositivi indossabili e tra -6,7% e 6,2% per le applicazioni per smartphones.

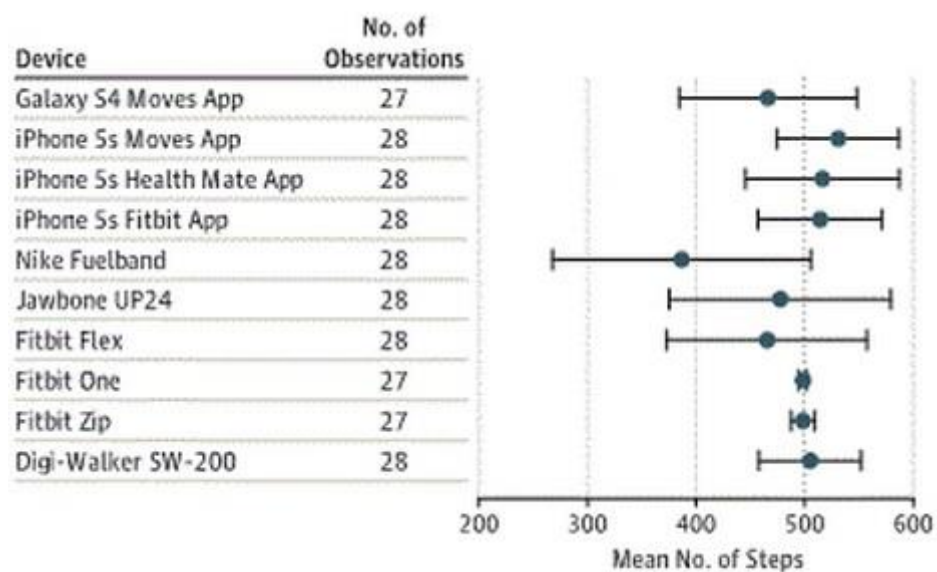


Figura 5.2: risultati dei dispositivi per i test effettuati sulla distanza di 500 passi [2]. La media e la deviazione standard sono state calcolate usando Excel.

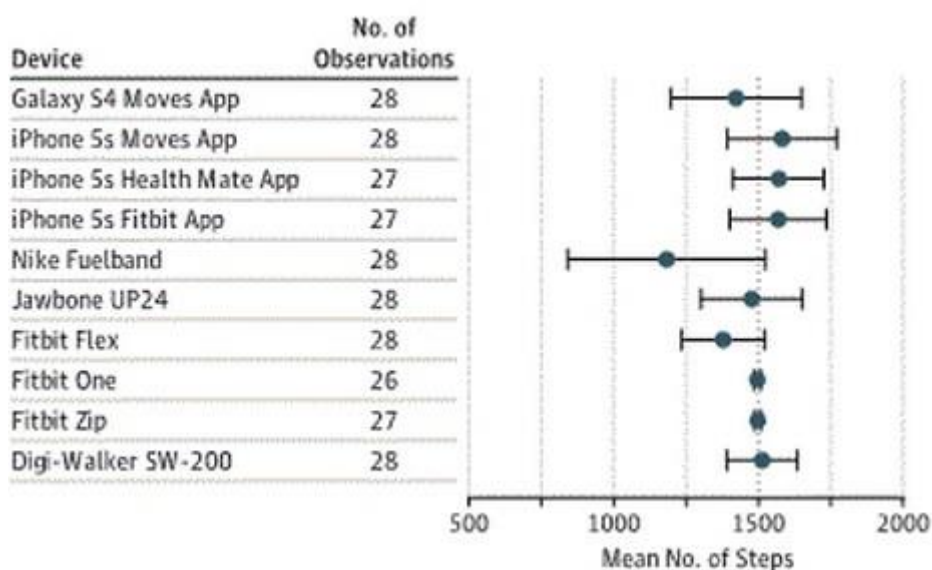


Figura 5.3: risultati dei dispositivi per i test effettuati sulla distanza di 1500 passi [2]. La media e la deviazione standard sono state calcolate usando Excel.

La prima osservazione che è possibile fare è la discreta differenza di prestazioni tra i vari dispositivi. Si può affermare che le applicazioni per smartphones si collocano in una fascia intermedia e, data la loro notevole diffusione, hanno riscontrato grande

successo nella popolazione. I risultati ottenuti con la mia app sono in linea con le prestazioni di quelle già esistenti, se non addirittura in alcuni casi migliori.

Naturalmente, possiamo comparare esclusivamente le prestazioni relative al contapassi, dato che la distanza percorsa viene calcolata in maniera differente a seconda dei casi. Sarebbe impensabile infatti equiparare la distanza percorsa calcolata con una lunghezza di passo costante e quella calcolata tramite il mio algoritmo.

Conclusioni

Come si può constatare dalla lettura dei vari capitoli presi in esame tale progetto può essere considerato un ottimo spunto per sviluppi futuri. Infatti l'idea che è alla base di tutto questo lavoro rappresenta un punto di rottura forte con la tecnologia esistente e apre le porte ad un nuovo modo di affrontare il problema.

Considerare un lunghezza di passo che varia durante la navigazione è fondamentale per una più precisa stima della distanza percorsa. L'algoritmo studiato può essere naturalmente modificato, in modo tale da migliorare le prestazioni dell'applicazione. Lo stesso contapassi può essere potenziato prendendo in considerazione diversi tipi di movimento a cui possiamo sottoporre il nostro dispositivo mobile.

Inoltre tale progetto può essere ulteriormente ampliato in varie direzioni:

- Si potrebbe studiare un sistema in grado di determinare anche la direzione secondo la quale ci stiamo dirigendo. Si potrebbe fare sfruttando ancora i sensori presenti sui device, quali bussola e giroscopi.
- Ottenere informazioni riguardo eventuali dislivelli che possono essere affrontati (ad esempio se si è all'interno di edifici a più piani bisogna capire se stiamo salendo, scendendo o rimanendo sullo stesso piano).

- Integrare l'applicazione con mappe esistenti in modo tale da permettere anche la localizzazione outdoor.
- Integrare la posizione trovata tramite gli accelerometri con il segnale GPS in caso di navigazione esterna.

Ad ogni modo i risultati ottenuti con questo primo progetto sono decisamente positivi e incoraggianti per un possibile sviluppo futuro che possa portare ad un miglioramento delle prestazioni attuali e ad un arricchimento delle funzionalità.

Bibliografia

Pagani, Matteo: Sviluppare applicazioni per Windows Phone 8. 2013. Edizioni FAG Milano.

[1] WHO World Health Organisation

URL: http://www.who.int/dietphysicalactivity/factsheet_recommendations/en/

[2] Accuracy of Smartphone Applications and Wearable Devices for Tracking Physical Activity Data

Meredith A. Case, BA; Holland A. Burwick; Kevin G. Volpp, MD, PhD; Mitesh S. Patel, MD, MBA, MS. JAMA: The Journal of the American Medical Association

URL(10/02/2015): <http://jama.jamanetwork.com/article.aspx?articleid=2108876>