Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in
Ingegneria Informatica

STIMA DELLA POSIZIONE IN DISPOSITIVI MOBILI, CON APPLICAZIONE A UNO STRUMENTO MUSICALE VIRTUALE

Studente:
Stefano Mazzocca

Relatore:
Carlo Fantozzi

12/10/2015

Sommario

In questa tesi verrà proposta la realizzazione di un algoritmo di *motion tracking* con l'obiettivo di creare un flauto di Pan virtuale su piattaforma mobile Android; lo scopo del nostro lavoro è infatti quello di ridare vita all'antico strumento a quattordici canne. Per fare ciò, abbiamo utilizzato diversi sensori (fotocamera, accelerometro, giroscopio, magnetometro) in modo da avere informazioni sul moto del dispositivo rispetto all'"esecutore".
Naturalmente, le fondamenta di questo lavoro potranno in futuro trovare applicazioni anche in campi diversi da quelli musicali. La stima del movimento può essere infatti impiegata per più scopi come ad esempio la stima del numero di metri percorsi da un robot, la conta di quanti oggetti mobili sono presenti nelle immagini, la stabilizzazione di scene disturbate dal tremolio della mano che impugna la telecamera ecc.

A mia mamma, che ha sempre dovuto lottare, che ha saputo crescere due figli da sola senza far mancare mai niente in casa, che dopo dodici anni di assenza dal mondo del lavoro ha saputo rientrarci con le sue forze e ricominciando tutto da zero.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 Panoramica

Lo scopo di questa tesi è quello di ottenere un Flauto di Pan virtuale per smartphone che possa essere suonato proprio come un vero flauto, cioè attraverso il movimento del dispositivo ed il soffio dell'utente. L'idea nasce dal fatto che nel 1930, in Egitto, è stato recuperato un antico Flauto di Pan, probabilmente di origini greche, attualmente esibito nel Museo di Scienze Archeologiche e d'Arte dell'Università degli Studi di Padova (vedi *Figura 1.1*). Naturalmente, trattandosi di un manufatto non più toccabile, si cerca in questo modo di superare le limitazioni dovute alla ridotta accessibilità, consentendo a chiunque di poterlo suonare.

Al momento sono due i progetti che, per conto del museo, cercano di ridare vita all'antico strumento: il primo, principale, mira a ricreare il flauto tramite tecniche di manipolazione tridimensionale, in modo da permettere al visitatore di afferrarlo virtualmente con le proprie mani e suonarlo soffiando su un set di microfoni; il secondo, oggetto di questo lavoro, si limita ad un'implementazione dello strumento per smartphone, sfruttando i sensori di cui il dispositivo è provvisto.

Purtroppo, ad oggi, non si sa molto del Flauto di Pan ma, grazie ad alcune rappresentazioni di età ellenistico-romana e posteriore, ci è noto che durante il periodo classico lo strumento era di forma quadrangolare, con canne di lunghezza esterna uguali tra loro, mentre, durante l'era ellenistica, esso era a forma d'ala con canne di lunghezza diversa. Studi chimici e morfologici ci hanno aiutato a rivelare alcuni segreti dello strumento ma diversi aspetti rimangono tuttora celati: le canne venivano infatti tappate internamente con materiale organico (cera, propoli...) per ottenere il suono desiderato ma il tempo ha rimosso quasi ogni loro traccia. Al momento dunque, sebbene siano possibili stime basate su dati certi, non so-

Figura 1.1: Antico flauto di Pan attualmente esibito nel Museo di Scienze Archeologiche e d'Arte dell'Università degli Studi di Padova; immagine gentilmente concessa dalla ditta "Nicola restauri S.r.l.", Aramengo (AT)

no note le lunghezze interne delle varie canne e ciò naturalmente cala un velo di mistero sul lato acustico. É tuttavia ragionevole supporre che, per estrarre i vari suoni, anche i concetti teorici di musica greca antica debbano ricoprire un ruolo non secondario. Sappiamo infatti che la struttura melodica dell'epoca era basata su tetracordi, ossia su di una serie di quattro note in cui tra la prima e l'ultima sussisteva un intervallo di *quarta giusta*, cioè un rapporto di 3:4, mentre tra le rimanenti il rapporto era variabile così da consentire l'esecuzione di stesse musiche in modi differenti.
Date tutte queste osservazioni, è quindi possibile ricreare con ragionevole accuratezza quelli che erano i suoni originali del flauto. In particolare, ciò può essere fatto tramite due tecniche distinte: il campionamento del suono prodotto da un flauto ricostruito oppure il ricorso a suoni sintetizzati generati tramite modelli matematici. Sebbene entrambe le soluzioni siano ugualmente valide, al momento nessuna delle due è stata realizzata e quindi la nostra app si limiterà a produrre suoni provvisori.

Tutto il nostro lavoro, più che dello studio e della realizzazione di uno strumento virtuale completo, si occupa del sottoproblema della stima del moto e tutti i sensori impiegati, cioè fotocamera, giroscopio, accelerometro e magnetometro, avranno il solo obiettivo di determinare di quanto il dispositivo si sia spostato rispetto all'u-

tente. In questo modo è infatti sempre possibile conoscere quale tra le quattordici canne (ordinate da quella che produce il suono più grave a quella che produce il suono più acuto) è quella situata sotto la bocca dell'"esecutore", permettendo così la riproduzione del suono corretto ogni volta che viene rilevato un soffio. Una piccola parte del progetto è infatti dedicata anche alla rilevazione del soffio ma deve essere chiaro che ciò viene fatto soltanto per motivi di completezza e non di ricerca.

1.2 Lo stato dell'arte

Per quanto riguarda il moto del dispositivo, sembra che ad oggi non ci siano studi approfonditi sulla stima di movimenti di piccola entità (cm) a partire da dati grezzi. Tutti i riferimenti bibliografici che abbiamo trovato si occupano infatti della stima di percorsi lunghi almeno qualche metro, e solo una minima parte di essi è inserita nel contesto degli smartphone. In [1] ad esempio viene usato un algoritmo basato sul filtro di Kalman esteso (EKF) che usa fotocamera, giroscopio ed accelerometro, mentre in [2] gli autori ricorrono all'utilizzo di una fotocamera e di un raggio laser bidimensionale. L'EKF viene ancora ripreso dagli autori di [3] per stimare con precisione percorsi di alcuni chilometri su terreni naturali grazie all'impiego simultaneo di fotocamera ed IMU (Inertial Measurement Unit), usata come inclinometro. Passando invece all'impiego degli smartphone, in [4] viene proposta una strategia in grado di stimare spostamenti di alcune decine di metri usando i dati registrati da accelerometro, giroscopio e magnetometro.
Per quanto riguarda invece la realizzazione di strumenti musicali virtuali, prima di noi moltissimi altri ricercatori si sono cimentati nel settore. In [5] ad esempio gli autori descrivono diversi strumenti virtuali ottenuti ricorrendo al solo sensore magnetico, cioè una chitarra, una mini-batteria, un'armonica e un theremin (per approfondimenti vedi ad esempio [6]). Anche in [7] viene descritta una mini batteria ma questa volta, oltre che al magnetometro per rilevare l'orientamento, è usato anche l'accelerometro per determinare l'intensità che il suono dovrà avere. Nello stesso articolo, gli autori fanno inoltre notare che, con le medesime basi teoriche, ulteriori strumenti come violini o tastiere possono essere implementati. Approfondimenti per quanto riguarda il touch screen sono riportati in [8], dove viene simulata una chitarra ed una mini-batteria, ed in [9], dove usando l'accelerometro oltre che, appunto, al touch screen, viene presentato un gioco che riproduce una consolle da DJ. Più simile al nostro contesto è il lavoro esposto in [10], che descrive *Ocarina*, un flauto virtuale per iPhone che fa uso del microfono per rilevare il soffio, del touch screen per selezionare la nota e dell'accelerometro per rilevare la presenza, ed eventualmente l'intensità, del vibrato (vedi *Figura 1.2*). Esso rappresenta al momento la massima espressione del flauto virtuale per smartphone e ciò

che noi ci prefiggiamo di fare è superare la limitazione del touchscreen per stabilire quale nota suonare. Col modello di flauto che noi abbiamo in mente infatti il touch screen sarebbe molto limitativo ed è per questo che cerchiamo di affrontare il problema in modo che l'app finale possa essere usata esattamente come un vero Flauto di Pan.

Figura 1.2: Rappresentazione del funzionamento di *Ocarina*; figura presa da [31]

1.3 Struttura della tesi

Prima di cominciare con la descrizione del lavoro svolto, viene di seguito introdotto un capitolo dedicato alla descrizione dello smartphone e dei suoi sensori, e un altro dedicato all'approfondimento delle precedenti ricerche disponibili in letteratura che ci sono sembrate utili come punto di partenza per i nostri obiettivi. Successivamente vengono trattati nel dettaglio le basi teoriche dei metodi che effettivamente hanno ricoperto un qualche ruolo nel nostro progetto e quindi, alla fine, vengono illustrati i vari algoritmi impiegati descrivendo l'app finale anche dal punto di vista dell'utente.

Capitolo 2

Sensori in uno smartphone

Prima di illustrare gli studi svolti, i procedimenti seguiti ed i risultati ottenuti è opportuno dare una panoramica su cosa sia uno smartphone e quali siano i suoi sensori potenzialmente utili al nostro scopo così da dare al lettore un'idea più chiara di ciò che circonda l'argomento.

2.1 Lo smartphone

Lo smartphone è un telefono cellulare che, grazie alla sua capacità di calcolo e di memoria, permette di navigare in Internet, riprodurre musica, scattare foto e girare video. Grazie ai suoi numerosi sensori è inoltre in grado di stimolare la fantasia e la creatività di molti sviluppatori di app che possono quindi dare vita a programmi di ritocco immagini, di fitness, di astronomia, e molto altro ancora, rendendo così lo smartphone sempre nuovo ed attraente. Di sistemi operativi appositamente concepiti per questo strumento ce ne sono molti (iOS, Windows Phone, Symbian, Blackberry 10...) e questa tesi è pensata per Android, sistema operativo per dispositivi mobili sviluppato da Google Inc. basato su kernel Linux. Tale scelta trova la sua giustificazione nel fatto che si tratta di software al momento molto diffuso, *open source* e per lo più gratuito, nel senso che gli sviluppatori sono tenuti a versare soltanto una piccola quota *una tantum* per essere presenti nello *store*, il quale consente agli utenti di scaricare le app.

2.2 L'accelerometro

2.2.1 Panoramica

L'accelerometro misura l'accelerazione applicata al dispositivo includendo sempre anche la forza di gravità. Per questa ragione l'output del sensore darà sempre

$$a_d = -g - \sum F_s/m \tag{2.1}$$

anziché

$$a_d = -\sum F_s/m \tag{2.2}$$

essendo a_d l'accelerazione applicata al dispositivo, F_s la forza applicata al sensore, g l'accelerazione gravitazionale di $9.81 m/s^2$ e m la massa del dispositivo.
Stando così le cose, quando lo smartphone rimane fermo su un tavolo, la lettura dell'accelerometro è di $g = 9.81 m/s^2$ mentre quando è in caduta libera è di $g = 0 m/s^2$. Questo almeno nel caso ideale, trascurando momentaneamente l'effetto del rumore purtroppo mai assente.
Il sistema di coordinate usato è quello standard, per cui quando il dispositivo giace sul tavolo con lo schermo rivolto verso l'alto, quando si muove verso sinistra e quando si muove verso il basso il valore letto sarà positivo lungo l'asse interessato (e negativo nei casi opposti).

2.2.2 Come funziona

Come si può vedere dalla *Figura 2.1*, le parti che compongono l'accelerometro sono due:

- Una sezione a forma di pettine detta *seismic mass* o anche *proof mass*, in grado di compiere piccolissimi movimenti in una sola direzione grazie alle molle a cui è attaccata;

- Una coppia di elettrodi fermi rispetto al chip che permettono il rilevamento del movimento della *seismic mass* grazie all'elettricità.

Quando la *seismic mass* è allontanata dalla sua posizione di riposo, la capacità tra essa e gli elettrodi viene alterata e ciò consente di quantificare l'accelerazione. Ciò è valido solo per l'asse in cui la *proof mass* può muoversi; per ottenere informazioni su tutti e tre gli assi geometrici si usano tre sensori di questo tipo opportunamente orientati.
Per quanto riguarda le dimensioni, non sorprende il fatto che queste componenti siano estremamente miniaturizzate. Nei modelli più vecchi infatti la loro lunghezza era di circa un millimetro ma di recente la dimensione è scesa sotto il mezzo millimetro.

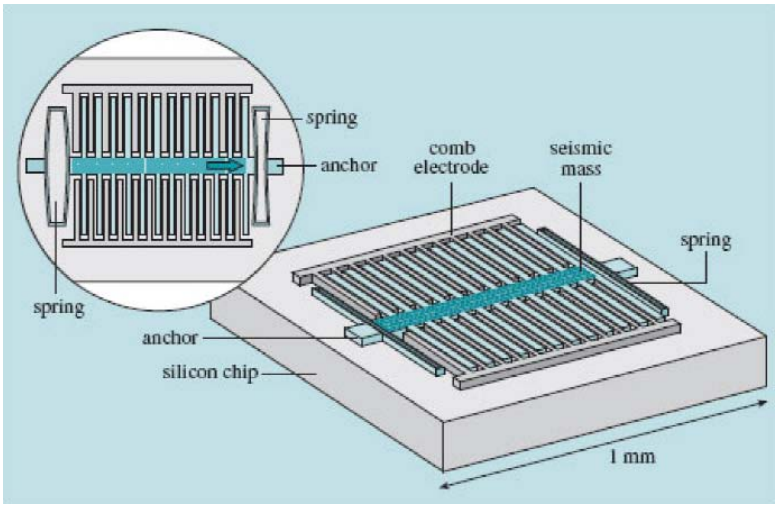


Figura 2.1: Rappresentazione del funzionamento di un accelerometro integrato negli smartphone (o tablet); figura presa da [30]

2.2.3 Limiti

L'accelerometro è noto per essere disturbato da un rumore di tipo gaussiano la cui media in genere è zero se il dispositivo giace su un tavolo. Può tuttavia accadere che, a causa di alcuni fattori come un montaggio non perfetto o errori interni, tale media possa essere non nulla rendendo così necessaria una fase di calibrazione. É inoltre da segnalare anche il fatto che, data l'eccezionale sensibilità del sensore, l'utilizzo di una cover può causare un'alterazione del valore medio dell'output dal momento che potrebbe inclinare leggermente il dispositivo.

2.3 Il giroscopio

2.3.1 Panoramica

Il giroscopio misura il tasso di rotazione in rad/s attorno agli assi X, Y e Z del dispositivo e la rotazione è definita positiva lungo l'asse d'interesse se all'osservatore appare che lo smartphone si stia muovendo in senso antiorario. É un sensore robusto e stabile e fornisce letture precise per veloci rotazioni ma non fornisce valori di riferimento assoluti.

2.3.2 Come funziona

Una componente a forma di pettine viene fatta oscillare lungo l'asse X e quindi viene sfruttato l'effetto di Coriolis: se un oggetto di massa m si muove con velocità

v lungo l'asse X ed ha una velocità angolare Ω lungo l'asse Z, allora si creerà una forza (apparente) lungo l'asse Y secondo l'equazione (2.3) (vedi *Figura 2.2*).

$$F_C = 2m\overrightarrow{v} \times \overrightarrow{\Omega} \tag{2.3}$$

Tale forza viene misurata in base al cambiamento della capacità del circuito dovuto dalla rotazione.

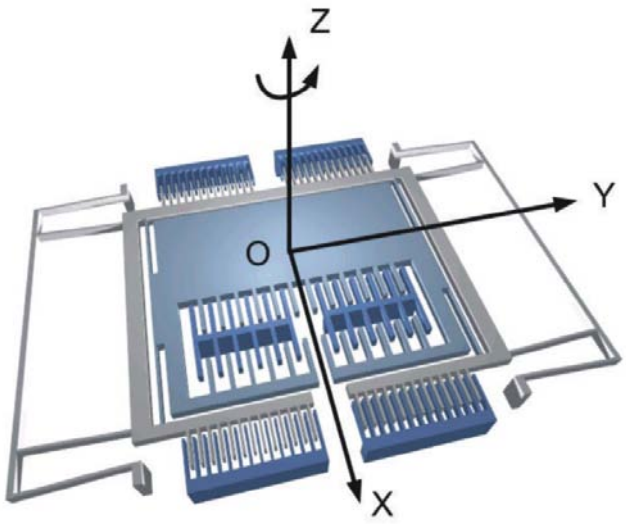


Figura 2.2: Rappresentazione del funzionamento di un giroscopio integrato negli smartphone (o tablet); figura presa da [32]

2.3.3 Limiti

Il giroscopio ha il pregio di essere molto reattivo ma non è adatto a calcolare da solo l'orientamento del dispositivo per lunghi periodi di tempo. Ciò è dovuto anche al rumore di tipo gaussiano cui è soggetto ma, soprattutto, all'effetto drift, ossia alla tendenza a dare in output valori la cui media si discosta sempre di più dallo zero nonostante il dispositivo sia tenuto a riposo.

2.4 Il magnetometro

2.4.1 Panoramica

Molto semplicemente, si tratta di un sensore che misura il campo magnetico circostante ed è quindi in grado di localizzare la direzione del nord magnetico terrestre.

Fornisce quindi una posizione di rotazione che è relativa alla posizione della Terra. Chiaramente si assume che nelle immediate vicinanze non siano presenti oggetti di disturbo (ad esempio magneti), altrimenti le sue letture sarebbero completamente inaffidabili.

2.4.2 Come funziona

Il magnetometro sfrutta la forza di Lorentz riassunta nell'equazione (2.4): se un conduttore immerso in un campo magnetico $\overrightarrow{B}$ viene percorso da una corrente elettrica stazionaria $\overrightarrow{I}$, allora la forza complessiva agente sul conduttore è data dalla somma vettoriale delle forze di Lorentz agenti sugli elettroni in moto con velocità $\overrightarrow{v}$.

$$\overrightarrow{F} = -e \cdot \overrightarrow{v} \times \overrightarrow{B} \tag{2.4}$$

2.4.3 Limiti

Ovviamente questo sensore può essere influenzato da qualsiasi oggetto che produca un campo magnetico ma, di solito, ciò non è un problema: non sono molte, infatti, le situazioni in cui una persona possa trovarsi in situazioni del genere. Un problema meno trascurabile è invece il rumore di tipo gaussiano cui è soggetto.

2.5 La fotocamera

2.5.1 Panoramica

Generalmente ogni smartphone è provvisto di due fotocamere, una anteriore ed una posteriore, e di solito quella posteriore offre una risoluzione migliore. Sebbene siano state concepite per immortalare scene e preservare ricordi, possono essere utili anche per stimare il movimento del dispositivo confrontando due o più immagini consecutive acquisite durante lo spostamento.

2.5.2 Come funziona

La fotocamera è formata da due parti: il sensore e la lente.
Il sensore è quello che effettivamente "cattura" l'immagine ed è un circuito integrato complesso che include fotorilevatori, amplificatori, transistor e spesso anche una qualche forma di hardware processing e gestione dell'energia. La grandissima maggioranza dei sensori oggi si basa su tecnologia CMOS (*Complementary Metal-Oxide-Semiconductor*) in cui i vari fotorilevatori corrispondenti ad ogni pixel dell'immagine catturano informazioni riguardo i fotoni che li colpiscono e tali

informazioni vengono poi amplificate e convertite in un segnale digitale che ne descrive la luminosità. Per ottenere quindi un'immagine a colori, un filtro di Bayer RGBG (vedi *Figura 2.3*) viene disposto sopra i fotorilevatori ed un software d'interpolazione provvederà a creare il risultato finale; tanto più grande sarà il numero di fotorilevatori e maggiori saranno i megapixel della fotocamera.
Per quanto concerne la lente, essa concentra la luce sul sensore in modo che l'immagine risulti nitida. Senza una lente infatti si otterrebbe soltanto una scena molto sfocata e ciò è dovuto al fatto che sarebbe permesso ai fotoni di colpire il sensore da qualsiasi angolazione. La sua composizione (vedi *Figura 2.4*) consiste in una serie di elementi di plastica o, per una migliore qualità, di vetro che, muovendosi in avanti e indietro rispetto al sensore, deviano successivamente il percorso dei fotoni favorendo la messa a fuoco di un soggetto piuttosto che un altro.

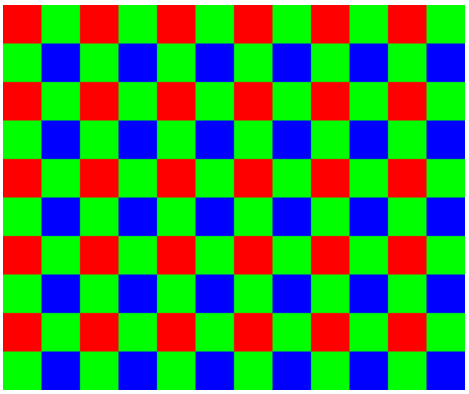


Figura 2.3: Schema RGBG di Bayer

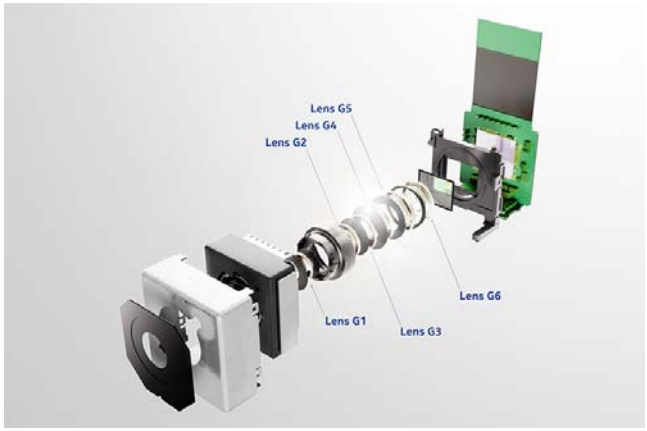


Figura 2.4: Esempio dei componenti di una lente all'interno di una fotocamera; figura presa da [33]

2.6 Il microfono

2.6.1 Panoramica

Trattandosi comunque di un telefono, allo smartphone non può mancare il microfono. Tutti i dispositivi ne hanno sempre almeno uno ed alcuni sono provvisti di un microfono extra per catturare solo i rumori ambientali e consentire così un filtraggio più accurato dei rumori di fondo indesiderati.

2.6.2 Come funziona

Le onde sonore altro non sono che un'oscillazione compiuta dalle particelle (atomi e molecole) in un mezzo, tipicamente l'aria. Lo scopo del microfono è quindi quello di catturare tale oscillazione grazie ad una membrana vibrante, detta diaframma, che si deforma a seconda del suono che riceve. Una spira fissata dietro tale membrana avvolge un magnete ed il suo movimento in avanti e indietro fa sì che si generi in essa una corrente indotta. In questo modo il segnale sonoro viene convertito in segnale elettrico che può dunque essere amplificato o comunque elaborato.

Capitolo 3

Lo studio della letteratura

Il primo passo verso la realizzazione di questo progetto è stato quello di dedicare molto tempo ad acquisire le tecniche proposte in letteratura per quanto riguarda la *motion estimation*, facendo particolare attenzione alle soluzioni con smartphone. Ciò che è emerso è che la fotocamera può essere molto utile per quantificare gli spostamenti subiti dal dispositivo. In moltissime ricerche infatti essa viene impiegata per estrarre dei punti altamente descrittivi dalle immagini acquisite, rendendo così possibile un confronto tra frame consecutivi. Così facendo, si può ricavare di quanto le due scene siano traslate l'una rispetto all'altra rendendo possibile la stima del moto o il suo problema complementare, cioè la stabilizzazione delle immagini. Per quanto riguarda invece l'accelerometro, molto spesso viene usato in combinazione con giroscopio e magnetometro per ricavare l'orientazione del dispositivo, ma questo verrà descritto nel capitolo successivo quando verrà introdotta la *sensor fusion*; in questo capitolo invece esso avrà un ruolo soltanto nella sezione 3.2.
Verranno qui di seguito riassunte le tecniche più significative proposte dai vari autori ed altri riferimenti utili sono consultabili nella bibliografia.

3.1 Stabilizzazione dell'immagine tramite le funzioni *Activation* e *MAD*

Il lavoro proposto dagli autori in [11] si propone di compensare quei piccoli movimenti non voluti durante una registrazione video per via di una mano poco ferma. Ciò viene fatto suddividendo la scena di input in due parti: primo piano e sfondo, come mostrato in *Figura 3.1*. Tali suddivisioni sono il risultato di una valutazione empirica di varie scene. Successivamente, dal confronto di due frame consecutivi, vengono valutati i singoli vettori di movimento per le due aree (*Figura 3.2*). Per calcolare il movimento globale di queste due zone, un istogramma quadrato rappresentante la frequenza dei vari vettori è più affidabile di due istogrammi lineari

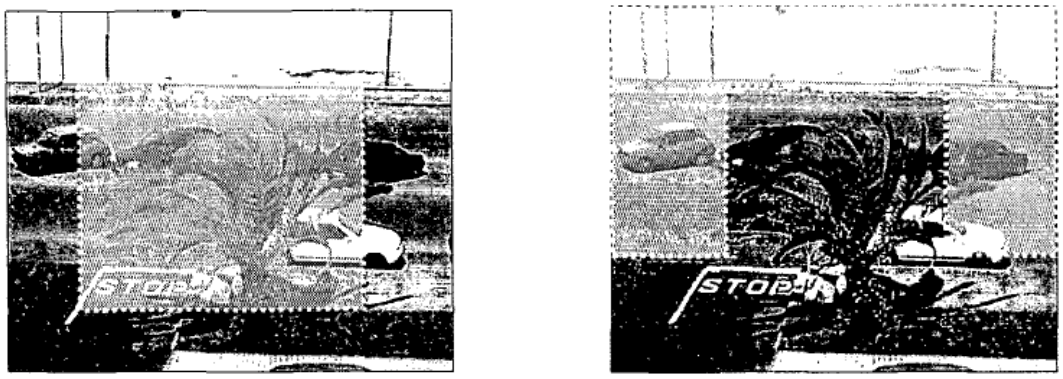


Figura 3.1: Stabilizzazione dell'immagine: selezione della zona di primo piano (a sinistra) e di sfondo (a destra); figura presa da [11]

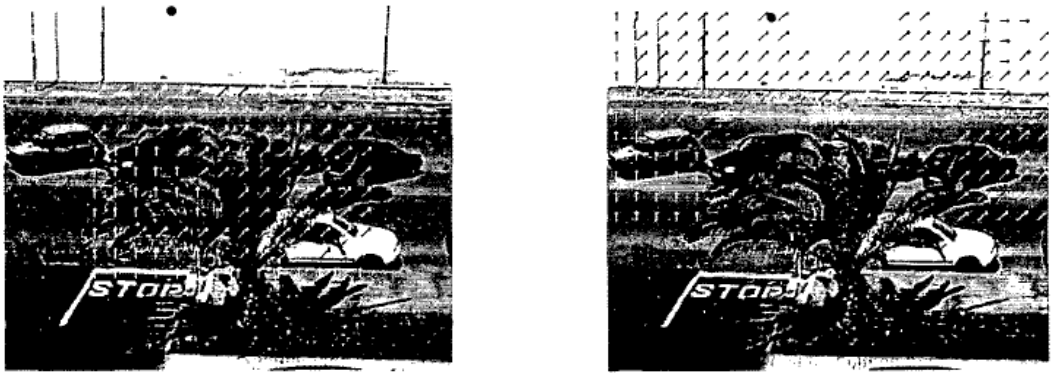


Figura 3.2: Vettori di movimento tra due frame consecutivi: a sinistra il primo piano e a destra lo sfondo; figura presa da [11]

da cui si può estrarre solo la componente più frequente. Ciò comunque non è sufficiente perché l'inclusione di vettori errati o provenienti da zone omogenee (come ad esempio il cielo) può portare a stime fuorvianti. É quindi utile distinguere i blocchi ad alta frequenza da quelli a bassa frequenza in modo da dare più importanza ai primi. Per fare ciò gli autori ricorrono a due funzioni : *Activation* e *Mean Absolute Difference(MAD)*, definite dalle equazioni 3.1 e 3.4 rispettivamente.

$$A = A_{horizontal} + A_{vertical} \tag{3.1}$$

$$A_{horizontal} = \sum_{i=0}^{N-1} \sum_{j=1}^{N-1} |Y_{i,j} - Y_{i,j-1}| \tag{3.2}$$

$$A_{vertical} = \sum_{j=0}^{N-1} \sum_{i=1}^{N-1} |Y_{i,j} - Y_{i-1,j}| \tag{3.3}$$

$$MAD = \frac{1}{N^2} \sum_{j=0}^{N-1} \sum_{i=0}^{N-1} |Y_{i,j} - \overline{Y}| \tag{3.4}$$

Nelle equazioni (3.1), (3.2) e (3.3), $Y_{i,j}$ è la luminanza del pixel del blocco indicizzato da i e j, N è la dimensione del blocco considerato e $\overline{Y}$ è la luminanza media nel blocco.
Il vantaggio di usare la (3.1) consiste nel fatto che il blocco è scansionato una volta sola, mentre il vantaggio di usare la (3.4) consiste nel fatto che essa è già presente nello standard *MPEG* e quindi parte del lavoro è già svolto.
Fatte quindi le dovute elaborazioni, vengono pesati i vari vettori di movimento in base ai risultati ottenuti con queste funzioni e il vettore di movimento globale prodotto dall'istogramma quadrato sarà più attendibile (vedi *Figura 3.3*). Infine, per

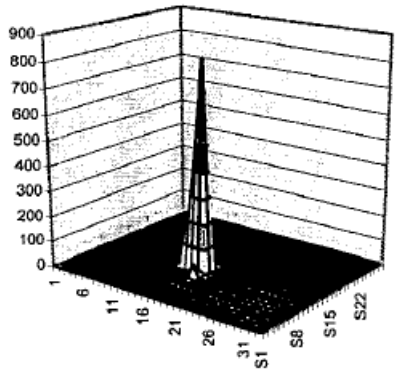


Figura 3.3: Istogramma pesato dei vettori di movimento; figura presa da [11]

decidere se correggere il movimento dello sfondo o quello del primo piano, viene preso in considerazione il numero di blocchi che producono il vettore di movimento globale: se è maggiore nello sfondo allora viene stabilizzato lo sfondo, altrimenti il primo piano.

3.2 NoShake

Sebbene lo scopo di *NoShake* (vedi [12]) sia lo stesso dell'algoritmo appena descritto, l'approccio qui risulta profondamente diverso. Gli autori infatti ignorano del tutto la fotocamera concentrandosi invece sulle letture dell'accelerometro. Il

modello presentato è quello di *Figura 3.4*, dove le immagini da visualizzare sono pensate come ancorate ad una molla e ad un ammortizzatore sia lungo l'asse X che lungo l'asse Y. Tale sistema può essere analizzato nei due assi in modo indi-

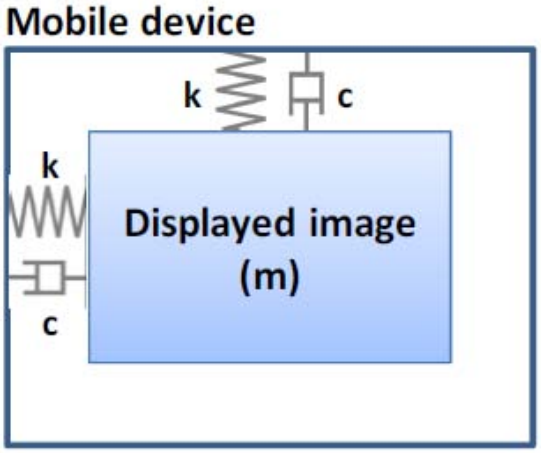


Figura 3.4: Modello con ammortizzatori e molle per la stabilizzazione dello schermo; figura presa da [12]

pendente come mostrato nell'equazione (3.5), dove k è la costante elastica della molla, c è il coefficiente di smorzamento, y è lo spostamento della massa m rispetto al dispositivo e x è lo spostamento del dispositivo dovuto allo scuotimento (vedi *Figura 3.5*).

$$m\ddot{y} + c\dot{y} + ky = -m\ddot{x} \tag{3.5}$$

Essendo che il rapporto di smorzamento $\zeta = c/\sqrt{km}$ influenza il comportamento del sistema, è opportuno notare che quando $\zeta < 1$ il sistema oscilla alla sua frequenza di risonanza e quando $\zeta > 1$ il sistema non oscilla ma richiede un tempo lungo per la stabilizzazione. La scelta è stata quindi quella di imporre $\zeta = 1$ e, per semplicità di conti, senza perdere generalità, $m = 1$. L'equazione del sistema diventa dunque la (3.6), dove $A(t)$ è l'accelerazione del dispositivo al tempo t, sostituendo così $\ddot{x}$.

$$\ddot{y} + 2\sqrt{k}\dot{y} + ky = -A(t) \tag{3.6}$$

Per calcolare la risposta del sistema $Y(t)$ è sufficiente convolvere l'input $A(t)$ con la risposta impulsiva del sistema $H(t)$ come riportato nell'equazione (3.7), essendo $H(t) = te^{-t\sqrt{k}}$.

$$Y(t) = H(t) * -A(t) \tag{3.7}$$

L'algoritmo funziona secondo lo schema di *Figura 3.6*: inizialmente viene rimossa la gravità con un semplice filtro passa alto in modo da avere soltanto le accele-

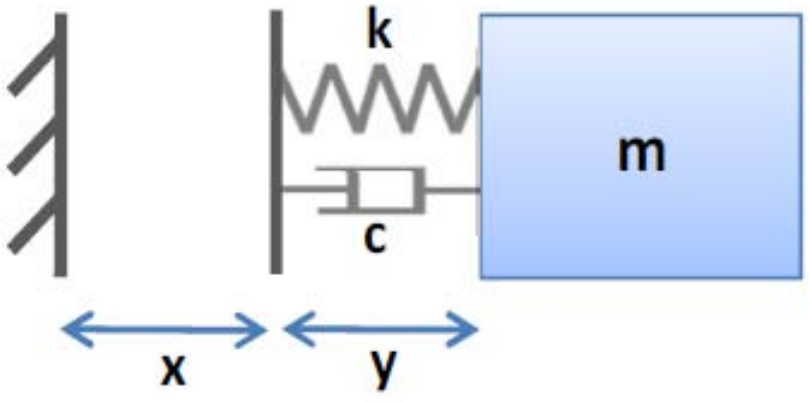


Figura 3.5: Modello fisico equivalente in una dimensione; figura presa da [12]

razioni dovute allo scuotimento, poi le letture vengono memorizzate in un buffer circolare per un utilizzo futuro. Se dalla valutazione dei valori ricevuti negli ultimi 1,8 secondi emerge la presenza di un movimento significativo, viene fatta la convoluzione con conseguente traslazione dell'immagine del display di una quantità proporzionale a $Y(t)$, determinata da un parametro di scala a. Sia a che k sono parametri personalizzabili che variano da dispositivo a dispositivo.

3.3 Stabilizzazione video tramite *Phase Correlation*

In [13] il problema della stima del moto viene risolto in modo concettualmente simile a come fatto in [11] ma con strumenti diversi. Come mostra la *Figura 3.7*, ogni frame viene diviso in quattro sottoimmagini a forma di quadrato avente il lato di lunghezza in pixel 2^n, con $n \in \mathbb{N}^+$; tipicamente $n = 6$. Questa suddivisione è fatta per determinare i vettori di movimento locali attraverso la correlazione di fase.
Siano G_1 e G_2 le trasformate discrete di Fourier (DFT) bidimensionali delle immagini g_1 e g_2 rispettivamente, per ogni componente di frequenza spaziale (u, v) la *cross power spectrum* normalizzata contenente informazioni sulla differenza di fase è definita secondo (3.8), dove " * " denota il complesso coniugato. Nel caso in cui l'immagine g_2 sia una versione traslata di g_1 allora deve valere la (3.9), essendo α un'eventuale differenza di contrasto e d_x e d_y lo spostamento orizzontale e verticale rispettivamente.

$$e^{j(\Phi_1(u,v)-\Phi_2(u,v))} = \frac{G_1(u,v) \cdot G_2^*(u,v)}{|G_1(u,v) \cdot G_2^*(u,v)|} \tag{3.8}$$

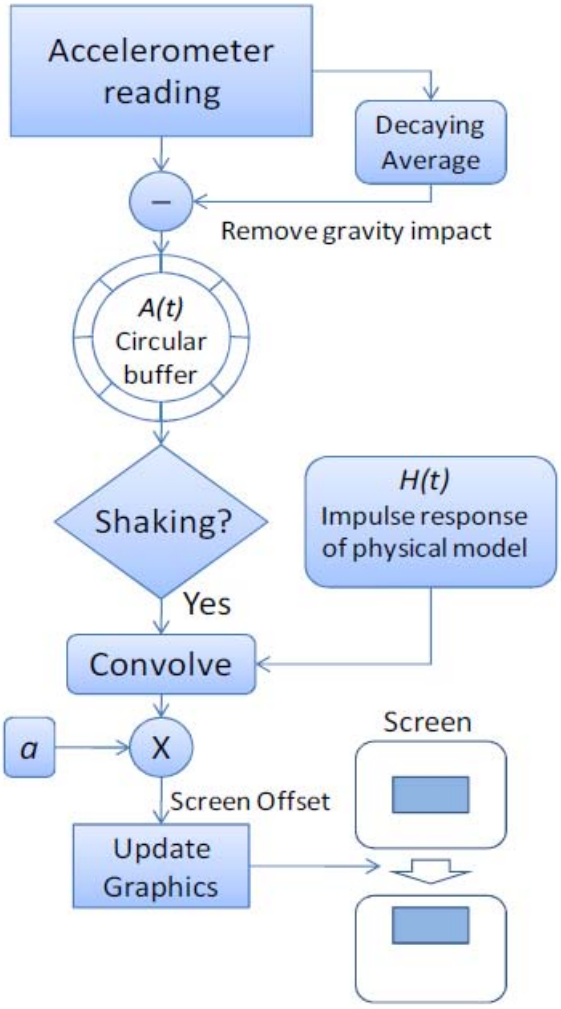


Figura 3.6: Rappresentazione grafica dell'algoritmo di NoShake; figura presa da [12]

$$g_2(x, y) = \alpha \cdot g_1(x - d_x, y - d_y) \tag{3.9}$$

La correlazione di fase $P(x, y)$ è definita quindi come l'inversa della trasformata discreta di Fourier (IDFT) della *cross power spectrum* normalizzata, come da equazione (3.10), dove F^{-1} denota l'operatore IDFT. Si deduce quindi che nel caso in cui g_2 sia una versione traslata di g_1, allora questa funzione risulta nulla ovunque a meno di una funzione δ in $(-d_x, -d_y)$, corrispondente allo spostamento.

$$P(x, y) = F^{-1}(e^{j\Phi_1(u,v)-\Phi_2(u,v)}) = F^{-1}(e^{j(ud_x+vd_y)}) = \delta(x + d_x, y + d_y) \tag{3.10}$$

Una proprietà interessante di questa tecnica è il fatto che l'ampiezza del picco è un ottimo indicatore dell'affidabilità del vettore di spostamento rilevato: maggiore

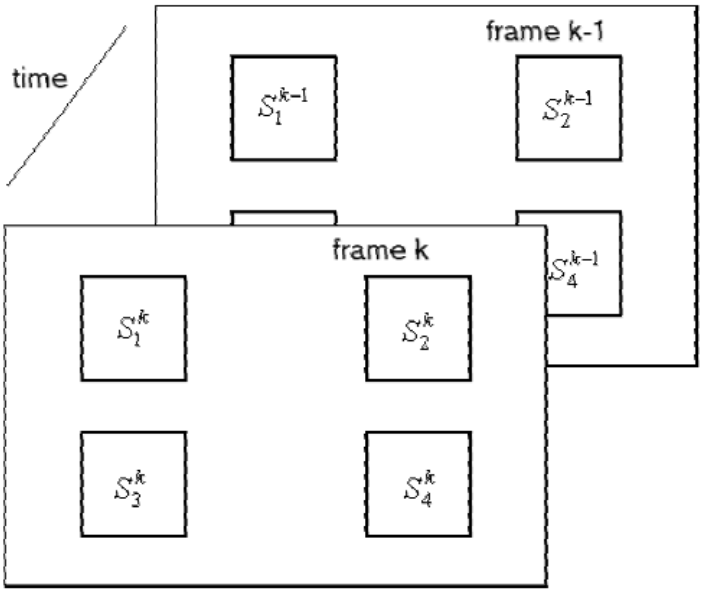


Figura 3.7: Locazione delle quattro sottoimmagini usate per la stima del moto; figura presa da [13]

sarà tale picco e maggiore sarà l'affidabilità della traslazione dedotta.
Calcolando la $P(x, y)$ per ognuna delle quattro sottoimmagini, gli autori propongono tre diversi approcci per stimare la traslazione globale, ossia:

- far coincidere il vettore di spostamento globale col vettore di spostamento locale avente la più elevata ampiezza di picco
- prendere i due vettori di spostamento locale con più alta ampiezza di picco e farne la media
- prendere tutti i vettori di spostamento locale la cui ampiezza di picco sia superiore ad una certa soglia e farne la media pesata secondo l'ampiezza di picco stessa

Qualsiasi di queste tre possibilità si scelga, rimane sempre il problema che, affidandosi solamente a $P(x, y)$ frame dopo frame, l'output sarà sempre affetto da alte frequenze indesiderate per cui è opportuno trovare un sistema che "levighi" questi risultati nel tempo. A questo scopo gli autori propongono l'utilizzo del filtro di Kalman riportato in (3.11), dove la coppia (x_1, x_2) rappresenta la posizione del frame (in orizzontale per la prima componente e in verticale per la seconda) e la coppia (dx_1, dx_2) rappresenta la corrispondente velocità. In particolare, la prima equazione rappresenta la predizione dello stadio con cui viene stimato lo stato e

la seconda rappresenta lo stadio di aggiornamento con la quale il filtro riceve un feedback sulla predizione.
Ulteriori approfondimenti sul filtro di Kalman sono esposti nella sezione 4.2.2.

$$\begin{cases} \begin{bmatrix} x_1(n) \\ x_2(n) \\ dx_1(n) \\ dx_2(n) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1(n-1) \\ x_2(n-1) \\ dx_1(n-1) \\ dx_2(n-1) \end{bmatrix} + \begin{bmatrix} wx_1 \\ wx_2 \\ wdx_1 \\ wdx_2 \end{bmatrix} \\ \\ \begin{bmatrix} z_1(n) \\ z_2(n) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1(n) \\ x_2(n) \\ dx_1(n) \\ dx_2(n) \end{bmatrix} + \begin{bmatrix} vx_1 vx_2 \end{bmatrix} \end{cases} \tag{3.11}$$

3.4 Stabilizzazione video tramite SIFT Keypoints

Il paper [14] si occupa ancora una volta della stabilizzazione di immagini video e in questo caso viene impiegata la *Scale Invariant Feature Transform*, cioè una funzione concepita per estrarre feature invarianti ed altamente descrittive di un'immagine (vedi capitolo 4 di [15] e capitolo 10 di [16]). L'elaborazione viene fatta su due frame consecutivi di una sequenza video in modo da prima calcolare la posizione delle feature e poi utilizzare un algoritmo di *feature matching* per associarle. Per fare ciò, tale algoritmo confronta i vettori descrittivi di ogni keypoint così da ottenere il vettore di spostamento locale per ogni associazione riuscita (vedi *Figura 3.8*). Il movimento totale può essere calcolato, almeno idealmente, tramite il sistema a due equazioni (3.12), dove la coppia (x_i, y_i) rappresenta la posizione della feature in un frame e (x_f, y_f) rappresenta la posizione della stessa feature ma nel frame successivo, λ è il parametro di rotazione, θ è l'angolo di rotazione, T_x è lo spostamento lungo l'asse X e T_y è lo spostamento lungo l'asse Y.

$$\begin{cases} x_f = x_i \lambda cos\theta - y_i \lambda sin\theta + T_x \\ y_f = x_i \lambda sin\theta + y_i \lambda cos\theta + T_y \end{cases} \tag{3.12}$$

Basterebbero dunque solo due associazioni per risolvere il sistema ma, per via del rumore che potrebbe affliggere i match, è utile applicare il metodo dei minimi quadrati a un set di equazioni ridondanti. L'algoritmo proposto è quindi il seguente. Inizialmente ogni associazione che presenti un vettore di spostamento troppo elevato viene scartata. Poi tutti i vettori di traslazione locali vengono elaborati usando il metodo dei minimi quadrati e contemporaneamente le feature sono tracciate

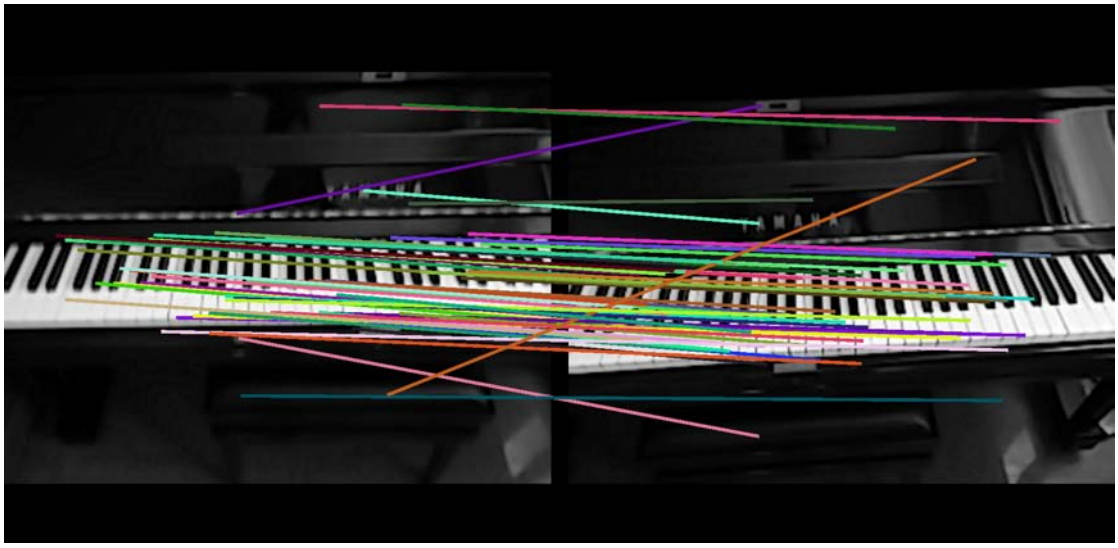


Figura 3.8: Associazione dei SIFT Keypoints in due frame consecutivi; non tutte le associazioni sono affidabili

lungo i vari frame. Feature note presenti in più frame saranno infatti più affidabili delle altre consentendo una maggiore accuratezza nella stima. Fatto ciò, ogni keypoint viene confrontato con i parametri trovati per trovare due tipi d'errore:

- Distanza Euclidea tra il punto atteso e quello reale
- Angolo rotazionale tra il punto atteso e quello reale

Naturalmente un errore troppo grande in almeno uno dei due controlli implica lo scarto dell'associazione in questione.
Quando una feature viene tracciata lungo frame consecutivi, il suo errore relativo è accumulato in modo da essere usato nel calcolo dei minimi quadrati. Più precisamente, se una feature k compare per la prima volta nel frame n, allora vengono valutati e tenuti i suoi errori ma tale feature non viene considerata nel calcolo dei minimi quadrati; se invece k è stata precedentemente tracciata anche nel frame $n-1$ ed ha un errore cumulativo E_{n-1}^{cum}, allora viene valutato il suo errore corrente $E_n(k)$ e viene poi effettuata l'interpolazione lineare per aggiornare gli errori di entrambe le misure come riportato in (3.13), essendo α un parametro empirico impostato a 0.35.

$$E_n^{cum}(k) = (1-\alpha)E_{n-1}^{cum}(k) + \alpha E_n^{cum} \tag{3.13}$$

Questa funzione è utile per discriminare ulteriormente i vettori affidabili da quelli meno affidabili. Vengono infatti ordinate le feature in base all'errore ottenuto e soltanto la prima metà di queste (con errore più basso) viene data in input al

metodo dei minimi quadrati. Essendoci due tipi di errori da calcolare, ogni feature deve soddisfare questa condizione per entrambi. Un esempio d'impatto che questo approccio ha è consultabile in *Figura 3.9*. Il vettore di movimento integrato del

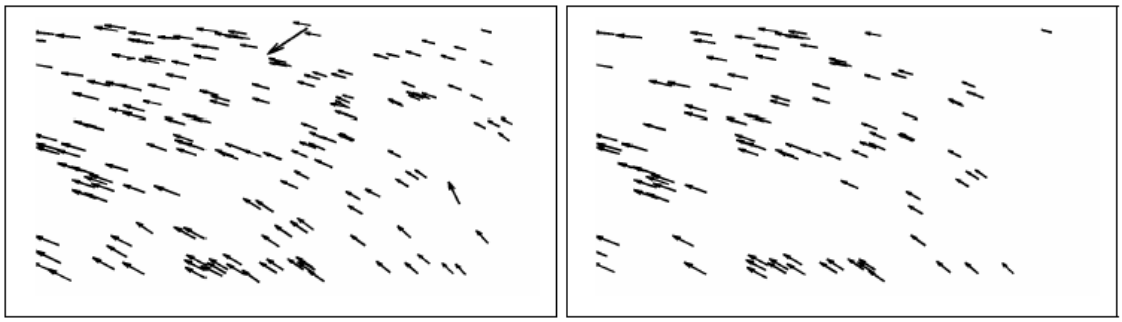


Figura 3.9: Esempio di filtraggio valutando l'errore; a sinistra i vettori non filtrati e a destra quelli rimasti dopo la computazione; figura presa da [14]

frame n $IMV(n)$, cioè lo spostamento tra il frame 0 ed il frame n, è ricavato con la (3.14), dove δ è un parametro di smorzamento compreso tra 0 e 1 e $GMV(n)$ è il vettore di movimento globale tra il frame n ed il frame $n-1$. La compensazione $C(n)$ da applicare al frame n, risultato finale di tutto questo lavoro, si ricava invece tramite la (3.15).

$$IMV(n) = \delta IMV(n-1) + GMV(n) \tag{3.14}$$

$$C(n) = IMV(n) + IMV(n-1) \tag{3.15}$$

3.5 Altre letture

I principali lavori che effettivamente potrebbero essere applicati al nostro progetto sono quelli appena descritti, tuttavia altri articoli scientifici hanno dimostrato di essere di un certo interesse per la nostra ricerca. Di seguito sono quindi riportati tali ricerche assieme ad un elenco di spunti utili in essi contenuti.

3.5.1 A Robust Block-Based Image/Video Registration Approach for Mobile Imaging Devices [17]

- Viene proposto un altro algoritmo simile a [14] ed i suoi passaggi principali sono esposti in *Figura 3.10*. Esso è formato da:
  - *Block Selector*: Seleziona dall'immagine alcuni blocchi quadrati da cui ricavare i vettori di movimento locali; tale selezione ha sia una componente storica (è ragionevole supporre che blocchi affidabili nel frame

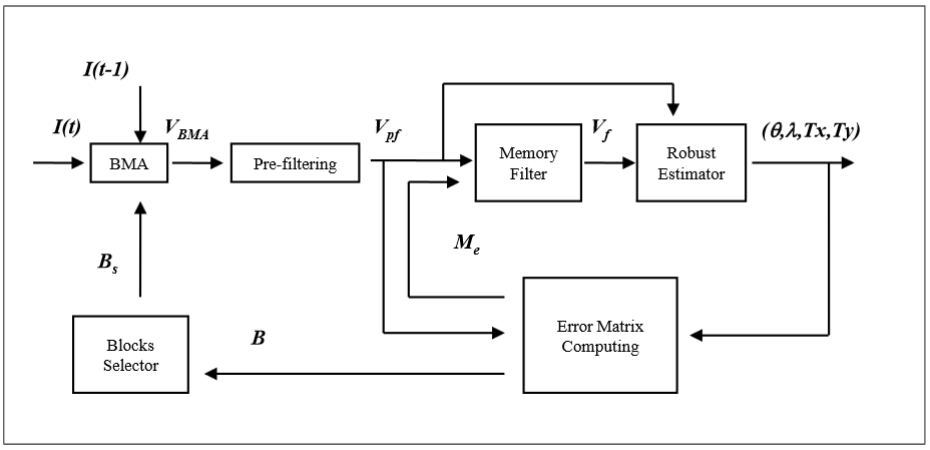


Figura 3.10: Architettura dell'algoritmo di motion estimation proposto in [17]; figura presa da [17]

n rimangano tali anche nel frame $n+1$) che una componente random, ed il numero totale di blocchi non deve superare una certa soglia che dipende dalla CPU, dalla velocità ecc.

- *BMA*: Ricava i vettori locali di movimento a partire dai blocchi precedentemente selezionati.
- *Per-filtering*: Elimina alcuni vettori trovati ritenuti poco affidabili in base alla bontà del matching, all'omogeneità del blocco associato e alla somiglianza coi vettori vicini.
- *Memory Filter*: Per evitare che oggetti in movimento (anche di grandi dimensioni rispetto all'intera immagine) contribuiscano a calcolare il movimento totale, viene fatta una divisione tra vettori presenti anche nella precedente computazione e vettori nuovi; i primi sono considerati più affidabili. Effettuando un filtraggio casuale per ragioni di leggerezza computazionale e valutando l'*Error Matrix Computing* dei vettori più vecchi per filtrare ulteriormente, viene creata la lista finale dei vettori utili alla stima del moto globale.
- *Robust Estimator*: I vettori di input vengono utilizzati per formare un sistema ridondante di equazioni come la (3.12) e viene quindi calcolata la stima della soluzione attraverso il metodo dei minimi quadrati. Fatto ciò, vengono calcolate due funzioni d'errore per ogni vettore, ossia la distanza bidimensionale (3.16) e il quadrato del coseno dell'angolo tra il movimento stimato e quello misurato (3.17), in modo da poter selezionare soltanto quelli con basso errore e rieffettuare il metodo dei minimi quadrati su questi.

– *Error Matrix Computing*: Per ogni vettore valido per il frame precedente viene calcolata la distanza euclidea tra il vettore di movimento locale, ottenuto da BMA, e quello ottenuto dalla stima dei parametri λ, θ, T_x e T_y. Così facendo, è possibile distinguere i vettori che appartengono alla scena in movimento da quelli che appartengono ad oggetti che stanno entrando in scena.

$$E_1 = (x_s - x_f)^2 + (y_s - y_f)^2 \tag{3.16}$$

$$E_2 = \frac{((x_s - x_i) \cdot (x_f - x_i) + (y_s - y_i) \cdot (y_f - y_i))^2}{((x_s - x_i)^2 + (y_s - y_i)^2) \cdot ((x_f - x_i)^2 + (y_f - y_i)^2)} \tag{3.17}$$

Nota: (x_i, y_i) è il centro del blocco (relativo a un vettore) nel frame $n-$, (x_f, y_f) è la posizione del blocco nel frame n e (x_s, y_s) è la posizione stimata prima di calcolare E_1 ed E_2.

3.5.2 Visual Odometry [18]

- Una stima robusta del movimento deve partire da un set di dati il più possibile privo di outliers.

- La calibrazione della fotocamera potrebbe essere un passaggio utile per evitare che eccessive distorsioni dell'immagine compromettano la stima.

- Le relazioni geometriche tra due immagini di una fotocamera calibrata sono descritte dall' *essential matrix* E (vedi (3.18)), dove $\hat{t}_k$ è la matrice di traslazione e R_k è la matrice di rotazione che racchiude i parametri di movimento (traslazione e rotazione) a meno di un fattore di scala; quest'ultima può essere calcolata tramite l'associazione di due feature in frame differenti.

$$E_k \propto \hat{t}_k R_k, \qquad \hat{t}_k = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} \tag{3.18}$$

- Un confronto tra diverse feature è riportato in *Figura 3.11*: dalla tabella emerge che le *SIFT features* sono poco consigliate per le situazioni *real time* con limitata capacità di elaborazione, come quella da noi studiata.

	Corner Detector	Blob Detector	Rotation Invariant	Scale Invariant	Affine Invariant	Repeatability	Localization Accuracy	Robustness	Efficiency
Haris	x		x			+++	+++	++	++
Shi-Tomasi	x		x			+++	+++	++	++
FAST	x		x	x		++	++	++	++++
SIFT		x	x	x	x	+++	++	+++	+
SURF		x	x	x	x	+++	++	++	++
CENSURE		x	x	x	x	+++	++	+++	+++

Figura 3.11: Confronto tra rilevatori di feature: proprietà e performance; figura presa da [18]

3.5.3 Interactivity for Mobile Music-Making [19]

- Un algoritmo di *optical flow* è in grado di rilevare i movimenti lineari dello smartphone nel piano dello schermo e i movimenti rotazionali lungo l'asse ottico.
- Il tempo di risposta degli accelerometri integrati negli smartphone è molto breve ed è un valido sensore per scopi musicali dal momento che non c'è alcun ritardo sensibile tra movimento ed output del sensore.

Capitolo 4

Tecniche utilizzate

Poiché dallo studio della letteratura è emerso che, per quanto riguarda la fotocamera, l'individuazione di *feature* nell'immagine costituisce un passaggio quasi imprescindibile, abbiamo deciso esplorare i comportamenti di tre loro tipologie, ossia *FAST*, *Good Features to Track* e *Speed Up Robust Features*. Queste sono state scelte in base alla loro differenziazione per velocità di elaborazione ed affidabilità, intesa come la capacità di rilevare in scene leggermente diverse sempre gli stessi agganci. Molto spesso infatti non è disponibile potenza di calcolo sufficiente per elaborare le proprietà di ogni pixel in tempi utili, quindi è necessario individuare su ogni frame dei punti chiave (*keypoints*) che permettano di confrontare scene diverse. In questo modo il carico computazionale viene considerevolmente ridotto e, in genere, ciò non ha un significativo impatto sulla qualità del risultato finale. Siccome ci serviva inoltre anche un metodo per associare le *feature* dedotte da immagini differenti, abbiamo preso in considerazione una tecnica molto consolidata ed affidabile: il così detto "metodo Lucas-Kanade".
Per quanto riguarda gli altri sensori invece, abbiamo constatato che l'unica strada percorribile era quella della *sensor fusion* poiché i dati rilevati da un unico sensore risultano essere troppo disturbati dai vari errori visti nel capitolo 2. Per ottenere poi risultati ancora più affidabili, abbiamo fatto ricorso anche al filtro di Kalman: si tratta di una tecnica altamente studiata e documentata in letteratura il cui scopo, molto generalmente, è quello di ridurre il rumore che affligge il segnale di input.
Naturalmente, tutte le varie tecniche descritte nel capitolo 3 sono ugualmente valide ma, per motivi di tempo, non abbiamo avuto modo di esplorarle tutte quante. Ciò che invece abbiamo potuto fare è stato decidere di seguire una strada e prendere spunti da un po' tutti i lavori là dove possibile, tenendo sempre in mente le limitate risorse del nostro contesto.

4.1 Tecniche per la fotocamera

4.1.1 FAST

Come suggerisce il nome, questa strategia, descritta in [26], è una di quelle più veloci al momento disponibili. Ogni pixel p di intensità I_p viene valutato in base ad una soglia T, per esempio impostata al 20% di I_p, e ad una circonferenza di 16 pixel attorno a p come mostra la *Figura 4.1*. Se almeno N su 16 pixel contigui hanno valore d'intensità maggiore di $I_p + T$ o minore di $I_p - T$, allora p può essere considerato punto d'interesse. In genere il valore N viene posto pari a 12.

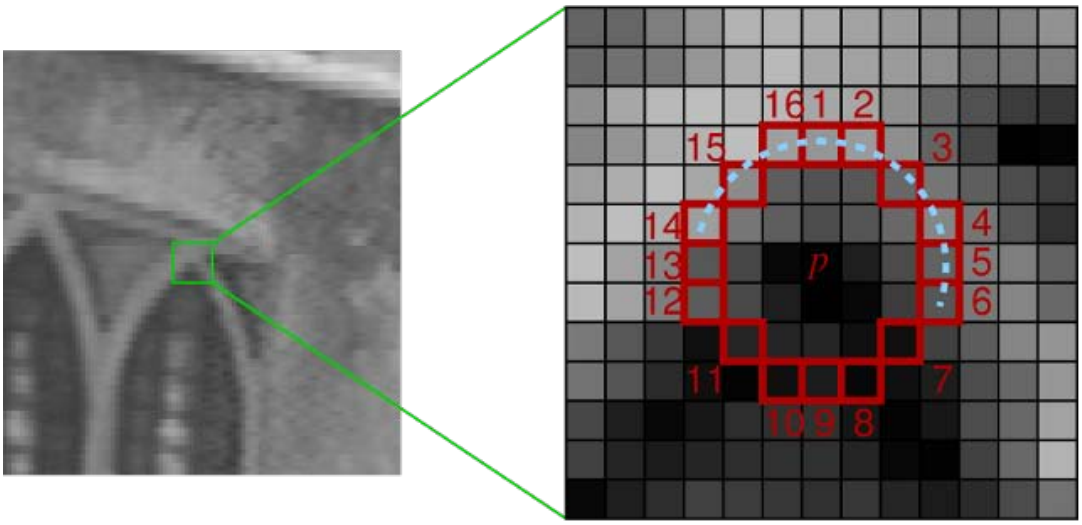


Figura 4.1: FAST Feature Detector: il pixel p è quello in esame mentre i pixel numerati da 1 a 16 sono quelli che determineranno se p è o meno un punto d'interesse; figura presa da [38]

4.1.2 Good Features to Track

Molto spesso in letteratura, quando si parla di flusso ottico di una scena ripresa (vedi sezione 4.1.4), si utilizzano queste feature descritte nell'omonimo paper [25] per individuare i punti su cui calcolare il flusso. In questo caso, se A è la matrice che rappresenta la derivata delle luminosità trovate lungo i due assi (vedi più avanti l'equazione (4.9)), allora tutte le considerazioni vengono svolte sui due autovalori di AA^T λ_1 e λ_2 che s'impongono superiori ad una data soglia λ, come mostra la (4.1). Il loro rapporto inoltre non deve essere eccessivamente elevato perché altrimenti ciò vorrebbe dire che il punto in questione è su di un lato, portando così al problema dell'apertura precedentemente menzionato; deve quindi valere anche la disequazione (4.2).

$$min(\lambda_1, \lambda_2) > \lambda \tag{4.1}$$

$$\frac{\lambda_1}{\lambda_2} < threshold \tag{4.2}$$

Il tracker basato sul metodo Lucas-Kanade che ricorre alle *Good Features to Track* viene detto "KLT-tracker".

4.1.3 Speed Up Robust Features

Per calcolare i punti d'interesse, il primo passo compiuto da quest'algoritmo descritto in [29] è quello di filtrare l'immagine con un filtro quadrato per ridurne il rumore: ogni pixel p_i viene quindi sostituito col valore medio dei pixel contenuti in un quadrato centrato in p_i .Tale filtro è già di per sé più veloce di tanti altri ma un altro vantaggio che offre è la possibilità di parallelizzare l'operazione per diverse scale della stessa immagine. Per selezionare poi la scala e la posizione, viene usato il determinante della matrice essiana; dato un punto p ed una scala σ, tale matrice è definita come mostrato nell'equazione (4.3), dove $L_{xx}(p, \sigma)$ ecc. sono le derivate del secondo ordine dell'immagine in scala di grigi.

$$H(p, \sigma) \equiv \begin{pmatrix} L_{xx}(p, \sigma) & L_{xy}(p, \sigma) \\ L_{xy}(p, \sigma) & L_{yy}(p, \sigma) \end{pmatrix} \tag{4.3}$$

Invece che cambiare la dimensione dell'immagine, per trovare punti d'interesse su diverse scale vengono applicati filtri di dimensione sempre più grande (9×9, 15×15, 21×21, 27×27 ecc.) ed infine vengono interpolati nella scala e nello spazio i valori massimi dei determinanti delle varie matrici essiane.
Per ottenere l'orientazione di ogni feature vengono calcolate le risposte Wavelet in entrambe le direzioni (x e y) entro un vicinato circolare di raggio $6s$, essendo s è la scala in cui è stata localizzata la feature. Tali risposte sono quindi pesate tramite una funzione gaussiana centrata nella feature e rappresentate su di un piano cartesiano avente nelle ascisse la risposta orizzontale e nelle ordinate quella verticale. L'orientazione dominante viene infine stimata sommando tutte le risposte entro un angolo di 60° come mostra la *Figura 4.2*.
Per la costruzione dei descrittori, che forniscono un'unica e robusta descrizione di una feature, viene estratta una regione grande $20s$ centrata nella feature ed orientata con l'orientazione precedentemente ricavata. Tale regione è quindi suddivisa in aree più piccole, 4×4, e per ognuna di queste vengono calcolate le risposte Wavelet ottenendo un vettore così formato: $v = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|)$. Il vettore totale avrà allora 64 dimensioni e sarà usato per stabilire se due feature di immagini distinte rappresentano lo stesso punto nel mondo.

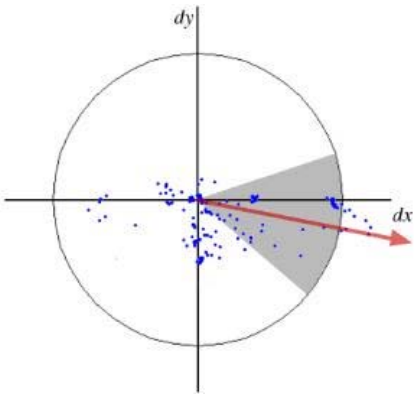


Figura 4.2: Esempio di come viene calcolato l'orientamento di un punto d'interesse con *SURF*; figura presa da [39]

4.1.4 Il metodo Lucas-Kanade

Nello studio delle immagini, ed in particolare di una sequenza d'immagini, un ruolo importante lo ricopre la stima del movimento che un oggetto (o una feature) ha compiuto tra un frame e quello successivo. A tal proposito si parla di *optical flow*, cioè di "flusso ottico": si tratta di determinare vettori di moto su ogni pixel (o un sottoinsieme di essi), come mostrato in *Figura 4.3*. Ciò che il metodo Lucas-Kanade si propone di fare è proprio questo.

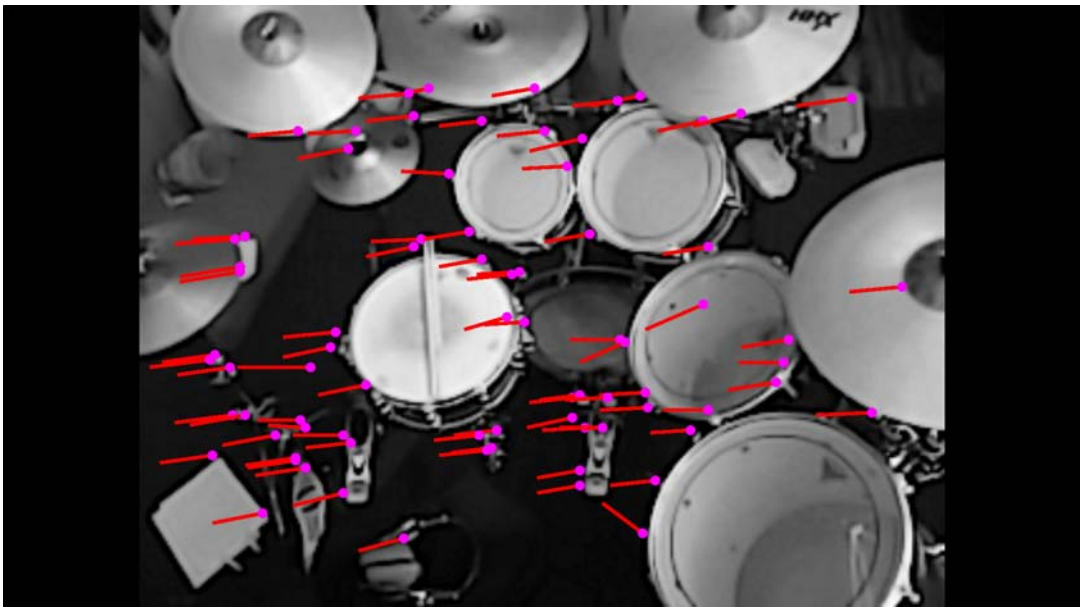


Figura 4.3: Optical flow: i vettori rappresentano lo spostamento delle feature

Le assunzioni alla base dell'algoritmo sono le seguenti:

- Luminosità costante: un pixel di un oggetto in movimento non cambia tra un frame e l'altro; nelle immagini grige, che poi saranno l'input della funzione implementata, ciò significa che la luminosità non cambia, ossia che vale la (4.4), e quindi la (4.5), dove $I(x, y, t)$ è la luminosità del pixel (x, y) al tempo t.

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \tag{4.4}$$

$$\frac{\partial I(x, y)}{\partial t} = 0 \tag{4.5}$$

- Piccoli movimenti: gli intervalli di tempo tra un frame e il successivo sono sufficientemente piccoli da non permettere che l'oggetto in questione si sposti troppo rispetto alla posizione precedente. Nel caso di una singola dimensione ciò significa che deve valere la (4.6), dove I_x è la derivata spaziale lungo la prima immagine, I_t è la derivata tra le immagini lungo il tempo e $\mathbf{v}$ è la velocità da trovare; per ottenerla è sufficiente prendere la (4.4), riadattarla nel caso monodimensionale ed applicare la regola della catena per le derivate parziali.

$$\underbrace{\left.\frac{\partial I}{\partial x}\right|_t}_{I_x} \underbrace{\left(\frac{\partial x}{\partial t}\right)}_{\mathbf{v}} + \underbrace{\left.\frac{\partial I}{\partial t}\right|_{x(t)}}_{I_t} = 0 \tag{4.6}$$

  Allora, rimanendo sempre nel caso monodimensionale, ciò vuol dire che la velocità si può ricavare con la (4.7).

$$\mathbf{v} = -\frac{I_t}{I_x} \tag{4.7}$$

  Purtroppo però il mondo reale risulta essere sempre un'approssimazione di quello ideale ed infatti è sperimentalmente provato che la luminosità non è davvero costante e a volte il frame rate non è sufficientemente veloce. Questo porta ad una stima inesatta della velocità (spostamento della feature da un frame all'altro), tuttavia, se essa rientra entro ragionevoli limiti, è possibile iterare il processo così da ottenere una soluzione più verosimile, ricalcolando ogni volta la I_t. Se la prima stima è sufficientemente buona basteranno circa cinque iterazioni (risultato empirico) ma, al contrario, se ciò non è vero

l'algoritmo divergerà. Tale procedimento è noto in letteratura come "metodo di Newton".

Nel caso bidimensionale la (4.6) diventa l'equazione (4.8) dove, cambiando notazione, u è la componente x della velocità e v la componente y della velocità: ogni pixel ha così due incognite. Quest'equazione descrive così una linea, riportata in *Figura 4.4*, e quindi può essere risolta solo la componente perpendicolare (normale) a tale linea.

$$I_x u + I_y v + I_t = 0 \tag{4.8}$$

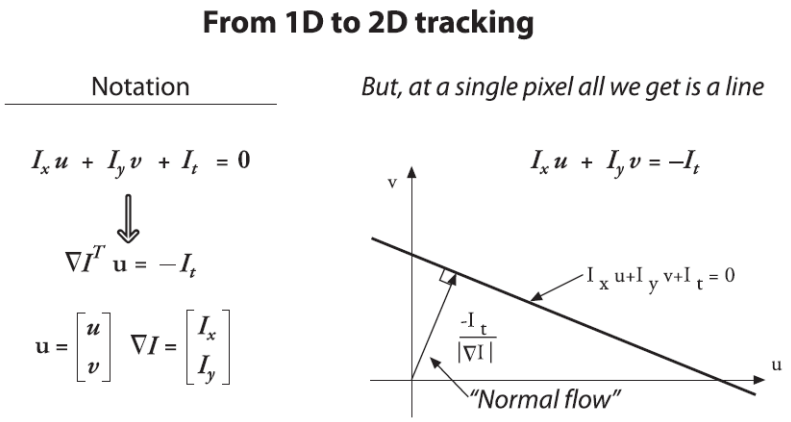


Figura 4.4: Optical flow bidimensionale in un singolo pixel; figura presa da [16]

Questa situazione è facilmente paragonabile al problema dell'apertura riportato in *Figura 4.5*: quando il movimento è rilevato attraverso una piccola apertura, quel che si vede di solito è un lato, non uno spigolo, e ciò non è sufficiente per determinare il movimento nella sua completezza. Fortunatamente l'ultima assunzione permette di aggirare quest'ostacolo.

- Coerenza spaziale: punti vicini che appartengono allo stesso oggetto hanno un movimento simile e vengono proiettati in punti vicini nel piano dell'immagine. Allora, se un gruppo di pixel verifica quest'ipotesi, è possibile creare un sistema ridondante di equazioni che permette di superare lo stallo del punto precedente. Se per esempio s'imposta una finestra 5×5 attorno al pixel corrente per i valori di luminosità, si possono impostare 25 equazioni come mostra la (4.9).

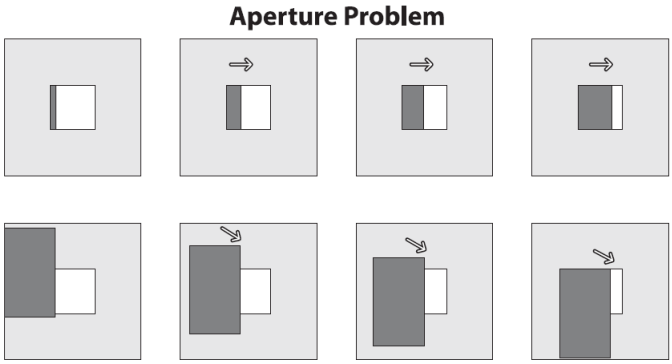


Figura 4.5: Problema dell'apertura: non è possibile stimare il moto di un lato che si muove dall'alto verso il basso e da sinistra a destra; figura presa da [16]

$$\underbrace{\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} u \\ v \end{bmatrix}}_{d} = \underbrace{\begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_{25}) \end{bmatrix}}_{b} \tag{4.9}$$

Tale sistema è allora risolvibile tramite la minimizzazione dei minimi quadrati, cioè risolvendo la (4.10), e così la soluzione che si ottiene è rappresentata dall'equazione (4.11).

$$(AA^T)d = A^T b \tag{4.10}$$

$$\begin{bmatrix} u \\ v \end{bmatrix} = (A^T A)^{-1} A^T b \tag{4.11}$$

Naturalmente, la (4.11) può essere risolta quando AA^T è invertibile, ossia quando ha rango pieno uguale a due, il che accade nelle finestre che comprendono oggetti che si stanno muovendo in due direzioni.
Un altro problema da considerare è inoltre il fatto che spesso nemmeno l'ipotesi di piccoli movimenti è valida. L'algoritmo appena descritto infatti non funzionerebbe bene nei casi reali ed è per questo che si è soliti far ricorso alla tecnica dell'immagine piramidale, illustrata in *Figura 4.6*. L'immagine di partenza viene sfocata e sottocampionata più volte, così da ottenere più immagini di diverse risoluzioni a partire dalla stessa scena. A questo punto, partendo dal livello più alto della piramide, cioè quello con minor dettaglio,

si calcola il flusso ottico ed il movimento risultante viene usato come punto di partenza per il livello successivo, cioè quello sottostante, fino a raggiungere il livello 0. Così facendo viene minimizzata la violazione di quest'ultima assunzione e si può dunque tener traccia di movimenti più veloci ed ampi.

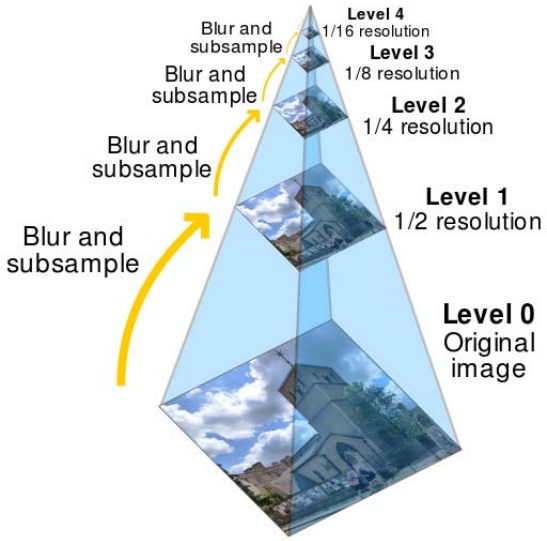


Figura 4.6: Immagine piramidale: riducendo il dettaglio dell'immagine è possibile ridurre i problemi causati dalla violazione dell'assunzione di piccoli movimenti; figura presa da [37]

4.2 Tecniche per gli altri sensori

4.2.1 Sensor Fusion

Come già accennato, accelerometro, giroscopio e magnetometro non sono molto affidabili se presi singolarmente per via del rumore che li affligge. Fortunatamente però, unendo i loro vari punti di forza, è possibile ottenere un output molto più robusto ed utile rispetto all'utilizzo di una singola fonte di dati. Questo approccio viene chiamato in letteratura *sensor fusion* e consiste appunto nell'utilizzare contemporaneamente sensori di natura diversa per stimare in modo più accurato una stessa grandezza. Come testimoniano ad esempio [20], [21], [22] e [23], l'idea più utilizzata di "*Sensor Fusion*" riguarda l'utilizzo simultaneo di accelerometro, giroscopio e magnetometro in modo da ottenere l'orientazione assoluta del dispositivo. Il risultato di questo lavoro sinergico viene poi spesso trattato con un filtro di Kalman (vedi sezione 4.2.2) in modo da avere una funzione finale "levigata" nel tempo.

In linea teorica sarebbero sufficienti soltanto accelerometro e magnetometro per determinare l'orientazione del dispositivo dal momento che il primo fornisce il vettore gravitazionale, cioè il vettore che punta verso il centro della Terra, e il secondo funge da bussola. Le informazioni di entrambi i sensori sarebbero quindi sufficienti per calcolare l'orientamento ma purtroppo il rumore che affligge gli output obbliga a percorrere altre strade.
Fortunatamente il giroscopio risulta essere molto più accurato e con un tempo di risposta molto breve per cui anche questo sensore riesce a fornire informazioni utili. Idealmente basterebbe integrare nel tempo i valori di tutti i risultati che fornisce ma il drift cui è soggetto distorcerebbe ancora una volta il risultato finale. Piccoli errori verrebbero infatti integrati di volta in volta portando in breve tempo ad una stima dell'orientamento inaccettabile.
Allora, per evitare sia i disagi creati dal rumore che quelli creati dal drift, l'output del giroscopio viene impiegato solamente per cambiamenti di orientamento in piccoli intervalli di tempo mentre i risultati dell'accelerometro e del magnetometro vengono usati come informazioni di supporto lungo periodi di tempo maggiori. Ciò equivale ad applicare un filtro passa-basso ai segnali dell'accelerometro e del magnetometro, e un filtro passa-alto al segnale del giroscopio. Lo schema concettuale risultante è quello di *Figura 4.7*.

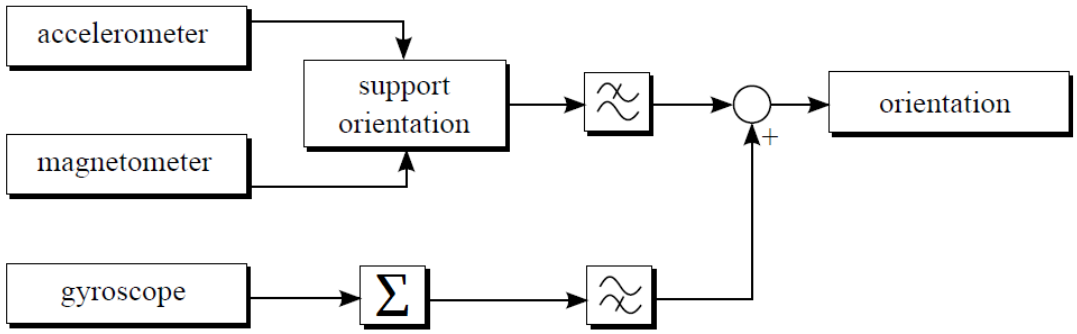


Figura 4.7: Sensor Fusion applicata ad accelerometro, giroscopio e magnetometro; la cooperazione di questi tre sensori riesce a svelare l'orientazione del dispositivo su cui sono installati; figura presa da [22]

Eliminando dunque le alte frequenze di un segnale rumoroso e le basse frequenze di un segnale affetto da drift, è possibile ottenere un segnale ibrido molto più "pulito" ed utile come mostra anche la *Figura 4.8*.

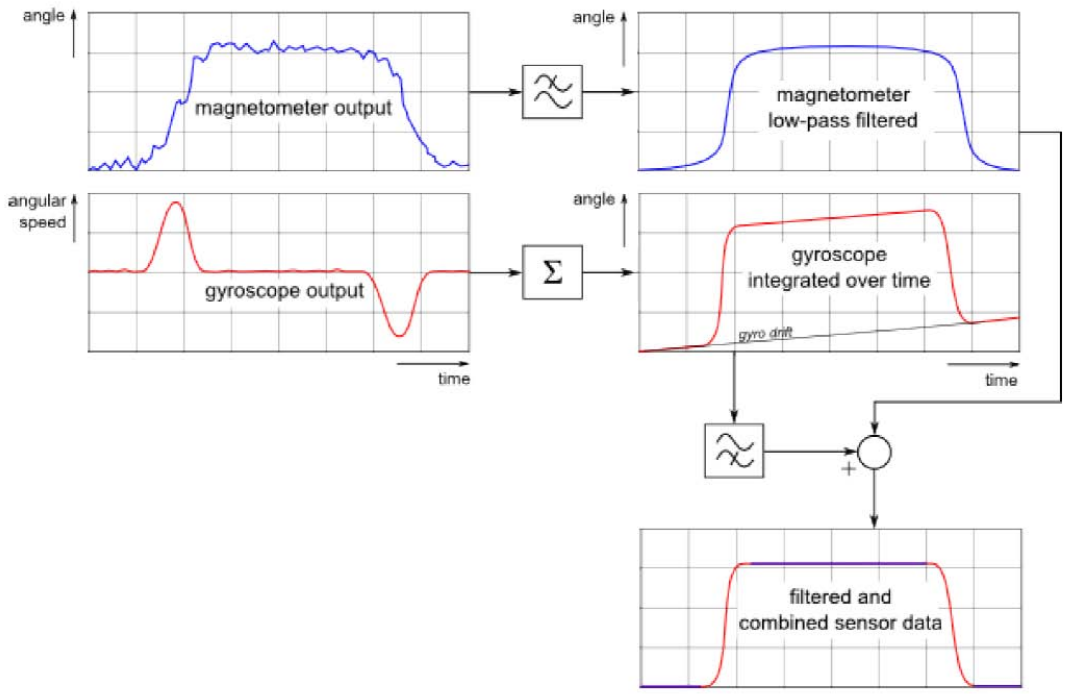


Figura 4.8: L'eliminazione di rumore e drift produce un segnale più affidabile. In questo caso il dispositivo prima è a riposo, poi viene inclinato di 90° rispetto a un certo asse e poi viene riportato a riposo; figura presa da [35]

4.2.2 Il Filtro di Kalman

Il filtro di Kalman è già stato accennato nella sezione 3.3 e viene qui esposto in modo più esaustivo. Si tratta di un efficiente filtro ricorsivo, utile in molti contesti, tra cui anche quello della *sensor fusion*, che valuta lo stato di un sistema dinamico a partire da una serie di misure soggette a rumore. In particolare, se il rumore è di tipo gaussiano e a media nulla, il filtro minimizza l'errore quadratico medio, divenendo così un filtro ottimo. Nel caso in cui non lo sia, il filtro di Kalman rimane comunque il miglior stimatore lineare dati soli la media e la deviazione standard del rumore, anche se stimatori non lineari potrebbero essere più adatti. Scopo di questo strumento è quello di cercare di ottenere lo stato $x \in \mathbb{R}^n$ di un processo a tempo discreto governato dall'equazione differenziale stocastica (4.12) con una misurazione $z \in \mathbb{R}^m$ del tipo di (4.13).

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \tag{4.12}$$

$$z_k = Hx_k + v_k \tag{4.13}$$

Le variabili casuali w_k e v_k rappresentano rispettivamente il rumore del processo e della misura e si suppongono essere vicendevolmente indipendenti, bianche e con distribuzioni $p(w)$ e $p(v)$ normali come nelle equazioni (4.14) e (4.15), dove Q ed R sono le matrici delle covarianze.

$$p(w) \sim N(0, Q) \tag{4.14}$$

$$p(v) \sim N(0, R) \tag{4.15}$$

La matrice A, di dimensione $n \times n$, associa lo stato x_{k-1} a quello successivo x_k, la matrice B associa l'opzionale input di controllo $u \in \mathbb{R}^l$ allo stato x_k e la matrice H associa lo stato x_k alla misurazione z_k. In teoria le matrici A, H, Q ed R possono cambiare ad ogni nuova misura ma quello che molto spesso viene fatto in pratica è considerarle delle costanti.
Sia poi $\hat{x}_k^- \in \mathbb{R}^n$ la stima a priori dello stato al passo k data la conoscenza del processo al passo precedente, e $\hat{x}_k \in \mathbb{R}^n$ la stima a posteriori dello stato sempre al passo k ottenuta grazie alla misurazione z_k. Si possono dunque definire i rispettivi errori di stima come (4.16) e (4.17), ottenendo così le relative covarianze (4.18) e (4.19).

$$e_k^- \equiv x_k - \hat{x}_k^- \tag{4.16}$$

$$e_k \equiv x_k - \hat{x}_k \tag{4.17}$$

$$P_k^- = E[e_k^- e_k^{-T}] \tag{4.18}$$

$$P_k = E[e_k e_k^T] \tag{4.19}$$

Allora l'equazione che calcola la stima a posteriori dello stato $\hat{x}_k$ con una combinazione lineare della stima a priori $\hat{x}_k^-$ e della differenza pesata tra l'effettiva misura z_k e la predizione $H\hat{x}_k^-$ è la (4.20)

$$\hat{x}_k = \hat{x}_k^- + K(z_k - H\hat{x}_k^-) \tag{4.20}$$

dove $(z_k - H\hat{x}_k^-)$ rappresenta la differenza tra la misura effettiva z_k e quella prevista $H\hat{x}_k^-$, mentre la matrice K, di dimensione $n \times m$, minimizza la covarianza dell'errore a posteriori (4.19). Tale minimizzazione può essere fatta inserendo la (4.20) nella (4.17), poi inserendo il risultato ottenuto nella (4.19) ed infine calcolando la derivata rispetto a K, ottenendo i valori di K che la annullano. Il risultato finale di questo procedimento è riportato nell'equazione (4.21).

$$K_k = \frac{P_k^- H^T}{H P_k^- H^T + R} \tag{4.21}$$

Più la covarianza dell'errore di misura R si avvicina a zero e più la misura z_k è considerata più affidabile della misura prevista $H\hat{x}_k^-$ ($\lim_{R_k \to 0} K_k = \frac{1}{H}$); più la covarianza dell'errore stimato a priori P_k^- si avvicina a zero e più la misura prevista $H\hat{x}_k^-$ è considerata più affidabile della misura effettiva z_k ($\lim_{P_k^- \to 0} K_k = 0$).

A questo punto è quindi possibile procedere con la descrizione delle equazioni che effettivamente compongono questo potentissimo strumento.
Il filtro di Kalman basa la stima dello stato usando una forma di controllo a retroazione: dopo che tale stima è stata fatta, utilizza infatti la misura (rumorosa) come indicazione della "bontà" della propria predizione. Le equazioni coinvolte si dividono così in due gruppi: *time update* e *measurement update*. Le prime si occupano di "proiettare in avanti (nel tempo)" lo stato corrente e la covarianza dell'errore della stima, in modo da ottenere la stima a priori del prossimo stato; le seconde si occupano di fornire il feedback, cioè di accorpare la nuova misura e la stima a priori. Così facendo, il risultato finale, cioè la stima a posteriori, terrà conto di entrambe.
Dal momento che le equazioni di *time update* possono essere pensate anche come equazioni che "predicono il futuro" e che quelle di *measurement update* possono essere pensate anche come equazioni che correggono le prime, è sovente in letteratura parlare di *predict phase* e *correct phase*. Andando più nel dettaglio, la *predict phase* è costituita dalle equazioni (4.22), mentre la *measurement phase* è costituita dalle equazioni (4.23).

$$\begin{cases} \hat{x}_k^- = A\hat{x}_{k-1} + Bu_k \\ P_k^- = AP_{k-1}A^T + Q \end{cases} \tag{4.22}$$

$$\begin{cases} K_k = P_k^- H^T (H P_k^- H^T + R)^{-1} \\ \hat{x}_k = \hat{x}_k + K_k(z_k - H\hat{x}_k^-) \\ P_k = (I - K_k H) P_k^- \end{cases} \tag{4.23}$$

Al passo 0, le condizioni iniziali dello stato sono impostate per esempio a 0 e la fase da cui si comincia è quella di predizione, stimando quindi lo stato a priori $\hat{x}_k^-$ e relativa covarianza dell'errore P_k^-. Fatto ciò si passa alla *measurement phase*: la prima cosa da fare è calcolare la matrice K_k, poi si legge la misura z_k in modo da ottenere la stima dello stato a posteriori $\hat{x}_k^-$ e relativa covarianza dell'errore P_k. Al termine di tutto questo si ritorna alla fase di predizione per poi ripetere questo ciclo indefinitamente.

Un esempio della potenza di questo strumento è riportata in *Figura 4.9* ed ulteriori approfondimenti si possono trovare in [24].

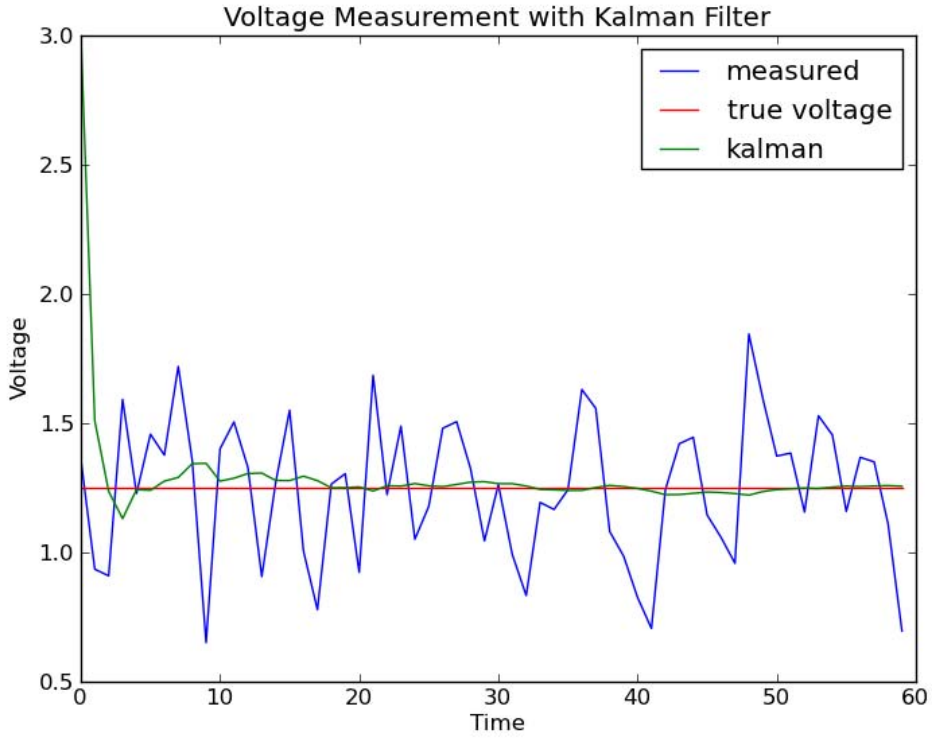


Figura 4.9: Comportamento del filtro di Kalman con una lettura rumorosa del voltaggio di uno strumento; notare l'inizio molto distorto per via delle condizioni iniziali lontane dalla realtà; immagine presa da [36]

Capitolo 5

Il procedimento svolto

Per ottenere un'applicazione che simulasse il flauto di Pan ricorrendo alla *sensor fusion*, abbiamo pensato di utilizzare la fotocamera per i movimenti piccoli e l'accelerometro per quelli grandi. Tenendo infatti uno dei lati in prossimità della bocca durante l'esecuzione (proprio come un vero flauto di Pan), la fotocamera è in grado di catturare di frame in frame le forme del vestito dell'utente, rendendo così possibile la stima del moto del dispositivo. Naturalmente questo discorso vale nel caso in cui il movimento non sia troppo ampio perché altrimenti, tra l'inevitabile sfocatura delle immagini ed il frame corrente troppo poco simile al precedente, le feature tracciate verrebbero perse.
Fortunatamente questa non è una limitazione che affligge l'accelerometro che, al contrario, potrebbe avere l'output troppo influenzato dal rumore per piccoli movimenti. Abbiamo quindi concentrato i nostri sforzi prima sulla stima del moto da fotocamera e successivamente da accelerometro, giroscopio e magnetometro. Di seguito vengono dunque riportati in ordine cronologico tutti gli esperimenti svolti con relativi risultati.
Ogni dato sperimentale descritto in questo lavoro è stato ricavato con uno smartphone LG G2 (Sistema Operativo: Android 4.4.2 KitKat; Videocamera posteriore: 13 MP OIS, risoluzione massima 4160 x 3120 pixel, autofocus (9 punti), 60 fps video 1080p; Videocamera anteriore: 8.1 MP; CPU: Snapdragon 800 MSM8974AA da 2.26 GHz; Memoria: 1.81GB RAM LP-DDR3 800mhz, interna 32 GB flash).

5.1 Fotocamera: FAST-Tracker

5.1.1 Introduzione

Per costruire una funzione che prenda in input una sequenza di immagini e dia in output il vettore del relativo movimento globale, abbiamo utilizzato *OpenCV*,

una libreria *open source* di *computer vision* sviluppata in *C++* e disponibile recentemente anche in *Java* per Android. Abbiamo così scelto di implementare un algoritmo basato sul metodo Lucas-Kanade (vedi capitolo 4.1.4) ricorrendo alla funzione `calcOpticalFlowPyrLK()`, della classe `Video`, che prende in input due immagini (frame consecutivi di una sequenza video), che noi abbiamo ridimensionato con il metodo `resize` della classe `Imgproc`, ed un insieme di punti per cui deve essere trovato il flusso. Molto spesso, tale insieme viene calcolato con la funzione `goodFeaturesToTrack()` (vedi paragrafo 4.1.2) ma è parso subito evidente che quella strada non avrebbe portato da nessuna parte. É emerso infatti che con tale funzione si riescono ad elaborare circa $2 \sim 3$ frame al secondo, con oscillazioni in base alla scena ripresa, ma ciò che a noi serve è almeno un tasso di 5 frame al secondo o, più auspicabilmente, anche 10.
Abbiamo quindi optato per il metodo `FAST`, della classe `FeatureDetector`, la quale riduce i tempi di calcolo dell'algoritmo finale di circa tre volte.

5.1.2 L'algoritmo implementato - versione 1

A questo punto non manca nessun elemento per la comprensione dell'algoritmo che abbiamo testato e quindi di seguito sono riportati tutti i passi di cui è composto:

1. Prendi l'immagine rilevata dalla fotocamera e convertila in una scala di grigi così da poter poi essere usata dalla funzione che calcola il flusso ottico
2. Porta la grandezza dell'immagine a 1/16 di quella originale per un'elaborazione più veloce
3. Calcola i punti d'interesse tramite la funzione `FAST`
4. Passa ogni punto così trovato alla funzione `cornerSubPix` della classe `Imgproc` in modo da avere le loro posizioni meglio definite, cioè non rappresentate da semplici numeri interi, usando una finestra di ricerca di 19×19 pixel (il parametro usato deve quindi essere `Size(9,9)`)
5. Se la corrente immagine processata è la prima di tutta la sequenza allora ripeti dall'inizio
6. Calcola il flusso ottico tra le ultime due immagini processate con la funzione `calcOpticalFlowPyrLK`, impostando la grandezza della finestra di ogni livello della piramide a 17×17 pixel ed il livello massimo della piramide a 4
7. Se il numero n dei vettori così trovati è maggiore di 10 allora ordinali in base alla loro componente orizzontale e scarta i primi e gli ultimi $\lfloor n/2.5 \rfloor$, altrimenti torna al primo passo

8. Definisci il movimento globale tra il frame corrente e quello precedente come la media aritmetica tra tutte le componenti orizzontali dei vettori rimasti (lo spostamento verticale non è d'interesse per l'app)

Tutti i parametri impiegati sono stati ricavati euristicamente, dopo numerosi e numerosi tentativi. I più importanti sono sicuramente quelli dei punti 2 e 6, ossia quelli che determinano il ridimensionamento dell'immagine, la grandezza della finestra di ricerca ed il livello massimo della piramide. Questi infatti sono quelli che maggiormente incidono sul frame rate e sulla qualità della stima: valori troppo elevati comportano un frame rate ridotto ma una migliore precisione e, viceversa, valori troppo ridotti comportano un frame rate più accettabile a discapito però dell'affidabilità della stima. Da notare, tra l'altro, che i parametri del punto 6 hanno valori molto simili a quelli di solito usati in letteratura. Abbiamo quindi cercato un connubio accettabile tra le opposte esigenze giungendo alle conclusioni appena descritte. I restanti due parametri invece, ossia la grandezza della finestra di `cornerSubPix` e la soglia con cui eliminare i vettori ordinati, sono risultati non influenzare significativamente la velocità di elaborazione; essi tuttavia contribuiscono ad aumentare la qualità dell'output. In particolar modo si noti che scartare i primi e gli ultimi $\lfloor n/2.5 \rfloor$ vettori ordinati significa non considerare quell'80% dei vettori che più si discosta dal valor medio ottenuto.

5.1.3 Risultati sperimentali

Per mettere alla prova l'algoritmo, abbiamo usato una griglia come quella di *Figura 5.1* perché ricca di punti ad alto contrasto. L'abbiamo scelta di grandezza $240mm \times 105mm$ in modo che potesse essere contenuta in un foglio A4, e lo smartphone è stato posto ad una distanza di 69cm dalla figura, come mostra la *Figura 5.2*. Ai tempi di questo test infatti non era ancora chiaro se la fotocamera avrebbe puntato in direzione del petto dell'"esecutore" o in quella opposta, per cui abbiamo optato per una distanza fotocamera - oggetto ripreso che fosse intermedia. Data comunque l'estrema semplicità della figura, la distanza tra lo smartphone e la griglia non ricopre un ruolo fondamentale per l'algoritmo.
I risultati ottenuti ripetendo cinque volte quattro traslazioni pure di diversa lunghezza sono quelli di *Tabella 5.1*, dove la prima colonna indica lo spostamento effettuato, la seconda indica la media delle quattro letture ottenute e la terza indica di quanto si sia discostata la peggiore lettura dal valore reale.

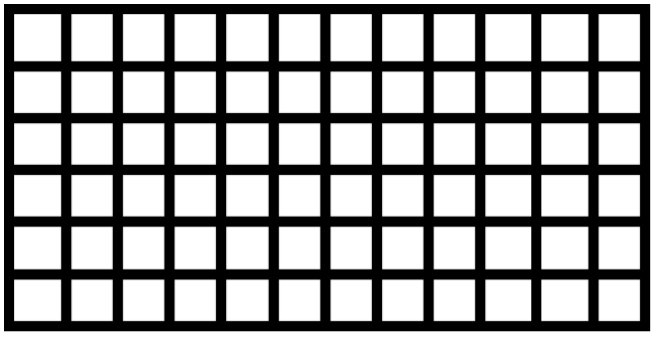


Figura 5.1: Griglia utilizzata nel primo test

Figura 5.2: Set allestito per condurre il primo test

spostam. reale (cm)	media spostam. stimato (cm)	discostam. max rilevato (cm)
1	1.02	0.07
2	2.06	0.17
5	4.97	0.61
10	10.00	0.46

Tabella 5.1: Risultati ottenuti con la prima versione del FAST-tracker

I risultati sembrano molto positivi ma un problema che abbiamo riscontrato è il fatto che l'output è effetto da drift: con smartphone fermo infatti la somma di tutte le letture, che serve a stimare di quanto il dispositivo si sia spostato dall'inizio del processo, non converge a zero ma, al contrario, tende a divergere. Altra cosa importante poi è anche che un errore di 0.61cm non può essere accettato se lo spazio utile, cioè la lunghezza dello schermo, è di 11cm e le canne dello strumento sono quattordici. Abbiamo quindi dovuto apportare qualche modifica all'algoritmo come riportato di seguito.

5.1.4 L'algoritmo implementato - versione 2

Questa seconda versione si differenzia dalla prima per l'aggiunta di un metodo per non avere troppe feature da gestire e per la diversa stima del movimento globale a partire dai vettori di movimento locale. Di seguito sono dunque riportati tutti i punti che la compongono.

1. Prendi l'immagine rilevata dalla fotocamera e convertila in una scala di grigi così da poter poi essere usata dalla funzione che calcola il flusso ottico

2. Ridimensiona l'immagine in modo che ci siano 888 pixel su ogni riga e 540 su ogni colonna

3. Calcola i punti d'interesse tramite la funzione `FAST`

4. Se il loro numero `numFeatures` è maggiore di 139, allora prendine uno ogni $\lfloor$`numFeatures`$/70\rfloor$

5. Passa ogni punto così trovato alla funzione `cornerSubPix` della classe `Imgproc` in modo da avere le loro posizioni meglio definite, cioè non rappresentate da semplici numeri interi, usando una finestra di ricerca di 19×19 pixel (il parametro usato deve quindi essere `Size(9,9)`)

6. Se la corrente immagine processata è la prima di tutta la sequenza allora ripeti dall'inizio

7. Calcola il flusso ottico tra le ultime due immagini processate con la funzione `calcOpticalFlowPyrLK`, impostando la grandezza della finestra di ogni livello della piramide a 17×17 pixel ed il livello massimo della piramide a 8

8. Crea un istogramma quadrato come quello di [11] e tieni traccia dei cinque vettori più frequenti

9. Se il numero di occorrenze del vettore più frequente è almeno 31, definisci la componente orizzontale del vettore di movimento globale come la media pesata delle componenti orizzontali dei primi cinque vettori più frequenti; idem per la componente verticale

10. Se la componente orizzontale o verticale del vettore di movimento globale ha modulo unitario, allora impostala a zero se il numero di occorrenze del vettore più frequente non è maggiore della metà della lunghezza del numero di vettori locali

A dire il vero, non è necessario che l'istogramma del punto 8 sia quadrato, infatti anche uno lineare andrebbe bene. Il motivo di questa scelta è il semplice fatto che, al tempo della stesura del codice, non era chiaro se lo spostamento verticale avrebbe ricoperto un qualche ruolo tra le funzionalità dell'app, per cui abbiamo affrontato il caso più generale. Il punto 10 invece potrebbe sembrare strano ma in realtà serve a dire che si accettano spostamenti di un singolo pixel solo nel caso in cui almeno la metà di tutti i vettori presenti segnalino tale spostamento; in questo modo viene attenuata l'influenza del rumore.
Per quanto riguarda i valori dei parametri, valgono tutte le considerazioni fatte per la versione 1. In particolare, quelli del punto 2 producono lo stesso ridimensionamento fatto nella prima versione, soltanto che in questo caso l'immagine avrà sempre la stessa grandezza finale indipendentemente dal dispositivo usato. Al numero massimo di feature invece è stato dato un tetto massimo perché abbiamo constatato che, oltre una certa soglia, i costi computazionali cominciavano ad essere più importanti dei benefici sul *tracking*. Da notare inoltre che in questa versione, poiché non implicava una significativa diminuzione del *frame rate*, abbiamo innalzato il livello massimo della piramide a 8 per cercare di rendere più accurata la stima.

5.1.5 Risultati sperimentali

Questa volta, invece che la griglia, abbiamo usato un'immagine grande $286mm \times 201mm$, così da poter essere contenuta in un foglio A4, con la scena di *Figura 5.3*. Tale scelta è giustificata dal fatto che in questo modo viene passato all'algoritmo un' immagine compatibile con quelle che saranno disponibili durante l'uso reale. L'abbondanza di feature tutte differenti presenti in *Figura 5.3* è infatti ben rappresentativa della situazione che si potrebbe ottenere inquadrando un abito avente una trama non eccessivamente anonima; l'unica differenza rispetto al caso reale consiste nel fatto che qui l'immagine è perfettamente piatta mentre inquadrando in direzione del petto di una persona ci saranno inevitabilmente alcune zone più sporgenti di altre.
Durante il test la distanza tra l'immagine e lo smartphone è stata di 9cm, come mostrato in *Figura 5.4*, essendo questo il discostamento da noi stimato dello smartphone dal petto durante l'"esecuzione", e gli spostamenti effettuati sono stati anche in questo caso pure traslazioni. In particolare abbiamo eseguito spostamenti di 1, 2 e 5 centimetri ed ognuno di questi è stato eseguito cinque volte in un verso e cinque volte in quello opposto. I risultati ottenuti sono riportati nelle tabelle 5.2, 5.3 e 5.4, dove ogni valore rappresenta lo spostamento registrato in centimetri.

Figura 5.3: Immagine utilizzata nel secondo test; figura presa da [40]

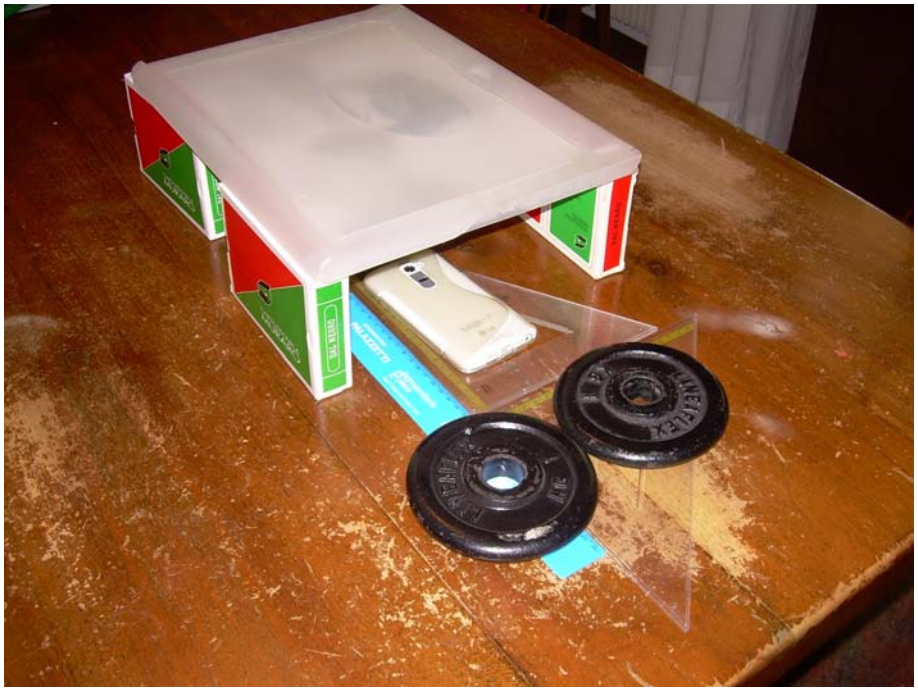


Figura 5.4: Set allestito per condurre i test successivi al primo; l'immagine di *Figura 5.3* è incollata al supporto superiore

1 cm	#1	#2	#3	#4	#5	media
avanti	1.02	1.01	0.98	1.01	0.99	1.00
indietro	0.91	0.90	0.91	0.91	0.87	0.90

Tabella 5.2: Spostamenti stimati in centimetri con la seconda versione del FAST-tracker per movimenti di 1cm

2 cm	#1	#2	#3	#4	#5	media
avanti	1.97	1.98	1.96	1.99	2.03	1.99
indietro	1.84	1.83	1.79	1.87	1.78	1.82

Tabella 5.3: Spostamenti stimati in centimetri con la seconda versione del FAST-tracker per movimenti di 2cm

5 cm	#1	#2	#3	#4	#5	media
avanti	4.92	4.92	4.88	4.97	4.93	4.92
indietro	4.74	4.57	4.68	4.74	4.72	4.72

Tabella 5.4: Spostamenti stimati in centimetri con la seconda versione del FAST-tracker per movimenti di 5cm

Sebbene il problema del drift sia stato risolto e il tasso di elaborazione sia accettabile (circa 8 frame al secondo), le stime del movimento differiscono ancora troppo le une dalle altre a parità di spostamento realmente effettuato. A questo si aggiunge il fatto che ci si aspetterebbe che la media dei valori ottenuti spostando il dispositivo di un centimetro sia cinque volte inferiore di quella ottenuta spostandolo di cinque centimetri, ma risulta evidente che non è esattamente così. Un'altra osservazione degna di nota è la sistematica differenza del valore ottenuto tra spostamenti in avanti e indietro. Nonostante abbiamo provato a ripetere i test più volte facendo in modo che il movimento sia il più fluido e uniforme possibile, curandoci di effettuare sempre la stessa quantità di traslazione mediante delle guide, non siamo riusciti a giustificare questo strano comportamento.
Abbiamo allora provato a fare un'altra revisione dell'algoritmo, contemplando questa volta dei punti di riferimento fissi come spiegato nella prossima sezione.

5.1.6 L'algoritmo implementato - versione 3

Questa nuova versione è molto simile alla precedente, solo che all'inizio prevede un'onerosa fase di calibrazione: all'utente infatti è chiesto di posizionarsi sulle

canne 1, 3, 5, 7, 9, 11 e 13 in modo da memorizzare le immagini e le feature corrispondenti. A questo punto l'algoritmo confronta l'immagine corrente con una di quelle in memoria per trovare qualche corrispondenza: se il match riesce, assumendo che una canna sia lunga 30 pixel, è possibile così stabilire quale sia quella posizionata sotto la bocca dell'"esecutore".
La ricerca del flusso viene fatta ogni volta "ancorandosi" ad una delle sette immagini di calibrazione, nel senso che il confronto è sempre tra l'immagine corrente ed una di quelle sette. Nel caso di fallimento, l'algoritmo proverà a rieffettuarlo cambiando "ancoraggio", muovendosi cioè sempre più lontano dal punto iniziale. Ad esempio, se l'immagine di riferimento è quella corrispondente alla canna 7 e non viene trovata nessuna corrispondenza col frame corrente, si proverà prima con l'immagine di riferimento della canna 5, poi con quella della canna 9, poi con quella della canna 3 e così via, fino ad esaurire tutte le possibilità e riprovare eventualmente con un un nuovo frame corrente.
Come si evince dal test riassunto nella tabella 5.5, svolto esattamente come il precedente, cioè tenendo l'immagine di figura 5.3 a 9cm di distanza dalla fotocamera, neanche in questo caso la situazione è migliorata particolarmente perché le *FAST features* sono risultate essere poco adatte per trovare il flusso su immagini che hanno un certo grado di diversità tra loro. Per ottenere un match valido infatti, l'immagine di input dell'algoritmo deve essere molto simile a una di quelle ottenute durante la calibrazione, cosa tra l'altro irrealistica nel caso reale. Comunque già il fatto che la parte di calibrazione fosse così inevitabilmente "invasiva" non lasciava particolari aspettative.
Non essendo dunque riusciti a migliorare sensibilmente la situazione, abbiamo momentaneamente accantonato la parte relativa al flusso ottico per concentrarci sugli altri sensori, nella speranza di riuscire ad integrare i vari dati ed ottenere una risposta più conforme alle nostre necessità.

canna corretta	canna rilevata
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
13	13
12	12
11	11
10	10
9	9
8	8
7	7
6	6
5	5
4	4
3	3
2	2
1	1
2	2
5	nessun match
8	8
11	11
14	14
11	nessun match
8	nessun match
5	nessun match
2	2
4	nessun match
9	nessun match
14	14
9	9
4	nessun match

Tabella 5.5: Stima della canna corretta con la terza versione del FAST-tracker

5.2 Sensori di movimento

5.2.1 Introduzione

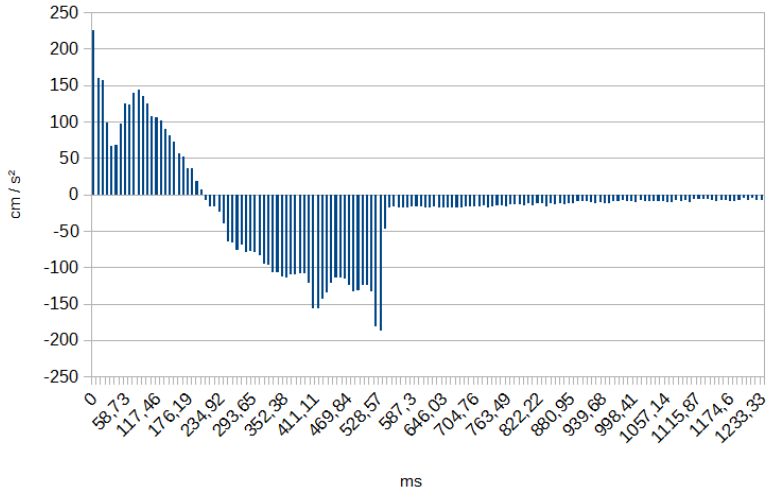
L'intento ora è quello di risolvere il problema della stima del moto grazie principalmente all'accelerometro. Conoscendo infatti i valori dell'accelerazione nel tempo e la distanza temporale tra un valore ed il successivo, è possibile, almeno teoricamente, fare un'integrazione nel tempo per ottenere la velocità e poi ancora un'altra integrazione nel tempo per ottenere lo spostamento.
Prima di cominciare il lavoro *ex novo*, abbiamo condotto qualche ricerca per vedere se in letteratura era già disponibile qualche studio utile al nostro caso ma, purtroppo, sembra che l'uso dell'accelerometro per piccoli movimenti (centimetri) negli smartphone sia un campo inesplorato. L'unica indicazione a noi nota era il fatto che una doppia integrazione delle letture dell'accelerometro sarebbe stata eccessivamente compromessa dal rumore, come spiegato in [27].
Ciò che abbiamo fatto è stato dunque mettere a confronto quattro diverse strategie per capire quale fosse quella più promettente a cui dedicare uno studio più approfondito. I test hanno riguardato l'utilizzo della funzione `TYPE_LINEAR_ACCELERATION` disponibile nelle librerie Android, una doppia integrazione dei trapezi, la strategia di *NoShake* (vedi [12] e sezione 3.2) ed il filtro di Kalman.
Tutti i test sono stati eseguiti allo stesso modo, cioè poggiando lo smartphone sopra un tavolo come mostrato in *Figura 5.4* e facendogli percorrere più volte una distanza 9cm, sia in avanti che indietro, il tutto per cinque volte, col supporto di una guida per assicurarsi che il moto sia di pura traslazione. Maggiori dettagli sui metodi con annessi risultati sono di seguito elencati.

5.2.2 L'accelerazione lineare

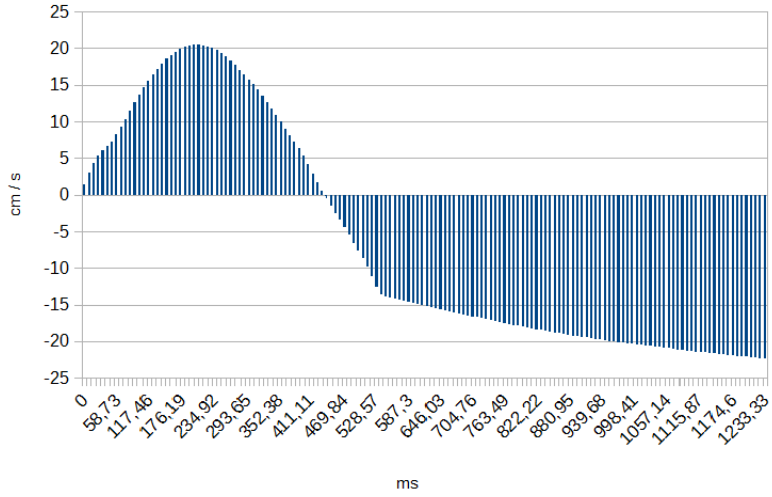
Questa strategia consiste nel separare le due componenti che costituiscono l'output dell'accelerometro, ossia quella gravitazionale e quella dovuta al moto del dispositivo. Per fare ciò, viene impiegato un filtro passa-basso per isolare la gravità a partire dai dati ricevuti come illustrato nell'algoritmo seguente; la variabile $total\_acceleration$ rappresenta l'accelerazione ottenuta dall'accelerometro e la variabile α è soltanto una costante posta a 0.8.

$$gravity \leftarrow gravity \cdot \alpha + (1 - \alpha) \cdot total\_acceleration$$
$$linear\_acceleration \leftarrow total\_acceleration - gravity$$

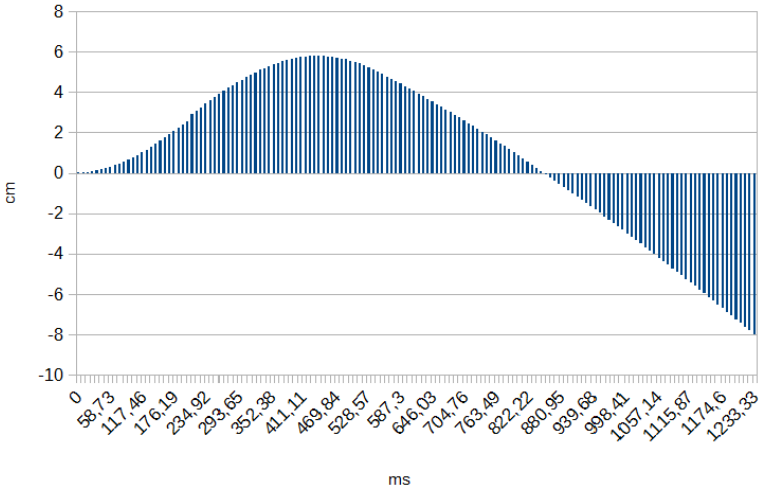
In questo modo il valore contenuto in $total\_acceleration$ dovrebbe restituire i valori dell'accelerometro liberi dall'influenza gravitazionale ma, purtroppo, tutti i tentativi fatti sono risultati essere simili a quelli di *Figura 5.5*, ossia inutilizzabili per via di uno "stazionamento" finale dell'accelerazione rilevata.



(a) Accelerazione (output dell'accelerometro lineare)



(b) Velocità (integrazione dell'accelerazione)



(c) Spostamento (integrazione della velocità)

Figura 5.5: Esempio delle stime ottenute con l'accelerazione lineare traslando lo smartphone di 9cm; ogni campione è stato prodotto ad una distanza di 8.3ms dal precedente

5.2.3 Metodo dei trapezi

Si tratta di implementare la semplicissima idea proposta in [28], in cui, per semplicità di codice, i dati in input, ossia i valori dell'accelerometro, sono resi tutti positivi grazie ad un offset. In tal modo è possibile calcolare l'area tra due successive letture ($Sample_n$ e $Sample_{n-1}$) tramite la (5.1) e per ottenere valori non distorti basta infine sottrarre il medesimo offset come mostra la *Figura 5.6*.

$$Area_n = Sample_n + \frac{|Sample_n - Sample_{n-1}|}{2} \times \Delta t \tag{5.1}$$

Così facendo, come mostra la *Figura 5.7*, le varie aree calcolate comprendono molto meno errore rispetto a quelle che si otterrebbero facendo una semplice somma.
Il riassunto dei dati raccolti con questa strategia è riportato in *Figura 5.8*. Appare subito evidente che sia il picco massimo che la posizione finale di ogni curva è soggetta ad un errore di diversi centimetri, anche più di tre, il che rende questa strategia inadeguata ai nostri scopi.

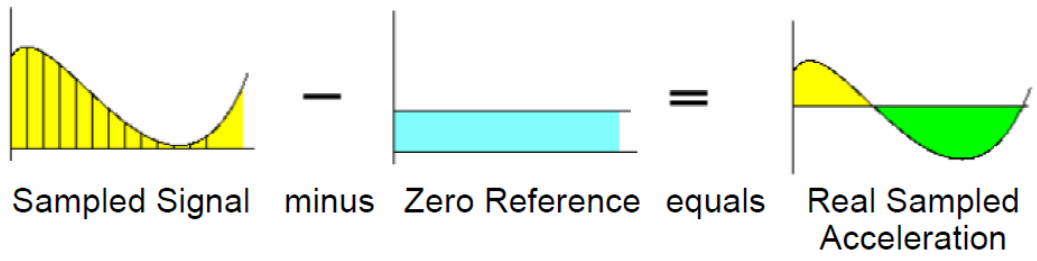


Figura 5.6: Metodo dei trapezi, alla fine i risultati vengono riportati ad un'altezza più consona; figura presa da [28]

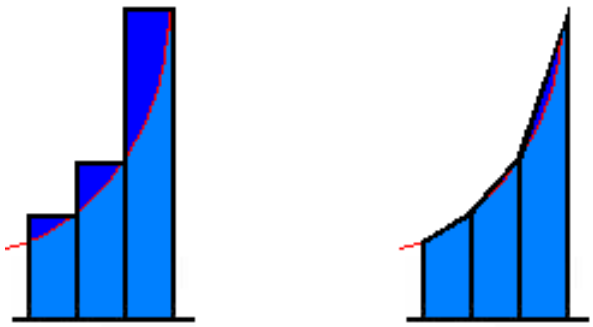


Figura 5.7: Metodo dei trapezi: a sinistra il prima e a destra il dopo elaborazione; figura presa da [28]

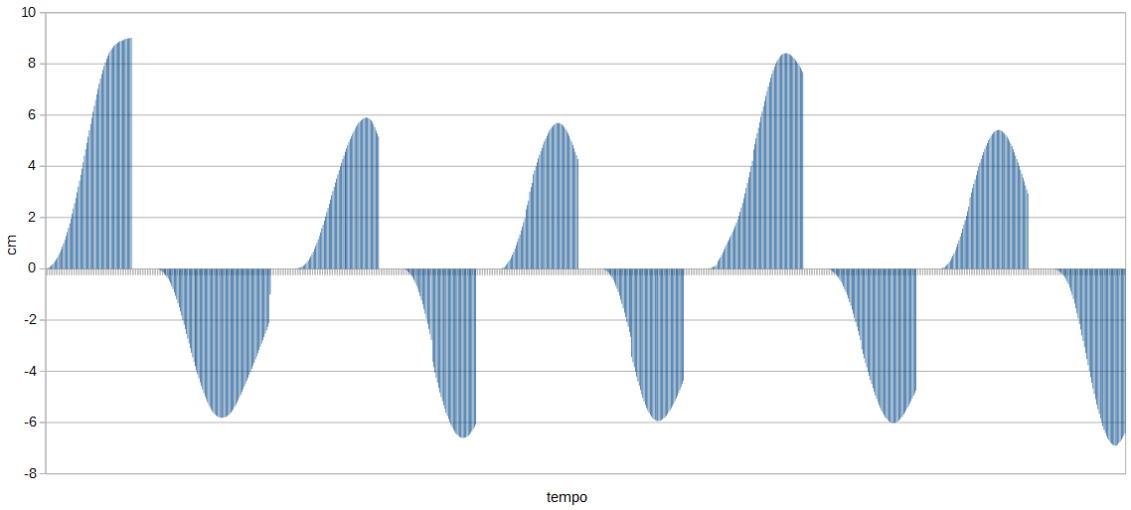


Figura 5.8: Risultati ottenuti col metodo dei trapezi: l'asse X rappresenta il tempo e l'asse Y lo spostamento stimato. I vari dati sono stati raccolti con passo di campionamento di 8.39ms ma, al contrario di quanto si possa pensare guardando il grafico, tra uno spostamento e quello successivo è stato fatto trascorrere un tempo di diversi secondi in modo da cominciare ogni stima con accelerometro a riposo

5.2.4 Metodo *NoShake*

Tale strategia è la stessa proposta in [12] e già descritta nella sezione 3.2. Il parametro k, dopo un po' di prove, è stato impostato a 3 e i risultati ottenuti sono quelli di *Figura 5.9*, dove ogni ordinata è "pura", cioè non moltiplicata per il fattore di scala a. É il caso più difficile da confrontare perché il suo output non rappresenta lo spostamento stimato ma il comportamento che dovrebbe avere la massa m in risposta allo spostamento subito. Ad ogni modo, in questa fase ci interessa soltanto l'andamento delle curve dei grafici per avere delle prime indicazioni, e anche questo caso può fornire informazioni utili. Si può osservare infatti che i picchi delle varie curve sono ancora una volta molto diversi fra loro e quindi nemmeno la convoluzione del segnale può essere una risposta valida al nostro problema.

5.2.5 Metodo del filtro di Kalman

Un altro tentativo è stato effettuato con il filtro di Kalman, descritto nella sezione 4.2.2. Il sistema che abbiamo impostato è quello di (5.2), dove la matrice quadrata della prima equazione rappresenta il modello utilizzato: tre semplici equazioni prese dalla fisica elementare legano vicendevolmente posizione, velocità ed accelerazione. Per quanto riguarda le due tipologie di rumore invece, dopo vari tentativi siamo giunti alla conclusione che una scelta ragionevole di parametri vuo-

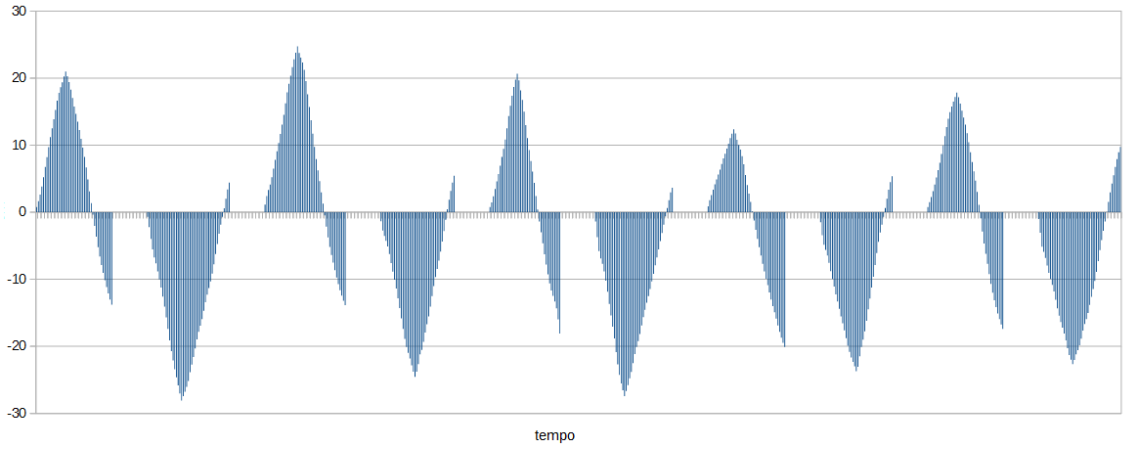


Figura 5.9: Risultati ottenuti col metodo di *NoShake*: l'asse X rappresenta il tempo e l'asse Y il risultato della convoluzione. I vari dati sono stati raccolti con passo di campionamento di 8.39ms ma, al contrario di quanto si possa pensare guardando il grafico, tra uno spostamento e quello successivo è stato fatto trascorrere un tempo di diversi secondi in modo da essere sicuri di cominciare ogni stima con accelerometro a riposo

le $Q = 0.001$ (affidabilità del modello) e $R = 0.007$ (affidabilità della misura); da notare che in entrambi i casi il valore 0 avrebbe significato "affidabilità assoluta".

$$\begin{cases} \begin{bmatrix} x_k \\ \dot{x}_k \\ \ddot{x}_k \end{bmatrix} = \begin{bmatrix} 1 & \Delta t & \frac{1}{2}\Delta t^2 \\ 0 & 1 & \Delta t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k-1} \\ \dot{x}_{k-1} \\ \ddot{x}_{k-1} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + w_{k-1} \\ z_k = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ \dot{x}_k \\ \ddot{x}_k \end{bmatrix} + v_k \end{cases} \tag{5.2}$$

I risultati ottenuti con questa strategia sono quelli di *Figura 5.10* e, come si può subito notare, sia l'accuratezza che la precisione sembrano lasciare spazio ad un proficuo sviluppo dell'algoritmo. Abbiamo perciò continuato i nostri studi in questa direzione, mettendo alla prova l'algoritmo su più distanze.
Il procedimento seguito è lo stesso dei casi precedenti (cinque traslazioni pure in avanti e indietro) ma questa volta il test ha avuto percorsi di 4, 9, 10 e 15 centimetri. Le distanze finali ottenute con relative tempistiche sono quelle riportate nelle tabelle che seguono.

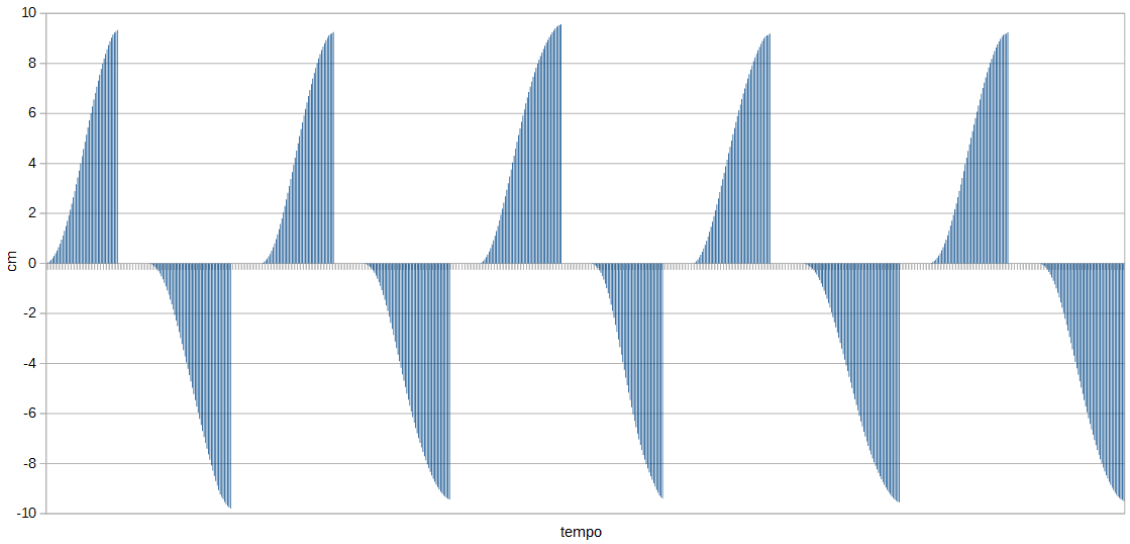


Figura 5.10: Risultati ottenuti col filtro di Kalman: l'asse X rappresenta il tempo e l'asse Y lo spostamento stimato. I vari dati sono stati raccolti con passo di campionamento di 8.39ms ma, al contrario di quanto si possa pensare guardando il grafico, tra uno spostamento e quello successivo è stato fatto trascorrere un tempo di diversi secondi in modo da cominciare ogni stima con accelerometro a riposo

Tempo trascorso (sec)	Spostamento stimato (cm)
0.318908691	4.3633461451
0.276947022	-4.1868395271
0.327301025	4.330184741
0.386047363	-3.9009596118
0.318908691	4.2608188268
0.394439697	-4.3658772311
0.327301025	4.1228409365
0.37765503	-4.0267686676
0.335693359	4.4073788731
0.352478027	-4.370373571

Tabella 5.6: Spostamenti stimati dal filtro di Kalman col set di *Figura 5.4* per traslazioni di 4cm in avanti e indietro

Tempo trascorso (sec)	Spostamento stimato (cm)
0.553894043	9.112644278
0.570678711	-9.2965138572
0.570678711	9.0388474765
0.528717041	-9.0761002211
0.486755371	9.7094124375
0.495147705	-9.2324617715
0.629425048	9.2985297975
0.528717041	-8.7408134585
0.537109375	9.1993599949
0.503540039	-9.5319624012

Tabella 5.7: Spostamenti stimati dal filtro di Kalman col set di *Figura 5.4* per traslazioni di 9cm in avanti e indietro

Tempo trascorso (sec)	Spostamento stimato (cm)
0.579071045	10.9229264888
0.570678711	-10.3260808026
0.545501709	9.9413715955
0.671386718	-10.4831180042
0.612640381	10.5633666319
0.579071044	-10.195717072
0.553894043	10.2318636151
0.654602051	-10.5023390658
0.604248047	10.4391779786
0.570646156	-10.3232469511

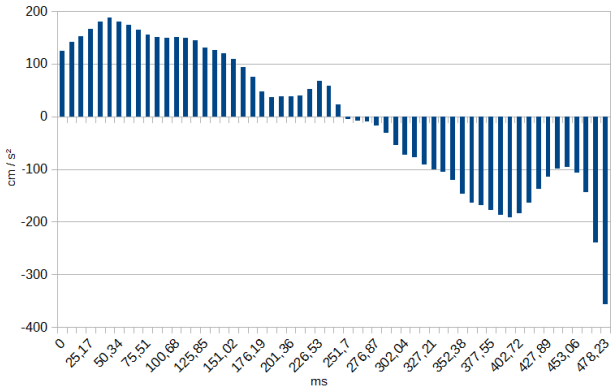
Tabella 5.8: Spostamenti stimati dal filtro di Kalman col set di *Figura 5.4* per traslazioni di 10cm in avanti e indietro

Tempo trascorso (sec)	Spostamento stimato (cm)
0.621032715	15.7139965727
0.730133057	-15.2645099255
0.579074078	15.3842827063
0.746917725	-15.4539908378
0.69656372	14.7615057441
0.679779052	-14.0623119324
0.746917724	15.1464848693
0.872802735	-15.3333426462
0.78048706	15.2087506862
0.797271729	-14.7810147801

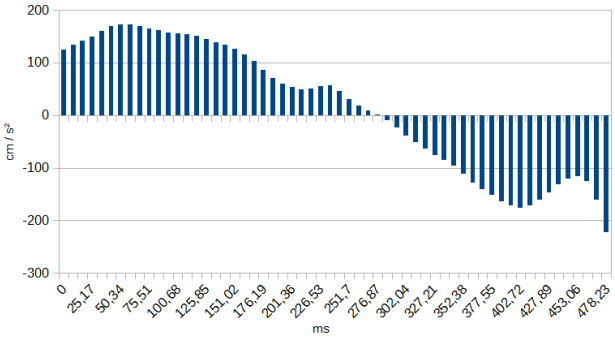
Tabella 5.9: Spostamenti stimati dal filtro di Kalman col set di *Figura 5.4* per traslazioni di 15cm in avanti e indietro

Si evince che gli errori massimi riscontrati durante queste prove sono di 0.37cm per spostamenti di 4cm, 0.71cm per spostamenti di 9cm, 0.92cm per spostamenti di 10cm e 0.94cm per spostamenti di 15cm: si tratta di errori del 6∼8%.
Per avere un'idea più completa, in *Figura 5.11* è riportato un esempio della stima della posizione a partire dalla stima della velocità, che a sua volta è ricavata dall'accelerazione.

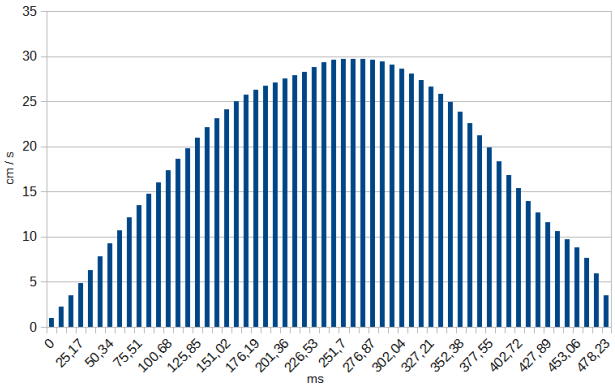
Le ultime verifiche svolte riguardano quelle con la gravità rimossa. Fin qui infatti l'influenza gravitazionale è sempre stata presente ma, poiché durante i test lo smartphone è sempre stato parallelo al tavolo, questa si è riversata tutta sull'ininfluente asse Z dell'accelerometro. Nel caso reale tuttavia non è pensabile che l'"esecutore" abbia una mano così ferma da far ricadere il vettore della gravità tutto su un asse per cui è opportuno escogitare un sistema per la sua rimozione.
Assumendo dunque la gravità costante in modulo, tale problema può essere risolto ricavando l'orientazione del dispositivo. Abbiamo così individuato due approcci che vengono incontro alle nostre esigenze: l'utilizzo dell'algoritmo sviluppato da Paul Lawitzki (descritto nella sezione 4.2.1, vedi anche [22]) e l'utilizzo della funzione `Sensor.TYPE_ROTATION_VECTOR` disponibile nelle API Android. Per semplicità di codice abbiamo scelto la seconda opzione.
Allora, conoscendo in ogni momento l'angolo θ che l'asse Y del dispositivo forma col terreno (da notare che deve essere tenuto in orizzontale), è possibile rimuovere l'influenza gravitazionale dalle letture dell'accelerometro sottraendo il seno di θ moltiplicato per g, dove la costante g può essere impostata manualmente a 9.8 o ricavata da una calibrazione. Per una maggiore accuratezza, tutte le nostre prove sono state svolte deducendo g da una serie di 1000 letture dell'accelerometro con



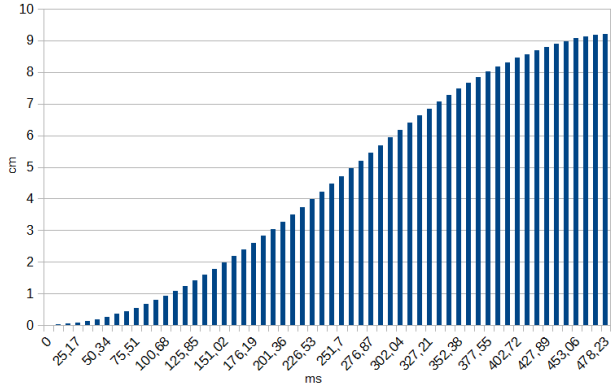
(a) Accelerazione pura (output dell'accelerometro)



(b) Accelerazione in dedotta col filtro di Kalman



(c) Velocità dedotta col filtro di Kalman



(d) Spostamento dedotto col filtro di Kalman

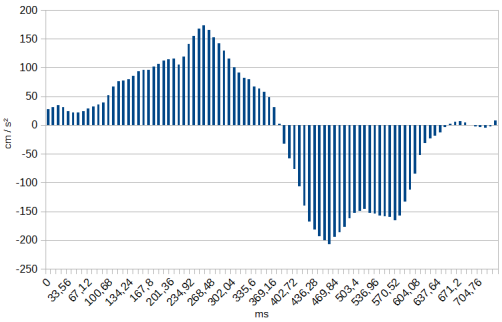
Figura 5.11: Esempio delle curve ottenute tramite il Filtro di Kalman su una traslazione di 9cm; ogni campione dista 8.3ms dal precedente

dispositivo fermo e appoggiato sul tavolo, concludendo che $g = 9.829892 m/s^2$.
Abbiamo quindi provveduto ad effettuare dei test su di un piano inclinato di 18° traslando lo smartphone di 9cm in avanti e indietro ma ci siamo subito imbattuti in una limitazione della stima dell'orientazione: sebbene l'inclinazione venisse correttamente stimata entro un errore di circa ±1°, questa (cioè la stima) purtroppo era soggetta a variazioni non trascurabili durante il movimento. Per meglio isolare questo comportamento abbiamo condotto alcune prove su di un piano orizzontale, cioè inclinato di 0°, ed i risultati ottenuti sono tutti riconducibili a quelli di *Figura 5.12*. É chiaro quindi che la stima finale viene gravemente compromessa da tale instabilità ma, fortunatamente, a riposo o durante micromovimenti, la stima dell'inclinazione è sufficientemente attendibile per cui abbiamo deciso di disabilitare l'aggiornamento dell'orientamento durante ogni "grande" movimento, assumendo quindi che non ci siano rotazioni durante gli spostamenti.
I risultati che sono emersi sono riportati nella *Tabella 5.10* e, a scopo dimostrativo, maggiori dettagli per quanto riguarda il primo caso sono esposti in *Figura 5.13*.

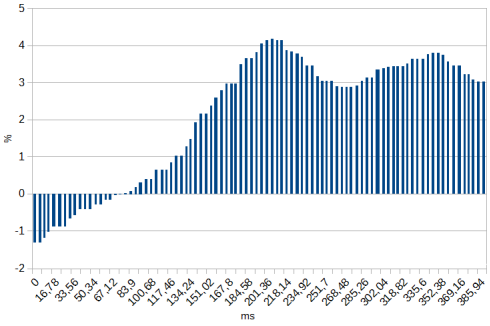
Inclinazione stimata (gradi)	Tempo trascorso (sec)	Spostamento stimato (cm)
17.25	0.402832031	8.938171182
17.51	0.411224365	-9.378208362
17.22	0.428009034	9.1419547257
17.11	0.419616699	-7.9282578363
18.18	0.377655029	7.9678705199
17.55	0.478363037	-9.4813642269
17.27	0.402832032	8.2095841135
17.40	0.411209137	-9.4929367739
17.28	0.419616699	8.8897581182
17.38	0.369262696	-9.5381924051

Tabella 5.10: Spostamenti stimati dal filtro di Kalman per traslazioni di 9cm in avanti e indietro su un piano inclinato di 18°

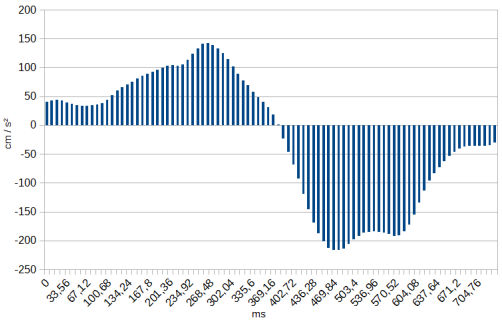
Abbiamo infine testato l'algoritmo su di un caso reale, facendo a mano libera, anziché con l'aiuto di una guida e di un tavolo, spostamenti orizzontali di circa 9cm in un verso e in quello opposto. Ciò che è emerso è riassunto nella *Tabella 5.11*. Nella *Figura 5.14* invece ci sono maggiori dettagli che riguardano il primo spostamento e idem nella *Figura 5.15*, ma in riferimento all'ultimo spostamento, cioè quello più insolito rispetto agli altri dal momento che stima uno spostamento finale molto incongruente con quello effettivo.



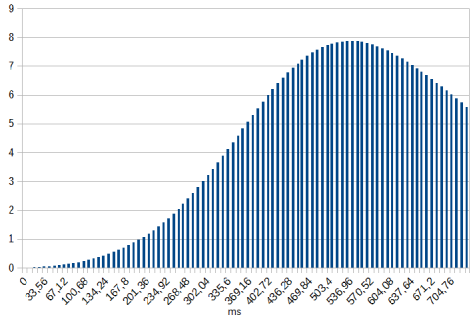
(a) Accelerazione pura (output dell'accelerometro)



(b) Percentuale dell'infuenza della gravità sull'asse Y (100 significa che l'asse Y è perpendicolare al terreno)

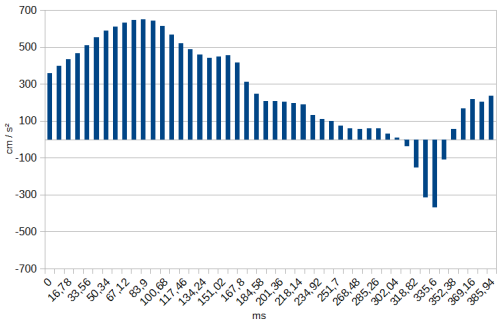


(c) Accelerazione in cm/s^2 dedotta col filtro di Kalman

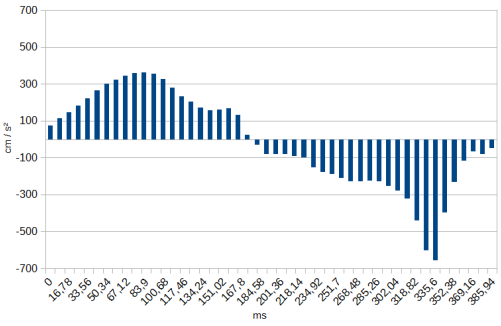


(d) Spostamento in cm dedotto col filtro di Kalman

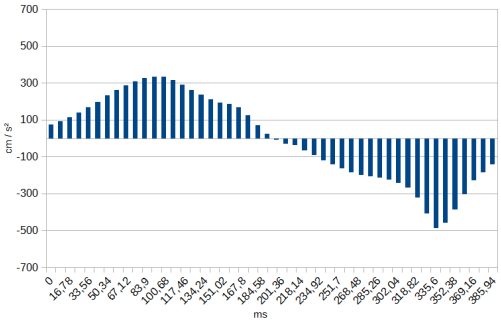
Figura 5.12: Esempio delle curve ottenute traslando lo smartphone di 9cm su di un piano non inclinato e rimuovendo l'influenza della gravità dalle letture dell'accelerometro; ogni campione è stato raccolto a 8.3ms dal precedente



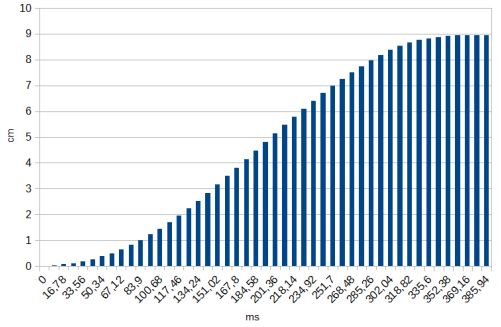
(a) Accelerazione pura (output dell'accelerometro)



(b) Accelerazione priva della componente gravitazionale

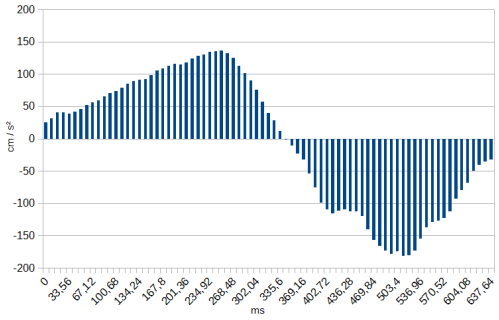


(c) Accelerazione dedotta col filtro di Kalman

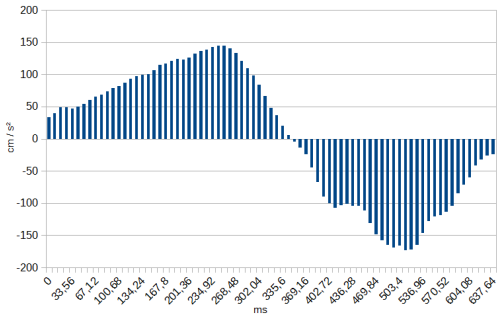


(d) Spostamento dedotto col filtro di Kalman

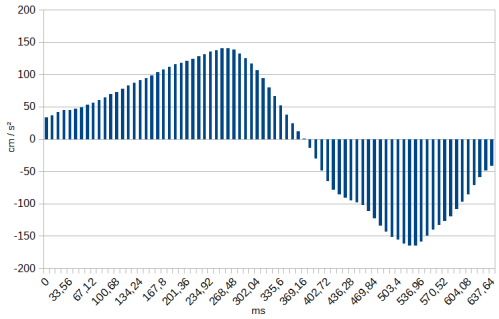
Figura 5.13: Esempio delle curve ottenute traslando lo smartphone di 9cm su di un piano inclinato e rimuovendo l'influenza della gravità dalle letture dell'accelerometro; ogni campione è stato raccolto a 8.3ms dal precedente



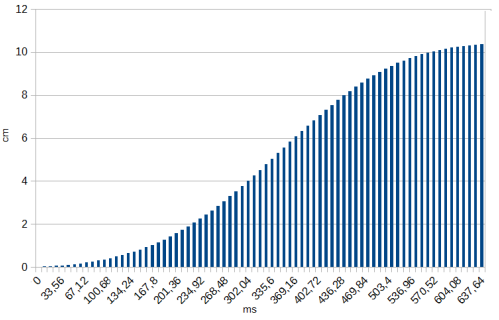
(a) Accelerazione pura (output dell'accelerometro)



(b) Accelerazione priva della componente gravitazionale in cm/s^2

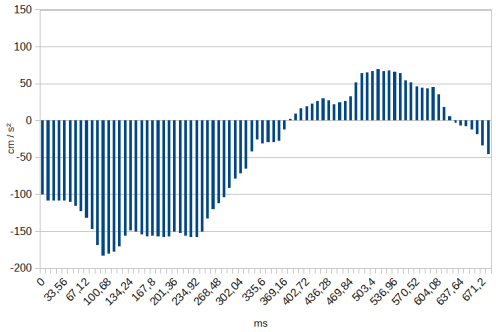


(c) Accelerazione dedotta col filtro di Kalman

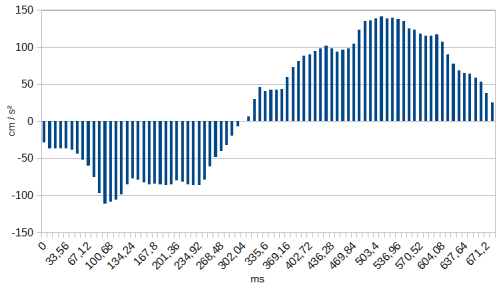


(d) Spostamento dedotto col filtro di Kalman

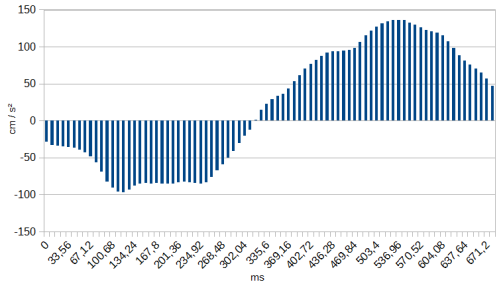
Figura 5.14: Esempio delle curve ottenute dal primo caso di *Tabella 5.11*; ogni campione è stato raccolto a 8.3ms dal precedente



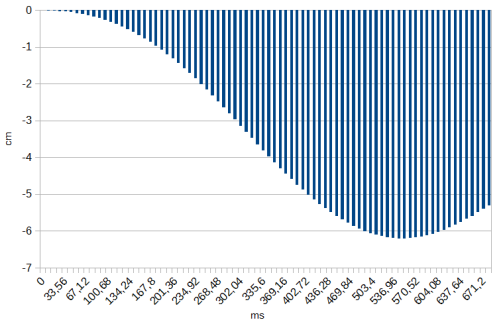
(a) Accelerazione pura (output dell'accelerometro)



(b) Accelerazione priva della componente gravitazionale



(c) Accelerazione in dedotta col filtro di Kalman



(d) Spostamento dedotto col filtro di Kalman

Figura 5.15: Esempio delle curve ottenute dall'ultimo caso di *Tabella 5.11*; ogni campione è stato raccolto a 8.3ms dal precedente

Inclinazione stimata (gradi)	Tempo trascorso (sec)	Spostamento stimato (cm)
-0.49	0.646211812	10.3752079541
-0.89	0.646209716	-10.4319301087
1.06	0.553894043	8.2391378003
-0.65	0.621032715	-9.0916791318
-1.98	0.604248047	10.3902810302
-2.35	0.763702393	-7.5913710078
-2.10	0.654613454	12.6151655848
-0.73	0.604248047	-8.335223095
-0.64	0.763702393	10.5419139806
-0.73	0.688171387	-5.1988419816

Tabella 5.11: Spostamenti stimati dal filtro di Kalman per traslazioni di 9cm circa in entrambi i sensi

Dopo tutti questi test, quella che emerge è nuovamente una situazione per nulla incoraggiante. Già dalla prime prove infatti l'errore non era piccolo e, proseguendo sempre di più verso test realistici, la situazione non ha fatto altro che peggiorare, portando addirittura ad un errore di circa il 42% come mostra l'ultimo dato di Tabella 5.11. Quello che abbiamo fatto quindi è stato abbandonare l'uso dell'accelerometro e ripartire da zero con la stima del moto a partire dalla fotocamera, ma questa volta con una libreria diversa.

5.3 Fotocamera: KLT-tracker

5.3.1 Introduzione

Avendo fino a qui riscontrato considerevoli errori di stima, abbiamo tentato di cambiare strumenti di lavoro per la fotocamera. In particolare, invece che usare *OpenCV*, abbiamo testato un'altra libreria di *computer vision*, sempre scritta in *Java* e sempre *open source*: *BoofCV*. Grazie a *BoofCV* siamo riusciti a ridare nuova vita all'algoritmo bastato sul metodo Lucas-Kanade, nel senso che ora, oltre che ad essere molto più fluido (circa 16 frame al secondo anziché 8), è anche utilizzabile nella sua versione più conosciuta, cioè la KLT (vedi capitolo 4.1.4).
Per raggiungere i nostri scopi abbiamo usato il metodo `createScaledBitmap` della classe `Bitmap`, disponibile nelle librerie Android, per ridimensionare le immagini, il metodo `klt` della classe `FactoryPointTracker` per implementare il KLT-tracker, il metodo `surfFast` della classe `FactoryDetectDescribe` per rilevare le SURF feature, il metodo `scoreEuclidean` della classe `FactoryAssociation` per fare la loro associazione su immagini differenti e il

metodo `greedy`, sempre della classe `FactoryAssociation`, per fare in modo che vengano valutate tutte le possibili combinazioni.
Vediamo quindi come funziona questo nuovo algoritmo.

5.3.2 L'algoritmo

Dal momento che la libreria *BoofCV* si è rivelata molto efficiente, l'algoritmo da noi implementato non ricorre alla sensor fusion: solo fotocamera e microfono sono infatti utilizzati, e quest'ultimo soltanto per rilevare in background la presenza di un soffio, senza quindi che venga "fuso" in alcun modo. Le parti principali sono tre, ossia calibrazione, stima del movimento e rilevamento del soffio. Qui di seguito vengono riportati maggiori dettagli per ognuna di esse.

CALIBRAZIONE

Prima di cominciare a suonare lo strumento, l'utente deve indicare all'algoritmo dove siano la quinta e la decima canna, soffiando prima sull'una e poi sull'altra. Così facendo vengono calcolati e salvati i *SURF Keypoints* delle relative immagini e viene quindi fatto un *match* tra le due, in modo da capire di quanti pixel lo smartphone si sia spostato. Avendo così la distanza tra le due canne, è possibile risalire alla grandezza Δx (sempre in pixel) di una singola canna e questa è un'informazione molto utile dal momento che consente di ricavare le posizioni di tutte le altre. Al termine della calibrazione infatti, la bocca sarà perfettamente centrata nella decima canna e allora, se viene rilevato uno spostamento di, ad esempio, $\frac{\Delta x}{2} + 1$, l'algoritmo saprà che la bocca si trova sull'orlo della canna successiva; se invece lo spostamento rilevato è di $\frac{3\Delta x}{2} + 1$, la canna sarà quella dopo ancora e così via.

STIMA DEL MOVIMENTO

Questa parte basata su KLT può essere eseguita in loop dopo aver concluso la parte di calibrazione. In tal modo è sempre noto il numero della canna sopra la quale è presente la bocca dell'esecutore così da poter sempre essere in grado di suonare la nota giusta.
Ogni immagine di input, prima di essere elaborata, è ridimensionata di una certa quantità modificabile attraverso l'activity *preferences* accessibile dalla schermata iniziale; di default è $320px \times 240px$. Per quanto riguarda invece la funzione che esegue effettivamente il tracking, la piramide usata è a tre livelli (il livello 0 corrisponde all'immagine non rielaborata) ed il numero massimo di feature consentito è 150 (oltre questo valore non abbiamo riscontrato particolari benefici per la stima).

Andando più nel dettaglio, le istruzioni che vengono eseguite ad ogni nuovo frame sono le seguenti:

1. Diminuisci la risoluzione dell'immagine in base al valore riportato nell'activity delle preferenze e convertila in scala di grigi
2. Esegui il KLT-tracker trovando le feature nel frame corrente e associandole con quelle del precedente (ammesso che ce ne sia uno)
3. Se il numero di match riusciti è inferiore a 50 fai *tabula rasa* di tutte le feature in memoria e comincia un nuovo processo di KLT-tracking
4. Se è stato possibile stimare i vettori di movimento locale attraverso l'associazione delle feature, calcola la componente di spostamento orizzontale più frequente
5. Se il numero di occorrenze di tale componente è almeno 10, allora essa rappresenta lo spostamento globale, altrimenti lo spostamento globale è 0

- In background, se è attivata la variabile booleana che segnala la necessità di calcolare lo spostamento con le SURF feature, disattivala e calcola le SURF feature sul frame corrente confrontandole con quelle ottenute in fase di calibrazione; se ci sono stati almeno 10 match, correggi lo spostamento rilevato dal KLT-tracker

Come negli algoritmi precedenti, anche in questo caso i valori dei vari parametri sono stati ricavati sperimentalmente, cercando di bilanciare le opposte esigenze di affidabilità e velocità. In particolare, abbiamo constatato che sia il valore "50" del punto 3 che il valore "10" del punto 5 potrebbero essere diminuiti senza compromettere sensibilmente la stima; abbiamo comunque deciso di tenere questi coefficienti per questioni di "sicurezza" dell'output.

RILEVAMENTO DEL SOFFIO

Il rilevamento del soffio avviene in background, esattamente come il calcolo delle SURF feature, e l'output del microfono è elaborato con la *Fast Fourier Transform* in modo da ottenere la lista delle frequenze coinvolte con relativa intensità. Dal momento che un soffio è composto da frequenze basse, soltanto quelle inferiori a 157Hz (soglia sperimentale) vengono considerate. In particolare, quando una di queste presenta un sufficiente valore d'intensità, viene incrementata di un'unità una variabile di guardia ed ogni volta che questa raggiunge un certo valore massimo viene fatto partire il suono della canna che, ricordiamo, al momento è un audio

provvisorio. Per consentire poi che questa riesca a riazzerarsi, ad ogni iterazione tale variabile viene decrementata di un'unità fino, appunto, a zero.
Se emerge che il soffio sta durando da un po' di tempo, l'algoritmo assume che lo smartphone sia fermo e che quindi sia un buon momento per stimare meglio la posizione corrente. In questo caso infatti viene attivata la variabile booleana che segnala la necessità di calcolare lo spostamento con le SURF feature, consentendo così una stima più accurata dello spostamento del dispositivo.

Capitolo 6

Confronto degli algoritmi

Nonostante siano abbastanza diversi, abbiamo voluto cercare di confrontare gli ultimi tre algoritmi esposti. L'idea iniziale era quella di usare la stessa sequenza d'immagini in tutti e tre i casi ma, purtroppo, al momento *OpenCV* per Android non offre la possibilità di elaborare video presenti in memoria. Siamo stati quindi costretti ad usare sequenze d'immagini diverse per ogni caso ma ogni volta l'abbiamo fatto cercando comunque di mantenere un certo grado di ripetibilità. Gli esperimenti che abbiamo condotto sono di tre tipi: spostamenti da una canna a quella successiva, spostamenti da una canna a quella tre posizioni più avanti, spostamenti da una canna a quella cinque posizioni più avanti. Ogni prova è stata condotta traslando lo smartphone in entrambi i sensi, mettendoci più o meno sempre lo stesso tempo per ogni spostamento (meno di mezzo secondo), ed utilizzando l'immagine di *Figura 5.3* posta a 9cm dal dispositivo come mostra la *Figura 5.4*. Per quanto riguarda le distanze, abbiamo assunto che i punti centrali di due canne consecutive distino 6.5mm.
Di seguito dunque sono riportati i risultati ottenuti per diversi test ma, ovviamente, le presenti tabelle hanno il solo scopo di dare un'idea generale della situazione perché, oltre al fatto che l'input non è mai esattamente lo stesso, la seconda versione dell'algoritmo FAST-tracker non dà in output il numero di canna corrente ma bensì la lo spostamento in pixel dal punto di partenza.

CANNA CORRETTA	FAST-tracker versione 2	FAST-tracker versione 3	KLT-tracker
14	0	14	14
13	-102	13	13
12	-231	12	12
11	-324	11	11
10	-427	10	10
9	-520	9	9
8	-628	8	8
7	-715	7	7
6	-820	6	6
5	-919	5	5
4	-1021	4	4
3	-1123	3	3
2	-1208	2	2
1	-1314	1	1
2	-1240	2	2
3	-1140	3	3
4	-1041	4	4
5	-957	5	5
6	-858	6	6
7	-765	7	7
8	-678	8	8
9	-566	9	9
10	-478	10	10
11	-374	11	11
12	-279	12	12
13	-193	13	13
14	-100	14	14

Tabella 6.1: Risultati dei tre algoritmi su spostamenti tra una canna e quella successiva

CANNA CORRETTA	FAST-tracker versione 2	FAST-tracker versione 3	KLT-tracker
14	0	14	14
11	-305	11	11
8	-607	8	8
5	-875	5	5
2	-1170	2	2
5	-913	5	5
8	-610	8	8
11	-343	11	11
14	-91	14	14

Tabella 6.2: Risultati dei tre algoritmi su spostamenti tra una canna e quella tre posizioni più avanti

CANNA CORRETTA	FAST-tracker versione 2	FAST-tracker versione 3	KLT-tracker
14	0	14	14
9	-302	nessun match	9
4	-387	4	5*
9	-110	nessun match	9
14	417	14	13*
9	98	nessun match	10*
4	-148	4	4
9	389	nessun match	9
14	835	14	12*

Tabella 6.3: Risultati dei tre algoritmi su spostamenti tra una canna e quella cinque posizioni più avanti; i risultati segnati con l'asterisco sono stati ogni volta corretti calcolando le SURF feature prima di continuare

Quello che emerge da questi dati è che col KLT-tracker siamo riusciti ad ottenere un algoritmo molto affidabile che ha saputo superare i limiti delle versioni precedenti. Per piccoli movimenti infatti il tracker è molto soddisfacente, e per quelli un po' più grandi il confronto con i *SURF Keypoints* riesce comunque a correggere alla perfezione qualsiasi deviazione.
La versione 2 dell'algoritmo invece ha dimostrato di non essere in grado di tornare al punto di partenza, soprattutto nel terzo caso, ed anche la versione 3 ha molti limiti: sebbene la maggior parte delle volte riesca ad indovinare correttamente la canna, soffre del fatto che il posizionamento deve essere quasi perfetto. Notiamo

infatti che nell'ultima tabella la canna 9 non viene mai riconosciuta. Ciò è dovuto al fatto che l'immagine corrente non trova molte corrispondenze né facendo il confronto con l'immagine memorizzata della canna 8 e né facendo il confronto con l'immagine memorizzata della canna 10. Il flusso ottico infatti, soprattutto se si serve dei *FAST Keypoints*, soffre del fatto che immagini troppo diverse danno luogo a un match povero, facendo così concludere all'algoritmo che le corrispondenze trovate sono troppo esigue per considerare il confronto riuscito.

Capitolo 7

L'app

L'app, che ricordiamo è basata sulla libreria *BoofCV*, presenta nella schermata iniziale un pulsante in alto a destra per le preferenze, dove poter scegliere la videocamera da usare e la dimensione di ogni frame da elaborare, ed un grande bottone *START* al centro che permette di cominciare la fase di calibrazione, dove l'utente è tenuto a soffiare prima sulla quinta canna e poi sulla decima, curandosi di puntare la videocamera che effettua le riprese verso il proprio petto (vedi *Figura 7.1*). Fatto ciò, è tutto pronto ed è quindi possibile iniziare a suonare il prototipo di flauto di Pan virtuale come mostrano le immagini di *Figura 7.2* e *Figura 7.3*.

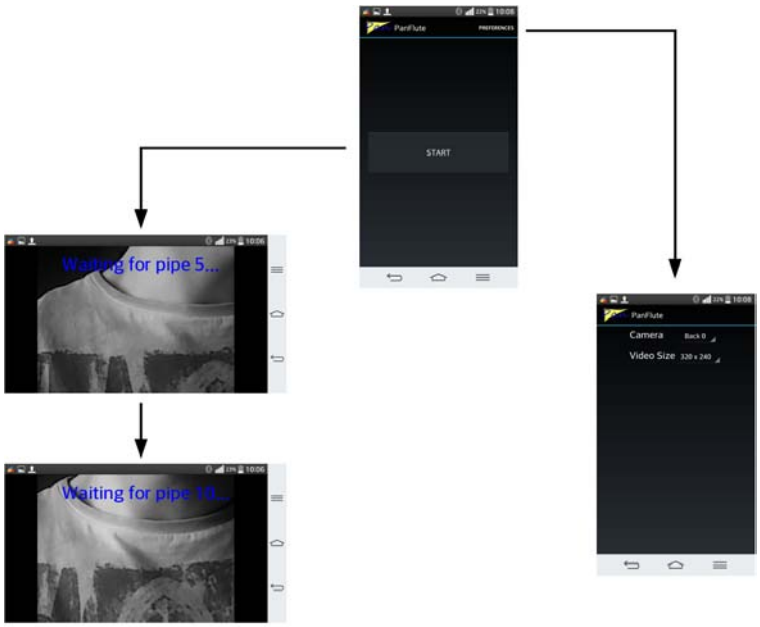


Figura 7.1: Struttura dell'app: dalla schermata iniziale è possibile accedere alle preferenze ed al prototipo del flauto di Pan facendo prima le due calibrazioni necessarie

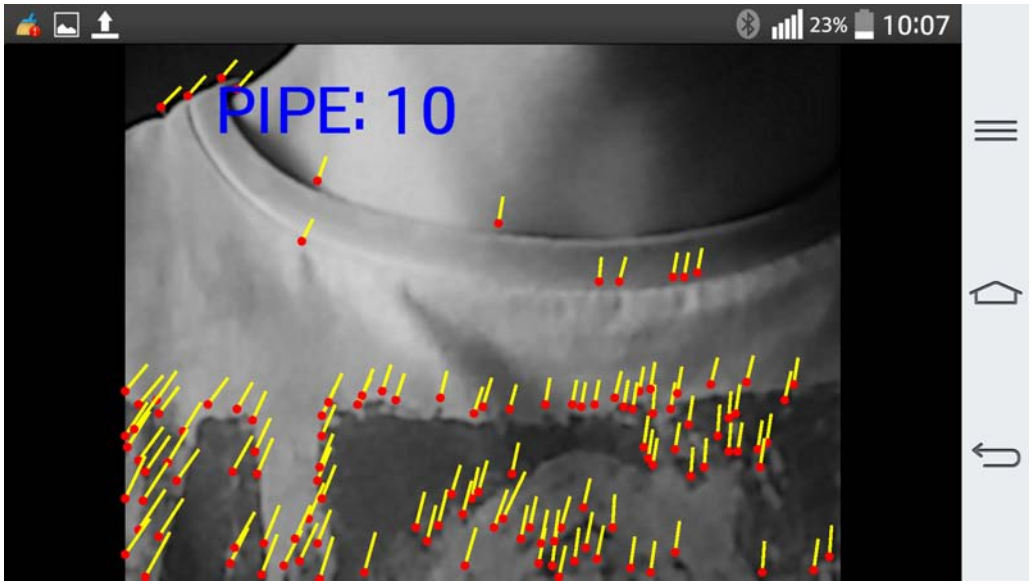


Figura 7.2: Dopo la fase di calibrazione l'app consente di produrre suoni in base alla canna rilevata (in questo caso la decima); i punti rossi rappresentano le feature trovate nel frame corrente e le linee gialle rappresentano i vettori di movimento locale

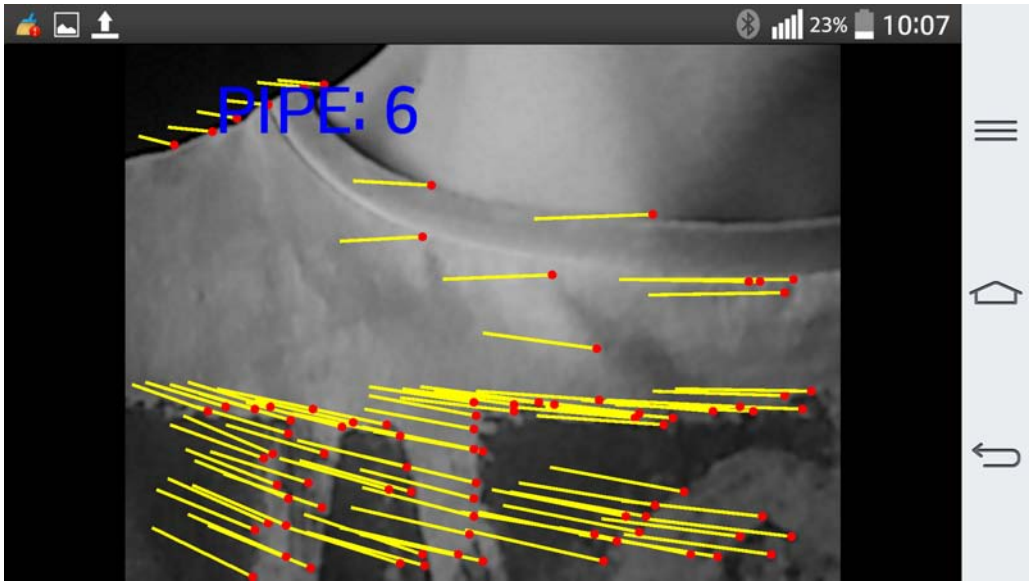


Figura 7.3: Dopo la fase di calibrazione l'app consente di produrre suoni in base alla canna rilevata (in questo caso la sesta); i punti rossi rappresentano le feature trovate nel frame corrente e le linee gialle rappresentano i vettori di movimento locale

Capitolo 8

Conclusioni e note finali

Abbiamo tentato di far rivivere su dispositivi Android l'antico flauto di Pan a quattordici canne, custodito nel Museo di Scienze Archeologiche e d'Arte dell'Università degli Studi di Padova. Per farlo, inizialmente abbiamo stimato i movimenti dello smartphone attraverso la fotocamera, usando la libreria *OpenCV* per Android, e l'accelerometro. Entrambi gli strumenti si sono rivelati inadeguati per spostamenti centimetrici: *OpenCV* perché computazionalmente oneroso e l'accelerometro, seppur aiutato dal filtro di Kalman, perché molto rumoroso. Quest'ultimo in particolare non ha tratto alcun vantaggio dalla rimozione dell'impatto gravitazionale grazie all'impiego simultaneo di giroscopio e magnetometro, anzi, abbiamo osservato che, così facendo, l'errore della stima non fa altro che aumentare. A tutto ciò si aggiunge anche il fatto che lo stress cui è sottoposta la CPU porta ad un surriscaldamento della stessa che può toccare i 56° C, causando di conseguenza uno sgradevole innalzamento della temperatura del dispositivo.
Abbiamo così rivisto l'intero progetto, utilizzando la libreria *BoofCV* per l'elaborazione delle immagini e abbandonando l'impiego dell'accelerometro. Così facendo abbiamo constatato un incremento sensibile delle prestazioni e dell'affidabilità che ci hanno consentito di sviluppare un'app molto più performante rispetto a quella che avremmo potuto sviluppare se avessimo perseguito sulla vecchia strada. In particolare, per movimenti lenti siamo riusciti ad ottenere risultati ottimi, nel senso che lo spostamento rilevato è sempre corretto; per movimenti più veloci invece il KLT-tracker risulta avere qualche difficoltà ad elaborare i dati correttamente, il che implica una stima del moto più o meno inesatta, e questa è sicuramente una questione che può essere meglio approfondita in successivi lavori. In ogni caso, siamo riusciti comunque a fare in modo che questo errore fosse sempre correttamente rettificato calcolando in poche centinaia di millisecondi le SURF feature. La fase di calibrazione iniziale inoltre offre un vantaggio non indifferente: permette infatti di non fissare a priori la lunghezza delle canne e/o la distanza tra la fotocamera e il petto dell'"esecutore", dando così all'utente una maggiore libertà di utilizzo.

Naturalmente, poiché abbiamo usato il metodo Lucas-Kanade come base della stima del moto, assumiamo che i movimenti siano sufficientemente fluidi e che la luminosità dell'ambiente sia più o meno costante nel tempo. Un altro limite ancora non esplicitato è il fatto che il vestito dell'"esecutore" dovrà essere il più ricco di *features* possibile, cioè il meno monocromatico possibile. Questo proprio perché tutto il lavoro viene fatto studiando l'immagine catturata dalla videocamera, e quindi la scena dovrà presentare dei buoni punti d'aggancio per una stima accurata. In genere comunque tutte queste assunzioni non sono troppo vincolanti e nella maggior parte dei casi non costituiscono un grave problema. Ciò che invece più spesso rappresenta un impedimento, e che sicuramente è una parte da migliorare prima di considerare il progetto del flauto di Pan virtuale concluso, è l'acquisizione audio che ancora fatica a distinguere un soffio da un rumore ambientale e che ha difficoltà a rilevare soffi non diretti ad uno dei microfoni, ma problematiche di questo tipo non sono oggetto della nostra ricerca.

Bibliografia

[1] Anastasios I. Mourikis and Stergios I. Roumeliotis, *A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation*

[2] Yunsu Bok, Youngbae Hwang, In So Kweon, *Accurate Motion Estimation and High-Precision 3D Reconstruction by Sensor Fusion*

[3] Kurt Konolige, Motilal Agrawal, Joan Solà, *Large Scale Visual Odometry for Rough Terrain*

[4] Pablo Esteban Quiroga Garcia Wenjie Li, *On Indoor Positioning for Mobile Devices*, Master's Thesis in Communication Engineering & Master's Thesis in Integrated Electronic Systems Design, Chalmers University of Technology

[5] Hamed Ketabdar, Amirhossein Jahnbekam, Kamer Ali Yuksel, *MagiMusic: Using Embedded Compass (Magnetic) Sensor for Touch-less Gesture Based Interaction with Digital Music Instruments in Mobile Devices*

[6] Hamed Ketabdar, Hengwei Chang, *MagiGuitar: A Guitar that is Played in Air!*

[7] Georg Essl, Michael Rohs, *Shamus - A Sensor-Based Integrated Mobile Phone Instrument*

[8] Gunter Geiger, *Using the Tough Screen as a Controller for Portable Computer Music Instruments*

[9] Nicholas Gillian, Sile O'Modhrain, Georg Essl, *Scratch-Off: A gesture based mobile music game with tactile feedback*

[10] Ge Wang, *Ocarina: Designing the iPhone's Magic Flute*

[11] Filippo Vella, Alfio Castorina, Massimo Mancuso, Giuseppe Messina, *Digital Image Stabilization by Adaptive Block Motion Vectors Filtering*

[12] Ahmad Rahmati, Clayton Shepard, Lin Zhong, *NoShake: Content Stabilization for Shaking Screens of Mobile Devices*, Dept. of Electrical and Computer Engineering Rice University, Houston, TX, United States

[13] S. Erturk, *Digital Image Stabilization with Sub-Image Phase Correlation Based Global Motion Estimation*

[14] Sebastiano Battiato, Giovanni Gallo, Giovanni Puglisi, *SIFT Features Tracking for Video Stabilization*, University of Catania

[15] Richard Szeliski, *Computer Vision: Algorithms and Applications*

[16] Gary Bradski, Adrian Kaehler, *Learning OpenCV*

[17] Sebastiano Battiato, Arcangelo Ranieri Bruna, Giovanni Puglisi, *A Robust Block-Based Image/Video Registration Approach for Mobile Imaging Devices*

[18] Davide Scaramuzza, Friedrich Fraundorfer, *Visual Odometry Part I & Visual Odometry Part II*

[19] Georg Essl, Michael Rohs, *Interactivity for Mobile Music-Making*

[20] Carolina Brum Medeiros, Marcelo M. Wanderley, *A Comprehensive Review of Sensors and Instrumentation Methods in Devices for Musical Expression*, Input Devices and Music Interaction Laboratory (IDMIL), Centre for Interdisciplinary Research in Music Media and Technology (CIRMMT), McGill University, 555 Sherbrooke St West, Montreal, Canada

[21] Fatemeh Abyarjoo, Armando Barreto, Jonathan Cofino, Francisco R. Ortega *Implementing a Sensor Fusion Algorithm for 3D Orientation Detection with Inertial/Magnetic Sensors*, Electrical and Computer Engineering Department, Florida International University, Miami, FL. USA; School of Computing and Information Science, Florida International University, Miami, FL. USA

[22] Paul Lawitzki, Joachim Charzinski (supervisor), *Application of Dynamic Binaural Signals in Acoustic Games (Master Thesis)*, Stuttgart Media University, Faculty Print and Media, Stuttgart

[23] Jay Esfandyari, Roberto De nuccio, Gang Xu, *Solutions for MEMS sensor fusion*, `http://www.mouser.com/applications/sensor_solutions_mems/`, STMicroelectronics, Coppell, TX USA

[24] Greg Welch, Gary Bishop *An Introduction to the Kalman Filter*, University of North Carolina at Chapel Hill, Department of Computer Science Chapel Hill

[25] Jianbo Shi, Carlo Tomasi, *Good Features To Track*, Computer Science Department; Cornell University - Ithaca (Shi), Stanford University - Stanford (Tomasi)

[26] Edward Rosten, Tom Drummond, *Machine learning for high-speed corner detection*, Department of Engineering, Cambridge University,UK

[27] David Sachs, *Sensor Fusion on Android Devices: A Revolution in Motion Processing [youtube video]*, https://www.youtube.com/watch?v=C7JQ7Rpwn2k

[28] Kurt Seifert, Oscar Camachoi, *Implementing Positioning Algorithms Using Accelerometers*

[29] Herbert Bay, Tinne Tuytelaars, Luc Van Gool *SURF: Speed Up Robust Features*

[30] Fabio Varesano, Luca Console (Advisor), Marco Grangetto (Co-Advisor), *Using Arduino for Tangible Human Computer Interaction (Master Thesis)*

[31] https://itunes.apple.com/us/app/ocarina/id293053479?mt=8

[32] http://cas1.elis.ugent.be/cas/en/mems/

[33] http://www.neowin.net/news/in-focus-a-closer-look-at-the-nokia-lumia-925-camera

[34] https://microphones.audiolinks.com/microphones.shtml#Microphones

[35] http://www.codeproject.com/Articles/729759/Android-Sensor-Fusion-Tutorial

[36] http://greg.czerniak.info/guides/kalman1/

[37] https://en.wikipedia.org/wiki/Pyramid_\%28image_processing\%29

[38] http://www.edwardrosten.com/work/fast.html

[39] http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html

[40] https://ianvanlan.wordpress.com/2012/08/11/gondole-gondola/