Università degli Studi di Padova
Scuola di Ingegneria
Dipartimento di Ingegneria dell'Informazione

Stima della posizione in dispositivi mobili: una analisi dettagliata delle informazioni fornite da accelerometro e fotocamera

Relatore: Prof. Fantozzi Carlo
Laureando: Sacilotto Simone

Laurea Magistrale in Ingegneria Informatica
12 Dicembre 2016
Anno Accademico 2015/2016

Indice

1 Introduzione

Il lavoro di questa tesi è legato al Flauto di Pan conservato nel Museo di Scienze Archeologiche e d'Arte dell'Università degli studi di Padova. Questo reperto, ritrovato in ottimo stato in Egitto durante le ricerche archeologiche condotte da Carlo Anti negli anni 1928-1936, è stato da poco oggetto di restauro ed attente analisi, ed è ora esposto nel museo. Lo strumento, di epoca ellenistico-romana, è composto da 14 canne palustri di lunghezza e diametri diversi, legate con fibre vegetali, e tappate ad una estremità con materiale organico (probabilmente cera o propoli, tuttavia nel tempo si sono perse quasi tutte le tracce del materiale effettivamente utilizzato). Vista la fragilità e la rarità di questo reperto (è uno dei soli tre flauti di età ellenistico-romana esistenti al mondo, comparabile con quello preservato al museo di Alessandria d'Egitto), esso è conservato in una teca con condizioni ambientali controllate, e quindi precluso ad un accesso diretto da parte del pubblico. Per ridare vita allo strumento, quindi, è stata realizzata, a fianco della teca in cui il flauto è esposto, una postazione interattiva che permette un esplorazione sia visiva che acustica, dando la possibilità al pubblico di suonare lo strumento per mezzo o di un'interfaccia touchscreen o di microfoni, grazie alla ricostruzione virtuale effettuata.
In parallelo a questo, è stato avviato un altro progetto, il cui obbiettivo è creare una versione virtuale dello strumento, che sia possibile suonare su smartphone. Tramite l'utilizzo di sensori come accelerometro, giroscopio, magnetometro e fotocamera, presenti nei dispositivi mobili moderni, si vuole realizzare un'app che simuli il Flauto di Pan, permettendo quindi di suonarlo con il proprio dispositivo, non limitando la fruizione alla sola visita al museo, ma dando anche la possibilità all'utente di "portare a casa" lo strumento. L'obiettivo dell'app, quindi, è simulare con la massima fedeltà possibile un flauto di Pan: essa dovrà permettere all'utente di suo-

nare una nota tramite un soffio e di manovrare il dispositivo come un vero flauto. In altre parole l'utente selezionerà la nota (ovvero la canna in cui si vuole soffiare) tramite lo spostamento del dispositivo, in modo tale che la bocca sia posizionata frontalmente alla canna virtuale, come se si stesse tenendo in mano un vero flauto.
Questo progetto è motivato dall'assenza, al momento, di app simili. Sono, infatti, disponibili applicazioni che permettono di suonare uno strumento virtuale tramite smartphone, tuttavia in tutte queste manca la componente di "movimento". Un esempio è l'app *Zampoña*[23]: essa permette di suonare uno strumento virtuale simile ad un flauto di Pan. Tuttavia l'utente, per suonare una nota, deve premere sullo schermo in corrispondenza della canna desiderata, vivendo, quindi, un esperienza utente molto lontana da quella da noi voluta. Un'app che si avvicina maggiormente all'esperienza utente da noi desiderata è *Ocarina*[24] (disponibile solamente per l'ambiente iOS): a differenza della precedente, in questa applicazione per suonare una nota è necessario soffiare nel microfono del dispositivo. Tuttavia la selezione della canna è ancora effettuata tramite la pressione sul touchscreen, risultando nuovamente in un esperienza utente differente da quella cercata.
Il problema principale che l'app deve risolvere, quindi, consiste nel tracciare nel tempo lo spostamento del dispositivo in modo da conoscerne la posizione e poter ricostruire il cammino percorso nello spazio: questo problema è noto come "motion tracking". Due diversi approcci per la risoluzione di questo problema sono l'utilizzo di hardware che fornisca dati direttamente correlati al moto, oppure tecniche di computer vision. Con il primo approccio si utilizzano sensori come per esempio accelerometri, giroscopi e sistemi di posizionamento satellitare, e si sfruttano le informazioni restituite per calcolare il cammino percorso, ad esempio integrando il segnale fornito dall'accelerometro per ottenere lo spostamento causato dall'accelerazione misurata. L'approccio che fa uso della computer vision, invece, mira ad utilizzare le informazioni fornite dalla fotocamera allo scopo di calcolare il movimento tra due immagini consecutive (per esempio il flusso ottico tra due frame consecutivi di un video ripreso in tempo reale) e da questa informazione sulle immagini ricostruisce il moto effettuato dalla fotocamera, e quindi dal dispositivo.
Il problema che deve essere affrontato, in particolare, è un caso speciale del

motion tracking, in quanto i movimenti che devono essere tracciati seguono una traiettoria approssimabile grossolanamente con una traslazione, la cui entità massima è nell'ordine della decina di centimetri. Questo sottoproblema, tuttavia, non è stato ancora studiato approfonditamente, e tutta la letteratura esaminata considera solamente spostamenti di entità nell'ordine di metri o superiore. Ciononostante, lavoro nell'ambito del progetto è già stato effettuato: in [15] si è affrontato il problema con l'obiettivo principale di realizzare una soluzione funzionante. Sono state considerate sia tecniche di computer vision, che tecniche basate su accelerometro, però, in quest'ultimo caso, si sono ottenuti risultati negativi, quindi maggiori risorse sono state investite nelle tecniche di computer vision. In questo modo si è ottenuta un'app funzionante, le cui prestazioni, però, risultano notevolmente limitate: la fluidità di utilizzo è scarsa, ed anche la precisione nella rilevazione dei movimenti non raggiunge la qualità desiderata.
Questi risultati motivano il lavoro della presente tesi: un'analisi più concentrata sulle informazioni fornite dai sensori stessi, con l'obiettivo di comprendere meglio le loro limitazioni e potenzialità. In particolare, l'analisi è stata concentrata sui sensori accelerometro e fotocamera, in quanto si è ritenuto che questi fossero i sensori più promettenti per ottenere l'informazione di posizione ricercata. Infatti l'accelerometro è la scelta più immediata per il calcolo della posizione, in quanto ne fornisce la derivata seconda e quindi, nel caso ideale di informazione perfetta, una doppia integrazione sarebbe sufficiente ad ottenere il dato cercato. La fotocamera, invece, è stata considerata in quanto anch'essa fornisce un'informazione strettamente legata al movimento, dato che questo si rispecchia direttamente ed in maniera prevedibile nelle immagini acquisite, per cui una loro adeguata elaborazione permette di ottenere la posizione nel tempo.
Il resto del documento è organizzato nel seguente modo: nel Capitolo 2 viene analizzato il sensore accelerometro, mentre in Capitolo 3 presenteremo i lavoro effettuato sulla fotocamera.
Precisiamo, infine, che tutte le analisi effettuate in questo lavoro sono state svolte utilizzando il software *Wolfram Mathematica* versione 10, eccetto le progettazioni dei filtri descritte nella sezione 2.5 e sezione 2.7, svolte con *Matlab* versione 2015a.

2 Analisi dell'accelerometro

Presentiamo in questo capitolo le analisi effettuate sul segnale fornito dal sensore accelerometro. In particolare, esamineremo i risultati delle elaborazioni effettuate sul segnale al fine di ottenere un risultato più adatto al calcolo della posizione, e le analisi effettuare con lo scopo di capire più a fondo le caratteristiche del segnale stesso.
Per prima cosa descriveremo brevemente il sensore fisico e come questo possa essere gestito in ambiente Android. Successivamente riporteremo i risultati di un lavoro svolto sullo stesso argomento, e preliminare allo studio effettuato, il quale verrà descritto subito dopo in tutto questo capitolo.

2.1 Caratteristiche generali del sensore

2.1.1 Sensore accelerometro

Un accelerometro è un dispositivo fisico che permette di misurare l'accelerazione propria a cui è sottoposto: questa è l'accelerazione relativa al sistema di riferimento di caduta libera. Di conseguenza, un sensore in stato di riposo sulla superficie terrestre riporterà un'accelerazione perpendicolare al suolo, orientata verso il cielo e di intensità pari alla forza di gravità, mentre se il sensore si trovasse in caduta libera, esso misurerebbe accelerazione nulla. Questo è dovuto al principio di equivalenza di Einstein, per cui gli effetti della gravità sono indistinguibili da una accelerazione: per fare un esempio, non è possibile distinguere la differenza tra il trovarsi su un razzo fermo sulla superficie della Terra, e l'accelerare con una forza pari a quella di gravità nello spazio aperto.
Per quanto riguarda il metodo di rilevazione dell'accelerazione, concettualmente il sensore si comporta come una massa collegata ad una mol-

la. Quando un'accelerazione è applicata, la molla si estende fino al punto in cui essa applica sulla massa una forza uguale ed opposta a quella che causa l'accelerazione: a questo punto, misurando l'estensione della molla dalla condizione di riposo è possibile calcolare l'accelerazione presente. Per realizzare fisicamente questo sistema, vari metodi sono stati adottati: alcuni dei più comuni utilizzano elementi piezoelettrici, piezoresistivi o capacitivi, trasformando, quindi, il segnale meccanico in elettrico. Con un elemento capacitivo, per esempio, si sfrutta la variazione di capacità di un condensatore piano al variare della distanza tra le sue armature: la massa, quindi, forma una delle due armature, mentre l'altra è costituita da un elemento fisso. In questo modo, rilevando la variazione della capacità nella struttura costruita è possibile ottenere il valore di accelerazione. La maggior parte dei sensori inclusi nei dispositivi mobili moderni utilizza sistemi basati sulla tecnologia MEMS (Micro Electro-Mechanical Systems) per realizzare un sistema capacitivo, il cui comportamento concettuale è lo stesso di quello descritto precedentemente, solo che esso è realizzato con strutture in silicio in dimensioni nell'ordine della frazione di millimetro.

2.1.2 Accelerometro in ambiente Android

Data l'abbondanza di sensori presente nei dispositivi mobili moderni, Android mette a disposizione degli sviluppatori un sistema unificato per accedere ai dati forniti dai possibili sensori. In particolare, tramite le classi Java Sensor e SensorManager è possibile gestire i sensori presenti sul dispositivo specifico, per esempio ottenendo la lista di sensori disponibili di un certo tipo, o registrandosi/deregistrandosi dalla ricezione di nuovi valori generati da un particolare sensore. Questa funzione di ricezione viene realizzata per mezzo di callback: alla generazione dell'evento di disponibilità di nuovo campione, la funzione viene invocata e riceve le informazioni sul sensore che ha generato l'evento, l'accuratezza ed il valore del campione, oltre al timestamp dell'istante in cui il dato è stato rilevato dal sensore. La frequenza con cui questo evento viene generato, tuttavia, è influenzato da vari fattori sia interni che esterni all'applicazione, come il sistema Android o altre applicazioni. Il controllo disponibile sulla frequenza della generazione degli eventi è la sola possibilità di indicare al sistema un suggerimento sul periodo tra un evento ed il successivo: questo avviene al momento della registrazione della callback, indicando una delle costanti *SEN-*

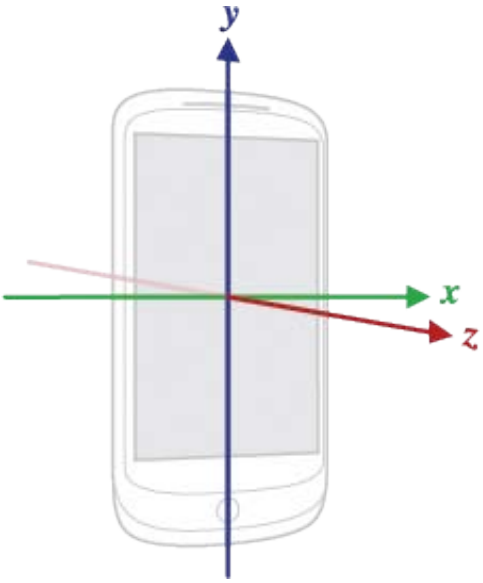


Figura 2.1: *Sistema di riferimento standard adottato da Android.*

SOR_DELAY_NORMAL, *SENSOR_DELAY_UI*, *SENSOR_DELAY_GAME*, *SENSOR_DELAY_FASTEST*. Queste costanti suggeriscono al sistema di consegnare l'evento con un periodo di rispettivamente $200ms$, $60ms$, $20ms$ e $0ms$, ovvero il prima possibile, tuttavia il tempo che effettivamente trascorre tra un evento ed il successivo non è costante o prevedibile. Per utilizzare correttamente i sensori, quindi, è importante tenere a mente questa differenza tra tempo di campionamento proprio del sensore e periodo di generazione degli eventi, oltre alla mancanza di garanzie su quest'ultimo tempo.

Per quanto riguarda l'accelerometro, Android mette a disposizione due diversi sensori: il tipo "Accelerometer" ed il "Linear Acceleration". Il primo sensore fornisce i valori raw restituiti dall'accelerometro fisico, comprendenti, quindi, la forza di gravità. Il secondo tipo, invece, è un sensore simulato, che fornisce la sola accelerazione applicata sul dispositivo senza la forza di gravità, rimossa automaticamente dal sistema. Entrambi i sensori adottano il sistema di coordinate standard definito da Android e visibile in Figura 2.1: posto il dispositivo nella sua orientazione naturale orizzontale o verticale, l'asse x risulta orizzontale e diretto verso la destra del dispositivo, l'asse y è verticale e diretto verso l'alto, mentre l'asse z è perpendicolare allo schermo in direzione uscente da questo.

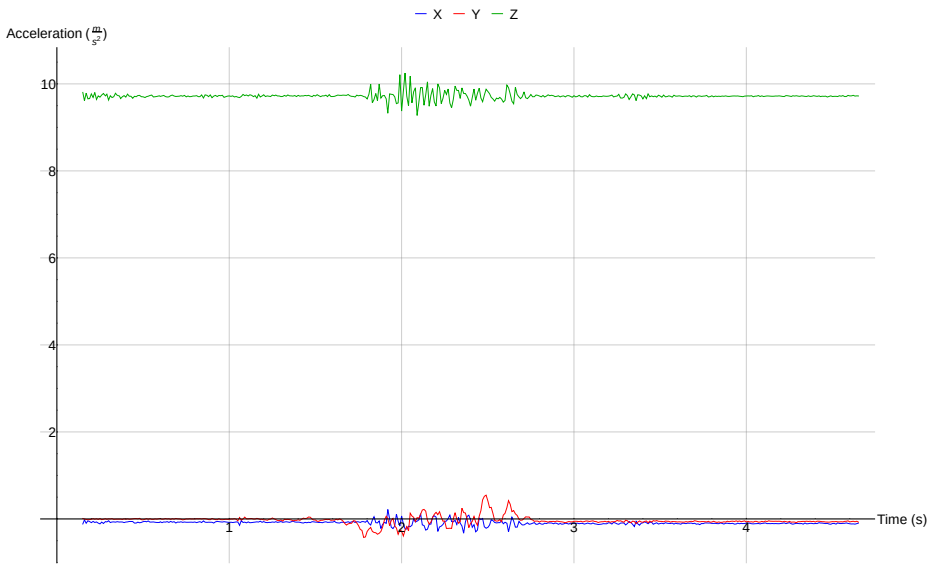


Figura 2.2: *Segnale di accelerazione raw sui tre assi cartesiani (x blu, y rosso, z verde) per un movimento di 5cm*

2.2 Lavoro preliminare

Riportiamo, prima di presentare le analisi svolte, alcuni risultati di un progetto svolto per il corso di "Programmazione di sistemi embedded" in cui è stato effettuato un primo studio del problema di calcolo della posizione a partire dai dati dell'accelerometro.
Per semplificare l'analisi si era deciso di studiare, almeno inizialmente, il caso ideale di moto puramente traslatorio, che approssima il movimento reale compiuto durante l'uso dell'app. Allo scopo di ottenere la posizione, quindi, si era deciso di utilizzare il dato "linear acceleration" fornito dal sistema Android, in quanto composto dalle sole forze di manipolazione agenti sul dispositivo. Per quanto riguarda la generazione ed acquisizione del segnale, invece, il dispositivo utilizzato per le rilevazioni, ed impiegato anche durante tutto lo svolgimento della presente tesi, è un tablet Samsung Galaxy Tab S2 con sistema operativo Android 6. Nell'acquisizione, inoltre, si è cercato di ricreare le condizioni il più possibile vicine a quelle ideali: si è costruita, quindi, una guida per un carrellino su cui poggiare il dispositivo, in modo tale da limitare la direzione di movimento e ridurre eventuali disturbi dovuti allo strisciamento del dispositivo sulla superficie. Tramite un'app appositamente realizzata, si è acquisito così il segnale

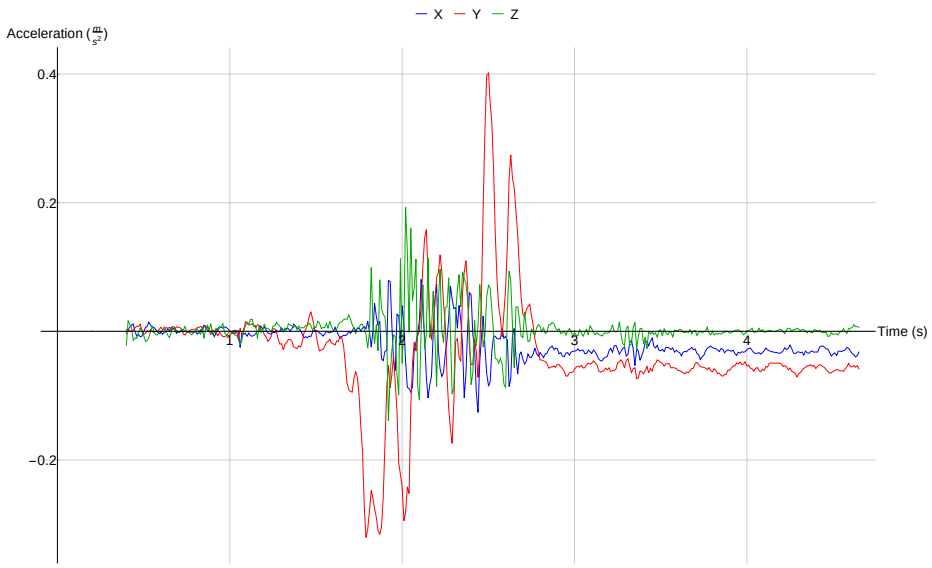


Figura 2.3: *Segnale di accelerazione sui tre assi cartesiani (x blu, y rosso, z verde) per un movimento di 5cm, elaborato con gli algoritmi di rimozione della gravità e filtraggio descritti in sezione 2.2.*

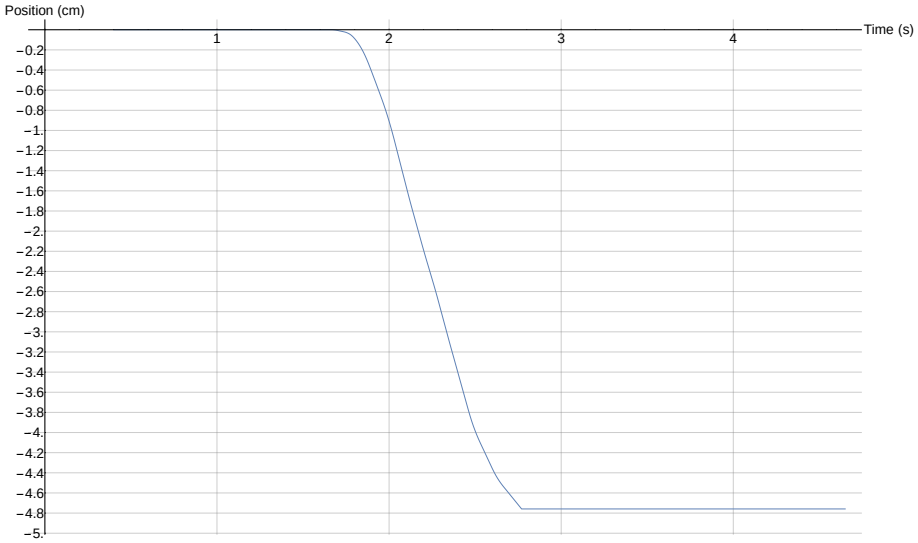


Figura 2.4: *Valore di posizione nel tempo, ottenuto integrando il segnale di Figura 2.3 con l'algoritmo descritto in sezione 2.2*

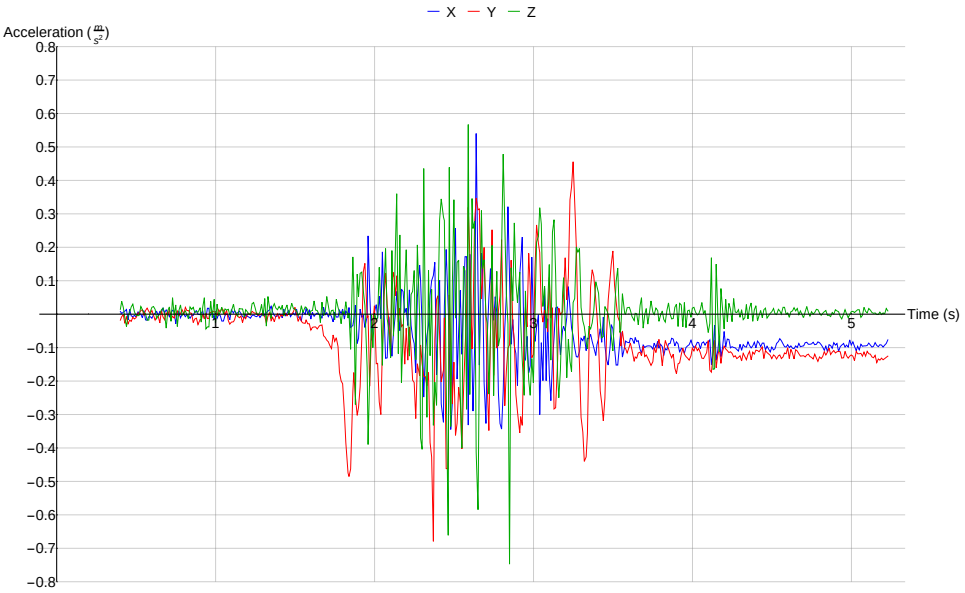


Figura 2.5: *Segnale di accelerazione sui tre assi cartesiani (x blu, y rosso, z verde) per un movimento di 10cm.*

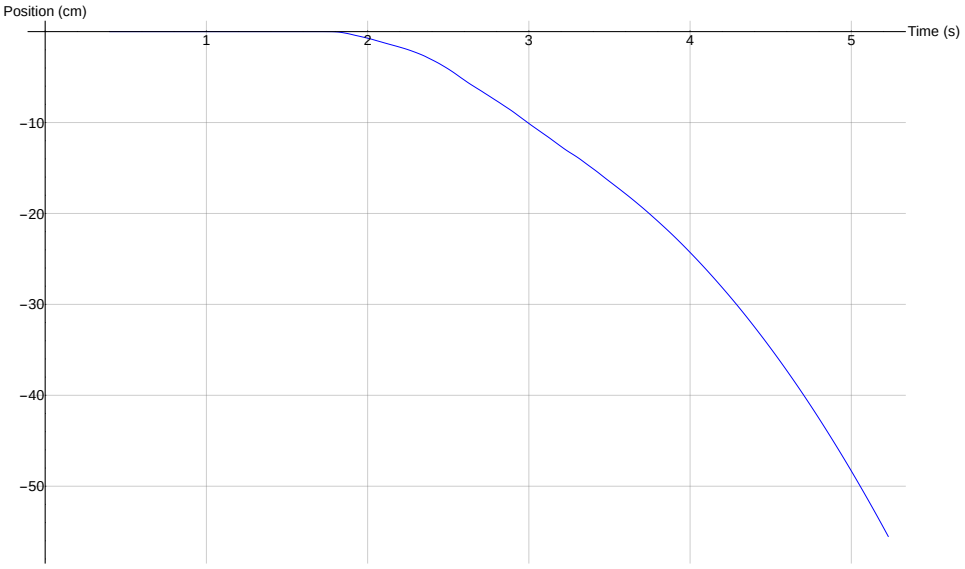


Figura 2.6: *Valore di posizione nel tempo, ottenuto integrando il segnale di Figura 2.5 con l'algoritmo descritto in sezione 2.2*

(e la sua ground truth) generato con un movimento parallelo all'orizzonte, rettilineo lungo l'asse y del sistema di riferimento del sensore (vedi sottosezione 2.1.2) e formato da un singolo spostamento di entità stabilita a priori (generando quindi il moto del dispositivo "fermo, movimento, fermo"). Utilizzando questo tipo di spostamento, si è registrato il segnale dell'accelerometro per alcune distanze ritenute significative (e.g. 2cm e 5cm), ed a partire da queste si sono testate varie elaborazioni allo scopo di ottenere il valore di posizione il più vicino possibile a quello corretto, senza tuttavia ottenere risultati positivi. Si è deciso, quindi, di abbandonare il segnale "linear acceleration" a favore del segnale raw del sensore, in modo da avere maggiore controllo sulle elaborazioni effettuate sul segnale. Il risultato di questo cambiamento è stato incoraggiante e si è riusciti, infine, ad ottenere dei risultati di posizione con errori accettabili da alcune delle misure considerate, utilizzando la catena di elaborazione formata dagli algoritmo riportati in sezione A.1 → sezione A.2 → sezione A.3. Per prima cosa si effettua una calibrazione: sfruttando il vincolo di dispositivo steso orizzontalmente si effettua una media dei primi campioni del segnale per estrarre il vettore di gravità, che può poi essere semplicemente sottratto dai successivi campioni grazie alla particolarità del moto, che avviene su un piano fissato. Successivamente viene applicato un filtro passa-basso del primo ordine per eliminare le variazioni rapide del segnale, che viene infine integrato nell'ultimo step della catena secondo l'algoritmo descritto in sezione A.3: questo è una doppia integrazione semplice a cui sono state apportate due modifiche chiave. Per prima cosa viene applicata una soglia al valore di accelerazione nella coordinata che si desidera integrare: se l'accelerazione non supera la soglia, essa viene posta a zero. Successivamente si tiene memoria e si conta il numero di campioni di accelerazioni nulli consecutivi: se questo supera una seconda soglia, si considera il movimento terminato e quindi si annulla la velocità e si mantiene costante la posizione. Se, invece, il numero di campioni non supera la soglia si effettua la doppia integrazione normalmente. Un ipotesi effettuata da questo algoritmo è che il movimento sia sufficientemente breve per cui la fase di accelerazione iniziale del moto e quella di decelerazione finale siano abbastanza vicine, in modo tale che non vi sia una fase intermedia di accelerazione nulla a velocità costante non nulla, ipotesi che è verificata dai segnali dei movimenti considerati. Un esempio di tale elaborazione viene riportato nelle seguenti

figure: in Figura 2.2 è mostrato il segnale raw fornito dal sensore per lo spostamento di 5cm, mentre in Figura 2.3 è riportato il segnale ottenuto dopo le elaborazioni. Infine, in Figura 2.4 è riportato il valore di posizione nel tempo, ottenuto dall'algoritmo di integrazione.
Questa elaborazione, tuttavia, non ha sempre successo, ed un esempio è il segnale dello spostamento di 10cm: in Figura 2.5 è riportato il segnale ottenuto dopo aver applicato le elaborazioni descritte precedentemente, mentre in Figura 2.6 è mostrato il risultato dell'integrazione utilizzando l'algoritmo presentato. Questo avviene a causa di alcune caratteristiche del segnale che portano l'algoritmo di integrazione a "divergere" e fornire un valore errato di posizione, tuttavia nell'ambito del progetto non si è riusciti a fare chiarezza su tutti i particolari che causano questo fenomeno. Un fatto che si riusciti ad appurare, però, è la necessità di scegliere con molta cura la prima soglia dell'algoritmo di integrazione, responsabile dell'annullamento dell'accelerazione: un valore troppo elevato porta a perdere una porzione significativa di informazione del segnale, causando una posizione restituita non corretta. Dall'altro lato, se si tiene la soglia troppo bassa si ottiene ancora un risultato scorretto, in quanto non si riescono ad eliminare le porzioni del segnale che non contengono informazione utile, portando quindi l'algoritmo ad un risultato errato anche in casi in cui una più accurata scelta della soglia permetterebbe di ottenere un risultato corretto (un esempio di questo è il segnale dello spostamento da 5cm: se la soglia viene abbassata troppo si ottiene un risultato analogo a quello di Figura 2.6).

2.3 Analisi dello spettro del segnale dell'accelerometro

Abbiamo visto, nella sezione precedente, un primo studio effettuato ed alcuni algoritmi che hanno portato ad un risultato positivo nel calcolo dello spostamento in condizioni prossime a quelle ideali. Tuttavia l'elaborazione realizzata prevede un operazione in frequenza, ovvero il filtraggio passa basso, che viene effettuata senza che nessuna analisi in frequenza sia stata svolta per avvalorare la correttezza dell'operazione. Per questo motivo, ed al fine di comprendere più a fondo il segnale, studiamo in questa sezione

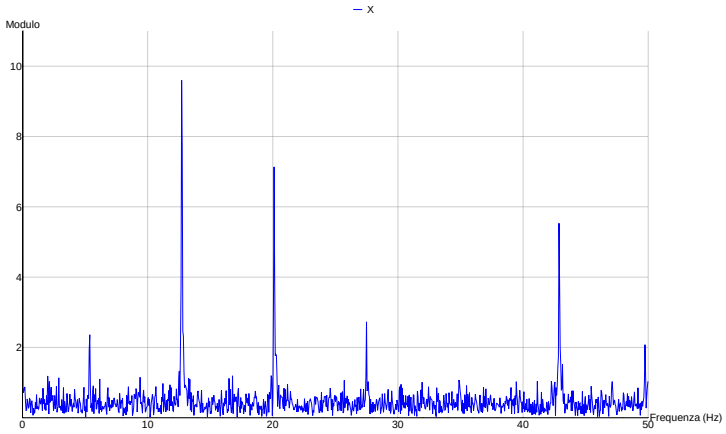
la risposta in frequenza, prima in condizioni statiche, in modo tale da analizzare il rumore di cui il sensore è affetto, e poi in condizioni dinamiche, studiando i segnali delle registrazioni acquisite.

2.3.1 Condizioni statiche

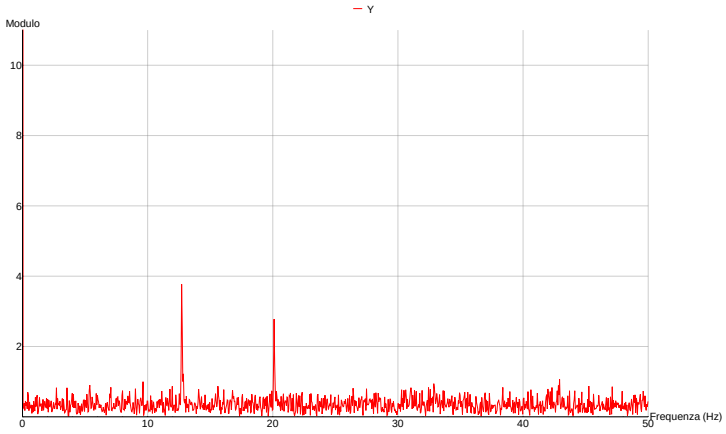
L'analisi descritta in questa sezione è stata effettuata sul segnale fornito dal sensore lasciando il dispositivo immobile, appoggiato su un tavolo orizzontale per una ventina di secondi. Prima di addentrarci nell'analisi, descriviamo la convenzione adottata per il calcolo della risposta in frequenza: come trasformata è stata utilizzata la Discrete Fourier Transform (DFT), in quanto il dato a nostra disposizione è una sequenza finita di campioni, e la definizione adottata è quella riportata in Equazione 2.1. Dati i campioni di una funzione $x(n) = (x_0, \ldots, x_{N-1})$, definiamo la DFT come

$$\mathcal{F}[x(n)] = X(k) = \sum_{n=0}^{N-1} x_n \cdot e^{-2\pi i \, k \frac{n}{N}} \quad \forall \; k = 0, \ldots, N-1 \tag{2.1}$$

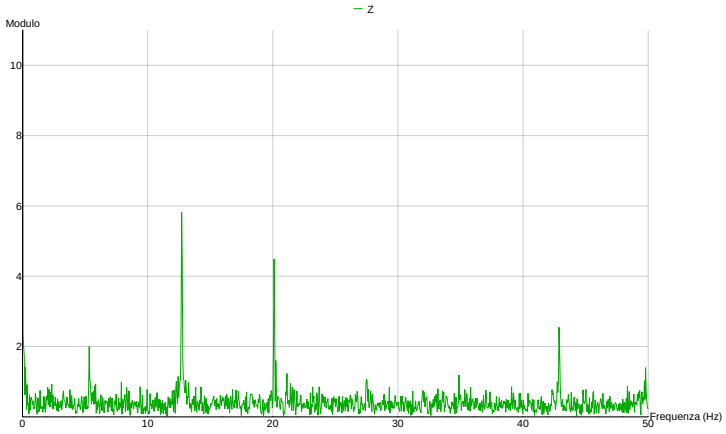
L'algoritmo realizzato per il calcolo della trasformata, invece, è riportato in sezione A.4: per la determinazione dei coefficienti della risposta in frequenza si è utilizzata una funzione built-in del software Wolfram Mathematica, mentre per il calcolo delle frequenze associate ai campioni del segnale trasformato si è utilizzata la proprietà della DFT per cui la differenza di frequenza tra due campioni successivi è pari alla frequenza di campionamento divisa per il numero totale di campioni. La frequenza di campionamento, infine, è stata calcolate semplicemente come inverso del tempo medio di interarrivo tra due sample consecutivi del segnale acquisito (in altre parole la differenza tra gli istanti di arrivo di due campioni consecutivi). Nonostante sia noto che il sistema Android non fornisce i campioni a multipli di un particolare periodo con consistenza, l'assunzione fatta è che questi arrivino con regolarità sufficiente da permettere di supporre la periodicità esatta. Entreremo nel dettaglio di questa ipotesi nella sezione 2.7.
Avendo precisato i dettagli dello strumento utilizzato, vediamo quindi i risultati ottenuti: in Figura 2.7 è riportato il modulo della risposta in frequenza del primo dei due segnali considerati. Per ottenere una vista più dettagliata e significativa si è troncata la parte superiore del grafico, in quanto vi è solamente un picco che raggiunge un'ampiezza maggiore di quella



(a) *Componente x.*



(b) *Componente y.*



(c) *Componente z.*

Figura 2.7: *Modulo della trasformata di Fourier delle tre componenti cartesiane del segnale di accelerazione raw in condizioni di immobilità del sensore (x blu, y rosso, z verde). Il picco presente alla frequenza nulla e le frequenze negative sono state tralasciate per permettere una migliore rappresentazione dei dettagli alle altre frequenze.*

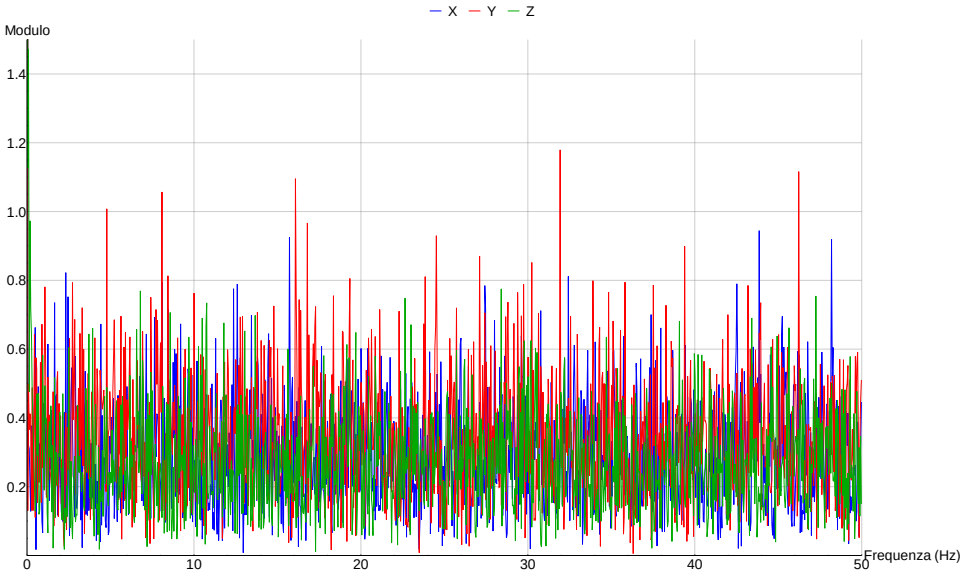


Figura 2.8: *Modulo della trasformata di Fourier delle tre componenti cartesiane di un secondo segnale di accelerazione raw in condizioni di immobilità del sensore (x blu, y rosso, z verde). Il picco presente alla frequenza nulla e le frequenze negative sono state tralasciate per permettere una migliore rappresentazione dei dettagli alle altre frequenze.*

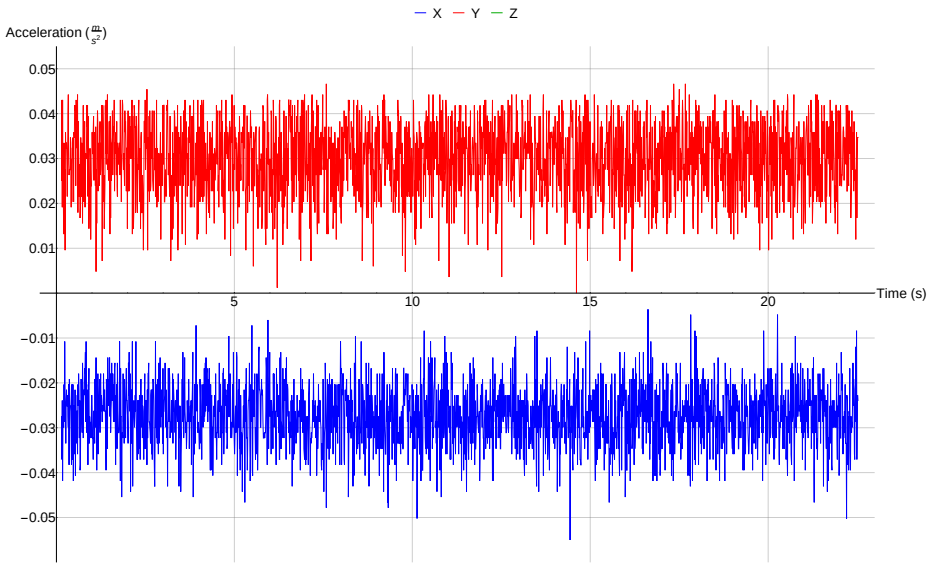


Figura 2.9: *Segnale di accelerazione raw sui tre assi cartesiani per il secondo segnale in condizioni statiche (x blu, y rosso, z verde). La componente z è fuori figura per poter visualizzare i dettagli del segnale: essa presenta una andamento analogo alle altre componenti, eccetto per il valore medio, che è nell'ordine dei* $9.75 \frac{m}{s^2}$.

considerata, che è quello a frequenza nulla, causato dalla componente continua del segnale: questo picco raggiunge un valore di circa 20000 che, se mostrato in scala, renderebbe inosservabili le altre variazioni di entità notevolmente minore. Si è scelto, inoltre, di mostrare le sole frequenze positive, in quanto il modulo è simmetrico rispetto all'asse delle ordinate. Com'è possibile osservare dalla figura, il modulo della risposta ha un'ampiezza analoga per tutte e tre le componenti, con un andamento piatto a tutte le frequenze, ad eccezione di alcuni picchi isolati corrispondenti a sinusoidi di frequenze 5.38 Hz, 12.72 Hz, 20.10 Hz, 27.49 Hz, 42.87 Hz e 49.74 Hz, che non sembrano presentare alcuna relazione tra loro. Per spiegare questi picchi, consideriamo la seconda misura studiata. Questo segnale è stato registrato in un secondo momento rispetto al primo, ma nelle stesse condizioni a meno di un particolare: durante la prima registrazione era presente un computer fisso acceso sullo stesso tavolo su cui il dispositivo era poggiato, mentre durante la seconda registrazione il PC era spento. In Figura 2.8 è riportato il modulo della risposta in frequenza di questo secondo segnale: come per Figura 2.7, anche in questo caso si sono mostrate le sole frequenze positive, ed il grafico è stato troncato, eliminandone la parte superiore formata dal solo picco a frequenza nulla, dovuto alla componente continua causata dalla gravità. Osservando la figura si può notare immediatamente che il modulo ha un andamento piatto a tutte le frequenze, con un'ampiezza analoga a quella del primo segnale, senza, tuttavia, presentare alcun picco (ad eccezione della frequenza nulla). Questo ci porta alla conclusione che le ventole del computer fisso abbiano causato delle vibrazioni nel tavolo nonostante il carico di lavoro scarso del PC, vibrazioni che la sensibilità del sensore gli ha permesso di rilevare e che si sono manifestate nella risposta in frequenza del segnale.
Riportiamo, infine, per completezza il secondo segnale in funzione del tempo: in Figura 2.9 sono visibili le componenti x ed y del secondo segnale considerato. La componente z non è stata mostrata in figura per motivi analoghi a quelli delle figure precedenti: avendo la componente z un valore medio nell'ordine dei $9.75 \frac{m}{s^2}$ la scala risultante nel grafico farebbe sparire tutti i dettagli di interesse del segnale. Quindi, dato che la componente z ha un andamento del tutto analogo a quello delle altre due (a meno del valore medio), si è deciso di riportare solamente queste ultime. Dalla figura è possibile osservare che entrambe le componenti x ed y presentano delle

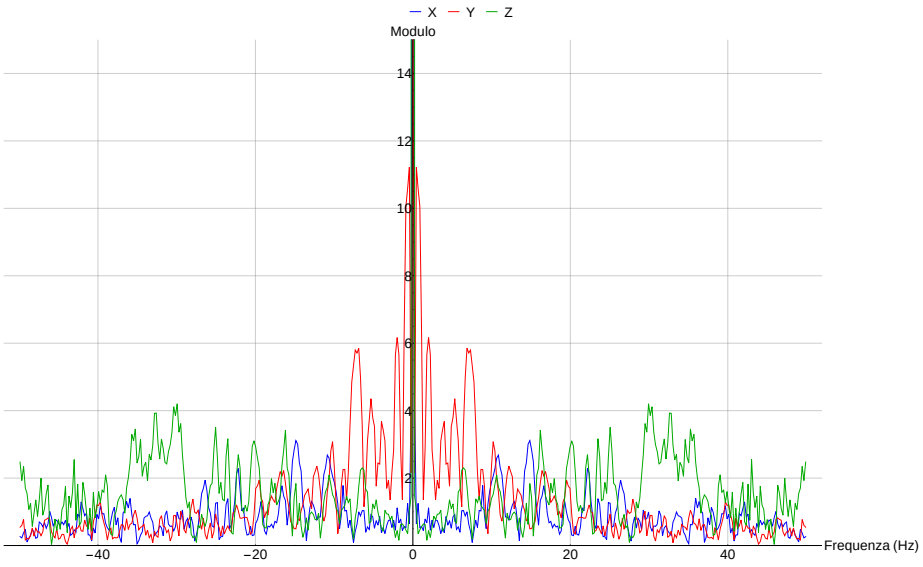


Figura 2.10: *Modulo della risposta in frequenza per il segnale dell'accelerometro, generato dallo spostamento di 5cm descritto in sezione 2.2. Il picco presente alla frequenza nulla (che raggiunge un valore di circa* 4000*) è stato tralasciato per permettere la rappresentazione dei dettagli alle altre frequenze.*

oscillazioni di ampiezza circa $0.035\frac{m}{s^2}$ (ovvero di $3.5\frac{cm}{s^2}$), causate dal rumore che affligge il sensore. Un'altra osservazione interessante è che sia la componente x, che quella y non hanno un valore medio nullo. Una verifica effettuata con una bolla conferma la presenza di una piccola inclinazione del tavolo, che quindi non è perfettamente orizzontale come supposto. Questo porta alla proiezione di una piccola parte del vettore di gravità sugli assi x ed y del sensore, tuttavia la mancanza di strumenti adeguati non ci ha permesso di misurare questa inclinazione. Possiamo effettuare la conclusione qualitativa, quindi, che il sensore è in grado di rilevare anche una tale inclinazione, senza poter sapere, però, il grado di precisione di questa misura.

2.3.2 Condizioni dinamiche

Avendo analizzato il segnale in condizioni statiche, mettendo in evidenza alcune caratteristiche del rumore che affligge il sensore, passiamo ora a studiare il segnale in condizioni dinamiche. In particolare, ci concentreremo sul segnale generato dal sensore durante lo spostamento di 5cm, descritto in sezione 2.2, in quanto i dati di tutti gli spostamenti presenta-

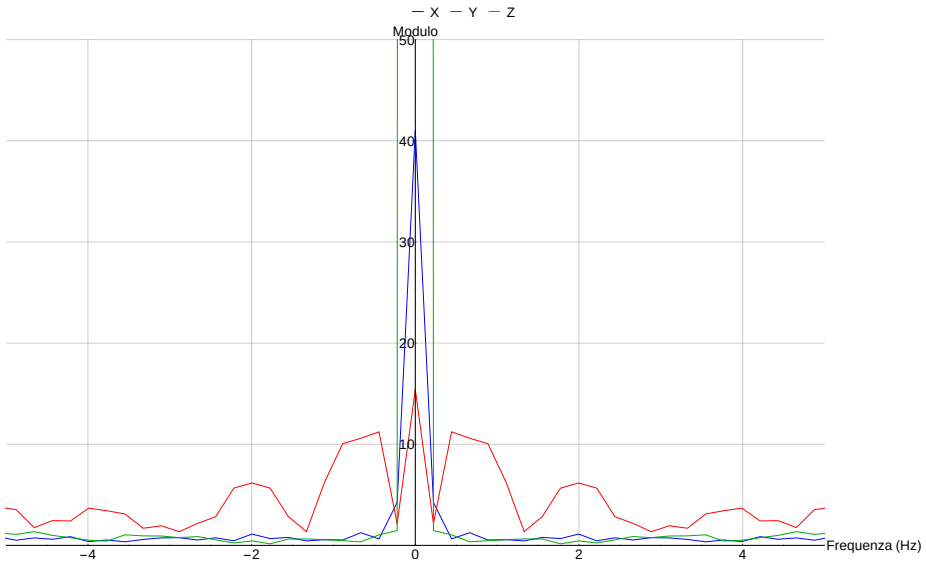


Figura 2.11: *Dettaglio, nell'intorno della frequenza nulla, del modulo della trasformata di Fourier per il segnale generato dallo spostamento di 5cm descritto in sezione 2.2. Il picco della componente z non è stato riportato per non perdere il dettaglio delle altre frequenze.*

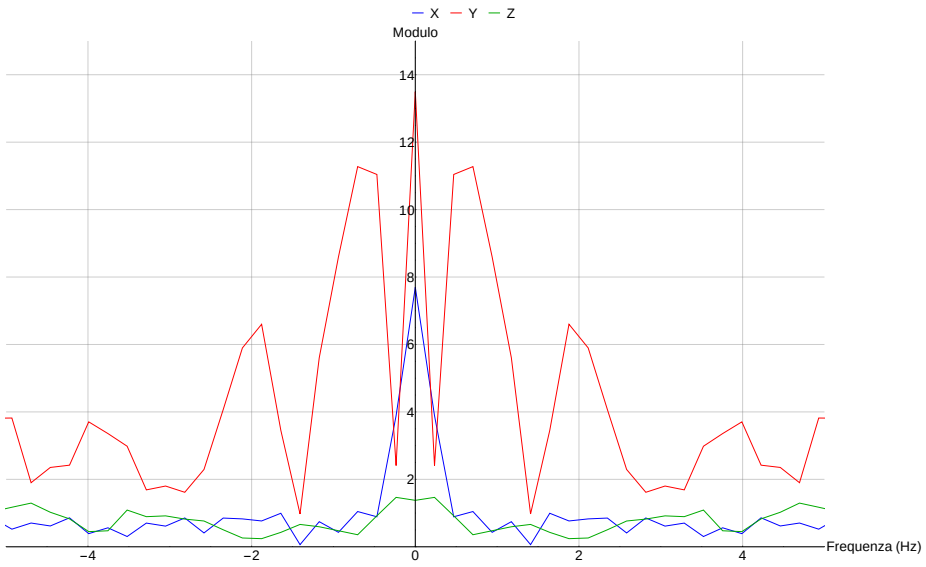


Figura 2.12: *Dettaglio, nell'intorno della frequenza nulla, del modulo della trasformata di Fourier per il segnale generato dallo spostamento di 5cm ed elaborato con l'algoritmo descritto in sezione A.1.*

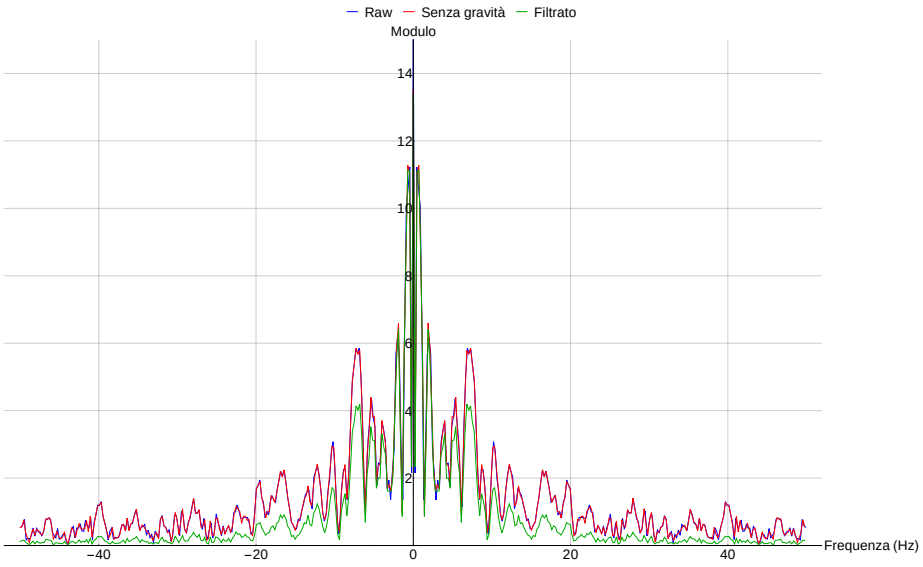


Figura 2.13: *Modulo della trasformata di Fourier per la componente y del segnale raw generato dallo spostamento di 5cm (blu), e per la sua elaborazione con gli algoritmo descritti in sezione A.1 (rosso) e sezione A.2 (verde)*

no caratteristiche analoghe tra loro: non si è, quindi, ritenuto significativo riportare i risultati di tutti, ma si è preferito concentrarsi su un solo caso esemplificativo.

Consideriamo per prima, allora, la Figura 2.10. Essa riporta il modulo della trasformata di Fourier del segnale raw: in blu è riportato il modulo della componente x del segnale, in rosso la componente y ed in verde la z. Come per gli altri casi, il picco alla frequenza nulla (che raggiunge un valore nell'ordine di 4000) è stato mantenuto fuori dalla figura per permettere di osservare con chiarezza i dettagli delle altre frequenze. Dalla figura è notare osservare come tutte le frequenze contribuiscono alla formazione del segnale, nonostante le frequenze più basse abbiano un peso maggiore, in quanto il modulo dello spettro ha un ampiezza più elevata in questa regione. Questo fatto concorda con il risultato empirico ottenuto dall'applicazione del filtro passa basso: le alte frequenze contribuiscono in maniera meno significativa al segnale, quindi un'attenuazione in questa regione produce un'alterazione del segnale minore di quella che si otterrebbe con un'attenuazione delle basse frequenze. Un altro fatto che è possibile notare osservando la figura è che tutte e tre le componenti del segnale presentano un valore medio non nullo: infatti osservando da più vicino la zona attorno

alla frequenza nulla, riportata in Figura 2.11, si può notare chiaramente che tutte e tre le componenti del segnale presentano un modulo non nullo alla frequenza zero, causato dalla presenza di una componente continua su tutti e tre gli assi e non solo su z. Va precisato, tuttavia, che la componente su quest'ultimo asse risulta comunque largamente predominante sulle altre, dato che il modulo della risposta per questo asse raggiunge un valore di 4000, mentre per gli assi x ed y è nell'ordine di qualche decina.
Avendo studiato le caratteristiche del segnale raw, analizziamo ora l'effetto che le elaborazioni effettuate hanno sullo spettro del segnale: studiamo per prima cosa la rimozione della gravità effettuata con l'algoritmo descritto in sezione A.1. Ricordiamo che l'algoritmo prevede l'estrazione del vettore gravità tramite una media dei primi campioni del segnale, ed una sua successiva eliminazione tramite la sottrazione da tutti i campioni successivi del vettore calcolato. Quindi, dato che la sottrazione di un valore costante è il fulcro dell'algoritmo, ci aspettiamo che il frequenza lo spettro rimanga inalterato, ad eccezione della frequenza nulla. In figura Figura 2.13 è possibile osservare il modulo della risposta in frequenza della componente y per il segnale raw (blu) e per il segnale senza gravità (rosso) ottenuto dall'algoritmo: per gli stessi motivi di Figura 2.10, il picco alla frequenza nulla non è stato mostrato, mentre le altre componenti (x e z) dei segnali sono state omesse per permettere una rappresentazione più chiara, grazie al fatto che l'analisi di una singola componente permette la caratterizzazione di tutte e tre. Come previsto, dalla figura si può notare che il modulo della trasformata del segnale a cui è stata rimossa la gravità segue fedelmente quello del segnale raw, tanto da rendere quasi indistinguibili le due curve, eccetto che alla frequenza nulla dove si può osservare la loro più rilevante differenza. In figura Figura 2.12 è riportato il dettaglio, nell'intorno della frequenza nulla, del modulo della trasformata del segnale senza gravità: come è possibile osservare confrontandolo con figura Figura 2.11, il modulo alla frequenza zero di tutte e tre le componenti del segnale è stato notevolmente attenuato. Notiamo, tuttavia, che una parte di componente continua è rimasta su tutti gli assi, indicatore che il valore medio del segnale è variato nel tempo, invece che rimanere costante al valore calcolato tramite i primi campioni. Un'analisi più approfondita di questo aspetto è affrontata nella sezione 2.8.
Consideriamo, infine, l'ultima elaborazione effettuata, ovvero il filtraggio

passa basso: in Figura 2.13 è possibile osservare il modulo della trasformata della componente y del segnale filtrato (colorato di verde), oltre a quello degli altri due segnali già citati. Il segnale mostrato in figura è stato ottenuto applicando un filtro passa basso secondo l'algoritmo descritto nella sezione A.2, con una costante RC pari a 0.018, che corrisponde ad una frequenza di taglio di 8.84194 Hz. Questo trova riscontro da quanto è possibile notare osservando il grafico: intorno ai 5 Hz il segnale filtrato inizia a discostarsi dal segnala raw, e poco prima dei 10 Hz il modulo raggiunge un'ampiezza pari a circa la metà dell'ampiezza del modulo del segnale raw. Notiamo, infine, che alle alte frequenze anche se il modulo è attenuato, la sua forma originale è in linea di massima preservata, invece che decrescere rapidamente a zero: il filtro applicato, infatti, è un semplice filtro RC del primo ordine, quindi l'attenuazione che esso imprime è modesta.

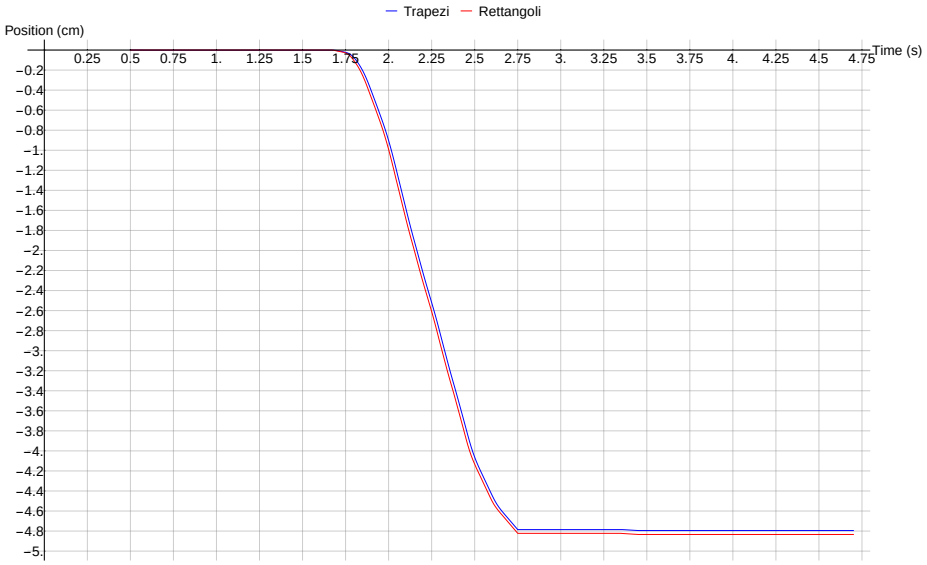
2.4 Integrazione del segnale

Abbiamo visto nelle sezioni precedenti l'algoritmo di integrazione del segnale utilizzato per ottenere la posizione, dopo una pre-elaborazione del segnale raw. In questa sezione studieremo il risultato del rilassamento di alcune ipotesi effettuate dall'algoritmo. Ricordiamo brevemente il funzionamento dell'algoritmo (riportato in sezione sezione A.3): gli input sono i campioni di accelerazione con i relativi timestamp, la coordinata su cui si vuole effettuare l'integrazione, e due soglie. L'algoritmo considera i campioni in ordine temporale ed effettua una sogliatura dell'accelerazione sulla coordinata desiderata, utilizzando la prima soglia di input. Successivamente il contatore delle accelerazioni nulle consecutive viene aggiornato: se il campione attuale è nullo il contatore viene incrementato, altrimenti viene azzerato. Dopodiché, se questo contatore supera la seconda soglia di input, si considera il movimento terminato e si annulla la velocità, mentre in caso contrario si effettua una doppia integrazione, utilizzando il metodo dei rettangoli: l'area tra due sample consecutivi viene calcolata tramite il rettangolo di base pari alla distanza temporale, ed altezza pari al valore del secondo sample.
Due scelte, quindi, sono state compiute in questo algoritmo: la prima è di concentrarsi su un singolo asse di interesse e scartare le informazioni delle altre componenti, mentre la seconda scelta è di utilizzare il metodo dei

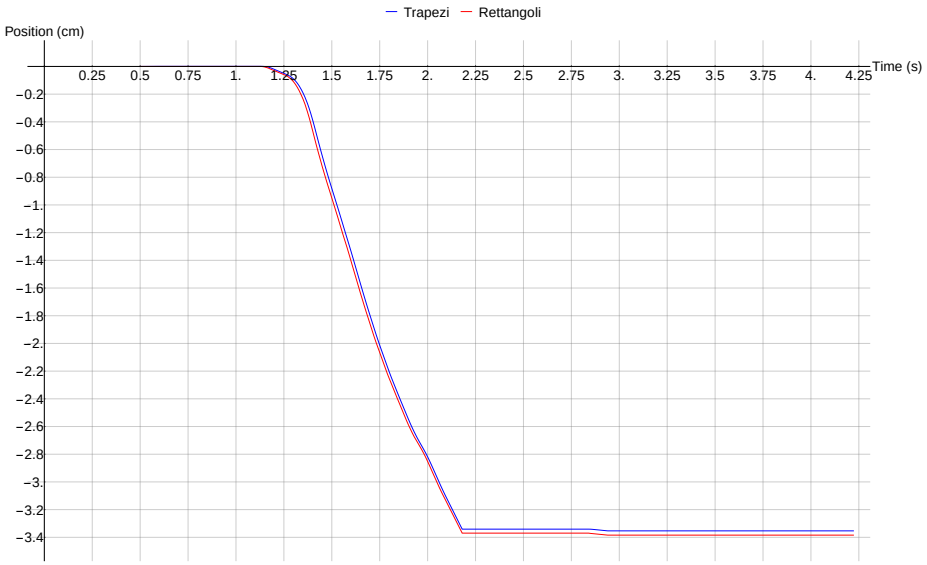
rettangoli per l'integrazione, supponendo che questo fornisca una buona approssimazione dell'area reale. Nelle prossime sezioni rilasseremo queste ipotesi, per verificare fino a che punto esse sono valide e per esplorare condizioni più generali.

2.4.1 Integrazione per trapezi

La prima ipotesi che rilassiamo è quella effettuata dal metodo dei rettangoli, che approssima l'area compresa tra due campioni consecutivi tramite appunto un rettangolo. Dato che questo metodo approssima il segnale reale tra due campioni consecutivi con il valore del secondo, a tutti gli effetti esso suppone che il segnale subisca una variazione a gradino nel passaggio da un sample al successivo, presentando quindi un andamento discontinuo. Nella realtà, tuttavia, ci aspettiamo che il segnale segua un andamento molto più regolare, e che si porti dal valore di un sample a quello del successivo in maniera più graduale, essendo il processo fisico continuo. Per questo motivo si è deciso di utilizzare il metodo dei trapezi al posto di quello dei rettangoli: esso assume che tra due sample consecutivi il segnale abbia un andamento lineare, formando quindi la figura di un trapezio se si tracciano le linee di ampiezza dei due sample consecutivi, e si congiungono con un segmento lineare. Questo metodo, quindi, assume un comportamento più regolare del segnale, che si presume sia maggiormente aderente alla realtà fisica. L'algoritmo che implementa l'integrazione con questo metodo è molto simile a quello che realizza l'integrazione per rettangoli, ed è riportato in sezione A.5: anche in questo caso vengono effettuate le operazioni di sogliatura dell'accelerazione e conteggio delle accelerazioni nulle, ma al momento di effettuare la doppia integrazione si calcola l'area utilizzando il valore del sample di accelerazione attuale e del precedente, di cui si è tenuta traccia, insieme alla velocità attuale e precedente secondo le formule $vel = vel + \frac{1}{2}\ dt\ (acc + lastAcc)$ e $pos = pos + \frac{1}{2}\ dt\ (vel + lastVel)$, dove acc e $lastAcc$ sono i campioni, rispettivamente attuale e precedente, di accelerazione sogliati, vel e pos i rispettivi valore di velocità e posizione e dt è l'intervallo temporale che separa i due campioni.
In Figura 2.14 riportiamo due casi significativi del risultato ottenuto dall'applicazione dell'algoritmo: in entrambe le figure sono riportati i grafici della posizione, espressa in centimetri, ottenuti tramite il metodo dei rettangoli (rosso) e dei trapezi (blu). In tutti e due i casi si è integrata l'acce-



(a) *Integrazione del segnale dello spostamento di 5cm.*



(b) *Integrazione del segnale dello spostamento di 3cm.*

Figura 2.14: *Confronto della posizione (in centimetri) ottenuta utilizzando il metodo dei rettangoli (rosso) e dei trapezi (blu).*

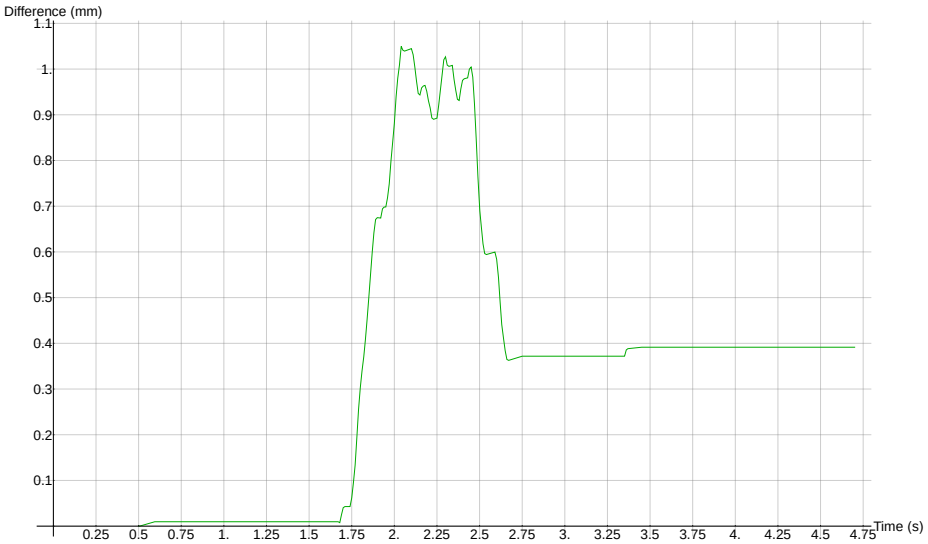


Figura 2.15: *Differenza (in millimetri) tra il valore di posizione ottenuto con il metodo dei rettangoli e quello dei trapezi, per lo spostamento di 5cm.*

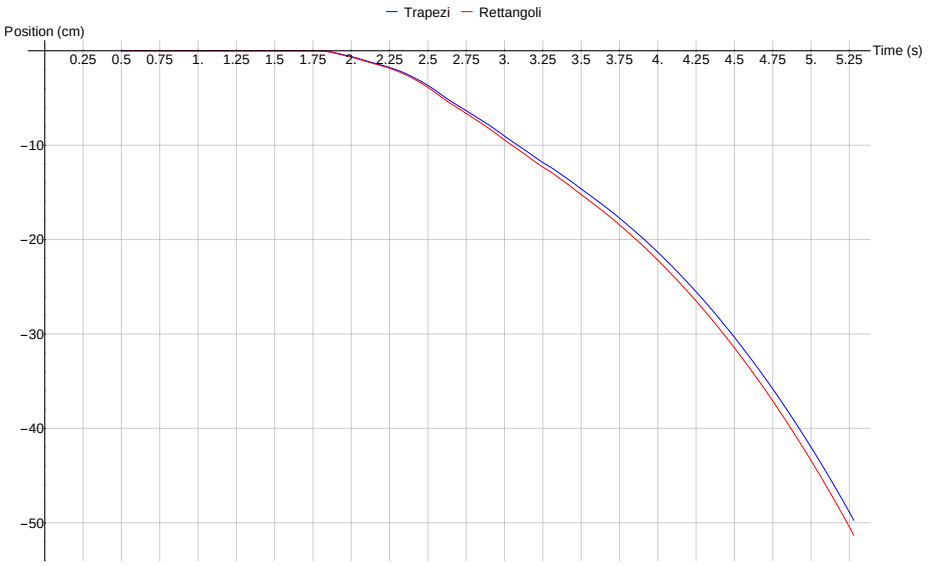


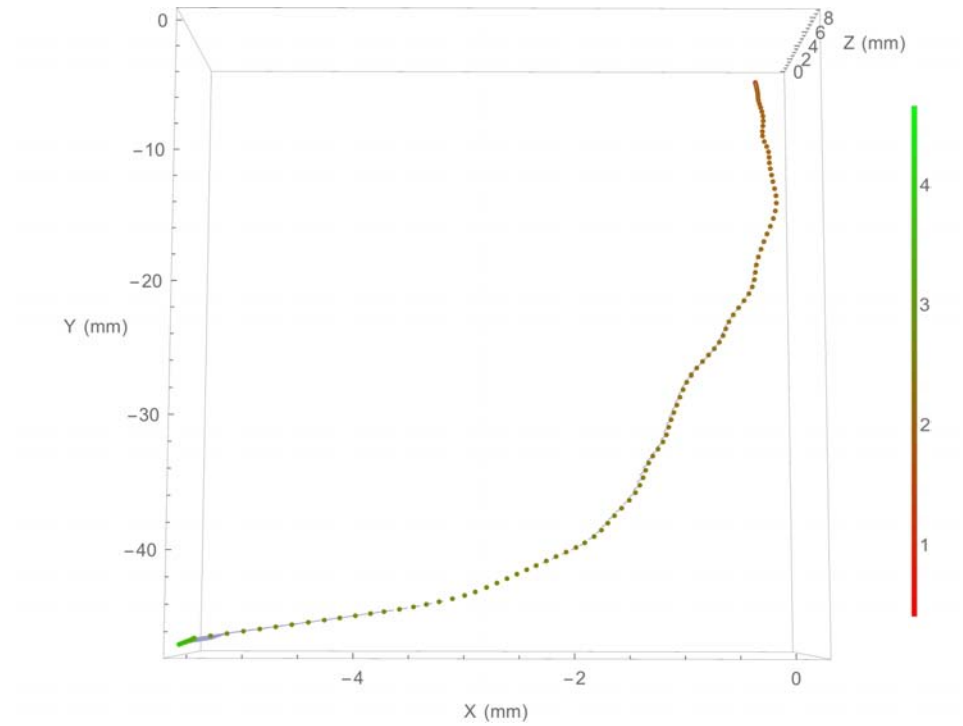
Figura 2.16: *Confronto della posizione (in centimetri) ottenuta utilizzando il metodo dei rettangoli (rosso) e dei trapezi (blu) per lo spostamento di 10cm.*

lerazione lungo la coordinata y, in quanto è quella su cui si è svolto il movimento, ma in Figura 2.14a è stato integrato il segnale dello spostamento di 5cm, mentre in Figura 2.14b quello di 3cm. Com'è possibile osservare da entrambe le figure, i valori ottenuti con i due metodi sono molto vicini tra loro, ma lo spostamento calcolato con il metodo dei trapezi è sempre leggermente inferiore a quello ottenuto col metodo dei rettangoli. Questo è dovuto alle caratteristiche degli algoritmi, in congiunzione alla particolarità del moto che si sta integrando: infatti l'andamento che la velocità segue si può dividere, qualitativamente, in due fasi successive, una in cui il modulo della velocità cresce, ed una seconda in cui il modulo decresce fino all'arresto del movimento. Per quanto riguarda gli algoritmi, invece, l'elemento chiave che causa questo fenomeno è la diversa ipotesi fatta nell'approssimare il segnale reale: infatti il metodo dei rettangoli lo approssima in ogni istante tra due sample consecutivi con il valore assunto dal secondo campione, ovvero il valore finale che il segnale raggiunge nell'intervallo, mentre il metodo dei trapezi approssima il segnale reale con la combinazione lineare dei due campioni. Congiuntamente questi due fatti causano una differenza sistematica tra i due metodi: infatti nella regione in cui il modulo della velocità è crescente, dati due campioni consecutivi, il secondo avrà sempre un modulo maggiore, quindi il metodo dei rettangoli approssimerà il segnale reale nell'intervallo con il suo valore massimo e questo risulterà sempre maggiore dell'approssimazione effettuata dal metodo dei trapezi, che suppone una combinazione lineare tra i due campioni. Quando si passa alla regione decrescente, invece, accade l'opposto, quindi l'approssimazione effettuata dal metodo dei rettangoli risulta sempre minore di quella calcolata dal metodo dei trapezi. Dato che l'integrazione prevede l'accumulo progressivo di tutte queste piccole aree tra i campioni, quello che accade è che nella prima regione crescente il metodo dei rettangoli accumula un'area maggiore rispetto al metodo dei trapezi e quindi la differenza tra le posizioni calcolate dai due metodi aumenta, fino al raggiungimento della seconda regione, dove è il metodo dei trapezi ad accumulare un'area maggiore rispetto al metodo dei rettangoli, causando una progressiva diminuzione della differenza tra le posizioni calcolate. Questo si può osservare anche dalla Figura 2.15, dove viene riportato il valore assoluto della differenza (espressa in millimetri) tra le posizioni ottenute con i due metodi: come atteso, la differenza inizialmente cresce fino al raggiun-

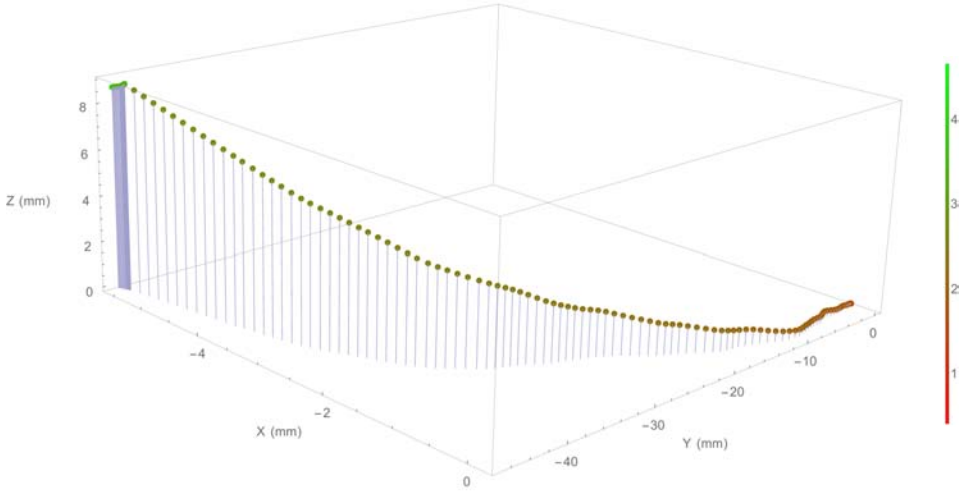
gimento della seconda metà del moto, dove inizia a calare. Dalla figura è possibile notare, inoltre, che l'entità di questa differenza è esigua: essa infatti raggiunge il millimetro al picco del movimento e si assesta, alla fine del movimento, sul mezzo millimetro di differenza per uno spostamento complessivo di 5cm (ovvero 1% del valore finale).
Infine, osserviamo che anche se si volesse sfruttare questa differenza per ottenere una stima, seppur di poco, migliore, nessuno dei due metodi fornirebbe un vantaggio rispetto all'altro. Mostriamo, infatti, in Figura 2.14 un controesempio: in Figura 2.14a entrambi gli algoritmi sottostimano la posizione reale, quindi il metodo dei rettangoli è quello che si avvicina di più al valore corretto. Nel caso riportato in Figura 2.14b, invece, gli algoritmi sovrastimano la posizione, quindi è il metodo dei trapezi a fornire il valore che si avvicina maggiormente a quello corretto.
Le considerazioni effettuate fin'ora, tuttavia, si applicano solo parzialmente al caso in cui la soglia scelta causi il fallimento dell'algoritmo di integrazione. In tutti i casi in cui questo succede, il risultato che si ottiene è analogo a quello mostrato in Figura 2.16, dove è riportato il risultato dell'integrazione dello spostamento di 10cm con i due metodi: quello che è possibile notare dalla figura è che la differenza tra la posizione stimata dai due metodi cresce nel tempo, e che in entrambi i casi l'andamento della posizione è correttamente parabolico, trattandosi di una doppia integrazione di una componente continua residua. Quest'ultimo fatto è in accordo con quanto si può osservare dal grafico dell'accelerazione integrata (visibile in Figura 2.5): come si può notare, nella parte finale del grafico l'accelerazione non si annulla, ma rimane una componente continua sopra la soglia di integrazione, e quando questa componente viene integrata il grafico della posizione acquisisce la forma parabolica osservata. L'incremento della differenza tra la stima delle posizioni con i due metodi, invece, è un risultato collegato alla precedente osservazione: essendo il segnale costante, osserviamo che la seconda integrazione viene effettuata su una funzione rampa, crescente in modulo, e ricordiamo che per questo tipo di funzioni il metodo dei rettangoli approssima il segnale reale con un area maggiore rispetto al metodo dei trapezi. Per questi motivi, al passare del tempo la posizione (ovvero l'area) stimata col metodo dei rettangoli si discosta sempre di più da quella calcolata col metodo dei trapezi.

2.4.2 Integrazione vettoriale

Avendo analizzato la scelta di usare il metodo dei rettangoli per l'integrazione, generalizziamo ora la decisione di concentrarsi su un singolo asse, scartando le informazioni degli altri: quello che faremo in questa sezione sarà integrare il segnale di accelerazione come valore vettoriale e non più come una funzione scalare. Osserviamo per prima cosa che il segnale a nostra disposizione è una sequenza di vettori tridimensionali a cui è associato un timestamp: in altre parole, dal punto di vista matematico, il segnale è una curva, ovvero una funzione $a(t) : \mathcal{R} \rightarrow \mathcal{R}^3$, che mappa ogni istante temporale nel suo corrispondente vettore di accelerazione. Dalla teoria sappiamo, inoltre, che l'integrale di una curva è definito come $\int a(t)dt = \left(\int a_x(t)dt,\ \int a_y(t)dt,\ \int a_z(t)dt\right)$, dove $a_x(t),\ a_y(t),\ a_z(t)$ sono le componenti della curva e sono funzioni reali di variabile reale. Quindi per integrare una curva è necessario integrare separatamente tutte le sue componenti: tuttavia, dato che il segnale a nostra disposizione è una curva campionata, per calcolarne l'integrale adatteremo uno degli algoritmi di calcolo numerico visti precedentemente per poter lavorare con valori vettoriali. In particolare, in sezione A.6 è riportato l'algoritmo di integrazione vettoriale realizzato: al contrario degli algoritmi precedenti, esso permette di scegliere il metodo di approssimazione da utilizzare (rettangoli o trapezi) tramite un parametro di ingresso. A parte questa differenza, la logica seguita dall'algoritmo è analoga a quella degli algoritmi precedenti: i campioni del segnale vengono letti in ordine temporale ed una soglia viene applicata al modulo del vettore accelerazione: se la norma del vettore non supera la soglia, il campione viene azzerato. Successivamente si aggiorna il contatore di vettori nulli consecutivi, analogamente agli algoritmi precedenti, per poi utilizzare questo contatore per rilevare la fine del moto: se il contatore è al di sotto della seconda soglia viene effettuata l'integrazione con il metodo indicato, in caso contrario la velocità viene azzerata. Per quanto riguarda l'integrazione vera e propria, le operazioni effettuate sono le stesse dei precedenti algoritmi, solo che ora gli argomenti sono dei vettori: per questo tipo di variabili le operazioni vengono eseguite separatamente, coordinata per coordinata, realizzando, quindi, a tutti gli effetti tre integrazioni, una per ogni componente del segnale.
In Figura 2.17 è riportato il risultato dell'integrazione, effettuata con l'algoritmo descritto, del segnale dello spostamento di 5cm, adottato come



(a) *Vista dall'alto.*



(b) *Vista laterale.*

Figura 2.17: *Posizione (in millimetri) nello spazio tridimensionale ottenuto integrando l'accelerazione dello spostamento di 5cm con l'algoritmo descritto in sezione A.6. A lato del grafico è riportato il colore utilizzato per associare la coordinata temporale ai campioni.*

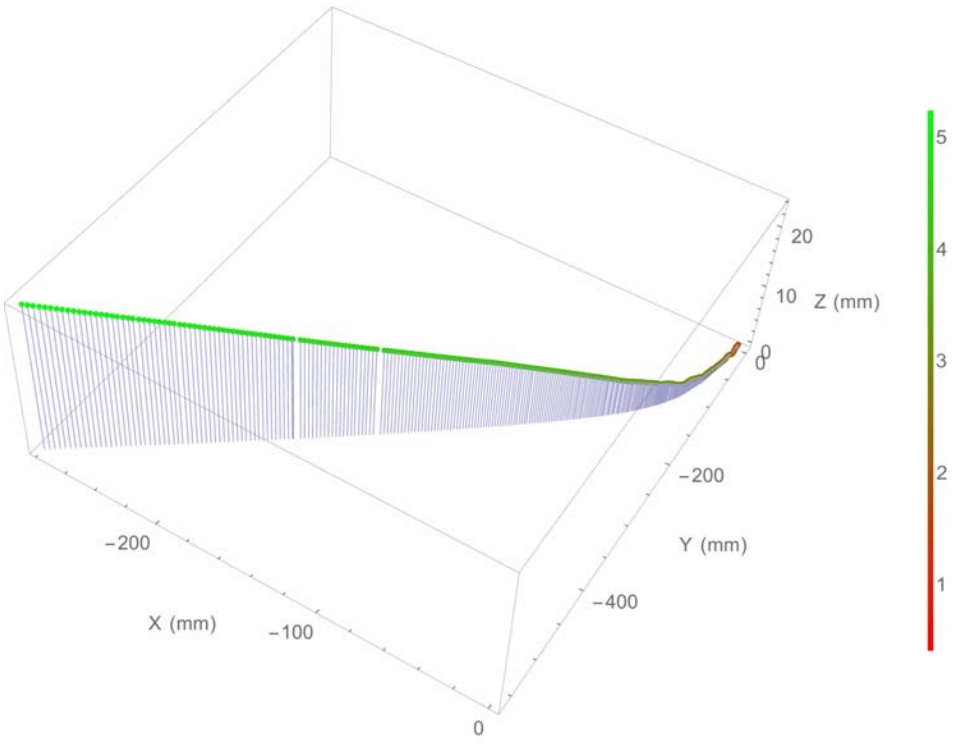


Figura 2.18: *Posizione (in millimetri) nello spazio tridimensionale ottenuto integrando l'accelerazione dello spostamento di 10cm. A lato del grafico è riportato il colore utilizzato per associare la coordinata temporale ai campioni.*

esemplificativo, in quanto gli altri presentano un comportamento analogo. Fissato lo zero come condizione iniziale dell'integrazione, ogni asse rappresenta lo spostamento, in millimetri, lungo l'asse corrispondente nel mondo reale a partire all'origine, per cui ogni punto del grafico corrisponde alla collocazione nello spazio, associata al tempo in cui ci si trovava nella posizione. Quest'ultima dimensione è stata rappresentata tramite il colore associato ai punti del grafico: la posizione al tempo zero è contrassegnata dal colore rosso, l'ultima posizione raggiunta è rappresentata con il verde, mentre tutte le posizioni intermedie hanno un colore composto da una combinazione di rosso e verde, pesata secondo il tempo associato al campione, come è possibile osservare dalla legenda riportata a margine del grafico, in cui è visibile il colore associato ad ogni istante temporale. Possiamo pensare, quindi, al grafico rappresentato come un grafico quadridimensionale che visualizza la posizione tridimensionale nel tempo (che generano quindi quattro gradi di libertà), ed in cui gli assi mostrati rappresentano

le coordinate spaziali, mentre la dimensione temporale è visualizzata mediante il colore della curva. Notiamo dalla Figura 2.17, che, come ci aspettavamo, il moto avviene principalmente lungo la direzione y: gli assi infatti non sono in scala, e su circa 47 millimetri di spostamento lungo l'asse y del movimento, solo 5/6 millimetri sono avvenuti lungo l'asse x e 8/9 lungo z. Su questi assi, tuttavia, non ci dovrebbe essere alcun movimento, ad eccezione di qualche eventuale piccola oscillazione dovuta alla non perfetta idealità del moto: appare evidente, invece, una sorta di "deriva" lungo gli assi x e z. Questa è causata da una componente continua presente nelle componenti del segnale: come risultato dell'integrazione queste costanti diventano un polinomio di secondo grado, mentre, approssimando qualitativamente l'accelerazione con una funzione lineare a tratti, risulta che su y il prodotto dell'integrazione è un polinomio di terzo grado. Questa disparità di grado tra i diversi assi nel risultato dell'integrazione causa la curvatura visibile: ricordiamo, infatti, che in figura è mostrato il supporto della curva (ovvero della funzione matematica), quindi se le funzioni che compongono la curva stessa fossero un polinomio di pari grado, quello che si potrebbe vedere sarebbe una linea retta, in quanto l'incremento su ogni asse sarebbe uguale a quello degli altri. Dato, invece, che il valore negli assi cresce con ritmi differenti in ogni coordinata, si ottiene il risultato visibile in figura, per cui il grafico "curva". Dal colore (e dal corrispondente tempo) dei campioni in cui avviene il movimento, notiamo, inoltre, che la deriva rilevata avviene nella parte centrale del segnale, ovvero nel pieno dello spostamento. Da questo possiamo concludere che, oltre alla componente continua presente alla fine del movimento, osservata dal grafico dell'accelerazione di Figura 2.3, una componente continua è presente sugli assi durante tutta la durata del movimento.
Come per l'algoritmo di integrazione scalare, tuttavia, anche per questo algoritmo ci sono casi in cui esso fallisce e restituisce un valore di posizione privo di significato: in particolare, questi casi coincidono con quelli in cui anche l'algoritmo scalare falliva. A titolo di esempio riportiamo in Figura 2.18 la posizione ottenuta integrando, con l'algoritmo presentato, il segnale dello spostamento di 10cm. Anche in questo caso sugli assi è riportata la posizione in millimetri rispetto all'origine, ed il tempo è rappresentato dal colore dei punti, codificato secondo la legenda riportata a fianco del grafico. Com'è possibile osservare, quando il grafico raggiunge

la posizione di circa 10cm lungo l'asse y (ovvero alla fine del movimento), la posizione inizia a "divergere" secondo una linea retta. Ricordando, come detto precedentemente, che la curva assume la forma di una retta nel caso in cui le componenti sugli assi siano un polinomio di pari grado, concludiamo che al termine del movimento le componenti continue presenti sugli assi formano un vettore la cui norma supera la soglia: questo vettore non viene azzerato, ed al contrario viene integrato, portando ad un valore di posizione privo di significato.

2.5 Rimozione della componente continua

Dalla sezione 2.4 sull'integrazione del segnale è emersa la presenza di componenti continue al termine del movimento, che disturbano l'integrazione e costringono a mantenere la soglia di integrazione alta, in modo che questa componente venga considerata rumore ed azzerata. Quello che cercheremo di fare in questa sezione è operare in frequenza per azzerare lo spetto alla sola frequenza nulla, in modo tale da rimuovere la componente continua. Per effettuare questa operazione solitamente viene utilizzato un filtro notch (o elimina banda): questi filtri, però, non hanno fase lineare, e questo significa che la forma del segnale non viene preservata. Per la nostra applicazione, però, questo è un effetto non desiderato e che non possiamo permetterci: infatti perdendo la forma del segnale non è più possibile ottenere il valore di posizione, che è legato all'area della curva. Utilizzeremo, allora, un filtro passa alto: idealmente, esso dovrà avere banda di transizione nulla, frequenza di taglio il più vicino possibile allo zero, attenuazione nulla in pass-band (ovvero banda passante) ed infinita in stop-band (ovvero al di fuori della banda passante). Al momento della realizzazione, tuttavia, ci saranno tutti gli effetti di non idealità, per cui ci saranno vincoli sulla banda di transizione, sull'attenuazione e sulla frequenza di taglio. Il filtro che progetteremo sarà di tipo FIR (Finite Impulse Response, ovvero filtro a risposta impulsiva finita), in quanto essi presentano delle caratteristiche volute, come la stabilità e possibilità di progettare il filtro con fase lineare (al contrario degli IIR, per cui la fase lineare è ottenibile solo come approssimazione).
Per quanto riguarda gli strumenti adottati per la progettazione: dato che gli algoritmi forniti da Mathematica non permettono molta libertà nella

progettazione, si è deciso di utilizzare lo "strumento di progettazione ed analisi filtri" messo a disposizione da Matlab. Tramite questo toolbox sono stati analizzati tre metodi base per la progettazione di un filtro FIR, ovvero equiripple, least-squares e metodo delle finestre.

- Equiripple è un algoritmo che, a partire dalle specifiche di progetto, permette di trovare il filtro ottimo in norma di Chebyshev o minimax: in altre parole l'algoritmo cerca di minimizzare l'errore massimo del filtro rispetto al filtro ideale. In particolare, Matlab mette a disposizione due varianti di questo algoritmo: una permette di trovare automaticamente il filtro di ordine minimo, mentre la seconda variante permette di fissare manualmente l'ordine del filtro.

- Il metodo least-squares fornisce ancora un filtro ottimo, ma l'errore che questo metodo minimizza è l'errore quadratico medio del filtro rispetto a quello ideale corrispondente alle specifiche. Al contrario del precedente, questo metodo impone di specificare l'ordine del filtro e non permette la ricerca automatica del filtro di ordine minimo che soddisfi le specifiche.

- Il metodo delle finestre, infine, è ancora un metodo least-squares, ovvero minimizza l'errore quadratico medio del filtro, e si basa sul troncamento della risposta in frequenza del filtro ideale tramite una funzione di finestratura, da cui il nome del metodo. Nella nostra progettazione, la funzione che verrà considerata è la "kaiser", in quanto per questo tipo di finestra, come per equiripple, Matlab permette di scegliere se fissare manualmente l'ordine del filtro o determinare con l'algoritmo il filtro di ordine minimo.

Una volta realizzato il filtro, si è copiato il suo kernel in Mathematica e lo si è applicato utilizzando l'algoritmo descritto in sezione A.7, che semplicemente carica il kernel del filtro e lo applica tramite una convoluzione alle singole componenti del segnale di input, restituendo, poi, il risultato in uscita.
Prima di entrare nei dettagli della progettazione del filtro con i vari metodi utilizzati, riportiamo le caratteristiche del segnale significative per la progettazione, ed il valore desiderato per i parametri del filtro: dalla Figura 2.20 possiamo osservare che il segnale da filtrare presenta una componente utile già alle bassissime frequenze. In particolare già alle frequenze

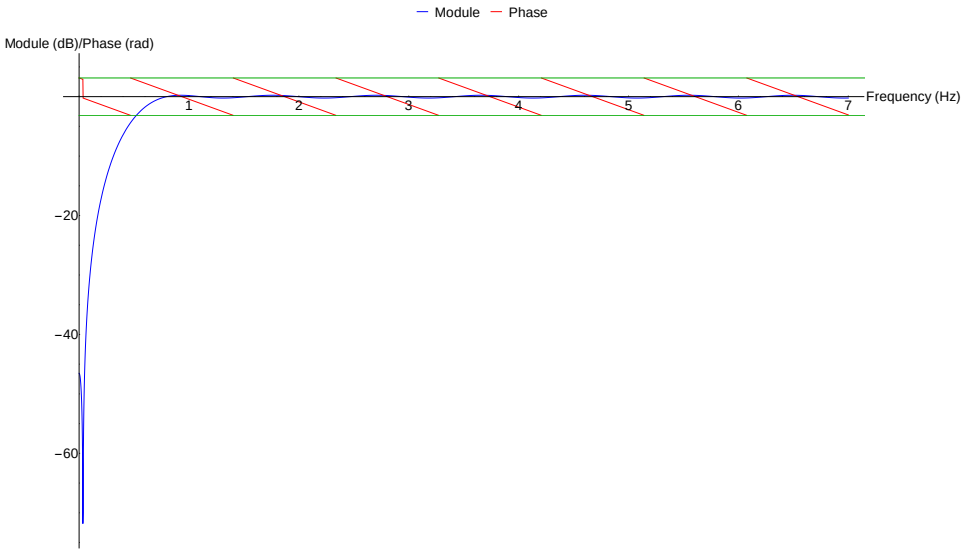
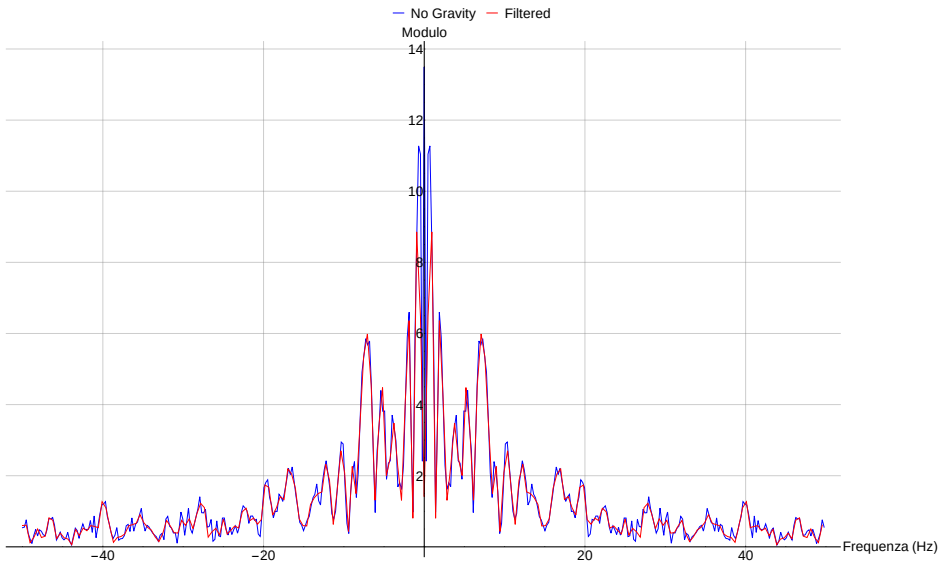
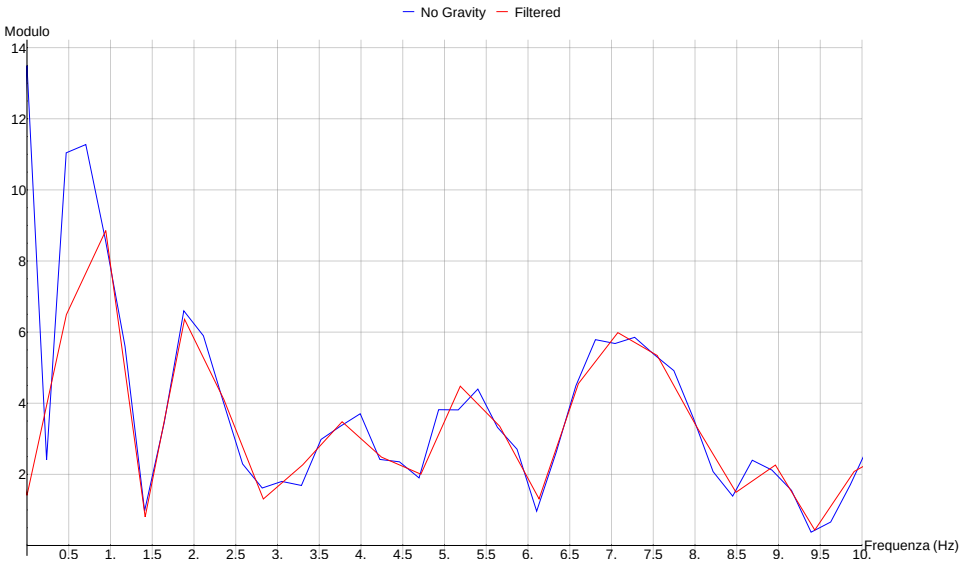


Figura 2.19: *Modulo in decibel (blu) e fase in radianti (rosso) della risposta in frequenza del filtro progettato con l'algoritmo equiripple e la specifica di trovare il filtro di ordine minimo. In verde sono riportati i valori costanti di* $\pm\pi$.

0.5-1 Hz il segnale presenta componenti significative, che rappresentano alcuni dei picchi del modulo dello spettro. La banda di transizione del filtro, quindi, dovrà iniziare il più possibile vicino allo zero ed avere ampiezza minima, in modo tale da non perdere informazione del segnale. In stopband, invece, cercheremo di ottenere la massima attenuazione possibile, con l'obiettivo di raggiungere un'attenuazione di almeno 80 dB. Per quanto riguarda l'ordine del filtro, infine, esso è strettamente legato al ritardo introdotto, dato che un filtro a fase lineare di ordine n introduce un ritardo pari a $n/2$ campioni, quindi cercheremo il filtro di ordine minimo. In particolare, il segnale di esempio considerato (come anche gli altri segnali in linea di massima) incomincia ad avere informazione di movimento utile dopo circa 1.5 secondi, quindi il massimo ordine del filtro che ci permette di non perdere informazione utile è nell'ordine del 200, considerando che anche la rimozione della gravità effettuata introduce ritardo. Procediamo, quindi, a descrivere la progettazione del filtro effettuata. Il primo metodo che consideriamo è la variante di equiripple che permette di trovare il filtro di ordine minimo: in questo caso, i parametri dell'algoritmo che possiamo controllare sono l'ampiezza della banda di transizione, il suo "punto di origine" (ovvero la frequenza di inizio della banda di transizione), l'atte-



(a) *Spettro completo dei segnali.*



(b) *Dettaglio della zona attorno alla frequenza nulla.*

Figura 2.20: *Modulo della risposta in frequenza della componente y del segnale prima (blu) e dopo (rosso) l'applicazione del filtro progettato (con risposta in frequenza mostrata in Figura 2.19).*

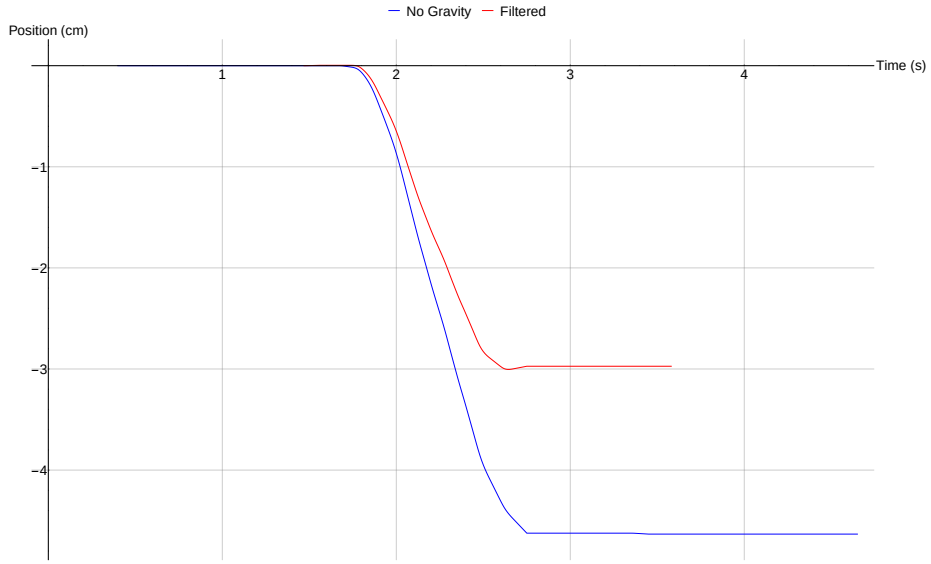


Figura 2.21: *Posizione ottenuta integrando lungo l'asse y, con l'algoritmo esposto in sottosezione 2.4.1, il segnale ottenuto tramite il filtraggio (rosso), ed il segnale con la componente continua rimossa tramite l'algoritmo descritto in sezione 2.2 (blu). Il segnale filtrato, inoltre, è stato traslato per allineare i due grafici.*

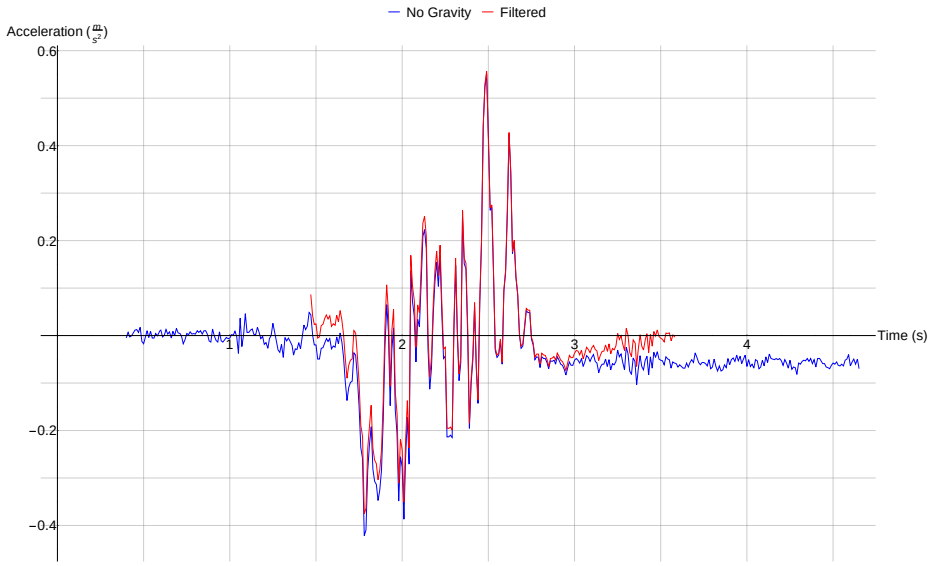


Figura 2.22: *Componente di accelerazione lungo l'asse y del segnale prima (blu) e dopo (rosso) il filtraggio. Il segnale filtrato è stato traslato per eliminare il ritardo introdotto e permettere un confronto più agevole.*

nuazione in stop-band ed il ripple in banda passante (che terremo al valore di default di 1 dB per semplificare l'analisi ed in quanto di minore interesse rispetto all'attenuazione imposta in stop-band), mentre l'ordine del filtro è stabilito dall'algoritmo, che cercherà di minimizzarlo. Per prima cosa, allora, cerchiamo la relazione esistente tra questi gradi di libertà e l'ordine del filtro prodotto, unico parametro che non possiamo controllare, in modo da poter stabilire consapevolmente i valori che ci permettono di ottenere il filtro migliore possibile. Fissiamo, quindi, l'origine e la larghezza della banda di transizione: come suggerito dall'intuizione, il risultato che si ottiene è che al diminuire dell'attenuazione imposta, anche l'ordine del filtro diminuisce. Per esempio, selezionando la banda di transizione $[0.2, 0.5]$ Hz si ottiene un filtro di ordine 916 con un'attenuazione di 80 dB, di ordine 710 con 60 dB e di ordine 498 con 40 dB. Come seconda verifica, fissiamo l'attenuazione imposta in stop-band e la frequenza di inizio della banda di transizione, e facciamo variare invece la sua ampiezza: quello che si ottiene è che all'aumentare dell'ampiezza della banda di transizione l'ordine del filtro cala. Per esempio, fissando l'attenuazione a 40 dB e l'origine della banda di transizione ad 1 Hz, il risultato è che per un'ampiezza di 0.5 Hz l'ordine risultante è di 300, per 1.5 Hz l'ordine è di 100, mentre per 2.5 Hz è di 60. Fissiamo, infine, l'ampiezza imposta in stop-band e la larghezza della banda di transizione, e ne variamo la frequenza di origine. Quello che si ottiene è che l'ordine del filtro non cambia al variare della frequenza di inizio della banda di transizione: per esempio, fissando l'attenuazione a 40 dB e l'ampiezza della banda di transizione ad 1 Hz si ottiene un filtro di ordine 76 sia che la banda di transizione inizi ad 1, 2 o 3 Hz.
Alla luce dei risultati trovati, analizziamo ora il filtro realizzato: dato che il punto di origine della banda di transizione non influisce sull'ordine del filtro, si è impostato questo a 0.05 Hz, mentre come ampiezza si è scelto 0.7 Hz e come attenuazione in stop-band 40 dB. Con questi parametri otteniamo un filtro di ordine 214 ed attenuazione in zero di circa 45 dB. Nonostante i valori adottati per i parametri siano lontani da quelli desiderati, i risultati ottenuti non ci concedono spazio di manovra: infatti sia aumentando l'attenuazione, che restringendo la banda di transizione, aumenteremmo l'ordine del filtro oltre la soglia stabilita, andando a perdere parte del segnale utile (infatti, per esempio, facendo terminare la banda di transizione a 0.5 Hz si ottiene un filtro di ordine 332).

Osserviamo, quindi, in Figura 2.19 il modulo (riportato in blu) e la fase (mostrata in rosso) del filtro ottenuto. Notiamo per prima cosa che la fase è lineare a tratti, con delle discontinuità di 2π dovute alla periodicità della fase: infatti in verde sono riportati i valori costanti $\pm\pi$, ed è possibile osservare che la fase presenta dei salti tra questi due valori. Per quanto riguarda il modulo, invece, notiamo come al termine della banda di transizione, a 0.75 Hz, il modulo raggiunga un valore vicino a zero ed in linea con i ripple presenti in banda passante, mentre alla frequenza nulla esso assume un valore di circa -45 dB. Applicando, invece, il filtro al segnale (consideriamo il segnale dello spostamento di 5cm a cui è stata rimossa la gravità come caso di studio), otteniamo il risultato mostrato in Figura 2.20, dove per chiarezza è stata riportata la sola componente y: da Figura 2.20a si può vedere che il modulo del segnale filtrato segue fedelmente quello originale, ad eccezione delle frequenze in banda di transizione e vicine allo zero. Dalla Figura 2.20a, infatti, si può osservare che tutte le frequenze nell'intervallo $[0,\ 1]$ Hz sono state attenuate, con la frequenza nulla che subisce l'attenuazione maggiore. Questa distorsione, tuttavia, ha causato una perdita di informazioni: infatti in Figura 2.21 è riportato il valore di posizione ricavato integrando il segnale ottenuto dal filtraggio (rosso), ed il segnale con la componente continua rimossa tramite l'algoritmo descritto in sezione 2.2 (blu). Com'è possibile osservare il segnale filtrato, quando integrato, restituisce una posizione che si assesta sui 3cm, quando lo spostamento reale è stato di 5cm. Per capire la causa di questo fenomeno, osserviamo Figura 2.22, dove viene riportato il segnale di accelerazione prima del filtraggio (blu) e filtrato (rosso): com'è possibile notare i due segnali sono molto simili, però vi sono delle piccole differenze tra i due, nell'ordine della seconda cifra decimale. Questo, tuttavia, significa che i due segnali hanno delle differenze nell'ordine di alcuni $\frac{cm}{s^2}$, che nella rilevazione di spostamenti dell'ordine di pochi centimetri non possono essere trascurate e diventano significative. Concludiamo, quindi, che questo filtro non è adeguato al nostro utilizzo, in quanto perde informazione significativa del segnale. Dato che con il metodo appena analizzato non siamo riusciti ad ottenere un filtro soddisfacente, studiamo ora la variante di equiripple che ci permette di fissare l'ordine del filtro: in questo caso i parametri dell'algoritmo che possiamo variare sono l'ordine del filtro, il punto di partenza e l'ampiezza della banda di transizione, mentre l'attenuazione nelle ban-

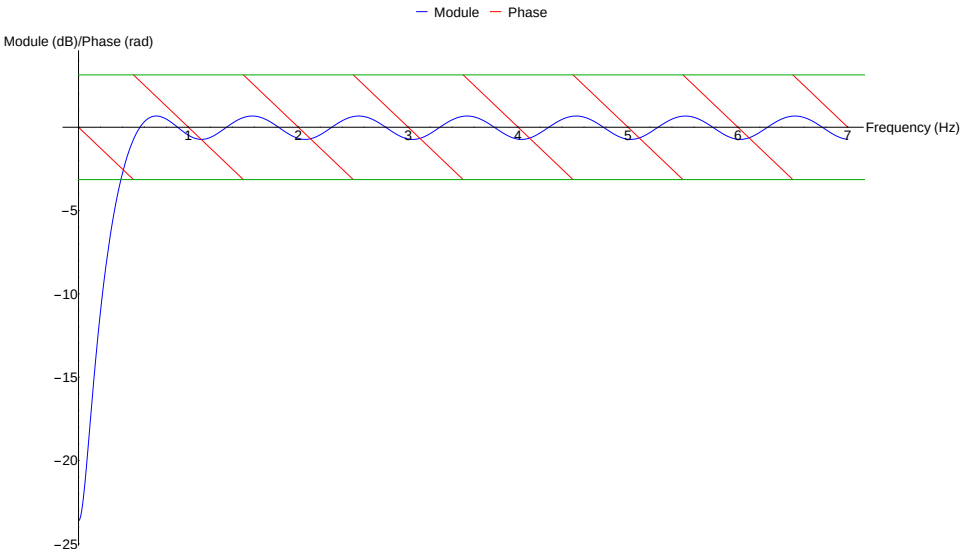
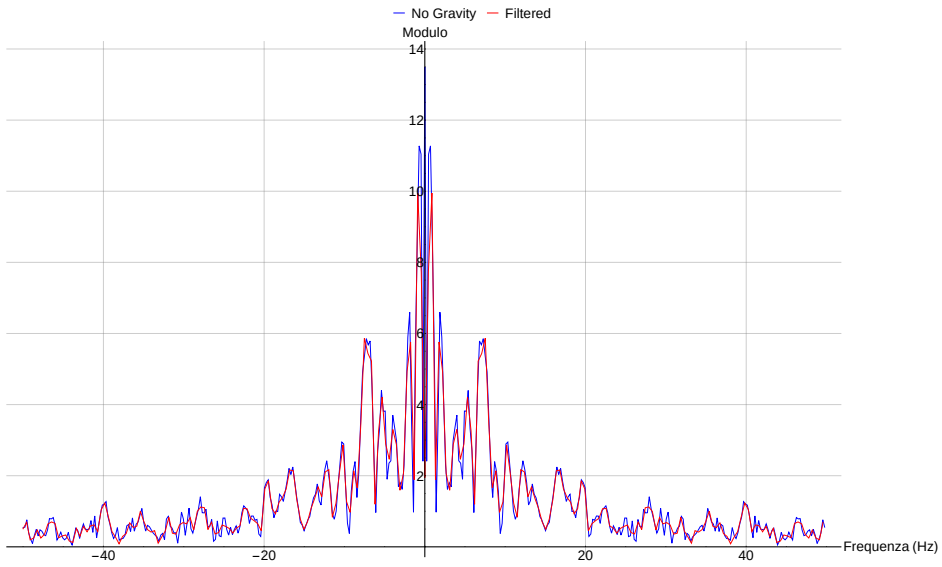
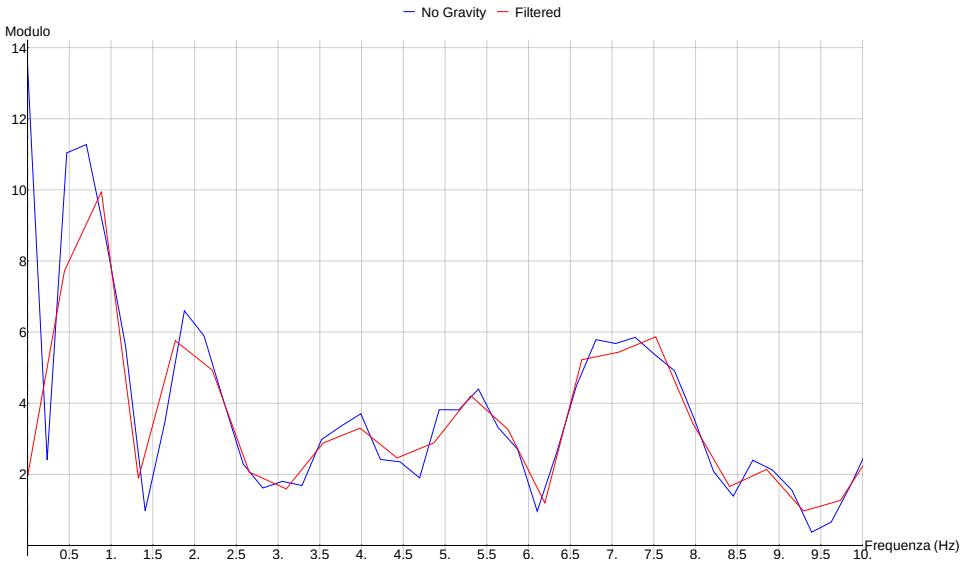


Figura 2.23: *Modulo in decibel (blu) e fase in radianti (rosso) della risposta in frequenza del filtro progettato con l'algoritmo equiripple ed ordine fissato a 200. In verde sono riportati i valori costanti di* $\pm\pi$.

de del filtro è stabilita completamente dall'algoritmo (ovviamente sempre mantenendo la forma di un passa alto).
Come fatto per la variante precedente, verifichiamo come varia l'attenuazione, specialmente in stop-band, al variare dei parametri a nostra disposizione. Per prima cosa verifichiamo la variazione al cambiamento di ordine del filtro, a parità degli altri parametri: come suggerito dall'intuizione, all'aumentare dell'ordine l'attenuazione in stop-band aumenta. Per esempio, fissando come banda di transizione l'intervallo $[1, 3.5]$ Hz, il risultato che si ottiene è che con ordine 50 l'attenuazione in zero raggiunge circa i 25 dB, con ordine 100 raggiunge circa 46 dB e con ordine 150 circa 65 dB. Quindi, dato che stiamo cercando l'attenuazione massima, fissiamo l'ordine del filtro al valore massimo, ovvero 200. Con questo ulteriore vincolo, osserviamo i due restanti parametri: fissiamo ora l'origine della banda di transizione e studiamo come varia l'attenuazione al variare dell'ampiezza della banda. Come nella variante precedente dell'algoritmo, rileviamo che l'aumento di ampiezza causa un aumento dell'attenuazione in stop-band: per esempio, fissando l'origine ad 1 Hz si ha che con un'ampiezza di 0.5 Hz l'attenuazione in zero è nell'ordine dei 30 dB, un'ampiezza di 1.5 Hz ci permette di ottenere circa 55 dB, mentre con 2.5 Hz raggiungiamo circa 80



(a) *Spettro completo dei segnali.*



(b) *Dettaglio della zona attorno alla frequenza nulla.*

Figura 2.24: *Modulo della risposta in frequenza della componente y del segnale prima (blu) e dopo (rosso) l'applicazione del filtro progettato (con risposta in frequenza mostrata in Figura 2.23).*

dB. Infine, fissiamo l'ampiezza della banda di transizione e ne variamo la frequenza di inizio: quello che si può osservare è che al variare dell'origine l'attenuazione in zero subisce variazioni modeste. Per esempio, la banda di transizione $[1,\ 2.8]$ Hz causa un'attenuazione in zero di circa 78 dB, mentre sia $[4,\ 5.8]$ Hz, che $[8.2,\ 10]$ Hz ottengono un'attenuazione nell'ordine dei 67 dB.
Alla luce dei risultati trovati, analizziamo ora il filtro realizzato: esso presenta ordine 200, per massimizzare l'attenuazione, ed una banda di transizione che inizia a 0.05 Hz, in quanto questo parametro influisce poco sull'attenuazione, ed una larghezza di banda di transizione pari a 0.45 Hz, in modo da ridurre l'attenuazione sulle basse frequenze del segnale. Con queste specifiche, il filtro ottenuto presenta in zero la deludente attenuazione di circa 24 dB. Tuttavia, l'unico parametro che possiamo aumentare per migliorare l'attenuazione è la larghezza di banda, in quanto l'ordine del filtro è già al valore massimo ammesso. Se, però, aumentassimo la larghezza di banda, aumenteremmo anche la distorsione del segnale, che, come vedremo a breve, risulta già elevata.
Riportiamo in Figura 2.23 il modulo (in blu) e la fase (in rosso) della risposta in frequenza del filtro ottenuto: com'è possibile osservare, il modulo alla frequenza nulla assume un valore di circa 24 dB e la fase è ancora lineare e presenta le stesse discontinuità di ampiezza 2π del filtro precedente. In Figura 2.24a, invece, è possibile osservare il modulo dello spettro del segnale prima (in blu) e dopo (in rosso) il filtraggio. Com'è possibile notare, lo spettro del segnale filtrato segue abbastanza fedelmente quello del segnale originale. Nelle vicinanze dello zero, invece, il segnale viene attenuato, anche se meno in confronto al filtro precedente, mentre la frequenza nulla risulta ancora la più attenuata. Nonostante questo, però, il risultato dell'integrazione del segnale filtrato ancora una volta non fornisce la posizione corretta, assestandosi ad un valore di circa 3.5cm. Questo è dovuto nuovamente dalle variazioni dell'accelerazione, simili a quelle rilevate nel caso del filtro precedente (visibili in Figura 2.22). Quindi anche questo filtro perde informazione significativa del segnale e non è adatto al nostro scopo.
Dato che con l'algoritmo equiripple non siamo riusciti a progettare un filtro che rispettasse le nostre specifiche e non distorcesse eccessivamente il segnale, analizziamo ora il metodo least squares: al contrario di equiripple, questo algoritmo non permette di trovare automaticamente il filtro di

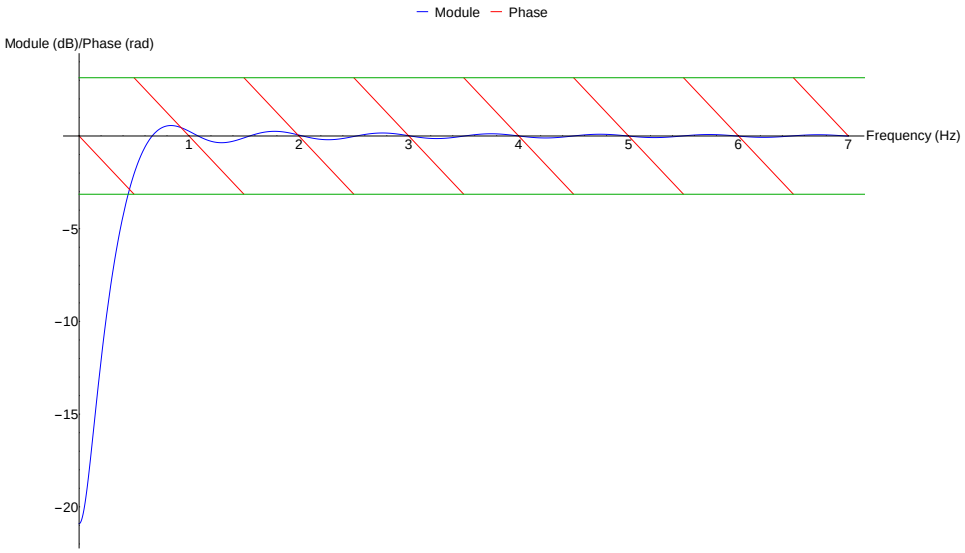
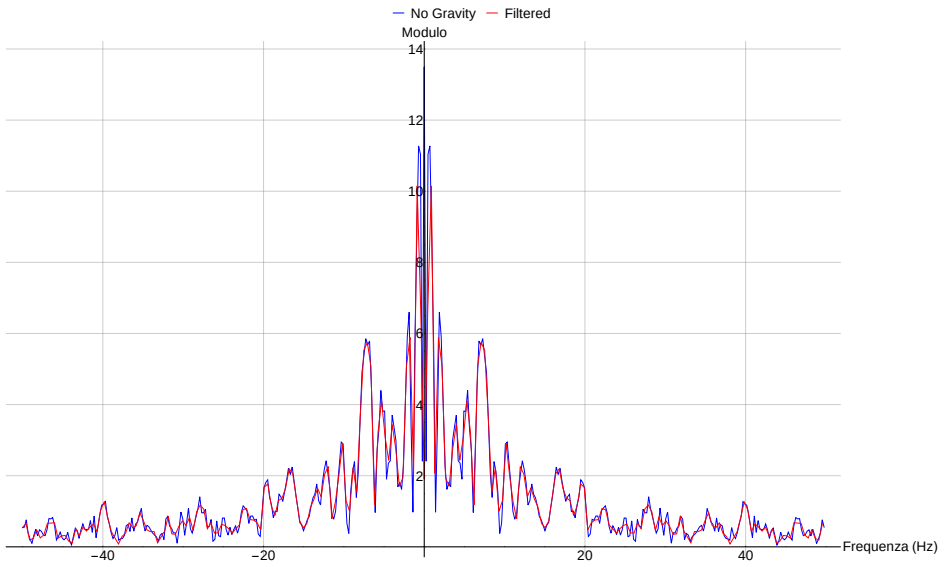
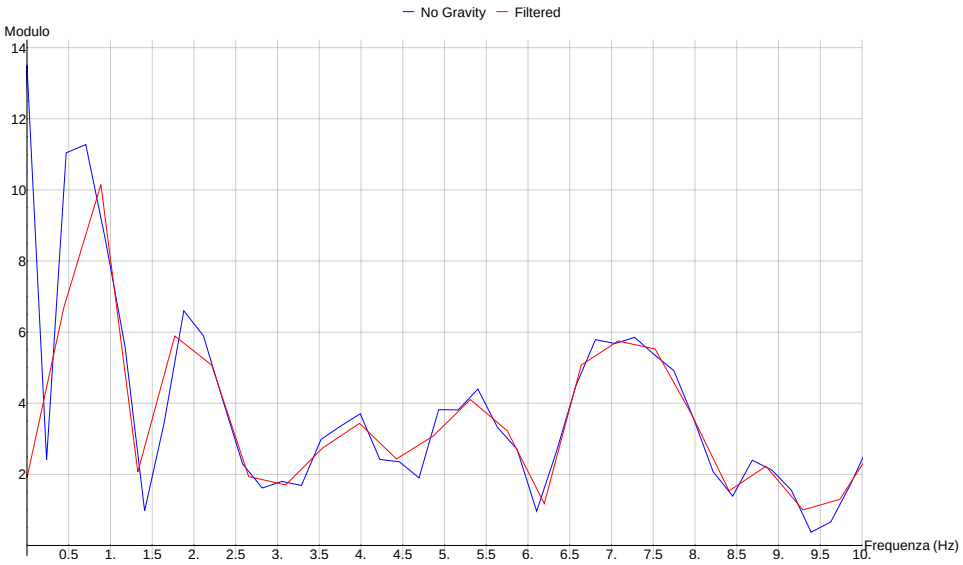


Figura 2.25: *Modulo in decibel (blu) e fase in radianti (rosso) della risposta in frequenza del filtro progettato con l'algoritmo least-squares ed ordine fissato a 200. In verde sono riportati i valori costanti di* $\pm\pi$.

ordine minimo, ma è necessario specificare manualmente l'ordine del filtro da realizzare. I parametri che è possibile variare sono l'ordine del filtro e origine ed ampiezza della banda di transizione, mentre l'attenuazione in stop-band è stabilita dall'algoritmo, che cercherà di fornire il filtro più possibile vicino all'ideale.
Come fatto con l'algoritmo precedente, verifichiamo come l'ordine del filtro influisce sull'attenuazione, a parità degli altri parametri: fissata la banda di transizione si è rilevato che all'aumentare dell'ordine anche l'attenuazione aumenta. Per esempio, fissato l'intervallo $[1, 3]$ Hz come banda di transizione, vale che con ordine 50 si ottiene un'attenuazione di 22 dB alla frequenza nulla, con ordine 100 si ottengono circa 40 dB, mentre con ordine 150 si raggiunge l'ordine dei 60 dB. Avendo verificato che questa relazione segue lo stesso andamento dei casi precedenti, scegliamo il massimo ordine ammesso, per poter ottenere la massima attenuazione, e studiamo le relazioni dei parametri rimanenti: per prima cosa, fissiamo l'origine della banda di transizione e studiamo l'attenuazione al variare della sua ampiezza. Quello che si può rilevare è che all'aumentare dell'ampiezza della banda di transizione, l'attenuazione alla frequenza zero (ed in generale nella stop-band) aumenta. Per esempio, fissando l'origine della banda di transizione



(a) *Spettro completo dei segnali.*



(b) *Dettaglio della zona attorno alla frequenza nulla.*

Figura 2.26: *Modulo della risposta in frequenza della componente y del segnale prima (blu) e dopo (rosso) l'applicazione del filtro progettato (con risposta in frequenza mostrata in Figura 2.25).*

a 1 Hz, con un'ampiezza di 0.5 Hz si ottiene un'attenuazione di circa 45 dB, con un'ampiezza di 1.5 Hz si raggiunge circa 60 dB, mentre con 2.5 Hz si ottiene un'attenuazione nell'ordine degli 80 dB. Verifichiamo ora l'attenuazione al variare del punto di origine: fissiamo la larghezza della banda di transizione e variamo il punto di inizio. Quello che si può rilevare è che l'attenuazione alla frequenza nulla non presenta differenze significative, indipendentemente dal punto di origine della banda di transizione. Per esempio, fissando l'ampiezza della banda di transizione a 1.5 Hz si ottiene un'attenuazione di circa 80 dB sia che essa inizi ad 1, 4 o 8 Hz.
Alla luce dei risultati trovati, analizziamo ora il filtro realizzato: come valori per i parametri si è scelto ordine 200, in modo da avere la massima attenuazione possibile, e banda di transizione $[0.05, 0.5]$, scelta con punto di origine molto vicino allo zero in quanto il suo valore influisce poco sull'attenuazione risultante, ed ampiezza tale da attenuare il meno possibile il segnale originale al di fuori dalla frequenza nulla. Con questi valori dei parametri, il filtro ottenuto ha un'attenuazione alla frequenza nulla pari a circa 22 dB: nonostante l'attenuazione sia deludente, non ci è possibile ottenere di meglio. Infatti l'ordine del filtro è già al valore massimo, mentre non è possibile aumentare la larghezza della banda di transizione perché, come vedremo a breve, già così avviene deformazione del segnale
In Figura 2.25 è possibile osservare il modulo (in blu) e la fase (in rosso) della risposta in frequenza del filtro progettato: come per gli altri filtri, è possibile notare che anche questo presenta una fase lineare a tratti, e si può osservare come alla frequenza nulla il modulo assuma un valore di circa 22 dB. In Figura 2.26, invece è riportato il modulo della risposta in frequenza del segnale prima (blu) e dopo (rosso) l'applicazione del filtro: com'è possibile osservare i due segnali sono quasi sovrapposti, ad eccezione delle frequenze nelle vicinanze dello zero, dov'è presente una leggera distorsione. Anche per questo filtro, tuttavia, si ottiene un comportamento analogo ai precedenti, per cui, una volta integrato, la posizione ottenuta si assesta a circa 3.5cm, invece che nelle vicinanze dei 5cm. Ancora una volta questo fenomeno si spiega osservando l'andamento dell'accelerazione, che presenta caratteristiche analoghe ai casi precedenti (e visibile in Figura 2.22 per il caso del primo filtro progettato), per cui le piccole differenze nell'ordine di alcuni $\frac{cm}{s^2}$ causano un risultato di posizione errato. Concludiamo, quindi, che neanche questo filtro è adatto all'utilizzo nel nostro contesto.

Non essendo riusciti ad ottenere il filtro desiderato neanche con il metodo least-squares, analizziamo ora l'ultimo algoritmo considerato, ovvero il metodo delle finestre. Come finestra è stata utilizzata la funzione denominata "kaiser", per la quale Matlab permette di scegliere se fissare l'ordine del filtro da progettare o far cercare all'algoritmo il filtro di ordine minimo. Come fatto con l'algoritmo equiripple, utilizziamo per prima la versione che permette di determinare il filtro di ordine minimo. In questo caso, i parametri che è possibile manipolare sono l'origine e l'ampiezza della banda di transizione, oltre all'attenuazione della stop-band, mentre l'ordine del filtro viene stabilito dall'algoritmo.
Come per i precedenti algoritmi, verifichiamo la relazione tra i vari parametri e l'ordine del filtro: quello che emerge è che le relazioni trovate sono le stesse rilevate nel caso della versione dell'algoritmo equiripple che cerca il filtro ad ordine minimo.
Analizziamo, quindi, il filtro realizzato: come attenuazione in stop-band si è imposto 30 dB, mentre la banda di transizione adottata è $[0.05,\ 0.75Hz]$. Con questi valori dei parametri si ottiene un filtro di ordine 220. Non è possibile ottenere un risultato migliore, in quanto un aumento dell'attenuazione farebbe crescere troppo l'ordine del filtro, mentre la larghezza di banda non può essere aumentata, in quanto già con questi valori le basse frequenze del segnale vengono alterate al punto tale da far ottenere un valore di posizione errato.
Studiamo, allora, la variante che ci permette di fissare l'ordine del filtro: in questo caso l'unico parametro che possiamo impostare, ad eccezione dell'ordine, è frequenza di taglio a metà della banda di transizione, in cui l'algoritmo impone un'attenuazione di 6 dB.Verifichiamo, quindi, come per i casi precedenti, la relazione tra ordine del filtro ed attenuazione in stop-band: ancora una volta, risulta che all'aumentare dell'ordine, a frequenza di taglio costante, l'attenuazione alla frequenza nulla ed in stop-band aumenta. Fissando, invece, l'ordine del filtro, all'aumentare della frequenza di taglio, l'attenuazione in stop-band aumenta (in particolare alle frequenze più basse).
Veniamo quindi alla progettazione: potendo solo specificare una frequenza di taglio con attenuazione 6 dB, siamo costretti a tenerne il valore molto basso, per evitare di distorcere il segnale. Il risultato che si ottiene, quindi, è che alle frequenze utilizzate (nell'ordine dei $0.1 - 0.2$ Hz), l'attenuazione

Tabella 2.1: *Riepilogo dei filtri progettati.*

Metodo	Banda di transizione[a]	Ordine	Attenuazione a 0 Hz
Equiripple (ordine minimo)	$[0.05, 0.75]$ Hz	214	45 dB
Equiripple (ordine fissato)	$[0.05, 0.5]$ Hz	200	24 dB
Least Squares	$[0.05, 0.5]$ Hz	200	22 dB
Finestre (ordine minimo)	$[0.05, 0.75]$ Hz	220	30 dB
Finestre (ordine fissato)	0.15 Hz	200	12 dB

[a] Nel caso del metodo delle finestre con ordine fissato, il valore indicato rappresenta la frequenza di taglio.

alla frequenza nulla (ed in stop-band) rimane nell'ordine dei 10/15 dB.
In conclusione, con nessuno degli algoritmi esaminati siamo riusciti ad ottenere un filtro che rispettasse le specifiche o non distorcesse eccessivamente il segnale. Possiamo concludere, quindi, che per ottenere un filtro con le caratteristiche desiderate è necessaria una progettazione più avanzata del filtro, dato che le specifiche necessarie alla nostra applicazione sono molto stringenti.

2.6 Analisi del rumore sonoro

Durante alcune delle analisi svolte nelle sezioni precedenti si è ricordato che le registrazioni delle accelerazioni utilizzate sono state prese in un ambiente esposto al rumore stradale, seppur in distanza. Emerge, quindi, la possibilità che il rumore sonoro possa aver influito sulla misura di accelerazione acquisita. Si è deciso, quindi, di verificare questa ipotesi osservando il segnale di accelerazione raw registrato dal sensore, in varie condizioni di rumore sonoro ritenute significative.
Per prima cosa si è voluto verificare se il rumore sonoro effettivamente influisce sull'accelerazione rilevata dal sensore. Una prima misura, quindi, è stata effettuata per verificare la sensibilità del sensore al rumore sonoro: posto il dispositivo in un ambiente molto rumoroso, lo si è reso di colpo silenzioso in modo da poter verificare la presenza della transizione nel segnale dell'accelerazione. A questo scopo si è utilizzato un televisore impo-

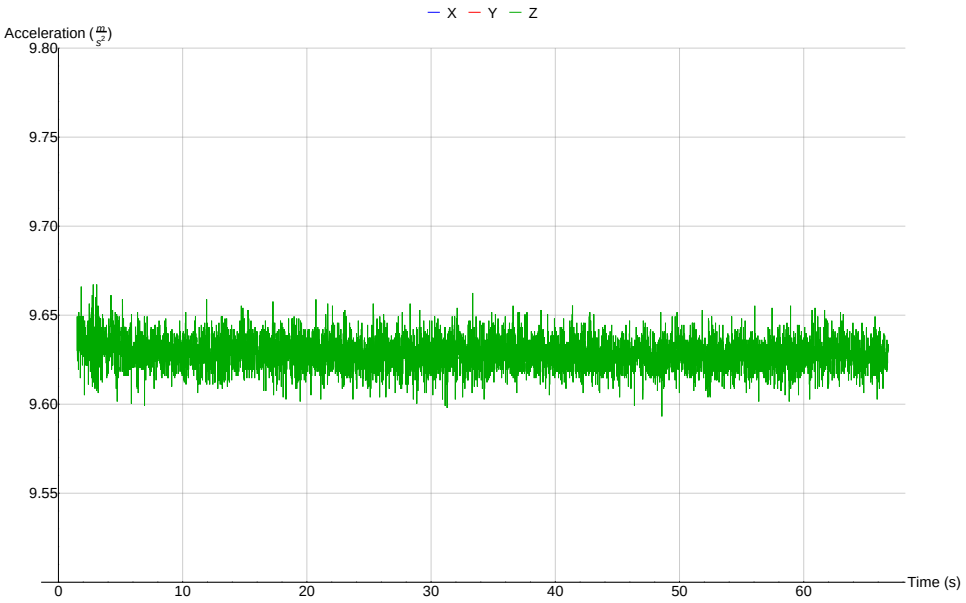


Figura 2.27: *Accelerazione registrata poggiando il dispositivo su una sedia di paglia di fronte ad un televisore al massimo volume, e silenziato a metà registrazione.*

stato ad un volume elevato, per poi silenziarlo con la funzione mute. In Figura 2.27 è possibile osservare il risultato della prima misura effettuata nelle condizioni descritte: il dispositivo è stato appoggiato su una sedia in paglia ad una distanza di circa mezzo metro dal televisore, impostato a volume elevato. In verde è riportata la componente z, mentre le altre due sono state lasciate fuori figura per poter vedere con maggiore dettaglio l'ampiezza del segnale, in ragione del fatto che tutte le componenti presentano un andamento analogo fra loro. Il televisore è stato impostato a mute dopo circa 33 secondi dall'inizio della registrazione, tuttavia, com'è possibile osservare dalla figura, il segnale non presenta nessuna variazione d'ampiezza tra quando il rumore sonoro era presente e quando c'era silenzio. Osserviamo, inoltre, in Figura 2.28 il modulo della risposta in frequenza di questo segnale: in blu è riportata la componente x, in rosso la y ed in verde la z. Notiamo che lo spettro non presenta nessuna frequenza dominante o picchi rilevanti, in accordo con il risultato ottenuto dall'analisi nel tempo.
È stata effettuata, quindi una seconda misura, poggiando il dispositivo sopra una scatola di plastica, a circa mezzo metro di distanza da un altro televisore con il volume impostato al massimo. Il risultato è mostrato in Figura 2.29: in verde è riportata la componente z dell'accelerazione, men-

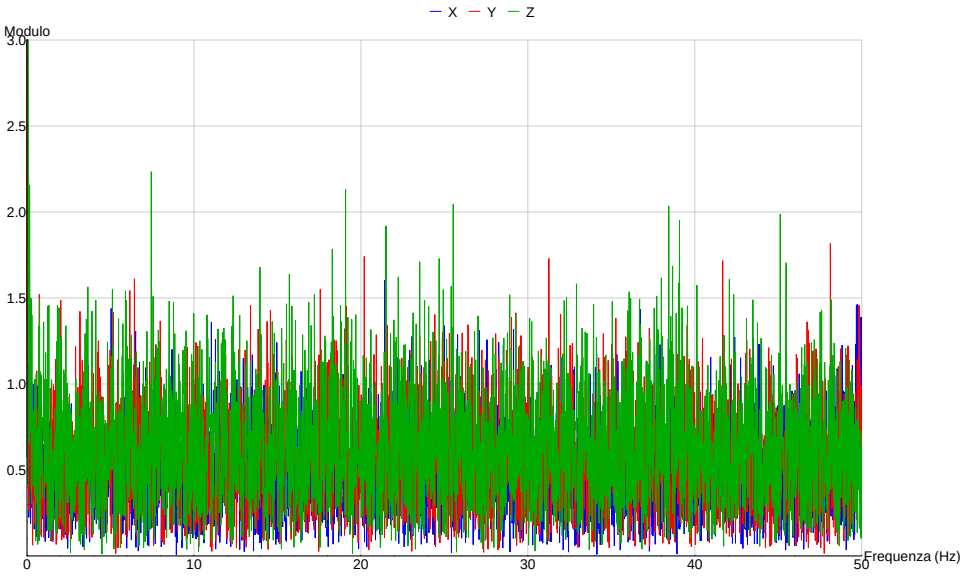


Figura 2.28: *Modulo dello spettro del segnale di Figura 2.27, ottenuto registrando un televisore a volume elevato da sopra una sedia di paglia.*

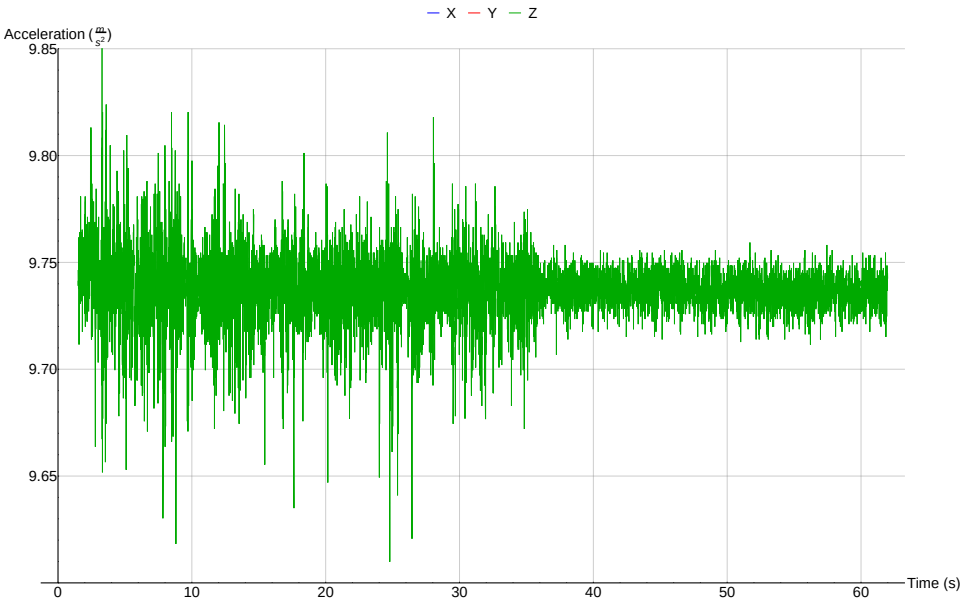


Figura 2.29: *Accelerazione registrata poggiando il dispositivo su una scatola di plastica a fianco ad un televisore al massimo volume, e silenziato a metà registrazione.*

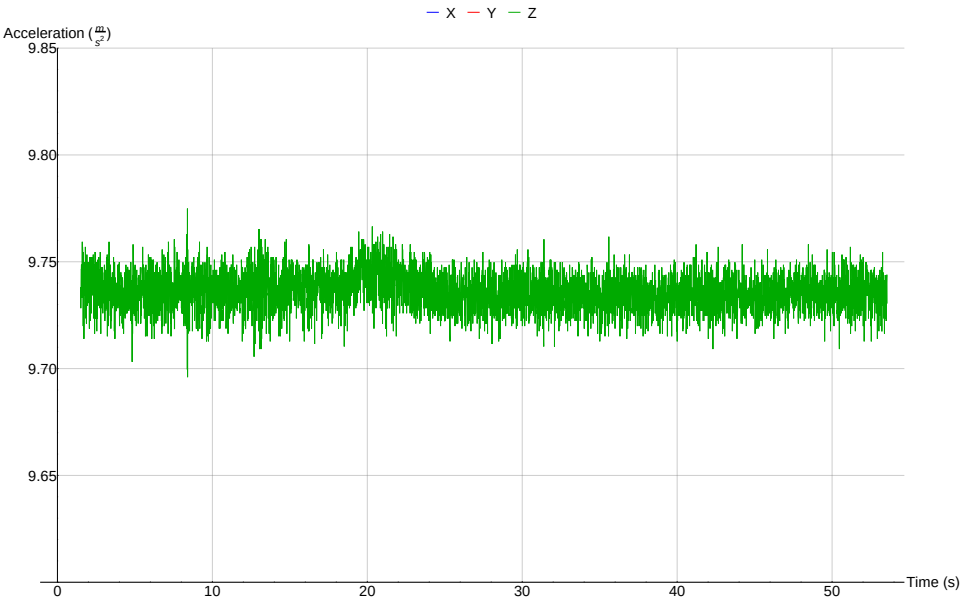
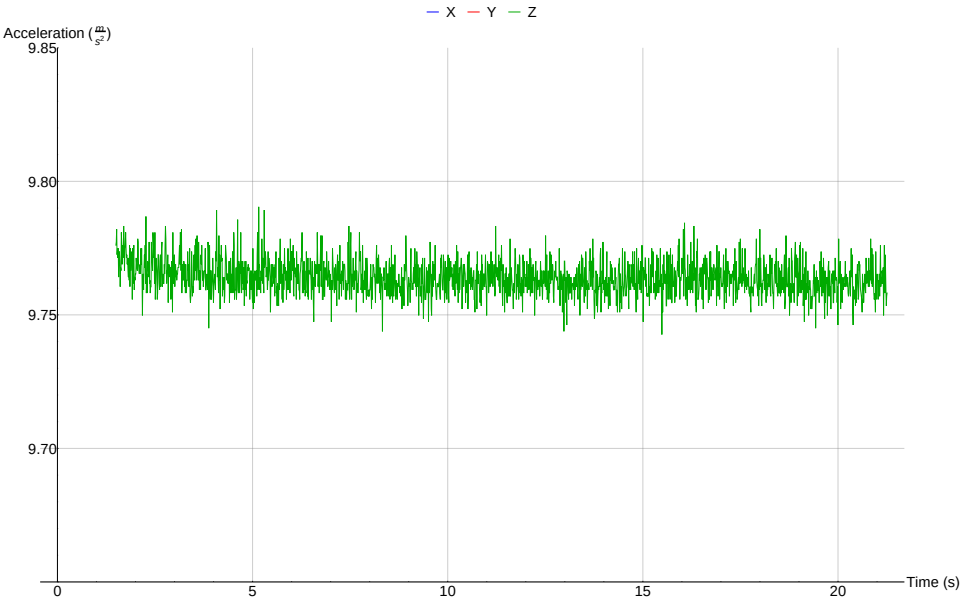


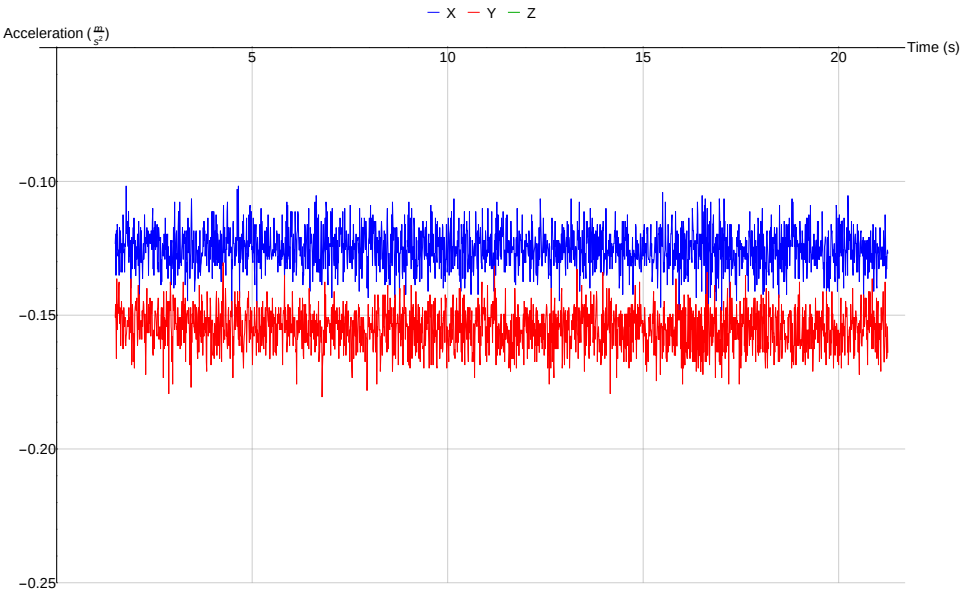
Figura 2.30: *Accelerazione registrata poggiando il dispositivo su un tavolo in legno ad una distanza di circa 4 metri da un televisore al massimo volume, e silenziato a metà registrazione.*

tre la x ed y sono state lasciate fuori figura per poter vedere con maggiore dettaglio il segnale, in quanto il fenomeno di interesse è più evidente sulla componente z. In questo caso il televisore è stato impostato a mute dopo 36 secondi dall'inizio del segnale: com'è possibile osservare dalla figura, in questo caso è chiaramente visibile la transizione dallo stato di rumore a quello di silenzio. Si è notato, tuttavia, che a causa del forte suono presente, la scatola di plastica presentava delle vibrazioni in presenza del rumore, che quindi hanno giocato un ruolo importante nel segnale registrato.
Infine, nell'ultima prova effettuata si è posto il dispositivo a circa quattro metri di distanza dal televisore, appoggiato sopra un tavolo di legno. Il segnale registrato è riportato in Figura 2.30: per gli stessi motivi del caso precedente è stata riportata la sola componente z dell'accelerazione. In questo caso il televisore è stato impostato a mute dopo 31 secondi dall'inizio della registrazione: com'è possibile osservare, anche in questo caso il sensore è riuscito a rilevare la transizione, nonostante questa risulti meno netta e sia più difficile distinguere l'accelerazione dovuta al suono dal rumore di fondo. Per questa misura, inoltre, contrariamente al caso precedente, non si sono notate vibrazioni sulla superficie di appoggio del dispositivo (per quanto possibile rilevare senza una strumentazione dedicata).

Concludiamo, quindi, che il sensore è effettivamente in grado di rilevare le vibrazioni dovute al suono ambientale, anche se la rilevazione è influenzata dalle vibrazioni della superficie di appoggio, ed il suono ambientale deve avere un volume molto elevato. Tuttavia, non è nei nostri scopi effettuare un'analisi approfondita su questo argomento, in quanto il nostro interesse è verificare la possibile presenza di un'accelerazione indesiderata nei segnali di spostamento registrati.
A questo scopo, dato che il rumore ambientale a cui le registrazioni degli spostamenti erano potenzialmente esposte era il rumore del passaggio di veicoli sulla strada, si è verificata questa condizione: per prima cosa si è considerato il caso del passaggio di un auto nelle vicinanze del dispositivo. Si è registrato, quindi, dal ciglio della strada il rumore di un auto in passaggio a circa 60-70 $\frac{km}{h}$ e distante approssimativamente 2 metri dal punto di registrazione. In Figura 2.31 e Figura 2.33 riportiamo i risultati più significativi ottenuti: in blu è riportata la componente x, in rosso la y ed in verde la z. Nella seconda figura non si è riportata la componente z per permettere di mostrare con più dettaglio le altre due componenti, che presentano tutta l'informazione di interesse. Quello che si è rilevato è che nella maggior parte delle misure effettuare il risultato ottenuto è quello mostrato nella prima figura, in cui il suono del passaggio dell'auto non ha causato variazioni nell'accelerazione. Infatti, per il caso particolare mostrato nella figura, il picco del rumore del passaggio dell'auto avviene nell'intervallo $[13, 16]$ secondi, ma, come si può notare confrontando l'accelerazione nell'intervallo con il resto del segnale, non vi sono differenze tra le due regioni, ed in entrambi i casi l'unico segnale visibile è il rumore del sensore. Questo è confermato anche dal modulo dello spettro del segnale, visibile in Figura 2.32, dove si può notare come l'unico segnale presente sia infatti il rumore. In Figura 2.33, invece, è riportato il caso in cui il suono del passaggio ha alterato maggiormente il segnale: il picco di intensità del suono dell'auto in passaggio avviene nell'intervallo $[16, 20]$ secondi, e quello che si può osservare dalla figura è che alla fine di questa regione è presente una piccola variazione del segnale, per quanto questa risulti difficile da distinguere dal rumore di fondo.
La seconda condizione considerata è quella che rispecchia le condizioni di acquisizione originali: si è registrato il rumore stradale nello stesso punto in cui sono state prese le prime registrazioni. In dettaglio il dispositivo è



(a) *Componente z.*



(b) *Componenti x ed y.*

Figura 2.31: *Accelerazione registrata sul ciglio della strada al passaggio di un'auto.*

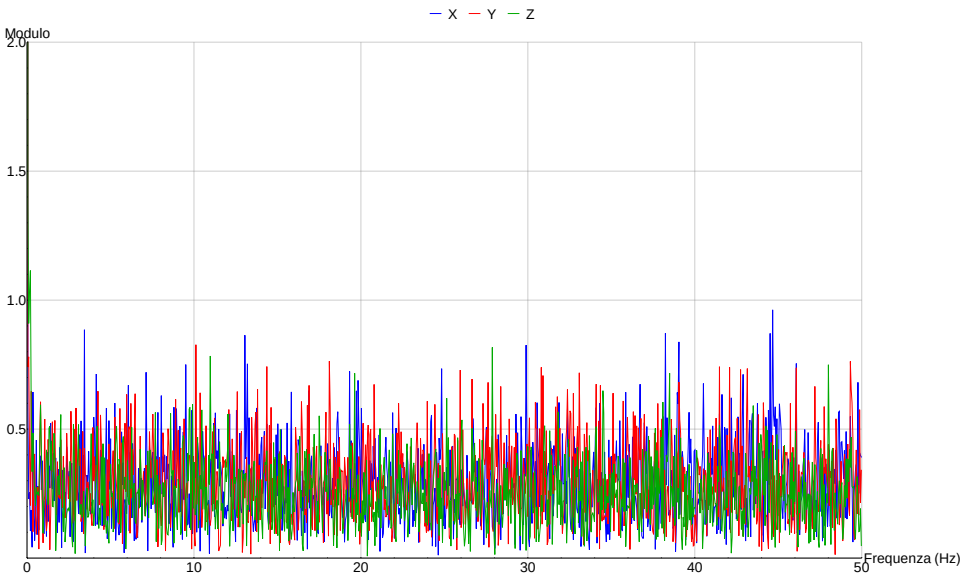


Figura 2.32: *Modulo dello spettro del segnale mostrato in Figura 2.31, in cui si è registrato il passaggio di un auto dal ciglio della strada.*

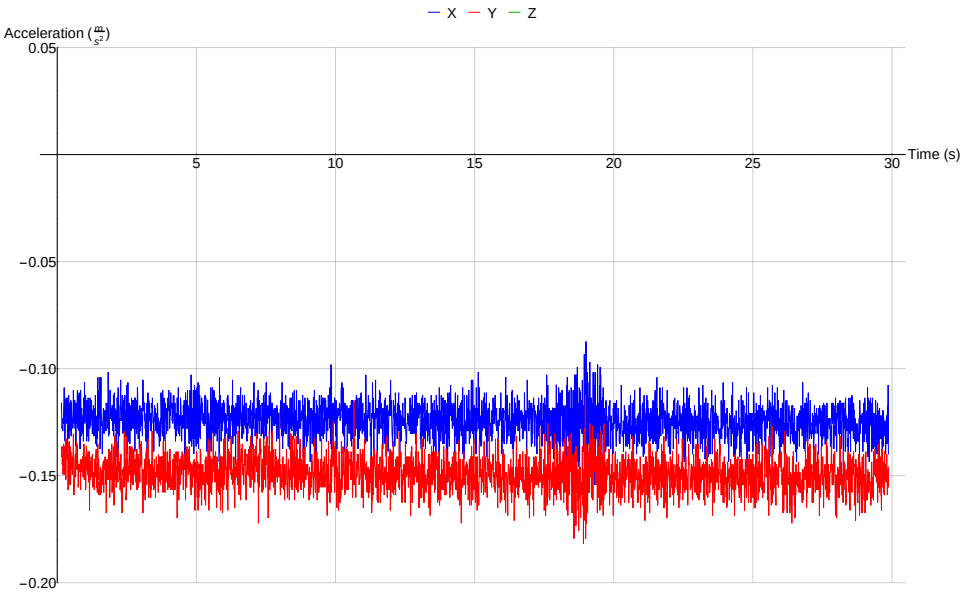
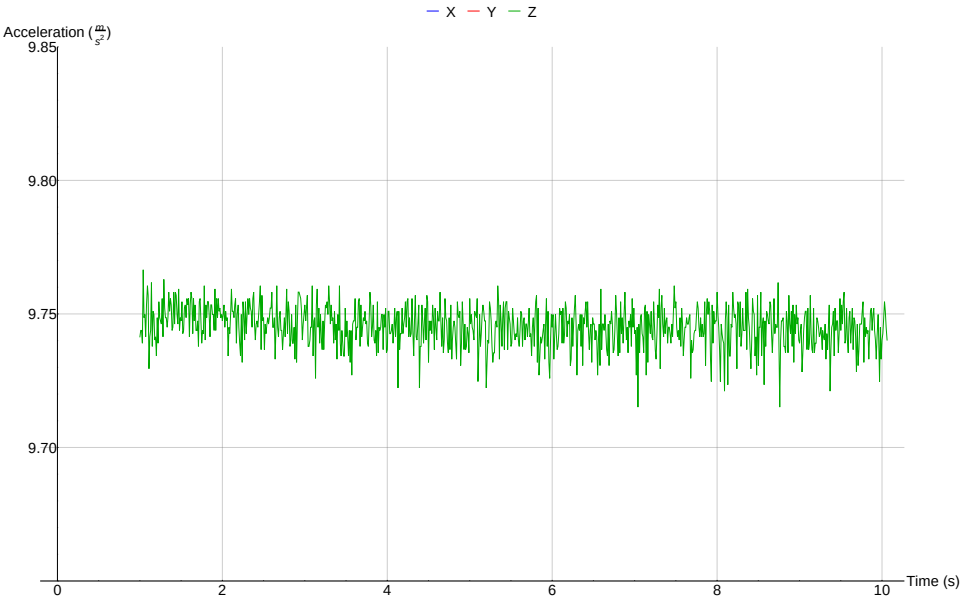
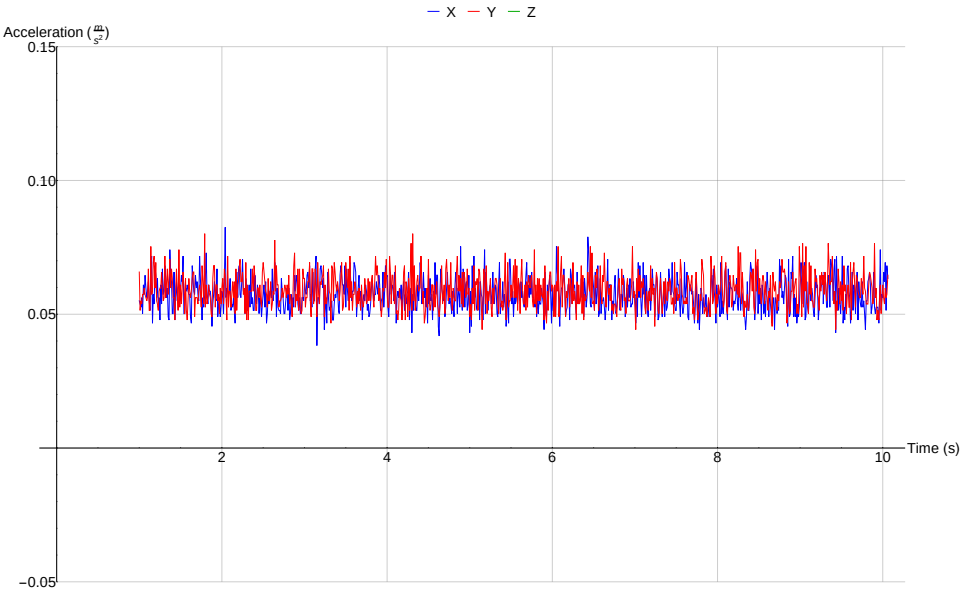


Figura 2.33: *Accelerazione registrata sul ciglio della strada al passaggio di una seconda auto.*



(a) *Componente z.*



(b) *Componenti x ed y.*

Figura 2.34: *Accelerazione registrata al passaggio di un'auto ad una distanza di circa 30 metri.*

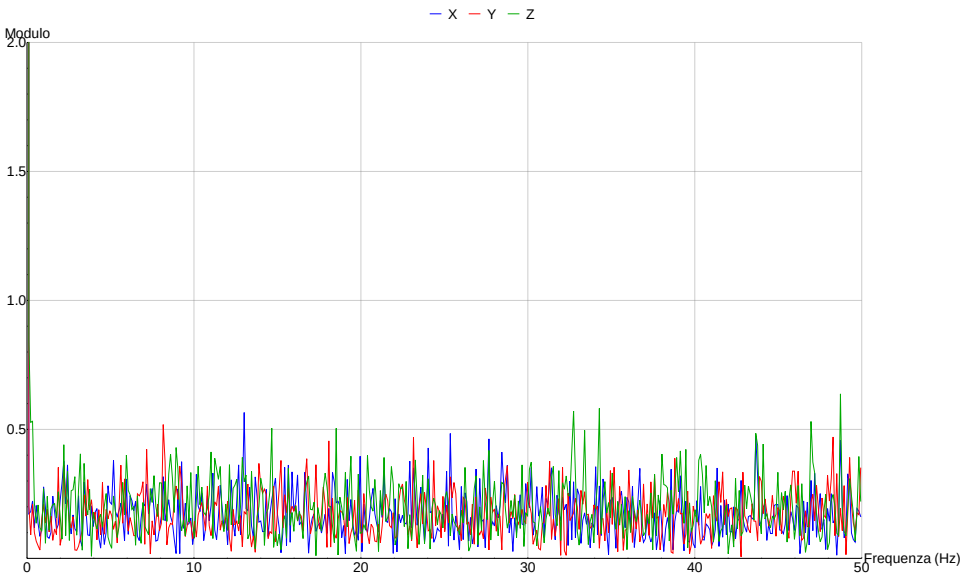


Figura 2.35: *Modulo dello spettro del segnale mostrato in Figura 2.34, in cui si è registrato il passaggio di un auto in lontananza.*

stato poggiato sopra un tavolo a circa 30 metri dalla strada e si è registrata l'accelerazione durante il passaggio di auto. I risultati sono visibili in Figura 2.34: com'è possibile notare, il segnale non presenta alcuna variazione, nemmeno nell'intervallo [3, 7] secondi dove è presente il picco di rumore dovuto al passaggio dell'auto. A conferma di questo, riportiamo in Figura 2.35 il modulo dello spettro del segnale, da cui si può osservare che l'unico elemento presente è il rumore. Concludiamo, quindi, che a questa distanza il rumore stradale non è rilevato dal sensore.

Per completezza, si è verificato anche come il rumore del parlato incide sul segnale: a questo scopo si sono effettuate alcune registrazioni in camera anecoica, gentilmente messa a disposizione dal Centro di Sonologia Computazionale dell'Università degli Studi di Padova, che ringraziamo. Nella prima condizione considerata si è poggiato il dispositivo sopra un tavolino e si è rimasti fermi sul posto, a circa un metro di distanza dal dispositivo, e da questa posizione si è parlato con un normale tono di voce nella direzione del tavolino. Il risultato di questa prova è visibile in Figura 2.36: in blu è riportata la componente x del segnale, in rosso la y, mentre la z (verde) è stata lasciata fuori dalla figura per poter vedere con maggior dettaglio il segnale, in quanto le tre componenti presentano un andamento analogo tra loro. Quello che si può notare dalla figura è l'assenza di variazioni

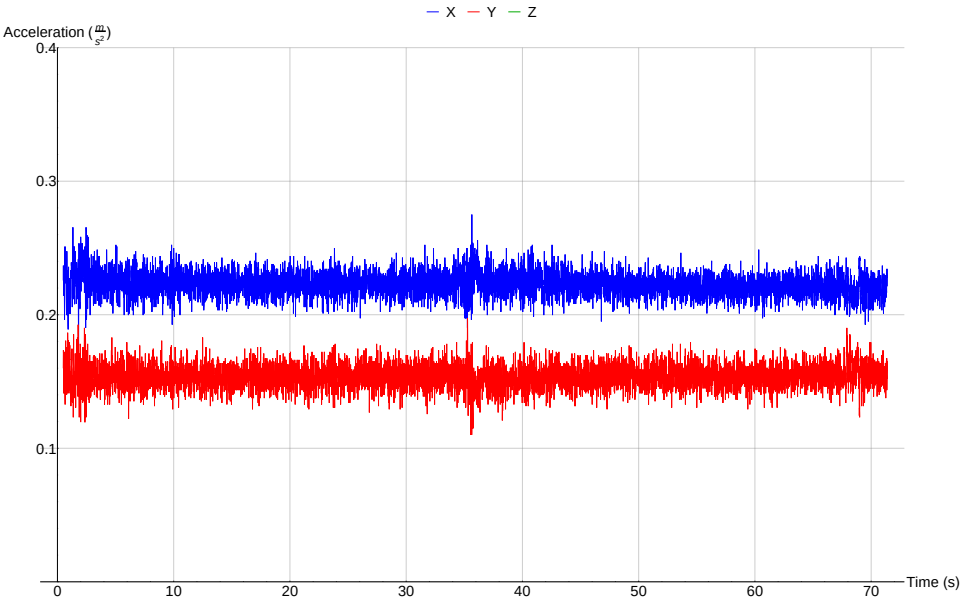


Figura 2.36: *Accelerazione registrata in camera anecoica, col dispositivo poggiato su un tavolino e ad un metro di distanza da una persona che parlava nella sua direzione.*

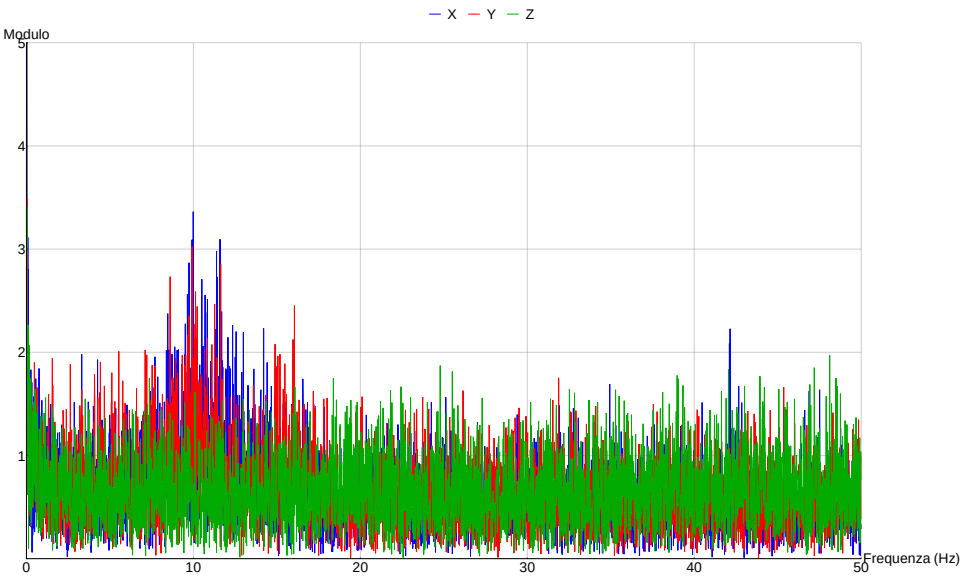


Figura 2.37: *Modulo dello spettro del segnale mostrato in Figura 2.36, registrato in camera anecoica parlando nella direzione del dispositivo.*

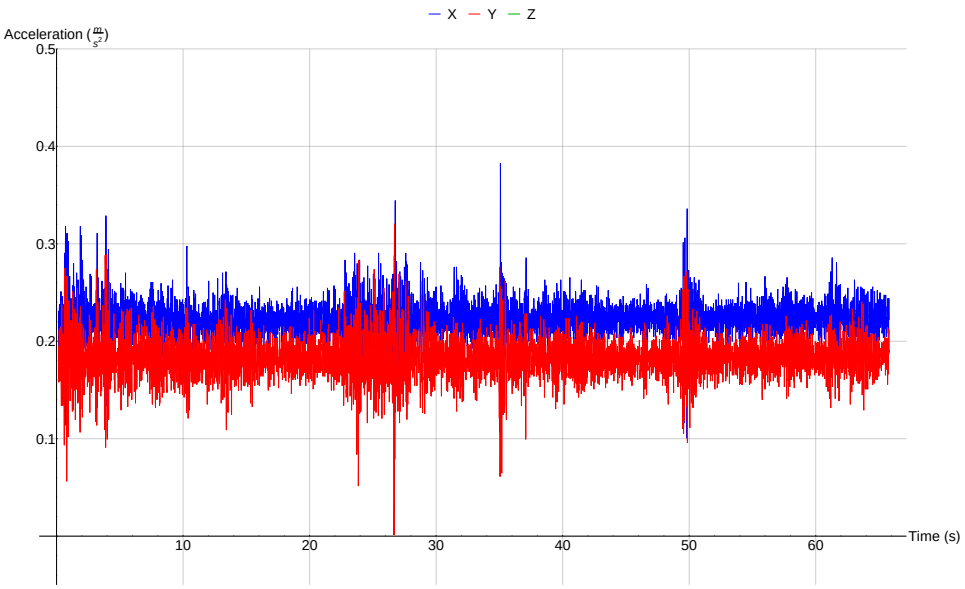


Figura 2.38: *Accelerazione registrata in camera anecoica col dispositivo poggiato su un tavolino, e parlando verticalmente sopra esso a varie distanze.*

nel segnale: l'unico elemento visibile è il rumore di fondo del sensore, ad eccezione della lieve increspatura vicino ai 35 secondi, dovuta ad un movimento che, a causa della flessibilità del pavimento della camera anecoica, ha causato una vibrazione del tavolino, rilevata dal sensore. A conferma di questo, osserviamo il modulo della risposta in frequenza del segnale, riportato in Figura 2.37: osserviamo la presenza di un lieve picco nella regione $[10, 15]$ Hz, al di fuori del range dell'udito umano e ragionevolmente dovuta all'increspatura del segnale osservata precedentemente.
Nella seconda condizione considerata, invece, si è posto ancora il dispositivo sopra il tavolino, ma contrariamente a prima ci si è posizionati al suo fianco e si è parlato verticalmente sopra il dispositivo a varie distanze. Il risultato è riportato in Figura 2.38: in blu è mostrata la componente x del segnale, in rosso la y, mentre la z (verde) è stata lasciata fuori dalla figura per poter mostrare il dettaglio del segnale. In questo caso è possibile identificare alcune regioni in cui l'accelerazione differisce: in particolare l'intervallo $[0, 15]$ secondi, quello $[22, 28]$ secondi ed i picchi a 35 e 50 secondi. La prima regione corrisponde ad un parlato effettuato con tono normale ad una distanza di circa 20cm dal dispositivo, la seconda regione corrisponde ad un parlato effettuato ad una distanza di 5/10cm, mentre i due picchi sono stati causati da dei colpi di tosse. Successivamente alla seconda regione si è

parlato ad una distanza di circa 50cm dal dispositivo, e si può osservare come in questa regione il segnale risulti pressoché indistinguibile dal rumore di fondo.

Possiamo concludere, quindi, che rumore sonoro ambientale ed il parlato vengono registrati dall'accelerometro. Tuttavia è necessario che il rumore ambientale abbia un volume molto elevato e che il parlato sia effettuato a distanza molto ravvicinata ed in direzione del dispositivo, affinché questi abbiano effetto sull'accelerazione rilevata. Questo, però, non è il caso delle condizioni in cui sono stati registrati i segnali di spostamento, che, quindi, possiamo concludere non risultino disturbati da un eventuale componente dovuta al rumore sonoro.

2.7 Sovracampionamento

Nelle analisi in frequenza effettuate nelle sezioni precedenti si è supposto costante il periodo di campionamento del segnale: in altre parole, si è supposto che tutti i campioni arrivassero ad intervalli regolari, cioè a multipli esatti di un fissato periodo di campionamento. Per trovare questo periodo è stata calcolata la media della differenza tra i tempi di arrivo dei campioni consecutivi. È noto, tuttavia, che Android non fornisce garanzie sul tempo di arrivo dei campioni, che quindi in linea di principio non risulta multiplo esatto di un periodo di campionamento. In questa sezione, quindi, esamineremo l'ipotesi fatta, per verificarne la validità e cercare di rilassarla. In particolare, per prima cosa verificheremo fino a che punto l'assunzione di arrivo di campioni ad intervalli costanti è valida, per poi considerare come riportarci a questa condizione, necessaria per gli algoritmi operanti in frequenza. Infatti, supponendo un periodo di campionamento costante, l'operazione logicamente effettuata è stata di "spostare" nel tempo i campioni al multiplo del periodo: per esempio, supponendo un periodo di campionamento di $10ms$, se un campione arriva dopo $12ms$, esso viene approssimato come se fosse arrivato dopo $10ms$. Quindi, per mantenere l'ipotesi di campioni multipli di un periodo (utilizzata dall'algoritmo di calcolo della trasformata di Fourier) e ridurre questo errore di approssimazione, l'operazione che è necessario effettuare è un sovracampionamento: aumentando il numero di campioni, la distanza tra di loro diminuisce, quindi al momento di approssimare un campione reale con uno multiplo

del periodo, l'errore commesso risulta minore. In letteratura, tuttavia, il sovracampionamento è un'operazione che assume esso stesso campioni a multipli di un periodo fissato, ed il cui obiettivo è proprio alterare questo periodo. L'algoritmo realizzato, quindi, è molto simile ad uno di sovracampionamento, pur presentando alcune differenze chiave: esso è descritto in sezione A.8. Esso accetta come input un segnale vettoriale di tre dimensioni, con la lista di timestamp associati ad ogni campione, oltre al periodo di campionamento originale del segnale (calcolato come descritto precedentemente) ed il periodo di campionamento desiderato per il segnale di uscita. Infine, l'ultimo parametro richiesto è il metodo da utilizzare per il calcolo del valore per i nuovi sample introdotti dall'algoritmo. Analizziamo, quindi, le azioni svolte dall'algoritmo. La prima operazione effettuata è quella di calcolare il numero esatto di campioni del segnale sovracampionato. Infatti, dovendo approssimare i sample del segnale originale con i più vicini di quello sovracampionato, c'è la possibilità che l'ultimo campione originale debba essere approssimato "per eccesso", e questo richiede un calcolo più attento del numero di campioni. Per prima cosa, quindi, si calcola il numero di campioni multipli del periodo di upsampling (ovvero del periodo del segnale di uscita) contenuti nell'intervallo di tempo compreso tra il primo e l'ultimo campione del segnale originale. Con questo numero, poi, si calcola il tempo dei due campioni del segnale sovracampionato, rispettivamente immediatamente precedente e successivo all'ultimo campione originale. Con questa informazione si può stabilire se approssimare l'ultimo campione originale per difetto o per eccesso: nel primo caso il numero di campioni totale del segnale sovracampionato è pari al numero calcolato precedentemente, in caso contrario il numero totale è semplicemente pari a quel valore incrementato di uno.
Avendo calcolato il numero di campioni del segnale di uscita, il passo successivo è di creare i campioni veri e propri del segnale: per prima cosa si imposta il primo campione uguale a quello originale. Successivamente si scorrono tutti i campioni del segnale sovracampionato: se il campione attuale è più vicino al prossimo campione originale rispetto al precedente ed al successivo sample del segnale di uscita, allora si dà al campione corrente il valore del sample originale, mentre lo si azzera in caso contrario. Si crea, in questo modo, un segnale in cui i campioni del segnale originale sono approssimati al più vicino tempo multiplo del periodo, mentre in mezzo a

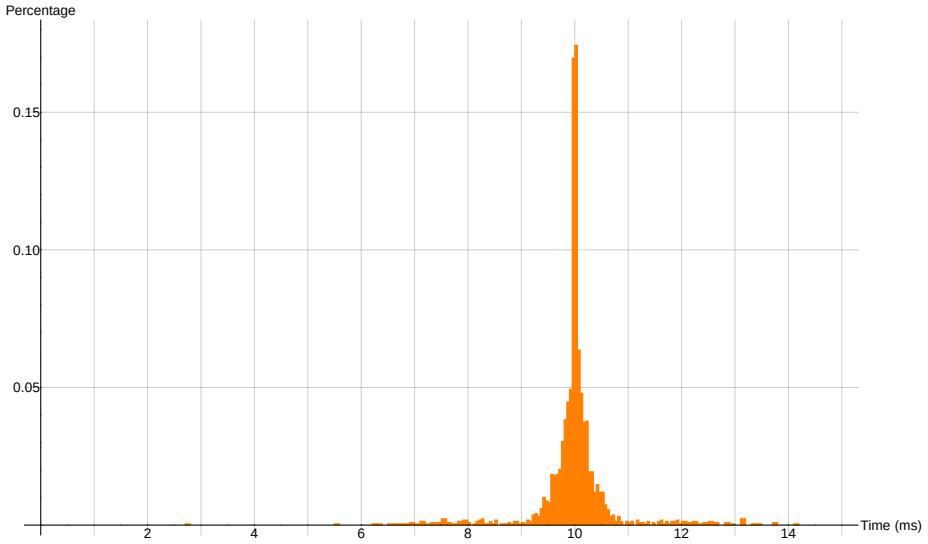


Figura 2.39: *Istogramma della durata dei tempi di inter-arrivo (tempo trascorso tra l'arrivo di un sample ed il successivo) espressa in millisecondi e per un periodo di campionamento nominale di* $10ms$.

tutti questi valori vi sono campioni nulli.
Ora, quello che è necessario fare è calcolare un valore adeguato per tutti i campioni nulli: per fare questo in letteratura esistono vari metodi, si veda per esempio [16, 18]. Quelli che si sono considerati sono un filtro interpolatore passa basso, ed una interpolazione lineare tramite i campioni adiacenti. La prima tecnica ragiona in frequenza: sovracampionando si sono create delle repliche dello spettro a multipli della frequenza di campionamento del segnale. Il filtro passa basso, quindi, ha lo scopo di eliminare queste ripetizioni periodiche, mantenendo lo spettro originale. Il secondo metodo, invece, opera nel tempo ed assume una variazione lineare tra un campione ed il successivo del segnale originale. La scelta di quale tecnica utilizzare è effettuata in base al parametro di input passato all'algoritmo.

2.7.1 Tempi di inter-arrivo dei campioni

Verifichiamo, quindi, per prima cosa la validità dell'ipotesi di periodo di campionamento costante: per fare questo studiamo il periodo di arrivo dei campioni, calcolando per ogni coppia di sample successivi la differenza tra i due tempi di arrivo e studiando la distribuzione di questi valori.
In particolare consideriamo il segnale registrato in condizioni di immobi-

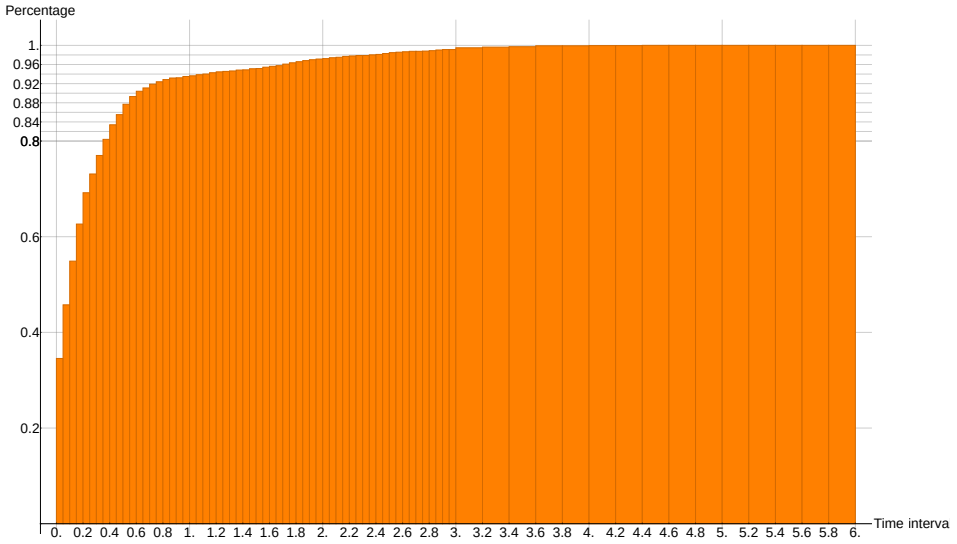


Figura 2.40: *Grafico della percentuale di tempi di inter-arrivo con ampiezza distante dalla media meno di un valore fissato, usato come variabile indipendente e riportato in ascissa.*

lità del dispositivo ed utilizzato nell'analisi esposta in sottosezione 2.3.1. In Figura 2.39 mostriamo l'istogramma dei tempi di inter-arrivo calcolati (ovvero dei tempi passati tra l'arrivo di un campione ed il successivo): in ascissa sono riportati i possibili valori di questi tempi (in millisecondi), mentre in ordinata è mostrata la percentuale di campioni che appartiene ad un particolare bin dell'istogramma. Come ampiezza dei bin utilizzati per suddividere i tempi è stata utilizzato un valore variabile: i bin nell'intervallo $[8, 12]$ millisecondi hanno un'ampiezza di $0.05ms$, mentre al di fuori dell'intervallo l'ampiezza è raddoppiata. Notiamo dalla figura che la distribuzione dei tempi di inter-arrivo segue una forma a campana ed osserviamo che il valore medio dei tempi di inter-arrivo è circa 10 millisecondi (9.9954 per essere esatti) mentre il valore massimo è 14.11 millisecondi ed il minimo è 2.73 millisecondi. È possibile osservare, infine, che qualitativamente la maggior parte dei tempi assume un valore entro un millisecondo dalla media. Al riguardo, osserviamo più in dettaglio quanto i tempi si discostano effettivamente dalla media: in Figura 2.40 è riportata la percentuale di tempi di inter-arrivo che assumono un valore distante dalla media non più di un parametro, visibile in ascissa ed utilizzato come variabile per la realizzazione del grafico. Dalla figura si può osservare, quindi, che entro un millisecondo dal valore medio sono racchiusi il 93% dei tempi: ovvero questa frazione dei tempi di inter-arrivo totali assume un valore nell'inter-

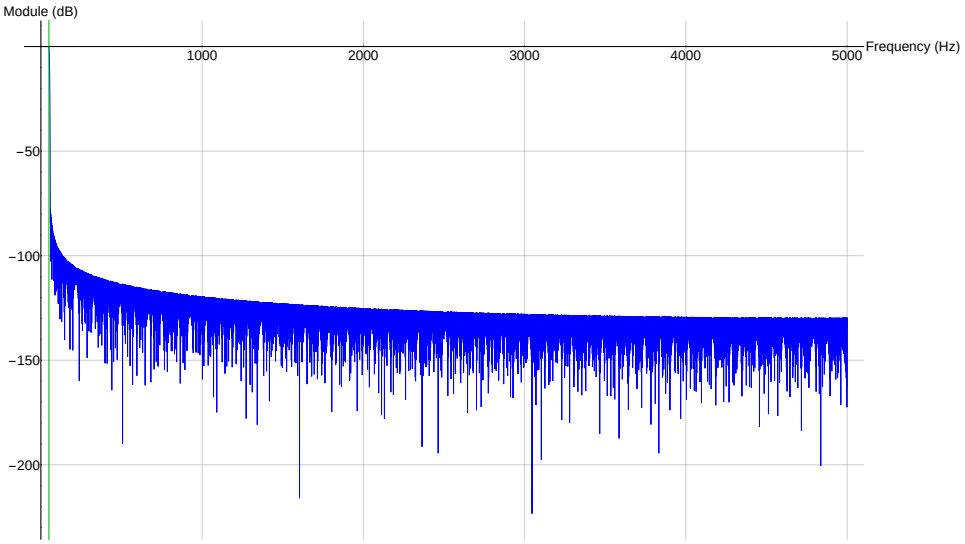


Figura 2.41: *Modulo della risposta in frequenza del filtro interpolatore utilizzato dall'algoritmo.*

vallo $[9, 11]$ millisecondi. Per ottenere il 95% dei tempi, invece, è necessario allargare l'intervallo ad $1.5ms$, mentre per il 99% bisogna salire a $3ms$. Concludiamo, quindi, che una certa variabilità nei tempi di arrivo è presente, e con essa la possibilità che l'ipotesi fatta precedentemente non fosse valida. Per esplorare questa possibilità, quindi, si rivela necessario effettuare l'operazione di upsampling descritta.

2.7.2 Filtro interpolatore

Analizziamo allora per prima cosa l'operazione di upsampling effettuata tramite l'algoritmo descritto precedentemente. Il metodo che studieremo in questa sezione è quello basato sul filtro interpolatore: esso, dopo aver approssimato i sample originali ed inserito i campioni nulli, elimina le ripetizioni periodiche dello spettro tramite l'applicazione di un filtro passa basso. Dalla sezione precedente, sappiamo che la frequenza di campionamento del sensore è circa 100 Hz, quindi la sua frequenza di Nyquist è circa 50 HZ: questa sarà la frequenza di taglio del filtro che utilizzeremo. Per la sua progettazione è stato utilizzato ancora una volta il tool fornito da Matlab: il filtro è stato realizzato con il metodo least squares ed ha ordine 4500 e banda di transizione $[48, 58]$ Hz. In Figura 2.41 riportiamo il modulo della risposta in frequenza del filtro utilizzato: in ascissa sono indicate

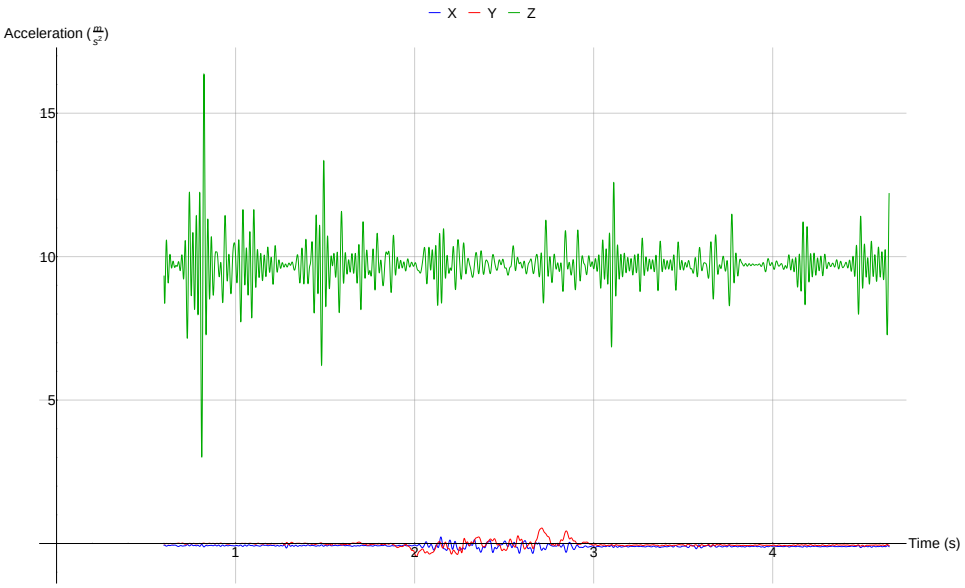


Figura 2.42: *Segnale di accelerazione ottenuto dall'algoritmo di upsampling con il metodo del filtro interpolatore.*

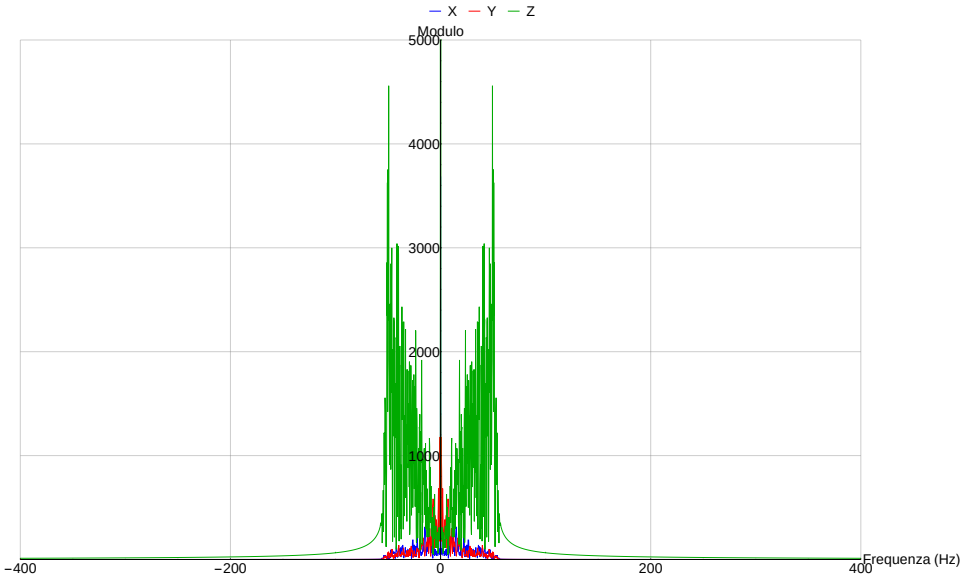
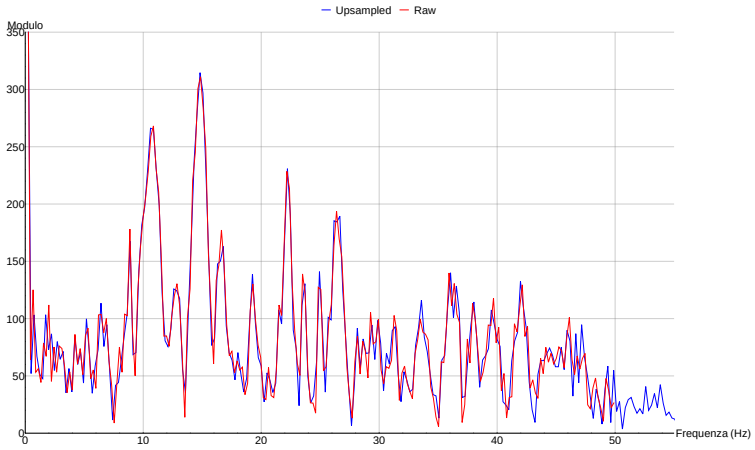
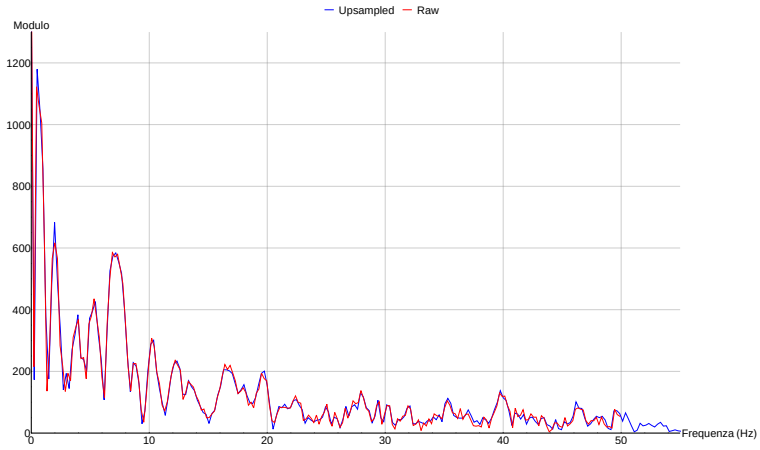


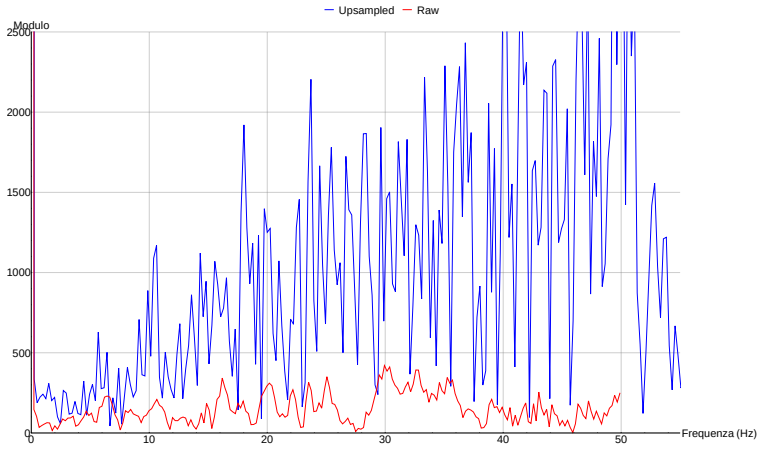
Figura 2.43: *Modulo della risposta in frequenza del segnale ottenuto dall'algoritmo di upsampling con il metodo del filtro interpolatore.*



(a) *Componente x.*



(b) *Componente y.*



(c) *Componente z.*

Figura 2.44: *Moduli degli spettri sulle tre componenti del segnale sovracampionato con il filtro interpolatore (blu) e originale (rosso).*

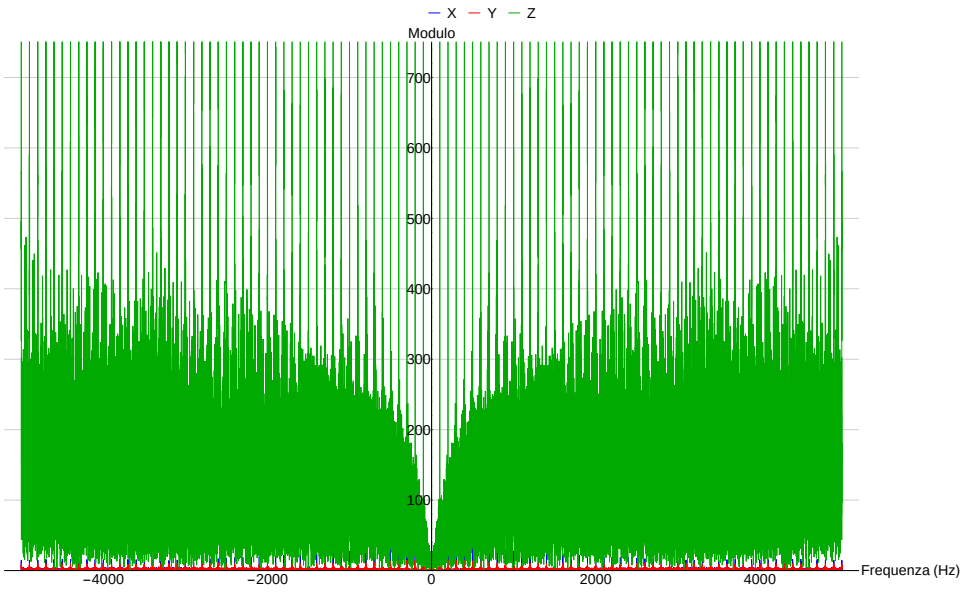


Figura 2.45: *Modulo della risposta in frequenza del segnale intermedio dell'algoritmo di upsampling, formato dai campioni del segnale originale approssimati, e dai sample nulli inseriti dall'algoritmo.*

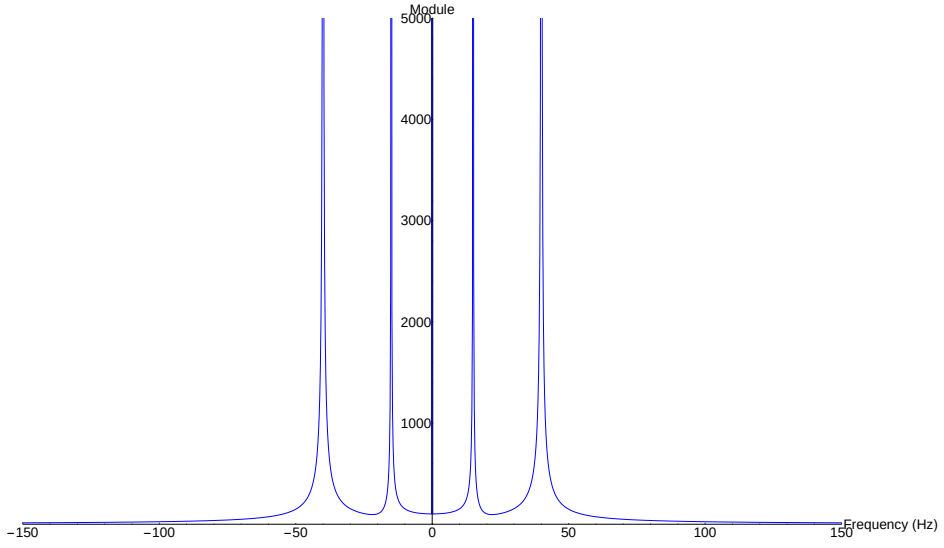


Figura 2.46: *Modulo della risposta in frequenza del segnale ottenuto sovracampionando il segnale sinusoidale considerato, con periodo di campionamento perfetto.*

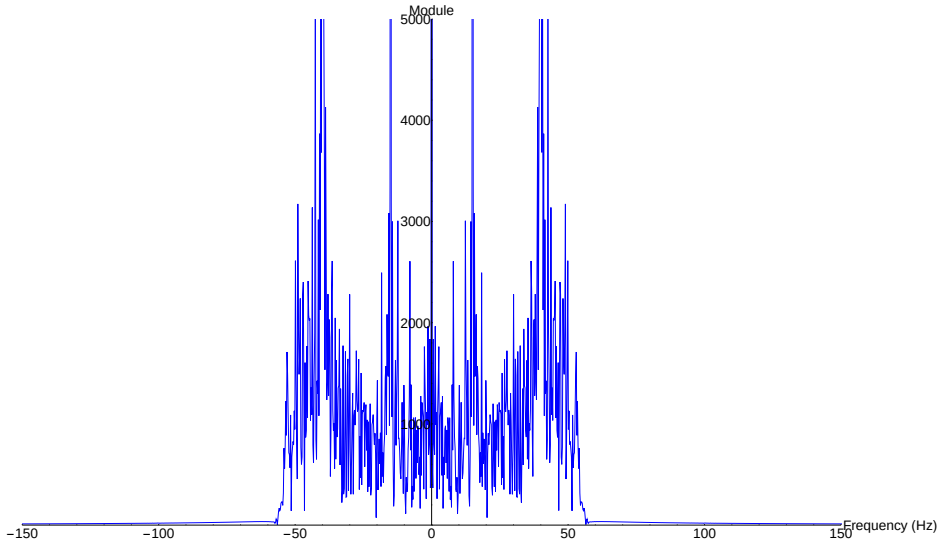


Figura 2.47: *Modulo della risposta in frequenza del segnale ottenuto sovracampionando il segnale sinusoidale considerato, con periodo di campionamento disturbato da rumore gaussiano.*

le frequenze, mentre in ordinata è riportato il modulo della risposta in frequenza, espresso in dB. Con una linea verde, inoltre, è stata evidenziata la frequenza di taglio di 50 Hz: com'è possibile osservare, in pass-band il filtro presenta attenuazione pressoché nulla, mentre in banda di transizione l'attenuazione cala rapidamente fino a raggiungere un valore nell'ordine degli 80 dB (che poi decresce ulteriormente in stop-band fino ad un valore di circa 120 dB).
Vediamo, quindi, il risultato dell'operazione di upsampling effettuata: in Figura 2.42 è riportato il segnale di accelerazione ottenuto dall'applicazione dell'algoritmo. In blu è mostrata la componente x del segnale, in rosso la y, mentre la z in verde. Com'è possibile osservare, le componenti x ed y ottenute seguono fedelmente il segnale raw (confrontando la figura con Figura 2.2); tuttavia, la componente z risulta completamente distorta.
Osserviamo allora la risposta in frequenza del segnale ottenuto, per capire il motivo di questa distorsione: in figura Figura 2.43 è riportato il modulo dello spettro del segnale ottenuto con l'algoritmo di upsampling. In blu è riportato il modulo della risposta della componente x, in rosso quello della componente y, mentre in verde quello della z. Per permettere una visione più dettagliata si sono lasciate fuori dalla figura le frequenze dell'intervallo $[400, 5000]$ e l'ampiezza è stata "troncata" a 5000, tagliando di fatto solo il

picco in zero della componente z. Notiamo che tutte e tre le componenti scendono velocemente a zero dopo i 50 Hz (corrispondenti alla frequenza di taglio), anche se la componente z decresce in maniera visibilmente più lenta delle altre. Osserviamo, inoltre, che le componenti x ed y presentano uno spettro simile a quello del segnale raw, mentre la componente z presenta un comportamento "anomalo": vediamo in Figura 2.44a e Figura 2.44b, infatti, che il modulo dello spettro delle componenti x ed y (mostrate in in blu) segue fedelmente quello del segnale raw (riportato in rosso). Il modulo della componente z (riportato in Figura 2.44c), al contrario, presenta un andamento completamente diverso da quello del segnale raw, crescendo al crescere della frequenza.
Per capire il motivo della forma del modulo dello spettro della componente z, osserviamo la risposta in frequenza prima dell'applicazione del filtro: in Figura 2.45 è riporto il modulo dello spettro del segnale intermedio nell'operazione di upsampling, composto dai campioni approssimati dal segnale raw e dai sample nulli inseriti. In blu è riportato il modulo della componente x, in rosso quello della componente y, ed in verde quello della z, mentre le ampiezza al di sopra dei 750 sono state lasciate fuori dalla figura per non perdere i dettagli dei segnali. Com'è possibile osservare, il modulo della risposta in frequenza presenta correttamente le ripetizioni della componente continua e dello spettro originale a multipli della frequenza di campionamento. Tuttavia, la componente z risulta fortemente distorta, presentando forti oscillazioni varianti dalla decina di Hertz fino all'ordine dei 300 Hz.
Non riuscendo a comprendere il motivo di questo fenomeno, si è deciso di analizzare un caso analogo, ma semplificato: si è scelto di utilizzare un segnale formato da un valore costante e due sinusoidi, rispettivamente con frequenze 15 Hz e 40 Hz. Questo segnale, successivamente, è stato campionato in due modi differenti: in un primo caso si è campionato il segnale ad intervalli perfettamente multipli di $10ms$, rispecchiando quindi il caso ideale. Nel secondo caso considerato, invece, si è campionato il segnale con un periodo di campionamento di $10ms$, ma disturbato da del rumore gaussiano, tale da presentare una distribuzione dei tempi di arrivo analoga a quella rilevata in sottosezione 2.7.1.
Analizziamo, quindi, il primo caso considerato, con i tempi di arrivo perfettamente multipli del periodo di campionamento. Per prima cosa applichiamo l'algoritmo di upsampling al segnale, portando il periodo di campio-

namento a $0.1ms$, e ne calcoliamo la risposta in frequenza: in Figura 2.46 è riportato il modulo dello spettro del segnale ottenuto. Per chiarezza le frequenze oltre i 150 Hz sono state omesse, in quanto in questa regione lo spettro presenta correttamente modulo nullo, mentre l'ampiezza del modulo è stata tagliata a 5000 per permettere di osservare i dettagli dello spettro. Notiamo che il modulo presenta correttamente dei picchi alle frequenze di 0, 15 e 40 Hz, corrispondenti alle tre componenti del segnale, e che anche le ripetizioni periodiche alle frequenze superiori alla frequenza di Nyquist sono state correttamente eliminate. Sono comparse, tuttavia, delle frequenze estranee tra i picchi del modulo: in queste regioni lo spettro dovrebbe essere nullo, in quanto composto da soli impulsi alle frequenze delle sinusoidi. Il modulo, invece, presenta una concavità tra i picchi, senza scendere, però, a zero. Si è ipotizzato che questo effetto sia dovuto alla non idealità del filtro, assieme al fatto di aver troncato il segnale sinusoidale, che idealmente dovrebbe essere infinito, tuttavia non si è stati in grado di confermare questa ipotesi.
Consideriamo, quindi, il secondo segnale realizzato, con il tempo di arrivo dei sample disturbato da rumore gaussiano. Anche per questo segnale effettuiamo l'operazione di upsampling e ne calcoliamo la risposta in frequenza: in Figura 2.47 è possibile osservare lo spettro ottenuto. Notiamo che anche in questo segnale i picchi dovuti alle sinusoidi ed alla componente continua sono presenti, e che per le frequenze superiori alla frequenza di Nyquist lo spettro è nullo. Quello che risulta evidente, però, è che ora, oltre al problema rappresentato dalle frequenze non nulle intermedie ai picchi, risulta anche che queste frequenze intermedie presentano un'ampiezza maggiore e molto più "rumorosa" rispetto al caso precedente. Possiamo concludere, quindi, che l'arrivo dei sample a tempi non perfettamente multipli di una frequenza di campionamento causa un disturbo nell'operazione di upsampling, tuttavia non siamo riusciti a comprendere più in dettaglio il motivo dei risultati ottenuti, ed in particolare non si è riusciti a capire la ragione della maggior distorsione della componente z rispetto alle altre, nel caso del segnale reale acquisito.

2.7.3 Interpolazione lineare

Abbiamo visto nella sezione precedente il comportamento dell'algoritmo di upsamping effettuato con il filtro interpolatore. In questa sezione

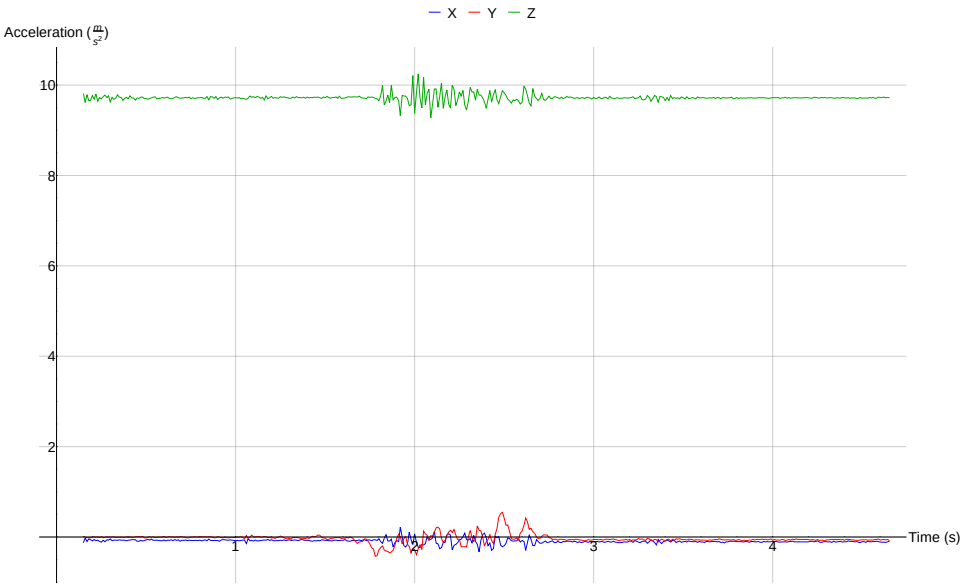


Figura 2.48: *Segnale di accelerazione ottenuto dall'algoritmo di upsampling con il metodo di interpolazione lineare*

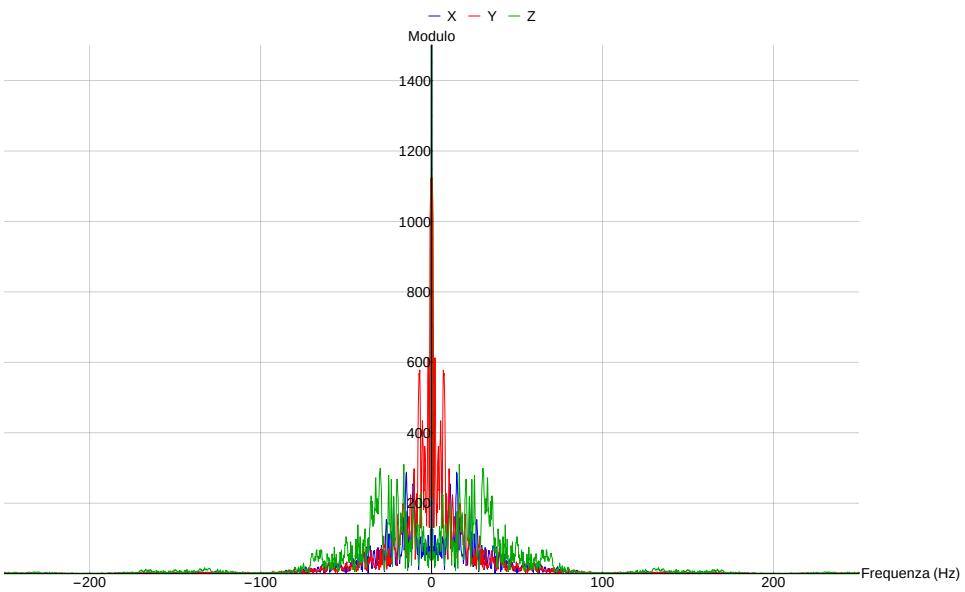
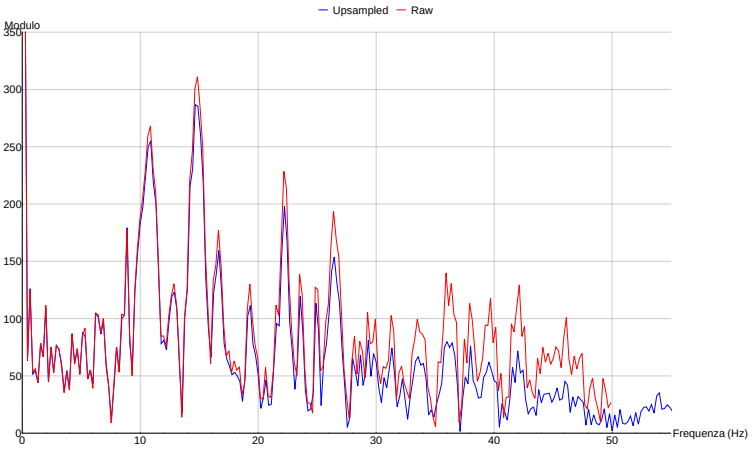
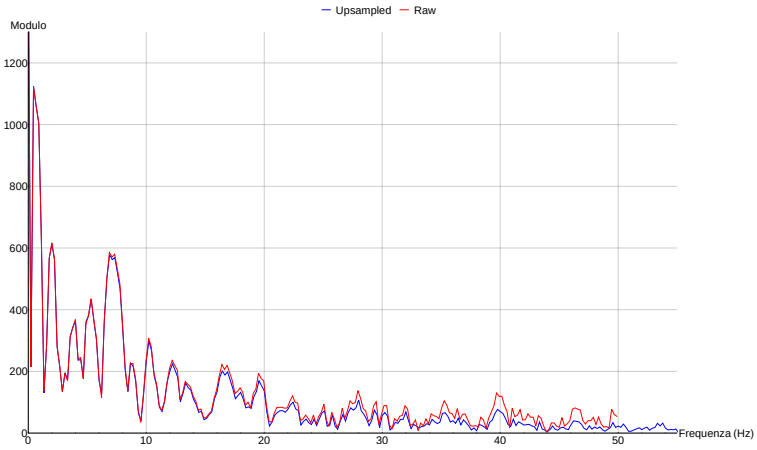


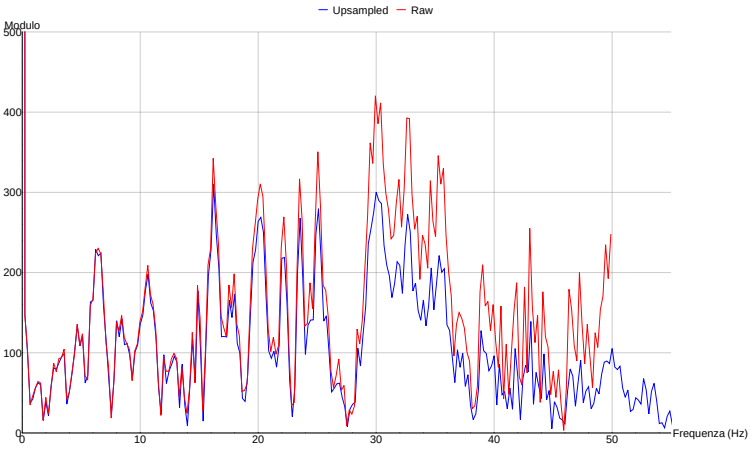
Figura 2.49: *Modulo della risposta in frequenza del segnale ottenuto dall'algoritmo di upsampling con il metodo di interpolazione lineare.*



(a) *Componente x.*



(b) *Componente y.*



(c) *Componente z.*

Figura 2.50: *Moduli degli spettri sulle tre componenti del segnale sovracampionato tramite interpolazione lineare (blu) e originale (rosso).*

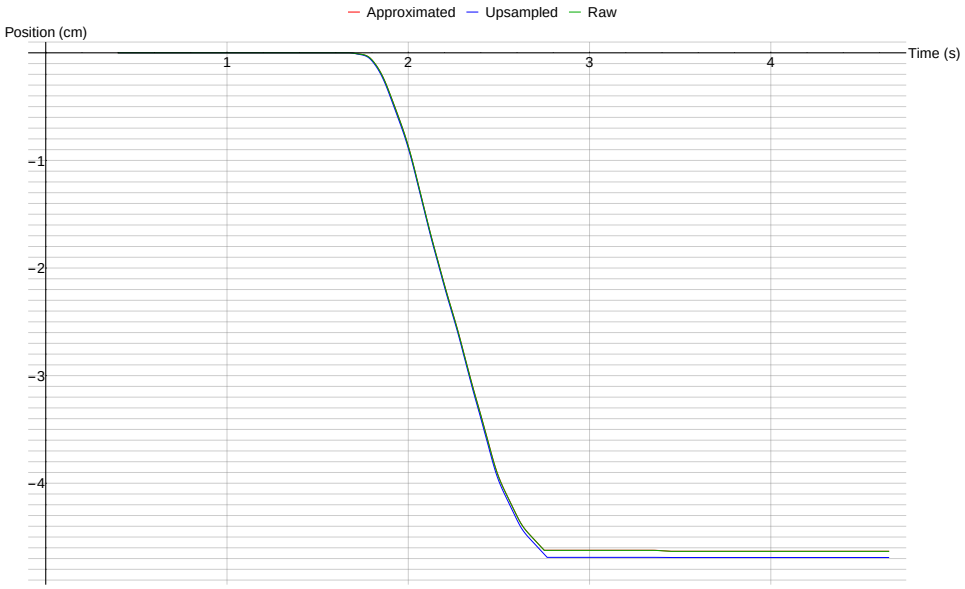


Figura 2.51: *Posizione, espressa in centimetri, ottenuta integrando il segnale sovracampionato (Upsampled, di colore blu), raw (di colore verde) e quello ottenuto applicando l'ipotesi di allineamento dei campioni a multipli del periodo di campionamento (Approximated, di colore rosso).*

consideriamo invece il metodo di interpolazione lineare: ricordiamo che questa tecnica assume una variazione lineare del segnale sovracampionato nell'intervallo di tempo compreso tra due campioni del segnale originale. Quindi, invece che operare in frequenza, questo metodo lavora nel dominio del tempo, assegnando ad ogni campione di uscita un valore corrispondente alla combinazione lineare tra i sample originali che lo precedono e lo seguono, pesandoli con la loro distanza dal campione. È possibile, comunque, osservare questa operazione anche dal punto di vista della risposta in frequenza: essa equivale ad un'operazione di filtraggio con la funzione $sinc^2(x)$.

Osserviamo, quindi, il risultato di questa operazione: in Figura 2.48 è riportato nel tempo il segnale sovracampionato. In blu è mostrata la componente x, in rosso la y, mentre la componente z è rappresentata in verde. A differenza del caso precedente, è possibile notare come tutte le componenti seguano fedelmente il segnale originale. A riconferma di questo, osserviamo in Figura 2.49 il modulo della risposta in frequenza del segnale sovracampionato: in blu è riportata la componente x, in rosso la y, mentre in verde quella z, ed osserviamo che anche in frequenza il segnale segue fedelmen-

te l'originale. Notiamo, tuttavia, che in questo caso il modulo raggiunge ampiezza nulla solamente intorno alla frequenza di 100 Hz, e nell'intervallo $[120, 180]$ Hz è presente una replica dello spettro, seppur con ampiezza notevolmente ridotta: questo è causato dal fatto che la funzione $sinc^2(x)$, praticamente utilizzata come filtro interpolatore, è un povero passa basso e presenta un'attenuazione modesta al di fuori del lobo centrale della funzione. Vediamo più in dettaglio, quindi, il modulo dello spettro delle singole componenti del segnale: in Figura 2.50a è riportato il modulo dello spettro delle componenti x segnale originale e sovracampionato, in Figura 2.50b quello delle componenti y ed in Figura 2.50c quello delle z. Com'è possibile osservare, in tutti e tre i casi il segnale sovracampionato segue molto fedelmente l'originale, ma al crescere della frequenza si discosta da questo, sempre a causa dell'attenuazione causata dal filtraggio con la funzione $sinc^2(x)$.
Infine, osserviamo il risultato dell'integrazione effettuata sul segnale sovracampionato: in Figura 2.51 è riportato (in blu) il valore di posizione calcolato integrando il segnale. In verde, inoltre, è riportata la posizione ottenuta integrando il segnale originale (in entrambi i casi dopo aver rimosso la gravità con l'algoritmo descritto in sezione 2.2): quello che si può notare è che i due valori differiscono di meno un millimetro. In figura, inoltre, è rappresentata (in rosso) anche la posizione che si otterrebbe dal segnale in cui i campioni sono realmente stati allineati a multipli del periodo di campionamento. Come si può osservare, tale segnale fornisce un valore di posizione identico a quello del segnale raw.
Da questi risultati possiamo concludere che assumere i tempi di arrivo dei sample forniti dal sensore come multipli di una frequenza di campionamento costante è un'ipotesi ragionevole.

2.8 Analisi dell'inclinazione

Si è rilevato nelle sezioni precedenti che l'accelerazione nei segnali registrati presenta una componente continua dopo la terminazione del moto. Si vuole, allora, capire meglio cosa causa questa componente, se è un problema di drift del sensore o se è causata dalla superficie su cui le misure sono state effettuate. A tale scopo sono state effettuate delle registrazioni dell'accelerazione con l'obiettivo di verificare la ripetibilità delle misure

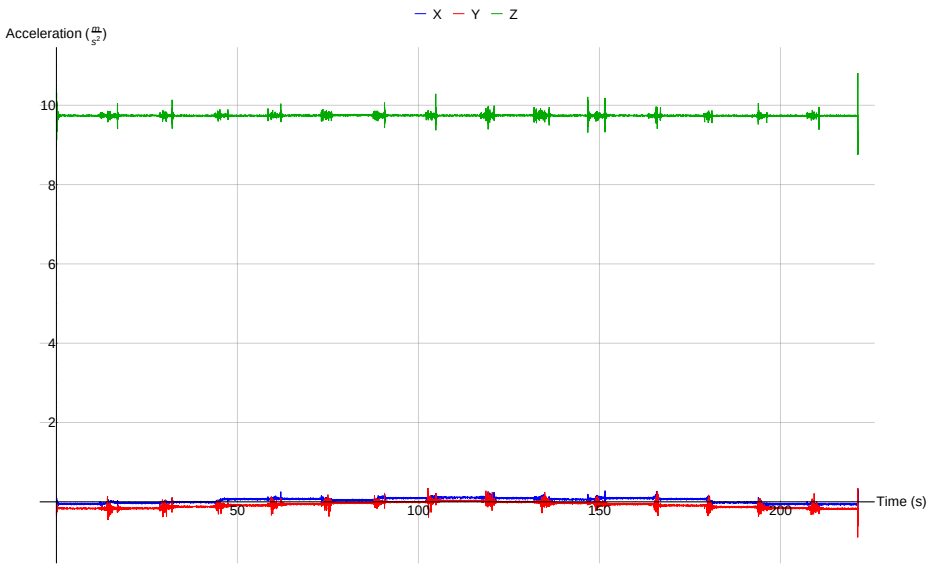


Figura 2.52: *Accelerazione del segnale ottenuto dal primo tipo di movimento considerato, in cui si sono percorsi 14cm.*

fornite dal sensore: una prima serie di misure è stata effettuata nella stessa zona dello stesso tavolo su cui sono state registrate le prime misure di accelerazione. Inoltre, si è utilizzata una guida per limitare il movimento lungo una singola direzione (corrispondente ancora una volta con l'asse y del sensore) e ci si è mossi in un verso ad intervalli di 2cm, fermandosi per vari secondi una volta finito il movimento. Dopo aver percorso una distanza di 14cm, si è invertito il verso del moto, muovendosi ancora ad intervalli di 2cm e, quindi, ripercorrendo le stesse posizioni in cui ci si era fermati durante il movimento nel primo verso. Nella seconda serie di misure acquisite, si è effettuato lo stesso tipo di movimento, ma in questo caso si è percorsa l'intera lunghezza del tavolo, effettuando movimenti di 10cm e coprendo la distanza di 110cm in un verso.

Vediamo, ora, i risultati ottenuti: In Figura 2.52 è riportato l'intero segnale acquisito per una misura del primo tipo di movimento considerato. In blu è riportata la coordinata x del segnale, in rosso la y ed in verde la z. Com'è possibile osservare, avendo percorso 14cm a passi di 2cm avanti ed indietro, nel segnale sono visibili 15 regioni in cui non è presente alcun movimento e l'unico segnale rilevato dal sensore è il vettore di gravità. Notiamo, inoltre, che, specialmente per le coordinate x ed y, nella prima metà del segnale dopo ogni movimento il valore medio dell'accelerazione

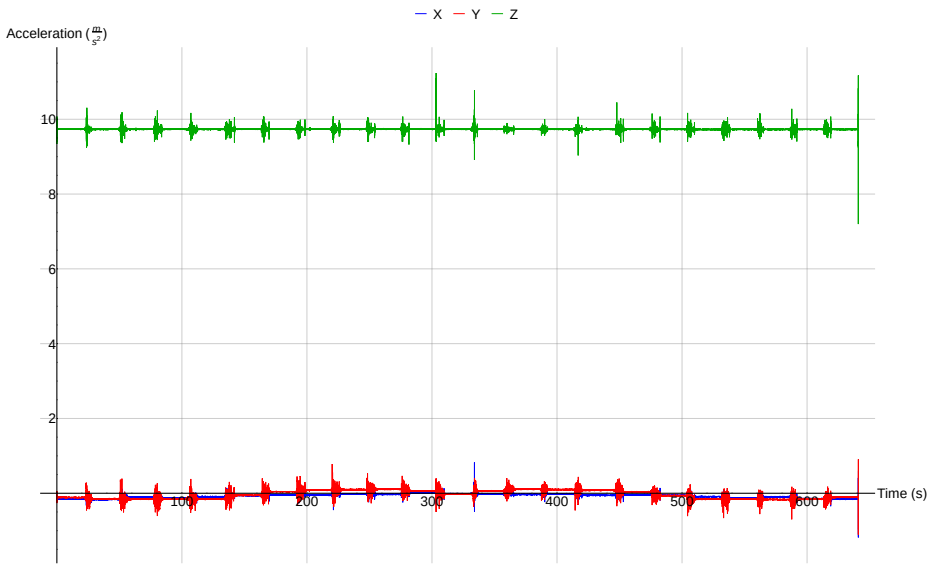


Figura 2.53: *Accelerazione ottenuta dal secondo movimento considerato, in cui si è percorsa l'intera lunghezza del tavolo.*

nella coordinata risulta avere un valore superiore rispetto a prima del movimento. Questo succede fino al raggiungimento della regione centrale, in cui avviene il cambio del verso di movimento: dopo questa regione, il valore medio delle componenti del segnale risultano avere intensità minore alla fine del movimento rispetto che all'inizio. Questo avviene in quanto nella seconda metà del segnale si sta ripercorrendo al contrario la strada fatta precedentemente, quindi, come ci si aspetta nel caso l'ipotesi di piano inclinato sia verificata, se all'andata l'accelerazione ha subito un incremento, muovendo nel senso inverso essa subisce un decremento. Osserviamo, infine, che considerando un singolo movimento, questo ha una forma molto simile alle registrazioni dell'accelerazione studiate precedentemente: ad una prima fase di accelerazione nulla segue il moto, e poi una seconda regione in cui è presente una sola componente continua, diversa da quella precedente il moto. In Figura 2.53, invece, è riportato l'intero segnale acquisito con il secondo tipo di movimento considerato, svolto su tutta la lunghezza del tavolo. In blu è riportata la coordinata x del segnale, in rosso la y ed in verde la z. È possibile notare come in questo caso l'andamento del valore medio delle componenti negli intervalli di immobilità segua un andamento più complesso rispetto al caso precedente. Vale anche in questo caso, tuttavia, la specularità del segnale rispetto all'intervallo centrale, per

cui il valore medio del segnale in una posizione durante la fase di "ritorno" assume un valore simile a quello registrato durante la fase di "andata".

Tabella 2.2: *Vettori di gravità, nelle regioni di condizione di immobilità e per le varie misure effettuate.*

	Misura 1	Misura 2	Misura 3
Regione 1	$\begin{pmatrix} -0.0525722 \\ -0.166272 \\ 9.7393 \end{pmatrix}$	$\begin{pmatrix} -0.0595648 \\ -0.172088 \\ 9.73276 \end{pmatrix}$	$\begin{pmatrix} -0.0760178 \\ -0.168217 \\ 9.73224 \end{pmatrix}$
Regione 2	$\begin{pmatrix} -0.0291784 \\ -0.166009 \\ 9.7414 \end{pmatrix}$	$\begin{pmatrix} -0.0621043 \\ -0.158697 \\ 9.73187 \end{pmatrix}$	$\begin{pmatrix} -0.0446148 \\ -0.158757 \\ 9.72809 \end{pmatrix}$
Regione 3	$\begin{pmatrix} -0.0104076 \\ -0.12057 \\ 9.73858 \end{pmatrix}$	$\begin{pmatrix} 0.00557868 \\ -0.130841 \\ 9.73386 \end{pmatrix}$	$\begin{pmatrix} -0.00198536 \\ -0.128223 \\ 9.73379 \end{pmatrix}$
Regione 4	$\begin{pmatrix} 0.065948 \\ -0.089572 \\ 9.739 \end{pmatrix}$	$\begin{pmatrix} 0.0674449 \\ -0.0926006 \\ 9.73204 \end{pmatrix}$	$\begin{pmatrix} 0.073468 \\ -0.0954 \\ 9.73161 \end{pmatrix}$
Regione 5	$\begin{pmatrix} 0.0752723 \\ -0.059812 \\ 9.74247 \end{pmatrix}$	$\begin{pmatrix} 0.097836 \\ -0.059436 \\ 9.73404 \end{pmatrix}$	$\begin{pmatrix} 0.087312 \\ -0.06342 \\ 9.73486 \end{pmatrix}$
Regione 6	$\begin{pmatrix} 0.041856 \\ -0.023572 \\ 9.74754 \end{pmatrix}$	$\begin{pmatrix} 0.0634956 \\ -0.0238084 \\ 9.73395 \end{pmatrix}$	$\begin{pmatrix} 0.075764 \\ -0.02506 \\ 9.73382 \end{pmatrix}$
Regione 7	$\begin{pmatrix} 0.0915 \\ -0.00384 \\ 9.74367 \end{pmatrix}$	$\begin{pmatrix} 0.091104 \\ -0.00144 \\ 9.73625 \end{pmatrix}$	$\begin{pmatrix} 0.0938204 \\ -0.00195095 \\ 9.73414 \end{pmatrix}$
Regione 8	$\begin{pmatrix} 0.109286 \\ 0.0271691 \\ 9.74205 \end{pmatrix}$	$\begin{pmatrix} 0.109627 \\ 0.0309569 \\ 9.73656 \end{pmatrix}$	$\begin{pmatrix} 0.128088 \\ 0.0320687 \\ 9.73431 \end{pmatrix}$
Regione 9	$\begin{pmatrix} 0.0902781 \\ -0.00298601 \\ 9.74441 \end{pmatrix}$	$\begin{pmatrix} 0.093844 \\ -0.0011 \\ 9.73562 \end{pmatrix}$	$\begin{pmatrix} 0.0925186 \\ -0.00464048 \\ 9.7355 \end{pmatrix}$

Tabella 2.2: *Vettori di gravità, nelle regioni di condizione di immobilità e per le varie misure effettuate.*

Regione 10	$\begin{pmatrix} 0.0598439 \\ -0.028995 \\ 9.7414 \end{pmatrix}$	$\begin{pmatrix} 0.0700019 \\ -0.0257539 \\ 9.7344 \end{pmatrix}$	$\begin{pmatrix} 0.061294 \\ -0.0274325 \\ 9.73194 \end{pmatrix}$
Regione 11	$\begin{pmatrix} 0.0907251 \\ -0.0595099 \\ 9.74203 \end{pmatrix}$	$\begin{pmatrix} 0.0805586 \\ -0.0610319 \\ 9.73565 \end{pmatrix}$	$\begin{pmatrix} 0.0542665 \\ -0.0625309 \\ 9.73591 \end{pmatrix}$
Regione 12	$\begin{pmatrix} 0.0693481 \\ -0.0926461 \\ 9.74165 \end{pmatrix}$	$\begin{pmatrix} 0.0763686 \\ -0.0920601 \\ 9.73586 \end{pmatrix}$	$\begin{pmatrix} 0.0469291 \\ -0.0968958 \\ 9.73789 \end{pmatrix}$
Regione 13	$\begin{pmatrix} -0.0174696 \\ -0.130679 \\ 9.73878 \end{pmatrix}$	$\begin{pmatrix} 0.00140262 \\ -0.129728 \\ 9.7335 \end{pmatrix}$	$\begin{pmatrix} 0.016784 \\ -0.135096 \\ 9.73168 \end{pmatrix}$
Regione 14	$\begin{pmatrix} -0.0589508 \\ -0.159012 \\ 9.73743 \end{pmatrix}$	$\begin{pmatrix} -0.0540911 \\ -0.15434 \\ 9.73044 \end{pmatrix}$	$\begin{pmatrix} -0.0539476 \\ -0.157246 \\ 9.72948 \end{pmatrix}$
Regione 15	$\begin{pmatrix} -0.0625986 \\ -0.172001 \\ 9.73684 \end{pmatrix}$	$\begin{pmatrix} -0.0759361 \\ -0.16862 \\ 9.73155 \end{pmatrix}$	$\begin{pmatrix} -0.071469 \\ -0.169362 \\ 9.73044 \end{pmatrix}$

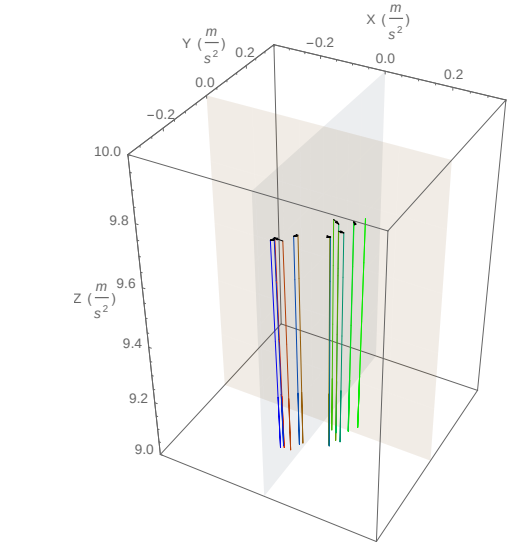
Analizziamo più in dettaglio, allora, il valore medio dell'accelerazione negli intervalli di immobilità: in queste regioni, dato che non è presente movimento, l'unico segnale registrato dal sensore è il vettore di gravità, oltre al rumore, sempre presente. Per ridurre l'effetto del rumore, quindi, si è calcolato il vettore gravità come valore medio del segnale nell'intervallo di tempo in cui il dispositivo è immobile. In Tabella 2.2 sono riportati i valori dei vettori ottenuti: ogni colonna corrisponde ad uno degli intervalli di immobilità nel segnale, numerati in modo crescente a partire dal primo presente nel segnale. In ogni riga, invece, è riportato il numero della rilevazione effettuata: sono, infatti, stati registrati più movimenti allo scopo di ottenere dati più affidabili. In Tabella 2.3, invece, sono mostrate le differenze tra i vettori di gravità ottenuti in una stessa posizione del movimento "all'andata" ed al "ritorno": la differenza su ogni coordinata del vettore assume la prima cifra significativa nella seconda o terza cifra decimale, mentre, nel caso ideale, essa dovrebbe essere nulla. Osservando, inoltre, il corrispondente errore percentuale (mostrato in Tabella 2.4 ed ot-

	Misura 1	Misura 2	Misura 3
Regione 1	$\begin{pmatrix} -0.0100265 \\ -0.00572884 \\ -0.00246374 \end{pmatrix}$	$\begin{pmatrix} -0.0163713 \\ 0.00346835 \\ -0.00120299 \end{pmatrix}$	$\begin{pmatrix} 0.00454879 \\ -0.00114417 \\ -0.00180593 \end{pmatrix}$
Regione 2	$\begin{pmatrix} -0.0297724 \\ 0.0069966 \\ -0.00397071 \end{pmatrix}$	$\begin{pmatrix} 0.0080132 \\ 0.00435743 \\ -0.00142323 \end{pmatrix}$	$\begin{pmatrix} -0.00933279 \\ 0.00151095 \\ 0.00138737 \end{pmatrix}$
Regione 3	$\begin{pmatrix} -0.007062 \\ -0.0101087 \\ 0.000192361 \end{pmatrix}$	$\begin{pmatrix} -0.00417607 \\ 0.00111256 \\ -0.000357618 \end{pmatrix}$	$\begin{pmatrix} 0.0187694 \\ -0.00687284 \\ -0.00211745 \end{pmatrix}$
Regione 4	$\begin{pmatrix} 0.00340007 \\ -0.00307415 \\ 0.00264314 \end{pmatrix}$	$\begin{pmatrix} 0.00892372 \\ 0.000540491 \\ 0.00381579 \end{pmatrix}$	$\begin{pmatrix} -0.0265389 \\ -0.0014958 \\ 0.00628339 \end{pmatrix}$
Regione 5	$\begin{pmatrix} 0.0154529 \\ 0.00030202 \\ -0.00044056 \end{pmatrix}$	$\begin{pmatrix} -0.0172774 \\ -0.00159588 \\ 0.0016113 \end{pmatrix}$	$\begin{pmatrix} -0.0330455 \\ 0.000889133 \\ 0.00105443 \end{pmatrix}$
Regione 6	$\begin{pmatrix} 0.0179879 \\ -0.00542303 \\ -0.00613592 \end{pmatrix}$	$\begin{pmatrix} 0.00650624 \\ -0.00194549 \\ 0.000454483 \end{pmatrix}$	$\begin{pmatrix} -0.01447 \\ -0.0023725 \\ -0.00187258 \end{pmatrix}$
Regione 7	$\begin{pmatrix} -0.00122193 \\ 0.000853987 \\ 0.000737966 \end{pmatrix}$	$\begin{pmatrix} 0.00274 \\ 0.00034 \\ -0.000627936 \end{pmatrix}$	$\begin{pmatrix} -0.00130187 \\ -0.00268953 \\ 0.00135518 \end{pmatrix}$

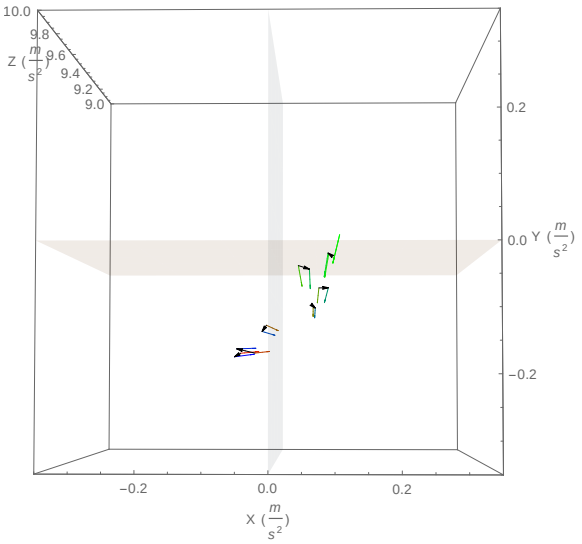
Tabella 2.3: *Differenze tra i vettori di gravità della stessa regione, letti "all'andata" ed "al ritorno".*

tenuto dividendo la differenza per il valore del primo vettore) possiamo notare come l'errore sulla coordinata z risulti molto basso, mentre per le coordinate x ed y l'errore è elevato e poco consistente. Infatti la differenza assoluta appena vista, anche se è nell'ordine del centesimo di $\frac{m}{s^2}$, è di entità paragonabile al valore assunto dal vettore.

Vediamo, quindi, in Figura 2.54 i vettori gravità rappresentati nello spazio: ogni asse rappresenta la corrispondente coordinata del vettore. Per maggiore chiarezza, inoltre, non è stato visualizzato il vettore nella sua interezza, ma si è mostrato solo il suo estremo opposto all'origine, in quanto essa è comune a tutti i vettori, nonostante nella realtà questi siano posizionati in punti diversi dello spazio. Possiamo osservare, così, la variazione del vettore gravità nel tempo: il primo vettore è rappresentato in rosso e man mano che ci si avvicina alla metà del segnale ed al vettore centrale, il colore assume una tonalità sempre più vicina la verde. Dal vettore centrale (di

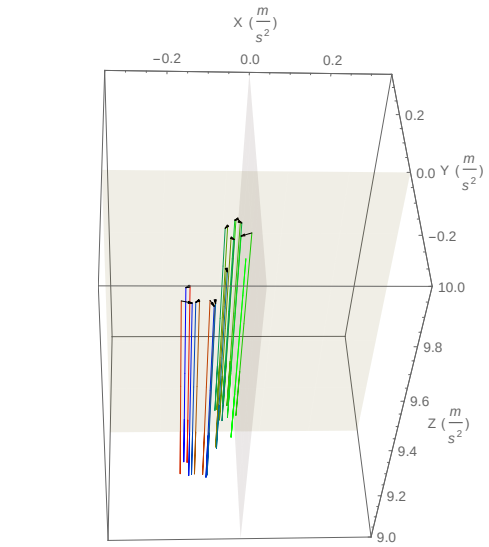


(a) *Vista di lato.*

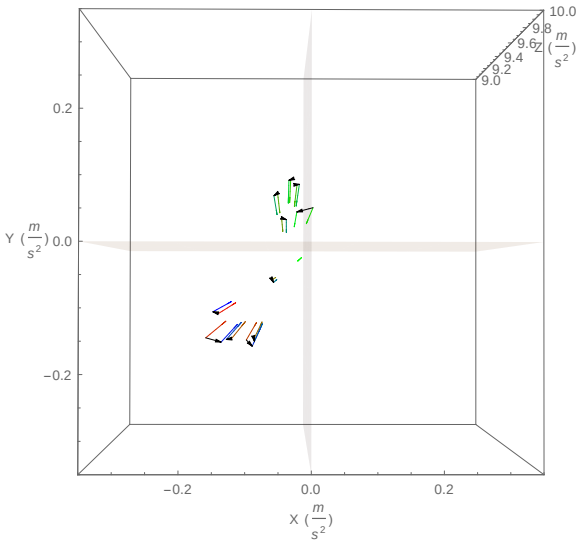


(b) *Vista dall'alto.*

Figura 2.54: *Vettori di gravità ottenuti con il primo movimento considerato. La variazione del vettore è indicata con il cambio nella tonalità del colore: dal rosso del primo vettore si arriva al verde di quello centrale, fino al blu dell'ultimo.*



(a) *Vista di lato.*



(b) *vista dall'alto*

Figura 2.55: *Vettori di gravità ottenuti con il secondo movimento considerato. La variazione del vettore è indicata con il cambio nella tonalità del colore: dal rosso del primo vettore si arriva al verde di quello centrale, fino al blu dell'ultimo.*

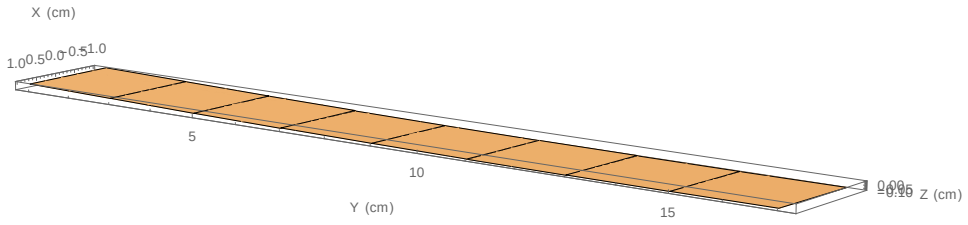
	Misura 1	Misura 2	Misura 3
Regione 1	$\begin{pmatrix} 19.0718 \\ 3.44546 \\ -0.0252969 \end{pmatrix}$ %	$\begin{pmatrix} 27.4848 \\ -2.01544 \\ -0.0123602 \end{pmatrix}$ %	$\begin{pmatrix} -5.98385 \\ 0.680175 \\ -0.0185561 \end{pmatrix}$ %
Regione 2	$\begin{pmatrix} 102.036 \\ -4.2146 \\ -0.0407612 \end{pmatrix}$ %	$\begin{pmatrix} -12.9028 \\ -2.74575 \\ -0.0146244 \end{pmatrix}$ %	$\begin{pmatrix} 20.9186 \\ -0.951735 \\ 0.0142615 \end{pmatrix}$ %
Regione 3	$\begin{pmatrix} 67.8544 \\ 8.38406 \\ 0.00197525 \end{pmatrix}$ %	$\begin{pmatrix} -74.8575 \\ -0.850314 \\ -0.00367396 \end{pmatrix}$ %	$\begin{pmatrix} -945.388 \\ 5.36006 \\ -0.0217536 \end{pmatrix}$ %
Regione 4	$\begin{pmatrix} 5.15568 \\ 3.43204 \\ 0.0271397 \end{pmatrix}$ %	$\begin{pmatrix} 13.2311 \\ -0.58368 \\ 0.0392086 \end{pmatrix}$ %	$\begin{pmatrix} -36.123 \\ 1.56793 \\ 0.0645668 \end{pmatrix}$ %
Regione 5	$\begin{pmatrix} 20.5293 \\ -0.504949 \\ -0.00452206 \end{pmatrix}$ %	$\begin{pmatrix} -17.6596 \\ 2.68504 \\ 0.0165532 \end{pmatrix}$ %	$\begin{pmatrix} -37.8476 \\ -1.40198 \\ 0.0108315 \end{pmatrix}$ %
Regione 6	$\begin{pmatrix} 42.9756 \\ 23.0062 \\ -0.0629484 \end{pmatrix}$ %	$\begin{pmatrix} 10.2468 \\ 8.17147 \\ 0.00466905 \end{pmatrix}$ %	$\begin{pmatrix} -19.0988 \\ 9.4673 \\ -0.0192379 \end{pmatrix}$ %
Regione 7	$\begin{pmatrix} -1.33545 \\ -22.2392 \\ 0.0075738 \end{pmatrix}$ %	$\begin{pmatrix} 3.00755 \\ -23.6111 \\ -0.00644947 \end{pmatrix}$ %	$\begin{pmatrix} -1.38761 \\ 137.857 \\ 0.0139219 \end{pmatrix}$ %

Tabella 2.4: *Differenze percentuali tra i vettori di gravità della stessa regione, letti "all'andata" ed "al ritorno".*

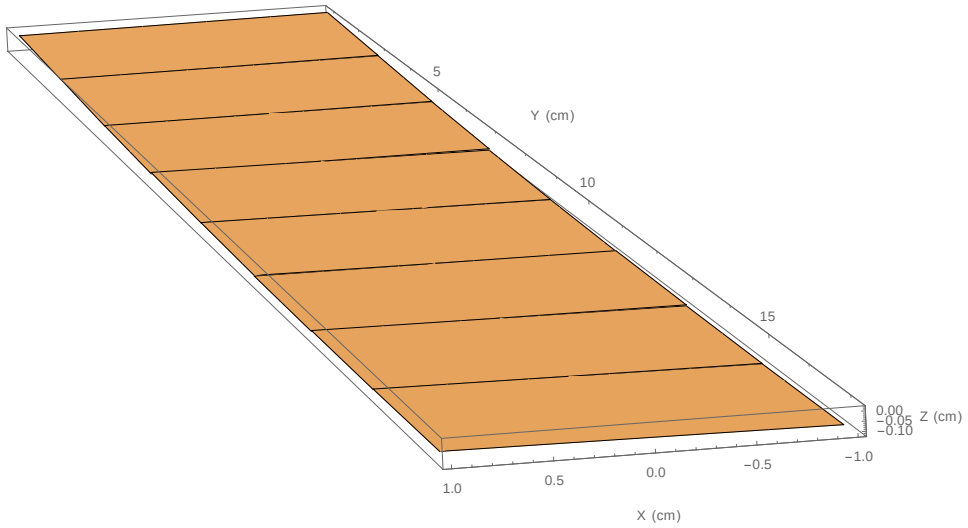
colore verde, quindi) inizia il movimento in verso opposto, ed i cui vettori sono rappresentati dal colore di tonalità tendente al blu. Per identificare le coppie di vettori associati alla stessa posizione, inoltre, una freccia è stata disegnata dal vettore rilevato nella fase di "andata", e con verso diretto sul vettore della fase di "ritorno". Tramite due piani colorati, infine, sono stati visualizzati i piani corrispondenti ad una coordinata nulla. Sia la fase di "andata" che quella di "ritorno" consistentemente indicano che il vettore gravità subisce una variazione lungo il piano verticale inclinato di 45° in senso orario rispetto al piano di coordinata x nulla. In Figura 2.55, invece, sono riportati i vettori di gravità ottenuti con il secondo tipo di movimento considerato: anche in questo caso è stata mostrata solamente la regione delle punte dei vettori, e la variazione nel tempo è stata rappresentata dalla variazione del colore dal rosso al verde per il vettore della regione centrale, fino al blu per l'ultimo vettore. Notiamo che per questo movimento, avve-

nuto su tutta la lunghezza del tavolo, la variazione del vettore normale è più complessa del caso precedente (come osservato anche da Figura 2.53). In particolare, si può notare come il vettore vari su un piano di poco inclinato rispetto a quello formato dalla coordinata x nulla, formando una curva simile ad una "S".

Osserviamo, infine, una ricostruzione della superficie del piano effettuata con i vettori normali: questi sono stati trovati calcolando la rotazione che allinea il vettore di gravità all'asse z (nel sistema di coordinate del sensore), ed applicando questa stessa rotazione al versore dell'asse z, nel sistema di coordinate del mondo. Si ottiene così il vettore normale al dispositivo, che coincide al vettore normale alla superficie, sotto l'ipotesi di dispositivo parallelo a questa. Da ogni vettore normale, quindi, è stato calcolato il piano perpendicolare ad esso e successivamente, conoscendo la direzione e l'entità di ogni movimento eseguito, si è ricostruita l'origine di ogni vettore: la posizione x è stata fissata a zero (in quanto il moto è avvenuto interamente lungo l'asse y), mentre la posizione y è stata incrementata per ogni vettore di una quantità pari allo spostamento noto. Per la posizione sulla coordinata z, infine, si è supposta una variazione lineare della superficie dall'origine del vettore normale (ovvero si è approssimata la superficie reale con il piano calcolato dal vettore normale). In questo modo si è approssimato l'incremento sull'asse z tra due vettori consecutivi come l'incremento in z lungo la direzione y su ciascuna delle due metà dei piani calcolati dai vettori in questione. Possiamo osservare, quindi, in Figura 2.56 la ricostruzione effettuata sul primo movimento considerato: com'è possibile notare, la distanza percorsa nella direzione y è esattamente pari a 14cm (l'origine del movimento corrisponde alla coordinata di 2cm nel grafico), mentre lungo z si percorre circa un millimetro (nonostante questo valore sia un'approssimazione della variazione reale). Osserviamo, inoltre, che la superficie ricostruita è consistente con quanto notato da Figura 2.54. In Figura 2.57, invece, è mostrata la ricostruzione eseguita sul secondo movimento considerato: anche in questo caso la distanza percorsa nella direzione y corrisponde al valore teorico percorso di 110cm, in quanto ricostruito manualmente (in questo caso l'origine corrisponde alla coordinata 10cm). Osserviamo che anche in questo caso, come nel precedente, la ricostruzione effettuata è consistente con la variazione dei vettori normali mostrata in Figura 2.55. Nonostante non ci sia stato possibile verificare la correttezza

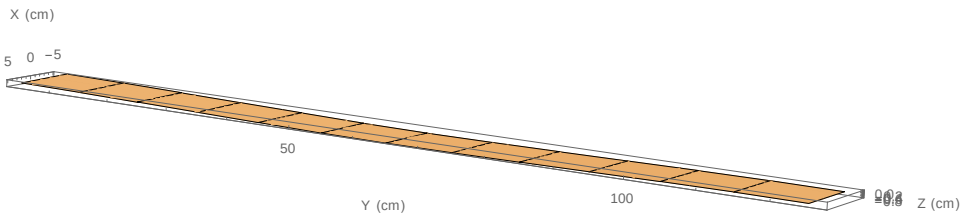


(a) *Prima vista.*

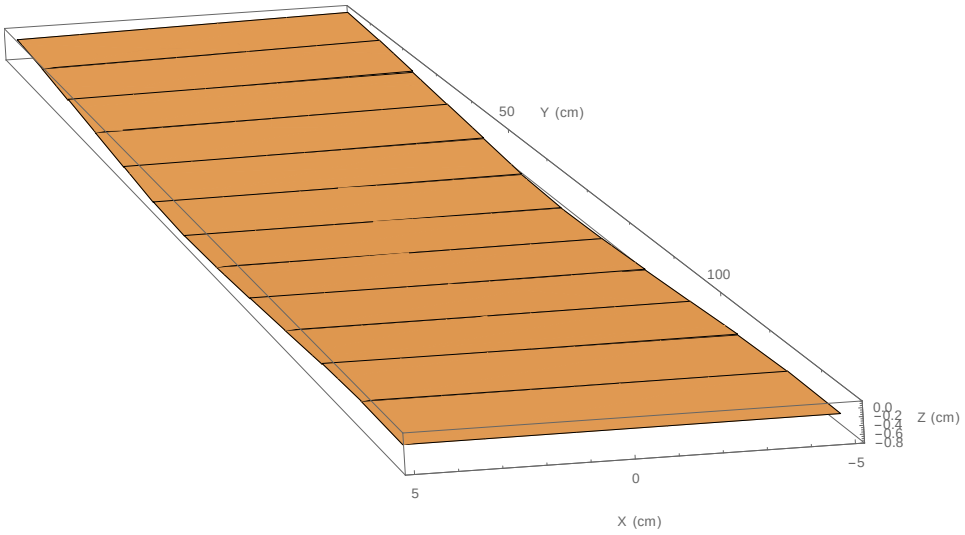


(b) *Seconda vista.*

Figura 2.56: *Ricostruzione della superficie nel caso del primo movimento considerato.*



(a) *Prima vista.*



(b) *Seconda vista.*

Figura 2.57: *Ricostruzione della superficie nel caso del secondo movimento considerato.*

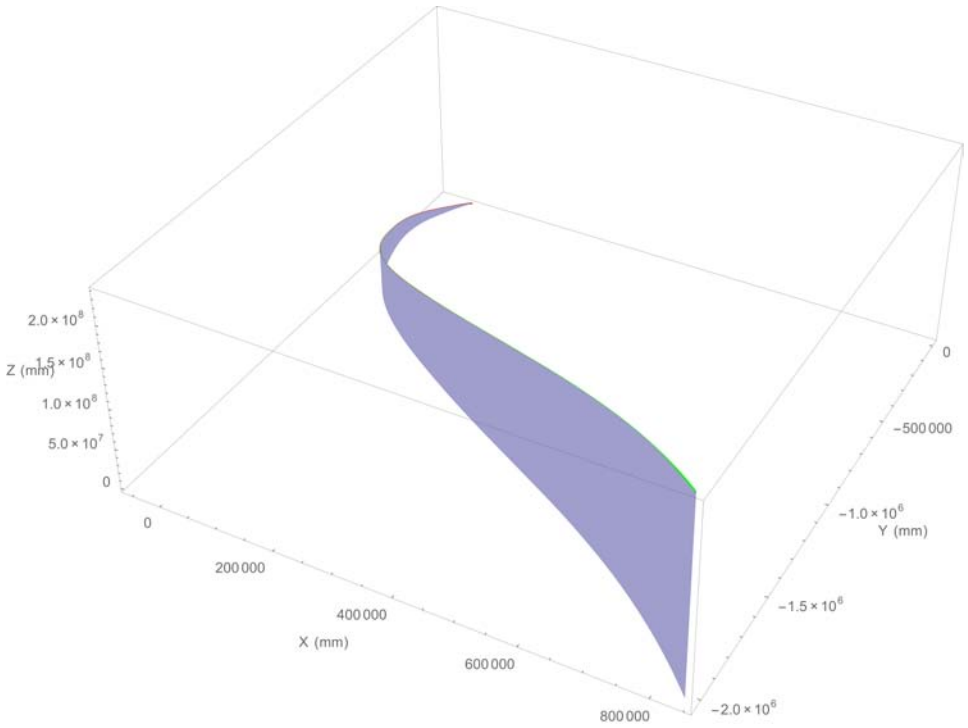
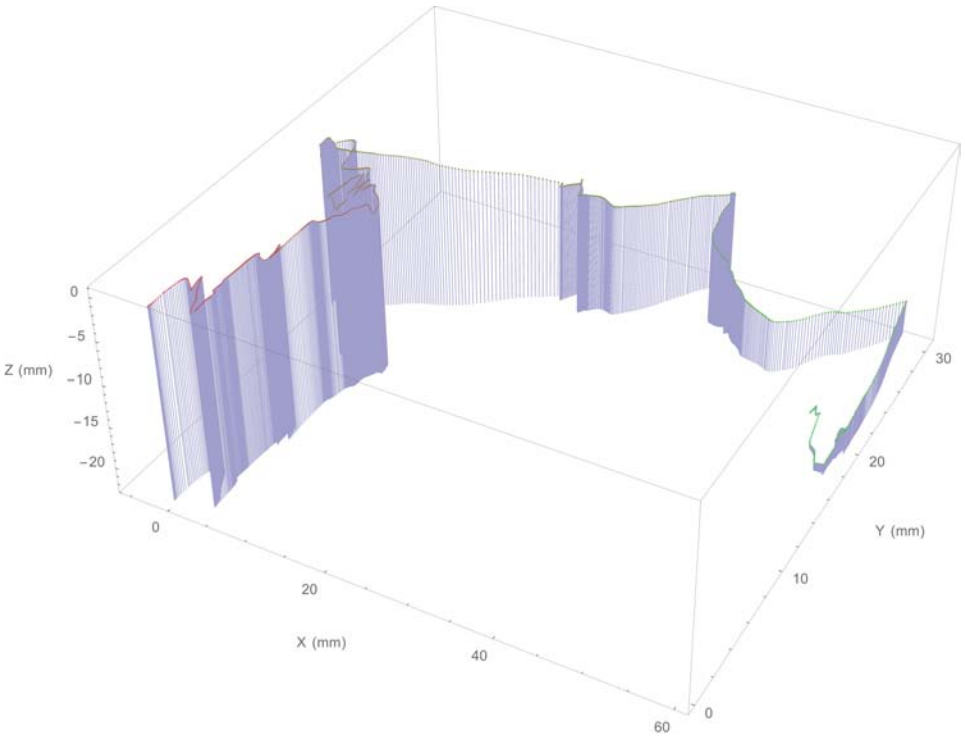


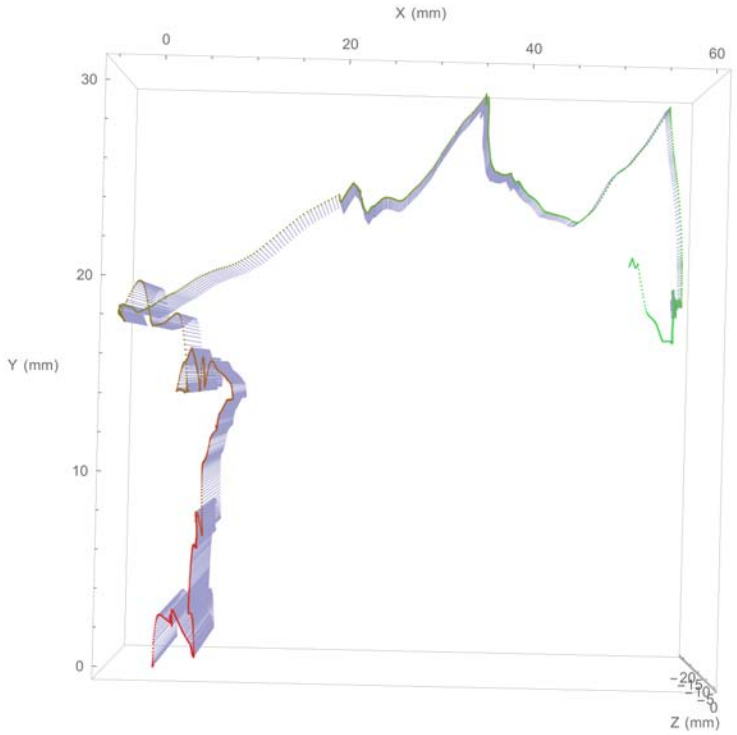
Figura 2.58: *Posizione ottenuta integrando il segnale di Figura 2.52 senza preelaborazione.*

della ricostruzione effettuata per l'impossibilità di rilevare la vera inclinazione della superficie, possiamo ragionevolmente concludere che il sensore non è affetto da drift, e che la componente continua presente alla fine dei segnali di spostamento studiati è dovuta ad una variazione dell'inclinazione della superficie di appoggio.

Consideriamo, ora, una seconda analisi effettuata sui segnali acquisiti per questa sezione e descritti precedentemente: in particolare, studiamo il risultato dell'integrazione del segnale al fine di verificare l'eventuale presenza di movimenti ricostruiti, ma incompatibili con il moto realmente avvenuto. Per prima cosa, quindi, analizziamo l'integrazione dell'intero segnale senza preelaborazioni: come algoritmo di integrazione utilizziamo l'algoritmo vettoriale descritto in sezione A.6 e come metodo adottiamo quello dei trapezi. In Figura 2.58 è mostrato il risultato ottenuto: ogni asse corrisponde alla rispettiva coordinata nel mondo reale, mentre la variazione nel tempo è indicata tramite la variazione del colore, dal rosso per il tempo zero fino al verde per l'ultimo campione. Com'è possibile osservare,

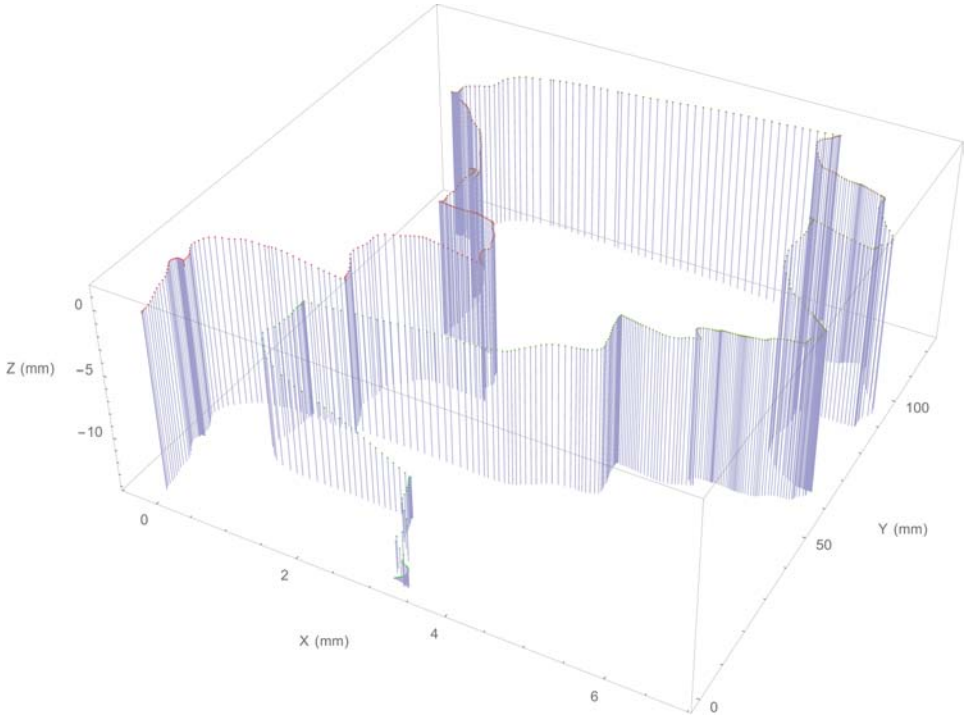


(a) *Vista di lato.*

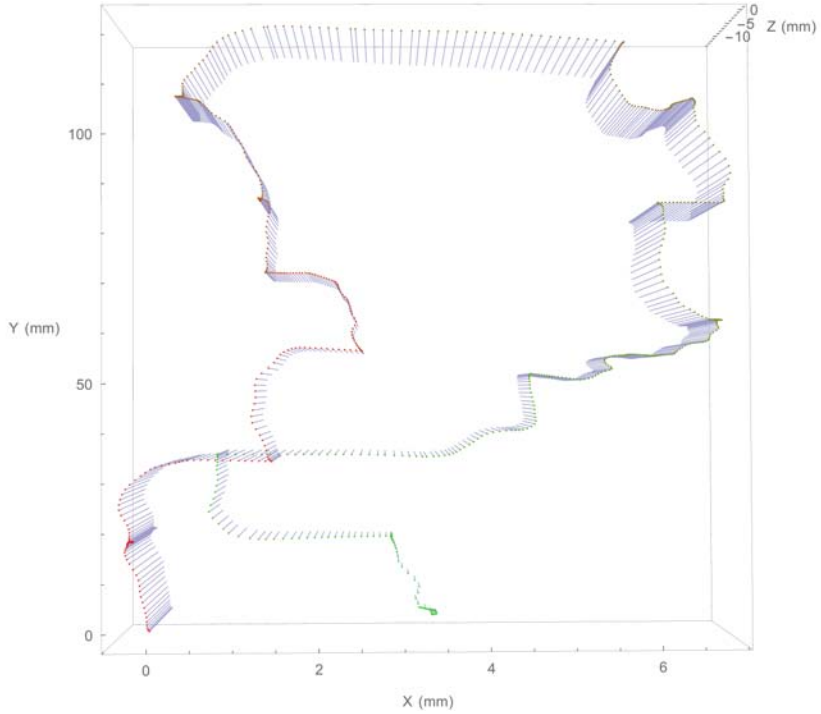


(b) *Vista dall'alto*

Figura 2.59: *Posizione ottenuta integrando i singoli segnali di movimento estratti manualmente dal segnale di Figura 2.52.*



(a) *Vista di lato.*



(b) *Vista dall'alto.*

Figura 2.60: *Posizione ottenuta integrando un segnale effettuato con le stesse modalità del primo considerato, ma con spostamenti veloci.*

nell'integrazione si è persa ogni informazione del movimento realmente eseguito: infatti su ogni asse il risultato di posizione finale è nell'ordine del chilometro. Non avendo rimosso la gravità dal segnale, ogni campione contribuisce all'integrazione, in quanto superiore come modulo al valore di soglia. Vengono contati nell'integrazione, quindi, anche tutti i campioni corrispondenti ai vettori gravità registrati negli intervalli di immobilità: in queste regioni i vettori presentano un valore medio non nullo anche sugli assi x ed y, oltre che sul z. Questo valore non nullo, quindi, quando integrato su un intervallo di circa 200 secondi, contribuisce significativamente al risultato dell'integrazione, tanto da portare al risultato visibile in figura. Non potendo utilizzare il segnale raw senza elaborazioni, si è deciso di estrarre manualmente gli intervalli di tempo in cui avviene il movimento ed integrare il segnale solamente in questi intervalli, accumulando i risultati ottenuti da ogni intervallo. Una volta estratto il singolo segnale di movimento, si è rimossa la componente continua con un procedimento simile a quello descritto in sezione A.1: avendo a disposizione l'informazione del vettore gravità prima e dopo il moto, si è supposta una variazione lineare dal primo vettore al secondo. Tramite questa ipotesi si è rimosso il vettore gravità sottraendo ad ogni campione la combinazione lineare dei due vettori, pesata per la posizione del sample nel segnale: in questo modo per i primi campioni del movimento ha più peso il vettore gravità che precede il moto, mentre per gli ultimi vale il viceversa. In Figura 2.59 è riportato il risultato di questa operazione: come prima gli assi corrispondono alle coordinate del mondo ed il tempo è rappresentato dalla variazione del colore, dal rosso al verde. Notiamo che ora la posizione ottenuta non raggiunge più valori elevati come nel caso precedente. Tuttavia, neanche in questo caso il moto ricostruito è compatibile con quello realmente avvenuto: in ciascuno degli assi, infatti, è riportato uno spostamento nell'ordine di pochi centimetri, addirittura di entità maggiore lungo l'asse x rispetto a quello y, dove è realmente avvenuto il moto. Ipotizziamo che questo sia dovuto alla modalità con cui il moto è stato effettuato: infatti, per realizzare uno spostamento preciso il movimento è avvenuto lentamente. In queste condizioni, tuttavia, il sensore fatica a fornire un buon segnale, in quanto esso viene largamente influenzato dal rumore.
Per verificare l'ipotesi, si è deciso quindi di ripetere la misura del primo tipo di movimento considerato, effettuando questa volta spostamenti rapidi,

a scapito della loro precisione. Dal segnale ottenuto, si sono nuovamente estratti manualmente gli intervalli in cui sono avvenuti i movimenti e, con la stessa procedura descritta precedentemente, si è rimossa la componente continua da ogni singolo segnale di movimento estratto. Si sono, quindi, integrati i segnali ed accumulata la posizione ottenuta: il risultato è visibile in Figura 2.60. Anche in questo caso, gli assi corrispondono alle coordinate del mondo reale ed il tempo è rappresentato dalla variazione di colore, dal rosso al verde. Com'è possibile notare, in questo caso il risultato di posizione è compatibile con il moto realmente avvenuto: infatti sulla coordinata y è riportato uno spostamento di circa 12cm, ovvero nell'ordine di grandezza di quello reale, mentre sugli altri assi lo spostamento calcolato è di alcuni millimetri, a causa di qualche accelerazione spuria presente sugli assi.

3 Analisi della fotocamera

Nel capitolo precedente abbiamo analizzato il segnale fornito dal sensore accelerometro, mettendo in luce alcune delle sue caratteristiche chiave per il nostro scopo. In questo capitolo, invece, ci focalizzeremo sulla fotocamera e le informazioni da lei fornite.
Ricordiamo che l'app finale che si vuole poter realizzare prevede che durante l'utilizzo l'utente tenga il dispositivo di fronte alla bocca, per simulare un vero flauto di pan, come mostrato a titolo d'esempio in Figura 3.1. Considerando che la maggior parte dei dispositivi moderni è fornita di due fotocamere, c'è la possibilità di scegliere quale utilizzare: una delle due fotocamere inquadrerà il petto dell'utente (quindi la sua maglia, un esempio è visibile in Figura 3.6), mentre la seconda mostrerà la scena dell'ambiente in cui l'utente si trova. In questo secondo caso, tuttavia, non è possibile effettuare alcuna assunzione sull'immagine acquisita, in quanto l'utente può utilizzare l'app in qualunque ambiente (per esempio una stanza di casa od un parco all'aperto). Inoltre gli oggetti inquadrati potrebbero essere molto distanti, per cui il loro spostamento risultare impercettibile, e tutto questo complica notevolmente le elaborazioni necessarie. Si è scelto, quindi, di utilizzare le immagini della maglia indossata dall'utente, facendo l'ipotesi che la distanza del petto dal dispositivo sia di circa 10cm. L'informazione disponibile fornita dalla fotocamera, quindi, è in generale l'immagine di una maglia che idealmente subisce un moto rigido di traslazione, ovvero il solo moto presente all'interno dell'immagine è una traslazione, applicata globalmente ad ogni punto. Un primo lavoro svolto su questo argomento è presentato in [15]: in questo lavoro, tuttavia, si è ignorata l'informazione di traslazione rigida e si è calcolato il movimento dell'immagine tramite flusso ottico. Per sfruttare l'intera informazione restituita, quindi, si è deciso di adottare un approccio differente, in particolare si è utilizzato l'algoritmo

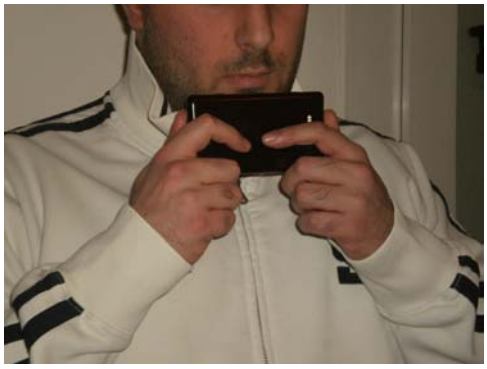


Figura 3.1: *Esempio della modalità d'uso desiderata per l'app.*

di registrazione di immagini "inverse compositional algorithm", che verrà descritto ed analizzato nel corso di questo capitolo.

3.1 Algoritmo di registrazione adottato

Vediamo, allora, l'algoritmo *Inverse compositional algorithm* adottato: esso è una variante dell'algoritmo presentato da Lucas e Kanade in [14]. L'idea alla base dell'algoritmo è di sfruttare l'informazione sul gradiente dell'intensità luminosa dell'immagine per indirizzare la ricerca della trasformazione di registrazione verso il valore ottimo, rispetto ad una metrica di errore. In particolare, l'algoritmo originale di Lucas-Kanade si basa su un approccio gradient descent ed incremento additivo dei parametri della trasformazione tramite un procedimento iterativo ed a partire da una stima iniziale di questi. Questo approccio, tuttavia, richiede di eseguire ad ogni iterazione una significativa quantità di calcoli matriciali, che rallentano l'esecuzione, per cui, dato che per il nostro obiettivo finale il tempo di esecuzione è un fattore importante, si è considerata la variante *inverse compositional*, che formula in maniera leggermente diversa il problema, rispetto alla versione "base", riuscendo ad ottenere prestazioni migliori.

3.1.1 Problema di registrazione di immagini

Definiamo formalmente, quindi, il problema di registrazione: sia I($\mathbf{x}$) una immagine a toni di grigio e T($\mathbf{x}$) un template contenuto in I($\mathbf{x}$) a meno di una deformazione $\mathbf{W}(\mathbf{x};\mathbf{p})$, dove $\mathbf{x} = (x, y)^T$ è il vettore colonna delle coordinate di un pixel. Quello che si vuole trovare è il vettore di parametri

p della funzione di deformazione **W**(**x**;**p**), tale che $T(\mathbf{x}) = I(\mathbf{W}(\mathbf{x};\mathbf{p}))\ \forall \mathbf{x} \in T$, dove **W**(**x**;**p**) è una funzione arbitrariamente complessa che descrive la deformazione subita dal template che si vuole allineare all'immagine. In altre parole, quello che si vuole determinare è il valore dei parametri di una deformazione (stabilita a priori) del template T, che mappi i suoi pixel nel sistema di coordinate dell'immagine I(**x**) in modo tale da allinearli con il minimo errore possibile.

3.1.2 Derivazione dell'algoritmo

Vediamo, allora, per prima cosa come si ottiene l'algoritmo *inverse compositional*, allo scopo di comprenderne le ipotesi alla base: l'algoritmo originale di Lucas-Kanade mira a minimizzare l'errore SSD

$$\sum_{\mathbf{x}\in T}\left[I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})\right]^2$$

Tuttavia questo è un problema di ottimizzazione non lineare, in quanto in generale il valore di I(**x**) non è lineare in **x**. Per semplificare il problema si suppone nota una stima dei parametri **p** e si minimizza iterativamente l'errore in Equazione 3.1 per la variazione dei parametri $\Delta\mathbf{p}$.

$$\sum_{\mathbf{x}\in T}\left[I(\mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p})) - T(\mathbf{x})\right]^2 \tag{3.1}$$

In questo modo, si riesce ad ottenere un algoritmo, che, come detto però, presenta delle inefficienze.
Per migliorare l'algoritmo, quindi, si formula il problema osservandolo da un'angolazione diversa: l'idea alla base della formulazione *inverse compositional* è di invertire i ruoli di immagine e template tramite un cambio di variabile. Il risultato è che l'algoritmo minimizza la funzione di errore mostrata in Equazione 3.2 rispetto a $\Delta\mathbf{p}$

$$\sum_{\mathbf{x}\in T}\left[T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))\right]^2 \tag{3.2}$$

Come l'algoritmo originale, quindi, anche la versione *inverse compositional* effettua l'importante assunzione di avere a disposizione una stima iniziale dei parametri della deformazione, che viene raffinata dall'algoritmo fino

alla minimizzazione dell'errore. Inoltre, altre due importanti assunzioni che emergono da Equazione 3.2 sono l'ipotesi che il template appaia completamente nell'immagine, senza che vi siano occlusioni, e l'assunzione di luminosità costante, ovvero che il template appaia nell'immagine con la stessa distribuzione di intensità presente nell'immagine template.
Proseguiamo, quindi, con la derivazione dell'algoritmo. Per prima cosa effettuiamo uno sviluppo in serie di Taylor su Equazione 3.2 per $\Delta\mathbf{p} \rightarrow 0$, ottenendo

$$\sum_{\mathbf{x}\in T}\left[T(\mathbf{W}(\mathbf{x};\mathbf{0})) + \nabla T(\mathbf{x}) J_{\mathbf{W}}(\mathbf{x};\mathbf{0})\Delta\mathbf{p} - I(\mathbf{W}(\mathbf{x};\mathbf{p}))\right]^2,$$

dove $\nabla T(\mathbf{x}) = (\frac{\partial T}{\partial x}(\mathbf{x}), \frac{\partial T}{\partial y}(\mathbf{x}))$ è il gradiente di T(**x**) e $J_{\mathbf{W}}(\mathbf{x};\mathbf{p})$ è la matrice jacobiana di **W**(**x**; **p**).
Questa espressione rappresenta l'errore di allineamento, che vogliamo minimizzare: a questo scopo calcoliamo la derivata parziale rispetto a $\Delta\mathbf{p}$ e ne prendiamo la trasposta, ottenendo l'espressione

$$2\sum_{\mathbf{x}\in T}\left[\nabla T(\mathbf{x}) J_{\mathbf{W}}(\mathbf{x};\mathbf{0})\right]^T \cdot \left[T(\mathbf{x}) + \nabla T(\mathbf{x}) J_{\mathbf{W}}(\mathbf{x};\mathbf{0})\Delta\mathbf{p} - I(\mathbf{W}(\mathbf{x};\mathbf{p}))\right]$$

in cui è stata fatta l'ipotesi che **W**(**x**; **0**) sia pari alla trasformazione identità. Ponendo l'espressione appena trovata pari a zero e risolvendo per $\Delta\mathbf{p}$ si ottiene infine

$$\Delta\mathbf{p} = H^{-1}\sum_{\mathbf{x}\in T}\left[\nabla T(\mathbf{x}) J_{\mathbf{W}}(\mathbf{x};\mathbf{0})\right]^T [I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})] \tag{3.3}$$

dove H è l'approssimazione della matrice hessiana, calcolata come

$$H = \sum_{\mathbf{x}\in T}\left[\nabla T(\mathbf{x}) J_{\mathbf{W}}(\mathbf{x};\mathbf{0})\right]^T \left[\nabla T(\mathbf{x}) J_{\mathbf{W}}(\mathbf{x};\mathbf{0})\right] \tag{3.4}$$

Riportiamo, infine, in algoritmo 3.1 lo pseudo-codice dell'algoritmo ottenuto. Per prima cosa vengono precalcolati il gradiente del template, la jacobiana della trasformazione, il prodotto tra queste due entità e la matrice H. Dopodiché, l'algoritmo iterativamente utilizza la trasformazione corrente per effettuare la deformazione dell'immagine I(**x**), con cui calcola l'immagine errore $I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})$, usata a sua volta per determinare la variazione dei parametri con cui aggiornare la trasformazione corrente per

l'iterazione successiva. L'algoritmo, infine, termina quando si è raggiunto un numero massimo di iterazioni o la variazione dei parametri scende sotto una soglia fornita come input all'algoritmo.

Algoritmo 3.1: Pseudo-codice per l'algoritmo *inverse compositional*

input : Immagine I(**x**)
Template T(**x**)
Soglia di errore ϵ
Numero massimo di iterazioni N
output : Funzione di trasformazione $\mathbf{W}(\mathbf{x};\mathbf{p})$

Calcola $\nabla T(\mathbf{x})$ a partire da T(**x**)
Calcola $J_{\mathbf{W}}(\mathbf{X};\mathbf{0})$
Calcola $\nabla T(\mathbf{x}) \cdot J_{\mathbf{W}}(\mathbf{X};\mathbf{0})$
Calcola H tramite Equazione 3.4
$numIterations \leftarrow 0$
do
  Calcola $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$
  Calcola $I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})$
  Calcola $\Delta\mathbf{p}$ tramite Equazione 3.3
  Aggiorna la funzione $\mathbf{W}(\mathbf{x};\mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}$
while $\|\Delta\boldsymbol{p}\| > \epsilon$ *and* $numIterations < N$

3.1.3 Varianti dell'algoritmo implementate

Abbiamo visto nella sezione precedente lo schema generale dell'algoritmo *inverse compositional* considerato, valido per qualsiasi trasformazione che soddisfi la proprietà di gruppo. Presentiamo ora i dettagli dell'implementazione realizzata. Essa è stata effettuata su PC tramite un programma scritto appositamente, il linguaggio che è stato adottato è il C++ (standard 2011), mentre per lo svolgimento dei calcoli matriciali si è utilizzata la libreria Eigen versione 3. Per la gestione e manipolazione delle immagini, come il caricamento da file ed applicazione della distorsione, invece, si è utilizzata la libreria OpenCV versione 2.4. L'algoritmo è stato realizzato adottando due diverse trasformazioni. In una prima implementazione è stato adottato un modello affine: questa trasformazione è caratterizzata da

sei parametri e le sue equazioni sono le seguenti

$$W_{aff}(x, y;\ a_{00}, a_{10}, a_{01}, a_{11}, t_x, t_y) = \begin{cases} x' = a_{00}\, x + a_{01}\, y + t_x \\ y' = a_{10}\, x + a_{11}\, y + t_y \end{cases}$$

Adottando, quindi, come vettore dei parametri $(a_{00},\ a_{10},\ a_{01},\ a_{11},\ t_x,\ t_y)$, la matrice jacobiana che si ottiene è

$$J_{W_{aff}}(x, y;\ a_{00}, a_{10}, a_{01}, a_{11}, t_x, t_y) = \begin{pmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{pmatrix}$$

Si è deciso di adottare questa trasformazione in quanto essa è abbastanza generale da approssimare la deformazione generata da una rototraslazione della fotocamera (per piccoli movimenti) e mantenere allo stesso tempo la proprietà di linearità. La trasformazione generale (omografia), infatti, non è lineare nelle coordinate dei pixel e risulta più computazionalmente impegnativa.
La trasformazione considerata per la seconda implementazione, invece, è una semplice traslazione: questa trasformazione è caratterizzata da due soli parametri e la sua funzione è la seguente:

$$W_{trasl}(x, y;\ t_x, t_y) = \begin{cases} x' = x + t_x \\ y' = y + t_y \end{cases}$$

Utilizzando il vettore dei parametri $(t_x,\ t_y)$, invece, si ottiene in questo caso che la matrice jacobiana è pari alla matrice identità:

$$J_{W_{trasl}}(x, y;\ t_x, t_y) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Com'è possibile notare, questa funzione è estremamente semplice: essa, tuttavia, caratterizza completamente il moto che siamo interessati a trattare, in quanto un movimento di traslazione della fotocamera si traduce in una traslazione nelle immagini generate.
Una terza versione implementata, infine, è una generalizzazione dell'algoritmo presentato, che permette di elaborare le immagini a colori: in particolare, le immagini vengono considerate non più come funzioni che associano ad ogni coordinata dei pixel un valore scalare, ma come funzioni vettoriali

che forniscono un vettore di colore per ogni pixel. Segue, quindi, che nelle formule presentate nella sezione precedente al posto del gradiente dell'immagine viene calcolata la sua jacobiana, o, vista in un'altra luce, il gradiente di ogni canale di colore dell'immagine:

$$J_T(\mathbf{x}) = \begin{pmatrix} \frac{\partial T_R}{\partial x}(\mathbf{x}) & \frac{\partial T_R}{\partial y}(\mathbf{x}) \\ \frac{\partial T_G}{\partial x}(\mathbf{x}) & \frac{\partial T_G}{\partial y}(\mathbf{x}) \\ \frac{\partial T_B}{\partial x}(\mathbf{x}) & \frac{\partial T_B}{\partial y}(\mathbf{x}) \end{pmatrix} = \begin{pmatrix} \nabla T_R(\mathbf{x}) \\ \nabla T_G(\mathbf{x}) \\ \nabla T_B(\mathbf{x}) \end{pmatrix}$$

Il resto delle formule presentate, invece, rimangono invariate, ad eccezione del fatto che ora alcune matrici hanno una dimensione maggiore e che ad ogni pixel dell'immagine è associato un vettore di colore. Per questa implementazione si è adottato il più generare modello affine descritto precedentemente.

Per ognuna delle versioni dell'algoritmo appena descritte, inoltre, è stata realizzata anche la corrispondente versione gerarchica, in quanto essa permette di riconoscere movimenti più ampi, rispetto alla versione normale: in questi casi, infatti, nelle immagini a bassa risoluzione l'effetto dovuto al movimento viene attenuato, e di conseguenza risulta più semplice per l'algoritmo convergere ad una stima corretta della trasformazione. Passando al livello successivo, inoltre, si riacquista parte del dettaglio perso col ridimensionamento, ma si ha anche a disposizione una buona stima di partenza per l'algoritmo. In questo modo, migliorando incrementalmente la stima della trasformazione, risulta possibile convergere al risultato corretto anche per immagini ottenute da movimenti ampi della telecamera, per cui la versione base degli algoritmi non riesce a calcolare correttamente la trasformazione avvenuta. In dettaglio, questa versione prevede di applicare ripetutamente l'algoritmo su una piramide di immagini ottenuta dai due input. In particolare, le implementazioni realizzate seguono lo schema dell'algoritmo descritto in [21]: per prima cosa viene costruita, sia per il template che per l'immagine di input, la piramide di immagini con diverse risoluzioni ai vari livelli, dopodiché, a partire dal livello con risoluzione minore, si applica l'algoritmo di allineamento, ottenendo una stima della trasformazione. Questa stima, prima di passare al livello successivo, viene aggiornata con la seguente modalità: per quanto riguarda il modello affine, i parametri a_{ij} vengono mantenuti inalterati, mentre sia per il modello affine, che per quello di traslazione i parametri t_x e t_y vengono moltiplicati per

il rapporto di risoluzione presente tra i livelli della piramide. Per esempio, se da un livello al successivo la risoluzione delle immagini raddoppia, allora i parametri vengono moltiplicati per due. Una volta aggiornati i parametri, l'algoritmo di allineamento viene nuovamente applicato, raffinando la stima di posizione fino al raggiungimento dell'ultimo livello della piramide.

3.2 Prestazioni dell'algoritmo

In questa sezione esamineremo le prestazioni degli algoritmi realizzati. Per fare questo, si sono considerate le condizioni il più possibile vicine a quelle ideali, ovvero moto di pura traslazione con la fotocamera che inquadra una scena fissa ad una distanza di 10cm. Per ottenere queste condizioni si è creato un sistema di guide per fissare la direzione del moto e tramite dei rialzi si è posta una maglia tesa sopra il dispositivo in modo tale che la fotocamera ne inquadrasse la parte centrale in buone condizioni di illuminazione. Con questo sistema si sono, quindi, registrati i video dalla fotocamera durante lo spostamento di alcune distanze significative: il movimento è stato effettuato il più lentamente possibile in modo tale da ridurre al minimo la sfocatura delle immagini. Alcuni frame del video realizzato per uno spostamento di 1.5cm sono riportati in Figura 3.2: com'è possibile osservare le condizioni di illuminazione sono costanti e non è presente un'evidente sfocatura dell'immagine durante il movimento.

Per valutare le prestazioni degli algoritmi implementati, quindi, la prima verifica effettuata è stata di allineare il primo frame con l'ultimo, per i vari video acquisiti. Tuttavia, dato che le immagini utilizzate hanno tutte la stessa risoluzione, essendo parte dello stesso video, l'ipotesi di template completamente contenuto nell'immagine da allineare è violata non appena avviene un movimento della fotocamera. Per questo motivo dall'immagine del template viene estratta una sottoimmagine della regione centrale, con dimensioni pari ad una frazione prestabilita dell'immagine originale: in questo modo, utilizzando come template nell'allineamento la sottoimmagine, il template risulta contenuto nell'immagine da allineare in qualunque direzione avvenga il moto. I risultati ottenuti sono riportati in Tabella 3.1: ogni riga corrisponde ad uno dei video realizzati, identificato con la distan-

(a) *Frame numero 10: prima del movimento.*

(b) *Frame numero 60: durante il movimento.*

(c) *Frame numero 80: dopo il movimento.*

Figura 3.2: *Esempi di frame del video di 1.5cm, presi nelle varie fasi del moto compiuto.*

Distanza	ICA affine		ICA traslazione		ICA colore	
	base	gerarchico	base	gerarchico	base	gerarchico
0.5						
1.0						
1.5						
2.0						
3.0						
4.0						
5.0						

(a) *Template con dimensioni pari al 33% di quelle dell'immagine.*

Distanza	ICA affine		ICA traslazione		ICA colore	
	base	gerarchico	base	gerarchico	base	gerarchico
0.5						
1.0						
1.5						
2.0						
3.0						
4.0						
5.0						

(b) *Template con dimensioni pari al 50% di quelle dell'immagine.*

Distanza	ICA affine		ICA traslazione		ICA colore	
	base	gerarchico	base	gerarchico	base	gerarchico
0.5						
1.0						
1.5						
2.0						
3.0						
4.0						
5.0						

(c) *Template con dimensioni pari al 100% di quelle dell'immagine (ovvero template ed immagine hanno uguale dimensione).*

Tabella 3.1: *Risultati ottenuti dall'allineamento del primo frame con l'ultimo, per ognuno dei video realizzati: nelle righe sono riportate le distanze percorse durante la ripresa, mentre nelle colonne le versioni degli algoritmi utilizzati. Le celle sono colorate in base al risultato ottenuto: in verde i casi in cui l'algoritmo ha correttamente allineato le immagini, mentre in rosso sono riportati i fallimenti.*

za percorsa durante la registrazione. Ogni colonna, invece, corrisponde ad una delle versioni degli algoritmi implementati. Le celle, inoltre, sono state colorate in base all'esito della registrazione: in verde sono riportati i casi in cui l'algoritmo ha allineato con successo le immagini, mentre in rosso sono mostrati i casi in cui l'algoritmo ha fallito l'allineamento. Ad ogni tabella, infine, è associata una diversa dimensione del template estratto. Per esempio, quindi, la versione gerarchica dell'algoritmo che si basa sul modello affine fallisce nell'allineare il primo e l'ultimo frame del video dello spostamento di 2cm se si utilizza un template di dimensione 33% o 100% della risoluzione originale, mentre ha successo se si adotta una dimensione del 50%. Quello che si può notare osservando le tabelle è che per tutti gli algoritmi la versione base riesce a calcolare la trasformazione corretta solamente per il video di 0.5cm, mentre in tutti gli altri casi la trasformazione trovata non corrisponde all'allineamento corretto. Un esempio di questo è visibile in figura Figura 3.3, dove è riportato l'input ed il risultato dell'algoritmo base con modello affine, dimensione del template del 33%, ed applicato al video di 0.5cm: in Figura 3.3b è mostrato il template da allineare all'immagine visibile in Figura 3.3a, mentre in Figura 3.3c è mostrato il template allineato tramite la trasformazione trovata, ed evidenziato con bordi rosso. Com'è possibile notare, il template risulta correttamente allineato all'immagine dalla trasformazione restituita dall'algoritmo, riportata di seguito:

$$W^{\text{base aff}}_{0.5,\,33\%} = \begin{pmatrix} 0.999273 & 0.00141565 & 97.1624 \\ 0.000514524 & 1.00169 & 63.6975 \\ 0 & 0 & 1 \end{pmatrix}$$

In Figura 3.4, invece, è riportato a titolo di esempio l'input ed il risultato della stessa configurazione applicata al video di 4cm: com'è possibile osservare, in questo caso la trasformazione trovata, e riportata di seguito, corrisponde ad un allineamento errato del template:

$$W^{\text{base aff}}_{4.0,\,33\%} = \begin{pmatrix} 0.511483 & 1.43254 & 96.5459 \\ 0.0892475 & 1.08814 & 51.6846 \\ 0 & 0 & 1 \end{pmatrix}$$

Per quanto riguarda le versioni gerarchiche, invece, possiamo notare come esse siano sostanzialmente equivalenti: fissata una dimensione del tem-

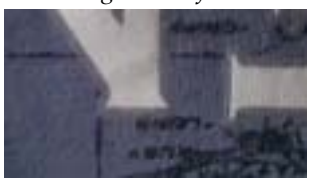
(a) *Immagine di riferimento.*

(b) *Template da allineare.*

(c) *Risultato dell'allineamento: il template deformato è evidenziato in rosso.*

Figura 3.3: *Risultato dell'allineamento del primo frame sull'ultimo, per il video di 0.5cm e dimensione del template del 33%.*

(a) *Immagine di riferimento.*



(b) *Template da allineare.*

(c) *Risultato dell'allineamento: il template deformato è evidenziato in rosso.*

Figura 3.4: *Risultato dell'allineamento del primo frame sull'ultimo, per il video di 4cm e dimensione del template del 33%.*

plate, ogni algoritmo fornisce la trasformazione corretta negli esatti casi in cui anche agli altri la forniscono e, viceversa, essi restituiscono un allineamento errato in tutti i casi in cui anche gli altri falliscono. Un'eccezione è rappresentata dall'allineamento per il video di 2cm e con dimensione del template pari a 100% dell'immagine originale: in questo caso l'algoritmo con modello traslatorio riesce a trovare con successo la trasformazione. Questo, tuttavia, è un caso isolato, dovuto, ragionevolmente, a condizioni iniziali particolarmente favorevoli che, in congiunzione alla semplicità del modello, hanno permesso la convergenza al valore corretto. Un altro fatto che emerge dalle tabelle è che per dimensioni del template del 50% gli algoritmi riescono a trovare l'allineamento corretto per i video fino a quello di 2cm, mentre per una dimensione del 33% non vanno oltre il video di 1.5cm, e nel caso di dimensione di 100% si ottengono sostanzialmente gli stessi risultati della versione base. Questo, tuttavia, è un risultato facilmente spiegabile: per dimensioni piccole del template, diventa più difficile per gli algoritmi convergere all'aumentare della differenza tra immagine e template, in quanto questo contiene poca informazione. Per dimensioni elevate, invece, il template contiene molta informazione, ma l'ipotesi di template completamente contenuto nell'immagine viene progressivamente meno, causando un degrado delle prestazioni. Per dimensioni intermedie, invece, il template contiene informazione sufficiente e l'ipotesi effettuata rimane valida.

Vediamo ora la seconda verifica fatta per stabilire le prestazioni degli algoritmi: in particolare, analizziamo ora i video completi. Nel dettaglio, consideriamo i frame nel loro ordine temporale: posto il primo frame come riferimento, si procede ad allineare il secondo frame al primo. In caso di successo si mantiene il primo frame come riferimento e si allinea il terzo frame al primo. Si procede con questa modalità fino a quando non avviene un fallimento nell'allineamento: in questo caso il frame che non si è riusciti ad allineare diventa il nuovo riferimento e si continua con la stessa modalità precedente, allineando il frame successivo al riferimento ed aggiornando questo solo in caso di fallimento. Per stabilire se è avvenuto un fallimento nell'allineamento si sono poste delle condizioni sui coefficienti della trasformazione restituita dall'algoritmo: in particolare si è verificato che i coefficienti di traslazione non fossero superiori alle dimensioni dell'immagine, in quanto in questo caso la trasformazione allineerebbe il template

"al di fuori" dell'immagine. La seconda condizione verificata, invece, è che i quattro coefficienti a_{ij} non distassero dal loro valore ideale più di 0.01: per esempio a_{00}, come a_{11}, in caso di pura traslazione dovrebbe assumere il valore 1 e si è controllato, quindi, che valesse $\|a_{00} - 1\| \leq 0.01$, mentre a_{10}, come a_{01}, dovrebbe assumere il valore 0, quindi la condizione diventa $\|a_{10}\| \leq 0.01$. Mostriamo, quindi, in Tabella 3.2 i risultati ottenuti con l'algoritmo gerarchico che usa il modello affine ed una dimensione del template pari a 50%: si sono riportati i risultati di questo solo algoritmo come rappresentativo di tutti, in quanto gli altri algoritmi e dimensioni del template forniscono risultati analoghi. In tabella è possibile osservare gli errori di allineamento ottenuti: ogni colonna corrisponde ad un video, la cui distanza identificativa è riportata nell'intestazione della tabella, mentre nel corpo sono elencati tutti i casi di errore nell'allineamento per ogni video, con la convenzione che $i \to j$ indica un errore nell'allineare il frame i al frame j. Com'è possibile osservare, tutti i video presentano errori di allineamento. Notiamo che questi sono concentrati nella parte centrale del video e che all'aumentare della distanza percorsa il numero di errori aumenta: si è rilevato manualmente, infatti, che in tutti i casi i fallimenti nell'allineamento avvengono in corrispondenza dei primi frame in cui inizia il movimento, mentre gli ultimi errori sono localizzati nei frame finali del moto. Infatti, a dispositivo fermo due fotogrammi consecutivi sono identici, quindi, come ci si aspetta, gli algoritmi li allineano correttamente. La localizzazione degli errori nel periodo in cui c'è movimento, inoltre, è in accordo con il fatto che video di distanze superiori presentano un maggior numero di errori: per questi video, infatti, un numero più elevato di frame contiene movimento, a causa della maggior distanza percorsa. Mostriamo, infine, a titolo di esempio uno degli allineamenti falliti. Consideriamo l'allineamento del frame 72 sul 69 per il video di 4cm: in Figura 3.5 è mostrato il frame 69, a cui il template estratto dal frame 72, e mostrato in Figura 3.5b , deve essere allineato. Il risultato fornito dall'algoritmo è mostrato in Figura 3.5c e la trasformazione è la seguente

$$W^{\text{ger aff}}_{4.0,\,50\%} = \begin{pmatrix} 1.58797 & -0.189102 & 47.2106 \\ -0.00805855 & 1.0233 & 47.0837 \\ 0 & 0 & 1 \end{pmatrix}$$

Tutto questo fa nascere l'ipotesi che l'algoritmo *inverse compositional* sia

0.5	1.0	1.5	2.0	3.0	4.0	5.0
$49 \to 0$	$39 \to 0$	$42 \to 0$	$42 \to 0$	$29 \to 0$	$26 \to 0$	$25 \to 0$
$50 \to 49$	$40 \to 39$	$43 \to 42$	$43 \to 42$	$30 \to 29$	$27 \to 26$	$26 \to 25$
$51 \to 50$	$41 \to 40$	$44 \to 43$	$44 \to 43$	$31 \to 30$	$28 \to 27$	$27 \to 26$
$52 \to 51$	$42 \to 41$	$47 \to 44$	$45 \to 44$	$32 \to 31$	$29 \to 28$	$28 \to 27$
$53 \to 52$	$43 \to 42$	$48 \to 47$	$46 \to 45$	$33 \to 32$	$31 \to 29$	$29 \to 28$
$54 \to 53$	$44 \to 43$	$52 \to 48$	$48 \to 46$	$36 \to 33$	$32 \to 31$	$30 \to 29$
	$45 \to 44$	$53 \to 52$	$50 \to 48$	$40 \to 36$	$33 \to 32$	$31 \to 30$
	$47 \to 45$	$54 \to 53$	$51 \to 50$	$42 \to 40$	$35 \to 33$	$32 \to 31$
	$50 \to 47$	$55 \to 54$	$54 \to 51$	$44 \to 42$	$38 \to 35$	$33 \to 32$
		$56 \to 55$	$55 \to 54$	$49 \to 44$	$40 \to 38$	$35 \to 33$
		$57 \to 56$	$56 \to 55$	$52 \to 49$	$42 \to 40$	$38 \to 35$
		$60 \to 57$	$59 \to 56$	$56 \to 52$	$47 \to 42$	$40 \to 38$
		$64 \to 60$	$61 \to 59$	$59 \to 56$	$48 \to 47$	$42 \to 40$
			$63 \to 61$	$61 \to 59$	$50 \to 48$	$44 \to 42$
				$62 \to 61$	$53 \to 50$	$46 \to 44$
				$65 \to 62$	$55 \to 53$	$48 \to 46$
				$66 \to 65$	$56 \to 55$	$50 \to 48$
				$70 \to 66$	$59 \to 56$	$51 \to 50$
					$66 \to 59$	$52 \to 51$
					$67 \to 66$	$53 \to 52$
					$69 \to 67$	$54 \to 53$
					$72 \to 69$	$56 \to 54$
					$74 \to 72$	$61 \to 56$
					$77 \to 74$	$63 \to 61$
					$82 \to 77$	$65 \to 63$
					$83 \to 82$	$71 \to 65$
						$72 \to 71$
						$73 \to 72$
						$75 \to 73$
						$86 \to 75$
						$89 \to 86$
						$90 \to 89$
						$91 \to 90$
						$96 \to 91$
						$99 \to 96$
						$101 \to 99$

Tabella 3.2

(a) *Immagine di riferimento corrispondente al frame numero 69.*

(b) *Template da allineare estratto dal frame 72.*

(c) *Risultato dell'allineamento: il template deformato è evidenziato in rosso.*

Figura 3.5: *Risultato dell'allineamento del frame 72 sul frame 69, per il video di 4cm e dimensione del template del 50%.*

Distanza	Base			Gerarchico		
	33%	50%	100%	33%	50%	100%
0.5						
1.0						
1.5						
2.0						
3.0						
4.0						
5.0						

Tabella 3.3: *Risultati dell'allineamento tramite implementazione Matlab dell'algoritmo, effettuato sul primo frame e l'ultimo per i vari video realizzati: nell'intestazione di riga è riportata la distanza percorsa nel video come suo identificativo, mentre sulle intestazioni delle colonne sono riportati gli algoritmi utilizzati. Il colore verde nella cella indica che l'algoritmo ha fornito l'allineamento corretto, e viceversa nel caso del rosso.*

poco robusto, e che la sua convergenza al risultato corretto sia largamente dipendente dalla stima iniziale della trasformazione, assieme alla "qualità" del template.
Prima di esplorare questa possibilità, tuttavia, si sono confrontati i risultati ottenuti con un'implementazione dell'algoritmo realizzata da terzi, in modo tale da poter escludere che le scarse prestazioni rilevate siano state causate dall'algoritmo implementato. Il programma utilizzato è reperibile da [3]: esso è un'implementazione dell'algoritmo in codice Matlab, realizzata nell'ambito di un progetto dell'istituto di robotica della Carnegie Mellon University. Quest'implementazione fornisce solamente la versione base con modello affine, tuttavia aggiungendo al programma una piccola porzione di codice che crea la piramide di immagini si è implementa la versione gerarchica, basata sempre, quindi, sul codice trovato. Con queste due versioni dell'algoritmo si sono ripetuti gli allineamenti eseguiti sul primo ed ultimo frame dei video utilizzati. I risultati sono visibili in Tabella 3.3: nelle righe sono riportati i video, identificati con la distanza percorsa nella registrazione, mentre sulle colonne sono mostrate le versioni degli algoritmi e le dimensioni del template utilizzate. Anche in questo caso si è identificato un allineamento corretto con una cella di colore verde, mentre con il rosso si è rappresentato il fallimento. Com'è possibile osservare, anche questa implementazione dell'algoritmo presenta risultati deludenti: la versione base ottiene prestazioni paragonabili a quella da noi realizzata, mentre la gerarchica si comporta notevolmente peggio. In

particolare, la versione base fornisce risultati numericamente simili a quelli della nostra implementazione, come per esempio il caso del video di 1.5cm e dimensione del template del 100%:

$$W^{\text{base aff}}_{1.5,\,100\%} = \begin{pmatrix} 0.960003 & -0.0512692 & 10.2794 \\ -0.0107005 & 0.990101 & 2.41649 \\ 0 & 0 & 1 \end{pmatrix}$$

$$W^{\text{Matlab base aff}}_{1.5,\,100\%} = \begin{pmatrix} 0.9596 & -0.0487 & 10.3015 \\ -0.0103 & 0.9897 & 2.3973 \\ 0 & 0 & 1.0000 \end{pmatrix}$$

con qualche eccezione isolata, come il caso del video di 0.5cm e dimensione del template del 50%:

$$W^{\text{base aff}}_{0.5,\,50\%} = \begin{pmatrix} 1.00713 & 0.0964193 & 65.2439 \\ 0.00282167 & 1.01298 & 47.1407 \\ 0 & 0 & 1 \end{pmatrix}$$

$$W^{\text{Matlab base aff}}_{0.5,\,50\%} = \begin{pmatrix} 1.0056 & 0.2140 & 62.4521 \\ 0.0012 & 1.0174 & 47.0933 \\ 0 & 0 & 1.0000 \end{pmatrix}$$

Per quanto riguarda la versione gerarchica, invece, le differenze sono più accentuate: in alcune configurazioni si ottengono valori simili, mentre in altre la differenza risulta marcata. Questo è dovuto, ragionevolmente, ad una combinazione di cause: la creazione della piramide in due ambienti diversi può causare una lieve differenza nei valori dei pixel. Inoltre, anche i metodi numerici utilizzati dagli algoritmi possono risultare in approssimazioni differenti, e questo, considerando le ripetizioni di queste operazioni introdotte dal metodo della piramide, può portare a valori discrepanti. Da queste osservazioni concludiamo, comunque, che le scarse prestazioni osservate non sono dovute all'implementazione dell'algoritmo realizzata.

Vediamo, ora, l'esito dell'allineamento effettuato su immagini differenti, al fine di escludere la possibilità che i risultati ottenuti siano dovuti alle immagini finora utilizzate. Mostriamo in Figura 3.6 una delle coppie di immagini utilizzate: esse sono state ottenute tramite un iPhone, ponendo il dispositivo di fronte alla bocca, ed acquisendo la seconda immagine

(a) *Prima immagine acquisita.*

(b) *Seconda immagine acquisita.*

Figura 3.6: *Coppia di immagini acquisita tramite iPhone: esse risultano "sottosopra" a causa del metodo di acquisizione dell'iPhone.*

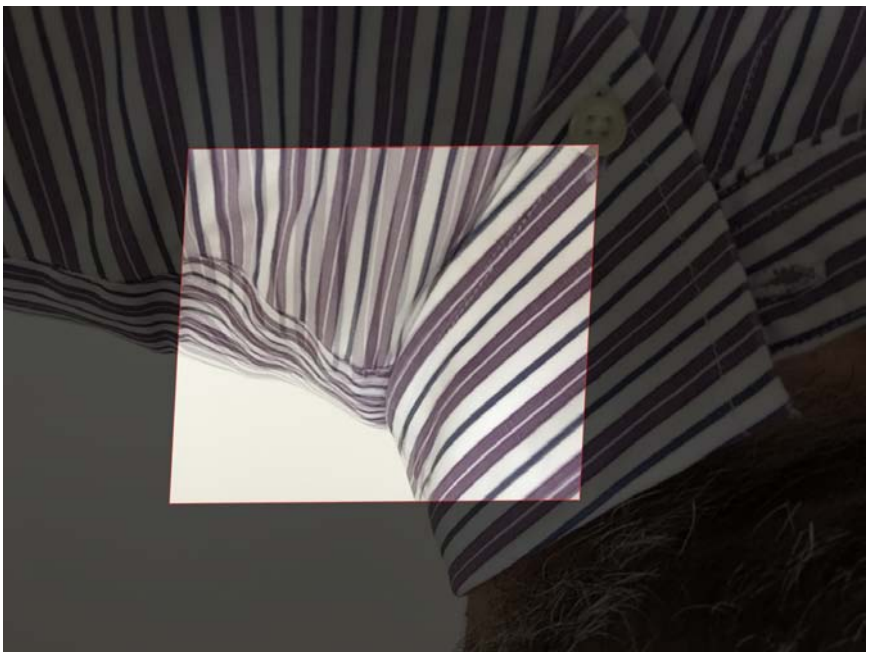


Figura 3.7: *Risultato dell'allineamento della prima immagine della camicia sulla seconda, utilizzando l'algoritmo gerarchico con modello affine e dimensione del template del 50%.*

Figura 3.8: *Risultato dell'allineamento delle immagini della camicia ruotate e salvate con formato PNG, utilizzando l'algoritmo gerarchico con modello affine e dimensione del template del 50%.*

traslando il dispositivo dopo aver registrato la prima. Riportiamo a titolo di esempio in Figura 3.7 il risultato ottenuto utilizzando l'algoritmo gerarchico con modello affine ed una dimensione del template del 50%: com'è possibile osservare, l'algoritmo ha raggiunto un ottimo locale, riuscendo ad allineare solo parzialmente le immagini, come si può notare osservando l'angolo del template in corrispondenza del bottone della camicia, e la regione sopra la spalla. Un risultato analogo, inoltre, viene raggiunto da tutti gli algoritmi, ed utilizzando una diversa dimensione del template il risultato non cambia, ottenendo sempre un allineamento parziale dovuto al raggiungimento di un ottimo locale. Riportiamo, inoltre, in Figura 3.8 il risultato ottenuto allineando una elaborazione delle immagini mostrate in Figura 3.6: tramite un software di elaborazione grafica (in dettaglio, GIMP) si sono raddrizzate le immagine, fisicamente memorizzate capovolte dall'iPhone, e le si sono salvate in formato PNG, al posto dell'originario JPEG. Queste nuove immagini sono poi state allineate con gli tutti algoritmi considerati: il risultato ottenuto è analogo al precedente, per cui gli algoritmi convergono ad un ottimo locale che non corrisponde alla trasformazione corretta. Nella figura, infatti, è riportato il caso dell'algoritmo gerarchico con modello affine e dimensione del template del 50%: com'è possibile notare, le immagini non sono state allineate correttamente, tuttavia il risultato ottenuto differisce da quello precedente. Infatti, in questo caso un angolo del template è stato allineato con la barba, mentre nel caso precedente non si superava il bordo del colletto della camicia. Ipotizziamo che l'algoritmo amplifichi una lieve differenza presente tra i pixel delle immagini ruotate e non, causata dall'operazione di rotazione, risultando nel diverso allineamento riscontrato.

Vediamo ora una seconda coppia di immagini considerata: in Figura 3.9 è riportata la prima immagine, utilizzata come riferimento nell'allineamento, assieme al template estratto dalla seconda. Queste foto ad alta risoluzione sono state ottenute tramite una fotocamera Reflex, effettuando più scatti consecutivamente alla stessa scena, ed ottenendo quindi immagini leggermente differenti tra loro. Quello che si ottiene utilizzando gli algoritmi per allineare il template all'immagine è che tutte e tre le versioni gerarchiche forniscono un risultato corretto, mentre le tre versioni base falliscono. Precisiamo, tuttavia, che l'algoritmo che utilizza il modello di traslazione commette un piccolo errore, dovuto al fatto che la trasformazione realmen-

te presente tra le immagini non è perfettamente approssimabile ad una traslazione. Consideriamo, inoltre, una seconda scena acquisita ancora con la fotocamera Reflex e con le stesse modalità della precedente: in Figura 3.10 è mostrata la prima immagine, utilizzata come riferimento nell'allineamento, assieme al template estratto dalla seconda immagine. In questo caso, gli algoritmi esibiscono un comportamento opposto al precedente: tutte e tre le versioni gerarchiche falliscono, mentre le tre base forniscono il risultato corretto (con le stesse considerazioni fatte precedentemente per la versione con modello traslatorio).
Da tutte queste osservazioni possiamo concludere che l'algoritmo *inverse compositional* presenta un comportamento poco stabile, e che il raggiungimento del risultato corretto è largamente dipendente dalle immagini da allineare, oltre che dalla necessità di avere una stima iniziale della trasformazione molto buona, per cui anche piccole variazioni nelle immagini possono portare a risultati molto diversi.

(a) *Immagine di riferimento.*

(b) *Template considerato.*

Figura 3.9: *Prima coppia di immagini ad alta risoluzione, acquisite con una fotocamera Reflex, ottenute scattando consecutivamente delle foto alla stessa scena.*

(a) *Immagine di riferimento.*

(b) *Template considerato.*

Figura 3.10: *Seconda coppia di immagini ad alta risoluzione, acquisite con una fotocamera Reflex, ottenute scattando consecutivamente delle foto alla stessa scena.*

A Codice Mathematica

A.1 Algoritmo di rimozione della gravità

```
RimuoviGravita[signal_, times_, numSample_]:=
Module[{gravity, out},
  Assert[Length[signal] == Length[times]];
  Assert[numSample ≥ 1];

  gravity = 1/numSample ∑_{k=1}^{numSample} signal[[k]];
  out = (# − gravity&)/@signal[[numSample + 1;;]];
  Return[< |"data" → out, "times" → times[[numSample + 1;;]]| >];
](* end module *)
```

A.2 Algoritmo del filtro passa basso di primo ordine

```
FiltroPassaBasso[signal_, times_, RC_]:=
Module[{α, out = {}, dt, k, filtered = 0},
  Assert[Length[signal] == Length[times]];
  Assert[Length[signal] ≥ 1];
  Assert[RC > 0];

  AppendTo[out, signal[[1]]];
  For[k = 2, k ≤ Length[signal], k++, (*do*)
    dt = times[[k]] − times[[k − 1]];
```

```
    α = dt/(RC+dt);
    filtered = filtered + α * (signal[[k]] - filtered);
    AppendTo[out, filtered];
  ];(* end for *)
  Return[<|"data" → out, "times" → times|>];
](* end module *)
```

A.3 Algoritmo di integrazione (metodo dei rettangoli)

```
IntegraRettangoli[signal_, times_, dimens_, threshold_, numZero_]:=
Module[{pos = 0, vel = 0, acc, k, dt, out = {}, countZero = 0},
  Assert[Length[signal] == Length[times]];
  Assert[1 ≤ dimens ≤ 3];
  Assert[threshold ≥ 0];
  Assert[numZero ≥ 0];

  AppendTo[out, 0];
  For[k = 2, k ≤ Length[signal], k++, (*do*)
    dt = times[[k]] - times[[k - 1]];
    If[Abs[signal[[k, dimens]]] < threshold, (*then*)
       acc = 0
    , (*else*)
       acc = signal[[k, dimens]]
    ];(* end if *)
    If[acc ≠ 0, (*then*)countZero = 0, (*else*)countZero = countZero + 1];
    If[countZero ≥ numZero, (*then*)
      vel = 0;
    , (*else*)
      vel = vel + acc * dt;
      pos = pos + vel * dt + 1/2 acc * dt^2;
    ];(* end if *)
    AppendTo[out, pos];
  ];(* end for *)
```

```
    Return[< |"data" → out, "times" → times| >];
]
```

A.4 Algoritmo per la Trasformata di Fourier

```
TrasformataFourier[signal_, times_, samplingFreq_]:=
Module[{transf, freq, k, df},
  Assert[Length[signal] == Length[times]];
  Assert[samplingFreq > 0];

  transf = Fourier[signal, FourierParameters → {1, −1}];
  transf = RotateRight [transf, ⌈(Length[transf]−1)/2⌉];
  df = samplingFreq/Length[transf];
  freq = Table [df ∗ (k − ⌈(Length[transf]−1)/2⌉), {k, 0, Length[transf] − 1}];
  Return[< |"data" → transf, "times" → freq| >];
](* end module *)
```

A.5 Algoritmo di integrazione (metodo dei trapezi)

```
IntegraTrapezi[signal_, times_, dimens_, threshold_, numZero_]:=
Module[{pos = 0, vel = 0, acc, k, dt, out = {}, timeout = {},
    countZero = 0, lastAcc, lastVel},
  Assert[Length[signal] == Length[times]];
  Assert[1 ≤ dimens ≤ 3];
  Assert[threshold ≥ 0];
  Assert[numZero ≥ 0];

  AppendTo[out, 0];
  lastAcc = signal[[1, dimens]];
  lastVel = 0;
  For[k = 2, k ≤ Length[signal], k++, (*do*)
```

```
    dt = times[[k]] − times[[k − 1]];
    If[Abs[signal[[k, dimens]]] < threshold, (*then*)
      acc = 0;
    , (*else*)
      acc = signal[[k, dimens]];
    ]; (* end if *)
    If[acc ≠ 0, (*then*) countZero = 0, (*else*) countZero = countZero + 1];
    If[countZero ≥ numZero, (*then*)
      vel = 0;
    , (*else*)
      vel = vel + dt ∗ acc+lastAcc/2;
      pos = pos + dt ∗ vel+lastVel/2;
    ]; (* end if *)
    lastAcc = acc;
    lastVel = vel;
    AppendTo[out, pos];
  ]; (* end for *)
  Return[< |"data" → out, "times" → times| >];
](* end module *)
```

A.6 Algoritmo di integrazione vettoriale

```
Integra[signal_, times_, method_, threshold_, numZero_]:=
Module[{pos, vel, numDimens, acc, k, dt, out = {}, countZero = 0,
    zero, lastAcc, lastVel},
  Assert[Length[signal] == Length[times]];
  Assert[Length[signal] ≥ 1];
  Assert[method == "Rettangoli" ∨ method == "Trapezi"];
  Assert[threshold ≥ 0];
  Assert[numZero ≥ 0];

  numDimens = Length[signal[[1]]];
  If[numDimens == 1, (*then*)
     zero = 0;
```

```
  , (*else*)
     zero = ConstantArray[0, numDimens];
  ]; (* end if *)
  AppendTo[out, zero];
  lastAcc = signal[[1]];
  lastVel = zero;
  pos = vel = zero;
  For[k = 2, k ≤ Length[signal], k++, (*do*)
    dt = times[[k]] − times[[k − 1]];
    If[Norm[signal[[k]]] < threshold, (*then*)
      acc = zero
    , (*else*)
      acc = signal[[k]]
    ]; (* end if *)
    If[acc ≠ 0, (*then*)countZero = 0, (*else*)countZero = countZero + 1];
    If[countZero ≥ numZero, (*then*)
      vel = zero;
    , (*else*)
      If[method == "Trapezi", (*then*)
        vel = vel + dt ∗ acc+lastAcc/2;
        pos = pos + dt ∗ vel+lastVel/2;
      , (*else*)
        vel = vel + acc ∗ dt;
        pos = pos + vel ∗ dt + 1/2 ∗ acc ∗ dt^2;
      ]; (* end if *)
    ]; (* end if *)
    lastAcc = acc;
    lastVel = vel;
    AppendTo[out, pos];
  ]; (* end for *)
  Return[< |"data" → out, "times" → times| >];
](* end module *)
```

A.7 Algoritmo per il filtro passa alto

```
FiltroPassaAlto[signal_, times_, krnlID_]:=
Module[{out = {}, krnl, x, y, z, ExtractDimens, JoinDimens},
  Assert[Length[signal] == Length[times]];

  ExtractDimens[a_, i_]:=Flatten[Take[a, All, {i}]];
  JoinDimens[x_, y_, z_]:=Table[{x[[k]], y[[k]], z[[k]]}, {k, 1, Length[x]}];

  krnl = GetFilterKernel[krnlID];

  x = ExtractDimens[signal, 1];
  y = ExtractDimens[signal, 2];
  z = ExtractDimens[signal, 3];
  x = ListConvolve[krnl, x];
  y = ListConvolve[krnl, y];
  z = ListConvolve[krnl, z];
  out = JoinDimens[x, y, z];
  Return[< |"data" → out, "times" → times[[Length[krnl] ;;]]| >];
](* end module *)
```

A.8 Algoritmo di upsampling

```
UpSample[signal_, samplingPeriod_, upsampledPeriod_, method_]:=
Module[{t, k, j, j1, numUpsample, outData, outTimes, zero, upSampledIndex},
  Assert[Head@signal == Association];
  Assert[Length[signal["data"]] == Length[signal["times"]]];
  Assert[MemberQ[{"Linear interpolation","Filtering","Zeros"}, method]];

  If[Length[signal["data"][[1]]] == 1, (*then*)
     zero = 0
  , (*else*)
     zero = ConstantArray[0, Length[signal["data"][[1]]]]
  ]; (* end if *)
```

```
Module[{numSamplesBeforeLastTime, upTimeBeforeLastTime,
    upTimeAfterLastTime, firstTime, lastTime,
    distLastTimePrevUpsamp, distLastTimeNextUpsamp},
  firstTime = First@signal["times"];
  lastTime = Last@signal["times"];
  numSamplesBeforeLastTime = ⌊(lastTime − firstTime)/upsampledPeriod⌋ + 1;
  upTimeBeforeLastTime = firstTime +
      upsampledPeriod * (numSamplesBeforeLastTime − 1);
  upTimeAfterLastTime = upTimeBeforeLastTime + upsampledPeriod;
  numUpsample = numSamplesBeforeLastTime;
  distLastTimePrevUpsamp = Abs[lastTime − upTimeBeforeLastTime];
  distLastTimeNextUpsamp = Abs[lastTime − upTimeAfterLastTime];
  If[distLastTimeNextUpsamp < distLastTimePrevUpsamp, (*then*)
    numUpsample = numUpsample + 1;
  ]; (* end if *)
]; (* end module *)

Module[{nextTime, currUpsampTime, prevUpsampTime,
    nextUpsampTime, distNextTimeCurrUpsamp,
    distNextTimePrevUpsamp, distNextTimeNextUpsamp},
  outData = {First@signal["data"]};
  outTimes = {First@signal["times"]};
  upSampledIndex = {1};
  t = 2;
  For[k = 2, k ≤ numUpsample, k++, (*do*)
    nextTime = signal["times"][[t]];
    currUpsampTime = First@outTimes + upsampledPeriod * (k − 1);
    prevUpsampTime = currUpsampTime − upsampledPeriod;
    nextUpsampTime = currUpsampTime + upsampledPeriod;
    distNextTimeCurrUpsamp = Abs[nextTime − currUpsampTime];
    distNextTimePrevUpsamp = Abs[nextTime − prevUpsampTime];
    distNextTimeNextUpsamp = Abs[nextTime − nextUpsampTime];

    If[(distNextTimeCurrUpsamp < distNextTimePrevUpsamp∧
    distNextTimeCurrUpsamp < distNextTimeNextUpsamp)
```

```
      ∨(t == Length[signal[“times”]] ∧ k == numUpsample),
      (*then*)
        AppendTo[outData, signal[“data”][[t]]];
        AppendTo[upSampledIndex, k];
        t = t + 1;
      , (*else*)
        AppendTo[outData, zero];
      ]; (* end if *)
      AppendTo[outTimes, currUpsampTime];
    ]; (* end for *)
  ]; (* end module *)

  Switch[method
    (*CASE*), “Linear interpolation”,
      t = 2;
      For[k = 2, k ≤ numUpsample, k++, (*do*)
        If[k ≠ upSampledIndex[[t]], (*then*)
          j = upSampledIndex[[t]];
          j1 = upSampledIndex[[t − 1]];
          outData[[k]] = k−j1/j−j1 (outData[[j]] − outData[[j1]]) + outData[[j1]];
        , (*else*)
          t = t + 1;
        ]; (* end if *)
      ]; (* end for *)

    (*CASE*), “Filtering”,
      Module[{krnl, dimx, dimy, dimz, ExtractDimens, JoinDimens},
        ExtractDimens[a_, i_]:=Flatten[Take[a, All, {i}]];
        JoinDimens[x_, y_, z_]:=Table[{x[[k]], y[[k]], z[[k]]}, {k, 1, Length[x]}];

        krnl = GetFilterKernel[];

        outData = samplingPeriod/upsampledPeriod outData;
        dimx = ExtractDimens[outData, 1];
        dimy = ExtractDimens[outData, 2];
        dimz = ExtractDimens[outData, 3];
```

```
        dimx = Chop/@ListConvolve[krnl, dimx];
        dimy = Chop/@ListConvolve[krnl, dimy];
        dimz = Chop/@ListConvolve[krnl, dimz];
        outData = JoinDimens[dimx, dimy, dimz];

        outTimes = outTimes[[Length[krnl];;]];
      ];(* end module *)

    (*CASE*),"Zeros",
      Null;(* do nothing *)
  ];(* end switch *)

  Return[< |"data" → outData,"times" → outTimes| >];
](* end module *)
```

B Codice Android

B.1 DataRecorder

DataRecorder.java

```
package DataRecorderPackage;

import android.Manifest;
import android.app.Activity;
import android.content.Context;
import android.content.pm.PackageManager;
import android.hardware.Camera;
import android.hardware.Camera.CameraInfo;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.media.CamcorderProfile;
import android.media.MediaRecorder;
import android.os.Bundle;
import android.os.Environment;
import android.support.v4.app.ActivityCompat;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.WindowManager;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.Toast;

import java.io.File;
```

```
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;


public class DataRecorder extends Activity implements
    SensorEventListener
{
private Camera mCamera;
private CameraPreview mPreview;
private MediaRecorder mediaRecorder;
private ImageView capture, switchCamera;
private Context myContext;
private LinearLayout cameraPreview;
private boolean cameraFront = false;
private static final int RC_HANDLE_CAMERA_PERM = 2;
private static final int
    RC_HANDLE_WRITE_EXTERNAL_STORAGE_PERM = 3;
private static final int RC_HANDLE_RECORD_AUDIO_PERM = 4;
private String filePath;
public static int cameraId = -1;
boolean recording = false;
private String StartDate;
private SensorManager mSensorManager;
private Sensor mLinearAcc;
private Sensor mAccelerometer;
private Sensor mRotation;
private ArrayList<String> readingsLinearAcc;
private ArrayList<String> readingsAccelerometer;
private ArrayList<String> readingsRotation;


@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_camera_capture);
getWindow().addFlags(WindowManager.LayoutParams.
    FLAG_KEEP_SCREEN_ON);
```

```
myContext = this;

filePath = Environment.getExternalStoragePublicDirectory(
    Environment.DIRECTORY_MOVIES).getAbsolutePath() + "/"
     + getResources().getString(R.string.app_name);
File file = new File(filePath);
file.mkdirs();

int rc = ActivityCompat.checkSelfPermission(this,
    Manifest.permission.CAMERA);
int sdCard = ActivityCompat.checkSelfPermission(this,
    Manifest.permission.WRITE_EXTERNAL_STORAGE);
if(sdCard == PackageManager.PERMISSION_GRANTED)
{
int recordAudio = ActivityCompat.checkSelfPermission(this
    , Manifest.permission.RECORD_AUDIO);
if(recordAudio == PackageManager.PERMISSION_GRANTED)
{
if (rc == PackageManager.PERMISSION_GRANTED){
initialize();
initializeCamera();
}
else
requestCameraPermission();
}
else
requestRecordAudioPermission();
}
else
requestSDCardPermission();

mSensorManager = (SensorManager)getSystemService(Context.
    SENSOR_SERVICE);
mLinearAcc = mSensorManager.getDefaultSensor(Sensor.
    TYPE_LINEAR_ACCELERATION);
mAccelerometer = mSensorManager.getDefaultSensor(Sensor.
    TYPE_ACCELEROMETER);
mRotation = mSensorManager.getDefaultSensor(Sensor.
    TYPE_ROTATION_VECTOR);
}
```

```
public void onResume()
{
super.onResume();
if(!hasCamera(myContext))
{
Toast toast = Toast.makeText(myContext, "Sorry, your
   phone does not have a camera!", Toast.LENGTH_LONG);
toast.show();
finish();
}
int rc = ActivityCompat.checkSelfPermission(this,
   Manifest.permission.CAMERA);
if (rc == PackageManager.PERMISSION_GRANTED)
initializeCamera();

mSensorManager.registerListener(this, mLinearAcc,
   SensorManager.SENSOR_DELAY_FASTEST);
mSensorManager.registerListener(this, mAccelerometer,
   SensorManager.SENSOR_DELAY_FASTEST);
mSensorManager.registerListener(this, mRotation,
   SensorManager.SENSOR_DELAY_FASTEST);
}

@Override
protected void onPause(){
super.onPause();
releaseCamera();
mSensorManager.unregisterListener(this);
}

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

@Override
public final void onSensorChanged(SensorEvent event)
{
if(recording)
{
if (event.sensor == mLinearAcc)
```

```
readingsLinearAcc.add(String.format("%f;%f;%f;%d", event.
    values[0], event.values[1], event.values[2], event.
    timestamp));
else if(event.sensor == mAccelerometer)
readingsAccelerometer.add(String.format("%f;%f;%f;%d",
    event.values[0], event.values[1], event.values[2],
    event.timestamp));
else if(event.sensor == mRotation)
readingsRotation.add(String.format("%f;%f;%f;%f;%d",
    event.values[0], event.values[1], event.values[2],
    event.values[3], event.timestamp));
else
Log.e(DataRecorderPackage.DataRecorder.class.getName(), "
    Received a sensor event while recording from an
    unregistered sensor.");
}

return;
}

OnClickListener captrureListener = new OnClickListener()
{
@Override
public void onClick(View v)
{
if(recording)
{
mediaRecorder.stop();
releaseMediaRecorder();
Toast.makeText(DataRecorderPackage.DataRecorder.this, "
    Video captured!", Toast.LENGTH_LONG).show();
recording = false;
capture.setSelected(true);

FileOutputStream fileLinearAcc = null;
FileOutputStream fileAccelerometer = null;
FileOutputStream fileRotation = null;
try {
fileLinearAcc = new FileOutputStream(filePath + "/" +
    StartDate + "_LinearAcceleration.csv");
```

```
fileAccelerometer = new FileOutputStream(filePath + "/" +
     StartDate + "_Accelerometer.csv");
fileRotation = new FileOutputStream(filePath + "/" +
    StartDate + "_Rotation.csv");
}catch(FileNotFoundException e){
Log.e(DataRecorderPackage.DataRecorder.class.getName(), "
    CAN'T OPEN FILE FOR SENSOR DATA");
e.printStackTrace();
finish();
}

try {
for(String str : readingsLinearAcc) fileLinearAcc.write((
    str+"\n").replace(',', '.').getBytes());
for(String str : readingsAccelerometer) fileAccelerometer
    .write((str+"\n").replace(',', '.').getBytes());
for(String str : readingsRotation) fileRotation.write((
    str+"\n").replace(',', '.').getBytes());

fileLinearAcc.close();
fileAccelerometer.close();
fileRotation.close();

}catch(IOException e){
Log.e(DataRecorderPackage.DataRecorder.class.getName(), "
    ERROR IN WRITING SENSOR DATA ON FILE");
e.printStackTrace();
finish();
}

}
else
{
StartDate = (new SimpleDateFormat("yyyy-MM-dd_HH-mm-ss"))
    .format(new Date());

if (!prepareMediaRecorder()){
Toast.makeText(DataRecorderPackage.DataRecorder.this, "
    Fail in prepareMediaRecorder()!\n - Ended -", Toast.
    LENGTH_LONG).show();
finish();
```

```
}

runOnUiThread(new Runnable()
{
public void run(){
try {
capture.setSelected(false);
mediaRecorder.start();
}
catch (final Exception ex) {}
}
});

readingsLinearAcc = new ArrayList<>();
readingsAccelerometer = new ArrayList<>();
readingsRotation = new ArrayList<>();

recording = true;
}
}
};

OnClickListener switchCameraListener = new
   OnClickListener()
{
@Override
public void onClick(View v)
{
if(!recording)
{
int camerasNumber = Camera.getNumberOfCameras();
if(camerasNumber > 1){
releaseCamera();
chooseCamera();
}
else{
Toast toast = Toast.makeText(myContext, "Sorry, your
   phone has only one camera!", Toast.LENGTH_LONG);
toast.show();
}
}
```

```
}
};

private boolean prepareMediaRecorder()
{

mediaRecorder = new MediaRecorder();

mCamera.unlock();
mediaRecorder.setCamera(mCamera);

mediaRecorder.setAudioSource(MediaRecorder.AudioSource.
    CAMCORDER);
mediaRecorder.setVideoSource(MediaRecorder.VideoSource.
    CAMERA);
mediaRecorder.setOrientationHint(CameraPreview.rotate);

mediaRecorder.setProfile(CamcorderProfile.get(
    CamcorderProfile.QUALITY_HIGH));

File file = new File(filePath + "/" + StartDate + "_Video
    .mp4");
try{ file.createNewFile(); Log.i(DataRecorderPackage.
    DataRecorder.class.getName(), "Video file created");
    }
catch(IOException e){
Log.e(DataRecorderPackage.DataRecorder.class.getName(), "
    ERROR CREATING VIDEO FILE \"" + filePath + "/" +
    StartDate + "_Video.mp4\"");
e.printStackTrace();
finish();
}
mediaRecorder.setOutputFile(file.getAbsolutePath());

try{
mediaRecorder.prepare();
}
catch (IllegalStateException e){
releaseMediaRecorder();
return false;
}
```

```
catch (IOException e){
releaseMediaRecorder();
return false;
}
return true;

}

public void initialize()
{
cameraPreview = (LinearLayout) findViewById(R.id.
   camera_preview);

mPreview = new CameraPreview(myContext, mCamera);
cameraPreview.addView(mPreview);

capture = (ImageView) findViewById(R.id.button_capture);
capture.setOnClickListener(captrureListener);

switchCamera = (ImageView) findViewById(R.id.
   button_ChangeCamera);
switchCamera.setOnClickListener(switchCameraListener);
capture.setSelected(true);
}

private int findFrontFacingCamera()
{
cameraId = -1;
int numberOfCameras = Camera.getNumberOfCameras();
for(int i = 0; i < numberOfCameras; i++)
{
CameraInfo info = new CameraInfo();
Camera.getCameraInfo(i, info);
if(info.facing == CameraInfo.CAMERA_FACING_FRONT)
{
cameraId = i;
cameraFront = true;
break;
}
}
return cameraId;
```

```
}

private int findBackFacingCamera()
{
cameraId = -1;
int numberOfCameras = Camera.getNumberOfCameras();
for(int i = 0; i < numberOfCameras; i++)
{
CameraInfo info = new CameraInfo();
Camera.getCameraInfo(i, info);
if(info.facing == CameraInfo.CAMERA_FACING_BACK)
{
cameraId = i;
cameraFront = false;
break;
}
}
return cameraId;
}

private void initializeCamera()
{
if(mCamera == null)
{
if(findFrontFacingCamera() < 0){
Toast.makeText(this, "No front facing camera found.",
   Toast.LENGTH_LONG).show();
switchCamera.setVisibility(View.GONE);
}
mCamera = Camera.open(findBackFacingCamera());
mPreview.refreshCamera(mCamera);
}
}

public void chooseCamera()
{
if(cameraFront)
{
cameraId = findBackFacingCamera();
if(cameraId >= 0){
mCamera = Camera.open(cameraId);
```

```
mPreview.refreshCamera(mCamera);
}
}
else
{
cameraId = findFrontFacingCamera();
if (cameraId >= 0) {
mCamera = Camera.open(cameraId);
mPreview.refreshCamera(mCamera);
}
}
}

private boolean hasCamera(Context context)
{
if (context.getPackageManager().hasSystemFeature(
   PackageManager.FEATURE_CAMERA))
return true;
else
return false;
}

private void releaseMediaRecorder()
{
if (mediaRecorder != null)
{
mediaRecorder.reset();
mediaRecorder.release();
mediaRecorder = null;
mCamera.lock();
}
}

private void releaseCamera()
{
if (mCamera != null){
mCamera.release();
mCamera = null;
}
}
```

```
private void requestCameraPermission(){
Log.w(DataRecorderPackage.DataRecorder.class.getName(), "
    Camera permission is not granted. Requesting
    permission");
final String[] permissions = new String[]{Manifest.
    permission.CAMERA};
ActivityCompat.requestPermissions(DataRecorderPackage.
    DataRecorder.this, permissions, RC_HANDLE_CAMERA_PERM
    );

}

private void requestSDCardPermission(){
Log.w(DataRecorderPackage.DataRecorder.class.getName(), "
    SDCard permission is not granted. Requesting
    permission");
final String[] permissions = new String[]{Manifest.
    permission.WRITE_EXTERNAL_STORAGE};
ActivityCompat.requestPermissions(DataRecorderPackage.
    DataRecorder.this, permissions,
    RC_HANDLE_WRITE_EXTERNAL_STORAGE_PERM);

}

private void requestRecordAudioPermission(){
Log.w(DataRecorderPackage.DataRecorder.class.getName(), "
    Record audio permission is not granted. Requesting
    permission");
final String[] permissions = new String[]{Manifest.
    permission.RECORD_AUDIO};
ActivityCompat.requestPermissions(DataRecorderPackage.
    DataRecorder.this, permissions,
    RC_HANDLE_RECORD_AUDIO_PERM);
}


@Override
public void onRequestPermissionsResult(int requestCode,
    String[] permissions, int[] grantResults)
{
switch(requestCode)
```

```
{
case RC_HANDLE_CAMERA_PERM:
if(grantResults[0] == PackageManager.PERMISSION_GRANTED){
initialize();
initializeCamera();
}
else
{
final android.app.AlertDialog.Builder builder = new
   android.app.AlertDialog.Builder(this);
builder.setMessage("This application cannot record video
   because it does not have the camera permission.");
builder.setPositiveButton(android.R.string.ok, null);
builder.show();
}
break;
case RC_HANDLE_WRITE_EXTERNAL_STORAGE_PERM:
if (grantResults[0] == PackageManager.PERMISSION_GRANTED)
requestRecordAudioPermission();
else
{
final android.app.AlertDialog.Builder builder = new
   android.app.AlertDialog.Builder(this);
builder.setMessage("This application cannot record video
   because it does not have the write external storage
   permission.");
builder.setPositiveButton(android.R.string.ok, null);
builder.show();
}
break;
case RC_HANDLE_RECORD_AUDIO_PERM:
if (grantResults[0] == PackageManager.PERMISSION_GRANTED)
requestCameraPermission();
else
{
final android.app.AlertDialog.Builder builder = new
   android.app.AlertDialog.Builder(this);
builder.setMessage("This application cannot record video
   because it does not have the record audio permission.
   ");
builder.setPositiveButton(android.R.string.ok, null);
```

```
builder.show();
}
break;
}
}
}
```

CameraPreview.java

```
package DataRecorderPackage;

import android.app.Activity;
import android.content.Context;
import android.hardware.Camera;
import android.util.Log;
import android.view.Surface;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import java.io.IOException;

public class CameraPreview extends SurfaceView implements
    SurfaceHolder.Callback
{
private SurfaceHolder mHolder;
private Camera mCamera;
public static int rotate;
private Context mContext;

public CameraPreview(Context context, Camera camera)
{
super(context);
mCamera = camera;
mHolder = getHolder();
mHolder.addCallback(this);
mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
mContext = context;
}

public void surfaceCreated(SurfaceHolder holder)
{
try
{
```

```
if(mCamera == null){
mCamera.setPreviewDisplay(holder);
mCamera.startPreview();
}
}
catch (IOException e){
Log.d(VIEW_LOG_TAG, "Error setting camera preview: " + e.
    getMessage());
}
}

public void refreshCamera(Camera camera)
{
if (mHolder.getSurface() == null)
return;

stopPreview();
setCamera(camera);
startPreview();
}

public void stopPreview()
{
try {
if(mCamera != null)
mCamera.stopPreview();
}
catch (Exception e) {
e.printStackTrace();
}
}

public void startPreview()
{
try
{
if(mCamera != null){
mCamera.setPreviewDisplay(mHolder);
mCamera.startPreview();
}
else
```

```
Log.d(VIEW_LOG_TAG, "Error starting camera preview: " );
}
catch (Exception e){
Log.d(VIEW_LOG_TAG, "Error starting camera preview: " + e
    .getMessage());
}
}

public void surfaceChanged(SurfaceHolder holder, int
    format, int w, int h){
refreshCamera(mCamera);
}

public void setCamera(Camera camera){
mCamera = camera;
setCameraRotation();
}

@Override
public void surfaceDestroyed(SurfaceHolder holder)
{
if(mCamera != null)
mCamera.release();
}

public void setCameraRotation()
{
try
{
Camera.CameraInfo camInfo = new Camera.CameraInfo();
if (DataRecorder.DataRecorder.cameraId == 0)
Camera.getCameraInfo(Camera.CameraInfo.CAMERA_FACING_BACK
    , camInfo);
else
Camera.getCameraInfo(Camera.CameraInfo.
    CAMERA_FACING_FRONT, camInfo);

int cameraRotationOffset = camInfo.orientation;
Camera.Parameters parameters = mCamera.getParameters();
```

```
int rotation = ((Activity)mContext).getWindowManager().
   getDefaultDisplay().getRotation();
int degrees = 0;
switch (rotation)
{
case Surface.ROTATION_0:
degrees = 0;
break;
case Surface.ROTATION_90:
degrees = 90;
break;
case Surface.ROTATION_180:
degrees = 180;
break;
case Surface.ROTATION_270:
degrees = 270;
break;
}
int displayRotation;
if (camInfo.facing == Camera.CameraInfo.
   CAMERA_FACING_FRONT){
displayRotation = (cameraRotationOffset + degrees) % 360;
displayRotation = (360 - displayRotation) % 360;
}
else
displayRotation = (cameraRotationOffset - degrees + 360)
   % 360;
mCamera.setDisplayOrientation(displayRotation);
if (camInfo.facing == Camera.CameraInfo.
   CAMERA_FACING_FRONT)
rotate = (360 + cameraRotationOffset + degrees) % 360;
else
rotate = (360 + cameraRotationOffset - degrees) % 360;
parameters.set("orientation", "portrait");
parameters.setRotation(rotate);
mCamera.setParameters(parameters);
}
catch (Exception e) {}
}
}
```

activityDataRecorder.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com
   /apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal" >
    <LinearLayout
        android:id="@+id/camera_preview"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_weight="1"
        android:orientation="horizontal"
        android:layout_centerInParent="true"/>
    <LinearLayout
        android:id="@+id/buttonsLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:orientation="horizontal"
        android:layout_alignParentBottom="true"
        android:weightSum="2"
        android:layout_marginBottom="10dp">
        <ImageView
            android:id="@+id/button_ChangeCamera"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:src="@mipmap/camera_switch_icon"
            android:layout_gravity="left" />
        <ImageView
            android:id="@+id/button_capture"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/
               video_start_stop_selector"
            android:layout_weight="1"
            android:layout_gravity="right" />
    </LinearLayout>
</RelativeLayout>
```

Bibliografia

[1] Simon Baker, Ralph Gross e Iain Matthews. «Lucas-Kanade 20 Years On: A Unifying Framework: Part 3». In: *International Journal of Computer Vision* 56 (2002), pp. 221–255.

[2] Simon Baker, Ralph Gross e Iain Matthews. «Lucas-Kanade 20 Years On: A Unifying Framework: Part 4». In: *International Journal of Computer Vision* 56 (2004), pp. 221–255.

[3] Simon Baker e Iain Matthews. *Implementazione Matlab di Inverse Compositional Algorithm*. URL: http://www.ri.cmu.edu/research_project_detail.html?project_id=515&menu_id=261 (visitato il 04/12/2016).

[4] Simon Baker e Iain Matthews. «Lucas-Kanade 20 Years On: A Unifying Framework». In: *Int. J. Comput. Vision* 56.3 (feb. 2004), pp. 221–255. ISSN: 0920-5691. DOI: 10.1023/B:VISI.0000011205.11775.fd. URL: http://dx.doi.org/10.1023/B:VISI.0000011205.11775.fd (visitato il 04/12/2016).

[5] Simon Baker et al. «Lucas-Kanade 20 Years On: A Unifying Framework: Part 2». In: *International Journal of Computer Vision* 56 (2003), pp. 221–255.

[6] Adrien Bartoli. «Groupwise geometric and photometric direct image registration». In: *in Proc. British Machine Vision Conference, (BMVC06*, p. 157.

[7] Miguel Bordallo López et al. «Interactive multi-frame reconstruction for mobile devices». In: *Multimedia Tools and Applications* 69.1 (2014), pp. 31–51. ISSN: 1573-7721. DOI: 10.1007/s11042-012-1252-4. URL: http://dx.doi.org/10.1007/s11042-012-1252-4 (visitato il 04/12/2016).

[8] *FIR Filter Design*. URL: https://it.mathworks.com/help/signal/ug/fir-filter-design.html (visitato il 04/12/2016).

[9] C. M. Huang, S. W. Lin e J. H. Chen. «Efficient Image Stitching of Continuous Image Sequence With Image and Seam Selections». In: *IEEE Sensors Journal* 15.10 (ott. 2015), pp. 5910–5918. ISSN: 1530-437X. DOI: 10.1109/JSEN.2015.2449879.

[10] JULIUS O. SMITH III. *INTRODUCTION TO DIGITAL FILTERS*. URL: https://ccrma.stanford.edu/~jos/filters/filters.html (visitato il 04/12/2016).

[11] *Introduction to Filter Designer*. URL: https://it.mathworks.com/help/signal/examples/introduction-to-filter-designer.html (visitato il 04/12/2016).

[12] Takahiro Ishikawa, Iain Matthews e Simon Baker. *Efficient image alignment with outlier rejection*. Carnegie Mellon University, The Robotics Institute, 2002.

[13] H. K. Kim et al. «A content-aware image stitching algorithm for mobile multimedia devices». In: *IEEE Transactions on Consumer Electronics* 57.4 (nov. 2011), pp. 1875–1882. ISSN: 0098-3063. DOI: 10.1109/TCE.2011.6131166.

[14] Bruce D. Lucas e Takeo Kanade. «An Iterative Image Registration Technique with an Application to Stereo Vision». In: *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI'81. Vancouver, BC, Canada: Morgan Kaufmann Publishers Inc., 1981, pp. 674–679. URL: http://dl.acm.org/citation.cfm?id=1623264.1623280 (visitato il 04/12/2016).

[15] Stefano Mazzocca. *Stima della posizione in dispositivi mobili, con applicazione a uno strumento musicale virtuale.* Ott. 2015. URL: http://tesi.cab.unipd.it/49624/ (visitato il 04/12/2016).

[16] Sanjit K. K. Mitra. *Digital Signal Processing: A Computer-Based Approach*. 2nd. McGraw-Hill Higher Education, 2000. ISBN: 0072321059.

[17] *Motion Sensors*. URL: https://developer.android.com/guide/topics/sensors/sensors_motion.html (visitato il 04/12/2016).

[18] John G. Proakis e Dimitris K. Manolakis. *Digital Signal Processing (4th Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006. ISBN: 0131873741.

[19] Zhong Qu et al. «The Improved Algorithm of Fast Panorama Stitching for Image Sequence and Reducing the Distortion Errors». In: *Mathematical Problems in Engineering* 2015 (2015). DOI: 10.1155/2015/428076.

[20] H. S. Qureshi et al. «Quantitative quality assessment of stitched panoramic images». In: *IET Image Processing* 6.9 (dic. 2012), pp. 1348–1358. ISSN: 1751-9659. DOI: 10.1049/iet-ipr.2011.0641.

[21] Javier Sánchez. «The Inverse Compositional Algorithm for Parametric Registration». In: *Image Processing On Line* 6 (2016), pp. 212–232. DOI: 10.5201/ipol.2016.153.

[22] *Sensors Overview*. URL: https://developer.android.com/guide/topics/sensors/sensors_overview.html (visitato il 04/12/2016).

[23] Uri Shaked. *Zampoña*. URL: https://play.google.com/store/apps/details?id=com.salsa4fun.zampona (visitato il 04/12/2016).

[24] Smule. *Ocarina*. URL: https://itunes.apple.com/us/app/ocarina/id293053479?mt=8 (visitato il 04/12/2016).

[25] Robin de Vries. *Mobile image stitching using ad-hoc networks on Android*. Giu. 2014. URL: https://esc.fnwi.uva.nl/thesis/centraal/files/f47916134.pdf (visitato il 04/12/2016).

[26] Y. Xiong e K. Pulli. «Fast panorama stitching for high-quality panoramic images on mobile phones». In: *IEEE Transactions on Consumer Electronics* 56.2 (mag. 2010), pp. 298–306. ISSN: 0098-3063. DOI: 10.1109/TCE.2010.5505931.

[27] C. H. Yuan et al. «Fast Image Blending and Deghosting for Panoramic Video». In: *2013 Ninth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*. Ott. 2013, pp. 104–107. DOI: 10.1109/IIH-MSP.2013.35.