Università degli studi di Urbino "Carlo Bo"


Tesi Dottorale

# Analisi e Modelli per il Monitoraggio del Manto Stradale

*Autore:*
Giacomo Alessandroni

*Tutori:*
Prof. Emanuele Lattanzi
Prof. Alberto Carini

*Una tesi presentata a compimento parziale per il conseguimento del titolo di Dottorato di ricerca*

*in*

Scienza della Complessità
Dipartimento di Scienze Pure e Applicate (DiSPeA)

a.a. 2015/2016
(XXVIII ciclo)

# Dichiarazione di paternità

Io, Giacomo Alessandroni, dichiaro che questa tesi dal titolo, "Analisi e Modelli per il Monitoraggio del Manto Stradale" e il lavoro presentato in esso è mio. Confermo che:

- Questo lavoro è stato svolto interamente o prevalentemente, durante la candidatura per un dottorato di ricerca presso l'Università degli studi di Urbino "Carlo Bo".
- Qualora sia stata precedentemente presentata una qualsiasi parte di questa tesi per una laurea o qualsiasi altro titolo presso l'Università degli studi di Urbino "Carlo Bo" o qualsiasi altra istituzione, questo è stato chiaramente indicato.
- Dove ho consultato il lavoro pubblicato da altri, questo è sempre chiaramente attribuito.
- Dove ho citato l'altrui lavoro, la fonte è sempre stata indicata. Fatta eccezione per le suddette citazioni, questa tesi è interamente frutto del mio lavoro.
- Ho preso atto di tutte le principali fonti di aiuto.
- Dove la tesi si basa sul lavoro svolto dal sottoscritto insieme ad altri, ho chiarito esattamente ciò che è stato fatto da altri e quello che io stesso ho realizzato.

Firma: _____

Data: _____

*"Innovation is everything. When you're on the forefront, you can see what the next innovation needs to be. When you're behind, you have to spend your energy catching up."*

Robert Norton Noyce

UNIVERSITÀ DEGLI STUDI DI URBINO "CARLO BO"

Informatica Applicata
Dipartimento di Scienze Pure e Applicate (DiSPeA)

Dottorato di ricerca

**Analisi e Modelli per il Monitoraggio del Manto Stradale**

di Giacomo Alessandroni

# *Abstract*

La profonda conoscenza delle condizioni del manto stradale è condizione essenziale per la sicurezza nella guida. Inoltre, le condizioni dello stato del manto stradale sono soggette a variazioni nel tempo dovute a usura, fenomeni atmosferici, eventi accidentali, cedimenti del terreno sottostante e numerosi altri fattori, non per ultimo il rifacimento del manto stradale stesso ad opera degli enti gestori.

La visione di un quadro dettagliato e aggiornato delle condizioni della rete viaria è pertanto un requisito essenziale per poter pianificare in maniera ottimale gli interventi di manutenzione e fronteggiare le eventuali emergenze. Conoscere le condizioni delle strade che si stanno percorrendo consente, infine, agli automobilisti e ai motociclisti di guidare in maggior sicurezza.

I dispositivi mobili di ultima generazione (smartphone) sono dotati di ricevitori GPS i quali consentono di implementare le funzioni di navigatori satellitari. Inoltre hanno al proprio interno accelerometri triassiali e bussole: dispositivi attraverso i quali è possibile estrarre—dopo opportuna elaborazione software dei dati, anch'essa possibile all'interno del dispositivo mobile—le caratteristiche del manto stradale. La sensibilità e la frequenza di campionamento degli accelerometri (circa 100 Hz, recentemente elevata—in alcuni modelli—fino a 300 Hz) sono tali da consentire di rilevare tutte le vibrazioni che noi stessi avvertiamo.

Nel progetto di ricerca *SmartRoadSense*, un dispositivo mobile viene utilizzato come navigatore satellitare e montato su un supporto rigido all'interno di un'autovettura. Così facendo, il suo accelerometro viene utilizzato per monitorare le asperità del manto stradale che provocano vibrazioni non riconducibili al funzionamento del motore stesso.

L'analisi dei segnali così campionati permette di effettuare una classificazione accurata della tipologia del terreno, riconoscere cedimenti o lesioni dell'asfalto. La dipendenza di

tali valutazioni dalla velocità del veicolo sono opportunamente compensate grazie alle informazioni fornite dal sistema di navigazione satellitare. Inoltre, le capacità di calcolo e comunicazione degli attuali dispositivi mobili consentono l'elaborazione—anche in tempo reale—dei dati e la trasmissione d'informazioni georeferenziate a un server centrale. Il server svolge la funzione di aggregare i dati forniti dai diversi dispositivi mobili operanti sul territorio e renderli disponibili per una visualizzazione più agevole su mappe stradali a beneficio degli enti gestori e degli automobilisti.

L'attendibilità dei dati è garantita dall'accuratezza degli strumenti e dal confronto delle rilevazioni fornite dai molteplici utenti che hanno percorso il medesimo tragitto stradale. La collaborazione con aziende di trasporto pubblico e l'installazione della tecnologia in oggetto su mezzi pubblici, come autobus di linea, rende il monitoraggio ulteriormente attendibile e sistematico.

Il progetto SmartRoadSense propone la realizzazione di un sistema collaborativo per di rilevamento automatico delle condizioni del manto stradale basato sull'uso combinato degli accelerometri triassiali e dei ricevitori GPS di cui sono dotati i moderni dispositivi mobili.

In particolare il progetto—nella sua interezza—può essere schematizzato in tre grandi blocchi principali:

1. Lo sviluppo di modelli matematici basati su algoritmi predittivi. Questi modelli—tramite l'elaborazione numerica dei dati forniti dall'accelerometro e la velocità fornita dal sensore GPS—permettono la classificazione automatica delle condizioni del manto stradale, il tutto senza alcun intervento dell'utente finale.

2. Lo sviluppo di un'applicazione (attualmente distribuita per Android, distribuita gratuitamente online) la quale invia i dati della rugosità del manto stradale georeferenziati a un server centrale.

3. Lo sviluppo di un'applicazione lato server che riceve i dati inviati dai vari dispositivi mobili e li aggrega eseguendone una media ponderata nel tempo e nello spazio. Questa applicazione restituisce i dati in formato aperto e aggregato, facendo particolare attenzione alla riservatezza degli utenti (con un approccio *zero–knowledge*) e consentendo agli utilizzatori finali di rielaborarli.

Il progetto di ricerca è stato condotto presso il Dipartimento di Scienze Pure e Applicate (DiSPeA) dell'Università di Urbino coinvolgendo numerosi studiosi con competenze informatiche, matematiche, elettroniche e geofisiche.

In questo lavoro viene presentata l'analisi del manto stradale utilizzata nel progetto SmartRoadSense, i modelli teorici sviluppati, le analisi sui dati fornite dal progetto stesso nelle sue varie fasi.

# *Ringraziamenti*

Il primo ringraziamento va ai numerosi contributori anonimi del progetto SmartRoadSense per l'acquisizione dati, sui quali questo progetto si fonda e a tutti i membri del gruppo di ricerca SmartRoadSense per i loro aiuti preziosi, le utili discussioni e le numerose critiche costruttive.

Un ringraziamento particolare va ai miei tutori Prof. Alberto Carini e Prof. Emanuele Lattanzi per avermi introdotto passo dopo passo nell'affascinante mondo del *signal processing*, per avermi guidato ed aiutato nelle mie ricerche, nella mia crescita professionale e—non meno importante—per il loro lato umano.

Infine, desidero ringraziare anche il Prof. Alessandro Bogliolo coordinatore del progetto SmartRoadSense e Lorenz Cuno Klopfenstein e Saverio Delpiori, amici, prima che compagni di studi, per i loro preziosi consigli disinteressati.

# Indice

**Dichiarazione di paternità** **iii**

**Abstract** **vii**

**Ringraziamenti** **xi**

**Indice** **xii**

**Elenco delle figure** **xvii**

**Elenco delle tabelle** **xix**

**Abbreviazioni** **xxi**

**Costanti fisiche** **xxiii**

**Simboli** **xxv**

**1 Introduzione al soggetto della tesi** **1**
1.1 Progetto SmartRoadSense . . . . . 1
1.2 Stato dell'arte nell'analisi del manto stradale . . . . . 4

**2 Informazioni di base e teoria** **9**
2.1 Analisi dei dati tramite modelli LPC . . . . . 10
2.1.1 Modelli autoregressivi del prim'ordine . . . . . 11
2.1.2 Modelli Autoregressivi oltre il prim'ordine . . . . . 12
2.1.3 Matrice di Toeplitz . . . . . 13
2.2 Potenza dell'accelerazione verticale . . . . . 14
2.3 Modello teorico del manto stradale . . . . . 15
2.4 Densità spettrale di potenza del profilo stradale . . . . . 16
2.5 Densità spettrale di potenza dell'accelerazione verticale . . . . . 18
2.6 Introduzione al modello "Quarter–car" . . . . . 19
2.6.1 Risposta del sistema agli smorzamenti . . . . . 21
2.6.2 Modello matematico delle sospensioni . . . . . 22
2.6.3 Soluzione del modello matematico . . . . . 24

2.6.4 Parametri tipici del modello Quarter–car . . . . . 26
2.6.5 Risposte in frequenza dei modelli Quarter–car . . . . . 26
2.7 Calcolo della potenza rilevata dall'accelerometro . . . . . 33
2.8 Funzione gamma . . . . . 35

**3 Sistema di analisi su dispositivi mobili 37**
3.1 Inizializzazione . . . . . 38
3.2 Calcolo della potenza dell'errore di predizione . . . . . 39
3.3 Stima del modello autoregressivo . . . . . 41
3.4 Indice di rugosità . . . . . 43
3.5 Costo computazionale dell'algoritmo . . . . . 44

**4 Primi esperimenti su segnali acquisiti 47**
4.1 Primi dati sperimentali . . . . . 48
4.2 Monitoraggio delle diverse accelerazioni . . . . . 49
4.3 Stima dell'ordine del filtro predittivo . . . . . 51
4.4 Calcolo dei coefficienti di autocorrelazione $R(k)$ . . . . . 53
4.5 Finestratura dei dati . . . . . 54
4.6 Overlapping . . . . . 55
4.7 Media mobile dei dati . . . . . 56

**5 Esperimenti su moli di dati 59**
5.1 Tipologie di esperimenti eseguiti . . . . . 59
5.2 Definizione della mappa colorimetrica . . . . . 61
5.3 Esperimenti su manto stradale omogeneo . . . . . 62
5.4 Esperimenti su manto stradale eterogeneo . . . . . 65
5.4.1 Analisi su autostrade . . . . . 67
5.4.2 Analisi su superstrade . . . . . 68
5.4.3 Analisi su strade primarie . . . . . 68
5.4.4 Analisi su strade secondarie . . . . . 70
5.5 Confronto dei risultati su manto stradale eterogeneo . . . . . 71
5.6 Modello indipendente dalla velocità . . . . . 74

**6 Conclusioni e scenari aperti 77**
6.1 Risultati ottenuti . . . . . 77
6.2 Scenari aperti . . . . . 78

**A Codice sorgente SmartRoadSense 81**
A.1 Codice MATLAB© . . . . . 81
A.1.1 `srs_test_no_plot.m` . . . . . 81
A.1.2 `extract_gps_data.m` . . . . . 84
A.1.3 `ppe.m` . . . . . 86
A.1.4 `ar_model.m` . . . . . 87
A.1.5 `autocorr2.m` . . . . . 88
A.1.6 `levinson_durbin.m` . . . . . 89
A.2 Codice Java . . . . . 90

A.2.1 `Manifest` . . . . . 91
A.2.2 `.project` . . . . . 91
A.2.3 `.classpath` . . . . . 91
A.2.4 `ComputationListener.java` . . . . . 91
A.2.5 `Buffer.java` . . . . . 92
A.2.6 `DataEntry.java` . . . . . 99
A.2.7 `DataHelpers.java` . . . . . 103
A.2.8 `Engine.java` . . . . . 105
A.2.9 `LogListener.java` . . . . . 112
A.2.10 `MathHelpers.java` . . . . . 113
A.2.11 `Matrix.java` . . . . . 117
A.2.12 `PrintHelpers.java` . . . . . 120
A.2.13 `Result.java` . . . . . 123
A.2.14 `ResultSimpleMean.java` . . . . . 125
A.2.15 `Settings.java` . . . . . 126
A.2.16 `DataLoader.java` . . . . . 129
A.2.17 `Program.java` . . . . . 132
A.2.18 `ResultDumper.java` . . . . . 135

**Bibliografia** **137**

# Elenco delle figure


2.1 Pseudocodice per la ricorsione Levinson–Durbin. . . . . 14
2.2 Sistema di riferimento. . . . . 17
2.3 Modello matematico del Quarter–car. . . . . 20
2.4 Analogo meccanico delle sospensioni scomposto. . . . . 22
2.5 Risposta in frequenza di Davis e Thompson. . . . . 28
2.6 Risposta in frequenza di Butsuen. . . . . 29
2.7 Risposta in frequenza di Gillespie. . . . . 30
2.8 Risposta in frequenza di Fialho e Balas. . . . . 30
2.9 Risposta in frequenza di Verros, Natsiavas e Papadimitriou. . . . . 31
2.10 Risposta in frequenza di Allison con i parametri “comfort”. . . . . 31
2.11 Risposta in frequenza di Allison con i parametri “handling”. . . . . 32
2.12 Risposta in frequenza di Salem e Aly. . . . . 32
2.13 Andamento di $P(v)$ al variare della velocità $v$ e del polo $p$. . . . . 34
2.14 Andamento di $\gamma$ al variare della velocità $v$ e del polo $p$. . . . . 35

3.1 Inizializzazione del programma. . . . . 38
3.2 Rimozione dei valori medi dalle accelerazioni. . . . . 39
3.3 Calcolo delle potenze dell’errore di predizione. . . . . 39
3.4 Calcolo delle potenze dell’errore di predizione. . . . . 40
3.5 Calcolo del filtro autoregressivo. . . . . 41
3.6 Calcolo dell’autocorrelazione senza trasformata di Fourier. . . . . 42
3.7 Calcolo del residuo finale di predizione con media mobile. . . . . 43
3.8 Calcolo della media mobile aritmetica. . . . . 44

4.1 Percorsi analizzati nella prima fase sperimentale. . . . . 49
4.2 Componenti dell’accelerazione $A_x$, $A_y$ e $A_z$ per il percorso `SRS-01`. . . . . 50
4.3 Frammento di codice per le analisi riportate in Figura 4.4 e Figura 4.5. . . 52
4.4 Analisi del FPE sui tracciati al variare dell’ordine del modello AR. . . . . 53
4.5 Variazioni del FPE rispetto all’ordine del modello AR. . . . . 53
4.6 Finestra di Hamming con $\alpha = 0.54$, $\beta = 0.46$ per 150 campioni. . . . . 55
4.7 Modello iterativo dell’overlap. Nella zona in cui vi è intersezione i campioni sono i medesimi. . . . . 56
4.8 $R_I$ (curva nera) e indice di rugosità smooth $P_{\mathrm{PE}}$ (curva rossa) di un veicolo che procede a 40 km/h. . . . . 57

5.1 Mappatura stradale effettuata tramite il progetto SmartRoadSense. . . . . 60
5.2 Frammento di codice per il calcolo della scala colorimetrica utilizzata nel progetto SmartRoadSense. . . . . 61
5.3 Confronto dei campionamenti di $|A_y|^2$ e $P_{\mathrm{PE}}$ su autostrada. . . . . 63

5.4 Confronto dei campionamenti di $|A_y|^2$ e $P_{\mathrm{PE}}$ su strada statale. . . . . 64
5.5 $P_{\mathrm{PE}}$ rispetto alla velocità e courve fit della legge gamma relativa alla classe "autostrada". . . . . 67
5.6 Istogramma della frequenza dei campioni raccolti per velocità relativi alla classe "autostrada". . . . . 68
5.7 $P_{\mathrm{PE}}$ rispetto alla velocità e courve fit della legge gamma relativa alla classe "superstrada". . . . . 69
5.8 Istogramma della frequenza dei campioni raccolti per velocità relativi alla classe "superstrada". . . . . 69
5.9 $P_{\mathrm{PE}}$ rispetto alla velocità e courve fit della legge gamma relativa alla classe "strada primaria". . . . . 70
5.10 Istogramma della frequenza dei campioni raccolti per velocità relativi alla classe "strada primaria". . . . . 70
5.11 $P_{\mathrm{PE}}$ rispetto alla velocità e courve fit della legge gamma relativa alla classe "strada secondaria". . . . . 71
5.12 Istogramma della frequenza dei campioni raccolti per velocità relativi alla classe "strada secondaria". . . . . 71
5.13 Fusione della Figura 5.5, Figura 5.7, Figura 5.9 e Figura 5.11 . . . . . 72
5.14 Andamento dei campioni rispetto dalla velocità per tutti i dati analizzati. 72
5.15 Confronto dei diversi curve fitting. . . . . 74
5.16 Confronto delle mappa cartografiche prima e dopo la correzione proposta per velocità e manto stradale. . . . . 76
5.17 Tipologia di manti stradali secondo la classificazione OpenStreetMap. . . 76

# Elenco delle tabelle


2.1 Principali pubblicazioni relative ai modelli Quarter–car . . . . . 26
2.2 Dati forniti da B. R. Davis e A. G. Thompson . . . . . 26
2.3 Dati forniti da Tetsuro Butsuen . . . . . 27
2.4 Dati forniti da Thomas D. Gillespie . . . . . 27
2.5 Dati forniti da Ian Fialho e Gary J. Balas . . . . . 27
2.6 Dati forniti da Natsiavas Verros e Papadimitriou . . . . . 27
2.7 Dati forniti da James T. Allison . . . . . 27
2.8 Dati forniti da M. M. M. Salem e A. Aly Ayman . . . . . 28
2.9 Valori numerici della risposta in frequenza del modello “Quarter–car” . . 28

3.1 Esempio di scomposizione di un vettore con dieci elementi per ogni finestra e overlap di due elementi. . . . . 40
3.2 Calcolo dell'autocorrelazione del vettore $\vec{x} = (1\ 2\ 3\ 4\ 5\ 6)$ . . . . . 42
3.3 Numero di moltiplicazioni necessarie per il calcolo della potenza del residuo di predizione per ciascuna componente dell'accelerazione. . . . . 45

4.1 Numero di campioni utili per ogni acquisizione. . . . . 49

5.1 Tavola colorimetrica generata dal codice di Figura 5.2 . . . . . 62
5.2 Curve fitting dei parametri $P(v) = \hat{q}v^{\gamma}$ per manti stradali omogenei. . . . 65
5.3 Tipologie di strade secondo la classificazione OpenStreetMap. . . . . 65
5.4 Interpolazione dei parametri $P(v) = \hat{q}v^{\gamma}$ per manti stradali eterogenei. . . 73

# Abbreviazioni

| | |
|---|---|
| **ACF** | **A**uto**c**orrelation **F**unction |
| **AR** | **A**uto**R**egressive |
| **CI** | **C**onfidence **I**nterval |
| **CSV** | **C**omma–**S**eparated **V**alues |
| **FFT** | **F**ast **F**ourier **T**ransform |
| **FPE** | **F**inal **P**rediction **E**rror |
| **GIS** | **G**eographical **I**nformation **S**ystem |
| **GPS** | **G**lobal **P**osition **S**ystem |
| **IRI** | **I**nternational **R**oughness **I**ndex |
| **LMA** | **L**aplace **M**oving **A**verage |
| **LPC** | **L**inear **P**redictive **C**oding |
| **LPF** | **L**ow–**P**ass **F**ilter |
| **PPE** | **P**ower **P**rediction **E**rror |
| **PSD** | **P**ower **S**pectral **D**ensity |
| **RI** | **R**oughness **I**ndex |
| **RR** | **R**ide **R**ate |
| **WGN** | **W**hite **G**ussian **N**oise |
| **WSS** | **W**ide **S**ense **S**tationary |

# Costanti fisiche

Accelerazione di gravità $g = 9.80665\ \mathrm{m/s^2}$

# Simboli

| | | |
|---|---|---|
| $a$ | accelerazione | m $s^{-2}$ |
| $C$ | costante di smorzamento | N s $m^{-1}$ |
| $K$ | costante elastica | N $m^{-1}$ |
| $f$ | frequenza | Hz |
| $m$ | massa | kg |
| $P$ | potenza | W (J $s^{-1}$) |
| $R$ | autocorrelazione | — |
| $t$ | tempo | s |
| $v$ | velocità | m $s^{-1}$ |
| $x$ | spazio | m |
| | | |
| $\lambda$ | parametri modello $AR$ | — |
| $\Lambda$ | pulsazione spaziale | rad $m^{-1}$ |
| $\Omega$ | pulsazione angolare | rad $s^{-1}$ |

*Dedico questo lavoro a mio figlio Giovanni e mia moglie Milena, per avermi dato questa opportunità e per la loro infinita pazienza. Senza di loro, senza il loro tempo, le loro attese e tantissime altre cose che soltanto loro conoscono nel profondo, non avrei mai completato questo lavoro.*

*Dedico questo lavoro anche a tutte quelle persone che, col cuore, mi hanno seguito in questo passo. Grazie a tutti voi che avete creduto in me fin dal primo giorno.*

# Capitolo 1

# Introduzione al soggetto della tesi

## 1.1 Progetto SmartRoadSense

SmartRoadSense [1] è un progetto di ricerca collaborativo. La sua finalità è il monitoraggio del manto stradale.

La rete viaria nazionale e internazionale è una componente strategica essenziale nell'economia di un Paese e nell'economia globale. Dal suo buon stato derivano benefici economici e sociali. L'adeguata manutenzione di questa infrastruttura è di fondamentale importanza per preservare e incrementare questi benefici (si pensi che i Paesi OCSE hanno una media di reti stradali di circa 500,000 km [2]). Investimenti inadeguati nella rete stradale si traducono in un impoverimento dell'economia e del benessere sociale. Senza un'adeguata manutenzione, l'elevatissimo valore insito nella rete stradale si erode velocemente.

Il monitoraggio delle condizioni del profilo stradale è un'attività fondamentale nella gestione delle infrastrutture di trasporto. A tal fine, in letteratura, sono state proposte molte soluzioni per monitorare automaticamente la qualità del manto stradale. La maggior parte di queste fanno uso di costosi sensori incorporati in veicoli, altre si concentrano principalmente sul rilevamento di specifiche anomalie durante l'attività di monitoraggio. In questo progetto, si propone un sistema di controllo della qualità del manto stradale collaborativo. L'architettura complessiva comprende un'applicazione per dispositivi mobili, un database connesso con un sistema informativo territoriale (GIS, Geographic Information System) e una visualizzazione finale dei dati aggregati.

La condizione del manto stradale viene descritta tramite un parametro di rugosità, calcolato con algoritmi di elaborazione numerica del segnale direttamente sui dispositivi

mobili. I valori di rugosità così ottenuti vengono successivamente trasmessi e memorizzati tramite un GIS, il quale consente l'elaborazione di tracce aggregate e la relativa visualizzazione delle condizioni del fondo stradale. L'approccio proposto è un sistema completamente integrato per il monitoraggio del manto stradale, adatto a soluzioni scalabili, di crowdsourcing, e collaborative.

Questo progetto introduce un nuovo sistema di misurazione della qualità del manto stradale. Un sistema basato su sensori a basso costo e un modello matematico concepito per estrarre un indice di qualità dei dati dai sensori. Infine un sistema di post–elaborazione dati, il quale consente di misurare, raccogliere e aggregare la media della stima delle asperità stradali.

Il primo contributo principale e originale nel progetto SmartRoadSense di questo lavoro di tesi è l'algoritmo di analisi della rugosità del manto stradale. L'algoritmo si basa sullo studio della potenza del residuo di predizione di un modello autoregressivo. L'idea di fondo è che quando l'asfalto è uniforme, le accelerazioni rilevate dall'accelerometro incorporato nel dispositivo mobile sono della stessa natura, pertanto predicibili. La potenza del residuo di predizione aumenta all'aumentare della rugosità del manto stradale. Quando—nel manto stradale—si presentano anomalie, si hanno valori più elevati nella potenza del residuo di predizione. Attraverso l'analisi di questi dati si ottiene un indice di rugosità del manto stradale. In una seconda fase di analisi e studio dei risultati ottenuti, questo algoritmo è stato ulteriormente migliorato con l'introduzione della dipendenza della rugosità del manto stradale dalla velocità del veicolo e dal tipo di manto stradale su cui si viaggia.

SmartRoadSense [3–7], come accennato, è un progetto di ricerca collaborativo. Propone un approccio collaborativo e a basso costo per il monitoraggio delle condizioni del manto stradale. SmartRoadSense controlla la qualità del manto stradale utilizzando un dispositivo mobile (smartphone) rigidamente ancorato all'interno della cabina auto. In particolare, i dati provenienti dai sensori del dispositivo mobile, quali accelerometro triassiale e navigatore GPS, sono elaborati al fine di estrarre un indice di rugosità, caratterizzante la superficie stradale. In particolare, l'analisi con codifica predittiva lineare (LPC, Linear Predictive Coding) viene eseguita sui segnali accelerometrici trama per trama. Successivamente, la potenza media del residuo di predizione viene utilizzata per stimare l'indice di rugosità della strada. Le coordinate GPS, la velocità del veicolo e l'indice di rugosità vengono trasmessi periodicamente a un server remoto il quale aggrega le informazioni provenienti dai diversi dispositivi mobili e ne compensa le diverse sensibilità. Infine, i dati ottenuti vengono memorizzati in un sistema informativo territoriale (GIS, Geographic Information System). Nel database ciascun punto viene calcolato

come la media ponderata, nel tempo e nello spazio, dei campioni ottenuti sulla stessa strada.

Altri progetti collaborativi proposti in letteratura si concentrano sulla strada e il monitoraggio del traffico. Molti studi hanno esplorato l'uso di accelerometri sia indipendenti, sia incorporati nei dispositivi mobili, per rilevare dossi, buche o strade dissestate. Tuttavia, la maggior parte di questi studi si focalizzano principalmente sull'identificazione e la localizzazione delle singole anomalie (o classi di anomalie). Diversamente, SmartRoadSense differisce da questi approcci e li integra, in quanto è volto a estrarre un parametro di qualità che caratterizza tutta la strada e lo stato della degradazione della pavimentazione.

Questo lavoro studia l'influenza della velocità del veicolo sull'accelerazione verticale del dispositivo mobile interno al veicolo stesso, di conseguenza, sull'indice rugosità di SmartRoadSense. Un punto materiale che segue da vicino il profilo stradale con velocità costante, percepisce una potenza dell'accelerazione verticale proporzionale alla quarta potenza della sua velocità [8]. Pertanto, la velocità del veicolo è un parametro molto importante, il cui effetto deve essere tenuto in considerazione durante l'elaborazione dei dati dell'accelerometro o durante la fase d'aggregazione dei dati provenienti da diversi dispositivi mobili. L'influenza della velocità sui dati dell'accelerometro è molto diversa da quella osservata considerando un punto materiale. Infatti, l'accelerometro incorporato in un dispositivo mobile rileva il profilo altimetrico stradale attraverso gli pneumatici, il sistema di sospensione, e l'accoppiamento meccanico con l'autoveicolo. La risposta di queste parti meccaniche influenza la dipendenza della velocità. Attraverso un modello teorico del profilo stradale, degli pneumatici e delle sospensioni, questo lavoro mostra come un dispositivo mobile, rigidamente ancorato al veicolo, rilevi una potenza di accelerazione verticale correlata alla velocità del veicolo secondo una legge gamma. La stessa legge gamma lega la misurazione dell'indice rugosità di SmartRoadSense rispetto alla velocità del veicolo. I risultati teorici vengono confermati da esperimenti effettuati con diversi dispositivi mobili utilizzando i dati raccolti nell'ambito del progetto SmartRoadSense.

Questo lavoro propone una nuova metodologia per migliorare la stima dell'indice di rugosità SmartRoadSense. Viene proposto un originale modello dell'indice di rugosità sulla base di uno studio teorico e dei dati del database SmartRoadSense (più di 6 milioni d'indici di rugosità basati su 1.8 miliardi di singole accelerazioni). I principali contributi sono i seguenti:

1. Modellando l'intero processo di acquisizione dei dati, la metodologia proposta migliora l'affidabilità dell'indice di rugosità ottenuto con SmartRoadSense.

2. Grazie a una linearizzazione originale del modello proposto, l'algoritmo è computazionalmente efficiente.

3. Lo studio esplora diversi tipologie di manti stradali, come autostrade, superstrade, strade statali, strade provinciali, e tutta la rete viaria italiana.

I risultati numerici ottenuti tramite l'analisi sia di singole vetture su singoli manti stradali, sia su popolazioni statistiche di grandi dati, sottolineano l'accuratezza del modello proposto ed evidenziano la nitidezza dei risultati teorici.

In una prima fase sperimentale, SmartRoadSense è stato progettato con un modello matematico indipendente dalla velocità. I risultati ottenuti da questo studio, evidenziano come l'indice di rugosità possa essere corretto mediante una legge gamma dipendente da due parametri correlati alla velocità della vettura e alla tipologia di asfalto.

I risultati sperimentali mostrano che la legge gamma dipende dal tipo di manto stradale, cioè, dall'asfalto. Diverse leggi gamma sono state ottenute per le autostrade, le superstrade, le strade statali e le strade provinciali. Le leggi gamma stimate attraverso i dati contenuti nel database, possono essere utilizzate per migliorare l'affidabilità dei dati di aggregazione di SmartRoadSense, riducendo così l'effetto della velocità del veicolo sui dati aggregati.

## 1.2 Stato dell'arte nell'analisi del manto stradale

Diversi studi, iniziati verso la fine degli anni '50, hanno dimostrato che la qualità della superficie stradale è il criterio più importante per la valutazione del percorso stradale e del suo comfort nella guida. Il deterioramento della superficie stradale, conduce a opere di manutenzione ordinarie e straordinarie talvolta molto costose, non solo per il rifacimento dell'asfalto. Ogni veicolo aumenta i suoi costi operativi: in primo luogo dovuti a un maggior consumo di carburante e, non meno importante, con maggiori emissioni inquinanti nell'ambiente. In secondo luogo, a causa dei cedimenti del manto stradale, dovuti ai carichi dinamici dei veicoli. Infine, strade dissestante conducono alla rottura di sospensioni e altri componenti meccanici degli autoveicoli [9–11].

L'elevazione del profilo stradale è stata modellata utilizzando, in primo luogo, funzioni impulsive, onde triangolari o segnali sinusoidali [12]. Solo in studi più accurati [8], come somme di segnali sinusoidali generati casualmente, con differenti ampiezze e fasi. In questi studi [8], è stato mostrato che la Densità Spettrale di Potenza (PSD) di una tipica superficie stradale ha una caratteristica passa–basso, la quale decresce con l'aumentare della frequenza spaziale.

Recentemente, sono stati proposti altri modelli stocastici, con caratteristiche passa–basso, al fine di descrivere la casualità dei profili stradali misurati, come processi gaussiani non omogenei [13], processi omogenei laplaciani in media mobile (LMA, Laplace Moving Average) e combinazioni ibride di questi [14]. Tuttavia, l'approccio che descrive il profilo stradale con un rumore gaussiano bianco filtrato con un filtro passa–basso del prim'ordine rimane il modello matematico più diffuso.

I primi approcci consolidati, per la stima delle condizioni del manto stradale, coinvolgono l'adozione di costose e sofisticate apparecchiature hardware. Ad esempio, profilometri a guida laser [15], o sofisticati accelerometri con sistemi di acquisizione ed elaborazione dati [16]. Il costo di queste apparecchiature, per ciascun veicolo (senza contare la calibrazione e l'installazione) può essere molto elevato.

Nel tentativo di ridurre i costi del monitoraggio stradale, in letteratura molti studi hanno esplorato l'uso di accelerometri autonomi o accelerometri incorporati nei dispositivi mobili. I loro obiettivi sono rilevare dossi, buche, strade accidentate e altre anomalie della superficie stradale. La maggior parte di questi studi, si concentra sulla localizzazione e identificazione di singole anomalie (o classi di anomalie).

In [17] viene utilizzato un accelerometro indipendente, al fine di determinare—mediante una simulazione—le condizioni delle asperità del manto stradale. Le simulazioni eseguite conducono alla conclusione che le asperità del manto stradale possono essere stimate attraverso i dati prodotti dal sensore stesso.

In [18] (a cui si fa riferimento col termine $P^2$, Pothole Patrol) viene sviluppato un sistema di filtri a cascata. Ciascun filtro analizza una differente classe di eventi. Nello studio viene utilizzato un accelerometro indipendente, ad un'elevata frequenza di campionamento (380 Hz) per analizzare le diverse anomalie del manto stradale.

In [19] (noto come "Nericell"[1]) vengono utilizzati diversi componenti del dispositivo mobile, come accelerometro, microfono, radio GSM e sistema GPS per analizzare la strada e le condizioni del traffico. Attraverso l'analisi di questi dati, forniti dai sensori, è possibile rilevare buche, dossi, frenate e il rumore dei clacson. In modo analogo a $P^2$, in Nericell viene utilizzato un accelerometro ad alta frequenza (310 Hz, una frequenza superiore a quella della maggior parte degli accelerometri incorporati negli attuali dispositivi mobili) per rilevare buche e dossi. Inoltre, vi è una suddivisione tra alte e basse velocità, con una discriminante pari a $\geq 25$ km/h. Tutte queste informazioni vengono infine assemblate per valutare le condizioni del manto stradale e del traffico.

[1] Nericell è un gioco di parole che deriva da 'Nerisal': congestione, in lingua tamil. Lingua parlata in India, Sri Lanka, Singapore e altri territori che si affacciano sull'Oceano Indiano.

Dispositivi mobili dotati di accelerometri sono utilizzati in [20] e in [21] per rilevare la posizione delle buche. Il loro approccio include alcuni semplici algoritmi per la rilevazione di eventi nei dati forniti dall'accelerometro. Il lavoro [20] ricalca molto $P^2$, creandone una versione ridotta per dispositivi mobili. Utilizza una frequenza di campionamento di 38 Hz (un decimo di quella di $P^2$).

Dispositivi mobili dotati di accelerometri sono utilizzati anche in [22] e [23]. In questo caso, i dati dell'accelerometro sono analizzati nel dominio della frequenza al fine di estrarre caratteristiche corrispondenti ai dossi stradali.

In [24] gli autori propongono un sistema di monitoraggio del manto stradale cooperativo, denominato CRSM[2]. CRSM è in grado di rilevare le buche e valutare i livelli di rugosità stradali utilizzando i moduli hardware montati sui veicoli. Il programma è stato testato su 100 taxi nell'area urbana di Shenzhen. I risultati degli esperimenti mostrano soltanto un 10 % di falsi positivi. L'analisi dell'indice di rugosità viene eseguita con un'algoritmo, denominato Gaussian Mixture Model. Infine viene calcolato l'IRI in modo statistico.

Sempre con l'ausilio di dispositivi mobili dotati di accelerometri, in [25] gli autori classificano diverse classi di anomalie stradali, quali dossi artificiali per il rallentamento, tombini, linee di decelerazione e buche. In questo caso, come fatto in Nericell [19] viene operata una distinzione tra basse e alte velocità con discriminante sempre pari a 25 km/h. La componente verticale dell'accelerazione viene individuata calcolando la componente media dell'accelerazione di gravità. Nell'articolo si modellano le oscillazioni causate dalle anomalie con un sistema oscillatorio senza smorzamenti. Gli eventi sono determinati confrontando la componente verticale dell'accelerazione, con la deviazione standard normale di questa componente. Infine, l'indice dell'evento viene ottenuto dividendo la deviazione standard durante il bump, con la deviazione standard normale.

Nello studio [26] viene introdotta una metodologia per rilevare le anomalie stradali analizzando i comportamenti del conducente. Si analizzano le attività del comportamento del conducente al fine d'individuare le sterzate. Il giroscopio e accelerometro del dispositivo mobile forniscono due dei suddetti parametri. Il giroscopio misura tutte le variazioni angolari dovute ai cambiamenti di direzione della vettura, mentre l'accelerometro misura l'accelerazione centripeta.

La maggior parte di questi studi si concentrano principalmente sulla identificazione e la localizzazione singole anomalie (o classi di anomalie). Diversamente, i progetti di ricerca più recenti [1, 3, 5, 7, 27–29] si sono concentrati sull'uso dei dati forniti dagli accelerometri incorporati nei dispositivi mobili, al fine di stimare un indice di rugosità che caratterizzi lo stato manto stradale.

[2]CRSM, dal titolo della pubblicazione: *Crowdsourcing based Road Surface Monitoring.*

In [27–29] i dati provenienti dall'accelerometro sono comparati con la misurazione dell'IRI (International Roughness Index) e viene eseguita nello stesso tratto di strada in analisi. I campioni provenienti dall'accelerometro sono prima filtrati con un filtro passa–alto, con lo scopo di eliminare o—quantomeno—ridurre l'effetto dell'accelerazione di gravità, dell'accelerazione del veicolo e l'accelerazione centrifuga presso le curve. Poi i campioni filtrati vengono analizzati nel dominio della frequenza in diverse bande con l'indice IRI che viene calcolato con l'algoritmo Vehicular Intelligent Monitoring System (VIMS) sulla stessa strada. Si dimostra che l'ampiezza dello spettro è approssimativamente proporzionale all'IRI della strada. La pubblicazione considera anche l'effetto della velocità, tramite un modello multilineare.

All'approccio scelto dal nostro gruppo di ricerca, simile in alcuni aspetti ad altri già presentati in letteratura, ma differente in molti punti strategici, è stato assegnato il nome "SmartRoadSense" [3]. In [3] i campioni dell'accelerometro sono suddivisi in finestre di 1 s. Un'analisi della codifica predittiva lineare (LPC, Linear Predictive Coding) viene eseguita sui segnali accelerometrici per ogni singola finestra, dai quali si estrae la potenza del residuo di predizione, utilizzata per stimare un indice di rugosità. L'analisi LPC rimuove l'influenza sul sensore dell'accelerazione di gravità, delle accelerazioni del veicolo, delle accelerazioni centrifughe dovute alle curve, il rollio, beccheggio e imbardata, nonché le accelerazioni dovute all'andamento stradale e le vibrazioni dovute al motore. Infatti, l'analisi LPC è in grado di rimuovere tutte le componenti predicibili del segnale, quelle che variano lentamente o periodiche. Il residuo di predizione dell'analisi LPC conserva la parte impredicibile delle accelerazioni, che può essere per lo più correlato all'effetto della superficie stradale sugli pneumatici e sulle sospensioni. Le coordinate GPS, la velocità del veicolo e l'indice di rugosità vengono trasmesse periodicamente a un centro di elaborazione dati remoto. Qui vengono aggregate tutte le informazioni provenienti dai diversi dispositivi mobili compensando le loro differenti sensibilità [4]. Infine, i dati risultanti vengono memorizzati in un sistema informativo territoriale, in cui ciascun dato è calcolato come la media ponderata nel tempo e nello spazio, dei dati mappati sulla stessa strada. Dopo un periodo di studio e prove iniziali, dal 21 febbraio 2015, l'applicazione SmartRoadSense è stata ufficialmente rilasciata ed è diventata pienamente operativa. In pochi mesi—fino al 31 dicembre 2015—SmartRoadSense ha raccolto più di 24,664.4 km, corrispondenti al 5.06 % dell'intera rete viaria italiana [1, 30]. Il database di SmartRoadSense contiene gli indici di rugosità di tutte queste strade, pari a 6,001,617 campionamenti di singoli indici di rugosità estratti da 1,800,485,100 valori di accelerazione.

Esistono anche lavori successivi a SmartRoadSense, sviluppati da altri enti di ricerca. Ad esempio, in [31] viene utilizzato un accelerometro indipendente collegato a un computer portatile tramite un sistema di acquisizione dati (i quali vengono poi analizzati in

laboratorio). Il lavoro si basa sullo studio della potenza dell'accelerazione verticale. In diversi aspetti ricalca la metodologia proposta in [3].

# Capitolo 2

# Informazioni di base e teoria

In questo capitolo sono illustrati i diversi fondamenti teorici a supporto del lavoro di ricerca.

In primo luogo, vengono riportate tutte quelle nozioni di base che non possono essere ritenute note al lettore. In particolare—tra queste—vi è l'analisi dei dati tramite modelli a codifica predittiva lineare (LPC, Linear Predictive Coding). Particolare attenzione viene dedicata ai modelli autoregressivi (AR, Autoregressive), di cui si farà massiccio uso, e all'approccio scelto per la loro soluzione.

A seguire, viene presentata la teoria adottata per modellare il manto stradale e la densità spettrale di potenza (PSD, Power Spectral Density) del profilo altimetrico stradale. Una volta nota questa densità spettrale di potenza, è possibile procedere con il calcolo della densità spettrale di potenza dell'accelerazione verticale relativa a un punto materiale che segue da vicino il profilo altimetrico stradale con velocità costante.

Ricordando che il nostro sensore è incorporato in un dispositivo mobile ancorato all'interno dell'abitacolo del veicolo, si procede con lo studio dettagliato delle sue sospensioni, deputate allo smorzamento delle accelerazioni verticali. Le sospensioni, infatti, sono un insieme di componenti meccanici viscoelastici che interagiscono con l'accelerazione verticale. La loro finalità è assorbire—quanto più possibile—le imperfezioni del terreno e garantire, come conseguenza, stabilità, tenuta di strada e comfort di guida.

Infatti, qualora l'intera massa del veicolo fosse rigidamente connessa agli pneumatici, tutti i movimenti di quest'ultimi verrebbero trasmessi direttamente al veicolo innescando pericolosi sobbalzi con conseguente perdita di controllo del mezzo da parte del conducente.

A questo punto, si procede con la stima della potenza rilevata dall'accelerometro e—di conseguenza—dell'indice di rugosità del manto stradale. Questo avviene sfruttando la

conoscenza del modello matematico delle sospensioni e del profilo altimetrico stradale. Eseguita questa operazione, è possibile determinare un indice di rugosità del profilo stradale stesso.

Infine, tale indice di rugosità viene corretto con una funzione gamma i cui parametri vengono estratti dalla grande massa di dati presente nel database (basato su oltre 1.8 miliardi di singole accelerazioni campionate), avente come scopo quello di eliminare la dipendenza dei valori ottenuti dell'indice di rugosità, dalla velocità del veicolo e dalla tipologia dell'asfalto percorso.

## 2.1 Analisi dei dati tramite modelli LPC

I modelli autoregressivi $AR(p)$, dove $p$ è l'ordine del modello, descrivono serie di dati statistici ricorrendo a una combinazione lineare dei loro $p$ campioni precedenti. Il modello autoregressivo specifica che la variabile in uscita all'istante $n$esimo, dipende dai suoi valori precedenti e da una termine aleatorio (impredicibile) $e$.

Nel caso generale, il modello autoregressivo $AR(p)$ si presenta come un'equazione alle differenze di ordine $p$, come descritto nella (2.1):

$$\begin{aligned} x(n) + \lambda_1 x(n-1) + \lambda_2 x(n-2) + \cdots + \lambda_p x(n-p) &= e(n) \\ x(n) + \sum_{i=1}^{p} \lambda_i x(n-i) &= e(n) \end{aligned} \quad . \tag{2.1}$$

Nella (2.1), $x(n)$ è il campione attuale del sistema stimato tramite il modello $AR(p)$, $\lambda_i$ sono i $p$ parametri del modello i quali vengono applicati ai suoi $p$ campioni precedenti, o—analogamente—definiscono la memoria del modello (in questo caso di ordine $p$). Infine $e(n)$ è il residuo di predizione del campione $x(n)$ che si desidera minimizzare, la componente non predicibile del sistema.

Nel caso di modello autoregressivo ideale, $e(n)$ restituisce un rumore gaussiano bianco (WGN, White Gaussian Noise). In caso contrario, $e(n)$ contiene informazioni sul sistema in esame. Questo significa che il modello ottenuto può essere ulteriormente perfezionato, tipicamente aumentando il suo ordine $p$, e—di conseguenza—rimuovendo così la componente predicibile in $e(n)$. Naturalmente, questo processo ha un andamento asintotico. Di conseguenza, questo lavoro acquista un significato nella misura in cui ampliare la memoria del modello autoregressivo produce benefici significativi. Quando all'aumentare dell'ordine del modello, non si ottengono più reali miglioramenti, in termini di predizione, il processo viene arrestato.

In ogni caso, il residuo di predizione $e(n)$ sarà sempre presente indipendentemente dall'ordine del modello autoregressivo. Infatti, questa tipologia di analisi, fornisce una stima—non una previsione esatta—del campione attuale del sistema.

Ai fini dello studio in esame, l'interesse è determinare la potenza del residuo di predizione. Questo parametro, infatti, è proporzionale alla componente impredicibile del segnale in esame, nella fattispecie, del manto stradale. L'idea di fondo è che più il manto stradale attraversato è omogeneo, minore sarà il residuo di predizione. Nel momento in cui il veicolo incontra un'anomalia nell'asfalto, questa si rifletterà in un evento non predicibile dal sistema, quindi in un aumento della potenza del residuo di predizione.

### 2.1.1 Modelli autoregressivi del prim'ordine

Il caso più elementare è quello relativo a un modello autoregressivo del prim'ordine, $AR(1)$. In questo caso la (2.1) si riduce ad una equazione alle differenze del prim'ordine, come riportato nella (2.2):

$$x(n) + \lambda x(n-1) = e(n). \tag{2.2}$$

In questo caso, il calcolo della potenza del residuo di predizione $P_E$ è immediato ed è pari alla (2.3):

$$P_E = \frac{1}{N} \sum_{n=1}^{N-1} e(n)^2, \tag{2.3}$$

la quale—al fine di identificare il sistema ottimo—deve essere minimizzata per tutti gli $N$ campioni presenti nella serie statistica, fatta eccezione per il termine di partenza. Sul primo termine della serie non è possibile fare predizioni: questo deve essere considerato come un termine noto, in quanto già trascorso. Diversamente non sarebbe possibile applicare la (2.2).

Pertanto, per stimare il filtro predittivo, si procede determinando il minimo della potenza del residuo di predizione. Questo valore è dato dal punto in cui la derivata prima della (2.3) si annulla rispetto a $\lambda$, come mostrato nella (2.4):

$$\frac{\partial P_E}{\partial \lambda} = \frac{\partial}{\partial \lambda} \frac{1}{N} \sum_{n=1}^{N-1} (x(n) + \lambda x(n-1))^2 = 0. \tag{2.4}$$

Ricercando il minimo della (2.4) si ottiene:

$$\frac{1}{N}\sum_{n=1}^{N-1} x(n)x(n-1) + \lambda\frac{1}{N}\sum_{i=1}^{N-1} x(n-1)x(n-1) = 0, \tag{2.5}$$

dove, ricordando che $R(k) = \dfrac{1}{N}\displaystyle\sum_{n=1}^{N-1} x(n)x(n-k)$ è la funzione di autocorrelazione (ACF, Autocorelation Function), la (2.5) può essere riscritta in forma compatta come segue:

$$R(1) + \lambda R(0) = 0. \tag{2.6}$$

Pertanto, dalla (2.6), si evince che il coefficiente ottimo di autoregressione $\lambda$ (per un modello autoregressivo del prim'ordine), è pari a:

$$\lambda = -\frac{R(1)}{R(0)}. \tag{2.7}$$

### 2.1.2 Modelli Autoregressivi oltre il prim'ordine

Nel caso generale della (2.1), con un modello $AR(p)$, la (2.6) diventa un sistema di $p$ equazioni in $p$ incognite del tipo:

$$\sum_{k=1}^{p} R(k-m)\lambda_k = -R(m), \tag{2.8}$$

con $m \in [1,\, p]$.

La (2.8) può essere riscritta in forma matriciale, ottenendo l'espressione riportata nella (2.9).

$$\begin{pmatrix} R(0) & R(1) & \cdots & R(p-1) \\ R(1) & R(0) & \cdots & R(p-2) \\ \vdots & \vdots & \ddots & \vdots \\ R(p-1) & R(p-2) & \cdots & R(0) \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_p \end{pmatrix} = - \begin{pmatrix} R(1) \\ R(2) \\ \vdots \\ R(p) \end{pmatrix}. \tag{2.9}$$

La (2.9) è la cosiddetta equazione di Yule–Walker [32, 33]. È importante sottolineare che al primo membro la matrice di correlazione assume la forma di una matrice di Toeplitz [34]. Questo fatto è molto importante (come si vedrà nel Paragrafo 2.1.3) poiché consente l'adozione di algoritmi veloci per la soluzione del sistema [35, 36] senza dover calcolare la matrice inversa.

Infatti, volendo adottare una soluzione diretta, i parametri autoregressivi $\lambda_{1,2,\ldots,p}$ sarebbero dati da:

$$\begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_p \end{pmatrix} = - \begin{pmatrix} R(0) & R(1) & \cdots & R(p-1) \\ R(1) & R(0) & \cdots & R(p-2) \\ \vdots & \vdots & \ddots & \vdots \\ R(p-1) & R(p-2) & \cdots & R(0) \end{pmatrix}^{-1} \begin{pmatrix} R(1) \\ R(2) \\ \vdots \\ R(p) \end{pmatrix}. \tag{2.10}$$

La particolare forma della matrice di correlazione, consente di calcolare il filtro predittivo $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_p)^T$ attraverso l'algoritmo ricorsivo di Levinson–Durbin [35, 36], presentato nel Paragrafo 2.1.3, dove $R(0)$, $R(1)$, $\ldots$, $R(p-1)$ sono i $p$ coefficienti di autocorrelazione che minimizzano la potenza del residuo di predizione $P_E$.

### 2.1.3 Matrice di Toeplitz

Si definisce *Matrice di Toeplitz* una matrice $\mathbf{M}$ avente una struttura come riportato nella (2.11):

$$\mathbf{M} = \begin{pmatrix} t_0 & t_{-1} & t_{-2} & \ldots & t_{-n+1} \\ t_1 & t_0 & t_{-1} & \ldots & t_{-n+2} \\ t_2 & t_1 & t_0 & \ldots & t_{-n+3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t_{n-1} & t_{n-2} & t_{n-3} & \ldots & t_0 \end{pmatrix}. \tag{2.11}$$

Da queste tipologie di matrici discende il cosiddetto "problema di Toeplitz", un'equazione matriciale posta nella seguente forma:

$$\vec{y} = \mathbf{M}\ \vec{x} \tag{2.12}$$

dove $\mathbf{M}$ è una matrice di Toeplitz con elementi non nulli nella diagonale principale, $\vec{y}$ un vettore noto e $\vec{x}$ un vettore incognito.

La particolare struttura di questa matrice consente di risolvere il problema senza ricorrere all'inversione della matrice $\mathbf{M}$, grazie a un algoritmo ricorsivo, proposto prima da Norman Levinson [35], ed implementato successivamente da James Durbin [36], in $\Theta(n^2)$ passi.

L'algoritmo ricorsivo di Levinson–Durbin [35, 36] è sintetizzato in Figura 2.1 (dove viene utilizzata la notazione MATLAB®). I parametri d'ingresso sono i coefficienti di autocorrelazione $R(0)$, $R(1)$, ..., $R(p-1)$.

```
k = R(2) / R(1);                        % Stima il primo coefficiente λ
λ = k;
E = (1 - k^2) * R(1);
for i = 2:p
    k = (R(i + 1) - λ * R(2:i)) / E; % Coefficienti di riflessione
    λ = [k, λ - k * λ(i-1:-1:1)];    % Stima i seguenti coefficienti
    E = (1 - k^2) * E;               % Aggiorna l'errore quadratico medio
end

λ = [1, - λ(N:-1:1)];                   % Restituisce il filtro predittivo
```

FIGURA 2.1: Pseudocodice per la ricorsione Levinson–Durbin.

L'algoritmo si basa sul calcolo dei coefficienti autoregressivi di matrici di ordine crescente. Si divide in due fasi: una prima d'inizializzazione per il calcolo del parametro $\lambda_1$ o—che è lo stesso—per il caso elementare della matrice $1 \times 1$. Successivamente, si procede con il calcolo iterativo dei parametri per matrici di ordine via, via crescente $2 \times 2$, $3 \times 3$,..., $p \times p$.

L'algoritmo mostrato in Figura 2.1, partendo dal vettore di autocorrelazione, restituisce il filtro predittivo per il modello $AR(p)$, l'errore di predizione e i coefficienti di riflessione $k$[1]. $E$, invece, è l'errore quadratico medio (MSE, Mean Square Error) sull'intervallo considerato.

## 2.2 Potenza dell'accelerazione verticale

Come mostrato nel Paragrafo 2.6, l'accelerazione percepita dal dispositivo mobile, posto all'interno di un autoveicolo, non è quella rilevata sulla superficie del manto stradale. Tale accelerazione viene filtrata dagli pneumatici, dalle sospensioni, dall'accoppiamento meccanico fino ad arrivare al dispositivo mobile.

Tuttavia, in ogni autoveicolo, il principale deputato allo smorzamento dell'accelerazione verticale è il sistema di sospensioni studiato in dettaglio nel Paragrafo 2.6. La conoscenza dell'accelerazione verticale presente sul manto stradale, unita alla risposta in frequenza del sistema di sospensioni, consente di stimare la potenza dell'accelerazione verticale

[1] I coefficienti di riflessione $k$, sono così definiti per analogia con le linee di trasmissione dove, in corrispondenza dell'interfaccia tra due mezzi di propagazione impedenza caratteristica differente, una parte dell'energia viene trasmessa, mentre la restante viene riflessa.

$P(v)$ percepita in un veicolo che procede in moto rettilineo uniforme con velocità costante pari a $v$.

Nei paragrafi seguenti, partendo dal suolo, sino ad arrivare al dispositivo mobile, viene descritto passo dopo passo il modello matematico dell'accelerazione verticale percepita da un dispositivo mobile rigidamente ancorato all'interno di un'automobile. L'obiettivo è mettere in relazione la potenza rilevata dell'accelerometro incorporato nel dispositivo mobile all'interno dell'abitacolo, con l'accelerazione esercitata sui pneumatici che poggiano sul manto stradale.

## 2.3 Modello teorico del manto stradale

I primi approcci per la modellistica e identificazione del profilo stradale considerano funzioni impulsive, onde triangolari o segnali sinusoidali [12]. In studi più accurati [8, 37], il manto stradale viene modellato come somme di segnali sinusoidali con ampiezze e fasi generate casualmente.

Sempre in [8], viene mostrato come la densità spettrale di potenza (PSD, Power Spectral Density) tipica di una superficie stradale, abbia una caratteristica passa–basso, la quale decresce con l'aumentare della frequenza spaziale. Questo modello è uno dei più diffusi in letteratura, per due semplici motivazioni:

1. descrive bene un manto stradale omogeneo, percorso a velocità costante;
2. e semplice da realizzare ed implementare.

Il modello descrive il manto stradale tramite un rumore gaussiano bianco filtrato attraverso un filtro passa–basso del prim'ordine. Nel dominio della frequenza il rumore bianco è una costante determinata dalla potenza del rumore stesso. Il filtro passa–basso del prim'ordine introduce un polo. Dalla conoscenza della potenza spettrale del rumore gaussiano bianco e del polo del filtro passa–basso, si determinano le proprietà statistiche del manto stradale.

Quindi, il modello gaussiano di [8] rimane uno dei modelli per l'identificazione del manto stradale più popolare per la sua capacità di rappresentare le caratteristiche superficiali dei manti stradali "non danneggiati" [14]. Pertanto, in questo studio, si considera un profilo altimetrico stradale $w(x)$ modellabile tramite un rumore gaussiano bianco (WGN, White Gaussian Noise) e successivamente filtrato con un fltro passa–basso (LPF, Low–Pass Filter) del prim'ordine.

Il rumore gaussiano bianco ha funzione di autocorrelazione (ACF, Autocorrelation Function) pari a $\rho_{ww} = q\delta(\Lambda)$, e la sua PSD è pari a:

$$S_{ww}(j\Lambda) = \int_{-\infty}^{+\infty} q\delta(\Lambda)e^{-j\Lambda}\mathrm{d}\Lambda = q, \tag{2.13}$$

dove $q$ è l'ampiezza della PSD, $\delta(\Lambda)$ è la funzione delta di Dirac e $\Lambda$ la pulsazione spaziale, misurata in radianti per metri.

Il filtro passa–basso del prim'ordine, nella sua formulazione generica, ha risposta in frequenza angolare pari a:

$$H(j\Lambda) = \frac{1}{p + j\Lambda}, \tag{2.14}$$

dove $p$ è il polo del filtro passa–basso.

Da quanto precede, segue che la densità spettrale di potenza spaziale, dell'elevazione del profilo stradale $S_{rr}(j\Lambda)$, è data dalla funzione di autocorrelazione del rumore gaussiano bianco seguita dal modulo al quadrato del filtro passa–basso del prim'ordine, ovvero:

$$S_{rr}(j\Lambda) = S_{ww}(j\Lambda)\left|H(j\Lambda)\right|^2 = q\left|\frac{1}{p + j\Lambda}\right|^2. \tag{2.15}$$

Dal modello descritto nella (2.15), si evince che le proprietà statistiche del profilo altimetrico stradale sono completamente caratterizzate una volta noti i parametri $q$ e $p$.

## 2.4 Densità spettrale di potenza del profilo stradale

In questo paragrafo si ipotizza un andamento del veicolo rettilineo uniforme, con velocità costante pari a $v$. Pertanto, si può scrivere $x(t) = v \cdot t$. Tale ipotesi, oltre a semplificare i calcoli, è plausibile poiché—nell'analisi che seguirà—si prendono in esame intervalli temporali dell'ordine del secondo.

In Figura 2.2, $x(t)$ rappresenta la distanza percorsa dal veicolo nella direzione di marcia, supposta unidirezionale, mentre $r\left[x(t)\right]$ è l'elevazione del profilo stradale, istante per istante.

Applicando, al rumore gaussiano bianco $W(j\Lambda)$, il filtro passa–basso del prim'ordine $H(\Lambda)$, come definito nella (2.14), si ottiene la rugosità del manto stradale:

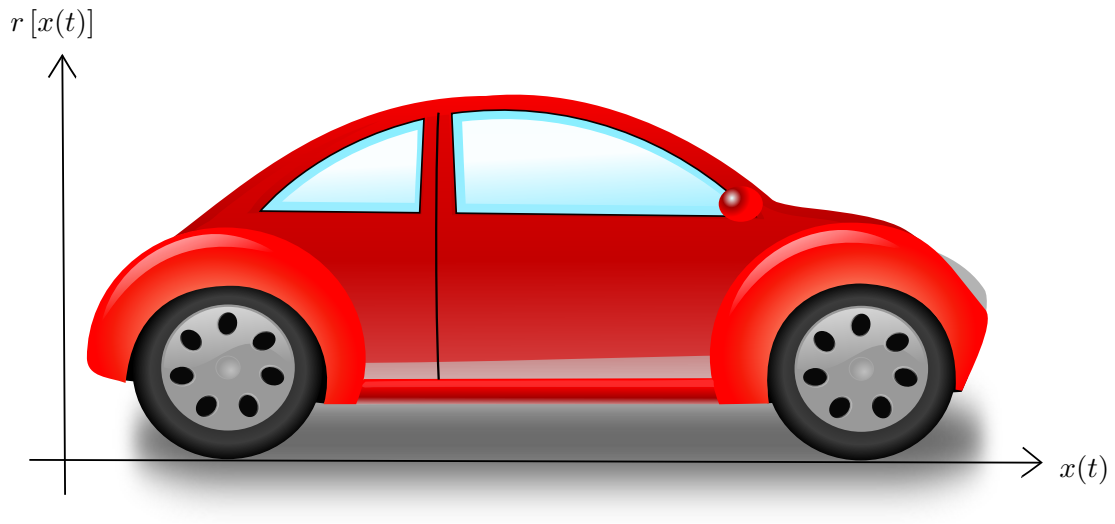



FIGURA 2.2: Sistema di riferimento.

$$r(j\Lambda) = \frac{1}{p + j\Lambda} W(j\Lambda). \tag{2.16}$$

Le equazioni che seguono, sono funzionali allo studio della rugosità del manto stradale nel dominio della pulsazione.

Si ha:

$$pr(j\Lambda) + j\Lambda r(j\Lambda) = W(j\Lambda). \tag{2.17}$$

Essendo $\frac{\mathrm{d}^n r(x)}{\mathrm{d}x^n} = (j\Lambda)^n r(j\Lambda)$, calcolando l'antitrasformata di Fourier si ottiene:

$$pr(x) + \frac{dr(x)}{dx} = W(x), \tag{2.18}$$

$$pr(x) + \frac{\mathrm{d}r(x)}{\mathrm{d}t}\frac{\mathrm{d}t}{\mathrm{d}x} = W(x), \tag{2.19}$$

$$pr(x) + \frac{\mathrm{d}r(x)}{\mathrm{d}t}\frac{1}{v} = W(x). \tag{2.20}$$

Come sopra, eseguendo la trasformata di Fourier—questa volta nel dominio del tempo—si ottiene:

$$pr(j\Omega) + \frac{j\Omega}{v} r(j\Omega) = W(j\Omega), \tag{2.21}$$

da cui si evince

$$r(j\Omega) = \frac{1}{p + \frac{j\Omega}{v}} W(j\Omega). \tag{2.22}$$

Dove $\Omega$ è la frequenza angolare, espressa in radianti al secondo.

Da quanto precede, è possibile riscrivere la densità spettrale di potenza, ora nel dominio del tempo. Il risultato è pari a:

$$S_{rr}(j\Omega) = S_{ww}(j\Omega)|H(j\Omega)|^2 = q\left|\frac{1}{p + \frac{j\Omega}{v}}\right|^2 = \tilde{S}_{rr}\left(\frac{j\Lambda}{v}\right) \tag{2.23}$$

Questo risultato mostra che la densità spettrale di potenza spaziale $\tilde{S}_{rr}(j\Lambda)$ dipende dalla densità spettrale di potenza temporale $S_{rr}(j\Omega)$ mediante la velocità del veicolo $v$ [12].

## 2.5 Densità spettrale di potenza dell'accelerazione verticale

Il calcolo della densità spettrale di potenza dell'accelerazione verticale è analogo a quello dell'elevazione del profilo stradale. Grazie ai risultati ottenuti nel Paragrafo 2.4, ora è possibile calcolare l'accelerazione percepita da un punto materiale che segue da vicino il profilo stradale. Il Paragrafo 2.6 integra e completa questo studio e quello del Paragrafo 2.4, al fine di poter calcolare, nel Paragrafo 2.7, la potenza dell'accelerazione verticale percepita dall'accelerometro all'interno del veicolo.

Il punto di partenza è il medesimo. La rugosità del manto stradale viene descritta dalla seguente espressione, ottenuta dalla (2.18):

$$pr(t) + \frac{\mathrm{d}r(t)}{\mathrm{d}t}\frac{1}{v} = W(t). \tag{2.24}$$

Derivando tutti i termini dell'equazione due volte rispetto al tempo, si ottiene l'accelerazione verticale $a_y(t)$:

$$p\frac{\mathrm{d}^2 r(t)}{\mathrm{d}t^2} + \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\mathrm{d}^2 r(t)}{\mathrm{d}t^2}\frac{1}{v}\right) = \frac{\mathrm{d}^2 W(t)}{\mathrm{d}t^2}, \tag{2.25}$$

$$p \cdot a_y(t) + \frac{\mathrm{d}}{\mathrm{d}t}\left(a_y(t)\frac{1}{v}\right) = \frac{\mathrm{d}^2 W(t)}{\mathrm{d}t^2}. \tag{2.26}$$

Calcolando la trasformata di Fourier della (2.26), nel dominio temporale, si ottiene la seguente espressione:

$$p \cdot A_y(j\Omega) + A_y(j\Omega)\frac{j\Omega}{v} = (j\Omega)^2 W(j\Omega), \tag{2.27}$$

dalla quale deriva

$$A_y(j\Omega) = \frac{(j\Omega)^2}{p + \frac{j\Omega}{v}} W(j\Omega). \tag{2.28}$$

La (2.28) evidenzia come l'accelerazione verticale—a differenza dell'elevazione altimetrica stradale—presenti una risposta in frequenza angolare di un filtro passa–alto.

Pertanto, l'accelerazione verticale di un punto materiale che segue da vicino il profilo altimetrico stradale, il quale si muove con velocità costante $v$, presenta una densità spettrale di potenza data da:

$$S_{A_y A_y}(j\Omega) = S_{ww}(j\Omega)\left|\frac{(j\Omega)^2}{p + \frac{j\Omega}{v}}\right|^2 = q\left|v\frac{(j\Omega)^2}{pv + j\Omega}\right|^2. \tag{2.29}$$

Pertanto, le proprietà statistiche del manto stradale, caratterizzate dai parametri $q$ e $p$, come mostrato nella (2.15), possono essere estratte anche attraverso l'analisi della densità spettrale di potenza fornita dallo studio dell'accelerazione verticale (2.29).

## 2.6 Introduzione al modello "Quarter–car"

Al suo livello più elementare, ogni veicolo presenta un'isolamento dal terreno fornitogli dal sistema di sospensioni connesso a ogni pneumatico. Le sospensioni hanno lo scopo di ridurre quanto più possibile l'accelerazione verticale della massa sospesa (*sprung mass*) quando il veicolo attraversa una strada sconnessa.

Le sospensioni sono uno dei componenti meno visibili dell'autoveicolo, ma determinanti per il comfort di guida. In un veicolo privato delle sospensioni, viaggiare sarebbe profondamente differente: è grazie a questo sistema d'isolamento che è possibile muoversi senza percepire un sussulto continuo da parte del veicolo stesso.

Il modello dinamico essenziale delle sospensioni è mostrato in Figura 2.3. Lo pneumatico (*tire*) può essere rappresentato con una molla (*spring*), anche se in alcuni modelli più complessi—talvolta—viene aggiunto, in parallelo, un'ammortizzatore (*dumper*) per includere la sua natura viscoelastica [9, 10]. Di seguito, la sospensione è rappresentata con una molla rigida (*suspension stiffness*) avente in parallelo un'ammortizzatore (*suspension dumper*).

Nella Figura 2.3, $M$ rappresenta la massa sospesa (*sprung mass*, al di sopra del sistema di sospensione) dove $Z$ rappresenta la sua quota. $K_s$ è la rigidità della sospensione, mentre $C_s$ rende conto della costante di smorzamento della sospensione che connette $M$ ad $m$, dove $m$ è la massa non sospesa (*unsprung mass*, posta sopra lo pneumatico e al di sotto della sospensione), mentre $Z_u$ è la sua quota. $K_t$ è la rigidità della molla che modella lo pneumatico. Infine, $Z_r$ è il livello del manto stradale.

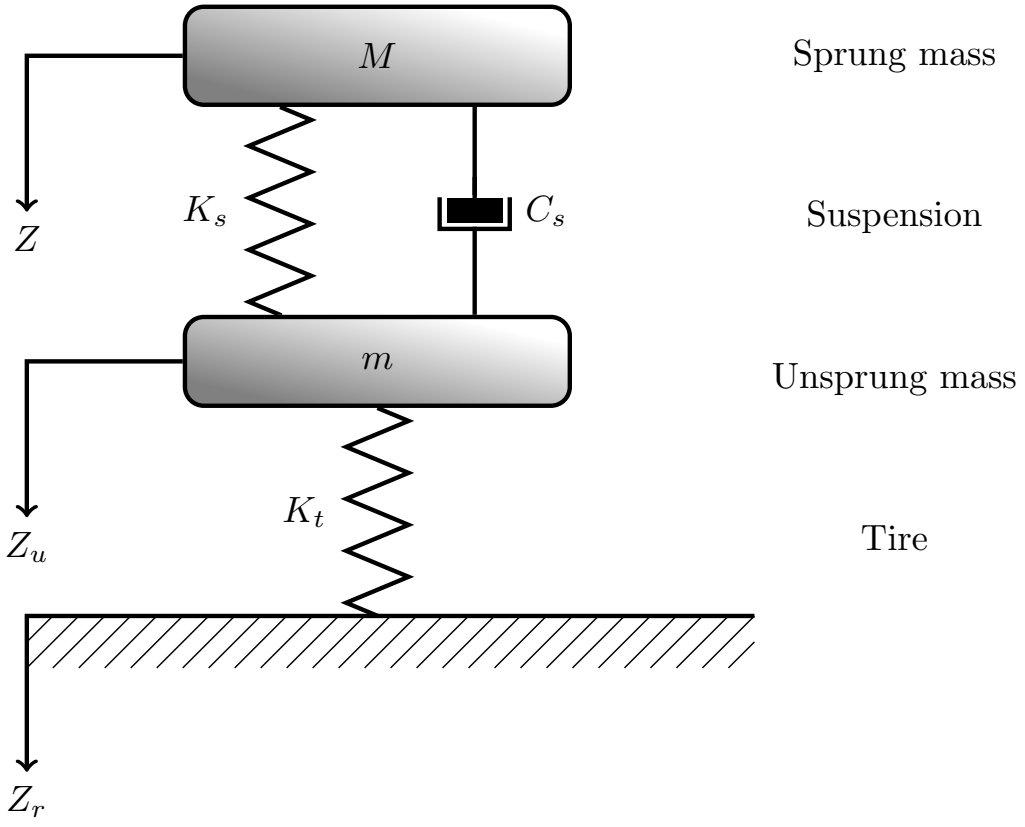



FIGURA 2.3: Modello matematico del Quarter–car.

Un modello così concepito, relativo a una singola sospensione, è noto in letteratura come "Quarter–car model" [38, 39], poiché rappresenta un quarto delle sospensioni effettivamente presenti nel veicolo.

Il sistema di sospensioni presenta un modello matematico che può essere risolto utilizzando la seconda legge di Newton, poiché è riconducibile a un'equazione differenziale ordinaria non omogenea del second'ordine (come mostrato nel paragrafo 2.6.2).

### 2.6.1 Risposta del sistema agli smorzamenti

Il modello dinamico della sospensione, mostrato in Figura 2.3, è deputato a svolgere un duplice effetto nell'assorbimento degli urti. Il suo primo compito viene svolto degli pneumatici stessi a contatto col suolo; il secondo—più significativo—è quello generato dalle sospensioni stesse.

La rigidità globale della sospensione può essere calcolata ponendo in serie la rigidità della sospensione con quella dello pneumatico. Tale parametro è noto come Ride Rate (RR), ed è dato da:

$$RR = \frac{K_s K_t}{K_s + K_t}. \quad \text{(N/m)} \tag{2.30}$$

La pulsazione naturale di oscillazione $\omega_n$, quella che si avrebbe senza alcuna azione ammortizzante, è data da:

$$\omega_n = \sqrt{\frac{R_R}{M}}, \quad \text{(rad/s)} \tag{2.31}$$

Dalla quale si ottiene la seguente frequenza naturale di oscillazione:

$$f_n = \frac{1}{2\pi}\sqrt{\frac{R_R}{W/g}}, \quad \text{(Hz)} \tag{2.32}$$

dove $W = M \cdot g$ costituisce il peso (*weight*) sostenuto dalla singola sospensione, mentre $g$ è l'accelerazione di gravità.

L'azione dell'ammortizzatore, quando presente—come nel caso delle sospensioni—ha pertanto lo scopo di ridurre la frequenza naturale di oscillazione. Quello che si ottiene è la cosiddetta frequenza naturale smorzata (*damped natural frequency*) $f_d$, determinata da:

$$f_d = f_n\sqrt{1 - \zeta_s^2}, \quad \text{(Hz)} \tag{2.33}$$

dove $\zeta_s$ è un parametro adimensionale detto rapporto di smorzamento (*damping ratio*). Il suo valore è pari a:

$$\zeta_s = \frac{C_s}{2\sqrt{K_s M}}. \tag{2.34}$$

Di seguito, vengono riportati alcuni valori tipici presenti in letteratura [8, Pag. 163, Fig. 5.25] dei parametri tipici del modello Quarter–Car di Figura 2.3. Un resoconto esaustivo, di tali parametri, è riportato nel paragrafo 2.6.4.

$$\begin{array}{ll} M = 240 \text{ kg} & \text{Sprung Mass} \\ m = 36 \text{ kg} & \text{Unsprung Mass} \\ K_t = 160,000 \text{ N/m} & \text{Tire Stiffness} \\ K_s = 16,000 \text{ N/m} & \text{Suspension Stiffness} \\ C_s = 980 \text{ Ns/m} & \text{Dumping constant} \end{array} \tag{2.35}$$

Per ottenere una sospensione con delle buone prestazioni, il rapporto di smorzamento della sospensione $\zeta_s$ deve essere superiore al 20 % ed inferiore al 40 %. Ovvero, compreso nell'intervallo $\zeta_s \in (0.2 \div 0.4)$. Con i parametri forniti nella (2.35) si ottiene $\zeta_s = 0.2501$, e—di conseguenza—una frequenza naturale smorzata pari a $f_d = 0.3831$ Hz.

### 2.6.2 Modello matematico delle sospensioni

Il sistema con sospensioni presenta un modello matematico che può essere scritto e risolto applicando la seconda legge di Newton a ogni massa del sistema in esame.

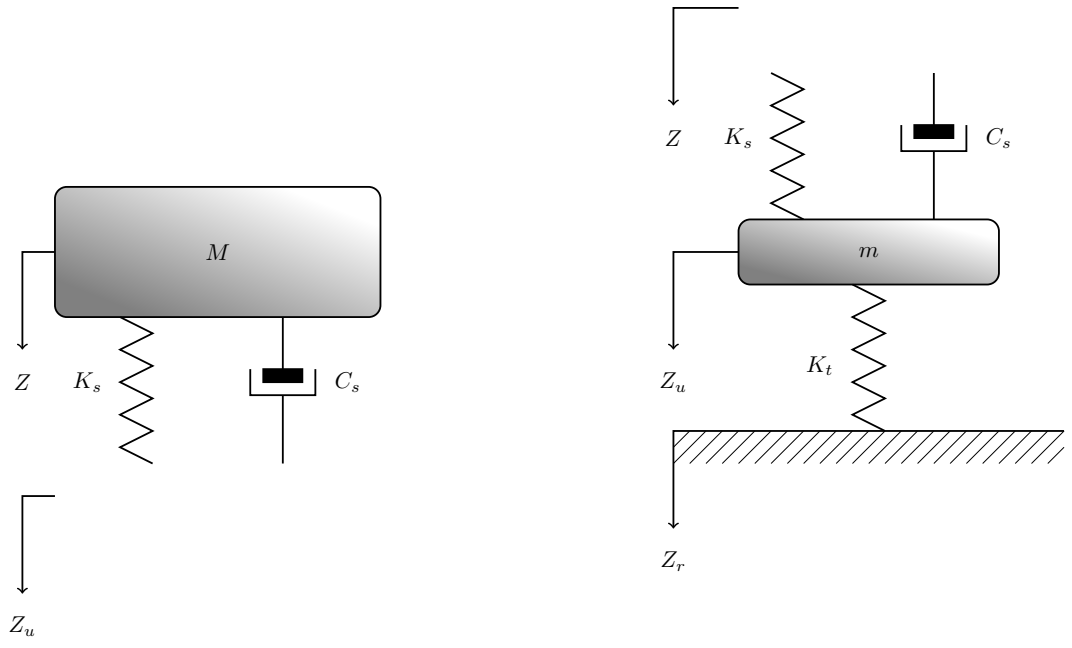



FIGURA 2.4: Analogo meccanico delle sospensioni scomposto.

Nella Figura 2.4 viene mostrato il medesimo modello matematico di Figura 2.3 ma scomposto, separando le forze che agiscono sulla massa sospesa $M$ e sulla massa non sospesa $m$.

La massa sospesa $M$ interagisce con la massa non sospesa $m$ mediante lo smorzatore, avente costante di smorzamento $C_s$, e la molla, la quale ha rigidità $K_s$.

Pertanto, l'equilibrio delle forze che si ottiene è pari a:

$$M\ddot{Z} = -C_s(\dot{Z} - \dot{Z_u}) - K_s(Z - Z_u), \tag{2.36}$$

dove $\dot{Z}$ e $\ddot{Z}$ sono, rispettivamente, la derivata prima e la derivata seconda della quota $Z$ rispetto al tempo $t$, nella notazione di Newton.

Similmente, la massa non sospesa $m$ al livello del pneumatico, interagisce sia con la massa sospesa $M$ posta al di sopra del sistema di sospensioni, mediante lo smorzatore, avente costante di smorzamento $C_s$ e con la molla avente rigidità $K_s$. Inoltre, interagisce con il suolo, attraverso la molla, che modella lo pneumatico, avente rigidità $K_t$.

Si ha quindi un equilibrio delle forze pari a:

$$m\ddot{Z_u} = -C_s(\dot{Z_u} - \dot{Z}) - K_s(Z_u - Z) - K_t(Z_u - Z_r). \tag{2.37}$$

Il modello matematico che descrive il comportamento dinamico del sistema quarter–car, è dato dalla (2.36) e dalla (2.37). Riscrivendo le due equazioni a sistema, al fine di evidenziare le forze esterne e le interazioni interne al modello in esame, si ottiene il seguente equilibrio:

$$\begin{cases} M\ddot{Z} + C_s(\dot{Z} - \dot{Z_u}) + K_s(Z - Z_u) = 0 \\ m\ddot{Z_u} - C_s(\dot{Z} - \dot{Z_u}) - K_s Z + (K_s + K_t)Z_u = K_t Z_r \end{cases}. \tag{2.38}$$

La (2.38), espressa in forma matriciale, diventa di più facile lettura:

$$\mathbf{m}\ddot{\mathbf{Z}} + \mathbf{C}\dot{\mathbf{Z}} + \mathbf{K}\mathbf{Z} = \mathbf{F}. \tag{2.39}$$

La (2.39), scritta in forma espansa, è pari a:

$$\begin{bmatrix} M \\ m \end{bmatrix} \begin{bmatrix} \ddot{Z} \\ \ddot{Z_u} \end{bmatrix} + \begin{bmatrix} C_s & -C_s \\ -C_s & C_s \end{bmatrix} \begin{bmatrix} \dot{Z} \\ \dot{Z_u} \end{bmatrix} + \begin{bmatrix} K_s & -K_s \\ -K_s & K_s + K_t \end{bmatrix} \begin{bmatrix} Z \\ Z_u \end{bmatrix} = \begin{bmatrix} 0 \\ K_t Z_r \end{bmatrix}, \tag{2.40}$$

dove vengono separate la forza risultante ($m\ddot{Z}$), dalla componente legata all'azione dello smorzatore ($C\dot{Z}$) e dalla forza elastica ($KZ$). Al secondo membro della (2.40) sono presenti le sollecitazioni esterne (non nulle), le quali fanno di questo un sistema di equazioni differenziali ordinarie del second'ordine non omogeneo.

### 2.6.3 Soluzione del modello matematico

Un metodo per ricavare la funzione di trasferimento tra accelerazione sulla superficie stradale ed accelerazione residua all'interno del veicolo $\ddot{Z}/\ddot{Z}_r$ è risolvere l'equazione (2.40) tramite la Trasformata di Laplace, trasformando il sistema di equazioni differenziali ordinarie del second'ordine in un sistema di equazioni di secondo grado.

Come primo passo si riscrive la (2.40) trasformandola secondo Laplace. Per agevolare il lavoro successivo, i termini vengono riordinati così da poter esprimere $Z$ in funzione di $Z_r$, ottenendo il seguente sistema di equazioni, dove—in entrambe le equazioni—è stata isolata $Z_u$, termine che si desidera elidere.

$$\begin{cases} s^2MZ + sC_sZ + K_sZ = sC_sZ_u + K_sZ_u \\ s^2mZ_u + sC_sZ_u + (K_s + K_t)Z_u = sC_sZ + K_sZ + K_tZ_r \end{cases} \tag{2.41}$$

Si scriverà, pertanto,

$$Z(s^2M + sC_s + K_s) = Z_u(sC_s + K_s) \tag{2.42a}$$

$$Z_u(s^2m + sC_s + K_s + K_t) = Z(sC_s + K_s) + K_tZ_r \tag{2.42b}$$

Isolando nella (2.42a) il termine $Z_u$, questo risulta pari a:

$$Z_u = Z\frac{s^2M + sC_s + K_s}{sC_s + K_s}. \tag{2.43}$$

Sostituendo la (2.43) nella (2.42b) si ottiene infine:

$$Z\frac{s^2M + sC_s + K_s}{sC_s + K_s}(s^2m + sC_s + K_s + K_t) = Z(sC_s + K_s) + K_tZ_r, \tag{2.44}$$

$$Z\frac{(s^2M + sC_s + K_s)(s^2m + sC_s + K_s + K_t) - (sC_s + K_s)^2}{sC_s + K_s} = K_tZ_r, \tag{2.45}$$

$$\frac{Z}{Z_r} = \frac{K_t(sC_s + K_s)}{(s^2M + sC_s + K_s)(s^2m + sC_s + K_s + K_t) - (sC_s + K_s)^2}. \tag{2.46}$$

La quale—con un po' di algebra—diventa:

$$\frac{Z}{Z_r} = \frac{sK_tC_s + K_tK_s}{s^4Mm + s^3C_s(M + m) + s^2(MK_t + mK_s + MK_s) + sK_tC_s + K_tK_s}. \tag{2.47}$$

Al fine di riscrivere la (2.47) in una forma più agevole, si divide numeratore e denominatore del secondo membro per la massa sospesa $M$, ottenendo i seguenti termini di sostituzione:

$$\begin{aligned} \chi &= \frac{m}{M} \\ c &= \frac{C_s}{M} && (1/\text{sec}) \\ K_1 &= \frac{K_t}{M} && (1/\text{sec}^2) \\ K_2 &= \frac{K_s}{M} && (1/\text{sec}^2) \end{aligned} \tag{2.48}$$

Con la (2.48), la (2.47) diventa:

$$\frac{Z}{Z_r} = \frac{sK_1c + K_1K_2}{s^4\chi + s^3c(1 + \chi) + s^2(K_1 + \chi K_2 + K_2) + sK_1c + K_1K_2}. \tag{2.49}$$

Raccogliendo a numeratore e denominatore nella parte destra della (2.49) $K_1K_2$ e moltiplicando a numeratore e denominatore nella parte sinistra della stessa per $s^2$ si ottiene:

$$\frac{\ddot{Z}(s)}{\ddot{Z}_r(s)} = \frac{Ds + 1}{As^4 + Bs^3 + Cs^2 + Ds + 1}, \tag{2.50}$$

dove i termini $A$, $B$, $C$ e $D$ della (2.50) sono pari a:

$$\begin{aligned} A &= \frac{\chi}{K_1K_2} && (\text{sec}^4) \\ B &= c\frac{1 + \chi}{K_1K_2} && (\text{sec}^3) \\ C &= \frac{K_1 + K_2\chi + K_2}{K_1K_2} && (\text{sec}^2) \\ D &= \frac{c}{K_2} && (\text{sec}) \end{aligned} \tag{2.51}$$

Naturalmente la (2.50) può essere riscritta anche nel dominio della frequenza:

$$\frac{\ddot{Z}(j\Omega)}{\ddot{Z}_r(j\Omega)} = \frac{Dj\Omega + 1}{A(j\Omega)^4 + B(j\Omega)^3 + C(j\Omega)^2 + Dj\Omega + 1}. \tag{2.52}$$

La (2.52) è la generica risposta in frequenza del modello quarter–car.

### 2.6.4 Parametri tipici del modello Quarter–car

In letteratura [8, 40–45] vengono suggeriti diversi valori tipici per i parametri $K_s$, $K_c$, $C_s$, $M$ e $m$, come riportato in Tabella 2.1.

Tabella 2.1: Principali pubblicazioni relative ai modelli Quarter–car

| Autore(i) | Pubblicazione |
|---|---|
| B. R. Davis e A. G. Thompson | [40, Pag. 193–210] |
| Tetsuro Butsuen | [41, § 2.4, Pag. 33] |
| Thomas D. Gillespie | [8, Fig. 5.25, Pag. 163] |
| Ian Fialho e Gary J. Balas | [42, Table 1] |
| Natsiavas Verros e Papadimitriou | [43, § 5.1.] |
| James T. Allison | [44, § 8.3.3. Pag. 169–172] |
| M. M. M. Salem e A. Aly Ayman | [45, Table 1] |

Con questi parametri è possibile calcolare i dati numerici per la risposta in frequenza della (2.52). Dalle serie di parametri che si ottengono $A$, $B$, $C$ e $D$ si ricavano le diverse risposte in frequenza.

In Tabella 2.2, Tabella 2.3, Tabella 2.4, Tabella 2.5, Tabella 2.6, Tabella 2.7 e Tabella 2.8, riportate in ordine cronologico di pubblicazione, forniamo i valori tipici dei parametri delle sospensioni come riportato in letteratura.

Tabella 2.2: Dati forniti da B. R. Davis e A. G. Thompson

| Grandezza fisica | Valore | Unità di misura |
|---|---|---|
| $M$ | 288.9 | kg |
| $K_s$ | 19,960.4 | N m$^{-1}$ |
| $C_s$ | 1,300 | N s m$^{-1}$ |
| $m$ | 28.58 | kg |
| $K_t$ | 155,900 | N m$^{-1}$ |

### 2.6.5 Risposte in frequenza dei modelli Quarter–car

Attraverso i dati forniti in Tabella 2.2, Tabella 2.3, Tabella 2.4, Tabella 2.5, Tabella 2.6, Tabella 2.7 e Tabella 2.8, inseriti nella (2.48) e nella (2.51), otteniamo le risposte in

Tabella 2.3: Dati forniti da Tetsuro Butsuen

| Grandezza fisica | Valore | Unità di misura |
|---|---|---|
| $M$ | 240 | kg |
| $K_s$ | 16,000 | N $m^{-1}$ |
| $C_s$ | 1,000 | N s $m^{-1}$ |
| $m$ | 36 | kg |
| $K_t$ | 160,000 | N $m^{-1}$ |

Tabella 2.4: Dati forniti da Thomas D. Gillespie

| Grandezza fisica | Valore | Unità di misura |
|---|---|---|
| $M$ | 240 | kg |
| $K_s$ | 16,000 | N $m^{-1}$ |
| $C_s$ | 980 | N s $m^{-1}$ |
| $m$ | 36 | kg |
| $K_t$ | 160,000 | N $m^{-1}$ |

Tabella 2.5: Dati forniti da Ian Fialho e Gary J. Balas

| Grandezza fisica | Valore | Unità di misura |
|---|---|---|
| $M$ | 290 | kg |
| $K_s$ | 16,812 | N $m^{-1}$ |
| $C_s$ | 1,000 | N s $m^{-1}$ |
| $m$ | 59 | kg |
| $K_t$ | 190,000 | N $m^{-1}$ |

Tabella 2.6: Dati forniti da Natsiavas Verros e Papadimitriou

| Grandezza fisica | Valore | Unità di misura |
|---|---|---|
| $M$ | 375 | kg |
| $K_s$ | 15,000 | N $m^{-1}$ |
| $C_s$ | 1,425 | N s $m^{-1}$ |
| $m$ | 60 | kg |
| $K_t$ | 200,000 | N $m^{-1}$ |

Tabella 2.7: Dati forniti da James T. Allison

| Grandezza fisica | Comfort | Handling | Non ottimale | Unità di misura |
|---|---|---|---|---|
| $M$ | 325 | 325 | 325 | kg |
| $K_s$ | 500.4 | 505.33 | 1,000 | N $m^{-1}$ |
| $C_s$ | 24.67 | 1,897.9 | 1,000 | N s $m^{-1}$ |
| $m$ | 65 | 65 | 65 | kg |
| $K_t$ | 323,500 | 323,500 | 323,500 | N $m^{-1}$ |

frequenza della (2.52). I parametri della (2.52) sono riportati, in ordine cronologico di pubblicazione, nella Tabella 2.9. Tutti questi parametri (a eccezione di quelli calcolati partendo dai dati forniti da Allison [44]) presentano parametri aventi il medesimo ordine di grandezza. Questi parametri forniscono risposte in frequenza molto simili.

Di seguito mostriamo le equazioni, i grafici delle risposte in frequenza e le larghezze di

Tabella 2.8: Dati forniti da M. M. M. Salem e A. Aly Ayman

| Grandezza fisica | Valore | Unità di misura |
|---|---|---|
| $M$ | 250 | kg |
| $K_s$ | 16,000 | N m$^{-1}$ |
| $C_s$ | 1,500 | N s m$^{-1}$ |
| $m$ | 50 | kg |
| $K_t$ | 160,000 | N m$^{-1}$ |

Tabella 2.9: Valori numerici della risposta in frequenza del modello "Quarter–car"

| $K_s$, $K_c$, $C_s$, $M$, e $m$ | $A$ (s$^4$) | $B$ (s$^3$) | $C$ (s$^2$) | $D$ (s) |
|---|---|---|---|---|
| Davis e Thompson [40] | 2.653 $10^{-6}$ | 0.1326 $10^{-3}$ | 0.01651 | 0.06513 |
| Butsuen [41] | 3.375 $10^{-6}$ | 0.1078 $10^{-3}$ | 0.01672 | 0.06250 |
| Gillespie [8] | 3.375 $10^{-6}$ | 0.1057 $10^{-3}$ | 0.01673 | 0.06125 |
| Fialho e Balas [42] | 5.356 $10^{-6}$ | 0.1093 $10^{-3}$ | 0.01909 | 0.05948 |
| Verros e altri [43] | 7.500 $10^{-6}$ | 0.2066 $10^{-3}$ | 0.02717 | 0.09500 |
| Allison (*Comfort*) [44] | 1.305 $10^{-4}$ | 0.5943 $10^{-4}$ | 0.65069 | 0.04930 |
| Allison (*Handling*) [44] | 0.129 $10^{-6}$ | 4.5278 $10^{-3}$ | 0.64435 | 3.75576 |
| Salem e Aly [45] | 4.883 $10^{-6}$ | 0.1758 $10^{-3}$ | 0.01750 | 0.09375 |

banda ottenute con i dati elaborati e riportati in Tabella 2.9.

Con i parametri di Davis e Thompson [40, Pag. 193–210] si ottiene questa risposta in frequenza:

$$S_1(s) = \frac{0.06513s + 1}{2.653\ 10^{-6}s^4 + 0.1326\ 10^{-3}s^3 + 0.01651s^2 + 0.06513s + 1}, \tag{2.53}$$

dalla quale si ricava la risposta in modulo e fase di Figura 2.5.

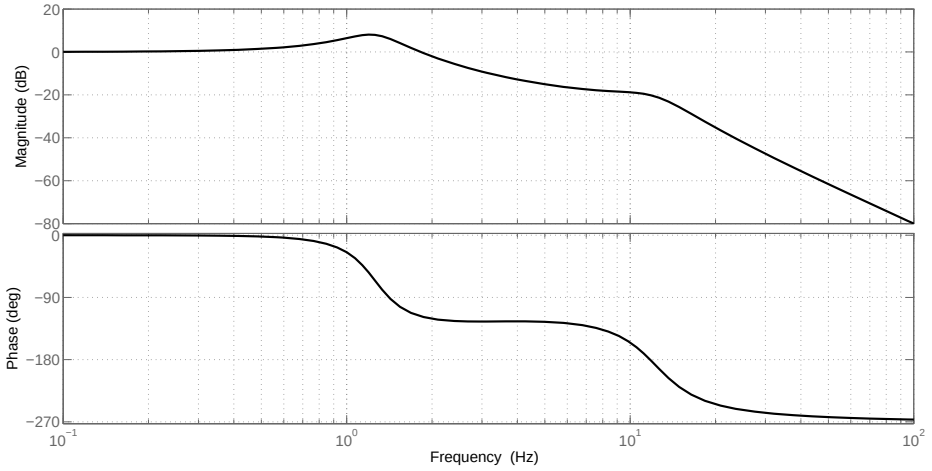



Figura 2.5: Risposta in frequenza di Davis e Thompson.

La larghezza di banda della (2.53) ha frequenza di taglio pari a $F_c = 2.0926$ Hz.

Con i parametri di Tetsuro Butsuen [41, § 2.4, Pag. 33] si ottiene la seguente risposta in frequenza:

$$S_2(s) = \frac{0.0625s + 1}{3.375\ 10^{-6}s^4 + 0.05943\ 10^{-3}s^3 + 0.01672s^2 + 0.0625s + 1}, \tag{2.54}$$

dalla quale si ricava la risposta in modulo e fase di Figura 2.6.

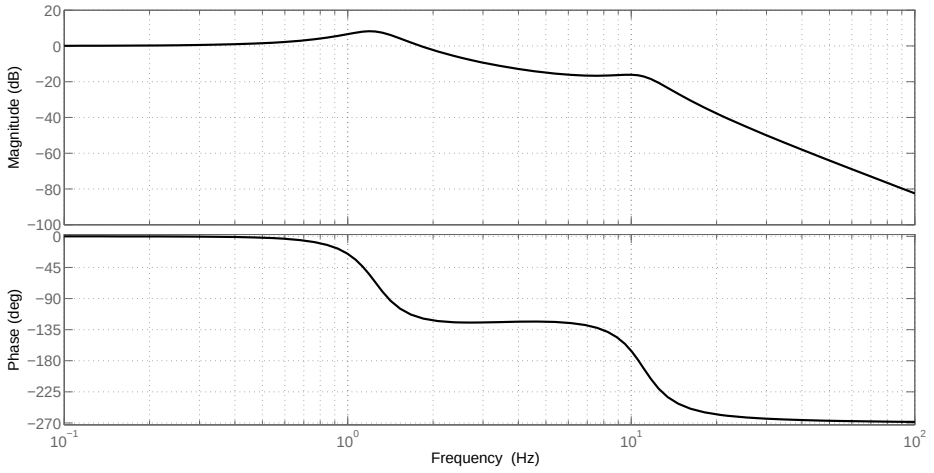



FIGURA 2.6: Risposta in frequenza di Butsuen.

La larghezza di banda della (2.54) ha frequenza di taglio pari a $F_c = 2.0674$ Hz.

Con i parametri di Thomas D. Gillespie [8, Figure 5.25, Pag. 163] si ottiene la seguente risposta in frequenza:

$$S_3(s) = \frac{0.06125s + 1}{3.375\ 10^{-6}s^4 + 0.1057\ 10^{-3}s^3 + 0.01672s^2 + 0.06125s + 1}, \tag{2.55}$$

dalla quale si ricava la risposta in modulo e fase di Figura 2.7.

La larghezza di banda della (2.55) ha frequenza di taglio pari a $F_c = 2.0623$ Hz.

Con i parametri di Ian Fialho e Gary J. Balas [42, Table 1] si ottiene la seguente risposta in frequenza:

$$S_4(s) = \frac{0.05948s + 1}{5.356\ 10^{-6}s^4 + 0.1093\ 10^{-3}s^3 + 0.01909s^2 + 0.05948s + 1}, \tag{2.56}$$

dalla quale si ricava la risposta in modulo e fase di Figura 2.8.

La larghezza di banda della (2.56) ha frequenza di taglio pari a $F_c = 1.9170$ Hz.

Con i parametri di Verros, Natsiavas e Papadimitriou [43, § 5.1.] si ottiene la seguente risposta in frequenza:

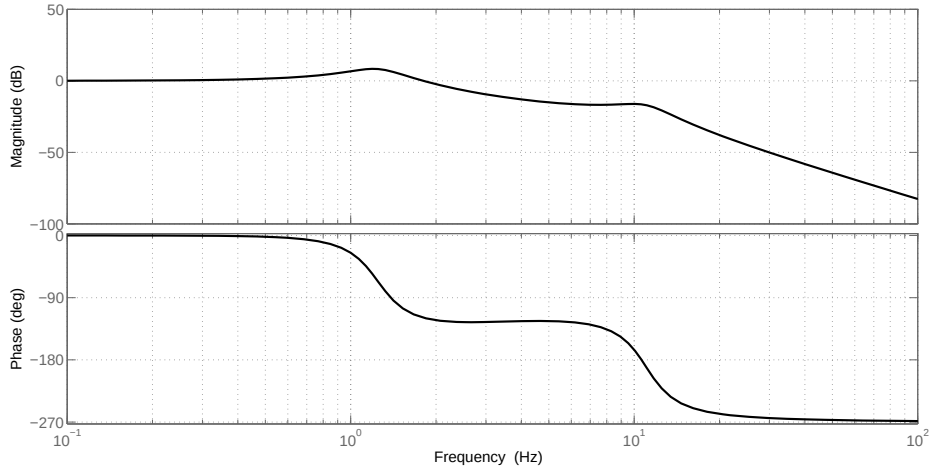



FIGURA 2.7: Risposta in frequenza di Gillespie.

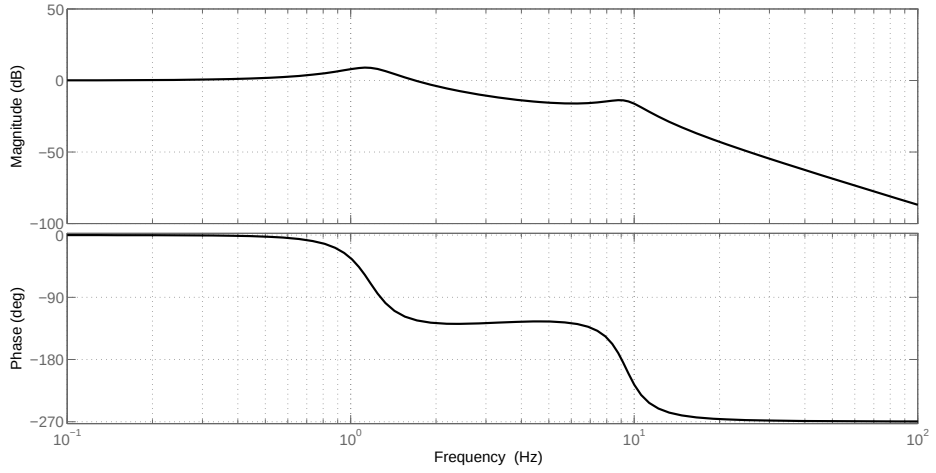



FIGURA 2.8: Risposta in frequenza di Fialho e Balas.

$$S_5(s) = \frac{0.095s + 1}{7.500\ 10^{-6}s^4 + 0.2066\ 10^{-3}s^3 + 0.02717s^2 + 0.095s + 1}, \tag{2.57}$$

dalla quale si ricava la risposta in modulo e fase di Figura 2.9.

La larghezza di banda della (2.57) ha frequenza di taglio pari a $F_c = 1.6497$ Hz.

Con i parametri di Allison [44, § 8.3.3. Pag. 169–172], utilizzando i parametri "comfort" (come da Tabella 2.7), si ottiene la seguente risposta in frequenza:

$$S_6(s) = \frac{0.0493s + 1}{0.1305\ 10^{-3}s^4 + 0.2066\ 10^{-3}s^3 + 0.6507s^2 + 0.0493s + 1}, \tag{2.58}$$

dalla quale si ricava la risposta in modulo e fase di Figura 2.10.

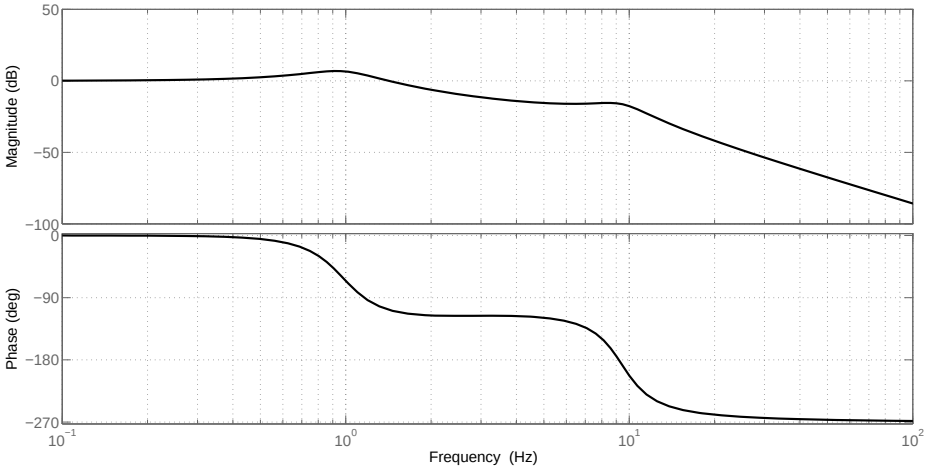



FIGURA 2.9: Risposta in frequenza di Verros, Natsiavas e Papadimitriou.

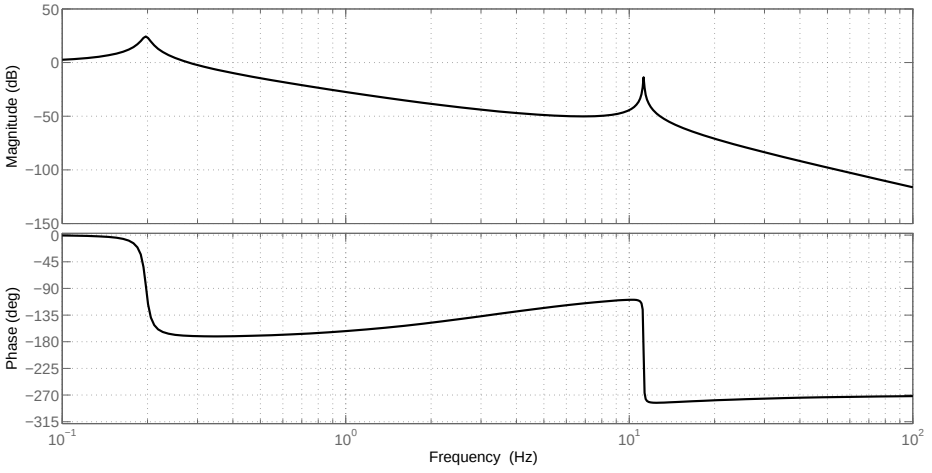



FIGURA 2.10: Risposta in frequenza di Allison con i parametri "comfort".

La larghezza di banda della (2.58), con parametri "comfort", ha frequenza di taglio pari a $F_c = 0.3068$ Hz.

Invece, utilizzando i parametri "handling" (come da Tabella 2.7), si ottiene la seguente risposta in frequenza:

$$S_6(s) = \frac{3.756s + 1}{0.1292\ 10^{-6}s^4 + 4.528\ 10^{-3}s^3 + 0.6443s^2 + 3.756s + 1}, \tag{2.59}$$

dalla quale si ricava la risposta in modulo e fase di Figura 2.11.

La larghezza di banda della (2.58), con parametri "handling", ha frequenza di taglio pari a $F_c = 1.0177$ Hz.

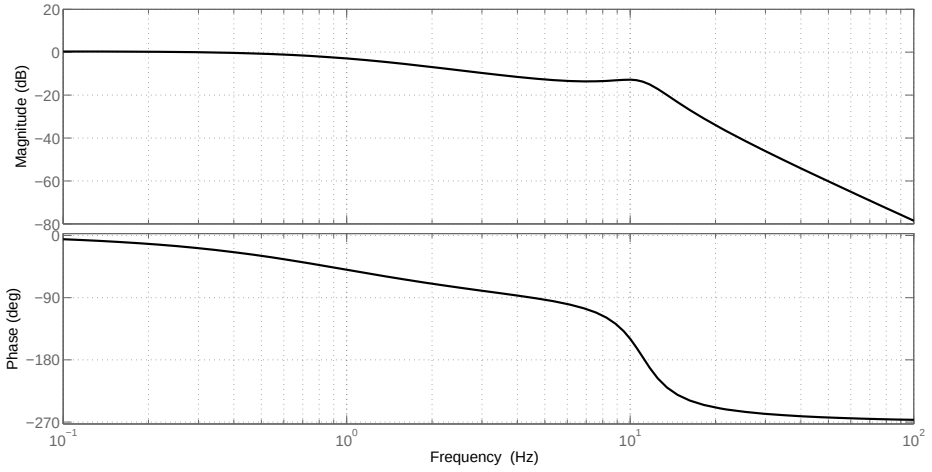



FIGURA 2.11: Risposta in frequenza di Allison con i parametri "handling".

Con i parametri di Salem e Aly [45, Table 1] si ottiene la seguente risposta in frequenza:

$$S_7(s) = \frac{0.09375s + 1}{4.883\ 10^{-6}s^4 + 0.1758\ 10^{-3}s^3 + 0.0175s^2 + 0.09375s + 1}, \tag{2.60}$$

dalla quale si ricava la risposta in modulo e fase di Figura 2.12.

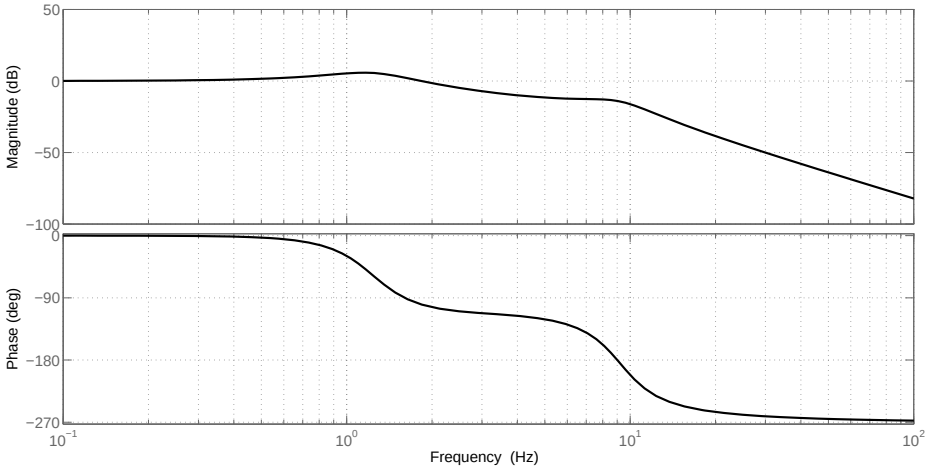



FIGURA 2.12: Risposta in frequenza di Salem e Aly.

La larghezza di banda della (2.60) ha frequenza di taglio pari a $F_c = 2.1959$ Hz.

Tutte le risposte in frequenza ottenute in questo paragrafo, calcolate partendo dai dati disponibili in letteratura, si sono rivelate analoghe in modulo e fase, a eccezione di quella ottenuta con i parametri forniti da [44]. Inoltre, tutte le risposte in frequenza mostrano una frequenza di taglio pari a circa $F_c = 2$ Hz.

È altrettanto interessante osservare anche i parametri numerici $A$, $B$, $C$ e $D$ delle risposte in frequenza che si ottengono elaborando i dati presenti in letteratura. Questi parametri presentano il medesimo ordine di grandezza e—in più casi—coincidono nelle prime due cifre significative.

## 2.7 Calcolo della potenza rilevata dall'accelerometro

I paragrafi precedenti forniscono tutti i dati necessari per il calcolo della potenza rilevata dall'accelerometro all'interno dell'abitacolo. È nota la densità spettrale di potenza che incontra un punto materiale quando attraversa il profilo stradale con velocità costante $v$. È altresì nota la risposta in frequenza delle sospensioni del veicolo.

Per il calcolo della potenza dell'accelerazione verticale misurata dall'accelerometro all'interno dell'abitacolo, si procede utilizzando il teorema di Parseval, scritto nella seguente forma:

$$P(v) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} S_{A_y A_y}(j\Omega) |S(j\Omega)|^2 \mathrm{d}\Omega, \tag{2.61}$$

dove, $S_{A_y A_y}(j\Omega)$ è la densità spettrale di potenza dell'accelerazione verticale rilevata sul manto stradale, calcolata attraverso la (2.29), la quale dipende dalla velocità $v$. $S(j\Omega)$, invece, è la risposta in frequenza delle sospensioni, data dalla (2.52).

La (2.61), scritta in forma completa, diventa pertanto:

$$P(v) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} q \left| v \frac{(j\Omega)^2}{pv + j\Omega} \right|^2 \left| \frac{Dj\Omega + 1}{A(j\Omega)^4 + B(j\Omega)^3 + C(j\Omega)^2 + Dj\Omega + 1} \right|^2 \mathrm{d}\Omega. \tag{2.62}$$

L'integrale di Parseval (2.62) viene calcolato attraverso il metodo della quadratura adattativa [46–48].

In Figura 2.13 sono mostrati gli andamenti della potenza dell'accelerazione verticale $P(v)$, calcolati per cinque diversi ordini di grandezza del polo $p$ al variare della velocità del veicolo $v$.

Nel caso di bassi valori del polo $p$, o per bassi valori di velocità $v$, $P(v)$ mostra una crescita costante fino a 20 dB/decade. Diversamente, quando $p$ o $v$ raggiungono valori elevati, $P(v)$ diminuisce la sua crescita, fino a mostrare un andamento costante.

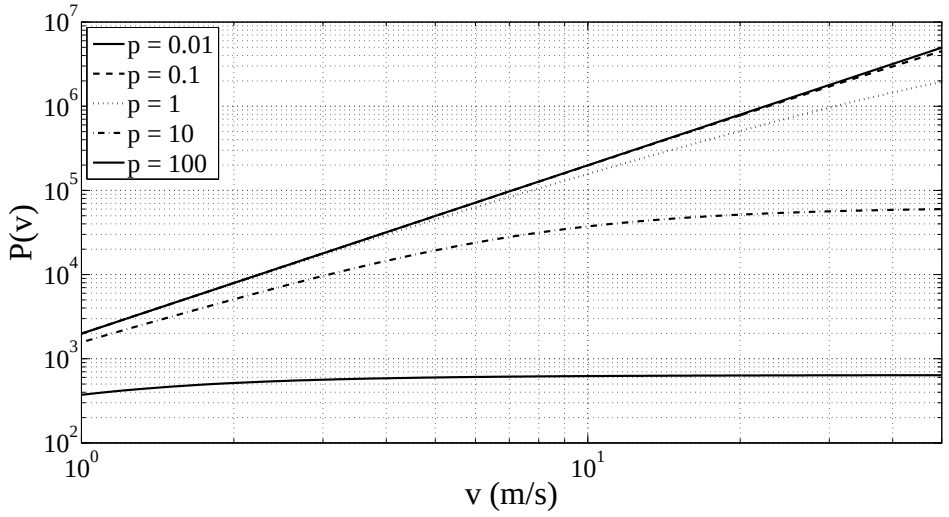



FIGURA 2.13: Andamento di $P(v)$ al variare della velocità $v$ e del polo $p$.

Questo comportamento può essere spiegato attraverso la (2.29), dove—per comodità di lettura—si definisce $\hat{p} = pv$, così da poter scrivere:

$$S_{A_y A_y}(j\Omega) = q \left| v \frac{(j\Omega)^2}{\hat{p} + j\Omega} \right|^2 . \tag{2.63}$$

Premesso questo, quando nella (2.63) $\hat{p} \ll j\Omega$, si ha:

$$S_{A_y A_y}(j\Omega) = q \left| v \frac{(j\Omega)^2}{\hat{p} + j\Omega} \right|^2 = v^2 q \Omega^2. \tag{2.64}$$

Diversamente, quando nella (2.63) si ha $\hat{p} \gg j\Omega$, si ottiene:

$$S_{A_y A_y}(j\Omega) = q \left| v \frac{(j\Omega)^2}{\hat{p} + j\Omega} \right|^2 = \frac{q}{p^2} \Omega^4. \tag{2.65}$$

Inserendo i risultati della (2.64) e della (2.65) nella (2.62)—e ricordando che $q$ e $S(j\Omega)$ sono indipendenti da $p$ e $v$—si evince che quando $\hat{p}$ tende a zero, allora $P(v)$ è proporzionale a $v^2$. Invece, quando $\hat{p}$ diverge, allora $P(v)$ risulta indipendente dalla velocità. Si ottiene infatti:

$$\begin{array}{lcl} \hat{p} \to 0 & \Rightarrow & P(v) \propto v^2 \\ \hat{p} \to +\infty & \Rightarrow & P(v) \propto v^0 \end{array} . \tag{2.66}$$

## 2.8 Funzione gamma

Nel caso generale, pertanto, è possibile approssimare localmente $P(v)$ con una legge gamma, che tenga conto della velocità del veicolo e della tipologia di asfalto, al fine di rendere $P(v)$ indipendente da questi due parametri. L'espressione usata è del tipo:

$$P(v) = \hat{q}v^{\gamma}, \tag{2.67}$$

dove ci si aspetta che l'esponente $\gamma$ cada nell'intervallo $[0, 2]$, come mostrato nella (2.66). A tal proposito, la Figura 2.14 mostra cinque diversi andamenti di $\gamma$ ottenuti dalla (2.62), per cinque ordini di grandezza del polo $p$ e differenti velocità del veicolo $v$.

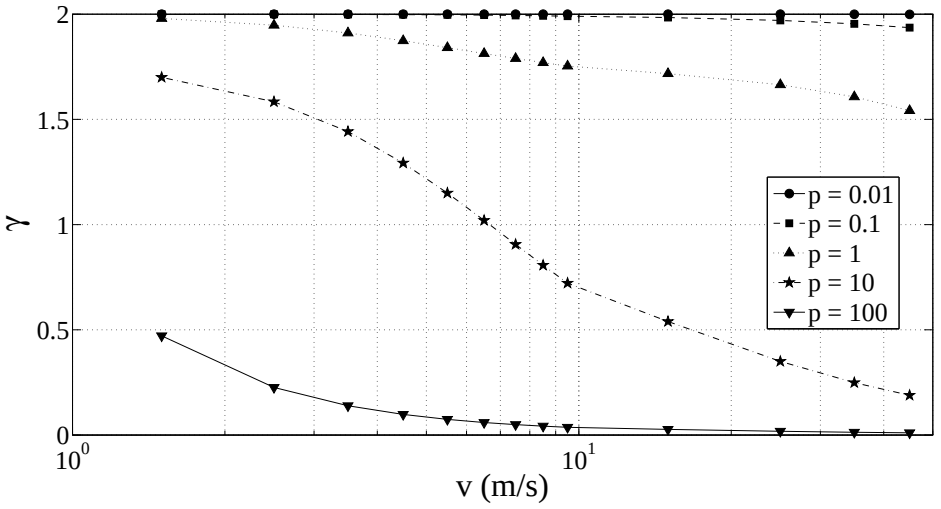



FIGURA 2.14: Andamento di $\gamma$ al variare della velocità $v$ e del polo $p$.

I valori di $\gamma$ mostrati in Figura 2.14 sono stati calcolati con una procedura di fitting lineare e ricorsivo del $\log[P(v)]$, il quale trasforma la (2.67), nella (2.68):

$$\log[P(v)] = \log(\hat{q}) + \gamma \log(v). \tag{2.68}$$

Come previsto e come mostrato nella Figura 2.14, $\gamma$ cade sempre all'interno dell'intervallo $[0, 2]$, ed ha un andamento monotono decrescente rispetto all'aumentare della velocità $v$. Per di più, alle velocità tipiche di un autoveicolo, in intervalli che vanno da 5 m/s sino a 30–40 m/s, il parametro $\gamma$ può spesso essere approssimato localmente con una costante.

Occorre notare che $\gamma$ è funzione della velocità del veicolo $v$, ma anche della qualità del manto stradale, come del sistema di sospensioni del veicolo. Infatti, vi è una dipendenza anche dal polo $p$ e da $S(j\Omega)$.

In sintesi, quando un veicolo percorre una strada con superficie uniforme (quindi con valori costanti dei parametri $p$ e $q$), ci si aspetta che la potenza dell'accelerazione verticale rilevata da un dispositivo mobile ancorato all'interno dell'abitacolo dipenda soltanto dalla velocità del veicolo. Questa dipendenza segue una legge gamma, con parametro $\gamma \in [0, 2]$ e con $\gamma$ che tende a decrescere all'aumentare del valore della velocità $v$ del veicolo.

Come mostrato nel Capitolo 5, i risultati sperimentali ottenuti con il database di Smart-RoadSense, confermano queste considerazioni teoriche.

# Capitolo 3

# Sistema di analisi su dispositivi mobili

Nel presente capitolo viene descritta la fase di sviluppo del *front–end* dell'applicativo SmartRoadSense. Nel Capitolo 4 e Capitolo 5 verranno mostrati i risultati sperimentali condotti con le tecniche e le metodologie descritte nel presente Capitolo.

Per determinare le anomalie del manto stradale, viene utilizzata una procedura che fa ricorso a un modello autoregressivo del sest'ordine, $AR(6)$ [49], con un approccio a finestre.

L'ipotesi di base è che quando l'asfalto percorso è omogeneo, le accelerazioni, all'interno di una finestra temporale limitata, sono predicibili. Quando non si verifica questa ipotesi è ragionevole supporre che nel manto stradale sia presente un'anomalia, anche se non è dato sapere quale. Lavorando con finestre temporali con ampiezza pari a un secondo e frequenze di campionamento pari a 100 Hz (quindi studiando 100 campioni per ogni secondo), si ha che quando un veicolo percorre una strada alla velocità di 180 km/h (la massima velocità presente nel database, corrispondente a 50 m/s) si registra un campionamento ogni 50 cm.

Infatti si ha:

$$d_{50} = v \cdot t = 50 \text{ m/s } 10 \cdot 10^{-3} \text{ s} = 0.5 \text{ m}. \tag{3.1}$$

Questo è quello che accade nel caso in cui si procede con la velocità è massima registrata. In tutti gli altri casi si ha un campionamento più fitto, fino al caso di 18 km/h (la velocità minima presa in considerazione) corrispondente a 5 m/s, dove si ha:

$$d_5 = v \cdot t = 5 \text{ m/s } 10 \cdot 10^{-3} \text{ s} = 0.05 \text{ m}. \tag{3.2}$$

Attraverso lo studio della potenza del residuo di predizione $P_{\mathrm{PE}}$ e dell'errore finale di predizione (sulla quale si basa la potenza del residuo di predizione, ottenuta tramite l'approccio di Ljung [50]) viene calcolata una stima della rugosità del manto stradale.

Gli *outliers*, dovuti ai dossi, vengono rimossi in una seconda fase, tramite una media mobile del decimo ordine.

## 3.1 Inizializzazione

La prima fase del programma, mostrata in Figura 3.1, consiste nello specificare le variabili e altre informazioni che verranno ereditate dalla parte restante del codice.

Questa porzione di codice, come altre che seguiranno, è stata collocata in questa sezione per due ragioni, entrambe strategiche:

1. poter accedere e variare i parametri del programma senza intervenire nel codice a più basso livello;
2. evitare di eseguire più volte stessi calcoli, con il conseguente effetto di rallentare l'esecuzione e richiedere un maggior consumo di batteria.

Il codice eseguito nella prima fase è il seguente:

```
1 s = 100;                      % Dimensione delle finestre
2 order = 6;                    % Ordine del modello AR(p)
3 overlap = 25;                 % Dimensione dell'overlap
4
5 Hs = s + overlap * 2 - 1;     % Creazione della finestra di Hamming
6 Ch = 2 * pi / Hs;
7 H = (0.54 - 0.46 * cos(Ch * (0:Hs))).';
```

FIGURA 3.1: Inizializzazione del programma.

In Figura 3.1, righe 5–7, viene creata la finestratura di Hamming per 150 campioni (tanti quanti la dimensione della finestra e degli overlap). Va da sé che se si desidera utilizzare una finestratura differente è sufficiente agire su queste righe che verranno ereditate nel resto del codice.

Dopo questa fase, si procede con l'elaborazione dei dati provenienti dall'accelerometro, finestra dopo finestra. I vettori contenenti questi campioni verranno chiamati $A_x$, $A_y$ e $A_z$.

```
1 A_x = A_x - mean(A_x);
2 A_y = A_y - mean(A_y);
3 A_z = A_z - mean(A_z);
```

FIGURA 3.2: Rimozione dei valori medi dalle accelerazioni.

Il primo passaggio, successivo all'acquisizione stessa, è la rimozione dei loro valor medi, come mostrato in Figura 3.2.

A questo punto è possibile procedere con il calcolo della potenza dell'errore di predizione $P_{\text{PE}}$ per ciascuna componente dell'accelerazione.

```
1 ppe_x = ppe(A_x, s, order, overlap, H);
2 ppe_y = ppe(A_y, s, order, overlap, H);
3 ppe_z = ppe(A_x, s, order, overlap, H);
```

FIGURA 3.3: Calcolo delle potenze dell'errore di predizione.

Si noti che il frammento di codice in Figura 3.3 richiede i valori della finestratura già calcolati, finestra che deve avere ampiezza pari a `length(H) = s + 2 · overlap`. Le funzioni che seguiranno non effettuano alcun controllo relativo a questa struttura. Questo—come anticipato—poiché le funzioni devono contenere il minor numero di operazioni possibili.

Questo modo di procedere, oltre a presentare un eccellente vantaggio computazionale, consente di poter utilizzare qualsiasi altro tipo di finestratura si ritenga adeguato e di cambiarla.

## 3.2 Calcolo della potenza dell'errore di predizione

Il primo passaggio è generare l'overlap dei campioni, suddividendo i dati acquisiti in colonne, dove ciascuna colonna rappresenta un elemento all'interno della finestra di osservazione.

Per semplicità, si illustra l'esempio di un vettore di 100 elementi da 1 a 100, dove si desidera ottenere finestre di 10 elementi con overlap (prima e dopo) di 2 elementi. Il risultato è mostrato in Tabella 3.1.

Si noti che i primi quattro elementi sono nulli, in modo tale da arrivare a 10 con la prima colonna. Così avviene anche nel caso pratico e la prima colonna di dati viene rigettata. Questo avviene perché l'overlap richiesto è generato su dati non noti. Inoltre, va da

TABELLA 3.1: Esempio di scomposizione di un vettore con dieci elementi per ogni finestra e overlap di due elementi.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 7 | 17 | 27 | 37 | 47 | 57 | 67 | 77 | 87 |
| 0 | 8 | 18 | 28 | 38 | 48 | 58 | 68 | 78 | 88 |
| 0 | 9 | 19 | 29 | 39 | 49 | 59 | 69 | 79 | 89 |
| 0 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 |
| 1 | 11 | 21 | 31 | 41 | 51 | 61 | 71 | 81 | 91 |
| 2 | 12 | 22 | 32 | 42 | 52 | 62 | 72 | 82 | 92 |
| 3 | 13 | 23 | 33 | 43 | 53 | 63 | 73 | 83 | 93 |
| 4 | 14 | 24 | 34 | 44 | 54 | 64 | 74 | 84 | 94 |
| 5 | 15 | 25 | 35 | 45 | 55 | 65 | 75 | 85 | 95 |
| 6 | 16 | 26 | 36 | 46 | 56 | 66 | 76 | 86 | 96 |
| 7 | 17 | 27 | 37 | 47 | 57 | 67 | 77 | 87 | 97 |
| 8 | 18 | 28 | 38 | 48 | 58 | 68 | 78 | 88 | 98 |
| 9 | 19 | 29 | 39 | 49 | 59 | 69 | 79 | 89 | 99 |
| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |

sé, che pure l'ultima colonna potrebbe contenere elementi nulli, poiché si arresta una volta raggiunto il suo valore massimo e forza la colonna stessa a contenere 14 campioni (2 + 10 + 2). Se anche nell'ultima colonna sono presenti elementi nulli, anche questa finestratura viene scartata.

```
1  temp = temp - sum(temp)/length(temp);    % Rimozione del valor medio
2  temp = temp .* window;                   % Applica la finestratura
3
4  λ = ar_model(temp, order);               % Stima del modello AR(order)
5
6  for j = 1:s                              % Calcola l'errore di predizione
7      k = overlap + j;
8      ϵ(j) = λ * temp(k : -1 : k - order);
9  end
10
11 y(i) = sum(ϵ.^2)/s;                      % Calcola la potenza dell'errore
12                                          % di predizione
```

FIGURA 3.4: Calcolo delle potenze dell'errore di predizione.

Non deve preoccupare il fatto che, nell'intero processo di acquisizione dati, vengano scartate due finestre. Si tratta—al più—di due secondi di campionamento rispetto all'intero tragitto (tipicamente della durata di decine di minuti, quando non di ore).

I dati dell'accelerometro vengono memorizzati in un buffer temporaneo, chiamato `temp`. Nel momento in cui viene elaborata una generica colonna di dati, si ricorre al codice riportato in Figura 3.4.

Come evidente dalla Figura 3.4, il passaggio successivo è la stima del filtro predittivo:

$$\lambda = \begin{pmatrix} 1 \\ \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \\ \lambda_5 \\ \lambda_6 \end{pmatrix}. \tag{3.3}$$

## 3.3 Stima del modello autoregressivo

La stima del filtro autoregressivo avviene sfruttando la ricorsione di Levinson–Durbin [35, 36]. Invece, per il calcolo dei coefficienti di autocorrelazione, in luogo di una antitrasformata e trasformata di Fourier, si ricorre al metodo diretto [51]. Il codice utilizzato è riassunto in Figura 3.5.

```
% Calcola i coefficienti di autocorrelazione
R = autocorr2(x, p);

% Rimuove la componente negativa del vettore
R = R(floor(max(size(R))/2)+1:end);

% Calcola il filtro autoregressivo
λ = levinson_durbin(R);
```

FIGURA 3.5: Calcolo del filtro autoregressivo.

Per il calcolo dei coefficienti di autocorrelazione si utilizza il codice riportato in Figura 3.6. Si è scelto di ricorrere a questo metodo poiché, nel caso più complesso di $R(6)$, il suo costo computazionale è pari a sole 921 moltiplicazioni.

Diversamente, se si fosse ricorso alla trasformata di Fourier veloce (Fast Fourier Transform, FFT), al prodotto dei due vettori risultanti e all'antitrasformata del risultato, il numero di moltiplicazioni sarebbe stato pari a:

$$M_{\text{TOT}} = \frac{N \log_2 N}{2} + N + \frac{N \log_2 N}{2}, \tag{3.4}$$

dove $N$ è la prima potenza intera di 2 successiva a 150, ovvero 256.

Pertanto—ricordando che le operazioni da eseguire sono complesse—di fatto ogni moltiplicazione complessa corrisponde a 4 moltiplicazioni reali. Il numero di moltiplicazioni, che determina la complessità computazione dell'algoritmo è pertanto pari a:

$$M_{\text{TOT}} = \left( \frac{256 \log_2 256}{2} + 1024 + \frac{256 \log_2 256}{2} \right) \times 4 = 12,288. \tag{3.5}$$

L'algoritmo di Figura 3.6 [51], richiede—invece—l'esecuzione di 921 moltiplicazioni. Pertanto, è circa dieci volte più veloce del metodo che ricorre alla trasformata di Fourier veloce.

```
1  R = zeros(1, maxlag + 1);    % Preallocazione dei dati
2  z = [];                      % Usato per spostare i dati
3
4  for k = 1:maxlag + 1
5      % Calcola il k-esimo elemento dell'autocorrelazione
6      R(k) = sum([z; x].*[x; z]);
7      z = zeros(k, 1);
8  end
9
10 R = [R(end:-1:2) R];         % Costruisce la parte negativa del vettore
11 R = R(:);                    % Forza il vettore in colonna
```

FIGURA 3.6: Calcolo dell'autocorrelazione senza trasformata di Fourier.

Per capire come opera l'algoritmo di Figura 3.6 è possibile fare riferimento ad un vettore (ad esempio di sei elementi). Questo vettore viene moltiplicato ogni volta per i suoi elementi (da qui il termine autocorrelazione, diversamente si parlerebbe di cross–correlazione) mentre questi dati "scorrono" in avanti. L'autocorrelazione, infine, è data dalla somma delle moltiplicazioni effettuate.

Nel caso ipotizzato, il calcolo si esegue come mostrato in Tabella 3.2.

TABELLA 3.2: Calcolo dell'autocorrelazione del vettore $\vec{x} = (1\ 2\ 3\ 4\ 5\ 6)$

| Dati | | | | | | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 6 | 12 | 18 | 24 | 30 | 36 | | | | | |
| | | 5 | 10 | 15 | 20 | 25 | 30 | | | | |
| | | | 4 | 8 | 12 | 16 | 20 | 24 | | | |
| | | | | 3 | 6 | 9 | 12 | 15 | 18 | | |
| | | | | | 2 | 4 | 6 | 8 | 10 | 12 | |
| | | | | | | 1 | 2 | 3 | 4 | 5 | 6 |
| Risultato | 6 | 17 | 32 | 50 | 70 | 91 | 70 | 50 | 32 | 17 | 6 |
| | $R(-5)$ | $R(-4)$ | $R(-3)$ | $R(-2)$ | $R(-1)$ | $R(0)$ | $R(1)$ | $R(2)$ | $R(3)$ | $R(4)$ | $R(5)$ |

È molto importante notare la simmetria del risultato. Se si osserva il codice di Figura 3.6, riga 10, si nota che i coefficienti di autocorrelazione vengono calcolati soltanto nella componente positiva, poiché in quella negativa risultano identici.

A questo punto, per il calcolo del filtro predittore, si ricorre alla ricorsione di Levinson–Durbin, il cui algoritmo è stato anticipato in Figura 2.1.

Di tale codice è opportuno ricordare che—per come è stato concepito—non calcola soltanto i coefficienti $\lambda_i$, ma l'intero filtro predittore, sfruttabile direttamente per il calcolo dell'errore di predizione, come avviene in Figura 3.4, riga 8 e—conseguentemente—della potenza dell'errore di predizione, riga 11.

## 3.4 Indice di rugosità

Una volta calcolate le potenze dei residui di predizione per ogni componente dell'accelerazione $ppe_x$, $ppe_y$ e $ppe_z$ se ne calcola il valor medio, poiché tutte le componenti dell'accelerazione forniscono un loro contributo.

$$P_{\mathrm{PE}} = \overline{ppe_x + ppe_y + ppe_z} \tag{3.6}$$

La (3.6) è l'indice di rugosità del manto stradale. Non è dato sapere quale sia l'orientamento del dispositivo mobile di un generico utente. Imporne una calibrazione iniziale è un'ipotesi praticabile, ma non è possibile sapere se questa avviene in un tratto di strada pianeggiante o in salita, poiché le componenti gravitazionali variano durante ogni finestratura, dove—a rigore—si dovrebbe rieseguire una nuova calibrazione ogni secondo. Va da sé che il costo computazionale di un siffatto algoritmo non è sostenibile se si desidera realizzare un'applicazione a basso consumo energetico.

Infine l'indice di rugosità si ottiene applicando una media mobile del decimo ordine a $P_{\mathrm{PE}}$, come mostrato in Figura 3.7, per rimuovere gli outlier dovuti ai bump, il cui codice è riportato in Figura 3.8. Le variabili `pre` e `post` servono per evitare che il risultato sia sfasato in ritardo rispetto ai dati d'ingresso.

```
1      % Ordine della media mobile
2      order = 10;
3
4      % Calcolo della media mobile della potenza del residuo di predizione
5      ppe_ma = moving_average(ppe_m, order);
```

FIGURA 3.7: Calcolo del residuo finale di predizione con media mobile.

La media mobile, viene calcolata una volta nota la potenza del residuo di predizione secondo dopo secondo. Quindi il suo calcolo è rinviato al server. Va da sé che potrebbe essere calcolata (con un approccio logaritmico) anche sul dispositivo mobile. Ma si

```
N = length(x);

pre = zeros(o/2, 1);
y = zeros(N-o, 1);
post = pre;

% Primi o/2 campioni
pre(1) = x(1);
for i = 2:2:o-2
    pre((i+2)/2) = sum(x(1:i))/(i+2)/2;
end

% Restante parte del segnale
for i = o+1:N
    y(i-o) = sum(x(i-o:i))/(o+1);
end

% Ultimi o/2 campioni
for i = 1:o/2-1
    post(i) = sum(x(end-2*i:end))/(2*i);
end
post(5) = x(end);

% Compilazione del vettore
y = [pre ; y ; post];
```

FIGURA 3.8: Calcolo della media mobile aritmetica.

preferisce questa soluzione, poiché alleggerisce il costo computazionale dell'algoritmo, relativamente al lato *client*.

## 3.5 Costo computazionale dell'algoritmo

In conclusione, è opportuno calcolare il numero di moltiplicazioni al secondo richieste dall'algoritmo, al fine di valutare le sue prestazioni.

Il calcolo per ogni componente $ppe_{\{x,y,z\}}$ richiede un numero di moltiplicazioni come mostrato in Tabella 3.3.

Pertanto, per il calcolo delle tre potenze dei residui di predizione, si ha un costo computazionale pari a $1,918 \times 3 + 150 = 5,904$ moltiplicazioni al secondo.

Per amor del vero, a queste vanno sommate anche le operazioni eseguite per il calcolo della media mobile. Tuttavia queste operazioni vengono eseguite in una seconda fase, una volta inviati i dati al server remoto, quindi non incidono nelle prestazioni del dispositivo mobile.

Tabella 3.3: Numero di moltiplicazioni necessarie per il calcolo della potenza del residuo di predizione per ciascuna componente dell'accelerazione.

| Operazione | Moltiplicazioni |
|---|---|
| Finestratura di Hamming | 150 |
| Autocorrelazione di 150 dati | 921 |
| Levinson–Durbin | 196 |
| Residuo di predizione | 700 |
| Potenza del residuo di predizione | 101 |
| TOTALE | 2,068 |
| TOTALE (SENZA FINESTRATURA) | 1,918 |

Una doverosa nota a margine. Il solo costo dell'autocorrelazione, se eseguito con la trasformata di Fourier veloce, sarebbe stato pari a 12,288 operazioni da, moltiplicare per un fattore 3, pari quindi a 36,864 moltiplicazioni.

Il calcolo dell'autocorrelazione attraverso la sua definizione, riduce questo costo computazionale di un fattore superiore a 40.

# Capitolo 4

# Primi esperimenti su segnali acquisiti

Nella prima fase di ricerca e acquisizione dati si è condotta una campagna preliminare di esperimenti aventi come finalità la taratura dei modelli matematici proposti nel Capitolo 2 e implementati mediante gli algoritmi illustrati nel Capitolo 3.

Pertanto, questa fase è servita a determinare quali valori numerici devono essere assegnati come parametri ottimali al modello matematico che verrà in seguito adottato per le acquisizioni su vasta scala.

Partendo da sette tracce, raccolte in Urbino, contenenti:

- timestamp;
- dati dell'accelerometro triassiale;
- dati del sistema di localizzazione GPS;

si è proceduto alla loro analisi per eseguire il settaggio del sistema di elaborazione dei segnali.

Ad onor del vero, il sistema forniva anche la possibilità di utilizzare due *flag* nei quali poteva essere specificata dall'utente la tipologia di manto stradale attraversato. Si è scelto, volutamente, di ignorare tali dati poiché—una volta avviata la campagna di acquisizione dati vera e propria—tale opzione sarebbe forzatamente scomparsa. È infatti improponibile chiedere all'utente finale di interagire col sistema durante la guida (oltre che contrario ad ogni norma e regola di sicurezza stradale). Infine, il sistema che si vuole tarare è "cieco" rispetto al manto stradale (viene utilizzato l'accelerometro, non la

telecamera). Per queste ragioni si è deciso di ignorare, fin dalla prima acquisizione, ogni eventuale dato aggiuntivo che—in una seconda fase—non sarebbe stato più possibile utilizzare.

Attraverso lo studio dell'errore di previsione finale (FPE, Final Prediction Error) è possibile determinare quale ordine del modello autoregressivo verrà adottato nell'elaborazione dei segnali provenienti dall'accelerometro.

L'analisi dei dati avviene mediate un approccio a finestrature. Questa scelta è strategica, al fine di poter considerare valida l'ipotesi di segnale stazionario. Naturalmente l'ampiezza della finestra deve—allo stesso tempo—aver durata sufficientemente al fine di consentire una stima accurata del filtro predittore.

Infine, l'indice di rugosità viene filtrato attraverso una media mobile, così da poter ottenere una serie statistica privata degli outlier generati dai bump che nulla hanno a che vedere col reale andamento del manto stradale.

## 4.1 Primi dati sperimentali

I primi dati sperimentali sono stati ottenuti con un'applicazione che registra soltanto i dati provenienti dall'accelerometro e il sistema di localizzazione GPS. Tale applicazione, dal momento del suo avvio, crea—nella memoria del dispositivo mobile—un archivio basato su file di testo (CSV, Comma–Separated Values). Questo archivio contiene il timestamp, le tre componenti dell'accelerazione, i dati georeferenziati provenienti dal sistema GPS (latitudine, lontitudine, quota, velocità).

SmartRoadSense, come protocollo, utilizza una frequenza di campionamento pari alla frequenza massima di lavoro dell'accelerometro triassiale (100 Hz[1]).

I dati di partenza riguardano tracciati di manti stradali eterogenei, siti nel comune di Urbino (Italia), riportati in Figura 4.1. Le strade sono state percorse sette volte con il medesimo veicolo e il medesimo dispositivo mobile. I percorsi, coinvolgono diverse strade e—volutamente—in più casi si sovrappongono. Questa scelta consente di eseguire una prima validazione dei parametri in esame, nel caso si ottenessero evidenze sperimentali palesemente in disaccordo tra loro.

I dati in possesso, sono stati ottenuti attraverso sette differenti acquisizioni, dove—scartando i dati non allocabili, poiché privi di segnale GPS—si hanno i seguenti campioni utili, riportati in Tabella 4.1.

[1] Va riportato che, oggigiorno, sono disponibili in commercio dispositivi mobili con frequenze di campionamento pari a 200 Hz. Tuttavia, nel progetto di ricerca, si è preferito uniformare la frequenza di campionamento a 100 Hz per avere la sicurezza di ottenere, da tutti i dispositivi mobili, dati congruenti.


FIGURA 4.1: Percorsi analizzati nella prima fase sperimentale.

TABELLA 4.1: Numero di campioni utili per ogni acquisizione.

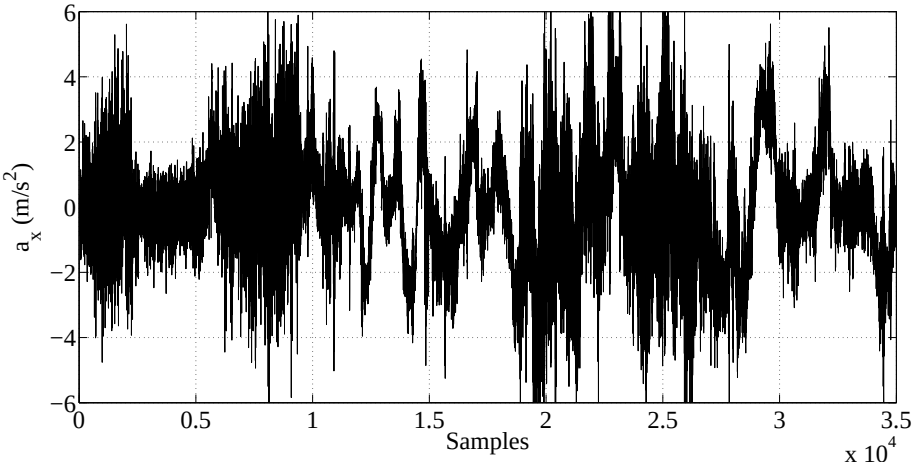
| Traccia | Campioni |
|---|---|
| SRS–01 | 46,464 |
| SRS–02 | 3,729 |
| SRS–03 | 13,920 |
| SRS–04 | 11,125 |
| SRS–05 | 11,106 |
| SRS–06 | 14,073 |
| SRS–07 | 9,995 |

L'insieme di questi dati, uniti ad altri acquisiti contestualmente con diversi veicoli e differenti dispositivi mobili, sempre ai fini di validazione, sono propedeutici agli esperimenti condotti nel Capitolo 5. La finalità degli studi effettuati in questo capitolo è:

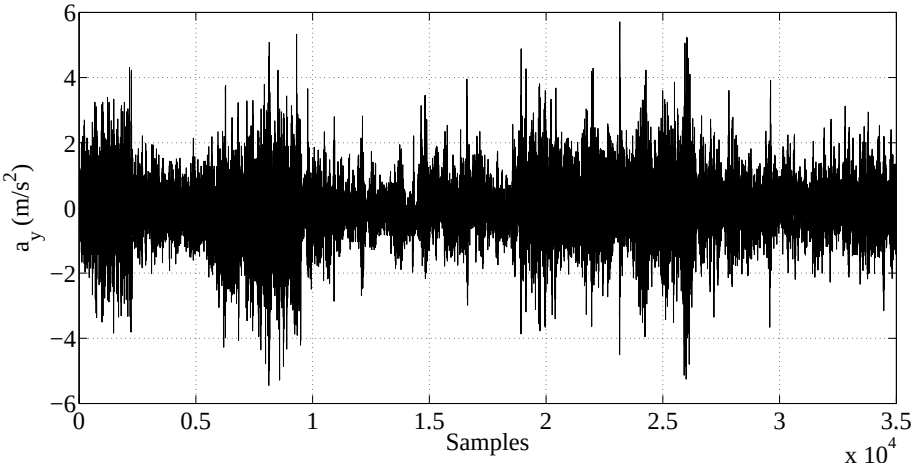
- individuare l'ordine ottimo del modello autoregressivo;
- scegliere il miglior tipo di finestratura dei dati;
- ottenere un algoritmo fruibile all'interno del progetto SmartRoadSense.

## 4.2 Monitoraggio delle diverse accelerazioni

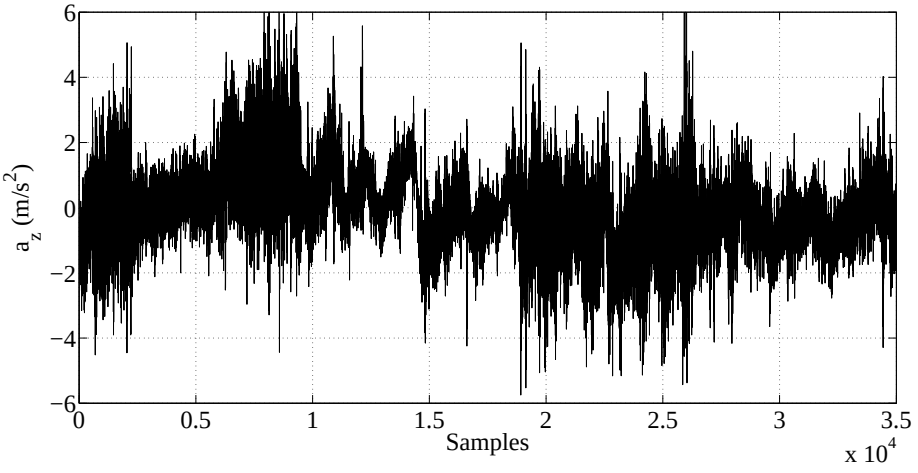
Molti studi presenti in letteratura si concentrano esclusivamente sull'analisi dell'accelerazione verticale per determinare le anomalie del manto stradale. Tuttavia la Figura 4.2 evidenzia come ogni componente dell'accelerazione percepita dall'accelerometro nell'abitacolo, contenga una componente utile alla descrizione del manto stradale. Diversamente, in questo lavoro, sono state prese in esame tutte le accelerazioni.




(a) Accelerazione orizzontale $A_x$.




(b) Accelerazione verticale $A_y$.




(c) Accelerazione trasversale $A_z$.

FIGURA 4.2: Componenti dell'accelerazione $A_x$, $A_y$ e $A_z$ per il percorso `SRS-01`.

Inoltre, nelle prove di laboratorio, è possibile eseguire tarature anche molto precise. La stessa cosa non può essere richiesta all'utente finale. Anche per questa ragione utilizzare tutte le componenti di accelerazione, ai fini del calcolo della rugosità del manto stradale, è strategico. Infatti non è possibile determinare a priori come verrà ancorato rispetto al suolo il dispositivo mobile. Vi sono—inoltre—salite e discese che fanno sì che anche una calibrazione eseguita ad inizio tragitto, possa essere fatta in modo errato.

Nella Figura 4.2(a), Figura 4.2(b) e Figura 4.2(c) viene presentata ogni componente dell'accelerazione dove stato rimosso il suo valor medio. Operazione che non solo elimina l'accelerazione di gravità, indipendentemente dalla pendenza della strada percorsa, ma rende possibile posizionare il dispositivo mobile con qualsiasi orientamento.

Va sottolineato che la procedura di rimozione del valor medio di ogni componente dell'accelerazione, oltre a rimuovere l'accelerazione di gravità, ovunque questa si rifletta, presenta ulteriori benefici.

L'azione di rimozione del valor medio dell'accelerazione, eseguita finestra dopo finestra, contente di rimuovere tutte le accelerazioni che presentano un valore costante all'interno della finestra temporale in esame, come le accelerazioni centrifughe dovute alle curve, accelerazioni non brusche, leggere frenate e altro. Tutte le accelerazioni che presentano un valore costante, all'interno della finestra di osservazione, vengono così rimosse a priori.

## 4.3 Stima dell'ordine del filtro predittivo

È possibile calcolare l'errore di previsione finale (FPE, Final Prediction Error) secondo Akaike [52], per il modello stimato, al variare dell'ordine del modello autoregressivo e per i percorsi stradali a disposizione.

L'errore di previsione finale secondo Akaike [50] fornisce una misurazione della qualità del modello, simulando la situazione in un contesto dove il modello viene testato con un set di dati diverso. Dopo aver testato diversi modelli, è possibile confrontarli utilizzando questo criterio. Secondo la teoria di Akaike, il modello più accurato restituisce il più piccolo errore di previsione finale.

Se si utilizzano gli stessi set di dati, sia per il modello di stima, sia per la validazione, il fit migliora con l'aumentare dell'ordine del modello. Di conseguenza, migliora la flessibilità della struttura del modello stesso.

L'errore di previsione finale secondo Akaike è definito dalla (4.1) [50, Paragrafi 7.4 e 16.4].

$$FPE = \det\left(\frac{1}{N}\sum_{n=1}^{N} e\left(n,\hat{\theta}_N\right) e\left(n,\hat{\theta}_N\right)^T\right)\frac{1+\frac{d}{N}}{1-\frac{d}{N}}, \tag{4.1}$$

dove $N$ è il numero di valori nel set di dati da stimare; $e(t)$ è un vettore degli errori di predizione di dimensione $n \times 1$; $\theta_N$ rappresenta la stima dei parametri; infine, $d$ è il numero di parametri stimati.

Questa analisi è stata ripetuta per tutti i percorsi e considerando ordini dei modelli autoregressivi crescenti da $AR(1)$ fino a $AR(20)$. La procedura genera, pertanto, un grafico con sette curve, una per ciascun percorso.

Tale grafico viene studiato per determinare il punto in cui le curve raggiungono il loro minimo. Tale punto rappresenta l'ordine ottimo per il modello $AR(p)$. Infatti, l'utilizzo di modelli di ordine più elevato non apporterebbe benefici. Verrebbe aumentata la complessità computazionale del sistema, senza migliorare la stima nella previsione dei dati.

Il frammento di codice utilizzato per questa analisi è riportato in Figura 4.3.

```
1  for k = 1:N                                % N = Number of paths (7)
2      load(['SRS-0' num2str(i)]);            % Load route data
3      AY = M(:, 3);                          % Extract A_y(t)
4      AY = AY - mean(AY);                    % Remove gravity from A_y(t)
5      for p = 1:20                           % FOR AR(1) TO AR(20)
6          m = ar(AY, p);                     % Compute AR(p) model
7          f(p, k) = fpe(m);                  % Compute FPE of AR(p) model
8      end
9  end
10
11 figure, plot(f, 'LineWidth', 2);           % Plot FPE vs order AR model
12 figure, plot(diff(f), 'LineWidth', 2);      % Plot derivative FPE vs AR model
```

FIGURA 4.3: Frammento di codice per le analisi riportate in Figura 4.4 e Figura 4.5.

I valori di uscita del codice di Figura 4.3, sono visibili in Figura 4.4. Rappresentano l'analisi dell'errore di previsione finale per i diversi tracciati di Tabella 4.1.

L'analisi della derivata discreta dell'andamento dell'errore di previsione finale, riportata in Figura 4.5, evidenzia come per tutti i tracciati presi in esame, con un modello autoregressivo oltre il sesto ordine non vengono restituite predizioni più accurate. Pertanto, nell'analisi LPC, si adotta il modello $AR(6)$: il più economico a livello computazionale e a parità di risultati ottenuti.

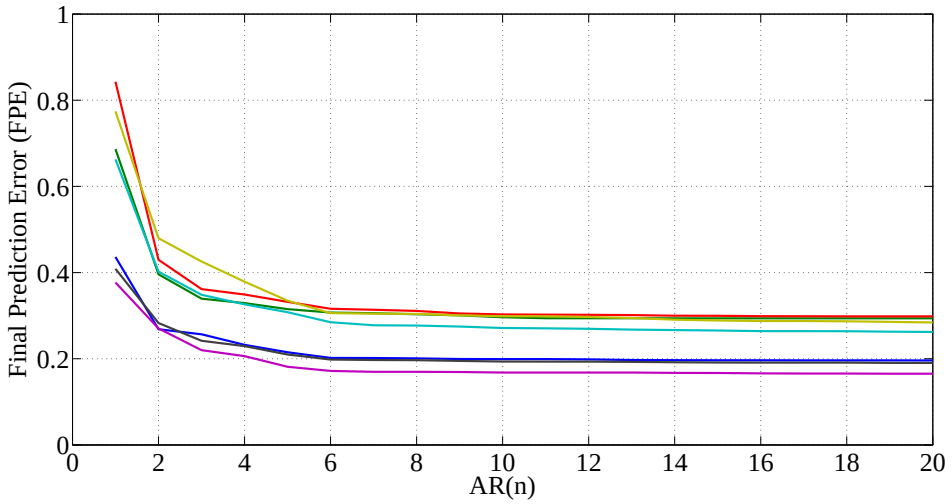



FIGURA 4.4: Analisi del FPE sui tracciati al variare dell'ordine del modello AR.

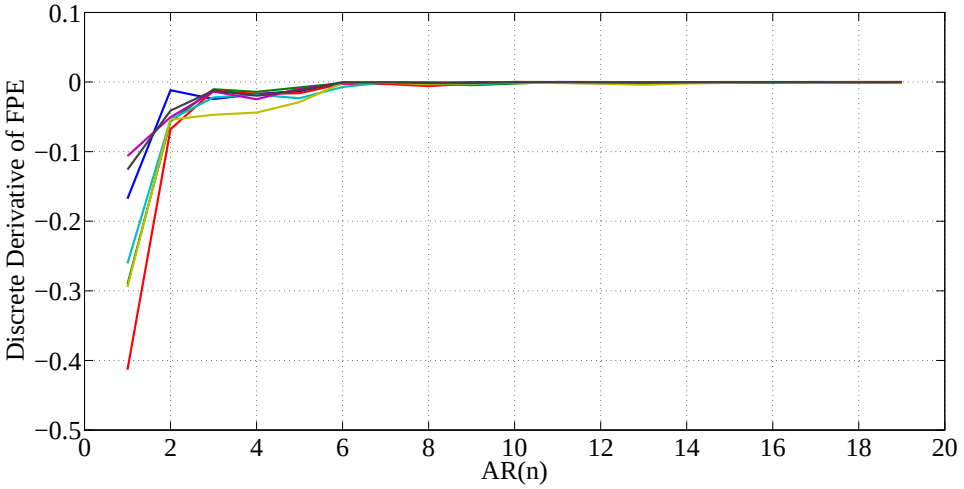



FIGURA 4.5: Variazioni del FPE rispetto all'ordine del modello AR.

## 4.4 Calcolo dei coefficienti di autocorrelazione $R(k)$

Un metodo molto utilizzato, per il calcolo dei coefficienti di autocorrelazione $R(k)$ di una generica variabile $x(n)$, si poggia sulle proprietà della trasformata di Fourier, in particolare sul teorema della convoluzione:

$$\mathcal{F}\{x(n) * y(n)\} = X(j\Omega) \cdot Y(j\Omega). \tag{4.2}$$

La (4.2) consente di calcolare i coefficienti di autocorrelazione $R(p)$ come segue:

$$R(p) = \mathcal{F}^{-1}\{\mathcal{F}\{x(n)\}\mathcal{F}^{*}\{x(n)\}\}. \tag{4.3}$$

Nel caso in esame, dovendo calcolare i coefficienti di autocorrelazione fino al sest'ordine, si è scelto di eseguire questi calcoli applicando la definizione. Così facendo l'algoritmo, computazionalmente, diventa molto più leggero. Quest'approccio consente di sviluppare applicazioni per dispositivi mobili garantendo una maggior durata della batteria.

## 4.5 Finestratura dei dati

Per calcolare il filtro predittore e l'errore di predizione è stato applicato un approccio a finestre del segnale campionato $a(n)$, diviso in segmenti di $M = 100$ campioni. Il numero di campioni è stato scelto pari a 100 per armonizzare il campionamento dei segnali provenienti dall'accelerometro (avente $F_s = 100$ Hz) con quelli provenienti dal GPS (avente $F_s = 1$ Hz).

Con questo modo di procedere, si ottiene un parametro $M$ sufficientemente ampio, tale d'avere un'accurata stima del filtro predittore e—allo stesso tempo—sufficientemente ridotto, per poter considerare il segnale stazionario.

Tuttavia, al diminuire della durata del segnale, si riduce la sua risoluzione in frequenza. Volendo selezionare soltanto una porzione di segnale, la sequenza dei campioni avrà necessariamente un inizio e una fine. Pertanto il numero dei campioni a disposizione sarà un numero finito.

Si consideri un approccio a finestre, applicato a un generico segnale campionato $a(n)$. Tale segnale viene prelevato mediante un'opportuna finestra $w(n)$ (*window*) chiamata anche "finestra di osservazione".

In letteratura esistono numerose tipologie di finestre, differenti dalla finestra rettangolare che preleva il segnale così com'è. L'applicazione di una generica finestra di osservazione sul segnale, può essere scritta come segue:

$$a_w(n) = a(n)w(n). \tag{4.4}$$

Nel dominio della frequenza, un segnale finestrato si presenta come la convoluzione dei due spettri:

$$A_w(j\Omega) = A(j\Omega) * W(j\Omega). \tag{4.5}$$

Per questa ragione, nell'elaborazione numerica dei segnali—in luogo della cosiddetta "finestra rettangolare"—si preferisce ricorrere all'adozione di curve non negative con decadimento "a campana" [53].

Un esempio di finestratura molto popolare in letteratura è la finestra di Hamming [54], nota anche come "coseno rialzato", descritta dalla (4.6) e mostrata in Figura 4.6.

$$H(n) = \alpha - \beta \cos\left(\frac{2\pi n}{N-1}\right), \tag{4.6}$$

dove $\alpha = 0.5383$, $\beta = 1 - \alpha = 0.4617$, $N$ è il numero di campioni, infine $n$ è il campione in esame.

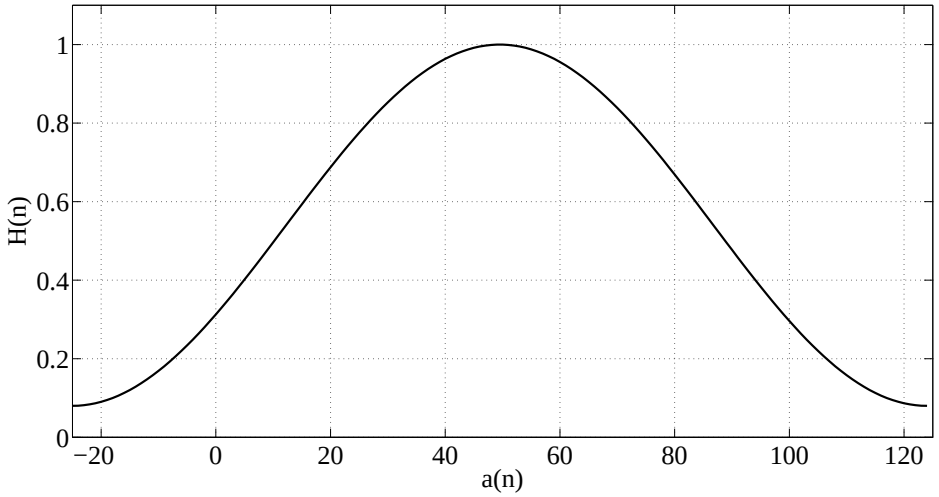



FIGURA 4.6: Finestra di Hamming con $\alpha = 0.54$, $\beta = 0.46$ per 150 campioni.

## 4.6 Overlapping

Nella fase di finestratura dei dati che verrà utilizzata in seguito, non viene considerata una finestra esatta di 100 campioni. A questa si aggiunge un *overlap*, in testa e in coda, di altri 25 campioni. La finestra di Hamming, quindi, viene applicata a 150 campioni complessivi.

L'overlap è strategico per l'analisi LPC, in quanto non è possibile fare predizioni sui campioni allo stato presente, né sui sei campioni precedenti—quelli necessari per stimare il campione $n$–esimo.

Premesso questo, si è scelto di considerare una finestra di 100 campioni, ai quali si aggiungono 25 campioni in testa e altrettanti 25 campioni in coda. Questi campioni

servono per inizializzare il processo autoregressivo, formando così una struttura iterativa come mostrato in Figura 4.7, dove i primi 25 campioni sono gli ultimi 25 del blocco precedente, mentre gli ultimi 25 campioni, sono i primi 25 del blocco successivo, e così via.

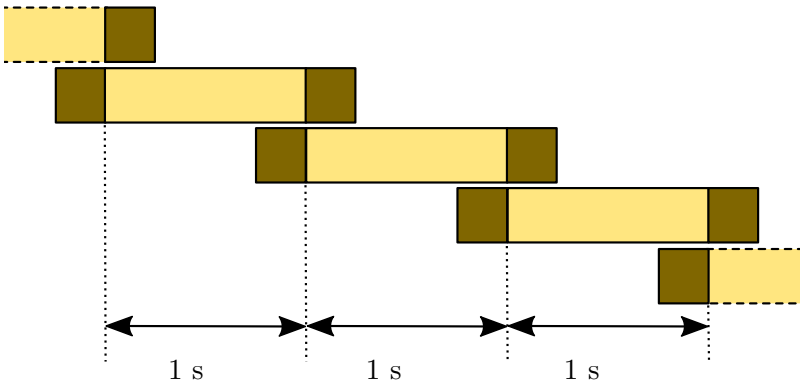



FIGURA 4.7: Modello iterativo dell'overlap. Nella zona in cui vi è intersezione i campioni sono i medesimi.

## 4.7 Media mobile dei dati

Il successivo passaggio consiste nel ridurre l'eccessiva sensibilità del modello sin qui realizzato che, se applicato letteralmente, fornirebbe in ogni istante una serie di valori difficili da utilizzare ai fini pratici. Per ottenere un risultato più coerente col manto stradale in esame si procede calcolando una media mobile dei risultati ottenuti al fine di eliminare gli outlier dovuti ai bump.

La Figura 4.8 mostra, nella sua curva nera, l'andamento dell'indice di rugosità $R_I$ estratto con l'analisi LPC senza elaborazioni. La curva smooth (disegnata in rosso) corrisponde al medesimo indice di rugosità filtrato attraverso una media mobile di ordine 10, ottenendo così un dato meno sensibile alle singole anomalie stradali. Il risultato, pertanto, è un parametro che fornisce una visione d'insieme del profilo altimetrico stradale.

Il caso in esame in Figura 4.8 è quello di un veicolo che percorre una strada alla velocità di 40 km/h.

Il generico campione $P_{\mathrm{PE}}(n)$ è dato dalla (4.7):

$$P_{\mathrm{PE}}(n) = \frac{1}{O+1} \sum_{k=n-O/2}^{n+O/2} R_I(k). \tag{4.7}$$

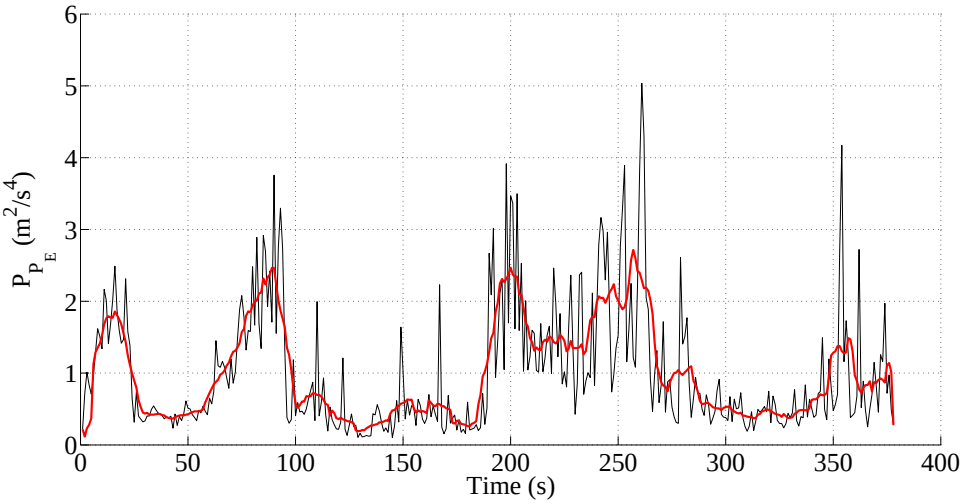



FIGURA 4.8: $R_I$ (curva nera) e indice di rugosità smooth $P_{\mathrm{PE}}$ (curva rossa) di un veicolo che procede a 40 km/h.

Questo parametro è sensibile al tipo di veicolo utilizzato, al dispositivo mobile e allo stile di guida del conducente. Per contro, le informazioni contenute in questo singolo parametro, possono essere facilmente raccolte e memorizzate. Sono i dati provenienti da diversi veicoli, dispositivi e conducenti che transitano sulla stessa strada a fornire un parametro espressivo della qualità del manto stradale.

# Capitolo 5

# Esperimenti su moli di dati

## 5.1 Tipologie di esperimenti eseguiti

Il Server di SmartRoadSense registra continuamente diversi pacchetti d'informazioni inviate dai dispositivi mobili degli utenti. In particolare, per ciascun pacchetto dati inviato, l'indice di rugosità, le coordinate GPS e la velocità del veicolo vengono raccolti e salvati. Ufficialmente, il progetto SmartRoadSense è iniziato nel febbraio 2015.

Di seguito vengono riportati due categorie di evidenze sperimentali, entrambe acquisite dal database di SmartRoadSense, il quale—attualmente (31 dicembre 2015)—ha al suo interno un patrimonio di 6,001,617 singole rilevazioni del manto stradale italiano, corrispondenti a 1.8 miliardi valori di accelerazioni.

Questi dati sono stati ottenuti tramite l'utilizzo di 432 differenti veicoli, equipaggiati con 147 diversi dispositivi mobili Android. Tutti questi dati, corrispondono a oltre 24,664.4 km di strade italiane, un valore corrispondente al 5.06 % dell'intera rete stradale nazionale [1, 30].

Al fine di verificare le assunzioni teoriche illustrate nel paragrafo 2.2, sono stati concepiti due serie di esperimenti. Il primo esperimento, discusso nel paragrafo 5.3, confronta due diversi dispositivi mobili, i quali percorrono un asfalto omogeneo (un'autostrada e una strada statale). Il primo esperimento, discusso nel paragrafo 5.3, prevede l'utilizzo di un'unica macchina e di un solo dispositivo mobile, oltre al percorso su un asfalto uniforme, riducendo così la variabilità delle condizioni sperimentali.

Nel paragrafo 5.4, vengono considerati tutti i 432 veicoli (equipaggiati con 147 diversi dispositivi mobili), per valutare i risultati ottenuti con diversi tipi di asfalto, differenti veicoli, e diversi dispositivi mobili.


FIGURA 5.1: Mappatura stradale effettuata tramite il progetto SmartRoadSense.

In Figura 5.1 viene mostrata la mappatura delle strade italiane ottenuta grazie ai dati forniti dai 432 veicoli che hanno aderito al progetto SmartRoadSense nei primi nove mesi di mappatura: dal 21 febbraio 2015, al 21 novembre 2015. Grazie a questa operazione di *crowdsourcing* si sono potuti ottenere questi risultati in un lasso di tempo molto breve, riuscendo così ad avere un quadro dettagliato dello stato del 5.06 % delle strade italiane (24,664.4 km).

## 5.2 Definizione della mappa colorimetrica

Nella mappa di Figura 5.1 i colori utilizzati sono dieci: dal verde, utilizzato per rappresentare i più bassi livelli di rugosità del manto stradale, fino al rosso, utilizzato per rappresentare gli elevati livelli di rugosità del manto stradale.

Questi colori sono definiti utilizzando il utilizzando il codice riportato in Figura 5.2, dal quale si ottiene la tavola colorimetrica adottata nell'applicazione SmartRoadSense, riportata in Tabella 5.1.

```
% Load all SmartRoadSense data
load('dump.mat')

% Removes all anomalous data
PPE = removenan(ppe);
PPE = PPE(PPE < 5);
PPE = PPE(PPE > 1E-3);

% Sorts the data
PPE2 = sort(PPE);

% Defines the vector of colorimetric scale
ID = zeros(10, 1);

% Sets the first nine steps
for k = 1:9
% L = half of elements
L = floor(length(PPE2)/2);
% ID(k) = level of step k
ID(k) = PPE2(L);
% Initializes the compute of next level
PPE2 = PPE2(L+1:end);
end

% Sets the last level
ID(10) = PPE2(end);
```

FIGURA 5.2: Frammento di codice per il calcolo della scala colorimetrica utilizzata nel progetto SmartRoadSense.

La tavola colorimetrica ha un andamento logaritmico e ogni colore copre metà dei campioni presenti nel database.

In prima istanza si era utilizzata una mappa colorimetrica definita senza riscontri con dati sperimentali. La grande mole di dati raccolti, ha consentito di generare una mappa colorimetrica più fedele alla realtà, in quanto data da oltre 6 milioni di indici di rugosità raccolti in tutt'Italia, con oltre 432 differenti veicoli sui quali sono stati installati oltre 147 differenti modelli di dispositivi mobili. Questi numeri sono la garanzia di una statistica dell'indice di rugosità affidabile.

Tabella 5.1: Tavola colorimetrica generata dal codice di Figura 5.2

| Livello | $P_{\text{PE}}$ | Colore |
|---|---|---|
| 1 | $\geq 0.0000$ | |
| 2 | $> 0.0811$ | |
| 3 | $> 0.2058$ | |
| 4 | $> 0.3824$ | |
| 5 | $> 0.5861$ | |
| 6 | $> 0.8311$ | |
| 7 | $> 1.1207$ | |
| 8 | $> 1.4040$ | |
| 9 | $> 1.6392$ | |
| 10 | $> 1.7985$ | |

Naturalmente, quando si raccolgono ulteriori dati, è fortemente consigliabile rieseguire questo calcolo, al fine di validare i valori della colonna 2 della Tabella 5.1.

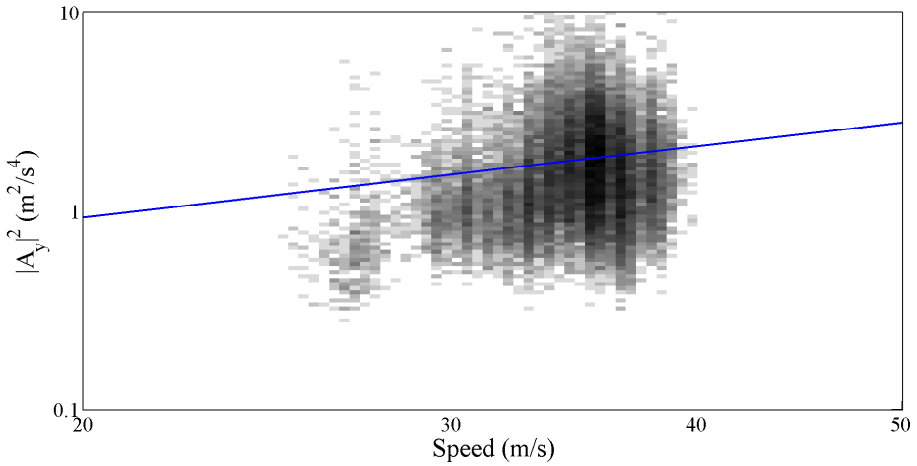
## 5.3 Esperimenti su manto stradale omogeneo

Il primo esperimento è stato realizzato estraendo dal database i dati prodotti da un veicolo noto, con a bordo un LG Nexus 4. Il veicolo ha viaggiato da Trieste fino a Pesaro (433 km), passando per le autostrade A13 e A14. L'autostrada è la strada con più alta performance in Italia. È caratterizzata da un controllo di accesso, ha due o più corsie indipendenti per ogni senso di marcia, e una pavimentazione uniforme realizzata mediante asfalto drenante. Questa situazione può essere interpretata come un caso di studio ideale.

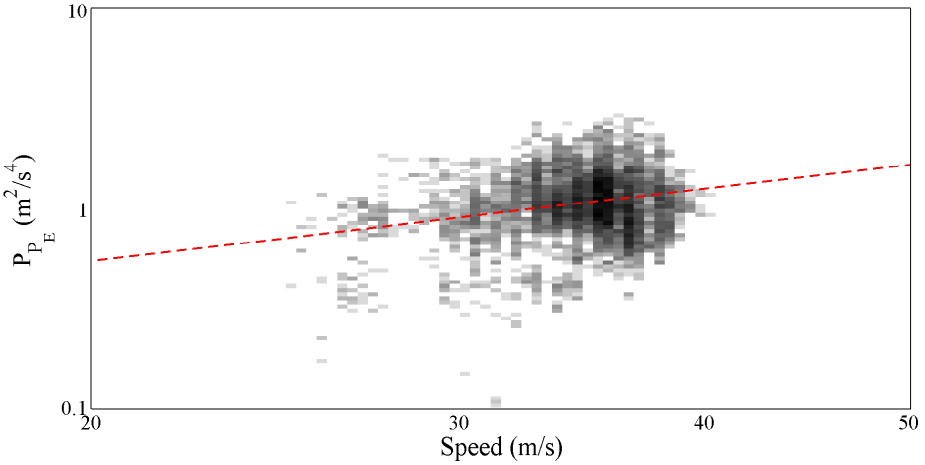
Durante questo esperimento il veicolo ha raccolto 2,485,800 campioni di accelerazione, i quali sono stati usati per calcolare i $P_{\text{PE}}$ su finestre di 100 campioni. La Figura 5.3 mostra sia i campioni dell'accelerazione verticale in Figura 5.3(a), sia i 24,858 valori di $P_{\text{PE}}$ calcolati in Figura 5.3(b).

Al fine di aumentare la facilità di lettura e interpretazione di queste figure—come di quelle che seguiranno—si è proceduto suddividendo l'asse delle ascisse e delle ordinate in 200 intervalli e colorando ogni singolo intervallo con una tonalità di grigio proporzionale al numero di campioni che cadono in quell'intervallo (bianco, nel caso di assenza di campioni; nero, quando il numero di campioni è massimo). Con questa procedura è possibile capire non solo dove i campioni sono presenti, ma soprattutto evidenziare dove la loro presenza è maggiore o minore.

In entrambi i grafici di Figura 5.3 viene tracciata la legge gamma calcolata in base alla (2.67). Come previsto, i campioni raccolti sono distribuiti in un breve intervallo di velocità, a partire da circa 25 m/s fino a circa 40 m/s.




(a) Campioni di $|A_y|^2$.



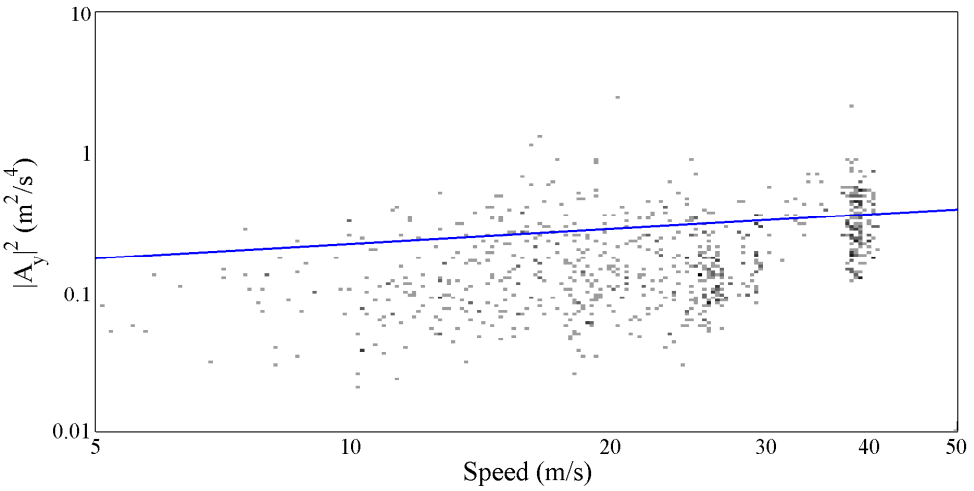

(b) Campioni di $P_{\mathrm{PE}}$.

FIGURA 5.3: Confronto dei campionamenti di $|A_y|^2$ e $P_{\mathrm{PE}}$ su autostrada.

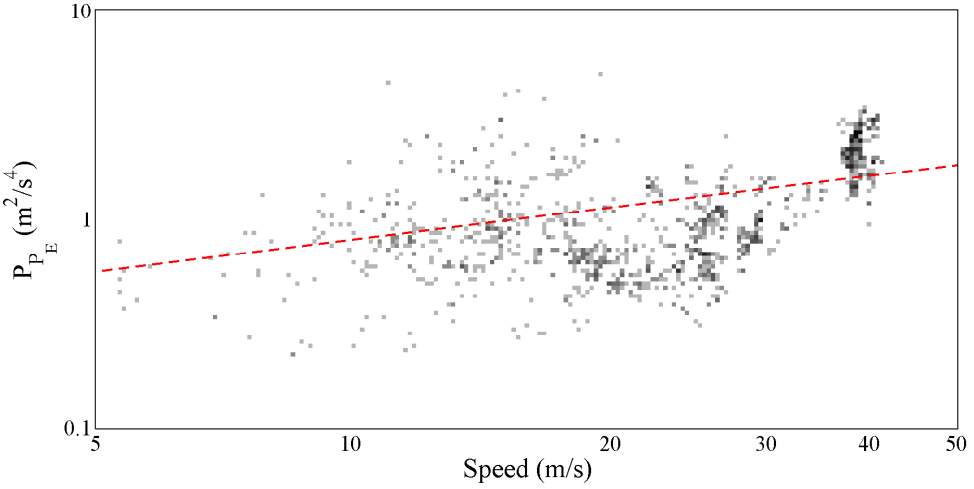
Un secondo esperimento, eseguito con la stessa metodologia, è stato condotto utilizzando un veicolo differente con a bordo un Samsung S3, il quale ha viaggiato nella strada statale che collega Pesaro ad Ancona, percorrendo circa 75 km, e raccogliendo 383,000 campioni di accelerazione. Una strada statale è una tipologia di strada deputata al collegamento tra grandi città, normalmente con due corsie non separate da una barriera centrale. Possiamo utilizzare questo tipo di strada come caso studio dove la qualità del manto stradale possa essere considerata quasi costante, ma di qualità inferiore rispetto a quella di un'autostrada.

La Figura 5.4 mostra i campionamenti delle accelerazioni verticali e dei relativi $P_{\mathrm{PE}}$ con

i rispettivi andamenti delle leggi gamma. Come previsto, i campioni raccolti sono distribuiti più uniformemente in una vasta gamma, da 10 m/s fino a 40 m/s, diversamente da quelli ottenuti in autostrada, i quali mostravano velocità maggiori.




(a) Campioni di $|A_y|^2$.




(b) Campioni di $P_{\mathrm{PE}}$.

FIGURA 5.4: Confronto dei campionamenti di $|A_y|^2$ e $P_{\mathrm{PE}}$ su strada statale.

La Tabella 5.2 riassume il curve fitting delle leggi gamma calcolate nei due precedenti esperimenti, sia per l'accelerazione verticale $|A_y|^2$, sia per la potenza del residuo di predizione $P_{\mathrm{PE}}$ con i rispettivi intervalli di confidenza.

Nell'esperimento condotto lungo un'autostrada il calcolo del parametro $\hat{q}$ ottenuto attraverso i dati dell'accelerazione $|A_y|^2$ è, approssimativamente, 26.60 $10^{-3}$. Mentre, l'esponente $\gamma$ è pari a circa 1.19. Il medesimo curve fitting, calcolato sui campioni della potenza del residuo di predizione $P_{\mathrm{PE}}$ restituisce un $\hat{q}$ e un esponente $\gamma$ rispettivamente di circa 14.93 $10^{-3}$ e 1.21 che sostengono le tesi teoriche, descritte nelParagrafo 2.7

TABELLA 5.2: Curve fitting dei parametri $P(v) = \hat{q}v^{\gamma}$ per manti stradali omogenei.

| Coefficienti (col 95% dell'intervallo di confidenza) | | | |
|---|---|---|---|
| Percorso | | $\hat{q}$ | $\gamma$ |
| Autostrada | $\lvert A_y\rvert^2$ | 26.60 $10^{-3}$ (15.78 $10^{-3}$, 18.01 $10^{-3}$) | 1.19 (1.06, 1.32) |
| | $P_{\mathrm{PE}}$ | 14.93 $10^{-3}$ (18.54 $10^{-3}$, 23.62 $10^{-3}$) | 1.21 (1.11, 1.31) |
| Strada Provinciale | $\lvert A_y\rvert^2$ | 101.2 $10^{-3}$ (71.76 $10^{-3}$, 130.6 $10^{-3}$) | 0.35 (0.25, 0.44) |
| | $P_{\mathrm{PE}}$ | 245.3 $10^{-3}$ (211.9 $10^{-3}$, 278.7 $10^{-3}$) | 0.51 (0.47, 0.55) |

e Paragrafo 2.8: i dati acquisiti seguono una legge gamma con $\gamma \in [0, 2]$.

Questo significa che il $P_{\mathrm{PE}}$ delle tre componenti di accelerazione triassiali (l'indice di rugosità del progetto SmartRoadSense), dovrebbe essere correlata alla velocità del veicolo secondo la stessa legge gamma dell'accelerazione verticale $|A_y|^2$.

Osservazioni analoghe, anche se con le dovute cautele per via del minor numero di campioni che è possibile raccogliere senza incorrere in un'eccessiva variabilità del manto stradale, possono essere ottenute analizzando il secondo esperimento, condotto lungo la strada primaria, anche se i valori assoluti dei parametri differiscono notevolmente, a causa dei diversi veicoli e delle diverse tipologie di asfalto. Rimangono analoghi i parametri relativi al best fit di $|A_y|^2$ e $P_{\mathrm{PE}}$ nelle rispettive strade. Questo significa che la metodologia per il calcolo della legge gamma nel caso della potenza dell'accelerazione verticale, è applicabile anche al caso della potenza del residuo di predizione.

## 5.4 Esperimenti su manto stradale eterogeneo

Al fine di studiare gli effetti che diverse tipologie di asfalto hanno sulla legge gamma, è stata implementata una nuova serie di esperimenti. In particolare, l'intero database di SmartRoadSense è stato suddiviso in quattro categorie, ciascuna delle quali contiene soltanto i campioni della $P_{\mathrm{PE}}$ raccolti lungo un particolare tipo di strada. Il raggruppamento è stato fatto seguendo la classificazione delle strade utilizzate dal database di OpenStreetMap [55] il quale suddivide le strade in quattro tipologie principali, come riportato in Tabella 5.3.

TABELLA 5.3: Tipologie di strade secondo la classificazione OpenStreetMap.

| Definizione internazionale | Corrispondenza italiana |
|---|---|
| *Motorways* | Autostrada |
| *Trunks road* | Superstrada |
| *Primary roads* | Strada Statale |
| *Secondary roads* | Strada Provinciale |

Per *motorways* si intendono tutte le autostrade, sia a pagamento, sia gratuite, compresi i raccordi autostradali, quali le tangenziali di Milano e il Grande Raccordo Anulare di Roma. All'estero sono l'equivalente delle Freeway statunitensi, le Autobahn tedesche, e simili. Normalmente sono classificate come A (Autostrada).

Per *trunks road* si fa riferimento a tipologie di strade poste a cavallo tra autostrade o tangenziali e le strade statali. Questa medesima classificazione viene assegnata anche ai tratti di svincolo di un'autostrada/tangenziale che conduce al centro della città. In questa categoria trovano la loro naturale collocazione le superstrade (per distinguerle dalle autostrade) come le strade extraurbane a una sola corsia per ogni senso di marcia, quando non presentano incroci a raso sul loro percorso e abbiano accessi e uscite mediati da corsie di accelerazione e decelerazione (per distinguerle dalle altre strade che potrebbero presentare incroci o rotatorie).

Per *primary roads* generalmente si fa riferimento a importanti arterie di comunicazione che collegano capoluoghi di provincia. Sono, pertanto, strade che rivestono una rilevanza nazionale o regionale, ma che non godono della classificazione di autostrada, tangenziale o superstrada. Le strade primarie collegano tra loro le città principali. Normalmente, vengono classificate come SS (Strada Statale) o SR (Strada Regionale), esistono eccezioni come in piccoli centri montani, dove la strada statale attraversa sì il paese, ma la strada primaria è una tangenziale di moderna costruzione che evita il centro abitato per decongestionare il traffico. Invece, in ambito urbano, le strade primarie costituiscono il primo anello della città, nelle quali vengono classificate come Viali.

Infine, per *secondary roads*, si fa riferimento a strade di importanza regionale e provinciale. Queste strade sono quelle che collegano i principali comuni di una regione. Sono normalmente classificate come SP (Strada Provinciale) anche se esistono eccezioni. In ambito urbano, normalmente sono classificate come vie importanti, anche a due corsie per ogni senso di marcia.

Come già descritto, l'autostrada è la strada con le maggiori prestazioni, progettata per gestire i volumi di traffico più elevati. L'autostrada è caratterizzata da un accesso controllato a due o più corsie per ogni senso di marcia. Solitamente è dotata di un asfalto drenante uniforme. Simile all'autostrada è la superstrada, la quale è sovente dotata di più corsie (tipicamente due per ogni senso di marcia), ma senza un rigoroso controllo di accesso e con uno standard di costruzione più basso. Scendendo nella classificazione delle strade per criteri e qualità costruttivi, incontriamo le strade primarie: principali vie di collegamento tra grandi città, normalmente con due corsie non separate da una barriera centrale. Infine, l'ultima ramificazione della rete viaria è la strada secondaria. Le strade secondarie sono strade caratterizzate tipicamente da una sola corsia per ogni senso di marcia.

Si noti che ciascuna classe di strada, corrispondente a una propria tipologia di manto stradale, contiene campioni raccolti da molti veicoli differenti, con all'interno diversi tipi di dispositivi mobili. Questa metodologia di analisi fa sì che le statistiche risultanti possano essere intese come un comportamento medio ottenuto in condizioni pseudo–uniformi.

### 5.4.1 Analisi su autostrade

La Figura 5.5 contiene oltre 476,000 campioni raccolti nelle autostrade italiane tramite il progetto SmartRoadSense fino a oggi (31 dicembre 2015). In particolare, tutti i dati estratti dal database, relativi alla categoria "autostrada", sono stati ordinati in base al loro valore della velocità. Come nel caso degli esperimenti precedenti, l'asse delle ascisse e delle ordinate è stato suddiviso in intervalli regolari, per colorare ogni singolo intervallo con una gradazione di grigio proporzionale al numero di campioni che vi cadono.

Per ciascuna classe di velocità è stato calcolata—e marcata in rosso—la potenza media del residuo di predizione $P_{\mathrm{PE}}$. Contestualmente, viene riportato il best fit ottenuto utilizzando la legge gamma (linea blu).

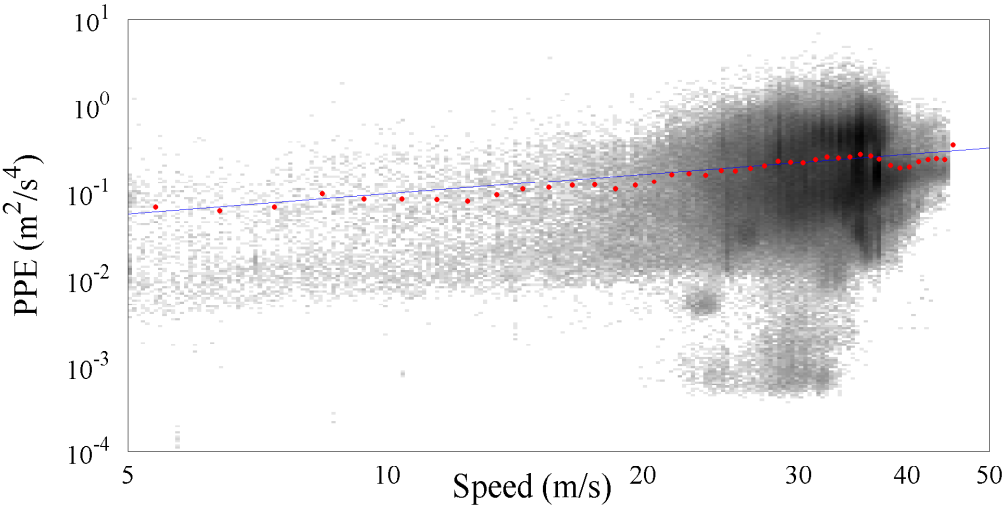



FIGURA 5.5: $P_{\mathrm{PE}}$ rispetto alla velocità e courve fit della legge gamma relativa alla classe "autostrada".

Nell'istogramma di Figura 5.6 sono messe in evidenza le ricorrenze di ogni singola classe di velocità. Le velocità maggiormente significative, ai fini dello studio e del calcolo del best fit di Figura 5.5 sono quelle comprese—approssimativamente—a partire da 20 m/s fino a 40 m/s.

I risultanti valori della legge gamma sono caratterizzati da un parametro $\hat{q}$ di circa 16.90 $10^{-3}$ e da un coefficiente $\gamma$ pari a circa 0.76, come riportato nella (5.1).

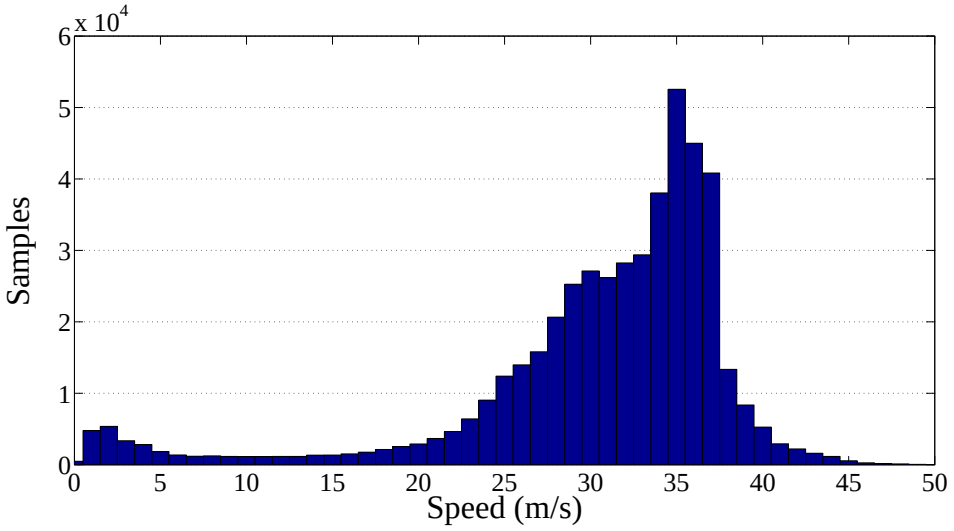



FIGURA 5.6: Istogramma della frequenza dei campioni raccolti per velocità relativi alla classe "autostrada".

$$P_{\text{motorway}}(v) = 16.90\ 10^{-3} v^{0.76}. \tag{5.1}$$

### 5.4.2 Analisi su superstrade

Nella Figura 5.7 e Figura 5.8 sono mostrati i risultati ottenuti dall'elaborazione della classe "superstrade", contenente oltre 170,000 campioni. Come previsto l'importanza dell'intervallo di velocità aumenta a causa della distribuzione più uniforme dei campioni lungo la gamma di velocità che partire da circa 10 m/s fino a 40 m/s.

Sono presenti—nell'istogramma di Figura 5.8—anche campioni a bassissima velocità (da 1 m/s, fino a circa 2 m/s), dovuti a casi di traffico congestionato. Come scelta metodologica, in tutte le analisi effettuale, tutti i campioni con velocità inferiore ai 5 m/s non sono stati analizzati.

Il valore così determinato dalla legge gamma restituisce coefficienti $\hat{q}$ e $\gamma$ rispettivamente pari a 21.08 $10^{-3}$ e 0.79, come riportato nella (5.2).

$$P_{\text{motorway}}(v) = 21.08\ 10^{-3} v^{0.79}. \tag{5.2}$$

### 5.4.3 Analisi su strade primarie

I risultati ottenuti elaborando la classe di "strade primarie", la quale contiene oltre 677.000 campioni, sono presentati in Figura 5.9 e Figura 5.10.

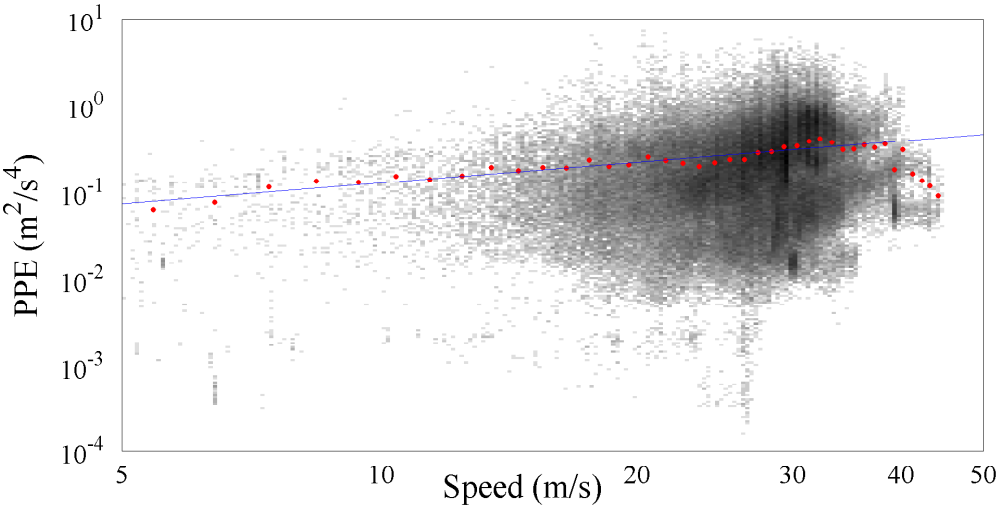



FIGURA 5.7: $P_{\mathrm{PE}}$ rispetto alla velocità e courve fit della legge gamma relativa alla classe "superstrada".

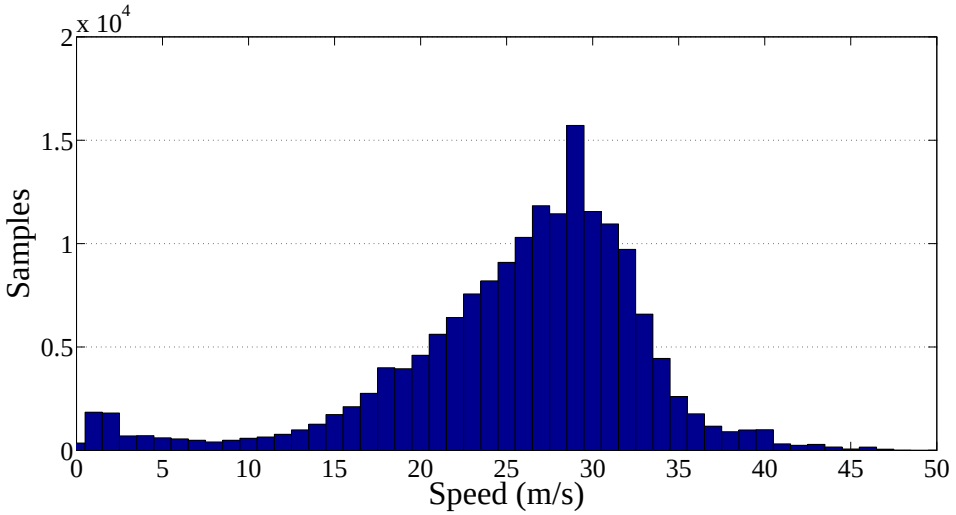



FIGURA 5.8: Istogramma della frequenza dei campioni raccolti per velocità relativi alla classe "superstrada".

La Figura 5.10 evidenzia un incremento di campioni nei bassi regimi di velocità, contestualmente a una riduzione dei campioni nelle velocità più elevate. Questo conduce ad avere velocità significative a partire da circa 5 m/s fino a circa 35 m/s.

Il valore così determinato dalla legge gamma restituisce coefficienti $\hat{q}$ e $\gamma$ rispettivamente pari a 20.21 $10^{-3}$ e 0.77, come riportato nella (5.3).

$$P_{\mathrm{motorway}}(v) = 20.21\ 10^{-3} v^{0.77}. \tag{5.3}$$

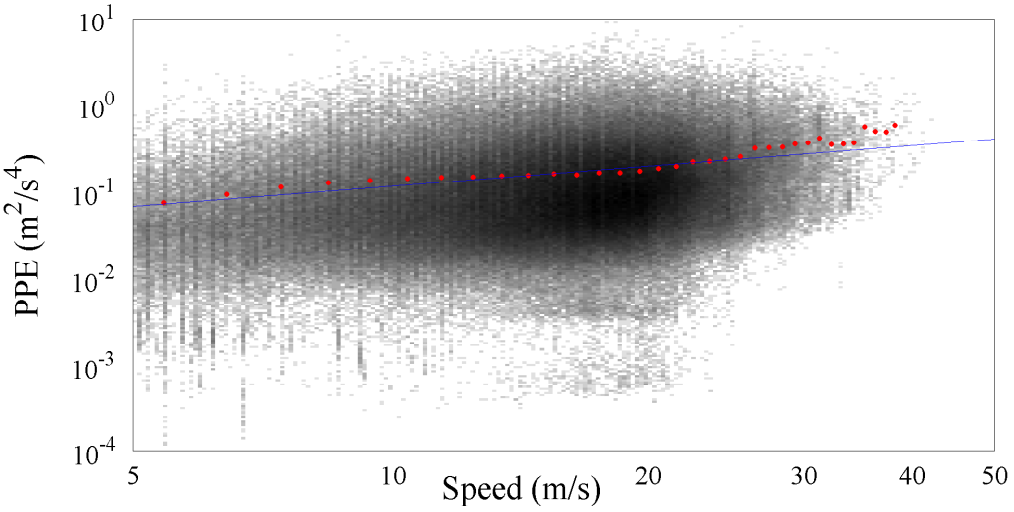



FIGURA 5.9: $P_{\mathrm{PE}}$ rispetto alla velocità e courve fit della legge gamma relativa alla classe "strada primaria".

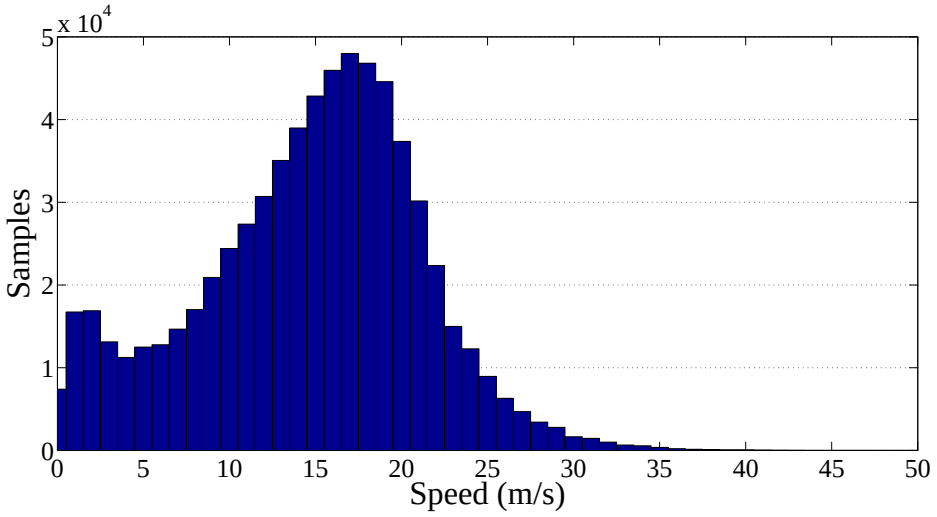



FIGURA 5.10: Istogramma della frequenza dei campioni raccolti per velocità relativi alla classe "strada primaria".

### 5.4.4 Analisi su strade secondarie

Infine è stata analizzata la classe "strade secondarie" e i grafici risultanti vengono riportati in Figura 5.11 e Figura 5.12. La quantità totale di campioni raccolti supera i 510.000 campioni, concentrati nelle velocità più basse. In particolare, la distribuzione dei campioni è relativamente ampia, mostrando una velocità di crociera compresa nell'intervallo 0 m/s fino a 25 m/s.

I valori risultanti dei coefficienti per la legge gamma sono $\hat{q} = 16.90\ 10^{-3}$ e $\gamma = 0.76$.

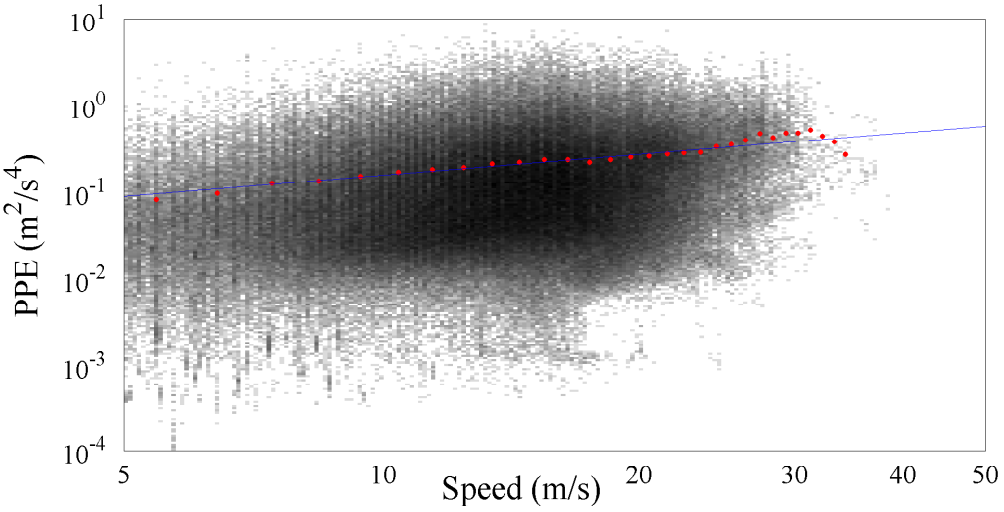



FIGURA 5.11: $P_{\mathrm{PE}}$ rispetto alla velocità e courve fit della legge gamma relativa alla classe "strada secondaria".

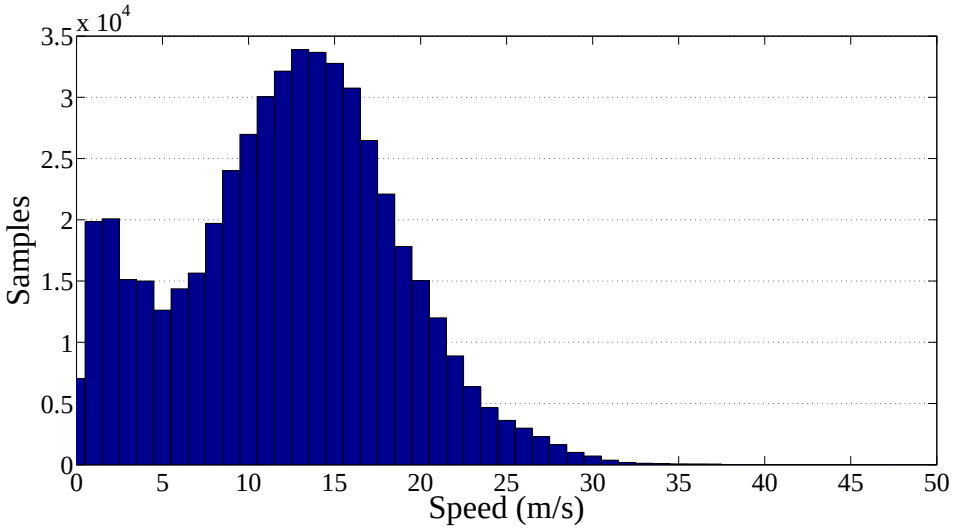



FIGURA 5.12: Istogramma della frequenza dei campioni raccolti per velocità relativi alla classe "strada secondaria".

## 5.5 Confronto dei risultati su manto stradale eterogeneo

La Figura 5.13 mostra tutti i dati analizzati nelle principali strade italiane. È ottenuta grazie alla sovrapposizione della Figura 5.5 (colorata con un gradiente grigio), della Figura 5.7 (colorata con un gradiente blu), della Figura 5.9 (colorata con un gradiente verde) e della Figura 5.11 (colorata con un gradiente rosso).

La Figura 5.14 mostra l'andamento di tutti i campioni studiati in queste quattro classi di strade, rispetto la loro velocità. A livello globale, sono stati usati 1,833,032 campioni utili, partendo da 5 m/s fino a circa 45 m/s. I campioni a velocità inferiori a 5 m/s,

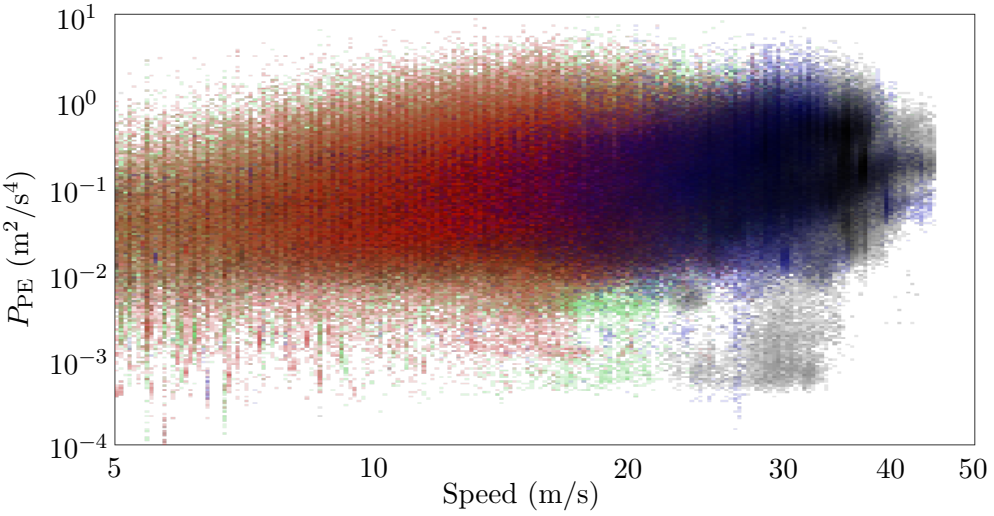



FIGURA 5.13: Fusione della Figura 5.5, Figura 5.7, Figura 5.9 e Figura 5.11

come in tutte le analisi—anche se presenti—sono stati eliminati poiché quasi tutti provenienti da traffico congestionato, code e altre condizioni di traffico che non si prestano a un'analisi del profilo stradale.

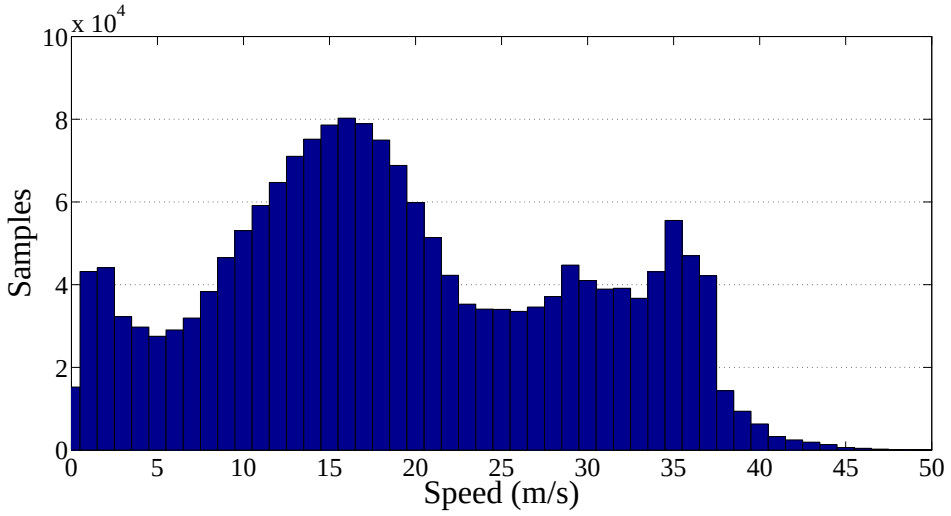



FIGURA 5.14: Andamento dei campioni rispetto dalla velocità per tutti i dati analizzati.

La legge gamma, ottenuta con tutti i dati analizzati assieme, non è applicabile al nostro caso di studio. I vari manti stradali sono differenti e questo provoca variazioni sia su $\hat{q}$, sia su $\gamma$. Inoltre, come mostrato nella Figura 5.6, Figura 5.8, Figura 5.10 e Figura 5.12 una siffatta metodologia implicherebbe lo studio di classi stradali con diverse velocità di crociera e differenti caratteristiche della pavimentazione stradale.

Per questa ragione, la soluzione migliore si ottiene cercando d'interpolare i dati di ogni velocità e secondo ciascuna tipologia di percorso, utilizzando una legge gamma per ogni tipo di percorso, i risultati ottenuti sono più affidabili.

Diversamente, se si volesse utilizzare un approccio che considera tutti i percorsi stradali assieme, si otterrebbe un parametro incongruente, avente coefficienti $\hat{q}$ e $\gamma$ rispettivamente pari a 49.62 $10^{-3}$ e 0.48, come riportato nella (5.4):

$$P_{\text{all}}(v) = 49.62\ 10^{-3} v^{0.48}. \tag{5.4}$$

Tale risultato è in disaccordo con quanto ottenuto nelle singole analisi precedenti. Tuttavia, è giustificabile poiché viene ottenuto analizzando differenti serie di dati eterogenei, partendo dalle autostrade (costruite per essere percorse alle più alte velocità e con asfalti drenanti) fino alle strade provinciali, dove i regimi di velocità sono notevolmente più bassi, come gli standard costruttivi dei manti stradali.

La Tabella 5.4 riporta tutti i coefficienti del curve fitting per l'espressione $P(v) = \hat{q}v^{\gamma}$ e i loro intervalli di confidenza al 95%. Tutti i fit hanno un andamento analogo (motorway, trunk road, primary road e secondary road), ma con valori di $\hat{q}$ e $\gamma$ crescenti al diminuire degli standard di costruzione del manto stradale.

Una dinamica simile non viene rilevata nel fit ottenuto con tutti i dati. Questo perché il fit globale tenta di approssimare ogni singolo fit. Per questa ragione questa soluzione deve essere rigettata. Nel database, gli indici di rugosità sono correlati con la relativa velocità e con i relativi tipi di strade. La pratica migliore è quella di correggere ogni campione con il suo valore di velocità, $\hat{p}$ e $\gamma$ relativi allo studio della tipologia di strada.

TABELLA 5.4: Interpolazione dei parametri $P(v) = \hat{q}v^{\gamma}$ per manti stradali eterogenei.

| Coefficienti (col 95% dell'intervallo di confidenza) | | |
|---|---|---|
| Strada | $\hat{q}$ | $\gamma$ |
| Autostrada | 16.89 $10^{-3}$ (15.78 $10^{-3}$, 18.01 $10^{-3}$) | 0.76 (0.74, 0.78) |
| Superstrada | 20.97 $10^{-3}$ (18.54 $10^{-3}$, 23.62 $10^{-3}$) | 0.79 (0.75, 0.83) |
| Strada Primaria | 20.20 $10^{-3}$ (19.32 $10^{-3}$, 21.10 $10^{-3}$) | 0.77 (0.76, 0.79) |
| Strada Secondaria | 24.82 $10^{-3}$ (23.64 $10^{-3}$, 26.03 $10^{-3}$) | 0.81 (0.79, 0.82) |

La Figura 5.15 riporta come le quattro diverse interpolazioni, per le quattro classi di manto stradale abbiano una risposta crescente in ragione della velocità e del deterioramento del manto stradale in esame. Si noti—infatti—che al diminuire degli standard costruttivi della pavimentazione stradale (dalle autostrade fino alle strade secondarie) si ha un progressivo aumento sia del parametro $\hat{q}$, sia del parametro $\gamma$. Questo significa che ogni tipologia di strada deve essere trattata come caso a sé stante. Quest'approccio è possibile. Infatti, il database di SmartRoadSense memorizza, oltre alle coordinate GPS, anche il codice identificativo della strada percorsa secondo la classificazione di

OpenStreetMap e—di conseguenza—la tipologia di manto stradale attraversato nel dato istante.

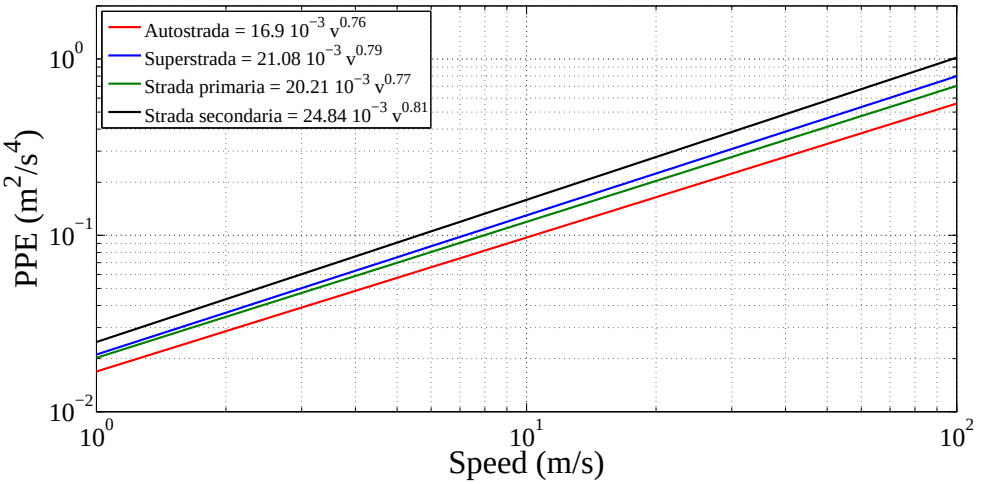



FIGURA 5.15: Confronto dei diversi curve fitting.

La pratica migliore è quella di correggere l'indice di rugosità di ogni singolo campione facendo riferimento al suo valore di velocità $v$, e ai suoi parametri $\hat{p}$ e $\gamma$ derivati dallo studio del tipo di stradale in cui è localizzato. Questo è possibile, poiché il database SmartRoadSense—nel momento in qui acquisisce i parametri georeferenziali—li colloca nella strada più vicina e, di questa, memorizza il suo codice identificativo OpenStreetMap, grazie al quale è possibile determinare la tipologia di asfalto.

## 5.6 Modello indipendente dalla velocità

La grande mole di dati presenti nel database permette di studiare i dati raccolti con rigore. Gli indici di rugosità presenti, a seconda della tipologia di manto stradale, possono essere corretti con la (5.5).

$$\widetilde{RI} = \frac{RI(v)}{\hat{q}v^{\gamma}}, \tag{5.5}$$

dove $\widetilde{RI}$ è un nuovo indice di rugosità, indipendente dalla velocità del veicolo e dal tipo di percorso, mentre $\hat{q}$ e $\gamma$ sono i parametri della legge gamma relativi al percorso in cui si trova il veicolo.

Tuttavia, questa soluzione—formalmente corretta—non risponde ad alcuni importanti requisiti:

1. il nuovo parametro $\widetilde{RI}$ ha un'equazione dimensionale differente dal precedente indice di rugosità $RI$, pertanto eseguire raffronti numerici è operazione tutt'altro che banale;

2. inoltre, le nuove mappe così generate, presentano valori che si discostano molto dai precedenti, impedendo un rapido confronto visivo costringendo all'adozione di una nuova mappa colorimetrica.

Per ovviare a questi due inconvenienti si è ritenuto opportuno definire un coefficiente normalizzato di questo tipo:

$$\widetilde{RI} = \bar{q}\bar{v}^{\bar{\gamma}} \frac{RI}{\hat{q}v^{\gamma}}, \tag{5.6}$$

dove $\bar{q}$, $\bar{v}$ e $\bar{\gamma}$ sono i parametri medi—rispettivamente—di $\hat{q}$, $v$ e $\gamma$. Il loro prodotto è un coefficiente la cui finalità è quella di restituire un indice di rugosità normalizzato e, al medesimo tempo, di generare un nuovo indice di rugosità $\widetilde{RI}$ con le medesime dimensioni della potenza di un'accelerazione ($\mathrm{m}^2/\mathrm{s}^4$) del precedente indice di rugosità $RI$.

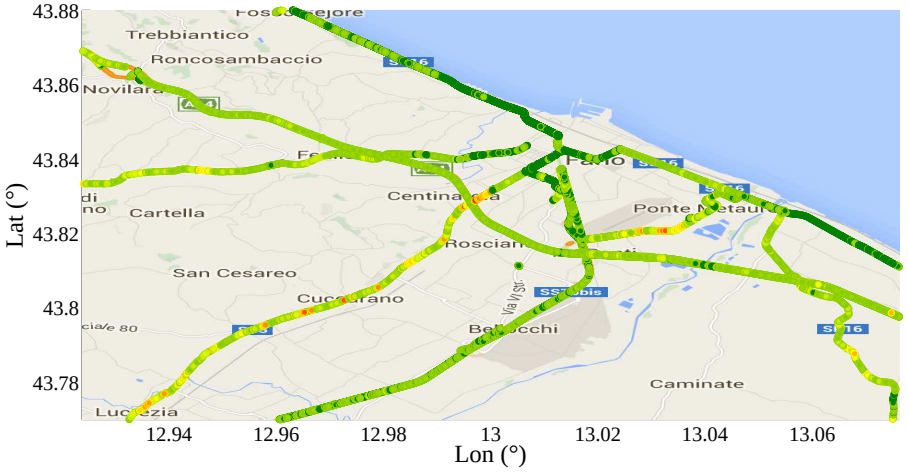
Nelle cartine geografiche è mostrata una sezione dell'Italia nel Centro–Est. Prima con il tipico indice di rugosità utilizzato nel progetto SmartRoadSense originale (come mostrato in Figura 5.16(a)), e—successivamente—come appare dopo l'applicazione della (5.6) (come visibile in Figura 5.16(b)).

Come riferimento è utile considerare la Figura 5.17, dove vengono evidenziate le diverse tipologie di manto stradale, utilizzando la stessa colorazione della Figura 5.15.

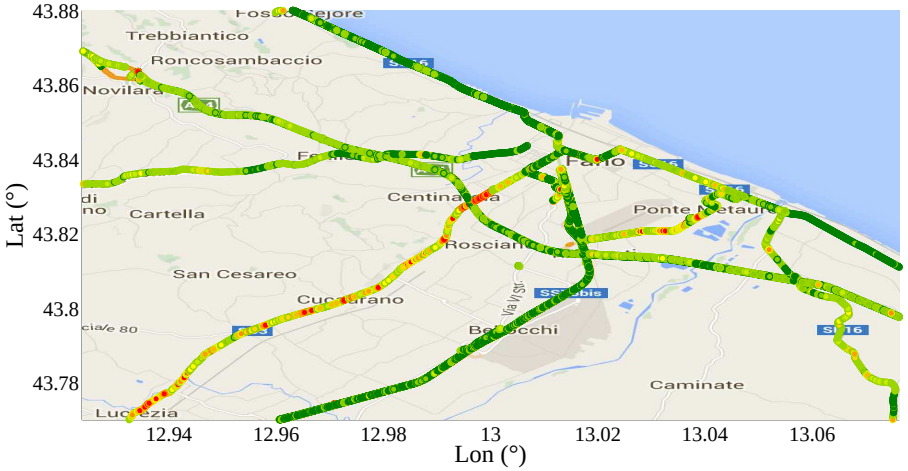
Utilizzando nell'applicazione la formula (5.6), le anomalie nelle strade statali e provinciali aumentano, mentre diminuiscono—come era lecito attendersi—le anomalie nelle autostrade e superstrade.

In questo capitolo si è—pertanto—sviluppato un nuovo modello per il calcolo dell'indice di rugosità del manto stradale. Ricorrendo all'utilizzo della (5.6) si ottiene un nuovo indice di rugosità $\widetilde{RI}$.

Questo nuovo indice di rugosità $\widetilde{RI}$ risulta indipendente sia dalla velocità del veicolo, sia dalla classe di manto stradale.




(a) Proiezione dei dati acquisiti senza correzione.




(b) Proiezione dei medesimi dati corretti attraverso la (5.6).

FIGURA 5.16: Confronto delle mappa cartografiche prima e dopo la correzione proposta per velocità e manto stradale.

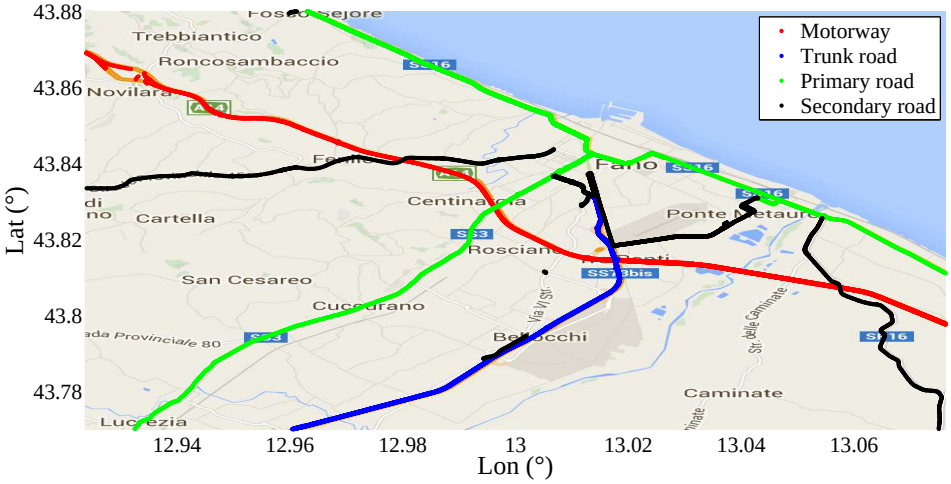



FIGURA 5.17: Tipologia di manti stradali secondo la classificazione OpenStreetMap.

# Capitolo 6

# Conclusioni e scenari aperti

## 6.1 Risultati ottenuti

In questa tesi è stato sviluppato e validato con successo un algoritmo per l'estrazione di un indice rugosità del manto stradale. Tale algoritmo prevede l'utilizzo di un originale approccio predittivo il quale è in grado di filtrare, dai dati acquisiti dall'accelerometro incorporato nel dispositivo mobile, le componenti delle accelerazioni non imputabili a imperfezioni del manto stradale.

In una seconda fase, questo stesso approccio è stato ulteriormente perfezionato, studiando le caratteristiche dell'influenza della velocità del veicolo, sull'accelerazione verticale rilevata da un dispositivo mobile ancorato nell'abitacolo. Questo studio ha fornito al progetto SmartRoadSense un indice di rugosità indipendente sia dalla velocità del veicolo, sia dalla tipologia del manto stradale.

I risultati teorici, così come descritti nel capitolo 2, indicano—infatti—che la potenza rilevata dell'accelerazione verticale e l'indice di rugosità di SmartRoadSense, dipendono dalla velocità del veicolo, secondo una legge gamma. Inoltre, il parametro $\gamma$ appartiene ad un intervallo noto e limitato. Per l'esattezza, si ha: $\gamma \in [0, 2]$ e $\gamma$, valore che tende a diminuire per valori crescenti della velocità del veicolo stesso.

I risultati sperimentali ottenuti dal database SmartRoadSense, ottenuti grazie alle enormi quantità di dati raccolti nei primi mesi di attività del progetto SmartRoadSense, confermano i risultati teorici. In particolare, le quattro principali tipologie di strade analizzate (autostrada, superstrada, strada statale e strada provinciale), restituiscono una legge gamma, come riportato in Tabella 5.4.

Si noti che l'autostrada, la strada con il miglior asfalto—tipicamente un asfalto drenante—presenta il minor coefficiente $\gamma$, questo perché è progettata per essere percorsa con le

più alte velocità e congestioni di traffico. Viceversa, le strade secondarie—attraversate con le velocità più basse—evidenziano il più alto coefficiente $\gamma$. La superstrada e le strade primarie, con velocità che spesso si sovrappongono, restituiscono un parametro $\gamma$ molto simile.

Le evidenze sperimentali sono in accordo con gli indici di rugosità raccolti corso del progetto SmartRoadSense.

## 6.2 Scenari aperti

Il coinvolgimento diretto degli utenti nel monitoraggio dello stato delle strade aumenta la consapevolezza nei confronti del valore del bene comune che, più di ogni altro, condiziona lo sviluppo del nostro paese. Dal punto di vista economico le strade italiane costituiscono un patrimonio dal valore altissimo, stimato in oltre 1,000 miliardi di Euro per la sola sovrastruttura [56]. Il ritardo negli interventi di manutenzione disperde questo immenso bene comune, creando—in prima istanza—problemi di sicurezza stradale, e comportando un aumento drastico dei costi operativi e dei consumi degli autoveicoli.

I dati aperti, prodotti dal progetto SmartRoadSense, possono essere utilizzati liberamente per realizzare ulteriori applicazioni e servizi, come la pianificazione degli interventi di manutenzione, la stima dei tempi di usura, cartografie temporali dello stato del manto stradale, fino a *plugin* di navigatori satellitari.

Il modello matematico sin qui elaborato risulta finalmente indipendente dalla tipologia di asfalto e dalla velocità del veicolo. Naturalmente la complessità del sistema è ancora fonte di studio. In primo luogo i veicoli non sono tutti uguali, a iniziare dalle loro sospensioni e masse. La legge gamma estratta potrà e dovrà essere rivista, mano a mano che si ottengono nuovi dati: infatti il database di SmartRoadSense è in continua espansione e ogni dato acquisito consolida le misurazioni già effettuate.

Questo non è uno svantaggio, ma un punto di forza di questo lavoro. Infatti i parametri della legge gamma sono stati ottenuti dallo studio dei dati presenti nel database. Ripetere il medesimo studio a intervalli regolari può produrre risultati via, via sempre più affidabili. Risultati che includono al loro interno il rifacimento di nuove strade come strade che si sono deteriorate nel corso degli anni e—pertanto—forniscono ora parametri differenti.

Così procedendo, i parametri $\hat{q}$ e $\gamma$, per ogni tipologia di asfalto, tenderanno a convergere ad un valore via, via, sempre più aderente alla realtà.

Infine, dove vi è una vasta copertura delle strade, non soltanto singoli passaggi, è auspicabile l'introduzione di modelli dinamici i quali consentano di studiare l'evoluzione del manto stradale, di determinare dove sono stati operati rifacimenti, come dove si sono verificati cedimenti strutturali. Queste informazioni—se automatizzate—si rivelerebbero un valore aggiunto per la Pubblica Amministrazione che già utilizza SmartRoadSense.

# Appendice A

# Codice sorgente SmartRoadSense

Nella presente appendice viene riportato il codice sorgente dell'applicazione SmartRoadSense. Le simulazioni di laboratorio, anticipate nei capitoli precedenti, sono state realizzate utilizzando MATLAB©.

Il codice relativo all'applicativo Android©—invece—è stato realizzato "traducendo" il codice MATLAB© in linguaggio Java.

## A.1 Codice MATLAB©

Di seguito viene riportato il codice MATLAB© utilizzato negli studi preliminari al fine di ottenere il codice Java utilizzato per nell'applicazione Android©. Il codice riportato qui di seguito è stato scritto in due versioni: una con output grafici e una seconda—quella che viene qui presentata—senza output grafici.

Questa viene ritenuta più interessante poiché si concentra sull'algoritmo e sulle sue performance.

### A.1.1 `srs_test_no_plot.m`

```
1 %SRS_TEST_NO_PLOT
2 %   SRS_TEST_NO_PLOT was developed to test traces of the GPS and
3 %   accelerometer for SmartRoadSense Project and estimates them
4 %   performance.
5 %
6 %   The MAT file in input MUST have a vector with ten (10) columns in
7 %   this order:
```

```
8  %
9  %   1.  Timestamp (ms)
10 %   2.  Accelerations x (m/s^2) Crosswise to the driving direction
11 %   3.  Accelerations y (m/s^2) Vertical acceleration
12 %   4.  Accelerations z (m/s^2) In line with the driving direction
13 %   5.  GPS latitude
14 %   6.  GPS longitude
15 %   7.  GPS quota (m)
16 %   8.  Velocity (m/s)
17 %   9.  Flag 1
18 %   10. Flag 2
19 %
20 %   Notes: The MAT file must be in the same folder of this program.
21 %
22 %   TEST use the modelling AR(6)[1] in a windowed approach. By its power
23 %   prediction error and final prediction error (based on power prediction
24 %   error, with Ljung formula[2]) compute an extimation of roughness
25 %   sourface.
26 %   Bumps are removed by 10th order Moving Average
27 %
28 %
29 %   References:
30 %
31 %   [1] Marple, Jr., S.L., Digital Spectral Analysis with Applications,
32 %   Prentice Hall, Englewood Cliffs, 1987, Chapter 8.
33 %   [2] Sections 7.4 and 16.4 in Ljung (1999).
34 %
35 %   See also:
36 %     EXTRACT_GPS_DATA (required)
37 %     PPE (required)
38 %     AR_MODEL (required)
39 %     MOVING_AVERAGE (required)
40 %
41 %   Author(s): G. Alessandroni, 01-07-14
42 %   Copyright 2014, University of Urbino
43
44 % Removes all variables, globals, functions and MEX links
45 % Closes all the open figure windows
46 % Clears the command window and homes the cursor
47 clear all;
48 close all;
49 clc;
50
51 filename = 'SRS-01';
52
53 % Load file .mat
54 M = load(filename);
55
```

```
56 % Extracts Time (in ms)
57 T = M.M(:, 1);
58
59 % Extracts matrix accelerations [Ax Ay Az]
60 A = [M.M(:, 2), M.M(:, 3), M.M(:, 4)];
61
62 % Extracts the individual accelerations
63 Ax = A(:, 1);
64 Ay = A(:, 2);
65 Az = A(:, 3);
66
67 % Extracts GPS data [lat lon quote]
68 GPS = [M.M(:,5), M.M(:,6), M.M(:,7)];
69
70 % Extracts Velocity data
71 V = M.M(:,8);
72
73 % Discard excess GPS data
74 [~, lat, lon , ~, index, Velocity] = extract_gps_data(T, GPS, V);
75
76 % Discard data without GPS trace
77 Ax = Ax(index(1):end);
78 Ay = Ay(index(1):end);
79 Az = Az(index(1):end);
80 t_sec = T(index(1):end)/1000;
81
82 % If Velocity = 0 then Velocity = 1
83 Velocity(Velocity < 1) = 1;
84
85 % Detrend mean value of accelerations
86 Ax = Ax - mean(Ax);
87 Ay = Ay - mean(Ay);
88 Az = Az - mean(Az);
89
90 % Size of windows
91 s = 100;
92
93 % Order of AR(p)
94 order = 6;
95
96 % Overlap of windows
97 overlap = 25;
98
99 % Hamming window
100 Hs = s + overlap * 2 - 1;
101 Ch = 2 * pi / Hs;
102 H = (0.54 - 0.46 * cos(Ch * (0:Hs)))';
103
```

```
104 % Compute Power Prediction Error for route
105 ppe_x = ppe(Ax, s, order, overlap, H);
106 ppe_y = ppe(Ay, s, order, overlap, H);
107 ppe_z = ppe(Az, s, order, overlap, H);
108
109 % Discard extras data
110 if size(lat, 1) ~= size(ppe_x, 1)
111     if size(lat, 1) > size(ppe_x, 1)
112         lat = lat(1:size(ppe_x, 1));
113         lon = lon(1:size(ppe_x, 1));
114         Velocity = Velocity(1:size(ppe_x, 1));
115     else
116         ppe_x = ppe_x(1:size(lat, 1));
117         ppe_y = ppe_y(1:size(lat, 1));
118         ppe_z = ppe_z(1:size(lat, 1));
119     end
120 end
121
122 % Compute Mean Power Prediction Error
123 ppe_m = (ppe_x + ppe_y + ppe_z)/3;
124
125 % Moving Average Order
126 order = 10;
127
128 % Compute Moving Average of Power Prediction Error
129 ppe_ma = moving_average(ppe_m, order);
130
131 % Output data
132 out.filename = filename;
133 out.window_size = s;
134 out.t_sec = t_sec;
135 out.ppe_x = ppe_x;
136 out.ppe_y = ppe_y;
137 out.ppe_z = ppe_z;
138 out.ppe_m = ppe_m;
139 out.ppe_ma = ppe_ma;
140 out.lat = lat;
141 out.lon = lon;
142 out.velocity = Velocity;
143
144 % [EOF] - SRS_TEST_NO_PLOT.M
```

### A.1.2 extract_gps_data.m

```
1 function [Time, lat, lon, quo, index, Velocity] = ...
2 extract_gps_data(T, GPS_data, V)
```

```
3  %EXTRACT_GPS_DATA
4  %   This function was developed to extract significative GPS data from a
5  %   matrix with ridondance.
6  %   Sometimes the GPS data are provided together at accelerometer ...
       data. But
7  %   the accelerometer has a sample frequency of 100 Hz and the GPS of ...
       1 Hz.
8  %   So is important reject all equal data.
9  %   Last but not least, this function rejects data without GPS.
10 %
11 %   Note: The vector Time, V and the Matrix (3 x N) GPS_data must have the
12 %   same number of rows.
13 %
14 %   Author(s): G. Alessandroni, 04-19-13
15 %   Copyright 2013, University of Urbino
16
17
18
19 % Rows of Time vector
20 [R_T, ~] = size(T);
21
22 % Discard empty GPS data
23 i = 1;
24 while GPS_data(i, 1) == 0
25     i = i + 1;
26 end
27
28 % Preallocating data for speed
29 Time = zeros(floor((R_T - i) / 100), 1);
30 lat = Time;
31 lon = Time;
32 quo = Time;
33 index = Time;
34 Velocity = Time;
35
36 j = 1;
37
38 % Extract data until end
39 while i + 1 < R_T
40
41     % Data extraction
42     Time(j) = T(i);
43     lat(j) = GPS_data(i, 1);
44     lon(j) = GPS_data(i, 2);
45     quo(j) = GPS_data(i, 3);
46     Velocity(j) = V(i);
47     index(j) = i;
48
```

```
49      % Find another data different from the last
50              while GPS_data(i, 1) == lat(j) && ...
51              GPS_data(i, 2) == lon(j) && ...
52              GPS_data(i, 3) == quo(j) && ...
53              V(i) == Velocity(j) && ...
54              i < R_T
55          i = i + 1;
56      end
57      j = j + 1;
58  end
59
60  end
61
62  % [OEF] - EXTRACT_GPS_DATA.M
```

### A.1.3 ppe.m

```
1   function y = ppe(x, s, order, overlap, window)
2   %PPE
3   %   PPE function compute Power Prediction Error of x data with
4   %   Autoregressive Model.
5   %   PPE use a serial of windows of size and overlap specify.
6   %
7   %   Y = PPE(X, S, ORDER, OVERLAP, WINDOW)
8   %
9   %   X         : Input data
10  %   S         : Window size
11  %   ORDER     : Order of AR model
12  %   OVERLAP   : overlap before and after window
13  %   WINDOW    : Hamming or other window
14  %
15  %   Y         : Power Prediction Error of X data
16  %
17  %   See also:
18  %     AR_MODEL (required)
19  %
20  %
21  %   Author(s): G. Alessandroni, 08-05-13
22  %   Copyright 2013, University of Urbino
23
24
25  % Resize data with overlap before and after data
26  b = buffer(x, s + overlap * 2, overlap * 2);
27
28  c = size(b, 2);
29
```

```
30 % Preallocation data for speed
31 e = zeros(s, 1);
32 y = zeros(c, 1);
33
34 for i = 1:c
35     % Stack data for current window
36     temp = b(:, i);
37     % Detrend data
38     temp = temp - sum(temp)/size(temp, 1);
39     % Apply window
40     temp = temp .* window;
41
42     % Estimates AR(order) model for current window
43     l = ar_model(temp, order);
44
45     % Compute prediction errors
46     for j = 1:s
47         k = overlap + j;
48         e(j) = l*temp(k : -1 : k - order);
49     end
50
51     % Compute window power prediction error
52     y(i) = sum(e.^2)/s;
53 end
54
55 end
56
57 % [EOF] - PPE.M
```

### A.1.4 ar_model.m

```
1  function lambda = ar_model(x, p)
2  %   AR_MODEL compute AR-models parameters of input signal using ...
       Yule-Walker
3  %   method.
4  %
5  %   LAMBDA = AR_MODEL(Y, N, T)
6  %     estimates an N:th order autoregressive polynomial model (AR) for ...
       time
7  %     series Y:
8  %        y(n) + l_1 * y(n-1) + l_2 * y(n-2) + ... +l_N * y(n-N) = e(t)
9  %
10 %     Inputs:
11 %      X: The time series to be modeled, a column vector of values. The
12 %      data must be uniformly sampled.
13 %      N: The order of the AR model (positive integer)
```

```
%       T: Toeplitz matrix index
%     Output:
%       LAMBDA: AR model delivered as an array where are
%       [1 l_1 l_2 l_3 ... l_N].
%     The model is estimated using "Yule-Walker" approach with no
%     windowing.
%
%   Author(s): G. Alessandroni, 01-08-13
%   Copyright 2013, University of Urbino

% Compute autocorrelation value without negative size
R = autocorr2(x, p);

% Remove negative size
R = R(floor(max(size(R))/2)+1:end);

% Compute AR parameters lambda = RL^-1 * RR
% Lambda parameters are computed via Yule-Walker method, without inverse
% matix.
lambda = levinson_durbin(R);

end

% [EOF] - AR_MODEL.M
```

### A.1.5 autocorr2.m

```
function c = autocorr2(x, maxlag)
%AUTOCORR2 Computate the Auto-correlation Sequence of x over the range
%   of lags:  -MAXLAG to MAXLAG, i.e., 2*MAXLAG+1 lags.
%
%   See also XCORR, XCOV, CORRCOEF, CONV, CCONV, COV and XCORR2.
%
%   Author(s): G. Alessandroni, 10-01-14
%   Copyright 2014, University of Urbino
%
%   References:
%     S.K. Mitra, "Digital Signal Processing. A Computer-Based Approach"
%     4nd Ed. Mc-Graw-Hill, 2011
%
%   Computational cost, for maxlag = 6, 921 products


% Preallocation data for speed
c = zeros(1, maxlag + 1);

```

```
20 % Used to shift datas
21 z = [];
22
23 for k = 1:maxlag + 1
24     % Compute the k-th element of autocorrelation
25     c(k) = sum([z; x] .* [x; z]);
26     z = zeros(k, 1);
27 end
28
29 %Build the negativ size of vector
30 c = [c(end:-1:2) c];
31
32 %Force vector to column
33 c = c(:);
34
35 end
36
37 % [EOF] - AUTOCORR2.M
```

### A.1.6 levinson_durbin.m

```
1  function [A, E, ref] = levinson_durbin(R, N)
2  %LEVINSON_DURBIN  Levinson-Durbin Recursion.
3  %   A = LEVINSON_DURBIN(R, N) solves the Hermitian Toeplitz system of
4  %   equations
5  %
6  %       [  R(1)   R(2)  ...  R(N)  ] [  A(2)  ]  =   [  R(2)  ]
7  %       [  R(2)   R(1)  ... R(N-1) ] [  A(3)  ]  =   [  R(3)  ]
8  %       [   .       .         .    ] [   .    ]  = - [   .    ]
9  %       [ R(N-1) R(N-2) ...  R(2)  ] [  A(N)  ]  =   [  R(N)  ]
10 %       [  R(N)  R(N-1) ...  R(1)  ] [ A(N+1) ]  =   [ R(N+1) ]
11 %
12 %   (also known as the Yule-Walker AR equations) using the Levinson-
13 %   Durbin recursion.  Input R is a vector of autocorrelation coefficients
14 %   with lag 0 as the first element.
15 %
16 %   N is the order of the recursion; if omitted, N = LENGTH(R)-1.
17 %   A will be a row vector of length N+1, with A(1) = 1.0.
18 %
19 %   [A, E] = LEVINSON_DURBIN(...) returns the prediction error, E, of ...
   order
20 %   N.
21 %
22 %   [A, E, K] = LEVINSON_DURBIN(...) returns the reflection ...
   coefficients K
23 %   as a column vector of length N. Since K is computed internally while
```

```
24 %   computing the A coefficients, then returning K simultaneously
25 %   is more efficient than converting A to K afterwards via TF2LATC.
26 %
27 %
28 %   Author(s): G. Alessandroni, 10-01-14
29 %   Copyright 2014, University of Urbino
30
31
32 % Force vector in coloumn
33 R = R(:);
34
35 if nargin == 1
36     N = length(R) - 1;
37 end
38
39 % Preallocation data for speed
40 ref = zeros (1, N);
41
42 % First step
43 k = R(2)/R(1);
44 A = k;
45 E = (1 - k^2) * R(1);
46 ref(1) = -k;
47
48 % Other recursive steps
49 for i = 2:N
50     k = (R(i+1) - A * R(2:i))/E;
51     % Compute the i-th solution
52     A = [k, A - k * A(i-1:-1:1)];
53     % Update the prediction error
54     E = (1 - k^2) * E;
55     % Compute the reflection coefficient
56     ref(i) = -k;
57 end
58
59 % Adds 1 to first column, changes sign and split order of vector
60 A = [1, -A(N:-1:1)];
61
62 end
63
64 % [EOF] - LEVINSON_DURBIN.M
```

## A.2 Codice Java

Di seguito viene riportato il codice Java utilizzata per l'applicazione SmartRoadSense. I primi tre codici sono di definizione del progetto, pertanto ospitati nella directory `root`.

I codici seguenti, sono relativi all'algoritmo vero e proprio. Pertanto sono ospitati in cartelle differenti, specificate nella prima riga durante il comando `package`.

### A.2.1 Manifest

```
Manifest-Version: 1.0
```

### A.2.2 .project

```
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
    <name>SrsAlgorithm</name>
    <comment></comment>
    <projects>
    </projects>
    <buildSpec>
        <buildCommand>
            <name>org.eclipse.jdt.core.javabuilder</name>
            <arguments></arguments>
        </buildCommand>
    </buildSpec>
    <natures>
    <nature>org.eclipse.jdt.core.javanature</nature>
    </natures>
</projectDescription>
```

### A.2.3 .classpath

```
<?xml version="1.0" encoding="UTF-8"?>
<classpath>
    <classpathentry kind="src" path="src"/>
    <classpathentry kind="con" ...
        path="org.eclipse.jdt.launching.JRE_CONTAINER"/>
    <classpathentry kind="output" path="bin"/>
</classpath>
```

### A.2.4 ComputationListener.java

```
package it.uniurb.smartroadsense.algorithm;

```

```
3  /**
4  * Allows external classes to listen for computation ...
       results.
5  *
6  */
7  public interface ComputationListener {
8
9  /**
10 * Signals that a new result has been generated.
11 */
12 public void resultGenerated(Result result);
13
14 }
```

### A.2.5 Buffer.java

```
1  package it.uniurb.smartroadsense.algorithm;
2
3  class Buffer {
4
5      public Buffer(int size){
6          if(size <= 0)
7          throw new IllegalArgumentException("Negative ...
               size");
8
9          _size = size;
10
11         _timestamps = new long[size];
12         _accX = new double[size];
13         _accY = new double[size];
14         _accZ = new double[size];
15         _longitude = new double[size];
16         _latitude = new double[size];
17         _speed = new float[size];
18         _bearing = new float[size];
19         _accuracy = new int[size];
20 }
21
```

```
22 private final int _size;
23
24 private int _fillIndex = 0;
25
26 public int getSize(){
27     return _size;
28 }
29
30 public int getIndex(){
31     return _fillIndex;
32 }
33
34 public boolean isFull(){
35     return (_fillIndex >= _size);
36 }
37
38 long[] _timestamps;
39 double[] _accX, _accY, _accZ;
40 double[] _longitude, _latitude;
41 float[] _speed;
42 float[] _bearing;
43 int[] _accuracy;
44
45 /**
46 * Clears the buffer.
47 */
48 public void clear(){
49     _prefillSize = 0;
50     _fillIndex = 0;
51 }
52
53 private final long TIMESTAMP_TOLERANCE_MS = 1500;
54
55 public void record(DataEntry entry){
56     if(isFull()){
57         throw new IllegalStateException();
58     }
59
60     long entryTimestamp = entry._timestamp;
```

```
61
62      if(_fillIndex > 0 && _timestamps[_fillIndex - 1] > ...
            entryTimestamp){
63          if(_timestamps[_fillIndex - 1] - ...
                entryTimestamp < TIMESTAMP_TOLERANCE_MS){
64              Engine.logError("Non monotonically ...
                    increasing timestamp compensated");
65              entryTimestamp = _timestamps[_fillIndex - 1];
66          }
67          else{
68              throw new ...
                    IllegalArgumentException("Timestamps ...
                    must be monotonically increasing");
69          }
70      }
71
72      _timestamps[_fillIndex] = entryTimestamp;
73      _accX[_fillIndex] = entry._accX;
74      _accY[_fillIndex] = entry._accY;
75      _accZ[_fillIndex] = entry._accZ;
76      _longitude[_fillIndex] = entry._longitude;
77      _latitude[_fillIndex] = entry._latitude;
78      _speed[_fillIndex] = entry._speed;
79      _bearing[_fillIndex] = entry._bearing;
80      _accuracy[_fillIndex] = entry._accuracy;
81
82      ++_fillIndex;
83  }
84
85  public void empty(){
86      _fillIndex = 0;
87  }
88
89  private int _prefillSize = 0;
90
91  /**
92  * Prefills a buffer with 0 values.
93  * @param prefillSize Total size of the data window to ...
       be prefilled.
```

```
94  */
95  public void prefill(int prefillSize) {
96      if(_fillIndex != 0)
97          throw new IllegalStateException("Buffer must ...
                be empty to be prefilled");
98      if(prefillSize < 0)
99          throw new IllegalArgumentException("Cannot ...
                prefill negative size");
100     if(prefillSize >= _size)
101         throw new IllegalArgumentException("Prefill ...
                size cannot match or exceed buffer size, ...
                leaves no space for data");
102
103     for(int i = 0; i < prefillSize; ++i){
104         _timestamps[i] = 0;
105         _accX[i] = _accY[i] = _accZ[i] = 0.0;
106         _longitude[i] = _latitude[i] = 0.0;
107         _speed[i] = 0;
108         _bearing[i] = 0.0f;
109         _accuracy[i] = 0;
110     }
111
112     _fillIndex = prefillSize;
113     _prefillSize = prefillSize;
114 }
115
116 void dump(String string) {
117     PrintHelpers.dumpData(string, new String[] {
118         "accX", "accY", "accZ", "long", "lang"
119     }, new double[][]{
120         _accX, _accY, _accZ, _longitude, _latitude
121     });
122 }
123
124 private int computeMiddleIndex(){
125     int filledSize = _size - _prefillSize;
126     return _prefillSize + (filledSize / 2);
127 }
128
```

```
129 public double getMeanLongitude(){
130     if(_fillIndex <= _prefillSize)
131         throw new IllegalStateException();
132 
133     double acc = 0.0;
134     for(int i = _prefillSize; i < _fillIndex; ++i){
135         acc += _longitude[i];
136     }
137 
138     return acc / (_fillIndex - _prefillSize);
139 }
140 
141 public double getMiddleLongitude(){
142     int middle = computeMiddleIndex();
143     if(_fillIndex <= middle)
144         throw new IllegalStateException();
145 
146     return _longitude[middle];
147 }
148 
149 public double getMeanLatitude(){
150     if(_fillIndex <= _prefillSize)
151         throw new IllegalStateException();
152 
153     double acc = 0.0;
154     for(int i = _prefillSize; i < _fillIndex; ++i){
155         acc += _latitude[i];
156     }
157 
158     return acc / (_fillIndex - _prefillSize);
159 }
160 
161 public double getMiddleLatitude(){
162     int middle = computeMiddleIndex();
163     if(_fillIndex <= middle)
164         throw new IllegalStateException();
165 
166     return _latitude[middle];
167 }
```

```
168
169 public long getFirstTimestamp(){
170     if(_fillIndex <= _prefillSize)
171         throw new IllegalStateException();
172
173     return _timestamps[_prefillSize];
174 }
175
176 public long getMeanTimestamp(){
177     if(_fillIndex <= _prefillSize)
178         throw new IllegalStateException();
179
180     double acc = 0.0;
181     for(int i = _prefillSize; i < _fillIndex; ++i){
182         acc += _timestamps[i];
183     }
184
185     return (long)(acc / (_fillIndex - _prefillSize));
186 }
187
188 public long getLastTimestamp(){
189     if(!isFull())
190         throw new IllegalStateException();
191
192     return _timestamps[_size-1];
193 }
194
195 public float getMeanSpeed(){
196     if(_fillIndex <= _prefillSize)
197         throw new IllegalStateException();
198
199     double acc = 0;
200     int count = 0;
201     for(int i = _prefillSize; i < _fillIndex; ++i){
202         if(_speed[i] > 0){
203             acc += _speed[i];
204             count++;
205         }
206     }
```

```
207
208         if(count == 0){
209             return 0;
210         }
211
212         return (float)(acc / (double)count);
213     }
214
215     public float getMeanBearing(){
216         if(_fillIndex <= _prefillSize)
217             throw new IllegalStateException();
218
219         double acc = 0;
220         for(int i = _prefillSize; i < _fillIndex; ++i){
221             acc += _bearing[i];
222         }
223
224         return (float)(acc / (_fillIndex - _prefillSize));
225     }
226
227     public int getMeanAccuracy(){
228         if(_fillIndex <= _prefillSize)
229             throw new IllegalStateException();
230
231         long acc = 0;
232         int count = 0;
233         for(int i = _prefillSize; i < _fillIndex; ++i){
234             if(_accuracy[i] > 0){
235                 acc += _accuracy[i];
236                 count++;
237             }
238         }
239
240         if(count == 0){
241             return 0;
242         }
243
244         return (int)((double)acc / (double)count);
245         }
```

```
246
247 }
```

### A.2.6 DataEntry.java

```
1  package it.uniurb.smartroadsense.algorithm;
2
3  import java.util.Date;
4
5  /**
6  * Instantaneous raw data entry.
7  */
8  public class DataEntry {
9
10     final static float DEFAULT_SPEED = 0.0f;
11     final static float DEFAULT_BEARING = 0.0f;
12     final static int DEFAULT_ACCURACY = 0;
13
14     /**
15     * Create a new data entry.
16     * @param timestamp Timestamp on which the entry ...
          was generated.
17     * @param accX X-axis accelerometer value.
18     * @param accY Y-axis accelerometer value.
19     * @param accZ Z-axis accelerometer value.
20     * @param latitude GPS latitude.
21     * @param longitude GPS longitude.
22     */
23     public DataEntry(Date timestamp,
24     double accX, double accY, double accZ,
25     double latitude, double longitude){
26
27         this(timestamp,
28         accX, accY, accZ,
29         latitude, longitude,
30         DEFAULT_SPEED,
31         DEFAULT_BEARING,
32         DEFAULT_ACCURACY);
```

```
33      }
34
35      /**
36      * Create a new data entry.
37      * @param timestamp Timestamp on which the entry ...
           was generated.
38      * @param accX X-axis accelerometer value.
39      * @param accY Y-axis accelerometer value.
40      * @param accZ Z-axis accelerometer value.
41      * @param latitude GPS latitude.
42      * @param longitude GPS longitude.
43      * @param speed GPS speed in m/s units.
44      */
45      public DataEntry(Date timestamp,
46      double accX, double accY, double accZ,
47      double latitude, double longitude,
48      float speed){
49
50          this(timestamp,
51          accX, accY, accZ,
52          latitude, longitude,
53          speed,
54          DEFAULT_BEARING,
55          DEFAULT_ACCURACY);
56      }
57
58      /**
59      * Create a new data entry.
60      * @param timestamp Timestamp on which the entry ...
           was generated.
61      * @param accX X-axis accelerometer value.
62      * @param accY Y-axis accelerometer value.
63      * @param accZ Z-axis accelerometer value.
64      * @param latitude GPS latitude.
65      * @param longitude GPS longitude.
66      * @param speed GPS speed in m/s units.
67      * @param bearing GPS bearing.
68      * @param accuracy GPS accuracy, in meters.
69      */
```

```
70      public DataEntry(Date timestamp,
71      double accX, double accY, double accZ,
72      double latitude, double longitude,
73      float speed,
74      float bearing,
75      int accuracy){
76
77          _timestamp = timestamp.getTime();
78
79          if(!isValid(accX) || !isValid(accY) || ...
                !isValid(accZ))
80          throw new ...
                IllegalArgumentException("Acceleration ...
                values not valid (NaN)");
81          _accX = accX;
82          _accY = accY;
83          _accZ = accZ;
84
85          if(!isValid(latitude) || !isValid(longitude))
86          throw new IllegalArgumentException("Geographic ...
                values not valid (NaN)");
87          _latitude = latitude;
88          _longitude = longitude;
89
90          _speed = speed;
91
92          if(!isValid(bearing))
93          throw new IllegalArgumentException("Bearing ...
                value not valid (NaN)");
94          _bearing = bearing;
95
96          _accuracy = accuracy;
97      }
98
99      private boolean isValid(double v){
100         if(Double.isInfinite(v) || Double.isNaN(v))
101         return false;
102         else
103         return true;
```

```
104     }
105
106     private boolean isValid(float v){
107         if(Float.isInfinite(v) || Float.isNaN(v))
108         return false;
109         else
110         return true;
111     }
112
113     final long _timestamp;
114
115     /**
116     * Gets the data entry timestamp.
117     */
118     public long getTimestamp(){
119         return _timestamp;
120     }
121
122     final double _accX, _accY, _accZ;
123
124     public double getAccelerationX(){
125         return _accX;
126     }
127
128     public double getAccelerationY(){
129         return _accY;
130     }
131
132     public double getAccelerationZ(){
133         return _accZ;
134     }
135
136     final double _latitude, _longitude;
137
138     public double getLatitude(){
139         return _latitude;
140     }
141
142     public double getLongitude(){
```

```
143             return _longitude;
144         }
145 
146         final float _speed;
147 
148         /**
149         * Gets GPS speed, in m/s units.
150         */
151         public float getSpeed(){
152             return _speed;
153         }
154 
155         final float _bearing;
156 
157         public float getBearing(){
158             return _bearing;
159         }
160 
161         final int _accuracy;
162 
163         /**
164         * Get GPS accuracy, in meters.
165         */
166         public int getAccuracy(){
167             return _accuracy;
168         }
169 
170 }
```

### A.2.7 DataHelpers.java

```
1 package it.uniurb.smartroadsense.algorithm;
2 
3 class DataHelpers {
4 
5     /**
6     * Extracts a sub array from a double array.
7     * @param origin Original data array to use as source.
```

```
8       * @param start Start point of the array.
9       * @param length Length of data to copy.
10      */
11      static double[] subarray(double[] origin, int ...
           start, int length){
12          if(start >= origin.length)
13          throw new IllegalArgumentException("Start of ...
               subarray out of bounds of original array");
14          if(start + length > origin.length)
15          throw new IllegalArgumentException("End of ...
               subarray out of bounds of original array");
16
17          double[] ret = new double[length];
18          for(int i = 0; i < ret.length; ++i){
19              ret[i] = origin[start + i];
20          }
21
22          return ret;
23      }
24
25      /**
26      * Reverses a double array.
27      */
28      static double[] reverse(double[] origin){
29          double[] ret = new double[origin.length];
30
31          for(int i = 0; i < ret.length; ++i){
32              ret[i] = origin[origin.length - i - 1];
33          }
34
35          return ret;
36      }
37
38      /**
39      * Appends an array to another array and returns a ...
           copy.
40      * @param a First array.
41      * @param b Second array.
```

```
42      * @return A new array that concatenates the ...
            entries of b to the entries of a.
43      */
44      static double[] append(double[] a, double[] b){
45          double[] ret = new double[a.length + b.length];
46          for(int i = 0; i < a.length; ++i){
47              ret[i] = a[i];
48          }
49          for(int i = 0; i < b.length; ++i){
50              ret[a.length + i] = b[i];
51          }
52          return ret;
53      }
54
55  }
```

### A.2.8 Engine.java

```
1   package it.uniurb.smartroadsense.algorithm;
2
3   import java.util.Date;
4
5   public class Engine {
6
7       public Engine(){
8           setNewSettings(new Settings());
9       }
10
11      public Engine(Settings settings){
12          if(settings == null){
13              throw new ...
                    IllegalArgumentException("Settings ...
                    cannot be null");
14          }
15
16          setNewSettings(settings);
17      }
18
```

```
19      private Settings _settings;
20
21      /**
22      * Resets the current computation completely.
23      */
24      public void reset() {
25          int bufferSize = _settings.getBufferSize();
26          _primaryBuffer = new Buffer(bufferSize);
27          _backgroundBuffer = new Buffer(bufferSize);
28
29          _primaryBuffer.prefill
30          (_settings.getWindowOverlap());
31
32          log("Engine reset with sampling buffers of ...
               size " + bufferSize);
33      }
34
35      /**
36      * Sets new settings for the algorithm.
37      * Resets the current computation completely.
38      */
39      public void setNewSettings(Settings settings){
40          _settings = settings;
41
42          this.reset();
43      }
44
45      private ComputationListener _listener = null;
46
47      /**
48      * Sets a new computation listener for results from ...
          the engine.
49      */
50      public void ...
          setComputationListener(ComputationListener ...
          listener){
51           _listener = listener;
52      }
53
```

```
54      private static LogListener _logListener = null;
55
56      /**
57      * Sets a new log listener.
58      */
59      public void setLogListener(LogListener listener){
60          _logListener = listener;
61      }
62
63      /**
64      * Logs a message through the external logging ...
           facility.
65      */
66      static void log(String message){
67          LogListener listener = _logListener;
68          if(listener == null)
69          return;
70
71          listener.log(message);
72      }
73
74      /**
75      * Logs an error message through the external ...
           logging facility.
76      */
77      static void logError(String message){
78          LogListener listener = _logListener;
79          if(listener == null)
80          return;
81
82          listener.logError(message);
83      }
84
85      private Buffer _primaryBuffer, _backgroundBuffer;
86
87      public void registerDataEntry(DataEntry entry){
88          //If we are in overflow position, record entry ...
               also in secondary buffer
```

```
89          if(_primaryBuffer.getIndex() >= ...
                _settings.getOverflowIndex()){
90              _backgroundBuffer.record(entry);
91          }
92
93          _primaryBuffer.record(entry);
94
95          //If buffer full, perform computation
96          if(_primaryBuffer.isFull()){
97              computePpeFromPrimaryBuffer();
98
99              _primaryBuffer.empty();
100
101             //Switch buffers
102             Buffer tmp = _primaryBuffer;
103             _primaryBuffer = _backgroundBuffer;
104             _backgroundBuffer = tmp;
105         }
106     }
107
108     private void computePpeFromPrimaryBuffer(){
109         log("Buffer full, computing PPE");
110
111         double ppeX = computePowerPredictionError
112         (_primaryBuffer._accX, _settings);
113         double ppeY = computePowerPredictionError
114         (_primaryBuffer._accY, _settings);
115         double ppeZ = computePowerPredictionError
116         (_primaryBuffer._accZ, _settings);
117
118         log(String.format("Raw PPE values are
119         x:%f y:%f z:%f", ppeX, ppeY, ppeZ));
120
121         Result result = new ResultSimpleMean(ppeX, ...
                ppeY, ppeZ);
122         result.FirstTimestamp = new ...
                Date(_primaryBuffer.getFirstTimestamp());
123         result.LastTimestamp = new ...
                Date(_primaryBuffer.getLastTimestamp());
```

```
124             result.Longitude = ...
                    _primaryBuffer.getMiddleLongitude();
125             result.Latitude = ...
                    _primaryBuffer.getMiddleLatitude();
126             result.MeanSpeed = _primaryBuffer.getMeanSpeed();
127             result.MeanBearing = ...
                    _primaryBuffer.getMeanBearing();
128             result.MeanAccuracy = ...
                    _primaryBuffer.getMeanAccuracy();
129
130             //Publish result
131             ComputationListener listener = _listener;
132             if(listener != null){
133                 listener.resultGenerated(result);
134             }
135         }
136
137         /**
138         * Computes the Power Prediction Error with an ...
                auto-regressive model.
139         */
140         private double ...
                computePowerPredictionError(double[] data,
141         Settings settings) {
142
143             MathHelpers.detrend(data);
144
145             double[] splitWindow = ...
                    MathHelpers.multiply(data, ...
                    settings.getHammingWindow());
146
147             double[] lambda = ...
                    estimateAutoRegressiveModel(splitWindow, ...
                    settings);
148
149             //Compute error
150             double errorAccumulator = 0.0;
151             for(int i = 0; i < ...
                    settings.getPpeWindowSize(); ++i){
```

```
152             // temp(k : -1 : k - order)
153             double[] section = ...
                    DataHelpers.subarray(splitWindow,
154             i + settings.getPpeWindowOverlap() - ...
                    settings.getOrderOfModel(),
155             settings.getOrderOfModel() + 1);
156             section = DataHelpers.reverse(section);
157
158             // l*temp(k : -1 : k - order)
159             double tmp = ...
                    MathHelpers.matrixProductSum(lambda, ...
                    section);
160
161             //Add squared error
162             errorAccumulator += tmp * tmp;
163         }
164
165         //Returns mean error for c-th split window
166         return (errorAccumulator / ...
                    settings.getPpeWindowSize());
167     }
168
169     private double[] ...
            estimateAutoRegressiveModel(double[] splitWindow,
170     Settings settings) {
171
172         double[] autoCorrelation = ...
                    computeAutoCorrelation(splitWindow, settings);
173
174         double[] lambda = ...
                    computeLevinsonDurbin(autoCorrelation, ...
                    settings);
175
176         return lambda;
177     }
178
179     /**
180     * Computes auto-correlation values with negative size.
181     */
```

```
182     private double[] computeAutoCorrelation(double[] ...
            splitWindow,
183     Settings settings) {
184
185         //Produces (order + 1) results
186         double[] results = new ...
                double[(settings.getOrderOfModel() + 1)];
187
188         //Final layout as follows:
189         //0 1 ... N-1 N
190         //where N is the order of model
191         for(int order = 0; order < ...
                settings.getOrderOfModel() + 1; ++order){
192             double[] arProduct =
193             MathHelpers.autoCorrelationProduct
194             (splitWindow, order);
195             results[order] = MathHelpers.sum(arProduct);
196         }
197
198         return results;
199     }
200
201     private double[] computeLevinsonDurbin(double[] data,
202     Settings settings) {
203
204         //First step
205         double k = data[1] / data[0];
206         double[] A = new double[] { k };
207         double e = (1 - k * k) * data[0];
208
209         for(int i = 1; i < data.length - 1; ++i){
210             double matrixPart = ...
                    MathHelpers.matrixProductSum(A, ...
                    DataHelpers.subarray(data, 1, i));
211             k = (data[i+1] - matrixPart)/e;
212
213             A = recomputeLevinsonDurbinA(A, k, i);
214
215             e = (1 - k * k) * e;
```

```
216             }
217
218             return finalizeLevinsonDurbinResult(A);
219         }
220
221         private double[] recomputeLevinsonDurbinA(double[] ...
                A, double k, int i) {
222             // k * A(i-1:-1:1)
223             double[] kRevA =
224             MathHelpers.multiply(DataHelpers.reverse(A), k);
225
226             // A = [k; A - kRevA]
227             return MathHelpers.subtractAndPrepend(A, ...
                    kRevA, k);
228         }
229
230         private double[] ...
                finalizeLevinsonDurbinResult(double[] A) {
231             // A = [1; -A(N:-1:1)];
232             return DataHelpers.append(new double[] { 1.0 },
233             MathHelpers.flipSign(DataHelpers.reverse(A))
234             );
235         }
236
237 }
```

### A.2.9 LogListener.java

```
1  package it.uniurb.smartroadsense.algorithm;
2
3  /**
4  * Listener that receives log messages from the ...
       algorithm engine.
5  */
6  public interface LogListener {
7
8      /**
9      * Signals a generic debug message.
```

```
10      * @param message The debug message.
11      */
12      public void log(String message);
13
14      /**
15      * Signals an error message.
16      * @param message The error message.
17      */
18      public void logError(String message);
19
20  }
```

### A.2.10 MathHelpers.java

```
1   package it.uniurb.smartroadsense.algorithm;
2
3   public class MathHelpers {
4
5       /**
6       * Multiplies array times another array.
7       * @return Returns array a.
8       */
9       public static double[] multiply(double[] a, ...
            double[] b){
10          if(a.length != b.length)
11          throw new IllegalArgumentException("Arrays ...
                have different length (a = " + a.length + ...
                ", b = " + b.length + ")");
12
13          double[] ret = new double[a.length];
14          for(int i = 0; i < a.length; ++i){
15              ret[i] = a[i] * b[i];
16          }
17
18          return ret;
19      }
20
21      /**
```

```
22      * Multiplies array times a scalar value, re-using ...
           the array (no copy).
23      * @return Returns array a.
24      */
25      public static double[] multiply(double[] a, double s){
26          for(int i = 0; i < a.length; ++i){
27              a[i] *= s;
28          }
29
30          return a;
31      }
32
33      /**
34      * Subtracts array B from array A, then prepends a ...
           scalar to the result.
35      * @return Returns new array [prepend; a - b]
36      */
37      public static double[] subtractAndPrepend(double[] ...
           a, double[] b, double prepend){
38          if(a.length != b.length)
39          throw new IllegalArgumentException("Operands A ...
              and B have different length");
40
41          double[] ret = new double[a.length + 1];
42          ret[0] = prepend;
43
44          for(int i = 0; i < a.length; ++i){
45              ret[i + 1] = a[i] - b[i];
46          }
47
48          return ret;
49      }
50
51      public static double[] ...
           autoCorrelationProduct(double[] x, int order){
52          double[] out = new double[x.length + order];
53
54          for(int i = 0; i < out.length; ++i){
55              //Operand A is prepended by zeros (# order)
```

```
56              double a = 0;
57              if(i - order >= 0)
58              a = x[i - order];
59
60              //Operand B is followed by equal number of ...
                   zeros
61              double b = 0;
62              if(i < x.length)
63              b = x[i];
64
65              out[i] = a * b;
66          }
67
68          return out;
69      }
70
71      /**
72      * Computes the sum of all values in an array.
73      */
74      public static double sum(double[] x){
75          double ret = 0.0;
76          for(int i = 0; i < x.length; ++i){
77              ret += x[i];
78          }
79          return ret;
80      }
81
82      /**
83      * Computes the mean of the values in an array.
84      */
85      public static double mean(double[] x){
86          return (sum(x) / x.length);
87      }
88
89      /**
90      * Computes the product sum of all values in two ...
          mono-dimensional matrices (row times columns).
91      */
```

```
92      public static double matrixProductSum(double[] a, ...
           double[] b){
93          if(a == null || b == null)
94          throw new IllegalArgumentException();
95          if(a.length != b.length)
96          throw new IllegalArgumentException("Operand A ...
               and B have different length (a = " + ...
               a.length + ", b = " + b.length + ")");
97          if(a.length == 0)
98          throw new IllegalArgumentException("Operands ...
               cannot have length 0");
99
100         double ret = 0.0;
101         for(int i = 0; i < a.length; ++i){
102             ret += (a[i] * b[i]);
103         }
104
105         return ret;
106     }
107
108     /**
109     * Flips sign of double array (re-using the ...
           original array).
110     */
111     public static double[] flipSign(double[] a){
112         for(int i = 0; i < a.length; ++i){
113             a[i] = -a[i];
114         }
115         return a;
116     }
117
118     /**
119     * Computes the average of a signal using a moving ...
           window.
120     * @param signal Input signal.
121     * @param order Moving window size.
122     */
123     public static double[] ...
           computeMovingAverage(double[] signal,
```

```
124     int order) {
125
126         if(order <= 1)
127         throw new IllegalArgumentException("Moving ...
               average window size must be greater than 1.");
128
129         double[] ret = new double[signal.length];
130
131         for(int i = 0; i < ret.length; ++i){
132             ret[i] = 0.0;
133             for(int j = Math.max(0, i - order + 1); j ...
                   <= i; ++j){
134                 ret[i] += signal[j];
135             }
136             ret[i] /= order;
137         }
138
139         return ret;
140     }
141
142     public static void detrend(double[] data){
143         double mean = mean(data);
144
145         for(int i = 0; i < data.length; ++i){
146             data[i] = data[i] - mean;
147         }
148     }
149
150 }
```

### A.2.11 Matrix.java

```
1 package it.uniurb.smartroadsense.algorithm;
2
3 /**
4 * Fixed format matrix.
5 */
6 class Matrix {
```

```
7       /**
8       * Effective data store.
9       * Indexed by row, column - i.e. _data[row][column]
10      */
11      private final double[][] _data;
12
13      private final int _colCount,
14      _rowCount;
15
16      public Matrix(int rows, int cols){
17          _rowCount = rows;
18          _colCount = cols;
19          _data = new double[_rowCount][_colCount];
20      }
21
22      public Matrix(double[][] data){
23          if(data.length < 1)
24          throw new IllegalArgumentException("Cannot ...
               create matrix with 0 rows");
25
26          int tmpColCount = data[0].length;
27          for(int i = 1; i < data.length; ++i){
28              if(data[i].length != tmpColCount)
29              throw new IllegalArgumentException("Cannot ...
                   create jagged matrix");
30          }
31
32          _rowCount = data.length;
33          _colCount = data[0].length;
34          _data = data;
35      }
36
37      public Matrix(double[][] data, int rows, int cols){
38          if(rows <= 0 || cols <= 0)
39          throw new IllegalArgumentException("Illegal ...
               number of rows or columns");
40          if(data.length < rows)
41          throw new IllegalArgumentException("Matrix ...
               data has less rows than specified for matrix");
```

```
42          for(int i = 0; i < data.length; ++i){
43              if(data[i].length < cols)
44              throw new IllegalArgumentException("Matrix ...
                    data has less columns than specified ...
                    for matrix");
45          }
46
47          _rowCount = rows;
48          _colCount = cols;
49          _data = data;
50      }
51
52      double[][] getData(){
53          return _data;
54      }
55
56      public double[] getRow(int r){
57          if(r >= _rowCount)
58          throw new IllegalArgumentException("Row " + r ...
                + " is out of bounds of matrix");
59
60          double[] ret = new double[_colCount];
61          for(int i = 0; i < _colCount; ++i){
62              ret[i] = _data[r][i];
63          }
64
65          return ret;
66      }
67
68      public double[] getColumn(int c){
69          if(c >= _colCount)
70          throw new IllegalArgumentException("Column " + ...
                c + " is out of bounds of matrix");
71
72          double[] ret = new double[_rowCount];
73          for(int i = 0; i < _rowCount; ++i){
74              ret[i] = _data[i][c];
75          }
76
```

```
77          return ret;
78      }
79
80      public void setAt(int row, int column, double value){
81          if(row < 0 || column < 0 ||
82          row >= _rowCount || column >= _colCount)
83          throw new IllegalArgumentException("Out of ...
                bounds matrix access");
84
85          _data[row][column] = value;
86      }
87
88      public double getAt(int row, int column){
89          if(row < 0 || column < 0 ||
90          row >= _rowCount || column >= _colCount)
91          throw new IllegalArgumentException("Out of ...
                bounds matrix access");
92
93          return _data[row][column];
94      }
95
96  }
```

### A.2.12 PrintHelpers.java

```
1  package it.uniurb.smartroadsense.algorithm;
2
3  import java.io.FileNotFoundException;
4  import java.io.FileOutputStream;
5  import java.io.PrintStream;
6
7  class PrintHelpers {
8
9      private PrintHelpers(){
10
11     }
12
```

```
13      static void writeCsvValue(PrintStream writer, ...
            double value){
14          writer.print(String.format("%.10f", ...
                value).replace('.', ','));
15          writer.print(";");
16      }
17
18      static void dumpData(String filename, String ...
            title, double[] data){
19          try {
20              PrintStream writer = new PrintStream(new ...
                    FileOutputStream(filename, false));
21
22              writer.println(title);
23
24              for(int i = 0; i < data.length; ++i){
25                  writeCsvValue(writer, data[i]);
26                  writer.println();
27              }
28
29              writer.close();
30          }
31          catch (FileNotFoundException e) {
32              e.printStackTrace();
33          }
34      }
35
36      static void dumpData(String filename, String ...
            titleA, String titleB, String titleC, double[] ...
            a, double[] b, double[] c){
37          try {
38              PrintStream writer = new PrintStream(new ...
                    FileOutputStream(filename, false));
39
40              writer.println(titleA + ";" + titleB + ";" ...
                    + titleC + ";");
41
42              int length = Math.min(a.length, ...
                    Math.min(b.length, c.length));
```

```
43
44              for(int i = 0; i < length; ++i){
45                  writeCsvValue(writer, a[i]);
46                  writeCsvValue(writer, b[i]);
47                  writeCsvValue(writer, c[i]);
48
49                  writer.println();
50              }
51
52              writer.close();
53          }
54          catch (FileNotFoundException e) {
55              e.printStackTrace();
56          }
57      }
58
59      static void dumpData(String filename, String[] ...
            titles, double[][] data){
60          if(titles != null && titles.length != data.length)
61          throw new IllegalArgumentException("Differing ...
               number of columns");
62
63          int length = data[0].length;
64          for(int i = 0; i < data.length; ++i){
65              if(data[i].length != length)
66              throw new ...
                   IllegalArgumentException("Differing ...
                   number of rows");
67          }
68
69          try {
70              PrintStream writer = new PrintStream(new ...
                   FileOutputStream(filename, false));
71
72              if(titles != null){
73                  for(int i = 0; i < titles.length; ++i){
74                      writer.print(titles[i] + ";");
75                  }
76                  writer.println();
```

```
77              }
78
79              for(int i = 0; i < length; ++i){
80                  for(int c = 0; c < data.length; ++c){
81                      writeCsvValue(writer, data[c][i]);
82                  }
83
84                  writer.println();
85              }
86
87              writer.close();
88          }
89          catch (FileNotFoundException e) {
90              e.printStackTrace();
91          }
92      }
93
94  }
```

### A.2.13 Result.java

```
1  package it.uniurb.smartroadsense.algorithm;
2
3  import java.util.Date;
4
5  /**
6  * Result of a computation run by the algorithm on a ...
       set of values.
7  */
8  public abstract class Result {
9
10     public Result(double ppeX, double ppeY, double ppeZ){
11         PpeX = ppeX;
12         PpeY = ppeY;
13         PpeZ = ppeZ;
14         Ppe = computePpe();
15
```

```
16          Engine.log(String.format("PPE is %f, computed ...
                using %s", Ppe, this.getClass().getName()));
17      }
18
19      protected abstract double computePpe();
20
21      final double PpeX, PpeY, PpeZ;
22      final double Ppe;
23
24      public double getPpeX(){
25          return PpeX;
26      }
27
28      public double getPpeY(){
29          return PpeY;
30      }
31
32      public double getPpeZ(){
33          return PpeZ;
34      }
35
36      public double getPpe(){
37          return Ppe;
38      }
39
40      double Longitude, Latitude;
41
42      public double getLongitude(){
43          return Longitude;
44      }
45
46      public double getLatitude(){
47          return Latitude;
48      }
49
50      Date FirstTimestamp, LastTimestamp;
51
52      public Date getFirstTimestamp(){
53          return FirstTimestamp;
```

```
54      }
55
56      public Date getLastTimestamp(){
57          return LastTimestamp;
58      }
59
60      float MeanSpeed;
61
62      /**
63      * Gets the mean speed, in m/s units.
64      */
65      public float getMeanSpeed(){
66          return MeanSpeed;
67      }
68
69      float MeanBearing;
70
71      public float getMeanBearing(){
72          return MeanBearing;
73      }
74
75      int MeanAccuracy;
76
77      /**
78      * Gets the mean GPS accuracy, in meters.
79      */
80      public int getMeanAccuracy(){
81          return MeanAccuracy;
82      }
83
84  }
```

### A.2.14 ResultSimpleMean.java

```
1  package it.uniurb.smartroadsense.algorithm;
2
3  class ResultSimpleMean extends Result {
4
```

```
5       public ResultSimpleMean(double ppeX, double ppeY, ...
            double ppeZ){
6           super(ppeX, ppeY, ppeZ);
7       }
8
9       @Override
10      protected double computePpe() {
11          return (PpeX + PpeY + PpeZ) / 3.0;
12      }
13
14  }
```

### A.2.15 Settings.java

```
1   package it.uniurb.smartroadsense.algorithm;
2
3   public class Settings {
4
5       private int _windowSize = 100;
6
7       /**
8       * Gets size of computation window in terms of new ...
            values per sampling.
9       * This is the number of actual values that are not ...
            repeated for
10      * each sampling.
11      */
12      public int getWindowSize(){
13          return _windowSize;
14      }
15
16      /**
17      * Sets size of computation window.
18      */
19      public void setWindowSize(int windowSize){
20          if(windowSize < 1)
21          throw new IllegalArgumentException();
22
```

```
23          _windowSize = windowSize;
24          _hammingWindow = null;
25      }
26
27      public int getPpeWindowOverlap(){
28          return _windowOverlap / 2;
29      }
30
31      public int getPpeWindowSize(){
32          return _windowSize;
33      }
34
35      private int _windowOverlap = 50;
36
37      /**
38      * Gets number of values that overlap between buffers.
39      */
40      public int getWindowOverlap(){
41          return _windowOverlap;
42      }
43
44      /**
45      * Sets number of overlapping values.
46      */
47      public void setWindowOverlap(int windowOverlap){
48          if(windowOverlap < 0)
49          throw new IllegalArgumentException();
50
51          _windowOverlap = windowOverlap;
52          _hammingWindow = null;
53      }
54
55      /**
56      * Gets the total size of a single buffer.
57      */
58      int getBufferSize(){
59          return _windowSize + _windowOverlap;
60      }
61
```

```
62      /**
63      * Gets the buffer index where values start ...
           overflowing into the next buffer.
64      */
65      int getOverflowIndex(){
66          return _windowSize;
67      }
68
69      private int _orderOfModel = 6;
70
71      /**
72      * Get order of auto-regressive model.
73      */
74      public int getOrderOfModel(){
75          return _orderOfModel;
76      }
77
78      /**
79      * Set order of auto-regressive model.
80      */
81      public void setOrderOfModel(int order){
82          if(order < 1)
83          throw new IllegalArgumentException();
84
85          _orderOfModel = order;
86      }
87
88      private int _movingAverageOrder = 10;
89
90      public int getMovingAverageOrder(){
91          return _movingAverageOrder;
92      }
93
94      public void setMovingAverageOrder(int order){
95          if(order < 1)
96          throw new IllegalArgumentException();
97
98          _movingAverageOrder = order;
99      }
```

```
100
101     private double[] _hammingWindow = null;
102
103     /**
104     * Computes a Hamming window matching the buffer ...
           length.
105     */
106     public double[] getHammingWindow(){
107         if(_hammingWindow == null){
108             int windowSize = this.getBufferSize();
109             double ch = 2.0 * Math.PI / (windowSize - 1);
110
111             double[] window = new double[windowSize];
112             for(int i = 0; i < windowSize; ++i){
113                 window[i] = (0.54 - 0.46 * Math.cos(ch ...
                       * (double)i));
114             }
115
116             _hammingWindow = window;
117         }
118
119         return _hammingWindow;
120     }
121
122 }
```

### A.2.16 DataLoader.java

```
1  package it.uniurb.smartroadsense.algorithm.filesystem;
2
3  import it.uniurb.smartroadsense.algorithm.DataEntry;
4
5  import java.io.BufferedReader;
6  import java.io.FileReader;
7  import java.io.IOException;
8  import java.util.ArrayList;
9  import java.util.Date;
10 import java.util.List;
```

```
11
12 public class DataLoader {
13
14     private final String _filename;
15     private boolean _prepared = false;
16     private FileReader _reader;
17
18     public DataLoader(String filename){
19         _filename = filename;
20     }
21
22     public boolean prepare(){
23         if(_prepared)
24         return true;
25
26         try {
27             _reader = new FileReader(_filename);
28         }
29         catch (Exception e) {
30             e.printStackTrace();
31             return false;
32         }
33
34         _prepared = true;
35         return true;
36     }
37
38     public List<DataEntry> load(){
39         if(!_prepared)
40         throw new IllegalStateException();
41
42         ArrayList<DataEntry> ret = new ...
               ArrayList<DataEntry>();
43
44         int skippedLines = 0;
45
46         BufferedReader reader = new ...
               BufferedReader(_reader);
47         do{
```

```
48              String line = readLine(reader);
49              if(line == null)
50              break;
51
52              String[] lineSplits = line.split(";");
53
54              long ts = Long.parseLong(lineSplits[0]);
55              double ax = Double.parseDouble(lineSplits[1]);
56              double ay = Double.parseDouble(lineSplits[2]);
57              double az = Double.parseDouble(lineSplits[3]);
58              if(lineSplits[4] == null || ...
                   lineSplits[4].length() <= 0){
59                  //Skip lines without GPS info
60                  skippedLines++;
61                  continue;
62              }
63              double gpsLat = ...
                   Double.parseDouble(lineSplits[4]);
64              double gpsLong = ...
                   Double.parseDouble(lineSplits[5]);
65              //double gpsQuota = ...
                   Double.parseDouble(lineSplits[6]);
66              double velocity = ...
                   Double.parseDouble(lineSplits[7]);
67
68              ret.add(new DataEntry(new Date(ts), ax, ...
                   ay, az, gpsLat, gpsLong, (int)velocity));
69          }
70          while(true);
71
72          System.out.println("Read file, " + ret.size() ...
               + " values, skipping " + skippedLines + " ...
               lines.");
73
74          return ret;
75      }
76
77      private String readLine(BufferedReader reader) {
78          String line = null;
```

```
79          try {
80              line = reader.readLine();
81          }
82          catch (IOException e) {
83              e.printStackTrace();
84              return null;
85          }
86
87          return line;
88      }
89
90  }
```

### A.2.17 Program.java

Questi ultimi due codici sono—come evidente dal `pakage`—finalizzati a operazioni di test sul programma stesso. Vengono riportati non solo per completezza, ma in quanto molto utili qualora si volesse testare l'applicativo con un emulatore di Android©.

```
1  package it.uniurb.smartroadsense.algorithm.tester;
2
3  import java.io.FileNotFoundException;
4  import java.util.List;
5  import java.util.Random;
6
7  import it.uniurb.smartroadsense.algorithm.DataEntry;
8  import it.uniurb.smartroadsense.algorithm.Engine;
9  import it.uniurb.smartroadsense.algorithm.LogListener;
10 import it.uniurb.smartroadsense.algorithm.MathHelpers;
11 import it.uniurb.smartroadsense.algorithm.Settings;
12 import
13  it.uniurb.smartroadsense.algorithm.filesystem.DataLoader;
14
15 public class Program {
16
17     final static String InputFile =
18     "C:\\Users\\Giacomo\\Documents\\Projects
19     \\SmartRoadSense\\Doc\\matlab\\SRS-01.csv";
```

```
20
21      private static void printHamming(int count) {
22          System.out.println("Hamming " + count);
23
24          Settings s = new Settings();
25          s.setWindowSize(count);
26
27          double[] hamming = s.getHammingWindow();
28          for(int i = 0; i < hamming.length; ++i)
29          System.out.print(hamming[i] + ", ");
30          System.out.println();
31      }
32
33      private static double[] getRandom(int size) {
34          Random rnd = new Random();
35          double[] ret = new double[size];
36          for(int i = 0; i < size; ++i) {
37              ret[i] = rnd.nextDouble() * 10.0;
38          }
39
40          return ret;
41      }
42
43      private static void dump(String title, double[] ...
            data) {
44          System.out.println(title);
45          for(int i = 0; i < data.length; ++i)
46          System.out.print(data[i] + ", ");
47          System.out.println();
48      }
49
50      private static void unitTestStub() {
51          Settings s = new Settings();
52          s.setWindowSize(100);
53          Engine tmp = new Engine(s);
54
55          double[] input = getRandom(150);
56          dump("input", input);
57
```

```
58          MathHelpers.detrend(input);
59          dump("detrend", input);
60
61          double[] yo = MathHelpers.multiply(input, ...
               s.getHammingWindow());
62          dump("final", yo);
63      }
64
65      public static void main(String[] args) {
66          unitTestStub();
67
68          System.out.println("Loading data file...");
69
70          DataLoader loader = new DataLoader(InputFile);
71          if(!loader.prepare()){
72              System.out.println("Failed to load file.");
73              return;
74          }
75          List<DataEntry> filedata = loader.load();
76
77          System.out.println("Loaded " + filedata.size() ...
               + " entries.");
78
79          Engine engine = new Engine();
80          engine.setLogListener(new LogListener() {
81
82              @Override
83              public void log(String message) {
84                  System.out.println("\t" + message);
85              }
86
87              @Override
88              public void logError(String message) {
89                  System.out.println("\tError: " + message);
90              }
91
92          });
93
94          ResultDumper dumper;
```

```
95          try {
96              dumper = new ResultDumper("C:\\
97              Users\\Giacomo\\Desktop\\output.csv");
98          } catch (FileNotFoundException e) {
99          e.printStackTrace();
100         return;
101     }
102     engine.setComputationListener(dumper);
103
104     //long start = System.nanoTime();
105
106     for(DataEntry entry : filedata){
107         engine.registerDataEntry(entry);
108     }
109
110     //long end = System.nanoTime();
111
112     //System.out.println("ns: " + (end - start));
113
114     dumper.close();
115
116     System.out.println("Done.");
117 }
118
119 }
```

### A.2.18 ResultDumper.java

```
1 package it.uniurb.smartroadsense.algorithm.tester;
2
3 import java.io.Closeable;
4 import java.io.FileNotFoundException;
5 import java.io.FileOutputStream;
6 import java.io.PrintStream;
7
8 import ...
    it.uniurb.smartroadsense.algorithm.ComputationListener;
```

```
9  import it.uniurb.smartroadsense.algorithm.Result;
10
11 public class ResultDumper implements ...
       ComputationListener, Closeable {
12
13     private final PrintStream _output;
14
15     public ResultDumper(String outputPath) throws ...
           FileNotFoundException {
16         _output = new PrintStream(new ...
               FileOutputStream(outputPath, false));
17
18         _output.println("ppe_mean;");
19     }
20
21     @Override
22     public void resultGenerated(Result value) {
23         //PrintHelpers.writeCsvValue(_output, value);
24         _output.println(value.getPpe());
25
26         System.out.println("PPE " + value.getPpe() + " ...
               speed " + value.getMeanSpeed());
27     }
28
29     @Override
30     public void close(){
31         _output.close();
32     }
33
34 }
```

# Bibliografia


[1] A. Bogliolo, A. Aldini, G. Alessandroni, A. Carini, S. Delpriori, V. Freschi, L. C. Klopfenstein, E. Lattanzi, G. Luchetti, B. D. Paolini, and A. Seraghiti. (2015) The SmartRoadSense website. University of Urbino. [Online]. Available: http://smartroadsense.it/

[2] Word Road Association, "The importance of road maintenance," Word Road Association, Tech. Rep. 2014R02EN, 2014.

[3] G. Alessandroni, L. Klopfenstein, S. Delpriori, M. Dromedari, G. Luchetti, B. Paolini, A. Seraghiti, E. Lattanzi, V. Freschi, A. Carini, and A. Bogliolo, "Smartroadsense: Collaborative road surface condition monitoring," in *Proceedings of the UBICOMM 2014: The Eighth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, 2014, pp. 210–215.

[4] V. Freschi, S. Delpriori, L. C. Klopfenstein, E. Lattanzi, G. Luchetti, and A. Bogliolo, "Geospatial data aggregation and reduction in vehicular sensing applications: the case of road surface monitoring," *Proceedings of the 3rd International Conference on Connected Vehicles & Expo*, 2014.

[5] G. Alessandroni, A. Bogliolo, A. Carini, S. Delpriori, V. Freschi, L. Klopfenstein, E. Lattanzi, G. Luchetti, B. Paolini, and A. Seraghiti, "Demo: Mobile crowdsensing of road surface roughness," in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2015, p. 439.

[6] S. Delpriori, V. Freschi, E. Lattanzi, and A. Bogliolo, "Efficient algorithms for accuracy improvement in mobile crowdsensing vehicular applications," *UBICOMM 2015*, pp. 145–150, 2015.

[7] G. Alessandroni, A. Carini, E. Lattanzi, and A. Bogliolo, "Sensing road roughness via mobile devices: a study on speed influence," in *9th International Symposium on Image and Signal Processing and Analysis (ISPA)*. IEEE, 2015, pp. 270–275.

[8] T. D. Gillespie, *Fundamentals of vehicle dynamics*. Society of Automotive Engineers, 1992.

[9] Society of Automotive Engineers. Vehicle Dynamics Committee, *Ride and Vibration Data Manual: SAE J6A: Report of Riding Comfort Research Committee Approved July 1946 and Last Revised by the Vehicle Dynamics Committee October 1965*, ser. SAE information report. Society of Automotive Engineers, 1965.

[10] T. Dahlberg, "Optimization criteria for vehicles travelling on a randomly profiled road–a survey," *Vehicle System Dynamics*, vol. 8, no. 4, pp. 239–352, 1979.

[11] N. Q. I. S. Committee *et al.*, *National highway user survey.* Coopers & Lybrand LLP, 1996.

[12] J. Y. Wong, *Theory of ground vehicles.* John Wiley & Sons, 2001.

[13] K. Bogsjö, "Road profile statistics relevant for vehicle fatigue," Ph.D. dissertation, Lund University, 2007.

[14] K. Bogsjö, K. Podgorski, and I. Rychlik, "Models for road surface roughness," vol. 50, no. 5, pp. 725–747, 2012.

[15] J. Laurent, M. Talbot, and M. Doucet, "Road surface inspection using laser scanners adapted for the high precision 3D measurements of large flat surfaces," in *3–D Digital Imaging and Modeling, 1997. Proceedings., International Conference on Recent Advances in.* IEEE, 1997, pp. 303–310.

[16] M. Ndoye, A. M. Barker, J. V. Krogmeier, and D. M. Bullock, "A recursive multi-scale correlation-averaging algorithm for an automated distributed road–condition–monitoring system," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 12, no. 3, pp. 795–808, 2011.

[17] A. González, E. J. O'brien, Y.-Y. Li, and K. Cashell, "The use of vehicle acceleration measurements to estimate road roughness," *Vehicle System Dynamics*, vol. 46, no. 6, pp. 483–499, 2008.

[18] J. Eriksson, L. Girod, B. Hull, R. Newton, S. Madden, and H. Balakrishnan, "The pothole patrol: using a mobile sensor network for road surface monitoring," in *Proceedings of the 6th international conference on Mobile systems, applications, and services.* ACM, 2008, pp. 29–39.

[19] P. Mohan, V. N. Padmanabhan, and R. Ramjee, "Nericell: rich monitoring of road and traffic conditions using mobile smartphones," in *Proceedings of the 6th ACM conference on Embedded network sensor systems.* ACM, 2008, pp. 323–336.

[20] A. Mednis, G. Strazdins, R. Zviedris, G. Kanonirs, and L. Selavo, "Real time pothole detection using android smartphones with accelerometers," in *Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on*. IEEE, 2011, pp. 1–6.

[21] G. Strazdins, A. Mednis, G. Kanonirs, R. Zviedris, and L. Selavo, "Towards vehicular sensor networks with android smartphones for road surface monitoring," in *2nd International Workshop on Networks of Cooperating Objects, Chicago, USA*, 2011.

[22] Y. Tai, C. Chan, and J. Y. Hsu, "Automatic road anomaly detection using smart mobile device," in *Conference on Technologies and Applications of Artificial Intelligence, Hsinchu, Taiwan*, 2010, pp. 1–8.

[23] M. Perttunen, O. Mazhelis, F. Cong, M. Kauppila, T. Leppänen, J. Kantola, J. Collin, S. Pirttikangas, J. Haverinen, T. Ristaniemi, and J. Riekki, "Distributed road surface condition monitoring using mobile phones," in *Ubiquitous Intelligence and Computing*. Springer, 2011, pp. 64–78.

[24] K. Chen, M. Lu, G. Tan, and J. Wu, "Crsm: Crowdsourcing based road surface monitoring," in *High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on*. IEEE, 2013, pp. 2151–2158.

[25] C.-W. Yi, Y.-T. Chuang, and C.-S. Nian, "Toward crowdsourcing-based road pavement monitoring by mobile sensing technologies," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 4, pp. 1905–1917, 2015.

[26] F. Seraj, K. Zhang, O. Turkes, N. Meratnia, and P. J. Havinga, "A smartphone based method to enhance road pavement anomaly detection by analyzing the driver behavior," in *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers*. ACM, 2015, pp. 1169–1177.

[27] V. Douangphachanh and H. Oneyama, "A study on the use of smartphones for road roughness condition estimation," *Journal of the Eastern Asia Society for Transportation Studies*, vol. 10, no. 0, pp. 1551–1564, 2013.

[28] ——, "Using smartphones to estimate road pavement condition," *International Symposium for Next Generation Infrastructure*, 2013.

[29] ——, "A study on the use of smartphones under realistic settings to estimate road roughness condition," *EURASIP Journal on Wireless Communications and Networking*, vol. 2014, no. 1, pp. 1–11, 2014.

[30] Central Intelligence Agency, *The World Factbook*. Washington, DC: Central Intelligence Agency, 2015.

[31] H. Bello-Salau, A. M. Aibinu, E. N. Onwuka, J. J. Dukiya, M. E. Bima, A. J. Onumanyi, and T. A. Folorunso, "A new measure for analysing accelerometer data towards developing efficient road defect profiling systems," *Journal of Scientific Research & Reports*, vol. 7, no. 2, pp. 108–116, 2015.

[32] G. U. Yule, "On a method of investigating periodicities in disturbed series, with special reference to wolfer's sunspot numbers," *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, pp. 267–298, 1927.

[33] G. Walker, "On periodicity in series of related terms," *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, pp. 518–532, 1931.

[34] R. M. Gray, *Toeplitz and circulant matrices: A review*. Information Systems Laboratory, Stanford Electronics Laboratories, Stanford University, 1971, vol. 1.

[35] N. Levinson, "The wiener RMS error criterion in filter design and prediction," *Journal of Mathematical Physics*, vol. 25, pp. 261–278, 1947.

[36] J. Durbin, "The fitting of time-series models," *Revue de l'Institut International de Statistique*, pp. 233–244, 1960.

[37] M. Ndoye, S. V. Vanjari, H. Huh, J. V. Krogmeier, D. M. Bullock, C. A. Hedges, and A. Adewunmi, "Sensing and signal processing for a distributed pavement monitoring system," in *Digital Signal Processing Workshop, 12th-Signal Processing Education Workshop, 4th*. IEEE, 2006, pp. 162–167.

[38] R. M. Chalasani, "Ride performance potential of active suspension systems–part I: Simplified analysis based on a quarter-car model," in *Proceedings of the 1986 ASME Winter Annual Meeting*, 1986.

[39] R. N. Jazar, *Vehicle dynamics: Theory and Application*. Springer, 2008.

[40] B. R. Davis and A. G. Thompson, "Optimal linear active suspensions with integral constraint," *Vehicle System Dynamics*, vol. 17, no. 6, pp. 193–210, 1988.

[41] T. Butsuen, "The design of semi–active suspensions for automotive vehicles," Ph.D. dissertation, Massachusetts Institute of Technology, 1989.

[42] I. Fialho and G. J. Balas, "Road adaptive active suspension design using linear parameter–varying gain-scheduling," *Control Systems Technology, IEEE Transactions on*, vol. 10, no. 1, pp. 43–54, 2002.

[43] G. Verros, S. Natsiavas, and C. Papadimitriou, "Design optimization of quarter–car models with passive and semi–active suspensions under random road excitation," *Journal of Vibration and Control*, vol. 11, no. 5, pp. 581–606, 2005.

[44] J. T. Allison, "Optimal partitioning and coordination decisions in decomposition–based design optimization," Ph.D. dissertation, The University of Michigan, 2008.

[45] M. M. M. Salem and A. A. Aly, "Fuzzy control of a quarter–car suspension system," *World Academy of Science, Engineering and Technology*, vol. 53, no. 5, pp. 258–263, 2009.

[46] L. F. Shampine, "Vectorized adaptive quadrature in MATLAB," *Journal of Computational and Applied Mathematics*, vol. 211, no. 2, pp. 131–140, 2008.

[47] ——, "Weighted quadrature by change of variable," *Neural, Parallel & Scientific Computations*, vol. 18, no. 2, pp. 195–206, 2010.

[48] J. R. Rice, "A metalgorithm for adaptive quadrature," *Journal of the ACM (JACM)*, vol. 22, no. 1, pp. 61–82, 1975.

[49] S. L. Marple Jr, *Digital spectral analysis with applications*. Englewood Cliffs, NJ, Prentice–Hall, Inc., 1987, vol. 1.

[50] L. Ljung, *System identification*. Wiley Online Library, 1999.

[51] S. K. Mitra, *Digital Signal Processing: a Computer–Based Approach*, 4th ed. McGraw–Hill, New York, 2011.

[52] H. Akaike, "Fitting autoregressive models for prediction," *Annals of the institute of Statistical Mathematics*, vol. 21, no. 1, pp. 243–247, 1969.

[53] C. Roads, *Microsound*. The MIT Press, 2002.

[54] R. W. Hamming, *Digital filters*. Courier Corporation, 1989.

[55] OpenStreetMap Wiki. (2015) Key:highway—OpenStreetMap Wiki. [Online]. Available: http://wiki.openstreetmap.org/wiki/Key:highway

[56] C. Giavarini, "Il valore delle nostre strade," *Rassegna del bitume*, vol. 71, pp. 21–24, 2012.