ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DIPARTIMENTO DI INGEGNERIA INDUSTRIALE - DIN

CORSO DI LAUREA IN INGEGNERIA MECCANICA (5724)

TESI DI LAUREA

in

MECCANICA DELLE MACCHINE M

PIANIFICAZIONE DI LEGGI DI MOTO ANTI-SLOSHING DEFINITE A TRATTI

CANDIDATO:
Jury Sardellini

RELATORE:
Prof. Ing. Marco Carricato

CORRELATORI:
Luca Guagliumi
Filippo Brasina

Anno accademico 2024/2025

Sessione V

A mia sorella Jenny e mio fratello Jerry

i miei mentori,
i miei confidenti,
i miei esempi,
i miei secondi genitori.

Abstract

Il presente lavoro affronta il problema della mitigazione dello *sloshing* in sistemi di automazione industriale, con riferimento alla movimentazione ad alte prestazioni di recipienti parzialmente riempiti di liquido su macchine automatiche. Il fenomeno, che consiste nelle oscillazioni del pelo libero indotte dall'eccitazione dinamica del contenitore, può compromettere la qualità del prodotto, causare fuoriuscite e limitare la produttività delle linee. L'approccio proposto si basa sulla pianificazione ottimale di leggi di moto definite a tratti — in particolare la legge trapezia standard a sette segmenti — le cui durate vengono ottimizzate in modo da ridurre l'energia trasferita ai modi di *sloshing*, mantenendo piena compatibilità con i controllori di movimento industriali. Il sistema fisico è descritto tramite un modello meccanico equivalente lineare e non lineare, derivato dalla teoria analitica dello *sloshing* per geometrie cilindriche. Il contributo metodologico centrale consiste nella derivazione in forma chiusa dello spettro analitico della legge trapezia, che consente di formulare obiettivi e vincoli spettrali continui rispetto alle variabili decisionali, evitando le approssimazioni introdotte da trasformate numeriche. Vengono sviluppate e confrontate tre famiglie di strategie di ottimizzazione: (i) attenuazione spettrale con notch singolo o multi-notch in banda attorno alla pulsazione naturale dominante ω_n; (ii) minimizzazione del jerk con vincolo spettrale in banda, che introduce un criterio di regolarità del profilo; (iii) ottimizzazione nel dominio del tempo sulla risposta del modello non lineare, con vincolo esplicito sull'oscillazione residua. I problemi di ottimizzazione sono risolti numericamente tramite il solver IPOPT e il framework CasADi con differenziazione automatica. L'efficacia dei metodi è validata sperimentalmente su un banco prova a guida lineare, mediante acquisizione video e post-processing per la stima dell'altezza di *sloshing*, in diverse condizioni di corsa e tempo di avanzamento. I risultati mostrano riduzioni significative dell'oscillazione residua e benefici complessivamente positivi fino a dinamiche moderate; al crescere dell'aggressività del moto emergono limiti fisici e di modellazione che riducono i margini di mitigazione, evidenziando il compromesso strutturale tra tempo ciclo e contenimento dello *sloshing*.

Indice

Capitolo 1

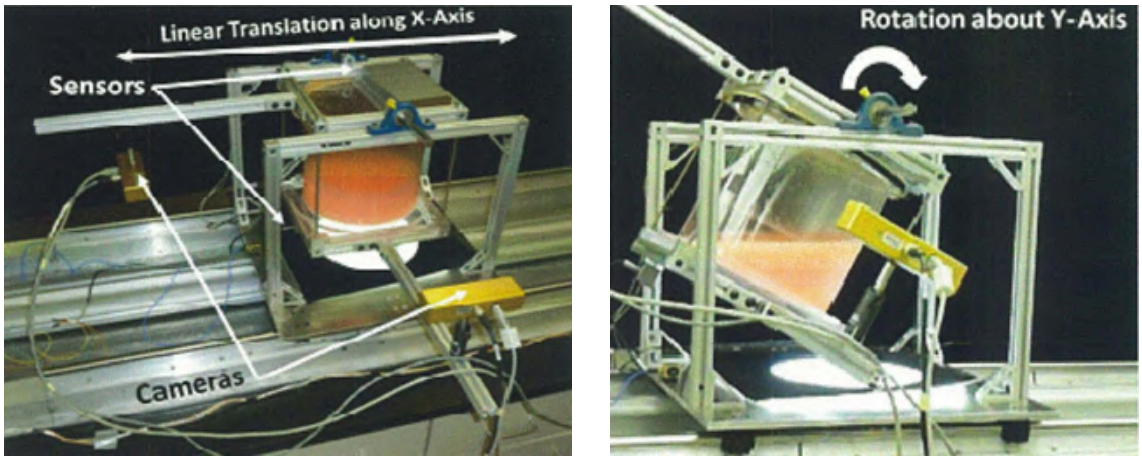
Introduzione

Il termine *sloshing* indica il moto oscillatorio del pelo libero di un liquido contenuto in un recipiente parzialmente riempito e soggetto a sollecitazioni dinamiche [1]. Quando il contenitore viene accelerato, frenato o varia la propria traiettoria, la massa fluida non può seguire istantaneamente il moto imposto dalle pareti, e ciò porta alla formazione di onde di superficie, come mostrato in Figura 1.1, e a spostamenti di massa che generano pressioni variabili sul bordo e sul fondo. In molti casi la risposta persiste anche dopo che l'eccitazione esterna si è esaurita, con una vibrazione residua che, a seconda delle condizioni operative, può risultare incompatibile con il funzionamento desiderato. Dal punto di vista ingegneristico, il fenomeno è importante perché introduce un accoppiamento tra la dinamica del liquido e quella del sistema che lo movimenta. L'attuatore e quindi il contenitore impongono una forzante (tipicamente un'accelerazione nel tempo), mentre il fluido risponde secondo modi propri che dipendono da geometria, livello di riempimento e proprietà del liquido. Ne consegue che la "bontà" di una traiettoria non è definita solo da criteri cinematici-dinamici classici (tempo ciclo, velocità, accelerazione e jerk), ma anche dalla capacità di limitare l'energia trasferita ai modi di sloshing. In altre parole, la progettazione della legge di moto diventa parte integrante del controllo del fenomeno.

Figura 1.1: Oscillazione di un liquido all'interno di un contenitore cilindrico movimentato dall'esterno.

L'effetto dello sloshing è rilevante in numerosi settori. In ambito navale, ad esempio, il moto del carico liquido nei serbatoi può influenzare stabilità e sollecitazioni, soprattutto in presenza di onde e manovre. Nei trasporti terrestri (autobotti e veicoli con serbatoi), l'inerzia del fluido può modificare la dinamica longitudinale e laterale del mezzo e contribuire a condizioni di rischio in frenata o in curva.

Un altro ambito di particolare interesse è quello spaziale e aerospaziale. La ricerca sul fenomeno dello sloshing si è sviluppata inizialmente proprio in questi due settori, dove la presenza di propellenti liquidi in serbatoi parzialmente riempiti ha imposto fin dagli anni '60 la necessità di comprenderne e controllarne gli effetti dinamici (Figura 1.2). In particolare, i programmi sperimentali promossi dalla NASA (National Aeronautics and Space Administration) in quel periodo hanno contribuito in modo decisivo alla costruzione della teoria di base del fenomeno e alla validazione di modelli semplificati e di metodi di calcolo dedicati. La motivazione è la seguente: durante le fasi di lancio e di manovra di lanciatori e veicoli spaziali, i serbatoi sono soggetti ad accelerazioni e variazioni di assetto tali da eccitare la superficie libera del liquido, generando oscillazioni che non possono essere considerate trascurabili. Lo sloshing del propellente nei serbatoi può interagire con la dinamica del veicolo e con i sistemi di controllo d'assetto. Le oscillazioni del liquido provocano spostamenti del baricentro e variazioni del momento d'inerzia, generando coppie indesiderate che il controllore deve compensare [2].
Anche su satelliti e sonde in orbita, dove il propellente residuo può essere una frazione significativa della massa totale, lo sloshing può contribuire a errori di puntamento, a disturbi sulle piattaforme di controllo e, più in generale, a prestazioni inferiori nelle operazioni di orientamento e di inseguimento. In questi contesti, soluzioni antisloshing sono spesso parte integrante del progetto, ma devono bilanciare benefici e penalità in massa, complessità e integrazione.



(a) Traslazione lineare forzata lungo l'asse x (b) Rotazione libera attorno all'asse y

Figura 1.2: Banco prova e configurazioni di eccitazione per prove di sloshing effettuate dalla NASA.

Su queste basi si è consolidata la modellazione fisica che, a seconda del livello di accuratezza richiesto, spazia da rappresentazioni equivalenti con modelli analitici per catturare i modi dominanti, fino ad approcci numerici come quello Computational Fluid Dynamics (CFD) accoppiato con la dinamica del corpo rigido [3]. L'ambito aerospaziale ha quindi fornito non solo le prime motivazioni applicative, ma anche una metodologia che ha reso lo sloshing un tema centrale nella dinamica dei sistemi con

liquidi in movimento, con risvolti successivi in molti altri settori dell'ingegneria. Per ridurre lo sloshing, molte soluzioni classiche agiscono sul recipiente o sul serbatoio, introducendo elementi interni quali: deflettori (baffles) o setti come, ad esempio, nelle autobotti e autocisterne. Altre soluzioni modificano invece la forma dei recipienti e le condizioni operative.

Spostando l'attenzione dal settore aerospaziale alle applicazioni di automazione industriale, il problema cambia natura: l'obiettivo rimane la mitigazione dello sloshing, ma i gradi di libertà progettuali si riducono. In ambito spaziale, dispositivi quali deflettori all'interno di serbatoi come quello mostrato in Figura 1.3 sono diffusi, ma devono essere giustificati rispetto a requisiti di massa e integrazione; in ambito industriale, invece, la geometria del recipiente coincide spesso con il prodotto commerciale e non è modificabile, mentre l'introduzione di elementi interni può essere incompatibile con vincoli igienici o di produzione.

Figura 1.3: Sezione del serbatoio del razzo spaziale Spica con baffles anti-sloshing.

Il progetto di tesi nasce da una necessità concreta tipica dell'automazione industriale, ossia la movimentazione rapida di recipienti contenenti liquidi lungo macchine automatiche. Proprio per questo il presente lavoro si inserisce nel filone di attività

sviluppate in collaborazione con Marchesini Group, realtà industriale che opera proprio nel settore delle macchine automatiche per il packaging primario e secondario, con particolare riferimento ai comparti farmaceutico e cosmetico. Marchesini Group è una multinazionale nata a Pianoro (BO) nel 1974 e, nelle proprie linee, gestisce frequentemente flaconi e contenitori riempiti con liquidi (medicinali, profumi, creme, ecc.), che devono essere movimentati a elevata produttività.

Una caratteristica tipica di questi sistemi è la presenza di leggi di moto fortemente condizionate dall'architettura della macchina. Si possono distinguere, in modo qualitativo, tre casi ricorrenti:

- moti rettilinei alternati tra stazioni, dove il contenitore viene accelerato e frenato lungo un asse (Figura 1.4a) ;
- moti circolari a velocità circa costante in macchine con architettura a giostra, dove agiscono accelerazioni centrifughe (Figura 1.4b);
- moti circolari non uniformi e transizioni tra ruote o stelle, in cui variano nel tempo sia accelerazione tangenziale sia accelerazione centripeta e in cui, nei casi più critici, possono comparire variazioni molto rapide dell'accelerazione.

In tale contesto, le esigenze di velocità impongono spesso leggi di moto caratterizzate da accelerazioni brusche, che equivalgono, dal punto di vista dinamico, a forzanti con contenuto spettrale ampio, capaci di eccitare i modi naturali del pelo libero e quindi generare sloshing. Le criticità risultanti sono sia funzionali sia qualitative, ed è dunque fondamentale mitigare il fenomeno per le seguenti ragioni:

- evitare la fuoriuscita del prodotto durante i trasferimenti, soprattutto quando i recipienti sono ancora aperti;
- ridurre la contaminazione e lo sporco sulle pareti interne del flacone, che può compromettere l'estetica del confezionato;
- prevenire depositi o residui (in particolare per prodotti in sospensione) che possono alterare la qualità del processo e la corretta lettura dei sensori di controllo/-scarto;
- limitare le oscillazioni residue del liquido, che possono propagarsi anche dopo la fine dell'eccitazione e interferire con le fasi successive della macchina;
- aumentare la massima produttività raggiungibile, evitando che lo sloshing diventi un vincolo sistematico nei cicli ad alta ripetitività.

Il problema è particolarmente delicato per prodotti farmaceutici in sospensione. In chimica una sospensione non è altro che una miscela in cui un materiale allo stato solido e finemente suddiviso è disperso in un altro materiale presente in quantità molto maggiori e allo stato liquido, in modo tale che il primo non sedimenti in tempo breve. Le sospensioni sono diverse dalle soluzioni in quanto le due fasi non si uniscono perfettamente dando origine ad un unico liquido, ma rimangono separate tra loro. Lo sloshing è un effetto intrinseco della movimentazione di liquidi in contenitori parzialmente riempiti; tuttavia, nel caso di prodotti farmaceutici in sospensione la sua mitigazione diventa particolarmente critica. In presenza di oscillazioni del pelo libero, infatti, la fase solida può depositarsi sulle pareti interne del flacone e, in alcuni casi,

(a) Moto rettilineo e stazione di riempimento.

(b) Moto rotatorio e stazione di tappatura.

Figura 1.4: Esempio di una macchina automatica prodotta da Marchesini Group. Nel primo tratto (figura a) i contenitori sono trasportati da una coclea e seguono quindi un moto rettilineo. In questa parte di macchina è presente inoltre la stazione di riempimento. Nella seconda parte del loro percorso (figura b), i recipienti si impegnano in una stella che dà loro un moto circolare. Questi passano poi in altre due stelle prima di tornare a compiere nuovamente un moto rettilineo. É durante questi passaggi che avviene la tappatura.

arrivare a cristallizzare, causando residui visibili e potenziali letture errate dei sensori di scarto, con conseguenti scarti non necessari di flaconi correttamente riempiti.

Nelle applicazioni considerate, non è in genere possibile intervenire né sulla geometria del recipiente (definita dal cliente) né sulla tipologia di liquido (che coincide con il prodotto finale di cui va ad usufruire il consumatore). Diventa quindi strategico considerare la traiettoria come un grado di libertà privilegiato: la legge di moto impartita al contenitore (forzante esterna) può essere progettata, compatibilmente con i vincoli industriali della macchina, in modo da trasferire meno energia ai modi di sloshing e ridurre così l'ampiezza delle oscillazioni.

Il parallelismo tra automazione e aerospazio mette in evidenza una prospettiva comune: lo sloshing è un fenomeno dinamico che dipende non solo dall'intensità della movimentazione, ma anche da come essa viene applicata nel tempo. Profili di moto con accelerazioni elevate e variazioni brusche dell'accelerazione tendono a trasferire più energia al liquido e, in determinate condizioni, possono amplificare le oscillazioni del pelo libero. Ne segue che non è sufficiente ragionare solo in termini di valori massimi di accelerazione: è altrettanto importante capire come la forzante è distribuita nel tempo e quale contenuto in frequenza essa introduce. Questa considerazione è alla base di molte strategie anti-sloshing (e più in generale anti-vibrazione) note, le quali possono essere ricondotte a tre famiglie principali: (i) tecniche di *input shaping* e filtri, che modellano il comando per attenuare le oscillazioni residue [4]; (ii) formulazioni di ottimizzazione della traiettoria, in cui si ricercano profili di moto che riducano l'energia trasferita al liquido compatibilmente con i vincoli di ciclo [5]; (iii) approcci numerici ad alta fedeltà (ad esempio CFD), più accurati ma spesso onerosi dal punto di vista computazionale [6]. Nelle applicazioni industriali reali, tuttavia, la legge di moto non può essere scelta in modo arbitrario: per ragioni di standardizzazione e compatibilità con i controllori, sono frequentemente impiegati profili predefiniti come ad esempio leggi trapezoidali o a tratti, i cui parametri vengono regolati dal progettista.

È importante chiarire fin da subito che l'obiettivo di questo lavoro non è sostituire approcci consolidati, quali tecniche di *input shaping* o soluzioni hardware dedicate, né proporre un metodo universalmente valido per qualunque geometria e condizione operativa. L'obiettivo è invece mostrare che, in un contesto industriale reale e vincolato, è possibile ottenere miglioramenti significativi agendo esclusivamente sulla pianificazione della traiettoria e mantenendo una legge di moto compatibile con i controlli esistenti. In questo solco si colloca il lavoro citato in [7], che propone una procedura per ottenere traiettorie anti-sloshing ottimizzando le durate dei segmenti di profili a tratti, riducendo in particolare le oscillazioni residue nella fase di quiete.

L'idea alla base di questo elaborato è perseguire una strategia analoga, mantenendo una legge di accelerazione standard e compatibile con i controllori industriali, ed eseguendo l'ottimizzazione *offline* dei suoi parametri, senza introdurre nuovo software macchina e senza intervenire sull'hardware. L'interesse per approcci *feedforward* basati su ottimizzazione *offline* della legge di moto è dovuto al fatto che nelle macchine automatiche ad alta produttività un controllo *online* in retroazione dello sloshing è spesso non praticabile, a causa della mancanza di misure dirette dello stato del liquido durante il ciclo. In particolare nel presente lavoro, si adotta una legge di accelerazione trapezia a sette segmenti, scelta perché semplice da parametrizzare e comunemente utilizzata in automazione industriale; l'ottimizzazione *offline* consente di ottenere un profilo di riferimento che il controllo può seguire con le funzionalità già disponibili. Rispetto all'approccio di [7], l'obiettivo è estendere il concetto di base includendo la

modellazione e l'ottimizzazione anche su un modello non lineare del fenomeno.

In un contesto industriale, inoltre, la movimentazione non avviene in condizioni "fisse", ma in funzione del *formato* di produzione impostato: per ciascun formato (identificato da codici/ricette macchina) sono noti e configurati parametri come geometria del recipiente e livello di riempimento, che influenzano direttamente la risposta dinamica del liquido. In quest'ottica, un obiettivo di medio periodo non è soltanto ottenere *un* set di parametri ottimi da caricare una volta per tutte, bensì rendere disponibile in macchina una procedura sufficientemente robusta e leggera da poter essere ri-eseguita automaticamente al cambio formato.
Un possibile sviluppo di interesse, coerente con l'impostazione del presente lavoro, consiste nel passare da un approccio *tabellare* (soluzioni ottime precalcolate per singoli casi) a un approccio *parametrico*, in cui i parametri della legge di moto vengono determinati a partire dalle caratteristiche del formato e del processo. Questa prospettiva è resa plausibile dal fatto che l'ottimizzazione agisce su un numero limitato di parametri e, in letteratura, approcci analoghi basati sull'ottimizzazione di profili a tratti mostrano tempi di calcolo contenuti.

1.1 Struttura e scopo del lavoro

Il prospetto del lavoro è duplice: da un lato, si imposta un problema di ottimizzazione numerica che agisce sui parametri temporali dei segmenti della legge, integrando vincoli di fattibilità (durate minime, limiti su velocità e accelerazione, continuità dei profili) e una funzione obiettivo legata alla risposta di sloshing. Dall'altro, si introduce esplicitamente un criterio nel dominio della frequenza: oltre alla valutazione nel dominio del tempo, la legge di moto viene analizzata in termini di spettro, con l'obiettivo di attenuare la componente della forzante in prossimità della pulsazione naturale del pelo libero.

Per ottimizzare la traiettoria è necessario disporre di un modello che, data in ingresso l'accelerazione del contenitore, fornisca una stima della risposta del liquido. Nell'ingegneria applicata allo sloshing sono spesso utilizzati modelli equivalenti meccanici, nei quali il fluido viene rappresentato tramite masse, molle e smorzatori [8]. I modelli sono gli elementi centrali per la valutazione delle prestazioni: per ogni candidata legge di moto, essi forniscono un indicatore della risposta (ad esempio l'altezza massima del pelo libero) che entra nella funzione obiettivo o nei vincoli dell'ottimizzazione. Nel progetto, e in particolare nel Capitolo 2, vengono considerati: sia un modello lineare, appropriato per piccole oscillazioni; sia un modello non lineare, utile quando l'ampiezza del moto del pelo libero non può essere considerata piccola o quando si vuole descrivere in modo più fedele l'evoluzione temporale dell'oscillazione. Questa scelta è coerente con scenari ad alte prestazioni, in cui tempi ciclo ridotti o manovre rapide possono portare a condizioni di eccitazione più severe.

Dal punto di vista operativo, l'ottimizzazione, di cui parleremo nel Capitolo 3, è stata sviluppata con una combinazione di strumenti di calcolo simbolico e numerico. La formulazione del problema di ottimo vincolato e la sua risoluzione numerica sono state implementate in ambiente Python tramite CasADi [9], framework che consente di costruire modelli e vincoli in forma simbolica e di ottenere automaticamente derivate utili ai solutori di ottimizzazione non lineare. L'uso di questo tipo di strumenti permette di integrare in modo naturale vincoli complessi e di iterare su molte configurazioni, mantenendo una base di codice riproducibile

Nel Capitolo 4 si richiamano i principi dell'input shaping, senza entrare in un dettaglio applicativo non necessario al presente progetto, e si discute la relazione tra tali tecniche e l'idea di agire sul contenuto in frequenza della forzante. Su questa base si introduce l'analisi spettrale della legge trapezia a tratti e il ragionamento sul vincolo di attenuazione in prossimità della frequenza naturale del pelo libero, includendo lo sviluppo analitico dello spettro. Quest'ultimo è stato implementato con supporto simbolico, tramite il software Mathematica utile alla derivazione e la verifica di espressioni analitiche.

Il Capitolo 5 presenta i risultati ottenuti. Si confrontano le traiettorie di riferimento con quelle ottimizzate sia a livello di simulazione sia tramite prove sperimentali su banco. Vengono descritte le caratteristiche essenziali del banco prova utilizzato e la procedura di test; a supporto dell'analisi vengono riportati frame estratti dai video delle prove, insieme all'andamento sperimentale dell'ampiezza di sloshing nel tempo ricavato tramite post-processing, così da affiancare una valutazione qualitativa a un confronto quantitativo tra traiettorie di riferimento e ottimizzate.

Infine, il Capitolo 6 riassume i contributi principali, presenta le conclusioni del lavoro e propone possibili sviluppi futuri.

Capitolo 2

Modellazione dello sloshing

Durante lo studio dei fenomeni fisici che regolano il comportamento dei sistemi naturali ci si imbatte spesso in problemi che, se affrontati in modo rigoroso ed esatto, risultano difficili da trattare dal punto di vista teorico e, in molti casi, eccessivamente onerosi dal punto di vista computazionale. Per questo motivo, in ambito ingegneristico è prassi adottare modelli equivalenti, costruiti su ipotesi semplificative controllate, che consentano di ottenere previsioni affidabili con un livello di complessità compatibile con gli obiettivi progettuali.

Il fenomeno dello sloshing non rappresenta un'eccezione. La descrizione completa del moto del fluido all'interno di un contenitore parzialmente riempito richiederebbe, in linea di principio, la risoluzione delle equazioni fondamentali della meccanica dei fluidi con condizioni al contorno non banali (superficie libera, dissipazioni, geometria del recipiente, ecc.). Per superare tali difficoltà, la letteratura ha proposto nel tempo modelli analitici semplificati che, pur non ricostruendo in dettaglio l'intero campo di pressione e di velocità, permettono di prevedere il comportamento globale del liquido e di fornire indicazioni utili senza ricorrere necessariamente a campagne sperimentali estese.

In questa cornice si collocano i modelli meccanici equivalenti. Lo scopo di un modello equivalente non è riprodurre puntualmente la distribuzione spaziale delle grandezze fluidodinamiche, bensì descrivere in modo realistico la risposta globale che si riflette sul sistema meccanico incaricato di movimentare il recipiente.
In questo capitolo si ripercorre sinteticamente il procedimento per ricavare i modelli appena citati. Per una trattazione analitica esaustiva si vedano [1] e [10].

2.1 Il modello analitico dello sloshing

L'equazione fondamentale che si considera come punto di partenza nello sloshing e in molti problemi relativi alla meccanica dei fluidi, è l'equazione di Navier.

$$\rho\left[\frac{\partial\overline{\mathbf{q}}}{\partial t}+(\overline{\mathbf{q}}\cdot\nabla)\overline{\mathbf{q}}\right]=-\rho\nabla(gz)-\nabla p+\mu\nabla^2\overline{\mathbf{q}}. \tag{2.1}$$

Sono elencati in [11] diversi modi per ricavarla. Quest'ultima è valida per un elemento di fluido infinitesimo all'interno del dominio considerato. Le grandezze presenti sono le seguenti:

- ρ: densità del fluido;

- g: accelerazione di gravità;
- z: quota del pelo libero;
- p: pressione;
- μ: viscosità dinamica del fluido;
- $\overline{\mathbf{q}} = \overline{\mathbf{q}}(x, y, z, t)$: vettore velocità del fluido rispetto ad un sistema di riferimento fisso.
- $\dfrac{D\overline{\mathbf{q}}}{Dt}$: derivata sostanziale del vettore velocità, pari a

$$\frac{D\overline{\mathbf{q}}}{Dt} = \frac{\partial \overline{\mathbf{q}}}{\partial t} + \left(\overline{\mathbf{q}} \cdot \nabla\right) \overline{\mathbf{q}}. \tag{2.2}$$

Per impostare il problema si parte dalla definizione delle ipotesi semplificative sulla base delle quali viene costruito il modello analitico. Queste ipotesi vengono di seguito riportate:

- contenitore chiuso, assunto rigido e impermeabile (conservazione della massa del sistema);
- fluido incomprimibile, poiché è un liquido; questa ipotesi, unita alla precedente, si traduce matematicamente nella divergenza nulla del vettore velocità:

$$\nabla \cdot \overline{\mathbf{q}} = 0; \tag{2.3}$$

- fluido irrotazionale, che matematicamente si traduce nel rotore nullo del vettore velocità:

$$\nabla \times \overline{\mathbf{q}} = 0; \tag{2.4}$$

- fluido non viscoso:

$$\mu = 0; \tag{2.5}$$

- capillarità e tensioni superficiali trascurate in campo gravitazionale;
- al serbatoio è permesso un moto piano senza rotazioni (ipotesi tipicamente corretta per i moti con cui si ha a che fare nel settore delle macchine automatiche).

Poichè vale l'eq. (2.4) per ipotesi, come [11] ci ricorda, il vettore velocità può essere espresso come il gradiente di una funzione scalare chiamata funzione potenziale di velocità Φ:

$$\overline{\mathbf{q}} = -\nabla\Phi \tag{2.6}$$

dove Φ costituisce la grandezza fisica che descrive il fenomeno dello sloshing. Inserendo la (2.6) nella (2.1) e considerando le ipotesi (2.3), (2.4) ed (2.5) si ottiene la cosiddetta equazione di Laplace:

$$\nabla^2\Phi = 0 \tag{2.7}$$

Quella appena ricavata rappresenta l'equazione differenziale che sta alla base dello studio semplificato del fenomeno dello sloshing: integrandola sotto le opportune condizioni al contorno si ricava l'espressione del potenziale Φ. Da quest'ultima si potranno trovare la forma del pelo libero e le sollecitazioni di forza e momento che il liquido in movimento esercita sulle pareti del contenitore. Il potenziale di velocità può essere espresso come somma di due contributi:

$$\Phi = \Phi_0 + \tilde{\Phi} \tag{2.8}$$

Il termine Φ_0 descrive il moto imposto al contenitore, mentre $\tilde{\Phi}$ rappresenta il potenziale di disturbo, associato al moto relativo del fluido rispetto al recipiente.

2.1.1 Geometria cilindrica e condizioni al contorno

Nel contesto applicativo di interesse si considerano prevalentemente recipienti cilindrici, poiché tale geometria approssima in modo soddisfacente un'ampia classe di contenitori per prodotti liquidi. In virtù della simmetria del problema e della geometria in questione, risulta conveniente esprimere la (2.6) in coordinate polari cilindriche (r,θ,z).

Si assuma un moto piano del recipiente nel piano X_0–Z_0 del sistema di riferimento fisso (O', X_0, Y_0, Z_0) e si introduca un sistema solidale al serbatoio (r,θ,z) con origine O posta alla quota del pelo libero indisturbato. Il tutto è chiarito in Figura 2.1, nella quale è riportata anche l'altezza del pelo libero h rispetto alla condizione indisturbata, valutata in corrispondenza di un generico valore della coordinata radiale r. Si consideri quindi un cilindro di raggio R e altezza di riempimento h. La superficie libera è descritta dalla quota $\eta = \eta(r,\theta,t)$ rispetto alla condizione indisturbata.

Poiché le condizioni al contorno sono imposte in un riferimento solidale al contenitore, esse vengono espresse in funzione del potenziale di disturbo $\tilde{\Phi}$. Nelle loro espressioni compaiono, oltre a ρ e g già definite, le seguenti grandezze:

- σ: tensione superficiale del liquido;
- R_1 e R_2: raggi principali di curvatura della superficie libera;
- $\ddot{X}_0$ e $\ddot{Z}_0$: componenti di accelerazione dell'origine O del sistema solidale al serbatoio, nell'ipotesi di moto nel piano X_0–Z_0 del sistema fisso.

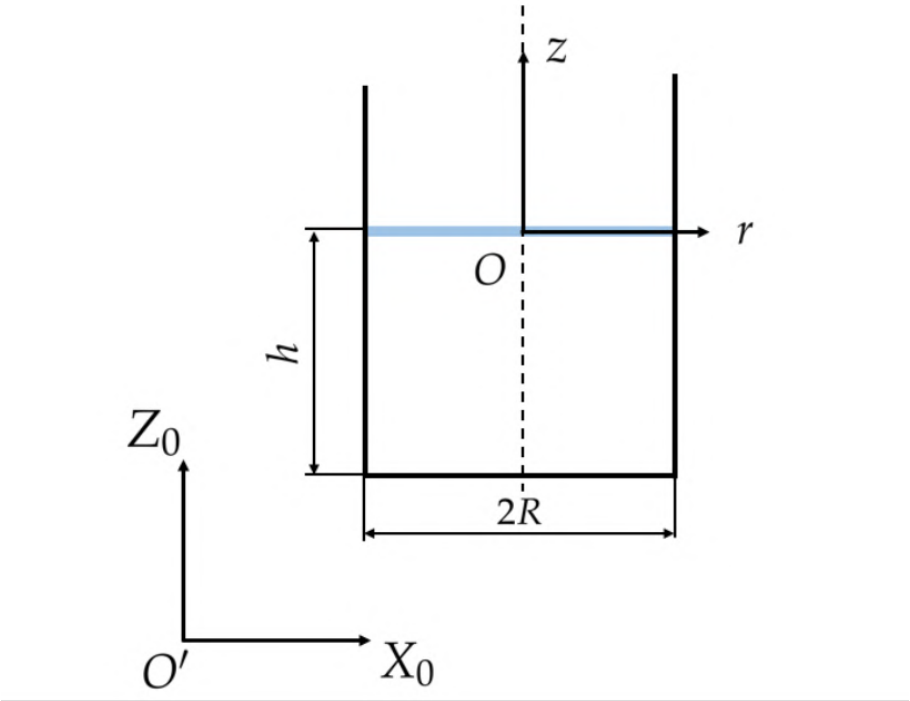


Figura 2.1: Recipiente cilindrico in moto piano lungo gli assi X_0 e Z_0 con altezza del pelo libero h misurata a partire dalla condizione indisturbata.

Le condizioni al contorno, il quale procedimento completo per ottenerle viene rimandato a [10], possono quindi essere riassunte come segue.

- Condizione dinamica al pelo libero (forma generale):

$$\frac{1}{2}\left(\nabla\tilde{\Phi}\cdot\nabla\tilde{\Phi}\right)-\frac{\sigma}{\rho}\left(\frac{1}{R_1}+\frac{1}{R_2}\right)+g\eta-\frac{\partial\tilde{\Phi}}{\partial t}+r\cos(\theta)\,\ddot{X}_0+\eta\,\ddot{Z}_0=0 \tag{2.9}$$

- Condizione cinematica al pelo libero:

$$-\frac{\partial\tilde{\Phi}}{\partial z}=\frac{\partial\eta}{\partial t}-\frac{\partial\tilde{\Phi}}{\partial r}\frac{\partial\eta}{\partial r}-\frac{1}{r^2}\frac{\partial\tilde{\Phi}}{\partial\theta}\frac{\partial\eta}{\partial\theta} \tag{2.10}$$

- Condizioni al contorno sulle pareti del recipiente (impermeabilità):

$$-\left.\frac{\partial\tilde{\Phi}}{\partial r}\right|_{r=R}=0,\qquad -\left.\frac{\partial\tilde{\Phi}}{\partial z}\right|_{z=-h}=0 \tag{2.11}$$

  delle quali la prima è la condizione ai margini, mentre la seconda è la condizione sul fondo.

Nelle relazioni (2.9) e (2.10), il termine $\eta=\eta(r,\theta,t)$ rappresenta la forma istantanea del pelo libero; di conseguenza, la coordinata verticale z viene sostituita con η nella descrizione della superficie libera.

Il problema analitico risulta quindi definito dall'equazione di Laplace (2.7) (scritta per il potenziale di disturbo) con le condizioni (2.9), (2.10) e (2.11). La soluzione del problema, in forma generale, porta a una rappresentazione modale del potenziale di disturbo, nella quale compaiono modi circonferenziali (indice m) e modi radiali (indice n).

2.1.2 Soluzione modale e funzioni di Bessel

Sotto opportune ipotesi di linearizzazione al pelo libero (piccole oscillazioni) e imponendo le condizioni al contorno sulle pareti, il potenziale di disturbo può essere espresso come serie doppia nei modi (m,n):

$$\begin{aligned}\tilde{\Phi}(r,\theta,z,t) = \sum_{m=0}^{\infty}\sum_{n=1}^{\infty}\Big[\alpha_{mn}(t)\cos(m\theta)+\beta_{mn}(t)\sin(m\theta)\Big] \\ \cdot J_m(\lambda_{mn}r)\,\frac{\cosh[\lambda_{mn}(h+z)]}{\cosh(\lambda_{mn}h)}.\end{aligned} \tag{2.12}$$

Qui $J_m(\cdot)$ è la funzione di Bessel di prima specie e ordine m. Il parametro λ_{mn} rappresenta il numero d'onda radiale ed è definito come:

$$\lambda_{mn} = \frac{\xi_{mn}}{R}$$

dove R è il raggio del recipiente cilindrico e ξ_{mn} è l'n-esimo zero della derivata radiale della funzione di Bessel $J'_m(\cdot)$, determinato dalla condizione di impermeabilità sulla parete laterale ($r = R$). In particolare, valgono le radici della seguente equazione:

$$\frac{\partial}{\partial r}J_m\left(\frac{\xi_{mn}}{R}\,r\right)\bigg|_{r=R} = 0 \tag{2.13}$$

L'espressione (2.12) evidenzia che lo sloshing può essere interpretato come un fenomeno di tipo modale (analogo, per certi aspetti, alla modellazione di sistemi continui come travi e piastre). Ne consegue che, in funzione della precisione desiderata, il sistema può essere approssimato considerando un numero finito di modi.

2.1.3 Frequenze naturali del pelo libero

Il primo passo dell'analisi consiste nel determinare le pulsazioni naturali del pelo libero e i corrispondenti modi propri di vibrare (nel seguito indicati come *modi di sloshing*). A tal fine si considera la *risposta libera* del sistema, imponendo l'assenza di eccitazioni esterne: le frequenze naturali e le forme modali dipendono infatti unicamente dalle caratteristiche del recipiente e del fluido, e non dalla specifica legge di eccitazione. Nel caso in esame, tale condizione equivale a porre nulle le accelerazioni imposte al contenitore, poiché la forzante del problema è rappresentata proprio dal moto del recipiente.

Mediante un'analisi modale della risposta libera, si ricavano le pulsazioni naturali associate ai modi di sloshing. L'analisi viene svolta inizialmente trascurando l'effetto della tensione superficiale (coerentemente con le ipotesi riportate, dunque $\sigma = 0$), che potrà essere eventualmente reintrodotto in un secondo momento. La pulsazione naturale del modo (m,n) è:

$$\omega_{mn} = \sqrt{\frac{g\,\xi_{mn}}{R}\tanh\left(\frac{\xi_{mn}}{R}\,h\right)} \tag{2.14}$$

Poiché ξ_{mn} dipende unicamente dagli indici modali ed è determinato dalla (2.13), la (2.14) mostra che la dinamica libera dipende dalla geometria del recipiente, tramite R e h. In molte applicazioni pratiche i contributi dei modi superiori risultano limitati e, per una prima approssimazione ingegneristica, si assume spesso il solo modo fondamentale.

2.1.4 Eccitazione armonica e risposta forzata

Una volta note le pulsazioni naturali, è possibile studiare la risposta forzata del sistema. Si consideri, a titolo esemplificativo, un'eccitazione armonica del recipiente lungo X_0 di ampiezza X e pulsazione Ω:

$$X_0(t) = X \sin(\Omega t) \tag{2.15}$$

A partire dalla forzante (2.15) e dalla rappresentazione modale del potenziale, si può determinare la forma del pelo libero, così come le forze e i momenti che genera il fluido sul contenitore a causa della sua agitazione dovuta alla legge di moto con cui è movimentato il recipiente. Per i passaggi analitici completi e le espressioni dettagliate delle grandezze di interesse si rimanda ai testi di riferimento [1], [2] e [10].

2.2 Modelli meccanici equivalenti

A partire dal modello analitico descritto nella sezione precedente, è possibile costruire un modello meccanico equivalente che riproduca le azioni idrodinamiche scambiate tra fluido e recipiente in forma più maneggevole per l'analisi dinamica e per le successive attività di sintesi/ottimizzazione.

Dal punto di vista degli effetti sulla dinamica complessiva, il modello equivalente e il sistema reale devono risultare tra loro coerenti; tuttavia, la rappresentazione meccanica risulta in genere molto più semplice da analizzare e da utilizzare nelle applicazioni successive. Inoltre, rispetto alla trattazione analitica introduttiva, il modello meccanico consente di includere in modo naturale l'effetto dissipativo associato alla viscosità del fluido (finora trascurato) e di descrivere il comportamento del sistema anche in presenza di forzanti generiche, non necessariamente limitate a una legge di moto armonica come quella assunta in precedenza. In particolare, un modello equivalente è ritenuto adeguato se: (i) presenta la stessa massa e lo stesso momento d'inerzia del sistema reale; (ii) riproduce la posizione del centro di massa per piccole oscillazioni; (iii) conserva le principali caratteristiche modali (pulsazioni naturali e smorzamenti effettivi); (iv) restituisce le stesse forze e coppie scambiate con il recipiente a fronte di sollecitazioni assegnate.

È inoltre utile distinguere tre regimi di moto del fluido durante lo sloshing:

- **regime lineare:** condizione di piccole oscillazioni in cui il pelo libero può essere approssimato come piano; l'oscillazione avviene attorno a un diametro nodale fissato e la risposta resta tipicamente lontana da condizioni di risonanza;

- **regime debolmente non lineare:** la superficie libera assume una forma non più approssimabile con un piano; possono manifestarsi fenomeni come lo sloshing rotante (*rotary sloshing*), in cui l'asse (o diametro) attorno al quale avviene l'oscillazione tende a ruotare, soprattutto in prossimità della risonanza e per ampiezze più elevate;

- **regime altamente non lineare:** si verifica per oscillazioni di grande ampiezza e/o in prossimità della risonanza, con variazioni molto rapide del pelo libero fino a condizioni di impatto sulle pareti, che generano picchi di pressione localizzati.

La complessità del modello equivalente cresce al crescere della non linearità richiesta: modelli più accurati consentono di descrivere una casistica più ampia, ma risultano anche più onerosi da identificare e impiegare.

Nel seguito si riporta dapprima il modello meccanico equivalente *lineare*, richiamato e simulato come riferimento per comprendere la fisica di base del fenomeno e per validare le grandezze modali principali. Successivamente, in continuità con il lavoro proposto in [7] e come sviluppo dell'impostazione già tracciata, l'attenzione viene spostata sulla risposta del modello *non lineare*, con particolare interesse per il caso *debolmente non lineare*, che rappresenta un buon compromesso tra fedeltà descrittiva e gestibilità ai fini dell'ottimizzazione.

2.2.1 Modello meccanico equivalente lineare

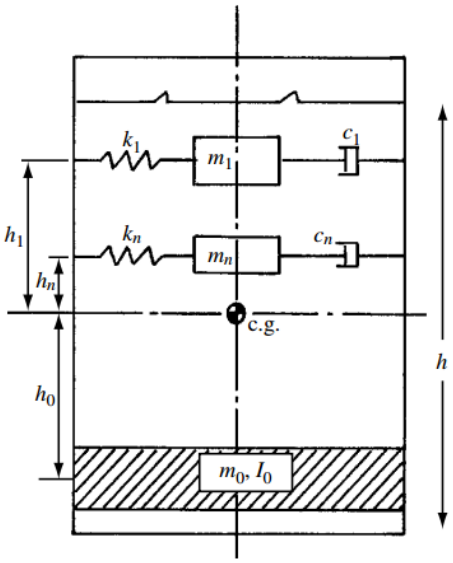


Figura 2.2: Modello meccanico lineare con rigidezze e smorzatori.

Nel modello discreto lineare, rappresentato in Figura 2.2, si considera una massa m_0 rigidamente solidale al recipiente, caratterizzata da un momento d'inerzia rispetto all'asse y (perpendicolare al piano del moto) pari a I_0. La massa m_0 rappresenta la porzione di liquido che segue il moto del contenitore senza partecipare alle oscillazioni di sloshing, mentre le n masse di sloshing modellano la frazione di fluido che oscilla durante la movimentazione del recipiente. Ciascuna massa di sloshing è associata a un modo radiale del sistema; per questo motivo alle masse di sloshing e ai modi radiali è assegnato il medesimo pedice n.

L'accoppiamento tra le masse di sloshing e il recipiente è descritto mediante rigidezze elastiche concentrate e smorzatori viscosi. La massa solidale m_0 è posta a distanza h_0 dal centro di massa del fluido nella configurazione indisturbata, mentre le masse di sloshing $m_1, m_2, \ldots, m_n$ si trovano a distanze $h_1, h_2, \ldots, h_n$.

Di seguito si riportano le condizioni necessarie per la costruzione del modello discreto lineare equivalente, in accordo con le ipotesi generali introdotte in precedenza:

- **conservazione della massa.** La massa totale del fluido deve essere uguale alla somma delle masse presenti nel modello discreto:

$$m_f = m_0 + \sum_{n=1}^{\infty} m_n \tag{2.16}$$

- **conservazione del momento d'inerzia.** Il momento d'inerzia rispetto all'asse y perpendicolare al piano del moto deve coincidere per il modello discreto e per il fluido considerato come un corpo rigido equivalente:

$$I_f = I_0 + m_0 h_0^2 + \sum_{n=1}^{\infty} m_n h_n^2; \tag{2.17}$$

- **coincidenza del centro di massa.** Il centro di massa del modello e quello del fluido reale devono coincidere. Inoltre, nel modello discreto lineare si trascura lo spostamento verticale del baricentro (ipotesi di piccole oscillazioni); ne consegue:

$$m_0 h_0 - \sum_{n=1}^{\infty} m_n h_n = 0; \tag{2.18}$$

- **riproduzione delle pulsazioni naturali.** La rigidezza k_n associata alla massa di sloshing m_n deve generare la corrispondente pulsazione naturale di sloshing ω_n. In particolare, assumendo il primo modo circonferenziale ($m = 1$) e omettendo tale pedice, si impone:

$$\omega_n^2 = \frac{k_n}{m_n}. \tag{2.19}$$

Sotto queste ipotesi si ricava l'equazione del moto che descrive la dinamica dell'n-esima massa di sloshing m_n. Il risultato (ottenibile, ad esempio, tramite equilibrio dinamico oppure con un approccio energetico mediante le equazioni di Lagrange, come mostrato in [10]) può essere scritto come:

$$m_n (\ddot{x}_n + \ddot{x}_0) + c_n \dot{x}_n + k_n x_n = 0. \tag{2.20}$$

Le grandezze x_0 e x_n rappresentano rispettivamente lo spostamento del recipiente rispetto a un sistema di riferimento fisso e lo spostamento della n-esima massa di sloshing rispetto al sistema solidale al contenitore, come schematizzato in Figura 2.3.

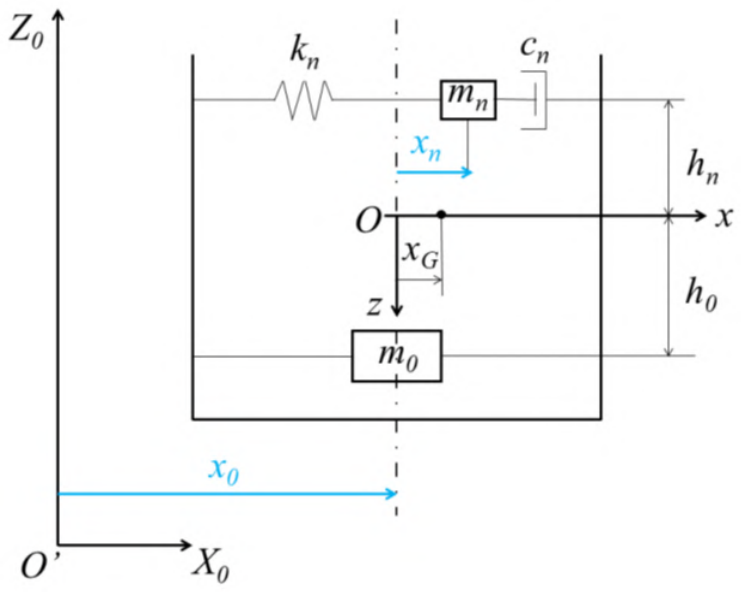


Figura 2.3: Schema del modello meccanico lineare considerando solo la n-esima massa di sloshing.

Il coefficiente viscoso c_n può essere espresso in funzione del coefficiente di smorzamento ζ_n come:

$$c_n = 2m_n\omega_n\zeta_n \tag{2.21}$$

Sostituendo (2.21) nell'equazione del moto (2.20), dividendo per m_n e ricordando la (2.19), si ottiene la forma standard:

$$\ddot{x}_n + 2\zeta_n\omega_n\dot{x}_n + \omega_n^2 x_n = -\ddot{x}_0 \tag{2.22}$$

L'accelerazione $\ddot{x}_0$ agisce dunque come forzante equivalente del sistema e genera lo spostamento della massa m_n. La (2.22) stabilisce quindi il legame tra la legge di moto imposta al recipiente e la risposta della massa di sloshing x_n. Si tratta di un'equazione differenziale lineare del secondo ordine a un grado di libertà nell'incognita x_n: noti i parametri del modello (ζ_n, ω_n) è possibile integrarla numericamente per qualunque profilo di accelerazione in ingresso.

Stima delle masse equivalenti. A questo punto è necessario stimare i parametri del modello discreto. Si ricorda che il modello analitico è stato introdotto assumendo fluido non viscoso; per rendere confrontabili le due rappresentazioni si può inizialmente considerare il caso non smorzato (ovvero $\zeta_n = 0$) ed un'eccitazione armonica analoga alla (2.15). Uguagliando l'espressione della forza scambiata tra fluido e recipiente ricavata dal modello analitico con quella generata dalla n-esima massa di sloshing nel modello discreto, si ottiene l'espressione della massa equivalente m_n:

$$m_n = \frac{m_f}{h\,(\xi_{1n}^2 - 1)}\,\frac{2R}{\xi_{1n}}\,\tanh\!\left(\frac{\xi_{1n}h}{R}\right) \tag{2.23}$$

Qui sono note le grandezze geometriche R e h, la massa totale del liquido m_f (ricavabile, ad esempio, come $m_f = \pi R^2 h\rho$) ed è noto il parametro ξ_{1n}. Per quanto riguarda la pulsazione naturale, essa coincide con quella ottenuta dal modello analitico, data dalla (2.14), considerando il modo fondamentale in direzione circonferenziale ($m = 1$).

In generale non è necessario conoscere tutti i parametri strutturali del modello discreto (m_n, k_n, c_n, m_0) per ricavare una grandezza direttamente confrontabile con la risposta reale. In particolare, la variabile x_n è una coordinata interna del modello equivalente e non è misurabile direttamente. È quindi utile introdurre una grandezza osservabile, quale l'altezza massima di sloshing rispetto alla posizione indisturbata del pelo libero, valutata in corrispondenza del bordo del contenitore e nel piano di eccitazione (si veda Figura 2.4, nella quale è indicata anche l'altezza di riempimento h). Seguendo i passaggi riportati in letteratura, si ottiene:

$$\bar{\eta} = \frac{4}{\pi R^3 \rho} \sum_{n=1}^{N} x_n m_n. \tag{2.24}$$

L'espressione (2.24) deriva dall'uguaglianza tra la posizione del baricentro nel modello discreto (Figura 2.3) e quella del fluido reale assumendo pelo libero di forma planare (Figura 2.4), coerentemente con l'ipotesi di regime lineare. Una volta determinato l'andamento di x_n integrando la (2.22) per una data legge di moto $\ddot{x}_0$, è quindi possibile ricavare la risposta del sistema in termini di altezza di sloshing alla parete applicando la (2.24).

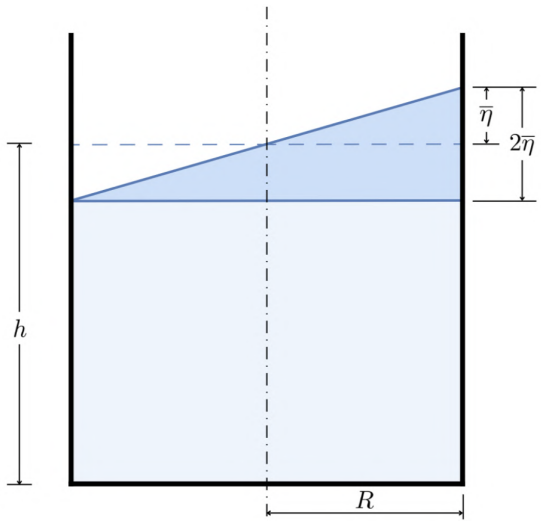


Figura 2.4: Altezza massima di sloshing nel caso di regime di moto lineare (pelo libero planare).

Stima dello smorzamento. Nel modello discreto lineare lo smorzamento è modellato mediante n smorzatori viscosi; il parametro adimensionale associato è ζ_n. Tale parametro può essere stimato sperimentalmente, ad esempio tramite la tecnica del decremento logaritmico, oppure attraverso approcci energetici. In ogni caso, i coefficienti di smorzamento non vengono generalmente determinati in forma chiusa a partire dal solo modello analitico. Il parametro fondamentale connesso alle perdite viscose è la viscosità cinematica ν, definita come rapporto tra viscosità dinamica μ e densità ρ:

$$\nu = \frac{\mu}{\rho} \tag{2.25}$$

Poiché nella trattazione introduttiva è stata assunta l'ipotesi $\mu = 0$, i valori di ζ stimati per via empirica o sperimentale vengono introdotti nel modello discreto equivalente al fine di rappresentare le perdite. Per serbatoi cilindrici sono disponibili in

letteratura [1] e [12], le seguenti espressioni empiriche per il coefficiente di smorzamento:

$$\zeta = 0.79\sqrt{Re_1}\left[1+\frac{0.318}{\sinh\left(1.84\frac{h}{R}\right)}\left(1+\frac{1-\frac{h}{R}}{\cosh\left(1.84\frac{h}{R}\right)}\right)\right] \tag{2.26}$$

$$\zeta = 0.83\sqrt{Re_1}\left\{\tanh\left(1.84\frac{h}{R}\right)\left[1+2\frac{1-\frac{h}{R}}{\cosh\left(3.68\frac{h}{R}\right)}\right]\right\} \tag{2.27}$$

$$\zeta = \frac{2.89}{\pi}\sqrt{Re_1}\left[1+\frac{0.318}{\sinh\left(1.84\frac{h}{R}\right)}\left(1+\frac{1-\frac{h}{R}}{\cosh\left(1.84\frac{h}{R}\right)}\right)\right] \tag{2.28}$$

dove il numero adimensionale Re_1 è definito come:

$$Re_1 = \frac{\nu}{\sqrt{gR^3}} \tag{2.29}$$

Nel seguito, il coefficiente di smorzamento ζ viene calcolato mediante l'espressione (2.26), adottata come riferimento nel modello discreto equivalente. Le altre formulazioni riportate sono incluse per completezza.
Essendo i coefficienti di smorzamento ricavati da correlazioni empiriche, è plausibile che uno dei limiti principali dei modelli equivalenti risieda proprio nella rappresentazione delle perdite energetiche. Tuttavia, nelle applicazioni in cui lo sloshing risulta più rilevante si ha spesso a che fare con liquidi a bassa viscosità; in tali condizioni l'impiego di correlazioni empiriche per ζ_n è generalmente ritenuto adeguato, soprattutto quando l'obiettivo è ridurre lo sloshing mediante opportune leggi di moto.

2.2.2 Modello meccanico equivalente non lineare

Nel caso in cui il sistema operi in un regime di moto *debolmente non lineare*, l'ipotesi di piccole oscillazioni alla base del modello discreto lineare non è più pienamente soddisfatta. In particolare, l'ampiezza del moto del pelo libero può diventare tale da rendere non trascurabili effetti che nel caso lineare erano secondari: variazioni della quota del baricentro del liquido, modifiche della forma della superficie libera e, soprattutto, la comparsa di fenomeni quali lo *rotary sloshing*, tipicamente osservati quando l'eccitazione si colloca in prossimità delle frequenze naturali del sistema. In queste condizioni, un modello equivalente puramente lineare rischia di non descrivere correttamente la risposta, con possibili sottostime (o errate previsioni) delle oscillazioni e delle azioni trasmesse alla struttura.

Per tenere conto di tali effetti, in questo lavoro si adotta un modello meccanico equivalente *non lineare* del tipo proposto in [13]. È costruito in modo tale che per piccole oscillazioni ricada in quello descritto precedentemente. Nel modello si riconoscono una massa fissa m_0 solidale al recipiente, associata a un'inerzia I_0, e un disco centrato nella posizione del baricentro della configurazione indisturbata, privo di massa ma caratterizzato da un'inerzia equivalente e collegato a uno smorzatore viscoso. La parte convettiva del fluido è rappresentata da n masse di sloshing (una per ciascun modo radiale), vincolate a muoversi su una superficie a paraboloide e, al contempo, accoppiate all'asse del cilindro mediante una molla a comportamento non lineare. Per modellare le perdite energetiche è infine necessario introdurre dissipatori viscosi associati alle masse di sloshing.

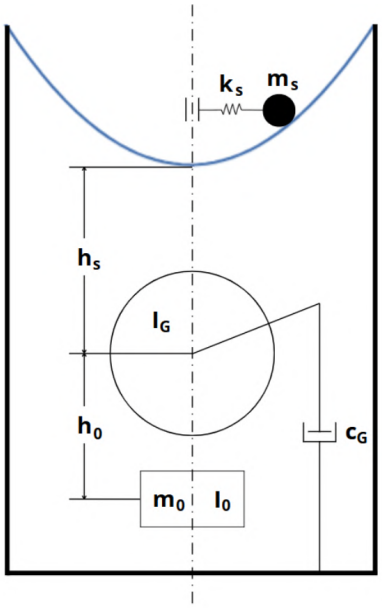


Figura 2.5: Altezza massima di sloshing nel caso di regime di moto non lineare, con pelo libero a profilo parabolico.

Lo schema di riferimento del modello adottato è riportato in Figura 2.5, che verrà assunta come base per lo sviluppo delle equazioni del moto e per la successiva analisi della risposta dinamica.

Nel seguito si considera il *solo modo fondamentale* di sloshing; per coerenza con la notazione del modello lineare si mantiene il pedice n, intendendo implicitamente $n = 1$.

Per esprimere il modello in forma adimensionale e rendere più generale la descrizione della dinamica, si introducono le coordinate normalizzate:

$$\bar{x}_n = \frac{x_n}{R}, \qquad \bar{y}_n = \frac{y_n}{R}, \tag{2.30}$$

dove R è il raggio del recipiente, mentre x_n e y_n rappresentano gli spostamenti della massa di sloshing sul piano orizzontale.

Equazioni del moto (modello completo 2D). Attraverso un approccio energetico, ad esempio mediante equazioni di Lagrange, e svolgendo le integrazioni necessarie sui domini definiti dagli estremi ricavabili dalla Figura 2.6 si ottengono le seguenti equazioni differenziali non lineari nelle variabili adimensionali:

$$\begin{aligned}&\ddot{\bar{x}}_n + 2\omega_n\zeta_n\left[\dot{\bar{x}}_n + C_n^2\left(\bar{x}_n^2\dot{\bar{x}}_n + \bar{y}_n\dot{\bar{y}}_n\bar{x}_n\right)\right] + C_n^2\left(\bar{x}_n\dot{\bar{x}}_n^2 + \bar{x}_n^2\ddot{\bar{x}}_n + \bar{x}_n\dot{\bar{y}}_n^2 + \bar{x}_n\ddot{\bar{y}}_n\bar{y}_n\right)\\ &\quad + \omega_n^2\bar{x}_n\left[1 + \alpha_n\left(\bar{x}_n^2 + \bar{y}_n^2\right)^{w-1}\right] + \frac{\ddot{x}_0}{R} = 0,\end{aligned} \tag{2.31}$$

$$\begin{aligned}&\ddot{\bar{y}}_n + 2\omega_n\zeta_n\left[\dot{\bar{y}}_n + C_n^2\left(\bar{y}_n^2\dot{\bar{y}}_n + \bar{x}_n\dot{\bar{x}}_n\bar{y}_n\right)\right] + C_n^2\left(\bar{y}_n\dot{\bar{y}}_n^2 + \bar{y}_n^2\ddot{\bar{y}}_n + \bar{y}_n\dot{\bar{x}}_n^2 + \bar{y}_n\ddot{\bar{x}}_n\bar{x}_n\right)\\ &\quad + \omega_n^2\bar{y}_n\left[1 + \alpha_n\left(\bar{x}_n^2 + \bar{y}_n^2\right)^{w-1}\right] = 0,\end{aligned} \tag{2.32}$$

dove ω_n e ζ_n sono rispettivamente la pulsazione naturale e il rapporto di smorzamento associati al modo fondamentale e definiti in modo coerente con il modello lineare. Nel presente lavoro si considera un moto puramente traslazionale lungo

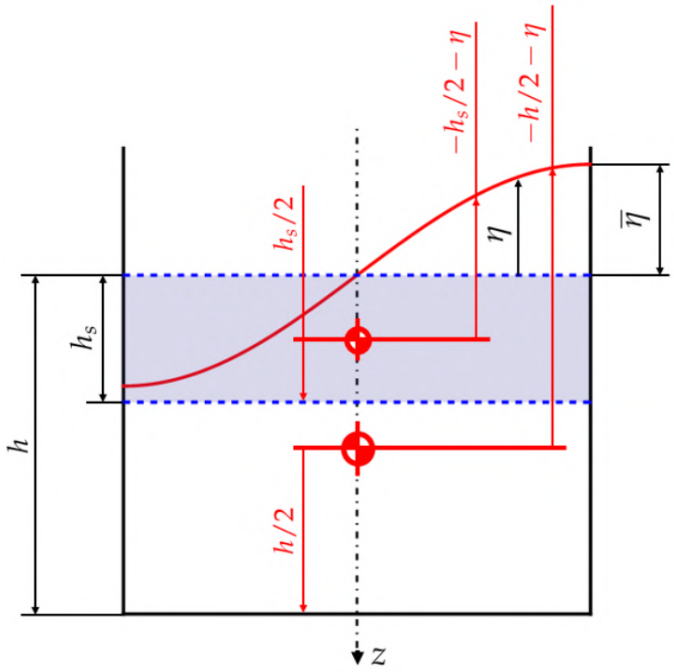


Figura 2.6: Rappresentazione dell'altezza di sloshing nel modello non lineare.

l'asse x (1 g.d.l.), per cui $y_0(t) \equiv 0$ e quindi $\ddot{y}_0 = 0$: il termine di eccitazione trasversale $\ddot{y}_0/R$ nella (2.32) risulta nullo e viene omesso nelle simulazioni. Inoltre, il parametro w è un esponente che regola il grado di non linearità del termine di richiamo; nella trattazione verrà considerato $w = 2$, come nel modello di riferimento, per cui $(\bar{x}_n^2 + \bar{y}_n^2)^{w-1} = (\bar{x}_n^2 + \bar{y}_n^2)$. Si definisce poi:

$$C_n = \frac{\omega_n^2 R}{g}, \tag{2.33}$$

e α_n, che è una costante adimensionale che regola l'entità della non linearità della molla, tipicamente compresa tra $\frac{1}{2}$ e $\frac{2}{3}$ nel modello di riferimento. In [14], così come in [15], si osserva che, ponendo $\alpha_n = 0$, si ottiene una stima più conservativa dell'altezza massima di sloshing. Nel presente lavoro, però, si mantiene $\alpha_n \neq 0$ in coerenza con il modello di riferimento. In questa maniera, si aggiunge un ulteriore termine di richiamo che può irrigidire la dinamica equivalente e, in alcuni casi, portare a una stima meno conservativa di $\eta_{\max}$ rispetto al caso $\alpha_n = 0$.

Riduzione al moto unidimensionale. In molte applicazioni industriali è utile semplificare il modello assumendo trascurabile il rotary sloshing; in tal caso il moto resta piano nel piano di eccitazione e non vi è eccitazione lungo y. Ponendo quindi $\bar{y}_n \equiv 0$ e annullando i termini associati, la (2.31) si riduce alla seguente equazione scalare:

$$\ddot{\bar{x}}_n + 2\omega_n\zeta_n\left(\dot{\bar{x}}_n + C_n^2\bar{x}_n^2\dot{\bar{x}}_n\right) + C_n^2\left(\bar{x}_n\dot{\bar{x}}_n^2 + \bar{x}_n^2\ddot{\bar{x}}_n\right) + \omega_n^2\bar{x}_n\left(1 + \alpha_n\bar{x}_n^2\right) + \frac{\ddot{x}_0}{R} = 0. \tag{2.34}$$

Nel seguito ci si concentra sul modo fondamentale di sloshing, che risulta dominante nelle condizioni operative di interesse. Pertanto, ponendo $n = 1$ e introducendo i parametri corrispondenti $(\omega_1, \zeta_1, C_1, \alpha_1)$, l'eq. (2.34) si riscrive come:

$$\ddot{\bar{x}}_1 + 2\omega_1\zeta_1\left(\dot{\bar{x}}_1 + C_1^2\bar{x}_1^2\dot{\bar{x}}_1\right) + C_1^2\left(\bar{x}_1\dot{\bar{x}}_1^2 + \bar{x}_1^2\ddot{\bar{x}}_1\right) + \omega_1^2\bar{x}_1\left[1 + \alpha_1\bar{x}_1^2\right] + \frac{\ddot{x}_0}{R} = 0, \tag{2.35}$$

che verrà adottata come modello di riferimento nelle sezioni successive.

Stima dell'altezza di sloshing. Come già discusso nel caso lineare, la coordinata del modello equivalente non coincide direttamente con una misura fisica del pelo libero. Per ottenere una quantità osservabile, si può stimare l'altezza massima di sloshing al bordo del recipiente. Nel modello di riferimento, seguendo i passaggi riportati in [10] e concentrandosi nuovamente sul modo fondamentale di sloshing (n=1), si ottiene la seguente espressione coerentemente alla forma esplicitata in [8]:

$$\overline{\eta} = \frac{\xi_{11}^2 \, x_1 \, m_1}{\pi R^3 \rho}, \tag{2.36}$$

dove $x_1(t)$ è la coordinata generalizzata del modello equivalente associata al modo fondamentale ($n = 1$), m_1 è la corrispondente massa modale, R è il raggio del recipiente, ρ la densità del liquido e ξ_1 è il parametro adimensionale del primo modo (legato al numero d'onda $k_1 = \xi_1 / R$). In questo modo, nota la legge di moto $\ddot{x}_0(t)$, l'integrazione numerica della (2.34) consente di ricostruire $\bar{x}_1(t)$ e quindi di stimare l'andamento di $\overline{\eta}(t)$ tramite (2.36).

Capitolo 3

Ottimizzazione di leggi di moto definite a tratti

3.1 Leggi di moto per l'automazione industriale

La pianificazione della traiettoria rappresenta un aspetto centrale nel controllo del moto di macchine automatiche e robot industriali. L'obiettivo è generare una funzione temporale $s(t)$ (posizione o una coordinata generalizzata equivalente) che realizzi lo spostamento richiesto garantendo il rispetto dei limiti cinematici e dinamici del sistema (ad esempio vincoli su velocità, accelerazione, jerk e coppie), oltre ai requisiti di qualità del moto e di sollecitazione meccanica. Nel controllo del moto di macchine automatiche e robot industriali, la variabile che eccita il fenomeno dello *sloshing* non è una forza applicata direttamente al fluido, ma l'accelerazione imposta al recipiente. In altre parole, la *legge di moto* del sistema costituisce la forzante che determina la risposta dinamica del pelo libero (cfr. Capitolo 2). Da un punto di vista ingegneristico, risulta quindi naturale affrontare la mitigazione dello sloshing agendo sulla parametrizzazione del profilo di moto, mantenendo però forme compatibili con l'implementazione tipica dei controllori industriali.

In letteratura esiste un'ampia varietà di profili di moto (polinomiali, cicloidali, B-spline, leggi a jerk minimo, profili time-optimal, ecc.). In questo lavoro si sceglie di analizzare tre famiglie rappresentative (**Poly5**, **trapezia** e **trapezoidale**) per tre motivi principali:

- sono largamente adottate in ambito industriale e nella pianificazione di traiettorie [16];
- rappresentano un buon compromesso tra regolarità del moto (continuità di velocità/accelerazione/jerk) e semplicità di implementazione;
- consentono una parametrizzazione compatta, elemento essenziale quando l'obiettivo è ottimizzare pochi parametri mantenendo invariata la "forma" della legge di moto.

Legge polinomiale quintica (Poly5). La legge polinomiale di quinto ordine è una scelta classica per moti *rest-to-rest* in quanto permette di imporre condizioni al contorno su posizione, velocità e accelerazione agli estremi dell'intervallo. Indicando con

T la durata del moto e con Δs lo spostamento totale, la posizione all'istante t è:

$$s(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5, \tag{3.1}$$

con vincoli $s(0) = 0,\ s(T) = \Delta s,\ \dot{s}(0) = \dot{s}(T) = 0,\ \ddot{s}(0) = \ddot{s}(T) = 0$. Imponendo le condizioni al contorno appena esplicitate e calcolando analiticamente i coefficienti a_i (con $i = 0, \ldots, 6$) presenti nell'eq. 3.1, si ottiene la seguente:

$$s(t) = \Delta s \left[10 \left(\frac{t}{T} \right)^3 - 15 \left(\frac{t}{T} \right)^4 + 6 \left(\frac{t}{T} \right)^5 \right], \tag{3.2}$$

da cui velocità e accelerazione seguono per derivazione.

Legge trapezia standard (accelerazione trapezia). La legge trapezia standard è molto diffusa perché semplice e facilmente parametrizzabile. Dal punto di vista dell'azionamento, essa consente di sfruttare in modo esplicito le prestazioni massime del sistema nei tratti a *accelerazione costante* e a *velocità costante*: variando le durate dei segmenti è possibile "spingere" intenzionalmente il moto verso i limiti di accelerazione e velocità imposti dal motore, mantenendo al contempo una struttura compatibile con l'implementazione tipica dei controllori industriali. Per confronto, nei profili polinomiali i massimi di velocità e accelerazione invece, non sono direttamente imposti tratto per tratto, ma emergono dalla forma globale del polinomio; ciò riduce il controllo sulla saturazione dei vincoli e rende meno immediata la regolazione del compromesso tra tempo ciclo e sollecitazione dinamica.
La traiettoria in questione, prende il nome dall'andamento dell'accelerazione, che ha la forma *trapezoidale* mostrata in Figura 3.1: rampe lineari e tratti a valore costante. In questa configurazione, l'accelerazione è continua a tratti, mentre il jerk risulta costante nei tratti di rampa e presenta discontinuità agli istanti di commutazione. Il vantaggio principale è che la legge si descrive con pochi parametri temporali e si presta bene all'ottimizzazione dei tratti.

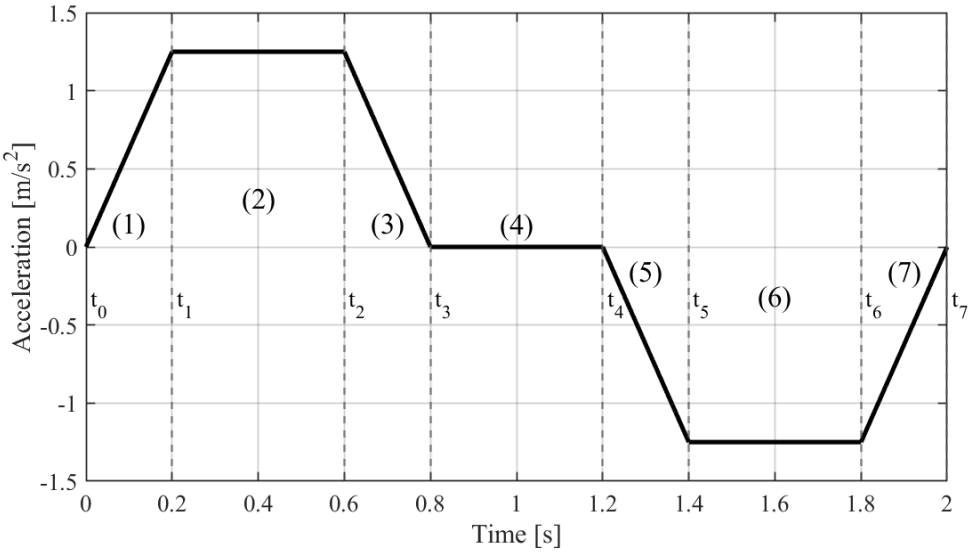


Figura 3.1: Profilo di accelerazione a sette segmenti per una legge trapezia standard (rampe lineari e tratti a valore costante). I tempi $t_0, \ldots, t_7$ delimitano i sette intervalli.

Legge trapezoidale raccordata (trapezoidale con raccordi sinusoidali). In molte applicazioni industriali è desiderabile che la legge di moto sia continua non solo nell'accelerazione, ma anche nel jerk, in modo da ridurre le oscillazioni strutturali dei meccanismi movimentati. Per questo motivo si adottano profili raccordati (ad esempio con raccordi sinusoidali o polinomiali) nelle fasi di variazione dell'accelerazione, ottenendo un andamento più "dolce" e, in pratica, un jerk limitato. La Figura 3.2 mostra una variante raccordata (trapezoidale) che, oltre a limitare il jerk, riduce anche il contenuto ad alta frequenza della forzante, con benefici in termini di eccitazione di modi strutturali e vibrazioni. Questa famiglia è particolarmente utile quando i limiti di jerk e le vibrazioni strutturali sono dominanti [16]. Nel presente lavoro viene considerata come riferimento metodologico e possibile estensione futura; in prospettiva, infatti, a parità di schema "a segmenti" si potrebbero ottimizzare non solo le durate dei tratti principali, ma anche quelle dei raccordi, migliorando ulteriormente la robustezza in presenza di vincoli stringenti sul jerk e di fenomeni vibratori. Tuttavia, si è scelto di concentrare l'ottimizzazione esclusivamente sulla legge trapezia standard, in quanto per quest'ultima, come verrà mostrato nella Sezione 4.2.2, è possibile ricavare in forma analitica lo spettro in funzione delle durate dei sette tratti ΔT_i, con $i = 1, \ldots, 7$, ottenendo così una formulazione particolarmente adatta al successivo processo di ottimizzazione.

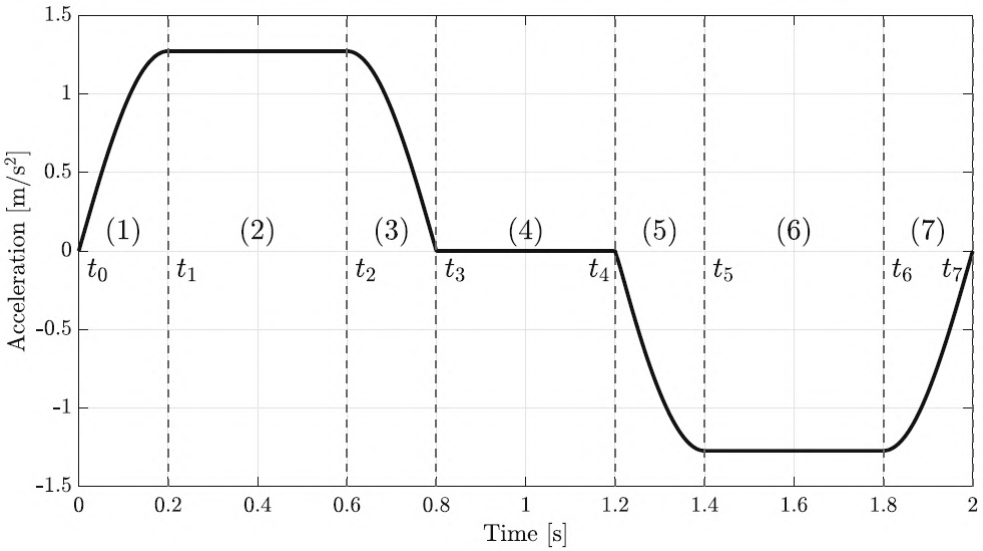


Figura 3.2: Esempio di profilo di accelerazione a sette segmenti con raccordi (legge trapezoidale raccordata). Rispetto alla trapezia standard, i tratti di transizione risultano più regolari e con jerk limitato.

La selezione di queste tre famiglie (Poly5, trapezia, trapezoidale raccordata) consente quindi di coprire tre livelli di complessità crescente—da un profilo polinomiale globale, a un profilo a tratti con parametri temporali espliciti, fino a una variante più regolare—senza disperdere l'analisi su un numero eccessivo di leggi alternative. Inoltre, queste famiglie sono sufficientemente generali da costituire un riferimento per estensioni successive del metodo.

3.2 Parametrizzazione a sette segmenti e collegamento con l'ottimizzazione

Come accennato in precedenza, nel presente lavoro l'attenzione viene posta sulla trapezia standard per mantenere una parametrizzazione minimale e mettere in evidenza il contributo dell'ottimizzazione basata sul modello non lineare di sloshing. Nel seguito, infatti, si considera una legge di accelerazione definita a tratti su un intervallo temporale $[t_0, t_7]$ (moto *rest-to-rest*), suddiviso in sette segmenti delimitati dagli istanti $t_0 < t_1 < \cdots < t_7$. Tale scelta è motivata dall'esigenza di disporre di una parametrizzazione sufficientemente flessibile da soddisfare vincoli cinematici di macchina e, al contempo, abbastanza compatta da rendere stabile l'ottimizzazione.

È conveniente introdurre come variabili temporali le durate dei segmenti:

$$\Delta T_i = t_i - t_{i-1}, \qquad i = 1, \dots, 7, \tag{3.3}$$

con i vincoli di consistenza:

$$\Delta T_i > 0, \qquad \sum_{i=1}^{7} \Delta T_i = T, \tag{3.4}$$

dove T è il tempo totale di esecuzione del moto.

Nel caso della legge trapezia standard in accelerazione (Figura 3.1), i sette segmenti corrispondono alle seguenti fasi:

1. **Rampa di accelerazione positiva** (ΔT_1): $a(t)$ cresce linearmente da 0 ad $a_{\max}$;
2. **Accelerazione positiva costante** (ΔT_2): $a(t) = a_{\max}$;
3. **Rampa verso zero** (ΔT_3): $a(t)$ decresce linearmente da $a_{\max}$ a 0;
4. **Tratto a accelerazione nulla** (ΔT_4): $a(t) = 0$ (velocità costante);
5. **Rampa di accelerazione negativa** (ΔT_5): $a(t)$ decresce linearmente da 0 ad $a_{\min}$;
6. **Accelerazione negativa costante** (ΔT_6): $a(t) = a_{\min}$;
7. **Rampa di ritorno a zero** (ΔT_7): $a(t)$ cresce linearmente da $a_{\min}$ a 0.

Una volta definito il profilo $a(t;\mathbf{x})$ in funzione delle durate $\mathbf{x} = [\Delta T_1, \dots, \Delta T_7]^T$, la velocità e la posizione si ottengono per integrazione:

$$v(t) = v(t_0) + \int_{t_0}^{t} a(\tau;\mathbf{x})\, d\tau, \qquad s(t) = s(t_0) + \int_{t_0}^{t} v(\tau)\, d\tau, \tag{3.5}$$

imponendo le condizioni al contorno tipiche dei moti industriali rest-to-rest:

$$v(t_0) = 0, \quad v(t_7) = 0, \qquad s(t_0) = 0, \quad s(t_7) = \Delta s. \tag{3.6}$$

Questa parametrizzazione è alla base delle procedure di ottimizzazione sviluppate nel progetto: i sette intervalli ΔT_i costituiscono le variabili decisionali, mentre obiettivo e vincoli dipendono dalla legge di moto risultante. La flessibilità della legge risiede

nella possibilità di variare la durata di ciascun tratto per soddisfare vincoli cinematici (ad esempio limiti su velocità e accelerazione massime) e, contemporaneamente, ottimizzare la risposta del sistema rispetto al modello di sloshing *non lineare* introdotto nel Capitolo 2. In questo modo si prosegue e si estende l'impostazione proposta in [7], portando l'ottimizzazione su un modello in grado di descrivere anche effetti non lineari nel range operativo di interesse [10]. Tale impostazione è inoltre coerente con l'implementazione numerica adottata negli strumenti di calcolo sviluppati per l'ottimizzazione, dove $\mathbf{x} = [\Delta T_1, \ldots, \Delta T_7]^T$ rappresenta la variabile da ottimizzare.

3.3 Formulazione del problema di ottimizzazione

Volendo dare una definizione generale dal punto di vista formale, un problema di ottimizzazione consiste nel determinare un vettore di variabili decisionali $\mathbf{x} \in \mathbb{R}^n$ che minimizzi (o massimizzi) una funzione scalare $f(\mathbf{x})$, detta *funzione obiettivo* o *funzione costo*. Tale funzione rappresenta una misura quantitativa della prestazione del sistema e dipende dai parametri che si intendono determinare. Nelle applicazioni ingegneristiche di interesse, le variabili non sono in genere libere di assumere qualunque valore, ma devono soddisfare un insieme di condizioni che esprimono requisiti fisici, geometrici o prestazionali. Si introduce quindi un insieme *ammissibile* (o *feasible set*) all'interno del quale è consentita la ricerca della soluzione ottima.

Una formulazione generale di problema vincolato [17] può essere espressa come:

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n} \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{g}(\mathbf{x}) = \mathbf{0}, \\ & \mathbf{h}(\mathbf{x}) \leq \mathbf{0}, \end{aligned} \tag{3.7}$$

dove $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^m$ raccoglie i vincoli di uguaglianza e $\mathbf{h} : \mathbb{R}^n \to \mathbb{R}^p$ i vincoli di disuguaglianza. L'insieme dei punti che soddisfano simultaneamente tutti i vincoli definisce la regione ammissibile. La geometria di un problema vincolato e l'interpretazione delle condizioni di ottimalità sono schematizzate in Figura 3.3.

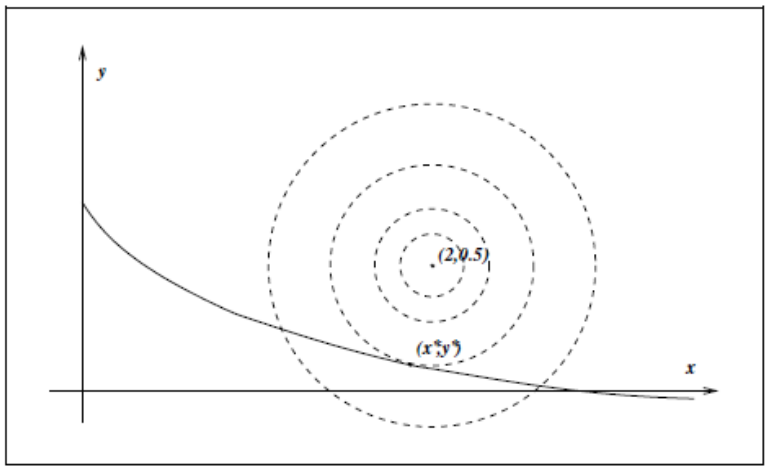


Figura 3.3: Rappresentazione geometrica di un problema di ottimizzazione vincolata: curve di livello dell'obiettivo e vincolo attivo al punto ottimo [17].

Nel caso di interesse di questo lavoro, l'obiettivo e/o i vincoli dipendono in modo non lineare dalle variabili decisionali; ciò conduce a un problema di *ottimizzazione*

non lineare (NLP, Nonlinear Programming), che in generale può risultare *non convesso*. In tali condizioni possono esistere più minimi locali e la soluzione numerica ottenuta da un algoritmo iterativo tende tipicamente a convergere verso un minimo locale, dipendente anche dalla formulazione del problema e dalla scelta dell'inizializzazione (initial guess).

3.3.1 Problema specifico (NLP): direzioni ammissibili e stazionarietà

La ricerca delle durate ottimali dei segmenti della legge di moto può essere formulata come un NLP. In forma compatta, un problema di minimizzazione con vincoli di uguaglianza e disuguaglianza si scrive come mostrato nell'eq. 3.7, dove $\mathbf{x}$ è il vettore delle variabili decisionali.

Nel caso in esame, le variabili decisionali sono le durate dei sette segmenti:

$$\mathbf{x} = [\Delta T_1, \Delta T_2, \ldots, \Delta T_7]^T, \tag{3.8}$$

(eventualmente ridotte a sei variabili imponendo un vincolo di dipendenza, ad esempio tramite il vincolo di somma sul tempo totale).

Nei problemi di ottimizzazione è utile richiamare brevemente il concetto di *direzione ammissibile*. Nel caso non vincolato, una condizione necessaria di minimo locale è $\nabla f(\mathbf{x}^\star) = \mathbf{0}$: se il gradiente non fosse nullo, esisterebbe una direzione in cui f diminuisce e sarebbe sempre possibile effettuare un piccolo passo in quella direzione migliorando il valore dell'obiettivo, in contraddizione con l'ipotesi di ottimalità locale.

Quando sono presenti vincoli, questo ragionamento non è più immediato perché le variabili non possono variare liberamente: gli spostamenti sono consentiti solo se mantengono (almeno localmente) il punto all'interno dell'insieme ammissibile. Di conseguenza, non tutte le direzioni nello spazio delle variabili sono *percorribili*: una direzione è detta ammissibile se, per un passo sufficientemente piccolo, non viola i vincoli di uguaglianza e non "esce" dalla regione definita dai vincoli di disuguaglianza.

Si consideri una perturbazione di $\mathbf{x}^\star$ nella direzione $\mathbf{d}$:

$$\mathbf{x} = \mathbf{x}^\star + \varepsilon \mathbf{d}, \qquad \varepsilon > 0 \text{ piccolo a piacere}, \tag{3.9}$$

e si analizzi l'effetto sui vincoli come in [17].

Vincoli di uguaglianza. Per un vincolo $g_i(\mathbf{x}) = 0$, uno sviluppo di Taylor al primo ordine fornisce:

$$g_i(\mathbf{x}^\star + \varepsilon \mathbf{d}) \simeq g_i(\mathbf{x}^\star) + \varepsilon \nabla g_i(\mathbf{x}^\star)^T \mathbf{d}. \tag{3.10}$$

Poiché $\mathbf{x}^\star$ è ammissibile, $g_i(\mathbf{x}^\star) = 0$. Affinché la perturbazione resti sul vincolo (al primo ordine) deve quindi valere:

$$\nabla g_i(\mathbf{x}^\star)^T \mathbf{d} = 0, \qquad i = 1, \ldots, m. \tag{3.11}$$

In altre parole, $\mathbf{d}$ deve essere tangente all'insieme ammissibile definito dai vincoli di uguaglianza.

Vincoli di disuguaglianza e vincoli attivi. Considerato un vincolo di disuguaglianza $h_j(\mathbf{x}) \leq 0$, si distinguono due casi:

- **Vincolo inattivo:** se $h_j(\mathbf{x}^\star) < 0$, il vincolo non è saturo in $\mathbf{x}^\star$ e non limita localmente le direzioni ammissibili.

- **Vincolo attivo:** se $h_j(\mathbf{x}^\star) = 0$, il vincolo è saturo in $\mathbf{x}^\star$ e restringe le direzioni ammissibili. Linearizzando in $\mathbf{x}^\star$, per una perturbazione $\mathbf{x} = \mathbf{x}^\star + \varepsilon\mathbf{d}$ con $\varepsilon > 0$ piccolo:
$$h_j(\mathbf{x}^\star + \varepsilon\mathbf{d}) \simeq h_j(\mathbf{x}^\star) + \varepsilon\nabla h_j(\mathbf{x}^\star)^T\mathbf{d} = \varepsilon\nabla h_j(\mathbf{x}^\star)^T\mathbf{d}.$$
Affinché la perturbazione resti ammissibile deve quindi valere:
$$\nabla h_j(\mathbf{x}^\star)^T\mathbf{d} \leq 0, \qquad \forall\, j \in \mathscr{A}(\mathbf{x}^\star),$$
dove $\mathscr{A}(\mathbf{x}^\star)$ è l'insieme dei vincoli attivi in $\mathbf{x}^\star$.

Condizione necessaria di primo ordine per un minimo locale. Un punto $\mathbf{x}^\star$ è candidato minimo locale vincolato se, tra tutte le direzioni ammissibili, non esiste alcuna direzione lungo cui f diminuisca. In forma differenziale, ciò implica che la derivata direzionale sia non negativa per ogni direzione ammissibile:
$$\nabla f(\mathbf{x}^\star)^T\mathbf{d} \geq 0 \qquad \forall\, \mathbf{d} \text{ ammissibile.} \tag{3.12}$$

Questo risultato conduce al fatto che il gradiente dell'obiettivo, nel punto ottimo, deve poter essere espresso come combinazione lineare dei gradienti dei vincoli attivi (uguaglianze e disuguaglianze attive). Tale osservazione motiva l'introduzione della Lagrangiana e dei moltiplicatori di Lagrange.

3.3.2 Lagrangiana e condizioni di Karush–Kuhn–Tucker

Introducendo i moltiplicatori di Lagrange $\boldsymbol{\lambda} \in \mathbb{R}^m$ per i vincoli di uguaglianza e $\boldsymbol{\mu} \in \mathbb{R}^p$ per i vincoli di disuguaglianza, si definisce la Lagrangiana [17]:
$$\mathscr{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T\mathbf{g}(\mathbf{x}) + \boldsymbol{\mu}^T\mathbf{h}(\mathbf{x}). \tag{3.13}$$

dove λ_i e μ_j sono i moltiplicatori associati rispettivamente ai vincoli di uguaglianza e di disuguaglianza.

Significato della stazionarietà vincolata. Nel caso non vincolato, l'ottimalità locale richiede che il gradiente dell'obiettivo sia nullo. In presenza di vincoli, invece, l'intuizione geometrica è che, nel punto ottimo, non sia possibile ridurre ulteriormente f muovendosi lungo direzioni ammissibili. Questo si traduce nel fatto che il gradiente di f deve essere "bilanciato" dai gradienti dei vincoli che risultano effettivamente limitanti nel punto considerato: l'ottimo non è caratterizzato da $\nabla f = \mathbf{0}$, ma da una condizione di equilibrio tra obiettivo e vincoli.

Vincoli attivi e moltiplicatori. I moltiplicatori possono essere interpretati anche come indicatori di sensibilità: misurano quanto varia il valore ottimo dell'obiettivo a fronte di una piccola variazione (rilassamento) dei vincoli. In particolare, per le disuguaglianze, il moltiplicatore associato è diverso da zero solo quando il vincolo è attivo (cioè "sul bordo" dell'insieme ammissibile); al contrario, vincoli inattivi non influenzano localmente la soluzione e hanno moltiplicatore nullo.

Sotto opportune ipotesi di regolarità sui vincoli (ad esempio una *constraint qualification*), le condizioni di Karush–Kuhn–Tucker (KKT) costituiscono condizioni necessarie di ottimalità locale per un problema di ottimizzazione non lineare vincolato.

Nel punto ottimo regolare, le condizioni di primo ordine (KKT) includono:

$$\nabla_{\mathbf{x}} \mathscr{L}(\mathbf{x}^\star, \boldsymbol{\lambda}^\star, \boldsymbol{\mu}^\star) = \mathbf{0}, \tag{3.14}$$

$$\mathbf{g}(\mathbf{x}^\star) = \mathbf{0}, \tag{3.15}$$

$$\mathbf{h}(\mathbf{x}^\star) \leq \mathbf{0}, \tag{3.16}$$

$$\boldsymbol{\mu}^\star \geq \mathbf{0}, \tag{3.17}$$

$$\boldsymbol{\mu}^\star \circ \mathbf{h}(\mathbf{x}^\star) = \mathbf{0}, \tag{3.18}$$

dove $\circ$ indica il prodotto elemento per elemento. In particolare, la condizione di complementarità impone che un vincolo di disuguaglianza sia *attivo* solo se il corrispondente moltiplicatore è positivo, mentre se il vincolo è *inattivo* il moltiplicatore associato è nullo. Il sistema KKT rappresenta il riferimento teorico per la costruzione dei principali metodi numerici per NLP vincolati.

3.3.3 IPOPT e metodi interior-point primal–dual

Per la risoluzione numerica di problemi di ottimizzazione non lineare vincolata (NLP) di grandi dimensioni è comune impiegare metodi *interior-point primal–dual*, come IPOPT. L'idea di base è trasformare i vincoli di disuguaglianza in condizioni "interne" all'insieme ammissibile, introducendo variabili di scarto (*slack*) e una funzione di barriera logaritmica. In questo modo si ottiene una successione di sottoproblemi *lisci* (cioè con funzioni obiettivo e vincoli almeno C^1, quindi derivabili in modo continuo [16]), per i quali è possibile applicare efficientemente un metodo di Newton. Nel linguaggio *primal–dual*, le *variabili primali* sono le variabili del problema originale (e, se presenti, le variabili di slack), mentre le *variabili duali* sono i moltiplicatori di Lagrange associati ai vincoli (moltiplicatori per vincoli di uguaglianza e di disuguaglianza). Il *parametro di barriera* $\mu > 0$ pesa il termine di barriera (logaritmico) e viene ridotto progressivamente: al diminuire di μ, le soluzioni dei sottoproblemi si avvicinano alla soluzione del NLP originale.
Da un punto di vista operativo, IPOPT risolve ad ogni iterazione un sistema lineare ottenuto linearizzando le condizioni di ottimalità (KKT) del problema con barriera, calcolando così una direzione di aggiornamento in stile Newton (ossia una correzione locale per primali e duali). In sintesi, a ogni iterazione:

1. costruisce i residui del sistema di ottimalità *primal–dual* al punto corrente (residui KKT, nella formulazione con barriera);

2. linearizza tali residui e risolve il sistema lineare associato per ottenere una direzione di aggiornamento in stile Newton;

3. applica una strategia di *globalizzazione* per determinare una lunghezza di passo α che garantisca avanzamento e robustezza;

4. aggiorna le variabili primali e duali e riduce progressivamente il parametro di barriera, verificando i criteri di convergenza.

Questa prospettiva chiarisce perché, in un solver primal–dual, la disponibilità di derivate accurate sia cruciale: il metodo di Newton è tanto più stabile ed efficiente quanto più fedeli sono le approssimazioni locali del sistema KKT, che dipendono direttamente dalla Jacobiana e dalla Hessiana. Per questo motivo, in un solver IPOPT è essenziale disporre delle derivate coerenti delle funzioni che definiscono il problema, in particolare:

- il gradiente dell'obiettivo $\nabla f(\mathbf{x})$;
- la Jacobiana dei vincoli $J_g(\mathbf{x}) = \nabla \mathbf{g}(\mathbf{x})^T$ (e analogamente per $\mathbf{h}$);
- la Hessiana della Lagrangiana $\nabla^2_{\mathbf{xx}} \mathscr{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$.

Ne consegue che la qualità delle derivate ha un impatto diretto su robustezza e velocità di convergenza del solver.
Nel contesto della pianificazione/ottimizzazione di traiettorie, l'impiego di IPOPT risulta particolarmente vantaggioso perché consente di gestire in modo simultaneo molti vincoli (cinematici e dinamici) e di sfruttare la struttura sparsa di Jacobiana e Hessiana, tipica dei problemi discretizzati nel tempo. D'altra parte, trattandosi in generale di problemi non convessi, la convergenza è tipicamente verso un *minimo locale*. La soluzione finale può quindi dipendere dall'inizializzazione (initial guess) e dalla scalatura delle variabili: una scelta coerente di tali elementi tende a migliorare stabilità numerica e ripetibilità della soluzione. In questo contesto risulta naturale ricorrere a un framework che consenta di definire obiettivo e vincoli in modo compatto e, soprattutto, di fornire automaticamente derivate accurate e coerenti (gradienti, Jacobiana e Hessiana della Lagrangiana) al solutore interior-point: questo è uno dei motivi principali dell'impiego di CasADi nel presente lavoro.

3.4 Risoluzione numerica con CasADi

CasADi è un framework progettato per modellare problemi di ottimizzazione e controllo ottimo in forma "symbolic-to-numeric": le espressioni matematiche vengono rappresentate tramite grafi aciclici diretti (DAG, Directed Acyclic Graph), in cui i nodi corrispondono a operazioni elementari e a chiamate di funzione. Un elemento chiave sono i *Function objects*, funzioni create a runtime che possono essere valutate numericamente, differenziate automaticamente e, se necessario, esportate in C (code generation) [9].

CasADi fornisce due principali tipi di variabili simboliche:

- **SX**: rappresentazione orientata a operazioni scalari, efficace per modelli compatti;
- **MX**: rappresentazione orientata a operazioni matriciali e a grafi più complessi, adatta a NLP di grandi dimensioni e a modelli con struttura composita.

Questa distinzione consente di bilanciare efficienza computazionale e flessibilità di modellazione. Nel proseguo del lavoro verranno utilizzate variabili simboliche di tipo MX.

3.4.1 Differenziazione automatica e sfruttamento della sparsità

Il vantaggio principale di CasADi, rispetto a formulazioni basate su *differenze finite*, è la capacità di produrre derivate *esatte a precisione macchina* (coerenti con le espressioni effettivamente implementate) mediante *Algorithmic Differentiation* (AD). Nei problemi di pianificazione/ottimizzazione di traiettorie, le funzioni $f(\mathbf{x})$ e i vincoli $\mathbf{g}(\mathbf{x})$, $\mathbf{h}(\mathbf{x})$ risultano spesso composti da termini cinematici e dinamici, fortemente accoppiati e valutati su una griglia temporale: in questi contesti, stimare le derivate con differenze finite può diventare problematico per tre motivi principali:

- **Scelta del passo:** un passo troppo grande introduce errore di troncamento, mentre un passo troppo piccolo amplifica gli errori di arrotondamento (cancellazione numerica), producendo stime rumorose di gradienti e Jacobiana;
- **Costo computazionale:** il calcolo di un gradiente in $\mathbb{R}^n$ richiede in generale n valutazioni aggiuntive della funzione; il costo cresce ulteriormente per Jacobiane e Hessiane, diventando rapidamente proibitivo all'aumentare della discretizzazione temporale;
- **Instabilità e incoerenza delle derivate:** derivate imprecise degradano le linearizzazioni del sistema KKT e possono portare a direzioni di Newton non affidabili, causando rallentamenti, oscillazioni o fallimenti di convergenza (non perché la matematica diverga, ma perché l'algoritmo usa informazione locale errata).

CasADi evita tali criticità grazie alla differenziazione automatica: le derivate vengono calcolate in modo coerente a partire dalla rappresentazione computazionale delle funzioni, senza introdurre un passo di discretizzazione per la derivata. In particolare, l'AD [9]:

- non richiede la scelta di un passo per la derivazione;
- produce derivate *coerenti* con le funzioni effettivamente implementate;
- sfrutta la *sparsità strutturale* di Jacobiane e Hessiane (tipica dei problemi su griglia temporale), riducendo drasticamente memoria e tempi di calcolo.

In questo modo, CasADi consente di generare in modo sistematico: $\nabla f(\mathbf{x})$, $J_g(\mathbf{x})$ e $\nabla^2_{xx}\mathscr{L}(\mathbf{x},\boldsymbol{\lambda})$.

Per problemi di grandi dimensioni, le Jacobiane e le Hessiane risultano tipicamente *sparse* e strutturate: CasADi sfrutta questa proprietà ricostruendo le derivate tramite prodotti Jacobiana-vettore e tecniche di *graph coloring*, riducendo il numero di valutazioni necessarie rispetto a una costruzione "colonna per colonna". Il *graph coloring* è una tecnica che assegna "colori" a nodi (o archi) di un grafo imponendo che elementi adiacenti abbiano colori diversi, così da individuare insiemi indipendenti che possono essere valutati insieme. Per Jacobiane sparse, CasADi adotta tecniche di colorazione per colonne o per righe, che sfruttano la struttura per ridurre il numero di valutazioni necessarie.
Per Hessiane (simmetriche), sfrutta simmetria e sparsità per ricostruire la matrice in modo efficiente, limitando ulteriormente i calcoli richiesti.

3.4.2 Discretizzazione temporale e problemi “simulation-based”

Nei problemi in cui vincoli e obiettivo dipendono dalla simulazione di un sistema dinamico, la prassi è trasformare il problema continuo in un NLP tramite discretizzazione (direct transcription). Due impostazioni comuni sono:

- **single shooting:** le variabili decisionali parametrizzano i comandi, e lo stato si ottiene integrando in avanti; è compatto ma può essere sensibile alle inizializzazioni;
- **multiple shooting / collocation:** si introducono variabili di stato in punti intermedi, imponendo vincoli di continuità; aumenta la dimensione del NLP ma migliora robustezza e sfrutta meglio la sparsità.

Nel presente progetto, l'impostazione adottata è di tipo **single shooting**: le variabili decisionali parametrizzano la legge di moto e la dinamica viene integrata in avanti a partire da una condizione iniziale, senza introdurre gli stati intermedi come ulteriori variabili del problema. Questa scelta mantiene compatta la dimensione dell'NLP ed è ben adatta quando il numero di parametri da ottimizzare è contenuto.

Un possibile sviluppo futuro consiste nell'impiegare una formulazione **multiple shooting** (oppure collocation) per aumentare la robustezza numerica in problemi più rigidi o più vincolati: introducendo variabili di stato ai nodi e imponendo vincoli di continuità, si riduce la sensibilità all'inizializzazione e si facilita la gestione di vincoli distribuiti lungo l'orizzonte (ad esempio limiti che devono valere in più punti temporali), sfruttando al contempo la sparsità strutturale del problema.

CasADi supporta in modo nativo entrambe le impostazioni tramite funzioni, mappe e integratori, mantenendo la derivabilità del problema complessivo.
Dal punto di vista implementativo, CasADi mette a disposizione strumenti particolarmente utili nella discretizzazione temporale di sistemi dinamici. In particolare:

- `Function` consente di incapsulare leggi di moto e dinamiche in oggetti valutabili e differenziabili;
- `map` permette di valutare una stessa funzione su molti istanti temporali in modo vettoriale, riducendo overhead;
- `mapaccum` consente di applicare iterativamente una funzione di step e accumulare l'evoluzione dello stato lungo l'orizzonte, evitando cicli espliciti e preservando la struttura differenziabile della pipeline.

Questi costrutti risultano particolarmente comodi quando la dinamica viene integrata con schemi a passo fisso, come ad esempio un integratore di tipo Runge–Kutta del quarto ordine (RK4), perché permettono di esprimere in forma compatta sia la valutazione della forzante su griglia, sia l'integrazione sequenziale dello stato.

Un esempio recente in ambito anti-sloshing mostra una pipeline tipica basata su discretizzazione dell'orizzonte, integrazione numerica e risoluzione via IPOPT, evidenziando come scelte quali schema di shooting e risoluzione temporale influenzino direttamente vincoli e convergenza [14].

La presenza di vincoli definiti su un intero intervallo temporale (e quindi non legati a un singolo istante, ma “distribuiti” lungo l'orizzonte) richiede una discretizzazione sufficientemente fine da garantire l'ammissibilità non solo nei nodi di griglia, ma anche tra i nodi; al contempo, tale scelta deve bilanciare accuratezza e costo computazionale, che cresce all'aumentare del numero di punti.

3.4.3 Funzione obiettivo e vincoli del problema anti-sloshing

Nel problema anti-sloshing la funzione obiettivo deve riflettere direttamente la qualità del trasporto del liquido e la sicurezza del processo. Per questo motivo si assume come grandezza di interesse l'altezza del pelo libero al bordo del contenitore $\overline{\eta}$ (introdotta nell'eq. (2.34) e si adottano metriche di tipo "worst-case", basate sul massimo valore assoluto lungo un intervallo temporale. Obiettivi di questo tipo sono in generale *non lisci* (non differenziabili in tutti i punti), poiché coinvolgono un operatore di massimo. In forma generale, il problema può essere scritto come

$$\left(\min_{\mathbf{x}}\right)\left(\max_{t\in[0,t_d]}\right)\phi(t;\mathbf{x}), \tag{3.19}$$

dove $\mathbf{x}$ è il vettore delle variabili decisionali e $\phi(t;\mathbf{x})$ è una funzione scalare generica che misura la prestazione istantanea (nel presente caso, ad esempio, $\phi(t;\mathbf{x}) = |\eta(t;\mathbf{x})|$). Una strategia standard per ricondurli a una forma più trattabile in un NLP consiste nell'introdurre una variabile ausiliaria γ e riscrivere il problema in forma *epigrafica*:

$$\min_{\mathbf{x},\gamma}\ \gamma \qquad \text{s.t.} \qquad \phi(t_k;\mathbf{x}) \le \gamma, \quad k = 1,\dots,N, \tag{3.20}$$

dove $\{t_k\}$ è una griglia temporale. In questo modo l'operatore di massimo viene sostituito da un insieme di vincoli che impongono a γ di essere un maggiorante dei valori campionati di ϕ, rendendo l'obiettivo liscio e gestibile da solutori NLP.

Nel presente lavoro, la funzione obiettivo viene costruita a partire dalla stima dell'altezza di sloshing al bordo del contenitore, $\overline{\eta}$, ottenuta risolvendo il modello dinamico descritto nel Capitolo 2 per una legge di moto di input definita dai tratti racchiusi nel vettore x. In continuità con la logica proposta in [7], una scelta efficace consiste nel minimizzare una misura di picco durante l'intero intervallo di moto:

$$f(\mathbf{x}) = \max_{t\in[0,t_d]} |\overline{\eta}(t;\mathbf{x})|. \tag{3.21}$$

In alternativa, qualora l'obiettivo applicativo sia la riduzione delle oscillazioni residue (ad esempio per migliorare operazioni in sosta come la pesatura), si può definire una finestra di osservazione post-moto e minimizzare il massimo in fase di riposo:

$$f(\mathbf{x}) = \max_{t\in[t_d,t_d+t_r]} |\overline{\eta}(t;\mathbf{x})|, \tag{3.22}$$

dove t_r è un orizzonte di osservazione durante la sosta. In entrambi i casi, se si desidera una formulazione più "liscia" e direttamente compatibile con i solutori NLP, è possibile ricorrere alla forma epigrafica (3.20) applicandola a $\phi(t;\mathbf{x}) = |\overline{\eta}(t;\mathbf{x})|$ sui nodi temporali considerati.

Oltre ai vincoli di consistenza (3.4), si introducono tipicamente vincoli che garantiscono sia l'eseguibilità della legge di moto sia il rispetto di limiti prestazionali:

Vincoli di durata minima dei segmenti.

$$\Delta T_i \ge \Delta T_{\min}, \qquad i = 1,\dots,7, \tag{3.23}$$

in modo da evitare segmenti eccessivamente brevi (con jerk elevato) e garantire robustezza di esecuzione. In particolare, per i tratti a jerk costante (rampe) $i \in \{1,3,5,7\}$ si impone una durata strettamente positiva $\Delta T_i \ge \Delta T_{\min} > 0$, così da evitare tratti degeneri e jerk non realistici. Per i tratti a accelerazione costante o nulla $i \in \{2,4,6\}$ è invece ammissibile anche $\Delta T_i = 0$, così da consentire profili "ridotti" quando una fase (plateau o sosta) non è necessaria.

Vincoli cinematici/dinamici di macchina. In funzione dei limiti dell'asse/robot e della catena cinematica, possono essere imposti vincoli su velocità, accelerazione e jerk:

$$|v(t)| \leq v_{\max}, \qquad |a(t)| \leq a_{\max}, \qquad |j(t)| \leq j_{\max}, \tag{3.24}$$

dove $j(t) = \dot{a}(t)$.

Vincoli sullo sloshing. In molte applicazioni è più importante garantire che lo sloshing rimanga entro soglie di sicurezza, piuttosto che minimizzarlo in senso assoluto. Si introduce quindi un vincolo del tipo:

$$\max_{t \in \mathcal{T}} |\overline{\eta}(t;\mathbf{x})| \leq \overline{\eta}_{\lim}, \tag{3.25}$$

dove $\mathcal{T}$ può coincidere con la fase di moto, con la fase di sosta o con entrambe.

Valutazione numerica di obiettivo e vincoli. Dato un vettore $\mathbf{x}$, la valutazione di $f(\mathbf{x})$ e dei vincoli (3.24)–(3.25) richiede una simulazione lungo l'orizzonte temporale:

- costruzione del profilo $a(t)$ a partire dai ΔT_i;
- integrazione per ottenere $v(t)$ e $s(t)$;
- risoluzione del modello di sloshing (nel presente lavoro *non lineare*) per ottenere $\overline{\eta}(t)$.

Ciò giustifica l'impiego di un framework come CasADi, in grado di gestire funzioni non lineari complesse, preservare la differenziabilità della catena di calcolo e fornire al solutore derivate accurate per migliorare robustezza e convergenza.

Ottimizzatore. In continuità con la formulazione NLP descritta nelle precedenti sezioni, nel presente lavoro è stato implementato un primo ottimizzatore nel dominio del tempo (`Opt_trapezia`) consultabile in Appendice A. L'obiettivo è determinare le durate ottimali dei sette tratti ΔT_i della legge trapezia standard, trattandole come variabili decisionali, imponendo vincoli di consistenza (somma dei tempi e durate minime) e vincoli prestazionali di macchina quando richiesto.

Operativamente, a ogni iterazione del solver l'insieme di parametri $\mathbf{x} = [\Delta T_1, \ldots, \Delta T_7]^T$ definisce il profilo di accelerazione $a(t;\mathbf{x})$; tale forzante viene quindi impiegata nella simulazione del modello dinamico di sloshing (cfr. Capitolo 2), ottenendo l'evoluzione di $\overline{\eta}(t;\mathbf{x})$ lungo l'orizzonte temporale. La funzione obiettivo è costruita in forma "worst-case"; ad esempio minimizzando il picco di $|\overline{\eta}(t)|$ durante il moto, eventualmente vincolando la risposta in sosta tramite soglie di accettabilità.

Capitolo 4

Metodi anti-sloshing

Nel controllo del moto di macchine automatiche e robot industriali, la mitigazione dello sloshing non richiede soltanto buone prestazioni in condizioni nominali, ma anche la capacità di mantenere comportamenti accettabili al variare dei parametri fisici e delle condizioni operative. In particolare, la frequenza naturale dominante del pelo libero e lo smorzamento effettivo possono variare con il livello di riempimento, le proprietà del liquido e la geometria del recipiente (cfr. Capitolo 2). In questo contesto, una soluzione ottenuta ottimizzando unicamente su un modello nominale può risultare efficace solo in un intorno ristretto delle ipotesi, mentre piccole variazioni parametriche possono degradare sensibilmente la prestazione.

Per perseguire un comportamento più *robusto*, è naturale ragionare in termini di contenuto in frequenza della forzante imposta al recipiente. L'idea di fondo è ridurre l'energia (o, più in generale, l'ampiezza spettrale) della legge di accelerazione in corrispondenza della frequenza naturale di sloshing ω_n, e possibilmente in un intervallo attorno ad essa, così da limitare l'eccitazione anche in presenza di incertezze su ω_n e ζ. Questo ragionamento richiama direttamente la logica delle tecniche di *input shaping*, che realizzano una modifica della traiettoria standard con l'obiettivo di attenuare (idealmente annullare) la vibrazione residua agendo in modo mirato sulle componenti armoniche responsabili dell'eccitazione del modo vibrante dominante.

4.1 Tecniche di input shaping

Le tecniche di *input shaping* costituiscono una classe di metodi in catena aperta che mirano a ridurre le oscillazioni residue di un sistema vibrante evitando (o cancellando) l'eccitazione dei suoi modi naturali [18]. Nella forma classica, l'idea è costruire un *filtro* nel dominio del tempo sotto forma di un treno di impulsi (input shaper) e convolvere tale filtro con un comando di riferimento (una traiettoria standard), ottenendo un comando modificato che produce vibrazioni residue ridotte.

La formulazione classica dell'input shaping fa riferimento a un sistema lineare del secondo ordine, rappresentativo del modo vibrante dominante del sistema fisico (nel caso dello sloshing, tipicamente il modo fondamentale del modello equivalente lineare). Assumendo un rapporto di smorzamento $0 < \zeta_n < 1$ e una pulsazione naturale $\omega_n > 0$, la risposta del sistema a un'eccitazione impulsiva può essere descritta in forma chiusa.

Per introdurre formalmente un input shaper, si richiama la definizione di impulso di Dirac $\delta(t)$, inteso come distribuzione tale che:

$$\delta(t) = 0 \ \text{ per } t \neq 0, \qquad \int_{-\infty}^{+\infty} \delta(t)\, dt = 1. \tag{4.1}$$

In particolare, un impulso unitario traslato all'istante t_j viene rappresentato come $\delta(t - t_j)$.
La risposta (normalizzata) di un sistema del secondo ordine sottosmorzato a un impulso unitario applicato all'istante t_j è una sinusoide esponenzialmente smorzata. Indicando con:

$$\omega_d = \omega_n \sqrt{1 - \zeta_n^2}, \tag{4.2}$$

Il periodo dell'oscillazione smorzata risulta:

$$T = \frac{2\pi}{\omega_d} = \frac{2\pi}{\omega_n \sqrt{1 - \zeta_n^2}}. \tag{4.3}$$

Treno di impulsi e ampiezza della vibrazione residua L'input shaper viene modellato come una sequenza di $N+1$ impulsi, ciascuno caratterizzato da un'ampiezza A_j e da un istante di applicazione t_j:

$$g(t) = \sum_{j=0}^{N} A_j\, \delta(t - t_j). \tag{4.4}$$

Per linearità, la risposta complessiva a tale treno di impulsi è la somma delle risposte parziali. Per quantificare la vibrazione residua dopo l'ultimo impulso (ossia al tempo $t = t_N$), si introduce una misura scalare $Z(\omega_n, \zeta_n)$ che rappresenta l'ampiezza risultante dell'oscillazione residua (Figura 4.1) . Tale grandezza è ricavabile in forma analitica per un sistema massa-molla-smorzatore come quello utilizzato nella modellazione lineare dello sloshing (cfr. Capitolo 2). Per maggiori informazioni si fa riferimento a [10].

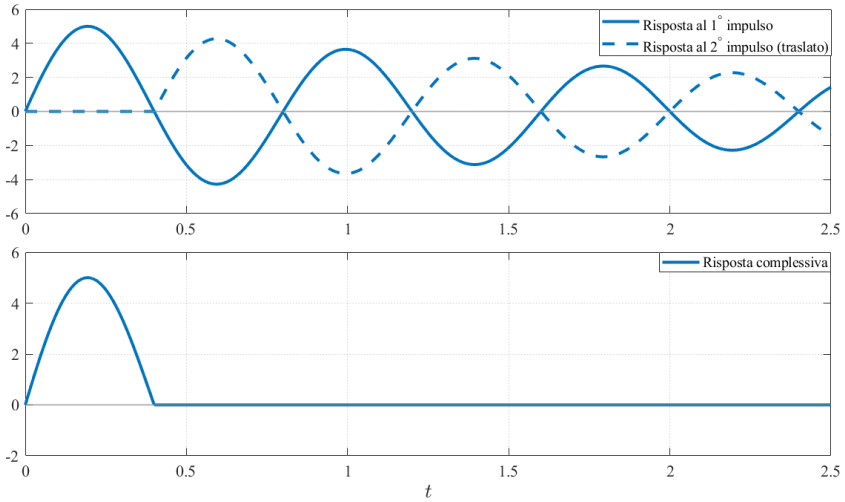


Figura 4.1: Risposta a due impulsi e cancellazione della vibrazione residua: le risposte parziali si sovrappongono fino ad annullarsi a regime.

Per ottenere uno *shaper* a vibrazione residua nulla (*Zero Vibration*, ZV) è sufficiente imporre che l'ampiezza della vibrazione residua del modo considerato sia nulla al termine della sequenza di impulsi, ovvero:

$$Z(\omega_n, \zeta_n) = 0. \tag{4.5}$$

Imponendo questa condizione e assumendo uno *shaper* costituito da due soli impulsi (cioè $N = 1$), ponendo il primo impulso all'istante $t_0 = 0$, si ricavano direttamente lo sfasamento temporale e le ampiezze degli impulsi del treno ZV:

$$t_0 = 0, \qquad t_1 = \frac{T}{2}, \tag{4.6}$$

$$A_0 = \frac{1}{1+K}, \qquad A_1 = \frac{K}{1+K}, \tag{4.7}$$

dove T è il periodo della risposta libera smorzata del modo dominante,

$$T = \frac{2\pi}{\omega_n \sqrt{1-\zeta_n^2}}, \tag{4.8}$$

e il termine

$$K = \exp\left(-\frac{\zeta_n \pi}{\sqrt{1-\zeta_n^2}}\right) \tag{4.9}$$

deriva dallo smorzamento del modo. La condizione di normalizzazione $A_0 + A_1 = 1$ garantisce inoltre che il comando "shapato" preservi il guadagno statico rispetto al comando originale.
Nella pratica, la cancellazione perfetta ottenuta imponendo (4.5) è garantita solo per parametri nominali esatti. Se ω_n (o ζ_n) è affetto da incertezza, lo shaper può perdere efficacia. Per aumentare la robustezza, si introduce l'idea di imporre condizioni aggiuntive sulla sensibilità della vibrazione residua rispetto ai parametri.

Se oltre alla condizione ZV, si impone l'annullamento della derivata della vibrazione residua rispetto a un parametro (tipicamente ω_n):

$$\frac{\partial Z(\omega_n, \zeta_n)}{\partial \omega_n} = 0. \tag{4.10}$$

Si ottiene così uno shaper a tre impulsi caratterizzato dai seguenti parametri:

$$\begin{aligned} t_0 &= 0, \quad A_0 = \frac{1}{1+2K+K^2}, \\ t_1 &= \frac{T}{2}, \quad A_1 = \frac{2K}{1+2K+K^2}, \\ t_2 &= T, \quad A_2 = \frac{K^2}{1+2K+K^2}. \end{aligned} \tag{4.11}$$

Per aumentare ulteriormente l'insensibilità a errori su ω_n, si può imporre anche la condizione sulla derivata seconda:

$$\frac{\partial^2 Z(\omega_n, \zeta_n)}{\partial \omega_n^2} = 0. \tag{4.12}$$

Il corrispondente shaper a quattro impulsi risulta:

$$\begin{aligned} t_0 &= 0, & A_0 &= \frac{1}{1+3K+3K^2+K^3}, \\ t_1 &= \frac{T}{2}, & A_1 &= \frac{3K}{1+3K+3K^2+K^3}, \\ t_2 &= T, & A_2 &= \frac{3K^2}{1+3K+3K^2+K^3}, \\ t_3 &= \frac{3T}{2}, & A_3 &= \frac{K^3}{1+3K+3K^2+K^3}. \end{aligned} \tag{4.13}$$

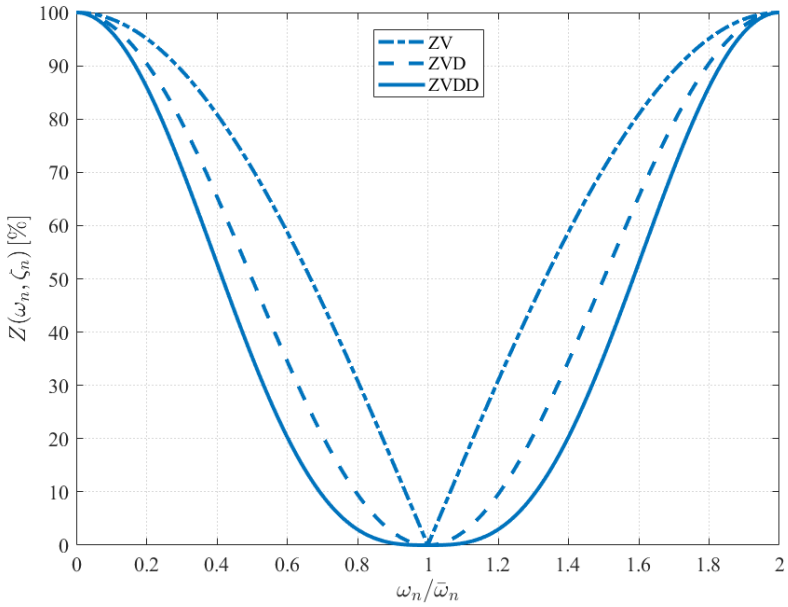


Figura 4.2: Curve di sensitività di *input shapers* ZV, ZVD e ZVDD per $\zeta_n = 0$.

Tutti gli shaper precedenti sono progettati per annullare la vibrazione residua alla frequenza nominale, ma differiscono per la loro *robustezza* rispetto a errori di modellazione: aumentando il numero di impulsi si amplia l'intorno di frequenze per cui l'attenuazione rimane efficace. Di contro, il tempo dell'ultimo impulso t_N cresce e lo shaper introduce un *ritardo* maggiore sul comando filtrato. La situazione è chiara in figura 4.2.

Un'ulteriore strategia, orientata esplicitamente alla robustezza, consiste nel non imporre l'annullamento esatto di Z solo in un punto, ma nel limitare la vibrazione residua a un livello tollerabile $\bar{z}$ su un intervallo più ampio di frequenze. In altre parole, si accetta una vibrazione residua non nulla ma controllata, in cambio di una banda di efficacia più larga [16].
Nel caso ideale non smorzato utilizzando un treno di tre impulsi si ottiene uno shaper detto Extra Insensitive (EI), definito dai seguenti parametri:

$$\begin{aligned} t_0 &= 0, & A_0 &= \frac{1+\bar{z}}{4}, \\ t_1 &= \frac{T}{2}, & A_1 &= \frac{1-\bar{z}}{2}, \\ t_2 &= T, & A_2 &= \frac{1+\bar{z}}{4}. \end{aligned} \tag{4.14}$$

La scelta di $\bar{z}$ regola direttamente il compromesso tra ampiezza residua accettata e robustezza: valori più piccoli richiedono una regione di attenuazione (notch) più profonda, ossia una riduzione più marcata della vibrazione residua in corrispondenza della frequenza nominale (comportamento più simile a ZVD/ZVDD); valori più grandi, invece, aumentano la banda di insensibilità.

Convoluzione: estensione a forzanti generiche nel tempo La trattazione impulsiva è utile per derivare gli shaper, ma nel caso reale l'ingresso al sistema è una funzione del tempo generica (ad esempio un profilo di accelerazione). Un segnale arbitrario può essere visto come sovrapposizione di impulsi, e si dimostra che applicare l'input shaping equivale a filtrare tale segnale tramite convoluzione con il treno di impulsi.

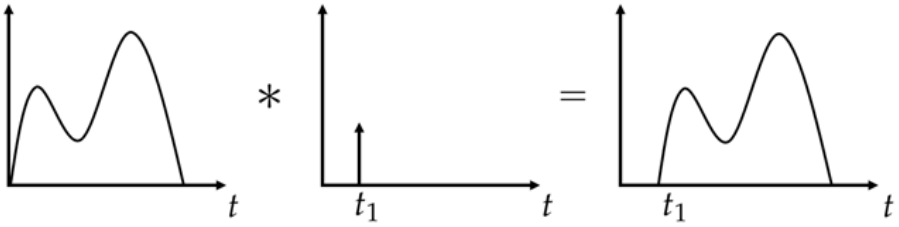


Figura 4.3: Operazione di convoluzione di un segnale generico del tempo con un impulso di Dirac unitario traslato rispetto all'origine.

Dato (4.4), la convoluzione tra una forzante $f(t)$ e lo shaper $g(t)$ produce:

$$y_0(t) = f(t) * g(t) = \int_{-\infty}^{+\infty} f(\tau) \sum_{j=0}^{N} A_j \, \delta(t - t_j - \tau) \, d\tau = \sum_{j=0}^{N} A_j \, f(t - t_j). \tag{4.15}$$

L'interpretazione è immediata: il comando filtrato è una combinazione lineare di copie ritardate del comando originale. Per una trattazione completa si rimanda a [10].

L'operazione di convoluzione (4.15) nel dominio del tempo corrisponde a una moltiplicazione nel dominio della frequenza: l'input shaper può quindi essere interpretato come un filtro che introduce zeri (notch) in corrispondenza delle frequenze indesiderate, attenuando l'eccitazione del modo naturale. Questa interpretazione è particolarmente rilevante per lo sloshing: pur non applicando direttamente un input shaper nel presente progetto, si adotta la stessa logica di fondo, ossia agire sul contenuto armonico della forzante (legge di accelerazione) per ridurre l'eccitazione in prossimità di ω_n e, in ottica di robustezza, in un intorno di tale valore.

4.2 Analisi e ottimizzazione spettrale della legge trapezia standard

Come discusso nella sezione precedente, l'idea dell'*input shaping* consiste nel ridurre le oscillazioni residue evitando di eccitare i modi naturali del sistema.

Nel presente lavoro, la forzante che eccita lo *sloshing* è l'accelerazione imposta al contenitore, $a(t)$, generata dalla legge trapezia standard a 7 tratti (Capitolo 2 e Capitolo 3). Per agire in modo mirato sui contributi che eccitano i modi propri del pelo libero, è quindi utile introdurre lo *spettro di ampiezza* della legge di accelerazione.

Trasformata di Fourier continua e spettro di ampiezza. Considerando un segnale $x(t)$ definito su una finestra temporale finita $[t_0, t_f]$, la trasformata di Fourier continua è:

$$X(\omega) = \int_{t_0}^{t_f} x(t)\, e^{-j\omega t}\, dt. \tag{4.16}$$

Usando l'identità di Eulero $e^{-j\omega t} = \cos(\omega t) - j\sin(\omega t)$, si può riscrivere:

$$X(\omega) = \int_{t_0}^{t_f} x(t)\cos(\omega t)\, dt \; - \; j\int_{t_0}^{t_f} x(t)\sin(\omega t)\, dt, \tag{4.17}$$

da cui seguono direttamente parte reale e immaginaria:

$$\begin{aligned} \Re\{X(\omega)\} &= \int_{t_0}^{t_f} x(t)\cos(\omega t)\, dt \\ \Im\{X(\omega)\} &= -\int_{t_0}^{t_f} x(t)\sin(\omega t)\, dt \end{aligned} \tag{4.18}$$

Nel seguito si applicano queste definizioni al segnale $x(t) = a(t)$, ossia all'accelerazione della legge trapezia: si indicherà quindi $X_a(\omega) \equiv \mathscr{F}\{a(t)\}$, dove $\mathscr{F}\{\cdot\}$ denota l'operatore di trasformata di Fourier, mentre $|X_a(\omega)|$ rappresenta lo spettro di ampiezza della forzante.

4.2.1 Determinazione di $a_{\max}$ e $a_{\min}$ per moto rest-to-rest

Si consideri la legge trapezia standard in accelerazione a sette tratti, definita dagli intervalli temporali $\Delta T_1, \ldots, \Delta T_7$ e dagli istanti di cambio tratto

$$\begin{aligned} &t_0 = 0, && t_1 = \Delta T_1, \\ &t_2 = t_1 + \Delta T_2, && t_3 = t_2 + \Delta T_3, \\ &t_4 = t_3 + \Delta T_4, && t_5 = t_4 + \Delta T_5, \\ &t_6 = t_5 + \Delta T_6, && t_7 = t_6 + \Delta T_7. \end{aligned}$$

Il profilo di accelerazione $a(t)$ è composto da rampe lineari (jerk costante) e tratti a accelerazione costante. Con una parametrizzazione coerente con la trapezia standard

si può scrivere, in forma a tratti:

$$a(t) = \begin{cases} a_{\max} \dfrac{t - t_0}{\Delta T_1}, & t \in [t_0, t_1] \\ a_{\max}, & t \in (t_1, t_2] \\ a_{\max} - a_{\max} \dfrac{t - t_2}{\Delta T_3}, & t \in (t_2, t_3] \\ 0, & t \in (t_2, t_4] \\ -a_{\min} \dfrac{t - t_4}{\Delta T_5}, & t \in (t_4, t_5] \\ -a_{\min}, & t \in (t_5, t_6] \\ -a_{\min} + a_{\min} \dfrac{t - t_6}{\Delta T_7}, & t \in (t_6, t_7] \\ 0, & \text{altrove.} \end{cases}$$

Vincolo di velocità finale nulla. Imponendo un moto *rest-to-rest*, la variazione complessiva di velocità deve essere nulla:

$$v(t_7) - v(t_0) = \int_{t_0}^{t_7} a(\tau)\, d\tau = 0. \tag{4.19}$$

Tale integrale coincide con l'area sottesa dal profilo di accelerazione $a(t)$. La parte positiva (tratti 1–3) fornisce:

$$A_+ = a_{\max}\left(\frac{\Delta T_1}{2} + \Delta T_2 + \frac{\Delta T_3}{2}\right) = a_{\max}\, d_3, \qquad d_3 = \frac{\Delta T_1}{2} + \Delta T_2 + \frac{\Delta T_3}{2},$$

mentre la parte negativa (tratti 5–7), essendo $a(t)$ negativa, dà:

$$A_- = -a_{\min}\left(\frac{\Delta T_5}{2} + \Delta T_6 + \frac{\Delta T_7}{2}\right) = a_{\min}\, d_4, \qquad d_4 = -\left(\frac{\Delta T_5}{2} + \Delta T_6 + \frac{\Delta T_7}{2}\right).$$

Il vincolo (4.19) diventa quindi:

$$A_+ + A_- = 0 \qquad \Longrightarrow \qquad d_3\, a_{\max} + d_4\, a_{\min} = 0. \tag{4.20}$$

Vincolo sullo spostamento totale. Lo spostamento totale imposto è

$$s_{\text{tot}} = x(t_7) - x(t_0), \tag{4.21}$$

dove, per definizione,

$$x(t_7) = \int_{t_0}^{t_7} v(t)\, dt, \qquad v(t) = \int_{t_0}^{t} a(\tau)\, d\tau. \tag{4.22}$$

Sostituendo (4.22) in (4.21) e scambiando l'ordine di integrazione nel seguente modo:

$$s_{\text{tot}} = \int_0^{t_7} v(s)\, ds = \int_0^{t_7} \left(\int_0^{s} a(\tau)\, d\tau\right) ds \tag{4.23}$$

dove il dominio di integrazione è il triangolo $\{(\tau, s) \mid 0 \le \tau \le s \le t_7\}$; invertendo l'ordine, τ varia da 0 a t_7 e, fissato τ, si ha $s \in [\tau, t_7]$:

$$\int_0^{t_7} \left(\int_\tau^{t_7} a(\tau)\, ds \right) d\tau \tag{4.24}$$

si ottiene una forma particolarmente utile:

$$s_{\text{tot}} = \int_{t_0}^{t_7} a(\tau)\,(t_7 - \tau)\, d\tau. \tag{4.25}$$

Come si nota, ciascun contributo di accelerazione viene pesato dal "tempo che resta" alla fine della manovra, $t_7 - \tau$.

Per rendere esplicita la dipendenza di s_{tot} dai soli parametri temporali ΔT_i, si riscrive il profilo trapezia in forma lineare rispetto ai livelli di accelerazione:

$$a(t) = a_{\max}\,\phi(t) + a_{\min}\,\psi(t), \tag{4.26}$$

dove $\phi(t)$ e $\psi(t)$ sono funzioni *normalizzate* (dipendono solo da ΔT_i e dagli istanti t_k) che descrivono, rispettivamente, il "lobo" positivo e quello negativo della legge:

$$\phi(t) = \begin{cases} \dfrac{t - t_0}{\Delta T_1}, & t \in [t_0, t_1], \\ 1, & t \in (t_1, t_2], \\ 1 - \dfrac{t - t_2}{\Delta T_3}, & t \in (t_2, t_3], \\ 0, & \text{altrove}, \end{cases} \qquad \psi(t) = \begin{cases} -\dfrac{t - t_4}{\Delta T_5}, & t \in (t_4, t_5], \\ -1, & t \in (t_5, t_6], \\ -\left(1 - \dfrac{t - t_6}{\Delta T_7}\right), & t \in (t_6, t_7], \\ 0, & \text{altrove}. \end{cases} \tag{4.27}$$

Sostituendo (4.26) nella forma pesata (4.25) si ottiene:

$$s_{\text{tot}} = a_{\max} \underbrace{\int_{t_0}^{t_7} \phi(\tau)(t_7 - \tau)\, d\tau}_{d_1} + a_{\min} \underbrace{\int_{t_0}^{t_7} \psi(\tau)(t_7 - \tau)\, d\tau}_{d_2}, \tag{4.28}$$

ovvero la combinazione lineare

$$s_{\text{tot}} = d_1\, a_{\max} + d_2\, a_{\min}. \tag{4.29}$$

Per calcolare d_1 e d_2 si integra a tratti (4.28). Ad esempio, sul primo tratto $\phi(t) = (t - t_0)/\Delta T_1$ e si ottiene:

$$\int_{t_0}^{t_1} \frac{t - t_0}{\Delta T_1}\,(t_7 - t)\, dt = t_7\, \frac{\Delta T_1}{2} - \frac{\Delta T_1^2}{3}. \tag{4.30}$$

Ripetendo lo stesso procedimento sui tratti 2 e 3 (per ϕ) e sui tratti 5, 6, 7 (per ψ), si ricavano espressioni chiuse in funzione delle sole durate ΔT_i. In particolare, risulta:

$$d_1 = t_7 \left(\frac{\Delta T_1}{2} + \Delta T_2 + \frac{\Delta T_3}{2} \right) - \left(\frac{\Delta T_1^2}{3} + \Delta T_1 \Delta T_2 + \frac{\Delta T_2^2}{2} + \frac{\Delta T_1 \Delta T_3}{2} + \frac{\Delta T_2 \Delta T_3}{2} + \frac{\Delta T_3^2}{6} \right), \tag{4.31}$$

$$d_2 = -\left(\frac{\Delta T_5^2}{6} + \frac{\Delta T_5 \Delta T_6}{2} + \frac{\Delta T_5 \Delta T_7}{2} + \frac{\Delta T_6^2}{2} + \Delta T_6 \Delta T_7 + \frac{\Delta T_7^2}{3} \right). \tag{4.32}$$

Le relazioni (4.31)–(4.32) mostrano quindi in modo esplicito che i coefficienti d_1 e d_2 dipendono unicamente dalle durate ΔT_i, rendendo la determinazione di $a_{\max}$ e $a_{\min}$ un problema lineare nelle due incognite, come discusso nel seguito.

Sistema lineare e soluzione. Combinando (4.20) e (4.29) si ottiene un sistema lineare in $a_{\max}$ e $a_{\min}$:

$$\begin{pmatrix} d_1 & d_2 \\ d_3 & d_4 \end{pmatrix} \begin{pmatrix} a_{\max} \\ a_{\min} \end{pmatrix} = \begin{pmatrix} s_{\text{tot}} \\ 0 \end{pmatrix}. \tag{4.33}$$

Assumendo $\det(M) = d_1 d_4 - d_2 d_3 \neq 0$, la soluzione si ricava immediatamente (ad esempio con il metodo di Cramer):

$$a_{\max} = \frac{d_4\, s_{\text{tot}}}{d_1 d_4 - d_2 d_3}, \qquad a_{\min} = -\frac{d_3\, s_{\text{tot}}}{d_1 d_4 - d_2 d_3}. \tag{4.34}$$

Tali espressioni consentono di determinare i livelli di accelerazione $a_{\max}$ e $a_{\min}$ in funzione delle durate ΔT_i e dello spostamento imposto, mantenendo esplicita la dipendenza parametrica necessaria per le analisi spettrali sviluppate nel seguito.

4.2.2 Spettro analitico della trapezia standard: derivazione in forma chiusa

In questa sezione si introduce una derivazione sintetica dello spettro analitico della legge trapezia standard a sette tratti. La legge di accelerazione $a(t)$ è quella definita nella Sezione 4.2.1, con livelli $a_{\max}$ e $a_{\min}$ determinati imponendo moto *rest-to-rest* e lo spostamento totale.
Si considera la trasformata di Fourier continua dell'accelerazione, limitata alla finestra temporale $[t_0, t_7]$ su cui la legge è definita:

$$X_a(\omega) = \int_{t_0}^{t_7} a(t)\, e^{-j\omega t}\, dt. \tag{4.35}$$

Poiché $a(t)$ è una funzione a tratti, l'integrale può essere scritto come somma dei contributi dei sette segmenti:

$$X_a(\omega) = \sum_{k=1}^{7} \int_{t_{k-1}}^{t_k} a_k(t)\, e^{-j\omega t}\, dt, \tag{4.36}$$

dove $a_k(t)$ è l'espressione di $a(t)$ sul tratto k-esimo.
Si consideri un segmento su cui l'accelerazione è costante:

$$a(t) = A_0, \qquad t \in [t_a, t_b].$$

Il contributo spettrale associato è:

$$X_{\text{const}}(\omega) = \int_{t_a}^{t_b} A_0\, e^{-j\omega t}\, dt = A_0 \left[\frac{e^{-j\omega t}}{-j\omega} \right]_{t_a}^{t_b} = A_0\, \frac{e^{-j\omega t_a} - e^{-j\omega t_b}}{j\omega}. \tag{4.37}$$

Separando parte reale e immaginaria tramite $e^{-j\omega t} = \cos(\omega t) - j\sin(\omega t)$, si ottiene:

$$X_{\text{const}}(\omega) = \frac{A_0}{\omega} \Big[\big(\sin(\omega t_b) - \sin(\omega t_a)\big) + j\big(\cos(\omega t_a) - \cos(\omega t_b)\big) \Big] \tag{4.38}$$

da cui:

$$\begin{aligned}\Re\{X_{\text{const}}\} &= \frac{A_0}{\omega}\big(\sin(\omega t_b) - \sin(\omega t_a)\big)\\ \Im\{X_{\text{const}}\} &= \frac{A_0}{\omega}\big(\cos(\omega t_a) - \cos(\omega t_b)\big)\end{aligned} \tag{4.39}$$

Si consideri ora un segmento in cui l'accelerazione è lineare:

$$a(t) = mt + q, \qquad t \in [t_a, t_b]$$

dove m e q sono due coefficienti reali generici che definiscono la retta nel tratto considerato: m è il coefficiente angolare della rampa di accelerazione e q la relativa intercetta. A questo punto, il contributo spettrale è:

$$X_{\text{lin}}(\omega) = \int_{t_a}^{t_b} (mt+q)\, e^{-j\omega t}\, dt = m\int_{t_a}^{t_b} t\, e^{-j\omega t}\, dt + q\int_{t_a}^{t_b} e^{-j\omega t}\, dt. \tag{4.40}$$

Il secondo integrale è quello del caso costante. Per il primo si può usare integrazione per parti, ottenendo la primitiva:

$$\int t\, e^{-j\omega t}\, dt = \frac{(j\omega t + 1)e^{-j\omega t}}{\omega^2}, \qquad (\omega \neq 0). \tag{4.41}$$

Valutando tra t_a e t_b e combinando i contributi, si ottiene una forma complessa chiusa:

$$X_{\text{lin}}(\omega) = m\left[\frac{(j\omega t+1)e^{-j\omega t}}{\omega^2}\right]_{t_a}^{t_b} + q\left[\frac{e^{-j\omega t}}{-j\omega}\right]_{t_a}^{t_b}. \tag{4.42}$$

Sostituendo $e^{-j\omega t} = \cos(\omega t) - j\sin(\omega t)$ e raccogliendo i termini reali e immaginari, si ottiene una combinazione di contributi del tipo

$$\Re\{X_{\text{lin}}\} = \alpha_1\frac{\cos(\omega t_a)}{\omega^2} + \alpha_2\frac{\cos(\omega t_b)}{\omega^2} + \alpha_3\frac{t_a\sin(\omega t_a)}{\omega} + \alpha_4\frac{t_b\sin(\omega t_b)}{\omega} \tag{4.43}$$

$$\Im\{X_{\text{lin}}\} = \beta_1\frac{\sin(\omega t_a)}{\omega^2} + \beta_2\frac{\sin(\omega t_b)}{\omega^2} + \beta_3\frac{t_a\cos(\omega t_a)}{\omega} + \beta_4\frac{t_b\cos(\omega t_b)}{\omega} \tag{4.44}$$

dove i coefficienti α_i e β_i si ottengono raccogliendo i termini in $\cos(\omega t_a), \cos(\omega t_b)$, $\sin(\omega t_a), \sin(\omega t_b)$ e risultano:

$$\begin{aligned}&\alpha_1 = -m, \quad \alpha_2 = m, \quad \alpha_3 = -m - \frac{q}{t_a}, \quad \alpha_4 = m + \frac{q}{t_b},\\ &\beta_1 = m, \quad \beta_2 = -m, \quad \beta_3 = -m - \frac{q}{t_a}, \quad \beta_4 = m + \frac{q}{t_b}.\end{aligned} \tag{4.45}$$

Per completezza: i prodotti che compaiono in (4.43) sono $\alpha_3 t_a = -(mt_a + q)$ e $\alpha_4 t_b = (mt_b + q)$, e analogamente $\beta_3 t_a = -(mt_a + q)$ e $\beta_4 t_b = (mt_b + q)$ in (4.44). Nel caso $t_a = 0$ la combinazione $\alpha_3 t_a$ resta ben definita perché coincide con $-a(t_a)$.

Applicazione ai 7 segmenti della trapezia standard. I tratti della legge trapezia si distinguono in:

- **rampe lineari** (tratti 1, 3, 5, 7), per cui si applica (4.42) con $(t_a, t_b) = (t_{k-1}, t_k)$ e con pendenza/intercetta (m, q) coerenti col profilo;
- **tratti costanti** (tratti 2 e 6), per cui si applica (4.37);
- **tratto nullo** (tratto 4), il cui contributo è nullo.

In particolare, per le rampe lineari si può scrivere $a(t) = mt + q$ con:

$$m = \pm\frac{a_{\max}}{\Delta T_1},\ \pm\frac{a_{\max}}{\Delta T_3},\ \pm\frac{a_{\min}}{\Delta T_5},\ \pm\frac{a_{\min}}{\Delta T_7},$$

e q determinato imponendo continuità del valore di accelerazione agli estremi del tratto.

Forma chiusa di $\Re\{X_a(\omega)\}$ e $\Im\{X_a(\omega)\}$. Il contributo complessivo è:

$$X_a(\omega) = \sum_{k=1}^{7} X_k(\omega), \qquad X_k(\omega) = \Re\{X_k(\omega)\} + j\,\Im\{X_k(\omega)\}. \tag{4.46}$$

Dopo l'espansione e la raccolta dei termini in forma simbolica sviluppate con il software Mathematica, si ottengono, a partire dalle eq. (4.43) e (4.44), due espressioni equivalenti alle forme chiuse riportate nelle eq. 4.47 e 4.48 (dove ricordiamo che $\Delta T_i = t_i - t_{i-1}$ sono le durate dei tratti, e $a_{\max}$ e $a_{\min}$ i livelli di accelerazione positiva e negativa):

Notazione compatta. Per rendere più leggibili le espressioni, si pone

$$s_k \triangleq \sin(\omega t_k), \qquad c_k \triangleq \cos(\omega t_k), \qquad k = 0, \ldots, 7.$$

$$\begin{aligned}\Re\{X_a(\omega)\} = &-\frac{a_{\min} t_6 s_7}{\Delta T_7\,\omega} + \frac{a_{\min} t_7 s_7}{\Delta T_7\,\omega} - \frac{a_{\min} t_5 s_7}{\Delta T_5\,\omega} + \frac{a_{\min} t_4 s_7}{\Delta T_5\,\omega} - \frac{a_{\max} t_3 s_3}{\Delta T_3\,\omega} \\ &+ \frac{a_{\max} t_2 s_3}{\Delta T_3\,\omega} - \frac{a_{\max} t_0 s_3}{\Delta T_1\,\omega} + \frac{a_{\max} t_1 s_3}{\Delta T_1\,\omega} + \frac{a_{\max}}{\omega^2}\left(\frac{c_1}{\Delta T_1} - \frac{c_0}{\Delta T_1}\right) \\ &+ \frac{a_{\max}}{\omega^2}\left(\frac{c_2}{\Delta T_3} - \frac{c_3}{\Delta T_3}\right) + \frac{a_{\min}}{\omega^2}\left(\frac{c_4}{\Delta T_5} - \frac{c_5}{\Delta T_5}\right) + \frac{a_{\min}}{\omega^2}\left(\frac{c_7}{\Delta T_7} - \frac{c_6}{\Delta T_7}\right).\end{aligned} \tag{4.47}$$

$$\begin{aligned}\Im\{X_a(\omega)\} = &-\frac{a_{\max} t_0 c_3}{\Delta T_1\,\omega} + \frac{a_{\max} t_1 c_3}{\Delta T_1\,\omega} - \frac{a_{\max} t_3 c_3}{\Delta T_3\,\omega} + \frac{a_{\max} t_2 c_3}{\Delta T_3\,\omega} - \frac{a_{\min} t_5 c_7}{\Delta T_5\,\omega} \\ &+ \frac{a_{\min} t_4 c_7}{\Delta T_5\,\omega} - \frac{a_{\min} t_6 c_7}{\Delta T_7\,\omega} + \frac{a_{\min} t_7 c_7}{\Delta T_7\,\omega} + \frac{a_{\max}}{\omega^2}\left(\frac{s_0}{\Delta T_1} - \frac{s_1}{\Delta T_1}\right) \\ &+ \frac{a_{\max}}{\omega^2}\left(-\frac{s_2}{\Delta T_3} + \frac{s_3}{\Delta T_3}\right) + \frac{a_{\min}}{\omega^2}\left(-\frac{s_4}{\Delta T_5} + \frac{s_5}{\Delta T_5}\right) + \frac{a_{\min}}{\omega^2}\left(\frac{s_6}{\Delta T_7} - \frac{s_7}{\Delta T_7}\right).\end{aligned} \tag{4.48}$$

A partire da (4.47)–(4.48) si ottiene quindi lo spettro di ampiezza:

$$|X_a(\omega)| = \sqrt{\Re\{X_a(\omega)\}^2 + \Im\{X_a(\omega)\}^2}. \tag{4.49}$$

Una trattazione analoga dello spettro della legge trapezia è riportata anche in [16]. Rispetto a tale impostazione, tuttavia, la formulazione qui ottenuta mantiene esplicitamente la dipendenza da tutte le durate ΔT_i dei singoli tratti e risulta quindi più generale e particolarmente adatta al problema di ottimizzazione considerato nel presente lavoro.

Spettro analitico e confronto con FFT. Le espressioni ricavate in (4.47)–(4.48) forniscono lo *spettro analitico* della legge trapezia standard, ossia una descrizione *continua* di $X_a(\omega)$ (e quindi di $|X_a(\omega)|$) in funzione delle durate dei sette tratti ΔT_i e delle condizioni al contorno del moto. Questa forma chiusa è particolarmente adatta quando lo spettro entra in un problema di ottimizzazione, perché varia in modo regolare al variare dei parametri ΔT_i e non introduce dipendenze numeriche legate a scelte di discretizzazione o campionamento.

In alternativa, lo spettro può essere stimato numericamente campionando il segnale $a(t)$ e calcolando la trasformata discreta tramite *Fast Fourier Transform* (FFT). Tuttavia, la stima effettuata in questa maniera dipende da come il segnale viene acquisito e pre-processato: passo di campionamento Δt (Nyquist/aliasing), durata della finestra T (risoluzione in frequenza $\Delta f \simeq 1/T$), eventuale finestratura e *leakage*, oltre a scelte di *zero-padding*. Questi aspetti, pur essendo gestibili, rendono la misura numerica più dipendente dalle scelte di campionamento rispetto alla descrizione analitica, quando si vuole imporre vincoli/obiettivi direttamente sul contenuto in frequenza.

Nel presente lavoro la FFT è quindi utilizzata come *verifica* (*sanity check*) del calcolo analitico. Per confrontare in modo coerente le ampiezze, si osservi che $X_a(\omega)$ definito dalla trasformata continua (4.16) (integrale nel tempo) ha unità di $[a]\cdot[s]$, mentre le ampiezze FFT sono spesso normalizzate per restituire un'ampiezza nella stessa unità di $a(t)$. Nel caso di spettro *single-sided*, il legame pratico tra le due definizioni può essere espresso nella forma:

$$A_{\mathrm{FFT,ss}}(\omega) \;\approx\; \frac{2}{T}\left|X_a(\omega)\right|, \tag{4.50}$$

dove T è la durata della finestra su cui si calcola la FFT (nel nostro caso, tipicamente la durata del tratto di moto).

4.2.3 Applicazione dello spettro analitico all'ottimizzazione della legge trapezia

Una volta ottenuta l'espressione analitica dello spettro $|X_a(\omega)|$ della legge trapezia, diventa possibile intervenire direttamente sul contenuto in frequenza della forzante responsabile dell'eccitazione del pelo libero. In particolare, l'obiettivo non è soltanto ridurre lo sloshing nel caso nominale, ma aumentare la robustezza del risultato rispetto alle inevitabili incertezze sul modello (ad esempio variazioni del livello di riempimento, dei parametri equivalenti, o dispersione della pulsazione naturale effettiva attorno al valore nominale ω_n). In termini ingegneristici, infatti, una soluzione "ottima" nel caso nominale può risultare poco robusta se ottenuta imponendo cancellazioni molto puntuali o sensitività elevata ai parametri.

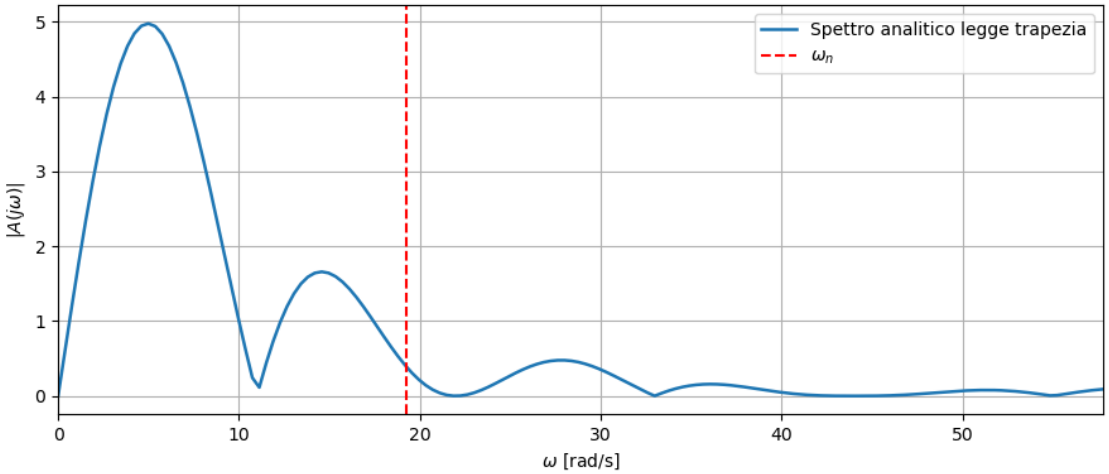


Figura 4.4: Spettro di ampiezza analitico della legge trapezia standard non ottimizzata. La linea verticale evidenzia la pulsazione naturale nominale ω_n associata al modo dominante del pelo libero.

Questa osservazione ha motivato la costruzione di ottimizzatori che agiscono sullo spettro della legge trapezia, ricercando configurazioni dei parametri temporali ΔT_i che riducano l'ampiezza in corrispondenza della risonanza dominante, e più in generale in una banda attorno a ω_n. A titolo illustrativo, la Figura 4.4 mostra lo spettro della legge standard non ottimizzata, in cui l'ampiezza in prossimità di ω_n è significativa; al contrario, le Figure 4.5, 4.6 e 4.7 evidenziano l'effetto dell'intervento spettrale su vari livelli.

In continuità con la formulazione NLP introdotta nel Capitolo 3, i parametri temporali ΔT_i vengono trattati come variabili decisionali: la funzione obiettivo e/o i vincoli vengono costruiti direttamente a partire da $|X_a(\omega)|$ valutato sulle frequenze di interesse. In questo modo, l'ottimizzazione ricerca leggi di moto che, pur mantenendo la stessa struttura a sette tratti compatibile con l'implementazione industriale, presentano un contenuto in frequenza ridotto (o controllato) in corrispondenza dei modi che governano lo sloshing. Come accennato, l'approccio sperimentale è stato articolato su tre livelli di intervento, descritti di seguito.

Metodo 1: ottimizzazione spettrale a notch singolo (strategia nominale). Il primo livello di intervento mira a ridurre l'ampiezza spettrale in corrispondenza della pulsazione naturale nominale ω_n (Figura 4.5). In pratica, l'obiettivo è imporre un'attenuazione localizzata ("notch") di $|X_a(\omega)|$ in prossimità di ω_n, poiché i contributi armonici attorno a tale pulsazione sono quelli che eccitano maggiormente il modo dominante del pelo libero. Una formulazione tipica per la funzione obiettivo è:

$$J_{\mathrm{n}} = |X_a(\omega_n)|^2 = \Re\{X_a(\omega_n)\}^2 + \Im\{X_a(\omega_n)\}^2. \tag{4.51}$$

Il parallelismo con l'*input shaping* è immediato: uno shaper di tipo ZV impone una cancellazione puntuale del contributo alla pulsazione naturale (condizione "nominale"). In questa formulazione l'attenuazione è perseguita *solo* nel dominio della frequenza e non si impongono vincoli diretti sulla grandezza di sloshing $\eta(t)$: l'obiettivo anti-sloshing è quindi *indiretto*, ottenuto agendo sulla forzante.

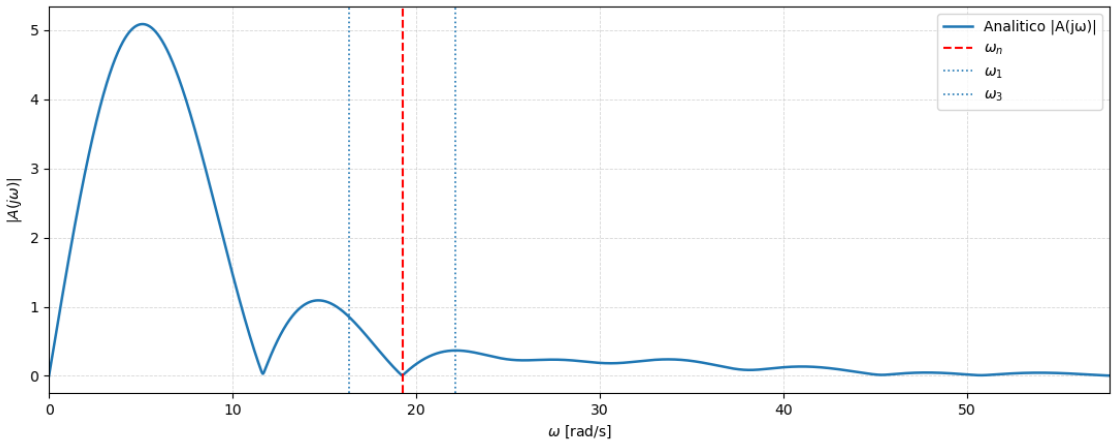


Figura 4.5: Esempio di spettro di ampiezza ottenuto imponendo l'attenuazione in corrispondenza della sola ω_n (strategia "nominale", analoga a uno shaper ZV).

Metodo 2: ottimizzazione spettrale notch multi-zero in banda (strategia robusta). Per ridurre la sensibilità a incertezze su ω_n (variazioni di riempimento, smorzamento, ecc.), si introduce una variante "robusta" in cui l'attenuazione non è richiesta solo nel punto ω_n, ma anche su pulsazioni adiacenti (Figura 4.6). Operativamente ciò equivale a imporre più condizioni di attenuazione, in analogia con gli shaper robusti (ZVD/EI). Una scelta semplice e molto efficace, utilizzata nel presente lavoro, consiste nel considerare tre pulsazioni:

$$\omega_1 = (1 - p_{\text{notch}})\,\omega_n, \qquad \omega_2 = \omega_n, \qquad \omega_3 = (1 + p_{\text{notch}})\,\omega_n, \tag{4.52}$$

dove p rappresenta una tolleranza relativa (incertezza) sulla frequenza naturale. Operativamente, si costruisce una funzione obiettivo che minimizza la somma dei moduli al quadrato:

$$J_{\text{mz}} = b_1|X_a(\omega_1)|^2 + b_2|X_a(\omega_2)|^2 + b_3|X_a(\omega_3)|^2. \tag{4.53}$$

con b_1, b_2, b_3 pesi scelti dall'utente. Lavorare su $|X_a(\omega)|^2 = \Re^2 + \Im^2$ evita la radice e migliora la regolarità numerica dell'obiettivo. Anche in questo caso non si impongono vincoli espliciti su $\eta(t)$: la robustezza è ottenuta *spalmando* l'attenuazione su un intorno della risonanza nominale, riducendo l'impatto di un eventuale disallineamento tra ω_n nominale ed effettiva. La logica può essere consultata in Appendice B.

Metodo 3: minimizzazione del jerk con vincolo spettrale in banda. Una terza strategia introduce esplicitamente un criterio di "qualità del moto". Si va ad agire direttamente sul jerk, *penalizzandolo* in modo da evitare profili troppo aggressivi e migliorare ripetibilità e compatibilità con vincoli meccanici, mantenendo però al contempo un controllo del contenuto in frequenza (Figura 4.7). Una scelta naturale della funzione obiettivo, in questo caso, è la minimizzazione del jerk quadratico:

$$J_{\text{jerk}} = \int_0^T j(t)^2\,dt, \tag{4.54}$$

affiancato da un vincolo di attenuazione spettrale "in banda" attorno a ω_n:

$$|X_a(\omega)| \le \bar{A}, \qquad \forall\,\omega \in \big[(1-\Delta)\omega_n, (1+\Delta)\omega_n\big], \tag{4.55}$$

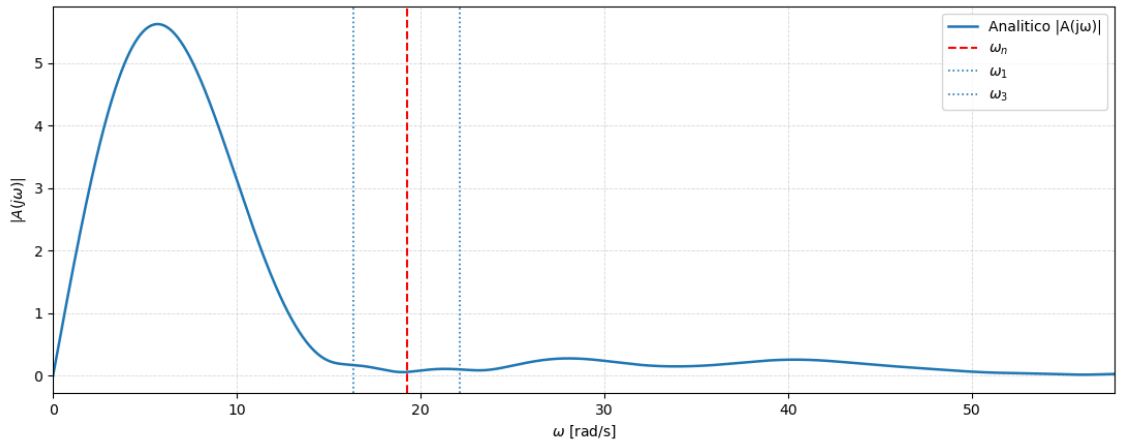


Figura 4.6: Esempio di spettro di ampiezza ottenuto con l'ottimizzazione *multi-zero*: l'ampiezza viene ridotta in corrispondenza di ω_n e su frequenze aggiuntive (p=0.15) attorno alla risonanza nominale per aumentare la robustezza rispetto a incertezze parametriche (strategia "robusta", analoga a shaper ZVD/EI.

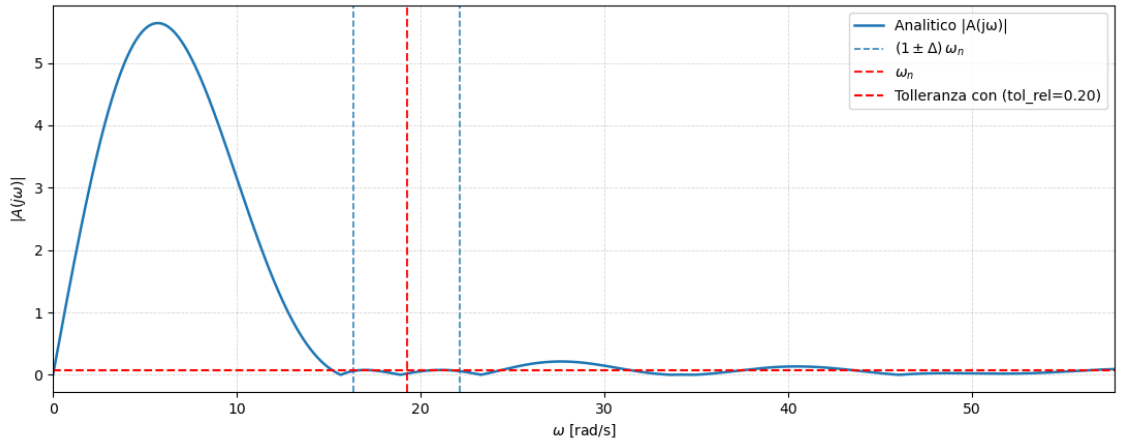


Figura 4.7: Esempio di spettro di ampiezza ottenuto con l'ottimizzazione *min-jerk*: oltre alla riduzione in banda attorno a ω_n, la formulazione integra vincoli/obiettivi di regolarità del profilo di accelerazione, coerenti con limiti di jerk e con l'esigenza di migliorare ripetibilità e qualità del moto.

dove Δ definisce l'ampiezza relativa della banda e $\bar{A}$ è una soglia di tolleranza.Nella pratica numerica, il vincolo continuo (4.55) viene imposto su una griglia di N_{band} punti nella banda:

$$|X_a(\omega_k)|^2 = \Re\{X_a(\omega_k)\}^2 + \Im\{X_a(\omega_k)\}^2 \;\le\; \bar{A}^2, \qquad k = 1, \ldots, N_{\text{band}}. \tag{4.56}$$

Questa formulazione, consultabile in Appendice C, rende esplicito il compromesso tra robustezza e "regolarità" della legge: la minimizzazione del jerk tende a evitare soluzioni troppo aggressive, mentre il vincolo in banda impone un'attenuazione effettiva attorno alla frequenza critica, riducendo la sensibilità a incertezze su ω_n.

In sintesi, la disponibilità dello spettro analitico della trapezia standard consente di trasformare il problema anti-sloshing in un controllo diretto del contenuto in frequenza della forzante, mantenendo una parametrizzazione minimale (i ΔT_i) compatibile con i controlli industriali e coerente con la logica degli input shaper.

Capitolo 5

Test sperimentali e risultati

La validazione sperimentale rappresenta un passaggio essenziale per verificare che i profili di moto ottenuti tramite ottimizzazione producano una effettiva riduzione misurabile dello sloshing in condizioni reali di macchina. In questo capitolo viene descritto il banco prova sviluppato per eseguire test ripetibili su traiettorie monodimensionali e viene introdotta l'impostazione sperimentale adottata (acquisizione video e post-processing). Nei paragrafi successivi saranno poi presentati i risultati, con confronto tra profili non ottimizzati e profili ottimizzati.

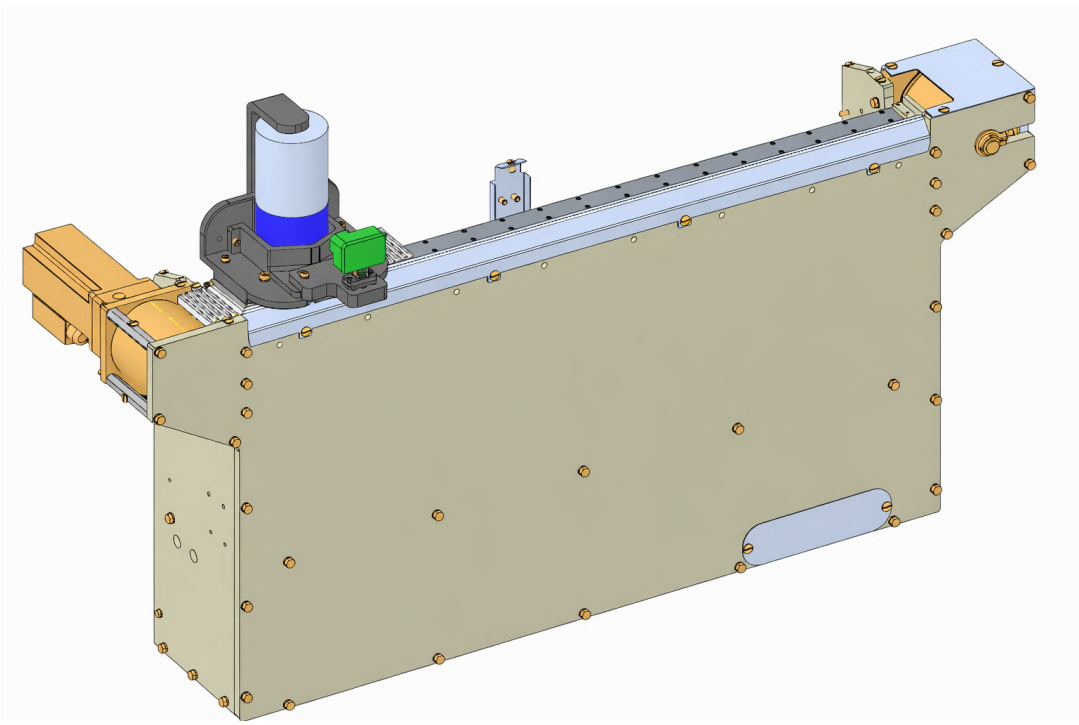


Figura 5.1: Modello CAD del banco prova: asse lineare e supporto recipiente. La geometria è progettata per garantire un moto monodimensionale lungo una corsa definita e per consentire acquisizione video frontale del pelo libero.

5.1 Il banco prova

Per eseguire prove controllate e ripetibili è stato prima progettato al CAD (Figura 5.1) e poi realizzato (Figura 5.2), un banco prova basato su un sistema di movimentazio-

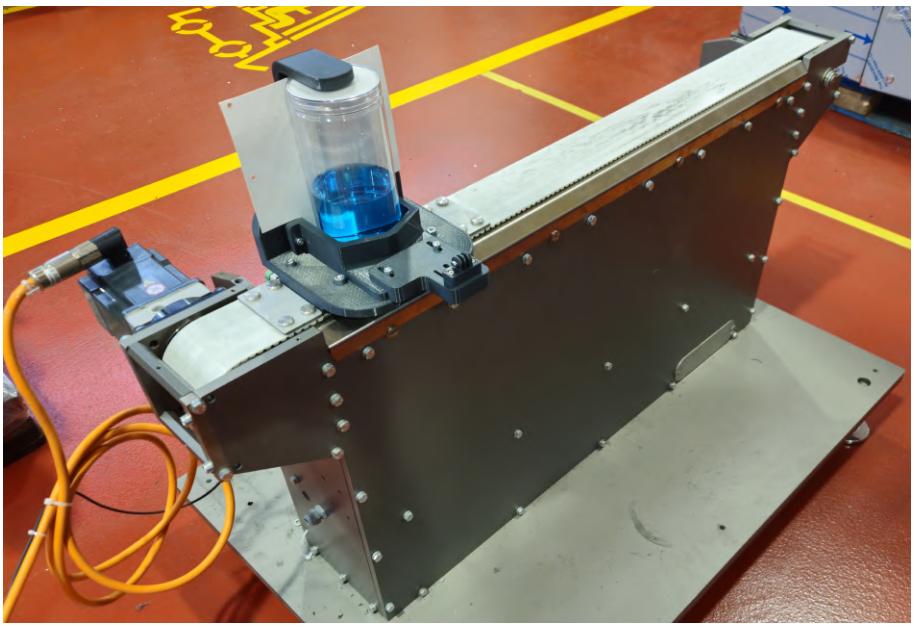


Figura 5.2: Banco prova allestito per i test

ne lineare, concepito per riprodurre fedelmente la tipica forzante che eccita il fenomeno dello *sloshing* nelle macchine automatiche, ovvero l'accelerazione imposta al recipiente lungo un asse di traslazione.

Il banco è stato disegnato a partire dalla *slitta* (guida lineare) utilizzata in azienda sul RoboCombi, uno dei primi robot industriali sviluppati da Marchesini Group per operazioni di handling e movimentazione in linea (ad esempio trasferimento/posizionamento di contenitori lungo stazioni di processo). In questo contesto, l'obiettivo non è replicare l'intera macchina, ma isolare il problema fondamentale: un recipiente parzialmente riempito che viene accelerato e decelerato lungo una corsa nota, con profili di moto programmabili.

Dal punto di vista architetturale, il banco è costituito da un asse lineare, sul cui carrello è fissato un supporto porta-recipienti. La componentistica industriale (sia meccanica che elettrica) consente di realizzare in modo affidabile e ripetibile profili *rest-to-rest* coerenti con quelli trattati nei capitoli precedenti, mantenendo al contempo un livello di eseguibilità comparabile a quello di un'applicazione reale.

La corsa utile del sistema è pari a 800 mm, valore sufficiente a riprodurre cicli di accelerazione e decelerazione rappresentativi del caso di studio. La struttura portante e il basamento sono stati progettati con l'obiettivo di garantire rigidezza e stabilità, in modo da limitare flessioni e vibrazioni parassite che potrebbero interferire con l'osservazione del pelo libero. Il supporto del recipiente è stato realizzato in configurazione modulare e, così come per gli elementi di fissaggio e centraggio (utili per adattare rapidamente il setup a diverse geometrie e facilitare la ripetibilità del posizionamento), si è fatto ricorso alla stampa 3D.

Per la campagna sperimentale riportata in questo elaborato è stato utilizzato un contenitore cilindrico con diametro $D = 97$ mm e un'altezza di riempimento $h = 82$ mm. Come fluido di prova è stata impiegata acqua addizionata con colorante blu: tale scelta è finalizzata ad aumentare il contrasto ottico del pelo libero, rendendo più robusta l'analisi dei filmati in post-processing. L'aggiunta di colorante modifica leggermente le caratteristiche del sistema, comunque, vista la quantità esigua inserita, si può trascurare la variazione nelle proprietà del modello rispetto al caso in cui si utilizzi semplice

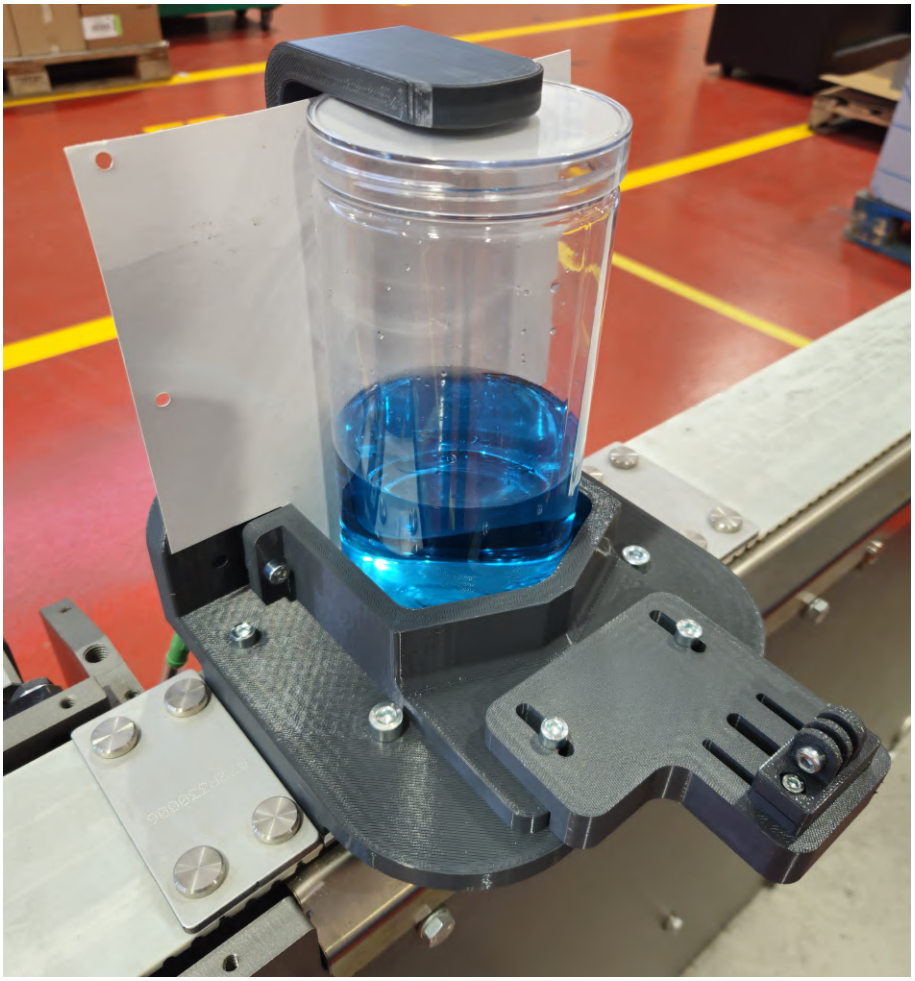


Figura 5.3: Banco prova allestito per i test: contenitore bloccato, tramite i supporti, sul carrello dell'asse lineare e configurazione per acquisizione video. La parete bianca sullo sfondo aumenta il contrasto del pelo libero.

acqua pura. La medesima semplificazione è stata adottata in [8] e [10]. L'acquisizione video è stata eseguita tramite action cam Easypix GoXtreme Vision Duo (4K Ultra HD) che, essendo solidale al carrello grazie ai supporti stampati in 3D, permette di mantenere costante l'inquadratura durante le prove. Per migliorare ulteriormente la qualità dell'estrazione del profilo della superficie libera, è stata inserita una parete bianca di cartoncino sullo sfondo (Figura 5.3), così da ridurre disturbi visivi e rendere più netto il contorno del liquido nei frame.

5.2 Risultati sperimentali

Una volta registrati i video, si passa all'analisi degli stessi in modo da ottenere degli andamenti dell'altezza di sloshing confrontabili con quelli ricavati dalle simulazioni numeriche [19]. A tale scopo è stata impiegata una procedura di post–processing (la stessa usata in [7] sviluppata in MATLAB (Figura 5.4), basata su tecniche di elaborazione immagini e tracciamento fotogramma per fotogramma del pelo libero (Figura 5.5).

In sintesi, il metodo prevede:

- la lettura del video e la conversione dei frame in una rappresentazione adatta

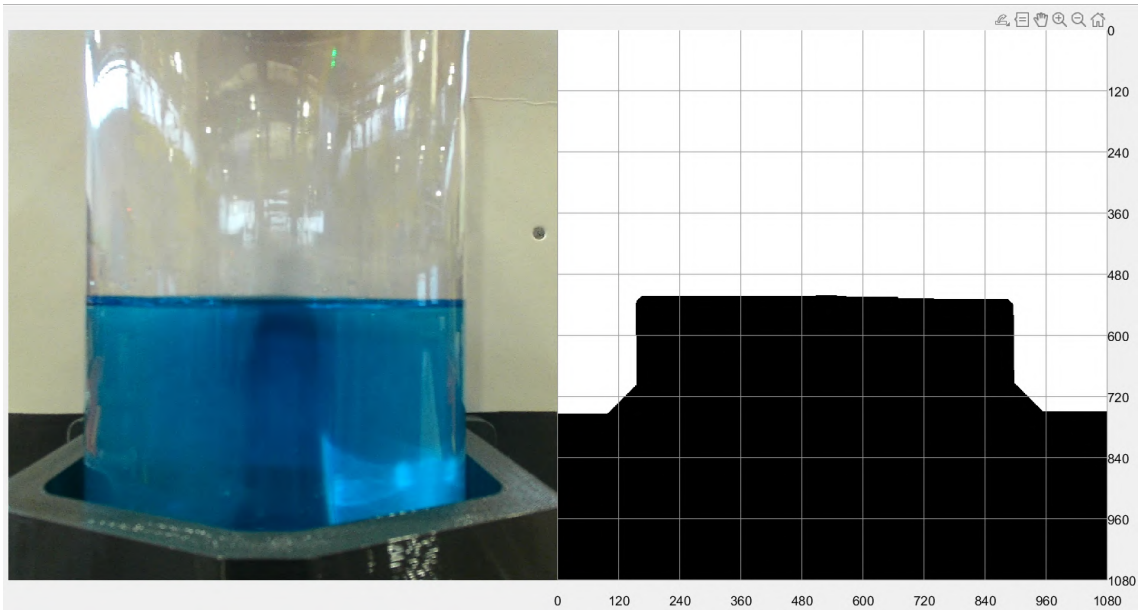


Figura 5.4: Primo step del post-processing: permette di controllare se il contrasto del liquido all'interno del recipiente è riconosciuto correttamente.

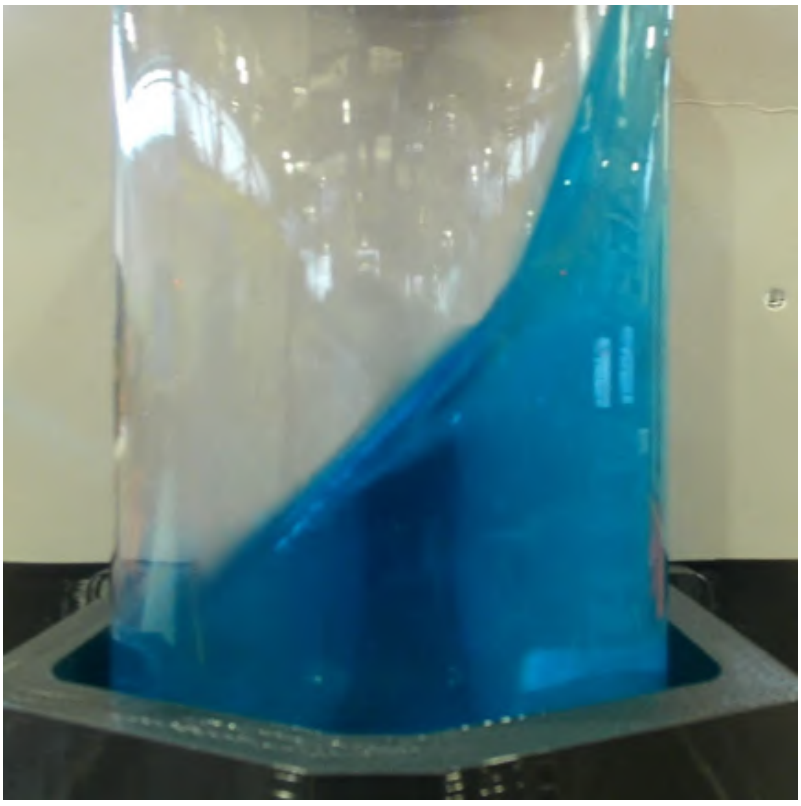


Figura 5.5: Istantanea di una legge di moto non ottimizzata dalla quale si evince che il liquido andrebbe a fuoriuscire dal recipiente se quest'ultimo fosse aperto.

alla segmentazione del liquido (sfruttando il contrasto cromatico introdotto dal colorante);

- l'applicazione di una binarizzazione e di filtri morfologici per ottenere una maschera pulita della regione di interesse;
- l'individuazione della quota del pelo libero in prossimità del bordo del recipiente, in modo coerente e ripetibile lungo tutta la sequenza.

Definita la scala di misura attraverso una grandezza geometrica nota del sistema (ad esempio il diametro del recipiente), l'altezza misurata in pixel viene convertita in millimetri e restituita in uscita come serie temporale $\eta(t)$. Una volta esportati i punti (t,η), questi vengono infine riportati in grafico in MATLAB per confrontare direttamente le prove sperimentali con i risultati di simulazione.

Le misure prese in questa maniera saranno affette da errori inevitabili. I principali fattori che li determinano sono ad esempio: la forma cilindrica del recipiente, che non facilita l'individuazione dell'effettivo punto in cui il pelo libero tocca il contenitore sul piano del moto; o inevitabili piccole vibrazioni che si innescano sulla lamiera di montaggio della fotocamera, che in alcuni frangenti possono determinare una leggera perdita di messa a fuoco. Mentre, altri errori che interessano la discrepanza tra il modello equivalente sviluppato e il sistema fisico reale sono dovuti agli scostamenti tra il posizionamento di riferimento e quello reale del carrello, e alla modifica che i parametri del modello subiscono a causa dell'aggiunta del colorante.
Nonostante questi fattori le misure ricavate risultano essere più che soddisfacenti. Ovviamente, per ottenere precisioni più elevate sarebbe stato necessario disporre di puntatori laser o strumenti di misura dedicati, che tuttavia non è stato possibile reperire nei tempi previsti per il completamento delle prove.

Nel presente progetto, all'interno della procedura di post–processing è stata adottata una scelta operativa leggermente diversa rispetto all'impostazione più "canonica" per un moto rettilineo a 1 g.d.l. In linea di principio, infatti, nel caso di eccitazione puramente traslazionale lungo un asse, la misura più rappresentativa dell'ampiezza di *sloshing* è l'altezza del pelo libero valutata *al bordo del recipiente sul lato di moto* (ad esempio sul lato destro se la direzione positiva dell'accelerazione è verso destra), poiché è in quel punto che si osservano tipicamente le massime escursioni durante la fase di eccitazione.

Durante l'analisi dei filmati, tuttavia, si è scelto di stimare per ciascun frame la *massima* altezza di sloshing lungo l'intero diametro del recipiente (nel piano di osservazione), anziché campionare la quota solo su un bordo. Questa scelta è stata adottata per due motivi principali: (i) sul lato destro del recipiente il software di segmentazione risulta generalmente più affidabile, poiché sono presenti meno zone d'ombra e riflessi legati ai supporti del banco prova; (ii) la grandezza di interesse ai fini del confronto e della validazione è proprio l'escursione massima del pelo libero. Si osservi inoltre che, mentre nel modello simulato il massimo di sloshing è tipicamente localizzato in prossimità della parete laterale, nel caso reale (soprattutto in fase di eccitazione) il picco può non trovarsi esattamente sulla parete laterale, ad esempio per dinamiche in cui si manifesta il rotary sloshing.

Coerentemente con questa scelta, i risultati sperimentali sono stati rappresentati in termini di *valore assoluto* dell'altezza di sloshing, ossia tramite $|\eta(t)|$. Tale rappresentazione perde l'informazione di segno utile a distinguere un innalzamento da un abbassamento rispetto alla quota indisturbata, ma consente di confrontare in modo più

robusto l'ampiezza del fenomeno tra profili di moto diversi, che è l'aspetto di interesse principale ai fini della validazione e del confronto con le simulazioni.

5.2.1 Sequenza sperimentale

Le prove sperimentali sono state condotte eseguendo cicli ripetuti di *andata e ritorno* lungo la guida lineare. Le due corse sono identiche in termini di parametri impartiti alla legge di moto (durate dei tratti e livelli di accelerazione), con l'obiettivo di sollecitare il sistema in modo ripetibile e verificare la stabilità del comportamento nel tempo. Nei grafici e nelle analisi riportate nel presente elaborato, tuttavia, vengono mostrati principalmente i risultati relativi al *primo moto* (la prima corsa), poiché rappresenta la condizione più coerente con l'ipotesi di progetto delle leggi *rest-to-rest*: il pelo libero parte infatti da una configurazione indisturbata e l'oscillazione osservata è attribuibile unicamente alla forzante imposta. Nei cicli successivi, invece, il moto inizia da una condizione già perturbata (residuo del ciclo precedente): se la legge non smorza adeguatamente, possono comparire comportamenti peggiorativi o fenomeni di deriva.

La scelta di registrare sequenze con movimenti ripetuti rimane comunque utile come *stress-test* sperimentale, perché consente di verificare la stabilità del comportamento su più ripetizioni e la presenza di eventuali accumuli di oscillazione. Si sottolinea infine che, nelle applicazioni industriali reali, le movimentazioni non consistono tipicamente in un moto alternato avanti–indietro, ma in trasferimenti tra stazioni; pertanto i cicli ripetuti qui impiegati hanno un ruolo principalmente diagnostico.

La campagna sperimentale è stata inizialmente impostata fissando la corsa $s_{\text{tot}} = 800$ mm e un tempo di avanzamento $t_{\text{avanz}} = 1$ s. A parità di queste specifiche, sono state poi confrontate diverse famiglie di profili di accelerazione, ottenute con strategie di ottimizzazione differenti e descritte dettagliatamente nelle Sezioni 3.4.3 e 4.2.3. Si rimanda per completezza anche alle Appendici A, B e C.

Come prima prova è stata implementata in macchina una legge trapezia *non ottimizzata* (Figura 5.6), utilizzata come baseline di confronto: la parametrizzazione è stata scelta assegnando durate uguali ai sette tratti (*pesi* uniformi, equivalenti a una ripartizione iniziale $\Delta T_i = 1/7$). Questo profilo baseline rappresenta un confronto necessario per quantificare in modo oggettivo il beneficio dei metodi anti-sloshing: in assenza di una progettazione mirata, lo smorzamento delle oscillazioni residue risulta in genere limitato e la risposta può presentare valori di $\eta(t)$ non compatibili con gli obiettivi di riduzione.

Ottimizzazione spettrale in ω_n e in un intorno controllato (Sezione 4.2.3, Appendice B). Il primo livello di intervento (Figura 5.7) adotta la strategia di ottimizzazione nel dominio della frequenza già descritta nel Capitolo 4, Sezione 4.2.3, e formalizzata in Appendice B. Qui si limita a richiamare il fatto che le variabili di decisione sono le durate dei sette segmenti ΔT_i e che l'intervento è costruito direttamente sullo spettro analitico $|X_a(\omega)|$ della forzante: nella versione "notch singolo" la riduzione è imposta in corrispondenza della pulsazione naturale nominale ω_n. I risultati sperimentali sono discussi nel seguito.

Successivamente, per aumentare la robustezza rispetto a possibili incertezze su ω_n, si è applicata la variante "robusta", ossia *multi-notch in banda*, descritta anch'essa nella Sez. 4.2.3 (Appendice B), mantenendo per coerenza sperimentale il parametro $p_{\text{notch}} = 15\%$. Il profilo risultante è riportato in Figura 5.8.

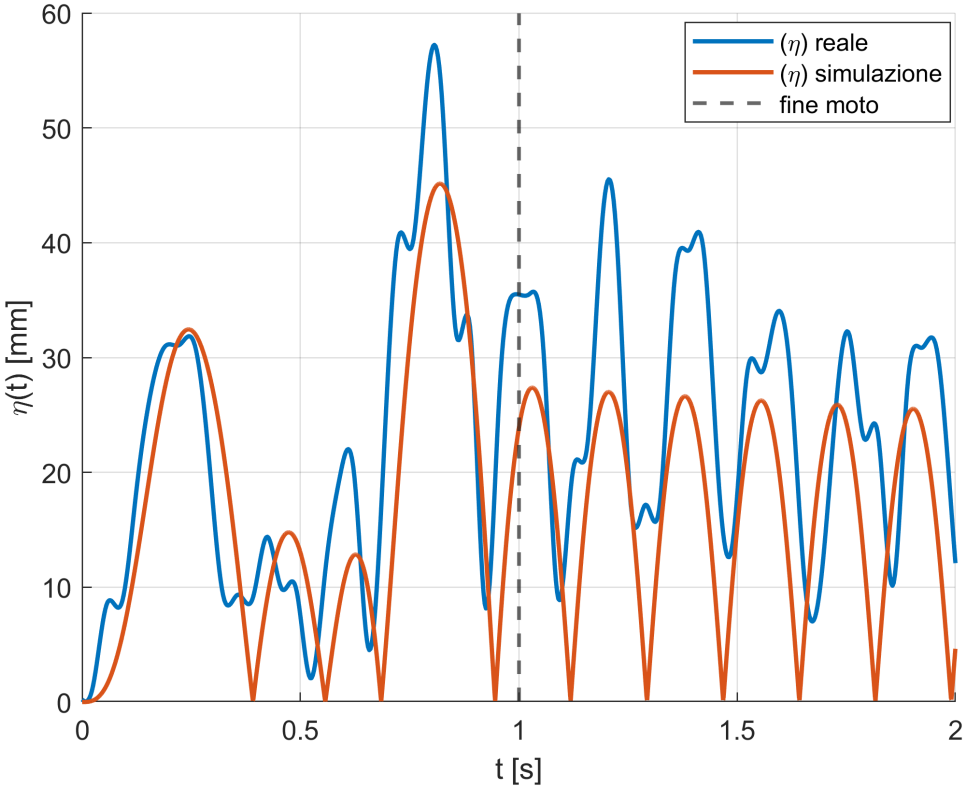


Figura 5.6: Profilo baseline non ottimizzato (ripartizione uniforme dei sette segmenti) per corsa $s_{\text{tot}} = 800$ mm e tempo di avanzamento $t_{\text{avanz}} = 1$ s (ID: `legge_100`).

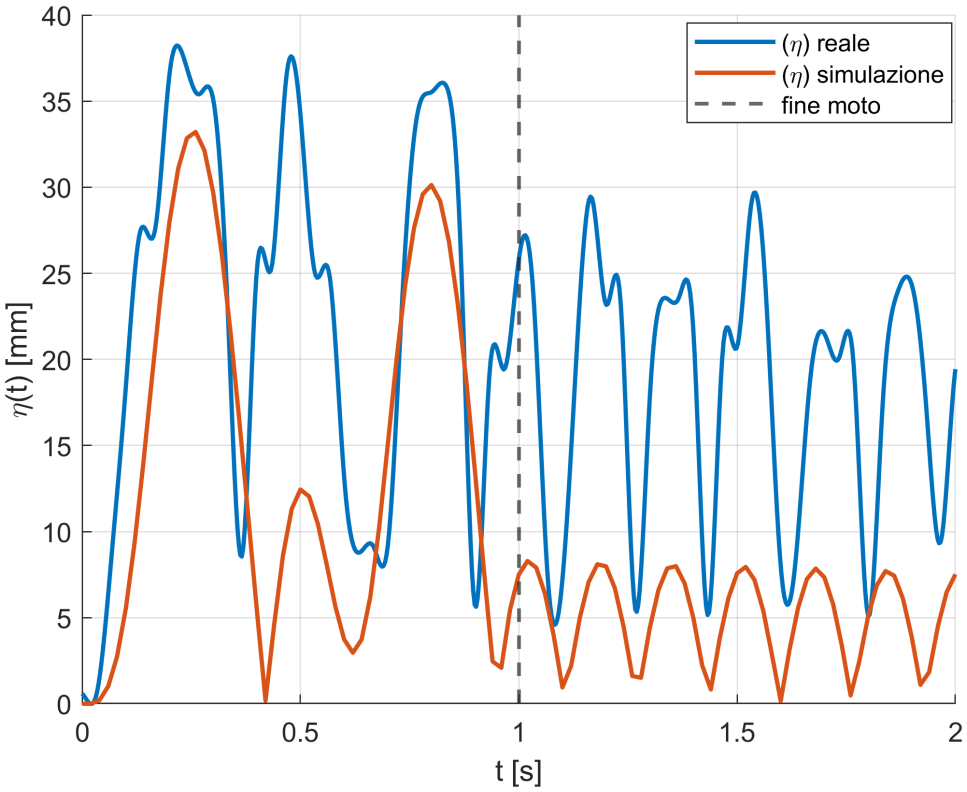


Figura 5.7: Profilo ottimizzato con notch singolo in corrispondenza di ω_n, per corsa $s_{\text{tot}} = 800$ mm e tempo di avanzamento $t_{\text{avanz}} = 1$ s (ID: `legge_1`).

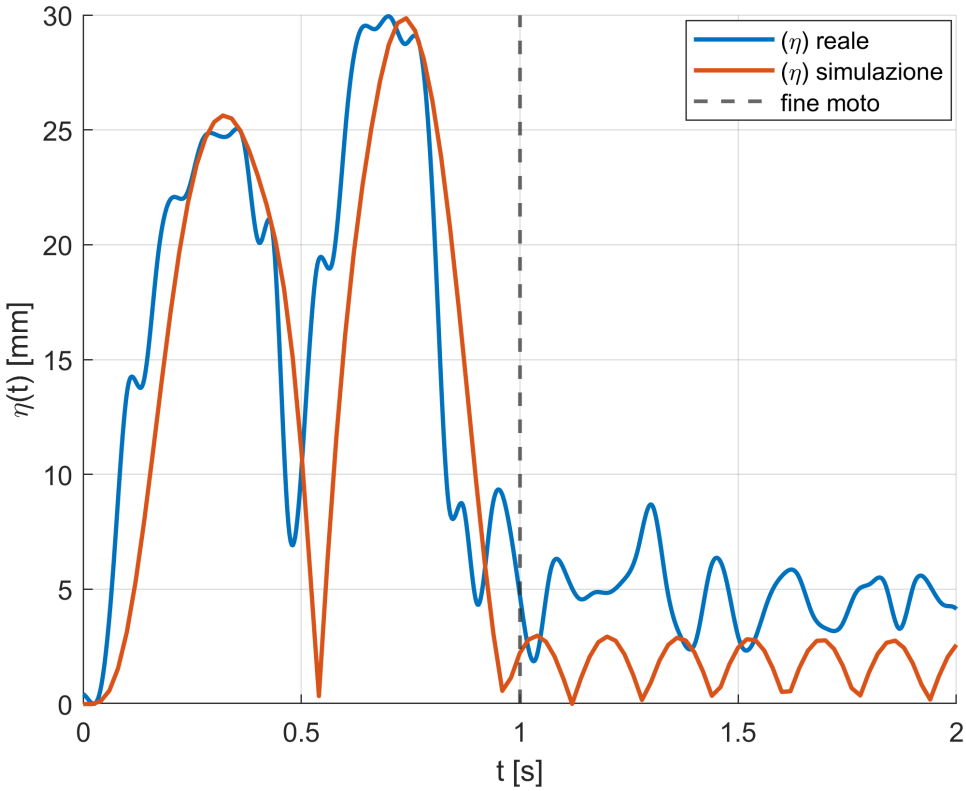


Figura 5.8: Profilo ottimizzato con notch multipli (strategia spettrale robusta in banda), per corsa $s_{\text{tot}} = 800$ mm e tempo di avanzamento $t_{\text{avanz}} = 1$ s: caso rappresentativo 1. (ID: `legge_3`).

Oltre a questo caso, sono state valutate due ulteriori traiettorie ottenute con la stessa logica di attenuazione in banda e con lo stesso valore di $p_{\text{notch}} = 15\%$), ma introducendo interventi mirati sulla parametrizzazione della legge.

In particolare, per entrambe le traiettorie qui discusse si è imposto che il *tratto centrale a accelerazione nulla* (tratto 4) sia *vincolato a zero*, così da eliminare la fase intermedia di sosta e rendere la legge più "compatta" a parità di t_{avanz}. Questa scelta consente di valutare l'effetto dell'ottimizzazione spettrale in condizioni più stringenti, in cui la ripartizione dei tratti deve contemporaneamente rispettare i vincoli cinematici e garantire la riduzione del contenuto in frequenza attorno a ω_n.

La prima variante presenta appunto il vincolo sul tratto 4 e, in aggiunta, introduce limiti sui tratti 2 e 6, imponendo limiti inferiori alle rispettive durate, così da evitare che i plateau di accelerazione si riducano eccessivamente durante l'ottimizzazione. L'attenuazione corrispondente è riportata in Figura 5.9.

La seconda variante mantiene anch'essa il vincolo sul tratto 4, ma rimuove i limiti aggiuntivi per lasciare maggiore libertà alla ricerca. In questo caso, inoltre, è stata accentuata l'azione sul punto ω_n, aumentando il peso ("b_2" nell'eq. 4.53) del termine di costo legato all'ampiezza spettrale in corrispondenza della pulsazione naturale dominante: l'intento è ottenere un notch più pronunciato in ω_n (pur rimanendo nel quadro di attenuazione in banda), a fronte di una minore regolarizzazione della soluzione. Il risultato è riportato in Figura 5.10.

Nel presente lavoro, per la campagna sperimentale è stato fissato il parametro $p_{\text{notch}} = 15\%$, che definisce la *semi-ampiezza relativa* della banda di attenuazione attorno alla pulsazione naturale: in altri termini, l'intervento spettrale viene richiesto non solo in

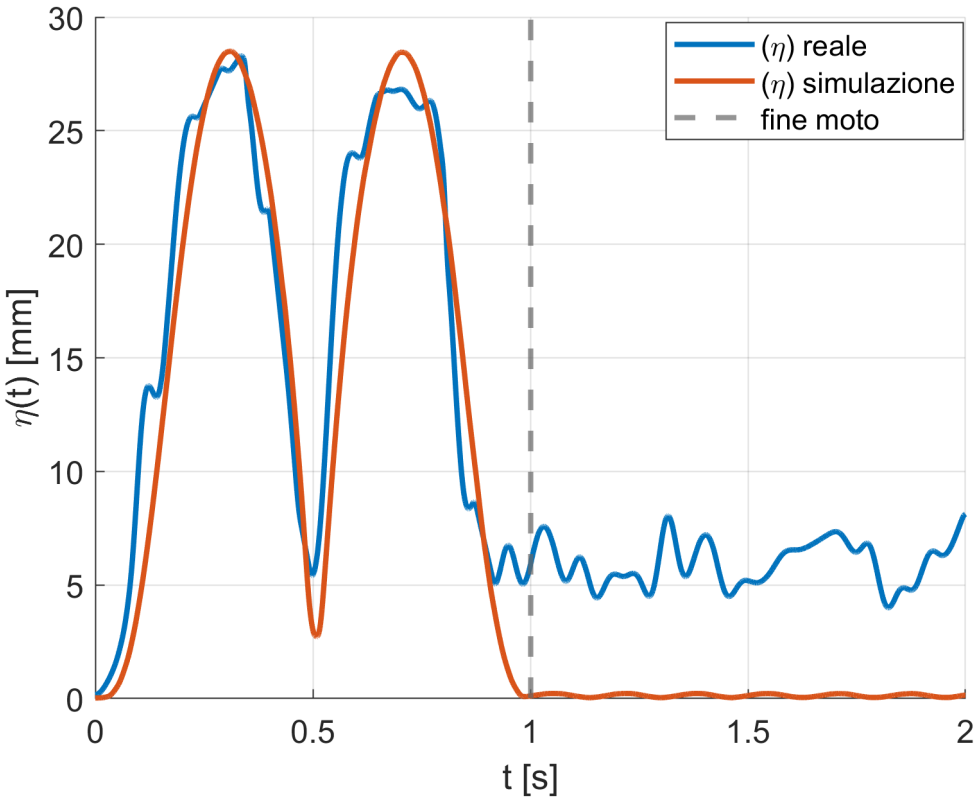


Figura 5.9: Profilo ottimizzato con notch multipli (strategia spettrale robusta in banda), per corsa s_{tot} = 800 mm e tempo di avanzamento t_{avanz} = 1 s: caso rappresentativo 2. (ID: `legge_5`).

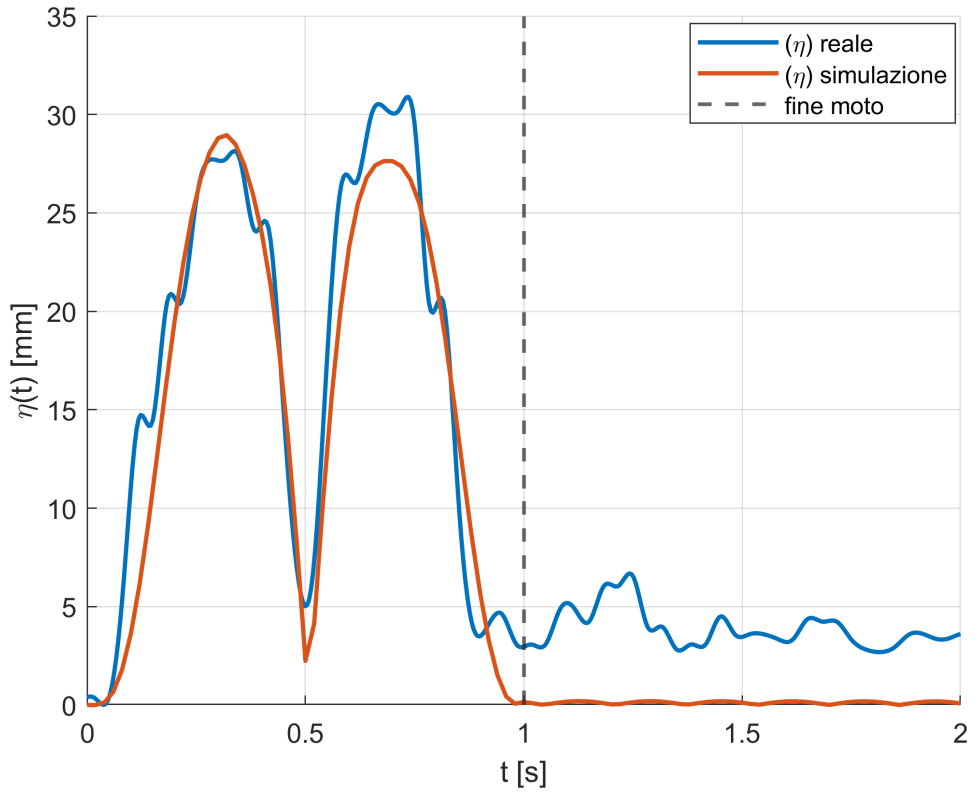


Figura 5.10: Profilo ottimizzato con notch multipli (strategia spettrale robusta in banda), per corsa s_{tot} = 800 mm e tempo di avanzamento t_{avanz} = 1 s: caso rappresentativo 3. (ID: `legge_7`).

ω_n, ma anche nell'intervallo

$$\omega \in \left[(1 - p_{\text{notch}})\,\omega_n,\ (1 + p_{\text{notch}})\,\omega_n\right],$$

così da imporre una riduzione del contenuto in frequenza su un intorno pari complessivamente al 30% di ω_n. Tale scelta rappresenta un compromesso tra robustezza (banda più ampia, minore sensibilità a errori su ω_n) e conservatività, visto che una banda troppo ampia può penalizzare eccessivamente la forma della legge, riducendo i gradi di libertà utili all'ottimizzazione.

Per coerenza metodologica e per rendere confrontabili i risultati, il valore $p_{\text{notch}} = 15\%$ viene mantenuto anche nelle prove successive condotte su tempi di avanzamento inferiori e, ove previsto, su corse inferiori. Al contempo, sono stati effettuati test anche con guess iniziali e percentuali di banda differenti; nel seguito vengono riportate solo le configurazioni più significative e prestazionalmente migliori, in modo da evitare di appesantire la trattazione con varianti marginali e da mantenere il confronto chiaro e leggibile.

Ottimizzazione spettrale con minimizzazione del jerk su banda (Sezione 4.2.3, Appendice C). Successivamente, è stata sperimentata una seconda famiglia di ottimizzazione spettrale, che introduce anche un criterio di regolarità del profilo, penalizzando variazioni troppo brusche e favorendo leggi con jerk più contenuti(Figura 5.11). In questa impostazione la banda attorno a ω_n viene discretizzata in un numero finito di punti: N_{band} rappresenta proprio il *numero di campioni in frequenza* utilizzati per valutare l'obiettivo/vincolo spettrale nella banda, ad esempio $N_{\text{band}} = 51$ punti equispaziati; mentre parametri quali Δ_{freq} e tol_{rel} regolano rispettivamente l'ampiezza della banda e la tolleranza relativa con cui si richiede l'attenuazione. Anche per questa strategia sono state testate diverse inizializzazioni e varianti dei parametri di vincolo, ma si riporta la configurazione con il miglior compromesso tra riduzione dello sloshing e regolarità del profilo.

Ottimizzazione nel dominio del tempo sul modello non lineare (Sezione 3.4.3, Appendice A). In aggiunta alle strategie nel dominio della frequenza, è stata testata anche la procedura di ottimizzazione *time-domain* basata sulla risposta del modello equivalente non lineare, descritta in Sezione 3.4.3. In questa fase sperimentale, si ricorda che la configurazione adottata è caratterizzata dall'imposizione esplicita di un vincolo sulle oscillazioni residue in fase di riposo:

$$\eta_{\text{max}}^{\text{riposo}} \leq 3 \text{ mm},$$

mentre il resto dell'impostazione del problema (parametrizzazione tramite le durate ΔT_i, simulazione della dinamica sotto la forzante $a(t)$ e settaggi numerici del solutore) è mantenuto coerente con quanto definito nel riferimento teorico. Il profilo ottenuto e impiegato nelle prove è mostrato in Figura 5.12. Per coerenza metodologica e per rendere confrontabili i risultati tra campagne di test, il vincolo $\eta_{\text{max}}^{\text{riposo}} \leq 3$ mm viene mantenuto invariato anche nelle prove successive.

Metriche di confronto e sintesi dei risultati. Per ciascuna legge testata, a partire dai dati video post-processati, sono stati estratti in MATLAB i massimi di interesse in due fasi: durante il moto (picco in avanzamento) e durante la sosta (oscillazione residua).

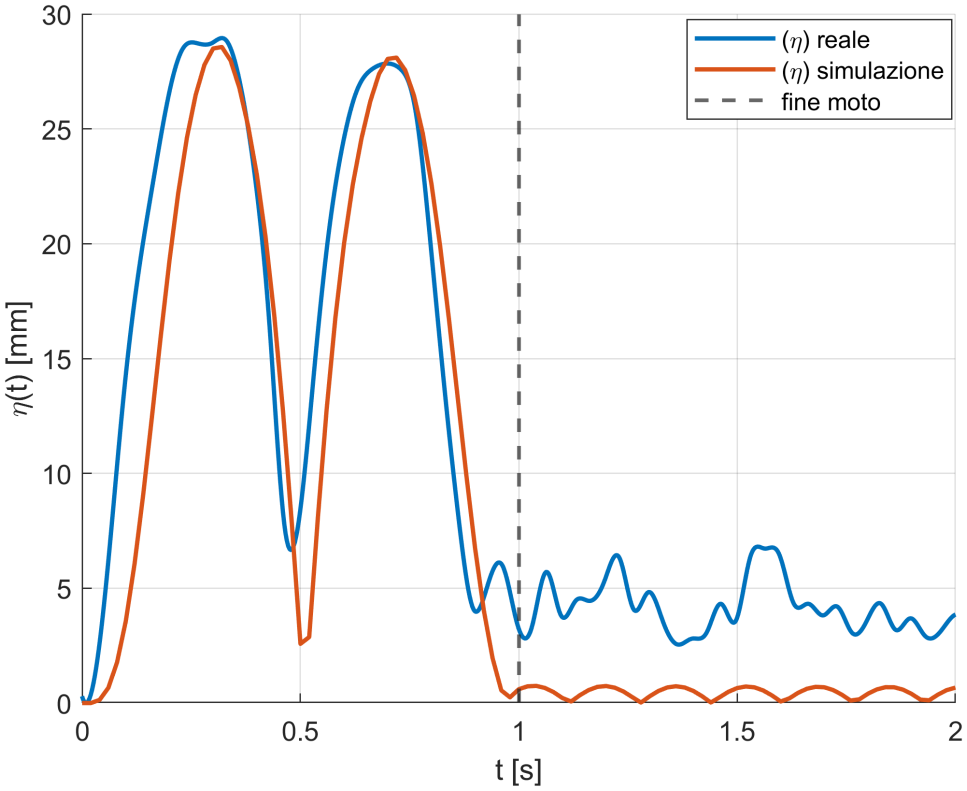


Figura 5.11: Profilo ottimizzato con criterio spettrale e penalizzazione del jerk per corsa $s_{\text{tot}} = 800$ mm e tempo di avanzamento $t_{\text{avanz}} = 1$ s (ID: `legge_10`).

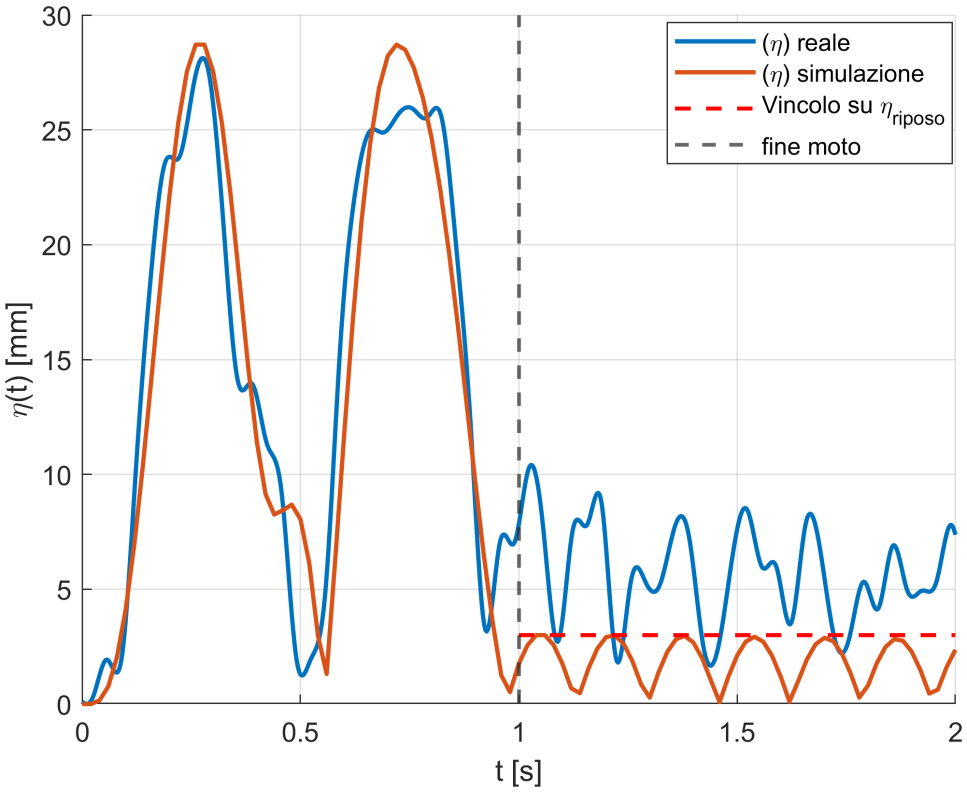


Figura 5.12: Profilo ottimizzato nel dominio del tempo sulla risposta del modello non lineare, con vincolo sulle oscillazioni residue in riposo, per corsa $s_{\text{tot}} = 800$ mm e tempo di avanzamento $t_{\text{avanz}} = 1$ s (ID: `legge_22`).

Tali valori sono stati poi confrontati con quelli della legge baseline non ottimizzata, calcolando miglioramenti percentuali separatamente per fase di moto e fase di riposo. La sintesi numerica delle leggi testate a $t_{\text{avanz}} = 1$ s è riportata nella tabella riassuntiva 5.1.

Tabella 5.1: Risultati sperimentali per corsa $s_{\text{tot}} = 800$ mm e $t_{\text{avanz}} = 1.0$ s. I miglioramenti percentuali sono calcolati rispetto alla traiettoria baseline (trapezia standard a 7 tratti con ripartizione uniforme; ID: `legge_100`) e sono rappresentati come riduzioni.

Traiettoria (caratteristiche principali)	η_{moto} [mm]	η_{riposo} [mm]	$\Delta\eta_{\text{moto}}$ [%]	$\Delta\eta_{\text{riposo}}$ [%]
Trapezia standard (baseline) ID: `legge_100`	57.2	45.5	–	–
Notch singolo in ω_n ID: `legge_1`	38.2	29.7	−33.2	−34.7
Notch multipli in banda, $p_{\text{notch}} = 15\%$ caso 1 ID: `legge_3`	29.9	8.7	−47.7	−80.9
Notch multipli in banda, $p_{\text{notch}} = 15\%$ caso 2 ID: `legge_5`	28.3	8.3	−50.6	−81.8
Notch multipli in banda, $p_{\text{notch}} = 15\%$ caso 3 ID: `legge_7`	30.9	6.7	−46.0	−85.3
Min-jerk in banda, $\Delta f = 15\%$, $\text{tol}_{\text{rel}} = 20\%$, $N_{\text{band}} = 51$ ID: `legge_10`	29.0	6.8	−49.4	−85.0
Ottimizzazione nel tempo sul modello non lineare, $\eta_{\text{rest}} \leq 3$ mm ID: `legge_22`	28.1	10.4	−50.8	−77.1

5.2.2 Prove con t_{avanz} inferiori

Dopo la campagna a $s_{\text{tot}} = 800$ mm e $t_{\text{avanz}} = 1.0$ s, si è scelto di ridurre progressivamente il tempo di avanzamento a parità di corsa, passando a $t_{\text{avanz}} = 0.9$ s e successivamente a $t_{\text{avanz}} = 0.8$ s. Questa scelta consente di verificare se le strategie proposte mantengono efficacia anche in condizioni più "spinose", caratterizzate da accelerazioni più elevate e quindi da un'eccitazione potenzialmente più marcata dei modi di sloshing. Per rendere i confronti metodologicamente significativi, si è deciso di riutilizzare i medesimi settaggi "migliori" individuati nel caso $t_{\text{avanz}} = 1.0$ s, replicandoli alle nuove condizioni (stessa corsa, ma tempo ridotto).

Caso $t_{\text{avanz}} = 0.9$ s. Anche per questa campagna viene introdotta una traiettoria *baseline* non ottimizzata, ottenuta con ripartizione uniforme dei sette tratti, che fornisce il riferimento per la valutazione dei miglioramenti percentuali. Successivamente sono state replicate le configurazioni più rappresentative emerse nel caso a 1 s, lasciando invariati tutti i parametri (consultabili in Tabella 5.2) utilizzati proprio per quel caso:

(i) strategia spettrale con notch singolo in corrispondenza di ω_n; (ii) strategia spettrale "robusta" con attenuazione imposta anche su frequenze adiacenti a ω_n; (iii) strategia min-jerk in banda; (iv) ottimizzazione nel dominio del tempo basata sul modello non lineare. Le Figure 5.13–5.17 riportano i profili di accelerazione associati alle leggi considerate.

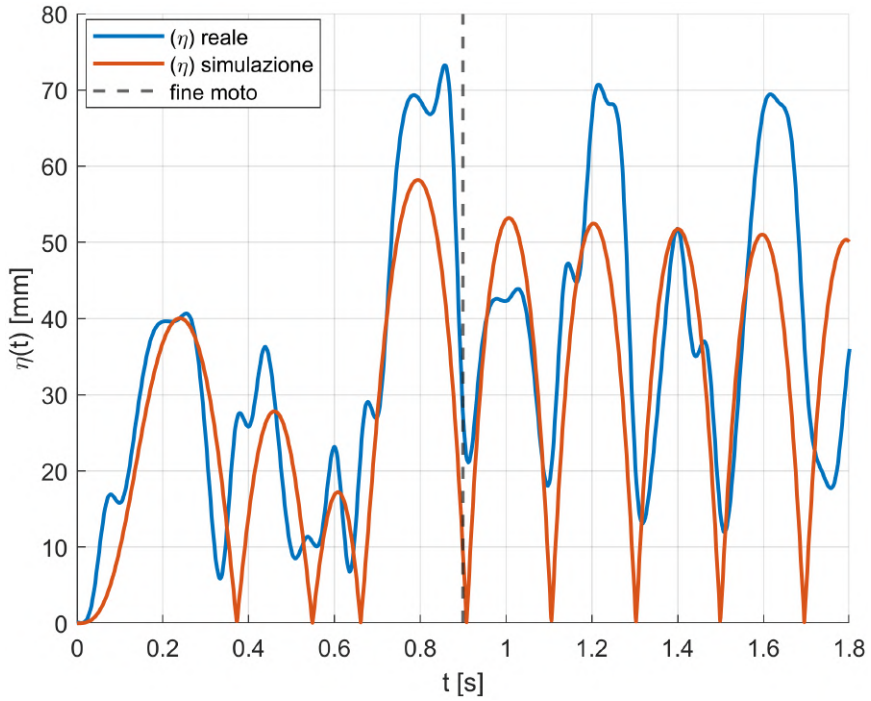


Figura 5.13: Profilo baseline non ottimizzato per corsa $s_{\text{tot}} \approx 800$ mm e tempo di avanzamento $t_{\text{avanz}} = 0.9$ s (ripartizione uniforme dei sette segmenti; ID: `legge_900`).

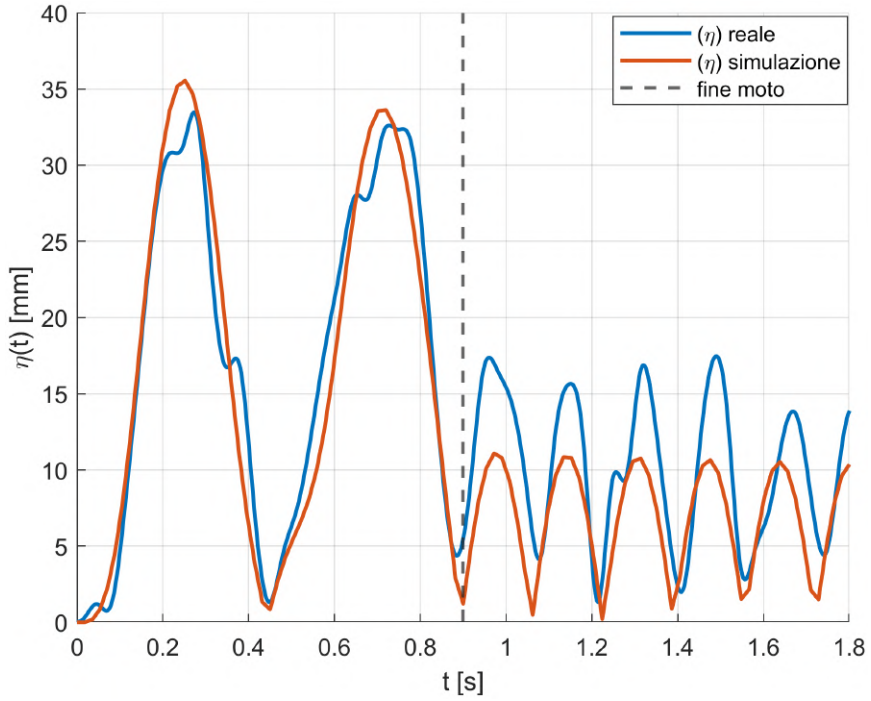


Figura 5.14: Profilo ottimizzato con notch singolo in corrispondenza di ω_n per corsa $s_{\text{tot}} \approx 800$ mm e $t_{\text{avanz}} = 0.9$ s (ID: `legge_36`).

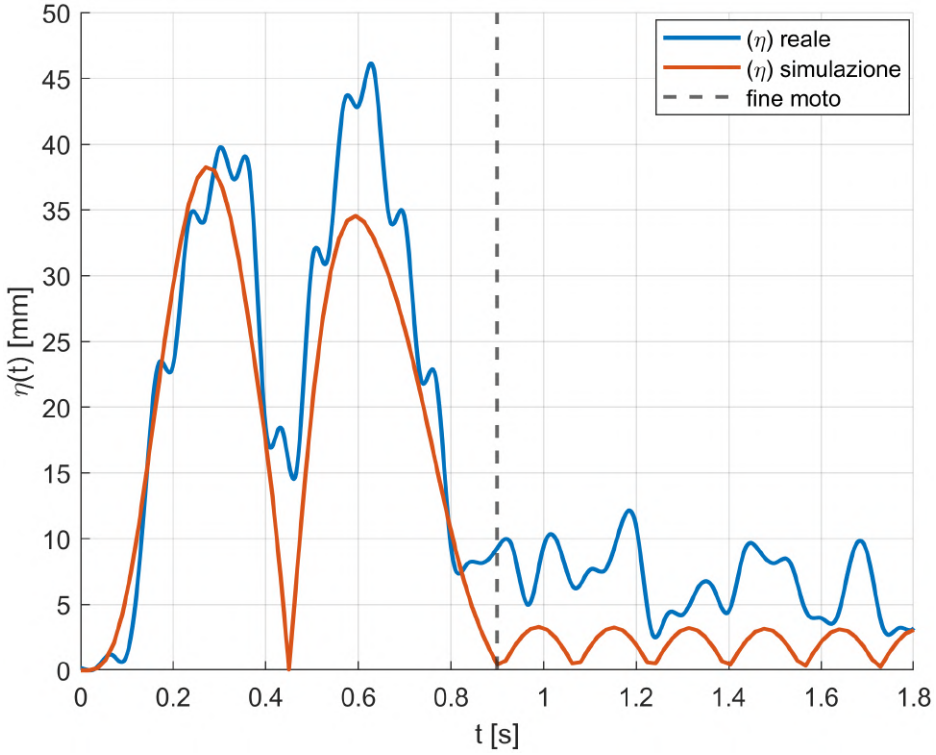


Figura 5.15: Profilo ottimizzato con strategia spettrale robusta (attenuazione su ω_n e frequenze adiacenti) per corsa $s_{\text{tot}} \approx 800$ mm e $t_{\text{avanz}} = 0.9$ s (ID: `legge_40`).

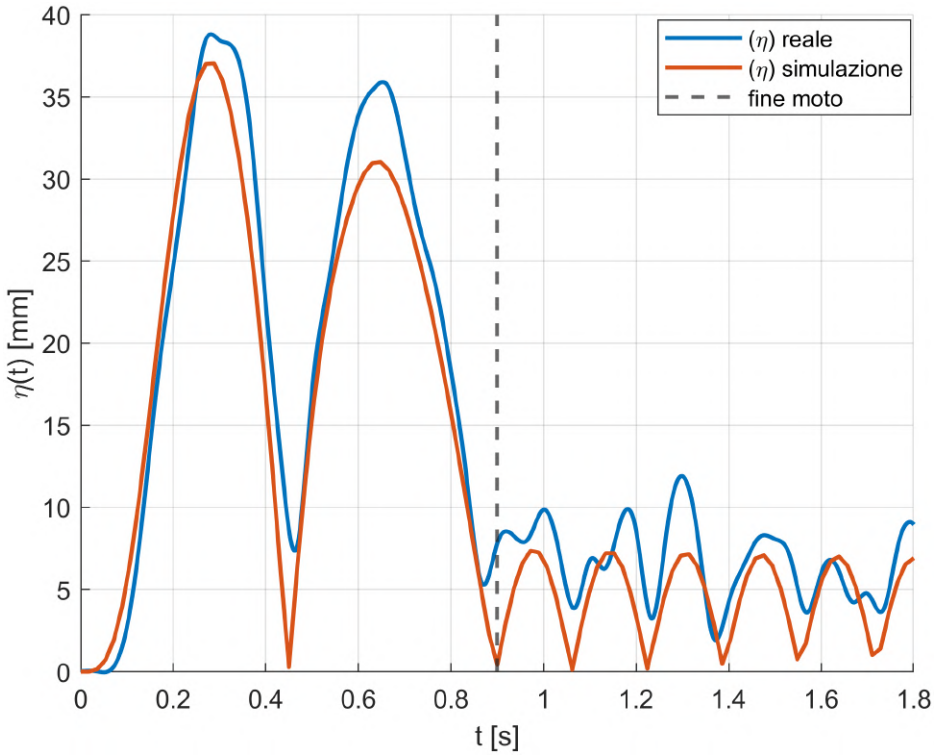


Figura 5.16: Profilo ottimizzato con strategia min-jerk in banda per corsa $s_{\text{tot}} \approx 800$ mm e $t_{\text{avanz}} = 0.9$ s (ID: `legge_41`).

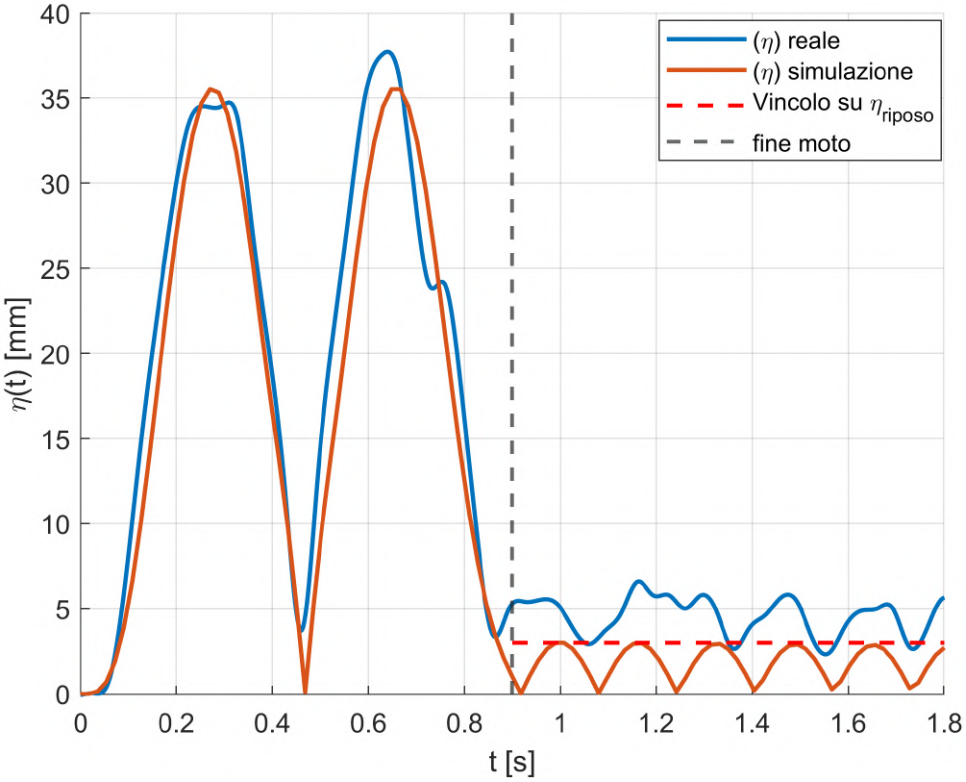


Figura 5.17: Profilo ottimizzato nel dominio del tempo sulla risposta del modello non lineare per corsa $s_{\text{tot}} \approx 800$ mm e $t_{\text{avanz}} = 0.9$ s (ID: `legge_50`).

Tabella 5.2: Risultati sperimentali per corsa $s_{\text{tot}} = 800$ mm e $t_{\text{avanz}} = 0.9$ s. I miglioramenti percentuali sono calcolati rispetto alla traiettoria baseline (trapezia standard a 7 tratti con ripartizione uniforme; ID interno: `legge_900`) e sono rappresentati come riduzioni.

Traiettoria (caratteristiche principali)	η_{moto} [mm]	η_{riposo} [mm]	$\Delta\eta_{\text{moto}}$ [%]	$\Delta\eta_{\text{riposo}}$ [%]
Trapezia standard (baseline) ID: `legge_900`	73.3	70.7	–	–
Notch singolo in ω_n ID: `legge_36`	33.5	17.5	−54.3	−75.3
Notch multipli in banda, $p_{\text{notch}} = 15\%$ ID: `legge_40`	46.1	12.1	−37.0	−82.8
Min-jerk in banda, $\Delta f = 15\%$, $\text{tol}_{\text{rel}} = 20\%$, $N_{\text{band}} = 51$ ID: `legge_41`	38.8	11.9	−47.0	−83.2
Ottimizzazione nel tempo sul modello non lineare, $\eta_{\text{rest}} \leq 3$ mm ID: `legge_50`	37.7	6.6	−48.5	−90.7

Caso $t_{avanz} = 0.8$ s. Riducendo ulteriormente il tempo a $t_{avanz} = 0.8$ s, l'obiettivo è verificare la tenuta dei metodi in una condizione ancora più dinamica. Come per il caso precedente, si definisce una baseline a pesi uniformi e si replicano le migliori configurazioni emerse nel caso a 1 s, mantenendo invariati i parametri di tuning: strategia spettrale direttamente robusta senza nuovamente fissare il confronto con il notch singolo, min-jerk in banda e ottimizzazione nel dominio del tempo sul modello non lineare. Per rendere il confronto coerente con le prove precedenti, si analizzano le stesse grandezze sintetiche già introdotte: i massimi di $\overline{\eta}(t)$ durante il moto e in fase di riposo. Le Figure 5.18–5.21 mostrano, per ciascun profilo, l'andamento temporale della quota di sloshing misurata e quella ottenuta in simulazione; anche qui la linea verticale tratteggiata indica l'istante di fine moto e consente di distinguere chiaramente le due fasi. La Tabella 5.3 riassume i valori massimi e i miglioramenti percentuali rispetto alla baseline, permettendo una lettura immediata dell'efficacia relativa delle strategie in una condizione caratterizzata da dinamiche più spinte.

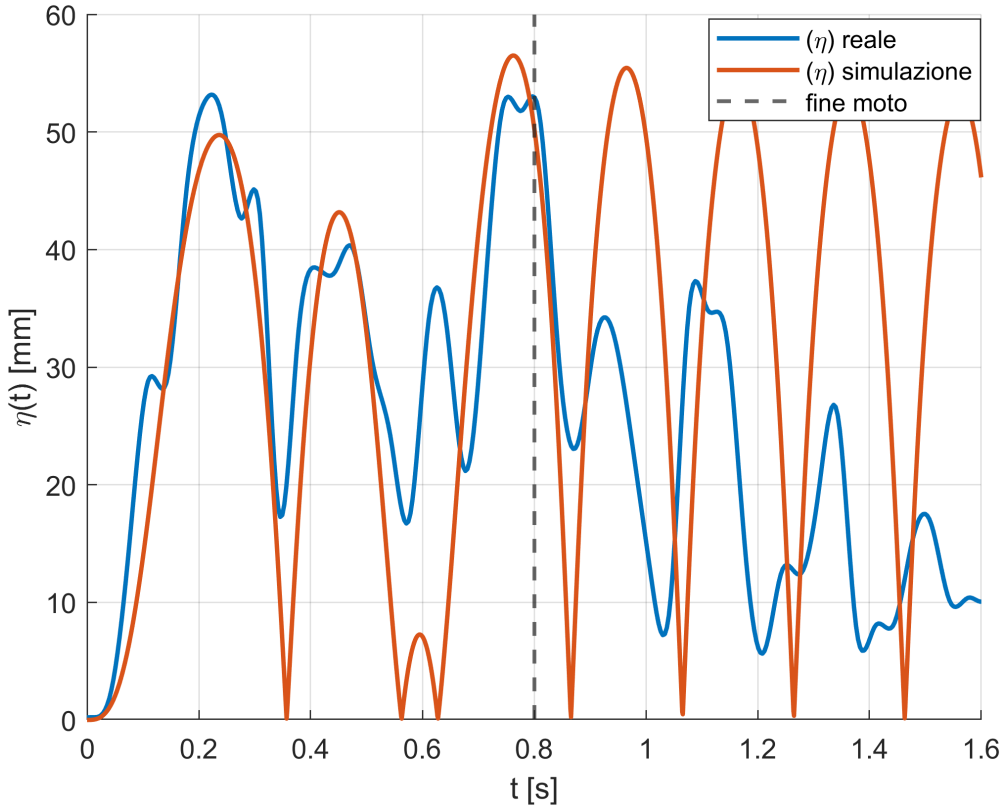


Figura 5.18: Profilo baseline non ottimizzato per corsa $s_{tot} \approx 800$ mm e tempo di avanzamento $t_{avanz} = 0.8$ s (ripartizione uniforme dei sette segmenti; ID: `legge_800`).

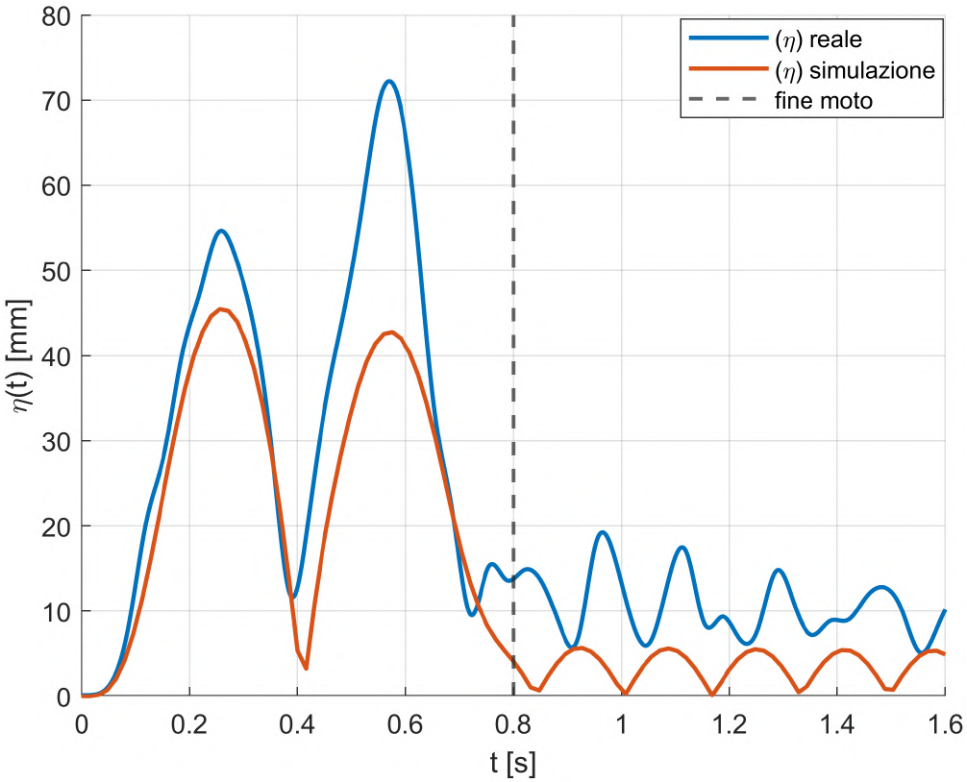


Figura 5.19: Profilo ottimizzato con strategia spettrale robusta (attenuazione in banda attorno a ω_n) per corsa $s_{\text{tot}} \approx 800$ mm e $t_{\text{avanz}} = 0.8$ s (ID: `legge_54`).

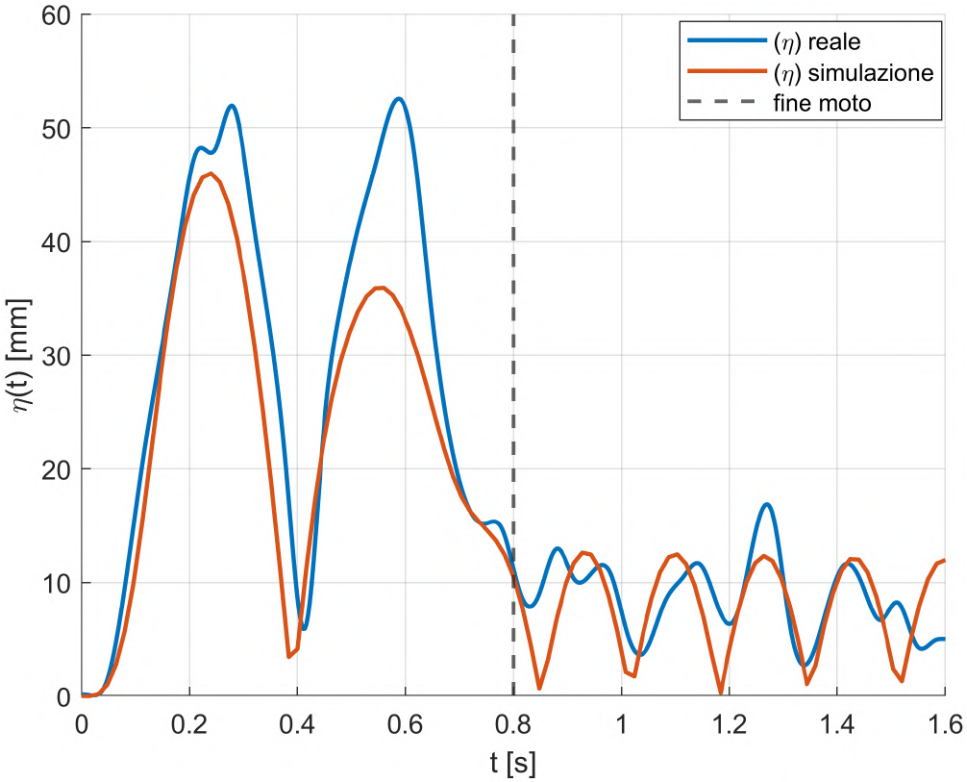


Figura 5.20: Profilo ottimizzato con strategia min-jerk in banda per corsa $s_{\text{tot}} \approx 800$ mm e $t_{\text{avanz}} = 0.8$ s (ID: `legge_51`).

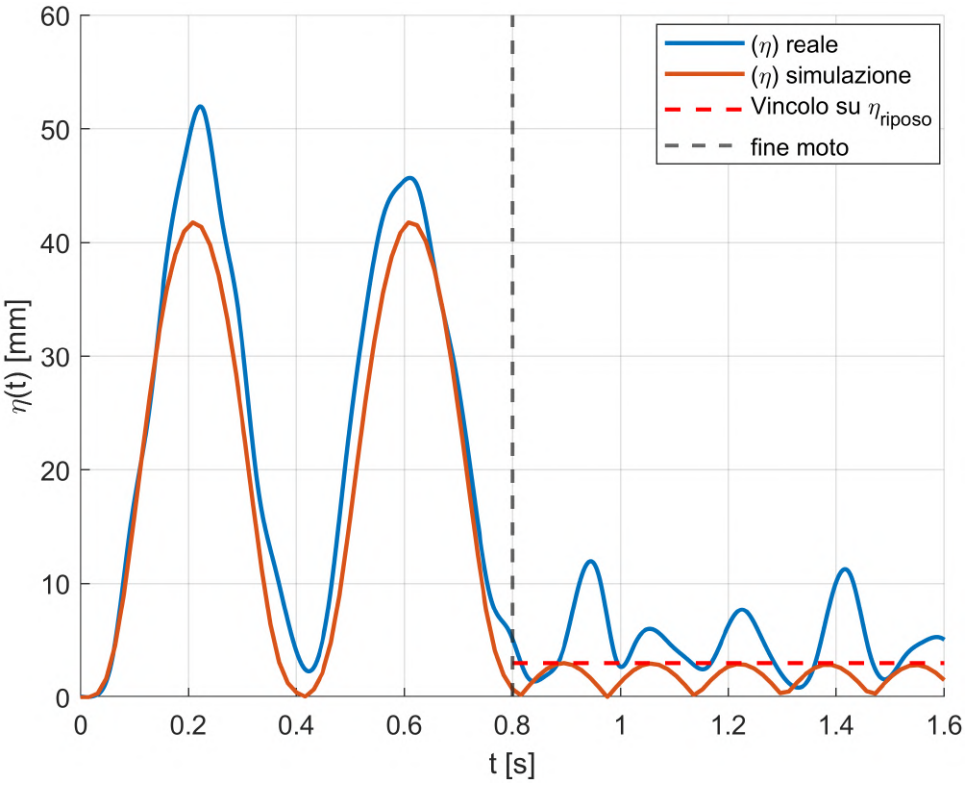


Figura 5.21: Profilo ottimizzato nel dominio del tempo sulla risposta del modello non lineare per corsa $s_{\text{tot}} \approx 800$ mm e $t_{\text{avanz}} = 0.8$ s (ID: `legge_57`).

Tabella 5.3: Risultati sperimentali per corsa $s_{\text{tot}} = 800$ mm e $t_{\text{avanz}} = 0.8$ s. I miglioramenti percentuali sono calcolati rispetto alla traiettoria baseline (trapezia standard a 7 tratti con ripartizione uniforme; ID interno: `legge_800`) e sono rappresentati come riduzioni.

Traiettoria (caratteristiche principali)	η_{moto} [mm]	η_{riposo} [mm]	$\Delta\eta_{\text{moto}}$ [%]	$\Delta\eta_{\text{riposo}}$ [%]
Trapezia standard (baseline) ID: `legge_800`	53.2	52.9	–	–
Notch multipli in banda, $p_{\text{notch}} = 15\%$ ID: `legge_54`	72.3	19.2	35.9	−63.7
Min-jerk in banda, $\Delta f = 15\%$, $\text{tol}_{\text{rel}} = 20\%$, $N_{\text{band}} = 51$ ID: `legge_51`	52.5	16.9	−1.2	−68.1
Ottimizzazione nel tempo sul modello non lineare, $\eta_{\text{rest}} \leq 3$ mm ID: `legge_57`	52.0	12.0	−2.3	−77.4

5.2.3 Prove su corsa ridotta: $s_{\text{tot}} = 500$ mm

Conclusa la campagna su corsa $s_{\text{tot}} = 800$ mm, si è ritenuto utile testare i metodi anche in condizioni differenti, riducendo la corsa e, in modo coerente con le esigenze di automazione, riducendo proporzionalmente il tempo di avanzamento. L'obiettivo di questa fase è valutare se le strategie risultate più efficaci nelle condizioni precedenti mantengano un comportamento migliorativo anche quando la dinamica diventa più spinta.

In prima battuta, sono state replicate le tre condizioni migliori emerse dai test a $t_{\text{avanz}} = 0.8$ s (i.e. notch multipli in banda, min-jerk con vincolo spettrale in banda e ottimizzazione nel dominio del tempo sul modello non lineare), applicandole ad una corsa di $s_{\text{tot}} = 500$ mm e a un tempo di avanzamento di $t_{\text{avanz}} = 0.65$ s. In questa configurazione si è riscontrato che l'elevata "spinta" della manovra porta a un livello di sollecitazione per cui i margini di mitigazione si riducono sensibilmente: gli interventi di ottimizzazione non risultano più efficaci in modo ripetibile e il beneficio rispetto alla traiettoria di riferimento tende a degradare. Le Figure 5.22–5.25 riportano i profili di accelerazione associati alle leggi considerate, mentre la Tabella 5.4 riassume i valori massimi e i miglioramenti percentuali rispetto alla baseline.

Influenza della durata del moto sui limiti di mitigazione. Dall'analisi complessiva delle prove sperimentali e numeriche, emerge un trend ricorrente: a parità di corsa imposta s_{tot}, riducendo il tempo di esecuzione della traiettoria il fenomeno dello *sloshing* diventa progressivamente più difficile da mitigare, e la corrispondenza tra risposta prevista dal modello e misura sperimentale tende a degradare. Un comportamento analogo si osserva sia con $s_{\text{tot}} = 800$ mm di corsa e tempi di avanzamento inferiori a 0.8 s, sia (in modo più marcato) nelle prove su corsa ridotta $s_{\text{tot}} = 500$ mm quando il tempo di avanzamento scende sotto 0.7 s.

Questa evidenza è coerente con un'osservazione di scala: per moto *rest-to-rest* su corsa fissata, una riduzione della durata complessiva T richiede, in media, profili più "aggressivi", con valori caratteristici di accelerazione e jerk che crescono rapidamente al diminuire di T. All'aumentare di tali sollecitazioni, la forzante $a(t)$ trasferisce più energia al fluido in tempi più brevi, favorendo l'eccitazione dei modi dominanti e, più in generale, di componenti dinamiche non pienamente descritte dal modello equivalente adottato (ad esempio contributi modali superiori, effetti tridimensionali, evoluzioni della superficie libera più complesse, fino a regimi con non linearità più marcata). A supporto di questa interpretazione, si menziona il fatto che i picchi di accelerazione sono dell'ordine di 7 m/s^2 per le prove a 500 mm in 0.65 s, contro circa 5 m/s^2 per 500 mm in 0.7 s: la maggiore sollecitazione si riflette in una ridotta "ottimizzabilità" del fenomeno, con benefici che tendono a ridursi rispetto ai casi meno spinti.

Dal punto di vista dell'ottimizzazione, ciò si traduce in un compromesso strutturale tra tempo ciclo e riduzione dello sloshing: restringendo il tempo disponibile, lo spazio delle soluzioni ammissibili si riduce (vincoli cinematici e di regolarità diventano più stringenti) e l'obiettivo basato su $\eta(t)$ presenta una diminuzione di efficacia "raggiungibile". In altre parole, oltre una certa aggressività del profilo, anche una legge ottimizzata può non riuscire a ridurre significativamente l'ampiezza di sloshing rispetto a profili standard, perché il limite dominante non è più solo l'ottimizzatore, ma la combinazione di vincoli di macchina e fisica del fenomeno.

Per questi motivi, le durate delle leggi di moto considerate nella campagna sperimentale sono state scelte in un intervallo che consenta, da un lato, di generare oscillazioni misurabili e significative, e dall'altro di evitare condizioni troppo estreme in cui la misura e la modellazione risultano meno affidabili. Tale scelta riflette inoltre un vincolo tipico delle applicazioni industriali: le movimentazioni devono comunque essere eseguite in tempi contenuti, quindi è importante valutare l'efficacia dei profili anti-sloshing proprio in un regime di moto "veloce" ma realisticamente eseguibile.

Per queste ragioni, si è scelto di ripetere la campagna mantenendo la corsa ridotta a 500 mm ma con un tempo di avanzamento leggermente meno aggressivo ($t_{\text{avanz}} = 0.7$ s), così da collocarsi in una condizione sperimentale più rappresentativa e maggiormente gestibile. Anche in questo caso sono state testate: (i) una traiettoria baseline non ottimizzata, (ii) una legge ottenuta con ottimizzazione spettrale robusta (notch multipli in banda con $p_{\text{notch}} = 15\%$), (iii) una legge ottenuta con minimizzazione del jerk con vincolo spettrale in banda ($\Delta f = 15\%$, $\text{tol}_{\text{rel}} = 20\%$, $N_{\text{band}} = 51$) e (iv) una legge ottenuta con ottimizzazione nel dominio del tempo sul modello non lineare con vincolo sulle oscillazioni residue $\eta_{\text{rest}} \leq 3$ mm. Nel seguito si riportano i profili sperimentati e la sintesi quantitativa dei risultati; come nelle sezioni precedenti, i valori percentuali sono riferiti alla baseline della stessa campagna.

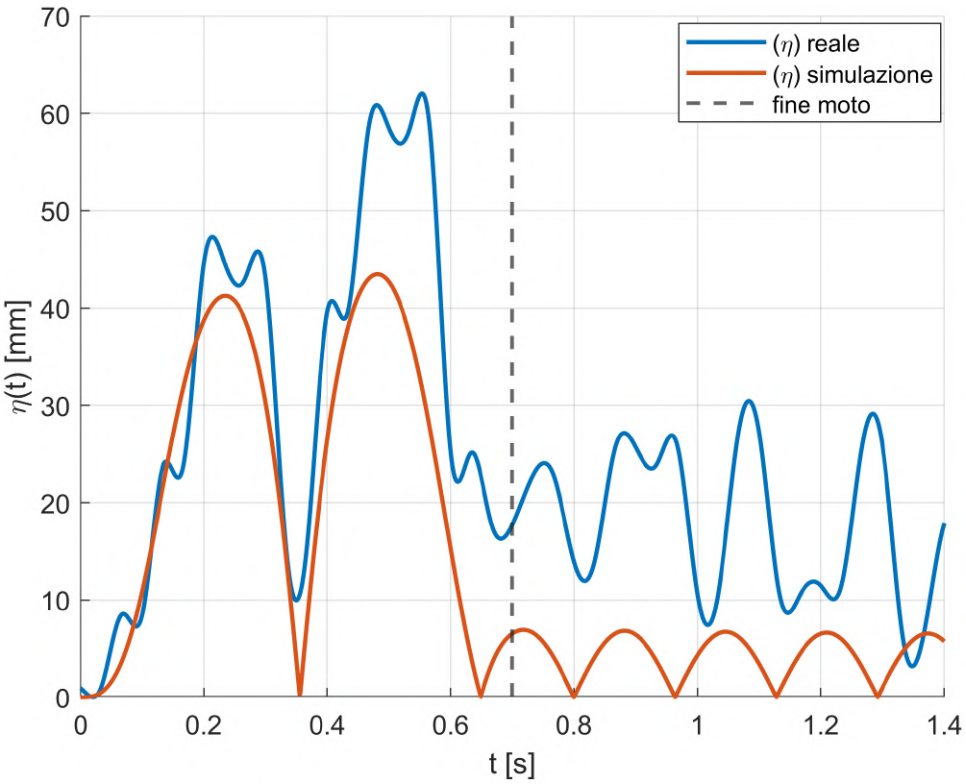


Figura 5.22: Profilo baseline non ottimizzato per corsa $s_{\text{tot}} = 500$ mm e tempo di avanzamento $t_{\text{avanz}} = 0.7$ s (ID interno: `legge_700`).

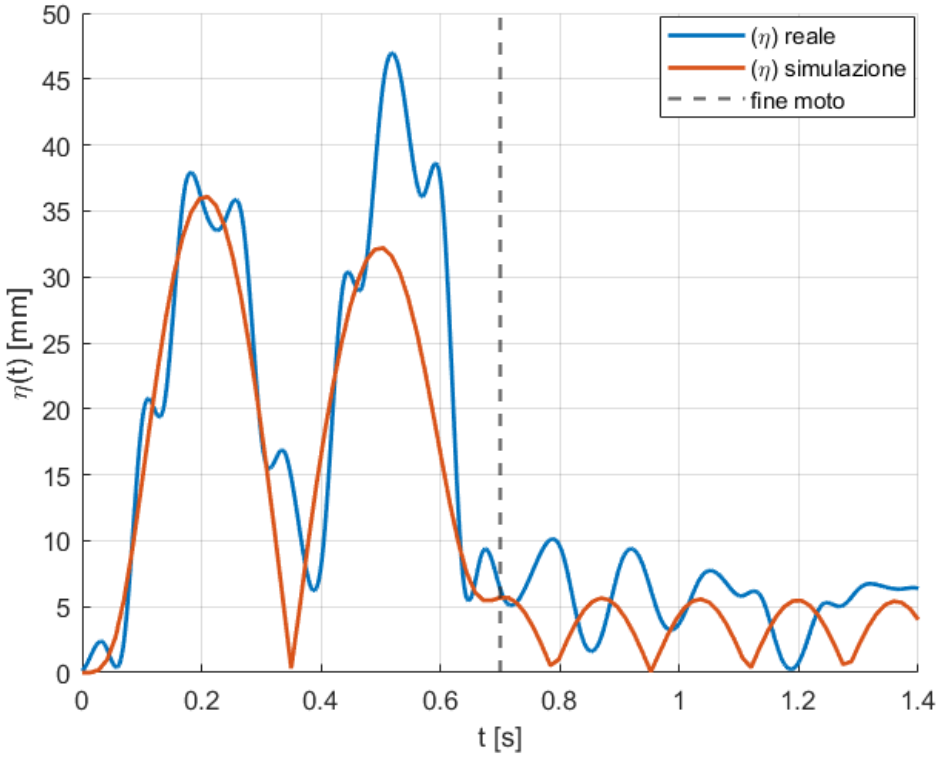


Figura 5.23: Profilo ottimizzato con notch multipli in banda ($p_{\text{notch}} = 15\%$, Appendice B) per corsa $s_{\text{tot}} = 500$ mm e $t_{\text{avanz}} = 0.7$ s (ID interno: `legge_67`).

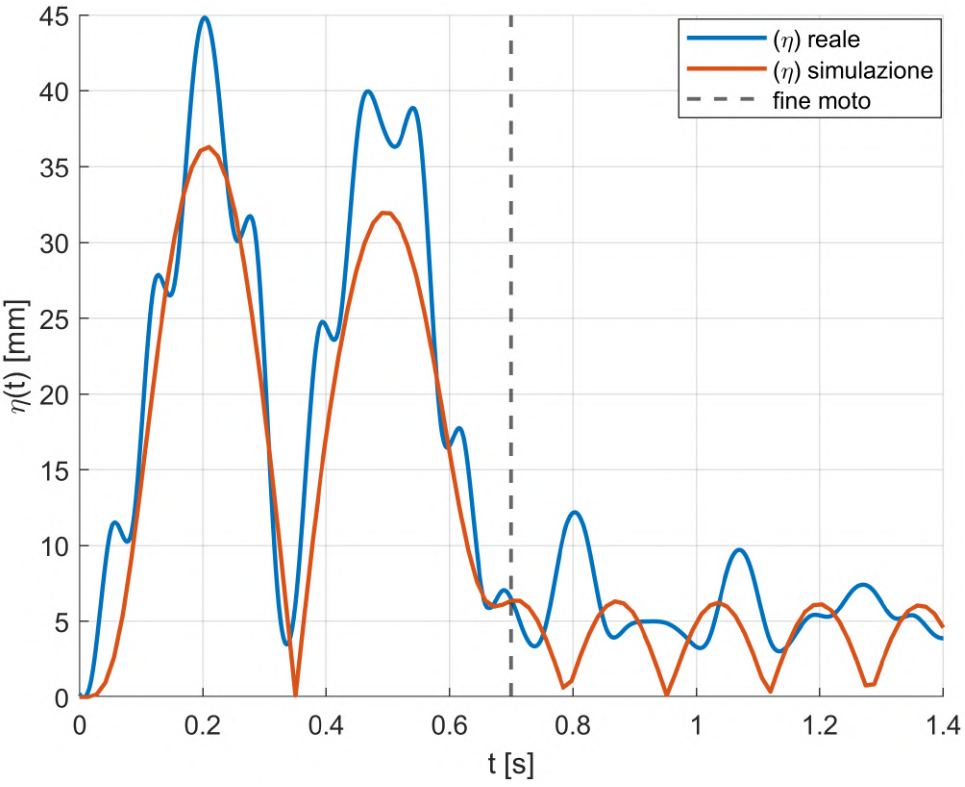


Figura 5.24: Profilo ottimizzato con min-jerk e vincolo spettrale in banda ($\Delta f = 15\%$, $\text{tol}_{\text{rel}} = 20\%$, $N_{\text{band}} = 51$, Appendice C) per corsa $s_{\text{tot}} = 500$ mm e $t_{\text{avanz}} = 0.7$ s (ID interno: `legge_66`).

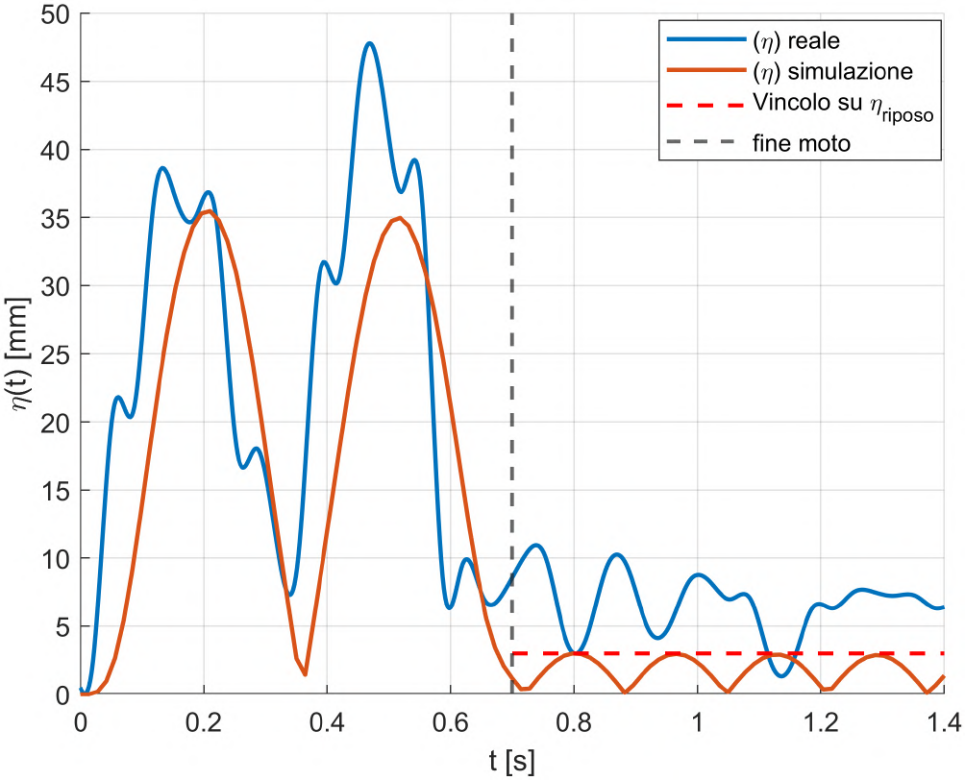


Figura 5.25: Profilo ottimizzato nel dominio del tempo sul modello non lineare con vincolo $\eta_{\text{rest}} \leq 3$ mm (Appendice A) per corsa $s_{\text{tot}} = 500$ mm e $t_{\text{avanz}} = 0.7$ s (ID interno: `legge_65`).

Tabella 5.4: Risultati sperimentali per corsa $s_{\text{tot}} = 500$ mm e $t_{\text{avanz}} = 0.7$ s. I miglioramenti percentuali sono calcolati rispetto alla traiettoria baseline (trapezia standard a 7 tratti con ripartizione uniforme; ID interno: `legge_700`) e sono rappresentati come riduzioni.

Traiettoria (caratteristiche principali)	η_{moto} [mm]	η_{riposo} [mm]	$\Delta\eta_{\text{moto}}$ [%]	$\Delta\eta_{\text{riposo}}$ [%]
Trapezia standard (baseline) ID: `legge_700`	62.0	30.4	–	–
Notch multipli in banda, $p_{\text{notch}} = 15\%$ ID: `legge_67`	46.7	9.8	−24.8	−67.7
Min-jerk in banda, $\Delta f = 15\%$, $\text{tol}_{\text{rel}} = 20\%$, $N_{\text{band}} = 51$ ID: `legge_66`	44.8	12.2	−27.7	−59.9
Ottimizzazione nel tempo sul modello non lineare, $\eta_{\text{rest}} \leq 3$ mm ID: `legge_65`	47.8	10.9	−23.0	−64.1

5.3 Discussione complessiva dei risultati sperimentali

Le tabelle di sintesi (Tabelle 5.1–5.4) mostrano in modo chiaro che tutte le strategie testate consentono, nelle condizioni considerate, di ridurre in misura significativa l'altezza di sloshing rispetto alle rispettive traiettorie baseline. In particolare, il beneficio risulta generalmente più marcato nella fase di *riposo* (oscillazione residua), mentre nella fase di *moto* i miglioramenti percentuali tendono ad essere più contenuti e, soprattutto, più sensibili alle condizioni dinamiche della prova e al contenuto armonico effettivamente introdotto dalla legge di accelerazione.

Nel caso $s_{\text{tot}} = 800$ mm e $t_{\text{avanz}} = 1.0$ s (Tabella 5.1), tutte le strategie ottengono una riduzione consistente sia in moto sia in riposo, con miglioramenti in riposo tipicamente molto elevati e miglioramenti in moto comunque importanti. In particolare, nella fase di riposo si osservano riduzioni che raggiungono circa l'85%, come per il caso della `legge_7`, mentre in fase di moto il miglioramento massimo si attesta intorno al 51% (per la `legge_22`, a conferma che la mitigazione dell'oscillazione residua è il beneficio più marcato nelle condizioni meno spinte.

Riducendo il tempo a $t_{\text{avanz}} = 0.9$ s (Tabella 5.2), il quadro resta complessivamente positivo: il notch singolo in ω_n produce un beneficio solamente in moto, mentre le strategie robuste in banda e min-jerk mostrano un'elevata efficacia sulla riduzione della vibrazione residua. La formulazione time-domain sul modello non lineare con vincolo $\eta_{\text{rest}} \leq 3$ mm si distingue per la forte riduzione in riposo anche in questa condizione. Dal confronto quantitativo, nella fase di riposo si arriva fino a circa il 91% di riduzione con la `legge_50`), mentre in fase di moto il miglioramento può ancora superare il 50% (fino a 54.3% nel caso della `legge_36`).

Quando la dinamica aumenta ulteriormente ($t_{\text{avanz}} = 0.8$ s su 800 mm, Tabella 5.3), emerge invece un aspetto importante: non tutte le strategie mantengono lo stesso comportamento. In particolare, la strategia multi-notch in banda (`legge_54`) risulta addirittura *controproducente* sulla fase di moto (aumenta l'altezza di sloshing del 36%), pur riducendo la vibrazione residua in riposo. Al contrario, sia la strategia min-jerk (`legge_51`) sia la strategia time-domain sul modello non lineare (`legge_57`) mantengono un comportamento migliorativo, con una riduzione più netta nella fase di riposo rispetto a quella di moto.

Un comportamento analogo si osserva anche per corsa ridotta $s_{\text{tot}} = 500$ mm e $t_{\text{avanz}} = 0.7$ s (Tabella 5.4): in questa configurazione, i miglioramenti in moto restano dell'ordine del 23–28%, mentre in riposo si raggiungono riduzioni nell'intervallo 60–68%, confermando che la fase di sosta beneficia maggiormente dell'ottimizzazione rispetto alla fase di moto. I tre metodi selezionati dalle prove più lente (multi-notch robusto, min-jerk in banda e time-domain sul modello non lineare) restano complessivamente efficaci, ma la ripartizione del beneficio tra moto e riposo cambia. In particolare, la strategia min-jerk mostra il miglioramento in moto più elevato tra i tre metodi, mentre le altre due strategie risultano particolarmente efficaci sul contenimento dell'oscillazione residua. Anche in questa casistica, complessivamente, seppur con valori più contenuti di miglioramento, i tre metodi isolati in quanto i più efficaci, portano a considerevoli migliorie di mitigazione e smorzamento.

Il caso della legge `legge_54` (Tabella 5.3) evidenzia che l'ottimizzazione basata sul solo contenuto spettrale può perdere efficacia quando la dinamica diventa più "aggressiva". Questo esito è coerente con la natura della traiettoria, che è ricavata considerando solo la frequenza naturale del pelo libero, parametro caratterizzante del modello

lineare, e risulta quindi più efficace quando la dinamica del sistema può essere approssimata bene in regime quasi-lineare; al crescere delle accelerazioni e delle non linearità, la riduzione spettrale "a notch" può perdere efficacia e, in casi estremi, peggiorare la risposta durante la fase di moto. In tali condizioni, imporre notch puntuali, o anche una banda relativamente stretta, può non essere sufficiente a garantire un miglioramento globale durante il moto, e in alcuni casi può produrre profili che, pur riducendo la risposta residua, risultano sfavorevoli nella fase di accelerazione/decelerazione.

Rispetto alla multi-notch, la strategia min-jerk in banda mostra una migliore "tenuta" all'aumentare delle dinamiche. Ciò è plausibile perché, oltre a imporre un'attenuazione spettrale in banda, la formulazione include un termine che penalizza il jerk, favorendo profili più regolari e riducendo la probabilità di introdurre contenuti ad alta frequenza che possono peggiorare la risposta durante il moto.

La formulazione time-domain sul modello non lineare con vincolo esplicito sulla fase di riposo $\eta_{\text{rest}} \leq 3$ mm risulta, nel complesso, la più coerente con l'obiettivo fisico di interesse (agire direttamente su $\eta(t)$) e mostra un comportamento particolarmente robusto nel mantenere bassa la vibrazione residua (fino al 90% in alcune configurazioni) anche al variare delle condizioni di prova. Il principale limite operativo osservato è legato alla fattibilità numerica: vincoli troppo stringenti sulla risposta possono ridurre lo spazio delle soluzioni ammissibili e portare più facilmente a fallimenti del solver. Per questa ragione, un valore di soglia "rilassato" ma ingegneristicamente significativo come $\eta_{\text{rest}} \leq 3$ mm, rappresenta un buon compromesso tra efficacia e robustezza numerica.

Le considerazioni precedenti aiutano anche a interpretare l'esito delle prove su $s_{\text{tot}} = 500$ mm e $t_{\text{avanz}} = 0.65$ s, dove i metodi risultano meno efficaci e meno ripetibili. Tale condizione è confrontabile con il caso a $t_{\text{avanz}} = 0.8$ s su corsa 800 mm: in entrambi i casi la manovra porta a sollecitazioni elevate e a un contenuto armonico più critico. In aggiunta, le misure a bordo macchina indicano che le accelerazioni tipiche nella campagna a 0.65 s su 500 mm raggiungono valori massimi dell'ordine di 7 m/s^2, mentre per 0.7 s su 500 mm scendono a circa 5 m/s^2: questa differenza contribuisce a spiegare perché, nella configurazione più "estrema", il margine di mitigazione si riduce e diventa più difficile ottenere benefici netti e robusti.

In sintesi, i risultati sperimentali confermano che le strategie proposte possono ridurre in modo significativo lo sloshing, ma mostrano anche che, aumentando la dinamica del moto, il problema diventa più "rigido" e la mitigazione in fase di moto tende a saturare prima rispetto alla fase di riposo. In tale regime, le strategie che integrano criteri di regolarità (min-jerk) o che lavorano direttamente sulla risposta del modello nel tempo (time-domain sul non lineare) risultano complessivamente più robuste rispetto a ottimizzazioni esclusivamente spettrali basate su notch.

Un aspetto ricorrente è la diversa "ottimizzabilità" delle due fasi: durante il moto, l'andamento di $\eta(t)$ è dominato dalla forzante imposta (accelerazione/decelerazione) e dai vincoli cinematici, quindi la riduzione del picco η_{moto} è limitata dal fatto che il profilo deve comunque completare la corsa nel tempo assegnato. Al contrario, nella fase di riposo l'obiettivo è prevalentemente ridurre l'energia residua del modo dominante; per questo motivo si osservano spesso miglioramenti percentuali molto più marcati su η_{riposo} rispetto a η_{moto}.

Capitolo 6

Conclusioni

Il presente lavoro di tesi ha affrontato il problema della mitigazione dello sloshing in movimentazioni ad alte prestazioni tipiche dell'automazione industriale, con l'obiettivo di ottenere leggi di moto compatibili con l'implementazione in macchina e, al tempo stesso, efficaci e *robuste* al variare di condizioni dinamiche. L'idea guida è stata quella di non limitarsi alla ricerca dell'"ottimo" in condizioni nominali, ma di costruire strategie che mantengano prestazioni accettabili anche in presenza di incertezze, in particolare lavorando sul contenuto in frequenza della forzante imposta al recipiente.

A partire dai modelli equivalenti meccanici lineare e non lineare dello sloshing, è stato formulato un problema di ottimizzazione su leggi di moto *a tratti* e più in particolare su trapezia standard a 7 segmenti, scegliendo come variabili decisionali le durate dei segmenti e imponendo i vincoli di fattibilità necessari per un moto *rest-to-rest*. Sul piano metodologico, il contributo centrale consiste nell'introduzione esplicita di una descrizione nel dominio della frequenza: è stato ricavato lo spettro analitico della legge trapezia e lo si è utilizzato per costruire obiettivi e vincoli spettrali direttamente in funzione dei parametri temporali della traiettoria. In parallelo, è stata mantenuta una strategia nel dominio del tempo basata sulla risposta del modello non lineare, in modo da disporre di un termine di confronto "fisico" quando la dinamica risulta più aggressiva.

Le campagne sperimentali su banco monodimensionale hanno consentito di confrontare in modo controllato traiettorie baseline e traiettorie ottimizzate, variando progressivamente la dinamica (corsa e tempo di avanzamento) e mantenendo costante la struttura della legge. Dall'analisi complessiva emerge che tutte le strategie considerate possono ridurre in modo significativo lo sloshing rispetto alle baseline nelle condizioni investigate, con un beneficio generalmente più marcato nella fase di riposo (oscillazione residua) rispetto alla fase di moto. Il miglioramento in moto risulta invece più variabile e tende a degradare al crescere dell'"aggressività" della manovra: riducendo il tempo a parità di corsa, aumentano accelerazioni e jerk caratteristici, la forzante trasferisce più energia al fluido in tempi più brevi e diventa più probabile eccitare componenti non pienamente rappresentate dal modello equivalente (modi superiori, effetti tridimensionali e non linearità più marcate). In tali condizioni, anche la corrispondenza tra risposta modellistica e misura sperimentale tende a peggiorare. I risultati ottenuti risultano inoltre coerenti con quelli riportati in [10] e in [4], in cui la riduzione dello sloshing è ottenuta mediante tecniche di input shaping applicate al recipiente di dimensioni maggiori. Questo confronto qualitativo suggerisce che l'approccio basato sull'analisi spettrale consente di ottenere prestazioni analoghe a quelle degli shaper,

pur senza modificare esplicitamente la forma della legge di moto.
All'interno di questo quadro, si distinguono tre livelli di intervento: (i) ottimizzazione spettrale "puntuale" (notch singolo in corrispondenza di ω_n); (ii) ottimizzazione spettrale robusta in banda (multi-notch attorno a ω_n, con banda definita da un parametro di ampiezza relativa fissato nella campagna sperimentale); (iii) strategie che introducono esplicitamente requisiti di regolarità/robustezza aggiuntivi, come la minimizzazione del jerk con vincolo spettrale in banda e l'ottimizzazione nel dominio del tempo sul modello non lineare con vincolo sulle oscillazioni residue.
In particolare, quando la dinamica diventa più spinta, l'ottimizzazione basata sul solo contenuto spettrale può perdere efficacia nella fase di moto e, in alcuni casi, diventare controproducente: ciò è coerente con il fatto che la logica "tipo input shaping" (attenuazione mirata del contenuto armonico) risulta più adatta a contesti dominati da pochi modi e da dinamiche prossime alla linearità. Viceversa, la strategia *min-jerk in banda* e soprattutto l'ottimizzazione time-domain sul modello non lineare hanno mostrato un comportamento più stabile al variare delle condizioni, mantenendo riduzioni significative della vibrazione residua anche per manovre più aggressive.
I principali elementi di forza del lavoro possono essere riassunti come segue:

- **Compatibilità industriale**: l'intervento avviene su un numero limitato di parametri (ΔT_i) e mantiene una struttura a tratti direttamente implementabile nei controllori di macchina.

- **Spettro analitico e ottimizzazione "intrinseca"**: la disponibilità di una forma chiusa evita dipendenze da scelte di campionamento quali: passo Δt, finestra, leakage o zero-padding e consente di costruire obiettivi/vincoli spettrali continui rispetto alle variabili decisionali.

- **Validazione sperimentale**: le prove su banco e il post-processing video hanno reso possibile un confronto quantitativo su più dinamiche e la traduzione dei benefici simulativi in riduzioni misurabili dello sloshing in condizioni operative concrete.

Dalle prove si evidenzia un limite strutturale: esiste un compromesso inevitabile tra tempo ciclo e riduzione dello sloshing. Riducendo troppo la durata del moto, lo spazio delle soluzioni ammissibili si restringe (vincoli più stringenti e profili più aggressivi) e la riduzione "raggiungibile" tende a diminuire; in queste condizioni il limite dominante può non essere l'ottimizzatore, ma la combinazione di vincoli di macchina e fisica del fenomeno. Inoltre, l'efficacia delle strategie spettrali dipende dalla validità dell'ipotesi modale sottostante (pochi modi dominanti) e dalla vicinanza a un regime "quasi lineare"; all'aumentare della non linearità effettiva, diventa più importante introdurre criteri che tengano conto direttamente della dinamica nel tempo o che rendano l'intervento spettrale meno "puntuale".
Inoltre, la campagna sperimentale ha utilizzato un unico contenitore cilindrico (D = 97 mm, h = 82 mm) con acqua addizionata di colorante, rappresentativo di una condizione specifica; in particolare, le prove sono state condotte con un solo valore del rapporto h/R, parametro centrale nel modello modale del liquido. Di conseguenza, la generalizzazione dei risultati a configurazioni caratterizzate da rapporti h/R differenti richiede ulteriori verifiche sperimentali, così come l'estensione a fluidi con proprietà reologiche differenti. Infine, la misura dell'altezza di sloshing tramite analisi video, pur robu-

sta e conveniente, introduce inevitabilmente incertezze di tipo ottico e di calibrazione che potrebbero essere ridotte con strumenti di misura dedicati.
A partire dai risultati conseguiti, diversi sviluppi futuri appaiono di interesse sia scientifico che applicativo. In primo luogo, sarebbe utile estendere la campagna sperimentale a configurazioni più variegate: da un lato, testare corse più brevi e tempi di avanzamento ulteriormente ridotti, per caratterizzare i limiti assoluti di mitigazione in regimi dinamici molto spinti; dall'altro, ripetere le stesse prove con contenitori di diametro diverso, al fine di verificare la scalabilità del metodo e la sua dipendenza dalla pulsazione naturale del sistema, che varia significativamente con la geometria del recipiente e il livello di riempimento. Altri sviluppi comprendono:

- **Estensione a profili raccordati e jerk-limited**: mantenendo l'idea di parametrizzazione a segmenti, ottimizzare anche leggi trapezoidali con raccordi ad esempio sinusoidali, per garantire continuità del jerk e ridurre ulteriormente vibrazioni strutturali e contenuti ad alta frequenza.

- **Modelli ancora più precisi**: estendere i modelli meccanici equivalenti includendo più modi di sloshing e/o effetti tridimensionali; in alternativa, adottare formulazioni non lineari più espressive quando la dinamica entra in regimi fortemente non lineari. In parallelo, sulla base dei risultati sperimentali si può prevedere di ottenere stime più accurate del rapporto di smorzamento effettivo introdotto (e della sua possibile dipendenza da riempimento, geometria e condizioni operative), migliorando l'aderenza tra simulazioni e prove.

- **Integrazione "format-driven" in macchina**: passare da soluzioni precalcolate a un approccio parametrico, in cui, dato un formato (geometria e riempimento noti e codificati a bordo linea), la macchina possa calcolare automaticamente i parametri della legge di moto tramite un metodo robusto pre-caricato, rendendo l'anti-sloshing adattivo al cambio formato.

In conclusione, il lavoro mostra che l'ottimizzazione di leggi a tratti può produrre benefici sperimentali rilevanti, soprattutto sulla vibrazione residua, e che la combinazione tra analisi spettrale e criteri di regolarità/dinamica nel tempo rappresenta una direzione promettente per ottenere prestazioni robuste in scenari industriali reali.

Appendici

Appendice A

Codice ottimizzazione nel dominio del tempo sul modello non lineare

```
import casadi as ca
import numpy as np
import math
import matplotlib.pyplot as plt

def calc_aMax_aMin_trapStd_casadi(DT, s_tot):
    DT1, DT2, DT3, DT4, DT5, DT6, DT7 = DT[0], DT[1], DT[2], DT[3], DT
    ↪ [4], DT[5], DT[6]

    c1 = (DT1*DT7**2 + 2*DT2*DT7**2 + DT3*DT7**2)/(2*DT7) + DT3*DT6/2 +
    ↪  DT2*DT6 + DT1*DT6/2 + (DT1*DT5**2 + 2*DT2*DT5**2 + DT3*DT5**2)
    ↪ /(2*DT5) + \
          (DT3/2 + DT2 + DT1/2)*DT4 + (DT1*DT3**2 + 2*DT2*DT3**2 +
    ↪ (2.0/3.0)*DT3**3)/(2*DT3) + DT1*DT2/2 + DT2**2/2 + DT1**2/6
    c2 = (-DT5*DT7**2 - 2*DT6*DT7**2 - (2.0/3.0)*DT7**3)/(2*DT7) - DT6
    ↪ **2/2 - DT5*DT6/2 - DT5**2/6
    c3 =  DT3/2 + DT2 + DT1/2
    c4 = -DT7/2 - DT6 - DT5/2

    denom = (c1*c4 - c2*c3)

    aMax =  c4 * s_tot / denom
    aMin = -c3 * s_tot / denom
    return aMax, aMin

def build_trap_coeffs_casadi(DT, s_tot_m):

    #Versione CasADi di calc_trap:
    #DT: MX(7,1) con [DT1..DT7]
    #s_tot_m: spostamento totale scalare [m]
    #Ritorna:
    #  coeff: MX(24,1) con coefficienti polinomiali e valori di jerk
    #  t_switch: MX(8,1) con [t0..t7]

```

```
30      DT1, DT2, DT3, DT4, DT5, DT6, DT7 = DT[0], DT[1], DT[2], DT[3], DT
        ↪ [4], DT[5], DT[6]
31
32      t0 = 0
33      t1 = DT1
34      t2 = DT1 + DT2
35      t3 = DT1 + DT2 + DT3
36      t4 = DT1 + DT2 + DT3 + DT4
37      t5 = DT1 + DT2 + DT3 + DT4 + DT5
38      t6 = DT1 + DT2 + DT3 + DT4 + DT5 + DT6
39      t7 = DT1 + DT2 + DT3 + DT4 + DT5 + DT6 + DT7
40
41      # condizioni al contorno
42      pos_0 = 0.0
43      vel_0 = 0.0
44      acc_0 = 0.0
45
46      pos_7 = s_tot_m
47      vel_7 = 0.0
48      acc_7 = 0.0
49
50      # questa legge ha il tratto 4 a accelerazione nulla (velocit
        ↪ costante)
51      A3 = 0.0
52
53      a0_1 = pos_0
54      a1_1 = vel_0
55      a2_1 = acc_0 / 2.0
56      a2_4 = A3 / 2.0
57      a2_5 = a2_4
58
59      A = ca.MX.zeros(24, 24)
60
61      A[0, 0]  = 6.0;         A[0, 20] = -1.0
62      A[1, 0]  = -DT1**3;     A[1, 1]  = 1.0
63      A[2, 2]  = 1.0;         A[2, 0]  = -3.0 * DT1**2
64      A[3, 3]  = 2.0;         A[3, 0]  = -6.0 * DT1
65
66      A[4, 4]  = -1.0;        A[4, 1]  = 1.0;  A[4, 2] = DT2;        A[4,
        ↪ 3] = DT2**2
67      A[5, 5]  = -1.0;        A[5, 2]  = 1.0;  A[5, 3] = 2.0 * DT2
68      A[6, 6]  = 1.0;         A[6, 3]  = -1.0
69
70      A[7, 7]  = 6.0;         A[7, 21] = -1.0
71      A[8, 8]  = -1.0;        A[8, 4]  = 1.0;  A[8, 5] = DT3;        A[8,
        ↪ 6] = DT3**2;      A[8, 7] = DT3**3
72      A[9, 9]  = -1.0;        A[9, 5]  = 1.0;  A[9, 6] = 2.0 * DT3; A[9,
        ↪ 7] = 3.0 * DT3**2
73      A[10, 6] = 1.0;         A[10, 7] = 3.0 * DT3
74
75      A[11, 10] = 1.0;        A[11, 8] = -1.0; A[11, 9] = -DT4
```

```
    A[12, 11] = 1.0;       A[12, 9] = -1.0

    A[13, 12] = 6.0;       A[13, 22] = -1.0
    A[14, 13] = 1.0;       A[14, 10] = -1.0; A[14, 11] = -DT5;       A
    ↪ [14, 12] = -DT5**3
    A[15, 14] = 1.0;       A[15, 11] = -1.0; A[15, 12] = -3.0 * DT5**2
    A[16, 15] = 1.0;       A[16, 12] = -3.0 * DT5

    A[17, 16] = -1.0;      A[17, 13] = 1.0; A[17, 14] = DT6;          A
    ↪ [17, 15] = DT6**2
    A[18, 17] = -1.0;      A[18, 14] = 1.0; A[18, 15] = 2.0 * DT6
    A[19, 18] = -1.0;      A[19, 15] = 1.0

    A[20, 19] = 6.0;       A[20, 23] = 1.0
    A[21, 16] = 1.0;       A[21, 17] = DT7;     A[21, 18] = DT7**2;
    ↪ A[21, 19] = DT7**3
    A[22, 17] = 1.0;       A[22, 18] = 2.0 * DT7; A[22, 19] = 3.0 * DT7
    ↪ **2
    A[23, 18] = 2.0;       A[23, 19] = 6.0 * DT7

    b = ca.vertcat(
        0,
        a0_1 + a1_1 * DT1 + a2_1 * DT1**2,
        a1_1 + 2.0 * a2_1 * DT1,
        2.0 * a2_1,
        0.0,
        0.0,
        0.0,
        0.0,
        0.0,
        0.0,
        a2_4,
        a2_4 * DT4**2,
        2.0 * a2_4 * DT4,
        0.0,
        a2_5 * DT5**2,
        2.0 * a2_5 * DT5,
        a2_5,
        0.0,
        0.0,
        0.0,
        0.0,
        pos_7,
        vel_7,
        acc_7
    )

    coeff = ca.solve(A, b)
    t_switch = ca.vertcat(t0, t1, t2, t3, t4, t5, t6, t7)
    return coeff, t_switch

```

```
123
124 def trapStd_kinematics_casadi(t, coeff, t_sw, s_tot_m):
125
126     #Ritorna p(t), v(t), a(t), j(t) (MX) per la legge a 7 tratti.
127
128     t0, t1, t2, t3, t4, t5, t6, t7 = [t_sw[i] for i in range(8)]
129
130     pos_0 = 0.0
131     vel_0 = 0.0
132     acc_0 = 0.0
133     A3    = 0.0
134
135     a0_1 = pos_0
136     a1_1 = vel_0
137     a2_1 = acc_0 / 2.0
138     a2_4 = A3 / 2.0
139     a2_5 = a2_4
140
141     # estrazione dei coefficienti
142     a3_1 = coeff[0]
143     a0_2, a1_2, a2_2 = coeff[1],  coeff[2],  coeff[3]
144     a0_3, a1_3, a2_3, a3_3 = coeff[4], coeff[5], coeff[6], coeff[7]
145     a0_4, a1_4 = coeff[8], coeff[9]
146     a0_5, a1_5, a3_5 = coeff[10], coeff[11], coeff[12]
147     a0_6, a1_6, a2_6 = coeff[13], coeff[14], coeff[15]
148     a0_7, a1_7, a2_7, a3_7 = coeff[16], coeff[17], coeff[18], coeff[19]
149
150     # tratto 1
151     tau1 = t - t0
152     p1 = a0_1 + a1_1*tau1 + a2_1*tau1**2 + a3_1*tau1**3
153     v1 =        a1_1      + 2*a2_1*tau1  + 3*a3_1*tau1**2
154     a1 =                    2*a2_1       + 6*a3_1*tau1
155     j1 =                                   6*a3_1
156
157     # tratto 2
158     tau2 = t - t1
159     p2 = a0_2 + a1_2*tau2 + a2_2*tau2**2
160     v2 =        a1_2      + 2*a2_2*tau2
161     a2 =                    2*a2_2
162     j2 = 0.0
163
164     # tratto 3
165     tau3 = t - t2
166     p3 = a0_3 + a1_3*tau3 + a2_3*tau3**2 + a3_3*tau3**3
167     v3 =        a1_3      + 2*a2_3*tau3  + 3*a3_3*tau3**2
168     a3 =                    2*a2_3       + 6*a3_3*tau3
169     j3 =                                   6*a3_3
170
171     # tratto 4 (accelerazione nulla se A3=0)
172     tau4 = t - t3
173     p4 = a0_4 + a1_4*tau4 + a2_4*tau4**2
```

```
    v4 =        a1_4      + 2*a2_4*tau4
    a4 =                    2*a2_4
    j4 = 0.0

    # tratto 5
    tau5 = t - t4
    p5 = a0_5 + a1_5*tau5 + a2_5*tau5**2 + a3_5*tau5**3
    v5 =        a1_5      + 2*a2_5*tau5  + 3*a3_5*tau5**2
    a5 =                    2*a2_5       + 6*a3_5*tau5
    j5 =                                   6*a3_5

    # tratto 6
    tau6 = t - t5
    p6 = a0_6 + a1_6*tau6 + a2_6*tau6**2
    v6 =        a1_6      + 2*a2_6*tau6
    a6 =                    2*a2_6
    j6 = 0.0

    # tratto 7
    tau7 = t - t6
    p7 = a0_7 + a1_7*tau7 + a2_7*tau7**2 + a3_7*tau7**3
    v7 =        a1_7      + 2*a2_7*tau7  + 3*a3_7*tau7**2
    a7 =                    2*a2_7       + 6*a3_7*tau7
    j7 =                                   6*a3_7

    p = ca.if_else(t <= t0, 0.0,
        ca.if_else(t <= t1, p1,
        ca.if_else(t <= t2, p2,
        ca.if_else(t <= t3, p3,
        ca.if_else(t <= t4, p4,
        ca.if_else(t <= t5, p5,
        ca.if_else(t <= t6, p6,
        ca.if_else(t <= t7, p7, s_tot_m)))))))

    v = ca.if_else(t <= t0, 0.0,
        ca.if_else(t <= t1, v1,
        ca.if_else(t <= t2, v2,
        ca.if_else(t <= t3, v3,
        ca.if_else(t <= t4, v4,
        ca.if_else(t <= t5, v5,
        ca.if_else(t <= t6, v6,
        ca.if_else(t <= t7, v7, 0.0)))))))

    a = ca.if_else(t <= t0, 0.0,
        ca.if_else(t <= t1, a1,
        ca.if_else(t <= t2, a2,
        ca.if_else(t <= t3, a3,
        ca.if_else(t <= t4, a4,
        ca.if_else(t <= t5, a5,
        ca.if_else(t <= t6, a6,
        ca.if_else(t <= t7, a7, 0.0)))))))
```

```
225
226     j = ca.if_else(t <= t0, 0.0,
227         ca.if_else(t <= t1, j1,
228         ca.if_else(t <= t2, j2,
229         ca.if_else(t <= t3, j3,
230         ca.if_else(t <= t4, j4,
231         ca.if_else(t <= t5, j5,
232         ca.if_else(t <= t6, j6,
233         ca.if_else(t <= t7, j7, 0.0)))))))
234
235     return p, v, a, j
236
237 # ================================================================================
    ↪ =================
238 # FFT (per confronto)
239 # ================================================================================
    ↪ =================
240
241 def calc_trajectory_fft_np(t, acc):
242     T = t[1] - t[0]
243     Fs = 1.0 / T
244     L = len(acc)
245     n = int(2**np.ceil(np.log2(10 * L)))
246
247     Y = np.fft.fft(acc, n)
248     trasf_calc = np.abs(Y / L)
249     trasf = trasf_calc[:n//2 + 1]
250     if trasf.size > 2:
251         trasf[1:-1] *= 2.0
252
253     f = Fs * np.arange(0, n//2 + 1) / n
254     w = f * 2*np.pi
255     return w, trasf
256
257
258 def trapStd_amplitude_spectrum_np(w, DT_vec, s_tot, t0=0.0):
259
260     #Versione NumPy (vettoriale) per plottare A() della trapezia
    ↪ standard
261
262     DT1, DT2, DT3, DT4, DT5, DT6, DT7 = DT_vec
263
264     t1 = t0 + DT1
265     t2 = t1 + DT2
266     t3 = t2 + DT3
267     t4 = t3 + DT4
268     t5 = t4 + DT5
269     t6 = t5 + DT6
270     t7 = t6 + DT7
271
272     # aMax/aMin
```

```
    c1 = (DT1*DT7**2 + 2*DT2*DT7**2 + DT3*DT7**2)/(2*DT7) + DT3*DT6/2 +
    ↪  DT2*DT6 + DT1*DT6/2 + (DT1*DT5**2 + 2*DT2*DT5**2 + DT3*DT5**2)
    ↪ /(2*DT5) + \
          (DT3/2 + DT2 + DT1/2)*DT4 + (DT1*DT3**2 + 2*DT2*DT3**2 +
    ↪ (2.0/3.0)*DT3**3)/(2*DT3) + DT1*DT2/2 + DT2**2/2 + DT1**2/6
    c2 = (-DT5*DT7**2 - 2*DT6*DT7**2 - (2.0/3.0)*DT7**3)/(2*DT7) - DT6
    ↪ **2/2 - DT5*DT6/2 - DT5**2/6
    c3 =  DT3/2 + DT2 + DT1/2
    c4 = -DT7/2 - DT6 - DT5/2
    denom = (c1*c4 - c2*c3)
    aMax =  c4 * s_tot / denom
    aMin = -c3 * s_tot / denom

    w = np.asarray(w, dtype=float)
    sin_wt0 = np.sin(w*t0);   cos_wt0 = np.cos(w*t0)
    sin_wt1 = np.sin(w*t1);   cos_wt1 = np.cos(w*t1)
    sin_wt2 = np.sin(w*t2);   cos_wt2 = np.cos(w*t2)
    sin_wt3 = np.sin(w*t3);   cos_wt3 = np.cos(w*t3)
    sin_wt4 = np.sin(w*t4);   cos_wt4 = np.cos(w*t4)
    sin_wt5 = np.sin(w*t5);   cos_wt5 = np.cos(w*t5)
    sin_wt6 = np.sin(w*t6);   cos_wt6 = np.cos(w*t6)
    sin_wt7 = np.sin(w*t7);   cos_wt7 = np.cos(w*t7)

    with np.errstate(divide='ignore', invalid='ignore'):
        RePart = -aMin*t6*sin_wt7/(DT7*w) + aMin*t7*sin_wt7/(DT7*w) -
    ↪ aMin*t5*sin_wt7/(DT5*w) - aMax*t3*sin_wt3/(DT3*w) + \
                aMax*t2*sin_wt3/(DT3*w) + aMin*t4*sin_wt7/(DT5*w) -
    ↪ aMax*t0*sin_wt3/(DT1*w) + aMax*t1*sin_wt3/(DT1*w) + \
                aMax*cos_wt1/(DT1*w**2) - aMax*cos_wt3/(DT3*w**2) +
    ↪ aMax*cos_wt2/(DT3*w**2) + aMin*cos_wt4/(DT5*w**2) - \
                aMin*cos_wt5/(DT5*w**2) + aMin*cos_wt7/(DT7*w**2) -
    ↪ aMin*cos_wt6/(DT7*w**2) - aMax*cos_wt0/(DT1*w**2)

        ImPart = -aMax*t0*cos_wt3/(DT1*w) - aMin*t5*cos_wt7/(DT5*w) +
    ↪ aMin*t7*cos_wt7/(DT7*w) - aMin*t6*cos_wt7/(DT7*w) + \
                aMax*t1*cos_wt3/(DT1*w) - aMax*t3*cos_wt3/(DT3*w) +
    ↪ aMax*t2*cos_wt3/(DT3*w) + aMin*t4*cos_wt7/(DT5*w) - \
                aMax*sin_wt1/(DT1*w**2) + aMax*sin_wt3/(DT3*w**2) -
    ↪ aMax*sin_wt2/(DT3*w**2) - aMin*sin_wt4/(DT5*w**2) + \
                aMin*sin_wt6/(DT7*w**2) - aMin*sin_wt7/(DT7*w**2) +
    ↪ aMin*sin_wt5/(DT5*w**2) + aMax*sin_wt0/(DT1*w**2)

        A = np.sqrt(RePart**2 + ImPart**2)

    # gestione ~0
    eps = 1e-6
    mask_small = np.abs(w) < eps
    if np.any(mask_small):
        A = np.asarray(A, dtype=float)
        A[mask_small] = 0.0
    return A
```

```
312
313
314 # ============================================================
315 # OPTI (problema di ottimizzazione)
316 # ============================================================
317
318 opti = ca.Opti()
319
320 # ============================================================
321 # Parametri sloshing
322 # ============================================================
323
324 R_mm = 48.5
325 h_mm = 82.0
326
327 R = R_mm / 1000.0
328 h = h_mm / 1000.0
329 rho = 1000.0
330 st = 0.073
331 mu = 1e-3
332 nu = mu / rho
333 g  = 9.81
334
335 xi = 1.84118
336 alpha_n = 0.58
337
338
339 # ============================================================
340 # Impostazioni moto
341 # ============================================================
342
343 s_tot_mm = 800.0
344 s_tot_m  = s_tot_mm / 1000.0
345
346 t_avanz = 1.0
347 t_rest  = 1.0
348 t_tot   = t_avanz + t_rest
349
350 eta_rest_max_mm = 3                              #mm
351 eta_rest_max_m  = eta_rest_max_mm / 1000.0
352
353 #eta_moto_max_mm = 40                               #mm
354 #eta_moto_max_m = eta_moto_max_mm / 1000.0
355
356
357 # griglia temporale
358 N  = 101
359 dt = t_tot / (N - 1)
360 t_grid = np.linspace(0.0, t_tot, N)
361
362 # integro su N-1 step, usando a(t_k) al nodo sinistro
```

```
t_matrix = ca.DM(t_grid[:-1]).reshape((1, N - 1))

# indice di separazione tra fase di moto e fase di sosta
idx_moto_end = int(np.searchsorted(t_grid, t_avanz, side="right"))
    ↪       # punti t <= t_avanz


# ============================================================
# Variabili decisionali:
# ============================================================

# durate grezze non negative
DT = opti.variable(7, 1)

opti.subject_to(ca.sum(DT) == t_avanz)

# durate minime (imposte sui DT effettivi)
opti.subject_to(DT[0] >= 0.05)
opti.subject_to(DT[1] >= 0.0)
opti.subject_to(DT[2] >= 0.05)
opti.subject_to(DT[3] >= 0.0)
opti.subject_to(DT[4] >= 0.05)
opti.subject_to(DT[5] >= 0.0)
opti.subject_to(DT[6] >= 0.05)

# Guess iniziale
w7 = np.array([0.14286, 0.14286, 0.14286, 0.14284, 0.14286, 0.14286,
    ↪ 0.14286])      #somma = 1

#configurazione del risolutore numerico
opti.solver('ipopt', {
    "ipopt.print_level": 3,          #livello di stampa
    "ipopt.max_iter": 2000,          #numero massimo di iterazioni
})

# ============================================================================
    ↪ ===
# Parametri fisici calcolati per caratterizzare la dinamica dello
    ↪ sloshing
# ============================================================================
    ↪ ===

m_f = math.pi * R**2 * h * rho
ms  = (m_f * 2.0 * R * math.tanh(xi * h / R)) / (xi * h * (xi**2 - 1.0)
    ↪ )
wn  = math.sqrt((g * xi / R) * math.tanh(xi * h / R))

Re1 = nu / math.sqrt(g * R**3)
zeta_1 = 0.79 * math.sqrt(Re1) * (1.0 + (0.318 / math.sinh(1.84 * h / R
    ↪ )) * (1.0 + (1.0 - h / R) / math.cosh(1.84 * h / R)))

```

```
407 def C_n_val(xi_val, h_val, R_val):
408     return xi_val * math.tanh(xi_val * (h_val / R_val))
409 
410 Cn_const = C_n_val(xi, h, R)
411 
412 # conversione da xn ad altezza di sloshing eta in (metri)
413 K_eta = (h * (xi**2) * ms) / m_f
414 
415 
416 # ============================================================
417 # Accelerazione trapezoidale a(t) come Function + map
418 # ============================================================
419 
420 t_sym  = ca.MX.sym('t')
421 DT_sym = ca.MX.sym('DT', 7)
422 
423 coeff_sym, t_sw_sym = build_trap_coeffs_casadi(DT_sym, s_tot_m)
424 _, _, a_trap_sym, _ = trapStd_kinematics_casadi(t_sym, coeff_sym,
    ↪ t_sw_sym, s_tot_m)
425 
426 a_fun = ca.Function('a_fun', [t_sym, DT_sym], [a_trap_sym])
427 
428 # Valuta a(t_k) su tutta la griglia in un colpo (map)
429 DT_rep = ca.repmat(DT, 1, N - 1)           # 7 x (N-1)
430 a_trap_grid = a_fun.map(N - 1)(t_matrix, DT_rep)  # 1 x (N-1)
431 
432 # prodotto elemento-per-elemento
433 a_grid = a_trap_grid  #* mask  # prodotto elemento-per-elemento (in
    ↪ CasADi Python *  elemento-per-elemento)
434 
435 # vettore dt (costante)
436 dt_grid = ca.DM.ones(1, N - 1) * dt
437 
438 
439 # ============================================================
440 # Dinamica + Function di step RK4
441 # ============================================================
442 
443 x_sym  = ca.MX.sym('x', 2)     # [xn, xnd]
444 a_in   = ca.MX.sym('a')        # ingresso: accelerazione a(t)
445 dt_sym = ca.MX.sym('dt')
446 
447 xn_sym  = x_sym[0]
448 xnd_sym = x_sym[1]
449 
450 Den = 1.0 + (Cn_const**2) * (xn_sym**2)
451 term1 = 2.0 * wn * zeta_1 * (xnd_sym + (Cn_const**2) * xnd_sym * (
    ↪ xn_sym**2))
452 term2 = (Cn_const**2) * xn_sym * (xnd_sym**2)
453 term3 = (wn**2) * xn_sym * (1.0 + alpha_n * (xn_sym**2))
454 term4 = a_in / R
```

```
455
456 Num  = -(term1 + term2 + term3 + term4)
457 xndd = Num / Den
458
459 xdot = ca.vertcat(xnd_sym, xndd)
460
461 f = ca.Function('f', [x_sym, a_in], [xdot])
462
463 k1 = f(x_sym, a_in)
464 k2 = f(x_sym + 0.5 * dt_sym * k1, a_in)
465 k3 = f(x_sym + 0.5 * dt_sym * k2, a_in)
466 k4 = f(x_sym + 1.0 * dt_sym * k3, a_in)
467
468 x_next = x_sym + (dt_sym / 6.0) * (k1 + 2.0 * k2 + 2.0 * k3 + k4)
469
470 RK4_step = ca.Function('RK4_step', [x_sym, a_in, dt_sym], [x_next])
471
472 # Integro l'intera traiettoria con mapaccum (sostituisce il ciclo for)
473 RK4_accum = RK4_step.mapaccum(N - 1)
474
475 x0 = ca.DM([0.0, 0.0])
476 x_acc = RK4_accum(x0, a_grid, dt_grid)    # atteso: 2 x (N-1)
477 X = ca.horzcat(x0, x_acc)                 # 2 x N
478
479 xn_grid  = X[0, :]                        # 1 x N
480 eta_grid = K_eta * xn_grid                # 1 x N
481
482
483 # ========================================================================
    ↪ ========================================================================
    ↪ =====
484 # Vincoli costruiti sull'intero vettore eta(t)----> Si pu scegliere di
    ↪ minimizzare l'altezza in moto vincolando quella a riposo e
    ↪ viceversa
485 # ========================================================================
    ↪ ========================================================================
    ↪ =====
486
487 eta_moto = eta_grid[0, :idx_moto_end]          # t in [0, t_avanz]
488 eta_rest = eta_grid[0, idx_moto_end:]          # t in (t_avanz, t_tot]
489
490 # Vincolo in sosta: -eta_rest_max <= eta_rest <= eta_rest_max
491 opti.subject_to( eta_rest <=  eta_rest_max_m )
492 opti.subject_to( eta_rest >=  -eta_rest_max_m )
493
494 #opti.subject_to( eta_moto <=  eta_moto_max_m )
495 #opti.subject_to( eta_moto >=  -eta_moto_max_m )
496
497
498 # Obiettivo: minimizzare il picco durante il moto tramite una variabile
    ↪  ausiliaria
```

```
499 eta_peak = opti.variable()
500 opti.subject_to(eta_peak >= 0.0)
501 opti.subject_to( eta_moto <=  eta_peak )
502 opti.subject_to( eta_moto >=  -eta_peak )
503
504 opti.minimize(eta_peak)
505
506 #eta_peak_rest = opti.variable()
507 #opti.subject_to(eta_peak_rest >= 0.0)
508 #opti.subject_to( eta_rest <=  eta_peak_rest )
509 #opti.subject_to( eta_rest >=  -eta_peak_rest )
510
511 #opti.minimize(eta_peak_rest)
512
513
514 # ============================================================
515 # Vincoli aggiuntivi (coerenza profilo, jerk)
516 # ============================================================
517
518 # Per sicurezza: verifico posizione finale a t_avanz
519 coeff, t_switch = build_trap_coeffs_casadi(DT, s_tot_m)
520 p_fin, _, _, _ = trapStd_kinematics_casadi(t_avanz, coeff, t_switch,
      ↪ s_tot_m)
521 opti.subject_to(p_fin == s_tot_m)
522
523
524 #Se si volessero considerare anche dei limiti su accelerazione massima
      ↪ e jerk massimo, uncommentare questa parte
525 #Consultare la Wiki relativa a questo script su GitLab nel caso di
      ↪ dubbi!
526 '''
527 a_lim = 10
528 K_jerk = 25.0
529
530 #aMax, aMin = calc_aMax_aMin_trapStd_casadi(DT, s_tot_m)
531 #opti.subject_to( aMax <= a_lim )
532 #opti.subject_to( aMin >= -a_lim )
533
534
535 # limiti sul jerk J1..J4 (salvati in coeff[20..23])
536 J1 = coeff[20]; J2 = coeff[21]; J3 = coeff[22]; J4 = coeff[23]
537 J_lim = K_jerk * a_lim
538
539 J_vec = ca.vertcat(J1, J2, J3, J4)
540 opti.subject_to( J_vec <=  J_lim )
541 opti.subject_to( J_vec >=  -J_lim )
542
543 '''
544
545
546 # ============================================================
```

```
# Solver basato sul guess iniziale
# ============================================================

w7 = w7 / w7.sum()
DT_init = t_avanz * w7


# uncommentare .set_initial di eta_rest o eta_peak in base a se si sta minimizzando l'altezza di riposo o quella in moto
opti.set_initial(DT, DT_init.reshape((7, 1)))
opti.set_initial(eta_peak, 1e-4)
#opti.set_initial(eta_peak_rest, 1e-4)

sol = opti.solve()

DT_opt = np.array(sol.value(DT)).flatten()
DT_opt_clean = np.where(np.abs(DT_opt) < 1e-8, 0.0, DT_opt)
# DT in percentuale (somma = 100) con 3 cifre significative
DT_pct = 100.0 * DT_opt_clean / DT_opt_clean.sum()

# arrotondo "a display" a 3 cifre significative e forzo la somma esatta a 100
DT_pct_disp = np.array([float(f"{p:.3g}") for p in DT_pct])
DT_pct_disp[-1] = 100.0 - DT_pct_disp[:-1].sum()

np.save("DT_opt.npy", DT_opt_clean)

eta_grid_num = np.array(sol.value(eta_grid)).ravel()
eta_rest_num = eta_grid_num[idx_moto_end:]

# ============================================================
# Stampe
# ============================================================

print("DT_opt = [" + " , ".join(f"{x:.6f}" for x in DT_opt_clean) + "]")
print("DT_opt_tot = ", DT_opt_clean.sum())
print("Space_tot = ", float(sol.value(p_fin)))
print("DT_opt [%] = [" + ", ".join(f"{p:.3g}" for p in DT_pct_disp) + "]")
print("DT_opt_tot = ", DT_pct_disp.sum())
print("eta_peak_opt [mm] = ", 1000.0 * float(sol.value(eta_peak)))
print("eta_rest_max [mm] = ", 1000.0 * np.max(np.abs(eta_rest_num)))



# ============================================================
# Plot
# ============================================================

# --- Ricostruisci a(t) su tutta la griglia (N punti) ---
```

```
t_eval = ca.DM(t_grid).reshape((1, len(t_grid)))          # 1 x N
DT_rep_full = ca.repmat(ca.DM(DT_opt), 1, len(t_grid))    # 7 x N

a_sol = np.array(a_fun.map(len(t_grid))(t_eval, DT_rep_full)).flatten()
    ↪   # (N,)

# --- Converto eta in mm per plot pi leggibile ---
eta_sol_mm = 1000.0 * eta_grid_num

# --- Figura unica con due subplot (una sola immagine) ---
fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True, figsize=(11, 7))

# 1) Accelerazione
ax1.plot(t_grid, a_sol, linewidth=1.5)
ax1.grid(True, which='both', linestyle="--", linewidth=0.6, alpha=0.5)
ax1.axvline(t_avanz, linestyle="--", linewidth=1.2)
ax1.set_ylabel("a(t) [m/s]")
ax1.set_title("Legge di accelerazione e altezza di sloshing")

# 2) Sloshing eta
ax2.plot(t_grid, eta_sol_mm, linewidth=1.2)
ax2.grid(True, which='both', linestyle="--", linewidth=0.6, alpha=0.5)
ax2.axvline(t_avanz, linestyle="--", linewidth=1.2, label="t_avanz")

# Vincoli SOLO dopo t_avanz (fase di riposo): due linee orizzontali
    ↪ rosse tratteggiate
ax2.hlines(+eta_rest_max_mm, xmin=t_avanz, xmax=t_tot, colors="red",
    ↪ linestyles="--", linewidth=1.6)
ax2.hlines(-eta_rest_max_mm, xmin=t_avanz, xmax=t_tot, colors="red",
    ↪ linestyles="--", linewidth=1.6)

ax2.set_xlabel("t [s]")
ax2.set_ylabel("eta(t) [mm]")

# Accelerazione a(t) come Function + map (riuso della stessa legge
    ↪ trapezia)
t_sym_sim  = ca.MX.sym('t')
DT_sym_sim = ca.MX.sym('DT', 7)
coeff_sym_sim, t_sw_sym_sim = build_trap_coeffs_casadi(DT_sym_sim,
    ↪ s_tot_m)
_, _, a_trap_sym_sim, _ = trapStd_kinematics_casadi(t_sym_sim,
    ↪ coeff_sym_sim, t_sw_sym_sim, s_tot_m)
a_fun_sim = ca.Function('a_fun_sim', [t_sym_sim, DT_sym_sim], [
    ↪ a_trap_sym_sim])

# ----------------------------
# FIGURA 2: spettro (analitico) + FFT per confronto
# ----------------------------
# Ricostruisco a(t) su una griglia fitta per fare una FFT pi pulita
N_plot = 2001
t_grid_plot = np.linspace(0.0, t_tot, N_plot)
```

```
637
638 t_eval = ca.DM(t_grid_plot).reshape((1, N_plot))
639 DT_rep = ca.repmat(ca.DM(DT_opt_clean).reshape((7, 1)), 1, N_plot)
640 a_sol_plot = np.array(a_fun_sim.map(N_plot)(t_eval, DT_rep)).ravel()
641
642 # Spettro analitico su un asse di frequenze
643 w_plot = np.linspace(0.0, 3.0 * wn, 2500)
644 A_plot = trapStd_amplitude_spectrum_np(w_plot, DT_opt_clean, s_tot_m,
    ↪ t0=0.0)
645
646 # per confrontarlo con la FFT (che  single-sided) uso lo stesso scaling
    ↪  gi adottato
647 A_plot_fft = (2.0 / t_avanz) * A_plot
648
649 # FFT numerica dell'accelerazione (solo tratto di moto)
650 mask_moto = t_grid_plot <= t_avanz + 1e-12
651 w_fft, A_fft = calc_trajectory_fft_np(t_grid_plot[mask_moto],
    ↪ a_sol_plot[mask_moto])
652
653 fig2, ax = plt.subplots(1, 1, figsize=(11, 4.5))
654 ax.plot(w_plot, A_plot_fft, linewidth=1.8, label="Analitico |A(j)|")
655 ax.plot(w_fft, A_fft, linestyle="--", linewidth=1.2, label="FFT (check
    ↪ numerico)")
656 ax.axvline(wn, linestyle="--", linewidth=1.5, label=r"$\omega_n$")
657 ax.set_xlim(0.0, 3.0 * wn)
658 ax.grid(True, which='both', linestyle="--", linewidth=0.6, alpha=0.5)
659 ax.set_xlabel(r"$\omega$ [rad/s]")
660 ax.set_ylabel(r"$|A(j\omega)|$")
661 ax.set_title("Spettro di ampiezza dell'accelerazione (focus su n)")
662 ax.legend()
663
664 plt.tight_layout()
665 plt.show()
```

Listing A.1: Script di ottimizzazione: Opt_trapezia

Appendice B

Codice ottimizzazione spettrale multi-notch in banda attorno a ω_n

```

import casadi as ca
import numpy as np
import math
import matplotlib.pyplot as plt


# ======================================================================
# Trapezia standard (7 segmenti): coefficienti polinomiali + cinematica
# ======================================================================

def build_trap_coeffs_casadi(DT, s_tot_m):

    #Versione CasADi di calc_trap:
    #DT: MX(7,1) con [DT1..DT7]
    #s_tot_m: spostamento totale scalare [m]
    #Ritorna:
    #  coeff: MX(24,1) con coefficienti polinomiali e valori di jerk
    #  t_switch: MX(8,1) con [t0..t7]

    DT1, DT2, DT3, DT4, DT5, DT6, DT7 = DT[0], DT[1], DT[2], DT[3], DT
    ↪ [4], DT[5], DT[6]

    t0 = 0
    t1 = DT1
    t2 = DT1 + DT2
    t3 = DT1 + DT2 + DT3
    t4 = DT1 + DT2 + DT3 + DT4
    t5 = DT1 + DT2 + DT3 + DT4 + DT5
    t6 = DT1 + DT2 + DT3 + DT4 + DT5 + DT6
    t7 = DT1 + DT2 + DT3 + DT4 + DT5 + DT6 + DT7

    # condizioni al contorno (rest-to-rest)
```

```
    pos_0 = 0.0
    vel_0 = 0.0
    acc_0 = 0.0

    pos_7 = s_tot_m
    vel_7 = 0.0
    acc_7 = 0.0

    # questa legge ha il tratto 4 a accelerazione nulla (velocit
    ↪ costante)
    A3 = 0.0

    a0_1 = pos_0
    a1_1 = vel_0
    a2_1 = acc_0 / 2.0
    a2_4 = A3 / 2.0
    a2_5 = a2_4

    A = ca.MX.zeros(24, 24)

    A[0, 0]  = 6.0;        A[0, 20] = -1.0
    A[1, 0]  = -DT1**3;    A[1, 1]  = 1.0
    A[2, 2]  = 1.0;        A[2, 0]  = -3.0 * DT1**2
    A[3, 3]  = 2.0;        A[3, 0]  = -6.0 * DT1

    A[4, 4]  = -1.0;       A[4, 1]  = 1.0;  A[4, 2] = DT2;       A[4,
    ↪ 3] = DT2**2
    A[5, 5]  = -1.0;       A[5, 2]  = 1.0;  A[5, 3] = 2.0 * DT2
    A[6, 6]  = 1.0;        A[6, 3]  = -1.0

    A[7, 7]  = 6.0;        A[7, 21] = -1.0
    A[8, 8]  = -1.0;       A[8, 4]  = 1.0;  A[8, 5] = DT3;       A[8,
    ↪ 6] = DT3**2;      A[8, 7] = DT3**3
    A[9, 9]  = -1.0;       A[9, 5]  = 1.0;  A[9, 6] = 2.0 * DT3; A[9,
    ↪ 7] = 3.0 * DT3**2
    A[10, 6] = 1.0;        A[10, 7] = 3.0 * DT3

    A[11, 10] = 1.0;       A[11, 8] = -1.0; A[11, 9] = -DT4
    A[12, 11] = 1.0;       A[12, 9] = -1.0

    A[13, 12] = 6.0;       A[13, 22] = -1.0
    A[14, 13] = 1.0;       A[14, 10] = -1.0; A[14, 11] = -DT5;       A
    ↪ [14, 12] = -DT5**3
    A[15, 14] = 1.0;       A[15, 11] = -1.0; A[15, 12] = -3.0 * DT5**2
    A[16, 15] = 1.0;       A[16, 12] = -3.0 * DT5

    A[17, 16] = -1.0;      A[17, 13] = 1.0; A[17, 14] = DT6;          A
    ↪ [17, 15] = DT6**2
    A[18, 17] = -1.0;      A[18, 14] = 1.0; A[18, 15] = 2.0 * DT6
    A[19, 18] = -1.0;      A[19, 15] = 1.0

```

```
78      A[20, 19] = 6.0;        A[20, 23] = 1.0
79      A[21, 16] = 1.0;        A[21, 17] = DT7;      A[21, 18] = DT7**2;
        ↪ A[21, 19] = DT7**3
80      A[22, 17] = 1.0;        A[22, 18] = 2.0 * DT7; A[22, 19] = 3.0 * DT7
        ↪ **2
81      A[23, 18] = 2.0;        A[23, 19] = 6.0 * DT7
82
83      b = ca.vertcat(
84          0,
85          a0_1 + a1_1 * DT1 + a2_1 * DT1**2,
86          a1_1 + 2.0 * a2_1 * DT1,
87          2.0 * a2_1,
88          0.0,
89          0.0,
90          0.0,
91          0.0,
92          0.0,
93          0.0,
94          a2_4,
95          a2_4 * DT4**2,
96          2.0 * a2_4 * DT4,
97          0.0,
98          a2_5 * DT5**2,
99          2.0 * a2_5 * DT5,
100         a2_5,
101         0.0,
102         0.0,
103         0.0,
104         0.0,
105         pos_7,
106         vel_7,
107         acc_7
108     )
109
110     coeff = ca.solve(A, b)
111     t_switch = ca.vertcat(t0, t1, t2, t3, t4, t5, t6, t7)
112     return coeff, t_switch
113
114
115 def trapStd_kinematics_casadi(t, coeff, t_sw, s_tot_m):
116
117     #Ritorna p(t), v(t), a(t), j(t) (MX) per la legge a 7 tratti.
118
119     t0, t1, t2, t3, t4, t5, t6, t7 = [t_sw[i] for i in range(8)]
120
121     pos_0 = 0.0
122     vel_0 = 0.0
123     acc_0 = 0.0
124     A3    = 0.0
125
126     a0_1 = pos_0
```

```
    a1_1 = vel_0
    a2_1 = acc_0 / 2.0
    a2_4 = A3 / 2.0
    a2_5 = a2_4

    # estrazione dei coefficienti
    a3_1 = coeff[0]
    a0_2, a1_2, a2_2 = coeff[1],  coeff[2],  coeff[3]
    a0_3, a1_3, a2_3, a3_3 = coeff[4], coeff[5], coeff[6], coeff[7]
    a0_4, a1_4 = coeff[8], coeff[9]
    a0_5, a1_5, a3_5 = coeff[10], coeff[11], coeff[12]
    a0_6, a1_6, a2_6 = coeff[13], coeff[14], coeff[15]
    a0_7, a1_7, a2_7, a3_7 = coeff[16], coeff[17], coeff[18], coeff[19]

    # tratto 1
    tau1 = t - t0
    p1 = a0_1 + a1_1*tau1 + a2_1*tau1**2 + a3_1*tau1**3
    v1 =        a1_1      + 2*a2_1*tau1  + 3*a3_1*tau1**2
    a1 =                    2*a2_1       + 6*a3_1*tau1
    j1 =                                   6*a3_1

    # tratto 2
    tau2 = t - t1
    p2 = a0_2 + a1_2*tau2 + a2_2*tau2**2
    v2 =        a1_2      + 2*a2_2*tau2
    a2 =                    2*a2_2
    j2 = 0.0

    # tratto 3
    tau3 = t - t2
    p3 = a0_3 + a1_3*tau3 + a2_3*tau3**2 + a3_3*tau3**3
    v3 =        a1_3      + 2*a2_3*tau3  + 3*a3_3*tau3**2
    a3 =                    2*a2_3       + 6*a3_3*tau3
    j3 =                                   6*a3_3

    # tratto 4 (accelerazione nulla se A3=0)
    tau4 = t - t3
    p4 = a0_4 + a1_4*tau4 + a2_4*tau4**2
    v4 =        a1_4      + 2*a2_4*tau4
    a4 =                    2*a2_4
    j4 = 0.0

    # tratto 5
    tau5 = t - t4
    p5 = a0_5 + a1_5*tau5 + a2_5*tau5**2 + a3_5*tau5**3
    v5 =        a1_5      + 2*a2_5*tau5  + 3*a3_5*tau5**2
    a5 =                    2*a2_5       + 6*a3_5*tau5
    j5 =                                   6*a3_5

    # tratto 6
    tau6 = t - t5
```

```
    p6 = a0_6 + a1_6*tau6 + a2_6*tau6**2
    v6 =        a1_6      + 2*a2_6*tau6
    a6 =                    2*a2_6
    j6 = 0.0

    # tratto 7
    tau7 = t - t6
    p7 = a0_7 + a1_7*tau7 + a2_7*tau7**2 + a3_7*tau7**3
    v7 =        a1_7      + 2*a2_7*tau7  + 3*a3_7*tau7**2
    a7 =                    2*a2_7       + 6*a3_7*tau7
    j7 =                                   6*a3_7

    p = ca.if_else(t <= t0, 0.0,
        ca.if_else(t <= t1, p1,
        ca.if_else(t <= t2, p2,
        ca.if_else(t <= t3, p3,
        ca.if_else(t <= t4, p4,
        ca.if_else(t <= t5, p5,
        ca.if_else(t <= t6, p6,
        ca.if_else(t <= t7, p7, s_tot_m)))))))

    v = ca.if_else(t <= t0, 0.0,
        ca.if_else(t <= t1, v1,
        ca.if_else(t <= t2, v2,
        ca.if_else(t <= t3, v3,
        ca.if_else(t <= t4, v4,
        ca.if_else(t <= t5, v5,
        ca.if_else(t <= t6, v6,
        ca.if_else(t <= t7, v7, 0.0)))))))

    a = ca.if_else(t <= t0, 0.0,
        ca.if_else(t <= t1, a1,
        ca.if_else(t <= t2, a2,
        ca.if_else(t <= t3, a3,
        ca.if_else(t <= t4, a4,
        ca.if_else(t <= t5, a5,
        ca.if_else(t <= t6, a6,
        ca.if_else(t <= t7, a7, 0.0)))))))

    j = ca.if_else(t <= t0, 0.0,
        ca.if_else(t <= t1, j1,
        ca.if_else(t <= t2, j2,
        ca.if_else(t <= t3, j3,
        ca.if_else(t <= t4, j4,
        ca.if_else(t <= t5, j5,
        ca.if_else(t <= t6, j6,
        ca.if_else(t <= t7, j7, 0.0)))))))

    return p, v, a, j
```

```
# ======================================================================
# Spettro analitico della trapezia standard in CasADi
# ======================================================================

def calc_aMax_aMin_trapStd_casadi(DT, s_tot):
    DT1, DT2, DT3, DT4, DT5, DT6, DT7 = DT[0], DT[1], DT[2], DT[3], DT[4], DT[5], DT[6]

    c1 = (DT1*DT7**2 + 2*DT2*DT7**2 + DT3*DT7**2)/(2*DT7) + DT3*DT6/2 + DT2*DT6 + DT1*DT6/2 + (DT1*DT5**2 + 2*DT2*DT5**2 + DT3*DT5**2)/(2*DT5) + \
        (DT3/2 + DT2 + DT1/2)*DT4 + (DT1*DT3**2 + 2*DT2*DT3**2 + (2.0/3.0)*DT3**3)/(2*DT3) + DT1*DT2/2 + DT2**2/2 + DT1**2/6
    c2 = (-DT5*DT7**2 - 2*DT6*DT7**2 - (2.0/3.0)*DT7**3)/(2*DT7) - DT6**2/2 - DT5*DT6/2 - DT5**2/6
    c3 =  DT3/2 + DT2 + DT1/2
    c4 = -DT7/2 - DT6 - DT5/2

    denom = (c1*c4 - c2*c3)

    aMax =  c4 * s_tot / denom
    aMin = -c3 * s_tot / denom
    return aMax, aMin


def trapStd_amplitude_at_omega_casadi(w, DT, s_tot, t0=0.0):

    #Ampiezza |A(j)| della trasformata della legge di accelerazione trapezia standard (7 segmenti),
    #valutata in un singolo  (rad/s). Formula analitica.

    DT1, DT2, DT3, DT4, DT5, DT6, DT7 = DT[0], DT[1], DT[2], DT[3], DT[4], DT[5], DT[6]

    # switch times
    t1 = t0 + DT1
    t2 = t1 + DT2
    t3 = t2 + DT3
    t4 = t3 + DT4
    t5 = t4 + DT5
    t6 = t5 + DT6
    t7 = t6 + DT7

    aMax, aMin = calc_aMax_aMin_trapStd_casadi(DT, s_tot)

    sin_wt0 = ca.sin(w*t0);   cos_wt0 = ca.cos(w*t0)
    sin_wt1 = ca.sin(w*t1);   cos_wt1 = ca.cos(w*t1)
    sin_wt2 = ca.sin(w*t2);   cos_wt2 = ca.cos(w*t2)
    sin_wt3 = ca.sin(w*t3);   cos_wt3 = ca.cos(w*t3)
```

```
271     sin_wt4 = ca.sin(w*t4);   cos_wt4 = ca.cos(w*t4)
272     sin_wt5 = ca.sin(w*t5);   cos_wt5 = ca.cos(w*t5)
273     sin_wt6 = ca.sin(w*t6);   cos_wt6 = ca.cos(w*t6)
274     sin_wt7 = ca.sin(w*t7);   cos_wt7 = ca.cos(w*t7)
275
276     RePart = -aMin*t6*sin_wt7/(DT7*w) + aMin*t7*sin_wt7/(DT7*w) - aMin*
        ↪ t5*sin_wt7/(DT5*w) - aMax*t3*sin_wt3/(DT3*w) + \
277             aMax*t2*sin_wt3/(DT3*w) + aMin*t4*sin_wt7/(DT5*w) - aMax*
        ↪ t0*sin_wt3/(DT1*w) + aMax*t1*sin_wt3/(DT1*w) + \
278             aMax*cos_wt1/(DT1*w**2) - aMax*cos_wt3/(DT3*w**2) + aMax*
        ↪ cos_wt2/(DT3*w**2) + aMin*cos_wt4/(DT5*w**2) - \
279             aMin*cos_wt5/(DT5*w**2) + aMin*cos_wt7/(DT7*w**2) - aMin*
        ↪ cos_wt6/(DT7*w**2) - aMax*cos_wt0/(DT1*w**2)
280
281     ImPart = -aMax*t0*cos_wt3/(DT1*w) - aMin*t5*cos_wt7/(DT5*w) + aMin*
        ↪ t7*cos_wt7/(DT7*w) - aMin*t6*cos_wt7/(DT7*w) + \
282             aMax*t1*cos_wt3/(DT1*w) - aMax*t3*cos_wt3/(DT3*w) + aMax*
        ↪ t2*cos_wt3/(DT3*w) + aMin*t4*cos_wt7/(DT5*w) - \
283             aMax*sin_wt1/(DT1*w**2) + aMax*sin_wt3/(DT3*w**2) - aMax*
        ↪ sin_wt2/(DT3*w**2) - aMin*sin_wt4/(DT5*w**2) + \
284             aMin*sin_wt6/(DT7*w**2) - aMin*sin_wt7/(DT7*w**2) + aMin*
        ↪ sin_wt5/(DT5*w**2) + aMax*sin_wt0/(DT1*w**2)
285
286     A = ca.sqrt(RePart**2 + ImPart**2)
287     return A
288
289
290 def trapStd_ReIm_at_omega_casadi(w, DT, s_tot, t0=0.0):
291
292     #Parte reale e immaginaria della trasformata (accelerazione) della
        ↪ trapezia standard a 7 segmenti.
293
294     #Per la robustezza  meglio lavorare su Re/Im:
295     #- A()=|X()| va a zero solo se Re()=0 e Im()=0
296     #- imporre/penalizzare anche d(Re)/d e d(Im)/d in =n rende lo zero
        ↪ "piatto" (notch pi largo localmente)
297     #
298     #Nota: la formula contiene termini 1/ e 1/, quindi va usata per >0
        ↪ (qui n>0).
299
300     DT1, DT2, DT3, DT4, DT5, DT6, DT7 = DT[0], DT[1], DT[2], DT[3], DT
        ↪ [4], DT[5], DT[6]
301
302     # switch times
303     t1 = t0 + DT1
304     t2 = t1 + DT2
305     t3 = t2 + DT3
306     t4 = t3 + DT4
307     t5 = t4 + DT5
308     t6 = t5 + DT6
309     t7 = t6 + DT7
```

```
310
311     aMax, aMin = calc_aMax_aMin_trapStd_casadi(DT, s_tot)
312
313     sin_wt0 = ca.sin(w*t0);   cos_wt0 = ca.cos(w*t0)
314     sin_wt1 = ca.sin(w*t1);   cos_wt1 = ca.cos(w*t1)
315     sin_wt2 = ca.sin(w*t2);   cos_wt2 = ca.cos(w*t2)
316     sin_wt3 = ca.sin(w*t3);   cos_wt3 = ca.cos(w*t3)
317     sin_wt4 = ca.sin(w*t4);   cos_wt4 = ca.cos(w*t4)
318     sin_wt5 = ca.sin(w*t5);   cos_wt5 = ca.cos(w*t5)
319     sin_wt6 = ca.sin(w*t6);   cos_wt6 = ca.cos(w*t6)
320     sin_wt7 = ca.sin(w*t7);   cos_wt7 = ca.cos(w*t7)
321
322     RePart = -aMin*t6*sin_wt7/(DT7*w) + aMin*t7*sin_wt7/(DT7*w) - aMin*
    ↪ t5*sin_wt7/(DT5*w) - aMax*t3*sin_wt3/(DT3*w) + \
323             aMax*t2*sin_wt3/(DT3*w) + aMin*t4*sin_wt7/(DT5*w) - aMax*
    ↪ t0*sin_wt3/(DT1*w) + aMax*t1*sin_wt3/(DT1*w) + \
324             aMax*cos_wt1/(DT1*w**2) - aMax*cos_wt3/(DT3*w**2) + aMax*
    ↪ cos_wt2/(DT3*w**2) + aMin*cos_wt4/(DT5*w**2) - \
325             aMin*cos_wt5/(DT5*w**2) + aMin*cos_wt7/(DT7*w**2) - aMin*
    ↪ cos_wt6/(DT7*w**2) - aMax*cos_wt0/(DT1*w**2)
326
327     ImPart = -aMax*t0*cos_wt3/(DT1*w) - aMin*t5*cos_wt7/(DT5*w) + aMin*
    ↪ t7*cos_wt7/(DT7*w) - aMin*t6*cos_wt7/(DT7*w) + \
328             aMax*t1*cos_wt3/(DT1*w) - aMax*t3*cos_wt3/(DT3*w) + aMax*
    ↪ t2*cos_wt3/(DT3*w) + aMin*t4*cos_wt7/(DT5*w) - \
329             aMax*sin_wt1/(DT1*w**2) + aMax*sin_wt3/(DT3*w**2) - aMax*
    ↪ sin_wt2/(DT3*w**2) - aMin*sin_wt4/(DT5*w**2) + \
330             aMin*sin_wt6/(DT7*w**2) - aMin*sin_wt7/(DT7*w**2) + aMin*
    ↪ sin_wt5/(DT5*w**2) + aMax*sin_wt0/(DT1*w**2)
331
332     return RePart, ImPart
333
334
335
336 # ================================================================
    ↪ =================
337 # FFT (per confronto)
338 # ================================================================
    ↪ =================
339
340 def calc_trajectory_fft_np(t, acc):
341     T = t[1] - t[0]
342     Fs = 1.0 / T
343     L = len(acc)
344     n = int(2**np.ceil(np.log2(10 * L)))
345
346     Y = np.fft.fft(acc, n)
347     trasf_calc = np.abs(Y / L)
348     trasf = trasf_calc[:n//2 + 1]
349     if trasf.size > 2:
350         trasf[1:-1] *= 2.0
```

```
351
352     f = Fs * np.arange(0, n//2 + 1) / n
353     w = f * 2*np.pi
354     return w, trasf
355
356
357 def trapStd_amplitude_spectrum_np(w, DT_vec, s_tot, t0=0.0):
358
359     #Versione NumPy (vettoriale) per plottare A() della trapezia
    ↪ standard
360
361     DT1, DT2, DT3, DT4, DT5, DT6, DT7 = DT_vec
362
363     t1 = t0 + DT1
364     t2 = t1 + DT2
365     t3 = t2 + DT3
366     t4 = t3 + DT4
367     t5 = t4 + DT5
368     t6 = t5 + DT6
369     t7 = t6 + DT7
370
371     # aMax/aMin
372     c1 = (DT1*DT7**2 + 2*DT2*DT7**2 + DT3*DT7**2)/(2*DT7) + DT3*DT6/2 +
    ↪  DT2*DT6 + DT1*DT6/2 + (DT1*DT5**2 + 2*DT2*DT5**2 + DT3*DT5**2)
    ↪ /(2*DT5) + \
373          (DT3/2 + DT2 + DT1/2)*DT4 + (DT1*DT3**2 + 2*DT2*DT3**2 +
    ↪ (2.0/3.0)*DT3**3)/(2*DT3) + DT1*DT2/2 + DT2**2/2 + DT1**2/6
374     c2 = (-DT5*DT7**2 - 2*DT6*DT7**2 - (2.0/3.0)*DT7**3)/(2*DT7) - DT6
    ↪ **2/2 - DT5*DT6/2 - DT5**2/6
375     c3 =  DT3/2 + DT2 + DT1/2
376     c4 = -DT7/2 - DT6 - DT5/2
377     denom = (c1*c4 - c2*c3)
378     aMax =  c4 * s_tot / denom
379     aMin = -c3 * s_tot / denom
380
381     w = np.asarray(w, dtype=float)
382     sin_wt0 = np.sin(w*t0);   cos_wt0 = np.cos(w*t0)
383     sin_wt1 = np.sin(w*t1);   cos_wt1 = np.cos(w*t1)
384     sin_wt2 = np.sin(w*t2);   cos_wt2 = np.cos(w*t2)
385     sin_wt3 = np.sin(w*t3);   cos_wt3 = np.cos(w*t3)
386     sin_wt4 = np.sin(w*t4);   cos_wt4 = np.cos(w*t4)
387     sin_wt5 = np.sin(w*t5);   cos_wt5 = np.cos(w*t5)
388     sin_wt6 = np.sin(w*t6);   cos_wt6 = np.cos(w*t6)
389     sin_wt7 = np.sin(w*t7);   cos_wt7 = np.cos(w*t7)
390
391     with np.errstate(divide='ignore', invalid='ignore'):
392         RePart = -aMin*t6*sin_wt7/(DT7*w) + aMin*t7*sin_wt7/(DT7*w) -
    ↪ aMin*t5*sin_wt7/(DT5*w) - aMax*t3*sin_wt3/(DT3*w) + \
393                 aMax*t2*sin_wt3/(DT3*w) + aMin*t4*sin_wt7/(DT5*w) -
    ↪ aMax*t0*sin_wt3/(DT1*w) + aMax*t1*sin_wt3/(DT1*w) + \
```

```
                aMax*cos_wt1/(DT1*w**2) - aMax*cos_wt3/(DT3*w**2) + aMax*cos_wt2/(DT3*w**2) + aMin*cos_wt4/(DT5*w**2) - \
                aMin*cos_wt5/(DT5*w**2) + aMin*cos_wt7/(DT7*w**2) - aMin*cos_wt6/(DT7*w**2) - aMax*cos_wt0/(DT1*w**2)

        ImPart = -aMax*t0*cos_wt3/(DT1*w) - aMin*t5*cos_wt7/(DT5*w) + aMin*t7*cos_wt7/(DT7*w) - aMin*t6*cos_wt7/(DT7*w) + \
                aMax*t1*cos_wt3/(DT1*w) - aMax*t3*cos_wt3/(DT3*w) + aMax*t2*cos_wt3/(DT3*w) + aMin*t4*cos_wt7/(DT5*w) - \
                aMax*sin_wt1/(DT1*w**2) + aMax*sin_wt3/(DT3*w**2) - aMax*sin_wt2/(DT3*w**2) - aMin*sin_wt4/(DT5*w**2) + \
                aMin*sin_wt6/(DT7*w**2) - aMin*sin_wt7/(DT7*w**2) + aMin*sin_wt5/(DT5*w**2) + aMax*sin_wt0/(DT1*w**2)

        A = np.sqrt(RePart**2 + ImPart**2)

    # gestione ~0
    eps = 1e-6
    mask_small = np.abs(w) < eps
    if np.any(mask_small):
        A = np.asarray(A, dtype=float)
        A[mask_small] = 0.0
    return A


# ================================================================================
# OPTI (problema di ottimizzazione)
# ================================================================================

opti = ca.Opti()

# ============================================================
# Parametri sloshing
# ============================================================

R_mm = 48.5
h_mm = 82.0

R = R_mm / 1000.0
h = h_mm / 1000.0
rho = 1000.0
st = 0.073
mu = 1e-3
nu = mu / rho
g  = 9.81

xi = 1.84118
alpha_n = 0.58

```

```
437
438 # ============================================================
439 # Impostazioni moto
440 # ============================================================
441
442 s_tot_mm = 800.0
443 s_tot_m  = s_tot_mm / 1000.0
444
445 t_avanz = 1.0
446 t_rest  = 1.0
447 t_tot   = t_avanz + t_rest
448
449 # ============================================================
450 # Variabili decisionali:
451 # ============================================================
452
453 #Numero di campioni usati per costruire la griglia temporale
454 N_sim = 101
455
456 # durate grezze non negative
457 DT = opti.variable(7, 1)
458
459 opti.subject_to(ca.sum(DT) == t_avanz)
460
461 # durate minime (imposte sui DT effettivi)
462 opti.subject_to(DT[0] >= 0.05)
463 opti.subject_to(DT[1] >= 0.0)
464 opti.subject_to(DT[2] >= 0.05)
465 opti.subject_to(DT[3] >= 0.0)
466 opti.subject_to(DT[4] >= 0.05)
467 opti.subject_to(DT[5] >= 0.0)
468 opti.subject_to(DT[6] >= 0.05)
469
470 # Guess iniziale
471 w7 = np.array([0.14286, 0.14286, 0.14286, 0.14284, 0.14286, 0.14286,
    ↪ 0.14286])      #somma = 1
472
473 #  percentuale di tolleranza intorno a wn che definisce altre due
    ↪ frequenze  (es. 0.03 = 3%)
474 p_notch = 0.15
475
476 #pesi da dare poi, all'interno della funzione costo, alle ampiezze
    ↪ dello spettro calcolate nelle 3 frequenze
477 #(si sceglie quale minimizzare maggiormente)
478 b1 = 1.0    #peso su (1-p)wn
479 b2 = 1.0    #peso su wn
480 b3 = 1.0    #peso su (1+p)wn
481
482 #configurazione del risolutore numerico
483 opti.solver('ipopt', {
484     "ipopt.print_level": 3,         #livello di stampa
```

```
    "ipopt.max_iter": 2000,          #numero massimo di iterazioni
})


# ======================================================================
# Parametri fisici calcolati per caratterizzare la dinamica dello sloshing
# ======================================================================

m_f = math.pi * R**2 * h * rho
ms  = (m_f * 2.0 * R * math.tanh(xi * h / R)) / (xi * h * (xi**2 - 1.0))
wn  = math.sqrt((g * xi / R) * math.tanh(xi * h / R))

Re1 = nu / math.sqrt(g * R**3)
zeta_1 = 0.79 * math.sqrt(Re1) * (1.0 + (0.318 / math.sinh(1.84 * h / R)) * (1.0 + (1.0 - h / R) / math.cosh(1.84 * h / R)))


def C_n_val(xi_val, h_val, R_val):
    return xi_val * math.tanh(xi_val * (h_val / R_val))

Cn_const = C_n_val(xi, h, R)

# conversione da xn ad altezza di sloshing eta (metri)
K_eta = (h * (xi**2) * ms) / m_f


# ============================================================
# Multi-zero notch (stile ZWD / ZWDD)
# ============================================================
# Invece di imporre solo lo zero in n (e/o la pendenza tramite derivate),
#    imposto un notch robusto annullando la trasformata
#    complessa X()=Re()+j Im() su 3 frequenze vicine a n:
#
#        1 = (1 - p) n
#        2 = n
#        3 = (1 + p) n
#
#    con p = percentuale di tolleranza sull'incertezza di n.
#
# L'obiettivo diventa la somma (pesi uguali) dei moduli al quadrato:
#
#        J = |X(1)|^2 + |X(2)|^2 + |X(3)|^2
#
#    dove |X()|^2 = Re()^2 + Im()^2.
#
# Nota:
```

```
#  - Lavoro su Re/Im per evitare la radice (pi stabile numericamente).
#  - p pu essere aumentato se si vuole un notch pi largo (a costo di
    ↪ maggiore vincolo).


# frequenze target (rad/s)
w1_dm = ca.DM((1.0 - p_notch) * wn)
wn_dm = ca.DM(wn)
w3_dm = ca.DM((1.0 + p_notch) * wn)

# Re/Im in ciascuna frequenza
Re_w1, Im_w1 = trapStd_ReIm_at_omega_casadi(w1_dm, DT, s_tot_m, t0=0.0)
Re_wn, Im_wn = trapStd_ReIm_at_omega_casadi(wn_dm, DT, s_tot_m, t0=0.0)
Re_w3, Im_w3 = trapStd_ReIm_at_omega_casadi(w3_dm, DT, s_tot_m, t0=0.0)

# moduli al quadrato (niente sqrt)
A2_w1 = Re_w1**2 + Im_w1**2
A2_wn = Re_wn**2 + Im_wn**2
A2_w3 = Re_w3**2 + Im_w3**2

# moduli (solo per stampa)
A_w1 = ca.sqrt(A2_w1)
A_wn = ca.sqrt(A2_wn)
A_w3 = ca.sqrt(A2_w3)

# Funzione costo
#J = b2 * A2_wn
J = b1 * A2_w1 + b2 * A2_wn + b3 * A2_w3

opti.minimize(J)

# ============================================================
# Vincolo posizione finale a t_avanz
# ============================================================

coeff, t_switch = build_trap_coeffs_casadi(DT, s_tot_m)
p_fin, _, _, _ = trapStd_kinematics_casadi(t_avanz, coeff, t_switch,
    ↪ s_tot_m)
opti.subject_to(p_fin == s_tot_m)

# ============================================================
# Solver basato sul guess iniziale
# ============================================================

w7 = w7 / w7.sum()
DT_init = t_avanz * w7


opti.set_initial(DT, DT_init.reshape((7, 1)))

sol = opti.solve()
```

```
579
580 DT_opt = np.array(sol.value(DT)).flatten()
581 DT_opt_clean = np.where(np.abs(DT_opt) < 1e-8, 0.0, DT_opt)
582 # DT in percentuale (somma = 100) con 3 cifre significative
583 DT_pct = 100.0 * DT_opt_clean / DT_opt_clean.sum()
584
585 # arrotondo "a display" a 3 cifre significative e forzo la somma esatta
    ↪  a 100
586 DT_pct_disp = np.array([float(f"{p:.3g}") for p in DT_pct])
587 DT_pct_disp[-1] = 100.0 - DT_pct_disp[:-1].sum()
588
589 np.save("DT_opt.npy", DT_opt_clean)
590
591
592 A_wn_num = float(sol.value(A_wn))
593 # --- diagnostica multi-zero notch ---
594 A_w1_num = float(sol.value(A_w1))
595 A_w3_num = float(sol.value(A_w3))
596 J_num    = float(sol.value(J))
597 # ------------------------------------------------
598
599
600
601 # ================================================================
    ↪ ==
602 # Post-processing sloshing: integrazione RK4 del modello non lineare
603 # (NON entra nell'ottimizzazione: serve solo per plot e stampa
    ↪ indicatori)
604 # ================================================================
    ↪ ==
605
606 # griglia temporale per simulazione sloshing
607 _dt_sim = t_tot / (N_sim - 1)
608 t_grid_sim = np.linspace(0.0, t_tot, N_sim)
609 idx_moto_end = int(np.searchsorted(t_grid_sim, t_avanz, side="right"))
610
611 # Accelerazione a(t) come Function + map (riuso della stessa legge
    ↪ trapezia)
612 t_sym_sim  = ca.MX.sym('t')
613 DT_sym_sim = ca.MX.sym('DT', 7)
614 coeff_sym_sim, t_sw_sym_sim = build_trap_coeffs_casadi(DT_sym_sim,
    ↪ s_tot_m)
615 _, _, a_trap_sym_sim, _ = trapStd_kinematics_casadi(t_sym_sim,
    ↪ coeff_sym_sim, t_sw_sym_sim, s_tot_m)
616 a_fun_sim = ca.Function('a_fun_sim', [t_sym_sim, DT_sym_sim], [
    ↪ a_trap_sym_sim])
617
618 # a(t_k) sui nodi (serve per plot)
619 t_eval_sim = ca.DM(t_grid_sim).reshape((1, N_sim))
620 DT_rep_sim_full = ca.repmat(ca.DM(DT_opt_clean).reshape((7, 1)), 1,
    ↪ N_sim)
```

```
621 a_sol_sim = np.array(a_fun_sim.map(N_sim)(t_eval_sim, DT_rep_sim_full))
        ↪ .ravel()
622
623 # a(t_k) sui nodi sinistri per integrare su N_sim-1 step
624 # uso a(t_k) per lo step [t_k, t_{k+1}])
625 t_left_sim = ca.DM(t_grid_sim[:-1]).reshape((1, N_sim - 1))
626 DT_rep_left = ca.repmat(ca.DM(DT_opt_clean).reshape((7, 1)), 1, N_sim -
        ↪  1)
627 a_left = np.array(a_fun_sim.map(N_sim - 1)(t_left_sim, DT_rep_left)).
        ↪ ravel()
628
629 a_grid = ca.DM(a_left).reshape((1, N_sim - 1))
630 dt_grid = ca.DM.ones(1, N_sim - 1) * _dt_sim
631
632 # ---- dinamica non lineare + RK4
633 x_sym  = ca.MX.sym('x', 2)     # [xn, xnd]
634 a_in   = ca.MX.sym('a')        # ingresso: accelerazione a(t)
635 dt_sym = ca.MX.sym('dt')
636
637 xn_sym  = x_sym[0]
638 xnd_sym = x_sym[1]
639
640 Den = 1.0 + (Cn_const**2) * (xn_sym**2)
641 term1 = 2.0 * wn * zeta_1 * (xnd_sym + (Cn_const**2) * xnd_sym * (
        ↪ xn_sym**2))
642 term2 = (Cn_const**2) * xn_sym * (xnd_sym**2)
643 term3 = (wn**2) * xn_sym * (1.0 + alpha_n * (xn_sym**2))
644 term4 = a_in / R
645
646 Num  = -(term1 + term2 + term3 + term4)
647 xndd = Num / Den
648
649 xdot = ca.vertcat(xnd_sym, xndd)
650 f = ca.Function('f', [x_sym, a_in], [xdot])
651
652 k1 = f(x_sym, a_in)
653 k2 = f(x_sym + 0.5 * dt_sym * k1, a_in)
654 k3 = f(x_sym + 0.5 * dt_sym * k2, a_in)
655 k4 = f(x_sym + 1.0 * dt_sym * k3, a_in)
656
657 x_next = x_sym + (dt_sym / 6.0) * (k1 + 2.0 * k2 + 2.0 * k3 + k4)
658 RK4_step = ca.Function('RK4_step', [x_sym, a_in, dt_sym], [x_next])
659 RK4_accum = RK4_step.mapaccum(N_sim - 1)
660
661 x0 = ca.DM([0.0, 0.0])
662 x_acc = RK4_accum(x0, a_grid, dt_grid)          # 2 x (N_sim-1)
663 X = ca.horzcat(x0, x_acc)                        # 2 x N_sim
664
665 xn_grid_num = np.array(X[0, :]).ravel()          # (N_sim,)
666 eta_grid_m = K_eta * xn_grid_num                 # (N_sim,) metri
667 eta_grid_mm = 1000.0 * eta_grid_m                # (N_sim,) mm
```

```
668
669 eta_moto_mm = eta_grid_mm[:idx_moto_end]
670 eta_rest_mm = eta_grid_mm[idx_moto_end:]
671
672 eta_peak_opt_mm = float(np.max(np.abs(eta_moto_mm)))
673 eta_rest_max_mm = float(np.max(np.abs(eta_rest_mm)))
674
675 # ============================================================
676 # Stampe
677 # ============================================================
678
679 print("DT_opt = [" + " , ".join(f"{x:.6f}" for x in DT_opt_clean) + "]"
    ↪ )
680 print("DT_opt_tot = ", DT_opt_clean.sum())
681 print("Space_tot  = ", float(sol.value(p_fin)))
682 print(f"wn [rad/s] = {wn:.6f}")
683 print(f"|A(j*wn)| (analitico) = {A_wn_num:.6e}")
684 print(f"p_notch = {p_notch*100:.1f}%")
685 print(f"1 = {(1.0-p_notch)*wn:.6f} rad/s, |A(j1)| = {A_w1_num:.6e}")
686 print(f"2 = {wn:.6f} rad/s, |A(j2)| = {A_wn_num:.6e}")
687 print(f"3 = {(1.0+p_notch)*wn:.6f} rad/s, |A(j3)| = {A_w3_num:.6e}")
688 print(f"Objective J = {J_num:.6e}")
689 print("DT_opt [%] = [" + ", ".join(f"{p:.3g}" for p in DT_pct_disp) + "
    ↪ ]")
690 print(f"eta_peak_opt [mm] = {eta_peak_opt_mm:.6f}")
691 print(f"eta_rest_max [mm] = {eta_rest_max_mm:.6f}")
692
693
694
695
696
697 # ============================================================================
    ↪ =================
698 # Plot
699 # ============================================================================
    ↪ =================
700
701 # ========================
702 # FIGURA 1: a(t) + eta(t)
703 # ========================
704
705 fig1, (ax1, ax2) = plt.subplots(2, 1, sharex=True, figsize=(11, 7))
706
707 ax1.plot(t_grid_sim, a_sol_sim, linewidth=1.5)
708 ax1.axvline(t_avanz, linestyle="--", linewidth=1.2)
709 ax1.grid(True, which='both', linestyle="--", linewidth=0.6, alpha=0.5)
710 ax1.set_ylabel("a(t) [m/s]")
711 ax1.set_title("Legge di accelerazione e altezza di sloshing (post-
    ↪ processing)")
712
713 ax2.plot(t_grid_sim, eta_grid_mm, linewidth=1.2)
```

```
ax2.axvline(t_avanz, linestyle="--", linewidth=1.2)
ax2.grid(True, which='both', linestyle="--", linewidth=0.6, alpha=0.5)
ax2.set_xlabel("t [s]")
ax2.set_ylabel("eta(t) [mm]")

# (opzionale) linee di riferimento nella fase di riposo
# Qui NON sono vincoli dell'ottimizzazione: sono solo un riferimento grafico.
#eta_rest_limit_mm = 1
#ax2.hlines(+eta_rest_limit_mm, xmin=t_avanz, xmax=t_tot, colors="red", linestyles="--", linewidth=1.6)
#ax2.hlines(-eta_rest_limit_mm, xmin=t_avanz, xmax=t_tot, colors="red", linestyles="--", linewidth=1.6)

plt.tight_layout()


# ================================================
# FIGURA 2: spettro (analitico) + FFT per confronto
# ================================================

# Ricostruisco a(t) su una griglia fitta per fare una FFT pi pulita
N_plot = 2001
t_grid_plot = np.linspace(0.0, t_tot, N_plot)

t_eval = ca.DM(t_grid_plot).reshape((1, N_plot))
DT_rep = ca.repmat(ca.DM(DT_opt_clean).reshape((7, 1)), 1, N_plot)
a_sol_plot = np.array(a_fun_sim.map(N_plot)(t_eval, DT_rep)).ravel()

# Spettro analitico su un asse di frequenze
w_plot = np.linspace(0.0, 3.0 * wn, 2500)
A_plot = trapStd_amplitude_spectrum_np(w_plot, DT_opt_clean, s_tot_m, t0=0.0)

# per confrontarlo con la FFT (che  single-sided) uso lo stesso scaling gi adottato
A_plot_fft = (2.0 / t_avanz) * A_plot

# FFT numerica dell'accelerazione (solo tratto di moto)
mask_moto = t_grid_plot <= t_avanz + 1e-12
w_fft, A_fft = calc_trajectory_fft_np(t_grid_plot[mask_moto], a_sol_plot[mask_moto])

fig2, ax = plt.subplots(1, 1, figsize=(11, 4.5))
ax.plot(w_plot, A_plot_fft, linewidth=1.8, label="Analitico |A(j)|")
#ax.plot(w_fft, A_fft, linestyle="--", linewidth=1.2, label="FFT (check numerico)")
ax.axvline(wn, linestyle="--", c='r', linewidth=1.5, label=r"$\omega_n$")
ax.axvline((1.0 - p_notch) * wn, linestyle=":", linewidth=1.2, label=r"$\omega_1$")
```

```
ax.axvline((1.0 + p_notch) * wn, linestyle=":", linewidth=1.2, label=r"
    ↪ $\omega_3$")
ax.set_xlim(0.0, 3.0 * wn)
ax.grid(True, which='both', linestyle="--", linewidth=0.6, alpha=0.5)
ax.set_xlabel(r"$\omega$ [rad/s]")
ax.set_ylabel(r"$|A(j\omega)|$")
#ax.set_title("Spettro di ampiezza dell'accelerazione (focus su n)")
ax.legend()

plt.tight_layout()
plt.show()
```

Listing B.1: Script di ottimizzazione: Opt_spettro_multizero

Appendice C

Codice ottimizzazione spettrale con minimizzazione jerk su banda

```
1
2  import casadi as ca
3  import numpy as np
4  import math
5  import matplotlib.pyplot as plt
6
7
8  # ================================================================
       ↪ ==================
9  # Trapezia standard (7 segmenti): coefficienti polinomiali + cinematica
10 # ================================================================
       ↪ ==================
11
12 def build_trap_coeffs_casadi(DT, s_tot_m):
13
14     #Versione CasADi di calc_trap:
15     #DT: MX(7,1) con [DT1..DT7]
16     #s_tot_m: spostamento totale scalare [m]
17     #Ritorna:
18     #  coeff: MX(24,1) con coefficienti polinomiali e valori di jerk
19     #  t_switch: MX(8,1) con [t0..t7]
20
21     DT1, DT2, DT3, DT4, DT5, DT6, DT7 = DT[0], DT[1], DT[2], DT[3], DT
       ↪ [4], DT[5], DT[6]
22
23     t0 = 0
24     t1 = DT1
25     t2 = DT1 + DT2
26     t3 = DT1 + DT2 + DT3
27     t4 = DT1 + DT2 + DT3 + DT4
28     t5 = DT1 + DT2 + DT3 + DT4 + DT5
29     t6 = DT1 + DT2 + DT3 + DT4 + DT5 + DT6
30     t7 = DT1 + DT2 + DT3 + DT4 + DT5 + DT6 + DT7
31
32     # condizioni al contorno (rest-to-rest)
```

```
    pos_0 = 0.0
    vel_0 = 0.0
    acc_0 = 0.0

    pos_7 = s_tot_m
    vel_7 = 0.0
    acc_7 = 0.0

    # questa legge ha il tratto 4 a accelerazione nulla (velocit
    ↪ costante)
    A3 = 0.0

    a0_1 = pos_0
    a1_1 = vel_0
    a2_1 = acc_0 / 2.0
    a2_4 = A3 / 2.0
    a2_5 = a2_4

    A = ca.MX.zeros(24, 24)

    A[0, 0]  = 6.0;        A[0, 20] = -1.0
    A[1, 0]  = -DT1**3;    A[1, 1]  = 1.0
    A[2, 2]  = 1.0;        A[2, 0]  = -3.0 * DT1**2
    A[3, 3]  = 2.0;        A[3, 0]  = -6.0 * DT1

    A[4, 4]  = -1.0;       A[4, 1]  = 1.0;  A[4, 2] = DT2;       A[4,
    ↪ 3] = DT2**2
    A[5, 5]  = -1.0;       A[5, 2]  = 1.0;  A[5, 3] = 2.0 * DT2
    A[6, 6]  = 1.0;        A[6, 3]  = -1.0

    A[7, 7]  = 6.0;        A[7, 21] = -1.0
    A[8, 8]  = -1.0;       A[8, 4]  = 1.0;  A[8, 5] = DT3;       A[8,
    ↪ 6] = DT3**2;      A[8, 7] = DT3**3
    A[9, 9]  = -1.0;       A[9, 5]  = 1.0;  A[9, 6] = 2.0 * DT3; A[9,
    ↪ 7] = 3.0 * DT3**2
    A[10, 6] = 1.0;        A[10, 7] = 3.0 * DT3

    A[11, 10] = 1.0;       A[11, 8] = -1.0; A[11, 9] = -DT4
    A[12, 11] = 1.0;       A[12, 9] = -1.0

    A[13, 12] = 6.0;       A[13, 22] = -1.0
    A[14, 13] = 1.0;       A[14, 10] = -1.0; A[14, 11] = -DT5;       A
    ↪ [14, 12] = -DT5**3
    A[15, 14] = 1.0;       A[15, 11] = -1.0; A[15, 12] = -3.0 * DT5**2
    A[16, 15] = 1.0;       A[16, 12] = -3.0 * DT5

    A[17, 16] = -1.0;      A[17, 13] = 1.0; A[17, 14] = DT6;          A
    ↪ [17, 15] = DT6**2
    A[18, 17] = -1.0;      A[18, 14] = 1.0; A[18, 15] = 2.0 * DT6
    A[19, 18] = -1.0;      A[19, 15] = 1.0

```

```
    A[20, 19] = 6.0;        A[20, 23] = 1.0
    A[21, 16] = 1.0;        A[21, 17] = DT7;      A[21, 18] = DT7**2;
    ↪ A[21, 19] = DT7**3
    A[22, 17] = 1.0;        A[22, 18] = 2.0 * DT7; A[22, 19] = 3.0 * DT7
    ↪ **2
    A[23, 18] = 2.0;        A[23, 19] = 6.0 * DT7

    b = ca.vertcat(
        0,
        a0_1 + a1_1 * DT1 + a2_1 * DT1**2,
        a1_1 + 2.0 * a2_1 * DT1,
        2.0 * a2_1,
        0.0,
        0.0,
        0.0,
        0.0,
        0.0,
        0.0,
        a2_4,
        a2_4 * DT4**2,
        2.0 * a2_4 * DT4,
        0.0,
        a2_5 * DT5**2,
        2.0 * a2_5 * DT5,
        a2_5,
        0.0,
        0.0,
        0.0,
        0.0,
        pos_7,
        vel_7,
        acc_7
    )

    coeff = ca.solve(A, b)
    t_switch = ca.vertcat(t0, t1, t2, t3, t4, t5, t6, t7)
    return coeff, t_switch


def trapStd_kinematics_casadi(t, coeff, t_sw, s_tot_m):

    #Ritorna p(t), v(t), a(t), j(t) (MX) per la legge a 7 tratti.

    t0, t1, t2, t3, t4, t5, t6, t7 = [t_sw[i] for i in range(8)]

    pos_0 = 0.0
    vel_0 = 0.0
    acc_0 = 0.0
    A3    = 0.0

    a0_1 = pos_0
```

```
    a1_1 = vel_0
    a2_1 = acc_0 / 2.0
    a2_4 = A3 / 2.0
    a2_5 = a2_4

    # estrazione dei coefficienti
    a3_1 = coeff[0]
    a0_2, a1_2, a2_2 = coeff[1],  coeff[2],  coeff[3]
    a0_3, a1_3, a2_3, a3_3 = coeff[4], coeff[5], coeff[6], coeff[7]
    a0_4, a1_4 = coeff[8], coeff[9]
    a0_5, a1_5, a3_5 = coeff[10], coeff[11], coeff[12]
    a0_6, a1_6, a2_6 = coeff[13], coeff[14], coeff[15]
    a0_7, a1_7, a2_7, a3_7 = coeff[16], coeff[17], coeff[18], coeff[19]

    # tratto 1
    tau1 = t - t0
    p1 = a0_1 + a1_1*tau1 + a2_1*tau1**2 + a3_1*tau1**3
    v1 =        a1_1      + 2*a2_1*tau1  + 3*a3_1*tau1**2
    a1 =                    2*a2_1       + 6*a3_1*tau1
    j1 =                                   6*a3_1

    # tratto 2
    tau2 = t - t1
    p2 = a0_2 + a1_2*tau2 + a2_2*tau2**2
    v2 =        a1_2      + 2*a2_2*tau2
    a2 =                    2*a2_2
    j2 = 0.0

    # tratto 3
    tau3 = t - t2
    p3 = a0_3 + a1_3*tau3 + a2_3*tau3**2 + a3_3*tau3**3
    v3 =        a1_3      + 2*a2_3*tau3  + 3*a3_3*tau3**2
    a3 =                    2*a2_3       + 6*a3_3*tau3
    j3 =                                   6*a3_3

    # tratto 4 (accelerazione nulla se A3=0)
    tau4 = t - t3
    p4 = a0_4 + a1_4*tau4 + a2_4*tau4**2
    v4 =        a1_4      + 2*a2_4*tau4
    a4 =                    2*a2_4
    j4 = 0.0

    # tratto 5
    tau5 = t - t4
    p5 = a0_5 + a1_5*tau5 + a2_5*tau5**2 + a3_5*tau5**3
    v5 =        a1_5      + 2*a2_5*tau5  + 3*a3_5*tau5**2
    a5 =                    2*a2_5       + 6*a3_5*tau5
    j5 =                                   6*a3_5

    # tratto 6
    tau6 = t - t5
```

```
178     p6 = a0_6 + a1_6*tau6 + a2_6*tau6**2
179     v6 =        a1_6      + 2*a2_6*tau6
180     a6 =                    2*a2_6
181     j6 = 0.0
182
183     # tratto 7
184     tau7 = t - t6
185     p7 = a0_7 + a1_7*tau7 + a2_7*tau7**2 + a3_7*tau7**3
186     v7 =        a1_7      + 2*a2_7*tau7  + 3*a3_7*tau7**2
187     a7 =                    2*a2_7       + 6*a3_7*tau7
188     j7 =                                   6*a3_7
189
190     p = ca.if_else(t <= t0, 0.0,
191         ca.if_else(t <= t1, p1,
192         ca.if_else(t <= t2, p2,
193         ca.if_else(t <= t3, p3,
194         ca.if_else(t <= t4, p4,
195         ca.if_else(t <= t5, p5,
196         ca.if_else(t <= t6, p6,
197         ca.if_else(t <= t7, p7, s_tot_m)))))))
198
199     v = ca.if_else(t <= t0, 0.0,
200         ca.if_else(t <= t1, v1,
201         ca.if_else(t <= t2, v2,
202         ca.if_else(t <= t3, v3,
203         ca.if_else(t <= t4, v4,
204         ca.if_else(t <= t5, v5,
205         ca.if_else(t <= t6, v6,
206         ca.if_else(t <= t7, v7, 0.0)))))))
207
208     a = ca.if_else(t <= t0, 0.0,
209         ca.if_else(t <= t1, a1,
210         ca.if_else(t <= t2, a2,
211         ca.if_else(t <= t3, a3,
212         ca.if_else(t <= t4, a4,
213         ca.if_else(t <= t5, a5,
214         ca.if_else(t <= t6, a6,
215         ca.if_else(t <= t7, a7, 0.0)))))))
216
217     j = ca.if_else(t <= t0, 0.0,
218         ca.if_else(t <= t1, j1,
219         ca.if_else(t <= t2, j2,
220         ca.if_else(t <= t3, j3,
221         ca.if_else(t <= t4, j4,
222         ca.if_else(t <= t5, j5,
223         ca.if_else(t <= t6, j6,
224         ca.if_else(t <= t7, j7, 0.0)))))))
225
226     return p, v, a, j
227
228
```

```
# ======================================================================
    ↪ =================
# Spettro analitico della trapezia standard in CasADi
# ======================================================================
    ↪ =================

def calc_aMax_aMin_trapStd_casadi(DT, s_tot):
    DT1, DT2, DT3, DT4, DT5, DT6, DT7 = DT[0], DT[1], DT[2], DT[3], DT
    ↪ [4], DT[5], DT[6]

    c1 = (DT1*DT7**2 + 2*DT2*DT7**2 + DT3*DT7**2)/(2*DT7) + DT3*DT6/2 +
    ↪  DT2*DT6 + DT1*DT6/2 + (DT1*DT5**2 + 2*DT2*DT5**2 + DT3*DT5**2)
    ↪ /(2*DT5) + \
          (DT3/2 + DT2 + DT1/2)*DT4 + (DT1*DT3**2 + 2*DT2*DT3**2 +
    ↪ (2.0/3.0)*DT3**3)/(2*DT3) + DT1*DT2/2 + DT2**2/2 + DT1**2/6
    c2 = (-DT5*DT7**2 - 2*DT6*DT7**2 - (2.0/3.0)*DT7**3)/(2*DT7) - DT6
    ↪ **2/2 - DT5*DT6/2 - DT5**2/6
    c3 =  DT3/2 + DT2 + DT1/2
    c4 = -DT7/2 - DT6 - DT5/2

    denom = (c1*c4 - c2*c3)

    aMax =  c4 * s_tot / denom
    aMin = -c3 * s_tot / denom
    return aMax, aMin


def trapStd_amplitude_at_omega_casadi(w, DT, s_tot, t0=0.0):

    #Ampiezza |A(j)| della trasformata della legge di accelerazione
    ↪ trapezia standard (7 segmenti),
    #valutata in un singolo  (rad/s). Formula analitica.

    DT1, DT2, DT3, DT4, DT5, DT6, DT7 = DT[0], DT[1], DT[2], DT[3], DT
    ↪ [4], DT[5], DT[6]

    # switch times
    t1 = t0 + DT1
    t2 = t1 + DT2
    t3 = t2 + DT3
    t4 = t3 + DT4
    t5 = t4 + DT5
    t6 = t5 + DT6
    t7 = t6 + DT7

    aMax, aMin = calc_aMax_aMin_trapStd_casadi(DT, s_tot)

    sin_wt0 = ca.sin(w*t0);   cos_wt0 = ca.cos(w*t0)
    sin_wt1 = ca.sin(w*t1);   cos_wt1 = ca.cos(w*t1)
    sin_wt2 = ca.sin(w*t2);   cos_wt2 = ca.cos(w*t2)
    sin_wt3 = ca.sin(w*t3);   cos_wt3 = ca.cos(w*t3)
```

```
271     sin_wt4 = ca.sin(w*t4);   cos_wt4 = ca.cos(w*t4)
272     sin_wt5 = ca.sin(w*t5);   cos_wt5 = ca.cos(w*t5)
273     sin_wt6 = ca.sin(w*t6);   cos_wt6 = ca.cos(w*t6)
274     sin_wt7 = ca.sin(w*t7);   cos_wt7 = ca.cos(w*t7)
275
276     RePart = -aMin*t6*sin_wt7/(DT7*w) + aMin*t7*sin_wt7/(DT7*w) - aMin*
        ↪ t5*sin_wt7/(DT5*w) - aMax*t3*sin_wt3/(DT3*w) + \
277             aMax*t2*sin_wt3/(DT3*w) + aMin*t4*sin_wt7/(DT5*w) - aMax*
        ↪ t0*sin_wt3/(DT1*w) + aMax*t1*sin_wt3/(DT1*w) + \
278             aMax*cos_wt1/(DT1*w**2) - aMax*cos_wt3/(DT3*w**2) + aMax*
        ↪ cos_wt2/(DT3*w**2) + aMin*cos_wt4/(DT5*w**2) - \
279             aMin*cos_wt5/(DT5*w**2) + aMin*cos_wt7/(DT7*w**2) - aMin*
        ↪ cos_wt6/(DT7*w**2) - aMax*cos_wt0/(DT1*w**2)
280
281     ImPart = -aMax*t0*cos_wt3/(DT1*w) - aMin*t5*cos_wt7/(DT5*w) + aMin*
        ↪ t7*cos_wt7/(DT7*w) - aMin*t6*cos_wt7/(DT7*w) + \
282             aMax*t1*cos_wt3/(DT1*w) - aMax*t3*cos_wt3/(DT3*w) + aMax*
        ↪ t2*cos_wt3/(DT3*w) + aMin*t4*cos_wt7/(DT5*w) - \
283             aMax*sin_wt1/(DT1*w**2) + aMax*sin_wt3/(DT3*w**2) - aMax*
        ↪ sin_wt2/(DT3*w**2) - aMin*sin_wt4/(DT5*w**2) + \
284             aMin*sin_wt6/(DT7*w**2) - aMin*sin_wt7/(DT7*w**2) + aMin*
        ↪ sin_wt5/(DT5*w**2) + aMax*sin_wt0/(DT1*w**2)
285
286     A = ca.sqrt(RePart**2 + ImPart**2)
287     return A
288
289
290 def trapStd_ReIm_at_omega_casadi(w, DT, s_tot, t0=0.0):
291
292     #Parte reale e immaginaria della trasformata (accelerazione) della
        ↪ trapezia standard a 7 segmenti.
293
294     #Per la robustezza  meglio lavorare su Re/Im:
295     #- A()=|X()| va a zero solo se Re()=0 e Im()=0
296     #- imporre/penalizzare anche d(Re)/d e d(Im)/d in =n rende lo zero
        ↪ "piatto" (notch pi largo localmente)
297     #
298     #Nota: la formula contiene termini 1/ e 1/, quindi va usata per >0
        ↪ (qui n>0).
299
300     DT1, DT2, DT3, DT4, DT5, DT6, DT7 = DT[0], DT[1], DT[2], DT[3], DT
        ↪ [4], DT[5], DT[6]
301
302     # switch times
303     t1 = t0 + DT1
304     t2 = t1 + DT2
305     t3 = t2 + DT3
306     t4 = t3 + DT4
307     t5 = t4 + DT5
308     t6 = t5 + DT6
309     t7 = t6 + DT7
```

```
310
311     aMax, aMin = calc_aMax_aMin_trapStd_casadi(DT, s_tot)
312
313     sin_wt0 = ca.sin(w*t0);   cos_wt0 = ca.cos(w*t0)
314     sin_wt1 = ca.sin(w*t1);   cos_wt1 = ca.cos(w*t1)
315     sin_wt2 = ca.sin(w*t2);   cos_wt2 = ca.cos(w*t2)
316     sin_wt3 = ca.sin(w*t3);   cos_wt3 = ca.cos(w*t3)
317     sin_wt4 = ca.sin(w*t4);   cos_wt4 = ca.cos(w*t4)
318     sin_wt5 = ca.sin(w*t5);   cos_wt5 = ca.cos(w*t5)
319     sin_wt6 = ca.sin(w*t6);   cos_wt6 = ca.cos(w*t6)
320     sin_wt7 = ca.sin(w*t7);   cos_wt7 = ca.cos(w*t7)
321
322     RePart = -aMin*t6*sin_wt7/(DT7*w) + aMin*t7*sin_wt7/(DT7*w) - aMin*
    ↪ t5*sin_wt7/(DT5*w) - aMax*t3*sin_wt3/(DT3*w) + \
323             aMax*t2*sin_wt3/(DT3*w) + aMin*t4*sin_wt7/(DT5*w) - aMax*
    ↪ t0*sin_wt3/(DT1*w) + aMax*t1*sin_wt3/(DT1*w) + \
324             aMax*cos_wt1/(DT1*w**2) - aMax*cos_wt3/(DT3*w**2) + aMax*
    ↪ cos_wt2/(DT3*w**2) + aMin*cos_wt4/(DT5*w**2) - \
325             aMin*cos_wt5/(DT5*w**2) + aMin*cos_wt7/(DT7*w**2) - aMin*
    ↪ cos_wt6/(DT7*w**2) - aMax*cos_wt0/(DT1*w**2)
326
327     ImPart = -aMax*t0*cos_wt3/(DT1*w) - aMin*t5*cos_wt7/(DT5*w) + aMin*
    ↪ t7*cos_wt7/(DT7*w) - aMin*t6*cos_wt7/(DT7*w) + \
328             aMax*t1*cos_wt3/(DT1*w) - aMax*t3*cos_wt3/(DT3*w) + aMax*
    ↪ t2*cos_wt3/(DT3*w) + aMin*t4*cos_wt7/(DT5*w) - \
329             aMax*sin_wt1/(DT1*w**2) + aMax*sin_wt3/(DT3*w**2) - aMax*
    ↪ sin_wt2/(DT3*w**2) - aMin*sin_wt4/(DT5*w**2) + \
330             aMin*sin_wt6/(DT7*w**2) - aMin*sin_wt7/(DT7*w**2) + aMin*
    ↪ sin_wt5/(DT5*w**2) + aMax*sin_wt0/(DT1*w**2)
331
332     return RePart, ImPart
333
334
335
336 # ================================================================
    ↪ =================
337 # FFT (per confronto)
338 # ================================================================
    ↪ =================
339
340 def calc_trajectory_fft_np(t, acc):
341     T = t[1] - t[0]
342     Fs = 1.0 / T
343     L = len(acc)
344     n = int(2**np.ceil(np.log2(10 * L)))
345
346     Y = np.fft.fft(acc, n)
347     trasf_calc = np.abs(Y / L)
348     trasf = trasf_calc[:n//2 + 1]
349     if trasf.size > 2:
350         trasf[1:-1] *= 2.0
```

```
351
352     f = Fs * np.arange(0, n//2 + 1) / n
353     w = f * 2*np.pi
354     return w, trasf
355
356
357 def trapStd_amplitude_spectrum_np(w, DT_vec, s_tot, t0=0.0):
358
359     #Versione NumPy (vettoriale) per plottare A() della trapezia
    ↪ standard
360
361     DT1, DT2, DT3, DT4, DT5, DT6, DT7 = DT_vec
362
363     t1 = t0 + DT1
364     t2 = t1 + DT2
365     t3 = t2 + DT3
366     t4 = t3 + DT4
367     t5 = t4 + DT5
368     t6 = t5 + DT6
369     t7 = t6 + DT7
370
371     # aMax/aMin
372     c1 = (DT1*DT7**2 + 2*DT2*DT7**2 + DT3*DT7**2)/(2*DT7) + DT3*DT6/2 +
    ↪  DT2*DT6 + DT1*DT6/2 + (DT1*DT5**2 + 2*DT2*DT5**2 + DT3*DT5**2)
    ↪ /(2*DT5) + \
373          (DT3/2 + DT2 + DT1/2)*DT4 + (DT1*DT3**2 + 2*DT2*DT3**2 +
    ↪ (2.0/3.0)*DT3**3)/(2*DT3) + DT1*DT2/2 + DT2**2/2 + DT1**2/6
374     c2 = (-DT5*DT7**2 - 2*DT6*DT7**2 - (2.0/3.0)*DT7**3)/(2*DT7) - DT6
    ↪ **2/2 - DT5*DT6/2 - DT5**2/6
375     c3 =  DT3/2 + DT2 + DT1/2
376     c4 = -DT7/2 - DT6 - DT5/2
377     denom = (c1*c4 - c2*c3)
378     aMax =  c4 * s_tot / denom
379     aMin = -c3 * s_tot / denom
380
381     w = np.asarray(w, dtype=float)
382     sin_wt0 = np.sin(w*t0);   cos_wt0 = np.cos(w*t0)
383     sin_wt1 = np.sin(w*t1);   cos_wt1 = np.cos(w*t1)
384     sin_wt2 = np.sin(w*t2);   cos_wt2 = np.cos(w*t2)
385     sin_wt3 = np.sin(w*t3);   cos_wt3 = np.cos(w*t3)
386     sin_wt4 = np.sin(w*t4);   cos_wt4 = np.cos(w*t4)
387     sin_wt5 = np.sin(w*t5);   cos_wt5 = np.cos(w*t5)
388     sin_wt6 = np.sin(w*t6);   cos_wt6 = np.cos(w*t6)
389     sin_wt7 = np.sin(w*t7);   cos_wt7 = np.cos(w*t7)
390
391     with np.errstate(divide='ignore', invalid='ignore'):
392         RePart = -aMin*t6*sin_wt7/(DT7*w) + aMin*t7*sin_wt7/(DT7*w) -
    ↪ aMin*t5*sin_wt7/(DT5*w) - aMax*t3*sin_wt3/(DT3*w) + \
393                 aMax*t2*sin_wt3/(DT3*w) + aMin*t4*sin_wt7/(DT5*w) -
    ↪ aMax*t0*sin_wt3/(DT1*w) + aMax*t1*sin_wt3/(DT1*w) + \
```

```
                aMax*cos_wt1/(DT1*w**2) - aMax*cos_wt3/(DT3*w**2) + aMax*cos_wt2/(DT3*w**2) + aMin*cos_wt4/(DT5*w**2) - \
                aMin*cos_wt5/(DT5*w**2) + aMin*cos_wt7/(DT7*w**2) - aMin*cos_wt6/(DT7*w**2) - aMax*cos_wt0/(DT1*w**2)

        ImPart = -aMax*t0*cos_wt3/(DT1*w) - aMin*t5*cos_wt7/(DT5*w) + aMin*t7*cos_wt7/(DT7*w) - aMin*t6*cos_wt7/(DT7*w) + \
                aMax*t1*cos_wt3/(DT1*w) - aMax*t3*cos_wt3/(DT3*w) + aMax*t2*cos_wt3/(DT3*w) + aMin*t4*cos_wt7/(DT5*w) - \
                aMax*sin_wt1/(DT1*w**2) + aMax*sin_wt3/(DT3*w**2) - aMax*sin_wt2/(DT3*w**2) - aMin*sin_wt4/(DT5*w**2) + \
                aMin*sin_wt6/(DT7*w**2) - aMin*sin_wt7/(DT7*w**2) + aMin*sin_wt5/(DT5*w**2) + aMax*sin_wt0/(DT1*w**2)

        A = np.sqrt(RePart**2 + ImPart**2)

    # gestione ~0
    eps = 1e-6
    mask_small = np.abs(w) < eps
    if np.any(mask_small):
        A = np.asarray(A, dtype=float)
        A[mask_small] = 0.0
    return A


# ================================================================================
# OPTI (problema di ottimizzazione)
# ================================================================================

opti = ca.Opti()

# ============================================================
# Parametri sloshing
# ============================================================

R_mm = 48.5
h_mm = 82.0

R = R_mm / 1000.0
h = h_mm / 1000.0
rho = 1000.0
st = 0.073
mu = 1e-3
nu = mu / rho
g  = 9.81

xi = 1.84118
alpha_n = 0.58

```

```
437
438 # ============================================================
439 # Impostazioni moto
440 # ============================================================
441
442 s_tot_mm = 800.0
443 s_tot_m  = s_tot_mm / 1000.0
444
445 t_avanz = 1.0
446 t_rest  = 1.0
447 t_tot   = t_avanz + t_rest
448
449 # ============================================================
450 # Variabili decisionali:
451 # ============================================================
452
453 #Numero di campioni usati per costruire la griglia temporale
454 N_sim = 101
455
456 # durate grezze non negative
457 DT = opti.variable(7, 1)
458
459 opti.subject_to(ca.sum(DT) == t_avanz)
460
461 # durate minime (imposte sui DT effettivi)
462 opti.subject_to(DT[0] >= 0.05)
463 opti.subject_to(DT[1] >= 0.0)
464 opti.subject_to(DT[2] >= 0.05)
465 opti.subject_to(DT[3] >= 0.0)
466 opti.subject_to(DT[4] >= 0.05)
467 opti.subject_to(DT[5] >= 0.0)
468 opti.subject_to(DT[6] >= 0.05)
469
470 # Guess iniziale
471 w7 = np.array([0.14286, 0.14286, 0.14286, 0.14284, 0.14286, 0.14286,
    ↪ 0.14286])      #somma = 1
472
473 #Larghezza (in %) del range di frequenze che voglio considerare nell'
    ↪ intorno di wn
474 Delta_freq = 0.15
475
476 #Numero di frequenze campionate nella banda  % attorno a wn
477 N_band = 51          # dispari perch cos wn sta al centro
478
479 #Fattore con cui trasformo l'ampiezza spettrale in una soglia di
    ↪ tolleranza da imporre come vincolo.
480 # relativo all'initial guess scelto
481 #Se il problema  infeasible, aumentare tol_rel. Se invece si vuole pi
    ↪ notch, ridurre.
482 tol_rel = 0.2
483
```

```
#configurazione del risolutore numerico
opti.solver('ipopt', {
    "ipopt.print_level": 3,          #livello di stampa
    "ipopt.max_iter": 2000,          #numero massimo di iterazioni
})


# ======================================================================
# Parametri fisici calcolati per caratterizzare la dinamica dello sloshing
# ======================================================================

m_f = math.pi * R**2 * h * rho
ms  = (m_f * 2.0 * R * math.tanh(xi * h / R)) / (xi * h * (xi**2 - 1.0))
wn  = math.sqrt((g * xi / R) * math.tanh(xi * h / R))

Re1 = nu / math.sqrt(g * R**3)
zeta_1 = 0.79 * math.sqrt(Re1) * (1.0 + (0.318 / math.sinh(1.84 * h / R)) * (1.0 + (1.0 - h / R) / math.cosh(1.84 * h / R)))


def C_n_val(xi_val, h_val, R_val):
    return xi_val * math.tanh(xi_val * (h_val / R_val))

Cn_const = C_n_val(xi, h, R)

# conversione da xn ad altezza di sloshing eta (metri)
K_eta = (h * (xi**2) * ms) / m_f


# ============================================================
# Minimizzazione del jerk + vincolo su spettro a wn
# ============================================================
# Il jerk  costante nei tratti 1,3,5,7. Minimizzo l'integrale del jerk^2:
#        J_jerk =  j(t)^2 dt = J1^2*DT1 + J2^2*DT3 + J3^2*DT5 + J4^2*DT7
#
# Anchde qui uso |X(wn)|^2 = Re(wn)^2 + Im(wn)^2 per evitare la radice.
# ============================================================

coeff, t_switch = build_trap_coeffs_casadi(DT, s_tot_m)

# solo per stampa
wn_dm = ca.DM(wn)
Re_wn, Im_wn = trapStd_ReIm_at_omega_casadi(wn_dm, DT, s_tot_m, t0=0.0)
A2_wn = Re_wn**2 + Im_wn**2          # |X(wn)|^2
A_wn  = ca.sqrt(A2_wn)               # solo per stampa
```

```
528
529 # ------------------------------------------------------------
530 # Vincolo spettrale su una banda  % attorno a wn (con .map)
531 # ------------------------------------------------------------
532
533 #crea N_band punti equispaziati dentro quellintervallo
534 w_band = np.linspace((1.0 - Delta_freq) * wn, (1.0 + Delta_freq) * wn,
        ↪ N_band)
535
536 # Function CasADi: (w, DT) -> (Re, Im)
537 w_sym  = ca.MX.sym('w', 1, 1)       #la rendo simbolica
538 DT_sym = ca.MX.sym('DT', 7, 1)      #la rendo simbolica
539 Re_sym, Im_sym = trapStd_ReIm_at_omega_casadi(w_sym, DT_sym, s_tot_m,
        ↪ t0=0.0)
540 fReIm = ca.Function('fReIm', [w_sym, DT_sym], [Re_sym, Im_sym])
        ↪  #creo questa function casadi che, data una coppia (, DT),
        ↪ ritorna (Re, Im)
541
542 # Map su N_band frequenze
543 fReIm_map = fReIm.map(N_band, 'serial')         #  come un for che
        ↪ valuta fReIm per N_band volte, una per ciascun punto
544
545 # Prepara gli Input al map: w come (1 x N_band), DT replicato come (7 x
        ↪  N_band)
546 w_vec  = ca.DM(w_band).reshape((1, N_band))         #cos contiene tutte
        ↪  le frequenze della banda
547 DT_mat = ca.repmat(DT, 1, N_band)                   #prende DT (71) e
        ↪ lao copia su N_band colonne
548
549 Re_vec, Im_vec = fReIm_map(w_vec, DT_mat)           #anche qui valutati
        ↪  su tutti i punti
550 A2_vec = Re_vec**2 + Im_vec**2
551
552 # Vincolo: |X(wk)|^2 <= tol^2 per tutti i wk nella banda (il valore
        ↪ della tolleranza glie lo do sotto, in base al guess iniziale)
553 A_wn_tol_par = opti.parameter()
554 opti.subject_to(A2_vec <= A_wn_tol_par**2)
555
556 # jerk nei tratti 1,3,5,7
557 J1 = coeff[20]   # tratto 1
558 J2 = coeff[21]   # tratto 3
559 J3 = coeff[22]   # tratto 5
560 J4 = coeff[23]   # tratto 7
561
562 # funzione costo: integrale del jerk^2
563 J_jerk = (J1**2) * DT[0] + (J2**2) * DT[2] + (J3**2) * DT[4] + (J4**2)
        ↪ * DT[6]
564
565 opti.minimize(J_jerk)
566
567 # ============================================================
```

```
# Vincolo posizione finale a t_avanz
# ============================================================

p_fin, _, _, _ = trapStd_kinematics_casadi(t_avanz, coeff, t_switch,
    ↪ s_tot_m)
opti.subject_to(p_fin == s_tot_m)

# ============================================================
# Solver basato sul guess iniziale
# ============================================================

w7 = w7 / w7.sum()
DT_init = t_avanz * w7

opti.set_initial(DT, DT_init.reshape((7, 1)))

# ---- Imposto la tolleranza su |X(wn)| relativamente allo spettro
    ↪ ottenuto col guess iniziale
# Se il problema  infeasible, aumentare tol_rel (es. 0.10). Se si vuole
    ↪  pi notch, ridurre (0.02, ...).
Aref_wn = float(trapStd_amplitude_spectrum_np(np.array([wn]), DT_init.
    ↪ flatten(), s_tot_m, t0=0.0)[0])
A_wn_tol_val = tol_rel * Aref_wn
opti.set_value(A_wn_tol_par, A_wn_tol_val)

print(f"[tolleranza spettro] Aref(wn)={Aref_wn:.6e}, tol_rel={tol_rel
    ↪ :.3f} -> A_wn_tol={A_wn_tol_val:.6e}")

sol = opti.solve()

DT_opt = np.array(sol.value(DT)).flatten()
DT_opt_clean = np.where(np.abs(DT_opt) < 1e-8, 0.0, DT_opt)
# DT in percentuale (somma = 100) con 3 cifre significative
DT_pct = 100.0 * DT_opt_clean / DT_opt_clean.sum()

# arrotondo "a display" a 3 cifre significative e forzo la somma esatta
    ↪  a 100
DT_pct_disp = np.array([float(f"{p:.3g}") for p in DT_pct])
DT_pct_disp[-1] = 100.0 - DT_pct_disp[:-1].sum()

np.save("DT_opt.npy", DT_opt_clean)


A_wn_num = float(sol.value(A_wn))
J_jerk_num = float(sol.value(J_jerk))

J1_num = float(sol.value(J1))
J2_num = float(sol.value(J2))
J3_num = float(sol.value(J3))
J4_num = float(sol.value(J4))
j_peak_num = max(abs(J1_num), abs(J2_num), abs(J3_num), abs(J4_num))
```

```
613
614
615 # ============================================================
616 # Post-processing sloshing: integrazione RK4 del modello non lineare
617 # (NON entra nell'ottimizzazione: serve solo per plot e stampa)
618 # ============================================================
619
620 # griglia temporale per simulazione sloshing
621 _dt_sim = t_tot / (N_sim - 1)
622 t_grid_sim = np.linspace(0.0, t_tot, N_sim)
623 idx_moto_end = int(np.searchsorted(t_grid_sim, t_avanz, side="right"))
624
625 # Accelerazione a(t) come Function + map (riuso della stessa legge
    ↪ trapezia)
626 t_sym_sim  = ca.MX.sym('t')
627 DT_sym_sim = ca.MX.sym('DT', 7)
628 coeff_sym_sim, t_sw_sym_sim = build_trap_coeffs_casadi(DT_sym_sim,
    ↪ s_tot_m)
629 _, _, a_trap_sym_sim, _ = trapStd_kinematics_casadi(t_sym_sim,
    ↪ coeff_sym_sim, t_sw_sym_sim, s_tot_m)
630 a_fun_sim = ca.Function('a_fun_sim', [t_sym_sim, DT_sym_sim], [
    ↪ a_trap_sym_sim])
631
632 # Jerk j(t) come Function (per plot)
633 _, _, _, j_trap_sym_sim = trapStd_kinematics_casadi(t_sym_sim,
    ↪ coeff_sym_sim, t_sw_sym_sim, s_tot_m)
634 j_fun_sim = ca.Function('j_fun_sim', [t_sym_sim, DT_sym_sim], [
    ↪ j_trap_sym_sim])
635
636 # a(t_k) sui nodi
637 t_eval_sim = ca.DM(t_grid_sim).reshape((1, N_sim))
638 DT_rep_sim_full = ca.repmat(ca.DM(DT_opt_clean).reshape((7, 1)), 1,
    ↪ N_sim)
639 a_sol_sim = np.array(a_fun_sim.map(N_sim)(t_eval_sim, DT_rep_sim_full))
    ↪ .ravel()
640
641 # j(t_k) sui nodi
642 j_sol_sim = np.array(j_fun_sim.map(N_sim)(t_eval_sim, DT_rep_sim_full))
    ↪ .ravel()
643
644 # a(t_k) sui nodi sinistri per integrare su N_sim-1 step
645 # uso a(t_k) per lo step [t_k, t_{k+1}])
646 t_left_sim = ca.DM(t_grid_sim[:-1]).reshape((1, N_sim - 1))
647 DT_rep_left = ca.repmat(ca.DM(DT_opt_clean).reshape((7, 1)), 1, N_sim -
    ↪  1)
648 a_left = np.array(a_fun_sim.map(N_sim - 1)(t_left_sim, DT_rep_left)).
    ↪ ravel()
649
650 a_grid = ca.DM(a_left).reshape((1, N_sim - 1))
651 dt_grid = ca.DM.ones(1, N_sim - 1) * _dt_sim
652
```

```
# ---- dinamica non lineare + RK4
x_sym  = ca.MX.sym('x', 2)     # [xn, xnd]
a_in   = ca.MX.sym('a')        # ingresso: accelerazione a(t)
dt_sym = ca.MX.sym('dt')

xn_sym  = x_sym[0]
xnd_sym = x_sym[1]

Den = 1.0 + (Cn_const**2) * (xn_sym**2)
term1 = 2.0 * wn * zeta_1 * (xnd_sym + (Cn_const**2) * xnd_sym * (xn_sym**2))
term2 = (Cn_const**2) * xn_sym * (xnd_sym**2)
term3 = (wn**2) * xn_sym * (1.0 + alpha_n * (xn_sym**2))
term4 = a_in / R

Num  = -(term1 + term2 + term3 + term4)
xndd = Num / Den

xdot = ca.vertcat(xnd_sym, xndd)
f = ca.Function('f', [x_sym, a_in], [xdot])

k1 = f(x_sym, a_in)
k2 = f(x_sym + 0.5 * dt_sym * k1, a_in)
k3 = f(x_sym + 0.5 * dt_sym * k2, a_in)
k4 = f(x_sym + 1.0 * dt_sym * k3, a_in)

x_next = x_sym + (dt_sym / 6.0) * (k1 + 2.0 * k2 + 2.0 * k3 + k4)
RK4_step = ca.Function('RK4_step', [x_sym, a_in, dt_sym], [x_next])
RK4_accum = RK4_step.mapaccum(N_sim - 1)

x0 = ca.DM([0.0, 0.0])
x_acc = RK4_accum(x0, a_grid, dt_grid)          # 2 x (N_sim-1)
X = ca.horzcat(x0, x_acc)                        # 2 x N_sim

xn_grid_num = np.array(X[0, :]).ravel()          # (N_sim,)
eta_grid_m = K_eta * xn_grid_num                 # (N_sim,) metri
eta_grid_mm = 1000.0 * eta_grid_m                # (N_sim,) mm

eta_moto_mm = eta_grid_mm[:idx_moto_end]
eta_rest_mm = eta_grid_mm[idx_moto_end:]

eta_peak_opt_mm = float(np.max(np.abs(eta_moto_mm)))
eta_rest_max_mm = float(np.max(np.abs(eta_rest_mm)))

# ============================================================
# Stampe
# ============================================================

print("DT_opt = [" + " , ".join(f"{x:.6f}" for x in DT_opt_clean) + "]")
print("DT_opt_tot = ", DT_opt_clean.sum())
```

```
print("Space_tot  = ", float(sol.value(p_fin)))
print(f"wn [rad/s] = {wn:.6f}")
print(f"|X(wn)| (analitico) = {A_wn_num:.6e}   (tolleranza = {A_wn_tol_val:.6e})")
print(f"j_peak = {j_peak_num:.6e}  [m/s^3]")
print(f"J_jerk = {J_jerk_num:.6e}")
print("DT_opt [%] = [" + ", ".join(f"{p:.3g}" for p in DT_pct_disp) + " ]")
print(f"eta_peak_opt [mm] = {eta_peak_opt_mm:.6f}")
print(f"eta_rest_max [mm] = {eta_rest_max_mm:.6f}")


# ======================================================================
# Plot
# ======================================================================

# ----------------------------
# FIGURA 1: a(t) + eta(t)
# ----------------------------
fig1, (ax1, ax3, ax2) = plt.subplots(3, 1, sharex=True, figsize=(11, 7))

ax1.plot(t_grid_sim, a_sol_sim, linewidth=1.5)
ax1.axvline(t_avanz, linestyle="--", linewidth=1.2)
ax1.grid(True, which='both', linestyle="--", linewidth=0.6, alpha=0.5)
ax1.set_ylabel("a(t) [m/s]")
ax1.set_title("Legge di accelerazione")

ax3.plot(t_grid_sim, j_sol_sim, linewidth=1.5)
ax3.axvline(t_avanz, linestyle="--", linewidth=1.2)
ax3.grid(True, which='both', linestyle="--", linewidth=0.6, alpha=0.5)
ax3.set_ylabel("j(t) [m/s]")
ax3.set_title("Jerk j(t) minimizzato")

ax2.plot(t_grid_sim, eta_grid_mm, linewidth=1.2)
ax2.axvline(t_avanz, linestyle="--", linewidth=1.2)
ax2.grid(True, which='both', linestyle="--", linewidth=0.6, alpha=0.5)
ax2.set_xlabel("t [s]")
ax2.set_ylabel("eta(t) [mm]")
ax2.set_title("Altezza di sloshing")

# (opzionale) linee di riferimento nella fase di riposo
# Qui NON sono vincoli dell'ottimizzazione: sono solo un riferimento grafico.
#eta_rest_limit_mm = 1
#ax2.hlines(+eta_rest_limit_mm, xmin=t_avanz, xmax=t_tot, colors="red", linestyles="--", linewidth=1.6)
#ax2.hlines(-eta_rest_limit_mm, xmin=t_avanz, xmax=t_tot, colors="red", linestyles="--", linewidth=1.6)
```

```
745
746 plt.tight_layout()
747
748
749 # ----------------------------
750 # FIGURA 2: spettro (analitico) + FFT per confronto
751 # ----------------------------
752 # Ricostruisco a(t) su una griglia fitta per fare una FFT pi pulita
753 N_plot = 2001
754 t_grid_plot = np.linspace(0.0, t_tot, N_plot)
755
756 t_eval = ca.DM(t_grid_plot).reshape((1, N_plot))
757 DT_rep = ca.repmat(ca.DM(DT_opt_clean).reshape((7, 1)), 1, N_plot)
758 a_sol_plot = np.array(a_fun_sim.map(N_plot)(t_eval, DT_rep)).ravel()
759
760 # Spettro analitico su un asse di frequenze
761 w_plot = np.linspace(0.0, 3.0 * wn, 2500)
762 A_plot = trapStd_amplitude_spectrum_np(w_plot, DT_opt_clean, s_tot_m,
    ↪ t0=0.0)
763
764 # per confrontarlo con la FFT (che  single-sided) uso lo stesso scaling
    ↪  gi adottato
765 A_plot_fft = (2.0 / t_avanz) * A_plot
766
767 # FFT numerica dell'accelerazione (solo tratto di moto)
768 mask_moto = t_grid_plot <= t_avanz + 1e-12
769 w_fft, A_fft = calc_trajectory_fft_np(t_grid_plot[mask_moto],
    ↪ a_sol_plot[mask_moto])
770
771 fig2, ax = plt.subplots(1, 1, figsize=(11, 4.5))
772 ax.plot(w_plot, A_plot_fft, linewidth=1.8, label="Analitico |A(j)|")
773 #ax.plot(w_fft, A_fft, linestyle="--", linewidth=1.2, label="FFT (check
    ↪  numerico)")
774
775 w_low  = (1.0 - Delta_freq) * wn
776 w_high = (1.0 + Delta_freq) * wn
777
778 ax.axvline(w_low,  linestyle="--", linewidth=1.2, alpha=0.9, color="C0"
    ↪ , label=r"$(1\pm\Delta)\,\omega_n$")
779 ax.axvline(wn,     linestyle="--", linewidth=1.5, c='r', alpha=0.9,
    ↪ color="C0", label=r"$\omega_n$")
780 ax.axvline(w_high, linestyle="--", linewidth=1.2, alpha=0.9, color="C0"
    ↪ )
781 A_tol_plot = (2.0 / t_avanz) * A_wn_tol_val
782 ax.axhline(A_tol_plot, linestyle="--", linewidth=1.5, color="red",
783            label=rf"Tolleranza con (tol_rel={tol_rel:.2f})")
784
785 ax.set_xlim(0.0, 3.0 * wn)
786 ax.grid(True, which='both', linestyle="--", linewidth=0.6, alpha=0.5)
787 ax.set_xlabel(r"$\omega$ [rad/s]")
788 ax.set_ylabel(r"$|A(j\omega)|$")
```

```
#ax.set_title("Spettro di ampiezza dell'accelerazione (focus su n)")
ax.legend()

plt.tight_layout()
plt.show()
```

Listing C.1: Script di ottimizzazione: Opt_spettro_minjerk_robusto

Appendice D

Dichiarazione sull'uso di strumenti di IA generativa

Durante la preparazione di questa tesi è stato utilizzato uno strumento di intelligenza artificiale generativa (*ChatGPT*, OpenAI), modello *GPT-5.2 Thinking*, nel periodo *[Ottobre 2025 – Marzo 2026]*.

Lo strumento è stato impiegato come supporto alla stesura e revisione linguistica (coerenza logica e correttezza grammaticale) e per ottenere indicazioni operative sul posizionamento e l'impaginazione di elementi grafici e di codice (in particolare nei Capitoli 3 e 4), senza sostituire l'attività di ricerca, analisi e sintesi svolta dall'autore.

Tutti i contenuti tecnici, i risultati sperimentali e numerici e le conclusioni restano di responsabilità dell'autore, che ha verificato criticamente e validato i contenuti prodotti dallo strumento prima dell'inserimento nell'elaborato.

Elenco delle figure

Bibliografia

[1] R. Ibrahim, *Liquid sloshing dynamics: Theory and Applications*. Cambridge University Press, 2005. doi:10.1017/CBO9780511536656.

[2] H. N. Abramson, "The dynamic behavior of liquids in moving containers, with applications to space vehicle technology," Tech. Rep. NASA-SP-106, National Aeronautics and Space Administration, 1966.

[3] M. I. Panevsky, B. J. Pataky, and P. T. Than, "Flight software, rigid body, and computational fluid dynamics closed loop simulation," in *CEAS Conference 2007*, 2007. Paper CEAS-2007-157. Publisher/affiliation: The Aerospace Corporation. Available via Prof. Scholz (HAW Hamburg) proceedings archive.

[4] L. Guagliumi, A. Berti, E. Monti, and M. Carricato, "Antisloshing trajectories for high-acceleration motions in automatic machines," *Journal of Dynamic Systems, Measurement, and Control*, vol. 144, no. 7, p. 071006, 2022. doi:10.1115/1.4054224.

[5] R. Di Leva, M. Carricato, H. Gattringer, and A. Müller, "Time-optimal trajectory planning for anti-sloshing 2-dimensional motions of an industrial robot," in *2021 20th International Conference on Advanced Robotics (ICAR)*, pp. 32–37, 2021. doi:10.1109/ICAR53236.2021.9659383.

[6] M. Schörgenhumer and A. Eitzlmayr, "Modeling of liquid sloshing with application in robotics and automation," *IFAC-PapersOnLine*, vol. 52, no. 15, pp. 253–258, 2019. doi:10.1016/j.ifacol.2019.11.683.

[7] F. Brasina, L. Guagliumi, R. Di Leva, and M. Carricato, "Anti-sloshing motion laws for one-dimensional piecewise trajectories," in *Proceedings of EuCoMeS 2024*, vol. 165 of *Mechanisms and Machine Science*, pp. 153–162, Springer Nature Switzerland AG, 2024. doi:10.1007/978-3-031-67295-8_18.

[8] L. Guagliumi, A. Berti, E. Monti, and M. Carricato, "A simple model-based method for sloshing estimation in liquid transfer in automatic machines," *IEEE Access*, vol. 9, pp. 129347–129357, 2021. doi:10.1109/ACCESS.2021.3113956.

[9] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "Casadi: a software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, pp. 1–36, 2019. doi:10.1007/s12532-018-0139-4.

[10] L. Guagliumi, "Metodi di pianificazione di traiettorie anti-sloshing," Master's thesis, 2019.

[11] Y. A. Çengel and J. M. Cimbala, *Fluid Mechanics: Fundamentals and Applications*. New York, NY: McGraw-Hill, 2005.

[12] F. T. Dodge, *The New "Dynamic Behavior of Liquids in Moving Containers"*. San Antonio, TX: Southwest Research Institute, 2000.

[13] H. F. Bauer, "Nonlinear mechanical model for the description of propellant sloshing," *AIAA Journal*, vol. 4, pp. 1662–1668, Sept. 1966. doi:10.2514/3.3752.

[14] A. Ferrari, R. Di Leva, S. Soprani, L. Biagiotti, G. Palli, and M. Carricato, "Time-optimal anti-sloshing trajectory planning for multiple liquid-filled containers subject to SCARA motion," *IEEE Robotics and Automation Letters*, vol. 11, no. 2, pp. 1762–1769, 2026. doi:10.1109/LRA.2025.3643281.

[15] R. Di Leva, S. Soprani, G. Palli, L. Biagiotti, and M. Carricato, "Sloshing height estimation for liquid-filled containers under four-dimensional motions including spatial translation and rotation about a fixed direction: Modeling and experimental validation," *Nonlinear Dynamics*, 2026. to appear.

[16] L. Biagiotti and C. Melchiorri, *Trajectory Planning for Automatic Machines and Robots*. Springer-Verlag Berlin Heidelberg, 2008. doi:10.1007/978-3-540-85629-0.

[17] J. Nocedal and S. J. Wright, *Numerical Optimization*. New York, NY: Springer, 2 ed., 2006.

[18] N. C. Singer and W. P. Seering, "Preshaping command inputs to reduce system vibration," *Journal of Dynamic Systems, Measurement, and Control*, vol. 112, no. 1, pp. 76–82, 1990. doi:10.1115/1.2894142.

[19] R. Di Leva, M. Carricato, H. Gattringer, and A. Müller, "Sloshing dynamics estimation for liquid-filled containers performing 3-dimensional motions: modeling and experimental validation," *Multibody System Dynamics*, vol. 56, pp. 153–171, 2022.