UNIVERSITÀ DEGLI STUDI DI ROMA "TOR VERGATA"

MACROAREA DI INGEGNERIA

CORSO DI STUDIO IN

INGEGNERIA MECCANICA

TESI DI LAUREA IN

ELEMENTI COSTRUTTIVI DELLE MACCHINE

TITOLO

Articolazione di THUMS mediante modelli di ordine ridotto

Relatore:
Prof.
Marco Evangelos Biancolini

Correlatore:
Ing.
Emanuele Di Meo

Laureando:
Samuele Stazi
Matricola: 0294945

Anno Accademico 2025/2026

APPROVAZIONE DELLA TESI

TITOLO:	Articolazione di THUMS mediante modelli di ordine ridotto
CANDIDATO:	Samuele Stazi
DATA PRESENTAZIONE:	16 Luglio 2026
RELATORE:	Prof. Marco Evangelos, Biancolini
CORRELATORE:	Ing. Emanuele Di Meo

SOMMARIO

Articolazione di THUMS mediante modelli di ordine ridotto

Samuele Stazi

Il presente lavoro di tesi si pone l'obiettivo di definire e validare una metodologia efficiente per l'articolazione del modello THUMS (versione AM50 v7), abbattendo i tempi di calcolo attraverso l'implementazione di Modelli di Ordine Ridotto (ROM). L'attività di simulazione, condotta in ambiente LS-DYNA, si è concentrata in modo particolare sulla complessa cinematica del distretto ginocchio-caviglia-piede. Per guidare il corretto posizionamento del modello, è stata ideata una strategia numerica basata sull'utilizzo di elementi *cable*, opportunamente configurati per imporre i vincoli cinematici desiderati. Sono state condotte diverse simulazioni volte al controllo della cinematica del distretto analizzato e alla raccolta dei dati necessari per le fasi successive. Per la costruzione del modello ridotto, le matrici di snapshot ottenute dai risultati FEM sono state elaborate matematicamente applicando il metodo della Proper Orthogonal Decomposition (POD). I risultati ottenuti dimostrano che l'applicazione della tecnica POD per la riduzione dell'ordine del modello consente un drastico abbattimento degli oneri computazionali richiesti per il riposizionamento dell'occupante, pur mantenendo un'elevata fedeltà strutturale e biomeccanica. Tale approccio si rivela uno strumento promettente per ottimizzare le fasi preliminari di setup nei crash test virtuali.

Indice

Pagina

Capitoli

Elenco delle tabelle

Tabella Pagina

Elenco delle figure

CAPITOLO 1

INTRODUZIONE

L'evoluzione dei modelli umani digitali, noti come HBM[1], ha rappresentato negli ultimi anni uno dei progressi più significativi nel campo della biomeccanica computazionale applicata alla sicurezza veicolare. Tra le architetture più avanzate e ampiamente validate a livello internazionale, il modello THUMS[2], sviluppato dai Toyota Central R&D Labs, costituisce un riferimento imprescindibile per lo studio del comportamento dinamico del corpo umano in condizioni di impatto. La versione AM50 v7, in particolare, riproducendo con elevato dettaglio anatomico e fisico le caratteristiche di un maschio adulto di statura e massa medie, si è affermata come strumento di elezione per l'analisi della crashworthiness[3], la progettazione di sistemi di protezione avanzati e la valutazione predittiva del rischio lesivo. Nonostante l'accuratezza dei risultati forniti dai modelli HBM, il loro impiego sistematico è ancora oggi limitato da un onere computazionale proibitivo, strettamente correlato all'elevato numero di elementi finiti, alla complessità dei materiali costitutivi e alla risoluzione di numerosi contatti non lineari. Tale criticità emerge con particolare evidenza durante la fase di pre-processing e posizionamento del modello: configurare l'HBM in posture differenti rispetto alla condizione di default richiede procedure di articolazione che, se affrontate con i solutori espliciti standard, risultano estremamente onerose e soggette a frequenti instabilità numeriche, quali il collasso degli elementi o l'insorgere di errori di volume negativo. Il presente lavoro di tesi si inserisce in questo contesto

[1]HBM:Human Body Models: modelli computazionali tridimensionali ad alta fedeltà che rappresentano l'anatomia umana per la simulazione di dinamiche d'impatto.

[2]THUMS:Total Human Model for Safety: modello FEA (Finite Element Analysis) avanzato sviluppato per simulare le lesioni umane in crash test virtuali.

[3]crashworthiness: Capacità di una struttura di proteggere gli occupanti durante un impatto, riducendo l'energia trasmessa e mitigando il rischio di lesioni.

tecnologico con l'obiettivo di definire una metodologia innovativa ed efficiente per l'articolazione del modello THUMS AM50, focalizzandosi in modo specifico sul distretto caviglia-piede e sulle ossa metatarsali. La strategia proposta prevede l'utilizzo di elementi [4] opportunamente configurati per imporre i vincoli cinematici necessari al corretto riposizionamento, garantendo una transizione posturale controllata e stabile. Al fine di superare i limiti computazionali intrinseci alla risoluzione del modello completo, è stata implementata una tecnica di MOR[5] basata sulla POD[6]. Tale approccio matematico permette di estrarre le dinamiche principali a partire da una base di configurazioni numeriche note, consentendo la costruzione di un modello ridotto capace di descrivere con elevata fedeltà la risposta meccanica del distretto anatomico analizzato, abbattendo drasticamente i tempi di calcolo. Il percorso di ricerca si articola attraverso lo sviluppo di un framework integrato in ambiente LS-DYNA[7], dove le fasi di generazione degli snapshot, estrazione della base modale POD e validazione del modello ridotto sono state condotte in modo sequenziale. L'efficacia della metodologia è stata valutata mediante un confronto sistematico tra la risposta del modello completo e quella del sistema ridotto, analizzando l'accuratezza cinematica e la robustezza numerica. Il contributo principale della tesi risiede pertanto nell'aver dimostrato come una riduzione selettiva, basata su criteri di dominanza energetica, possa costituire una soluzione vincente per migliorare l'efficienza dei flussi di lavoro in ambito biomeccanico, fornendo uno strumento versatile per il posizionamento rapido

[4]cable: Elementi strutturali monodimensionali in grado di trasmettere esclusivamente sforzi di trazione, impiegati per imporre vincoli cinematici e controllare il posizionamento del modello.

[5]MOR: Model Order Reduction: procedura matematica per ridurre la dimensione di un sistema dinamico complesso preservandone la dinamica dominante.

[6]POD: Proper Orthogonal Decomposition: tecnica statistica che estrae i "modi" o pattern spaziali che contengono la maggior parte dell'energia del sistema da un insieme di simulazioni di riferimento (snapshot).

[7]LS-DYNA: Software di simulazione a elementi finiti (FEA) esplicito, standard industriale per l'analisi di fenomeni altamente non lineari e transienti.

dell'occupante nei test di impatto virtuali e aprendo la strada a studi di sensibilità parametrica precedentemente non praticabili.

CAPITOLO 2

THUMS:TOTAL HUMAN BODY FOR SAFETY

In questo capitolo vengono presentati i principali riferimenti teorici e metodologici relativi ai modelli umani digitali, al modello THUMS AM50.

2.1 Modelli umani digitali

Lo sviluppo dei modelli umani digitali (Human Body Models, HBM) rappresenta il risultato di un'evoluzione progressiva che ha interessato la biomeccanica computazionale negli ultimi decenni. Inizialmente, la valutazione della sicurezza passiva si basava esclusivamente sui manichini antropomorfi fisici (Anthropomorphic Test Devices, ATD[1]), strumenti capaci di fornire misure cinematiche e dinamiche ma intrinsecamente limitati nella possibilità di analizzare grandezze interne come stress, strain o pressioni tissutali. L'avvento di tecniche avanzate di imagin medico[2], unito alla crescente disponibilità di dati sperimentali su tessuti e strutture anatomiche, ha reso possibile la costruzione di modelli numerici sempre più accurati, in grado di riprodurre la complessità del corpo umano con un livello di dettaglio prima irraggiungibile. La transizione dagli ATD agli HBM ha comportato un cambiamento radicale nell'approccio alla simulazione degli impatti. I modelli digitali hanno introdotto la possibilità di rappresentare fedelmente la geometria scheletrica, la distribuzione dei tessuti molli, la configurazione articolare e il comportamento non lineare dei materiali biologici. Tale evoluzione ha permesso di superare i limiti dei manichini fisici,

[1] ATD: dispositivo fisico strumentato progettato per simulare la risposta cinematica e dinamica del corpo umano in caso di impatto.

[2] imagin medico: il ponte tecnologico che permette la trasposizione del paziente reale in un ambiente virtuale ad alta fedeltà, fornendo il set di dati geometrici necessari per la discretizzazione del modello agli elementi finiti.

consentendo analisi interne, valutazioni localizzate del rischio di lesione e simulazioni personalizzate su differenti morfologie corporee.

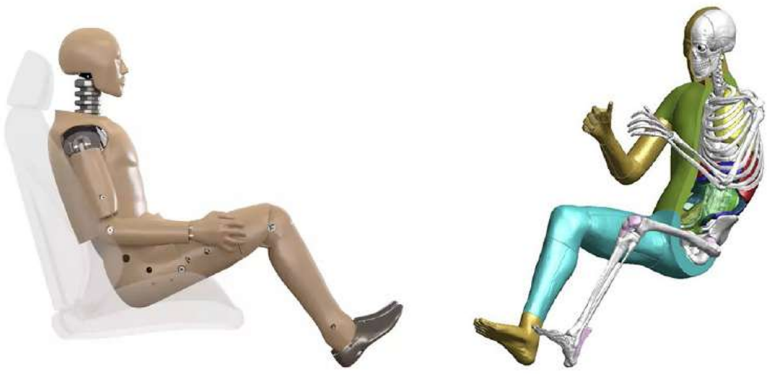


Figura 2.1: Dai Manichini fisici (ATD) ai modelli umani digitali (HBM).

Tra le famiglie di HBM più diffuse, la serie THUMS rappresenta uno dei riferimenti principali nel settore della sicurezza automobilistica. La sua evoluzione, articolata in sette versioni successive, riflette un processo continuo di raffinamento sia dal punto di vista anatomico sia da quello numerico. La prima versione (v1[3]), completata nel 2000, rappresentava un modello in cui le principali strutture ossee e i legamenti erano già descritti con buona accuratezza, mentre il cervello e gli organi interni erano ancora schematizzati come volumi solidi privi di dettagli interni. La cinematica articolare veniva riprodotta attraverso il moto relativo tra le ossa, senza l'impiego di elementi cinematici dedicati. Il modello comprendeva circa 80.000 elementi con una dimensione media della mesh[4] pari a 15 mm e risultava idoneo allo studio delle fratture ossee e delle rotture legamentose in scenari di impatto automobilistico.

[3]v1: denominazione utilizzata per indicare la versione del modello,in questo caso la prima.

[4]mesh:Griglia di elementi geometrici (solitamente tetraedri o esaedri) che discretizza un dominio continuo, permettendo la risoluzione numerica delle equazioni del moto in ambito FEA

Nel 2004 fu sviluppata la seconda versione, caratterizzata da una revisione delle strutture ossee del volto al fine di migliorare la capacità del modello di riprodurre i meccanismi di frattura cranio-facciale.

Un ulteriore avanzamento si ebbe nel 2008 con la terza versione, che introdusse per la prima volta un modello dettagliato del cervello, consentendo analisi più accurate delle lesioni cerebrali traumatiche. In questa configurazione il numero totale di elementi raggiungeva circa 130.000.

La quarta versione, completata nel 2010, integrò una modellazione più realistica degli organi interni, permettendo di simulare in modo più affidabile le lesioni addominali e toraciche. Il modello raggiunse in questa fase una complessità di circa due milioni di elementi.

Nel 2015 fu rilasciata la quinta versione, che incorporava la spalla, le colonne toracica e lombare, gli organi interni e l'intera muscolatura del corpo, basandosi sulla struttura della Versione 3. Il numero di elementi risultava in questo caso pari a circa 280.000.

L'ultima evoluzione, la sesta versione, completata nel 2019, integrò la muscolatura completa del corpo all'interno dell'architettura della Versione 4, raggiungendo nuovamente una complessità dell'ordine di due milioni di elementi. Le Versioni 5 e 6 sono disponibili esclusivamente nella postura seduta, configurazione pensata per rappresentare un occupante di veicolo. Oltre ai modelli commerciali, sono state sviluppate anche varianti destinate esclusivamente alla ricerca, tra cui un modello dell'intero corpo con muscolatura rappresentata tramite elementi solidi e un modello dedicato a un'occupante in gravidanza di piccola taglia. Tali versioni non sono mai state rese disponibili commercialmente.

La v7, utilizzata nel presente lavoro, integra una revisione completa dei tessuti molli, una migliore definizione delle articolazioni e una calibrazione aggiornata delle

proprietà meccaniche, rendendo il modello particolarmente adatto a studi di crashworthiness avanzati. La Figura 2.2 mostra una panoramica comparativa dell'evoluzione delle versioni THUMS dalla v1 alla v7.

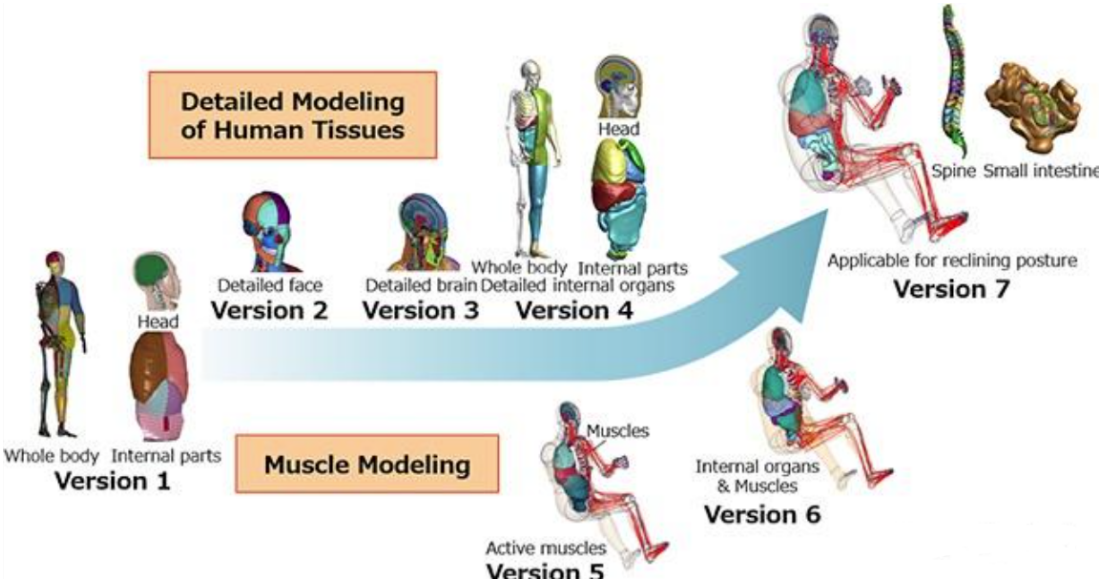


Figura 2.2: Evoluzione delle versioni del modello THUMS, dalla v1 alla v7.

Un ulteriore elemento di rilevanza nella modellazione digitale del corpo umano riguarda la disponibilità di versioni antropometriche riferite a differenti percentili della popolazione. I modelli del 50°(AM50[5]) e del 95°(AM95[6]) percentile rappresentano due casi particolarmente significativi per la valutazione della sicurezza dei sistemi di protezione. Il modello del 50° percentile riproduce un adulto maschio di corporatura media ed è generalmente considerato il riferimento standard nelle simulazioni di im-

[5]AM50: Average Male 50th percentile: modello che rappresenta un maschio adulto di statura e massa medie (media statistica della popolazione), utilizzato come standard di riferimento principale nei crash test. Esso è ampiamente utilizzato come standard di progettazione nei test di sicurezza passiva poiché riflette le caratteristiche antropometriche centrali della popolazione maschile adulta. Sebbene non rappresenti l'intero spettro di utenza, la sua geometria bilanciata e la distribuzione delle masse segmentali lo rendono il riferimento ideale per calibrare le prestazioni dei sistemi di ritenuta (cinture di sicurezza, airbag) in condizioni di impatto nominali.

[6]Average Male 95th percentile: modello che rappresenta un individuo di corporatura superiore alla media (95% della popolazione è più piccola), utilizzato per verificare il funzionamento dei sistemi di sicurezza in condizioni limite di ingombro. rappresenta un individuo di corporatura significativamente più robusta. L'impiego del modello AM95 è essenziale per valutare scenari di "worst-case" riguardanti, ad esempio, lo spazio disponibile nell'abitacolo (clearance) o l'efficacia del posizionamento della cintura di sicurezza su soggetti di grandi dimensioni

patto. Esso presenta proporzioni corporee equilibrate e una distribuzione delle masse segmentali rappresentativa della popolazione maschile adulta.

Il modello del 95° percentile, al contrario, è caratterizzato da una corporatura più robusta, con un incremento significativo della massa corporea complessiva e delle dimensioni segmentali. Le differenze non si limitano all'altezza e al peso, ma riguardano anche la geometria scheletrica, la distribuzione dei tessuti molli e la risposta dinamica durante un impatto. L'aumento della massa segmentale comporta variazioni nelle accelerazioni locali, nelle forze di contatto e nelle deformazioni interne, influenzando in modo sostanziale la cinematica dell'impatto e il rischio di lesioni.

Il confronto tra i modelli del 50° e del 95° percentile risulta quindi essenziale per comprendere come la variabilità antropometrica influenzi la risposta biomeccanica e per garantire che i sistemi di sicurezza siano efficaci su un'ampia gamma di utenti. La Figura 2.3 mostra un confronto visivo tra i vari modelli (AM30,AM50,AM95), evidenziando le principali differenze antropometriche.

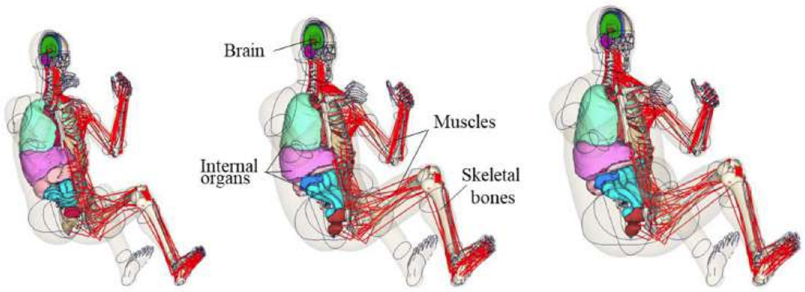


Figura 2.3: Panoramica Generale delle versioni 7.

2.2 Il modello THUMS AM50 V7: caratteristiche, versioni e validazioni

L'obiettivo primario della metodologia di modellazione risiede nella rappresentazione geometrica ad alta fedeltà del corpo umano, minimizzando le semplificazioni morfologiche per garantire la biofedeltà della risposta strutturale del modello numerico. Di

seguito vengono dettagliate le fasi di acquisizione, segmentazione e discretizzazione geometrica.

2.2.1 Acquisizione dei Dati e Definizione dell'Antropometria

La ricostruzione della struttura interna del tronco è stata eseguita a partire da un dataset di tomografie computerizzate (TC[7]) ad alta risoluzione, originariamente acquisite per scopi clinici e fornite dall'Università del Michigan, previo ottenimento delle necessarie autorizzazioni da parte dell'omonimo *Institutional Review Board*. Per la configurazione del modello **AM50**, rappresentativo del percentile medio della popolazione maschile adulta, è stato selezionato un soggetto con le seguenti caratteristiche antropometriche:

- **Età:** 39 anni
- **Altezza:** 173 cm
- **Peso:** 77.3 kg
- **Indice di Massa Corporea (BMI):** 25.8 kg/m^2

Il processo di scansione TC è stato condotto adottando un passo (pitch[8]) differenziato in base al distretto anatomico analizzato, al fine di ottimizzare la risoluzione spaziale.

[7]Tomografia Computerizzata: metodica diagnostica per immagini che utilizza radiazioni X per acquisire sezioni trasversali del corpo umano

[8]pitch: parametro definito come il rapporto tra la distanza di avanzamento del lettino in una rotazione del tubo radiogeno e la larghezza del fascio di radiazioni. Valori di pitch differenti influenzano direttamente la risoluzione spaziale del dataset TC e la dose radiante somministrata

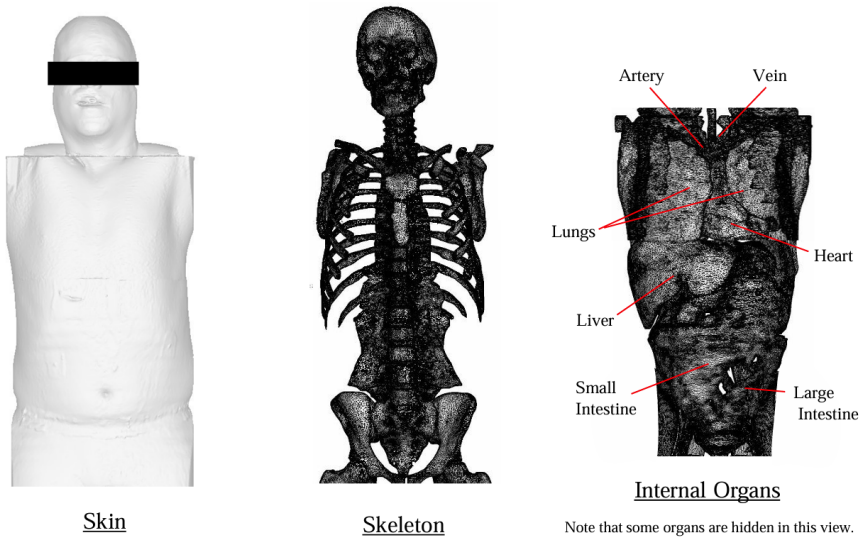


Figura 2.4: maasse ossee, masse muscolari, masse adipose e masse organiche.

La definizione geometrica dell'intero apparato muscolare è stata successivamente integrata e validata mediante il confronto con la letteratura anatomica di riferimento. Le geometrie dei muscoli sono determinate a partire dalla letteratura dedicata e infine le geometrie di arti e testa vengono ottenute raffinando le stesse impiegate in THUMS V3.

2.2.2 Segmentazione e Generazione del Modello Geometrico

I dati volumetrici grezzi derivanti dalla TC sono stati elaborati mediante tecniche di segmentazione dell'immagine[9]. Applicando algoritmi di mascheratura e calibrazione dei valori di soglia (*thresholding*) basati sulla densità radiografica dei tessuti, è stato possibile isolare i singoli distretti anatomici. Le geometrie così ottenute sono state convertite in superfici poligonali in formato STL[10]. A titolo esemplificativo di quanto

[9]segmentazione dell'immagine: Processo di elaborazione digitale volto a suddividere un'immagine in regioni distinte, basato sulla distinzione dei livelli di grigio (densità radiografica). Nel contesto HBM, permette di isolare le strutture anatomiche (es. ossa o organi) dal rumore di fondo e dai tessuti circostanti, operazione fondamentale per la successiva ricostruzione geometrica 3D.

[10]Standard Triangulated Language: formato file che descrive la geometria di una superficie 3D mediante una tassellazione di triangoli (mesh di superficie). Rappresenta lo standard di interfaccia

detto si mostra in Figura 2.5 la geometria degli organi interni e in Figura 2.6 quella delle parti scheletriche afferenti al torso.

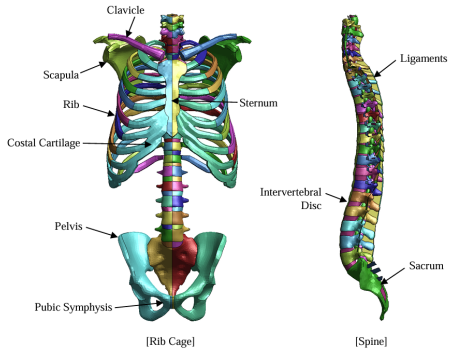


Figura 2.5: Ossa toraciche.

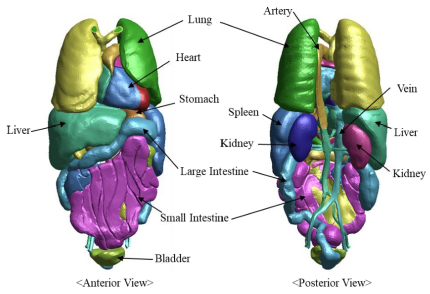


Figura 2.6: Organi interni distretto torace.

2.2.3 Validazione del Modello Numerico THUMS

Il modello THUMS rappresenta uno degli strumenti di calcolo biomeccanico più avanzati e affidabili nello scenario internazionale, grazie a un rigoroso e continuo processo di validazione scientifica. Poiché non è biologicamente né eticamente possibile condurre test d'impatto ad alta energia su volontari umani, la risposta bio-fedele del THUMS è stata calibrata e verificata replicando numericamente i più celebri esperimenti della storia della biomeccanica automotive, condotti originariamente su campioni PMHS[11].

Gli sviluppatori del modello hanno adottato un approccio gerarchico di tipo *bottom-up*: i singoli tessuti biologici (ossa, legamenti, organi e muscoli) sono stati inizialmente validati tramite prove di trazione, compressione e flessione a diversi *strain rate*. Successivamente, i macro-distretti anatomici isolati sono stati messi alla

tra il software di segmentazione e gli strumenti di pre-processing FEA, dove la superficie verrà poi convertita in un volume discretizzato (mesh volumetrica).

[11]Post-Mortem Human Subjects: soggetti umani deceduti utilizzati in esperimenti di biomeccanica storica. I dati derivanti da questi test rappresentano il riferimento sperimentale per la validazione della bio-fedeltà dei modelli numerici (HBM), non essendo eticamente possibile eseguire test ad alta energia su volontari viventi

prova simulando impatti storici di riferimento, come i test di Kroell [12]per valutare la deformabilità e la rigidezza della gabbia toracica, o i test di Nahum[13] per la misurazione delle pressioni intracraniche nella testa in seguito a impatti diretti.

Infine, la cinematica globale a corpo intero è stata validata riproducendo complessi test sperimentali su slitta d'impatto (sled tests[14]), verificando che le curve temporali di accelerazione, forza e spostamento generate dal solutore esplicito LS-DYNA si sovrapponessero fedelmente ai corridoi sperimentali definiti in laboratorio. Questo immenso lavoro di calibrazione garantisce l'assoluta affidabilità del modello nell'estrazione dei dati cinematici e strutturali che verranno successivamente elaborati all'interno di questo studio.

[12]Il test di Kroell rappresenta uno studio fondamentale sulla risposta biomeccanica del torace umano, basato su impatti condotti su cadaveri per determinare le soglie di tolleranza alla compressione e alla forza durante collisioni frontali.

[13]Il test di Nahum rappresenta una pietra miliare in ambito biomeccanico, focalizzato sulla risposta dinamica del torace e del complesso toraco-addominale sottoposto a impatti controllati, fornendo dati essenziali per la calibrazione dei modelli numerici di crash

[14]sled tests: configurazione sperimentale utilizzata per riprodurre le dinamiche d'impatto di un veicolo in ambiente controllato. Il sistema oggetto di studio (manichino o PMHS) viene posizionato su una slitta che subisce un profilo di accelerazione o decelerazione predefinito, permettendo l'acquisizione di dati cinematici e dinamici ad alta precisione, essenziali per la validazione della risposta di modelli HBM come il THUMS

CAPITOLO 3

METODO DI RIDUZIONE DELL'ORDINE: PROPER ORTHOGONAL DECOMPOSITION (POD)

La **Model Order Reduction (MOR)** riveste un ruolo cruciale nella simulazione numerica di sistemi complessi ad alta dimensionalità, come i modelli umani digitali utilizzati nella sicurezza passiva. Nelle simulazioni agli elementi finiti (FEA) su larga scala, l'elevato numero di gradi di libertà necessari per discretizzare accuratamente le strutture anatomiche comporta un onere computazionale spesso proibitivo per analisi iterative o di ottimizzazione. Tra le varie tecniche disponibili in letteratura, la POD rappresenta uno dei metodi più diffusi ed efficienti per estrarre le caratteristiche dominanti — i cosiddetti modi transienti[1] — da un insieme di dati cinematici o dinamici ottenuti da simulazioni full-order[2]. Attraverso la proiezione del sistema su una base ortogonale ridotta, la POD permette di descrivere il comportamento dinamico del sistema preservando le proprietà energetiche fondamentali, riducendo drasticamente la dimensionalità del problema senza sacrificare l'accuratezza necessaria per la validazione biomeccanica..

[1]modi transienti:Rappresentazione della dinamica temporale del sistema all'interno di una decomposizione POD. Sebbene i modi spaziali definiscano la geometria delle deformazioni, il termine 'modi transienti' si riferisce all'evoluzione temporale dei coefficienti associati a tali modi, descrivendo l'intera cinematica del movimento come una sovrapposizione di risposte variabili nel tempo

[2]full-order:Modello originale ad alta fedeltà (spesso indicato come FOM) che mantiene tutti i gradi di libertà originali del sistema agli elementi finiti. Nel contesto della tua tesi, rappresenta la sorgente di dati (snapshot) necessaria per addestrare il modello ridotto e il termine di paragone per validarne l'accuratezza

In questo capitolo verrà presentata la formulazione matematica della POD, il metodo degli snapshot introdotto da Sirovich[3] e i criteri di selezione energetica dei modi per la costruzione di uno spazio a dimensionalità ridotta.

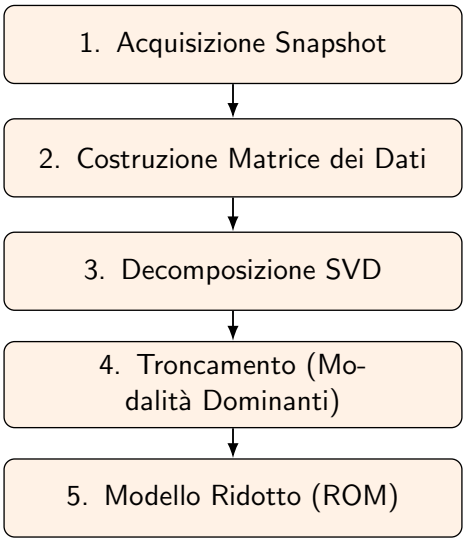


Figura 3.1: Procedura di Model Order Reduction (ROM) basata su POD.

3.1 Il Concetto di Riduzione dell'Ordine e Spazio Ridotto

L'obiettivo fondamentale della POD è approssimare un campo ad alta dimensionalità $u(\mathbf{x}, t)$, che dipende dallo spazio $\mathbf{x} \in \Omega$(Dominio spaziale[4])e dal tempo $t \in [0, T]$, mediante una combinazione lineare di un numero ridotto di funzioni spaziali ortogonali, note come *modi POD* o *funzioni base*.

[3]Sirovich: Matematico che ha introdotto nel 1987 il *Method of Snapshots*, superando il limite computazionale della POD classica. Il suo approccio permette di estrarre i modi di un sistema riducendo il problema di autovalori da una matrice di dimensione $N \times N$ (gradi di libertà) a una di dimensione $M \times M$ (numero di snapshot), rendendo possibile l'analisi di modelli complessi come il THUMS

[4]Dominio spaziale:insieme dei punti che definiscono la geometria del sistema in studio. Nell'ambito della modellazione FEA, rappresenta il volume occupato dal modello discreto (es. THUMS)

L'approssimazione si esprime nella forma a variabili separate:

$$u(\mathbf{x}, t) \approx \sum_{i=1}^{r} a_i(t)\phi_i(\mathbf{x}) \tag{3.1}$$

dove $\phi_i(\mathbf{x})$ sono i modi spaziali, $a_i(t)$ sono i coefficienti temporali (o ampiezze modali) e $r \ll N$ è il numero di modi trattenuti nel modello ridotto, con N pari ai gradi di libertà del sistema completo (*Full Order Model*, FOM).

—

3.2 Il Metodo degli Snapshot e la Formulazione Matematica

Nella pratica numerica, il campo continuo viene discretizzato nello spazio e nel tempo. Si supponga di disporre di un insieme di M soluzioni temporali del sistema, denominate *snapshot* (istantanee), campionate agli istanti di tempo $t_1, t_2, \ldots, t_M$.

Ogni snapshot viene espresso come un vettore colonna $\mathbf{u}_j \in \mathbb{R}^N$, con $j = 1, \ldots, M$. La collezione di tutti gli snapshot definisce la *matrice degli snapshot* $\mathbf{X}$:

$$\mathbf{X} = \begin{bmatrix} | & | & & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_M \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{N \times M} \tag{3.2}$$

Per estrarre le basi ottimali da questa matrice si fa ricorso alla decomposizione a valori singolari (*Singular Value Decomposition*, SVD).

3.3 Decomposizione a Valori Singolari (SVD)

La matrice degli snapshot $\mathbf{X}$ può essere decomposta come:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{3.3}$$

dove:

- $\mathbf{U} \in \mathbb{R}^{N\times N}$ è una matrice ortogonale le cui colonne contengono i vettori singolari sinistri, che corrispondono esattamente ai *modi spaziali POD* ϕ_i.
- $\mathbf{\Sigma} \in \mathbb{R}^{N\times M}$ è una matrice diagonale rettangolare contenente i valori singolari $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k \geq 0$, disposti in ordine decrescente.
- $\mathbf{V} \in \mathbb{R}^{M\times M}$ è una matrice ortogonale i cui vettori colonna rappresentano l'evoluzione temporale dei modi.

3.4 Contenuto Energetico e Criterio di Troncamento

Ogni valore singolare σ_i (o il suo quadrato $\lambda_i = \sigma_i^2$, nel caso si utilizzi la matrice di correlazione) è direttamente proporzionale alla "energia" o varianza statistica catturata dal rispettivo modo POD. La scelta del numero di modi r da trattenere per la ricostruzione del sistema si basa sull'indice di energia accumulata (RIC[5]):

$$E(r) = \frac{\sum_{i=1}^{r} \sigma_i^2}{\sum_{j=1}^{K} \sigma_j^2} \tag{3.4}$$

dove $K = \min(N, M)$. Tipicamente, si sceglie un valore di r tale che $E(r)$ superi una determinata soglia percentuale (ad esempio, il 95% o il 99%), garantendo così

[5]Relative Information Content:rappresenta la frazione di varianza (o energia) contenuta in ogni modo, permettendo di identificare quali componenti descrivono le dinamiche dominanti del sistema e quali possono essere trascurate in quanto contributi trascurabili o rumore

che la dinamica macroscopica del sistema sia preservata e il rumore numerico ad alta frequenza venga filtrato.

3.5 Ricostruzione del Campo e Errore di Proiezione

Una volta selezionati i primi r modi dominanti, la matrice di proiezione ridotta $\mathbf{U}_r \in \mathbb{R}^{N\times r}$ viene utilizzata per mappare il sistema ad alta dimensionalità nello spazio ridotto. Il campo approssimato $\mathbf{X}_r$ è dato da:

$$\mathbf{X}_r = \mathbf{U}_r \mathbf{U}_r^T \mathbf{X} \tag{3.5}$$

L'errore di troncamento (o di proiezione) commesso può essere quantificato in norma di Frobenius come:

$$|\mathbf{X} - \mathbf{X}_r\|_F^2 = \sum_{i=r+1}^{K} \sigma_i^2 \tag{3.6}$$

La proprietà fondamentale della POD, sancita dal teorema di Schmidt-Mirsky-Eckart-Young[6], garantisce che l'approssimazione ottenuta tramite la SVD sia l'ottima in assoluto tra tutte le possibili approssimazioni di rango r su base lineare, minimizzando l'errore dei minimi quadrati.

3.6 Applicazioni della POD

La versatilità della POD trova riscontro in numerosi ambiti dell'ingegneria dove è richiesta un'analisi rapida di sistemi complessi. In fluidodinamica, ad esempio, la POD è impiegata per identificare le strutture coerenti in flussi turbolenti, permettendo di semplificare modelli di calcolo altrimenti proibitivi. In ambito strutturale, la tecnica

[6]teorema di Schmidt-Mirsky-Eckart-Young: stabilisce che la migliore approssimazione di rango r di una matrice X — intesa come quella che minimizza l'errore in norma di Frobenius $\|X - X_r\|_F$ — è ottenuta mediante il troncamento della sua decomposizione SVD ai primi r valori singolari

viene utilizzata per l'analisi dinamica di sistemi con elevato numero di gradi di libertà, facilitando la sintesi di modelli ridotti efficaci per la stima della risposta a vibrazioni aleatorie. Nel presente lavoro, tale approccio viene declinato nella biomeccanica computazionale per affrontare la complessità del modello THUMS. In questo contesto, la POD non funge solo da strumento di riduzione d'ordine (ROM), ma abilita una valutazione efficiente della cinematica dei distretti di interesse, attraverso l'estrazione di snapshot significativi dalle simulazioni LS-DYNA. Tale metodologia permette di sintetizzare la risposta dinamica del sistema in una forma compatta.

CAPITOLO 4

STRUMENTI UTILIZZATI

Il presente capitolo è dedicato alla descrizione della metodologia computazionale e degli strumenti software impiegati nello svolgimento del lavoro di tesi. In particolare, si illustra l'utilizzo di ANSYS LS-DYNA come solutore agli elementi finiti per la simulazione dei fenomeni dinamici indagati. L'attività di pre-processing e la gestione del modello numerico all'interno dei file K-FILE sono state condotte mediante l'ausilio di *LS-PrePost*, software di pre- e post-processing nativo per LS-DYNA. Successivamente, viene presentata la libreria Lasso-Python, adottata come ambiente di sviluppo per l'automazione del post-processing, la manipolazione dei dati in uscita e l'implementazione degli algoritmi di riduzione di ordine di modello (ROM) basati sulla decomposizione POD precedentemente discussa.

4.1 Analisi Dinamica Non Lineare e Metodi Espliciti: Il Ruolo di LS-DYNA nella Simulazione CAE

L'ingegneria moderna si affida a strumenti di calcolo ad alte prestazioni per prevedere il comportamento di sistemi complessi sottoposti a sollecitazioni estreme. In questo panorama, LS-DYNA si configura come uno dei solutori agli elementi finiti (FEM) più avanzati per la simulazione dinamica esplicita, universalmente riconosciuto come leader nel settore. Originariamente sviluppato dalla Livermore Software Technology Corporation (LSTC) e acquisito da Ansys nel 2019, LS-DYNA rappresenta lo standard de facto per applicazioni caratterizzate da forti non linearità, come crash test automobilistici, esplosioni e analisi balistiche.

4.1.1 Classificazione delle Analisi Transitorie

Nell'ambito della dinamica agli elementi finiti, LS-DYNA permette di affrontare problemi transitori distinguendo tra approcci impliciti ed espliciti, a seconda dell'algoritmo di integrazione temporale impiegato.

Metodi Impliciti La soluzione al tempo $t + \Delta t$ viene calcolata imponendo l'equazione di governo al tempo $t + \Delta t$ stesso. Questo richiede processi iterativi, come il metodo di Newton-Raphson[1], rendendo tali metodi adatti a eventi di lunga durata con non linearità moderate (ad esempio prove di trazione quasi-statiche).

Metodi Espliciti La soluzione al tempo $t + \Delta t$ è ottenuta direttamente dalle condizioni al tempo t, senza necessità di iterazioni. Tale approccio è ideale per eventi di brevissima durata, nell'ordine dei millisecondi, dove le non linearità sono intense e repentine, come nel caso degli impatti frontali.

4.1.2 Formulazione Matematica: Il Metodo delle Differenze Centrali

Il cuore pulsante di LS-DYNA è lo schema di integrazione esplicita basato sul metodo delle differenze centrali[2]. Data l'equazione di governo:

[1]Newton-Raphson: algoritmo iterativo di ricerca delle radici, impiegato nei solutori impliciti per risolvere il sistema di equazioni non lineari derivante dall'equilibrio statico o quasi-statico. A ogni passo temporale, il metodo linearizza il sistema calcolando la matrice di rigidezza tangente e aggiornando iterativamente lo spostamento finché il residuo delle forze (differenza tra forze esterne e forze interne) non risulta inferiore a una tolleranza prefissata.

[2]metodo delle differenze centrali: algoritmo di integrazione numerica esplicita a secondo ordine di precisione. Esso si basa sull'approssimazione delle derivate temporali mediante espansioni in serie di Taylor troncate. La sua efficacia nei codici come LS-DYNA risiede nella natura non iterativa: poiché lo spostamento al passo futuro $x_{t+\Delta t}$ dipende esclusivamente da variabili note (spostamenti e accelerazioni ai tempi precedenti), non è richiesta la risoluzione di un sistema di equazioni non lineari (come accade nei metodi impliciti), garantendo un costo computazionale per iterazione estremamente ridotto

$$M_t\,\ddot{x}_t + C_t\,\dot{x}_t + K_t\,x_t = f(t), \tag{4.1}$$

dove M, C e K rappresentano rispettivamente le matrici di massa, smorzamento e rigidezza, il metodo approssima accelerazioni e velocità come:

$$\ddot{x}_t = \frac{1}{\Delta t^2}(x_{t+\Delta t} - 2x_t + x_{t-\Delta t}), \tag{4.2}$$

$$\dot{x}_t = \frac{1}{2\Delta t}(x_{t+\Delta t} - x_{t-\Delta t}). \tag{4.3}$$

Sostituendo tali espressioni nell'equazione di governo, è possibile isolare lo spostamento incognito al passo successivo:

$$x_{t+\Delta t} = \left(\frac{1}{\Delta t^2}M_t + \frac{1}{2\Delta t}C_t\right)^{-1}\left[f(t) - \left(K_t - \frac{1}{\Delta t^2}M_t\right)x_t - \left(\frac{1}{\Delta t^2}M_t + \frac{1}{2\Delta t}C_t\right)x_{t-\Delta t}\right]. \tag{4.4}$$

L'efficienza computazionale di LS-DYNA deriva dall'uso di matrici di massa *lumped* (diagonali), che rendono l'inversione della matrice un'operazione estremamente rapida.

4.1.3 Requisiti di Stabilità e Time Step Critico

L'integrazione esplicita è condizionatamente stabile, il che impone un vincolo rigoroso sulla dimensione del passo temporale Δt. La soluzione rimane stabile solo se:

$$\Delta t \leq \Delta t_{\text{critico}} = \frac{\pi}{\nu_{\max}}, \tag{4.5}$$

dove $\nu_{\max}$ è la massima frequenza propria del sistema. Dal punto di vista ingegneristico, ciò implica che mesh estremamente raffinate (elementi molto piccoli) aumentano la frequenza del sistema, richiedendo time step più piccoli e incrementando proporzionalmente il costo computazionale.

4.1.4 Struttura del Dato: Il K-FILE

LS-DYNA utilizza un formato di input testuale basato su parole chiave, noto come *K-FILE* (estensione `.k`). Questo file contiene l'intera definizione del modello numerico, organizzata in sezioni quali:

- `*NODE`: elenco dei nodi con numerazione e coordinate spaziali;
- `*ELEMENT_SOLID` / `*ELEMENT_SHELL`: definizione della connettività degli elementi finiti;
- `*SET_NODE`: raggruppamenti logici di nodi per l'applicazione di carichi o vincoli;
- `*MAT`: definizione delle leggi costitutive dei materiali (es. elasto-plastici per lo scheletro, iperelastici per i tessuti molli).

4.1.5 *LS-PrePost*: Ambiente di Pre e Post-Processing

LS-PrePost è l'interfaccia grafica avanzata sviluppata nativamente per LS-DYNA, progettata per gestire l'intero ciclo di vita di un modello numerico. Per quanto concerne il *pre-processing*, il software permette la creazione, la modifica e la verifica dei file `.k` (*K-FILE*), consentendo di definire con precisione geometrica le proprietà dei materiali, le condizioni al contorno e i parametri di contatto tra le componenti. In fase di *post-processing*, *LS-PrePost* offre strumenti di visualizzazione 3D in tempo reale e analisi dei time-history plottime-history plot: rappresenta l'evoluzione temporale di una grandezza fisica (quale spostamento, velocità, accelerazione o energia)

calcolata in uno specifico nodo o elemento. A differenza della visualizzazione spaziale delle deformazioni, il grafico temporale consente di isolare la dinamica locale, risultando fondamentale per verificare la convergenza della soluzione e l'andamento delle grandezze critiche durante l'intera durata del transitorio, risultando essenziale per l'interpretazione dei risultati dinamici. La sua perfetta integrazione con il solutore rende questo strumento indispensabile per la validazione della coerenza sintattica e fisica dei modelli prima dell'avvio della simulazione.

4.1.6 Lasso-Python: Automazione e Analisi dei Dati

Lasso-Python è una libreria specializzata, concepita per colmare il divario tra i risultati numerici complessi generati da LS-DYNA e le necessità di analisi avanzata dei dati. A differenza delle interfacce grafiche tradizionali, Lasso-Python opera a livello di programmazione, permettendo l'automazione su larga scala del post-processing. Tale strumento è stato adottato in questo lavoro per la gestione e l'estrazione sistematica degli *snapshot* numerici dai database di uscita (*d3plot*), consentendo una manipolazione efficiente delle matrici di dati necessarie per l'implementazione degli algoritmi di riduzione di ordine di modello (ROM) e per lo sviluppo delle pipeline di analisi statistica richieste dal metodo POD.

4.1.7 Paraview

ParaView è un'applicazione open source di analisi e visualizzazione scientifica basata sul toolkit VTK, ampiamente utilizzata nel settore della meccanica computazionale per il post-processing di dataset complessi. Grazie alla sua architettura modulare e alla completa programmabilità tramite interfaccia Python, esso consente l'esplorazione qualitativa e quantitativa di configurazioni deformate e campi fisici derivanti da simulazioni numeriche agli elementi finiti.

CAPITOLO 5

METODOLOGIA DI PRE-PROCESSING E SETUP DEL MODELLO

Il presente capitolo descrive le operazioni di pre-processing necessarie per la configurazione del modello numerico, con particolare attenzione alla definizione della cinematica di movimentazione. Per simulare in modo controllato le sollecitazioni dinamiche sul modello THUMS, si è fatto ricorso all'implementazione di sistemi di vincolo basati su elementi *cable*, capaci di trasmettere spostamenti imposti mantenendo un'elevata efficienza computazionale. Verranno inoltre discusse le strategie adottate per la selezione dei gradi di libertà, distinguendo tra zone di ancoraggio rigido e segmenti del corpo liberi di deformarsi in risposta al transitorio dinamico.

5.1 Introduzione al Setup Computazionale

Il workflow di pre-processing adottato per la movimentazione del modello umano THUMS v7 ha come base di riferimento la configurazione anatomica standard rilasciata da Toyota, garantendo così la coerenza del modello di partenza con i protocolli di sicurezza accademici e industriali. La complessità intrinseca nel movimentare un modello biomeccanico di tale dettaglio — che conta milioni di elementi finiti — richiede un approccio rigoroso per evitare instabilità numeriche o artefatti cinematici dovuti a vincoli eccessivamente rigidi.

Per superare tali criticità, si è scelto di implementare una strategia di movimentazione basata su un elemento *cable* , ispirata alle funzionalità di automazione presenti in Oasys Primer[1], che permette di guidare il movimento in modo fluido e controllato.

[1]Oasys Primer: è uno dei software di pre-processing leader nel settore automotive per la preparazione di modelli di simulazione destinati al solutore LS-DYNA. Il software è progettato specificamente per gestire l'elevata complessità dei modelli a elementi finiti, offrendo strumenti dedicati alla verifica della qualità della mesh, alla gestione delle gerarchie di componenti (Dummy Tree) e all'automazione

Date le ampie risorse computazionali necessarie per una movimentazione globale, il presente lavoro si focalizza sulla cinematica specifica del distretto ginocchio, caviglia, punta di piede. Tale scelta permette di validare l'efficacia del metodo di movimentazione tramite *cable* su un distretto anatomico complesso, garantendo al contempo la stabilità del modello durante il transitorio dinamico e la precisione nel raggiungimento della configurazione desiderata.

5.2 La Strategia di Movimentazione: Il *cable*

In questo lavoro, l' elemento *cable* è stato implementato non come componente strutturale del modello biofedele, bensì come dummy tree[2]cinematico, ovvero manipolatore virtuale finalizzato al posizionamento controllato del THUMS v7. A differenza delle condizioni al contorno imposte direttamente ai nodi — che spesso inducono distorsioni locali o instabilità numeriche in geometrie complesse come quella della caviglia — il *cable* agisce come un'interfaccia di movimentazione "a basso impatto". Esso permette di trasformare una legge di moto prescritta in uno spostamento fluido, distribuendo il carico su una regione definita e filtrando le accelerazioni impulsive che potrebbero compromettere la stabilità del transitorio dinamico. Tale approccio, ispirato alle tecniche di automazione proprie di Oasys Primer, trasforma il *cable* in un vero e proprio strumento di posizionamento cinematico che garantisce precisione nel raggiungimento della configurazione desiderata e, al contempo, permette di monitorare le forze di reazione necessarie alla movimentazione attraverso l'analisi della *time-history* assiale.

---

del posizionamento dei manichini, garantendo la compatibilità e la coerenza dei dati di input per le simulazioni di crash dinamico

[2]dummy tree:una struttura logica gerarchica utilizzata in software di pre-processing avanzati, come *Oasys Primer*, per organizzare i modelli numerici complessi (come gli HBM o i manichini) in sottosistemi articolati. Questa gerarchia permette di gestire la cinematica del modello semplificando la selezione delle componenti anatomiche e facilitando il posizionamento tramite rotazioni e traslazioni di macro-segmenti (es. arto inferiore, tronco)

5.2.1 Definizione della Part

La *Part* rappresenta l'entità fondamentale del database di LS-DYNA, fungendo da "contenitore" logico che associa in modo univoco una geometria a una specifica formulazione strutturale e materiale. Tale configurazione è stata interamente gestita attraverso l'interfaccia di pre-processing *LS-PrePost*, che permette di definire il suo specifico *Part ID* (PID) con l'obiettivo di isolare l'elemento di controllo dalla struttura originale del modello umano, evitando conflitti di identificazione e garantendo la modularità del setup. L'importanza della definizione della *Part* tramite *LS-PrePost* risiede nella sua capacità di aggregare le proprietà fondamentali in modo strutturato:

Tabella 5.1: Configurazione della Part del cavo di movimentazione.

Parametro	**Valore**	**Descrizione**
TITLE	cavo-cable	Etichetta identificativa della parte nel modello.
PID	94300002	Identificativo univoco della parte.
SECID	15	ID della sezione (*Section*) che definisce la formulazione dell'elemento.
MID	2001	ID del materiale (*Material*) che definisce la legge costitutiva.
EOSID	0	Equazione di stato (non utilizzata per elementi discreti).
HGID	0	ID del controllo *Hourglass* .
GRAV	0	Attivazione effetti gravitazionali .
ADPOPT	0	Opzione di *Adaptive Refinement* (disabilitata).
TMID	0	ID del materiale per la gestione termica.

Tale approccio modulare, supportato dalle funzionalità di gestione gerarchica di *LS-PrePost*, non solo ottimizza la stabilità computazionale, ma rende il setup facilmente riproducibile e modificabile in fase di test, costituendo il pilastro su cui poggia l'intera strategia di movimentazione assistita.

5.2.2 Definizione del Materiale

La definizione del materiale è avvenuta mediante la selezione del modello costitutivo *Type 071: CABLE_DISCRETE_BEAM*. Questa scelta non è casuale: a differenza dei modelli elastici standard, la formulazione *Type 071* è progettata specificamente per simulare il comportamento di funi o tiranti, gestendo nativamente la non-linearità geometrica e, soprattutto, l'assenza di rigidezza in compressione. Come illustrato nella Tabella 5.2, tale modello permette al solutore di calcolare la forza di trazione in funzione dell'allungamento relativo tra due nodi, garantendo che, nel momento in cui la distanza tra i nodi si riduce sotto la lunghezza di riferimento (condizione di compressione), l'elemento si disattivi istantaneamente. Questo comportamento "tensione-solo" è fondamentale per assicurare che il cavo agisca esclusivamente come attuatore cinematico, senza influenzare la dinamica della struttura anatomica del THUMS v7 attraverso forze di spinta parassite.

Tabella 5.2: Parametri costitutivi del materiale per l'elemento cavo (MID: 2001).

Parametro	**Valore**	**Descrizione**
TITLE	cable	Etichetta del materiale.
MID	2001	Identificativo univoco del materiale.
RO	1.0	Densità di massa del materiale.
E	5000.0	Rigidezza elastica (fattore di scala).
LCID	3	ID della curva forza-spostamento (comportamento).
F0	0.0	Forza iniziale pre-esistente nel cavo.(assente)
TMAXF0	0.0	Tempo di applicazione della forza iniziale.
TRAMP	0.0	Tempo di rampa per la forza iniziale.
IREAD	0	Flag di lettura aggiuntiva (default).

La curva è stata definita per simulare il comportamento di un attuatore a trazione, imponendo una legge di forza crescente che permette di controllare lo spostamento meccanico durante la fase di posizionamento. La curva impone un comportamento asimmetrico in cui l'attuatore genera forza esclusivamente in trazione ($\epsilon > 0$), mentre in compressione ($\epsilon \leq 0$) la rigidezza è nulla, impedendo al cavo di trasmettere sforzi

quando è allentato e assicurando che l'attuazione avvenga solo attraverso l'effettivo tiro delle componenti collegate.

Tabella 5.3: Definizione della curva forza-deformazione per *MAT_CABLE(LCID 3).

Deformazione (Strain)	**Forza [N]**
0.00	0.0
0.05	500.0
0.10	1200.0
0.20	2500.0

5.2.3 Definizione della Section

La definizione della SECTION rappresenta lo step logico successivo alla configurazione del materiale, essendo volta a stabilire la formulazione numerica che il solutore utilizzerà per interpretare l'elemento 122. In questo ambito, la scelta della scheda *SECTION_DISCRETE con ELFORM 6 è determinante per il corretto comportamento cinematico del cavo all'interno dell'ambiente di simulazione. La configurazione dei parametri nella sezione ELFORM 6, come riportato nella Tabella 5.4, è stata definita per garantire la massima fedeltà cinematica nel trasferimento della forza tra il punto di ancoraggio e la struttura anatomica del THUMS. La selezione dell'ELFORM 6 risponde alla necessità di utilizzare una formulazione *Discrete Beam* ottimizzata per la modellazione di cavi, in quanto capace di gestire la rigidezza assiale senza indurre effetti flessionali indesiderati. Il fattore di correzione per il taglio (SHRF), impostato al valore unitario, è lo standard per elementi discreti in cui non è richiesta una decomposizione complessa delle tensioni, semplificando così il calcolo della forza di trazione pura. Parallelamente, il parametro QR/IRID pari a 2 è stato adottato per bilanciare il costo computazionale con l'accuratezza del solutore durante l'integrazione temporale. Infine, si è scelto di mantenere a zero i restanti parametri, quali massa non strutturale (NSM) o coordinate locali (CID), al fine di escludere contributi non necessari; tale approccio assicura che l'elemento operi esclusivamente come un dispo-

sitivo di movimentazione privo di inerzia propria, minimizzando ogni interferenza con le proprietà di massa del segmento corporeo analizzato e garantendo una stabilità numerica ottimale durante il transitorio dinamico.

Tabella 5.4: Configurazione della section per elemento cable.

Parametro	**Valore**	**Descrizione**
SECID	15	Identificativo univoco della sezione.
ELFORM	6	Formulazione discreta per elementi trave (Cable).
SHRF	1.0	Fattore di correzione per il taglio (Shear Factor).
QR/IRID	2	Tipo di smorzamento (Hourglass/Integration).
CST	0	Tipo di trave.
SCOOR	0.0	Coordinate locali per la sezione.
NSM	0.0	Massa non strutturale aggiunta.
NAUPD	0	Opzioni di aggiornamento dei nodi.

5.2.4 Definizione del nodo attivo

Al fine di gestire cinematicamente il movimento del sistema,il nodo 81007636 è stato impostato come *master* (`PNODE`) del componente `CNBR`, configurandolo come elemento di controllo del blocco rigido di interfaccia. Questa impostazione garantisce una trasmissione univoca delle forze di trazione e un posizionamento controllato dell'intero set di nodi, permettendo una simulazione accurata della cinematica del calcagno.

Il carico non viene applicato direttamente su un singolo nodo, scelta che genererebbe una singolarità di tensione localizzata e una distorsione non fisica della mesh. Si è optato per la creazione di un nodo *master*, inserito all'interno di un Nodal Rigid Body[3]. Tale approccio è fondamentale poiché permette di governare il movimento del sistema di vincoli come un corpo unico, garantendo che la forza di trazione, generata dall'elemento cable, sia ripartita equamente su un set di nodi superficiali, simulando così l'inserzione di un tendine sull'osso in modo più realistico rispetto a un carico

[3]Nodal Rigid Body:costituisce una formulazione cinematica che impone a un set di nodi, definiti *slave*, di comportarsi come un unico corpo rigido.

puntuale. La regione di applicazione è stata selezionata sulla superficie dell'osso del calcagno del modello.

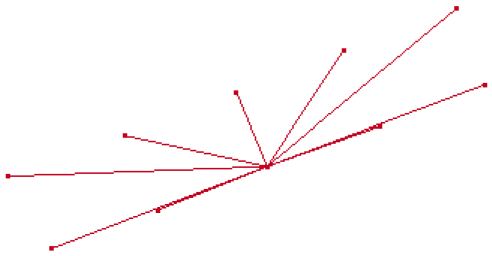
Figura 5.1: Osso Calcagno (R_Calcaneus_Cort).

Questa scelta è dettata dalla necessità di replicare la cinematica naturale della caviglia, identificando un'area che consenta di trasmettere il momento risultante del movimento in modo anatomicamente coerente, minimizzando le interferenze con le strutture articolari adiacenti e rispettando la biomeccanica del piede nel manichino. Di seguito è riportata una tabella esemplificativa del cnbr e una figura illustrativa; i parametri riportati definiscono la configurazione del SET_NODE, utilizzato per identificare univocamente il set di nodi (SID) che compongono il blocco rigido. Tale configurazione permette di distribuire l'azione di trazione dell'attuatore su più punti anatomici del modello THUMS, garantendo un posizionamento cinematica controllato e realistico durante le fasi di simulazione.

Tabella 5.5: Parametri di configurazione del CNRB.

PID	CID	NSID	PNODE	IPRT	DRFLAG	RRFLAG
81002101	0	81007636	99007636	0	0	0

Tabella 5.6: Rappresentazione del CNRB.



Il parametro `NSID` (*Node Set ID*) identifica l'insieme dei nodi coinvolti nell'attuazione, i cui identificativi sono riportati nella tabella sottostante.

Tabella 5.7: Configurazione `CNBR_SET_NODE`.

SID	DA1	DA2	DA3	DA4	SOLVER	ITS
92500014	0.0	0.0	0.0	0.0	MECH	1

NID1	NID2	NID3	NID4	NID5	NID6	NID7	NID8
81007636	81006587	81005720	81006588	81005724	81006595	81007641	81006596

5.2.5 Definizione del nodo passivo

Parallelamente, è stato scelto un nodo passivo (ID 33) con funzione di *anchor* (ancoraggio), posizionato alle coordinate (-1112.69, 127.49, -0.55). Tale nodo presenta tutti i gradi di libertà vincolati, garantendo un punto di riferimento fisso e stabile, essenziale per la reazione vincolare e la corretta cinematica del blocco rigido durante la simulazione.

5.2.6 Definizione dell' Elemento

Una volta completata la definizione dei parametri caratteristici del componente — nello specifico la *part*, la *section*, il materiale (*mat*) e la configurazione dei tre nodi chiave (il *master* node, i nodi appartenenti all'insieme *NSID* e il nodo di *anchor*) — è possibile procedere con la formalizzazione dell'elemento `ELEMENT_SEATBELT_RETRACTOR`. Tale passaggio permette di integrare correttamente le proprietà meccaniche e geometriche definite, abilitando la corretta trasmissione del moto e delle forze di tensione all'interno del blocco rigido del modello.

Tabella 5.8: Definizione dell'elemento cable

Parametro	**Valore/ID**	**Descrizione**
EID	122	Identificativo univoco dell'elemento nel modello.
PID	94300002	Identificativo della Part (richiama Section e Material).
N1	81007636	Nodo attivo .
N2	33	Nodo passivo .
N3	0	Nodo di orientamento (nessuno in questo caso).
RT1	0	Grado di libertà traslazionale rilasciato (vincolato).
RR1	0	Grado di libertà rotazionale rilasciato (vincolato).
RT2	0	Grado di libertà traslazionale rilasciato (vincolato).
RR2	0	Grado di libertà rotazionale rilasciato (vincolato).
LOCAL	2	Sistema di coordinate locale per il calcolo della forza.

5.3 Gestione dei Vincoli e Condizioni al Contorno

Nel capitolo dedicato alla modellazione dei vincoli, si è operata una semplificazione cinematica finalizzata a isolare la risposta biomeccanica dell'arto inferiore. È stata imposta una condizione di vincolo rigido sull'intera parte superiore del corpo, la quale è stata, di fatto, "ingessata" per impedire ogni movimento parassita, lasciando libera di muoversi esclusivamente la gamba.

Il sistema di vincoli si completa attraverso tre interazioni fondamentali:

- L'azione esercitata dal nodo *cable* (`CNBR`), che funge da attuatore per il controllo delle trazioni (illustrato nel capitolo 4.2);
- L'azione del nodo di *anchor* (ID 33), che fornisce la reazione vincolare necessaria alla stabilità dell'intero blocco rigido di interfaccia (illustrato nel capitolo 4.2).
- La restrizione ad un dominio di interesse ridotto (dalla testa del femore alla punta di piede), il quale permette di rendere le simulazioni meno onerose.

Tale configurazione permette di confinare il campo di analisi cinematica e dinamica al solo segmento anatomico di interesse, garantendo la coerenza dei risultati numerici rispetto alle sollecitazioni applicate. Al fine di isolare la risposta biomeccanica dell'arto inferiore, è stata applicata una condizione di vincolo sulla parte superiore del modello THUMS. Tale operazione è stata eseguita tramite il comando `*BOUNDARY_SPC_SET`, applicato a un set di nodi opportunamente definito (graficamente) per l'intero tronco e i segmenti superiori. L'imposizione di tale vincolo ha permesso di inibire tutti i gradi di libertà (*degrees of freedom*) della porzione superiore del corpo, rendendola solidale al sistema di riferimento globale e garantendo che ogni variazione cinematica registrata sia riconducibile esclusivamente alla dinamica della gamba.

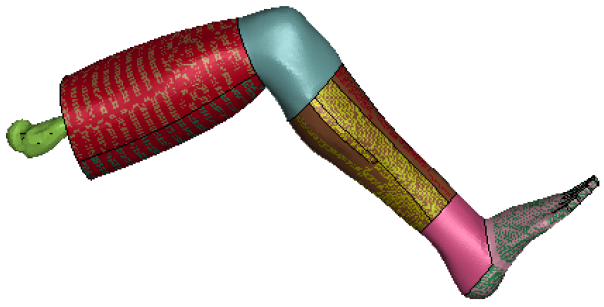


Figura 5.2: Elementi non vincolati.

5.4 Cinematica e Controllo del Transitorio

Nel presente paragrafo viene illustrata la strategia adottata per l'imposizione della cinematica sul modello. Per azionare il nodo sulla caviglia (ID 81007636), è stata implementata una coppia di condizioni al contorno tramite `BOUNDARY PRESCRIBED MOTION NODE ID`. Nello specifico, le due curve di moto (*Load Curve* 145 e 146) agiscono rispettivamente sui gradi di libertà traslazionali 1 e 3 (asse x e z); tali componenti sono state opportunamente modellate per operare come le proiezioni cartesiane di un unico vettore spostamento risultante. Al fine di costruire il database necessario per la successiva fase di riduzione modale, è stata eseguita una campagna di simulazioni numeriche. Dopo aver definito la strategia di movimentazione, sono state condotte simulazioni per le cinque configurazioni riportate nella Tabella 5.9, selezionate per coprire una varietà rappresentativa di spostamenti nello spazio dei parametri.

Tabella 5.9: Configurazioni geometriche analizzate tramite simulazioni LS-DYNA.

Configurazione	**DX [mm]**	**DZ [mm]**	$\lvert D \rvert$ **[mm]**	**Note**
Caso A	-148.6	+133.8	≈ 200	Lungo direzione originale
Caso B	-100.0	+173.0	≈ 200	Più in alto
Caso C	-180.0	+87.0	≈ 200	Più orizzontale
Caso D	+45.0	+20.0	≈ 49	Corto, verso l'interno
Caso E	-120.0	-90.0	150.0	Abbassamento in Z

La Tabella 5.10 illustra la card utilizzata per imporre la condizione di moto. Tale configurazione risulta invariata per tutti e cinque i casi analizzati, differenziandosi unicamente per la definizione delle curve di carico (LCID) associate.

Tabella 5.10: `*BOUNDARY_PRESCRIBED_MOTION_NODE_ID`.

ID	**NID**	**DOF**	**VAD**	**LCID**	**SF**
2323	81007636	1	2	145	1.0
2324	81007636	3	2	146	1.0

Di seguito sono riportate le tabelle esplicative relative ai carichi cinematici applicati al sistema. Si precisa che la giustificazione tecnica riguardante la scelta degli

intervalli temporali e della relativa discretizzazione verrà trattata approfonditamente nei paragrafi successivi.

5.4.1 Caso A

Il caso A si riferisce alla configurazione della curva di riferimento .

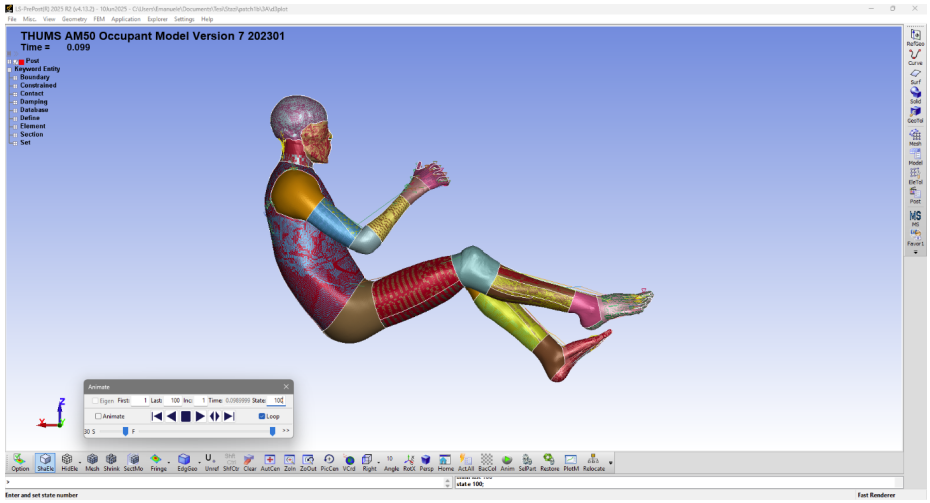


Figura 5.3: visualizzazione LS-prepost posizione finale caso A.

L'analisi del caso A ha evidenziato una piena convergenza numerica, associata a una cinematica del distretto analizzato pienamente coerente con la risposta biomeccanica attesa.

Tabella 5.11: LCID 145 (direzione X)

Tempo [s]	**spost. [mm/s]**
0.000	0.0
0.025	-20.0
0.050	-72.0
0.075	-118.0
0.100	-148.6

Tabella 5.12: LCID 146 (direzione Z)

Tempo [s]	**spost. [mm/s]**
0.000	0.0
0.025	18.0
0.050	64.8
0.075	106.0
0.100	133.8

5.4.2 Caso B

Il caso B è caratterizzato da uno spostamento di 200 mm più in alto rispetto alla configurazione di riferimento.

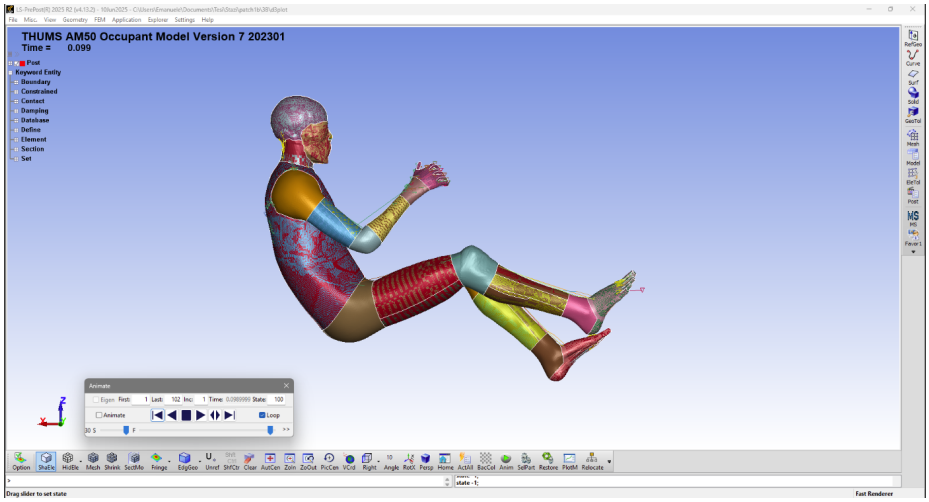


Figura 5.4: visualizzazione LS-prepost posizione finale caso B.

L'analisi del caso B ha evidenziato una piena convergenza numerica, associata a una cinematica del distretto analizzato pienamente coerente con la risposta biomeccanica attesa.

Tabella 5.13: LCID 145 per il Caso B

Tempo [s]	**Valore (o1)**
0.000	0.0
0.025	-13.5
0.050	-48.5
0.075	-79.5
0.100	-100.0

Tabella 5.14: LCID 146 per il Caso B

Tempo [s]	**Valore (o1)**
0.000	0.0
0.025	23.4
0.050	83.8
0.075	137.5
0.100	173.0

5.4.3 Caso C

Il caso C è caratterizzato da uno spostamento 200 mm più a destra rispetto alla configurazione di riferimento.

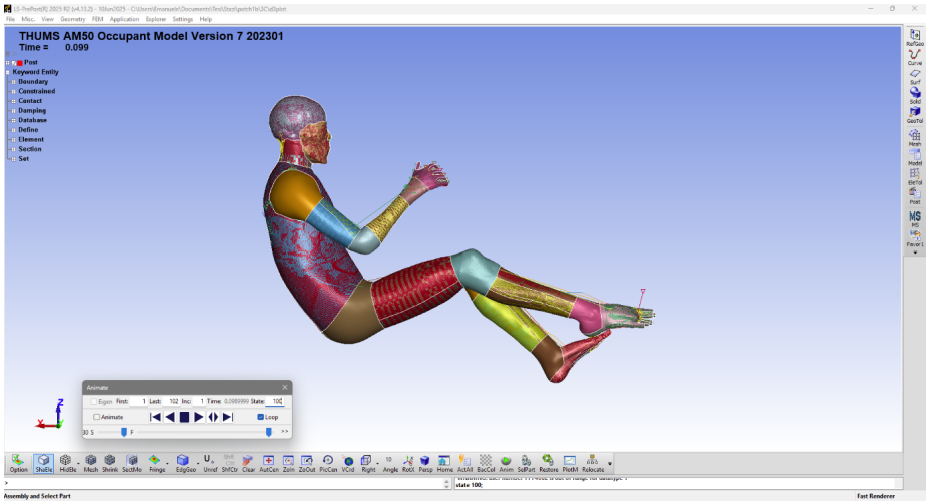


Figura 5.5: visualizzazione LS-prepost posizione finale caso C.

L'analisi del caso C ha evidenziato una piena convergenza numerica, associata a una cinematica del distretto analizzato pienamente coerente con la risposta biomeccanica attesa.

Tabella 5.15: LCID 145 per il Caso C

Tempo [s]	**Valore (o1)**
0.000	0.0
0.025	-24.3
0.050	-87.3
0.075	-143.1
0.100	-180.0

Tabella 5.16: LCID 146 per il Caso C

Tempo [s]	**Valore (o1)**
0.000	0.0
0.025	11.7
0.050	42.2
0.075	69.2
0.100	87.0

5.4.4 Caso D

Il caso D si caratterizza per una corsa molto breve (circa 50 mm) in direzione X positiva (verso l'interno).

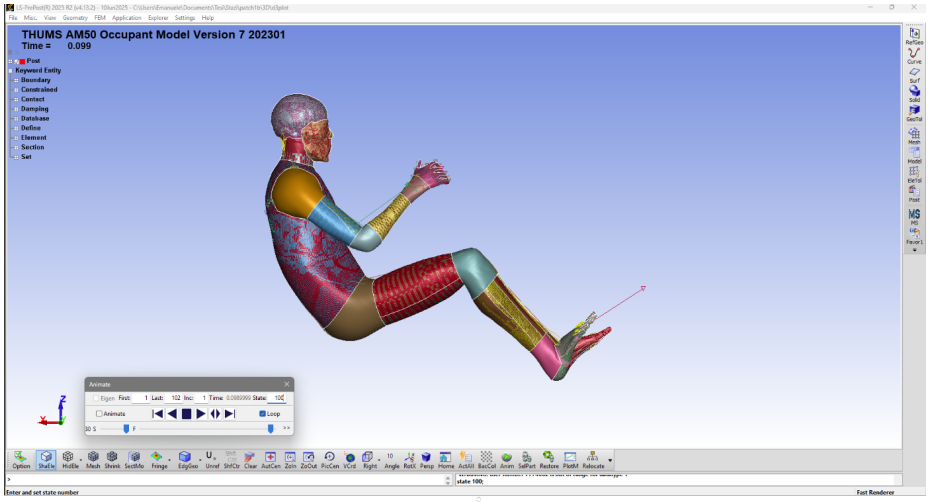


Figura 5.6: visualizzazione LS-prepost posizione finale caso D

L'analisi del caso D ha evidenziato una piena convergenza numerica, associata a una cinematica del distretto analizzato pienamente coerente con la risposta biomeccanica attesa.

Tabella 5.17: LCID 145 per il Caso D

Tempo [s]	**Valore (o1)**
0.000	0.0
0.025	6.1
0.050	21.8
0.075	35.8
0.100	45.0

Tabella 5.18: LCID 146 per il Caso D

Tempo [s]	**Valore (o1)**
0.000	0.0
0.025	2.7
0.050	9.7
0.075	15.9
0.100	20.0

5.4.5 Caso E

Il caso E presenta la seguente configurazione delle curve di spostamento.

Tabella 5.19: LCID 145 per il Caso E

Tempo [s]	Valore (o1)
0.000	0.0
0.025	-16.2
0.050	-58.2
0.075	-95.4
0.100	-120.0

Tabella 5.20: LCID 146 per il Caso E

Tempo [s]	Valore (o1)
0.000	0.0
0.025	-12.2
0.050	-43.7
0.075	-71.6
0.100	-90.0

Sebbene la simulazione E abbia raggiunto la convergenza numerica, essa è stata esclusa dall'analisi in quanto, a partire dallo step 75, la cinematica del modello ha manifestato una perdita di plausibilità biomeccanica, rendendo i dati successivi non rappresentativi del fenomeno reale. Di seguito è riportato il posizionamento dell'occupante allo step 75.

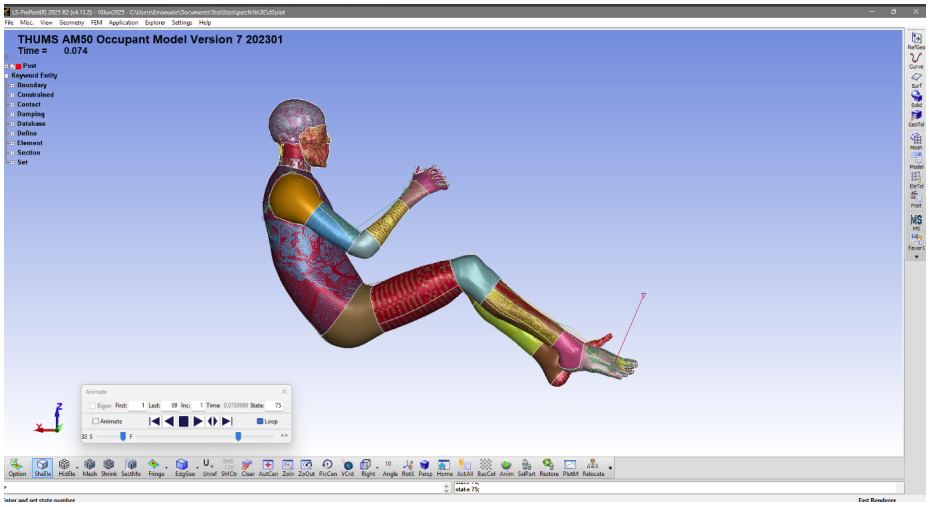


Figura 5.7: Visualizzazione step 75 caso E (LS-prepost).

5.5 Configurazione dei parametri di simulazione e criteri di campionamento

Al fine di garantire un'adeguata risoluzione temporale nell'estrazione dei dati, la simulazione è stata impostata con una durata complessiva di 0.1 s, prevedendo un campionamento dei risultati con una frequenza che consente di ottenere circa 100 istanti di analisi. Tale configurazione è stata definita imponendo un intervallo di estrazione costante pari a 0.001 s. Parallelamente, la definizione delle *load curve* (*LCID*) è stata calibrata in funzione della durata totale della simulazione, assicurando che la cinematica imposta si sviluppi in modo coerente con l'intervallo temporale prescelto. Per ogni istante di analisi la simulazione fornisce una serie di output, ognuno stampato in un file separato.

Tabella 5.21: Panoramica dei file di output generati da LS-DYNA.

Nome File	**Descrizione**
`d3plot`	Database binario per analisi post-processo e visualizzazione.
`glstat`	Statistiche globali su energie e bilancio temporale.
`nodout`	Storia temporale di spostamenti, velocità e accelerazioni.
`matsum`	Sintesi dell'energia dissipata per ogni materiale.

Tra questi, il file *d3plot* riveste un ruolo cruciale: esso costituisce l'unico output di interesse per l'analisi post-processo, in quanto contiene l'intera storia cinematica e deformativa del modello THUMS. Grazie alla sua capacità di memorizzare la configurazione geometrica e le variabili di stato in istanti temporali discreti, il *d3plot* permette un'analisi spaziale dettagliata del comportamento biomeccanico, rendendo superflua la generazione di altri database binari o file di output ausiliari ai fini della presente trattazione. Le impostazioni relative alla frequenza di salvataggio dei dati nel file binario *d3plot* sono riassunte nella tabella seguente.

Tabella 5.22: Configurazione della keyword per il salvataggio dei file d3plot.

Keyword	Parametro	Valore/Descrizione
`*DATABASE_BINARY_D3PLOT`	DT	0.001 (intervallo di scrittura)
	LCDT	0 (impostazione costante)

5.6 Sintesi del Workflow di Pre-processing

La procedura seguita per la configurazione del modello e la preparazione della simulazione numerica è schematizzata nel diagramma di flusso seguente, che illustra le fasi sequenziali dal caricamento del modello THUMS fino alla creazione del nuovo modello, il quale sarà sovrascritto in un nuovo file.k.

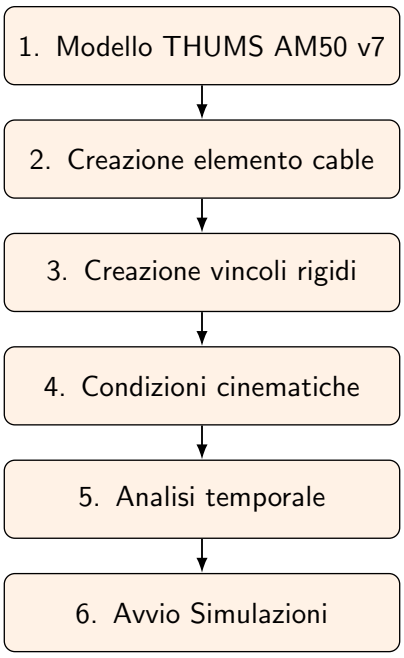


Figura 5.8: Workflow della procedura di preprocessing e setup della simulazione.

5.7 Esecuzione della simulazione

In questa sezione viene descritta l'esecuzione delle simulazioni numeriche precedentemente configurate. Il file di input `.k`, completo di tutti i vincoli e le condizioni al

contorno, è stato inserito all'interno della directory di lavoro di LS-Run[4], da cui è stato avviato il processo di risoluzione.

[4]LS-Run è l'interfaccia grafica fornita da LSTC per la gestione e l'esecuzione dei job di calcolo con il solutore LS-DYNA.

CAPITOLO 6

POST-PROCESSING E WORKFLOW COMPUTAZIONALE

Il presente capitolo descrive le procedure adottate per il trattamento dei risultati numerici e la successiva implementazione del workflow computazionale. Si analizzeranno le fasi di post-processing, necessarie alla standardizzazione dei dati, e la metodologia di costruzione del modello ridotto, finalizzata all'esplorazione parametrica in ambiente di realtà virtuale.

6.1 Gestione degli output e conversione dati

Il workflow computazionale ha inizio con l'elaborazione dei risultati transitori delle simulazioni LS-DYNA, archiviati nei file `d3plot`. Tali database contengono la cronistoria completa dell'evoluzione geometrica e delle grandezze fisiche del modello THUMS. Per ottimizzare l'analisi, si è fatto ricorso a uno script Python dedicato (si veda listing A.1), basato sull'utilizzo della libreria `lasso`, che permette di interfacciare in modo efficiente i file binari prodotti dal solutore e trasformare i risultati in file .vtk . Tale script, analizzando il file `keyword` originale, isola gli elementi *shell* appartenenti al `*SET_PART_LIST` della pelle, consentendo di escludere le componenti strutturali non rilevanti ai fini del modello ridotto e contenendo drasticamente il peso computazionale. Un requisito metodologico fondamentale in questa fase è il mantenimento dell'isotopologia: tutti gli *snapshot* devono presentare un numero di nodi, un ordinamento nodale e una connettività del tutto identici. Tale condizione è essenziale affinché ogni configurazione possa essere rappresentata come un vettore numerico di dimensione costante, garantendo una corrispondenza fisica univoca tra i gradi di libertà in tutto lo spazio temporale. Per tale motivo, è stata definita una mesh compatta con connettività fissata *a priori*, rendendo le variazioni tra gli istanti

di campionamento dipendenti esclusivamente dallo spostamento dei nodi e dall'evoluzione dei campi fisici. Durante la conversione in formato `VTK`, lo script Python estrae le coordinate deformate, i vettori spostamento . Il processo si conclude con un campionamento temporale personalizzato, che genera una sequenza ordinata di file `VTK` unitamente a un file indice `.pvd`, consentendo una gestione fluida delle serie temporali e la relativa visualizzazione all'interno dell'ambiente *ParaView*.

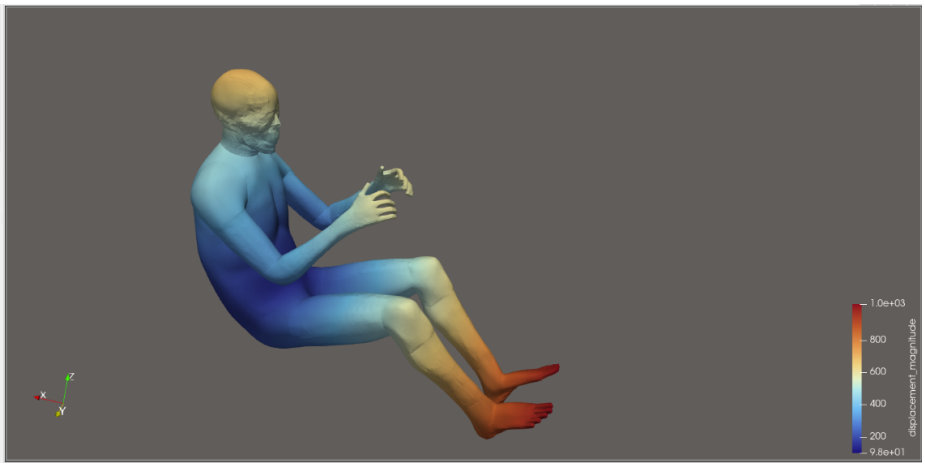


Figura 6.1: Visualizzazione di un d3plot nell'ambiente Paraview.

6.2 Definizione e Ruolo della Tabella di input

Il file di input rappresenta la struttura logica e metodologica dell'intero progetto, fungendo da vera e propria "mappa" dello spazio dei parametri entro il quale il modello ridotto può muoversi e operare. Si configura come un database strutturato che raccoglie le istruzioni relative a ogni configurazione simulata. La sua struttura è organizzata come segue:

- **Indice del Design Point (Prima colonna):** ogni riga esordisce con un identificativo numerico univoco (DP), che associa in modo biunivoco la riga a una specifica configurazione o istante temporale della simulazione.

- **Parametri Indipendenti (Colonne centrali):** questa sezione riporta i valori delle variabili di input (grandezze fisiche, geometriche o di carico) modificate sistematicamente per generare le risposte del modello. Tali parametri definiscono lo spazio di progetto, ovvero i confini entro i quali il modello ridotto è in grado di predire nuovi risultati tramite l'interpolazione.

In accordo con la metodologia descrittiva presentata, la Tabella 6.1 illustra la struttura del file `Input.csv`. Tale database è stato generato mediante l'estrazione automatizzata dei parametri caratteristici direttamente dai file `d3plot`, avvalendosi dello script Python riportato integralmente nel listing A.2. Parallelamente, al fine di esplicitare la configurazione geometrica del sistema, si riportano in Figura 6.2 le rappresentazioni grafiche che definiscono gli angoli α e β, utilizzati come variabili indipendenti all'interno del piano sperimentale.

Tabella 6.1: Struttura del database `input.csv` con parametri angolari e dominio temporale.

DP	α **[rad]**	β **[rad]**
0	-0.50	0.30
1	-0.45	0.30
2	-0.40	0.30

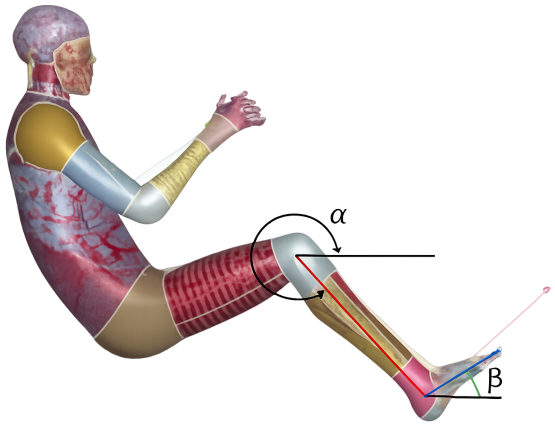


Figura 6.2: Rappresentazione geometrica degli angoli α e β nel sistema di riferimento locale.

6.3 Costruzione della matrice degli Snapshot

Il workflow implementato prevede la generazione di due distinte matrici di snapshot, fondamentali per descrivere tanto la configurazione spaziale quanto la risposta fisica del modello. Come introdotto nel Capitolo 3, la decomposizione modale richiede un'organizzazione rigorosa del dato, la cui caratterizzazione nel nostro caso è di seguito descritta. Uno degli aspetti metodologici più critici riguarda la garanzia di una corrispondenza biunivoca tra le configurazioni cinematiche, rappresentate dai file in formato VTK, e il rispettivo set di parametri di input. Tale accoppiamento è mediato dall'adozione di una rigorosa convenzione di nomenclatura. In particolare la matrice $S \in \mathbb{R}^{n \times m}$, è stata assemblata aggregando i risultati estratti dalle simulazioni numeriche relative ai casi A, B, C e D, previa esclusione del caso E dall'analisi (vedi capitolo 5.4.5). Lo script di generazione degli snapshot (parte del listing A.1) associa infatti ogni geometria estratta alla convenzione riportata in tabella 6.2. Si vuole precisare che è stato selezionato un unico istante temporale comune (tra i casi A B C D) ($t = 0$), d3plot_00 archiviato come `vtk1`, per garantire l'unicità del dataset di partenza e prevenire l'insorgenza di ridondanze che renderebbero la matrice singolare. Si è adottata una nomenclatura progressiva sincronizzata tra file di input (d3plot) e output (`.vtk`), estraendo dati ogni 0.004s.

Tabella 6.2: Convenzione nomenclatura

d3plot	vtk	dp
d3plot_00a	vtk_00	dp_00
d3plot_01a	vtk_01	dp_01
⋮	⋮	⋮
d3plot_100a	vtk_25	dp_25
d3plot_1b	vtk_26	dp_26
⋮	⋮	⋮
d3plot_100b	vtk_50	dp_50
d3plot_1c	vtk_51	dp_51
⋮	⋮	⋮
d3plot100c_	vtk_75	dp_75
d3plot_1d	vtk_76	dp_76
⋮	⋮	⋮
d3plot_100d	vtk_100	dp_100

Con questa covenzione si stabilisce un legame diretto e univoco tra il file fisico e la corrispondente riga del database di input, identificata dal medesimo indice numerico. La robustezza di tale procedura, che deriva i parametri geometrici direttamente dai file `d3plot`, è garantita dallo script di estrazione del database riportato nel Listato A.2.

L'ordinamento sequenziale dei file, basato su tale indice, garantisce che la struttura della matrice degli snapshot sia intrinsecamente coerente con la sequenza dei parametri di addestramento. L'adozione di questo criterio costituisce una condizione necessaria per preservare l'integrità del dato; un eventuale disallineamento in questa fase comporterebbe infatti un'errata associazione tra geometria e parametri di input, inficiando la validità fisica del modello ridotto (ROM) e compromettendone drasticamente l'affidabilità nei successivi processi di inferenza.

6.3.1 Dataset geometrico

La trasformazione del dataset geometrico discreto in una struttura algebrica idonea alle tecniche di riduzione dell'ordine, avviene attraverso la costruzione della matrice

degli snapshot $S \in \mathbb{R}^{n \times m}$. Come anticipato nel Capitolo 3, dove è stata presentata la formulazione generale del metodo, la caratterizzazione della matrice nel nostro caso specifico è la seguente. Ogni file VTK, relativo a un istante temporale i-esimo, viene sottoposto a un'operazione di *flattening* (appiattimento), trasformando la geometria della mesh in un vettore colonna $\mathbf{u}_i \in \mathbb{R}^n$, dove $n = 3N$ rappresenta il numero totale di gradi di libertà (GDL) del sistema, dato dal prodotto tra il numero di nodi N e le tre componenti spaziali (x, y, z). La struttura del vettore è la seguente:

$$\mathbf{u}_i = [x_{1,i}, y_{1,i}, z_{1,i}, \ldots, x_{N,i}, y_{N,i}, z_{N,i}]^T \tag{6.1}$$

La matrice degli snapshot S si ottiene concatenando orizzontalmente i vettori colonna ottenuti per ogni istante temporale m presente nel database input, seguendo rigorosamente l'ordinamento temporale imposto dall'indice progressivo:

$$S = \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_m \end{bmatrix} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ y_{1,1} & y_{1,2} & \cdots & y_{1,m} \\ z_{1,1} & z_{1,2} & \cdots & z_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,1} & x_{N,2} & \cdots & x_{N,m} \\ y_{N,1} & y_{N,2} & \cdots & y_{N,m} \\ z_{N,1} & z_{N,2} & \cdots & z_{N,m} \end{bmatrix} \tag{6.2}$$

In tale struttura, le dimensioni risultanti sono:

- n **(Righe):** definisce lo spazio dei gradi di libertà del sistema, pari a $3 \times N$.
- m **(Colonne):** indica il numero totale di campionamenti geometrici estratti.

6.3.2 Dataset fisico

Ai fini dell'analisi dei campi di spostamento, il workflow prevede la costruzione della matrice degli snapshot fisici $S_{phys} \in \mathbb{R}^{n \times m}$. Tale struttura differisce dalla matrice delle configurazioni geometriche S_{geo} per il contenuto informativo: mentre quest'ultima raccoglie le coordinate assolute, S_{phys} è dedicata alla descrizione della risposta meccanica del modello, isolando le variazioni cinematiche rispetto alla configurazione indeformata $\mathbf{x}_0$.

Ogni colonna della matrice, indicata come snapshot fisico $\mathbf{u}_i \in \mathbb{R}^n$, è definita come il vettore degli spostamenti nodali:

$$\mathbf{u}_i = \mathbf{x}_i - \mathbf{x}_0 \tag{6.3}$$

dove $\mathbf{x}_i$ rappresenta la configurazione campionata all'istante i-esimo. La matrice assume quindi la forma:

$$S_{phys} = \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_m \end{bmatrix} \tag{6.4}$$

In tale formulazione, la dimensione n del vettore colonna risulta pari a $3 \times N$, essendo il campo degli spostamenti una grandezza vettoriale definita per ciascuno degli N nodi della mesh.

6.4 Generazione del modello ridotto tramite POD

Una volta strutturato il database numerico nelle matrici degli snapshot, la fase centrale del workflow consiste nell'applicazione della *Proper Orthogonal Decomposition* (POD). Tale tecnica permette di operare una drastica riduzione della dimensionalità del problema, estraendo le caratteristiche salienti del sistema da un insieme di soluzioni ad alta fedeltà. L'obiettivo è rappresentare la configurazione generica del

modello come una combinazione lineare di un numero limitato di funzioni ortonormali, denominate modi principali.

6.4.1 Decomposizione modale della geometria e del campo fisico

Il processo di decomposizione viene applicato in modo indipendente sia alla matrice delle configurazioni geometriche S_{geo} che alla matrice degli snapshot fisici S_{phys}. Questa separazione metodologica consente di generare due modelli ridotti distinti ma tra loro accoppiati:

- **Modello Geometrico:** finalizzato alla ricostruzione della morfologia deformata della skin del modello THUMS.
- **Modello del Campo:** dedicato alla ricostruzione della distribuzione spaziale delle grandezze fisiche (quali spostamenti nodali) sulla superficie stessa.

Dal punto di vista analitico, ogni snapshot $\mathbf{x}(\boldsymbol{\mu})$ associato a un vettore di parametri di input $\boldsymbol{\mu}$ viene approssimato secondo la seguente espressione:

$$\mathbf{x}(\boldsymbol{\mu}) \approx \sum_{i=1}^{r} a_i(\boldsymbol{\mu})\boldsymbol{\phi}_i \tag{6.5}$$

dove $\boldsymbol{\phi}_i$ rappresentano i vettori della base ridotta (modi POD), $a_i(\boldsymbol{\mu})$ sono i corrispondenti coefficienti o pesi modali, e r indica il numero di modi mantenuti nel modello, con $r \ll n$.

6.4.2 rbfROM

La decomposizione precedentemente descritta è implementata tramite il software rbfROM, il quale automatizza il calcolo della base ortonormale e la determinazione dei relativi pesi modali. Tale ambiente computazionale consente di integrare in modo

efficiente l'approccio Reduced Order Modeling, garantendo la ricostruzione del campo di moto attraverso la combinazione lineare delle componenti estratte.Tale elaborazione consente di operare una drastica riduzione dimensionale, convertendo il considerevole volume di dati associato ai file VTK originali in una rappresentazione compatta, definita esclusivamente dai modi dominanti e dai relativi coefficienti. Solo a valle di questa sintesi informativa è possibile procedere alla fase successiva di preparazione dei dati funzionale all'inferenza[1].

6.5 Inferenza

A valle della generazione del modello ridotto, si procede all'organizzazione sistematica dei dati al fine di rendere il sistema operativo per l'inferenza tramite il software rbfROM. In tale fase, il ROM non viene ricompilato, bensì strutturato per consentire l'estrapolazione predittiva su configurazioni non incluse nel database di addestramento.

6.5.1 Input e integrazione delle informazioni

Il punto di partenza è costituito dai dati estratti dal processo di decomposizione modale gestito da rbfROM:

- **Modi e coefficienti modali:** riferiti sia alla geometria che ai campi fisici di interesse.
- **Architettura del input e parametri di *input*:** integrati con la geometria di riferimento e il file VTK relativo alla configurazione *baseline*.

[1]inferenza: si intende il processo di stima dei coefficienti modali per configurazioni di input non incluse nel campione di addestramento (snapshot). Tale operazione trasforma la rappresentazione ridotta in un modello predittivo capace di approssimare la risposta del sistema fisico, superando la necessità di eseguire nuove simulazioni high-fidelity

Tale architettura permette di stabilire una corrispondenza biunivoca tra lo spazio dei parametri e la ricostruzione dei domini geometrici e strutturali.

6.5.2 Meccanismo di ricostruzione dei campi

Per stimare la risposta del sistema in configurazioni inedite, il modello implementa, tramite le funzionalità di rbfROM, una funzione interpolante che mette in relazione i punti del input con i relativi coefficienti modali. Per ogni nuova configurazione richiesta:

- Viene effettuata la stima dei coefficienti modali target.
- Tali coefficienti sono combinati linearmente con i modi POD per ricostruire la geometria e il campo fisico, evitando il costo computazionale di una simulazione numerica *high-fidelity*.

6.5.3 Output finale e risultati numerici

Al termine della procedura impostando un errore di mesh massima di 5mm sono stati estratti dal softwere (rbfRom) 8 modi. L'esito di tale procedura è la generazione di un *dataset* strutturato, contenente i modi, i coefficienti, i punti sorgente normalizzati e le funzioni interpolanti. Tale pacchetto costituisce lo strumento operativo del modello ridotto, permettendo un'interrogazione rapida e accurata per configurazioni mai simulate in precedenza. in figura è mostrato il modello Ridotto visualizzato nel softwere rbfCAE.

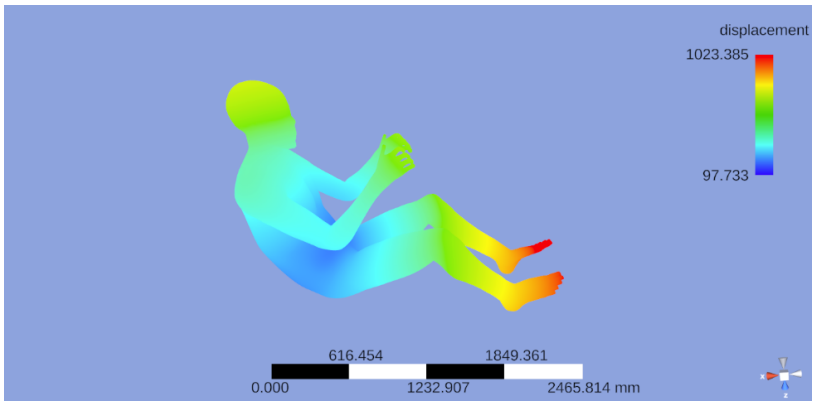


Figura 6.3: Visualizzazione del modello ridotto su rbfCAE

A scopo illustrativo, sono stati riportati in figura 6.4 i due slider integrati nell'interfaccia del software rbfCAE, i quali consentono la variazione in tempo reale degli angoli α e β. Tale funzionalità permette di aggiornare istantaneamente la configurazione del modello ridotto, garantendo un posizionamento coerente rispetto ai parametri selezionati. Poiché l'azione di tali variabili risulta disaccoppiata, è possibile esplorare un ampio spazio di configurazioni, combinando i due gradi di libertà in modo indipendente.

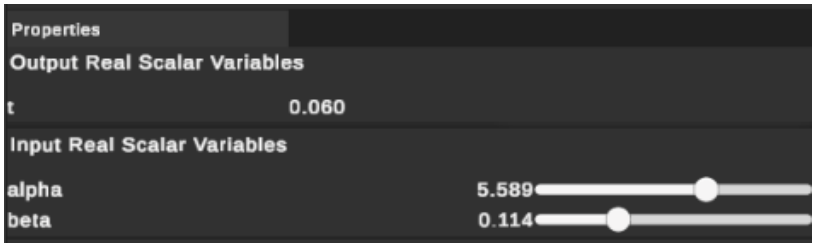


Figura 6.4: Slider interfaccia rbfCAE.

6.6 Dai d3plot al ROM

Il workflow metodologico adottato, illustrato nella Figura 5.3, sintetizza il processo di trasformazione dei dati dalla scala di simulazione ad alta fedeltà fino alla definizione del modello predittivo finale. Il percorso si articola attraverso l'estrazione e il processamento dei dati dai file `d3plot`, la costruzione della base di conoscenza mediante la

matrice degli snapshot e la successiva applicazione della procedura di riduzione POD. Tale iterazione culmina nell'inferenza basata su rbfROM, che permette di ottenere un'agile capacità predittiva riducendo drasticamente l'onere computazionale rispetto al modello originale.

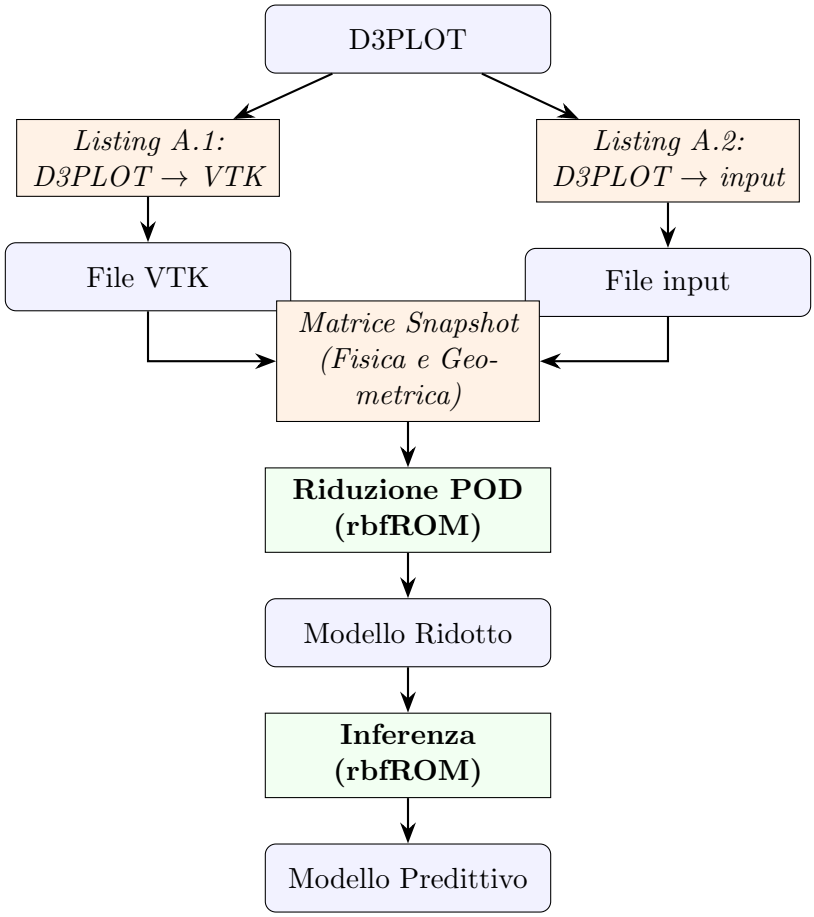


Figura 6.5: Workflow metodologico: dalla simulazione al modello predittivo.

CONCLUSIONI

Il presente lavoro di tesi ha dimostrato l'efficacia dell'approccio basato su Reduced Order Modeling (ROM) per l'analisi biomeccanica del modello THUMS. L'integrazione della tecnica di Proper Orthogonal Decomposition (POD) con l'inferenza tramite RBF-ROM ha permesso di superare i limiti computazionali intrinseci alle simulazioni dinamiche ad alta fedeltà. I risultati ottenuti confermano la validità del metodo proposto: la riduzione dell'onere computazionale è drastica, consentendo di passare da tempi di elaborazione dell'ordine delle ore, necessari per le simulazioni numeriche tradizionali, a tempi di risposta quasi istantanei. Tale incremento di efficienza non avviene a discapito della precisione, garantendo una fedeltà dei risultati comparabile a quella del modello originale. Un punto di forza fondamentale della metodologia sviluppata risiede nella sua estrema modularità. L'approccio implementato non è limitato al singolo distretto corporeo analizzato, ma risulta facilmente estensibile a qualsiasi altra regione anatomica di interesse. La flessibilità della procedura permette, infatti, di scalare il modello su una porzione estesa del corpo umano o sull'intero apparato, semplicemente mediante l'integrazione di un maggior numero di punti di controllo o di ulteriori vincoli cinematici. È opportuno sottolineare che l'efficienza computazionale raggiunta deriva anche da una precisa strategia di post-processing: l'estrazione dai file `d3plot` si è focalizzata esclusivamente sulla superficie esterna del modello, ottenendo così una rappresentazione snella e ottimizzata. Tuttavia, la metodologia è intrinsecamente agnostica rispetto alla geometria selezionata; qualora gli obiettivi di ricerca richiedessero una maggiore fedeltà strutturale interna, il framework consentirebbe l'integrazione di ulteriori componenti — come il sistema scheletrico o gli organi interni — semplicemente variando i nodi di interesse in fase di esportazione, a costo di un incremento dimensionale dei file `vtk` e della matrice degli snapshot, ma mantenendo invariata l'architettura logica del modello ridotto. In conclusione, il

framework costruito non si configura come una soluzione isolata, ma come una base metodologica robusta e versatile, pronta per essere impiegata in futuri studi di ottimizzazione dei dispositivi di protezione o nell'analisi di scenari di impatto complessi dove la rapidità di valutazione è un requisito imprescindibile.

APPENDICI

LISTA DEI PROGRAMMI

A.1 Script di conversione

Nella presente appendice si riporta lo script Python utilizzato per la conversione dei file binari in formato `.vtk`.

Listing A.1: Script di conversione dati

```
import os, re, json, warnings, numpy as np
from pathlib import Path
from lasso.dyna import D3plot, ArrayType

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Export LS-DYNA d3plot time states to isotopological VTK skin-shell meshes.

Goal:
- Export only shell elements belonging to THUMS skin.
- Export compact mesh: only nodes used by skin shell elements.
- Preserve isotopology over time:
    same nodes, same node order, same cells, same connectivity, same cell types.
- Write nodal point_data:
    node_id
    displacement_magnitude
    VM stress averaged from shell elements to nodes when available.
- Write cell_data:
    element_id
    part_id
    shell_vm_cell when available.
- Write a PVD collection for ParaView time series.

Requirements:
    pip install lasso-python numpy

Usage:
    python export_thums_skin_vtk_from_d3plot.py

Edit the USER SETTINGS section below.
"""

from __future__ import annotations

import os
import re
import math
import json
import warnings
from pathlib import Path
from typing import Dict, List, Tuple, Optional, Iterable

import numpy as np

try:
    from lasso.dyna import D3plot, ArrayType
except Exception as exc:
    raise ImportError(
        "Cannot import lasso-python. Install it with: pip install lasso-python"
    ) from exc


# ============================================================================
# USER SETTINGS
# ============================================================================

CASE_DIR = Path(r"./3E")
K_FILE = CASE_DIR / "main_THUMS_AM50_V7u3E.key"
D3PLOT_FILE = CASE_DIR / "d3plot"

OUT_DIR = CASE_DIR / "Thums"
```

```
63
64  # THUMS skin contact/set id found in your main key:
65  # *SET_PART_LIST
66  #   80000001 ... SKIN
67  SKIN_PART_SET_ID = 80000001
68
69  # Time sampling.
70  T_START = 0.0
71  T_END = None          # None = last d3plot time
72  DT_EXPORT = 0.005     # seconds
73
74  # Field selection:
75  # "auto"  -> export shell VM stress if available; otherwise export displacement magnitude only.
76  # "vm"    -> require shell VM stress, raise error if unavailable.
77  # "disp"  -> export displacement magnitude only.
78  FIELD_MODE = "auto"
79
80  # How to collapse shell stress over layers / integration points.
81  # "mean" or "max"
82  SHELL_STRESS_REDUCTION = "mean"
83
84  # Legacy VTK ASCII is robust and ParaView-friendly but large.
85  # For large databases keep DT_EXPORT coarse first.
86  WRITE_ASCII_VTK = True
87
88  # Write one diagnostic JSON containing available d3plot arrays and template info.
89  WRITE_DIAGNOSTICS = True
90
91
92  # ============================================================================
93  # KEYWORD PARSER
94  # ============================================================================
95
96  def strip_comment(line: str) -> str:
97      return line.split("$", 1)[0].rstrip("\n\r")
98
99
100 def is_keyword(line: str) -> bool:
101     return line.strip().startswith("*")
102
103
104 def clean_fields(line: str) -> List[str]:
105     line = strip_comment(line)
106     line = line.replace(",", " ")
107     return [x for x in line.split() if x]
108
109
110 def to_ints(fields: Iterable[str]) -> List[int]:
111     out = []
112     for f in fields:
113         try:
114             out.append(int(float(f)))
115         except Exception:
116             pass
117     return out
118
119
120 def read_key_with_includes(path: Path, already_read: Optional[set] = None) -> List[str]:
121     """
122     Read keyword file and recursively append included file contents.
123
124     Handles simple *INCLUDE cards where the next non-comment line is the include path.
125     For complex *INCLUDE_TRANSFORM workflows this may need adaptation.
126     """
127     path = Path(path).resolve()
128     if already_read is None:
129         already_read = set()
130     if path in already_read:
131         return []
132     already_read.add(path)
133
134     if not path.exists():
135         warnings.warn(f"Include/key not found: {path}")
136         return []
137
138     lines = path.read_text(errors="ignore").splitlines()
139     out = []
140
141     i = 0
142     while i < len(lines):
143         line = lines[i]
144         s = line.strip()
145         out.append(line)
146
147         if s.upper().startswith("*INCLUDE"):
148             i += 1
149             while i < len(lines):
150                 inc_line_raw = lines[i]
151                 inc_line = inc_line_raw.strip()
152                 out.append(inc_line_raw)
153
154                 if inc_line and not inc_line.startswith("$") and not inc_line.startswith("*"):
155                     inc_file = inc_line.strip().strip('"').strip("'")
156                     inc_path = (path.parent / inc_file).resolve()
157                     out.extend(read_key_with_includes(inc_path, already_read))
158                     break
159                 i += 1
160
161         i += 1
162
163     return out
```

```
164 |
165 |
166 | def find_set_part_list(lines: List[str], sid: int) -> List[int]:
167 |     """
168 |     Find *SET_PART_LIST with given sid and return all listed PIDs.
169 |     Stops at next keyword.
170 |     """
171 |     pids = []
172 |     i = 0
173 |     while i < len(lines):
174 |         s = lines[i].strip().upper()
175 |         if s.startswith("*SET_PART_LIST"):
176 |             # Next numeric non-comment line should contain sid.
177 |             j = i + 1
178 |             sid_found = None
179 |             while j < len(lines):
180 |                 sj = lines[j].strip()
181 |                 if not sj or sj.startswith("$"):
182 |                     j += 1
183 |                     continue
184 |                 if sj.startswith("*"):
185 |                     break
186 |                 vals = to_ints(clean_fields(lines[j]))
187 |                 if vals:
188 |                     sid_found = vals[0]
189 |                     break
190 |                 j += 1
191 |
192 |             if sid_found == sid:
193 |                 # Now read subsequent numeric lines until next keyword.
194 |                 j += 1
195 |                 while j < len(lines):
196 |                     sj = lines[j].strip()
197 |                     if sj.startswith("*"):
198 |                         break
199 |                     if sj and not sj.startswith("$"):
200 |                         vals = to_ints(clean_fields(lines[j]))
201 |                         # Skip the sid line if encountered by chance.
202 |                         for v in vals:
203 |                             if v > 0 and v != sid:
204 |                                 pids.append(v)
205 |                     j += 1
206 |                 break
207 |         i += 1
208 |
209 |     # Remove duplicates preserving order.
210 |     seen = set()
211 |     unique = []
212 |     for p in pids:
213 |         if p not in seen:
214 |             seen.add(p)
215 |             unique.append(p)
216 |     return unique
217 |
218 |
219 | def parse_lsdyna_int_fields(line: str, width: int = 8) -> List[int]:
220 |     """
221 |     Parse LS-DYNA integer cards.
222 |     Supports both whitespace-separated and fixed-width 8-character fields.
223 |     """
224 |     line = strip_comment(line).rstrip("\n\r")
225 |     if not line.strip():
226 |         return []
227 |
228 |     vals_split = to_ints(line.replace(",", " ").split())
229 |
230 |     vals_fixed = []
231 |     for i in range(0, len(line), width):
232 |         chunk = line[i:i + width].strip()
233 |         if not chunk:
234 |             continue
235 |         try:
236 |             vals_fixed.append(int(float(chunk)))
237 |         except Exception:
238 |             pass
239 |
240 |     # If split sees one huge concatenated number, use fixed-width fields.
241 |     if len(vals_fixed) >= 6 and (
242 |         len(vals_split) < 6 or any(abs(v) > 999999999 for v in vals_split)
243 |     ):
244 |         return vals_fixed
245 |
246 |     return vals_split
247 |
248 |
249 | def parse_shell_elements_for_pids(lines: List[str], skin_pids: set[int]) -> List[dict]:
250 |     """
251 |     Parse *ELEMENT_SHELL entries whose PID is in skin_pids.
252 |     Supports standard LS-DYNA whitespace and fixed-width formatting.
253 |     Returns list of {eid, pid, nodes, vtk_type}.
254 |     """
255 |     cells = []
256 |     current_kw = None
257 |
258 |     for raw in lines:
259 |         s = raw.strip()
260 |         if not s or s.startswith("$"):
261 |             continue
262 |
263 |         if s.startswith("*"):
264 |             current_kw = s.upper().split()[0]
```

```
            continue

        if current_kw and current_kw.startswith("*ELEMENT_SHELL"):
            vals = parse_lsdyna_int_fields(raw, width=8)

            if len(vals) < 6:
                continue

            eid = vals[0]
            pid = vals[1]

            if pid not in skin_pids:
                continue

            n = vals[2:6]
            n = [x for x in n if x > 0]

            unique_nodes = []
            for node in n:
                if node not in unique_nodes:
                    unique_nodes.append(node)

            if len(unique_nodes) == 3:
                vtk_type = 5   # VTK_TRIANGLE
            elif len(unique_nodes) == 4:
                vtk_type = 9   # VTK_QUAD
            else:
                continue

            cells.append({
                "eid": int(eid),
                "pid": int(pid),
                "nodes": [int(x) for x in unique_nodes],
                "vtk_type": int(vtk_type),
            })

    cells.sort(key=lambda c: c["eid"])
    return cells


# ============================================================================
# D3PLOT HELPERS
# ============================================================================

def array_key_name(k) -> str:
    try:
        return k.name
    except Exception:
        return str(k)


def get_array_by_tokens(d3: D3plot, all_tokens: List[str], any_tokens: Optional[List[str]] = None):
    """
    Find array whose key string contains all_tokens and at least one of any_tokens, if provided.
    Returns (key, array) or (None, None).
    """
    all_tokens = [t.lower() for t in all_tokens]
    any_tokens = [t.lower() for t in any_tokens] if any_tokens else None

    for k, arr in d3.arrays.items():
        name = array_key_name(k).lower()
        if all(t in name for t in all_tokens):
            if any_tokens is None or any(t in name for t in any_tokens):
                return k, arr
    return None, None


def get_required_array(d3: D3plot, enum_member, fallback_tokens: List[str]):
    try:
        return d3.arrays[enum_member]
    except Exception:
        _, arr = get_array_by_tokens(d3, fallback_tokens)
        if arr is None:
            raise KeyError(f"Cannot find array {enum_member} or tokens {fallback_tokens}")
        return arr


def describe_d3_arrays(d3: D3plot) -> Dict[str, dict]:
    info = {}
    for k, arr in d3.arrays.items():
        name = array_key_name(k)
        try:
            shape = tuple(arr.shape)
            dtype = str(arr.dtype)
        except Exception:
            shape = None
            dtype = str(type(arr))
        info[name] = {"shape": shape, "dtype": dtype}
    return info


def find_shell_ids_array(d3: D3plot):
    candidates = [
        ["element", "shell", "ids"],
        ["element_shell", "ids"],
        ["shell", "ids"],
    ]
    for toks in candidates:
        key, arr = get_array_by_tokens(d3, toks)
        if arr is not None:
            return key, arr
```

```
    return None, None


def find_shell_stress_array(d3: D3plot):
    # Different lasso versions use slightly different ArrayType names.
    candidates = [
        ["element", "shell", "stress"],
        ["element_shell", "stress"],
        ["shell", "stress"],
    ]
    for toks in candidates:
        key, arr = get_array_by_tokens(d3, toks)
        if arr is not None:
            return key, arr
    return None, None

def find_node_displacement_array(d3: D3plot):
    """
    Find nodal displacement array in lasso d3plot arrays.
    Needed when node_coordinates is static with shape (n_nodes, 3).
    """
    candidates = [
        ["node", "displacement"],
        ["node", "displacements"],
    ]

    for toks in candidates:
        key, arr = get_array_by_tokens(d3, toks)
        if arr is not None:
            return key, arr

    # Fallback for possible ArrayType names.
    for attr in ("node_displacement", "node_displacements"):
        try:
            enum_member = getattr(ArrayType, attr)
            if enum_member in d3.arrays:
                return enum_member, d3.arrays[enum_member]
        except Exception:
            pass

    return None, None

def von_mises_from_stress_components(stress: np.ndarray) -> np.ndarray:
    """
    Compute von Mises from last dimension containing stress components.

    Supports:
    - 6 components: sx, sy, sz, sxy, syz, sxz
    - 3 components: sx, sy, sxy, plane stress
    - already scalar: returned as absolute value
    """
    a = np.asarray(stress)

    if a.shape[-1] >= 6:
        sx = a[..., 0]
        sy = a[..., 1]
        sz = a[..., 2]
        txy = a[..., 3]
        tyz = a[..., 4]
        txz = a[..., 5]
        vm2 = 0.5 * ((sx - sy)**2 + (sy - sz)**2 + (sz - sx)**2) + 3.0 * (txy**2 + tyz**2 + txz**2)
        return np.sqrt(np.maximum(vm2, 0.0))

    if a.shape[-1] == 3:
        sx = a[..., 0]
        sy = a[..., 1]
        txy = a[..., 2]
        vm2 = sx**2 - sx * sy + sy**2 + 3.0 * txy**2
        return np.sqrt(np.maximum(vm2, 0.0))

    # scalar-like
    return np.abs(a)


def reduce_shell_vm_for_state(shell_stress: np.ndarray, state_idx: int, reduction: str) -> np.ndarray:
    """
    Convert shell stress for one state to one VM value per shell element.

    Expected shell_stress forms may be:
    - (n_states, n_shells, ..., ncomp)
    - (n_shells, ..., ncomp) if only one state loaded
    """
    arr = np.asarray(shell_stress)

    if arr.ndim >= 3 and arr.shape[0] > state_idx:
        s = arr[state_idx]
    else:
        s = arr

    vm = von_mises_from_stress_components(s)

    # vm now shape: (n_shells, extra_dims...)
    if vm.ndim == 1:
        return vm.astype(float)

    axes = tuple(range(1, vm.ndim))
    if reduction.lower() == "max":
        return np.nanmax(vm, axis=axes).astype(float)
    return np.nanmean(vm, axis=axes).astype(float)
```

```
# ============================================================================
# VTK WRITER
# ============================================================================

def write_vtk_unstructured_grid_ascii(
    path: Path,
    points: np.ndarray,
    cells_conn: List[List[int]],
    cell_types: np.ndarray,
    point_data: Dict[str, np.ndarray],
    cell_data: Dict[str, np.ndarray],
    title: str,
):
    """
    Write legacy VTK ASCII unstructured grid.
    """
    n_points = points.shape[0]
    n_cells = len(cells_conn)
    total_cell_size = sum(len(c) + 1 for c in cells_conn)

    with open(path, "w", newline="\n") as f:
        f.write("# vtk DataFile Version 3.0\n")
        f.write(title[:255] + "\n")
        f.write("ASCII\n")
        f.write("DATASET UNSTRUCTURED_GRID\n")

        f.write(f"POINTS {n_points} float\n")
        for x, y, z in points:
            f.write(f"{x:.9e} {y:.9e} {z:.9e}\n")

        f.write(f"CELLS {n_cells} {total_cell_size}\n")
        for conn in cells_conn:
            f.write(str(len(conn)) + " " + " ".join(str(int(i)) for i in conn) + "\n")

        f.write(f"CELL_TYPES {n_cells}\n")
        for ct in cell_types:
            f.write(f"{int(ct)}\n")

        if point_data:
            f.write(f"POINT_DATA {n_points}\n")
            for name, arr in point_data.items():
                arr = np.asarray(arr)
                safe_name = re.sub(r"[^A-Za-z0-9_]", "_", name)

                if arr.ndim == 1:
                    dtype = "int" if np.issubdtype(arr.dtype, np.integer) else "float"
                    f.write(f"SCALARS {safe_name} {dtype} 1\n")
                    f.write("LOOKUP_TABLE default\n")
                    if dtype == "int":
                        for v in arr:
                            f.write(f"{int(v)}\n")
                    else:
                        for v in arr:
                            f.write(f"{float(v):.9e}\n")
                elif arr.ndim == 2 and arr.shape[1] == 3:
                    f.write(f"VECTORS {safe_name} float\n")
                    for x, y, z in arr:
                        f.write(f"{float(x):.9e} {float(y):.9e} {float(z):.9e}\n")
                else:
                    warnings.warn(f"Skipping unsupported point_data {name} shape {arr.shape}")

        if cell_data:
            f.write(f"CELL_DATA {n_cells}\n")
            for name, arr in cell_data.items():
                arr = np.asarray(arr)
                safe_name = re.sub(r"[^A-Za-z0-9_]", "_", name)

                if arr.ndim == 1:
                    dtype = "int" if np.issubdtype(arr.dtype, np.integer) else "float"
                    f.write(f"SCALARS {safe_name} {dtype} 1\n")
                    f.write("LOOKUP_TABLE default\n")
                    if dtype == "int":
                        for v in arr:
                            f.write(f"{int(v)}\n")
                    else:
                        for v in arr:
                            f.write(f"{float(v):.9e}\n")
                else:
                    warnings.warn(f"Skipping unsupported cell_data {name} shape {arr.shape}")


# ============================================================================
# MAIN EXPORT
# ============================================================================

def main():
    OUT_DIR.mkdir(parents=True, exist_ok=True)

    print("[1/7] Reading keyword and skin part set...")
    lines = read_key_with_includes(K_FILE)

    skin_pids = find_set_part_list(lines, SKIN_PART_SET_ID)
    if not skin_pids:
        raise RuntimeError(
            f"No PIDs found for *SET_PART_LIST sid={SKIN_PART_SET_ID}. "
            "Check SKIN_PART_SET_ID or define skin_pids manually."
        )

    print(f"      Skin PIDs found: {len(skin_pids)}")
    print(f"      First PIDs: {skin_pids[:10]}")

```

```
    print("[2/7] Parsing skin shell elements from keyword...")
    skin_cells = parse_shell_elements_for_pids(lines, set(skin_pids))
    if not skin_cells:
        raise RuntimeError("No *ELEMENT_SHELL found for the selected skin PIDs.")

    print(f"      Skin shell cells: {len(skin_cells)}")

    # Stable compact node order: ascending LS-DYNA node id.
    used_node_ids = sorted({nid for c in skin_cells for nid in c["nodes"]})
    node_id_to_vtk = {nid: i for i, nid in enumerate(used_node_ids)}

    vtk_cells_conn = [[node_id_to_vtk[nid] for nid in c["nodes"]] for c in skin_cells]
    vtk_cell_types = np.asarray([c["vtk_type"] for c in skin_cells], dtype=np.int32)
    element_ids = np.asarray([c["eid"] for c in skin_cells], dtype=np.int64)
    part_ids = np.asarray([c["pid"] for c in skin_cells], dtype=np.int64)

    print(f"      Compact skin nodes: {len(used_node_ids)}")
    print(f"      Cell types exported: {sorted(set(vtk_cell_types.tolist()))} (5=tri, 9=quad)")

    print("[3/7] Reading d3plot...")
    d3 = D3plot(str(D3PLOT_FILE))

    # Standard arrays.
    node_ids_all = get_required_array(d3, ArrayType.node_ids, ["node", "ids"]).astype(np.int64)
    times = get_required_array(d3, ArrayType.global_timesteps, ["global", "timesteps"]).astype(float)
    coords = get_required_array(d3, ArrayType.node_coordinates, ["node", "coordinates"])
    coords = np.asarray(coords)

    node_disp_key, node_disp = find_node_displacement_array(d3)

    if coords.ndim == 2:
        if node_disp is None:
            raise RuntimeError(
                "node_coordinates has shape (n_nodes, 3), quindi contiene solo la geometria iniziale. "
                "Non trovo per  un array di displacement nodale nel d3plot."
            )

        node_disp = np.asarray(node_disp)

        if node_disp.ndim != 3 or node_disp.shape[-1] != 3:
            raise RuntimeError(
                f"Array displacement nodale trovato, ma con shape non supportata: {node_disp.shape}"
            )

        print(f"      Node coordinates shape: {coords.shape} static initial coordinates")
        print(f"      Node displacement source: {array_key_name(node_disp_key)} shape={node_disp.shape}")

    elif coords.ndim == 3:
        node_disp = None
        print(f"      Node coordinates shape: {coords.shape} time-dependent coordinates")

    else:
        raise RuntimeError(f"Unsupported node_coordinates shape: {coords.shape}")

    node_id_to_d3 = {int(nid): i for i, nid in enumerate(node_ids_all)}

    missing_nodes = [nid for nid in used_node_ids if nid not in node_id_to_d3]
    if missing_nodes:
        raise RuntimeError(
            f"{len(missing_nodes)} skin nodes are not present in d3plot node_ids. "
            f"First missing: {missing_nodes[:10]}"
        )

    skin_d3_indices = np.asarray([node_id_to_d3[nid] for nid in used_node_ids], dtype=np.int64)

    print(f"      D3plot states: {len(times)}")
    print(f"      D3plot nodes: {len(node_ids_all)}")

    print("[4/7] Looking for shell stress arrays...")
    shell_ids_key, shell_ids = find_shell_ids_array(d3)
    shell_stress_key, shell_stress = find_shell_stress_array(d3)

    use_vm = False
    skin_shell_to_d3_shell = None

    if FIELD_MODE.lower() in ("auto", "vm"):
        if shell_stress is None:
            msg = "No shell stress array found in d3plot."
            if FIELD_MODE.lower() == "vm":
                raise RuntimeError(msg)
            print("      WARNING:", msg, "Will export displacement field only.")
        elif shell_ids is None:
            msg = "Shell stress exists but no shell element ID array was found for mapping."
            if FIELD_MODE.lower() == "vm":
                raise RuntimeError(msg)
            print("      WARNING:", msg, "Will export displacement field only.")
        else:
            shell_ids = np.asarray(shell_ids).astype(np.int64)
            shell_id_to_d3 = {int(eid): i for i, eid in enumerate(shell_ids)}

            missing_shells = [int(eid) for eid in element_ids if int(eid) not in shell_id_to_d3]
            if missing_shells:
                msg = (
                    f"{len(missing_shells)} skin shell EIDs are not present in d3plot shell_ids. "
                    f"First missing: {missing_shells[:10]}"
                )
                if FIELD_MODE.lower() == "vm":
                    raise RuntimeError(msg)
                print("      WARNING:", msg, "Will export displacement field only.")
            else:
```

```
                skin_shell_to_d3_shell = np.asarray([shell_id_to_d3[int(eid)] for eid in element_ids], dtype=
                    np.int64)
                use_vm = True
                print(f"      VM source: {array_key_name(shell_stress_key)} shape={np.asarray(shell_stress).
                    shape}")
                print(f"      Shell IDs source: {array_key_name(shell_ids_key)} shape={np.asarray(shell_ids).
                    shape}")

    # Precompute element-to-node averaging graph.
    node_sum_indices = []
    elem_source_indices = []
    for eidx, conn in enumerate(vtk_cells_conn):
        for pidx in conn:
            node_sum_indices.append(pidx)
            elem_source_indices.append(eidx)
    node_sum_indices = np.asarray(node_sum_indices, dtype=np.int64)
    elem_source_indices = np.asarray(elem_source_indices, dtype=np.int64)
    node_counts = np.bincount(node_sum_indices, minlength=len(used_node_ids)).astype(float)
    node_counts[node_counts == 0.0] = 1.0

    print("[5/7] Selecting output states...")
    t_end = float(times[-1]) if T_END is None else float(T_END)
    target_times = np.arange(float(T_START), t_end + 0.5 * float(DT_EXPORT), float(DT_EXPORT))

    selected = {}
    for t_req in target_times:
        idx = int(np.argmin(np.abs(times - t_req)))
        selected[idx] = float(times[idx])

    selected_items = list(selected.items())
    print(f"      Requested states: {len(target_times)}")
    print(f"      Unique d3plot states exported: {len(selected_items)}")

    if coords.ndim == 3:
        initial_skin_points = coords[0, skin_d3_indices, :].astype(float)
    else:
        initial_skin_points = coords[skin_d3_indices, :].astype(float)

    if WRITE_DIAGNOSTICS:
        diag = {
            "k_file": str(K_FILE),
            "d3plot_file": str(D3PLOT_FILE),
            "skin_part_set_id": SKIN_PART_SET_ID,
            "n_skin_pids": len(skin_pids),
            "skin_pids": skin_pids,
            "n_skin_nodes": len(used_node_ids),
            "n_skin_shell_cells": len(skin_cells),
            "cell_types": sorted(set(vtk_cell_types.tolist())),
            "n_d3plot_states": int(len(times)),
            "n_d3plot_nodes": int(len(node_ids_all)),
            "field_mode": FIELD_MODE,
            "vm_exported": bool(use_vm),
            "shell_stress_key": array_key_name(shell_stress_key) if shell_stress_key is not None else None,
            "shell_stress_shape": tuple(np.asarray(shell_stress).shape) if shell_stress is not None else None
                ,
            "available_arrays": describe_d3_arrays(d3),
        }
        (OUT_DIR / "export_diagnostics.json").write_text(json.dumps(diag, indent=2), encoding="utf-8")

    print("[6/7] Writing VTK states...")
    pvd_entries = []

    used_node_ids_arr = np.asarray(used_node_ids, dtype=np.int64)

    for out_i, (state_idx, actual_time) in enumerate(selected_items):
        if coords.ndim == 3:
            points = coords[state_idx, skin_d3_indices, :].astype(float)
        else:
            points = (
                coords[skin_d3_indices, :].astype(float)
                + node_disp[state_idx, skin_d3_indices, :].astype(float)
            )

        disp = points - initial_skin_points
        disp_mag = np.linalg.norm(disp, axis=1)

        point_data = {
            "node_id": used_node_ids_arr,
            "displacement": disp,
            "displacement_magnitude": disp_mag,
            "time": np.full(len(used_node_ids), actual_time, dtype=float),
        }

        cell_data = {
            "element_id": element_ids,
            "part_id": part_ids,
        }

        if use_vm:
            vm_all_shells = reduce_shell_vm_for_state(shell_stress, state_idx, SHELL_STRESS_REDUCTION)
            vm_skin_cell = vm_all_shells[skin_shell_to_d3_shell]
            cell_data["shell_vm_cell"] = vm_skin_cell

            # Average element VM to connected nodes -> point data.
            node_vm_sum = np.zeros(len(used_node_ids), dtype=float)
            np.add.at(node_vm_sum, node_sum_indices, vm_skin_cell[elem_source_indices])
            node_vm = node_vm_sum / node_counts
            point_data["shell_vm_nodal_avg"] = node_vm

        vtk_name = f"DP{out_i}-Thums.vtk"
        vtk_path = OUT_DIR / vtk_name
```

```
765
766         write_vtk_unstructured_grid_ascii(
767             vtk_path,
768             points,
769             vtk_cells_conn,
770             vtk_cell_types,
771             point_data,
772             cell_data,
773             title=f"THUMS skin shell state {state_idx}, time {actual_time:.9e}",
774         )
775
776         pvd_entries.append((actual_time, vtk_name))
777         print(f"      {vtk_name}: state={state_idx}, time={actual_time:.9e}")
778
779     print("[7/7] Writing PVD series...")
780     pvd_path = OUT_DIR / "ThumsSkin_series.pvd"
781     with open(pvd_path, "w", newline="\n") as f:
782         f.write('<?xml version="1.0"?>\n')
783         f.write('<VTKFile type="Collection" version="0.1" byte_order="LittleEndian">\n')
784         f.write("  <Collection>\n")
785         for t, fname in pvd_entries:
786             f.write(f'    <DataSet timestep="{t:.9e}" group="" part="0" file="{fname}"/>\n')
787         f.write("  </Collection>\n")
788         f.write("</VTKFile>\n")
789
790     print("\nDone.")
791     print(f"Open in ParaView: {pvd_path}")
792     print(f"Diagnostics:       {OUT_DIR / 'export_diagnostics.json'}")
793     print("\nIsotopology guarantee:")
794     print("  - same compact node_id list for all states")
795     print("  - same node order for all states")
796     print("  - same shell cells/connectivity for all states")
797     print("  - only VTK cell types 5 and 9 are written")
798
799
800 if __name__ == "__main__":
801     main()
```

A.2 Script di generazione del file di input

Lo script seguente automatizza l'estrazione delle coordinate nodali e il calcolo degli angoli caratteristici (α e β), associando ogni istante temporale a una configurazione geometrica identificata da un indice progressivo univoco.

Listing A.2: Script per la generazione automatizzata del file input

```
1  import os
2  import numpy as np
3  import pandas as pd
4  from lasso.dyna import D3plot
5  from pathlib import Path
6
7
8  # ============================================================
9  # USER SETTINGS
10 # ============================================================
11
12 input_PATH = Path(r"./3E/d3plot")
13 OUT_CSV = Path(r"./3E/input.csv")
14
15 T_START = 0.0
16 T_END = None
17 DT_EXPORT = 0.005
18
19 # Scegli unit  angolare:
20 # "deg" = gradi
21 # "rad" = radianti
22 ANGLE_UNIT = "rad"
23
24 NODES = {
25     "rotula": 81079163,
26     "caviglia": 81004730,
27     "punta": 81009670,
28 }
29
30
31 # ============================================================
32 # ANGLE FUNCTIONS
33 # ============================================================
34
35 def convert_angle(angle_deg):
```

```
    """
    Converte l'angolo in base allo switch ANGLE_UNIT.
    Internamente gli angoli sono calcolati in gradi.
    """

    if ANGLE_UNIT.lower() == "deg":
        return angle_deg

    if ANGLE_UNIT.lower() == "rad":
        return np.radians(angle_deg)

    raise ValueError(
        f"ANGLE_UNIT non valido: {ANGLE_UNIT}. Usa 'deg' oppure 'rad'."
    )


def calc_angle_xz_360(v):
    """
    Angolo del vettore v proiettato sul piano X-Z.

    Convenzione:
    - 0 = direzione -X globale, cio  verso destra nell'immagine
    - positivo = verso +Z globale, cio  verso l'alto nell'immagine
    - output base in gradi [0, 360)

    Poi viene convertito in gradi o radianti tramite ANGLE_UNIT.
    """

    x_loc = -v[0]   # asse locale orizzontale positivo = -X globale
    z_loc =  v[2]   # asse locale verticale positivo = +Z globale

    angle_deg = np.degrees(np.arctan2(z_loc, x_loc)) % 360.0
    return convert_angle(angle_deg)
```

Bibliografia

[1] Toyota Motor Corporation, *THUMS User's Guide Version 7*, 2023.

[2] J. input , *Reduced Order Modeling for Biomechanics*, Journal of Engineering, 2025.

[3] Toyota Motor Corporation, *THUMS AM50 V7: Validation Report and Biofidelity Analysis*, Toyota Internal Technical Report, 2023.

[4] University of Michigan Transportation Research Institute (UMTRI), *Biomechanical Response of Human Subjects in Impact Scenarios*, Final Research Report, 2021.

[5] LS-DYNA Repository 2025-r2.