UNIVERSITÀ DI PISA

DIPARTIMENTO DI INGEGNERIA CIVILE E INDUSTRIALE
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA MECCANICA

Tesi di Laurea

Controllo reattivo delle forze di presa durante il trasferimento di un oggetto in uno scenario di manipolazione bimanuale

Candidato:
ANTONIO DI LALLO

Relatore:
Ing. MARCO GABICCINI

Anno Accademico 2014-2015

SOMMARIO

Il presente lavoro di tesi parte dall'analisi di un modello quasi-statico per la manipolazione robotica, passando in rassegna le equazioni che regolano la presa di un oggetto. Lo studio è condotto a livello del tutto generale e include il caso di power grasping e la possibilità di avere sottoattuazione nel sistema di manipolazione. Si passa poi alla simulazione dinamica del trasferimento di un oggetto da una mano all'altra in uno scenario di manipolazione bimanuale. In particolare, si prende in esame un semplice modello bidimensionale e si utilizza un algoritmo di ottimizzazione per ottenere una distribuzione pianificata delle forze di contatto che sposti gradualmente l'onere della presa dalla mano sinistra a quella destra, tenendo conto anche della eventuale presenza di un disturbo esterno. Alla pianificazione si affianca poi un sistema di controllo reattivo che permette di agire sulla posizione e sulla velocità delle dita alla luce delle informazioni ricavabili da opportuni sensori tattili.

INDICE

1 INTRODUZIONE

Nonostante si tratti di un tema classico in robotica, lo studio della destrezza nella manipolazione dei robot continua a dare impulso a una fervida attività di ricerca.

Nell'ambito di tale attività si colloca il progetto di ricerca europeo PaCMan, all'interno del quale è stata sviluppata questa tesi.

L'obiettivo che essa si prefigge è l'analisi del trasferimento di un oggetto in uno scenario di manipolazione bimanuale e la creazione di un modello matematico per la simulazione.

Si tratta di un gesto quotidiano, di per sé banale, che pratichiamo continuamente senza porvi attenzione. Dietro ad esso, però, si cela tutto un processo di valutazione delle forze di presa che noi eseguiamo inconsciamente ma che non è immediato riprodurre efficacemente in ambito robotico.

In via del tutto esemplificativa si può pensare al passaggio del testimone in una staffetta di atletica, dove tale gesto acquisisce un valore cruciale ai fini dell'esito della gara.

In particolare, nell'ambito del seguente lavoro si vuole ricavare uno strumento di ottimizzazione per la pianificazione di una distribuzione ottima delle forze di contatto dita-oggetto nell'intervallo di tempo in cui la presa passa da una mano all'altra.

Si procederà dapprima con l'esposizione di un modello quasi-statico generale per poi passare alla simulazione di un semplice modello dinamico bidimensionale.

2 PROGETTO PACMAN

PaCMan (Probabilistic and Compositional Representations of Objects for Robotic Manipulation), è un progetto di ricerca finanziato dalla Commissione Europea e portato avanti da un consorzio che raggruppa le università di Birmingham (Regno Unito), di Pisa (Italia) e di Innsbruck (Austria).

L'obiettivo è quello di sviluppare algoritmi per rendere i robot capaci di svolgere in modo affidabile semplici compiti di manipolazione su oggetti nuovi ai robot stessi. L'aggettivo "nuovi" si riferisce al fatto che gli oggetti non sono noti, ma il robot ne apprende le proprietà attraverso la visione e il contatto. Il modo di rappresentare internamente al robot le caratteristiche acquisite dell'oggetto costituisce uno dei punti chiave su cui si focalizza il progetto. Esso si basa su due idee fondamentali: scomponibilità e incertezza.

Scomponibilità significa che gli oggetti sono costituiti da una gerarchia di parti, in modo che il robot possa trasferire informazioni tra oggetti alquanto differenti, purché dotati di parti simili. Si tratta quindi di insegnare al robot a scomporre in parti i modelli degli oggetti, come vengono acquisiti dalla visione e dal contatto, e a usare le informazioni associate a una certa parte ogni volta che questa viene riconosciuta in un oggetto. Si pensi ad esempio a oggetti domestici. Essi condividono molte parti: le più semplici sono i bordi, mentre le più complesse includono manichi o in generale parti apposite per l'afferraggio.

Per quanto riguarda l'incertezza, invece, essa ha una duplice importanza: da un lato il robot ha bisogno di conoscere l'incertezza legata alla posizione e alla forma dell'oggetto, dall'altro lato deve sapere come questa incertezza può condizionare le sue azioni pianificate.

Questi due aspetti sono dunque combinati e testati su dei robot, prendendo come problema test l'operazione di caricamento di una lavastoviglie.

In sostanza, quindi, il progetto PaCMan si propone di rendere più robusta la manipolazione robotica di oggetti, familiari e non. Il robot deve essere in grado di

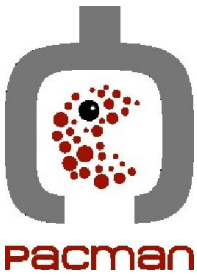
ottenere autonomamente informazioni sull'oggetto ricavandole dalla visione e dal contatto, deve saper pianificare di conseguenza le sue attività, e verificare in tempo reale l'andamento delle sue azioni.

Per raggiungere tale obiettivo il progetto si articola in 7 Work Packages (WP), che sono illustrati nella tabella sottostante:

	Work Packages	Lead
WP1	Learning and Estimation of Compositional Representations of Objects from Visual Data	University of Birmingham
WP2	Multi-modal Compositional Representations of Objects	Universitaet Innsbruck
WP3	Active Haptic and Visual Information Gathering Under Uncertainty	University of Birmingham
WP4	Grasping Under Uncertainty	Universita di Pisa
WP5	Scenario Based Evaluation	Universitaet Innsbruck
WP6	Management	University of Birmingham
WP7	Dissemination and Exploitation	Universita di Pisa

Tabella 1: Piano di lavoro del progetto PaCMan

In particolare il presente lavoro si colloca nell'ambito del WP4, volto a sviluppare metodi di controllo delle azioni di presa, basati sul rilevamento dell'oggetto tramite opportuni sensori tattili.



3 MODELLO QUASI-STATICO DEL GRASP

Come assunzione di base si considera ogni elemento come un corpo rigido, in modo da poter sfruttare tutti gli strumenti matematici sviluppati per descrivere il moto di un corpo rigido. Tuttavia, la componente elastica del sistema viene salvaguardata introducendo, ad esempio, molle virtuali per descrivere la cedevolezza del contatto tra i polpastrelli e l'oggetto afferrato.

In questo modo, dunque, è possibile separare la componente rigida del sistema da quella elastica, così da facilitare la trattazione matematica senza precludere la possibilità di considerare oggetti localmente deformabili.

Le dita che compongono la mano risultano essere tutte cinematicamente uguali ma posizionate diversamente rispetto al palmo. Pertanto, sarà sufficiente studiare la cinematica di un singolo dito, schematizzabile come una catena seriale, e poi riportarne l'origine nella posizione desiderata rispetto al palmo attraverso una matrice di rototraslazione.

Per descrivere completamente la cinematica di un corpo rigido è sufficiente conoscere la velocità di un suo punto qualsiasi P e la sua velocità angolare $\underline{\omega}$. Queste informazioni possono essere raccolte in un'unica variabile detta twist, o velocità generalizzata, $\underline{\xi}_P = \begin{bmatrix} \underline{v}_P \\ \underline{\omega} \end{bmatrix}$.

In maniera analoga, è conveniente definire una variabile che colleziona le informazioni sulle risultanti di forza e momento (rispetto a un polo generico P) in un'unica scrittura; tale grandezza prende il nome di wrench, o forza generalizzata, ed è indicata come $\underline{w}_P = \begin{bmatrix} \underline{F} \\ \underline{M}_P \end{bmatrix}$.

Si osservi che twist e wrench sono grandezze vettoriali formate da componenti fisicamente disomogenee.

Per il resto del capitolo si omette la sottolineatura per le grandezze vettoriali.

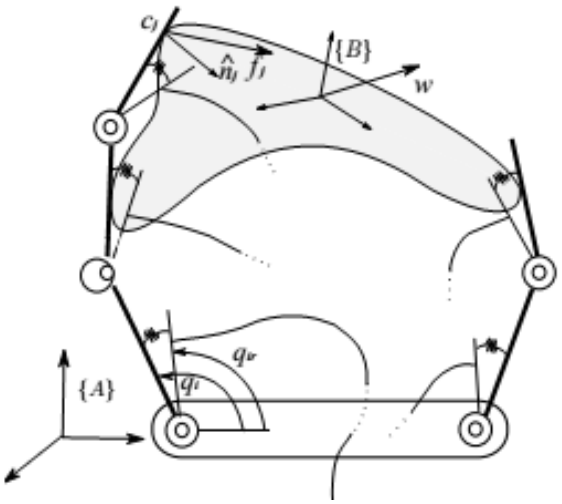


Figura 3.1: Modello schematico di mano robotica

Con riferimento alla Figura 3.1, introduciamo i seguenti sistemi di riferimento:

$\{A\} = \{O_a; x_a, y_a, z_a\}$ frame inerziale solidale al palmo della mano

$\{B\} = \{O_b; x_b, y_b, z_b\}$ frame solidale all'oggetto afferrato

$\{C_i^h\} = \{O_{c_i^h}; x_{c_i^h}, y_{c_i^h}, z_{c_i^h}\}$ frame attaccato al dito in corrispondenza dell' *i-esimo* punto di contatto

$\{C_i^o\} = \{O_{c_i^o}; x_{c_i^o}, y_{c_i^o}, z_{c_i^o}\}$ frame attaccato all'oggetto in corrispondenza dell'*i-esimo* punto di contatto

Inoltre, si utilizza la seguente notazione:

ξ_{xy}^z twist del frame $\{Y\}$ rispetto a $\{X\}$, descritto da un osservatore solidale a $\{Z\}$

w_x^y wrench che il frame $\{X\}$ esercita sul frame $\{Y\}$ espresso in componenti $\{Y\}$

$g_{tz} \in SE(3)$ postura del frame $\{Z\}$ rispetto a $\{T\}$

Si definisce poi la matrice aggiunta $Ad_g \in \mathbb{R}^{6\times6}: se(3) \to se(3)$, come un operatore in grado di trasformare i twist secondo la legge

$$\xi_{xy}^t = Ad_{g_{tz}} \xi_{xy}^z, \tag{1}$$

e la matrice co-aggiunta $Ad_g^{-T} \in \mathbb{R}^{6\times6}: se^*(3) \to se^*(3)$, che trasforma i wrench in maniera analoga [1]:

$$w_x^t = Ad_{g_{ty}}^{-T} w_x^y. \tag{2}$$

Fatte le dovute premesse si può ora passare ad analizzare le equazioni che regolano il grasp, ovvero la presa:

- Equazione di equilibrio dell'oggetto
- Equazione di congruenza dell'oggetto
- Equazione di equilibrio della mano
- Equazione di congruenza della mano
- Modello di interazione mano/oggetto

L'unione di tutte queste equazioni permetterà poi di definire quella che è l'Equazione Fondamentale del Grasp (FGE).

3.1 Equazione di equilibrio dell'oggetto

Indichiamo con $w_e^b \in \mathbb{R}^6$ il wrench esterno che agisce sull'oggetto, con componenti espresse in $\{B\}$, e con $w_{c_i^h}^{c_i^o}$ il wrench che la mano esercita sull'oggetto in corrispondenza dell'*i-esimo* punto di contatto, con componenti in $\{C_i^o\}$. L'equilibrio dell'oggetto richiede che la somma di tutti i contributi di forza e momento sia nulla. Esprimendo le componenti in $\{B\}$, tale condizione può essere scritta come

$$w_e^b + \sum_{i=1}^{p} Ad_{g_{bc_i^o}}^{-T} w_{c_i^h}^{c_i^o} = 0 \qquad (3)$$

dove *p* è il numero dei punti di contatto. Tenendo conto della natura del tipo di contatto, tuttavia, è possibile descrivere l'interazione di contatto con un vettore $f_{c_i^h}^{c_i^o} \in \mathbb{R}^{c_i}$, con $c_i \leq 6$, al posto del wrench completo, in modo da prendere in considerazione le sole direzioni vincolate. Più in dettaglio per ogni punto di contatto si introduce una matrice $H_i \in \mathbb{R}^{6\times c_i}$ che mappa l'interazione locale nel wrench completo come segue:

$$w_{c_i^h}^{c_i^o} = H_i f_{c_i^h}^{c_i^o} \qquad (4)$$

A seconda del tipo di contatto la matrice H_i espressa in terna locale può assumere diverse forme, come si vedrà successivamente.

Sostituendo (4) in (3) si ottiene

$$w_e^b + \sum_{i=1}^{p} Ad_{g_{bc_i^o}}^{-T} H_i f_{c_i^h}^{c_i^o} = 0, \qquad (5)$$

e definendo la matrice di Grasp nel frame $\{B\}$, ${}^{b}G \in \mathbb{R}^{6\times c}$, come

$$ {}^{b}G = \left[Ad_{g_{bc_1^o}}^{-T} H_1 \ \cdots \ Ad_{g_{bc_i^o}}^{-T} H_i \ \cdots \ Ad_{g_{bc_p^o}}^{-T} H_p \right], \tag{6} $$

con $c = \sum_{i=1}^{p} c_i$ pari al numero totale dei vincoli di contatto che effettivamente agiscono sull'oggetto, si giunge finalmente all'equazione di equilibrio dell'oggetto nella forma

$$ w_e^b + {}^{b}G f_{c^h}^{c^o} = 0, \tag{7} $$

dove $f_{c^h}^{c^o} = \left[f_{c_1^h}^{c_1^{oT}}, \cdots, f_{c_i^h}^{c_i^{oT}}, \cdots, f_{c_p^h}^{c_p^{oT}} \right]^T \in \mathbb{R}^c$ è il vettore che colleziona tutte le azioni esercitate dalla mano sull'oggetto. Si noti che se i punti di contatto dell'oggetto restano fissi, la matrice di Grasp nel sistema di riferimento dell'oggetto è costante. Di conseguenza è immediato calcolare il differenziale di (7), trovando una relazione che lega piccole perturbazioni delle variabili coinvolte. In altri termini è possibile scrivere l'equazione di equilibrio perturbato dell'oggetto come

$$ \delta w_e^b + {}^{b}G \delta f_{c^h}^{c^o} = 0. \tag{8} $$

3.2 Equazione di congruenza dell'oggetto

Il twist dell'*i-esimo* frame di contatto sull'oggetto $\{C_i^o\}$ può essere espresso in funzione del twist del frame dell'oggetto {B} per mezzo della matrice aggiunta:

$$ \xi_{ab}^{c_i^o} = Ad_{g_{c_i^o b}} \xi_{ab}^b. \tag{9} $$

Inoltre, in genere i vincoli di contatto non interessano tutte le direzioni di spostamento. Più precisamente si verifica che la matrice ${H_i}^T$ è in grado di selezionare dal twist completo le sole componenti di velocità che violano i vincoli di contatto [2]. In formule questo legame si esprime come

$$ v_{ab}^{c_i^o} = {H_i}^T \xi_{ab}^{c_i^o} = {H_i}^T Ad_{g_{c_i^o b}} \xi_{ab}^b. \tag{10} $$

A questo punto, definendo $v_{ab}^{c^o} = \left[v_{ab}^{c_1^{oT}}, \cdots, v_{ab}^{c_i^{oT}}, \cdots, v_{ab}^{c_p^{oT}} \right]^T \in \mathbb{R}^c$ e ricordando la definizione della matrice di Grasp in (6), è immediato ricavare

$$ v_{ab}^{c^o} = {}^{b}G^T \xi_{ab}^b. \tag{11} $$

Un ulteriore passo avanti si può fare introducendo una catena cinematica virtuale per descrivere la configurazione dell'oggetto rispetto al frame assoluto $\{A\}$. In questo

modo il Jacobiano della catena virtuale può essere usato per esprimere il twist dell'oggetto in funzione di un vettore $u \in \mathbb{R}^6$, capace di parametrizzare $SE(3)$, come segue:

$$\xi_{ab}^{b} = J_o(u)\dot{u}. \tag{12}$$

Pertanto, sostituendo (12) in (11) e definendo la variazione del frame di contatto dell'oggetto come $\delta C_{ab}^{c^o} = v_{ab}^{c^o} dt$ l'equazione di congruenza dell'oggetto diventa

$$\delta C_{ac^o}^{c^o} = {}^{b}G^{T} J_o(u)\delta u. \tag{13}$$

3.3 Equazione di congruenza della mano

Ciascun dito della mano si schematizza come un braccio seriale e quindi per il generico frame *i-esimo* di contatto della mano il twist risulta

$$\xi_{ac_i^h}^{a} = {}^{a}J_i(q_i)\dot{q}_i \tag{14}$$

dove si è indicato con ${}^{a}J_i(q_i) \in \mathbb{R}^{6\times \#q_i}$ il Jacobiano spaziale della catena cinematica seriale relativa all'*i-esimo* punto di contatto della mano e con $\#q_i$ il numero di giunti che precedono il punto stesso.

Successivamente conviene riportare il twist nel sistema di riferimento del punto di contatto dell'oggetto, ossia, usando la matrice aggiunta,

$$\xi_{ac_i^h}^{c_i^o} = Ad_{g_{c_i^o a}(u)} \xi_{ac_i^h}^{a} = Ad_{g_{c_i^o a}(u)}\ {}^{a}J_i(q_i)\dot{q}_i. \tag{15}$$

A questo punto è possibile selezionare, tramite la matrice ${H_i}^T$, le componenti della velocità nelle direzioni che violano i vincoli di contatto, ottenendo

$$v_{ac_i^h}^{c_i^o} = {H_i}^T Ad_{g_{c_i^o a}(u)}\ {}^{a}J_i(q_i)\dot{q}_i = {}^{c_i^o}J_i(q_i, u)\dot{q}_i. \tag{16}$$

Dunque, mettendo insieme le equazioni corrispondenti a ciascun punto di contatto, si ricava

$$v_{ac^h}^{c^o} = {}^{c^o}J(q, u)\dot{q}, \tag{17}$$

dove si sono utilizzate le seguenti grandezze:

- $v_{ac^h}^{c^o} = \left[{v_{ac_1^h}^{c_1^o}}^{T}, \cdots, {v_{ac_i^h}^{c_i^o}}^{T}, \cdots, {v_{ac_p^h}^{c_p^o}}^{T} \right]^T \in \mathbb{R}^{\mathrm{c}}$;
- $q \in \mathbb{R}^{\#q}$, vettore che colleziona tutte le variabili di giunto;
- ${}^{c^o}J(q, u)$, Jacobiano dell'intera mano, riferito ai frame di contatto dell'oggetto.

Infine, moltiplicando per dt, l'equazione di congruenza della mano diventa

$$\delta C_{ac^h}^{c^o} = {}^{c^o}J(q, u)\delta q. \tag{18}$$

3.4 Equazione di equilibrio della mano

Sfruttando la dualità cineto-statica è immediato ricavare l'equazione di equilibrio della mano, che permette di mappare le forze esercitate dalla mano sull'oggetto $f_{c^h}^{c^o}$ nelle coppie ai giunti $\tau \in \mathbb{R}^{\#q}$:

$$\tau = {}^{c^o}J^T(q,u) f_{c^h}^{c^o}. \tag{19}$$

Inoltre, differenziando la (19) è possibile descrivere piccole perturbazioni del sistema come

$$\delta\tau = Q\delta q + U\delta u + {}^{c^o}J^T \delta f_{c^h}^{c^o}, \tag{20}$$

avendo definito le matrici

- $Q = \frac{\partial {}^{c^o}J^T(q,u) f_{c^h}^{c^o}}{\partial q} \in \mathbb{R}^{\#q \times \#q}$;
- $U = \frac{\partial {}^{c^o}J^T(q,u) f_{c^h}^{c^o}}{\partial u} \in \mathbb{R}^{\#q \times 6}$.

Si noti che in una configurazione di equilibrio i termini Q e U diventano trascurabili se le forze di contatto iniziali sono piccole. Inoltre esiste un metodo per calcolarli a partire dal Jacobiano spaziale e dalle matrici aggiunte, senza bisogno di ricorrere al calcolo simbolico [3].

3.5 Modello di interazione mano/oggetto

Il contatto tra la mano e l'oggetto può essere interpretato come un vincolo di moto. A seconda delle direzioni vincolate si distinguono diverse tipologie di contatto [4]:

- *frictionless contact point*, in cui solo la direzione normale alla superficie nel punto di contatto è vincolata e il vettore $f_{c_i^h}^{c_i^o}$ ha dimensione unitaria ($c_i = 1$);

- *point contact with friction* or *hard finger*, che permette la presenza di una forza generica a tre componenti, ma non di un momento, con $c_i = 3$;

- *soft finger*, che rispetto all'hard finger consente in più la possibilità di trasmettere un momento (di spin) attorno alla normale alla superficie nel punto di contatto, equivalente a un vettore $f_{c_i^h}^{c_i^o}$ di dimensione quattro.

In Tabella 2 si mostra un riepilogo delle varie tipologie di contatto dita/oggetto con le basi del twist relativo permesso e del wrench esercitabile dal vincolo, espresse entrambe in terna locale.

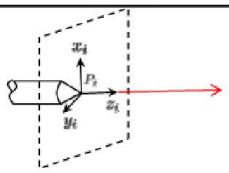
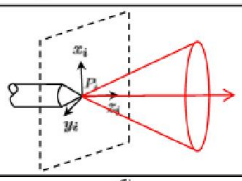
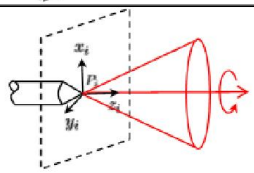
Tipologia	Schema	Base twist relativo (cols)	Base wrench (rows)
Frictionless point contact		${}^{i}F_i = \left[\begin{array}{c:cc:cc} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{array}\right]$	${}^{i}H_i = \left[\begin{array}{cc:c:cc} 0 & 0 & 1 & 0 & 0 & 0 \end{array}\right]$
Point contact with friction		${}^{i}F_i = \left[\begin{array}{c:c:c} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \hdashline 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}\right]$	${}^{i}H_i = \left[\begin{array}{cc:c:cc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 1 & 0 & 0 & 0 \end{array}\right]$
Soft-finger		${}^{i}F_i = \left[\begin{array}{c:c} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \hdashline 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{array}\right]$	${}^{i}H_i = \left[\begin{array}{cc:c:cc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right]$

Tabella 2: Tipi di vincoli di contatto

In genere il numero totale dei vincoli è maggiore del numero dei gradi di libertà dell'oggetto (sei nel caso 3D e tre nel caso piano), vale a dire che il problema è iperstatico o staticamente indeterminato. Sotto questa condizione le equazioni di equilibrio non sono sufficienti a determinare univocamente le forze di contatto. Per risolvere questo problema, quindi, è necessario fare ricorso a delle equazioni costitutive: si assume un modello lineare per la forza di contatto, in modo che questa sia direttamente proporzionale all'interferenza tra la mano e l'oggetto, come se si introducessero delle molle virtuali nei contatti. Più specificamente, per l'*i-esimo* punto di contatto si introduce una matrice di rigidezza $K_{c_i} \in \mathbb{R}^{c_i \times c_i}$ e la forza che la mano esercita sull'oggetto può essere espressa come

$$\delta f_{c_i^h}^{c_i^o} = K_{c_i}(\delta C_{ac_i^h}^{c_i^o} - \delta C_{ac_i^o}^{c_i^o}). \tag{21}$$

Mettendo tutto insieme e definendo la matrice $K_c = \text{blkdiag}(K_{c_1}, \dots, K_{c_p}) \in \mathbb{R}^{c \times c}$, l'equazione costitutiva delle forze di contatto diventa

$$\delta f_{c^h}^{c^o} = K_c(\delta C_{ac^h}^{c^o} - \delta C_{ac^o}^{c^o}). \tag{22}$$

Si osservi che nessuna specificazione è stata fatta sui punti di contatto, che perciò non coincidono per forza con le estremità delle dita, ma possono essere localizzati anche sulle falangi medie o prossimali o ancora sul palmo della mano. Anzi l'idea di sfruttare tutte le parti del sistema di manipolazione è un modo di accrescerne ulteriormente le capacità di presa e le potenzialità di applicazione. Essa viene direttamente dall'osservazione di esempi umani e animali, che dimostrano l'utilità di

tale manipolazione cosiddetta *whole-limb* (o *power grasping*) in natura, consentendo una maggiore robustezza e destrezza.

3.6 L'Equazione Fondamentale del Grasp

Le equazioni analizzate sopra sono in grado di descrivere una approssimazione lineare di tutti i possibili spostamenti del sistema nell'intorno di una configurazione di equilibrio. Assemblandole tutte insieme in forma matriciale si ottiene la cosiddetta Equazione Fondamentale del Grasp (GFE):

$$\begin{bmatrix} I_{\#f} & 0 & 0 & -K_c & K_c & 0 & 0 \\ -{}^{c^o}\bar{J}^T & I_{\#\tau} & -\bar{U} & 0 & 0 & 0 & -\bar{Q} \\ 0 & 0 & -{}^{b}GK_c\,{}^{b}G^T\bar{J}_o & 0 & 0 & I_{\#w} & {}^{b}GK_c\,{}^{c^o}\bar{J} \\ 0 & 0 & 0 & I_{\#C^h} & 0 & 0 & -{}^{c^o}\bar{J} \\ 0 & 0 & -{}^{b}G^T\bar{J}_o & 0 & I_{\#C^o} & 0 & 0 \end{bmatrix} \begin{bmatrix} \delta f_{c^h}^{c^o} \\ \delta\tau \\ \delta u \\ \delta C_{ac^h}^{c^o} \\ \delta C_{ac^o}^{c^o} \\ \delta w_e^b \\ \delta q \end{bmatrix} = 0, \qquad (23)$$

dove si è indicato con $\bar{\nu}$ il valore della variabile ν nel punto di equilibrio iniziale, e con $I_{\#\nu}$ una matrice identica di dimensione $\#\nu$.

Si tratta dunque di un sistema lineare omogeneo nella forma $A\delta y = 0$, in cui la matrice dei coefficienti $A \in \mathbb{R}^{r_a \times c_a}$ prende il nome di Matrice Fondamentale del Grasp (FGM), e il vettore $y \in \mathbb{R}^{c_a}$, che colleziona tutte le variabili in gioco, è la configurazione aumentata.

Per la sua risoluzione esistono diversi metodi, a seconda che si voglia prediligere l'efficacia dal punto di vista numerico o il senso fisico, attraverso uno studio in forma simbolica [3].

Infatti, l'analisi della FGM permette già di ricavare importanti proprietà del grasp, basate sull'esistenza di sottoinsiemi rilevanti di soluzione, attraverso lo studio dello spazio nullo. È possibile verificare, ad esempio, la presenza di forze interne e forze strutturali esterne, la possibilità di casi di pure squeeze e di moti ridondanti della mano, nonché di spostamenti labili dell'oggetto [5]. Una proprietà fondamentale che in genere si richiede alla presa è la cosiddetta *force closure*, indice di robustezza del grasp. Essa infatti consiste nella capacità del robot di impedire moti dell'oggetto in qualsiasi direzione a dispetto di forze esterne applicate e quindi è connessa alla capacità dello stesso manipolatore di esercitare forze arbitrarie attraverso i contatti [6].

3.7 Modello di attuazione dei giunti

Il modello fin qui proposto può essere reso ancora più completo e complesso prevedendo anche un modello di attuazione dei giunti.

Ad esempio per il giunto *i-esimo* introduciamo un comportamento elastico che lega la configurazione reale q_i a un valore di riferimento $q_{r_i} \in \mathbb{R}$, descritto da una molla a torsione con costante di rigidezza $k_{q_i} \in \mathbb{R}$:

$$\tau_i = k_{q_i}(q_{r_i} - q_i). \tag{24}$$

Definendo $K_q = diag(k_{q_i}) \in \mathbb{R}^{\#q \times \#q}$ come la matrice che colleziona tutte le rigidezze di giunto e $q_r \in \mathbb{R}^{\#q}$ come il vettore delle variabili di riferimento dei giunti, per differenziazione possiamo scrivere la legge di attuazione quasi-statica come

$$\delta\tau = K_q(\delta q_r - \delta q). \tag{25}$$

Aggiungendo anche la (25) al sistema (23), l'Equazione Fondamentale del Grasp che descrive il caso di una mano con giunti elastici diventa

$$\begin{bmatrix} I_{\#f} & 0 & 0 & -K_c & K_c & 0 & 0 & 0 \\ -{}^{c^o}\bar{J}^T & I_{\#\tau} & -\bar{U} & 0 & 0 & -\bar{Q} & 0 & 0 \\ 0 & 0 & -{}^{b}GK_c\,{}^{b}G^T\bar{J}_o & 0 & 0 & {}^{b}GK_c\,{}^{c^o}\bar{J} & I_{\#w} & 0 \\ 0 & 0 & 0 & I_{\#C^h} & 0 & -{}^{c^o}\bar{J} & 0 & 0 \\ 0 & 0 & -{}^{b}G^T\bar{J}_o & 0 & I_{\#C^o} & 0 & 0 & 0 \\ 0 & I_{\#\tau} & 0 & 0 & 0 & K_q & 0 & -K_q \end{bmatrix} \begin{bmatrix} \delta f_{c^h}^{c^o} \\ \delta\tau \\ \delta u \\ \delta C_{ac^h}^{c^o} \\ \delta C_{ac^o}^{c^o} \\ \delta q \\ \delta w_e^b \\ \delta q_r \end{bmatrix} = 0 \tag{26}$$

3.8 Sottoattuazione

Si noti che finora la trattazione è avvenuta nell'ipotesi di mano completamente attuata, che dispone cioè di un attuatore per ogni giunto. Tuttavia, per quanto desiderabile, questa non è una pista molto praticabile, né tantomeno naturale se si prende a modello la struttura anatomica dell'uomo, in cui è affidata ai tendini la funzione di trasmettere alle dita la forza esercitata dai muscoli del braccio. La remotizzazione degli attuatori ha infatti il vantaggio di alleggerire gli end-effectors, consentendo di raggiungere alte prestazioni dinamiche. Per questo motivo sono sorti numerosi progetti di mani robotiche basate su un sistema tendineo e sono stati proposti diversi approcci alla modellazione e al controllo di tali sistemi bio-ispirati [7].

Tutti questi studi hanno in comune l'introduzione di un certo grado di sottoattuazione, che comporta conseguenze interessanti a livello del grasp [8].

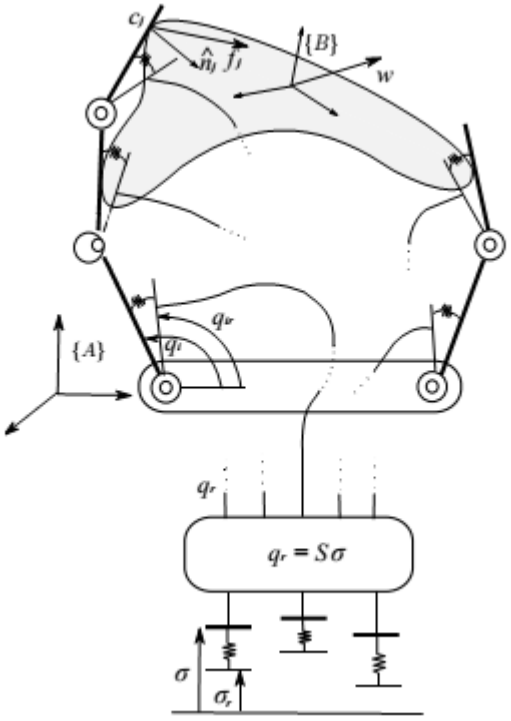


Figura 3.2: Modello schematico di mano robotica con sottoattuazione sinergica

Dunque, per inserire quest'altro aspetto nel modello imponiamo che le posizioni di riferimento dei giunti $q_r \in \mathbb{R}^{\#q}$ siano funzione di alcuni valori di sinergia posturale $\sigma \in \mathbb{R}^{\#\sigma}$, con $\#\sigma \leq \#q$, cioè

$$q_r = f(\sigma), \tag{27}$$

dove $f: \mathbb{R}^{\#\sigma} \to \mathbb{R}^{\#q}$ è la funzione di sinergia. Calcolando il differenziale di (27), e definendo la matrice di sinergia come $S(\sigma) = \frac{\partial f(\sigma)}{\partial \sigma} \in \mathbb{R}^{\#q \times \#\sigma}$, gli spostamenti di riferimento dei giunti sono descritti come segue:

$$\delta q_r = S(\sigma)\delta\sigma. \tag{28}$$

Di nuovo, in virtù della dualità cineto-statica, possiamo trovare che il vettore $\eta \in \mathbb{R}^{\#\sigma}$ delle forze generalizzate al livello di sinergia è legato alle coppie ai giunti dalla relazione

$$\eta = S^T(\sigma)\tau. \tag{29}$$

Differenziando (29) e introducendo la matrice $\Sigma(\sigma, \tau) = \frac{\partial S^T(\sigma)\tau}{\partial \sigma} \in \mathbb{R}^{\#\sigma \times \#\sigma}$, per le forze di attuazione otteniamo

$$\delta\eta = S^T(\sigma)\delta\tau + \Sigma(\sigma, \tau)\delta\sigma. \tag{30}$$

Per generalità possiamo introdurre un modello elastico anche per la sottoattuazione sinergica, similmente a quanto fatto per i giunti. Definendo la matrice di rigidezza di sinergia come la matrice $K_\sigma = diag(k_{\sigma_i}) \in \mathbb{R}^{\#\sigma \times \#\sigma}$, e introducendo il vettore dei valori di riferimento di sinergia $\sigma_r \in \mathbb{R}^{\#\sigma}$, la legge di attuazione risulta

$$\delta\eta = K_\sigma(\delta\sigma_r - \delta\sigma). \qquad (31)$$

In conclusione, è possibile inglobare anche quest'ultimo aspetto nell'Equazione Fondamentale del Grasp $A\delta y = 0$: per una mano con giunti elastici e con sottoattuazione sinergica l'Equazione Fondamentale del Grasp assume la forma

$$\begin{bmatrix} I_{\#f} & 0 & 0 & 0 & -K_c & K_c & 0 & 0 & 0 & 0 & 0 \\ -{}^{c^o}\bar{J}^T & I_{\#\tau} & 0 & -\bar{U} & 0 & 0 & -\bar{Q} & 0 & 0 & 0 & 0 \\ 0 & -\bar{S}^T & I_{\#\eta} & 0 & 0 & 0 & 0 & 0 & -\bar{\Sigma} & 0 & 0 \\ {}^{b}G & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I_{\#w} & 0 \\ 0 & 0 & 0 & 0 & I_{\#C^h} & 0 & -{}^{c^o}\bar{J} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -{}^{b}G^T\bar{J}_o & 0 & I_{\#C^o} & 0 & 0 & 0 & 0 & 0 \\ 0 & I_{\#\tau} & 0 & 0 & 0 & 0 & K_q & -K_q & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & I_{\#q_r} & -\bar{\mathrm{S}} & 0 & 0 \\ 0 & 0 & I_{\#\sigma} & 0 & 0 & 0 & 0 & 0 & K_\sigma & 0 & -K_\sigma \end{bmatrix} \begin{bmatrix} \delta f_{c^h}^{c^o} \\ \delta\tau \\ \delta\eta \\ \delta u \\ \delta C_{ac^h}^{c^o} \\ \delta C_{ac^o}^{c^o} \\ \delta q \\ \delta q_r \\ \delta\sigma \\ \delta w_e^b \\ \delta\sigma_r \end{bmatrix} = 0. \qquad (32)$$

4 OTTIMIZZAZIONE QUASI-STATICA DELLA DISTRIBUZIONE DELLE FORZE DI PRESA

Finora abbiamo visto come si interfaccia con l'oggetto la singola mano, ma il punto cruciale della trattazione è rappresentato dal processo in cui la presa passa da una mano all'altra. Vogliamo che le forze di contatto della mano sinistra, inizialmente da sola in presa, decrescano gradualmente fino ad annullarsi, mentre quelle della mano destra crescano in maniera complementare, consentendo alla fine di poter disimpegnare la mano sinistra e affidare completamente la presa alla sola mano destra. Si tratta di un processo che per un certo verso è simile al regrasping, ovvero a quelle applicazioni in cui al manipolatore è richiesto di disimpegnare e rimpegnare le dita per eseguire certi compiti [9]. Per esempio, si pensi all'operazione di avvitare una lampadina in una presa: è chiaro che per eseguirla c'è bisogno di sganciare la mano per poi riafferrare la lampadina. Durante il trasferimento dell'oggetto, dunque, come per il regrasping, deve essere ancora fornito il wrench (risultante e momento risultante) desiderato e devono essere rispettati tutti i vincoli relativi ai coni d'attrito statico. Inoltre, le forze di contatto devono essere continue dappertutto, altrimenti l'integrità del grasp può essere compromessa.

4.1 Ottimizzazione in condizioni nominali

Affinché il passaggio dell'oggetto avvenga stabilmente, quindi, è necessario che ad ogni istante sia verificato l'equilibrio dell'oggetto:

$$G\underline{f} + \underline{w} = \underline{0}, \qquad (33)$$

dove si è indicato con $\underline{w}$ il wrench esterno, che nel nostro caso è rappresentato dalla sola forza peso, con $\underline{f}$ il vettore delle forze di contatto dita-oggetto relative a entrambe le mani, e con G la matrice di Grasp, che condensa l'effetto delle forze di contatto rispetto a un unico polo. Nel nostro caso abbiamo preso come polo l'origine O_a del

sistema di riferimento assoluto, perché questa scelta garantisce che l'espressione della matrice di Grasp resti costante al variare dei punti di contatto dita-oggetto.

Utilizzando il modello hard finger, ciascun dito può esercitare una componente di forza normale f_{i_n}, non negativa, e una tangenziale f_{i_t}; tuttavia le forze di contatto non possono assumere qualsiasi direzione, ma sono vincolate ad essere all'interno dei coni d'attrito, cioè deve valere

$$f_{i_n} \geq 0 \qquad \forall i, \tag{34}$$

$$\mu_i f_{i_n} \geq |f_{i_t}| \qquad \forall i, \tag{35}$$

dove μ_i è il coefficiente di attrito tra il dito e l'oggetto in corrispondenza dell'*i-esimo* punto di contatto. Per semplicità assumiamo che si abbia lo stesso coefficiente di attrito μ per ogni contatto, e che quello dinamico sia uguale a quello statico.

Inoltre, affinché il passaggio sia graduale, introduciamo un limite superiore f_{lim} alla componente normale delle forze di contatto e imponiamo che per la mano sinistra questo limite decresca linearmente a partire dall'istante t_0 fino ad annullarsi all'istante t_1, mentre per la mano destra esso cresca nello stesso intervallo di tempo con pendenza opposta, come riportato nella Figura 4.1.

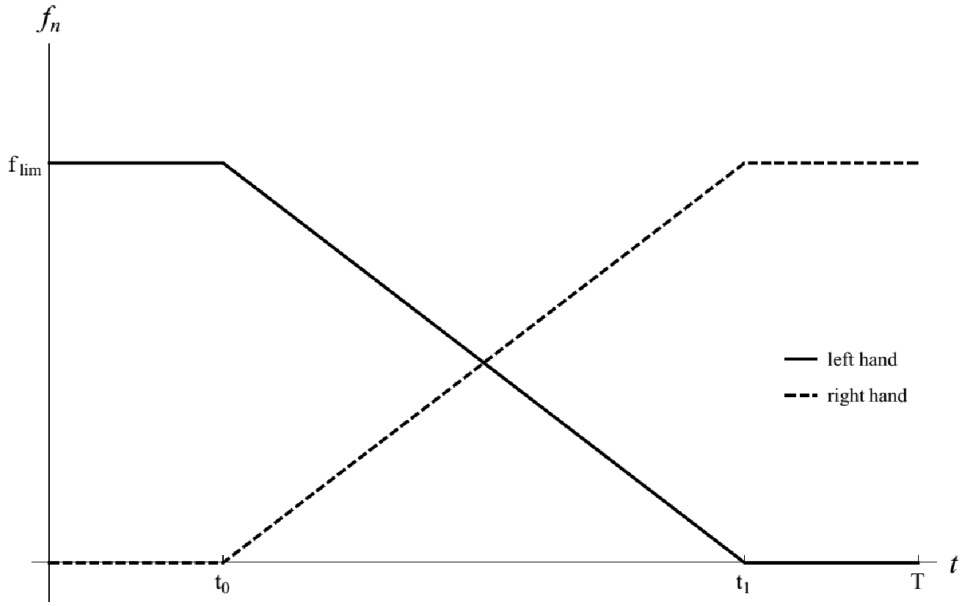


Figura 4.1: Andamento del valore massimo ammissibile delle forze di presa

Quindi, pensando di applicare il controllo a un istante di tempo congelato τ, tale condizione si traduce in disequazioni del tipo

$$f_{i_n}(\tau) \leq f_{\lim} \cdot g_h(\tau) \qquad \forall i; h \in \{\text{right}, \text{left}\}, \tag{36}$$

con g_h definita diversamente per le due mani:

$$g_{\text{left}}(\tau) = \begin{cases} 1 & \tau \le t_0 \\ \dfrac{\tau - t_0}{t_1 - t_0} & t_0 < \tau < t_1 \\ 0 & \tau \ge t_1 \end{cases} \tag{37}$$

$$g_{\text{right}}(\tau) = \begin{cases} 0 & \tau \le t_0 \\ 1 - \dfrac{\tau - t_0}{t_1 - t_0} & t_0 < \tau < t_1 \\ 1 & \tau \ge t_1 \end{cases} \tag{38}$$

Si osserva che in questo modo ad ogni istante risulta

$$g_{\text{right}}(\tau) + g_{\text{left}}(\tau) = 1 \tag{39}$$

cioè le (50) e (51) sono due funzioni complementari a 1.

Mettendo tutto insieme e scegliendo di minimizzare la norma 2 delle forze di presa, in modo da ridurre le coppie ai giunti ed evitare eccessive forze interne, che potrebbero compromettere oggetti delicati, il problema di ottimizzazione quasi-statica del grasp al generico istante τ può essere posto nella forma

$$\begin{aligned} & \underline{f}^* = \text{argmin}\left(\underline{f}^T \underline{f}\right) \\ \text{s.t.} \quad & G\underline{f} + \underline{w} = \underline{0} \\ & f_{i_n} \ge 0 \qquad\qquad \forall i \\ & \mu_i f_{i_n} \ge \left|f_{i_t}\right| \qquad \forall i \\ & f_{i_n}(\tau) \le f_{\text{lim}} \cdot g_h(\tau) \qquad \forall i \end{aligned} \tag{40}$$

Si tratta di un problema di ottimizzazione quadratica, con funzione obiettivo quadratica e vincoli lineari di uguaglianza e di disuguaglianza, riconducibile alla classica forma

$$\begin{aligned} & \underline{x}^* = \text{argmin}\left(\underline{c}^T \underline{x} + \frac{1}{2}\underline{x}^T Q \underline{x}\right) \\ & \text{s.t.} \quad A\underline{x} = \underline{b} \\ & \qquad C\underline{x} \ge \underline{d} \end{aligned} \tag{41}$$

4.2 Ottimizzazione robusta

Il problema (41) assicura la presa in condizioni nominali, ma un minimo disturbo esterno sarebbe sufficiente a comprometterne l'efficacia. Per rendere il grasp più robusto si può pensare di considerare contemporaneamente più situazioni, ad ognuna delle quali è associata l'azione di un disturbo. Tale disturbo può essere schematizzato come una forza applicata al baricentro dell'oggetto; essa avrà modulo pari alla massima intensità contro cui ci si vuole tutelare e direzione che varia all'interno del range in cui ci si aspetta che si manifesti. In assenza di direzioni privilegiate, invece,

l'idea è quella di considerare direzioni equamente spaziate all'interno dell'angolo giro. Ad esempio, supponiamo di prevedere un disturbo con intensità massima pari a *d* e con orientazione compresa tra $\alpha_{\min}$ e $\alpha_{\max}$ rispetto all'asse delle ascisse, e ipotizziamo di considerare *n* situazioni. Allora, al caso *k-esimo* assoceremo la presenza di un disturbo $\underline{d_i}$ di modulo *d* e orientazione $\alpha_i = \alpha_{\min} + \frac{k-1}{n-1}(\alpha_{\max} - \alpha_{\min})$, con $k = 1, \dots, n$.

Per ogni caso è richiesto che sia rispettata l'equazione di equilibrio

$$G\underline{f} + \underline{w} + \underline{d}_k = \underline{0} \qquad k = 1, \dots, n. \tag{42}$$

Risolvendole insieme si ottiene un sistema di *n* equazioni vettoriali nell'incognita $\underline{f}$. Pertanto, per valori significativi di *n* c'è da aspettarsi che il sistema non ammetta soluzione. Per far fronte a questo inconveniente modifichiamo le equazioni (42) introducendo il vettore dei residui $\underline{\varepsilon_k}$, tale che si abbia

$$G\underline{f} + \underline{w} + \underline{d}_k = \underline{\varepsilon_k} \qquad k = 1, \dots, n \tag{43}$$

e definiamo una nuova funzione obiettivo nella forma

$$\frac{1}{2}\underline{f}^T K_f \underline{f} + \frac{1}{2}\underline{\varepsilon}^T K_\varepsilon \underline{\varepsilon} \tag{44}$$

dove $\underline{\varepsilon} = [\underline{\varepsilon_1}^T \quad \cdots \quad \underline{\varepsilon_n}^T]^T$ è il vettore che impila tutti gli $\underline{\varepsilon_k}$, e K_f e K_ε sono matrici che consentono di attribuire pesi diversi ai vari termini. Così, ad esempio, se si ha una probabilità più alta di avere un disturbo con una certa direzione, conviene che sia maggiore il coefficiente di peso relativo a quella orientazione.

In questo modo abbiamo eliminato il vincolo costituito dalle *n* equazioni di equilibrio (42), sostituendolo con l'obiettivo di minimizzazione dei residui.

In definitiva, il problema di ottimizzazione delle forze di presa in presenza di eventuali disturbi esterni può essere posto come

$$\begin{aligned} \underline{f}^* = \operatorname{argmin} & \left(\frac{1}{2}\underline{f}^T K_f \underline{f} + \frac{1}{2}\underline{\varepsilon}^T K_\varepsilon \underline{\varepsilon}\right) \\ \text{s.t.} \quad & G\underline{f} + \underline{w} + \underline{d}_k = \underline{\varepsilon_k} \qquad k = 1, \dots, n \\ & f_{i_n} \geq 0 \qquad \forall i \\ & \mu_i f_{i_n} \geq |f_{i_t}| \qquad \forall i \\ & f_{i_n}(\tau) \leq f_{\lim} \cdot g_h(\tau) \qquad \forall i \end{aligned} \tag{45}$$

ed è anch'esso riconducibile alla scrittura tradizionale (41).

4.3 Ottimizzazione robusta "estesa"

Per rendere il modello ancora più affidabile e la presa più robusta, un altro interessante accorgimento è quello di introdurre un certo grado di incertezza per quanto riguarda la posizione del baricentro dell'oggetto. Ipotizziamo di conoscerla con un errore massimo ξ: possiamo pensare di procedere analogamente a prima, ma scrivendo le (42) per diverse posizioni del baricentro e imponendo che siano rispettate contemporaneamente. Consideriamo ad esempio *m* posizioni del baricentro oltre a quella nominale G_0 e scegliamole in modo che siano equamente distribuite sulla circonferenza di centro G_0 e raggio ξ. Come vincolo per l'equilibrio otteniamo quindi un sistema di $n \cdot (m+1)$ equazioni vettoriali del tipo

$$G\underline{f} + \underline{w_z} + \underline{d}_{k,z} = \underline{\varepsilon}_{k,z} \qquad k = 1, \dots, n;\ z = 0, \dots, m \qquad (46)$$

ma la struttura del problema di ottimizzazione resta la stessa di (50), con la definizione aggiornata di $\underline{\varepsilon} = [\underline{\varepsilon}_{1,0}{}^T \quad \cdots \quad \underline{\varepsilon}_{n,0}{}^T \quad \underline{\varepsilon}_{1,1}{}^T \quad \cdots \quad \underline{\varepsilon}_{n,1}{}^T \quad \cdots \quad \underline{\varepsilon}_{1,m}{}^T \quad \cdots \quad \underline{\varepsilon}_{n,m}{}^T]^T$.

5 AMBIENTE DI SIMULAZIONE

5.1 CasADi

CasADi (Computer algebra systems for Algorithimc Differentiation) è un ambiente di calcolo simbolico/numerico che si avvale della sintassi dei sistemi di algebra computazionale per la differenziazione automatica (o algoritmica) e l'ottimizzazione numerica. Si tratta cioè di uno strumento finalizzato all'implementazione veloce, ma altamente efficiente, di algoritmi per l'ottimizzazione numerica non lineare, in particolare per l'ottimizzazione dinamica (controllo ottimo, model predictive control, etc.) [10].

CasADi è un pacchetto software open-source (LGPL) scritto in C++ e fruibile sia direttamente in C++, sia tramite l'interfaccia Python, come nel nostro caso.

Uno dei vantaggi connesso al suo utilizzo consiste nella possibilità di gestire i dati tramite strutture, che rendono più agevole la programmazione. Tutti i dati sono di tipo matriciale, allo stesso modo della sintassi di Matlab: ad esempio, i vettori sono trattati come matrici $n \times 1$ e gli scalari come matrici 1×1 [11].

Per quanto riguarda la simulazione, inoltre, CasADi dispone di una apposita classe *Simulator*. Definiti gli stati, i controlli e i parametri del sistema, e note le condizioni al contorno, questa istanza consente di valutare la soluzione in differenti istanti di tempo usando opportuni integratori. Si interfaccia, infatti, con i due popolari integratori Sundials, CVodes e IDAS, per problemi di tipo ODE (Equazioni Differenziali Ordinarie) e DAE (Equazioni Differenziali Algebriche) rispettivamente.

Infine, il punto di forza del CasADi è l'ottimizzazione, grazie all'utilizzo della differenziazione algoritmica (AD). Essa non è né differenziazione simbolica (e.g. Maple, Mathematica, etc.), né quella numerica, basata sull'approssimazione alle differenze finite, ma sfrutta il fatto che ogni programma esegue una sequenza di operazioni aritmetiche elementari (addizione, sottrazione, moltiplicazione, divisione, etc.) e funzioni elementari (exp, log, sin, cos, etc.). Applicando ripetutamente la regola

di composizione a queste operazioni possono essere calcolate automaticamente derivate di ordine arbitrario con la precisione di macchina [12].

Per avere un'idea di base facciamo un esempio: sia data la funzione $f(x) = (x_1 x_2 \sin x_3 + e^{x_1 x_2})/x_3$.

Ad essa vengono associate le seguenti variabili ausiliare (intermedie), ottenendo il grafo computazionale riportato in Figura 5.1:

$x_4 = x_1 * x_2$

$x_5 = \sin x_3$

$x_6 = e^{x_4}$

$x_7 = x_4 * x_5$

$x_8 = x_6 + x_7$

$x_9 = x_8/x_3$

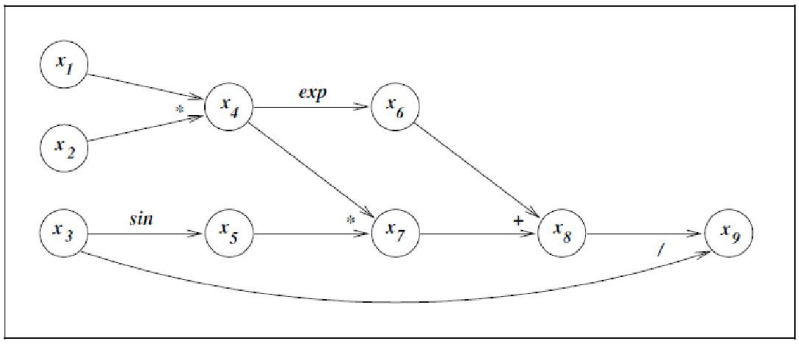


Figura 5.1: Grafo computazionale per f(x)

Dopodiché si effettua il calcolo della derivata direzionale lungo *p* per tutti i nodi:

$$D_p x_i \equiv (\nabla x_i)^T p = \sum_{j=1}^{3} \frac{\partial x_i}{\partial x_j} p_j \qquad i = 1,2,\dots,9$$

e si applica la regola di composizione.

Si noti che è possibile utilizzare il grafo computazionale e applicare la regola di composizione anche per calcolo intrinsecamente numerico. Questo viene effettuato costruendo il grafo di una porzione di codice (tape) automaticamente al momento della compilazione.

Nel CasADi i programmi di ottimizzazione quadratica (QPs) sono rappresentati nella forma

$$\begin{aligned} \underline{x}^* = \operatorname{argmin} & \left(\frac{1}{2}\underline{x}^T H \underline{x} + \underline{g}^T \underline{x}\right) \\ \text{s.t.} \quad & \underline{x}_{\text{lb}} \le \underline{x} \le \underline{x}_{\text{ub}} \\ & \underline{a}_{\text{lb}} \le A\underline{x} \le \underline{a}_{\text{ub}} \end{aligned} \qquad (47)$$

dove *H* denota una matrice Hessiana simmetrica (tipicamente semidefinita positiva). In questa rappresentazione per ottenere un vincolo di uguaglianza basta imporre che il limite superiore e inferiore siano uguali, cioè $\underline{a}_{\text{lb}}^{(j)} = \underline{a}_{\text{lb}}^{(j)}$ per qualche *j*.

Per la sua risoluzione poi abbiamo utilizzato il solutore IPOPT, generico per problemi non lineari; maggiore efficienza si potrebbe ottenere ricorrendo a solutori specifici per programmazione quadratica, come ad esempio qpOASES.

5.2 Modello di simulazione

Il modello di simulazione comprende un oggetto a forma di capsula e un manipolatore bimanuale, in cui ciascuna mano è formata da due dita. L'oggetto ha lunghezza l_o, raggio di arrotondamento r_o e massa m_o. Per quanto riguarda le dita, invece, abbiamo considerato esclusivamente i polpastrelli, che costituiscono l'end-effector della relativa catena seriale. Questi sono rappresentati in prima approssimazione come circonferenze di raggio r_f e possono essere controllati in posizione e velocità dal robot. In Figura 5.2 si riporta una rappresentazione del sistema.

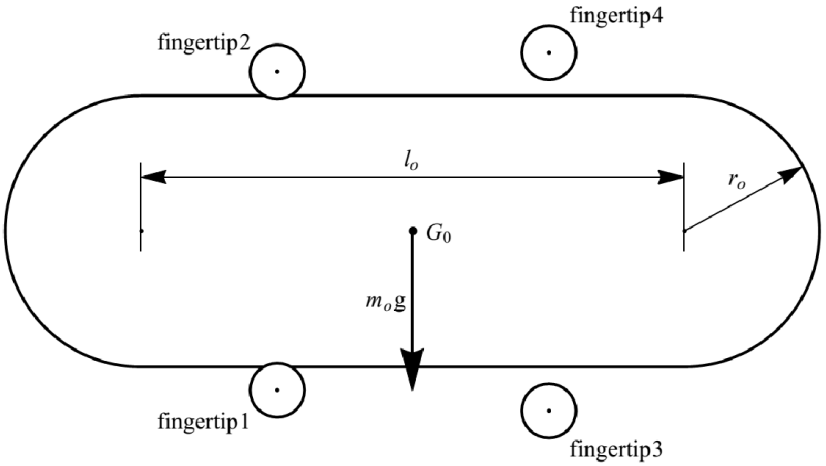


Figura 5.2: Rappresentazione del modello di simulazione

Si è indicato con 1 e 2 le dita della mano sinistra e con 3 e 4 quelle della mano destra.

La durata dell'intero intervallo di simulazione sia pari a *T*. Esso comprende una prima fase in cui è in presa solo la mano sinistra, fino all'istante t_0; a partire da questo momento entra in gioco anche la mano destra, il cui contributo al mantenimento dell'oggetto cresce gradualmente come esposto precedentemente in 4.1. Dall'istante t_1 l'oggetto è affidato alla sola mano destra.

Volendo ottimizzare le forze di contatto durante tutto il processo, suddividiamo l'intervallo di simulazione in *N* sottointervalli di durata $\Delta T = \frac{T}{N}$. Per ciascuno di essi, quindi, congeliamo l'istante iniziale e applichiamo l'algoritmo di ottimizzazione quasi-statica della distribuzione di forze analizzato nel capitolo precedente. Trovate le forze ottime, invertendo le equazioni costitutive del contatto ricaviamo la posizione e la velocità richieste alle estremità delle dita, che costituiscono le nostre variabili di controllo. Ovviamente, all'aumentare di N aumenta la stabilità del processo, ma

diminuisce il tempo di computazione a disposizione del controllo ottimo, che nella simulazione viene assunto istantaneo. Pertanto nella pratica sarà necessario trovare un compromesso tra questi due aspetti contrastanti.

Applicato il controllo, si lascia correre la simulazione utilizzando l'apposita classe *simulator* integrata nel CasADi, in cui il sottointervallo ΔT viene a sua volta discretizzato in M steps di durata $\delta T = \frac{\Delta T}{M} = \frac{T}{M \cdot N}$.

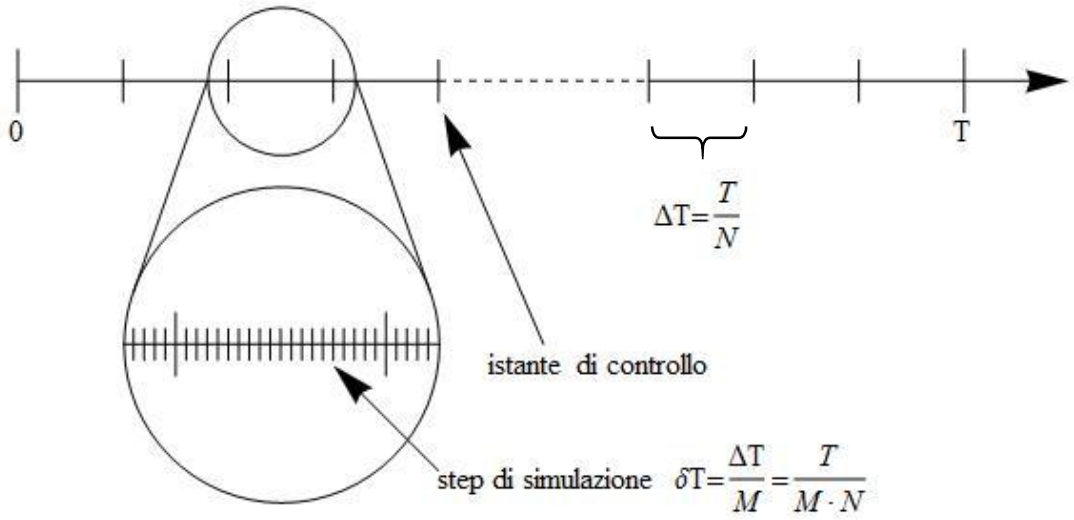


Figura 5.3: Discretizzazione dell'intervallo di simulazione

Si fa notare la possibilità di usare coefficienti di attrito diversi per il controllo e per la simulazione vera e propria. Questa distinzione permette di analizzare la situazione critica in cui il coefficiente di attrito reale sia minore rispetto al valore stimato ai fini del controllo, che comporta l'impossibilità da parte del manipolatore di applicare effettivamente le forze ottime di contatto. Consente altresì di lasciare cautelativamente un certo margine di sicurezza per la presa impiegando nel controllo una sottostima del coefficiente di attrito reale, in modo da assicurare l'effettiva applicabilità delle forze ottime.

5.3 Collision detection

Nella pratica opportuni sensori tattili di forza permettono di rilevare il contatto delle dita con l'oggetto. Per implementare tale possibilità nella simulazione si ricorre a una funzione di collision detection. Nel nostro caso si tratta di rilevare il contatto tra capsula e circonferenza.

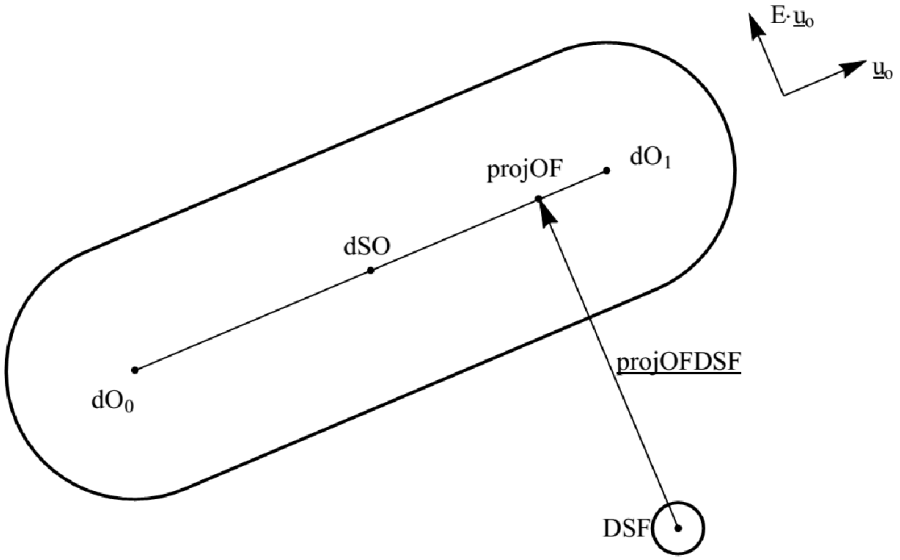


Figura 5.4:Schema per la collision detection

Definendo la matrice $E = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$ di rotazione antioraria di 90° e utilizzando la notazione illustrata in Figura 5.4 si ha:

$$\mathrm{DSF} = \mathrm{dO}_0 + \lambda\, \underline{u_o} + \eta\, E\underline{u_o} \tag{48}$$

ossia

$$\left[\underline{u_o} \middle| E\underline{u_o}\right] \begin{pmatrix} \lambda \\ \eta \end{pmatrix} = \mathrm{DSF} - \mathrm{dO}_0 \tag{49}$$

che in componenti risulta

$$\begin{bmatrix} u_{o_x} & -u_{o_y} \\ u_{o_y} & u_{o_x} \end{bmatrix} \begin{pmatrix} \lambda \\ \eta \end{pmatrix} = \mathrm{DSF} - \mathrm{dO}_0 \tag{50}$$

Si noti che la matrice $\left[\underline{u_o} \middle| E\underline{u_o}\right]$ appartiene al gruppo $\mathrm{SO}(2)$, formato dalle matrici ortogonali con determinante pari a +1; pertanto la sua inversa è uguale alla trasposta ed è immediato ottenere

$$\begin{pmatrix} \lambda \\ \eta \end{pmatrix} = \begin{bmatrix} u_{o_x} & u_{o_y} \\ -u_{o_y} & u_{o_x} \end{bmatrix} (\mathrm{DSF} - \mathrm{dO}_0) \tag{51}$$

Nel nostro caso siamo interessati a ricavare λ, parametro necessario per ottenere la proiezione di DSF sull'asse della capsula. Chiamiamo dunque

$$\mathrm{laOF} = \underline{u_o}^T (\mathrm{DSF} - \mathrm{dO}_0) \tag{52}$$

e poi effettuiamo una operazione di clamping per restringerne il valore al range $[0, l_o]$:

$$\mathrm{LAOF} = \begin{cases} 0, & if\ \mathrm{laOF} < 0 \\ l_o, & if\ \mathrm{laOF} < l_o \\ \mathrm{laOF}, & \mathrm{otherwise} \end{cases} \tag{53}$$

In questo modo è possibile ottenere la proiezione di DSF sull'asse dell'oggetto come

$$\mathrm{projOF} = \mathrm{dO}_0 + \underline{u_o} \cdot \mathrm{LAOF} \tag{54}$$

dopodichè è immediato calcolare il versore normale $\underline{u_{OF}}$ nell'eventuale punto di contatto normalizzando il vettore

$$\underline{\mathrm{projOFDSF}} = \mathrm{projOF} - \mathrm{DSF} \tag{55}$$

Poi, ruotando il versore normale di 90° si ottiene anche il versore tangenziale.

Abbiamo così definito il sistema di riferimento associato al generico punto di contatto dito-oggetto.

Si osservi che per disporre di tali informazioni nella pratica sarà necessario munire le dita di idonei sensori tattili, capaci di distinguere tra componente normale e tangenziale della forza di contatto.

5.4 Equazioni costitutive per il contatto

Una volta rilevato il contatto e definito il sistema di riferimento ad esso associato tramite lo strumento di collision detection, resta da modellare l'interazione tra il dito e l'oggetto. A tal fine si è scelto un contatto di tipo hard finger, per cui ciascun dito è in grado di esercitare una forza con componente normale e tangenziale.

5.4.1 Interazione normale

Una relazione approssimata molto usata permette di stimare la forza normale f_n a partire dal valore del gap g_n, tramite una legge di tipo esponenziale della forma

$$f_n(g_n) = f_{n_0} \cdot e^{\alpha \cdot g_n} \tag{56}$$

dove f_{n_0} corrisponde al valore della forza normale in condizione di tangenza ($g_n = 0$), mentre

$$\alpha = \frac{\ln\left(\frac{f_{n_{max}}}{f_{n_0}}\right)}{g_{n_{min}}} \tag{57}$$

con $f_{n_{max}}$ pari al valore che la forza normale assume al gap $g_{n_{min}}$, in cui si verifica la massima compenetrazione ammissibile.

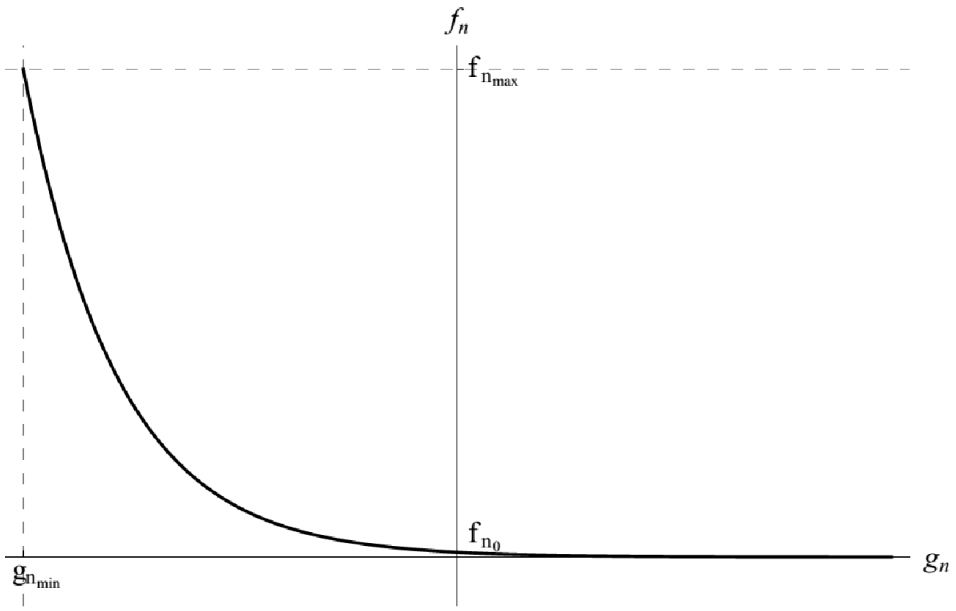


Figura 5.5: Grafico dell'equazione costitutiva per il contatto normale

Come si vede dal grafico in Figura 5.5, la forza normale è giustamente sempre positiva; essa cresce al crescere dell'interferenza (valori negativi del gap), ma risulta non nulla anche per valori positivi del gap. Questo è ovviamente un inconveniente, tuttavia scegliendo valori opportuni per i parametri in gioco è possibile fare in modo che la sua intensità decresca rapidamente, in modo da diventare subito trascurabile. D'altra parte questa equazione ha il vantaggio di essere regolare e facilmente invertibile, prestandosi quindi molto bene ad essere implementata in maniera efficiente in un codice di calcolo.

5.4.2 Interazione tangenziale

Per quanto riguarda l'interazione tangenziale, invece, si fa riferimento alla classica legge di Coulomb, caratterizzata da un comportamento discontinuo adesione-strisciamento. Pertanto, per superare l'inconveniente della non differenziabilità si utilizza una sua regolarizzazione di tipo iperbolico, che consente di descrivere una transizione regolare dalla condizione di adesione a quella di strisciamento. Essa esprime la forza tangenziale f_t in funzione del coefficiente di attrito μ, della forza normale f_n e della velocità di strisciamento v_t nel modo seguente:

$$f_t(\mu, f_n, v_t) = -\mu \cdot f_n \cdot \tanh\left(\frac{v_t}{\kappa}\right) \qquad (58)$$

dove κ è un parametro con le dimensioni della velocità che consente di regolare la "dolcezza" della curva: per $\kappa \to 0$ come caso limite si ottiene la legge di Coulomb.

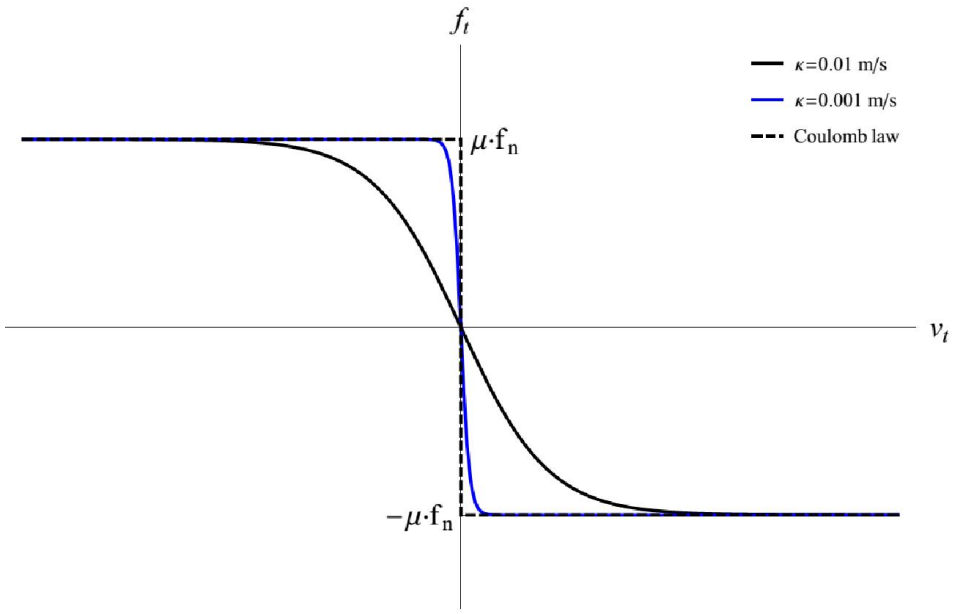


Figura 5.6: Grafico dell'equazione costitutiva per il contatto tangenziale

Dal grafico riportato in Figura 5.6 è evidente che la funzione presenta due asintoti orizzontali, che rappresentano i valori limite della forza tangenziale, di modulo μf_n. Si nota inoltre la differenziabilità di questa formulazione rispetto alla legge di Coulomb, proprietà che rende tale modello adatto ad algoritmi numerici. L'inconveniente di questa legge sta però nel fatto che qualsiasi forza è in grado di generare moto relativo, caratteristica ovviamente in contraddizione con la realtà, anche se, come detto, un ridotto valore di κ permette di rendere trascurabile questo aspetto. Si osserva, ad esempio, che già per $\kappa = 10^{-4}\ m/s$ l'equazione (58) approssima benissimo la legge di Coulomb.

6 RISULTATI

In questo capitolo si propone un confronto tra i risultati ottenuti effettuando diverse prove di simulazione in cui si valuta la risposta del sistema prendendo in esame diverse condizioni di simulazione e impiegando i diversi approcci di ottimizzazione esposti nel capitolo 4.

6.1 Dati di simulazione

Prima di procedere riportiamo per completezza i valori numerici utilizzati per le grandezze fisiche e geometriche della simulazione:

Dati geometrici

$m_o = 0.3\ kg$

$l_o = 0.2\ m$

$r_o = 0.05\ m$

$r_f = 0.01\ m$

Modello costitutivo

$f_{n_0} = 1\ N$

$f_{n_{max}} = 100\ N$

$g_{n_{min}} = -3\ mm$

$\kappa = 1 \cdot 10^{-4}\ m/s$

$\mu = 0.4$

Parametri di simulazione

$T = 4\ s$

$t_0 = 0.6\ s$

$t_1 = 3.4\ s$

$N = 20$

$M = 20$

6.2 Simulazione con controllo nominale

Partiamo dall'analisi del sistema in condizioni nominali con il controllo studiato appositamente per quelle condizioni. Si riportano l'andamento temporale di posizione e orientazione dell'oggetto e delle componenti normali delle forze di contatto.

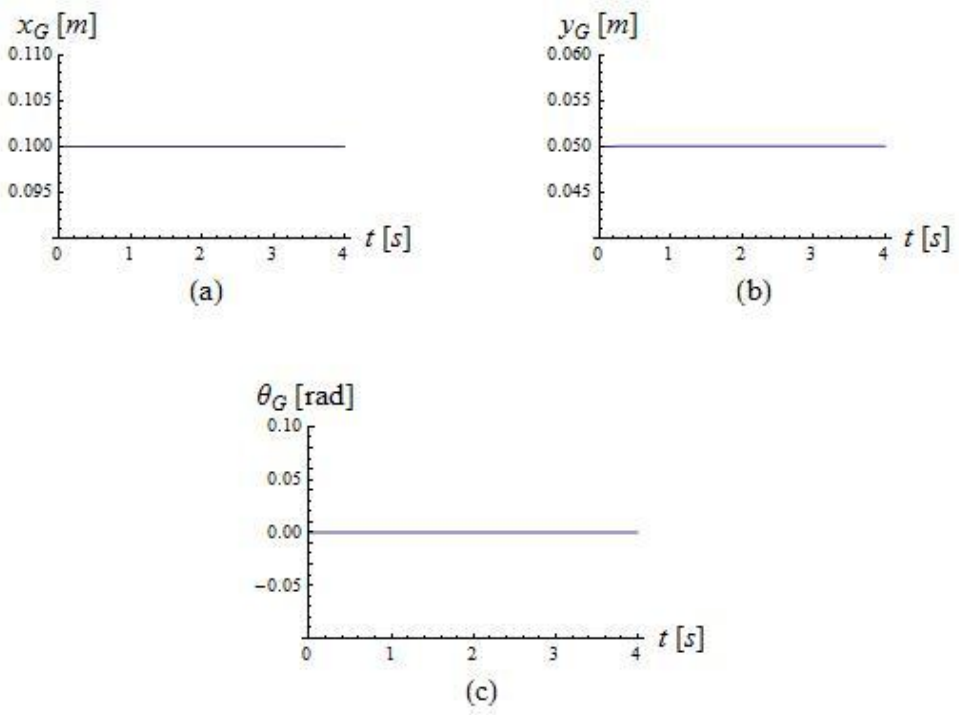


Figura 6.1: Andamento della postura dell'oggetto in condizioni nominali e con controllo nominale: (a) ascissa e (b) ordinata del baricentro, e (c) orientazione

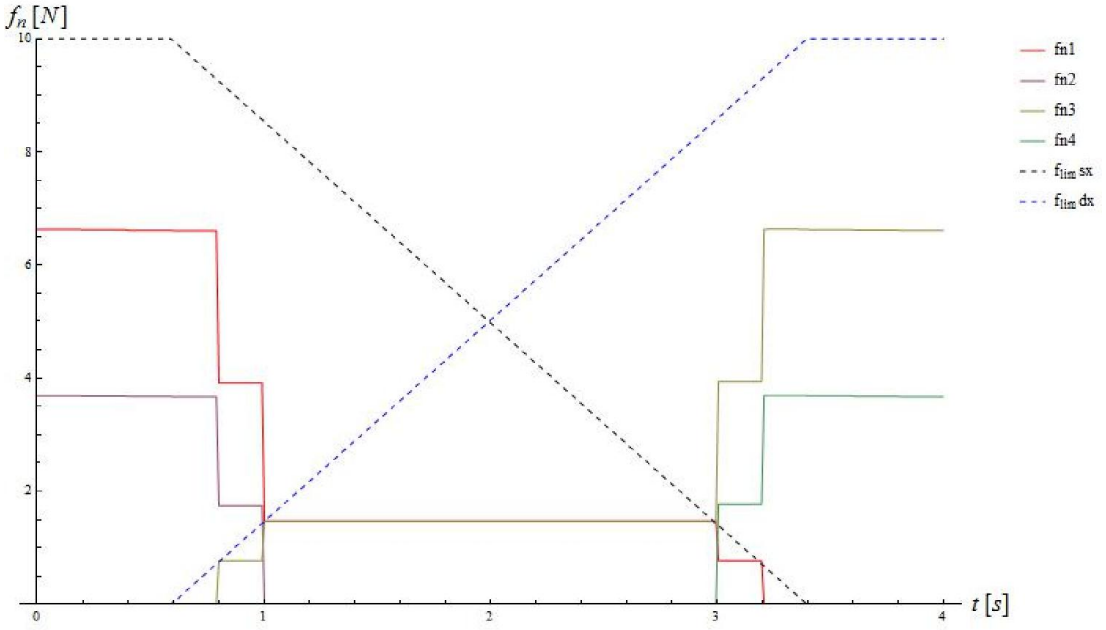


Figura 6.2: Andamento delle componenti normali delle forze di presa in condizioni nominali e con controllo nominale

Come si vede in Figura 6.1, l'oggetto non subisce nessuno spostamento, mentre dalla Figura 6.2 si nota che le forze sono stabili e tendono a mantenersi costanti, confinando la variazione in due brevissimi intervalli che coincidono con la discontinuità della funzione di vincolo sul valore massimo ammissibile per le forze stesse. Da questa stessa figura, inoltre, si osserva un comportamento insolito: nella fase centrale, in cui entrambe le mani contribuiscono alla presa, le forze sulle dita superiori (2 e 4) sono nulle, ovvero si verifica una situazione del tipo di quella rappresentata in Figura 6.3, in cui l'oggetto risulta semplicemente appoggiato sulle dita inferiori.

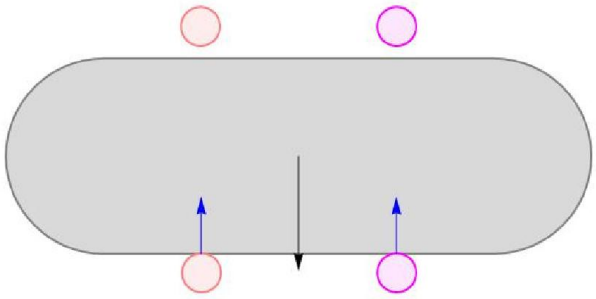


Figura 6.3: Situazione all'istante centrale della simulazione in condizioni nominali e con controllo nominale

Se da un lato si tratta di un risultato ragionevole dal punto di vista fisico, dal momento che le dita inferiori sono sufficienti ad equilibrare la forza peso e che l'obiettivo del controllo adoperato è la minimizzazione della norma 2 delle forze di contatto,

dall'altro lato è impensabile di applicarlo nella pratica, perché un minimo disturbo esterno sarebbe sufficiente a compromettere la presa.

A riscontro di questo, inseriamo nell'ambiente di simulazione una forza di disturbo $\underline{r} = \frac{m_o g}{2}\hat{x}$, applicata al baricentro dell'oggetto e avente modulo pari alla metà del peso e direzione coincidente con quella dell'asse x,. Come conseguenza si registrano spostamenti significativi dell'oggetto (Figura 6.4), che producono un andamento instabile delle forze di contatto (Figura 6.5).

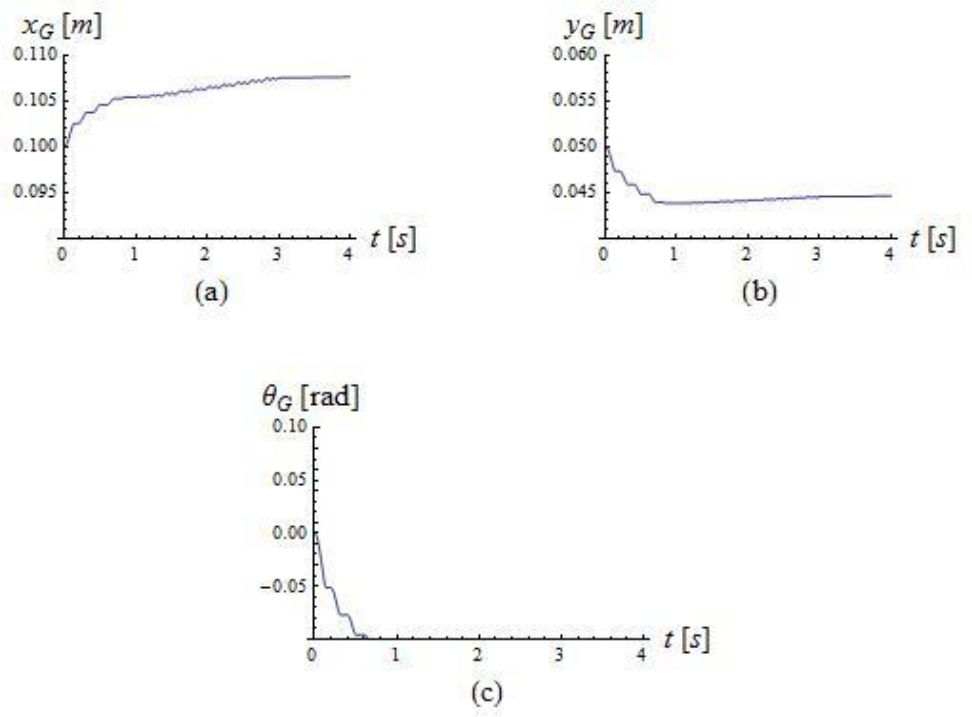


Figura 6.4: Andamento della postura dell'oggetto in presenza di disturbo e con controllo nominale: (a) ascissa e (b) ordinata del baricentro, e (c) orientazione

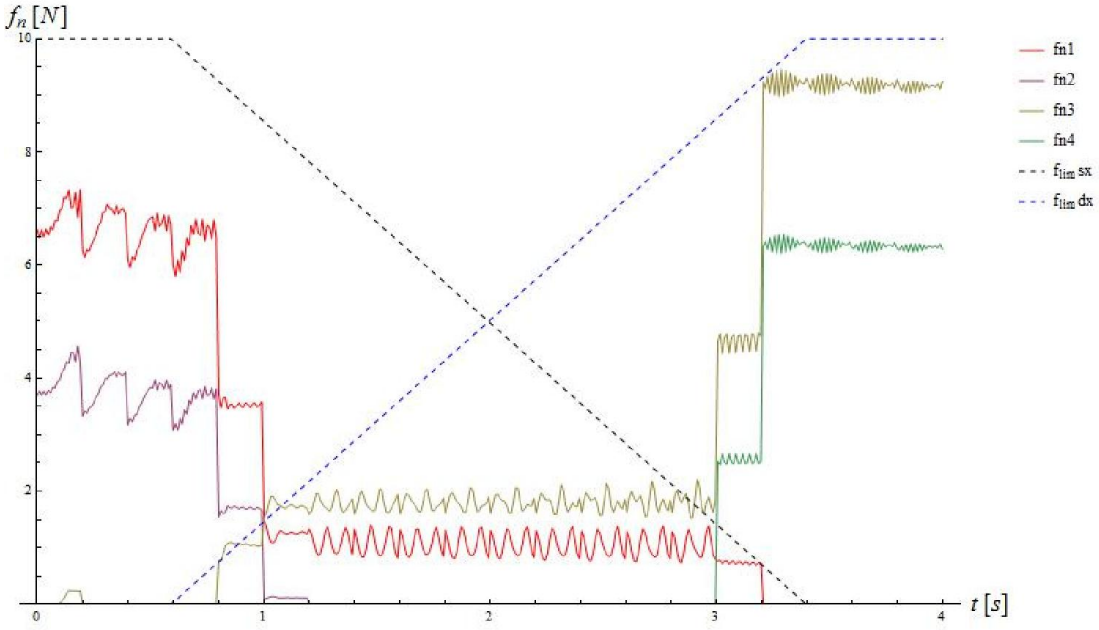


Figura 6.5: Andamento delle componenti normali delle forze di presa in presenza di disturbo e con controllo nominale

È evidente quindi la necessità di rendere la presa più robusta, così da garantire un *force closure* grasp. Di qui l'idea di modificare il controllo in modo da prevedere l'eventuale presenza di disturbi esterni, come illustrato nel paragrafo 4.2.

6.3 Simulazione con controllo robusto

Ripetendo la simulazione in condizioni nominali, ma utilizzando la nuova strategia di controllo robusto con $n = 10$ e $d = \frac{1}{2} m_o g$, si ottengono i risultati in Figura 6.6.

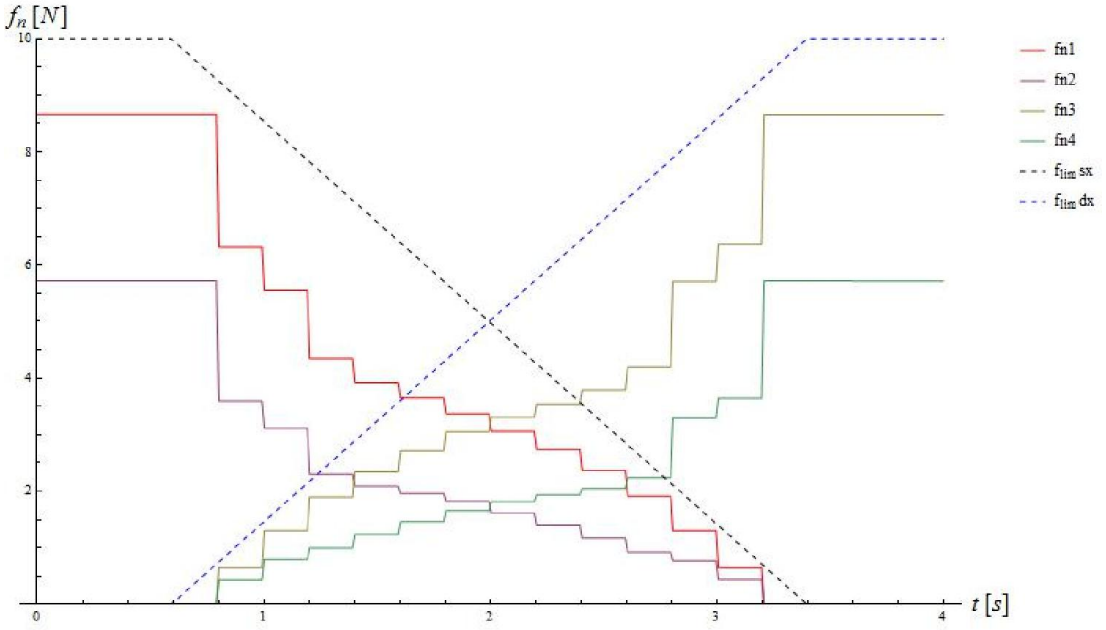


Figura 6.6: Andamento delle componenti normali delle forze di presa in condizioni nominali e con controllo robusto

Si nota una variazione molto più graduale delle forze di contatto, che inoltre sono nettamente più vicine al limite superiore imposto nel controllo; infatti, affinché la presa sia più salda ovviamente occorre stringere maggiormente l'oggetto.

L'altra sostanziale differenza che si può constatare riguarda la fase intermedia del trasferimento dell'oggetto, in cui si ha il superamento dell'inconveniente mostrato in Figura 6.3 a favore di una presa più robusta, con tutte e quattro le dita a contatto, come in Figura 6.7.

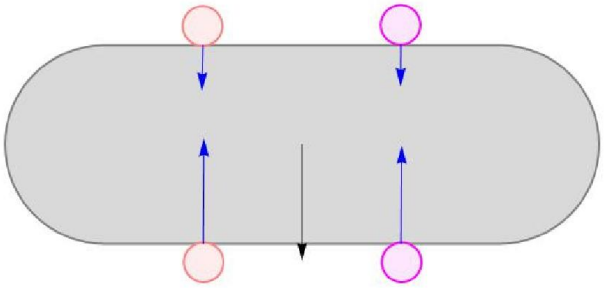


Figura 6.7: Situazione all'istante centrale della simulazione in condizioni nominali e con controllo in previsione di disturbi

Per quanto riguarda la postura dell'oggetto, invece, i risultati sono identici a quelli in Figura 6.1, cioè l'oggetto non si muove per nulla, a conferma della buona stabilità della presa. Un esito analogo si riscontra stavolta anche in presenza del disturbo $\underline{r} = \frac{m_o g}{2}\hat{x}$,

a parte una leggera perturbazione nel momento in cui entra in azione la mano destra (Figure 6.8 e 6.9).

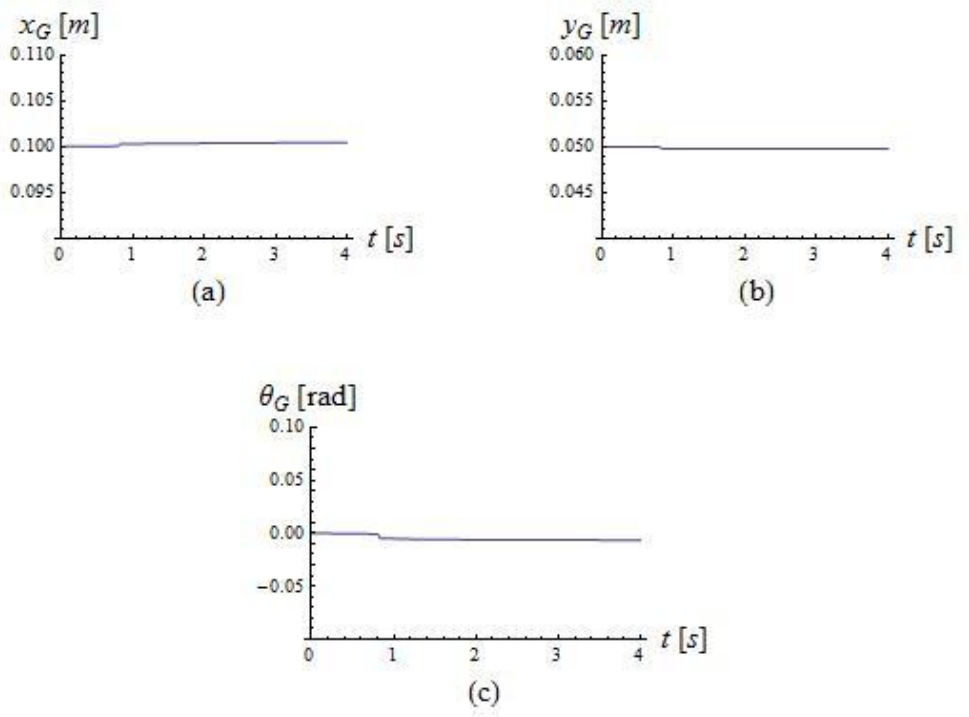


Figura 6.8: Andamento della postura dell'oggetto in presenza di disturbo e con controllo robusto: (a) ascissa e (b) ordinata del baricentro, e (c) orientazione

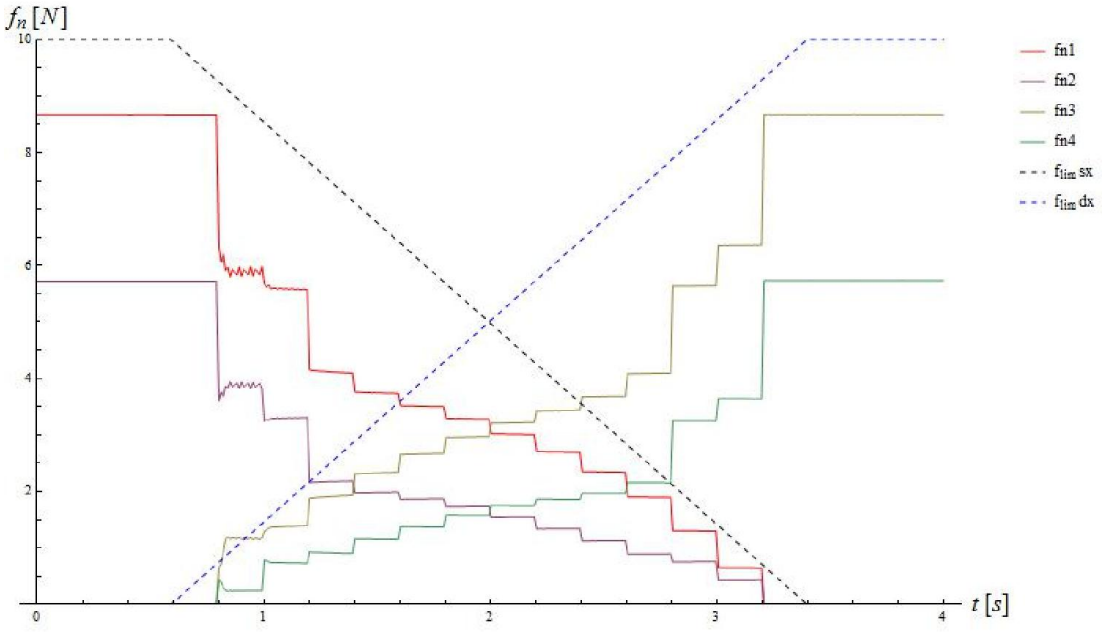


Figura 6.9: Andamento delle componenti normali delle forze di presa in presenza di disturbo e con controllo robusto

6.4 Simulazione con controllo robusto "esteso"

Per rendere il processo ancora più affidabile o comunque più vicino alla realtà, come detto, abbiamo supposto di conoscere la posizione del baricentro con un errore massimo $\xi = \frac{r_o}{5}$. Abbiamo ripetuto quindi le stesse prove di simulazione utilizzando lo strumento di ottimizzazione definito nel paragrafo 4.3 ($n = 6, m = 4$). Ancora una volta il sistema risponde in maniera stabile in condizioni nominali, mentre mostra relativamente piccole oscillazioni sia in presenza del solo disturbo orizzontale sia in presenza del disturbo e di un errore nell'individuazione del baricentro, che abbiamo supposto spostato di $\underline{e} = 0{,}7 \cdot \frac{r_o}{5}(\hat{x} + \hat{y})$ rispetto alla posizione nominale G_0. Nelle Figure 6.10 e 6.11 sono graficati i risultati relativi a quest'ultima situazione.

Tuttavia, si nota che tale strategia comporta un aumento considerevole del costo computazionale e quindi del tempo di calcolo a causa del maggior numero di equazioni di vincolo nel problema di ottimizzazione.

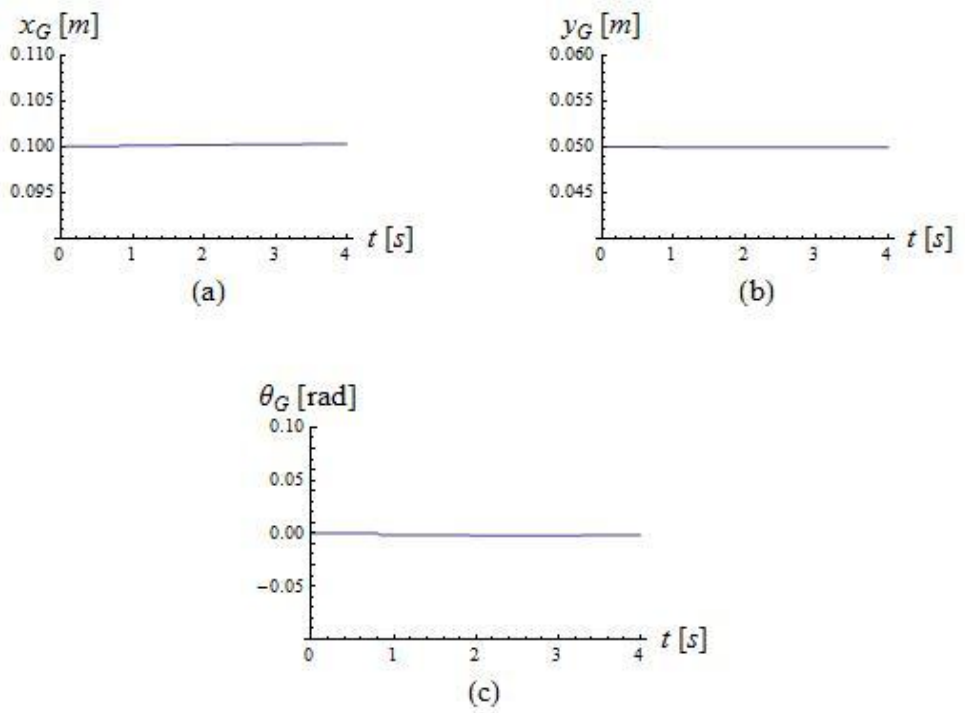


Figura 6.10: Andamento della postura dell'oggetto in presenza di disturbo e incertezza nella posizione del baricentro con controllo robusto "esteso": (a) ascissa e (b) ordinata del baricentro, e (c) orientazione

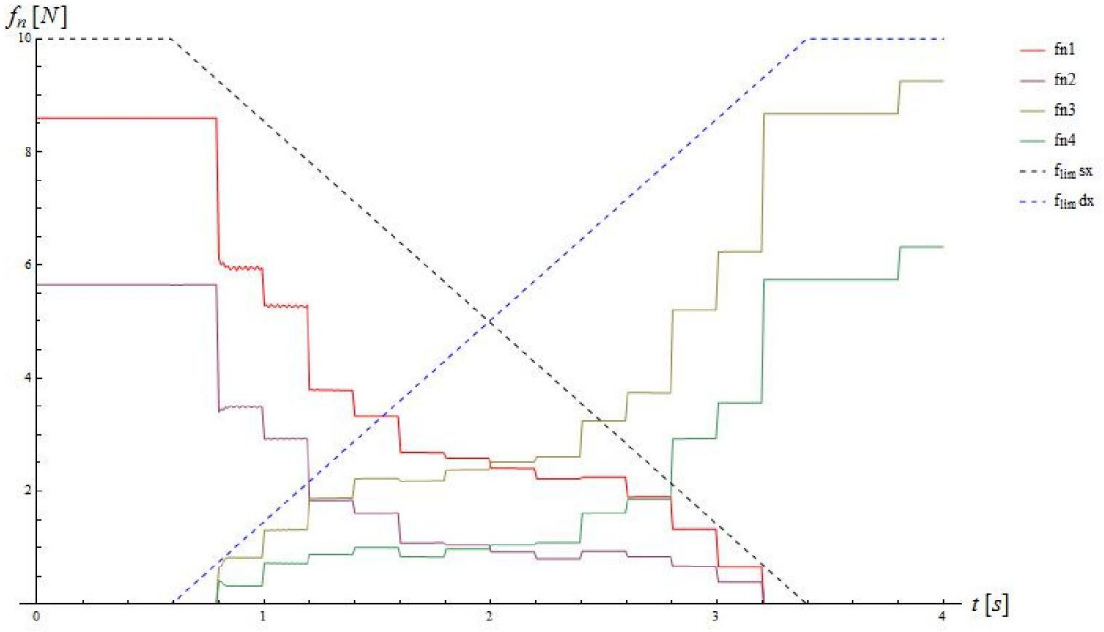


Figura 6.11: Andamento delle componenti normali delle forze di presa in presenza di disturbo e incertezza nella posizione del baricentro con controllo robusto"esteso"

7 CONCLUSIONI

In base ai risultati osservati nel capitolo precedente il modello sembra rispondere in maniera ragionevole nelle diverse situazioni simulate; pertanto costituisce un buon punto di partenza per uno studio più approfondito, da estendere al caso tridimensionale, specie in virtù della notevole velocità di esecuzione che il CasADi offre nella risoluzione dei problemi di ottimizzazione.

Per dare un'idea della sua estrema efficienza si riportano i dati relativi al problema di ottimizzazione robusta nel caso di simulazione in condizioni nominali:

Singola ottimizzazione

- Numero di variabili = 38
- Numero di iterazioni a convergenza ≈ 15
- Tempo di calcolo in IPOPT ≈ 0,15 s

Come sviluppi futuri si potrebbe pensare di utilizzare una descrizione più accurata dell'interazione normale o di considerare altre strategie di ottimizzazione, ad esempio introducendo la modellazione per intero delle dita in modo da inglobare nel controllo come ulteriore funzione obiettivo la minimizzazione delle coppie ai giunti. O ancora, sarebbe interessante testare differenti approcci alla ottimizzazione robusta, eventualmente caratterizzando in modo più preciso la natura statistica dei disturbi durante task specifici.

APPENDICE A:
CODICE PYTHON

```
from casadi import *
from casadi.tools import *
import numpy as np

INDICE = 0
'''
0 per ottimizzazione nominale
1 per ottimizzazione robusta
2 per ottimizzazione robusta “estesa”
'''
ACTIVATOR = 0 #permette di attivare (1) o disattivare (0) il disturbo nella simulazione
ErrorBaric = 1 #permette di attivare (1) o disattivare (0) l’errore nella posizione del baricentro

# capsule object quantities
lo = 0.20
ro = 0.05
mo = 0.3
Jo = (1./12.)*mo*lo**2
poO0 = np.array([ [-lo/2., 0., 1.] ]).transpose()
poO1 = np.array([ [ lo/2., 0., 1.] ]).transpose()

# finger object quantities
rf = 0.01

# physical constants
gY = 9.81 # gravity

# Valori da impostare per INDICE = 1
Settori = 10
ModuloDisturbo = mo*gY*0.5

# Valori da impostare per INDICE = 2
MagnitudeDisturb = mo*gY*0.5
UncertBaric = ro/5.
N_disturb = 6 #4
N_bari = 6 #4

if N_disturb == 0:
        N_disturb = 1
        MagnitudeDisturb = 0.
N_baric = N_bari+1

Nsteps = 14
ns = 3
ne = 3
```

```
TotalTime = 4.
Nsubsteps = 20
scale = 0.02 # fattore di scala per la visualizzazione grafica delle forze
N = Nsteps + ns + ne
Tstep = TotalTime/N

muFOreal = 0.4 # friction coefficient of the simulation (reality)
muFOcontrollore = 0.4 # friction coefficient of the controller (internal model)
muFOottimizzazione = 0.4 #- 0.001

# Parametri di fnLaw
fn0 = 1. # fn alla tangenza geometrica
fnMax = 100. # fn alla max compenetrazione
minGap = -0.003 # max compenetrazione a cui si ha fnMax

# Parametri di ftLaw
coeffTanh = 0.0001 # più si aumenta più è dolce

# Initial conditions
qOb = 0.; vqOb = 0.
dSOxb = 0.1; dSOyb = 0.05; dSOb = np.array([[dSOxb], [dSOyb]])
vdSOxb = 0.; vdSOyb = 0.; vdSOb = np.array([[vdSOxb], [vdSOyb]])

qH1b = 0.; vqH1b = 0.
dSH1xb = dSOxb-lo/4; dSH1yb = dSOyb-(ro+rf); dSH1b = np.array([[dSH1xb], [dSH1yb]])
vdSH1xb = 0.; vdSH1yb = 0.; vdSH1b = np.array([[vdSH1xb], [vdSH1yb]])

qH2b = pi; vqH2b = 0.
dSH2xb = dSOxb-lo/4; dSH2yb = dSOyb+(ro+rf); dSH2b = np.array([[dSH2xb], [dSH2yb]])
vdSH2xb = 0.; vdSH2yb = 0.; vdSH2b = np.array([[vdSH2xb], [vdSH2yb]])

qH3b = 0.; vqH3b = 0.
dSH3xb = dSOxb+lo/4; dSH3yb = dSOyb-(ro+rf); dSH3b = np.array([[dSH3xb], [dSH3yb]])
vdSH3xb = 0.; vdSH3yb = 0.; vdSH3b = np.array([[vdSH3xb], [vdSH3yb]])

qH4b = pi; vqH4b = 0.
dSH4xb = dSOxb+lo/4; dSH4yb = dSOyb+(ro+rf); dSH4b = np.array([[dSH4xb], [dSH4yb]])
vdSH4xb = 0.; vdSH4yb = 0.; vdSH4b = np.array([[vdSH4xb], [vdSH4yb]])

EE = np.array( [ [0,-1], [1,0] ] )
EEE = np.array( [ [0,-1,0], [1,0,0], [0,0,0] ] )

# Definizione delle equazioni costitutive di forza normale e tangenziale
def fnLaw(gN):
        alpha = log( fnMax / fn0 ) / minGap
        result = fn0*exp( alpha*gN )
        return result
def gNlaw(fn):
        alpha = log( fnMax / fn0 ) / minGap
        result = log(fn/fn0)/alpha
        return result

def ftLaw(vgT, fn, mu):
        result = -mu*fn*tanh(vgT/coeffTanh)
        return result
def vgTlaw(ft, fn, mu):
        result = coeffTanh*arctanh(-ft/(mu*fn))
        return result
```

```
def qpsolve(H,g,lbx,ubx,A=np.zeros((0,0)),lba=np.zeros(0),uba=np.zeros(0)):
        H = DMatrix(H)
        g = DMatrix(g)
        lbx = DMatrix(lbx)
        ubx = DMatrix(ubx)
        A = DMatrix(A)
        A = A.reshape((A.size1(),H.size1())) # Make sure matching dimensions
        lba = DMatrix(lba)
        uba = DMatrix(uba)
        qp = qpStruct(h=H.sparsity(),a=A.sparsity())
        if False:
                solver = QpSolver("qpoases",qp)
        else:
                solver = QpSolver("nlp",qp)
                solver.setOption("nlp_solver","ipopt")
        solver.init()
        solver.setInput(H,"h")
        solver.setInput(g,"g")
        solver.setInput(A,"a")
        solver.setInput(lbx,"lbx")
        solver.setInput(ubx,"ubx")
        solver.setInput(lba,"lba")
        solver.setInput(uba,"uba")
        solver.evaluate()
        return np.array(solver.getOutput("x"))

def wopt0(step, mu):
        H = np.identity(8)
        g = np.zeros(8)
        if step < ns:
                bound = 0.
        elif step > N-ne-1:
                bound = 1.
        else:
                bound = (step-ns)/(N-1-ns-ne)
        lbx = np.zeros(8)
        lbx[1] = lbx[3] = 1-bound
        lbx[5] = lbx[7] = bound
        lbx = -flim*lbx
        ubx = np.zeros(8)
        ubx[0] = ubx[1] = ubx[2] = ubx[3] = 1-bound
        ubx[4] = ubx[5] = ubx[6] = ubx[7] = bound
        ubx = flim*ubx
        A_fx = np.array([en1[0], et1[0], en2[0], et2[0], en3[0], et3[0], en4[0], et4[0]]).T
        A_fy = np.array([en1[1], et1[1], en2[1], et2[1], en3[1], et3[1], en4[1], et4[1]]).T
        A_mz = np.array([-en1[0]*GF1[1] + en1[1]*GF1[0], -et1[0]*GF1[1] + et1[1]*GF1[0], -en2[0]*GF2[1] +
        en2[1]*GF2[0], -et2[0]*GF2[1] + et2[1]*GF2[0], -en3[0]*GF3[1] + en3[1]*GF3[0], -et3[0]*GF3[1] +
        et3[1]*GF3[0], -en4[0]*GF4[1] + en4[1]*GF4[0], -et4[0]*GF4[1] + et4[1]*GF4[0]]).T
        A = np.zeros((11,8))
        A[0,0] = A[1,0] = A[4,4] = A[5,4] = -mu
        A[2,2] = A[3,2] = A[6,6] = A[7,6] = -mu
        A[0,1] = A[2,3] = A[4,5] = A[6,7] = 1.0
        A[1,1] = A[3,3] = A[5,5] = A[7,7] = -1.0
        A[8,:] = A_fx
        A[9,:] = A_fy
        A[10,:] = A_mz
        lba = -np.inf*np.ones(11)
        lba[8] = 0.
```

```
        lba[9] = mo*gY
        lba[10] = 0.
        uba = np.zeros(11)
        uba[8] = 0.
        uba[9] = mo*gY
        uba[10] = 0.
        fnt = qpsolve(H,g,lbx,ubx,A,lba,uba)
        result = fnt
        return result

def wopt1(step, mu):
        dis = np.zeros((3,Settori))
        for alpha in range(Settori):
                theta = 2*pi*alpha/Settori
                dis[0][alpha] = ModuloDisturbo*cos(theta)
                dis[1][alpha] = ModuloDisturbo*sin(theta)
                dis[2][alpha] = 0.
        H = np.identity(8+3*Settori)
        for index in range(8):
                H[index,index] = 0.
        g = np.zeros(8+3*Settori)
        if step < ns:
                bound = 0.
        elif step > N-ne-1:
                bound = 1.
        else:
                bound = (step-ns)/(N-1-ns-ne)
        lbx = -np.inf*np.ones(8+3*Settori)
        lbx[1] = lbx[3] = -flim*(1-bound)
        lbx[5] = lbx[7] = -flim*bound
        lbx[0] = lbx[2] = lbx[4] = lbx[6] = 0.
        ubx = np.inf*np.ones(8+3*Settori)
        ubx[0] = ubx[1] = ubx[2] = ubx[3] = flim*(1-bound)
        ubx[4] = ubx[5] = ubx[6] = ubx[7] = flim*bound
        A = np.zeros((8+3*Settori,8+3*Settori))
        A[0,0] = A[1,0] = A[4,4] = A[5,4] = -mu
        A[2,2] = A[3,2] = A[6,6] = A[7,6] = -mu
        A[0,1] = A[2,3] = A[4,5] = A[6,7] = 1.0
        A[1,1] = A[3,3] = A[5,5] = A[7,7] = -1.0
        lba = -np.inf*np.ones(8+3*Settori)
        uba = np.zeros(8+3*Settori)
        for alpha in range(Settori):
                A[8+3*alpha,0] = en1[0][0]
                A[8+3*alpha,1] = et1[0][0]
                A[8+3*alpha,2] = en2[0][0]
                A[8+3*alpha,3] = et2[0][0]
                A[8+3*alpha,4] = en3[0][0]
                A[8+3*alpha,5] = et3[0][0]
                A[8+3*alpha,6] = en4[0][0]
                A[8+3*alpha,7] = et4[0][0]
                A[8+3*alpha,8+3*alpha] = -1.
                A[9+3*alpha,0] = en1[1][0]
                A[9+3*alpha,1] = et1[1][0]
                A[9+3*alpha,2] = en2[1][0]
                A[9+3*alpha,3] = et2[1][0]
                A[9+3*alpha,4] = en3[1][0]
                A[9+3*alpha,5] = et3[1][0]
                A[9+3*alpha,6] = en4[1][0]
```

```
                A[9+3*alpha,7] = et4[1][0]
                A[9+3*alpha,9+3*alpha] = -1.
                A[10+3*alpha,0] = - en1[0][0]*OF1[1][0] + en1[1][0]*OF1[0][0]
                A[10+3*alpha,1] = - et1[0][0]*OF1[1][0] + et1[1][0]*OF1[0][0]
                A[10+3*alpha,2] = - en2[0][0]*OF2[1][0] + en2[1][0]*OF2[0][0]
                A[10+3*alpha,3] = - et2[0][0]*OF2[1][0] + et2[1][0]*OF2[0][0]
                A[10+3*alpha,4] = - en3[0][0]*OF3[1][0] + en3[1][0]*OF3[0][0]
                A[10+3*alpha,5] = - et3[0][0]*OF3[1][0] + et3[1][0]*OF3[0][0]
                A[10+3*alpha,6] = - en4[0][0]*OF4[1][0] + en4[1][0]*OF4[0][0]
                A[10+3*alpha,7] = - et4[0][0]*OF4[1][0] + et4[1][0]*OF4[0][0]
                A[10+3*alpha,10+3*alpha] = -1.
                lba[8+3*alpha] = - dis[0][alpha]
                lba[9+3*alpha] = mo*gY - dis[1][alpha]
                lba[10+3*alpha] = mo*gY*OG[0][0] + dis[0][alpha]*OG[1][0] - dis[1][alpha]*OG[0][0] - dis[2][0]
                uba[8+3*alpha] = - dis[0][alpha]
                uba[9+3*alpha] = mo*gY - dis[1][alpha]
                uba[10+3*alpha] = mo*gY*OG[0][0] + dis[0][alpha]*OG[1][0] - dis[1][alpha]*OG[0][0] - dis[2][0]
        fnt = qpsolve(H,g,lbx,ubx,A,lba,uba)
        result = fnt[0:8]
        return result

def wopt2(step, mu):
        dis = np.zeros((3,N_disturb))
        for alpha in range(N_disturb):
                theta = 2*pi*alpha/N_disturb
                dis[0][alpha] = MagnitudeDisturb*cos(theta)
                dis[1][alpha] = MagnitudeDisturb*sin(theta)
                dis[2][alpha] = 0.
        H = np.identity(8+3*N_disturb*N_baric)
        for index in range(8):
                H[index,index] = 0.
        g = np.zeros(8+3*N_disturb*N_baric)
        if step < ns:
                bound = 0.
        elif step > N-ne-1:
                bound = 1.
        else:
                bound = (step-ns)/(N-1-ns-ne)
        lbx = -np.inf*np.ones(8+3*N_disturb*N_baric)
        lbx[1] = lbx[3] = -flim*(1-bound)
        lbx[5] = lbx[7] = -flim*bound
        lbx[0] = lbx[2] = lbx[4] = lbx[6] = 0.
        ubx = np.inf*np.ones(8+3*N_disturb*N_baric)
        ubx[0] = ubx[1] = ubx[2] = ubx[3] = flim*(1-bound)
        ubx[4] = ubx[5] = ubx[6] = ubx[7] = flim*bound
        A = np.zeros((8+3*N_disturb*N_baric,8+3*N_disturb*N_baric))
        A[0,0] = A[1,0] = A[4,4] = A[5,4] = -mu
        A[2,2] = A[3,2] = A[6,6] = A[7,6] = -mu
        A[0,1] = A[2,3] = A[4,5] = A[6,7] = 1.0
        A[1,1] = A[3,3] = A[5,5] = A[7,7] = -1.0
        lba = -np.inf*np.ones(8+3*N_disturb*N_baric)
        uba = np.zeros(8+3*N_disturb*N_baric)
        OGx = OG[0][0]
        OGy = OG[1][0]
        OGi = np.zeros((2,N_baric))
        for beta in range(N_baric):
                fhi = 2*pi*beta/N_baric
                if beta == 0:
```

```
                        OGi[0][beta] = OGx
                        OGi[1][beta] = OGy
                else:
                        OGi[0][beta] = OGx + UncertBaric*cos(fhi)
                        OGi[1][beta] = OGy + UncertBaric*sin(fhi)
                for alpha in range(N_disturb):
                        A[8+3*alpha*N_baric+3*beta,0] = en1[0][0]
                        A[8+3*alpha*N_baric+3*beta,1] = et1[0][0]
                        A[8+3*alpha*N_baric+3*beta,2] = en2[0][0]
                        A[8+3*alpha*N_baric+3*beta,3] = et2[0][0]
                        A[8+3*alpha*N_baric+3*beta,4] = en3[0][0]
                        A[8+3*alpha*N_baric+3*beta,5] = et3[0][0]
                        A[8+3*alpha*N_baric+3*beta,6] = en4[0][0]
                        A[8+3*alpha*N_baric+3*beta,7] = et4[0][0]
                        A[8+3*alpha*N_baric+3*beta,8+3*N_baric*alpha+3*beta] = -1.
                        A[9+3*alpha*N_baric+3*beta,0] = en1[1][0]
                        A[9+3*alpha*N_baric+3*beta,1] = et1[1][0]
                        A[9+3*alpha*N_baric+3*beta,2] = en2[1][0]
                        A[9+3*alpha*N_baric+3*beta,3] = et2[1][0]
                        A[9+3*alpha*N_baric+3*beta,4] = en3[1][0]
                        A[9+3*alpha*N_baric+3*beta,5] = et3[1][0]
                        A[9+3*alpha*N_baric+3*beta,6] = en4[1][0]
                        A[9+3*alpha*N_baric+3*beta,7] = et4[1][0]
                        A[9+3*alpha*N_baric+3*beta,9+3*alpha*N_baric+3*beta] = -1.
                        A[10+3*alpha*N_baric+3*beta,0] = - en1[0][0]*OF1[1][0] + en1[1][0]*OF1[0][0]
                        A[10+3*alpha*N_baric+3*beta,1] = - et1[0][0]*OF1[1][0] + et1[1][0]*OF1[0][0]
                        A[10+3*alpha*N_baric+3*beta,2] = - en2[0][0]*OF2[1][0] + en2[1][0]*OF2[0][0]
                        A[10+3*alpha*N_baric+3*beta,3] = - et2[0][0]*OF2[1][0] + et2[1][0]*OF2[0][0]
                        A[10+3*alpha*N_baric+3*beta,4] = - en3[0][0]*OF3[1][0] + en3[1][0]*OF3[0][0]
                        A[10+3*alpha*N_baric+3*beta,5] = - et3[0][0]*OF3[1][0] + et3[1][0]*OF3[0][0]
                        A[10+3*alpha*N_baric+3*beta,6] = - en4[0][0]*OF4[1][0] + en4[1][0]*OF4[0][0]
                        A[10+3*alpha*N_baric+3*beta,7] = - et4[0][0]*OF4[1][0] + et4[1][0]*OF4[0][0]
                        A[10+3*alpha*N_baric+3*beta,10+3*alpha*N_baric+3*beta] = -1.
                        lba[8+3*alpha*N_baric+3*beta] = - dis[0][alpha]
                        lba[9+3*alpha*N_baric+3*beta] = mo*gY - dis[1][alpha]
                        lba[10+3*alpha*N_baric+3*beta] = mo*gY*OGi[0][beta] + dis[0][alpha]*OGi[1][beta] -
                dis[1][alpha]*OGi[0][beta] - dis[2][alpha]
                        uba[8+3*alpha*N_baric+3*beta] = - dis[0][alpha]
                        uba[9+3*alpha*N_baric+3*beta] = mo*gY - dis[1][alpha]
                        uba[10+3*alpha*N_baric+3*beta] = mo*gY*OGi[0][beta] + dis[0][alpha]*OGi[1][beta]
                - dis[1][alpha]*OGi[0][beta] - dis[2][alpha]
        fnt = qpsolve(H,g,lbx,ubx,A,lba,uba)
        result = fnt[0:8]
        return result

def R2D(q):
        cq = np.cos(q)
        sq = np.sin(q)
        result = np.array([[cq, -sq],[sq, cq]])
        return result

def g2D(q, d):
        cq = np.cos(q)
        sq = np.sin(q)
        result = np.array([[cq, -sq, d[0][0]],[sq, cq, d[1][0]],[0., 0., 1. ]])
        return result
```

```
def Vhat(vq, d, vd):
	vOrig = np.array(vd) - vq*np.dot(EE, d)
	temp = np.bmat([[ vq*EE , vOrig ],[[[0, 0]], [[0]]]])
	result = np.array(temp)
	return result

def N_distHandCircCap(qH, dSH, qO, dSO):
	gSO = g2D(qO, dSO)
	uo = gSO[ 0:3, [0] ]
	dO0 = np.dot(gSO, poO0)
	DSF = np.concatenate( (dSH, [[1.]]), axis=0)
	laOF = np.dot( uo.T, -dO0 + DSF )
	if laOF < 0.:
		LAOF = 0.
	elif laOF > lo:
		LAOF = lo
	else:
		LAOF = laOF
	projOF = dO0 + uo*LAOF
	projOFdSF = projOF - DSF
	normprojOFdSF = norm(projOFdSF)
	uOF = np.dot(projOFdSF, 1./normprojOFdSF)
	distOF = normprojOFdSF - ( ro + rf )
	FcandOF = DSF + uOF*rf
	OcandOF = projOF - uOF*ro
	inwardNormal = uOF
	positiveTangent = -np.dot( EEE, uOF)
	result = [distOF, [DSF, projOF], [FcandOF, OcandOF], [inwardNormal, positiveTangent] ]
	return result

def N_diffHandCircCap(qH, dSH, qO, dSO, vqH, vdSH, vqO, vdSO):
	geoHandCircCap = N_distHandCircCap(qH, dSH, qO, dSO)
	[PSF, PSO] = geoHandCircCap[2]
	[enO, etO] = geoHandCircCap[3]
	VSFS = Vhat( vqH, dSH, vdSH )
	VSOS = Vhat( vqO, dSO, vdSO )
	vFO = np.dot( VSOS, PSO ) - np.dot( VSFS, PSF )
	vFOn = np.dot( vFO.transpose(), enO )
	vFOt = np.dot( vFO.transpose(), etO )
	geoHandCircCap.append( [vFOn, vFOt] )
	return geoHandCircCap

def N_single_dist(qH, dSHx, dSHy, qO, dSOx, dSOy):
	dSH = np.array([[dSHx], [dSHy]])
	dSO = np.array([[dSOx], [dSOy]])
	gSO = g2D(qO, dSO)
	uo = gSO[ 0:3, [0] ]
	dO0 = np.dot(gSO, poO0)
	DSF = np.concatenate( (dSH, [[1]]), axis=0)
	laOF = np.dot( uo.transpose(), -dO0 + DSF ).item()
	if laOF < 0.:
		LAOF = 0.
	elif laOF > lo:
		LAOF = lo
	else:
		LAOF = laOF
	projOF = dO0 + uo*LAOF
	projOFdSF = projOF - DSF
```

```
        normprojOFdSF = norm(projOFdSF)
        uOF = projOFdSF / normprojOFdSF
        OcandOF = projOF - uOF*ro
        return OcandOF

def N_single_wrench(qH, dSHx, dSHy, qO, dSOx, dSOy, vqH, vdSHx, vdSHy, vqO, vdSOx, vdSOy):
        dSH = np.array([[dSHx], [dSHy]])
        dSO = np.array([[dSOx], [dSOy]])
        vdSH = np.array([[vdSHx], [vdSHy]])
        vdSO = np.array([[vdSOx], [vdSOy]])
        kin = N_diffHandCircCap(qH, dSH, qO, dSO, vqH, vdSH, vqO, vdSO)
        gNRigid = kin[0]
        fn = fnLaw(gNRigid)
        vgT = (kin[4][1]).item()
        ft = ftLaw( vgT, fn, muFOreal )
        [enO, etO] = kin[3]
        f = np.array( fn*enO + ft*etO )
        result = [f, fn, ft]
        return result

def distHandCircCap(qH, dSH, qO, dSO):
        gSO = g2D(qO, dSO)
        uo = gSO[ 0:3, [0] ]
        dO0 = np.dot(gSO, poO0)
        DSF = np.concatenate( (dSH, [[1]]), axis=0)
        laOF = np.dot( uo.transpose(), -dO0 + DSF ).item()
        LAOF = np.array( if_else( laOF < 0., 0., if_else( laOF > lo, lo, laOF ) ) )
        projOF = dO0 + uo*LAOF
        projOFdSF = projOF - DSF
        normprojOFdSF = norm(projOFdSF)
        uOF = projOFdSF / normprojOFdSF
        distOF = ( normprojOFdSF - ( ro + rf ) )
        FcandOF = DSF + uOF*rf
        OcandOF = projOF - uOF*ro
        inwardNormal = uOF
        positiveTangent = -np.dot( EEE, uOF)
        result = [ distOF, [ DSF, projOF ], [ FcandOF, OcandOF ], [inwardNormal, positiveTangent] ]
        return result

def diffHandCircCap(qH, dSH, qO, dSO, vqH, vdSH, vqO, vdSO):
        geoHandCircCap = distHandCircCap(qH, dSH, qO, dSO)
        [PSF, PSO] = geoHandCircCap[2]
        [enO, etO] = geoHandCircCap[3]
        VSFS = Vhat( vqH, dSH, vdSH )
        VSOS = Vhat( vqO, dSO, vdSO )
        vFO = np.dot( VSOS, PSO ) - np.dot( VSFS, PSF )
        vFOn = np.dot( vFO.transpose(), enO )
        vFOt = np.dot( vFO.transpose(), etO )
        geoHandCircCap.append( [vFOn, vFOt] )
        return geoHandCircCap

def wrenchHandCircCap( qH, dSH, qO, dSO, vqH, vdSH, vqO, vdSO ):
        kin = diffHandCircCap(qH, dSH, qO, dSO, vqH, vdSH, vqO, vdSO)
        gNRigid = kin[0]
        fn = fnLaw(gNRigid)
        vgT = (kin[4][1]).item()
        ft = np.array( ftLaw( vgT, fn, muFO ) )
        [enO, etO] = kin[3]
```

```
        f = np.array( fn*enO + ft*etO )
        fX = f[0][0]
        fY = f[1][0]
        arm = kin[2][1][0:2, [0]] - np.array(dSO)
        mZ = -arm[1][0]*fX + arm[0][0]*fY
        result = np.array( [ [fX], [fY], [mZ] ] )
        return result

        def RHS_wrench_Cap(qH1, dSH1x, dSH1y, qH2, dSH2x, dSH2y, qH3, dSH3x, dSH3y, qH4, dSH4x,
        dSH4y, qO, dSOx, dSOy, vqH1, vdSH1x, vdSH1y, vqH2, vdSH2x, vdSH2y, vqH3, vdSH3x, vdSH3y,
        vqH4, vdSH4x, vdSH4y, vqO, vdSOx, vdSOy):
        dSH1 = np.array([[dSH1x],[dSH1y]]); vdSH1 = np.array([[vdSH1x],[vdSH1y]])
        dSH2 = np.array([[dSH2x],[dSH2y]]); vdSH2 = np.array([[vdSH2x],[vdSH2y]])
        dSH3 = np.array([[dSH3x],[dSH3y]]); vdSH3 = np.array([[vdSH3x],[vdSH3y]])
        dSH4 = np.array([[dSH4x],[dSH4y]]); vdSH4 = np.array([[vdSH4x],[vdSH4y]])
        dSO = np.array([[dSOx],[dSOy]]); vdSO = np.array([[vdSOx],[vdSOy]])
        wrenchHandCap = wrenchHandCircCap( qH1, dSH1, qO, dSO, vqH1, vdSH1, vqO, vdSO )
        wrenchHandCap += wrenchHandCircCap( qH2, dSH2, qO, dSO, vqH2, vdSH2, vqO, vdSO )
        wrenchHandCap += wrenchHandCircCap( qH3, dSH3, qO, dSO, vqH3, vdSH3, vqO, vdSO )
        wrenchHandCap += wrenchHandCircCap( qH4, dSH4, qO, dSO, vqH4, vdSH4, vqO, vdSO )
        result = wrenchHandCap
        return result

############################ INITIAL CONDITIONS ########################
C1xseq = []; C1yseq = []; C2xseq = []; C2yseq = []; C3xseq = []; C3yseq = []; C4xseq = []; C4yseq = []

f1xseq = []; f1yseq = []; f2xseq = []; f2yseq = []; f3xseq = []; f3yseq = []; f4xseq = []; f4yseq = []

dSH1bseq = np.array([[], []]); vdSH1bseq = np.array([[], []])
dSH2bseq = np.array([[], []]); vdSH2bseq = np.array([[], []])
dSH3bseq = np.array([[], []]); vdSH3bseq = np.array([[], []])
dSH4bseq = np.array([[], []]); vdSH4bseq = np.array([[], []])
qH1bseq = []; qH2bseq = []; qH3bseq = []; qH4bseq = []
vqH1bseq = []; vqH2bseq = []; vqH3bseq = []; vqH4bseq = []

qOseq = []; vqOseq = []; dSOxseq = []; dSOyseq = []; vdSOxseq = []; vdSOyseq = []
dSH1xseq = []; dSH1yseq = []; vdSH1xseq = []; vdSH1yseq = []
dSH2xseq = []; dSH2yseq = []; vdSH2xseq = []; vdSH2yseq = []
dSH3xseq = []; dSH3yseq = []; vdSH3xseq = []; vdSH3yseq = []
dSH4xseq = []; dSH4yseq = []; vdSH4xseq = []; vdSH4yseq = []

RealDisturbseq = np.array([[],[]])
fntoptseq = np.array([[], [], [], [], [], [], [], []])

myZero = 0.8e-4
N = Nsteps + ns + ne
flim = 10.

for w in range(N):
        k = w*1.0
        fn0 = 1.
        pos1 = N_distHandCircCap(qH1b, dSH1b, qOb, dSOb)
        pos2 = N_distHandCircCap(qH2b, dSH2b, qOb, dSOb)
        pos3 = N_distHandCircCap(qH3b, dSH3b, qOb, dSOb)
        pos4 = N_distHandCircCap(qH4b, dSH4b, qOb, dSOb)
        OG = dSOb
        Vh = Vhat(vqOb, [[dSOxb],[dSOyb]], [[vdSOxb],[vdSOyb]])
        gN1 = pos1[0]; gN2 = pos2[0]; gN3 = pos3[0]; gN4 = pos4[0]
```

```
[PSF1, PSO1] = pos1[2]; [PSF2, PSO2] = pos3[2]; [PSF3, PSO3] = pos2[2]; [PSF4, PSO4] = pos4[2]
un1 = pos1[3][0]; ut1 = pos1[3][1]; en1 = un1[0:2]; et1 = ut1[0:2]
un2 = pos2[3][0]; ut2 = pos2[3][1]; en2 = un2[0:2]; et2 = ut2[0:2]
un3 = pos3[3][0]; ut3 = pos3[3][1]; en3 = un3[0:2]; et3 = ut3[0:2]
un4 = pos4[3][0]; ut4 = pos4[3][1]; en4 = un4[0:2]; et4 = ut4[0:2]
OF1 = pos1[2][1][0:2]; OF2 = pos2[2][1][0:2]; OF3 = pos3[2][1][0:2]; OF4 = pos4[2][1][0:2]
GF1 = OF1-OG; GF2 = OF2-OG; GF3 = OF3-OG; GF4 = OF4-OG

if INDICE == 0:
        fntopt = wopt0(k, muFOottimizzazione)
elif INDICE == 1:
        fntopt = wopt1(k, muFOottimizzazione)
elif INDICE == 2:
        fntopt = wopt2(k, muFOottimizzazione)
else:
        print "error INDICE"

fntoptseq = np.concatenate((fntoptseq, fntopt), axis=1)
fn1opt = fntopt[0][0]; ft1opt = fntopt[1][0]
fn2opt = fntopt[2][0]; ft2opt = fntopt[3][0]
fn3opt = fntopt[4][0]; ft3opt = fntopt[5][0]
fn4opt = fntopt[6][0]; ft4opt = fntopt[7][0]

if fn1opt < fn0*myZero :
        fn1opt = fn0*myZero
        gN1opt = gNlaw(fn1opt)
        vgT1opt = vgTlaw(ft1opt, fn1opt, muFOcontrollore)
        vdSH1xb = - vgT1opt*et1[0][0]; vdSH1yb = - vgT1opt*et1[1][0]
if fn2opt < fn0*myZero :
        fn2opt = fn0*myZero
        gN2opt = gNlaw(fn2opt)
        vgT2opt = vgTlaw(ft2opt, fn2opt, muFOcontrollore)
        vdSH2xb = - vgT2opt*et2[0][0]; vdSH2yb = - vgT2opt*et2[1][0]
if fn3opt < fn0*myZero :
        fn3opt = fn0*myZero
        gN3opt = gNlaw(fn3opt)
        vgT3opt = vgTlaw(ft3opt, fn3opt, muFOcontrollore)
        vdSH3xb = - vgT3opt*et3[0][0]; vdSH3yb = - vgT3opt*et3[1][0]
if fn4opt < fn0*myZero :
        fn4opt = fn0*myZero
        gN4opt = gNlaw(fn4opt)
        vgT4opt = vgTlaw(ft4opt, fn4opt, muFOcontrollore)
        vdSH4xb = - vgT4opt*et4[0][0]; vdSH4yb = - vgT4opt*et4[1][0]

deltan1 = (gN1opt-gN1); deltan2 = (gN2opt-gN2); deltan3 = (gN3opt-gN3); deltan4 = (gN4opt-gN4)
dSH1b = dSH1b - deltan1*en1
dSH2b = dSH2b - deltan2*en2
dSH3b = dSH3b - deltan3*en3
dSH4b = dSH4b - deltan4*en4
dSH1xb = dSH1b[0][0]; dSH1yb = dSH1b[1][0]
dSH2xb = dSH2b[0][0]; dSH2yb = dSH2b[1][0]
dSH3xb = dSH3b[0][0]; dSH3yb = dSH3b[1][0]
dSH4xb = dSH4b[0][0]; dSH4yb = dSH4b[1][0]

################################ SIMULAZIONE ############################
        states=struct_symSX(["dSOx","dSOy","qO","vdSOx","vdSOy","vqO","dSH1x","dSH1y","dSH2x","dSH2y
","dSH3x","dSH3y","dSH4x","dSH4y","vdSH1x","vdSH1y","vdSH2x","vdSH2y","vdSH3x","vdSH3y","vdSH4x","vdS
H4y"])
```

```
        dSOx,dSOy,qO,vdSOx,vdSOy,vqO,dSH1x,dSH1y,dSH2x,dSH2y,dSH3x,dSH3y,dSH4x,dSH4y,vdSH1x
,vdSH1y,vdSH2x,vdSH2y,vdSH3x,vdSH3y,vdSH4x,vdSH4y = states[...]

        controls = struct_symSX(["qH1", "qH2", "qH3", "qH4", "vqH1", "vqH2", "vqH3", "vqH4"])
        qH1,qH2,qH3,qH4,vqH1,vqH2,vqH3,vqH4 = controls[...]

        parameters = struct_symSX(["fn0", "muFO"])
        fn0, muFO = parameters[...]
        parameters_ = parameters()
        parameters_["fn0"] = 1.
        parameters_["muFO"] = muFOreal

        RealDisturb = ACTIVATOR*np.array([[0.5*mo*gY],[0]])
        wr = RHS_wrench_Cap(qH1, dSH1x, dSH1y, qH2, dSH2x, dSH2y, qH3, dSH3x, dSH3y, qH4, dSH4x,
dSH4y, qO, dSOx, dSOy, vqH1, vdSH1x, vdSH1y, vqH2, vdSH2x, vdSH2y, vqH3, vdSH3x, vdSH3y, vqH4,
vdSH4x, vdSH4y, vqO, vdSOx, vdSOy)

        xdot=struct_SX(states)
        xdot["dSOx"] = vdSOx
        xdot["dSOy"] = vdSOy
        xdot["qO"] = vqO
        xdot["vdSOx"] = (1./mo)*wr[0][0] + RealDisturb[0][0]/mo
        xdot["vdSOy"] = (1./mo)*wr[1][0] - gY + RealDisturb[1][0]/mo
        if ErrorBaric == 0:
                xdot["vqO"] = (1./Jo)*wr[2][0]
        else:
                xdot["vqO"] = (1./Jo)*wr[2][0] + (UncertBaric*0.7/Jo)*(mo*gY+RealDisturb[0][0]-
        RealDisturb[1][0])
        xdot["dSH1x"] = vdSH1x; xdot["dSH1y"] = vdSH1y
        xdot["dSH2x"] = vdSH2x; xdot["dSH2y"] = vdSH2y
        xdot["dSH3x"] = vdSH3x; xdot["dSH3y"] = vdSH3y
        xdot["dSH4x"] = vdSH4x; xdot["dSH4y"] = vdSH4y
        xdot["vdSH1x"] = 0.; xdot["vdSH1y"] = 0.
        xdot["vdSH2x"] = 0.; xdot["vdSH2y"] = 0.
        xdot["vdSH3x"] = 0.; xdot["vdSH3y"] = 0.
        xdot["vdSH4x"] = 0.; xdot["vdSH4y"] = 0.

        f = SXFunction( controldaeIn( x=states, p=parameters, u=controls ), daeOut( ode=xdot ) )
        f.init()

        # Simulation output grid
        M = Nsubsteps #numero di sottointervalli
        T = Tstep #durata di ciascun intervallo
        tgrid = linspace(0,T,M)

        # ControlSimulator will output on each node of the timegrid
        csim = ControlSimulator(f,tgrid)
        csim.setOption("integrator", "cvodes")
        csim.setOption("integrator_options",{"abstol":1e-6,"reltol":1e-6})
        csim.init()
        x0 = [dSOxb, dSOyb, qOb, vdSOxb, vdSOyb, vqOb, dSH1xb, dSH1yb, dSH2xb, dSH2yb, dSH3xb,
dSH3yb, dSH4xb, dSH4yb, vdSH1xb, vdSH1yb, vdSH2xb, vdSH2yb, vdSH3xb, vdSH3yb, vdSH4xb, vdSH4yb]

        # Create input profile
        controls_ = controls.repeated(csim.getInput("u"))
        for k in range(M-1):
        controls_[k,"qH1"] = qH1b; controls_[k,"vqH1"] = vqH1b
        controls_[k,"qH2"] = qH2b; controls_[k,"vqH2"] = vqH2b
```

```
controls_[k,"qH3"] = qH3b; controls_[k,"vqH3"] = vqH3b
controls_[k,"qH4"] = qH4b; controls_[k,"vqH4"] = vqH4b

# Pure simulation
csim.setInput(x0, "x0")
csim.setInput(parameters_,"p")
csim.setInput(controls_,"u")
csim.evaluate()

output = states.repeated(csim.getOutput())

dSOxb = output[-1, "dSOx"]; dSOyb = output[-1, "dSOy"]
qOb = output[-1, "qO"]
vdSOxb = output[-1, "vdSOx"]; vdSOyb = output[-1, "vdSOy"]
vqOb = output[-1, "vqO"]
dSH1xb = output[-1, "dSH1x"]; dSH1yb = output[-1, "dSH1y"]
dSH2xb = output[-1, "dSH2x"]; dSH2yb = output[-1, "dSH2y"]
dSH3xb = output[-1, "dSH3x"]; dSH3yb = output[-1, "dSH3y"]
dSH4xb = output[-1, "dSH4x"]; dSH4yb = output[-1, "dSH4y"]
vdSH1xb = output[-1, "vdSH1x"]; vdSH1yb = output[-1, "vdSH1y"]
vdSH2xb = output[-1, "vdSH2x"]; vdSH2yb = output[-1, "vdSH2y"]
vdSH3xb = output[-1, "vdSH3x"]; vdSH3yb = output[-1, "vdSH3y"]
vdSH4xb = output[-1, "vdSH4x"]; vdSH4yb = output[-1, "vdSH4y"]

dSOb = np.array([[dSOxb], [dSOyb]]); vdSOb = np.array([[vdSOxb], [vdSOyb]])
dSH1b = np.array([[dSH1xb], [dSH1yb]]); vdSH1b = np.array([[vdSH1xb], [vdSH1yb]])
dSH2b = np.array([[dSH2xb], [dSH2yb]]); vdSH2b = np.array([[vdSH2xb], [vdSH2yb]])
dSH3b = np.array([[dSH3xb], [dSH3yb]]); vdSH3b = np.array([[vdSH3xb], [vdSH3yb]])
dSH4b = np.array([[dSH4xb], [dSH4yb]]); vdSH4b = np.array([[vdSH4xb], [vdSH4yb]])

dSOxseq = np.append(dSOxseq, output[vertcat,:, "dSOx"])
dSOyseq = np.append(dSOyseq, output[vertcat,:, "dSOy"])
qOseq = np.append(qOseq, output[vertcat,:, "qO"])
vdSOxseq = np.append(vdSOxseq, output[vertcat,:, "vdSOx"])
vdSOyseq = np.append(vdSOyseq, output[vertcat,:, "vdSOy"])
vqOseq = np.append(vqOseq, output[vertcat,:, "vqO"])

dSH1xseq = np.append(dSH1xseq, output[vertcat,:, "dSH1x"])
dSH1yseq = np.append(dSH1yseq, output[vertcat,:, "dSH1y"])
dSH2xseq = np.append(dSH2xseq, output[vertcat,:, "dSH2x"])
dSH2yseq = np.append(dSH2yseq, output[vertcat,:, "dSH2y"])
dSH3xseq = np.append(dSH3xseq, output[vertcat,:, "dSH3x"])
dSH3yseq = np.append(dSH3yseq, output[vertcat,:, "dSH3y"])
dSH4xseq = np.append(dSH4xseq, output[vertcat,:, "dSH4x"])
dSH4yseq = np.append(dSH4yseq, output[vertcat,:, "dSH4y"])

vdSH1xseq = np.append(vdSH1xseq, output[vertcat,:, "vdSH1x"])
vdSH1yseq = np.append(vdSH1yseq, output[vertcat,:, "vdSH1y"])
vdSH2xseq = np.append(vdSH2xseq, output[vertcat,:, "vdSH2x"])
vdSH2yseq = np.append(vdSH2yseq, output[vertcat,:, "vdSH2y"])
vdSH3xseq = np.append(vdSH3xseq, output[vertcat,:, "vdSH3x"])
vdSH3yseq = np.append(vdSH3yseq, output[vertcat,:, "vdSH3y"])
vdSH4xseq = np.append(vdSH4xseq, output[vertcat,:, "vdSH4x"])
vdSH4yseq = np.append(vdSH4yseq, output[vertcat,:, "vdSH4y"])

for s in range(M):
        qH1bseq = np.append(qH1bseq, qH1b)
        qH2bseq = np.append(qH2bseq, qH2b)
```

```
                qH3bseq = np.append(qH3bseq, qH3b)
                qH4bseq = np.append(qH4bseq, qH4b)
        vqH1bseq = np.append(vqH1bseq, vqH1b)
        vqH2bseq = np.append(vqH2bseq, vqH2b)
        vqH3bseq = np.append(vqH3bseq, vqH3b)
        vqH4bseq = np.append(vqH4bseq, vqH4b)
        RealDisturbseq = np.concatenate((RealDisturbseq, RealDisturb), axis=1)

Nnodes = dSOxseq.shape[0]

C1xseq = np.zeros(Nnodes); C1yseq = np.zeros(Nnodes);
C2xseq = np.zeros(Nnodes); C2yseq = np.zeros(Nnodes)
C3xseq = np.zeros(Nnodes); C3yseq = np.zeros(Nnodes);
C4xseq = np.zeros(Nnodes); C4yseq = np.zeros(Nnodes)

f1xseq = np.zeros(Nnodes); f1yseq = np.zeros(Nnodes);
f2xseq = np.zeros(Nnodes); f2yseq = np.zeros(Nnodes)
f3xseq = np.zeros(Nnodes); f3yseq = np.zeros(Nnodes);
f4xseq = np.zeros(Nnodes); f4yseq = np.zeros(Nnodes)

f1nseq = np.zeros(Nnodes); f1tseq = np.zeros(Nnodes);
f2nseq = np.zeros(Nnodes); f2tseq = np.zeros(Nnodes)
f3nseq = np.zeros(Nnodes); f3tseq = np.zeros(Nnodes);
f4nseq = np.zeros(Nnodes); f4tseq = np.zeros(Nnodes)

for z in range(Nnodes):
        C1xseq[z] = N_single_dist(qH1bseq[z], dSH1xseq[z], dSH1yseq[z], qOseq[z], dSOxseq[z],
dSOyseq[z])[0][0]
        C1yseq[z] = N_single_dist(qH1bseq[z], dSH1xseq[z], dSH1yseq[z], qOseq[z], dSOxseq[z],
dSOyseq[z])[1][0]
        C2xseq[z] = N_single_dist(qH2bseq[z], dSH2xseq[z], dSH2yseq[z], qOseq[z], dSOxseq[z],
dSOyseq[z])[0][0]
        C2yseq[z] = N_single_dist(qH2bseq[z], dSH2xseq[z], dSH2yseq[z], qOseq[z], dSOxseq[z],
dSOyseq[z])[1][0]
        C3xseq[z] = N_single_dist(qH3bseq[z], dSH3xseq[z], dSH3yseq[z], qOseq[z], dSOxseq[z],
dSOyseq[z])[0][0]
        C3yseq[z] = N_single_dist(qH3bseq[z], dSH3xseq[z], dSH3yseq[z], qOseq[z], dSOxseq[z],
dSOyseq[z])[1][0]
        C4xseq[z] = N_single_dist(qH4bseq[z], dSH4xseq[z], dSH4yseq[z], qOseq[z], dSOxseq[z],
dSOyseq[z])[0][0]
        C4yseq[z] = N_single_dist(qH4bseq[z], dSH4xseq[z], dSH4yseq[z], qOseq[z], dSOxseq[z],
dSOyseq[z])[1][0]

        fo1 = N_single_wrench(qH1bseq[z], dSH1xseq[z], dSH1yseq[z], qOseq[z], dSOxseq[z], dSOyseq[z],
vqH1bseq[z], vdSH1xseq[z], vdSH1yseq[z], vqOseq[z], vdSOxseq[z], vdSOyseq[z])
        fo2 = N_single_wrench(qH2bseq[z], dSH2xseq[z], dSH2yseq[z], qOseq[z], dSOxseq[z], dSOyseq[z],
vqH2bseq[z], vdSH2xseq[z], vdSH2yseq[z], vqOseq[z], vdSOxseq[z], vdSOyseq[z])
        fo3 = N_single_wrench(qH3bseq[z], dSH3xseq[z], dSH3yseq[z], qOseq[z], dSOxseq[z], dSOyseq[z],
vqH3bseq[z], vdSH3xseq[z], vdSH3yseq[z], vqOseq[z], vdSOxseq[z], vdSOyseq[z])
        fo4 = N_single_wrench(qH4bseq[z], dSH4xseq[z], dSH4yseq[z], qOseq[z], dSOxseq[z], dSOyseq[z],
vqH4bseq[z], vdSH4xseq[z], vdSH4yseq[z], vqOseq[z], vdSOxseq[z], vdSOyseq[z])

        f1xseq[z] = fo1[0][0][0]; f1yseq[z] = fo1[0][1][0]
        f2xseq[z] = fo2[0][0][0]; f2yseq[z] = fo2[0][1][0]
        f3xseq[z] = fo3[0][0][0]; f3yseq[z] = fo3[0][1][0]
        f4xseq[z] = fo4[0][0][0]; f4yseq[z] = fo4[0][1][0]
```

```
            f1nseq[z] = fo1[1]; f1tseq[z] = fo1[2]
            f2nseq[z] = fo2[1]; f2tseq[z] = fo2[2]
            f3nseq[z] = fo3[1]; f3tseq[z] = fo3[2]
            f4nseq[z] = fo4[1]; f4tseq[z] = fo4[2]

D1xseq = C1xseq + scale*f1xseq; D1yseq = C1yseq + scale*f1yseq
D2xseq = C2xseq + scale*f2xseq; D2yseq = C2yseq + scale*f2yseq
D3xseq = C3xseq + scale*f3xseq; D3yseq = C3yseq + scale*f3yseq
D4xseq = C4xseq + scale*f4xseq; D4yseq = C4yseq + scale*f4yseq

dataSimulation = [T, N, M, scale, TotalTime, ns, ne, flim]
np.savetxt('dataSimulation.dat', dataSimulation)
dataHand = [rf]
np.savetxt('dataHand.dat', dataHand )
dataObject = [lo, ro, mo]
np.savetxt('dataObject.dat', dataObject )
poseObjectList = np.concatenate( ([dSOxseq], [dSOyseq], [qOseq]), axis=0 )
np.savetxt('poseObjectList.dat', poseObjectList )

poseHand1List = np.concatenate( ([dSH1xseq], [dSH1yseq]), axis=0 )
np.savetxt('poseHand1List.dat', poseHand1List )
poseHand2List = np.concatenate( ([dSH2xseq], [dSH2yseq]), axis=0 )
np.savetxt('poseHand2List.dat', poseHand2List )
poseHand3List = np.concatenate( ([dSH3xseq], [dSH3yseq]), axis=0 )
np.savetxt('poseHand3List.dat', poseHand3List )
poseHand4List = np.concatenate( ([dSH4xseq], [dSH4yseq]), axis=0 )
np.savetxt('poseHand4List.dat', poseHand4List )

velHand1List = np.concatenate( ([vdSH1xseq], [vdSH1yseq]), axis=0 )
np.savetxt('velHand1List.dat', velHand1List )
velHand2List = np.concatenate( ([vdSH2xseq], [vdSH2yseq]), axis=0 )
np.savetxt('velHand2List.dat', velHand2List )
velHand3List = np.concatenate( ([vdSH3xseq], [vdSH3yseq]), axis=0 )
np.savetxt('velHand3List.dat', velHand3List )
velHand4List = np.concatenate( ([vdSH4xseq], [vdSH4yseq]), axis=0 )
np.savetxt('velHand4List.dat', velHand4List )

Force1List = np.concatenate( ([C1xseq], [C1yseq], [D1xseq], [D1yseq], [f1xseq], [f1yseq], [f1nseq], [f1tseq]),
axis=0 )
np.savetxt('Force1List.dat', Force1List )
Force2List = np.concatenate( ([C2xseq], [C2yseq], [D2xseq], [D2yseq], [f2xseq], [f2yseq], [f2nseq], [f2tseq]),
axis=0 )
np.savetxt('Force2List.dat', Force2List )
Force3List = np.concatenate( ([C3xseq], [C3yseq], [D3xseq], [D3yseq], [f3xseq], [f3yseq], [f3nseq], [f3tseq]),
axis=0 )
np.savetxt('Force3List.dat', Force3List )
Force4List = np.concatenate( ([C4xseq], [C4yseq], [D4xseq], [D4yseq], [f4xseq], [f4yseq], [f4nseq], [f4tseq]),
axis=0 )
np.savetxt('Force4List.dat', Force4List )
DisturbList = RealDisturbseq
np.savetxt('DisturbList.dat', DisturbList)
```

APPENDICE B:
CODICE MATHEMATICA PER IL VISUALIZZATORE GRAFICO

Read in simulation results

Geometrical data

Hand geometry
```
rf=Import["dataHand.dat"][[1,1]]
0.0100000000000000002
```

Capsule geometry
```
lo=Import["dataObject.dat"][[1,1]]
0.200000000000000011
ro=Import["dataObject.dat"][[2,1]]
0.05000000000000000278
```

Physical data
```
gY=9.81
9.81
mo=Import["dataObject.dat"][[3,1]]
0.299999999999999989
```

Simulation data

Hand
```
poseHand1List=Import["poseHand1List.dat"];
poseHand2List=Import["poseHand2List.dat"];
poseHand3List=Import["poseHand3List.dat"];
poseHand4List=Import["poseHand4List.dat"];
velHand1List=Import["velHand1List.dat"];
velHand2List=Import["velHand2List.dat"];
velHand3List=Import["velHand3List.dat"];
velHand4List=Import["velHand4List.dat"];
```

Object
```
poseObjectList=Import["poseObjectList.dat"];
```

Forces
```
Force1List=Import["Force1List.dat"];
Force2List=Import["Force2List.dat"];
Force3List=Import["Force3List.dat"];
Force4List=Import["Force4List.dat"];
DisturbList=Import["DisturbList.dat"];
```

Parameters
```
Tf=Import["dataSimulation.dat"][[1,1]]
0.200000000000000011
Nsteps=Floor[Import["dataSimulation.dat"][[2,1]]]
20
M=Floor[Import["dataSimulation.dat"][[3,1]]]
20
ns=Floor[Import["dataSimulation.dat"][[6,1]]]
3
ne=Floor[Import["dataSimulation.dat"][[7,1]]]
3
DeltaN=Nsteps-ns-ne
14
TotalTime = Import["dataSimulation.dat"][[5,1]]
4.00000000000000000
s=Import["dataSimulation.dat"][[4,1]]
0.0200000000000000004
flim=Import["dataSimulation.dat"][[8,1]]
10.0000000000000000

Arrowvalue=0.03;
ZeroForce=0.001;
```

Utils

2D Rotation matrix
```
R2D[q_]:=Module[{cq=Cos[q],sq=Sin[q]},{{cq,-sq},{sq,cq}}];
```

Graphical Utility Functions

Capsule Object
```
GraphicsCapsule[dx_,dy_,q_]:=
Graphics[{EdgeForm[{Gray,Thick}],FaceForm[LightGray],
GeometricTransformation[{{Rectangle[{-(lo/2.+ro),-ro}, {(lo/2.+ro),ro},
RoundingRadius->ro]}}, {R2D[q],{dx,dy}}]}]
```

Capsule Object test

```
Manipulate[Show[GraphicsCapsule[dSOx,dSOy,qO],
PlotRange-> {{-2.lo ,2.lo},{-2.lo,2.lo}}],
{{dSOx,0,"dSOx"},-1.lo,1.lo}, {{dSOy,0,"dSOy"},-1.lo,1.lo}, {{qO,0,"qO"},-1.Pi,1.Pi}]
```

Forces

```
GraphicsForce1[C1x_,C1y_,D1x_,D1y_]:=
If[Sqrt[(D1x-C1x)^2+(D1y-C1y)^2]<ZeroForce,
Graphics[{Gray,PointSize[0.001],Point[{C1x,C1y}]}],
Graphics[{Blue,Arrowheads[Arrowvalue],Arrow[{{C1x,C1y},{D1x,D1y}}]}]]

GraphicsForce2[C2x_,C2y_,D2x_,D2y_]:=
If[Sqrt[(D2x-C2x)^2+(D2y-C2y)^2]<ZeroForce,
Graphics[{Gray,PointSize[0.001],Point[{C2x,C2y}]}],
Graphics[{Blue,Arrowheads[Arrowvalue],Arrow[{{C2x,C2y},{D2x,D2y}}]}]]

GraphicsForce3[C3x_,C3y_,D3x_,D3y_]:=
If[Sqrt[(D3x-C3x)^2+(D3y-C3y)^2]<ZeroForce,
Graphics[{Gray,PointSize[0.001],Point[{C3x,C3y}]}],
Graphics[{Blue,Arrowheads[Arrowvalue],Arrow[{{C3x,C3y},{D3x,D3y}}]}]]

GraphicsForce4[C4x_,C4y_,D4x_,D4y_]:=
If[Sqrt[(D4x-C4x)^2+(D4y-C4y)^2]<ZeroForce,
Graphics[{Gray,PointSize[0.001],Point[{C4x,C4y}]}],
Graphics[{Blue,Arrowheads[Arrowvalue],Arrow[{{C4x,C4y},{D4x,D4y}}]}]]

GraphicsWeight[Gx_,Gy_]:=
Graphics[{Black,Arrowheads[Arrowvalue], Arrow[{{Gx,Gy},{Gx,Gy-s*mo*gY}}]}]

GraphicsDisturb[{Gx_,Gy_},{Hx_,Hy_}]:=
Graphics[{Brown,Thickness[0.003],Arrowheads[0.02],
Arrow[{{Gx,Gy},{Gx+s*Hx,Gy+s*Hy}}]}]
```

Force test

```
Manipulate[Show[GraphicsForce1[C1x,C1y,D1x,D1y],
PlotRange-> {{-10. rf,10. rf},{-10.rf,10. rf}}],{{C1x,0,"C1x"},-5.rf,5.rf},
{{C1y,0,"C1y"},-5.rf,5.rf}, {{D1x,0,"D1x"},-5.rf,5.rf}, {{D1y,0,"D1y"},-5.rf,5.rf}]
```

Hand

```
GraphicsHand1[dx_,dy_]:=Graphics[GeometricTransformation[
{{EdgeForm[{Thick,Pink}],FaceForm[LightPink],Disk[{0,0},rf]}},{{dx,dy}}]]

GraphicsHand2[dx_,dy_]:=Graphics[GeometricTransformation[
{{EdgeForm[{Thick,Pink}],FaceForm[LightPink],Disk[{0,0},rf]}},{{dx,dy}}]]

GraphicsHand3[dx_,dy_]:=Graphics[GeometricTransformation[
{{EdgeForm[{Thick,Magenta}],FaceForm[LightMagenta],Disk[{0,0},rf]}},{{dx,dy}}]]

GraphicsHand4[dx_,dy_]:=Graphics[GeometricTransformation[
{{EdgeForm[{Thick,Magenta}],FaceForm[LightMagenta],Disk[{0,0},rf]}},{{dx,dy}}]]
```

Hand test

```
Manipulate[Show[GraphicsHand1[dSHx,dSHy],
PlotRange-> {{-10. rf,10. rf},{-10. rf,10. rf}}],
{{dSHx,0,"dSHx"},0,5. rf},{{dSHy,0,"dSHy"},0,5. rf}]
```

Show results

```
Nnodes=Dimensions[poseHand1List][[2]]
400

Screenshot[i_]:={GraphicsCapsule@@poseObjectList[[{1,2,3},i]],
GraphicsHand1@@poseHand1List[[{1,2},i]],
GraphicsHand2@@poseHand2List[[{1,2},i]],
GraphicsHand3@@poseHand3List[[{1,2},i]],
GraphicsHand4@@poseHand4List[[{1,2},i]],
GraphicsForce1@@Force1List[[{1,2,3,4},i]],
GraphicsForce2@@Force2List[[{1,2,3,4},i]],
GraphicsForce3@@Force3List[[{1,2,3,4},i]],
GraphicsForce4@@Force4List[[{1,2,3,4},i]],
GraphicsWeight@@poseObjectList[[{1,2},i]],
GraphicsDisturb@@{poseObjectList[[{1,2},i]],DisturbList[[{1,2},i]]}}

video=Manipulate[Show[Screenshot[i],PlotRange-> {{-0.7lo,1.7lo},{-0.5lo,1.lo}},
ImageSize-> {800,500}], {i,1,Nnodes,1, AnimationRate->40., RefreshRate->100.,
AutorunSequencing->1,Appearance-> "Open"},
Bookmarks->{start->{i=1},end->{i=Nnodes}}]
```

RINGRAZIAMENTI

Giunto al termine di questa tesi che chiude un intero percorso di studi, desidero ringraziare tutti coloro che mi sono stati accanto, fisicamente e/o moralmente, nel corso di questi anni: senza di loro non sarei potuto arrivare fino a questo punto.

Intendo ringraziare innanzitutto il professore Gabiccini Marco, in qualità di relatore, per la grande disponibilità e l'interesse mostrato al mio lavoro. Oltre ad avermi fornito preziosi insegnamenti, ha saputo guidarmi pazientemente e costantemente in ogni fase di questo lavoro, trasmettendomi fiducia incondizionata nelle mie capacità.

Ringrazio poi tutti i miei familiari per il sostegno illimitato e la presenza continua nonostante la distanza. Hanno sempre creduto in me, rappresentando un porto sicuro in ogni situazione, il focolare domestico in cui ritrovare tranquillità e incitamento.

Infine, last but not least, un ringraziamento particolare va ad Alessia e agli amici, di Pisa e di Petrella, dello studio e del calcetto, della mensa e delle uscite serali, perché mi accettano per come sono, e anzi mi aiutano a migliorare sempre di più. Loro sanno rendere meno brutti i momenti difficili e più belli quelli già piacevoli, ed è per questo che voglio condividere con loro la gioia di questo traguardo.

Grazie!

Antonio

BIBLIOGRAFIA

[1] R. M. Murray, Z. Li e S. S. Sastry, A Mathematical Introduction to, CRC Press, 1994, p. 480.

[2] A. Bicchi, «On the Problem of Decomposing Grasp and Manipulation Forces in Multiple Whole-Limb Manipulation,» *International Journal of Robotics and Autonomous Systems,* vol. 13, n. 2, pp. 127-147, 1994.

[3] M. Gabiccini, E. Farnioli e A. Bicchi, «Grasp Analysis Tools for Synergistic Underactuated Robotic Hands,» *International Journal of Robotics Research,* vol. 32, n. 13, pp. 1553 - 1576, 11 2013.

[4] I. Kao, K. Lynch e J. W. Burdick, «Contact Modeling and Manipulation,» in *Handbook of Robotics*, Springer, 2008, pp. 647 - 669.

[5] M. Gabiccini, E. Farnioli e A. Bicchi, «Grasp and Manipulation Analysis for Synergistic Underactuated Hands Under General Loading Conditions,» in *International Conference on Robotics and Automation*, Saint Paul, MN, USA, 2012.

[6] A. Bicchi, «On the Closure Properties of Robotic Grasping,» *International Journal of Robotics Research,* vol. 14, pp. 319-334, 8 1995.

[7] M. Gabiccini, M. Branchetti e A. Bicchi, «Dynamic Optimization of Tendon Tensions in Biomorphically Designed Hands with Rolling Constraints,» in *International Conference on Robotics and Automation*, Shanghai, China, 2011.

[8] M. Gabiccini, A. Bicchi, D. Prattichizzo e M. Malvezzi, «On the role of hand synergies in the optimal choice of grasping forces,» *Autonomous Robots,* vol. 31, n. 2-3, pp. 235 - 252, 10 2011.

[9] H. Borgstrom, M. A. Batalin, G. S. Sukhatme e W. J. Kaiser, «Weighted Barrier Functions for Computation of Force Distributions with Friction Cone Constraints,» in *International Conference on Robotics and Automation*, Anchorage, AK, USA, 2010.

[10] J. Andersson, «A General-Purpose Software Framework for Dynamic Optimization,» Heverlee, Belgium, 2013.

[11] J. Andersson, J. Gillis e M. Diehl, User Documentation for CasADi v2.2.0, Optimization in Engineering Center (OPTEC): KU Leuven, 2015, p. 39.

[12] J. Nocedal e S. J. Wright, Numerical Optimization, Springer, 2006, pp. 204-219.

[13] M. Gabiccini, «A twist exponential approach to gear generation with general spatial motions,» *Mechanism and Machine Theory,* vol. 44, n. 2, pp. 382 - 400, 2009.

[14] M. R. Cutkosky, R. D. Howe e W. R. Provancher, «Force and Tactile Sensors,» in *Handbook of Robotics*, Springer, 2008, pp. 455 - 476.

[15] D. Prattichizzo e J. C. Trinkle, «Grasping,» in *Handbook of Robotics*, Springer, 2008, pp. 671 - 700.

[16] F. Caccavale e M. Uchiyama, «Cooperative Manipulators,» in *Handbook of Robotics*, Springer, 2008, pp. 701 - 718.