UNIVERSITA' DI PISA
CENTRO INTERDIPARTIMENTALE "E. PIAGGIO"
FACOLTA' DI INGEGNERIA
CORSO DI LAUREA IN INGEGNERIA DELL'AUTOMAZIONE

Sensore di Forza e Coppia

Progetto di Robotica

Supervisori	**Studenti**
Prof. Antonio Bicchi	Davide De Carli
Ing. Marco Gabiccini	Marco Fredianelli
Ing. Giorgio Grioli	

Anno Accademico 2007/2008

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

La manipolazione richiede il contatto tra il robot e un oggetto. In questi casi il contatto è l'interazione fondamentale; la maggior parte dei sistemi robotici non presenta un'adeguata sensibilità oppure non utilizza informazioni riguardanti il contatto. In molte situazioni l'utilizzo di sistemi di visione è una valida risorsa per avere una retroazione sullo stato del *task* che si sta effettuando. Comunque in operazioni di *grasping* o di *pushing* sono richieste informazioni più precise riguardanti la meccanica del contatto. L'approccio tradizionale a questo tipo di problema si sviluppa in due aree di ricerca: *force sensing* e *tactile sensing*.

L'approccio *force sensing* richiede l'impiego di elementi *strain sensors* nel polso, nei riduttori e nella struttura del braccio robotico, che permettano l'identificazione dell'entità delle interazioni e il controllo del contatto utilizzando la misura delle forze. Questo metodo focalizza l'attenzione sull'entità delle forze trasmesse senza considerare la locazione e la geometria del contatto. Invece, adoperando il metodo *tactile sensing*, i sensori forniscono i dettagli del particolare contatto: la posizione del punto di contatto e la distribuzione delle pressioni.

Un approccio alternativo, sviluppato in [1], consiste nella misurazione di forza e coppia per ottenere la posizione del punto di contatto e le componenti della forza. L'obiettivo di questo metodo *force-based* consiste nel misurare la forza agente su un corpo, mentre l'obiettivo dei metodi tradizionali è una misura della distribuzione delle pressioni sull'area di contatto per dedurre successivamente i dettagli riguardanti la posizione del punto di contatto. I sensori utilizzati nel metodo *force-based*

sono posti all'interno della superficie di contatto e vengono anche chiamati *intrinsic tactile sensors*.

In [1] si presentano gli aspetti teorici dell'*intrinsic contact sensing* come base per la progettazione di questo genere di dispositivi analizzando i tre modelli base di contatto: contatto puntuale senza attrito (*point contact without friction*), contatto puntuale con attrito (*point contact with friction*) e contatto *soft finger*. Quest'ultima tipologia di contatto è particolarmente importante perché descrive la situazione più comune che si incontra nei problemi di manipolazione.

Nel capitolo 2 si affronta il problema della percezione del contatto dal punto di vista matematico e meccanico di base e si discutono le ipotesi necessarie per l'esistenza della soluzione che determina la posizione del punto di contatto e la risultante delle forze e dei momenti di contatto, date le misure di un sensore di contatto. Una possibile implementazione di un sensore di contatto consiste in una superficie fissata sopra un sensore di forza/coppia a sei assi che misuri le tre componenti di forza e momento risultanti (per ulteriori informazioni sul sensore utilizzato, si consiglia di consultare [2]). Inoltre si analizzano le scelte effettuate per la definizione delle superfici di contatto sia in forma implicita che in forma parametrica.

Nel terzo capitolo si presenta la soluzione in forma chiusa per superfici ellissoidali, come esposto in [1].

Nel quarto si estende la soluzione iterativa proposta in [1] per superfici generiche, prendendo in considerazione superfici definite in forma parametrica, anziché in forma implicita. Questa scelta è stata effettuata sulla base di considerazioni riguardanti il diffuso utilizzo di superfici definite in forma parametrica (B-spline e NURBS) nei software CAD.

Nel capitolo 5 si presenta un algoritmo utile per l'importazione di un file creato da un programma CAD in Matlab, in modo che i dati importati siano immediatamente utilizzabili negli schemi Matlab-Simulink adoperati nelle simulazioni e nelle prove sperimentali effettuate sul sensore.

Nel capitolo 6 sono analizzati i dati ottenuti nelle simulazioni e sono paragonati i risultati relativi alle diverse soluzioni implementate, evidenziando pregi e difetti di ogni soluzione.

Infine nel capitolo 7 si riportano delle conclusioni sul lavoro svolto, sugli eventuali sviluppi futuri e sulle possibili applicazioni.

Capitolo 2

Formulazione del problema

Il contatto *soft finger* è il caso più generale tra le principali tipologie di contatto. Il contatto puntuale con o senza attrito è un caso particolare del *soft finger contact*. Un contatto di questo tipo avviene tra due corpi non rigidi e possibilmente non elastici. Le interazioni tra i corpi sono considerate essere di compressione, quindi tali azioni sono rivolte verso l'interno dei corpi. Inoltre tale modello non considera le forze di adesione (attrito statico) che si hanno durante il contatto.

Come evidenziato in [1], un punto molto utile per la rappresentazione di un contatto *soft finger* è il *contact centroid* (baricentro di contatto). La risultante delle forze di compressione può essere considerata applicata a tale punto. Questa è una buona approssimazione se l'area di contatto è sufficientemente piccola.

Una possibile implementazione di un sensore di contatto consiste in una superficie (chiamata *fingertip*) fissata sopra un sensore di forza/coppia a sei assi, che misuri le tre componenti di forza $\mathbf{f}$ e momento $\mathbf{m}$ risultanti. Queste ultime sono definite rispetto ad un sistema di riferimento di assi cartesiani avente l'origine coincidente con il centro del sensore (Fig. 2.1).

2.1 Superfici in forma implicita

L'approccio adoperato in [1] si basa su una descrizione in forma implicita della *fingertip surface*:

$$S(\mathbf{r}) = 0, \tag{2.1}$$

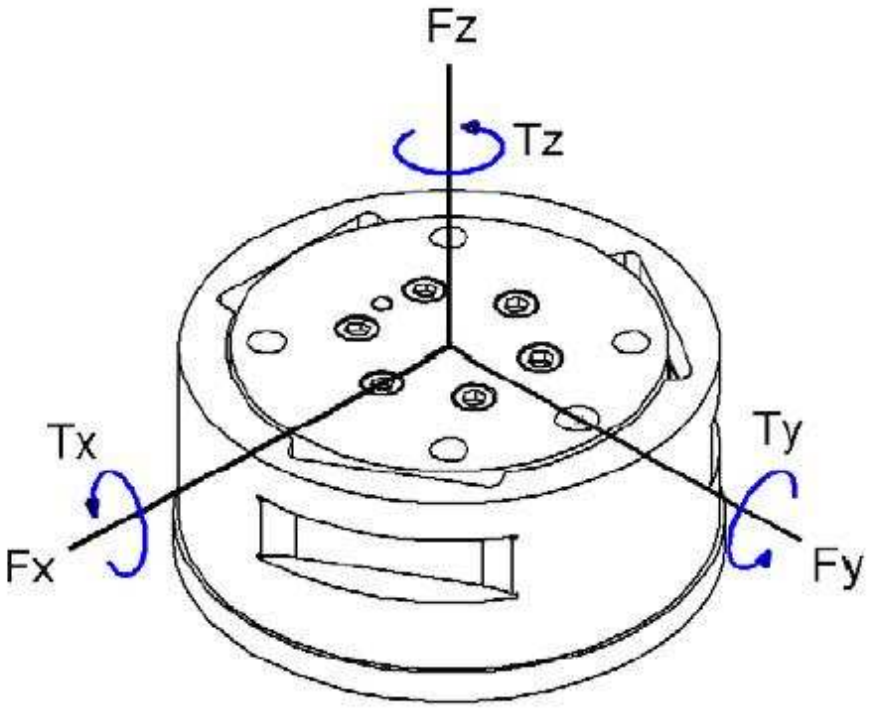

Figura 2.1: Sistema di riferimento del sensore di forza/coppia

dove $\mathbf{r}$ è il vettore posizione definito rispetto al sistema di riferimento mostrato in Figura 1. Se la superficie S ha derivata prima continua, il versore normale $\mathbf{n}$ può essere definito, in ogni punto di S, dalla relazione:

$$\mathbf{n} = \frac{\nabla S(\mathbf{r})}{\|\nabla S(\mathbf{r})\|}, \tag{2.2}$$

dove $\nabla S(\mathbf{r})$ indica il gradiente di S.

Sia $\mathbf{c}$ il vettore delle coordinate del baricentro di contatto e siano $\mathbf{p}$ e $\mathbf{q}$ la forza e il momento, applicati in $\mathbf{c}$, equivalenti al contatto *soft finger*.

Le grandezze misurabili $\mathbf{f}$ e $\mathbf{m}$ permettono di ricavare le incognite $\mathbf{c}$, $\mathbf{p}$ e $\mathbf{q}$ utilizzando le equazioni di equilibrio:

$$\begin{cases} \mathbf{f} = \mathbf{p} \\ \mathbf{m} = \mathbf{q} + \mathbf{c} \times \mathbf{p} \end{cases} \tag{2.3}$$

l'equazione $S(\mathbf{c}) = 0$ e l'equazione che definisce il momento $\mathbf{q}$, che nei contatti *soft finger* è parallelo al versore normale $\mathbf{n}$:

$$\mathbf{q} = K \cdot \frac{\nabla S(\mathbf{c})}{\|\nabla S(\mathbf{c})\|} = K \cdot \mathbf{n}, \tag{2.4}$$

dove K è il modulo del momento di attrito $\mathbf{q}$.

Esplicitando le relazioni dalla (2.1) alla (2.4), si ottiene un sistema non lineare di dieci equazioni in dieci incognite: le nove componenti di $\mathbf{p}$, $\mathbf{q}$ e $\mathbf{c}$ e lo scalare K. Poiché il problema è non lineare, è necessario determinare l'esistenza della soluzione e, nel caso questa esista, l'eventuale unicità. Se la forza $\mathbf{f}$ e il momento $\mathbf{m}$ risultanti sono misure consistenti con il modello di contatto *soft finger*, l'esistenza è verificata. Se esiste una soluzione, la sua unicità è garantita se e solo se la superficie del sensore è convessa, come dimostrato in Appendice 2 di [1].

2.2 Superfici in forma parametrica

Come accennato nell'Introduzione, è stato ritenuto opportuno definire il problema adoperando superfici in forma parametrica, anziché in forma implicita. L'equazione che definisce la superficie è del tipo:

$$\mathbf{S} = \mathbf{S}(u, v) \tag{2.5}$$

dove u e v sono i parametri che definiscono la superficie.

Il versore normale alla superficie è:

$$\mathbf{n} = \frac{\mathbf{S}_u \times \mathbf{S}_v}{\|\mathbf{S}_u \times \mathbf{S}_v\|}, \tag{2.6}$$

dove $\mathbf{S}_u = \frac{\partial \mathbf{S}}{\partial u}$ e $\mathbf{S}_v = \frac{\partial \mathbf{S}}{\partial v}$.

In questo modo le equazioni che definiscono il problema sono

$$\begin{cases} \mathbf{S}(u,v) = \mathbf{c}, \\ \mathbf{f} = \mathbf{p}, \\ \mathbf{m} = \mathbf{q} + \mathbf{c} \times \mathbf{p}, \\ \mathbf{n} = \frac{\mathbf{S}_u \times \mathbf{S}_v}{\|\mathbf{S}_u \times \mathbf{S}_v\|}, \\ \mathbf{q} = K \cdot \mathbf{n}, \end{cases} \tag{2.7}$$

dove le incognite sono $\mathbf{c}$, $\mathbf{q}$, $\mathbf{p}$, u, v e K. La soluzione al problema posto in questi termini è analizzata nel Capitolo 4.

Capitolo 3

Soluzione per superfici ellissoidali

Riferendosi alla formulazione del problema esposta nel paragrafo 2.1, si propone una soluzione del problema considerando superfici ellissoidali: in particolare si trattano piano, sfera e cilindro.

Tali superfici permettono di trovare una soluzione in forma chiusa, focalizzando l'attenzione su forme quadratiche del tipo

$$S(\mathbf{r}) = \mathbf{r}^T \mathbf{A}^T \mathbf{A} \mathbf{r} - R^2 = 0, \tag{3.1}$$

dove $\mathbf{A}$ è una matrice costante dei coefficienti e R è un opportuno fattore di scala.

Poiché il sistema di riferimento rispetto al quale è stata definita la superficie di contatto può essere spostato arbitrariamente, si può assumere, senza perdita di generalità, che $\mathbf{A}$ possa essere in forma diagonale

$$\mathbf{A} = \begin{bmatrix} 1/\alpha & 0 & 0 \\ 0 & 1/\beta & 0 \\ 0 & 0 & 1/\gamma \end{bmatrix} \tag{3.2}$$

con $0 < 1/\alpha \leq 1$, $0 < 1/\beta \leq 1$, $0 < 1/\gamma \leq 1$. In questo modo si limita lo studio a porzioni convesse di forma quadratica; in particolare $2\alpha R$, $2\beta R$ e $2\gamma R$ rappresentano gli assi principali dell'ellissoide considerato.

Sostituendo (3.1) in (2.2), si ottiene

$$\mathbf{n} = \frac{\mathbf{A}^2\mathbf{c}}{\left\|\mathbf{A}^2\mathbf{c}\right\|} \propto \mathbf{q} = K\mathbf{A}^2\mathbf{c} \tag{3.3}$$

e sostituendo (3.3) in (2.3) risulta

$$\mathbf{m} = K\mathbf{A}^2\mathbf{r} + \mathbf{r} \times \mathbf{f} \tag{3.4}$$

Il sistema di equazioni che risolve il problema può essere riscritto nella forma

$$\begin{cases} \Gamma\mathbf{c} = \mathbf{m} \\ \mathbf{c}^T\mathbf{A}^2\mathbf{c} = R^2 \end{cases} \tag{3.5}$$

dove Γ è una matrice 3x3 i cui elementi sono funzione di K e delle componenti della forza misurata $\mathbf{f} = [f_1, f_2, f_3]^T$:

$$\Gamma(K) = \begin{bmatrix} K/\alpha^2 & f_3 & -f_2 \\ -f_3 & K/\beta^2 & f_1 \\ f_2 & -f_1 & K/\gamma^2 \end{bmatrix} \tag{3.6}$$

Il determinante della $\Gamma(K)$ è dato da

$$det\ \Gamma(K) = K(K^2D^2 + \|\mathbf{Af}\|^2), \tag{3.7}$$

dove $D = det\ \mathbf{A}$. La matrice $\Gamma(K)$ è singolare per $K = 0$, quando il momento d'attrito $\mathbf{q}$ è uguale a zero. In questo caso, come esposto nel capitolo 4 di [1], si utilizzano il modello di contatto puntuale con attrito e il metodo del *wrench-axis*; si riportano le equazioni che sono adoperate da questo metodo per individuare il punto di contatto.

$$\mathbf{c} = \mathbf{r}_0 + \lambda\mathbf{f}$$
$$\mathbf{r}_0 = \frac{\mathbf{f} \times \mathbf{m}}{\|\mathbf{f}\|^2}$$

dove λ è un parametro che dipende dalla geometria della superficie considerata. Il valore del parametro λ per superfici ellissoidali è dato da:

$$\lambda = \frac{-\mathbf{f}_*^T\mathbf{r}_0' - \sqrt{(\mathbf{f}_*^T\mathbf{r}_0')^2 - \|\mathbf{f}_*\|^2 (\|\mathbf{r}_0'\|^2 - R^2)}}{\|\mathbf{f}_*\|^2} \tag{3.8}$$

con $\mathbf{f}_* = \mathbf{Af}$ e $\mathbf{r}_0' = \mathbf{Ar}_0$.

Ritornando al caso in cui $K \neq 0$ ed esplicitando la (3.5) rispetto a $\mathbf{c}$, si ottiene

$\mathbf{c} = \Gamma(K)^{-1}\mathbf{m}$; il valore di K che è consistente con l'ipotesi di contatto non adesivo è dato da (cfr. [1]):

$$\begin{aligned} K &= \frac{-sgn(\mathbf{f}^T\mathbf{m})}{\sqrt{2}RD}\sqrt{\sigma + \sqrt{\sigma^2 + 4D^2R^2(\mathbf{f}^T\mathbf{m})^2}} \\ \sigma &= D^2\left\|\mathbf{A}^{-1}\mathbf{m}\right\|^2 - R^2\left\|\mathbf{A}\mathbf{f}\right\|^2 \end{aligned} \tag{3.9}$$

L'implementazione in Matlab/Simulink della soluzione illustrata è presentata successivamente nel Capitolo 6. In particolare sono state considerate le soluzioni relative al caso di superficie sferica, cilindrica e planare, che sono casi particolari di superfici ellissoidali. Infatti, per quanto riguarda la sfera, la matrice $\mathbf{A}$ dei coefficienti risulta:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Nel caso del cilindro

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1/\gamma \end{bmatrix}$$

dove $\gamma \to \infty$. Infine per il piano si ha

$$\mathbf{A} = \begin{bmatrix} 1/\gamma & 0 & 0 \\ 0 & 1/\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

per $\gamma \to \infty$.

Per l'implementazione di questa soluzione in Matlab/Simulink e i risultati delle simulazioni si rimanda al Paragrafo 6.1.

Capitolo 4

Soluzione iterativa

Nella prima parte di questo Capitolo si prende in considerazione una soluzione iterativa del problema esposto nel Paragrafo 2.1, come proposto in [1]. Tale soluzione generalizza le precedenti e permette di considerare superfici non necessariamente di tipo ellissoidale. Nella seconda parte si presenta un algoritmo iterativo che utilizza superfici in forma parametrica piuttosto che in forma implicita.

4.1 Soluzione iterativa per superfici in forma implicita

Dovendo trovare la soluzione di funzioni vettoriali non lineari a più variabili, si riscrivono le equazioni che definiscono il problema nella forma

$$\mathbf{g}(\mathbf{x}) = \mathbf{0}, \tag{4.1}$$

dove

$$\begin{aligned} \mathbf{x}^T &= (x_1, x_2, x_3, x_4)^T = (\mathbf{c}^T, K) \\ g_1(\mathbf{x}) &= x_4 \nabla S_1 - f_2 x_3 + f_3 x_2 - m_1 \\ g_2(\mathbf{x}) &= x_4 \nabla S_2 - f_3 x_1 + f_1 x_3 - m_2 \\ g_3(\mathbf{x}) &= x_4 \nabla S_3 - f_1 x_2 + f_2 x_1 - m_3 \\ g_4(\mathbf{x}) &= S(x_1, x_2, x_3). \end{aligned} \tag{4.2}$$

4.1.1 Metodo del gradiente

La matrice Jacobiana $\mathbf{G}$ associata al problema risulta

$$\mathbf{G}(\mathbf{x}) = \frac{\partial \mathbf{g}}{\partial \mathbf{x}} = \begin{bmatrix} x_4\mathbf{H} - \mathbf{f}_{\otimes} & \nabla S \\ \nabla S^T & 0 \end{bmatrix} \tag{4.3}$$

dove $\mathbf{H}$ è la matrice Hessiana della superficie S ($H_{ij} = \frac{\partial^2 S}{\partial x_i \partial x_j}$, con $i, j = 1, 2, 3$) e $\mathbf{f}_{\otimes}$ è la matrice di cross-prodotto relativa al vettore $\mathbf{f}$, tale che $\mathbf{f}_{\otimes} \cdot \mathbf{c} = \mathbf{f} \times \mathbf{c}$. Poiché la matrice Jacobiana $\mathbf{G}$ non è simmetrica, $\mathbf{g}$ non può essere direttamente considerata come il gradiente di una funzione di tipo quadratico da minimizzare. Comunque, se consideriamo l'equazione

$$\mathbf{G}^T\mathbf{g}(\mathbf{x}) = 0, \tag{4.4}$$

si ha che tutti gli zeri di $\mathbf{g}$ sono anche zeri di $\mathbf{G}^T\mathbf{g}$ e $\mathbf{G}^T\mathbf{g}$ è il gradiente associato alla funzione scalare definita positiva $V = \frac{1}{2}\mathbf{g}^T\mathbf{g}$ il cui minimo assoluto è la soluzione desiderata.

Si utilizza il metodo numerico del *gradient descent* la cui legge di aggiornamento al passo k è definita dalla relazione

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \lambda \nabla V(\mathbf{x}) = \mathbf{x}_k - \lambda \mathbf{G}_k^T \mathbf{g}_k. \tag{4.5}$$

Questo metodo è localmente asintoticamente convergente alla soluzione desiderata scegliendo un valore opportuno della costante λ; infatti, per λ abbastanza piccoli nella direzione di *steepest descent* della funzione V, quest'ultima risulta sempre decrescente fino al raggiungimento del minimo.

La locale e asintotica convergenza dell'algoritmo prevede che la condizione iniziale $\mathbf{x}_0$ sia abbastanza vicina al punto di contatto che deve essere individuato.

Riferendosi al problema affrontato, emergono due principali problemi: l'eventuale presenza di soluzioni multiple che annullano i residui $\mathbf{g}^T\mathbf{g}$ e l'eventuale convergenza verso "false" soluzioni individuate da minimi locali della funzione obiettivo.

4.1.2 Metodo di Levenberg-Marquardt

Al fine di ottenere una maggiore robustezza dell'algoritmo in presenza di soluzioni multiple il cui residuo sia diverso da zero, si è preso in considerazione il metodo di

Levenberg-Marquardt. Questo metodo numerico utilizza una direzione di ricerca d_k che è soluzione delle equazioni lineari

$$(\mathbf{J}(x_k)^T\mathbf{J}(x_k) + \lambda_k\mathbf{I})d_k = -\mathbf{J}(x_k)\mathbf{F}(x_k) \tag{4.6}$$

dove $\mathbf{F}(x_k)$ è la funzione dei residui, $\mathbf{J}(x_k)$ è il Jacobiano della funzione da minimizzare e λ_k è un parametro scalare che controlla sia l'intensità che la direzione di $d_k = x_{k+1} - x_k$. Quando λ_k è zero, la direzione d_k è identica a quella del metodo di Gauss-Newton. Al crescere di λ_k, d_k tende a un vettore di zeri e fornisce la direzione di *steepest descent*. Il termine λ_k può quindi essere controllato per assicurare la convergenza anche quando l'efficienza del metodo di Gauss-Newton viene meno, ad esempio nel caso in cui $\mathbf{J}(x_k)$ risulti singolare o malcondizionata. Il metodo di Gauss-Newton è comunque più efficiente quando il residuo è zero nella soluzione; poiché non è sempre possibile sapere a priori se il residuo corrispondente alla soluzione sia nullo, la occasionale minore efficienza del metodo di Levenberg-Marquardt è compensata dalla sua maggiore robustezza.
L'utilizzo di questo metodo di ottimizzazione e i confronti con i metodi già descritti sono esposti nel Capitolo 6.

4.2 Soluzione iterativa per superfici in forma parametrica

In riferimento al Paragrafo 2.2 è possibile descrivere il problema adoperando l'equazione 4.1, dove

$$\begin{aligned}
\mathbf{x}^T &= (x_1, x_2, x_3)^T = (u, v, K) \\
\mathbf{c}^T &= (c_1, c_2, c_3)^T = \mathbf{c}(u, v) \\
\mathbf{n}^T &= (n_1, n_2, n_3)^T = \mathbf{n}(u, v) \\
g_1(\mathbf{x}) &= x_3n_1 - f_2c_3 + f_3c_2 - m_1 \\
g_2(\mathbf{x}) &= x_3n_2 - f_3c_1 + f_1c_3 - m_2 \\
g_3(\mathbf{x}) &= x_3n_3 - f_1c_2 + f_2c_1 - m_3.
\end{aligned} \tag{4.7}$$

$\mathbf{n}$ è il versore normale alla superficie $S(u, v)$ definito in 2.6 e $\mathbf{c}$ è il punto di contatto. Come si può notare, il sistema considerato è costituito da tre equazioni in tre incognite. Il problema così formulato può essere ricondotto, come è stato fatto nel paragrafo precedente, a un problema di ottimizzazione. Anche in questo caso emergono le problematiche già citate e soprattutto la presenza di soluzioni multiple che annullano i residui $\mathbf{g}^T\mathbf{g}$. Si è quindi deciso di ricondurre il problema ad una minimizzazione di una funzione multivariabile con vincolo non lineare:

$$\begin{cases} \min_x \left(\mathbf{g}^T\mathbf{g}\right) \\ v(\mathbf{x}) \leqslant 0 \\ \mathbf{lb} \leqslant \mathbf{x} \leqslant \mathbf{ub} \end{cases} \tag{4.8}$$

dove $v(\mathbf{x})$ è il vincolo non lineare che impone che la forza di contatto $\mathbf{f}$ sia di compressione e diretta verso l'interno della superficie. In particolare si ha $v(\mathbf{x}) = \mathbf{f}^T\mathbf{n}(u, v)$. Inoltre **lb** e **ub** rappresentano rispettivamente il vincolo inferiore e superiore sui parametri u e v che definiscono la superficie ($u \in [0, 1]$, $v \in [0, 1]$).

Dunque è stato adoperato l'Optimization Toolbox incluso in Matlab e in particolare la funzione `fmincon`. La funzione `fmincon` risolve il problema di ottimizzazione utilizzando il metodo dei moltiplicatori di Lagrange; tale metodo consente di trovare massimi e minimi di una funzione di più variabili soggetta a una o più vincoli ed è lo strumento di base nell'ottimizzazione non lineare vincolata. L'algoritmo sviluppato in questo modo è efficace fuori linea, ma l'implementazione in tempo reale con il sensore non è stata possibile a causa dell'eccessivo costo computazionale della funzione `fmincon`.

Al fine di risolvere il problema dell'imposizione del vincolo non lineare mantenendo tempi di calcolo accettabili in relazione alla frequenza massima di campionamento del sensore (50 Hz), è stata adoperata la funzione Matlab `lsqnonlin` che implementa il metodo numerico di Levenberg-Marquardt. Questa funzione non permette l'imposizione diretta di vincoli, siano essi lineari o non lineari e quindi si è ovviato al problema della molteplicità della soluzione imponendo un initial guess opportuno in relazione al bacino di attrazione della soluzione desiderata. In particolare è stata considerata una griglia quadrata di punti sulla superficie NURBS e in ognuno di essi

è stata valutata la normale locale prima di iniziare la prova sperimentale. Successivamente vengono forniti come valori iniziali dei parametri della funzione `lsqnonlin` quelli corrispondenti ai punti in corrispondenza dei quali risulta essere minimo il prodotto scalare della forza misurata dal sensore con il versore normale calcolato precedentemente fuori linea. L'implementazione di questo algoritmo è ripresa nel Capitolo 6.

Capitolo 5

Importazione di file CAD in Matlab

Al fine di rendere più rapida la definizione di superfici NURBS, anche complesse, in Matlab è stata realizzata una procedura che permette di costruire tale superficie interfacciandosi con un programma CAD. Il programma CAD che è stato adoperato è ProEngineer, standard industriale per la progettazione meccanica. In questo modo l'utente può definire una superficie qualsiasi, purchè si rispettino le condizioni necessarie per l'esistenza della soluzione discusse nel Capitolo 2. Tale procedura non può essere automatizzata completamente in quanto i vari passi da eseguire devono essere contestualizzati dall'utente rispetto alla superficie trattata; questo aspetto sarà più chiaro in seguito quando verrà spiegato l'iter da seguire per l'importazione. Si riporta dunque il procedimento da seguire dalla progettazione della superficie sul programma CAD alla sua definizione su MatLab.

Come primo passo deve essere definita la superficie desiderata utilizzando ProEngineer; quindi il file creato deve essere esportato utilizzando il formato STEP (*.stp); la scelta di questo tipo di formato risiede nella sua compatibilità con la maggior parte dei programmi CAD in commercio e soprattutto con il software PTC Geomagic, suite della casa produttrice di ProEngineer che permette di trattare ed elaborare nuvole di punti ed utilizzato dagli esperti di *reverse engineering*. Successivamente il file .stp è stato aperto con Geomagic con lo scopo di creare una griglia ordinata di punti che interpolano la superficie; per far ciò si utilizza il comando `cross section`

che permette di dividere la superficie in esame in sezioni delle quali può essere specificato il numero, la distanza reciproca e il piano rispetto alle quali sono parallele (*align plane*). In questo modo viene salvato un oggetto che rappresenta l'insieme delle curve che sono intersezione della superficie con le varie sezioni (di default il nome di tale oggetto è Curves). A questo punto per esportare tale oggetto è sufficiente fare click con il tasto destro sull'oggetto stesso e selezionare Save...; nella finestra di salvataggio infine scegliere il formato Wavefront File (`*.obj`). La scelta di quest'ultimo formato di esportazione è stata dettata dalla sua pulizia e dalla sua essenzialità; infatti i punti delle varie sezioni vengono salvati ordinatamente e con lo stesso numero di punti per ogni sezione, il che permette una più semplice elaborazione degli stessi nonostante le difficoltà che nascono dall'esigenza di adattare la procedura a superfici qualsiasi. La fase appena descritta, come già accennato, rende poco automatizzabile la procedura di creazione di una griglia di punti interpolanti; infatti le sezioni, e quindi il piano di allineamento che le genera, devono essere tali da intersecare la superficie una sola volta, cioè l'intersezione tra la superficie e il piano che identifica la sezione deve poter essere descritta da una sola curva. Il suddetto problema si ha solamente nel caso in cui la superficie considerata non sia convessa. Inoltre il piano di allineamento deve essere uno dei tre piani identificati dagli assi cartesiani in quanto i punti relativi alla stessa sezione devono avere almeno una coordinata fissa.

Il file *.obj così costruito può essere utilizzato per costruire la struttura che definisce la NURBS in Matlab. A questo scopo si è implementata una funzione (`costruzione_nurbs('nome_file.obj')`che riceve come ingresso il nome del file *.obj e restituisce la struttura NURBS che rappresenta la superficie; tale rappresentazione è dovuta all'utilizzo di NURBS Toolbox per la definizione delle Splines. Di seguito si riporta lo pseudo-codice Matlab utile a chiarire l'implementazione della funzione considerata:

```
function y=costruzione_nurbs(nome_file)
pnt=ordinamento_punti(nome_file);
[P,U,V,uk,vl] = bspline_surf_interp(pnt);
nurbs = nrbmak(P,{U,V});
y=nurbs;
end
```

La funzione `ordinamento_punti()` riceve in ingresso il file .obj e restituisce una matrice di dimensioni NxMx3 dove N è il numero delle sezioni considerate in Geomagic ed M è il numero di punti appartenenti ad ogni sezione. Questa matrice viene poi passata come argomento alla funzione `bspline_surf_interp()` che, utilizzando un algoritmo di interpolazione proposto nel The NURBS Book ([3]), restituisce il poliedro di controllo definito dai punti contenuti nella matrice P e i vettori dei nodi U e V relativi rispettivamente alle due coordinate parametriche u e v di una superficie NURBS di quarto grado. Quest'ultima funzione è stata realizzata presso il Dipartimento di Ingegneria Meccanica, Nucleare e della Produzione dall'Ing. Andrea Bracci e dall'Ing. Marco Gabiccini. I punti del poliedro di controllo e i vettori dei nodi sono successivamente utilizzati dalla funzione `nrbmak()` del Nurbs Toolbox che restituisce la struttura NURBS desiderata.

Si riportano gli *screenshots* relativi alla procedura appena esposta applicata a una superficie che modella il polpastrello di un dito. In figura 5.1 si presenta la superficie realizzata in ProEngineer e la sua esportazione utilizzando il formato STEP.

La figura 5.2 rappresenta il file .stp che è stato aperto con Geomagic e utilizzando il comando `cross section` la superficie considerata è sezionata specificando il numero dei piani di sezione, la distanza reciproca e il piano rispetto al quale essi sono paralleli (*align plane*).

La figura 5.3 mostra il salvataggio dell'oggetto che rappresenta l'insieme delle curve che costituiscono l'intersezione della superficie con i piani di sezione e l'esportazione di tale oggetto nel formato Wavefront File (.obj).

Infine si riporta il plot ottenuto adoperando il comando `nrbplot()` contenuto all'interno del NURBS Toolbox della superficie in forma parametrica importata in Matlab (5.4).

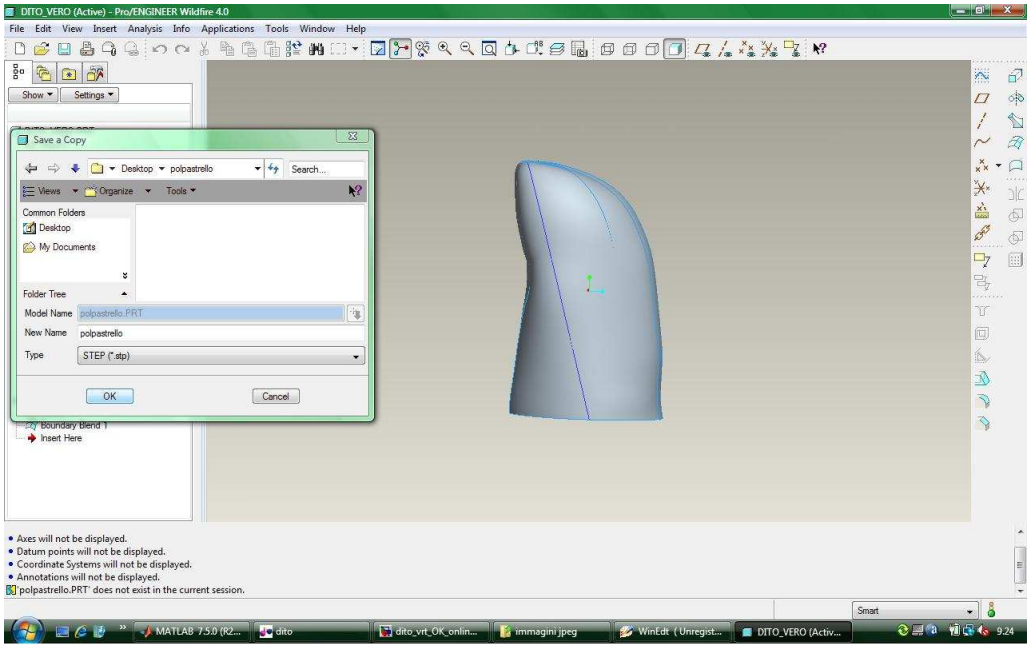

Figura 5.1: Modello CAD ProEngineer del polpastrello di un dito

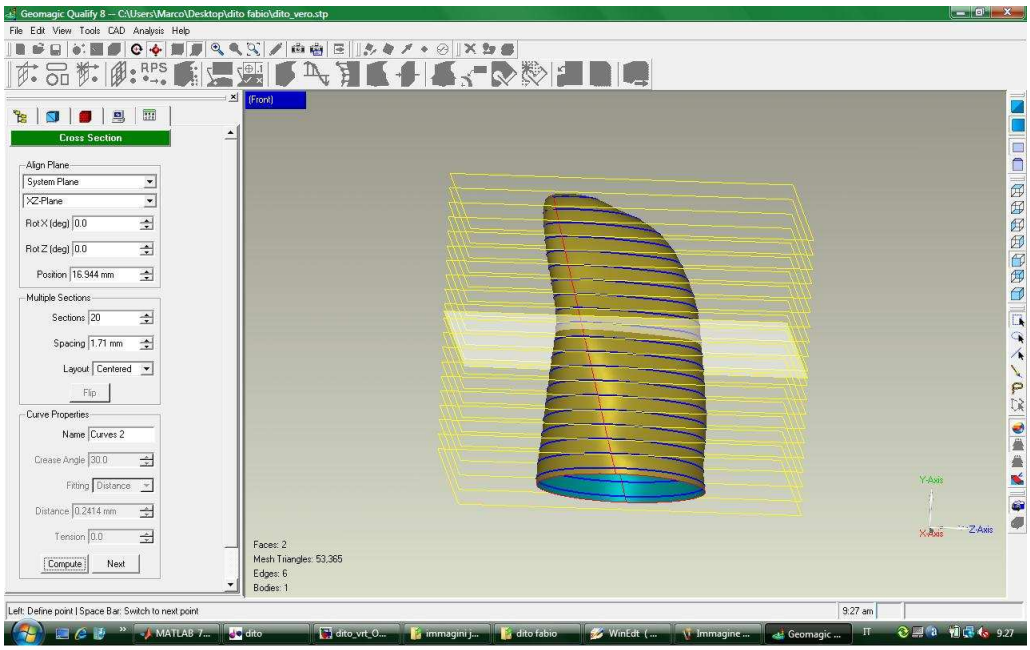

Figura 5.2: Importazione modello CAD in Geomagic e scelta delle sezioni

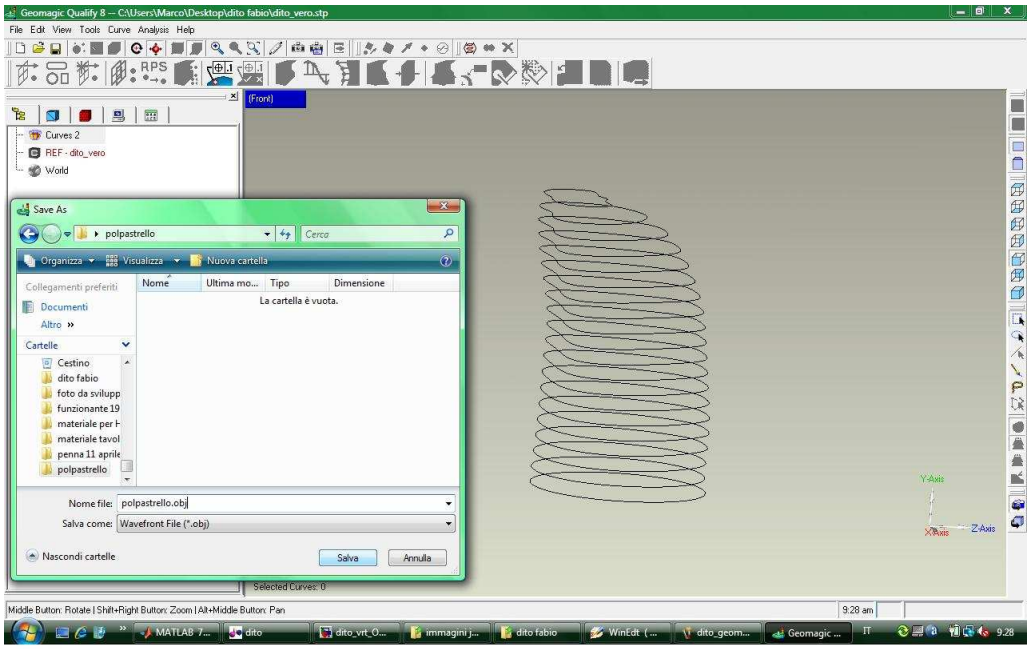

Figura 5.3: Salvataggio nuvole di punti in formato .obj

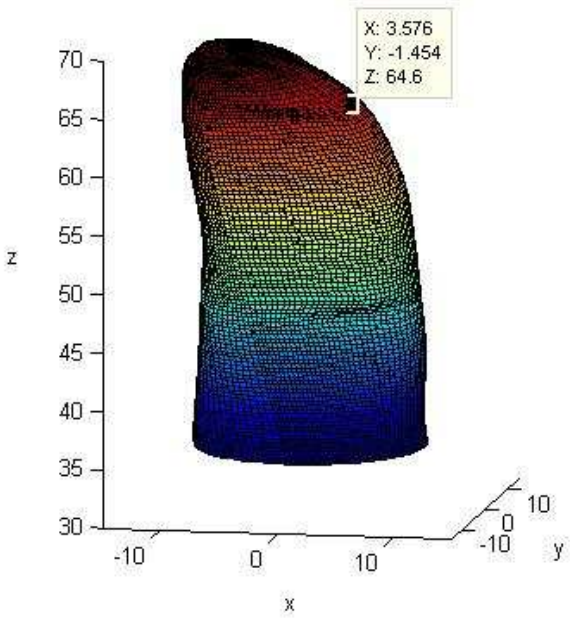

Figura 5.4: Superficie NURBS del polpastrello realizzata in Matlab

Capitolo 6

Implementazione e Simulazioni

In questo Capitolo si illustrano nel dettaglio gli algoritmi presentati in precedenza e le loro implementazioni in Matlab/Simulink. Inoltre sono analizzati i risultati ottenuti nelle simulazioni e con le prove sperimentali.

6.1 Implementazione della soluzione in forma chiusa per superfici ellissoidali

Riferendosi alla soluzione esposta nel paragrafo 3, si propone una implementazione della soluzione considerando superfici ellissoidali: in particolare si trattano piano, sfera e cilindro.
Il modello Simulink utilizzato è presentato in figura 6.1.

Il blocco celeste implementa la S-Function che riceve i dati dal sensore di forza/coppia attraverso la porta seriale RS-232, elabora il pacchetto ricevuto e restituisce in output le tre componenti di forza e momento misurati e un bit di validità (blocco verde) utile per capire se il dato è valido oppure no. L'unico parametro che riceve in ingresso questo blocco è il *sample time*, cioè il tempo di campionamento della S-Function. Dalle prove effettuate e sulla base delle caratteristiche costruttive del sensore, si è individuata una frequenza di funzionamento del sensore di 50 Hz; quindi, per avere una corretta sincronizzazione con il sensore, è opportuno impostare un *sample time* di 0.02 secondi. Inoltre le uscite fornite dal sensore sono state tarate

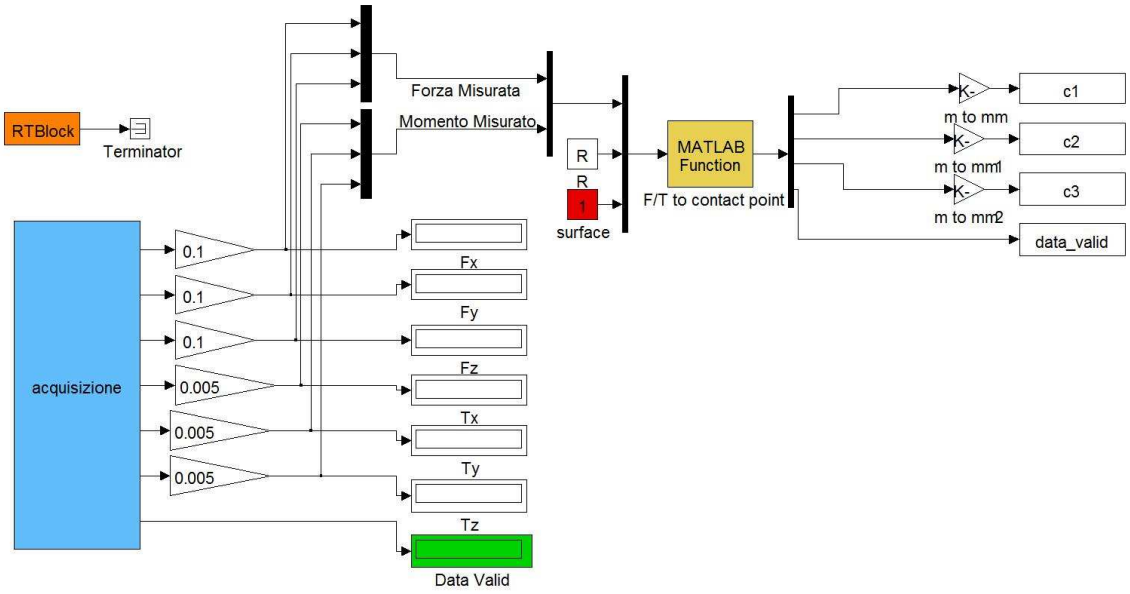

Figura 6.1: Schema Simulink soluzione in forma chiusa per superfici ellissoidali

opportunamente in maniera sperimentale per ottenere la forza misurata in N e il momento misurato in $N \cdot m$.

Il blocchetto arancione implementa un algoritmo che permette l'esecuzione in tempo reale di modelli Simulink. La S-Function scritta in C++ che implementa tale algoritmo è disponibile su Matlab Exchange ed è stato implementato dall'Ing. Leonardo Daga. Questo blocco è basato sul semplice concetto che, per far sì che la simulazione abbia una temporizzazione real-time, il tempo di ciclo (il tempo di cui Simulink ha bisogno per calcolare un passo di simulazione) debba essere inferiore al *sample time* del risolutore numerico di Simulink. Se questa ipotesi non è rispettata, non è possibile avere una simulazione real-time. Il blocco in questione mantiene semplicemente l'esecuzione della simulazione di Simulink in linea con il *time flow*; se il tempo di ciclo è inferiore al *simulation step*, il blocco aspetta il tempo necessario per giungere al termine del *simulation step* stesso, lasciando la CPU disponibile per altri processi correnti.

Proseguendo nell'analisi del modello Simulink illustrato, si trovano le costanti R e surface. La costante R rappresenta il raggio della superficie considerata nel caso di

sfera e cilindro e nel caso del piano è l'offset al quale si trova il piano stesso lungo l'asse z del sensore. La costante surface serve per comandare lo switch interno al blocchetto Matlab Function in modo da scegliere in maniera opportuna la superficie da trattare e in particolare deve assumere il valore 1 nel caso del cilindro, 2 nel caso del piano e 3 nel caso della sfera.
Infine il blocco giallo implementa la soluzione per le tre superfici considerate. Per completezza si riporta il codice del file .m utilizzato dal blocchetto considerato.

```
 function [y] = sup_semplici(u) %(f, m, R, surface)

%ingressi
f=u(1:3);
m=u(4:6);
R=u(7);
surface=u(8);

switch surface
    case 1
        % CILINDRO
        %matrice dei coefficienti del cilindro
        Ac=[1 0 0;0 1 0;0 0 0];

        %implementazione della soluzione
        f_ort=[f(1) f(2) 0]';
        m_par=abs(m(3));
        norma=norm(f_ort);
        K= -(f'*m)/(sqrt(R^2*norma^2-m_par^2));

        if K==0
            %metodo wrench axis
            r0=(cross(f,m))/((norm(f))^2);
            f_p=Ac*f;
            r0_p=Ac*r0;
            lambda = (-f_p'*r0_p - sqrt((f_p'*r0_p)^2 -
                     ((norm(f_p))^2)*(((norm(r0_p))^2)-R^2)))/((norm(f_p))^2);
            r=r0+lambda*f;
            valid=1;
            c=r;
            y(1:3)=c; y(4)=valid;
        else
            c=(K^2*m_par + cross((K*f_ort),m) + (f'*m)*f)/(K*(norm(f_ort))^2);
            valid=1;
            y(1:3)=c; y(4)=valid;
        end
```

```
    case 2
        % PIANO
        % Matrice dei coefficienti del piano
        Ap=[0 0 0;0 0 0;0 0 1];

        %implementazione della soluzione
        f_par=[0 0 f(3)]';
        valid=1;
        c=(cross(f_par,m) + R*norm(f_par)*f)/((norm(f_par))^2);
        y(1:3)=c; y(4)=valid;

    case 3
        %SFERA
        offset=[0 0 0.027]'; %posizione del centro della sfera rispetto al punto sensore
        m_offset=cross(f,offset);
        m = m+m_offset;

        % Matrice dei coefficienti della sfera
        As=eye(3);

        %implementazione della soluzione
        sigma = (norm(m))^2 - R^2*(norm(f))^2;
        K = ((-sign(f'*m)) * sqrt(sigma + sqrt(sigma^2 + 4*R^2*(f'*m)^2)))/(R*sqrt(2));
        if K==0
            lambda=- (sqrt(R^2 - ((norm(cross(f,m)))^2)/((norm(f))^4))) / (norm(f));
            r0=(cross(f,m))/((norm(f))^2);
            r=r0+lambda*f;  %r=asse di wrench
            valid=1;
            c=r;
        else
            valid=1;
            K*(K^2+(norm(f))^2);
            c = (K^2*m + K*cross(f,m) + (f'*m)*f) / (K*(K^2+(norm(f))^2));
        end

% uscite
y(1:3)=c; y(4)=valid;

end
```

L'analisi dei risultati ottenuti adoperando l'algoritmo esposto è presentata nel paragrafo 6.4.

6.2 Implementazione della soluzione iterativa per superfici ellissoidali

In questo paragrafo si prende in considerazione una soluzione iterativa del problema esposta in 4.1.1 e proposta in [1].

Lo schema Simulink è presentato in figura 6.2. Il blocco Matlab Function implemen-

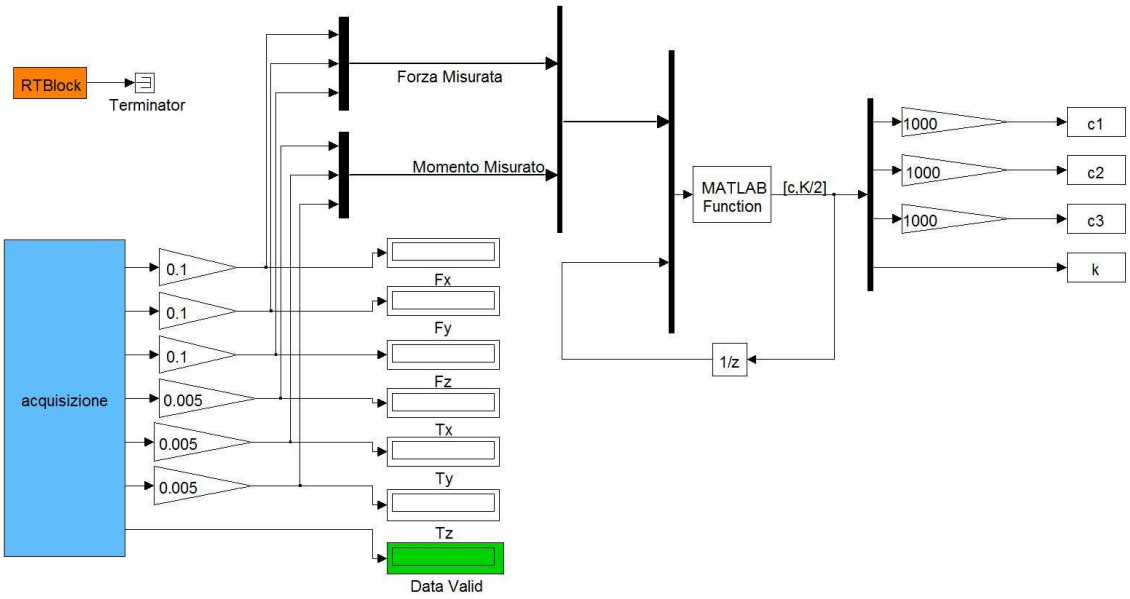

Figura 6.2: Schema Simulink soluzione iterativa con il metodo del gradiente per superfici ellissoidali

ta l'algoritmo iterativo con il metodo del gradiente; l'unica differenza significativa rispetto allo schema precedente 6.1 è la presenza del blocco $1/z$ sul filo che retroaziona la soluzione trovata al passo precedente, necessaria per la legge di aggiornamento definita dal metodo iterativo. Si riporta di seguito il codice in formato .m.

```
function [y] = iterativo(u)

%ingressi
f=u(1:3);
m=u(4:6);
x=u(7:10);
```

```
lambda=1;           %step lenght
R=0.01;             % raggio cilindro
err_star= 4e-4 ;
err = 1;
cont = 0;

while ( (err > err_star) & (cont < 10000) )

    % calcolo della g
    g(1) = 2*x(4)*x(1) - f(2)*x(3) + f(3)*x(2) - m(1);
    g(2) = 2*x(2)*x(4) - f(3)*x(1) + f(1)*x(3) - m(2);
    g(3) = f(2)*x(1) - f(1)*x(2) - m(3);
    g(4) = x(1)^2 + x(2)^2 - R^2;

    % calcolo di G^T*g
    Gtg(1) = 2*x(4)*g(1) - f(3)*g(2) +f(2)*g(3) +2*x(1)*g(4);
    Gtg(2) = f(3)*g(1) + 2*x(4)*g(2) - f(1)*g(3) + 2*x(2)*g(4);
    Gtg(3)=-f(2)*g(1)+ f(1)*g(2);
    Gtg(4)=2*x(1)*g(1)+2*x(2)*g(2);

    % aggiornamento
    x_new(1) =  x(1) - lambda * Gtg(1);
    x_new(2) =  x(2) - lambda * Gtg(2);
    x_new(3) =  x(3) - lambda * Gtg(3);
    x_new(4) =  x(4) - lambda * Gtg(4);

    err = sqrt((x_new(1)-x(1))^2 + (x_new(2)-x(2))^2 + (x_new(3)-x(3))^2);
    cont = cont + 1;

end
%uscite
y(1)=x_new(1);   % c1
y(2)=x_new(2);   % c2
y(3)=x_new(3);   % c3
y(4)=x_new(4);   % k
end
```

L'algoritmo sopra riportato richiede l'impostazione del parametro `lambda`, cioè la definizione della lunghezza del passo (*step length*) utilizzata dalla legge di aggiornamento. Dalle simulazioni si evince che, adoperando più superfici di dimensioni differenti, non può essere scelto a priori uno *step length* che sia efficiente in qualsiasi caso. Anche a causa di questi motivi si è preferito utilizzare metodi come quelli di Gauss-Newton e Levenberg-Marquardt.

L'analisi dei risultati ottenuti adoperando il metodo esposto e i confronti con gli altri approcci analizzati sono presentati nel paragrafo 6.4.

6.3 Implementazione della soluzione iterativa per superfici generiche

In questo paragrafo l'algoritmo sviluppato è testato considerando delle superfici generiche; inizialmente sono eseguite delle prove fuori linea imponendo punto di contatto, momento di attrito e misure dei sensori e successivamente l'algoritmo utilizzato è validato in linea. Le superfici considerate devono rispettare, quantomeno nelle zone in cui si suppone avvenga il contatto, le condizioni necessarie per non avere problemi riguardanti la molteplicità e l'esistenza della soluzione: convessità della superficie e derivata prima continua.
Come già spiegato nel Paragrafo 4.2, per la risoluzione del problema di minimizzazione è stato utilizzata la funzione Matlab `lsqnonlin()`, piuttosto che la funzione `fmincon()`, per le maggiori prestazioni in termini di tempo di calcolo e di velocità di convergenza. L'utilizzo di `lsqnonlin()` conduce alla necessità di ovviare al problema della molteplicità della soluzione imponendo un initial guess opportuno in relazione al bacino di attrazione della soluzione desiderata. In particolare è stata considerata una griglia quadrata di punti sulla superficie NURBS e in ognuno di essi è stata valutata la normale locale prima di iniziare la prova sperimentale. Successivamente vengono forniti come valori iniziali dei parametri della funzione `lsqnonlin` quelli corrispondenti ai punti in corrispondenza dei quali risulta essere minimo il prodotto scalare della forza misurata dal sensore con il versore normale calcolato precedentemente fuori linea. Si riporta di seguito la parte di codice che si occupa della costruzione della matrice A i cui elementi sono versori normali alla superficie dei punti appartenenti alla griglia quadrata che è stata considerata. Tale funzionalità viene eseguita durante l'inizializzazione dello schema Simulink (figura 6.3) in modo da non gravare sulle prestazioni dell'algoritmo.

```
i=1;
j=1;
```

```
for u=0:0.01:1
    for v=0:0.01:1
       [pnt, jac] = nrbdeval(nurbs, dnurbs, {u, v});
        c_u = jac(:,1){:};
        c_v = jac(:,2){:};
        num = cross( c_u, c_v );
        den = norm( num );
        n = num / den;

        A{i,j}=n;
        j=j+1;
    end
    i=i+1;
    j=1;
end
```

In riferimento al codice sopra riportato, `nurbs` e `dnurbs` sono strutture NURBS costruite precedentemente e rappresentano rispettivamente la superficie considerata (costruita con la funzione `nrbmak()` di Nurbs Toolbox) e la sua derivata prima (costruita con la funzione `nrbderiv(nurbs)`). La funzione `nrbdeval()` calcola le derivate parziali rispetto ai parametri u e v; `c_u` e `c_v` sono tali derivate parziali. Infine `n` è il versore normale alla superficie valutato nel punto parametrizzato dai particolari valori di u e v; `A` è una matrice i cui elementi sono una cella contenente le componenti del versore normale.

In riferimento allo schema Simulink 6.3, i blocchi evidenziati in arancione sono i filtri tempo-discreti che agiscono sulle uscite di forza e coppia del sensore riducendo le oscillazioni delle misure. Il blocco azzurro rappresenta la Matlab Function che implementa l'algoritmo di minimizzazione riportato di seguito:

```
function [y] = contact_point(in)

f=in(1:3);
m=in(4:6);

u_prec=in(7);
v_prec=in(8);
K=in(9);

global dnurbs;
global nurbs;
global A;
```

```
% definizione dei bound su  u, v, k
LB=[0 0 -1000];
UB=[0.999 0.999 1000];

%calcolo initial guess
vinc=norm(f);
u_0=0;
v_0=0;
i=1;
j=1;
for u=0:0.01:1
    for v=0:0.01:1
        vinc_new=f'*A{i,j};
        if vinc_new<vinc
            vinc=vinc_new;
            u_0=u;
            v_0=v;
        end
        j=j+1;
    end
    i=i+1;
    j=1;
end

%LEVENBERG MARQUARDT
options=optimset;
options.MaxFunEvals=10;
options.MaxIter=10;
options.Display='off';
x_min = lsqnonlin(@funzione,[u_0,v_0,K],LB,UB,options);

    % funzione da minimizzare
    function g = funzione(ingresso)
        u_p = ingresso(1);
        v_p = ingresso(2);
        K_p = ingresso(3);
        [pnt, jac] = nrbdeval(nurbs, dnurbs, {u_p, v_p});
        c_u = jac(:,1){:};
        c_v= jac(:,2){:};
        num = cross( c_u, c_v );
        den = norm( num );
        n = num / den;
        c = nrbeval(nurbs,{u_p,v_p});
        g = K_p*n + cross(c,f) - m;
    end

c_new = nrbeval(nurbs,{x_min(1),x_min(2)});
```

```
% uscite y(1) -> y(6)
y(1) = c_new(1);
y(2) = c_new(2);
y(3) = c_new(3);
y(4) = x_min(1);
y(5) = x_min(2);
y(6) = x_min(3);

%calcolo del vettore normale "finale" per VRT
[pnt, jac] = nrbdeval(nurbs, dnurbs, {u_p, v_p});
c_u = jac(:,1){:};
c_v= jac(:,2){:};
num = cross( c_u, c_v );
den = norm( num );
n_momento = num / den;

% uscite y(7) -> y(9)
y(7)=n_momento(1);
y(8)=n_momento(2);
y(9)=n_momento(3);

end
```

Il blocco verde è il filtro sulle uscite della funzione `contact_point()` sopra riportata ed è stato utilizzato per avere una migliore visualizzazione in Virtual Reality Sink. Infine i blocchi gialli e rossi si occupano dell'animazione nell'ambiente Virtual Reality rispettivamente della freccia che rappresenta il momento di attrito normale e della freccia che rappresenta la forza applicata nel punto di contatto. L'ambiente Virtual Reality in cui si realizzano le animazioni è stato costruito a partire dal modello CAD della superficie salvato in formato .wrl e permette di visualizzare in maniera dinamica l'evoluzione del punto di contatto, le variazioni di intensità e direzione della forza di contatto e del momento di attrito ad essa associato.
Di seguito si riportano i risultati di alcune simulazioni eseguite su diverse superfici generiche.
Si utilizza la superficie NURBS rappresentata in figura 6.4, che è stata realizzata importando un file CAD ProE in Geomagic e seguendo il procedimento analizzato nel capitolo 5. La superficie considerata è un solido di rivoluzione ottenuto facendo ruotare un profilo intorno all'asse y e assomiglia ad una sfera avente una parte terminale

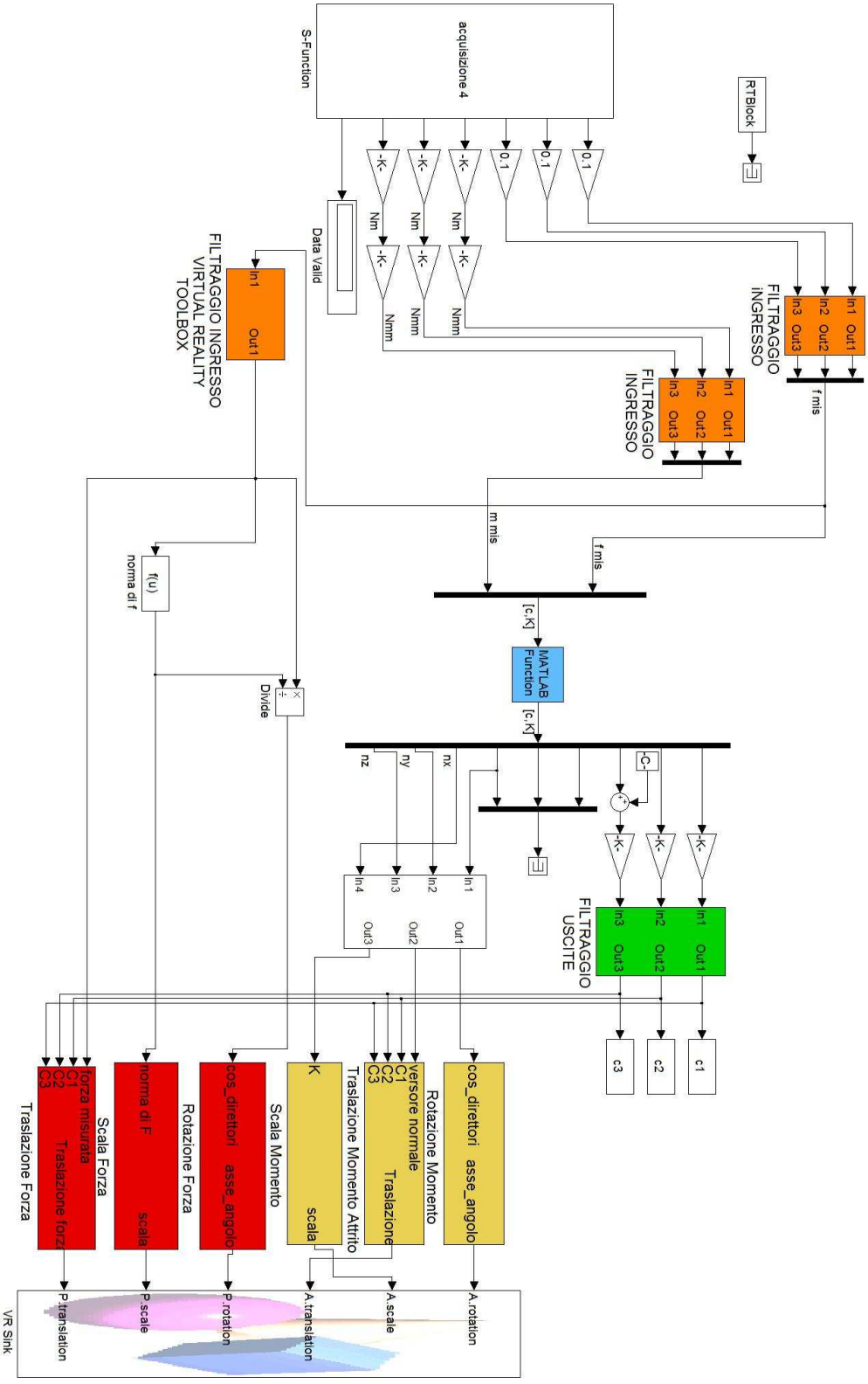

Figura 6.3: Schema Simulink soluzione iterativa per superfici in forma parametrica

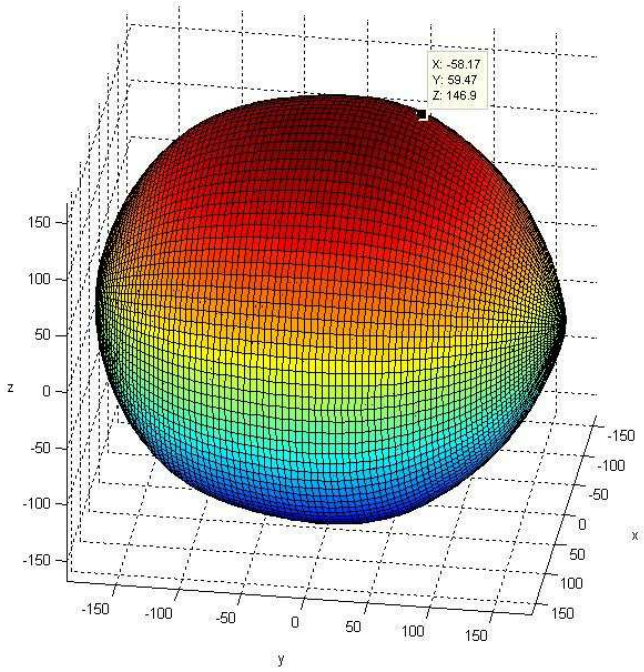

Figura 6.4: Superficie NURBS realizzata in Matlab

acuminata in corrispondenza di uno dei due poli. Considerando il punto della superficie avente le coordinate x, y e z (espresse in millimetri) mostrate in figura 6.4 e, imponendo una forza di contatto $F = [1 - 0.2 - 1]N$, otteniamo la soluzione esatta come possiamo notare nella tabella 6.1. Il *guess* iniziale che è stato considerato è definito dai valori dei parametri $[u, v, K] = [0.3, 0.5, 0]$, dove u e v sono i parametri che individuano un punto sulla superficie definita in forma parametrica e K è il modulo del momento d'attrito. Tali valori iniziali dei parametri u e v corrispondono al punto c_0, espresso in coordinate cartesiane, $c_0 = [x, y, z] = [-146.132, -103.763, 0.0008]mm$. Nella simulazione è stato considerato uno *step size* fisso di 0.01 secondi e, come si può notare dalla tabella 6.1, la convergenza alla soluzione esatta si ha in un *sample time* e diventa perfettamente stabile in tempi di simulazione inferiori ai 0.05 secondi (5 passi). Dopo aver verificato l'efficienza dell'algoritmo fuori linea è stata scelta un'altra superficie che potesse essere fisicamente realizzata in laboratorio e che permettesse di testare l'algoritmo in linea. In seguito ad alcune valutazioni riguardanti le possibili applicazioni ingegneristiche dell'algoritmo sviluppato, si è deciso di considerare quale superficie di prova il polpastrello di un dito, che è stato realizzato in

c_1 [mm]	c_2 [mm]	c_3 [mm]
-58.597920	59.447565	146.787939
-58.597917	59.447585	146.787932
-58.597927	59.447584	146.787928
-58.597927	59.447585	146.787928
-58.597927	59.447585	146.787927

Tabella 6.1: Componenti di c nei sample times iniziali

resina epossidica adoperando un apposito calco. Successivamente è stato costruito il modello CAD del dito ed è stato importato in Matlab utilizzando Geomagic e il Nurbs Toolbox. La visualizzazione dell'oggetto in Simulink è ottenuta tramite Virtual Reality Toolbox; in questo modo è possibile controllare l'efficienza dell'algoritmo in linea anche considerando punti di contatto che varino rapidamente.
Inizialmente è imposto fuori linea il punto di contatto le cui componenti in coordinate cartesiane sono evidenziate in figura 6.5 ($c = [3.756\ -1.454\ 64.6]\, mm$) e la forza applicata $F = [-1\ 0\ -0.1]\, N$; dunque si ottengono risultati soddisfacenti mostrati in figura 6.6 (la soluzione converge al punto di contatto desiderato al primo passo).
Successivamente l'algoritmo è stato validato a livello sperimentale utilizzando quale superficie di prova il polpastrello di un dito in resina epossidica. Uno dei punti cruciali emerso in questa fase è stato il problema del costo computazionale dell'algoritmo di minimizzazione. Infatti la soluzione esposta che adopera il calcolo dell'*initial guess* è stata scelta dopo una serie di sessioni sperimentali in cui sono state utilizzate altre varianti dell'algoritmo. Come emerge dal codice della funzione `contact_point` riportato in questo paragrafo è stata fatta la scelta di calcolare diecimila possibili *guess* iniziali scegliendo quello che risulti essere il più lontano dalla frontiera definita dal vincolo non lineare $f^T \cdot n < 0$. In altri termini si individua il punto iniziale come quello avente versore normale con componente minima nella direzione ortogonale alla forza misurata. Per chiarezza, si riporta la figura 6.7 dove è raffigurata la griglia quadrata di punti considerata sul polpastrello (in blu), la forza misurata (in rosso) e il punto scelto come *guess* iniziale (in verde). Con questa scelta del *guess* iniziale è stato possibile ridurre drasticamente il numero di iterazioni

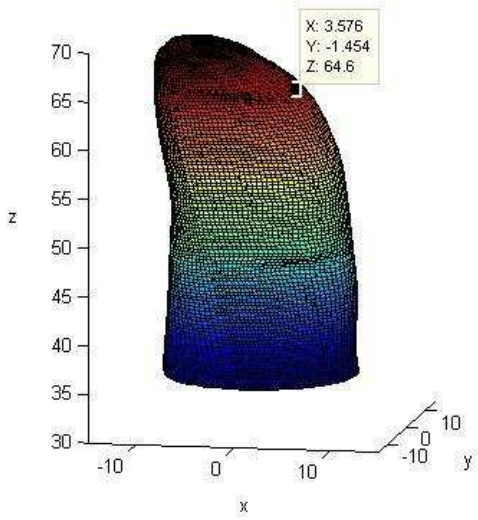

Figura 6.5: Superficie NURBS polpastrello dito in Matlab con punto di contatto di prova

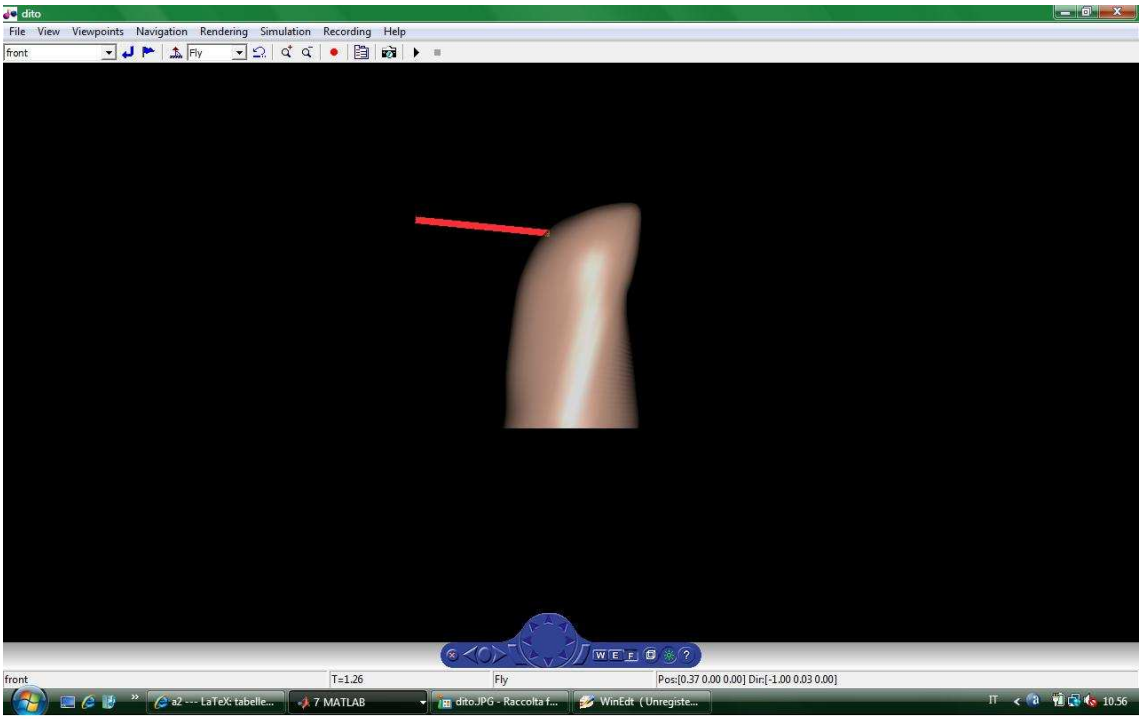

Figura 6.6: Visualizzazione soluzione utilizzando Virtual Reality Toolbox

effettuate ad ogni ciclo dalla funzione `lsqnonlin()` in modo da rispettare il vincolo sul tempo di ciclo. In particolare, limitando il numero di iterazioni a dieci, le prestazioni dell'algoritmo non sono risultate essere degradate in termini di precisione della soluzione e sono stati ottenuti risultati soddisfacenti nelle prove sperimentali effettuate considerando diverse superfici quali un cilindro, un parallelepipedo a base quadrata e un polpastrello.

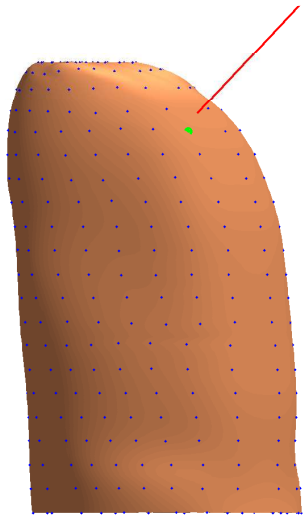

Figura 6.7: Scelta dell'*initial guess* per il polpastrello

6.4 Confronti

In questo Paragrafo si analizzano le prestazioni degli algoritmi esposti evidenziandone pregi e difetti. Le simulazioni effettuate trattano una superficie di contatto cilindrica. E' stato considerato, come soluzione del problema, il punto di contatto $c = [-7.467 \quad -13.58 \quad 50.57]\, mm$ ed è stata imposta una forza di contatto $F = [1 \ 1.1 \ 0.2]\, N$. Si riportano nella tabella 6.2 le soluzioni c trovate da ciascun algoritmo e i relativi tempi di calcolo dell'intera Matlab Function.

Analizzando la Tabella 6.2, si può notare innanzitutto la correttezza degli algoritmi proposti in termini di soluzione trovata. In particolare l'algoritmo che implementa la soluzione del problema in forma chiusa risulta, ovviamente, il più veloce in ter-

Algoritmo	c_1 [mm]	c_2 [mm]	c_3 [mm]	Tempo di calcolo [s]
Forma chiusa	-7.81	-13.96	50.50	0.000487
Metodo del gradiente	-7.75	-13.83	50.51	0.333654
Metodo dei molt. di Lagrange	-7.47	-13.58	50.57	0.283179
Metodo di Levenberg Marquardt (no guess)	-7.47	-13.58	50.57	0.084969
Metodo di Levenberg Marquardt (con guess)	-7.47	-13.58	50.56	0.066318

Tabella 6.2: Confronto tra algoritmi diversi (punto di contatto e tempo di calcolo)

mini di tempi di calcolo; la differenza tra la soluzione trovata da questo algoritmo e la soluzione esatta è da ricercarsi nelle approssimazioni numeriche fatte durante i calcoli implementati e nell'assenza completa di un feedback. Inoltre tale soluzione rimane legata al particolare tipo di superficie, in quanto risulta valida solo per superfici ellissoidali e in particolare per piano, sfera e cilindro.

La seconda soluzione proposta, quella che utilizza il metodo del gradiente, è più precisa in termini di punto di contatto trovato, ma troppo lenta. Questo algoritmo infatti, essendo stato implementato senza l'utilizzo di nessuna funzione dell'Optimization Toolbox di Matlab, non è stato ottimizzato in termini di velocità di convergenza. Inoltre l'algoritmo richiede la definizione dello *step length*, il quale influenza sia la convergenza (valori di *step length* troppo alti fanno divergere la soluzione), che la sua velocità (più elevato è lo *step length* e maggiore è la velocità di convergenza). Ovviamente il valore di tale parametro non può essere universalmente individuato, infatti il suo valore ottimale dipende dalla superficie considerata e dalle altre grandezze in gioco.

Come terza soluzione considerata nella tabella, è stato implementato un algoritmo che utilizza la funzione `fmincon()` dell'Optimization Toolbox di Matlab, la quale cerca il minimo di una funzione vincolata con i moltiplicatori di Lagrange. La scelta di tale funzione è legata all'imposizione del vincolo non lineare utilizzato per risol-

vere il problema della molteplicità della soluzione (4.2). Come si può vedere dai risultati ottenuti, la soluzione trovata gode di un'ottima precisione. Inoltre questo algoritmo riesce anche ad avere un'ottima robustezza rispetto a variazioni dei parametri iniziali; in altre parole, l'algoritmo riesce a trovare la soluzione giusta partendo da qualsiasi punto della superficie, sia esso vicino o lontano dal punto desiderato. Lo svantaggio maggiore di tale funzione, che emerge dai dati disponibili, è l'elevato tempo di convergenza che comporta un costo computazionale troppo elevato per un'applicazione real-time come quella per cui questo tipo di algoritmo si ritiene possa essere utile.

Come è stato discusso nel Paragrafo 4.2, si è optato per l'utilizzo della funzione `lsqnonlin()`, quindi per una ottimizzazione ai minimi quadrati con il metodo di Levenberg-Marquardt. La soluzione trovata anche in questo caso risulta praticamente coincidente con il punto di contatto desiderato; inoltre la velocità di convergenza risulta essere maggiore rispetto a quella rilevata con le soluzioni precedentemente considerate. La funzione utilizzata però non prevede l'imposizione di vincoli di alcun genere, quindi non è possibile discriminare a priori circa la correttezza della soluzione; soltanto al termine dell'ottimizzazione è possibile verificare se il vincolo è soddisfatto oppure no. Quindi è evidente che tale soluzione non è ideale per un'applicazione real-time.

L'ultima soluzione proposta risulta essere una variante di quella precedente. Infatti, sotto l'ipotesi di superficie convessa, si è notato che, considerando come punto iniziale dell'ottimizzazione (con la funzione `lsqnonlin()`) il punto più lontano possibile dalla frontiera corrispondente al vincolo, il risultato trovato rispetta il vincolo. Inoltre i passi necessari a trovare la soluzione sono drasticamente diminuiti grazie alla vicinanza tra il punto di contatto desiderato e il punto considerato come *guess* iniziale. Per questo motivo è stato ridotto il numero massimo di iterazioni disponibili per la funzione `lsqnonlin()` ad un valore compreso tra dieci e venti. In questo modo si ottiene una soluzione ancora accettabile in termini di precisione (l'errore è di circa $2 \cdot 10^{-3}$ mm) e un costo computazionale che risulta essere inferiore a quello associato a tutti gli altri algoritmi iterativi implementati. Infine la correttezza della soluzione non risulta essere più legata alla scelta del punto iniziale dell'ottimizza-

zione in quanto tale punto iniziale cambia ad ogni minimizzazione e viene scelto automaticamente; in questo modo la robustezza dell'algoritmo utilizzato per il calcolo della soluzione rispetto alla variazione dei parametri iniziali è paragonabile a quella che si aveva utilizzando la funzione `fmincon()`.

Capitolo 7

Conclusioni

Con questo lavoro si estende il metodo iterativo proposto in [1] per superfici generiche, prendendo in considerazione superfici definite in forma parametrica. Inoltre si presenta un algoritmo utile per l'importazione di un file creato da un programma CAD in Matlab, in modo che i dati importati possano essere adoperati per costruire la superficie NURBS con Nurbs Toolbox e Virtual Reality Toolbox. Sono state dunque effettuati dei confronti tra i metodi iterativi trattati, evidenziando pregi e difetti di ognuno e individuando l'algoritmo migliore sulla base dell'applicazione sperimentale considerata. Gli sviluppi futuri potrebbero riguardare l'implementazione di una procedura di importazione della superficie in Matlab che faccia uso di strumenti di scansione 3D e di *Reverse Engineering*. In questo modo si avrebbe un metodo più rapido per la costruzione della griglia dei punti di controllo della superficie NURBS trattata e una immediata corrispondenza tra le caratteristiche geometriche della superficie di contatto e la sua rappresentazione mediante strumenti software. Inoltre, per quanto riguarda il campo di applicazione di questo tipo di algoritmo, avendo considerato un modello di contatto di tipo *soft finger*, la manipolazione robotica è una delle principali. Per l'analisi di altre eventuali applicazioni si rimanda a [1].

Bibliografia

[1] A. Bicchi, J. K. Salisbury, D. L. Brock, *Contact Sensing from Force Measurements*, The International Journal of Robotics Research .

[2] ATI Industrial Automation, Pinnacle Park, 1031 Goodworth Drive, Apex, NC 27539, USA, *Force/Torque Sensor System - Installation and Operation Manual.*

[3] L. Piegl, W. Tiller, *The NURBS Book*, 2nd Edition, 1997.