Cinematica del subriflettore di SRT

Franco Buffa

INAF - Osservatorio Astronomico di Cagliari
fbuffa@oa-cagliari.inaf.it
Rev. Settembre 2011

Cinematica di M2

Introduzione

La movimentazione cinematica del subriflettore (M2) del Sardinia Radio Telescope avviene attraverso la composizione dei movimenti di sei attuatori che conferiscono al sistema sei gradi di libertà. gli attuatori verranno guidati in retroazione attraverso il sistema di metrologia laser previsto (cfr. TM-1385-005 Laser Metrology). Nel presente studio si propone un semplice modello dei moti di M2 e quindi un'analisi della catena cinematica nella quale ogni attuatore può essere decomposto. Queste pagine costituiscono anche una guida essenziale all'utilizzo della libreria *hexlib* da utilizzarsi per il controllo della movimentazione del subriflettore. La fig.1 schematizza la geometria del sistema, nella notazione adottata i sei attuatori risultano

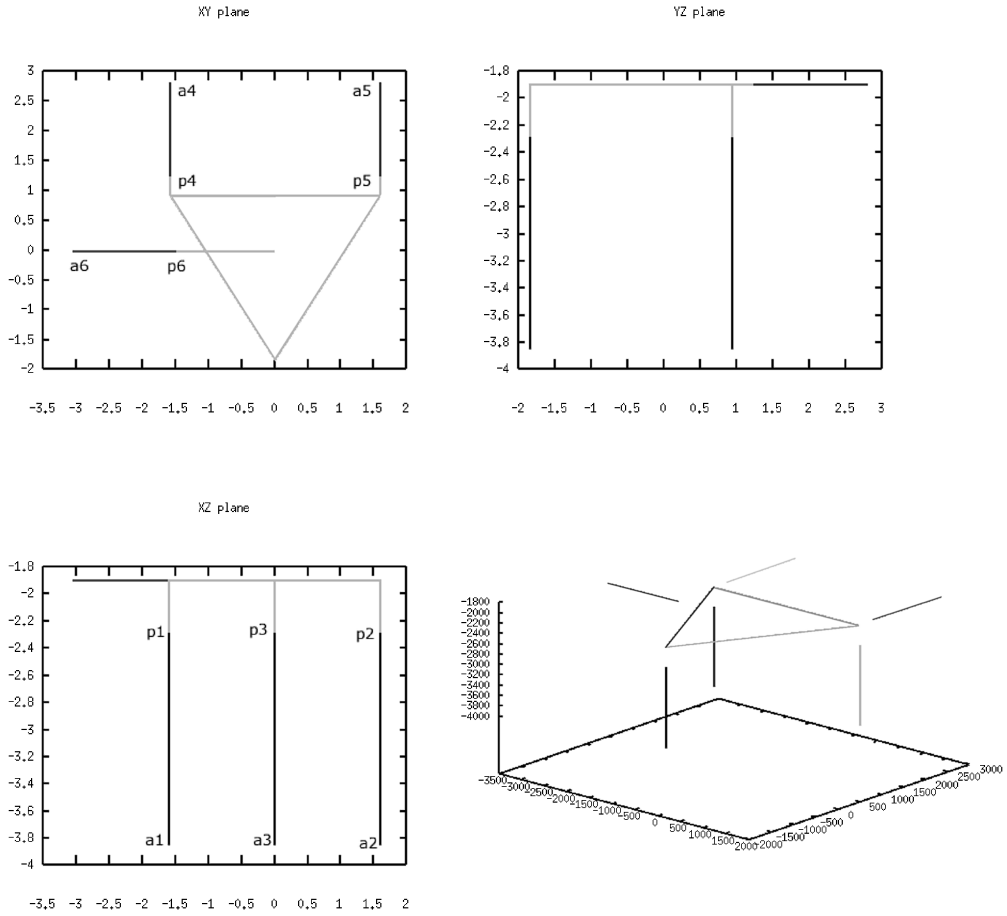

Figura 1: Rappresentazione schematica dei sei attuatori che movimentano il subriflettore di SRT

identificati da coppie di punti riferiti all'origine degli assi posta convenzionalmente nel vertice di M2.

attuatore	tipo	x_p	y_p	z_p	x_a	y_a	z_a
a_1p_1	mov z	-1602.0	924.9	-2287.6	-1602.0	924.9	-3857.6
a_2p_2	mov z	1602.0	924.9	-2287.6	1602.0	924.9	-3857.6
a_3p_3	mov z	0.0	-1849.8	-2287.6	0.0	-1849.8	-3857.6
a_4p_4	mov y	-1602.0	1230.9	-1893.6	-1602.0	2800.9	-1893.6
a_5p_5	mov y	1602.0	1230.9	-1893.6	1602.0	2800.9	-1893.6
a_6p_6	mov x	-1474.0	0.0	-1893.6	-3044.0	0.0	-1893.6

Tabella 1: Coordinate progettuali espresse in mm e riferite al vertice di M2 dei punti di attacco degli attuatori all'APEX (a_i) e al subriflettore (p_i)

Studio cinematico

Lo studio cinematico del subriflettore ha in definitiva due scopi:

1. descrivere la trasformazione che data una roto-traslazione di M2 permette di stimare le corrispondenti elongazioni degli attuatori;
2. determinare la trasformazione inversa che dalle elongazioni permette di ottenere le corrispondenti roto-traslazioni.

Assumendo come sistema di riferimento convenzionale quello di fig. 1, in virtù di una generica movimentazione degli attuatori i punti a_i risultano vincolati (tramite giunti sferici) alla struttura, mentre i punti p_i assumono nuove coordinate p'_i definite dalla trasformazione:

$$p'_i = [x'_i, y'_i, z'_i]^T = R_z R_y R_x [x_i, y_i, z_i]^T + [B_x, B_y, B_z]^T \quad i = 1, 2, ...6 \tag{1}$$

R_j è la matrice di rotazione rispetto all'asse $j (j = x, y, z)$ e B è il vettore di traslazione. Le misure delle elongazioni di ciascun attuatore, noto il vettore di roto-traslazione r, sono date dalle relazioni:

$$d_i = [(a_i p'_i)(a_i p'_i)^T]^{\frac{1}{2}} \quad i = 1, 2, ...6 \tag{2}$$

Il problema inverso: determinazione degli angoli e della traslazione a partire dalle elongazioni degli attuatori, risulta complicato dalla non invertibilità in senso analitico del problema.
Si consideri il set di sei equazioni non lineari nelle incognite rappresentate dalle componenti di r, per una data configurazione degli attuatori d_i^*, il problema puo essere scritto:

$$F_i = d_i^* - [(a_i p'_i)(a_i p'_i)^T]^{\frac{1}{2}} = 0 \quad i = 1, 2, ...6 \tag{3}$$

che in forma compatta diventa:

$$F(r) = 0 \tag{4}$$

$$r = [\theta_x, \theta_y, \theta_z, B_x, B_y, B_z]^T$$

dove F è un'applicazione non lineare in $\Re^6$ continua con la sua derivata $J = F'(r)$.
I metodi che risolvono la (4) sono di tipo ricorsivo: data una approssimazione iniziale r_0, si generano sucessive soluzioni r_k fino al raggiungimento di una buona approssimazione della soluzione cercata. Linearizzando la (4) si ottiene:

$$r_{k+1} = r_k - J^{-1} F(r_k) \tag{5}$$

o in forma equivalente:

$$J dr = -F(r) \tag{6}$$

La (6) rappresenta il metodo di Newton che risulta applicabile se lo Jacobiano J non è singolare. Nel corso di alcuni test si è riscontrato che, in questo caso, il metodo di Newton raggiunge con relativa difficoltà la convergenza e questo effetto è tanto più marcato quanto più r_0 si discosta dalla soluzione ottimale.
Il Powells Hybrid method è un algoritmo quasi-Newton che supera le limitazioni del metodo classico. Ad ogni iterazione l'algoritmo determina una soluzione alla Newton risolvendo il sistema (6), se la nuova soluzione è tale da migliorare la convergenza questa viene accettata, altrimenti l'algoritmo utilizza una combinazione lineare del metodo di Newton con il metodo di ricerca gradientale di un minimo (steepest descending):

$$dr = \alpha J^{-1} F(r) - \beta \nabla \|F(r)\|^2 \tag{7}$$

La (7) è stata implementata in ambiente Octave (http://www.octave.org) e ancora in Octave si sono scritti i moduli di visualizzazione 3D delle parti meccaniche di interesse.
A titolo di verifica si è operata una mappatura della superficie (6D) degli errori di stima dei parametri generando 10^4 configurazioni casuali di parametri di roto-traslazione, calcolando le corrispondenti elongazioni e quindi invertendo il problema per ottenere un nuovo set di parametri da confrontare con quelli di partenza. L'esito dei test è stato del tutto soddisfacente in quanto in nessun caso l'algorimo ha manifestato problemi di convergenza. Ulteriori verifiche sono state effettuate in ambiente CAD al fine di controllare la coerenza dei calcoli rispetto a un sistema simulato.

Catena cinematica

Gli attuatori del subriflettore di SRT (fig. 2) sono dotati alle estremità di giunti sferici. Ogni giunto sferico è dotato di 3 gradi di libertà e pertanto ogni attuatore è in grado di assumere una qualsivoglia posizione nello spazio compatibilmente con il range angolare meccanicamente concesso ai giunti sferici. Tali angoli

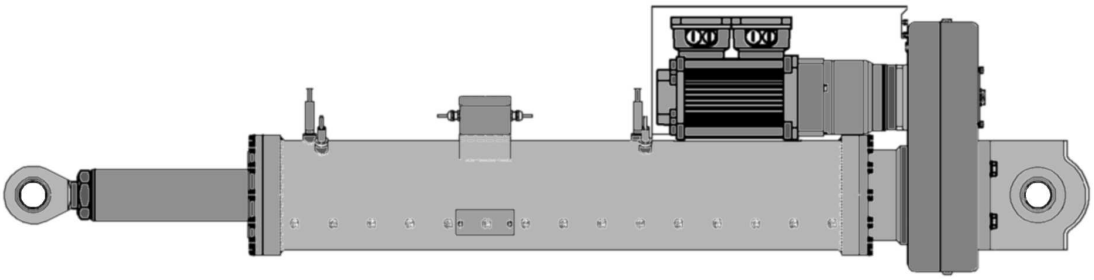

Figura 2: Uno dei sei attuatori del sistema di movimentazione di M2

possono essere studiati in funzione delle elongazioni imposte ai sei attuatori in modo da simulare particolari configurazioni estreme che possano determinare rotture o stress meccanici al sistema.
Per facilitare lo studio e la caratterizzazione di tali moti é possibile scomporre i moti di ciascun attuatore in una *catena cinematica* rappresentando i moti dei giunti sferici per mezzo di una composizione di *moti cardanici* equivalenti. L'obiettivo è quello di riuscire a determinare gli angoli di torsione di ciascun giunto per una data posizione nello spazio del subriflettore definita dal vettore r nella (4) o in maniera equivalente dalle sei elongazioni definite nella (2).
In fig. 3 viene mostrato uno dei sei attuatori del subriflettore di SRT in una rappresentazione che permette di scomporne i moti in moti elementari secondo la convenzione di Denavit-Hartenberg. Si individuano le parti mobili (*giunti*) distinguendo tra quelle a cui corrisponde un moto circolare, *giunto di rivoluzione* (rappresentato da un cilindro) e quelle a cui corrisponde una traslazione, *giunto prismatico* (rappresentato in figura da un cubo). I *giunti* sono tra loro collegati da *link* che sono le parti non mobili della catena cinematica. Nel caso mostrato in fig. 3 sono rappresentati sei giunti (cinque di rivoluzione ed uno prismatico)

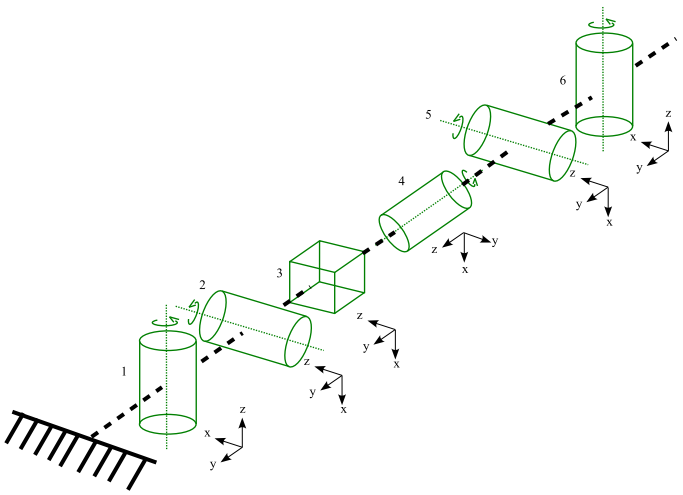

Figura 3: Rappresentazione della scomposizione in moti elementari di un attuatore

corrispondenti ai gradi di mobilità dell'attuatore. Il giunto di rivoluzione #4 in particolare descrive la risultante delle rotazioni dei due giunti sferici attorno all'asse longitudinale dell'attuatore.
Si può notare che ad ogni giunto viene associato un sistema di assi cartesiani (numerati dal #1 corrispondente al primo cardano vincolato all'APEX, fino al #6 corrispondente all'ultimo cardano solidale a M2). Sia allora A_i l'operatore di roto-traslazione che permette il passaggio dal sistema di assi con indice i a quello con indice $i-1$, si avrà che l'operatore T_1, espresso dalla relazione:

$$T_1 = \prod_{i=2}^{6} A_i \tag{8}$$

rappresenta l'operatore di passaggio dal sistema #6 al sistema #1. Si consideri il punto materiale q_6 nel sistema #6 e quindi solidale al subriflettore non essendoci altri moti *a valle* di quello considerato. In virtù di una generica movimentazione del subriflettore, le coordinate del punto rimangono invariate rispetto al sistema #6, ma variano rispetto a tutti gli altri sistemi e in particolare rispetto al sistema #1. Il sistema #1 è solidale (ma non coincidente) con il sistema di riferimento di M2 definito nel precedente paragrafo (che verrà indicato come il sistema #0) quindi per una qualunque movimentazione del subriflettore le nuove coordinate q_1 del punto materiale espresse nel sistema #1 sono ottenibili a partire dalle trasformazioni definite nel paragrafo precedente (a meno di un'ulteriore roto-traslazione che trasferisce il sistema #1 sul sistema #0. Queste considerazioni sono valide per ciascun attuatore anche se ovviamente la roto-traslazione che trasferisce il sistema #1 di ciascun attuatore sul sistema #0 di M2 sarà definita da parametri differenti, ma noti a priori.
Applicando la (8) a q_6 si ottiene dunque $q_1 = T_1 q_6$: qui tanto q_1 quanto q_6 sono noti a priori, mentre incogniti sono i valori dei parametri associati alla trasformazione T_1 (ovvero le posizioni angolari cercate dei giunti di rivoluzione).
La tabella 2 riporta i parametri associati alla catena cinematica tramite la trasformazione T_1. Con riferimento alla fig. 3, se si considera il giunto #1 a questo compete una rotazione libera γ_1 attorno all'asse z del suo sistema di riferimento, al giunto #2 compete una rotazione *libera* dello stesso tipo, ma preceduta da una rotazione *fissata* di $\frac{\pi}{2}$ attorno all'asse y, con considerazioni analoghe si può passare da un sistema all'altro sino al #6 utilizzando i parametri riportati in tabella. Gli unici movimenti liberi corrispondono dunque alle rotazioni attorno all'asse z (γ_i) dei corrispondenti cardani e alla traslazione lungo l'asse y del giunto prismatico #3 (il cui valore è comunque noto perché deducibile dalla trattazione cinematica del precedente capitolo); le rotazioni attorno agli assi x e y (α e β) sono solo di tipo *fissato*. Si noti ancora che i sistemi di riferimento dei primi due cardani e degli ultimi due sono uniti da link di misura nulla, e ciò per tener conto

joint	α	β	γ	d_x	d_y	d_z
revol. #1	0	0	γ_1	0	0	0
revol. #2	0	$\frac{\pi}{2}$	γ_2	0	0	0
prism. #3	0	0	0	0	d_3	0
revol. #4	$-\frac{\pi}{2}$	0	γ_4	0	0	0
revol. #5	$\frac{\pi}{2}$	0	γ_5	0	0	0
revol. #6	0	0	γ_6	0	0	0

Tabella 2: Parametri cinematici, α, β e γ sono le rotazione attorno agli assi x, y e z di ciascun giunto e d_x, d_y e d_z le relative traslazioni

che nell'attuatore *reale* di fig. 2 i moti dei cardani rappresentano in realtà un giunto sferico equivalente.
Per testare la catena cinematica descritta è dunque necessario risolvere l'eq. (8) per i valori consentiti al vettore di roto-traslazione r definito nella (4). In questo studio i range nominali applicati alle componenti di r sono $\theta_x = \pm 0.25$ (deg), $\theta_y = \pm 0.25$ (deg), $\theta_z = \pm 0.25$ (deg), $B_x = \pm 0.05$ (m), $B_y = \pm 0.11$ (m) e $B_z = \pm 0.05$ (m). I risultati di seguito esposti sono riferiti all'attuatore a_5p_5, ma naturalmete l'analisi potrebbe essere facilmente estesa anche agli altri cinque attuatori.
Nelle figure successive sono presentate tre serie ciascuna delle quali è composta da sei grafici corrispondenti ai moti dei sei giunti dell'attuatore a_5p_5. I moti sono espressi in funzione degli angoli di rotazione del sistema θ_x, θ_y e θ_z (indicati nei grafici come ROT(x), ROT(y) e ROT(z)). Nella simulazione le tre componenti di traslazione del vettore r erano poste uguali a zero.
I grafici mostrano i range di lavoro dei singoli cardani, ma anche le correlazioni esistenti tra il moto di ciascun giunto e le componenti angolari del vettore r. Se si considera ad esempio il giunto di rivoluzione #1 si può notare che questo risulta non correlato a θ_x e correlato a θ_y e θ_z, nel caso del giunto #4 questo risulta strettamente correlato soltanto a θ_y.

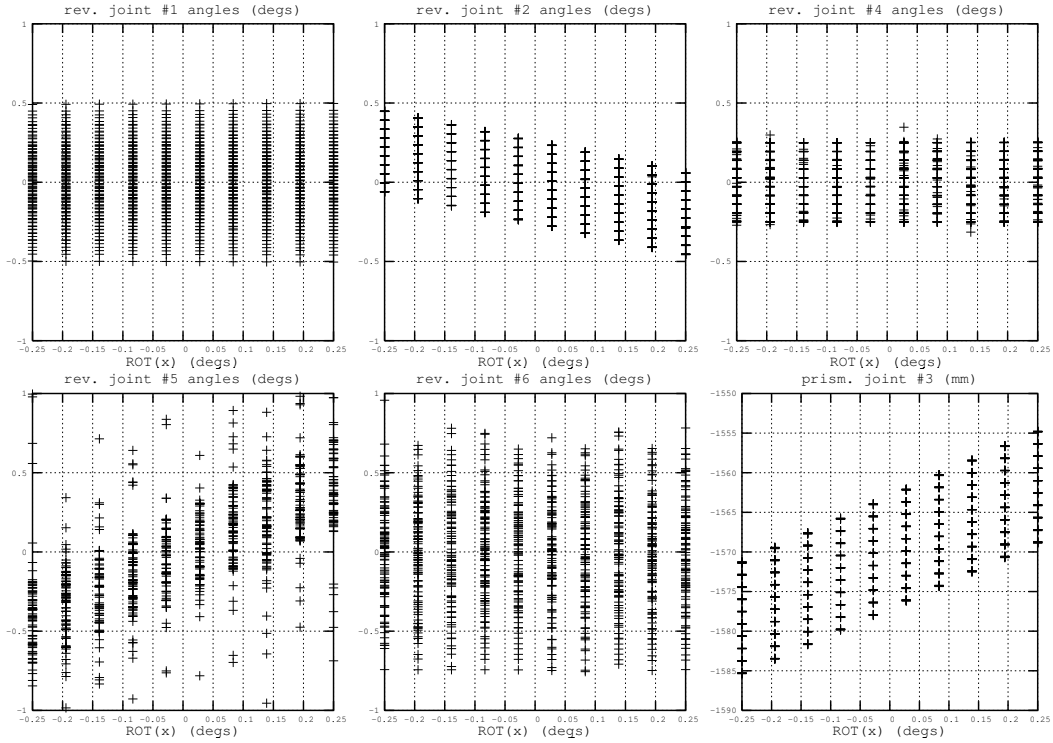

Figura 4: Valori dei giunti prismatici e di rivoluzione in funzione di θ_x (ROT(x) nel grafico).

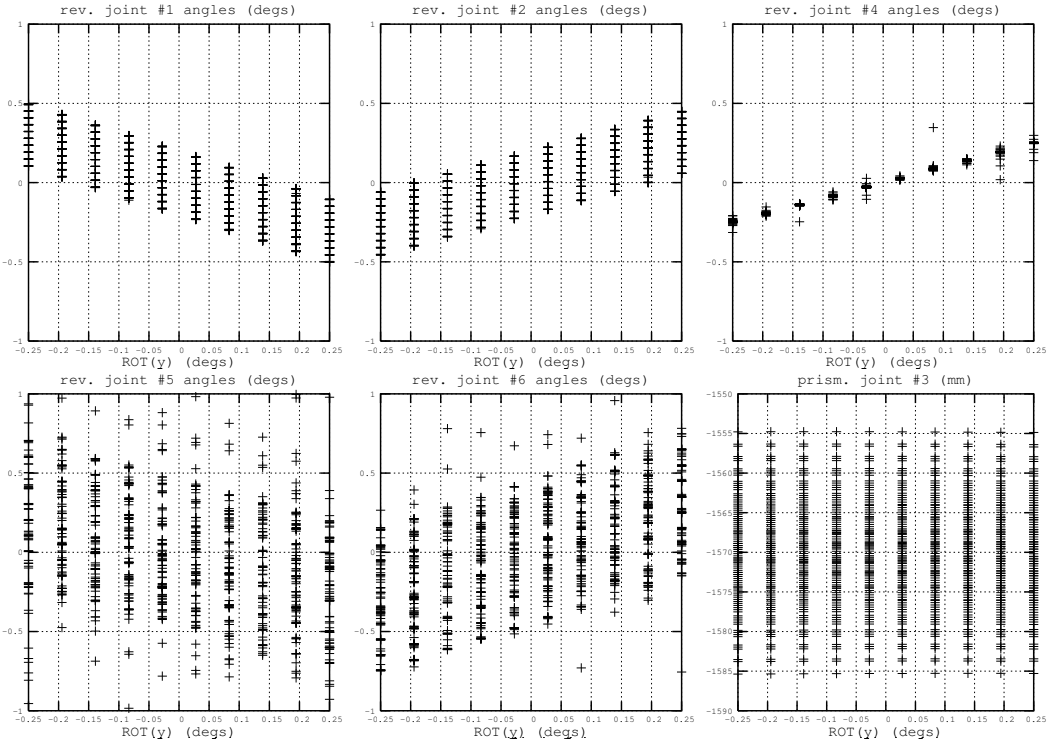

Figura 5: Valori dei giunti prismatici e di rivoluzione in funzione di θ_y (ROT(y) nel grafico).

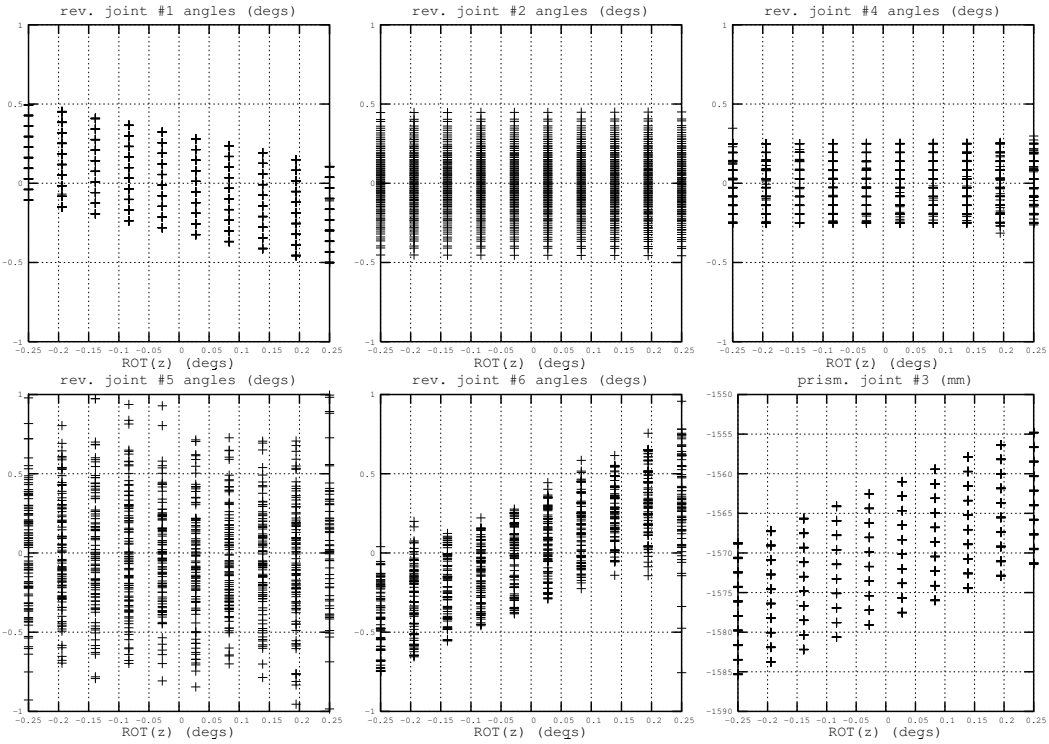

Figura 6: Valori dei giunti prismatici e di rivoluzione in funzione di θ_z (ROT(z) nel grafico).

Libreria hexlib

Funzioni di libreria

La libreria hexlib consente di trattare il problema cinematico ed è quindi utilizzabile per il controllo di M2. Le due funzioni di libreria *dir* e *inv* permettono di risolvere rispettivamente il problema diretto ed inverso, le altre funzioni svolgono il compito di inizializzazione dei parametri e di visualizzazione dei risultati del calcolo. La libreria non è, per il momento, utilizzabile per la decomposizione dei moti cardanici.
L'unico input richiesto dalla libreria è costituito da un file contenente le coordinate dei punti d'attacco degli attuatori, cioè le coordinate delle 6 coppie di punti a_i e p_i che in pratica definiscono la posizione a riposo dei sei attuatori nello spazio. Queste possono essere inserite nel file in qualsivoglia ordine, questo stesso ordine, ovviamente, si ritroverá negli output.
Per il porting in C della libreria si è utilizzata la GNU Scientific Library (GSL), questa libreria dovrà ovviamente essere pre-installata sulla macchina nella quale si intende utilizzare la hexlib. La GSL è disponibile all'indirizzo: *http://www.gnu.org/software/gsl/*, sia per Linux sia per Winxx.

struct rparams

```
typedef struct
{
   double p1[3],p2[3],p3[3],p4[3],p5[3],p6[3]; // coordinate lato M2
   double a1[3],a2[3],a3[3],a4[3],a5[3],a6[3]; // coordinate lato APEX
   int status;  // calcolo OK?
   size_t iter; // numero di iterazioni
/*
   x[_n] componenti del vettore r = [tx ty tx rx ry rz]
   y[_n] residui calcolati da fsolver
   d[_n] elongazioni degli attuatori
*/
   double x[_n],y[_n],d[_n];
}rparams;
```

La struttura rparams contiene i parametri necessari per il calcolo quali le coordinate dei punti che definiscono la geometria del sistema. I dati sono caricati dalle funzioni di libreia *init_p* e *load_p*.

init_p

```
#include "hexlib.h"
int init_p(p)
rparams *p;
```

Inizializza la struttura dei parametri con i valori di default ovvero le coordinate di progetto dei punti che definiscono la culla di M2 più altri parametri utilizzati nel calcolo. L'inizializzazione deve essere necessariamente eseguita all'inzio del calcolo. Nel caso non si intendano utilizzare le coordinate progettuali si dovrà utilizzare in alternativa *load_p*.

load_p

```
#include "hexlib.h"
int load_p(p, fname)
rparams *p;
char *fname;
```

Inizializza la struttura dei parametri con i valori contenuti nel file fname. L'inizializzazione deve necessariamente avvenire all'inizio di un calcolo. Il file fname è un file di testo contenente le coordinate dei punti p_i e a_i espresse in mm. Le prime 6 righe contengono le coordinate dei punti p_1, p_2,..., p_6, le successive 6 quelle dei punti a_1, a_2,..., a_6. Le coordinate sono separate da tabulazioni. L'utente deve predisporre questo file utilizzando coordinate riferite convenzionalmente al vertice di M2. Questa funzione non è equivalente a init_p. La funzione *init_p*, infatti, inizializza il calcolo con le coordinate di progetto, mentre se le coordinate dei punti p_i e a_i sono state misurate con tecniche alternative (per es. con una stazione totale), queste andranno necessariamente salvate su un file e caricate con la funzione *load_p*. La funzione restituisce il valore 0 se il file è stato aperto con successo, 1 altrimenti.

print_p

```
#include "hexlib.h"
int print_p(p)
rparams *p;
```

Stampa a schermo le coordinate dei punti a_i e p_i. I punti sono nello stesso ordine utilizzato nella funzione *load_p*.
Esempio:

```
#include "hexlib.h"
int main ()
{
  rparams p;
  init_p(&p);
  print_p(&p);
  return 0;
}
```

dir

```
#include "hexlib.h"
void dir(p)
rparams *p;
```

Calcola le elongazioni degli attuatori a partire da una roto-traslazione data (problema diretto). I valori corrispondenti alla roto-traslazione (espressi in radianti e in millimetri) vanno preliminarmente caricati in *p.x*.
Esempio:
vedi il programma test01.c.

inv

```
#include "hexlib.h"
void inv(p)
rparams *p;
```

Calcola i parametri di roto-traslazione (vettore r) a partire dalle elongazioni espresse in millimetri (problema inverso). I valori delle elongazioni vanno caricati preliminarmente in *p.d*.
Esempio:
vedi i programmi test02.c e test03.c.

print_state

```
#include "hexlib.h"
int print_state (p)
rparams *p;
```

Restituisce alcune informazioni relative alla soluzione del problema non lineare, quali in numero di iterazioni, gli scarti ecc.
Esempio:
vedi il programma test03.c.

gotopos

```
#include "hexlib.h"
int gotopos(p,pos1,pos2,nstp,x)
rparams *p;
double *pos1;
double *pos2;
int nstp;
h_array *x;
```

Calcola le elongazioni intermendie durante il moto tra due differenti posizioni del subriflettore.
Esempio:
vedi il programma test04.c.

set_rot

```
#include "hexlib.h"
int set_rot(rot_s)
int rot_s;
```

Seleziona la matrice di rotazione da utilizare nei calcoli. I due valori consentiti sono definiti dalle macro RZRYRX e RXRYRZ. La funzione va invocata prima del calcolo altrimenti viene utilizzato il valore di default RZRYRX.
Esempio:
vedi il programma test01.c.

Esempi di utilizzo

test01.c

Il codice test01.c è un esempio di trasformazione diretta da una roto-traslazione alla lunghezza delle elongazioni degli attuatori.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "hexlib.h"

int main (int argc, char *argv[])
{
  int i;
  const size_t n = _n;  // la macro _n contiene il numero di attuatori (_n=6)
  rparams p;

//init_p(&p);
```

```
  if(load_p(&p,"hexdata.txt"))
  {
    printf("non trovo il file attuatori\n");
    exit(0);
  }

//rotazione di default (la chiamata puo' essere omessa)
//commentare se si vuole usare RXRYRZ
  set_rot(RZRYRX);

//rotazione alternativa a quella di default
//scommentare se si vuole usare RXRYRZ
//set_rot(RXRYRZ);

  if(argc<7)
  {
     printf("\n\nCalcolo delle elongazioni assolute (in mm) degli\n");
     printf("attuatori della movimentazione cinematica di M2.\n");
     printf("uso:\n%s tx ty tz rx ry rz -v\n\n",argv[0]);
     printf("l'opzione -v fa il dump della struttura dei parametri\n");
     printf("tx=traslazione x\n");
     printf("ty=traslazione y\n");
     printf("tz=traslazione z\n");
     printf("x=rotazione x\n");
     printf("y=rotazione y\n");
     printf("z=rotazione z\n");
     exit(0);
  }
  for(i=0;i<n;i++) p.x[i]=atof(argv[i+1]); // carica i parametri di roto-traslazione
  dir(&p);
  for (i=0;i<n;i++) printf("d[%d]=%f\n",i,p.d[i]); // stampa le elongazioni
  if(argc>7) if (!strcmp(argv[7],"-v")) print_p(&p);

  return 0;
}
```

hexdata.txt

Il programma test01.c utilizza il file hexdata.txt per caricare le coordinate dei punti di attacco degli attuatori. Come detto l'ordine di inserimento degli attuatori non ha effetto sul calcolo ma solo sull'ordine con cui i risultati saranno visualizzati. Nel seguente esempio gli attuatori sono inseriti nell'ordine: Z_1, Z_2, Z_3, X_1, Y_1, Y_2.

```
-1597.80   917.85 -2276.10
 1603.70   919.00 -2271.85
   -0.20 -1853.30 -2262.85
-1474.75    -8.90 -1892.25
-1598.25  1224.70 -1893.35
 1605.25  1225.65 -1890.30
-1595.10   908.40 -3845.80
 1612.70   917.10 -3841.10
   16.85 -1843.65 -3834.40
-3044.75    -8.90 -1892.25
-1595.75  2795.95 -1885.55
 1596.45  2797.20 -1891.25
```

test02.c

Il programma test02.c è un esempio di trasformazione inversa dalla lunghezza delle elongazioni degli attuatori alla corrispondente roto-traslazione.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "hexlib.h"

int main (int argc, char *argv[])
{
  int j;
  const size_t n = _n;
  rparams p;

  init_p(&p);  // uso le coordinate da pregetto

  if(argc<7)
  {
     printf("\n\nCalcola le traslazioni (mm) e gli angoli di rotazione (deg) date\n");
     printf("le elongazioni assolute degli attuatori della movimentazione\n");
     printf("cinematica di M2. \n");
     printf("uso:\n%s d1 d2 d3 d4 d5 d6\n\n",argv[0]);
     printf("d1=attuatore #1\n");
     printf("d2=attuatore #2\n");
     printf("d3=attuatore #3\n");
     printf("d4=attuatore #4\n");
     printf("d5=attuatore #5\n");
     printf("d6=attuatore #6\n");
     exit(0);
  }
  for(j=0;j<n;j++) p.d[j]=atof(argv[j+1]);
  inv(&p);
  print_state (&p);

  return 0;
```

test03.c

Il codice test03.c verifica l'accuratezza dell'algoritmo calcolando le elongazioni a partire da un input dato da linea di programma e quindi invertendo il problema per ottenere un nuovo set di parametri da confrontare con quelli di partenza. I parametri vengono forniti dall'utente attraverso il file hexdata.txt.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "hexlib.h"

int main (int argc, char *argv[])
{
  int i;
  const size_t n = _n;
  double x0[n];
  rparams p;
```

```
  if(load_p(&p,"hexdata.txt"))
  {
    printf("non trovo il file attuatori\n");
    exit(0);
  }

  if(argc<7)
  {
     printf("\n\nCalcolo delle elongazioni assolute (in mm) degli\n");
     printf("attuatori della movimentazione cinematica di M2.\n");
     printf("uso:\n%s tx ty tz rx ry rz -v\n\n",argv[0]);
     printf("l'opzione -v fa il dump della struttura dei parametri\n");
     printf("tx=traslazione x\n");
     printf("ty=traslazione y\n");
     printf("tz=traslazione z\n");
     printf("x=rotazione x\n");
     printf("y=rotazione y\n");
     printf("z=rotazione z\n");
     exit(0);
  }
  for(i=0;i<n;i++) p.x[i]=atof(argv[i+1]);
  dir(&p);
  printf("Calcolo diretto, rototrasl.->elongazioni\n");
  for (i=0;i<n;i++) printf("d[%d]=%f\n",i,p.d[i]);

  printf("Calcolo inverso, elongazioni->rototrasl.\n");
  inv(&p);
  print_state (&p);

  if(argc>7) if (!strcmp(argv[7],"-v")) print_p(&p);

  return 0;
}
```

test04.c

Il codice test04.c calcola la traiettoria tra le due posizioni pos1 e pos2.

```
#include <stdlib.h>
#include "hexlib.h"
int main()
{
  int i,j,status,m=25;  // m = numero di pos. intermedie
  double y[12];
  h_array *x=new_array(m);  // inizializzo l'array che conterra' i risultati
  double pos1[6]={-1.93,-40.45,-3.1,-631,120,314}; // tx ty tz rx ry rz iniziali
  double pos2[6]={-1.5,16.62,5.86,125,132,244};    // tx ty tz rx ry rz finali
  rparams p;
  if(load_p(&p,"hexdata.txt"))
  {
    printf("non trovo il file attuatori\n");
    exit(0);
  }
  for(i=3;i<6;i++) pos1[i]=pos1[i]*PI/180.0/3600;  // le pos. angolari sono espresse in arc sec
  for(i=3;i<6;i++) pos2[i]=pos2[i]*PI/180.0/3600;
```

```
  gotopos(&p,pos1, pos2,m,x);
  for(i=0;i<m;i++)
  {
    status=get_array(i,x,y);  //  y = [tx ty tz rx ry rz d1 d2 d3 d4 d5 d6] per ogni i
    for(j=3;j<6;j++) y[j]=y[j]*180.0/PI*3600;
    for(j=0;j<12;j++) printf("%f\t",y[j]);
//  for(j=6;j<12;j++) printf("%f\t",y[j]);
    printf("%d\n",status);
  }
  free_array(x);
  return 0;
}
```