UNIVERSITÀ DEGLI STUDI DI PADOVA

FACOLTÀ DI INGEGNERIA
CORSO DI LAUREA IN INGEGNERIA MECCATRONICA

TESI DI LAUREA TRIENNALE

SVILUPPO DI UN SISTEMA APTICO PER APPLICAZIONI ECOGRAFICHE

Relatore: Prof. Ing. ROBERTO OBOE

Correlatore: Ing. OMAR ANDRES DAUD

Laureando: NICOLA FERRO

Matricola 563367-IMC

ANNO ACCADEMICO 2010-2011

Sommario

Scopo di questa tesi è la realizzazione di un'interfaccia per un sistema aptico, con finalità di movimentazione di una sonda ecografica, mantenendola ad una distanza costante da una superficie di ispezione. Il sistema è composto da:

- un robot antropomorfo a 6 gradi di libertà (**CRS A465**),
- un dispositivo aptico (**PHANTOM Omni Device**) il cui movimento dell'end-effector, impugnato dall'operatore, fà corrispondere un movimento cartesiano equivalente, opportunamente scalato, all'end-effector del robot su cui potrà essere installata una sonda ecografica.

Sull'end-effector del robot è stato installato un sensore ottico di prossimità, che segnala l'avvicinarsi della sonda alla superficie di ispezione producendo un feedback proporzionale-derivativo al dispositivo aptico, opportunamente scomposto, genera le coppie necessarie per contrastare l'avanzare dell'end-effector impugnato dell'operatore, creando una sensazione di vincolo reale. In altri termini si è realizzato un sistema master-slave con feedback in retroazione al master, che condiziona il movimento dell'end-effector impugnato dall'operatore e di conseguenza l'avanzare della sonda se il sensore avverte un ostacolo, mantenendola quindi ad una distanza prefissata dalla superficie di ispezione/ostacolo. La difficoltà di questo elaborato consiste nel riuscire a far comunicare i dispositivi provenienti da architetture e ambienti completamenti differenti, grazie ad un pc con installata una scheda di acquisizione e controllo MultiQ-3 e all'ambiente di sviluppo MATLAB in cui si è creato un modello Simulink: grazie a cui si è potuto gestire tutti i segnali di input/output dei vari dispositivi utilizzati e realizzare quindi l'interfaccia.
Per poter movimentare il robor CRS A465 in oper-architecture è stato necessario realizzare un controllo di posizione su tutti i 6 giunti che lo compongono, per poi implementare una funzione di cinematica inversa che ha permesso di passare da un controllo di riferimento sui singoli giunti ad un controllo di riferimento cartesiano dell'end-effector.
Il PHANTOM per essere riconosciuto e per comunicare con il pc, ha richiesto l'installazione del software Handshake proSENSE^{TM} che, grazie alle sue librerie WinCon, contenenti specifici blocchi per Simulink, si è potuto inviare e ricevere dati dal dispositivo. Ultimo passo è stato aggiungere un sensore ottico di prossimità sull'end-effector del robot antropomorfo che, tramite la scheda di aquisizione e adeguato condizionamento del feedback, ha permesso di chiudere la catena realizzando l'interfaccia aptica; la sensazione di vincolo viene quindi percepita dall'operatore solo quando il sensore è ad una determinata vicinanza da un ostacolo.

Indice

Elenco delle tabelle

Elenco delle figure

Introduzione

Le patologie infiammatorie reumatiche, come l'**Artrite Reumatoide** (AR), sono tra le principali cause di disabilità e costituiscono una frequente causa di inabilità al lavoro. Recentemente è stato dimostrato che tramite operazioni di scanning si possono individuare fattori prognostici ritenuti il segno precettore dell'AR. La **risonanza magnetica** (*MRI*), non è adatta a questo tipo di scansione in quanto l'enorme mole del macchinario richiede un tempo di esecuzione molto lungo, con lunghe liste di attesa, unito a costi non indifferenti; presenta inoltre controindicazioni nel caso di impianti metallici ed elettronici, nel caso di pazienti disabilitati, e nei casi in cui si debbano controllare più articolazioni in un follow-up. L'**ecografica a contrasto** (*EGC*), è comunemente utilizzata per la diagnosi di vascolarizzazione legata a tumori, ed è stata considerato come miglior candidata per le operazioni di scanning di Artrite Reumatoide in quanto: il tracciante utilizzato ha un rapidissimo assorbimento caratterizzato da un'ottimo contrasto delle zone d'interesse, la sonda è particolarmente piccola da poter essere facilmente impugnata dall'operatore, riduce drasticamente i tempi e i costi di scansione. Per contro l'ECG richiede che la scansione avvenga in immersione e che la sonda resti ad una determinata distanza dalla superfice d'ispezione, per cui la qualità delle immagini e quindi la possibilità dell'operatore di determinare anomalie, dipende dalle movimentazioni che devono assolutamente essere prive di vibrazioni e mantenute a distanza costante; inoltre la scansione in acqua provoca ulteriore difficoltà all'operatore, in quanto la presenza di rifrazione tra aria e acqua sfalsa la posizione percepita da quella reale.

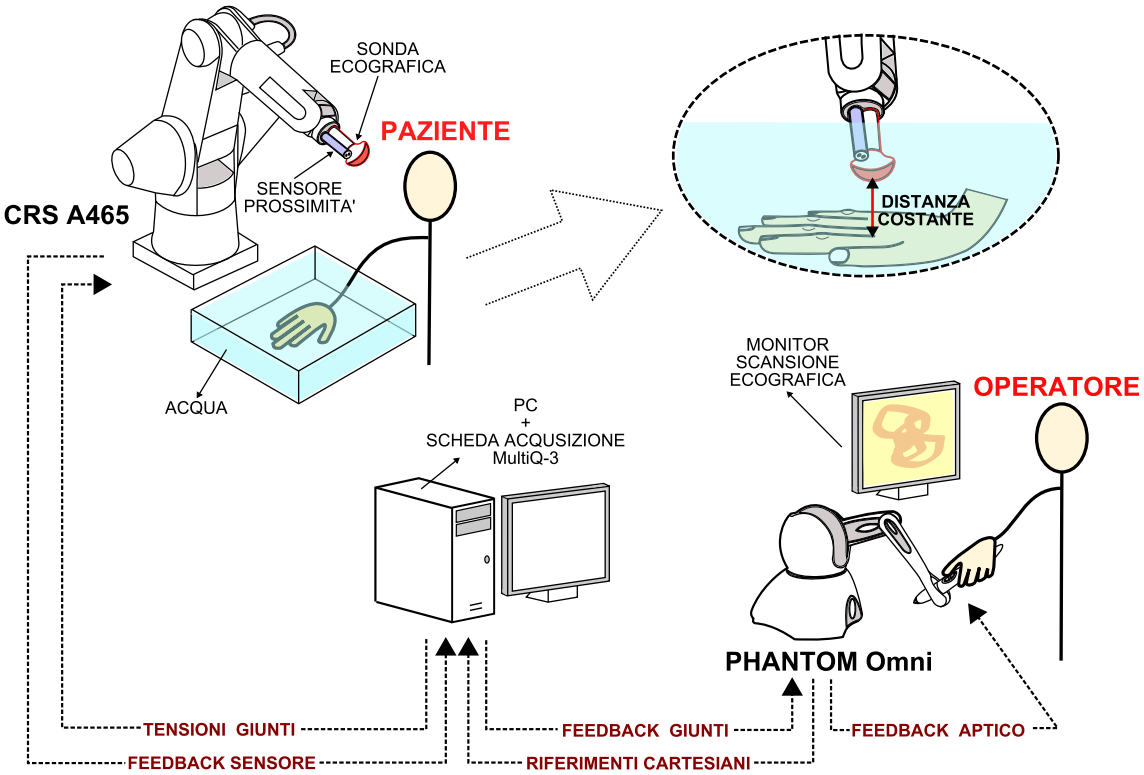

Fig. 1: Introduzione di un sistema aptico nelle analisi ecografiche

Lo sviluppo di questo elaborato si basa sull'idea che l'ecografia a immersione possa ovviare a molti dei problemi presenti attualmente nella analisi delle articolazioni. Per rendere tale tecnica utilizzabile nella routine clinica è però necessario sviluppare degli strumenti

che consentano un'acquisizione rapida, ripetibile e standardizzabile, ed un'interpretazione accurata dei risultati dell'esame.
Al fine di garantire un posizionamento arbitrario della sonda ecografia, al di sopra della zone di indagine, si rende necessaria l'implementazione di un dispositivo robotizzato per sostenere e movimentare con precisione la sonda ecografica; tali movimenti saranno comandati a distanza da un dispositivo aptico, impugnato dalla mano dell'operatore.
La peculiarità del sistema risiede anche nel fatto che la forza riprodotta dal sistema aptico non sarà la forza di contatto, come in un normale sistema di telemanipolazione con ritorno di forza, bensì quella che segnalerà all'operatore la presenza di un vincolo virtuale, cioè la distanza ottimale della sonda. Quest'ultima quantità è misurata da un sensore ottico di prossimità installato a fianco della sonda ecografica e rivolto verso l'area d'indagine.
Si richiede di fatto la realizzazione di un sistema master-slave aptico, in cui il robot CRS A465 su cui verrà installata la sonda ecografica e sensore ottico di prossimità, essendo il dispositivo slave, deve seguire i riferimenti di posizione ed orientazione provenienti dal dispositivo master: PHANTOM Omni. Il sistema aptico consentirà da una parte di muovere la sonda ecografia lungo le varie zone di indagine e dall'altra di guidare il movimento dell'operatore, al fine di mantenere la posizione della sonda ad una distanza ottimale, potendo avvicinarsi ulteriormente in uno specifico punto d'interesse semplicemente applicando maggiore forza contro le coppie di feedback che impediscono l'avanzare.
Scopo di questa tesi è la realizzazione di un interfaccia che permette la comunicazione tra i dispositivi master e slave caratterizzati da architetture completamente diverse, attuando le necessarie codifiche di conversione per unificare i sistemi cartesiani di riferimento e permettere la corretta movimentazione nello spazio. Tale interfaccia è stata realizza in ambiente MATLAB/Simulink e le operazioni di I/O tra i dispositivi e sensore sono stati possibile grazie all'utilizzo di schede di acquisizione della Quanser rispettivamente tra pc e CRS A465 e tra pc e sensore.
I riferimenti che il dispositivo PHANTOM Omni pone in uscita, prima d'essere inviati al dispositivo slave, devono essere convertiti in coordinate di posizione catesiane x y z ed orientazioni dell'end-effector espresse in *yaw pitch roll*, secondo il sistema di riferimento del robot.
Il CRS A465 provenendo dal mondo industriale, ha richiesto particolare attenzione per il suo utilizzo in open-architecture, in quanto passando in tale modalità, il suo controllore disabilita tutti i controlli sui giunti, lasciando all'utente il compito di gestire i riferimenti di tensione che attuano le coppie sui 6 motori. Nelle sue librerie allegate, mette a disposizione dei controllori PD di posizione angolare e cartesiana, ma utilizzabili solo con sistema operativo Windows NT; al contratrio il dispositivo PHANTOM Omni non è utilizzabile in tale sistema operativo, quindi decidendo di lavorare con sistema Windows XP, è stato necessario realizzare anche i controllori PD di posizione per gestire i motori dei singoli giunti dell'A465.

L'elaborato si compone di 3 capitoli:

- il primo per la descrizione dei dispositivi hardware utillizzati,
- nel secondo viene descritta l'intera interfaccia in tutte le sue funzionalità, realizzata in ambiente MATLAB/Simulink,
- nell'ultimo capitolo si trovano i grafici con i confronti tra riferimenti generati dal dispositivo PHANTOM e movimento effettivo del robot A465, con i relativi errori, e degli esempi di segnali di feedback scomposti sugli assi cartesiani per pilotare le coppie dei primi tre giunti.

Capitolo 1

Hardware utilizzati

1.1 PHANTOM Omni

Il PHANTOM Omni è un dispositivo aptico prodotto da SensAble Technologies; permette di toccare e manipolare oggetti virtuali con molta facilità grazie alle dimensioni compatte e ai suoi 6 gradi di libertà. Le coppie prodotte dal dispositivo, generate dai feedback, risultano sufficientemente potenti per una chiara percezione di un vincolo virtuale. È costituito da componenti di metallo e plastica che ne giustificano il costo contenuto ed accessibile. Il dispositivo risulta idoneo alle funzionalità richieste da questo elaborato.

Fig. 1.1: PHANTOM Omni Device

1.1.1 Caratteristiche Tecniche

- certificazione CE
- sei gradi di libertà di rilevamento posizionale
- design compatto e portatile
- area di lavoro compatta per una migliore facilità d'uso
- comodo stilo in gomma strutturato per uso a lungo termine e presa sicura
- due interruttori integrati nello stilo per una miglior facilità d'uso e personalizzazione degli utenti finali
- Stylus-docking per la calibrazione automatica dell'area di lavoro

Tab. 1.1: PHANTOM Omni Specifiche Tecniche

Area di lavoro della forza di retroazione	160 W x 120 H x 70 D [mm]
Footprint(area fisica che la base del dispositivo occupa sulla scrivania)	168 W x 203 D [mm]
Peso	1,786 Kg
Range di movimento	Il movimento della mano con perno al polso
Risoluzione nominale di posizione	0.055 mm
Attrito giunti	$\leq$ 0,26 N
Forza massima esercitabile alla posizione nominale (bracci ortogonali)	3.3 N
Forza continua esercitabile (24ore)	0,88 N
Rigidezza	Asse X 1,26 N/mm Asse Y 2,31 N/mm Asse Z 1.02 N/mm
Inerzia (massa apparente alla punta)	45 g
Forza di retroazione (feedback)	x, y, z
Rilevamento di posizione [Stylus cardanico]	x, y, z (encoder digitale) Pitch, roll, yaw (potenziometri linearità $\pm$5%)
Interfaccia	IEEE-1394 FireWire ® porta: da 6-pin a 6-pin
Le piattaforme supportate	PC Intel o AMD

1.2 Thermo CRS A465

Fig. 1.2: Thermo CRS A465

Il CRS A465 è un robot antropomorfo. Dispone di 6 giunti alimentati da 6 motori.

É stato progettato con lo stesso range di movimento e di carico utile del braccio umano, che lo rende ideale per applicazioni che richiedono un range di movimento articolato in entrambi i piani orizzontale e verticale. Il robot offre una elevata combinazione di automazione flessibile ad alta velocità, alta affidabilità e facilità d'uso. Possiede servomotori duraturi e potenti che lo rendono sia veloce che robusto.

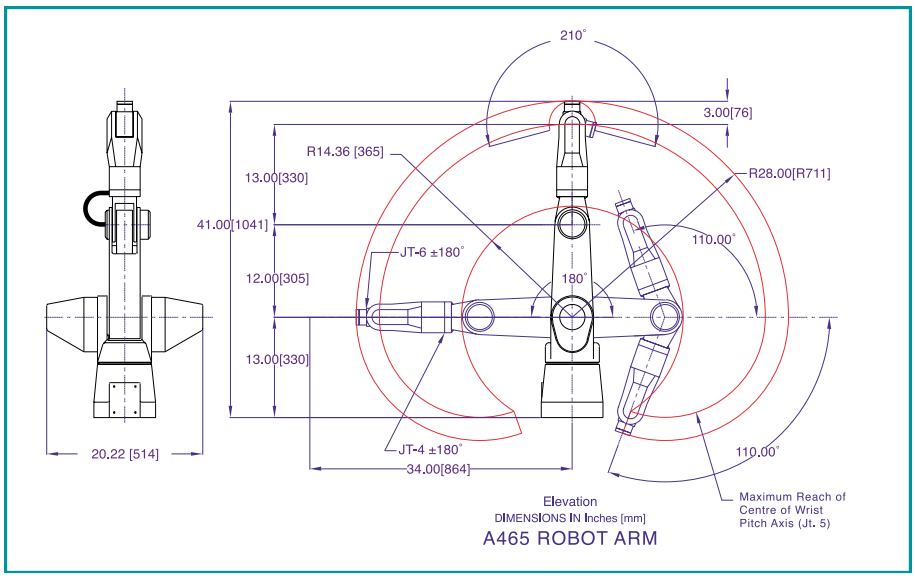

Fig. 1.3: CRS A465: vista laterale, range giunti J2 J3 J5

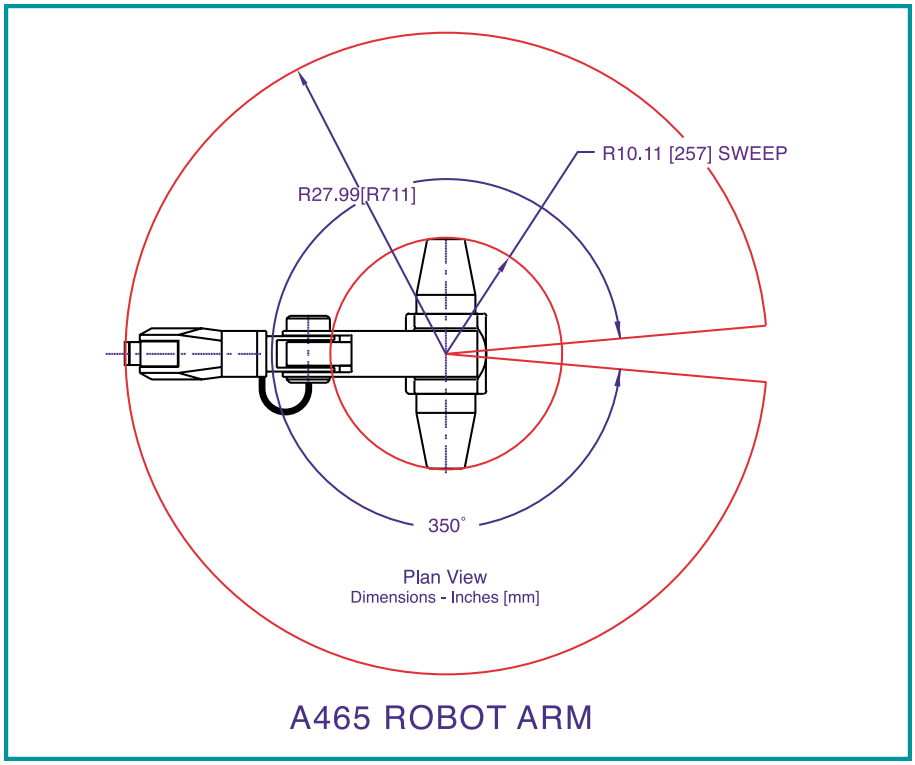

Fig. 1.4: CRS A465: vista dall'alto, range giunto J1

Tab. 1.2: CRS A465: Caratteristiche

Giunti del robot	Articolati
Configurazione	6 gradi di libertà
Drive	Servomotori Encoder di prossimità per ogni giunto
Trasmissione	Harmonic drive Cinghia di distribuzione
End-effector	Connettore pneumatico Servo gripper
C500C	Circuito integrato E-stop
Controller safety	Rilevazione continua dei guasti

Tab. 1.3: CRS A465 - Specifiche tecniche

Carico utile nominale	2 kg
Estensione (no gripper)	711 mm
Estensione (con gripper)	864 mm
Ripetibilità	±0.05 mm
Peso	31 kg

Tab. 1.4: CRS A465 - Caratteristiche giunti

J_1(vita)	±175°	180°/sec
J_2(spalla)	±90°	180°/sec
J_3(gomito)	±110°	180°/sec
J_4(yaw)	±180°	171°/sec
J_5(pitch)	±105°	173°/sec
J_6(roll)	±180°	171°/sec

1.2.1 Controllore C500C

Fig. 1.5: CRS A465: Controllore C500C

Il sistema viene fornito con il controllore C500C quest'ultimo è un semplice controllore

PID che opera su ogni motore del robot ed è in grado di eseguire oltre 30 processi contemporaneamente per il controllo completo del robot. Robcomm3 e ActiveX sono software di sviluppo che permettono di programmare il robot CRS-A465 in close architecture mediante l'utilizzo del linguaggio di programmazione RAPLIII che permette di insegnare al robot ad eseguire compiti specifici. Questo può avvenire interfacciandosi con un PC attraverso una porta seriale oppure mediante il teach pendant in dotazione. Per utilizzare il robot in open-architecture il controllore è fornito di una CRS/MultiQ Adapter Board (CRSMQ), cioè una scheda sul retro del telaio del controllere che ha due slot per connettori flat che servono per comuncare con la scheda MultiQ installata sul pc.

CRS/MultiQ Adapter Board (CRSMQ)

Per passare dalla modalità close-architecture alla open-architecture, si deve agire sull'interruttore etichettato CRS/MultiQ che effettua il passaggio di controllo da C500C al proprio controllo eseguito attraverso Simulink. L'interruttore può anche essere controllato digitalmente dal PC utilizzando la porta digitale numero #0 della scheda MultiQ. Si fa presente che nel momento in cui si passa in modalità open-architecture, tutti i PID interni del C500C vengono disattivati lasciando il completo controllo dei singoli giunti al pc che potrebbe far cadere a terra i giunti se non li si alimenta in maniera appropriata o peggio potrebbero sbattere violentemente se erroneamente alimentati a tensione massima.

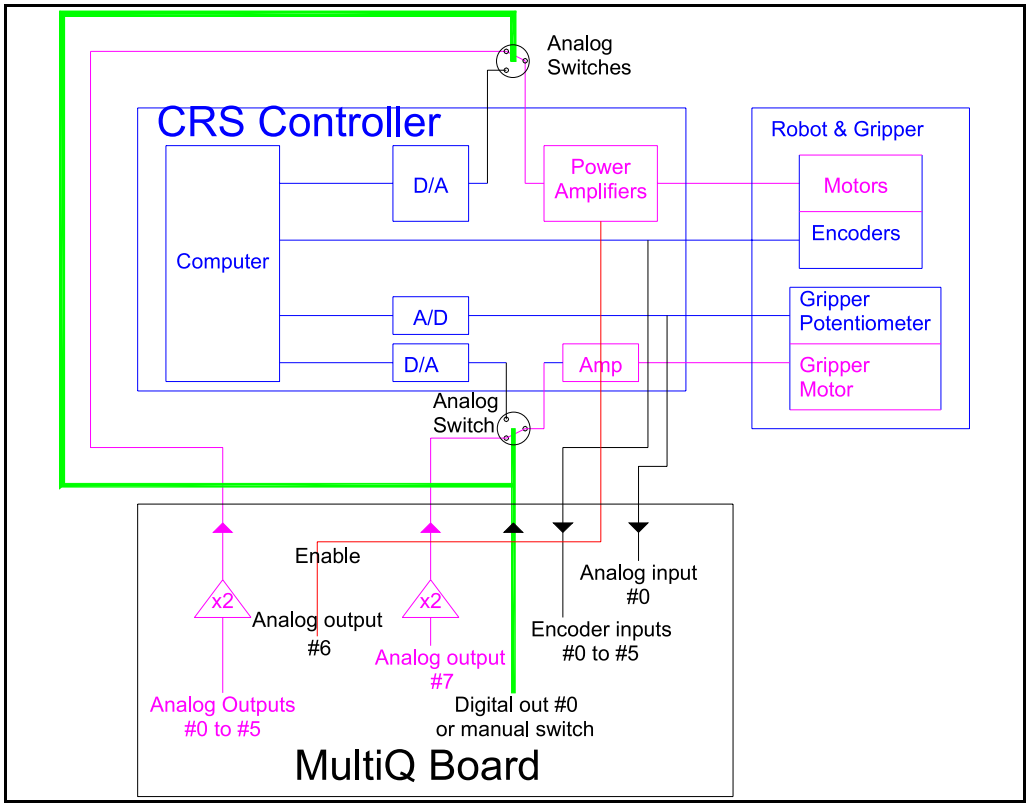

Fig. 1.6: MultiQ Analog e Digital I/O

1.3 Scheda MultiQ-3

Il dispositivo MultiQ-3 è una una scheda di acquisizione e controllo dati multiuso che dispone di 8 ingressi analogici single ended, 8 uscite analogiche, 16 bit per gli ingressi/uscite digitali, 3 timer programmabili e 8 ingressi encoder decodificati in quadratura.

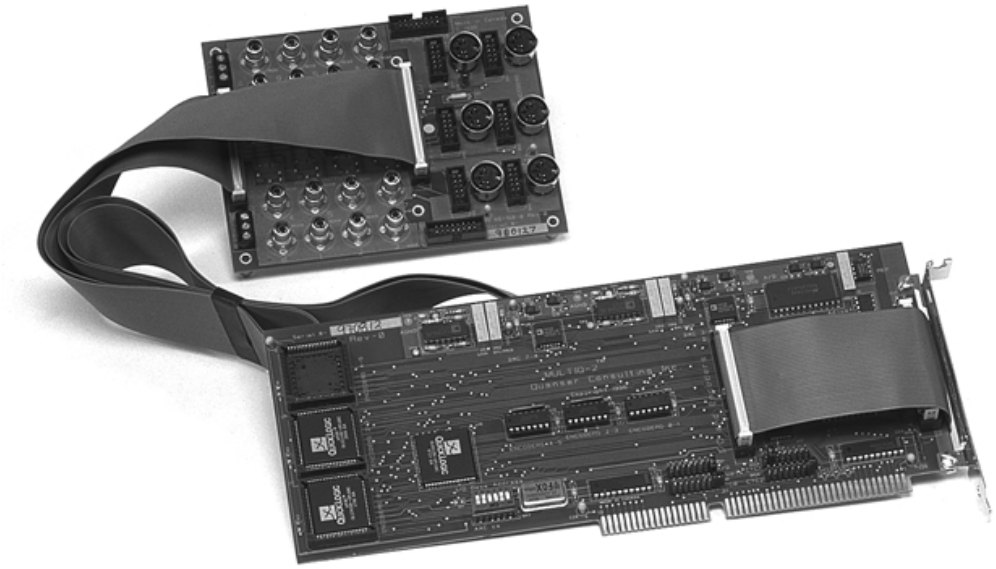

Fig. 1.7: Scheda MultiQ-3

Conversione analogico digitale

La conversione A/D del MultiQ-3 è di tipo single ended bipolare con segno a 13 bit (12 bit più segno). É possibile eseguire una conversione in uno degli 8 canali, selezionando il canale e iniziando la conversione. Il bit EOC_I (end of conversion interrupt) nel STATUS REGISTER indica che il dato è pronto e può essere letto. I dati vengono letti dal AD_DATA register. I dati restituiti sono due parole a 8 bit che devono essere combinati per dare come risultato una parola a 16 bit con segno. Il range accettabile in ingresso alla scheda di acquisizione dati è ±5 V. Un ingresso di +5 V viene codificato con 0xFFF mentre 0 V con 0x0 e -5 V con 0xFFFF000. Tutti gli ingressi del multiplexer dell' A / D sono single ended nel range ±5 Volt e dovrebbero essere collegati ai jack RCA etichettati con ingressi analogici.

Uscite analogiche

I convertitori digitali-analogici (D/A) sono a 12 bit, binari, senza segno. Un input di -5V è codificato con 0x000, 0V con 0x3FF, +5V con 0xFFF. I programmi che dovranno essere scritti quindi dovranno scrivere un numero a 12 bit (0-4.095) in un apposito registro. Le uscite analogiche cambiano quando i dati del registro sono aggiornati.

Ingressi encoder

La scheda può essere equipaggiata con un massimo di otto decoder encoder (Modelli-2E, 4E, 6E e 8E). I dati dell'encoder vengono decodificati in quadratura e utilizzati per incrementare o diminuire un contatore a 24 bit. Con 24 bit, è possibile ottenere 16.777.215 conteggi. Con un encoder lineare in quadratura e risoluzione pari a 2000 inpulsi/giro, questo si traduce in una risoluzione effettiva pari a 8.000inpulsi/giro il che può essere misurata senza overflow dei contatori. Conteggi superiori possono essere gestiti tramite un software.

Ingressi digitali

La scheda può leggere 16 linee di ingresso digitale connesse a un indirizzo di I/O. L'ingresso digitale è normalmente alta ('1') e risulta basso ('0') quando la linea è portata al potenziale GND. La linea di ingresso digitale #0 può essere legata ad un interrupt mediante i jumper in dotazione.

Uscite digitali

La scheda in grado di controllare 16 uscite digitali mappate a un indirizzo di I / O. Se si scrive uno '0' per il bit appropriato risulta 0V (TTL LOW) in uscita, mentre la scrittura di un '1' porta l'uscita a 5 volt (TTL HIGH).

Clock realtime

La scheda è dotata di 3 timer clock indipendenti e programmabili. Ogni timer può essere programmato per lavorare ad una frequenza compresa tra 2 MHz e 30,52 Hz. Il principio di funzionamento è quello di scrivere un divisore (N) al clock desiderato e la frequenza di uscita sarà 2.0 / N MHz. (N) è un valore intero a 16 bit compreso tra 2 e 65535 (0xFFFF). L'uscita di uno dei tre clock può essere legata ad una linea di interrupt con un jumper sulla scheda.

1.4 Sensore ottico di prossimità

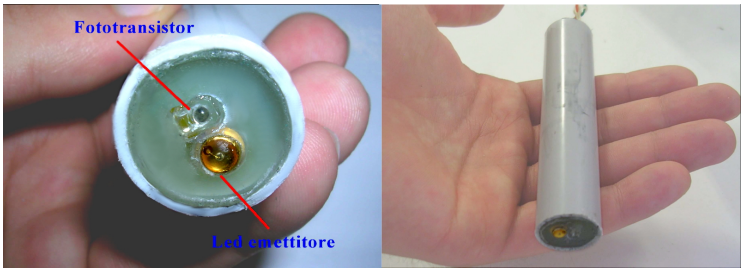

Fig. 1.8: Sensore ottico di prossimità

Il sensore ottico (o proximity ottico) utilizzato è stato realizzato in una tesi precedente, e si compone di:

- un diodo led infrarosso LD 242 7800 della OSRAM
- un fototransistor PT480F prodotto dalla SHARP

Tra le varie tipologie di sensori di prossimità è stato scelto il proximity ottico, perché ha un basso costo, una portata nominale sufficiente al caso in questione e grazie ad un contenitore stagno, possono essere utilizzati in acqua.
Il fascio viene riflesso dalla superficie dell'oggetto rilevato, per lo stesso fenomeno per cui la luce visibile può essere riflessa e percepita dai nostri occhi. Il problema è che la quantità di radiazione riflessa dipende dalla composizione e dall'orientamento della superficie; pertanto il campo sensibile di questi proximity dipende sostanzialmente dalla natura della superficie dell'oggetto da rilevare. Va inoltre posta attenzione al posizionamento di fonti di luce artificiale: la proiezione di una forte luce su questi sensori ne può provocare l'accecamento. Ecco quindi l'esigenza di far funzionare l'emettitore ad una frequenza di 10kHz, in modo che la ricezione del segnale riflesso, filtrando la sola componente a 10kHz, si priva di tutto il segnale di disturbo che si somma al segnale utile.

1.4.1 Analisi circuitale

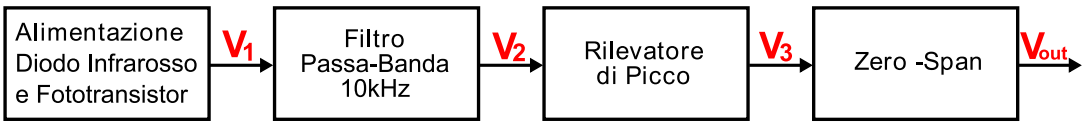

Fig. 1.9: Schema circuito sensore prossimita

Il circuito, progettato in una precedente tesi e assemblato su breadboard, è stato quindi creato su una più compatta scheda millefori che ne ha permesso un uso più pratico senza il continuo pericolo che qualche filo di collegamento si possa staccare.

Alimentazione Componenti

Si alimenta in maniera opportuna il diodo e il fototransistor:

- il diodo è alimentato da una tensione continua poco minore di 5V, poi in serie, un transistor la cui base dovrà ricevere un onda quadra con frequenza 10kHz e 5Vpp; questa configurazione permette l'intermittenza del diodo alla frequenza imposta.

- il fototransistor è direttamente collegato alla tensione continua di 5V; la base è sensibile alle onde emesse dal diodo, che vengono riflesse dalla superficie di un ostacolo quando il sensore raggiunge una determinata distanza, chiudendo quindi il circuito che trasmette tensione ai circuiti a valle. Tale tensione è legata in maniera inversamente proporzionale a tale distanza ma non linearmente.
  Zona fino a V_{out1} nella Fig. 1.10

Filtro Passa-Banda

Si filtra, attenuando tramite un filtro passabanda centrato alla frequenza di 10kHz, la tensione (V_{out1} in Fig. 1.10) inviata dal fototransistor che rappresenta la portante emessa dal diodo infrarosso. É fondamentale filtrare in maniera accurata il segnale, per eliminare tutti quei contributi di disturbo ambientale causati da lampade al neon, luce solare e da qualsiasi altra sorgente che emetta infrarossi a frequenza diversa da 10kHz che potrebbero rendere troppo rumoroso il segnale ricevuto.

Tenuta del Picco

La tensione in uscita dal filtro passa-banda (V_{out2} in Fig. 1.10) è sinusoidale, per cui è stato necessario linearizzarlo mantenendolo per una costante di tempo $\tau = R * C$, pari a circa 50 volte la frequenza del segnale in ingresso; in questo modo si ottiene una tensione (V_{out3} in Fig. 1.10) dipendente solamente dal variare del picco del segnale.

Zero-Span

Il segnale però ancora non va bene, la scheda di acquisizione MultiQ-3 vuole in ingresso un segnale analogico compreso tra 0 e 5V, quindi è necessario adattare l'ampiezza e sommare un offset.
Circuito tra V_{out3} e V_{out} in Fig. 1.10

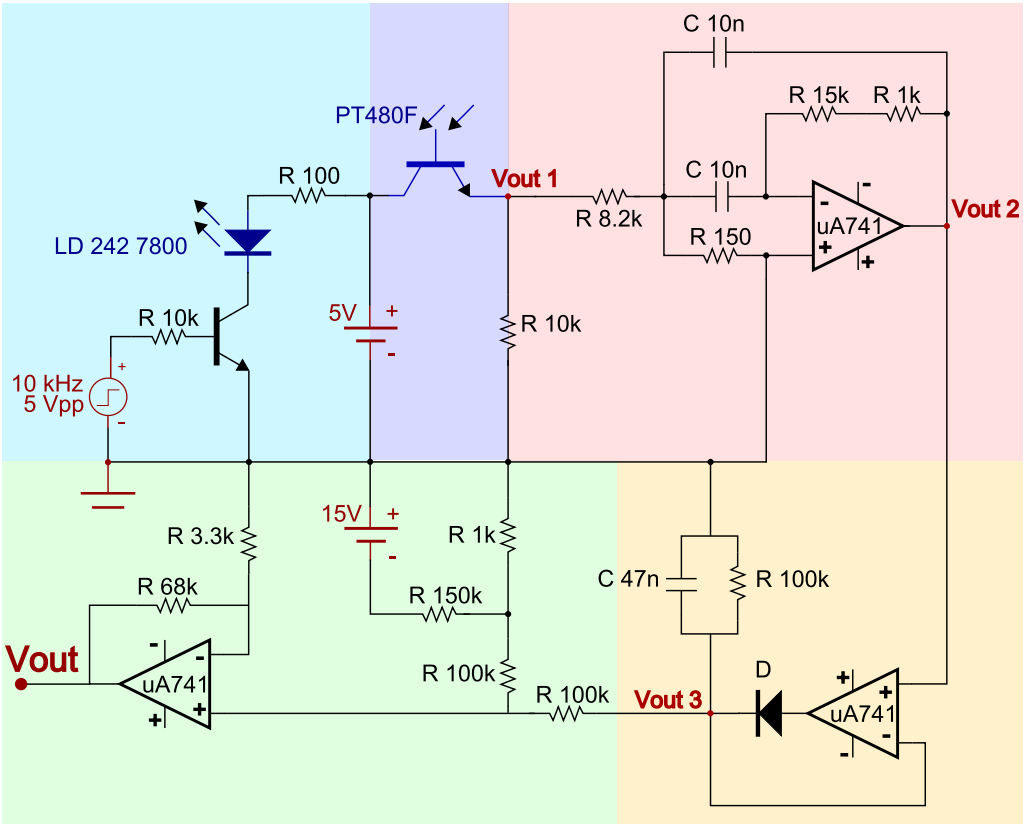

Fig. 1.10: Circuito sensore

1.4.2 Funzione

Il sensore segnalerà quindi la presenza di un ostacolo solo quando questo avrà raggiunto una determinata vicinanza, tale distanza dipende fondamentalmente dai componendi adottati per il sensore: potenza led e sensibilità fototransistor.
Raggiunta tale soglia il sensore, tramite circuito di condizionamento, invierà un segnale inversamente proporzionale (non lineare) al diminuire della distanza tra sensore e ostacolo.
La tensione inviata alla scheda di acquisizione sarà:

- 0V quando il sensore non capta nessun ostacolo nelle vicinanze,
- 5V quando il sensore è a meno di 0.5mm dalla superficie di un ostacolo,
- $0V < V_{out} < 5V$ quando sono nelle vicinanze di un ostacolo.

Si fa notare che il fototransistor converte in segnale solo le onde infrarosse che si riflettono su una superficie di ostacolo, quindi più questa è scura e meno si riflette la luce e viceversa. In altri termini la trans-caratteristica distanza/tensione dipende dal tipo di superficie e dal fluido (aria, acqua, ecc...) in cui è immerso ostacolo e sensore.

Capitolo 2

Interfacciamento dispositivi

Si analizzano ora tutti i blocchi funzionali che hanno permesso l'interfacciamento tra i dispositivi precedentemente descritti per ottenere il sistema aptico.
La differente architettura dei due bracci (CRS A465 e PHANTOM Omni), nonostante entrambi presentino 6 gradi di libertà, hanno reso necessarie delle scelte preliminari al fine di ottenere un sistema di riferimento comune in grado di far comunicare i due dispositivi. Si è poi dovuto tener conto della diversa orientazione degli assi cartesiani *xyz* e del diverso range di movimento tra i due bracci, molto limitato nel PHANTOM perchè, come da descrizione, il suo spazio di lavoro è delimitato dalla sola area coperata dal solo movimento del polso della mano; si è poi dovuto riscalare il movimento di quest'ultimo in scala del'A465.
L'intero sistema aptico è stato sviluppato in ambiente Simulink di MATLAB, che grazie alla sua interfacia grafica minimalista e modulare, ha permesso un'implementazione standalone dei vari blocchi funzionali man mano che venivano realizzati. Questa flessibilità ha reso possibile lo sviluppo del progetto a livelli di astrazione, in modo tale da mascherare la complessità di una funzione, precedentemente realizzata e supposta funzionante in tutti i possibili casi, e sfruttarne le funzioni all'interno di un altro blocco funzionale più esteso.

2.1 Dispositivo Slave: Thermo CRS A465

Il robot richiede particolare attenzione nel passaggio del controllo da close-architecture a quello open-architecture, in quanto passando a quest'ultima modalità il controllore C500C rilascia tutti i suoi controlli interni, mantenendo la funzione di amplificatore di tensione dei motori sui giunti e la funzione di collegamento tra gli encoder dei giunti e la scheda di acquisizione MultiQ-3, lasciando all'utente la completa gestione di questi.

2.1.1 Encoder Input

Il blocco S-Function ***Encoder Input*** permette la lettura e la successiva codifica dei segnali provenienti da ognuno degli otto ingressi encoder della scheda MultiQ (installata sul retro del controllore C500C). Gli encoder dell'A465 vengono decodificati in quadratura (cioè in doppia traccia) e utilizzati per incrementare o diminuire i rispettivi contatori a 24 bit. L'uscita del blocco è un vettore di dimensione otto, in quanto ogni elemento di questo vettore costituisce il risultato di un contatore; tali elementi dovranno essere moltiplicati per una costante per ricavare un valore angolare espresso in gradi , tuttavia queste costanti dipendono dal tipo di encoder utilizzato. Si ricorda che essendo encoder a impulsi, i contatori devono essere abilitati solo quando tutti i giunti sono in una posizione specifica, pena il disaccoppiamento tra posizione effettiva e valori di encoder.
Il file ***Sfun_ENC_Res.c*** implementato in questa S-Function, gestisce anche questa problematica; avendo come ingresso un segnale binario che va a zero quando voglio passare in open-architecture, si assume che finchè tale valore è alto, i contatori si mantengano resettati. L'utente, prima di passare in modalità aperta, deve assicurarsi che il robot sia in una particolare posizione definita dal costruttore, chiamata *Ready Position*, raggiungibile solamente tramite il comando remoto (teach pendant) del controllore C500C, premendo il tasto indicato come *ready*. Tenendo premuto tale pulsante, una procedura interna automatica del C500C, porta tutti i giunti alla posizione prestabilita che sarà per noi lo zero di

riferimento di tutti gli encoder; quando tutti i giunti si arrestano per aver raggiungono la ready position, l'utente può rilasciare il tasto e passare all'architettura aperta. Passando in quest'ultima, la variabile d'ingresso abilita quindi i contatori che varieranno al muoversi dei giunti; potendo trasmettere tali valori al controllo di posizione che si occuperà di generare le tensioni per pilotare correttamente i giunti secondo i riferimenti.
Prima di poter essere utilizzato, il vettore d'uscita, deve passare per il blocco *Convert* che lo converte in formato *word*, in modo che altri blocchi possano leggere tali valori.

2.1.2 Controllori PD di Posizione

Il controllo di posizione dei giunti è un punto molto importante dell'elaborato perchè da questo dipende la correta movimentazione del CRS A465.
Il blocco riceve in ingresso i riferimenti angolari e i valori di encoder (in gradi) di tutti i 6 giunti. Si è scelto di implementare un controllore di tipo PD (proporzionale-derivativo) per limitare il più possibile l'errore tra riferimento e posizione, e per assicurare una buona stabilità del sistema.

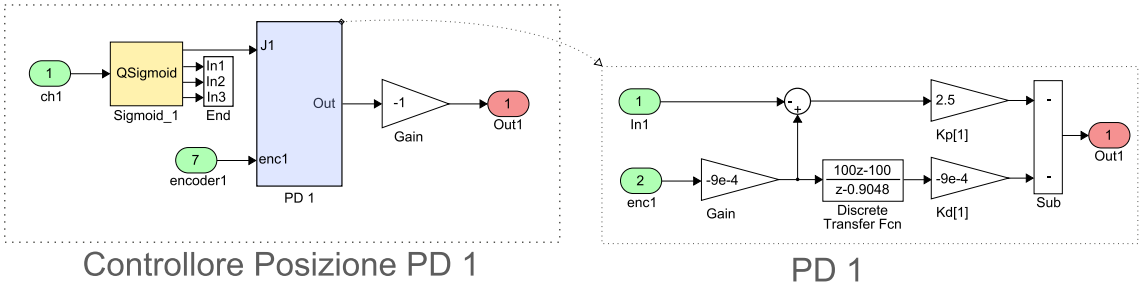

Fig. 2.1: Blocco: Controllo Posizione

Sigmoidi

Prima di entrare nei subsystem PD, i 6 riferimenti angolari entrano in altrettanti blocchi S-Function ***Sigmoid***, in cui è caricata la funzione *QSigmoid.c*, presa dalla libreria del robot. Tale funzione è molto utile perchè permettendo di impostare i parametri di massima velocità e accelerazione del giunto, se il riferimento in ingresso ha velocità o accelerazione maggiore di quanto indicato nei parametri (espressi in $gradi/sec$ e in $gradi/sec^2$), la funzione ricalcola il riferimento con tali valori massimi. L'utilizzo della sigmoide è di estrema utilità soprattutto quando trasmettendo al robot riferimenti angolari a gradino distanti da quelli di encoder, evita che i PD generino gradini di tensione (con il conseguente movimento a scatto dei giunti), creando un riferimento con profilo a 'S' con velocità e accelerazione massima stabilita, dando di fatto il tempo al PD di vincere le inerzie in gioco ottendo movimenti sinuosi. Va sottolineato che la funzione genera di fatto un riferimento alternativo che cerca di inseguire nel tempo il riferimento in ingresso, ma avendo valori di velocità e accelerazione limitati, smussa le variazioni troppo rapide che non riesce ad inseguire; questo si traduce in effetto negativo se, impostando valori troppo lenti, si vuole attuare un movimento di andata e ritorno, il riferimento d'uscita non raggiungerà il punto finale stabilito se il tempo a disposizione per effettuare il percorso è troppo breve, mentre si traduce in effetto positivo se il segnale di riferimento presenta disturbi che vogliamo filtrare linearizzandoli.

Costanti dei Controllori

Per ognuno dei 6 giunti il robot ha un servomotore e un encoder di prossimità I valori delle costanti proporzionali e derivative utilizzate per i 6 giunti sono:

$$k_p = [\ 2.5\ 2.5\ 2.5\ 0.5\ 0.5\ 0.3\ ] \tag{2.1.1}$$

$$k_d = [\ 0.05\ 0.05\ 0.05\ 0.005\ 0.005\ 0.002\ ] \tag{2.1.2}$$

I valori delle costanti per ricavare le posizioni in gradi dei 6 encoder sono:

$$K_{encoder} = [\ -0.0009\ \ -0.0009\ \ -0.0009\ 0.0018\ 0.0018\ 0.0018\ ] \tag{2.1.3}$$

L'errore in uscita è quindi un vettore di dimensione 6, i cui valori rappresentano le tensioni da inviare al blocco *Analog Output* che le invierà al controllore per effettuare un movimento o per autosorreggersi dalla gravità.

2.1.3 Analog Output

Per poter trasmettere un segnale analogico in tensione al controllore C500C si è utilizzato il DAC della scheda MultiQ-3, è stato quindi necessario realizzare un blocco che codifichi un valore in tensione, compreso tra $\pm 5V$, in un valore compreso tra 0 e 4095 incluso, in quanto la scheda effettua una conversione D/A a 12bit senza segno.

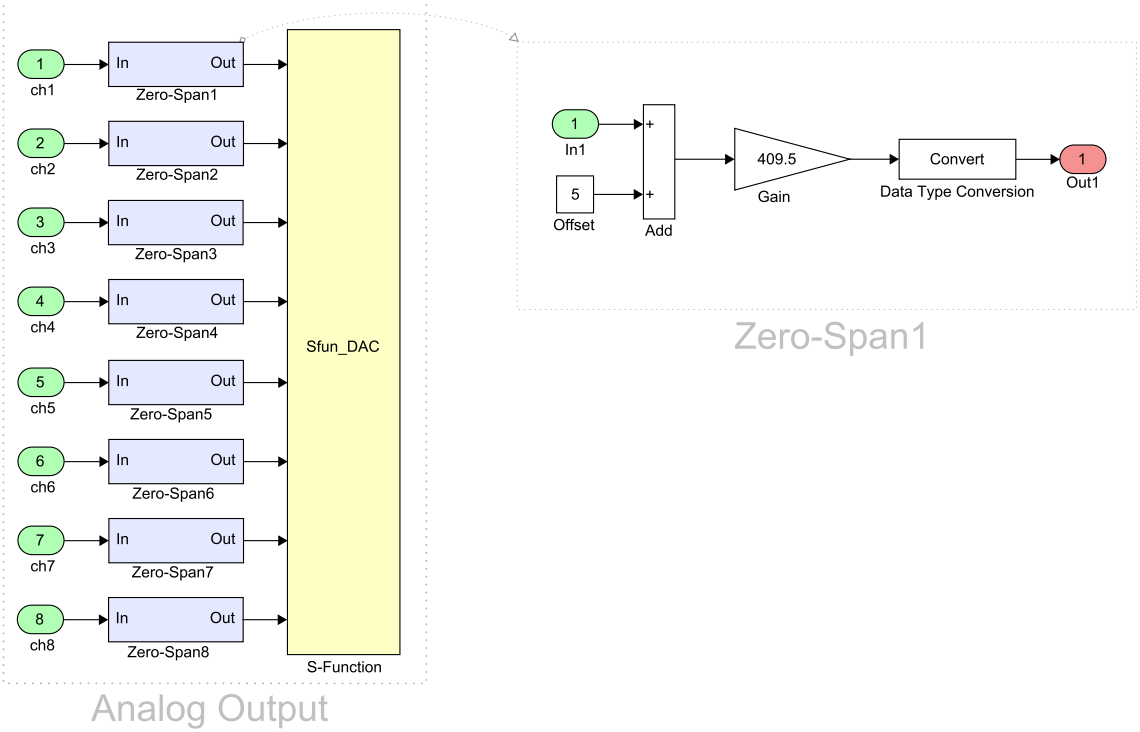

Fig. 2.2: Blocco Analog Output

L'adattamento del segnale avviene tramite l'aggiunta di un offset e un riscalamento come scritto in seguito:

$$k = \frac{x'_{max} - x'_{min}}{x''_{max} - x''_{min}} = \frac{4095 - 0}{5 - (-5)} = 409,5 \tag{2.1.4}$$

$$offset = 5 \tag{2.1.5}$$

Il blocco ***Analog Output*** si presenta con 8 ingressi, che sono gli 8 canali gestibili dal DAC della scheda, i segnali entrano poi nei blocchi ***Zero-Span*** dove vengono incrementati di un offset, amplificati e convertiti dal tipo *double* al tipo *word*; i segnali sono quindi nella

forma corretta e pronti per essere trasmessi alla scheda. Tale compito spetta al blocco ***S-Function*** in cui è stato caricato il file **Sfun_DAC.c**: un programma scritto in codice C in cui vengono definiti gli indirizzi di memoria, la frequenza di clock del D/A e varie funzioni per la corretta inizializzazione del canale e per la trasmissione dei valori.

2.1.4 CRS Controller

Il SubSystem ***CRS Controller*** inglobando i blocchi precedentemente descritti, ci nasconde la complessità delle specifiche funzioni svolte e ci permette di assumere che: dati in ingresso i 6 riferimenti di posizone angolare (espressi in gradi) e lette le 6 variabili di encoder, mandando queste in ingresso al blocco *Controllori PD* si ottengono le tensioni per azionare i motori dei giunti; ma prima di mandarle al blocco *Analog Output*, devono passare per un blocco **saturazione** che limita il segnale in modulo a 5V, perchè i valori d'uscita dei PD dipendendo dall'errore tra riferimento e posizione di encoder potrebbero risultare valori ben più elevati.

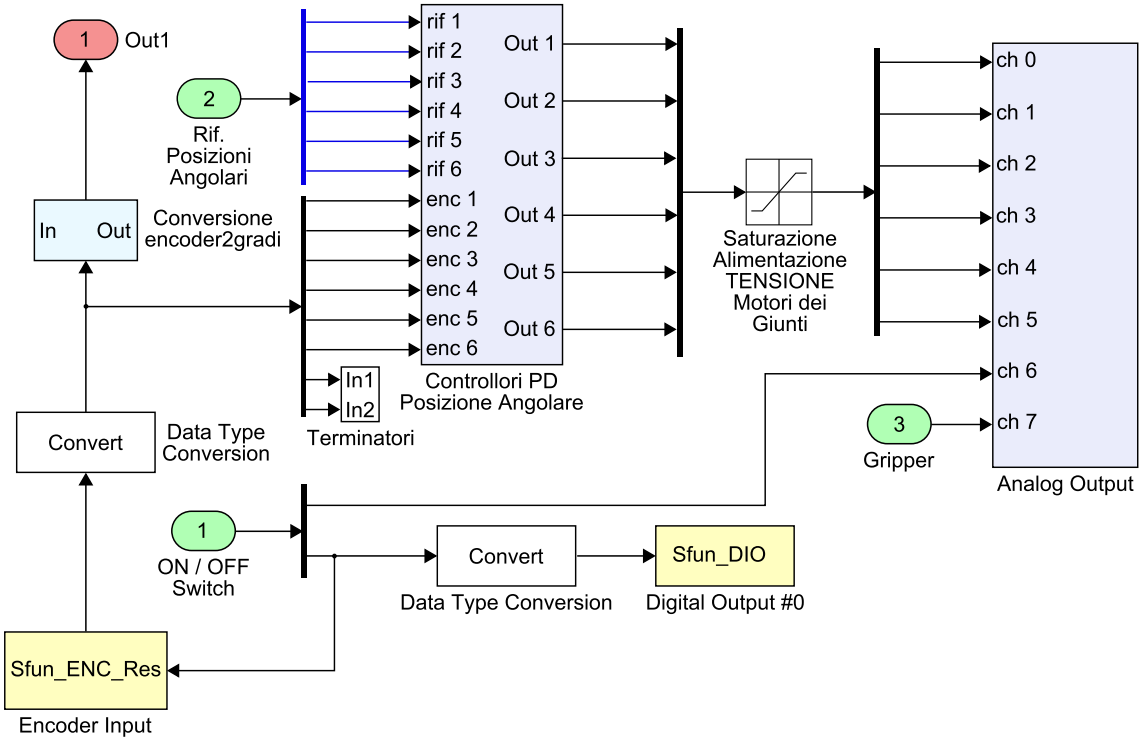

Fig. 2.3: CRS Controller

Digital Output

Il blocco ***Digital Output*** permette di pilotare, tramite la funzione '*SFunf_DIO.c*', una delle sedici uscite digitali presenti nella scheda MultiQ. Un valore d'ingresso pari a 0 porta l'uscita digitale a 0V (*low*) mentre un valore d'ingresso pari a 1 porta l'uscita a 5V (*high*).

Close/Open Architecture

Come definito nelle specifiche della scheda MultiQ per passare dalla modalità *Close-Architecture* alla modalità *Open-Architecture*, la Digital Output #0 deve passare da valore alto a valore basso (da 1 a 0) e l'Analog Output #6 che deve passare da 5 a 0.
Vale il viceversa.

Queste due uscite vengono pilotate dai blocchi *'Analog Output'* e *'Digital Output #0'* che

Tab. 2.1: CRS A465 OA-mode

	Analog Out #6	**Digital Out #0**
Close-Architecture	5	1 (*High*)
Open-Architecture	0	0 (*Low*)

ricevono i valori da un vettore di dimensione due in ingresso al SubSystem *CRS Controller*. Vedi Fig. 2.3.
Arrivati a questo punto, il robot CRS è in grado di raggiungere e mantenere le posizioni angolari che riceve, superando quindi la gravità del peso proprio.

2.1.5 Blocco CRS-A465

I livelli fin ora descritti si sono occupati di garantire una corretta movimentazione dei giunti del robot, dati dei riferimenti in ingresso; ora ad un livello più alto, si analizza la fase transitoria che intercorre tra la modalità close-architecture all'open-architecture. Il blocco ***CRS_A465*** riceve in ingresso tre segnali:

- *ON/OFF*, il vettore di dimensione due contenente i valori per l'uscita digitale e analogica per passare il controllo dal controllore C500C al controllo utente;
- *Buttons*, anch'esso un vettore di dimensione due, contenente gli stati binari dei due pulsanti del dispositivo master (PHANTOM Omni Device);
- *Joint*, il vettore dimensione sei con i rispettivi riferimenti angolari provenienti dal dispositivo master.

Nell'istante in cui i segnali di ON/OFF comandano l'architettura aperta, essendo il robot in posizione di *Ready*, devono essere inviati ai controllori PD riferimenti nulli, in modo tale che il CRS mantenga la stessa posizione nel passaggio di controllo. Se nel passare da una modalità all'altra i riferimenti fossero distanti da tale posizione, il robot effettuerebbe un movimento a potenza massima troppo brusco, quindi da evitare.
Ottenuto il controllo e mantenendo il robot in posizione di Ready, per lo stesso problema di cambio di riferimenti, se l'operatore switcha a quelli generati dal PHANTOM, non potendo assumere una posizione identica, con angoli tutti nulli sui giunti, si genererebbe un gradino e il CRS inseguirebbe tale posizione con movimento troppo rapido, poichè le funzioni Sigmoide poste a valle hanno parametri di velocità e accelerazioni adatte ad un inseguimento rapido dei riferimenti, ma troppo rapide per un gradino, sopratutto in fase di cambio riferimenti. In favore di sicureza quindi è stato introdotto un by-pass che inserisce in serie una funzione Sigmoide molto lenta per ogni riferimento, in questo modo i giunti dell CRS si muoveranno lentamente; l'operatore può quindi permettersi di assumere una posizione e un'orientazione più approssimata. Una volta raggiunti tutti i riferimenti del PHANTOM le Sigmoidi lente si disabilitano permettendo una movimentazione molto più rapida limitata solamente dalla velocità dell'operatore nel muovere il dispositivo master o dall'intervento delle sigmoidi 'veloci'.
Vale anche il viceversa, per passare dai riferimenti del dispositivo master ai riferimenti nulli di Ready-Position.

Per poter gestire correttamente i passaggi tra riferimenti nulli e riferimenti provenienti dal Phantom, e per l'attivazione/disattivazione delle Sigmoidi lente, è stata realizzata una '*macchina a stati initi*'.

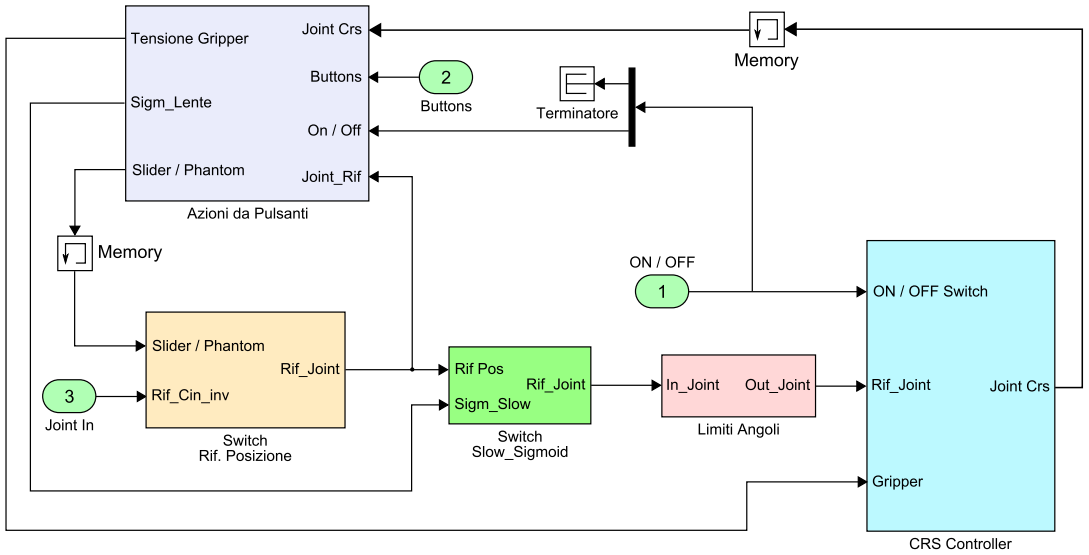

Fig. 2.4: Blocco: CRS_A465

2.1.6 Macchina a Stati Finiti

Il blocco **Azioni da Pulsanti** racchiude tutte le funzioni ottenibili dai pulsanti del Phantom Omni. Inizialmente sull'end-effector dell'A465, prima di installare il sensore ottico di prossimità, era installato un Gripper, per cui è stato implementato un controllo per aprire e chiudere le pinze tramite i due pulsanti sull'end-effector del Phantom. Qui si trova anche la funzione che permette all'utente di iniziare o fermare la movimentazione del CRS switchando tra i riferimenti nulli (*Ready Position*) e i riferimenti del Phantom. Si analizzano ora le funzioni implementate in questo blocco.

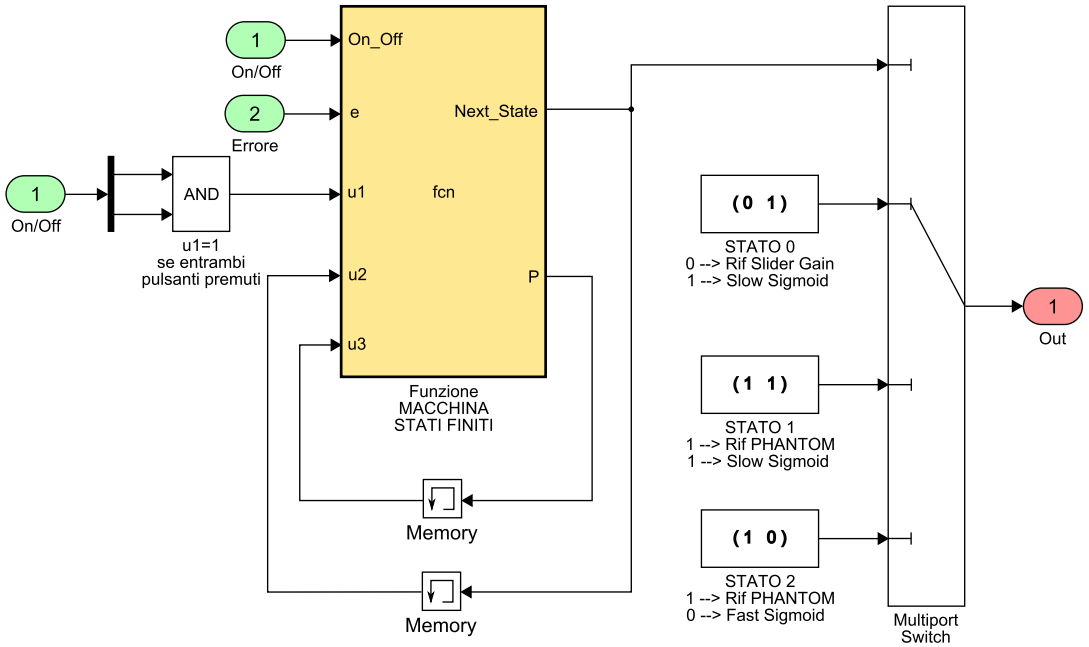

Fig. 2.5: Blocco: Azioni da Pulsanti

Gripper Controll

Questo blocco è stato realizzato per poter utilizzare il Gripper inizialmente installato sul robot. La posizione delle pinze viene letta tramite la funzione '*Sfun_ADC.c*' che prende il valore discretizzato dalla scheda MultiQ-3 a sua volta inviato dal sensore di prossimità del gripper. Per aprire e chiudere le pinze si incrementa/decrementa la variabile di riferimento tramite la pressione di uno dei due pulsanti sull'end-effector del PHANTOM e tramite un controllore PD si calcola la tensione da inviare al blocco *Analog Output*. Se le pinze, chiudendosi, trovano un ostacolo che ne impedisce la chiusura, incrementano la differenza tra riferimento ed encoder che, superando un determinato valore, congela il riferimento dall'ulteriore decremento e mantenendo quindi tale differenza per esercitare la forza necessaria per non far scivolare l'oggetto dalle pinze durante il sollevamento.

Errore in Modulo

Il blocco contiene una funzione (vedi Appendice C) che riceve in ingresso il vettore con le posizioni angolari di encoder e il vettore con i riferimenti angolari; per tutti i sei giunti calcola l'errore e salva nella varibile *err1* la norma dei primi tre giunti e in *err2* la norma degli ultimi tre giunti, che riguardano l'orientazione.

$$e_1 = enc_1 - rif_1;$$

$$err1 = \sqrt{(e_1)^2 + (e_2)^2 + (e_3)^2}$$

Poi una condizione verifica se i giunti hanno raggiunto tutti i riferimenti angolari, in particolare i primi tre giunti devono accumulare una norma inferiore al grado, mentre nei giunti d'orientazione si può assumere una tolleranza più larga considerando lo posizione raggiunta con una norma inferiore a 5°.

$$if(err1 < 1 \;\&\&\; err2 < 5)$$

Se la condizione viene soddisfatta la funzione pone l'uscita a 1, altrimenti rimane a 0. Tale uscita è fondamentale per il corretto funzionamento della macchina a stati finiti in quanto segnala l'avvenuto posizionamento dei giunti ai riferimenti potendo quindi disabilitare il by-pass delle sigmoidi lente che rallenta di molto il movimento dei giunti. La stessa procedura si ripete nel caso inverso, quando si vuole passare dai riferimenti del Phantom ai riferimenti nulli di ready: l'utente premendo insieme i pulsanti del dispositivo aptico, la macchina a stati finiti attiva le sigmoidi lente e i rifeirmenti nulli in modo tale che il robot si rposizioni lentamente in Ready-Position da qualsiasi posizione si trovi.

Macchina di Moore

Le variabili in ingresso sono:

- la linea On/Off, in particolare la linea che comanda le uscite digitali, che varia tra 0 e 1;
- l'errore in modulo che esce dall'omoninma funzione il quale mi segnala con un uscita binaria alta l'avvenuto posizionamento del robot ai riferimenti sotto ad un certo errore di margine;
- una linea binaria che va alta quando vengono premuti i due pulsanti del Phantom insieme.

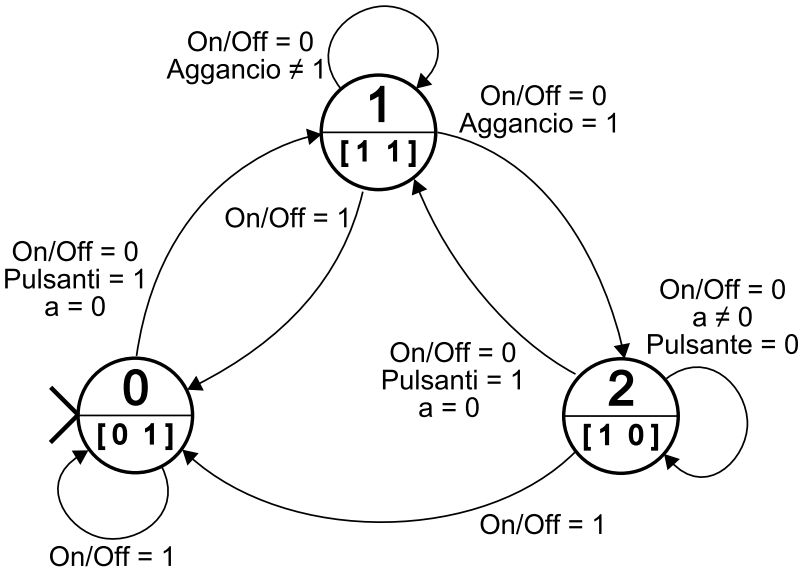

Fig. 2.6: Macchina di Moore

L'uscita della funzione (vedi Appendice D) è un numero intero compreso tra 0 e 2, che tramite un blocco '*Multiport Switch*', pilota l'uscita della macchina tra tre ingressi predefiniti.
Tale uscita è composta da un vettore binario di dimensione 2, in cui il primo elemento tramite un altro Multiport Switch comanda in uscita i riferimenti nulli di Ready Position o i riferimenti del Phantom; il secondo elemento invece attiva/disattiva le sigmoidi lente necessarie per un movimento in sicurezza tra il cambio di riferimenti.
L'Automa parte dallo **Stato 0** con l'uscita [0 1], quindi si presentano al robot riferimenti nulli e si attivano le sigmoidi lente; quando l'utente 'switcha' la linea On/Off iniziando la modalità OA, il robot insegue i riferimenti nulli e quindi non rimane nella stessa posizione, per poter comandare il robot tramite il dispositivo Phantom devono essere premuti contemporaneamente i pulsanti dell'end-effector passando quindi allo Stato 1. **Stato 1**, l'uscita assume i valori [1 1], quindi il Crs con velicità e accelerazioni dei giunti molto limitate insegue i riferimenti dinamici inviati dal dispositivo master; in questo stato l'utente può partire con il dispositivo in qualsiasi posizione e orientazione, con l'unico accorgimento di non assumere posizioni troppo basse altrimenti il CRS andrebbe a sbattere contro il tavolo di supporto, motivo per cui si è reso necessario l'utilizzo di movimenti lenti in questa fase di aggancio, cosi da permettere di correggere la posizione/orientazione del Phantom in caso l'utente si accorgesse di essere in una zona di probabile collisione del robot con un ostacolo o il piano di supporto. Quando i giunti dei due dispositivi hanno assunto posizioni angolari con scarto minimo, la funzione 'Errore in Modulo' ci segnala l'avvenuto aggancio ponendo l'uscita alta, che permette all'automa di passare allo Stato 2.
Lo **Stato 2** pone in uscita il vettore [1 0], che disabilita le sigmoidi lente permettendo un veloce inseguimento dei riferimenti del Phantom; solo in questo stato l'operatore può pilotare l'eventuale sonda ecografica installata sull'end-effector del robot tramite la movimentazione dell'end-effector del Phantom Omni. Per poter terminare la movimentazione e riportare il CRS in Ready-Position disaccoppiando l'inseguimento dei riferimenti dinamici, l'utente deve premere nuovamente entrambe i pulsanti, riportando quindi l'automa allo stato iniziale.
Si deduce quindi che per iniziare o terminare la movimentazione del CRS tramite il dispositivo Phantom, si devono premere entrambi i pulsanti.
Nel caso in cui l'utente tenga premuto i pulsanti anche quando i due dispositivi hanno

raggiunto il posizionamento, sarebbero verificate tutte le condizioni di passagio tra i tre stati, causando loop. Per evitare questo inconveninte, si è reso necessario l'utilizzo di una variabile semaforo 'a', vedi Appendice D, che viene settata a 1 quando rileva un fronte di salita sulla variabile dei pulsanti premuti insieme e a 0 quando vengono rilasciati; quindi il problema di loop viene eliminato.

2.1.7 Switch Riferimenti di Posizione

Questo blocco riceve in ingresso una linea binaria *Slider/Phantom* e il vettore dei sei riferimenti generati dal Phantom. Tramite un Multiport Switch a due ingressi, l'uscita dipende dal valore della linea binaria proveniente dalla macchina a stati finiti:

- se è a 0 vengono posti in uscita tutti riferimenti nulli; tali valori sono dati dai blocchi slider gain, utilizzati in fase iniziale per verificare il corretto inseguimento giunto per giunto.
- se invece ha valore 1 si inviano i riferimenti provenienti dal Phantom; per ognuno dei 6 riferimenti, grazie all'inserimento in serie di Switch, è possibile decidere giunto per giunto se inviare il riferimento proveniente dal master o inviare una costante.

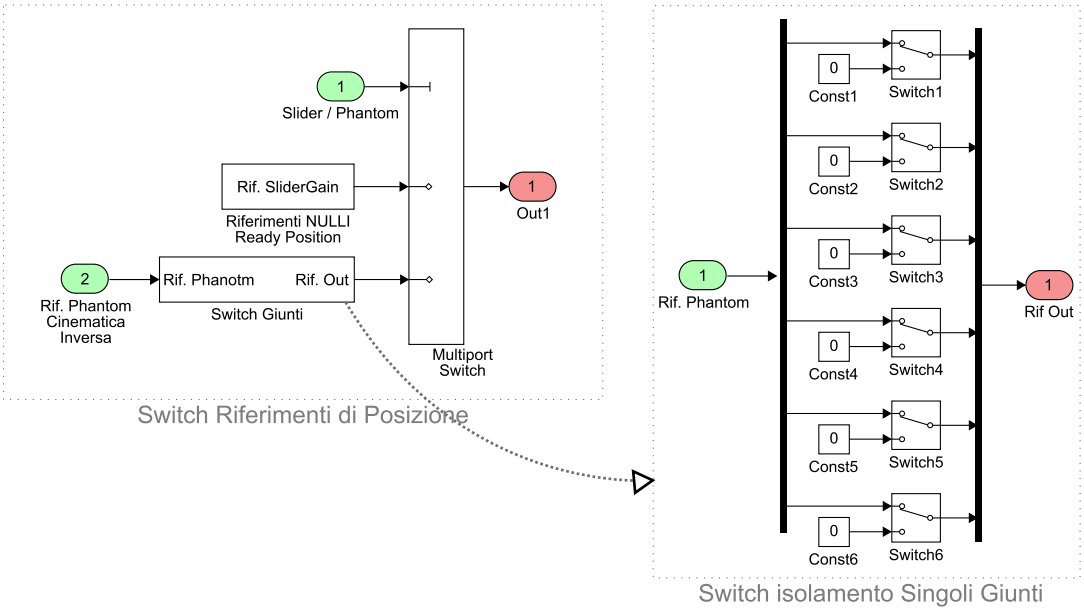

Fig. 2.7: Switch Riferimenti Ready Position / Phantom

2.1.8 Switch Slow Sigmoid

Anche questo blocco, come il precedente tramite un Multiport Switch a due ingressi, l'uscita è pilotata dalla linea binaria in ingresso provenienta dalla macchina a stati finiti che:

- se ha valore 0, i riferimenti vengono posti direttamente in uscita;
- se ha valore 1, ogni riferimento passa per un blocco Sigmoide, in cui sono stati settati parametri di velocità e accelerazione molto bassi, ottenendo i posizionamenti molto lenti voluti nelle fasi transitorie di cambio riferimenti.

2.1.9 Limiti Angoli

Avendo i giunti del robot A465 un range angolare ben definito, si è ritenuto opportuno implementare questo blocco contenente 6 blocchi *Saturazione*, prima dell'ingresso al blocco che ne attua il movimento. Tali blocchi, saturando i riferimenti con limiti di qualche grado inferiore ai fine corsa meccanico, sono di estrema utilità in quanto se un riferimento avesse valore massimo maggiore del limite fisico di un giunto, il controllore continuerebbe ad alimentarne il motore rischiando la rottura. Questo caso non è banale perchè i giunti del dispositivo Phantom Omni hanno range sui giunti diversi dai range del robot A465.

Tab. 2.2: CRS A465 - Limiti delle Saturazioni sui Riferimenti dei Giunti

Giunto	**Limite Saturazione**	**Fine corsa meccanico**
J_1 (vita)	-170°< J_1 <170°	-175°< J_1 <175°
J_2 (spalla)	-85°< J_2 <85°	-90°< J_2 <90°
J_3 (gomito)	-23°< J_3 <203°	-25°< J_3 <205°
J_4 (yaw)	-180°< J_4 <175°	-180°< J_4 <180°
J_5 (pitch)	-100°< J_5 <100°	-105°< J_5 <105°
J_6 (roll)	-180°< J_6 <175°	-180°< J_6 <180°

2.2 Cinematica Inversa

Completato il blocco CRS_A465, si deve realizzare il blocco che realizzi la cinematica inversa per ricavare i riferimenti angolari per il CRS-A465 dato in ingresso posizione ed orientazione cartesiana. La funzione **A465_WorldToJoint** proviene dalla libreria WinCon del robot e caricata in un blocco funzione (Fig. 2.8) e calcola dunque i riferimenti angolari di tutti sei i giunti del robot, ricevendo in ingresso un vettore di dimensione 6 di cui:

- i primi tre elementi corrispondono alle coordinate x y z,
- gli ultimi tre si riferiscono all'orientazione espressa rispettivamente nell'ordine *Yaw*, *Pitch* e *Roll*.

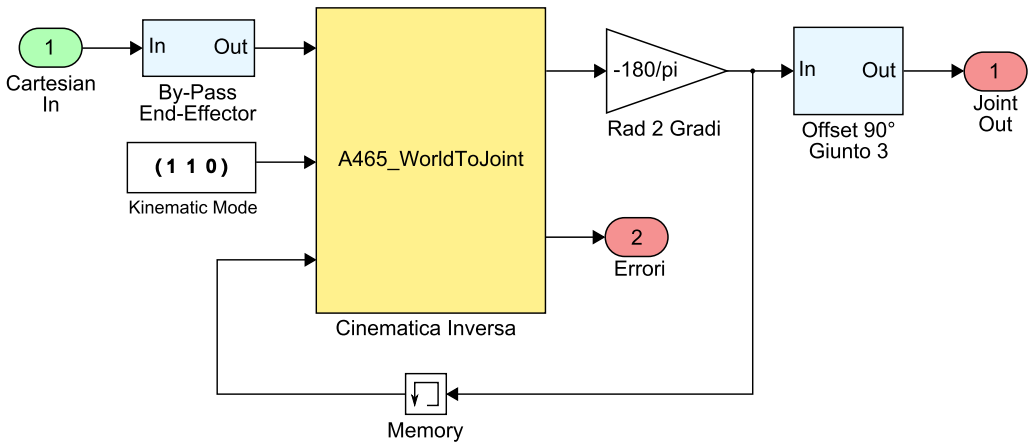

Fig. 2.8: CRS A465 - Cinematica Inversa

Per ovviare ai problemi di configurazione multipla della cinematica inversa, la funzione riceve in ingresso un vettore di dimensione tre i cui valori stabiliscono la modalità con cui il robot raggiunge la posizione desiderata.

Tab. 2.3: CRS A465: Modalità cinematica inversa

Valore	0	1	-1
Braccio	indietro	avanti	La cinematica inversa troverà la posizione migliore, se esiste
Gomito	sotto	sopra	
Polso	capovolto	no capovolto	

Fig. 2.9: Configurazioni Cinematica Inversa

La modalità più adatta a questo interfaciamento è sicuramente la *[1 1 0]*.
La funzione restituisce in uscita due vettori: uno di dimensione 6 che è il risultato della cinematica inversa, quindi contiene le posizioni angolari dei giunti del robot, il secondo vettore segnala la presenza di errori: se è nullo non ci sono errori, altrimenti presenta un numero negativo.
Gli errori che la funzione potrebbe porre in uscita sono i seguenti:

- *0xx1 giunto fuori portata*: uno o più giunti hanno superato il loro limite positivo o negativo del range di azione. Il giunto incriminato è indicato dal secondo numero del codice di errore.
  Ad esempio se il giunto fuori portata è J3 il codice di errore risulterà 0x31;
- *0xx2 trasformazione illecita*: nessuna soluzione di cinematica inversa soddisfa i vincoli dei giunti;
- *0xx3 errore nella cinematica inversa, fuori dalla portata*: la posizione data non può essere assolutamente raggiunta dal robot.

2.3 Dispositivo Master: Phantom Omni

Il Phantom Omni Device (descritto al cap:1.1), ha un implementazione più semplice rispetto al dispositivo precedente, in quanto grazie al blocco '*OmniDevice*' che pone in uscita le posizioni angolari e riceve in ingresso le coppie di feedback che attua ai primi tre giunti, non è stata necessaria un interfaccia a basso livello per la lettura degli encoder e per la gestione dei motori che permettendo di effettuare l'interfaccia aptica.
Per permettere la comunicazione tra le due interfacce dei dispositivi, essendo realizzati con architetture diverse, è stato necessario fissare un riferimento cartesiano univoco ed effetturare delle codifiche in seguito descritte.

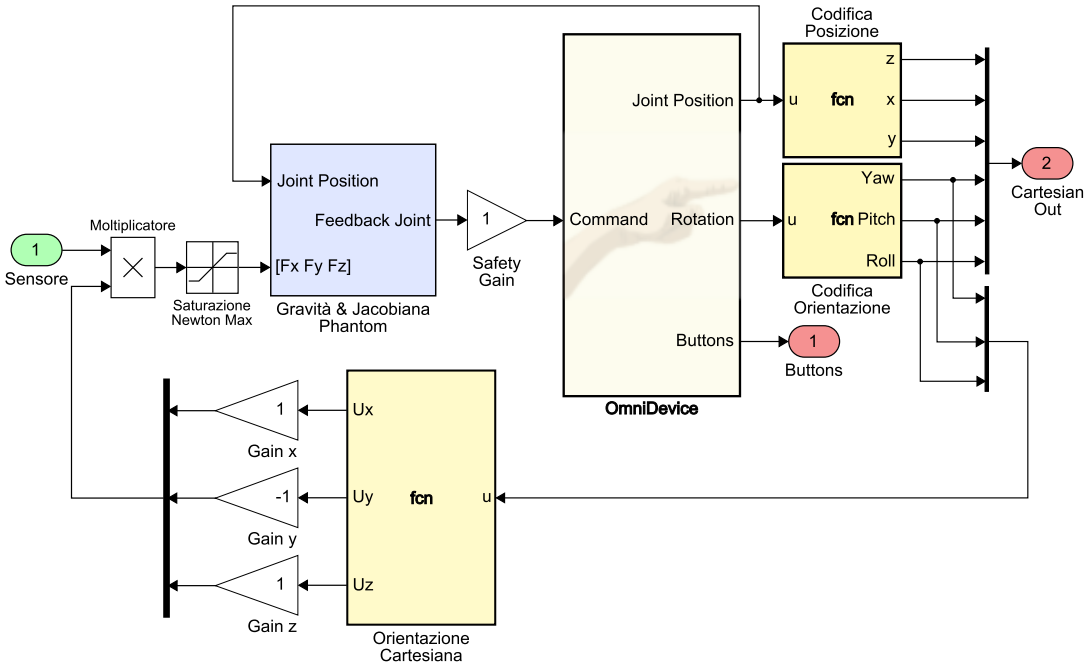

Fig. 2.10: Interfaccia dispositivo Phantom Omni

2.3.1 Blocco: OmniDevice

Il blocco è implementabile dalla libreria di Simulink solo installando il software Handshake proSENSE^{TM} (Fig. 2.10) e pone in uscita:

- *Joint Position*: un vettore di dimensione 3 contenente la posizione dell'end-effector in coordinate angolari dei giunti, espresse in radianti
- *Rotation*: un vettore di dimensione 4 che esprime l'orientazione dell'end-effector espressa in *Axis Angles* [1]
- *Buttons*: fornisce lo stato binario dei pulsanti sul dispositivo, caratterizato da un vettore di dimensione 2 rispettivamente associati ai 2 tasti.

Il blocco richiede l'impostazione del parametro *Sample time* in cui è stato inserito il valore di 0.001 secondi, analogamente all'intera interfaccia che utilizza lo stesso tempo.
L'ingresso del blocco riceve un vettore di dimensione 3, i cui valori pilotano la coppia sui motori dei tre giunti. I valori devono essere compresi tra -32768 e e 32767 (2_{15} più bit di segno) e attuano rispettivamente coppia massima negativa e la coppia massima positiva, al valore nullo naturalmente corrisponde coppia nulla. Grazie a questo ingresso è possibile togliere la gravità e inviare i segnali di feedback realizzando cosi un interfaccia aptica.

2.3.2 Codifiche dei Riferimenti

Si analizzano quindi le codifiche necessarie per ottenere i riferimenti adatti per poter pilotare il dispositivo slave A465; l'obiettivo è di porre in uscita la posizione in coordinate cartesiane xyz coerenti con le cordinate del dispositivo slave, e l'orientazione espressa in Yaw, Pitch e Roll dell'end-effector del dispositivo Omni.

[1] Gli axis angles sono una rappresentazione di una rotazione, sono costituiti da due valori: un versore che indica la direzione di un asse (linea retta), e un angolo che descrive la entità della rotazione attorno all'asse appena definito. La rotazione avviene nel senso previsto dalla regola della mano destra.

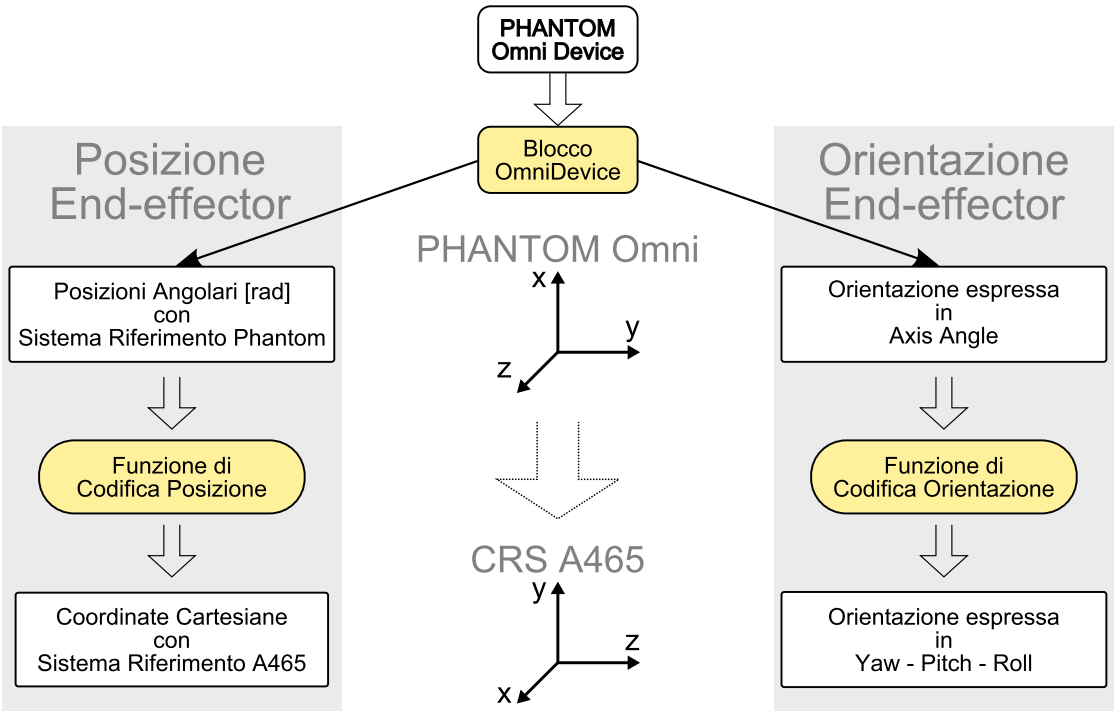

Fig. 2.11: Conversione Riferimenti

2.3.3 Codifica di Posizione

Come già accennato, il Phantom esprime la posizione dell'end-effector in coordinate angolari, espresse in radianti, quindi si dovrà convertire tali riferimenti in coordinate cartesiane opportunamente scalate con le lunghezze dei giunti del robot A465. Si evidenzia anche la necessità di un cambiamento del sistema di riferimento in quanto tra i due dispositivi essi sono differenti (vedi Fig. 2.11). Il blocco funzione *Codifica di Posizione* (listato all'Appendice A.1) di fatto realizza una Cinematica Diretta: utilizzando i valori angolari dei primi tre giunti del Phantom in ingresso e definendo lunghezze dei primi 3 giunti del CRS, generando quindi le matrici di rotazione (R_{base}, R_{link1}, R_{link2}) secondo il relativo asse e sommando i vari vettori adeguatamente ruotati e sommandoci pure il vettore che identifica la base d'appoggio del robot, è possibile ottenere le coordinate cartesiane del end-effector secondo il sistema di riferimento del Phantom.

$$R_{base} = \begin{bmatrix} cos(u[1]) & 0 & sin(u[1]) \\ 0 & 1 & 0 \\ -sin(u[1]) & 0 & cos(u[1]) \end{bmatrix} \tag{2.3.1}$$

$$R_{link1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(u[2]) & -sin(u[2]) \\ 0 & sin(u[2]) & cos([2]) \end{bmatrix} \tag{2.3.2}$$

$$R_{link2} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(u[3]) & -sin(u[3]) \\ 0 & sin(u[3]) & cos([3]) \end{bmatrix} \tag{2.3.3}$$

Per ottenere le coordinate nel sistema di riferimento del robot CRS è bastato invertire la l'ordine delle tre linee d'uscita, associando:

$$X_{CRS} = Z_{PHANTOM}$$

$$Y_{CRS} = X_{PHANTOM}$$
$$Z_{CRS} = Y_{PHANTOM}$$

2.3.4 Codifica Orientazione

La soluzione adottata per la conversione da *Axis Angles* a *Yaw-Pitch-Roll* prevede una conversione divisa in due stadi (vedi Fig. 2.12), passando cioè per la rappresentazione d'orientazione in *Matrice di Rotazione*.

Fig. 2.12: Stati di Conversione Orientazione End-Effector

Codifica da Axis Angles a Matrice di Rotazione

La rappresentazione dell'orientazione secondo gli Axis Angles è definita da un vettore di dimensione quattro:

$$\mathbf{w} = [\ k_x\ k_y\ k_z\ \theta\ ] \tag{2.3.4}$$

La matrice di rotazione è una matrice di dimensioni 3x3 che può essere ottenuta mediante i valori del vettore (2.3.4) con la seguente formula:

$$\begin{aligned} \mathbf{R} &= \begin{bmatrix} k_x k_x v\theta + c\theta & k_x k_y v\theta - k_z s\theta & k_x k_z v\theta - k_y s\theta \\ k_x k_y v\theta + k_z s\theta & k_y k_y v\theta - c\theta & k_y k_z v\theta - k_x s\theta \\ k_x k_z v\theta + k_y s\theta & k_y k_z v\theta + k_x \theta & k_z k_z v\theta + c\theta \end{bmatrix} \\ &= \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \end{aligned} \tag{2.3.5}$$

dove $c\theta = cos(\theta)$, $s\theta = sin(\theta)$, $v\theta = 1 - cos(\theta)$.

Codifica da Matrice di Rotazione a Yaw - Pitch - Roll

Una rotazione generica può essere ottenuta come somma di tre differenti rotazioni rispettivamente sui tre assi di un sistema di riferimento fisso secondo gli angoli roll, pitch e yaw. Quindi potendo esprimere una rotazione intorno ad un asse con una matrice 3x3, una generica rotazione essendo il susseguirsi di tre rotazioni intorno ad a tre assi, può essere rappresentata da un unica matrice avente stessa dimensione 3x3 risultante dal prodotto delle tre matrici di rotazione.

$$\begin{aligned} \mathbf{R}(\gamma, \beta, \alpha) &= \mathbf{R}_z(\alpha)\mathbf{R}_y(\beta)\mathbf{R}_x(\gamma) \\ &= \begin{bmatrix} c\alpha & -s\alpha & 0 \\ s\alpha & c\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\beta & 0 & s\beta \\ 0 & 1 & 0 \\ -s\beta & 0 & c\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\gamma & -s\gamma \\ 0 & s\gamma & c\gamma \end{bmatrix} \end{aligned} \tag{2.3.6}$$

Dove $c\alpha$ sta per $cos(\alpha)$, $s\alpha$ per $sen(\alpha)$ e cosi via. Naturalmente la moltiplacazione tra matrici non gode della proprietà commutativa, quindi è fondamentale disporre le matrici

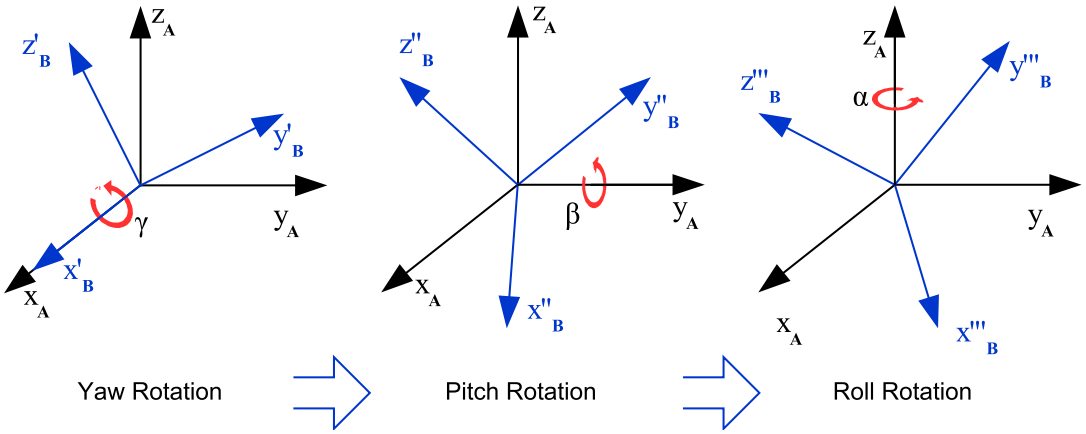

Fig. 2.13: Rappresentazione di una generica rotazione ottenuta tramite rotazioni successive intorno agli assi x(roll), y(yaw) e z(roll)

secondo l'ordine esatto in cui vengono effettuate le rotazioni. Svolgendo il prodotta tra le tre matrici si ottiene la seguente:

$$\begin{aligned}\mathbf{R}(\gamma,\beta,\alpha) &= \begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix} \\ &= \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \end{aligned} \tag{2.3.7}$$

Ora grazie a questa matrice, che riassume le tre rotazioni, è possibile ricavare i valori di α β γ tramite la soluzione di un sistema a nove equazioni in tre incognite, ma essendo sei di queste linearmente dipendenti, il sistema diventa a tre equazioni in tre incognite. Guardando con attenzione la matrice (2.3.7) si può dedurre che sommando il quadrato dei valori r_{11} e r_{21}, grazie alla proprietà di Eulero $(cos(x)^2 + sen(x)^2 = 1)$, si ottiene $cos(\beta)$, da cui si può ricavare β dalla funzione *arcotangente* sfruttanto l'elemento $r_{31} = -sen(\beta)$. Dal valore di $\beta \neq \pm 90°$ si ricavano facilmente anche i valori di α e γ dai valori di r_{11}, r_{21}, r_{32} e r_{33}.

$$\begin{aligned}\beta &= Atan2(\ -r_{31},\ \sqrt{r_{11}^2\ +\ r_{21}^2}\ ) \\ \alpha &= Atan2(\ r_{21}/c\beta,\ r_{11}/c\beta) \\ \gamma &= Atan2(\ r_{32}/c\beta,\ r_{33}/c\beta)\end{aligned} \tag{2.3.8}$$

In caso in cui $\beta = \pm 90°$ le equazioni divergono, per cui tenendo conto che può essere eseguita somma e prodotto tra α e γ; in questi casi è convenzione considerare uno dei due nullo (assumiamo $\alpha = 0$) e ricavare α dall'arcotangente di r_{12} e r_{22}.

Se $\beta = 90°$

$$\begin{aligned}\beta &= 90 \\ \alpha &= 0 \\ \gamma &= Atan2(\ r_{12},\ r_{22})\end{aligned} \tag{2.3.9}$$

Se $\beta = -90°$

$$\beta = -90$$

$$\begin{aligned} \alpha &= 0 \\ \gamma &= -Atan2(\, r_{12},\, r_{22}) \end{aligned} \tag{2.3.10}$$

Ora si hanno le coordinate cartesiane *xyz* e l'orientazione dell'end-effector nel corretto sistema di riferimento del robot A465, pronte quindi per essere inviate al blocco successivo di cinematica inversa; ma prima si analizzano i restanti blocchi realtivi al dispositivo PHANTOM, come: la funzione che compensa la gravità dei giunti e la funzione che scompone il segnale di feedback del sensore secondo l'orientazione dell'end-effector.

2.3.5 Scomposizione Cartesiana del segnale Feedback

Quando l'end-effector del robot A465 è in prossimità di un ostacolo, il sensore ottico genera un segnale di feedback che, prima d'essere condizionato per l'ingresso del blocco Omni-Device, dev'essere scomposto secondo le proiezioni sugli assi cartesiani dell'orientazine dell'end-effector. Tali proiezioni vengono calcolate dal blocco funzione *Orientazione Cartesiana* (vedi Fig. 2.10) che tramite i valori di *Yaw* e *Pitch* (Roll è influente alla proiezione) calcola i versori $\mathbf{u}_x$, $\mathbf{u}_y$ e $\mathbf{u}_z$.

$$\begin{aligned} \mathbf{u}_x &= sign(cos(pitch))sign(sin(yaw))(cos(pitch)sin(yaw))^2 \\ \mathbf{u}_y &= sign(sin(pitch))sin(pitch)^2 \\ \mathbf{u}_z &= sign(cos(pitch))sign(cos(yaw))(cos(pitch)cos(yaw))^2 \end{aligned} \tag{2.3.11}$$

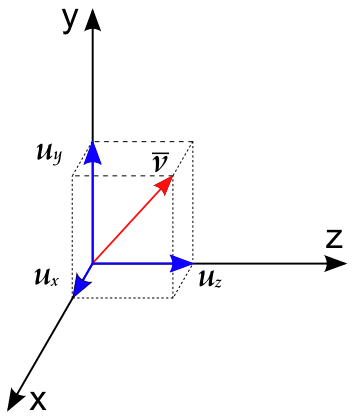

Fig. 2.14: Scomposizione in versori dell'orientazione dell'end-effector

Moltiplicando il segnale del sensore per questi versori si ottiene un vettore di dimensione 3 contenente il feedback correttamente scomposto

$$\mathbf{F} = [\mathbf{f}_x\ \mathbf{f}_y\ \mathbf{f}_z] \tag{2.3.12}$$

Definendo la matrice jacobiana in un blocco funzione, si possono calcolare le tre coppie relativa ai tre motori che il blocco OmniDevice attuerà al dispositivo Phantom, semplicemente moltiplicando la jacobiana trasposta per il vettore dei versori (2.3.12)

$$\mathbf{J} = \begin{bmatrix} -c(\theta_1)(L_2 s(\theta_3) + L_1 c(\theta_2)) & L_1 s(\theta_1) s(\theta_2) & -L_2 s(\theta_1) c(\theta_3) \\ 0 & L_1 c(\theta_2) & L_2 s(\theta_3) \\ -L_2 s(\theta_1) s(\theta_3) - l_1 s(\theta_1) c(\theta_2) & -L_1 c(\theta_1) s(\theta_2) & L_2 c(\theta_1) c(\theta_3) \end{bmatrix} \tag{2.3.13}$$

$$\tau = J^T \mathbf{U} = [\tau_1\ \tau_2\ \tau_3] \tag{2.3.14}$$

2.3.6 Compensazione Gravitá dei Giunti

Parlando di dispositivi aptici, ci si rende subito conto di quando sia fondamentale curare con attenzione il feedback che dovrà poi essere percepito dall'operatore impugnando l'end-effector. In particolare, applicando una forza verticale rivolta verso l'alto, il peso dei giunti influenzia non di poco la sensazione percepita perchè le coppie opposte attenuano quelle generate. É necessario quindi sommare alle coppie di feedback un vettore che vada a compensare le coppiecausate dal peso dei due giunti del Phantom con valori crescenti al crescere della leva. La particolare architettura fisica del dispositivo non ha permesso l'implementazione di una funzione che stimasse le coppie relativamente ai valori angolari dei due link, quindi si è optato per una procedura che tramite un controllore in posizione, memorizzi i valori delle coppie mandate in feedback che sorreggono i giunti in una determinata posizione. Memorizzare i valori delle coppie nel range dei primi 2 link con un intervallo di discretizzazione di qualche millimetro, può risultare un operazione troppo dispendiosa, ma una volta svolta, caricando le matrici con le coppie corrispondenti ai valori angolari in blocco 'Lookup Table' si ottiene un ottima compensazione di gravità.

2.4 Sensore Ottico di Prossimitá

Ora che i dispositivi master e slave sono correttamente interfacciati, per realizzare il sistema aptico si deve leggere dalla scheda MultiQ-3 la tensione inviata dal sensore ottico di prossimità.

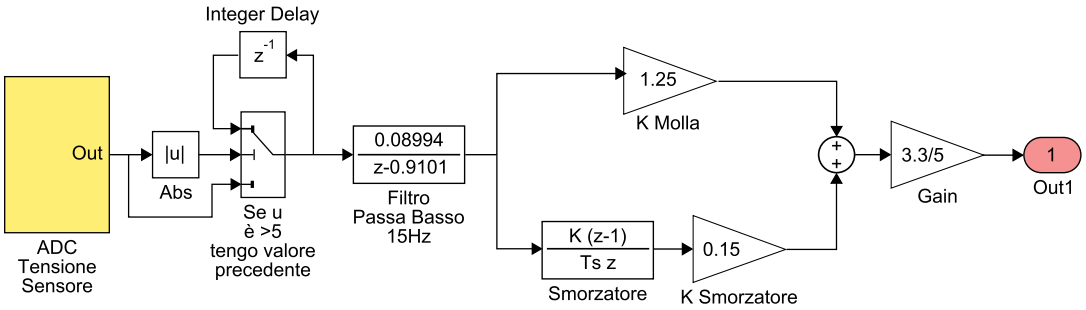

Fig. 2.15: Blocco lettura e gestione Sensore di Prossimità

Tramite la funzione *SFun_Qadc.c* il blocco legge il valore quantizzato in ingresso alla scheda con un range compreso tra $\pm 5V$, tale valore restituito è ancora in valore di registro quindi va moltiplicato per la costante:

$$k = 5/4096$$

quindi si passa da un valore compreso tra ± 4096 (12bit più 1 di segno) ad un valore $\pm 5V$ che sarà il valore utilizzato per il feedback. Facendo delle prove preliminari per valutare il segnale, ci si è accorti della presenza di picchi molto elevati con tempo di salita nell'ordine di campinamento, ma per non rallentare troppo il segnale è stato inserito uno Switch che manda in uscita il valore campionato precedente se quello corrente supera i 5V; poi dal segnale vengono attenuate le componenti rumorose tramite un filtro passa-basso con frequenza di taglio a 15Hz:

$$f(z) = \frac{0.08994}{z - 9101} \tag{2.4.1}$$

I valori sono stati calcolati tramite discretizzazione del filtro in continua:

$$f(s) = \frac{15 \cdot 2\pi}{s + 15 \cdot 2\pi} \tag{2.4.2}$$

tramite l'istruzione Matlab *c2d(f,0.001,'zoh')*.
Il segnale filtrato entra quindi in un sistema Molla-Smorzatore le cui costanti permetto di indurire/ammorbidire e smorzare eventuali oscillazioni del feedback, evitando quindi il propagarsi dell'ondulazioni tipicamente generate dalla molla.

Capitolo 3

Grafici Sperimentali

Realizzato l'interfacciamento, per poter documentare la bontà dei riferimenti generati e dai movimenti effettuati dal dispositivo slave, durante una movimentazione d'esempio compiendo movimenti casuali si sono registrati:

- i riferimenti cartesiani in uscita dal dispositivo PHANTOM, rispettivamente: x, y, z, yaw, pitch, yaw;
- le posizioni cartesiane del robot CSR A465 ottenute tramite cinematica diretta dai valori di encoder dei giunti;
- il vettore di feedback ottenuto moltiplicando il segnale condizionato del sensore con i versori dell'orientazione dell'end-effector, rappresentate le forze cartesiane x, y e z, espresse in Newton, prima della conversione in coppie.

Le nette differenze che si notano tra i segnali, all'inizio e alla fine della simulazione, sono da attribuire alle fasi descritte nella macchina a stati finiti (2.1.6) in cui avviene la fase transitoria di switch tra i riferimenti statici e quelli dinamici del Phantom; infatti il CRS A465 (segnale blu) si trova sempre nella posizione nulla e poi, switchando i riferimenti, insegue i riferimenti del Phantom (segnale rosso) per poi ritornare in posizione nulla al termine.

3.1 Confronto tra Riferimenti PHANTOM e Posizioni CRS

3.1.1 Assi Cartesiani

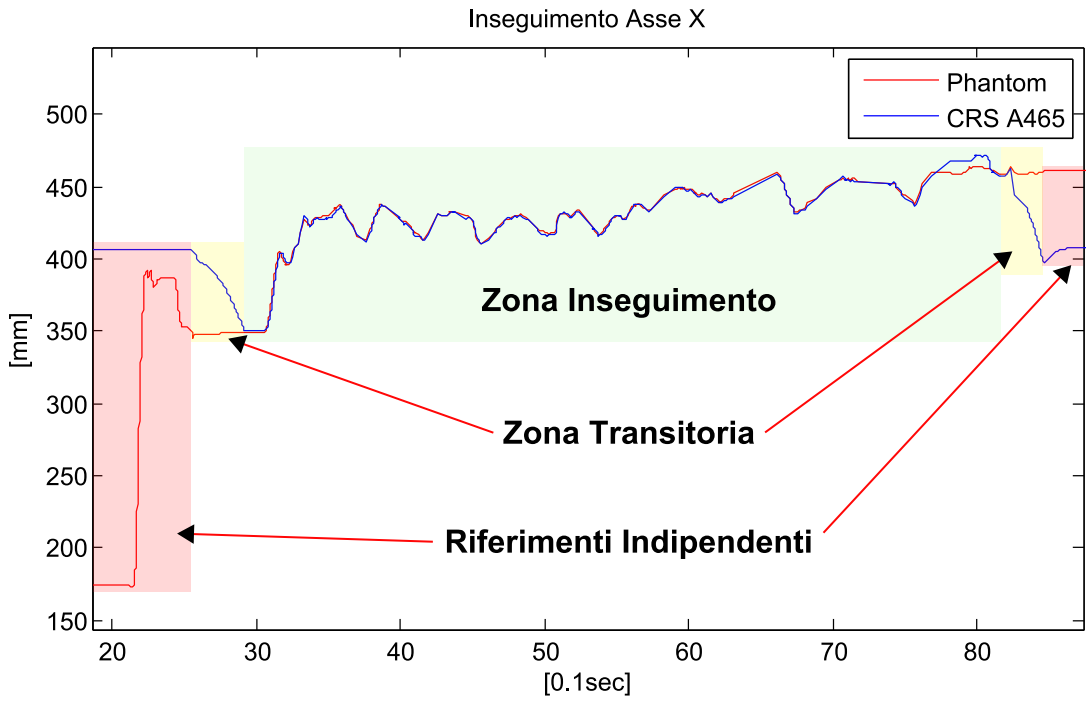

Fig. 3.1: Inseguimento dei dispositivi sull'asse X

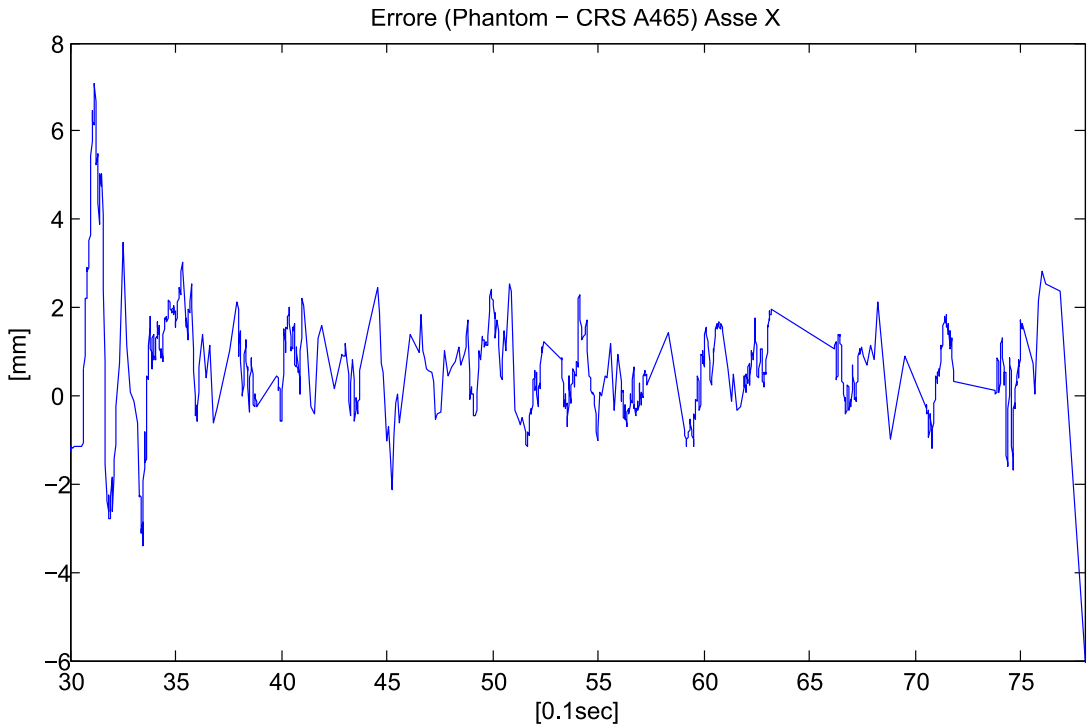

Fig. 3.2: Errore sull'asse X

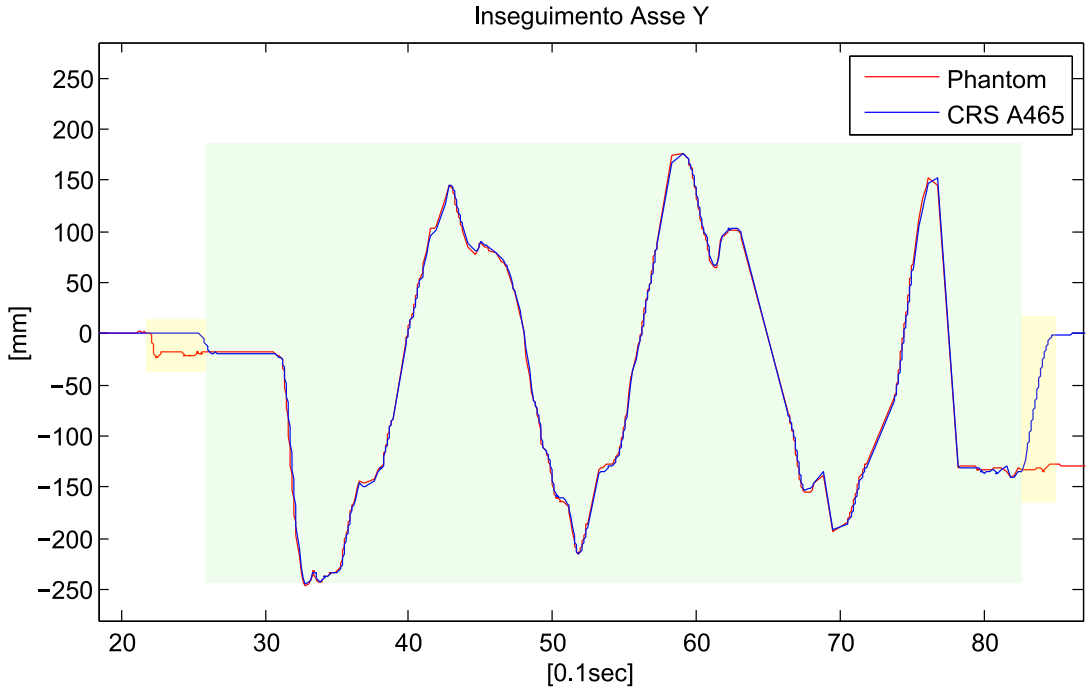

Fig. 3.3: Inseguimento dei dispositivi sull'asse Y

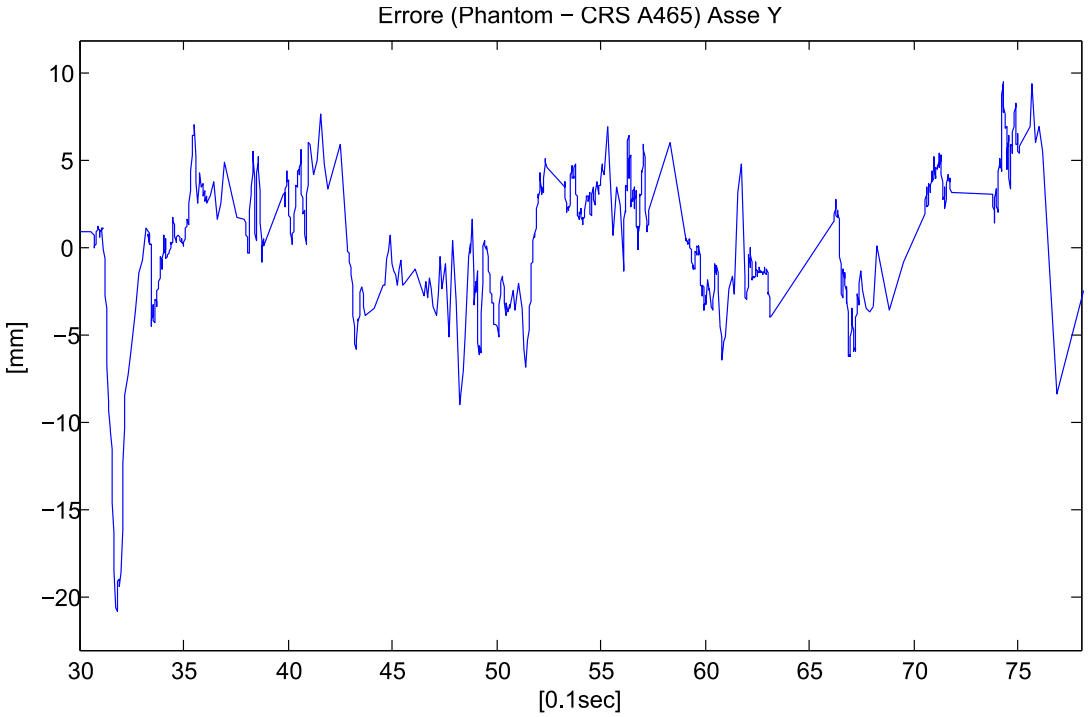

Fig. 3.4: Errore sull'asse Y

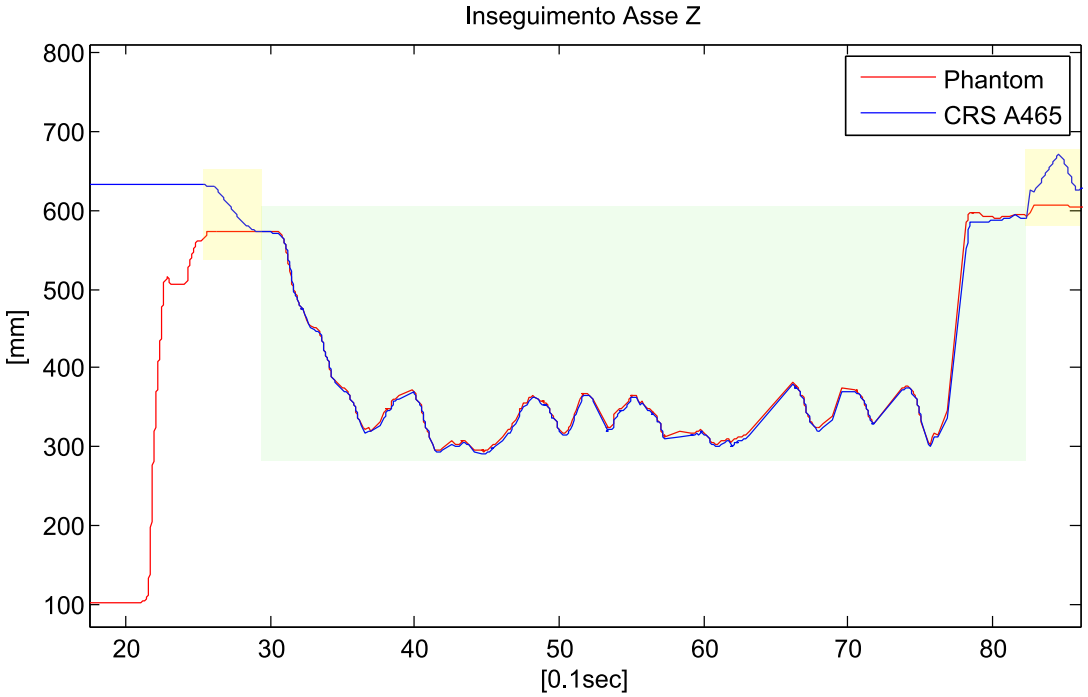

Fig. 3.5: Inseguimento dei dispositivi sull'asse Z

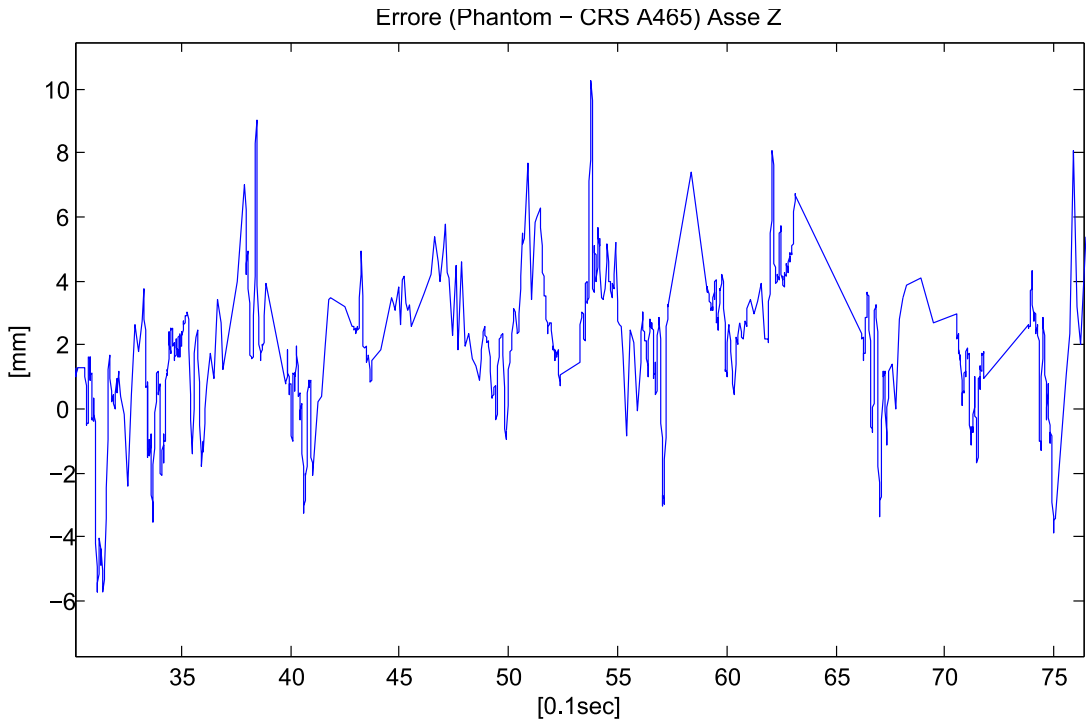

Fig. 3.6: Errore sull'asse Z

3.1.2 Orientazioni

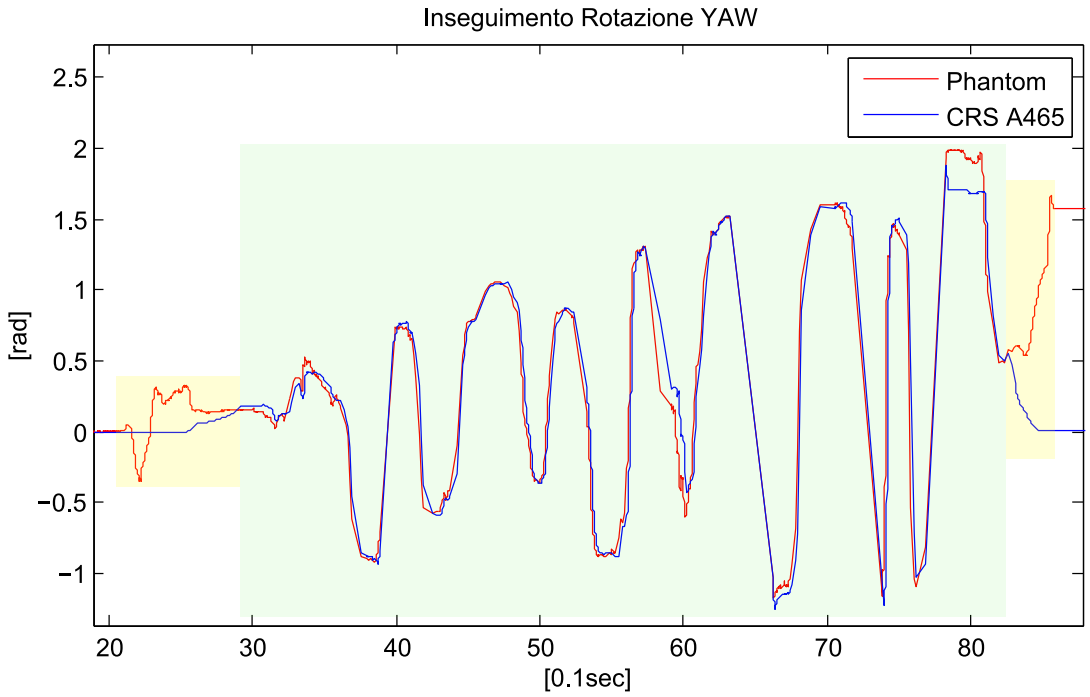

Fig. 3.7: Inseguimento Orientazione YAW

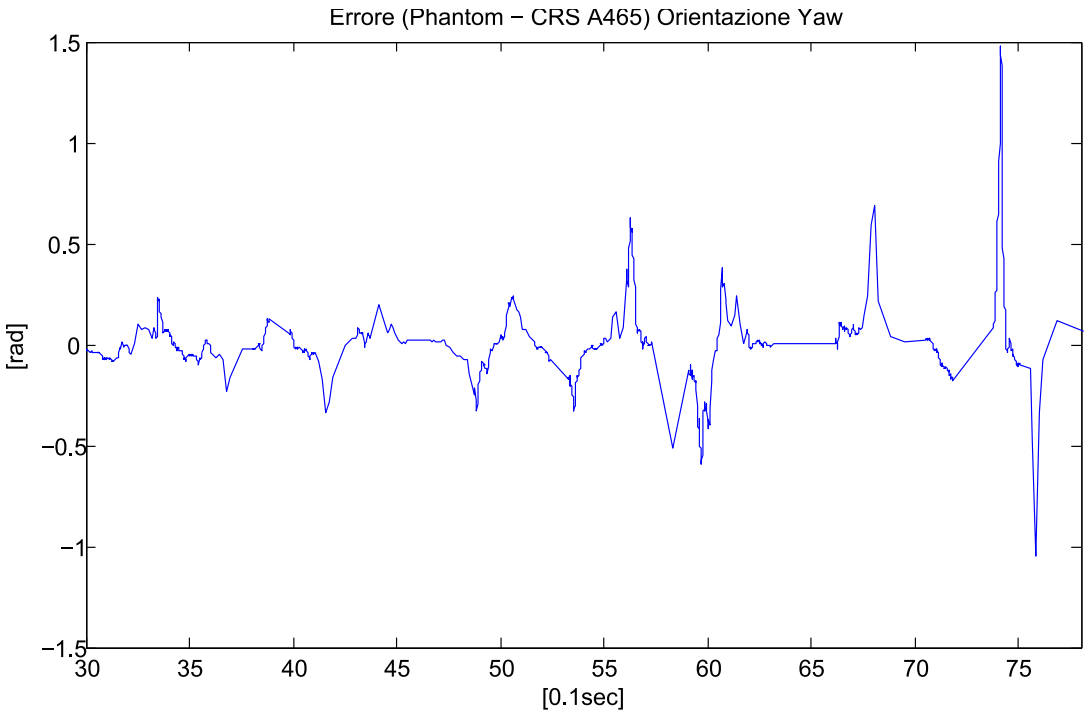

Fig. 3.8: Errore orientazione YAW

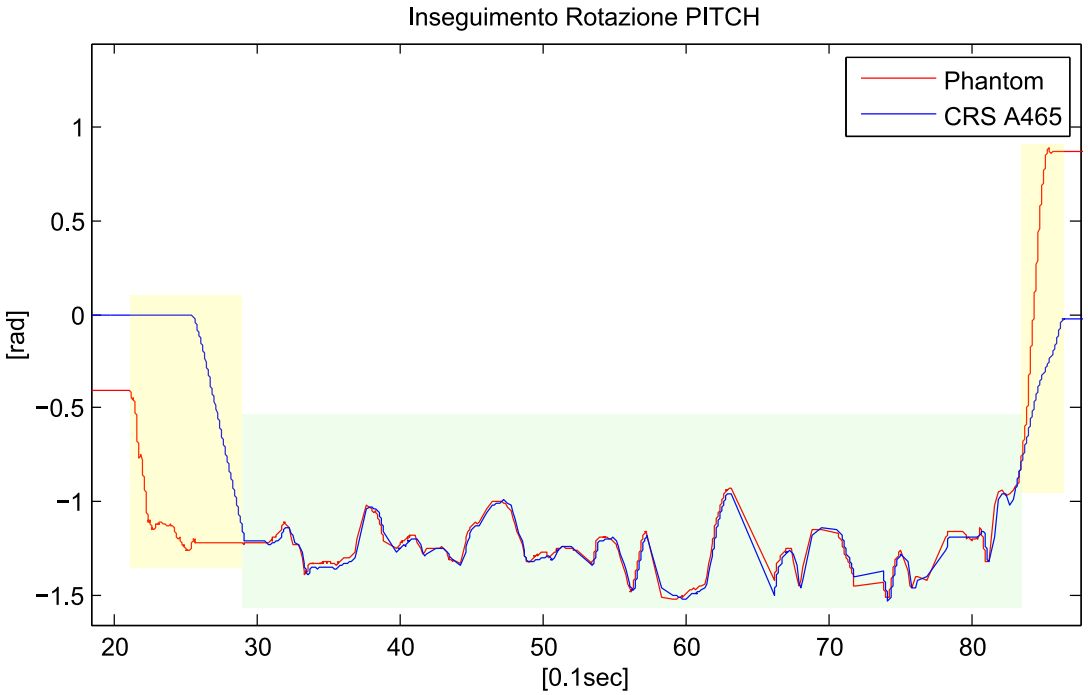

Fig. 3.9: Inseguimento Orientazione PITCH

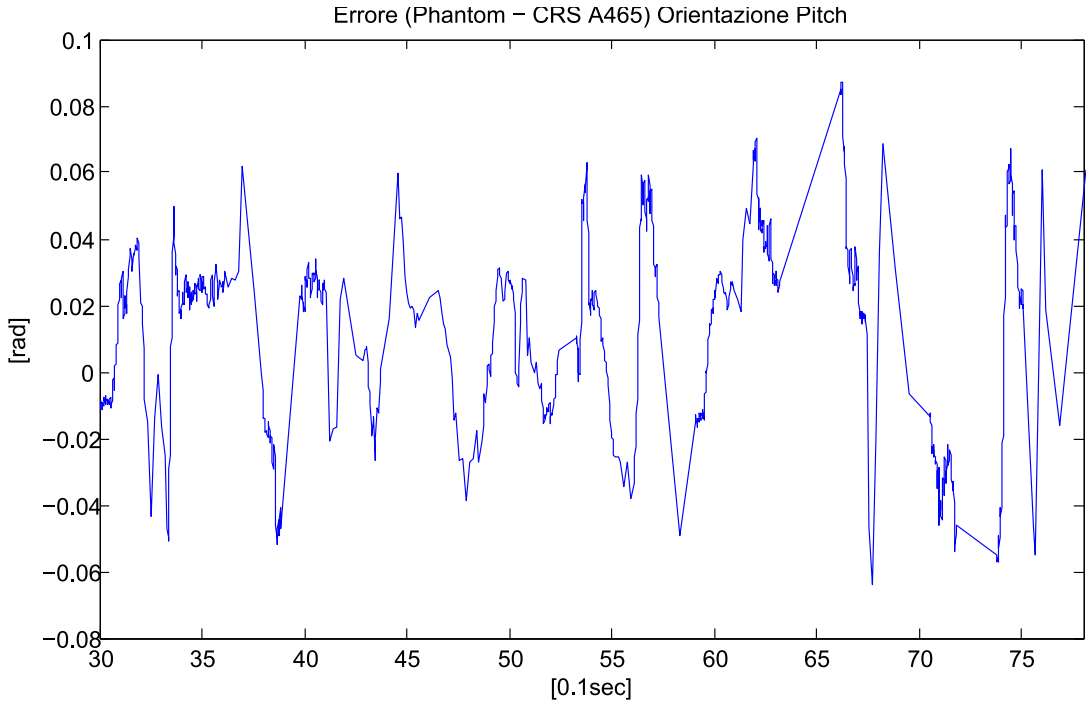

Fig. 3.10: Errore orientazione PITCH

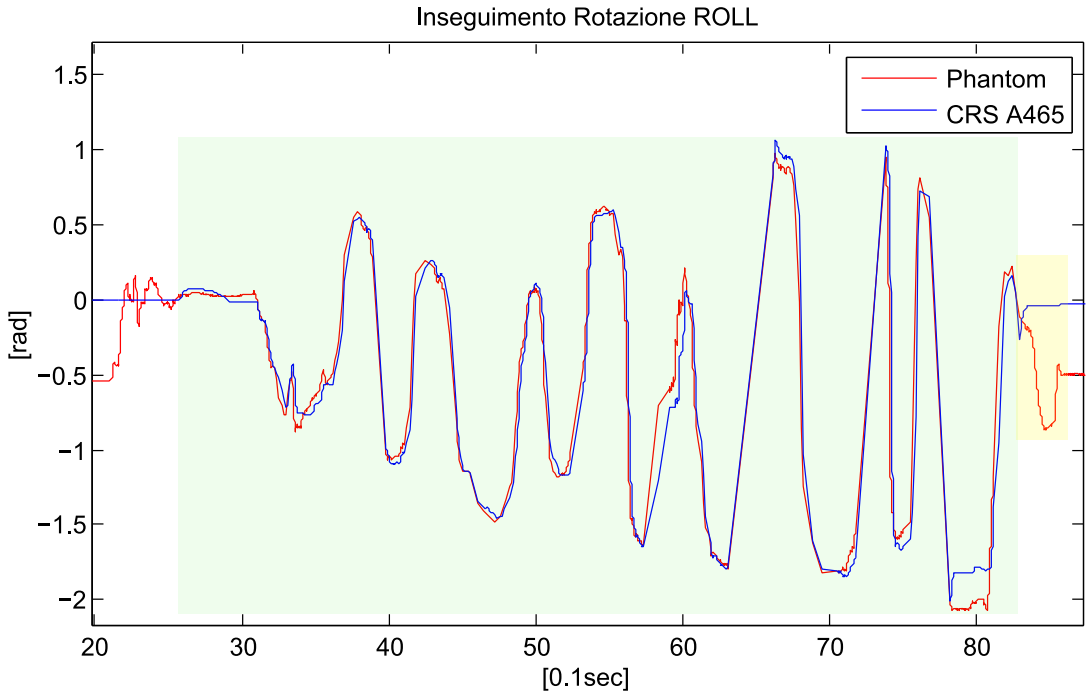

Fig. 3.11: Inseguimento Orientazione ROLL

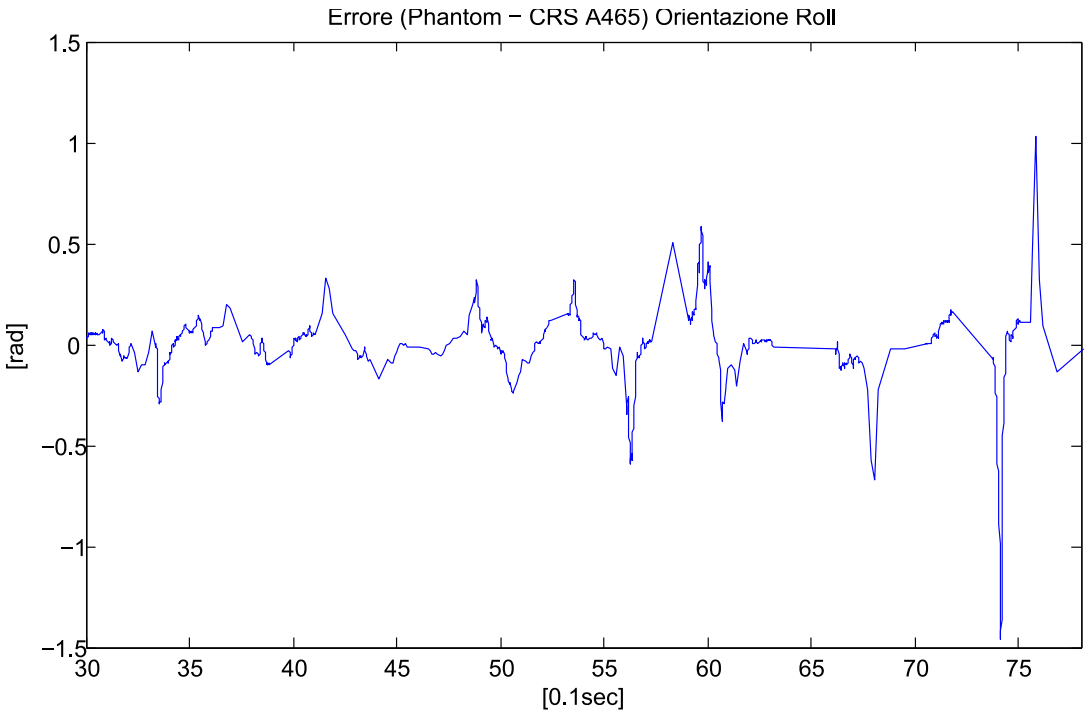

Fig. 3.12: Errore orientazione ROLL

3.2 Feedback Cartesiano

Per documentare il segnale di feedback è stata eseguita una prova in cui il robot CRS si avvicina ad un oggetto con superficie piana appoggiato al tavolo, quando il sensore si avvicina all'ostacolo superando la distanza limite, l'end-effector del Phantom viene fermato dal feedback aptico. In Fig. 3.14 il primo gradino del segnale corrisponde al raggiungimento della soglia, il secondo è dovuto alla volontà di avvicinarsi maggiormente spingendo contro le forze del feedback.

Il segnale andrà poi moltiplicato per i versori d'orientazione dell'end-effector del Phantom per poi essere convertito in coppie tramite la jacobiana trasposta. Il feedback nel grafico si presenta molto rumoroso nonostante il filtraggio con il filtro passa-basso con taglio a 15hz, tuttavia l'ampiezza del disturbo non ha ampiezza apprezzabile dalla sensibilità della mano dell'operatore che percepisce forze decise ma senza vibrazioni.

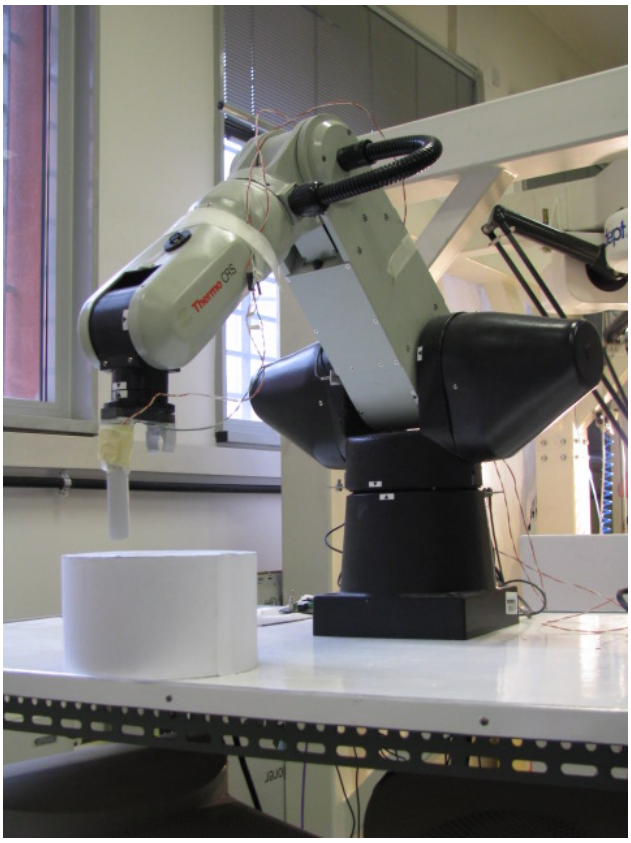

Fig. 3.13: CRS A465 durante la simulazione

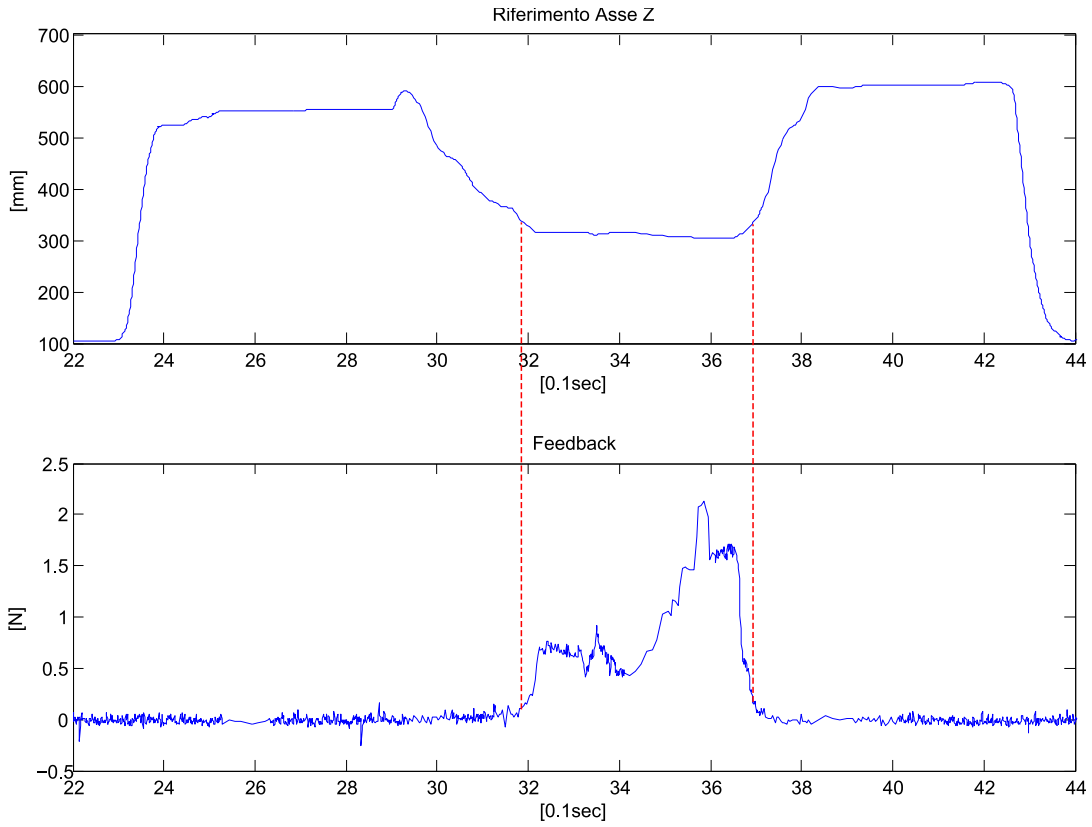

Fig. 3.14: Feedback condizionato prima di esssere scomposto e trasformato in coppie

Conclusioni

La realizzazione di quest'interfaccia aptica ha richiesto una notevole mole di tempo ed energia, in quanto il robot CRS A465, provenendo dal settore industriale e non essendo quindi nato per lavorare in open-architeture ha un'esigua documentazione a riguardo che, unita ad un esperienza prettamente didattica, ha costretto molto spesso a provare soluzioni verificate solo in teoria che ha portato a conseguenze impreviste.
Ulteriore difficoltà si è riscontrata nel dover realizzare un sistema di riferimento comune tra i dispositivi, con le opportune codifiche di posizione ed orientazione, rispettando i diversi limiti dei giunti tra i due dispositivi e dovendo gestire con particolare attenzione le zone che generano singolarità nella cinematica.
Il segnale proveniente dal sensore è risultato molto più disturbato di quanto previsto nonostante il tentativo di filtraggio con svariati valori di taglio, ma si infine è trovato per un giusto compromesso tra errore e velocità.
Nella speranza che questa interfaccia possa essere in un futuro realmente utilizzata in applicazioni di scansione ecografica e in generale in tutte le applicazioni che richiedono dispositivi aptici, ritengo questa esperienza estremamente utile in quanto ho messo in pratica quasi tutte le nozioni apprese nelle varie lezioni di questo corso di laurea in Meccatronica, dovendo spesso rivederle e approfondirle su libri e in rete.

Bibliografia

[1] John J. Craig, " Introduction to robotics mechanics and control", Second edition, Addison Wesley Longman;

[2] Handshake proSENSE^{TM}, "Virtual Touch Toolbox v2.0, User Guide", Handshake VR Inc., 2001-2006;

[3] David Wang, George Lee, Mehrdad Hosseini-Zadeh, James Kunz, "Handshake proSENSE^{TM} Lab Series", Handshake VRInc., 2001-2005;

[4] Quanser, "Quanser / CRS Model 465Open Architecture Robot";

[5] Quanser, "MultiQ-3, Programming manual";

[6] Sensable Tecnologies (http://www.sensable.com);

[7] http://www.robotsdotcom.com/;

Appendice A

Listati Codifiche

A.1 Codifica di Posizione

```
function [z,x,y]= fcn(u)

% definizione lunghezza link con valori del CRS
base=[0,330,0];
braccio1=[0 0 305];
braccio2=[0 -330 0];

% Generazione delle matrici di rotazione
% rot base -> rotazione risp l'asse y
% rot primo link -> rot risp l'asse x
% rot secondo link -> rot risp l'asse x
rot1=[cos(u(1)) 0 sin(u(1));0 1 0;-sin(u(1)) 0 cos(u(1))];
rot2=[1 0 0; 0 cos(u(2)) -sin(u(2)); 0 sin(u(2)) cos(u(2))];
rot3=[1 0 0; 0 cos(u(3)) -sin(u(3)); 0 sin(u(3)) cos(u(3))];

%Rotazioni
bra1=braccio1*rot2;   % rotazione del primo link
bra2=braccio2*rot3;   % rotazione del secondo link
vet=base+bra1+bra2;   % somma dei 3 link orientati
vet1=vet*rot1;        % rotazione della base

% assegnazione delle uscite
x=vet1(1);
y=vet1(2);
z=vet1(3);
```

A.2 Codifica di Orientazione

```
function [yaw, pitch, roll] = fcn(u)

% Da vettore di Axis Angle a variabili kx ky kz e theta
kx = u(3); ky = u(1); kz = u(2); theta = u(4);

% Definizione matrice di rotazione in funzione di kx ky kz e theta,
% permette la conversione da axis angles a matrice di rotazione
ctheta = cos(theta);
stheta = sin(theta);
vtheta = 1-cos(theta);

Rk = [kx*kx*vtheta+ctheta kx*ky*vtheta-kz*stheta kx*kz*vtheta+ky*stheta;
      kx*ky*vtheta+kz*stheta ky*ky*vtheta+ctheta ky*kz*vtheta-kx*stheta;
      kx*kz*vtheta-ky*stheta ky*kz*vtheta+kx*stheta kz*kz*vtheta+ctheta];

% Conversione da matrice di rotazione a roll-pitch-yaw
    pitch_x = sqrt(Rk(1,1)^2 + Rk(2,1)^2);
    pitch_y = -Rk(3,1);
    pitch = atan2(pitch_y,pitch_x);

    yaw_x = Rk(1,1)/cos(pitch);
    yaw_y = Rk(2,1)/cos(pitch);
    yaw = atan2(yaw_y,yaw_x);
```

```
    roll_x =Rk(3,3)/cos(pitch);
    roll_y = Rk(3,2)/cos(pitch);
    roll = atan2(roll_y,roll_x);

if (pitch == pi/2)
    yaw = 0;
    roll = atan2(Rk(1,2),Rk(2,2));
end
if (pitch == -pi/2)
    yaw = 0;
    roll = -atan2(Rk(1,2),Rk(2,2));
end
```

Appendice B

Proiezioni Cartesiane Orientazione End-Effector

```
function [Ux,Uy,Uz] = fcn(u)
% This block supports an embeddable subset of the MATLAB language.
% See the help menu for details.
yaw=u(1);
pitch=u(2);

Ux= sign(cos(pitch))* sign(sin(yaw))*(cos(pitch)*sin(yaw))^2;
Uy= sign(sin(pitch))* sin(pitch)^2;
Uz= sign(cos(pitch))*sign(cos(yaw))*(cos(pitch)*cos(yaw))^2;
```

Appendice C

Funzione: Errore in Modulo

```
function y = fcn(Crs, Phantom)

u11 = Phantom(1);
u12 = Phantom(2);
u13 = Phantom(3);
u14 = Phantom(4);
u15 = Phantom(5);
u16 = Phantom(6);
u21 = Crs(1);
u22 = Crs(2);
u23 = Crs(3);
u24 = Crs(4);
u25 = Crs(5);
u26 = Crs(6);
a = 0;

e1 = (u11-u21)^2;
e2 = (u12-u22)^2;
e3 = (u13-u23)^2;
e4 = (u14-u24)^2;
e5 = (u15-u25)^2;
e6 = (u16-u26)^2;

 err1 = sqrt( e1 + e2 + e3 );
 err2 = sqrt( e4 + e5 + e6 );

if( (err1 < 1) && (err2 < 5) )
    a = 1;
else
    a = 0;
end;

y = a;
```

Appendice D

Funzione: Macchina a Stati Finiti

```
function [NextS,P]= fcn(off,e,u1,u2,u3)

Aggancio = e;
Pulsante = u1;
currstate = u2;
nextstate = 0;
a = u3;
Sw = off;

if( Sw == 1 )
    nextstate = 0;
    currstate = 0;
end;

%finchè non rilascio il pulsante, "a" resta =1 appena rilascio a=0
if ( Pulsante == 0 ) && ( a == 1 )
    a = 0;
end;

switch currstate,

    case 2,
        % CRS insegue Phantom con Sigmoidi Veloci
                                if ( Pulsante == 1 ) && ( a == 0 )
            nextstate = 0;
            a = 1;
        else
            nextstate = 2;
        end; % end if

    case 1,
        % CRS insegue Phantom con Sigmoidi Lente,
        % finchè non aggancia la posizione del Phantom

        if  ( Aggancio == 1 )
            nextstate = 2;
            a = 1;
        else
            nextstate = 1;

        end; % end if

    case 0, % CRS in READY
        if ( Pulsante == 1 ) && ( a == 0 )
            nextstate = 1;
            a = 1;
        else
            nextstate = 0;
        end; % end if

end; % end switch

NextS = nextstate;
P = a;
```

Appendice E

Funzione: A465_WorldToJoint.c

```
/*
 * File : A465_WorldToJoint.c
 */

#define S_FUNCTION_NAME  A465_WorldToJoint
#define S_FUNCTION_LEVEL 2

/*
 * Need to include simstruc.h for the definition of the SimStruct and
 * its associated macro definitions.
 */
#include <string.h>
#include <math.h>
#include "tmwtypes.h"
#include "A465.h"

#if defined(__cplusplus)
extern "C"
{
#endif

#include "simstruc.h"

/*====================*
 * Build checking *
 *====================*/

/*************************************************

   S-Function Parameters
   ---------------------

*************************************************/

enum
{
    NUM_PARAMS
};

/*============================================================*
 * Macros to access the S-function parameter values *
 *============================================================*/

enum
{
    WORLD_COORDS,
    // World coordinates to convert to joint angles
    STANCE,
    // Desired stance (second input to match JointToWorld outputs)
    OLD_JOINT_ANGLES,
    // Old joint angles (third input to allow pretty feedback loops)

    NUMBER_OF_INPUTS
};

enum
```

```
{
    JOINT_ANGLES,
    ERROR_OUTPUT,

    NUMBER_OF_OUTPUTS
};

enum StanceElements
{
        REACH,
        ELBOW,
        WRIST,

        NUMBER_OF_STANCES
};

/*====================*
 * S-function methods *
 *====================*/


/* Function: mdlInitializeSizes ========================================
 * Abstract:
 *    Call mdlCheckParameters to verify that the parameters are okay,
 *    then setup sizes of the various vectors.
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, NUM_PARAMS);
    if (ssGetNumSFcnParams(S) == ssGetSFcnParamsCount(S))
    {
        ssSetNumContStates( S, 0);
        ssSetNumDiscStates( S, 0);
        ssSetNumSampleTimes(S, 0);

        // Configure the inputs
        if (!ssSetNumInputPorts(S, NUMBER_OF_INPUTS))
            return;

        ssSetInputPortWidth(              S, WORLD_COORDS, 6);
        ssSetInputPortDirectFeedThrough( S, WORLD_COORDS, 1);
        //ssSetInputPortRequiredContiguous(S, WORLD_COORDS);
        // Release 12 only

        ssSetInputPortWidth( S, STANCE, NUMBER_OF_STANCES);
        ssSetInputPortDirectFeedThrough( S, STANCE, 1);
        //ssSetInputPortRequiredContiguous(S, STANCE);
        // Release 12 only

        ssSetInputPortWidth(  S, OLD_JOINT_ANGLES, NUM_AXES);
        ssSetInputPortDirectFeedThrough( S, OLD_JOINT_ANGLES, 1);
        //ssSetInputPortRequiredContiguous(S, OLD_JOINT_ANGLES);
        // Release 12 only

        // Configure the outputs
        if (!ssSetNumOutputPorts(S, NUMBER_OF_OUTPUTS))
            return;
        ssSetOutputPortWidth(S, JOINT_ANGLES, NUM_AXES);
        // joint angles
        ssSetOutputPortWidth(S, ERROR_OUTPUT, 2);
        // any error produced

        ssSetNumIWork(  S, 0);
```

```
        ssSetNumRWork(  S, 0);
        ssSetNumPWork(  S, 0);
        ssSetNumModes(  S, 0);
        ssSetNumNonsampledZCs( S, 0);
        ssSetOptions( S, SS_OPTION_EXCEPTION_FREE_CODE);
    }

} /* end mdlInitializeSizes */

/* Function: mdlInitializeSampleTimes ==========================
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
} /* end mdlInitializeSampleTimes */

/*
*
*       This function multiplies a transform matrix '_A' by a
*       transform matrix '_B' to give a transform matrix '_C'.
*       All transform matricies are expressed as twelve element
*       single dimensional arrays.
*       Note that transform matricies are usually 4x4; however
*       since the last row of the matrix is always [0 0 0 1] the sixteen
*       elements of the transform matrix can be compacted to twelve.
*/
static void matrix_mult(const real_T _A[XFORM_SIZE],
                        const real_T _B[XFORM_SIZE],
                        real_T _C[XFORM_SIZE])
{

  _C[r11]  = _A[r11]*_B[r11] + _A[r12]*_B[r21] + _A[r13]*_B[r31];
  _C[r21]  = _A[r21]*_B[r11] + _A[r22]*_B[r21] + _A[r23]*_B[r31];
  _C[r31]  = _A[r31]*_B[r11] + _A[r32]*_B[r21] + _A[r33]*_B[r31];

  _C[r12]  = _A[r11]*_B[r12] + _A[r12]*_B[r22] + _A[r13]*_B[r32];
  _C[r22]  = _A[r21]*_B[r12] + _A[r22]*_B[r22] + _A[r23]*_B[r32];
  _C[r32]  = _A[r31]*_B[r12] + _A[r32]*_B[r22] + _A[r33]*_B[r32];

  _C[r13]  = _A[r11]*_B[r13] + _A[r12]*_B[r23] + _A[r13]*_B[r33];
  _C[r23]  = _A[r21]*_B[r13] + _A[r22]*_B[r23] + _A[r23]*_B[r33];
  _C[r33]  = _A[r31]*_B[r13] + _A[r32]*_B[r23] + _A[r33]*_B[r33];

  _C[Px]  = _A[r11]*_B[Px] + _A[r12]*_B[Py] + _A[r13]*_B[Pz] + _A[Px];
  _C[Py]  = _A[r21]*_B[Px] + _A[r22]*_B[Py] + _A[r23]*_B[Pz] + _A[Py];
  _C[Pz]  = _A[r31]*_B[Px] + _A[r32]*_B[Py] + _A[r33]*_B[Pz] + _A[Pz];

    return;

}

/*
 * Function that maps a joint angle into the range of +/- PI
 * Note that this function does NOT yield the same results
 * as fmod(in_angle, PI)!
 */
static real_T map_to_PI( real_T in_angle )
{
    real_T newangle;

    newangle = in_angle;

    while( newangle > PI)
        newangle -= PIX2;
```

```
    while( newangle <= -PI)
        newangle += PIX2;

    return(newangle);
}

/* This routine maps the a465 arm's joint angles from physical
 * joint limit space (user space) to kinematics space
*/
static void A465_map_user_to_kin_space(
                const real_T user_angle[NUM_AXES],
                real_T kin_angle[NUM_AXES])
{
    int_T i;
    real_T disp;

/* most joints are the same in joint space and in kin space */
/* however we must ensure that they're mapped between +/- PI */
/* the exceptions are joints 2 and 3  */
    for(i=0;i<6;i++)
    {
        if(i==J2)
            disp = PID2;
        else if (i==J3)
            disp = -PID2;
        else
            disp = 0;

        kin_angle[i]=map_to_PI(user_angle[i] + disp);

    }

    /* before we're finished we need to ensure that our range is
     * is correct for joints 2 and 3
    */
    if( kin_angle[J2] <= -PID2 )
            kin_angle[J2] += PIX2;
    if( kin_angle[J3] >   PID2 )
            kin_angle[J3] -= PIX2;

    return;
}

/*
*   Converts a position and orientation array expressed in x, y, z, yaw,
*   pitch, roll representation to a homogeneous transform matrix.  See
*   Craig pp45-50 for more information.
*/
static void convert_worldcoords_to_homotransform(
                        const real_T worldcoords[6],
                        real_T homotransform[XFORM_SIZE])
{
         real_T cy,sy,cp,sp,cr,sr;

         cy = cos(worldcoords[YAW]);
         sy = sin(worldcoords[YAW]);

         cp = cos(worldcoords[PITCH]);
         sp = sin(worldcoords[PITCH]);

         cr = cos(worldcoords[ROLL]);
         sr = sin(worldcoords[ROLL]);
```

```
        homotransform[r11] = cy*cp;
        homotransform[r21] = sy*cp;
        homotransform[r31] = -sp;

        homotransform[r12] = cy*sp*sr - sy*cr;
        homotransform[r22] = sy*sp*sr + cy*cr;
        homotransform[r32] = cp*sr;

        homotransform[r13] = cy*sp*cr + sy*sr;
        homotransform[r23] = sy*sp*cr - cy*sr;
        homotransform[r33] = cp*cr;

        homotransform[Px] = worldcoords[XDIM];
        homotransform[Py] = worldcoords[YDIM];
        homotransform[Pz] = worldcoords[ZDIM];
}

/* A465_inverse_kinematics:
 * Computes the kinematic joint angles 'jointsoln' from
 * the basic arm transform 'transform'.
 *
 *Note:'old_joints' is an array representing the robots current kinematic
 *joint angles.  It is used when a singularity is encountered in joint
 *1 (wrist directly above origin) or joint 5 (when J5 is exactly 0).
 *In each case, we appeal to the previous joint angle for a reasonable
 *approximation of where the next joint angle should be.  Without an
 *approach like this the arm may move erratically and dangerously.
 */
static int_T A465_inverse_kinematics(
        real_T transform[XFORM_SIZE],
        Joint_Solution jointsoln[NUM_SOLNS],
        real_T old_joints[NUM_AXES])
{
    real_T linkvar1, linkvar2;
    real_T j1soln1, j1soln2;
    real_T s1, s1_F, s1_B, c1, c1_F, c1_B;
    real_T j3tmp1, j3tmp2, j3tmp3, j3tmp4;
    real_T j3soln1, j3soln2;
    real_T s3, c3_BUFD, c3_FUBD;
    real_T threespace_dist_sqr;
    real_T inv_3space_dist_sqr;
    real_T baseplane_dist;
    real_T j2tmp1_BUFD, j2tmp1_FUBD;
    real_T j2tmp1, j2tmp2;
    real_T s23, c23;
    real_T basedist_26;
    real_T j2soln;
    real_T j5_singularity;
    real_T s5, c5, s4, c4, s6, c6;
    real_T j4tmp1, j4tmp2, j4tmp3;
    real_T j4soln1, j4soln2;
    real_T j5soln1, j5soln2;
    real_T j6tmp1, j6tmp2;
    real_T j6soln1, j6soln2;
    int_T index;


    /* Joint 1 */
#ifndef TESTING
  if(fabs(transform[Px])<EPSILON && fabs(transform[Py])<EPSILON)
    {
/* we're in the joint 1 singularity directly above the origin */
        j1soln1 = old_joints[J1];
```

```
            j1soln2 = j1soln1 + PI;
    }
    else
#endif
    {
            j1soln1 = atan2(transform[Py],transform[Px]);
            j1soln2 = j1soln1 + PI;
    }

    /* reach forward solutions */
    jointsoln[FUN].joint[J1] = j1soln1;
    jointsoln[FUF].joint[J1] = j1soln1;
    jointsoln[FDN].joint[J1] = j1soln1;
    jointsoln[FDF].joint[J1] = j1soln1;

    /* reach backward solutions */
    jointsoln[BUN].joint[J1] = j1soln2;
    jointsoln[BUF].joint[J1] = j1soln2;
    jointsoln[BDN].joint[J1] = j1soln2;
    jointsoln[BDF].joint[J1] = j1soln2;

    /* set up some variables for later */

    s1_F=sin(j1soln1); /* forward */
    c1_F=cos(j1soln1);
    s1_B = -s1_F;        /* backward */
    c1_B = -c1_F;



    /* Joint 3 */

    linkvar1     = A2*A2+D4*D4;
    linkvar2     = 4.0*A2*A2*D4*D4;

    baseplane_dist = c1_F*transform[Px] + s1_F*transform[Py];
    /* was f11p  */
     threespace_dist_sqr =
        baseplane_dist*baseplane_dist+transform[Pz]*transform[Pz];
    j3tmp1 = linkvar1 - threespace_dist_sqr;
    j3tmp2 = linkvar2 - j3tmp1*j3tmp1 ;

    /* ensure we're within reach */
    if( j3tmp2 < -linkvar2*EPSILON )
    /* EPSILON defined in header file */
        return( INVKIN_OUTOFREACH );
    else if( j3tmp2 < 0.0 )
         j3tmp2 = 0.0;

    /* now determine the Joint 3 angle */
    j3tmp3 = sqrt( j3tmp2 );
    j3tmp4 = atan2( -j3tmp3,j3tmp1);
    j3soln1 = PID2 + j3tmp4;
    j3soln2 = PID2 - j3tmp4;

    /* backward/up and forward/down solutions */
    jointsoln[BUN].joint[J3] = j3soln1;
    jointsoln[BUF].joint[J3] = j3soln1;
    jointsoln[FDN].joint[J3] = j3soln1;
    jointsoln[FDF].joint[J3] = j3soln1;

    /* foward/up and backward/down solutions */
    jointsoln[FUN].joint[J3] = j3soln2;
```

```
    jointsoln[FUF].joint[J3] = j3soln2;
    jointsoln[BDN].joint[J3] = j3soln2;
    jointsoln[BDF].joint[J3] = j3soln2;


    /* now prepare some variables for further stages. */
    s3      =sin(j3soln1);  /* was tmp2 */
    c3_BUFD =cos(j3soln1);  /* was tmp1 */
    c3_FUBD = -c3_BUFD;     /* same as cos(j3soln2) */

 /* pre-calculating the inverse here saves cpu cycles later on */
    inv_3space_dist_sqr = 1/threespace_dist_sqr;


    /* Joint 2 --> Joint 6 */

    /* set up joint 2 variables */
    j2tmp1_BUFD = A2*c3_BUFD;
    j2tmp1_FUBD = A2*c3_FUBD;

    j2tmp2 = D4 - A2*s3;


/*We have two solutions for each of J1 and J3 (corresponding
 *to reach forward/back and elbow up/down) so there are now a
 *total of 4 stance configurations that we can attain:
 *FU, BU, FD, BD.
 *We must find separate solutions for each configuration
 *for all the remaining joints.
 *To simplify this process we will loop through the following
 *calculations 4 times, once for each of the configurations.
 */


/* Because of the way the solution array is organized, we must
 * index through with a step size of two to hit each of the FU, BU
 * FD, and BD solutions.  See A465.h for more info    */
    for (index=FUN;index<=6;index+=2)
    {
        switch(index)
        {
            case(FUN):
            {
                c1          = c1_F;
                s1          = s1_F;
                basedist_26 = baseplane_dist;
                j2tmp1      = j2tmp1_FUBD;
                break;
            }
            case(FDN):
            {
                c1          = c1_F;
                s1          = s1_F;
                basedist_26 = baseplane_dist;
                j2tmp1      = j2tmp1_BUFD;
                break;
            }
            case(BUN):
            {
                c1          = c1_B;
                s1          = s1_B;
                basedist_26 = -baseplane_dist;
                j2tmp1      = j2tmp1_BUFD;
```

```
                break;
            }
            case(BDN):
            {
                c1          = c1_B;
                s1          = s1_B;
                basedist_26 = -baseplane_dist;
                j2tmp1      = j2tmp1_FUBD;
                break;
            }
            default:
                return CASE_ERROR;
                break;
        }


        /* joint 2 */

  s23 = (j2tmp1*transform[Pz]-j2tmp2*basedist_26)*inv_3space_dist_sqr;
  c23 = ((j2tmp2*transform[Pz])+(j2tmp1*basedist_26))*inv_3space_dist_sqr;
  j2soln = atan2(s23, c23) - jointsoln[index].joint[J3];

        jointsoln[index].joint[J2] = j2soln;
        /* noflip solution */
        jointsoln[index+1].joint[J2] = j2soln;
        /* flip solution */


        /* Joint 4 */


        j4tmp1 = -s1*transform[r13] + c1*transform[r23];
        j4tmp2 =  c1*transform[r13] + s1*transform[r23];
        j4tmp3 =  c23*j4tmp2 + s23*transform[r33];


/* first check if we're in the J5 singularity (i.e. J5 ~= 0)  */
#ifndef TESTING
    if( (fabs(j4tmp1) < EPSILON) && (fabs(j4tmp3) < EPSILON) )
      {
/* we're in the singularity -- set flag and set J5 to exactly 0 */
            j5_singularity = 1;
            jointsoln[index].joint[J5]      = 0.0;
        /* noflip solution */
            jointsoln[index+1].joint[J5]    = 0.0;
            /* flip solution */

        s5 = 0.0;
        c5 = 1.0;

/* set joint 4 to it's old value since we have no way of knowing
 * where else it should be (position is ambiguous)
 */
  jointsoln[index].joint[J4] = old_joints[J4];
  /* noflip solution */
  jointsoln[index+1].joint[J4] = old_joints[J4];
  /* flip solution */

            s4 = sin(old_joints[J4]);
            c4 = cos(old_joints[J4]);


    }
```

```
    else
#endif
    {
            /* we're not singular in jt 5 */
            j5_singularity = 0;
            j4soln1 = atan2(j4tmp1,j4tmp3);
       j4soln2 = j4soln1 + PI;

    jointsoln[index+1].joint[J4]    = j4soln1;
    /* wrist is flipped   */
    jointsoln[index].joint[J4]      = j4soln2;
    /* wrist is not flipped */

            s4 = sin(j4soln2);
            c4 = cos(j4soln2);
     }

    /* Joint 5 */

 /* now, if we're not in a singularity, we must still solve for J5 */
    if( !j5_singularity )
    {
        s5 = -c4*j4tmp3 - s4*j4tmp1;
        c5 = -s23*j4tmp2 + c23*transform[r33];
            j5soln1 = atan2(s5,c5);
            j5soln2 = -j5soln1;
        jointsoln[index].joint[J5]   = j5soln1;
        /* noflip solution */
        jointsoln[index+1].joint[J5] = j5soln2;
        /* flip solution */
    }

    /* Joint 6 */

    j6tmp1 = c1*transform[r12] + s1*transform[r22];
        j6tmp2 = -s1*transform[r12] + c1*transform[r22];

    s6= -c5*(c4*(c23*j6tmp1+s23*transform[r32]) + s4*j6tmp2)
            + s5*(s23*j6tmp1-c23*transform[r32]);

    c6= -s4*(c23*j6tmp1 + s23*transform[r32]) + c4*j6tmp2;

        j6soln1 = atan2(s6, c6);
        j6soln2 = j6soln1 + PI;

        jointsoln[index].joint[J6]= j6soln1;
        /* wrist not flipped */

       if( j5_singularity )
/* in singularity -- flip solution same as noflip solution */
            jointsoln[index+1].joint[J6] = j6soln1;
        else
            jointsoln[index+1].joint[J6] = j6soln2;
            /* wrist flipped */



    } /* end of Joints 2-6 for loop*/

    return(OK);

}   /* end of A465_inverse_kinematics() */
```

```
/* cost is based solely on the sum of the squares of
 * the distance (in radians, mapped into PI) between
 * the new j1..j6 and the old j1..j6. The closer they
 * are together, the lower the cost
*/
static real_T A465_calc_cost_function(
                      const real_T new_joints[NUM_AXES],
                      const real_T old_joints[NUM_AXES])
{
    real_T cost;
    real_T diff;
    int i;

    cost = 0.0;
    for(i=0;i<NUM_AXES;i++)
    {
    diff = map_to_PI( new_joints[i]-old_joints[i] );
    cost += (diff*diff);
    }
    return(cost);
}

/* This routine maps the A465 joint angles from kinematics space
 * to joint limit space (user space).                          */
static void A465_map_kin_to_user_space(
                const real_T kin_angle[NUM_AXES],
                real_T user_angle[NUM_AXES])
{
    int_T i;
    real_T offset;
    real_T temp;


    /* most joints are the same in joint space and in kin space
     * however we must ensure that they're mapped between +/- PI
     * The exceptions are joints 2 and 3 which are shifted by 90 degrees
     * from the kinematic model.
    */
    for(i=0;i<6;i++)
    {
        if(i==J2)
            offset = -PID2;
        else if(i==J3)
            offset = PID2;
        else
            offset = 0;

        temp = map_to_PI(kin_angle[i] + offset);

        user_angle[i] = temp;
    }

    return;
}

/* Checks each joint to ensure that all are within joint limits.
 * Returns OK if we're within the joint limits, or an error code
 * indicating which joint is out of range.
 * Note:  based on user joint angles, NOT kinematic joint angles.
*/
static int_T A465_user_joint_check(const real_T user_joints[NUM_AXES])
{
    int_T i;
```

```
    for (i=0;i<NUM_AXES;i++)
    {
    /* check joint angle against limits */
  if((user_joints[i]>A465_posjointlim[i])||
                          (A465_negjointlim[i]>user_joints[i]))
    {
        /* joint out of range --send this back as an error */
        return( -(JOINT_OUT_OF_RANGE | ((i + 1) << 4 ) ) );
    }
    }

    return(OK);
}

/* Identical to A465_user_joint_check(), however takes kinematic
 * joint angles as arguments
*/
static int_T A465_kin_joint_check(const real_T kin_joints[NUM_AXES])
{
    real_T u_joints[NUM_AXES];

    A465_map_kin_to_user_space(kin_joints, u_joints);

    return(A465_user_joint_check(u_joints));


}

/* Function that chooses the best solution
 *(of the possibilities generated by the inverse kinematics routine)
 * based on previous joints and desired stance.
 * returns the best solution (if any)
 * or NO_SOLUTION if one can't be found
*/

static int_T A465_search_solution(
                          const Joint_Solution soln_array[NUM_SOLNS],
                          uint16_T stance,
                          real_T old_joints[NUM_AXES],
                          int_T *bestindex)
{

    real_T cost = 9.9E7;
    /* initialize to an arbitrary high number */
    real_T new_cost;
    int_T solutionlist[NUM_SOLN] = {1,1,1,1,1,1,1,1};
    int_T i;

    *bestindex = NO_SOLUTION;

/* at this point ALL solutions are possibilities.  We must
 * determine which are allowed and which are not (based on given
 * stance), eliminating the latter  */

    /* begin with the reach variable*/
    if(!REACH_FREE(stance))
    {
        if(REACH_BACK(stance))
            /* eliminate all the reach forward solutions */
            for(i=FUN;i<=FDF;i++)
                solutionlist[i] = 0;
        else
```

```
            /* eliminate all the reach back solutions  */
            for(i=BUN;i<=BDF;i++)
                solutionlist[i] = 0;
    }

    /* now check the elbow */
    if(!ELBOW_FREE(stance))
    {
        if(ELBOW_UP(stance))
        {
            /* eliminate all the elbow down solutions */
            solutionlist[FDN] = 0;
            solutionlist[FDF] = 0;
            solutionlist[BDN] = 0;
            solutionlist[BDF] = 0;
        }
        else
        {
            /* eliminate all the elbow up solutions */
            solutionlist[FUN] = 0;
            solutionlist[FUF] = 0;
            solutionlist[BUN] = 0;
            solutionlist[BUF] = 0;
        }
    }

    /* lastly check the wrist */
    if(!WRIST_FREE(stance))
    {
        if(WRIST_FLIP(stance))
            /* eliminate all the wrist not-flipped solutions */
            for(i=FUN;i<=BDN;i+=2)
                solutionlist[i] = 0;
        else
            /* eliminate all the wrist flipped solutions */
            for(i=FUF;i<=BDF;i+=2)
                solutionlist[i] = 0;
    }


/* now we have left an array with between 1 and 8
 *possible solutions. All are allowed,
 *however not all are best. We will check each of
 * them to determine which is best. */

    for(i=0;i<NUM_SOLN;i++)
    {
    if((solutionlist[i]!=0)&&
            (A465_kin_joint_check(&soln_array[i].joint[J1])==OK))
        {
            new_cost =
             A465_calc_cost_function(
                        &soln_array[i].joint[J1],
                         old_joints);
            if(new_cost < cost)
            {
                cost = new_cost;
                *bestindex = i;
            }
        }
    }

    if(*bestindex == NO_SOLUTION)
```

```
        return(-ILLEGAL_TRANSFORM);
    else
        return OK;


}


/*
*This is the gateway to the inverse kinematics algorithms.
*A world coordinate position and orientation is given,
* along with stance information and old joint angles.
* A pointer to a set of new joint angles is returned.
*/

static int_T A465_world_to_joint(
                            uint16_T stance,
                            const real_T old_joint_angles[NUM_AXES],
                            const real_T world_coords[NUM_AXES],
                            real_T joint_angles[NUM_AXES])
{
    real_T inv_shoulder_column[XFORM_SIZE]=INV_SHOULDER_MATRIX;
    real_T inv_flange_length[XFORM_SIZE]=INV_FLANGE_MATRIX;
    real_T inv_bottom_transform[XFORM_SIZE];
    real_T inv_terminal_transform[XFORM_SIZE];
    real_T arm_without_terminal[XFORM_SIZE];
    real_T basic_arm_transform[XFORM_SIZE];
    real_T complete_arm_transform[XFORM_SIZE];
    real_T old_kin_joint_angles[NUM_AXES];
    Joint_Solution kin_joint_solution[NUM_SOLN];
    int_T best_soln;
    int_T retcode=0;



/* Convert the old joint angles from the user frame the kinematic frame */
A465_map_user_to_kin_space(old_joint_angles, old_kin_joint_angles);

/* get homogeneous transform of current position */
convert_worldcoords_to_homotransform(world_coords, complete_arm_transform);

/*Next we need to find out what other transforms and offsets are in place.
 *We are best to get the inverse of each of the transforms so we don't have
 *to invert matricies in real time.
 *                                                    */
/*  a) inverse base offset                            */

    /*  b) inverse shoulder column--initialized above    */

    /*  c) inverse flange offset--initialized above      */

    /*  d) inverse tool offset                           */


/*Now strip off the extra transforms to leave only
* the transform between the intersection of joints 1&2
* and joints 5&6.  Result is the basic arm transform.
*/

matrix_mult(
            inv_shoulder_column,
            A465_inv_basematrix,
            inv_bottom_transform);
```

```
matrix_mult(
            A465_inv_toolmatrix,
            inv_flange_length,
            inv_terminal_transform);

matrix_mult(
            complete_arm_transform,
            inv_terminal_transform,
            arm_without_terminal);
matrix_mult(
            inv_bottom_transform,
            arm_without_terminal,
            basic_arm_transform);


/*Call inverse kinematics to produce all possible joint solutions*/
    retcode = A465_inverse_kinematics(
                                        basic_arm_transform,
                                        kin_joint_solution,
                                        old_kin_joint_angles);

    /* Bail out immediately if we have an invkin error */
    if (retcode != OK)
        return retcode;


/*Now, choose the appropriate solution according
 * to given stance constraints*/
    retcode = A465_search_solution(
                                    kin_joint_solution,
                                    stance,
                                    old_kin_joint_angles,
                                    &best_soln);
    if( retcode == OK)
    {
/* We have a solution: convert joint angles to user frame */
        A465_map_kin_to_user_space(
                        &kin_joint_solution[best_soln].joint[J1],
                        joint_angles);
    }
    else  /* can not find a valid solution */
        return(retcode);


    return(OK);




}

static uint16_T computeStanceMask(
                    SimStruct *S,
                    InputRealPtrsType ppnStance)
{
 uint16_T nStance = 0;

  switch ((int_T)*ppnStance[REACH])
  {
    case -1: // free
    nStance |= FREE_REACH_BIT;
    break;
```

```
    case 1:
     nStance |= REACH_BIT;
     break;

    case 0:
     break;

     default:
     ssSetErrorStatus(S, "Illegal reach stance");
        break;
}

switch ((int_T)*ppnStance[ELBOW])
{
     case -1:  // free
     nStance |= FREE_ELBOW_BIT;
break;

case 1:
     nStance |= ELBOW_BIT;
     break;

     case 0:
     break;

     default:
     ssSetErrorStatus(S, "Illegal elbow stance");
     break;
     }

     switch ((int_T)*ppnStance[WRIST])
     {
     case -1: // free
     nStance |= FREE_WRIST_BIT;
     break;

     case 1:
     nStance |= WRIST_BIT;
     break;

     case 0:
     break;

     default:
     ssSetErrorStatus(S, "Illegal wrist stance");
     break;
     }

    return nStance;
}

/* Function: mdlOutputs =======================================
 * Abstract:
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
  InputRealPtrsType ppnWorldCoords =
              ssGetInputPortRealSignalPtrs(S, WORLD_COORDS);
  InputRealPtrsType ppnStance      =
              ssGetInputPortRealSignalPtrs(S, STANCE);
  InputRealPtrsType ppnOldJointAngles =
               ssGetInputPortRealSignalPtrs(S, OLD_JOINT_ANGLES);
  real_T *pnJointAngles  =
```

```
                ssGetOutputPortRealSignal(S, JOINT_ANGLES);
  real_T *pnError         =
                ssGetOutputPortRealSignal(S, ERROR_OUTPUT);
  int_T   nError;

    real_T    nWorldCoords[6];
    uint16_T  nStance;
    real_T    nOldJointAngles[NUM_AXES];
    int_T     i;

    for (i=0; i < 6; i++)
        nWorldCoords[i] = *ppnWorldCoords[i];
    for (i=0; i < NUM_AXES; i++)
        nOldJointAngles[i] = *ppnOldJointAngles[i];

    nStance = computeStanceMask(S, ppnStance);

    nError = -A465_world_to_joint(
                                  nStance,
                                  nOldJointAngles,
                                  nWorldCoords,
                                  pnJointAngles);
    pnError[0] = nError & 0x0f;
    // error code
    pnError[1] = (nError & 0xf0) >> 4;
    // offending joint angle

} /* end mdlOutputs */

/* Function: mdlTerminate =====================================
 * Abstract:
 *      Free the user data.
 */
static void mdlTerminate(SimStruct *S)
{
}

/*=============================*
 * Required S-function trailer *
 *=============================*/

#ifdef  MATLAB_MEX_FILE
/* Is this file being compiled as a MEX-file? */
#include "simulink.c"
/* MEX-file interface mechanism */
#else
#include "cg_sfun.h"
/* Code generation registration function */
#endif

#if defined(__cplusplus)
}
#endif
```

Appendice F

Funzione: Sfun_Qadc

```
/*
 * File: Sfun_Qadc1
 */


#define S_FUNCTION_NAME Sfun_Qadc1
#define S_FUNCTION_LEVEL 2
/*<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<*/
/* %%%-SFUNWIZ_defines_Changes_BEGIN --- EDIT HERE TO _END */
#define NUM_INPUTS              0

#define NUM_OUTPUTS             1
/* Output Port  0 */
#define OUT_PORT_0_NAME         y0
#define OUTPUT_0_WIDTH          1
#define OUTPUT_DIMS_0_COL       1
#define OUTPUT_0_DTYPE          int16_T
#define OUTPUT_0_COMPLEX        COMPLEX_NO
#define OUT_0_FRAME_BASED       FRAME_NO
#define OUT_0_DIMS              1-D
#define OUT_0_ISSIGNED           1
#define OUT_0_WORDLENGTH         8
#define OUT_0_FIXPOINTSCALING    1
#define OUT_0_FRACTIONLENGTH     3
#define OUT_0_BIAS               0
#define OUT_0_SLOPE              0.125

#define NPARAMS                 0

#define SAMPLE_TIME_0           INHERITED_SAMPLE_TIME
#define NUM_DISC_STATES         0
#define DISC_STATES_IC          [0]
#define NUM_CONT_STATES         0
#define CONT_STATES_IC          [0]

#define SFUNWIZ_GENERATE_TLC    1
#define SOURCEFILES "__SFB__"
#define PANELINDEX              6
#define USE_SIMSTRUCT           0
#define SHOW_COMPILE_STEPS      0
#define CREATE_DEBUG_MEXFILE    0
#define SAVE_CODE_ONLY          0
#define SFUNWIZ_REVISION        3.0
/* %%%-SFUNWIZ_defines_Changes_END --- EDIT HERE TO _BEGIN */
/*<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<*/
#include "simstruc.h"

// /*declare prototypes */
#include <windows.h>
#include <stdio.h>
#include <math.h>
#include "WinIo.h"

#define base_port 0x320
#define digin_port base_port + 0x00
#define digout_port base_port + 0x00
```

```
#define dac_cs base_port + 0x02
#define ad_cs base_port + 0x04
#define status_reg base_port + 0x06
#define control_reg base_port + 0x06
#define clk_reg base_port + 0x08
#define enc_reg1 base_port + 0x0c
#define enc_reg2 base_port + 0x0e

#define AD_SH 0x200 /* active low */
#define AD_AUTOCAL 0x100 /* active high */
#define AD_AUTOZ 0x80 /* active high */
#define AD_MUX_EN 0x40 /* active high */
#define AD_CLOCK_4M 0x400 /* high = 4 MHz */

/* IMPORTANT */
/* sample and hold disabled to prevent exrtaneous sampling */
/* and fix the clock speed to 4 MHz */

#define CONTROL_MUST (AD_SH | AD_CLOCK_4M)
unsigned int control_word = CONTROL_MUST;

static int open_dll = 0;


/*====================*
 * S-function methods *
 *====================*/
/* Function: mdlInitializeSizes ===============================
 * Abstract:
 *   Setup sizes of the various vectors.
 */
static void mdlInitializeSizes(SimStruct *S)
{

    DECL_AND_INIT_DIMSINFO(outputDimsInfo);
    ssSetNumSFcnParams(S, NPARAMS);
     if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
return;
/* Parameter mismatch will be reported by Simulink */
     }

    ssSetNumContStates(S, NUM_CONT_STATES);
    ssSetNumDiscStates(S, NUM_DISC_STATES);

    if (!ssSetNumInputPorts(S, NUM_INPUTS)) return;

    if (!ssSetNumOutputPorts(S, NUM_OUTPUTS)) return;
    ssSetOutputPortWidth(S, 0, OUTPUT_0_WIDTH);
    ssSetOutputPortDataType(S, 0, SS_INT16);
    ssSetOutputPortComplexSignal(S, 0, OUTPUT_0_COMPLEX);
    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);

/* Take care when specifying exception free code - see sfuntmpl_doc.c*/
    ssSetOptions(S, (SS_OPTION_EXCEPTION_FREE_CODE |
                     SS_OPTION_USE_TLC_WITH_ACCELERATOR |
                     SS_OPTION_WORKS_WITH_CODE_REUSE));
}
```

```
/* Function: mdlInitializeSampleTimes =========================================
 * Abstract:
 *    Specifiy  the sample time.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, SAMPLE_TIME_0);
    ssSetOffsetTime(S, 0, 0.0);
}

#define MDL_SET_OUTPUT_PORT_DATA_TYPE
static void mdlSetOutputPortDataType(



{
    ssSetOutputPortDataType(S, 0, dType);
}

#define MDL_SET_DEFAULT_PORT_DATA_TYPES
static void mdlSetDefaultPortDataTypes(SimStruct *S)
{
   ssSetOutputPortDataType(S, 0, SS_DOUBLE);
}
/* Function: mdlOutputs =======================================================
 *
*/
static void mdlOutputs(SimStruct *S, int_T tid)
{
   int16_T  *y0  = (int16_T *)ssGetOutputPortRealSignal(S,0);

//     int ch = 0;
    int ch, toolong, maxcnt;

    DWORD lbw, lb, hb; //, hbw;
    PDWORD lbw_p, lb_p, hb_p; // *hbw_p;
    lbw_p = &lbw;
//     hbw_p = &hbw;
    lb_p = &lb;
    hb_p = &hb;

    ch = 2;
    maxcnt = 30;
//     nosound();


    if ( open_dll == 0 )
    {
        InitializeWinIo();

        open_dll = 1;
    }
    control_word = CONTROL_MUST | AD_MUX_EN | (ch<<3);
    /* select channel and enable mux start S/H*/
    SetPortVal(control_reg, control_word, 2);

    toolong = 0;
    GetPortVal(status_reg, lbw_p, 2);

    while( (lbw&0x0008 == 0x0000 ) && (toolong <maxcnt) ) toolong++;
    if(toolong>=maxcnt) Beep(750,300);
    SetPortVal(ad_cs, 0, 1);
    GetPortVal(status_reg, lbw_p, 2);
```

```
    while( lbw&0x0010 == 0x0000 );
    GetPortVal(ad_cs, hb_p, 1);
        //hb_p = &hb;
    //hb = hb & 0xff;
    //y0[0]= *hb_p;
    GetPortVal(ad_cs, lb_p, 1);

    //lb_p = &lb;
    //lb = lb & 0xff;
    SetPortVal(control_reg, CONTROL_MUST, 2);
//  lb = 10;
//  hb = 11;


    *y0 = (int16_T)( ((hb&0x00ff)<<8) | lb&0x00ff);

    // y0[0] = (hb<<8); // | (lb&0xff);


}



/* Function: mdlTerminate =====================================
 * Abstract:
 *In this function, you should perform any actions that are necessary
 *at the termination of a simulation.  For example, if memory was
 *allocated in mdlStart, this is the place to free it.
 */
static void mdlTerminate(SimStruct *S)
{
    if ( open_dll == 1)
    {
        ShutdownWinIo();
       open_dll = 0;
    }
}

#ifdef  MATLAB_MEX_FILE
#include "simulink.c"
#else
#include "cg_sfun.h"
#endif
```

Appendice G

Sfun_DAC.c

```
/*
 * File: Sfun_DAC.c
 */

#define S_FUNCTION_NAME Sfun_DAC
#define S_FUNCTION_LEVEL 2
/*<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<*/
/* %%%-SFUNWIZ_defines_Changes_BEGIN --- EDIT HERE TO _END */
#define NUM_INPUTS              8
/* Input Port  0 */
#define IN_PORT_0_NAME          ivalue0
#define INPUT_0_WIDTH           1
#define INPUT_DIMS_0_COL        1
#define INPUT_0_DTYPE           int16_T
#define INPUT_0_COMPLEX         COMPLEX_NO
#define IN_0_FRAME_BASED        FRAME_NO
#define IN_0_DIMS               1-D
#define INPUT_0_FEEDTHROUGH     1
#define IN_0_ISSIGNED            0
#define IN_0_WORDLENGTH          8
#define IN_0_FIXPOINTSCALING     1
#define IN_0_FRACTIONLENGTH      9
#define IN_0_BIAS                0
#define IN_0_SLOPE               0.125

/* Input Port  1 */
#define IN_PORT_1_NAME          ivalue1
#define INPUT_1_WIDTH           1
#define INPUT_DIMS_1_COL        1
#define INPUT_1_DTYPE           int16_T
#define INPUT_1_COMPLEX         COMPLEX_NO
#define IN_1_FRAME_BASED        FRAME_NO
#define IN_1_DIMS               1-D
#define INPUT_1_FEEDTHROUGH     1
#define IN_1_ISSIGNED            0
#define IN_1_WORDLENGTH          8
#define IN_1_FIXPOINTSCALING     1
#define IN_1_FRACTIONLENGTH      9
#define IN_1_BIAS                0
#define IN_1_SLOPE               0.125

/* Input Port  2 */
#define IN_PORT_2_NAME          ivalue2
#define INPUT_2_WIDTH           1
#define INPUT_DIMS_2_COL        1
#define INPUT_2_DTYPE           int16_T
#define INPUT_2_COMPLEX         COMPLEX_NO
#define IN_2_FRAME_BASED        FRAME_NO
#define IN_2_DIMS               1-D
#define INPUT_2_FEEDTHROUGH     1
#define IN_2_ISSIGNED            0
#define IN_2_WORDLENGTH          8
#define IN_2_FIXPOINTSCALING     1
#define IN_2_FRACTIONLENGTH      9
#define IN_2_BIAS                0
#define IN_2_SLOPE               0.125
```

```
/* Input Port  3 */
#define IN_PORT_3_NAME          ivalue3
#define INPUT_3_WIDTH           1
#define INPUT_DIMS_3_COL        1
#define INPUT_3_DTYPE           int16_T
#define INPUT_3_COMPLEX         COMPLEX_NO
#define IN_3_FRAME_BASED        FRAME_NO
#define IN_3_DIMS               1-D
#define INPUT_3_FEEDTHROUGH     1
#define IN_3_ISSIGNED            0
#define IN_3_WORDLENGTH          8
#define IN_3_FIXPOINTSCALING     1
#define IN_3_FRACTIONLENGTH      9
#define IN_3_BIAS                0
#define IN_3_SLOPE               0.125

/* Input Port  4 */
#define IN_PORT_4_NAME          ivalue4
#define INPUT_4_WIDTH           1
#define INPUT_DIMS_4_COL        1
#define INPUT_4_DTYPE           int16_T
#define INPUT_4_COMPLEX         COMPLEX_NO
#define IN_4_FRAME_BASED        FRAME_NO
#define IN_4_DIMS               1-D
#define INPUT_4_FEEDTHROUGH     1
#define IN_4_ISSIGNED            0
#define IN_4_WORDLENGTH          8
#define IN_4_FIXPOINTSCALING     1
#define IN_4_FRACTIONLENGTH      9
#define IN_4_BIAS                0
#define IN_4_SLOPE               0.125

/* Input Port  5 */
#define IN_PORT_5_NAME          ivalue5
#define INPUT_5_WIDTH           1
#define INPUT_DIMS_5_COL        1
#define INPUT_5_DTYPE           int16_T
#define INPUT_5_COMPLEX         COMPLEX_NO
#define IN_5_FRAME_BASED        FRAME_NO
#define IN_5_DIMS               1-D
#define INPUT_5_FEEDTHROUGH     1
#define IN_5_ISSIGNED            0
#define IN_5_WORDLENGTH          8
#define IN_5_FIXPOINTSCALING     1
#define IN_5_FRACTIONLENGTH      9
#define IN_5_BIAS                0
#define IN_5_SLOPE               0.125

/* Input Port  6 */
#define IN_PORT_6_NAME          ivalue6
#define INPUT_6_WIDTH           1
#define INPUT_DIMS_6_COL        1
#define INPUT_6_DTYPE           int16_T
#define INPUT_6_COMPLEX         COMPLEX_NO
#define IN_6_FRAME_BASED        FRAME_NO
#define IN_6_DIMS               1-D
#define INPUT_6_FEEDTHROUGH     1
#define IN_6_ISSIGNED            0
#define IN_6_WORDLENGTH          8
#define IN_6_FIXPOINTSCALING     1
#define IN_6_FRACTIONLENGTH      9
#define IN_6_BIAS                0
```

```
#define IN_6_SLOPE              0.125

/* Input Port  7 */
#define IN_PORT_7_NAME          ivalue7
#define INPUT_7_WIDTH           1
#define INPUT_DIMS_7_COL        1
#define INPUT_7_DTYPE           int16_T
#define INPUT_7_COMPLEX         COMPLEX_NO
#define IN_7_FRAME_BASED        FRAME_NO
#define IN_7_DIMS               1-D
#define INPUT_7_FEEDTHROUGH     1
#define IN_7_ISSIGNED           0
#define IN_7_WORDLENGTH         8
#define IN_7_FIXPOINTSCALING    1
#define IN_7_FRACTIONLENGTH     9
#define IN_7_BIAS               0
#define IN_7_SLOPE              0.125

#define NUM_OUTPUTS             0

#define NPARAMS                 0

#define SAMPLE_TIME_0           INHERITED_SAMPLE_TIME
#define NUM_DISC_STATES         0
#define DISC_STATES_IC          [0]
#define NUM_CONT_STATES         0
#define CONT_STATES_IC          [0]

#define SFUNWIZ_GENERATE_TLC    1
#define SOURCEFILES "__SFB__"
#define PANELINDEX              6
#define USE_SIMSTRUCT           0
#define SHOW_COMPILE_STEPS      0
#define CREATE_DEBUG_MEXFILE    0
#define SAVE_CODE_ONLY          0
#define SFUNWIZ_REVISION        3.0
/* %%%-SFUNWIZ_defines_Changes_END --- EDIT HERE TO _BEGIN */
/*<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<*/
#include "simstruc.h"

// /*declare prototypes */
#include <windows.h>
#include <stdio.h>
#include <math.h>
#include "WinIo.h"

/* Definition of BASE_ADDRESS */

#define BASE                    0x320


/* Definition of Quanser Register Adresses for DA conversion */

#define DA_DATA                 (BASE + 0x02)
/*16bit write (only lowest 12bits matter) data for DA conversion*/
#define CONTROL_REG             (BASE + 0x06)
/* the address of the CONTROL REGISTER */
#define STATUS_REG              (BASE + 0x06)
/*address of CONTROL REGISTER (or STATUS REGISTER in case of reading)*/

/* Definition of 16 bit CONTROL REGISTER values for the Quanser card */

#define AD_SH                   0x200
```

```
/*(S/H) = disable sample and hold on the A/D (keep this 1 all the time)*/
#define LATCH_ON            0x1800
/*LD0 and LD1 on CONTROL REGISTER latched before DA conversion */
#define AD_CLOCK_4M         0x400
/* (CLK) = choose clock frequency of the A/D, (1 = 4MHz, 0 = 2 MHz) */
#define CONTROL_MUST        (AD_SH | AD_CLOCK_4M)
/* bit wise OR, so we keep both S/H and CLK high all the time */

static int open_dll = 0;

/*====================*
 * S-function methods *
 *====================*/
/* Function: mdlInitializeSizes ========================
 * Abstract:
 *   Setup sizes of the various vectors.
 */
static void mdlInitializeSizes(SimStruct *S)
{

    DECL_AND_INIT_DIMSINFO(inputDimsInfo);
    DECL_AND_INIT_DIMSINFO(outputDimsInfo);
    ssSetNumSFcnParams(S, NPARAMS);
     if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
         return;
         /* Parameter mismatch will be reported by Simulink */
     }

    ssSetNumContStates(S, NUM_CONT_STATES);
    ssSetNumDiscStates(S, NUM_DISC_STATES);

    if (!ssSetNumInputPorts(S, NUM_INPUTS)) return;
    /*Input Port 0*/
    ssSetInputPortWidth(S, 0, INPUT_0_WIDTH);
    ssSetInputPortDataType(S, 0, SS_INT16);
    ssSetInputPortComplexSignal(S, 0, INPUT_0_COMPLEX);
    ssSetInputPortDirectFeedThrough(S, 0, INPUT_0_FEEDTHROUGH);
    ssSetInputPortRequiredContiguous(S, 0, 1);
    /*direct input signal access*/

    /*Input Port 1*/
    ssSetInputPortWidth(S, 1, INPUT_0_WIDTH);
    ssSetInputPortDataType(S, 1, SS_INT16);
    ssSetInputPortComplexSignal(S, 1, INPUT_0_COMPLEX);
    ssSetInputPortDirectFeedThrough(S, 1, INPUT_0_FEEDTHROUGH);
    ssSetInputPortRequiredContiguous(S, 1, 1);
    /*direct input signal access*/

    /*Input Port 2*/
    ssSetInputPortWidth(S, 2, INPUT_0_WIDTH);
    ssSetInputPortDataType(S, 2, SS_INT16);
    ssSetInputPortComplexSignal(S, 2, INPUT_0_COMPLEX);
    ssSetInputPortDirectFeedThrough(S, 2, INPUT_0_FEEDTHROUGH);
    ssSetInputPortRequiredContiguous(S, 2, 1);
    /*direct input signal access*/

    /*Input Port 3*/
    ssSetInputPortWidth(S, 3, INPUT_0_WIDTH);
    ssSetInputPortDataType(S, 3, SS_INT16);
    ssSetInputPortComplexSignal(S, 3, INPUT_0_COMPLEX);
    ssSetInputPortDirectFeedThrough(S, 3, INPUT_0_FEEDTHROUGH);
    ssSetInputPortRequiredContiguous(S, 3, 1);
    /*direct input signal access*/
```

```
    /*Input Port 4*/
    ssSetInputPortWidth(S, 4, INPUT_0_WIDTH);
    ssSetInputPortDataType(S, 4, SS_INT16);
    ssSetInputPortComplexSignal(S, 4, INPUT_0_COMPLEX);
    ssSetInputPortDirectFeedThrough(S, 4, INPUT_0_FEEDTHROUGH);
    ssSetInputPortRequiredContiguous(S, 4, 1);
    /*direct input signal access*/

    /*Input Port 5*/
    ssSetInputPortWidth(S, 5, INPUT_0_WIDTH);
    ssSetInputPortDataType(S, 5, SS_INT16);
    ssSetInputPortComplexSignal(S, 5, INPUT_0_COMPLEX);
    ssSetInputPortDirectFeedThrough(S, 5, INPUT_0_FEEDTHROUGH);
    ssSetInputPortRequiredContiguous(S, 5, 1);
    /*direct input signal access*/

    /*Input Port 6*/
    ssSetInputPortWidth(S, 6, INPUT_0_WIDTH);
    ssSetInputPortDataType(S, 6, SS_INT16);
    ssSetInputPortComplexSignal(S, 6, INPUT_0_COMPLEX);
    ssSetInputPortDirectFeedThrough(S, 6, INPUT_0_FEEDTHROUGH);
    ssSetInputPortRequiredContiguous(S, 6, 1);
    /*direct input signal access*/

    /*Input Port 7*/
    ssSetInputPortWidth(S, 7, INPUT_0_WIDTH);
    ssSetInputPortDataType(S, 7, SS_INT16);
    ssSetInputPortComplexSignal(S, 7, INPUT_0_COMPLEX);
    ssSetInputPortDirectFeedThrough(S, 7, INPUT_0_FEEDTHROUGH);
    ssSetInputPortRequiredContiguous(S, 7, 1);
    /*direct input signal access*/


    if (!ssSetNumOutputPorts(S, NUM_OUTPUTS)) return;

    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);

/*Take care when specifying exception free code, see sfuntmpl_doc.c */
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE );
}

# define MDL_SET_INPUT_PORT_FRAME_DATA
static void mdlSetInputPortFrameData(SimStruct  *S,
                                     int_T      port,
                                     Frame_T    frameData)
{
    ssSetInputPortFrameData(S, port, frameData);
}
/* Function: mdlInitializeSampleTimes =========================
 * Abstract:
 *    Specifiy  the sample time.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, SAMPLE_TIME_0);
    ssSetOffsetTime(S, 0, 0.0);
}
```

```
#define MDL_START
#if defined(MDL_START)
static void mdlStart(SimStruct *S)
{
}
#endif /*MDL_START*/

#define MDL_SET_INPUT_PORT_DATA_TYPE
static void mdlSetInputPortDataType(SimStruct *S, int port, DTypeId dType)
{
    ssSetInputPortDataType( S, 0, dType);
}
#define MDL_SET_OUTPUT_PORT_DATA_TYPE
static void mdlSetOutputPortDataType(SimStruct *S, int port, DTypeId dType)
{
    ssSetOutputPortDataType(S, 0, dType);
}

#define MDL_SET_DEFAULT_PORT_DATA_TYPES
static void mdlSetDefaultPortDataTypes(SimStruct *S)
{
    ssSetInputPortDataType( S, 0, SS_DOUBLE);
    ssSetOutputPortDataType(S, 0, SS_DOUBLE);
}
/* Function: mdlOutputs =======================================
 *
*/
static void mdlOutputs(SimStruct *S, int_T tid)
{

    const int16_T   *ivalue0  = (const int16_T*) ssGetInputPortSignal(S,0);
    const int16_T   *ivalue1  = (const int16_T*) ssGetInputPortSignal(S,1);
    const int16_T   *ivalue2  = (const int16_T*) ssGetInputPortSignal(S,2);
    const int16_T   *ivalue3  = (const int16_T*) ssGetInputPortSignal(S,3);
    const int16_T   *ivalue4  = (const int16_T*) ssGetInputPortSignal(S,4);
    const int16_T   *ivalue5  = (const int16_T*) ssGetInputPortSignal(S,5);
    const int16_T   *ivalue6  = (const int16_T*) ssGetInputPortSignal(S,6);
    const int16_T   *ivalue7  = (const int16_T*) ssGetInputPortSignal(S,7);

    if ( open_dll == 0 )
    {
        InitializeWinIo();
        open_dll = 1;
    }

    SetPortVal(CONTROL_REG, LATCH_ON | 0 | CONTROL_MUST, 2);
    SetPortVal(DA_DATA, *ivalue0, 2);
    SetPortVal(CONTROL_REG, CONTROL_MUST, 2);

    SetPortVal(CONTROL_REG, LATCH_ON | 1 | CONTROL_MUST, 2);
    SetPortVal(DA_DATA, *ivalue1, 2);
    SetPortVal(CONTROL_REG, CONTROL_MUST, 2);

    SetPortVal(CONTROL_REG, LATCH_ON | 2 | CONTROL_MUST, 2);
    SetPortVal(DA_DATA, *ivalue2, 2);
    SetPortVal(CONTROL_REG, CONTROL_MUST, 2);

    SetPortVal(CONTROL_REG, LATCH_ON | 3 | CONTROL_MUST, 2);
    SetPortVal(DA_DATA, *ivalue3, 2);
    SetPortVal(CONTROL_REG, CONTROL_MUST, 2);

    SetPortVal(CONTROL_REG, LATCH_ON | 4 | CONTROL_MUST, 2);
```

```
    SetPortVal(DA_DATA, *ivalue4, 2);
    SetPortVal(CONTROL_REG, CONTROL_MUST, 2);

    SetPortVal(CONTROL_REG, LATCH_ON | 5 | CONTROL_MUST, 2);
    SetPortVal(DA_DATA, *ivalue5, 2);
    SetPortVal(CONTROL_REG, CONTROL_MUST, 2);

    SetPortVal(CONTROL_REG, LATCH_ON | 6 | CONTROL_MUST, 2);
    SetPortVal(DA_DATA, *ivalue6, 2);
    SetPortVal(CONTROL_REG, CONTROL_MUST, 2);

    SetPortVal(CONTROL_REG, LATCH_ON | 7 | CONTROL_MUST, 2);
    SetPortVal(DA_DATA, *ivalue7, 2);
    SetPortVal(CONTROL_REG, CONTROL_MUST, 2);

}



/* Function: mdlTerminate =====================================
 * Abstract:
 *    In this function, you should perform any actions that are necessary
 *    at the termination of a simulation.  For example, if memory was
 *    allocated in mdlStart, this is the place to free it.
 */
static void mdlTerminate(SimStruct *S)
{
    if ( open_dll == 1)
    {
        SetPortVal(CONTROL_REG, LATCH_ON | 0 | CONTROL_MUST, 2);
        SetPortVal(DA_DATA, ceil(4095/2), 2);
        SetPortVal(CONTROL_REG, CONTROL_MUST, 2);

        SetPortVal(CONTROL_REG, LATCH_ON | 1 | CONTROL_MUST, 2);
        SetPortVal(DA_DATA, ceil(4095/2), 2);
        SetPortVal(CONTROL_REG, CONTROL_MUST, 2);

        SetPortVal(CONTROL_REG, LATCH_ON | 2 | CONTROL_MUST, 2);
        SetPortVal(DA_DATA, ceil(4095/2), 2);
        SetPortVal(CONTROL_REG, CONTROL_MUST, 2);

        SetPortVal(CONTROL_REG, LATCH_ON | 3 | CONTROL_MUST, 2);
        SetPortVal(DA_DATA, ceil(4095/2), 2);
        SetPortVal(CONTROL_REG, CONTROL_MUST, 2);

        SetPortVal(CONTROL_REG, LATCH_ON | 4 | CONTROL_MUST, 2);
        SetPortVal(DA_DATA, ceil(4095/2), 2);
        SetPortVal(CONTROL_REG, CONTROL_MUST, 2);

        SetPortVal(CONTROL_REG, LATCH_ON | 5 | CONTROL_MUST, 2);
        SetPortVal(DA_DATA, ceil(4095/2), 2);
        SetPortVal(CONTROL_REG, CONTROL_MUST, 2);

        SetPortVal(CONTROL_REG, LATCH_ON | 6 | CONTROL_MUST, 2);
        SetPortVal(DA_DATA, ceil(4095/2), 2);
        SetPortVal(CONTROL_REG, CONTROL_MUST, 2);

        SetPortVal(CONTROL_REG, LATCH_ON | 7 | CONTROL_MUST, 2);
        SetPortVal(DA_DATA, ceil(4095/2), 2);
        SetPortVal(CONTROL_REG, CONTROL_MUST, 2);

        ShutdownWinIo();
```

```
        open_dll = 0;
    }
}

#ifdef  MATLAB_MEX_FILE
#include "simulink.c"
#else
#include "cg_sfun.h"
#endif
```

Appendice H

Sfun_ENC_Res.c

```
/*
 * File: Sfun_ENC_Res.c
 */


#define S_FUNCTION_NAME Sfun_ENC_Res
#define S_FUNCTION_LEVEL 2
/*<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<*/
/* %%%-SFUNWIZ_defines_Changes_BEGIN --- EDIT HERE TO _END */
#define NUM_INPUTS              1
/* Input Port  0 */
#define IN_PORT_0_NAME          u0
#define INPUT_0_WIDTH           1
#define INPUT_DIMS_0_COL        1
#define INPUT_0_DTYPE           real_T
#define INPUT_0_COMPLEX         COMPLEX_NO
#define IN_0_FRAME_BASED        FRAME_NO
#define IN_0_DIMS               1-D
#define INPUT_0_FEEDTHROUGH     1
#define IN_0_ISSIGNED            0
#define IN_0_WORDLENGTH          8
#define IN_0_FIXPOINTSCALING     1
#define IN_0_FRACTIONLENGTH      9
#define IN_0_BIAS                0
#define IN_0_SLOPE               0.125

#define NUM_OUTPUTS             1
/* Output Port  0 */
#define OUT_PORT_0_NAME         y0
#define OUTPUT_0_WIDTH          8
#define OUTPUT_DIMS_0_COL       1
#define OUTPUT_0_DTYPE          int32_T
#define OUTPUT_0_COMPLEX        COMPLEX_NO
#define OUT_0_FRAME_BASED       FRAME_NO
#define OUT_0_DIMS              1-D
#define OUT_0_ISSIGNED           1
#define OUT_0_WORDLENGTH         8
#define OUT_0_FIXPOINTSCALING    1
#define OUT_0_FRACTIONLENGTH     3
#define OUT_0_BIAS               0
#define OUT_0_SLOPE              0.125

#define NPARAMS                 0

#define SAMPLE_TIME_0           INHERITED_SAMPLE_TIME
#define NUM_DISC_STATES         0
#define DISC_STATES_IC          [0]
#define NUM_CONT_STATES         0
#define CONT_STATES_IC          [0]

#define SFUNWIZ_GENERATE_TLC    1
#define SOURCEFILES             "__SFB__"
#define PANELINDEX              6
#define USE_SIMSTRUCT           0
#define SHOW_COMPILE_STEPS      0
#define CREATE_DEBUG_MEXFILE    0
```

```
#define SAVE_CODE_ONLY        0
#define SFUNWIZ_REVISION      3.0
/* %%%-SFUNWIZ_defines_Changes_END --- EDIT HERE TO _BEGIN */
/*<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<*/
#include "simstruc.h"

// /*declare prototypes */
#include <windows.h>
#include <stdio.h>
#include <math.h>
#include "WinIo.h"

#define base_port 0x320

#define digin_port    base_port + 0x00
#define digout_port   base_port + 0x00
#define dac_cs        base_port + 0x02
#define ad_cs         base_port + 0x04
#define status_reg    base_port + 0x06
#define control_reg   base_port + 0x06
#define clk_reg       base_port + 0x08
#define enc_reg1      base_port + 0x0c
#define enc_reg2      base_port + 0x0e

#define AD_SH        0x200 /* active low  */
#define AD_AUTOCAL   0x100 /* active high */
#define AD_AUTOZ     0x80  /* active high */
#define AD_MUX_EN    0x40  /* active high */
#define AD_CLOCK_4M 0x400 /* high = 4 MHz */

/* ENCODER addresses */
#define ENC_DATA      base_port+0xc
// data register of ENCODER CHANNEL
#define ENC_CONTROL   base_port+0xe
// control register ENCODER CHANNEL

/* ENCODER CHIP COMMANDS */
#define CLOCK_DATA       0      // FCK frequency divider
#define CLOCK_SETUP      0X98   //18 transfer PR0 to PSC
#define INPUT_SETUP      0XC1   //41 enable inputs A and B
#define QUAD_X4          0XB8   //38 quadrature
#define BP_RESET         0X01   // reset byte pointer
#define CNTR_RESET       0X02   // reset counter
#define TRSFRPR_CNTR     0X08   // transf preset regist to count
#define TRSFRCNTR_OL     0X10   // transf CNTR to OL (x and y)
#define EFLAG_RESET      0X06   // reset E bit of flag register


/* IMPORTANT */
/* sample and hold disabled to prevent exrtaneous sampling */
/* and fix the clock speed to 4 MHz */
#define CONTROL_MUST (AD_SH | AD_CLOCK_4M)

unsigned int control_word0 = CONTROL_MUST;
unsigned int control_word1 = CONTROL_MUST;
unsigned int control_word2 = CONTROL_MUST;
unsigned int control_word3 = CONTROL_MUST;
unsigned int control_word4 = CONTROL_MUST;
unsigned int control_word5 = CONTROL_MUST;
unsigned int control_word6 = CONTROL_MUST;
unsigned int control_word7 = CONTROL_MUST;

static int open_dll = 0;
```

```
int ch0 = 0; int ch1 = 1; int ch2 = 2; int ch3 = 3; int ch4 = 4;
int ch5 = 5; int ch6 = 6; int ch7 = 7;

/*====================*
 * S-function methods *
 *====================*/
/* Function: mdlInitializeSizes ========================
 * Abstract:
 *   Setup sizes of the various vectors.
 */
static void mdlInitializeSizes(SimStruct *S)
{

    DECL_AND_INIT_DIMSINFO(inputDimsInfo);
    DECL_AND_INIT_DIMSINFO(outputDimsInfo);
    ssSetNumSFcnParams(S, NPARAMS);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        return;
        /* Parameter mismatch will be reported by Simulink */
    }

    ssSetNumContStates(S, NUM_CONT_STATES);
    ssSetNumDiscStates(S, NUM_DISC_STATES);

    if (!ssSetNumInputPorts(S, NUM_INPUTS)) return;
    ssSetInputPortWidth(S, 0, INPUT_0_WIDTH);
    ssSetInputPortDataType(S, 0, SS_DOUBLE);
    ssSetInputPortComplexSignal(S, 0, INPUT_0_COMPLEX);
    ssSetInputPortDirectFeedThrough(S, 0, INPUT_0_FEEDTHROUGH);
    ssSetInputPortRequiredContiguous(S, 0, 1);
    /*direct input signal access*/

    if (!ssSetNumOutputPorts(S, NUM_OUTPUTS)) return;
    ssSetOutputPortWidth(S, 0, OUTPUT_0_WIDTH);
    ssSetOutputPortDataType(S, 0, SS_INT32);
    ssSetOutputPortComplexSignal(S, 0, OUTPUT_0_COMPLEX);
    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);

 /* Take care when specifying exception free code, sfuntmpl_doc.c */
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}

# define MDL_SET_INPUT_PORT_FRAME_DATA
static void mdlSetInputPortFrameData(SimStruct  *S,
int_T       port,
Frame_T     frameData)
{
    ssSetInputPortFrameData(S, port, frameData);
}
/* Function: mdlInitializeSampleTimes ==================
 * Abstract:
 *    Specifiy  the sample time.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, SAMPLE_TIME_0);
    ssSetOffsetTime(S, 0, 0.0);
```

```
}


#define MDL_START
/* Change to #undef to remove function */
#if defined(MDL_START)
  /* Function: mdlStart =======================================
   * Abstract:
   *    This function is called once at start of model execution. If you
   *    have states that should be initialized once, this is the place
   *    to do it.
   */
static void mdlStart(SimStruct *S)
{
}
#endif /*  MDL_START */

#define MDL_SET_INPUT_PORT_DATA_TYPE
static void mdlSetInputPortDataType(SimStruct *S, int port, DTypeId dType)
{
    ssSetInputPortDataType( S, 0, dType);
}
#define MDL_SET_OUTPUT_PORT_DATA_TYPE
static void mdlSetOutputPortDataType(SimStruct *S, int port, DTypeId dType)
{
    ssSetOutputPortDataType(S, 0, dType);
}

#define MDL_SET_DEFAULT_PORT_DATA_TYPES
static void mdlSetDefaultPortDataTypes(SimStruct *S)
{
    ssSetInputPortDataType( S, 0, SS_DOUBLE);
    ssSetOutputPortDataType(S, 0, SS_DOUBLE);
}
/* Function: mdlOutputs =======================================
 *
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
    const real_T   *u0  = (const real_T*) ssGetInputPortSignal(S,0);
    int32_T        *y0  = (int32_T *)ssGetOutputPortRealSignal(S,0);

    DWORD low_byte0, mid_byte0, high_byte0;
    DWORD low_byte1, mid_byte1, high_byte1;
    DWORD low_byte2, mid_byte2, high_byte2;
    DWORD low_byte3, mid_byte3, high_byte3;
    DWORD low_byte4, mid_byte4, high_byte4;
    DWORD low_byte5, mid_byte5, high_byte5;
    DWORD low_byte6, mid_byte6, high_byte6;
    DWORD low_byte7, mid_byte7, high_byte7;

    PDWORD low_byte0_p, mid_byte0_p, high_byte0_p;
    PDWORD low_byte1_p, mid_byte1_p, high_byte1_p;
    PDWORD low_byte2_p, mid_byte2_p, high_byte2_p;
    PDWORD low_byte3_p, mid_byte3_p, high_byte3_p;
    PDWORD low_byte4_p, mid_byte4_p, high_byte4_p;
    PDWORD low_byte5_p, mid_byte5_p, high_byte5_p;
    PDWORD low_byte6_p, mid_byte6_p, high_byte6_p;
    PDWORD low_byte7_p, mid_byte7_p, high_byte7_p;

    unsigned int high_word0, low_word0;
    unsigned int high_word1, low_word1;
    unsigned int high_word2, low_word2;
```

```
unsigned int high_word3, low_word3;
unsigned int high_word4, low_word4;
unsigned int high_word5, low_word5;
unsigned int high_word6, low_word6;
unsigned int high_word7, low_word7;


low_byte0_p = &low_byte0;
mid_byte0_p = &mid_byte0;
high_byte0_p = &high_byte0;

low_byte1_p = &low_byte1;
mid_byte1_p = &mid_byte1;
high_byte1_p = &high_byte1;

low_byte2_p = &low_byte2;
mid_byte2_p = &mid_byte2;
high_byte2_p = &high_byte2;

low_byte3_p = &low_byte3;
mid_byte3_p = &mid_byte3;
high_byte3_p = &high_byte3;

low_byte4_p = &low_byte4;
mid_byte4_p = &mid_byte4;
high_byte4_p = &high_byte4;

low_byte5_p = &low_byte5;
mid_byte5_p = &mid_byte5;
high_byte5_p = &high_byte5;

low_byte6_p = &low_byte6;
mid_byte6_p = &mid_byte6;
high_byte6_p = &high_byte6;

low_byte7_p = &low_byte7;
mid_byte7_p = &mid_byte7;
high_byte7_p = &high_byte7;

if ( open_dll == 0 )
{
    InitializeWinIo();
    open_dll = 1;
}

/*Encoder 0*/

if (u0[0] == 1) /*reset encoder0*/
{

    control_word0 = CONTROL_MUST | (ch0<<3);
    // select channel and enable mux start S and H
    SetPortVal(control_reg, control_word0, 1);
    /* select the encoder channel */
        /*initialize the ENCODER CHIP*/
    SetPortVal(ENC_CONTROL, EFLAG_RESET, 1);
    // reset E bit of flag register
    SetPortVal(ENC_CONTROL, BP_RESET, 1);
    // reset byte pointer (x and y)
    SetPortVal(ENC_DATA, CLOCK_DATA, 1);
    // FCK frequency divider
    SetPortVal(ENC_CONTROL, CLOCK_SETUP, 1);
    // transfer PR0 to PSC (x and y)
```

```
    SetPortVal(ENC_CONTROL, INPUT_SETUP, 1);
    // enable inputs A and B (x and y)
    SetPortVal(ENC_CONTROL, QUAD_X4, 1);
    // quadrature multiplier to 4 (x and y)
    SetPortVal(ENC_CONTROL, CNTR_RESET, 1);
    // reset counter (x and y)
}

/*read data encoder0*/
control_word0 = CONTROL_MUST | AD_MUX_EN | (ch0<<3);
// select channel
SetPortVal(control_reg, control_word0, 1);
/*SELECT THE ENCODER CHANNEL USING THE MUX */
SetPortVal(ENC_CONTROL, BP_RESET, 1);
// reset byte pointer
SetPortVal(ENC_CONTROL, TRSFRCNTR_OL, 1);
// latch the data
GetPortVal(ENC_DATA, low_byte0_p, 1);
// least significant byte
low_byte0=low_byte0&0xff;
GetPortVal(ENC_DATA, mid_byte0_p, 1);
mid_byte0=mid_byte0&0xff;
low_word0 = low_byte0 | (mid_byte0 << 8)&0xffff;
GetPortVal(ENC_DATA, high_byte0_p, 1);
// most significant byte
high_byte0=high_byte0&0xff;
high_word0 = high_byte0&0xffff;

if(high_word0 & 0x80) high_word0 = high_word0 | 0xff00;
/*convert to signed 32 bit*/
y0[0] = ((uint32_T)high_word0 << 16) | low_word0;

/*Encoder 1*/

if (u0[0] == 1)  /*reset encoder1*/
{
    control_word1 = CONTROL_MUST | (ch1<<3);
    // select channel and enable mux start S and H
    SetPortVal(control_reg, control_word1, 1);
    /* select the encoder channel */
        /*initialize the ENCODER CHIP*/
    SetPortVal(ENC_CONTROL, EFLAG_RESET, 1);
    // reset E bit of flag register
    SetPortVal(ENC_CONTROL, BP_RESET, 1);
    // reset byte pointer (x and y)
    SetPortVal(ENC_DATA, CLOCK_DATA, 1);
    // FCK frequency divider
    SetPortVal(ENC_CONTROL, CLOCK_SETUP, 1);
    // transfer PR0 to PSC (x and y)
    SetPortVal(ENC_CONTROL, INPUT_SETUP, 1);
    // enable inputs A and B (x and y)
    SetPortVal(ENC_CONTROL, QUAD_X4, 1);
    // quadrature multiplier to 4 (x and y)
    SetPortVal(ENC_CONTROL, CNTR_RESET, 1);
    // reset counter (x and y)
}

/*read data encoder1*/
control_word1 = CONTROL_MUST | AD_MUX_EN | (ch1<<3);
// select channel
SetPortVal(control_reg, control_word1, 1);
/*SELECT THE ENCODER CHANNEL USING THE MUX */
SetPortVal(ENC_CONTROL, BP_RESET, 1);
```

```
    // reset byte pointer
    SetPortVal(ENC_CONTROL, TRSFRCNTR_OL, 1);
    // latch the data
    GetPortVal(ENC_DATA, low_byte1_p, 1);
    // least significant byte
    low_byte1=low_byte1&0xff;
    GetPortVal(ENC_DATA, mid_byte1_p, 1);
    mid_byte1=mid_byte1&0xff;
    low_word1 = low_byte1 | (mid_byte1 << 8)&0xffff;
    GetPortVal(ENC_DATA, high_byte1_p, 1);
    // most significant byte
    high_byte1=high_byte1&0xff;
    high_word1 = high_byte1&0xffff;

    if(high_word1 & 0x80) high_word1 = high_word1 | 0xff00;
    /*convert to signed 32 bit*/
    y0[1] = ((uint32_T)high_word1 << 16) | low_word1;

/*Encoder 2*/

    if (u0[0] == 1)  /*reset encoder2*/
    {
        control_word2 = CONTROL_MUST | (ch2<<3);
        // select channel and enable mux start S and H
        SetPortVal(control_reg, control_word2, 1);
         /* select the encoder channel */
            /*initialize the ENCODER CHIP*/
        SetPortVal(ENC_CONTROL, EFLAG_RESET, 1);
        // reset E bit of flag register
        SetPortVal(ENC_CONTROL, BP_RESET, 1);
        // reset byte pointer (x and y)
        SetPortVal(ENC_DATA, CLOCK_DATA, 1);
        // FCK frequency divider
        SetPortVal(ENC_CONTROL, CLOCK_SETUP, 1);
        // transfer PR0 to PSC (x and y)
        SetPortVal(ENC_CONTROL, INPUT_SETUP, 1);
         // enable inputs A and B (x and y)
        SetPortVal(ENC_CONTROL, QUAD_X4, 1);
        // quadrature multiplier to 4 (x and y)
        SetPortVal(ENC_CONTROL, CNTR_RESET, 1);
        // reset counter (x and y)
    }


    /*read data encoder2*/
    control_word2 = CONTROL_MUST | AD_MUX_EN | (ch2<<3);
    // select channel
    SetPortVal(control_reg, control_word2, 1);
    /*SELECT THE ENCODER CHANNEL USING THE MUX */
    SetPortVal(ENC_CONTROL, BP_RESET, 1);
    // reset byte pointer
    SetPortVal(ENC_CONTROL, TRSFRCNTR_OL, 1);
    // latch the data
    GetPortVal(ENC_DATA, low_byte2_p, 1);
    // least significant byte
    low_byte2=low_byte2&0xff;
    GetPortVal(ENC_DATA, mid_byte2_p, 1);
    mid_byte2=mid_byte2&0xff;
    low_word2 = low_byte2 | (mid_byte2 << 8)&0xffff;
    GetPortVal(ENC_DATA, high_byte2_p, 1);
    // most significant byte
    high_byte2=high_byte2&0xff;
    high_word2 = high_byte2&0xffff;
```

```
    if(high_word2 & 0x80) high_word2 = high_word2 | 0xff00;
    /*convert to signed 32 bit*/
    y0[2] = ((uint32_T)high_word2 << 16) | low_word2;

/*Encoder 3*/

    if (u0[0] == 1)  /*reset encoder3*/
    {
        control_word3 = CONTROL_MUST | (ch3<<3);
        // select channel and enable mux start S and H
        SetPortVal(control_reg, control_word3, 1);
        /* select the encoder channel */
            /*initialize the ENCODER CHIP*/
        SetPortVal(ENC_CONTROL, EFLAG_RESET, 1);
        // reset E bit of flag register
        SetPortVal(ENC_CONTROL, BP_RESET, 1);
        // reset byte pointer (x and y)
        SetPortVal(ENC_DATA, CLOCK_DATA, 1);
        // FCK frequency divider
        SetPortVal(ENC_CONTROL, CLOCK_SETUP, 1);
        // transfer PR0 to PSC (x and y)
        SetPortVal(ENC_CONTROL, INPUT_SETUP, 1);
        // enable inputs A and B (x and y)
        SetPortVal(ENC_CONTROL, QUAD_X4, 1);
        // quadrature multiplier to 4 (x and y)
        SetPortVal(ENC_CONTROL, CNTR_RESET, 1);
        // reset counter (x and y)
    }

    /*read data encoder3*/
    control_word3 = CONTROL_MUST | AD_MUX_EN | (ch3<<3);
    // select channel
    SetPortVal(control_reg, control_word3, 1);
    /*SELECT THE ENCODER CHANNEL USING THE MUX */
    SetPortVal(ENC_CONTROL, BP_RESET, 1);
    // reset byte pointer
    SetPortVal(ENC_CONTROL, TRSFRCNTR_OL, 1);
    // latch the data
    GetPortVal(ENC_DATA, low_byte3_p, 1);
    // least significant byte
    low_byte3=low_byte3&0xff;
    GetPortVal(ENC_DATA, mid_byte3_p, 1);
    mid_byte3=mid_byte3&0xff;
    low_word3 = low_byte3 | (mid_byte3 << 8)&0xffff;
    GetPortVal(ENC_DATA, high_byte3_p, 1);
    // most significant byte
    high_byte3=high_byte3&0xff;
    high_word3 = high_byte3&0xffff;

    if(high_word3 & 0x80) high_word3 = high_word3 | 0xff00;
    /*convert to signed 32 bit*/
    y0[3] = ((uint32_T)high_word3 << 16) | low_word3;

/*Encoder 4*/

    if (u0[0] == 1)  /*reset encoder4*/
    {
        control_word4 = CONTROL_MUST | (ch4<<3);
        // select channel and enable mux start S and H
        SetPortVal(control_reg, control_word4, 1);
        /* select the encoder channel */
            /*initialize the ENCODER CHIP*/
```

```
        SetPortVal(ENC_CONTROL, EFLAG_RESET, 1);
        // reset E bit of flag register
        SetPortVal(ENC_CONTROL, BP_RESET, 1);
        // reset byte pointer (x and y)
        SetPortVal(ENC_DATA, CLOCK_DATA, 1);
        // FCK frequency divider
        SetPortVal(ENC_CONTROL, CLOCK_SETUP, 1);
        // transfer PR0 to PSC (x and y)
        SetPortVal(ENC_CONTROL, INPUT_SETUP, 1);
         // enable inputs A and B (x and y)
        SetPortVal(ENC_CONTROL, QUAD_X4, 1);
        // quadrature multiplier to 4 (x and y)
        SetPortVal(ENC_CONTROL, CNTR_RESET, 1);
        // reset counter (x and y)
    }

    /*read data encoder4*/
    control_word4 = CONTROL_MUST | AD_MUX_EN | (ch4<<3);
    // select channel
    SetPortVal(control_reg, control_word4, 1);
     /*SELECT THE ENCODER CHANNEL USING THE MUX */
    SetPortVal(ENC_CONTROL, BP_RESET, 1);
    // reset byte pointer
    SetPortVal(ENC_CONTROL, TRSFRCNTR_OL, 1);
     // latch the data
    GetPortVal(ENC_DATA, low_byte4_p, 1);
    // least significant byte
    low_byte4=low_byte4&0xff;
    GetPortVal(ENC_DATA, mid_byte4_p, 1);
    mid_byte4=mid_byte4&0xff;
    low_word4 = low_byte4 | (mid_byte4 << 8)&0xffff;
    GetPortVal(ENC_DATA, high_byte4_p, 1);
    // most significant byte
    high_byte4=high_byte4&0xff;
    high_word4 = high_byte4&0xffff;

    if(high_word4 & 0x80) high_word4 = high_word4 | 0xff00;
    /*convert to signed 32 bit*/
    y0[4] = ((uint32_T)high_word4 << 16) | low_word4;

/*Encoder 5*/

    if (u0[0] == 1)  /*reset encoder5*/
    {
        control_word5 = CONTROL_MUST | (ch5<<3);
         // select channel and enable mux start S and H
        SetPortVal(control_reg, control_word5, 1);
         /* select the encoder channel */
            /*initialize the ENCODER CHIP*/
        SetPortVal(ENC_CONTROL, EFLAG_RESET, 1);
        // reset E bit of flag register
        SetPortVal(ENC_CONTROL, BP_RESET, 1);
        // reset byte pointer (x and y)
        SetPortVal(ENC_DATA, CLOCK_DATA, 1);
        // FCK frequency divider
        SetPortVal(ENC_CONTROL, CLOCK_SETUP, 1);
         // transfer PR0 to PSC (x and y)
        SetPortVal(ENC_CONTROL, INPUT_SETUP, 1);
         // enable inputs A and B (x and y)
        SetPortVal(ENC_CONTROL, QUAD_X4, 1);
        // quadrature multiplier to 4 (x and y)
        SetPortVal(ENC_CONTROL, CNTR_RESET, 1);
         // reset counter (x and y)
```

```
    }

    /*read data encoder5*/
    control_word5 = CONTROL_MUST | AD_MUX_EN | (ch5<<3);
    // select channel
    SetPortVal(control_reg, control_word5, 1);
     /*SELECT THE ENCODER CHANNEL USING THE MUX */
    SetPortVal(ENC_CONTROL, BP_RESET, 1);
    // reset byte pointer
    SetPortVal(ENC_CONTROL, TRSFRCNTR_OL, 1);
    // latch the data
    GetPortVal(ENC_DATA, low_byte5_p, 1);
    // least significant byte
    low_byte5=low_byte5&0xff;
    GetPortVal(ENC_DATA, mid_byte5_p, 1);
    mid_byte5=mid_byte5&0xff;
    low_word5 = low_byte5 | (mid_byte5 << 8)&0xffff;
    GetPortVal(ENC_DATA, high_byte5_p, 1);
    // most significant byte
    high_byte5=high_byte5&0xff;
    high_word5 = high_byte5&0xffff;

    if(high_word5 & 0x80) high_word5 = high_word5 | 0xff00;
     /*convert to signed 32 bit*/
    y0[5] = ((uint32_T)high_word5 << 16) | low_word5;

/*Encoder 6*/

    if (u0[0] == 1)  /*reset encoder6*/
    {
        control_word6 = CONTROL_MUST | (ch6<<3);
        // select channel and enable mux start S and H
        SetPortVal(control_reg, control_word6, 1);
         /* select the encoder channel */
            /*initialize the ENCODER CHIP*/
        SetPortVal(ENC_CONTROL, EFLAG_RESET, 1);
        // reset E bit of flag register
        SetPortVal(ENC_CONTROL, BP_RESET, 1);
        // reset byte pointer (x and y)
        SetPortVal(ENC_DATA, CLOCK_DATA, 1);
        // FCK frequency divider
        SetPortVal(ENC_CONTROL, CLOCK_SETUP, 1);
         // transfer PR0 to PSC (x and y)
        SetPortVal(ENC_CONTROL, INPUT_SETUP, 1);
         // enable inputs A and B (x and y)
        SetPortVal(ENC_CONTROL, QUAD_X4, 1);
        // quadrature multiplier to 4 (x and y)
        SetPortVal(ENC_CONTROL, CNTR_RESET, 1);
         // reset counter (x and y)
    }

    /*read data encoder6*/
    control_word6 = CONTROL_MUST | AD_MUX_EN | (ch6<<3);
     // select channel
    SetPortVal(control_reg, control_word6, 1);
     /*SELECT THE ENCODER CHANNEL USING THE MUX */
    SetPortVal(ENC_CONTROL, BP_RESET, 1);
     // reset byte pointer
    SetPortVal(ENC_CONTROL, TRSFRCNTR_OL, 1);
     // latch the data
    GetPortVal(ENC_DATA, low_byte6_p, 1);
     // least significant byte
    low_byte6=low_byte6&0xff;
```

```
    GetPortVal(ENC_DATA, mid_byte6_p, 1);
    mid_byte6=mid_byte6&0xff;
    low_word6 = low_byte6 | (mid_byte6 << 8)&0xffff;
    GetPortVal(ENC_DATA, high_byte6_p, 1);
     // most significant byte
    high_byte6=high_byte6&0xff;
    high_word6 = high_byte6&0xffff;

    if(high_word6 & 0x80) high_word6 = high_word6 | 0xff00;
    /*convert to signed 32 bit*/
    y0[6] = ((uint32_T)high_word6 << 16) | low_word6;

/*Encoder 7*/

    if (u0[0] == 1)  /*reset encoder7*/
    {
        control_word7 = CONTROL_MUST | (ch7<<3);
         // select channel and enable mux start S and H
        SetPortVal(control_reg, control_word7, 1);
         /* select the encoder channel */
            /*initialize the ENCODER CHIP*/
        SetPortVal(ENC_CONTROL, EFLAG_RESET, 1);
         // reset E bit of flag register
        SetPortVal(ENC_CONTROL, BP_RESET, 1);
        // reset byte pointer (x and y)
        SetPortVal(ENC_DATA, CLOCK_DATA, 1);
         // FCK frequency divider
        SetPortVal(ENC_CONTROL, CLOCK_SETUP, 1);
         // transfer PR0 to PSC (x and y)
        SetPortVal(ENC_CONTROL, INPUT_SETUP, 1);
         // enable inputs A and B (x and y)
        SetPortVal(ENC_CONTROL, QUAD_X4, 1);
         // quadrature multiplier to 4 (x and y)
        SetPortVal(ENC_CONTROL, CNTR_RESET, 1);
         // reset counter (x and y)
    }

    /*read data encoder7*/
    control_word7 = CONTROL_MUST | AD_MUX_EN | (ch7<<3);
     // select channel
    SetPortVal(control_reg, control_word7, 1);
     /*SELECT THE ENCODER CHANNEL USING THE MUX */
    SetPortVal(ENC_CONTROL, BP_RESET, 1);
     // reset byte pointer
    SetPortVal(ENC_CONTROL, TRSFRCNTR_OL, 1);
     // latch the data
    GetPortVal(ENC_DATA, low_byte7_p, 1);
     // least significant byte
    low_byte7=low_byte7&0xff;
    GetPortVal(ENC_DATA, mid_byte7_p, 1);
    mid_byte7=mid_byte7&0xff;
    low_word7 = low_byte7 | (mid_byte7 << 8)&0xffff;
    GetPortVal(ENC_DATA, high_byte7_p, 1);
     // most significant byte
    high_byte7=high_byte7&0xff;
    high_word7 = high_byte7&0xffff;

    if(high_word7 & 0x80) high_word7 = high_word7 | 0xff00;
     /*convert to signed 32 bit*/
    y0[7] = ((uint32_T)high_word7 << 16) | low_word7;

}
```

```
static void mdlTerminate(SimStruct *S)
{
    if ( open_dll == 1)
    {
        ShutdownWinIo();

        open_dll = 0;
    }
}

#ifdef  MATLAB_MEX_FILE
#include "simulink.c"
#else
#include "cg_sfun.h"
#endif
```