Università degli Studi di Padova
Facoltà di Ingegneria

Finito di scrivere il giorno 22 luglio 2010 utilizzando LaTeX 2ε

UNIVERSITÀ DEGLI STUDI DI PADOVA
FACOLTÀ DI INGEGNERIA

—

DIPARTIMENTO DI INNOVAZIONE MECCANICA E GESTIONALE

—

TESI DI LAUREA TRIENNALE IN INGEGNERIA DELL' AUTOMAZIONE

CONTROLLO DI MANIPOLATORE A TRE GRADI DI LIBERTÀ

RELATORE: CH.MO PROF. ING. GIULIO ROSATI

LAUREANDO: ROMERES DIEGO

ANNO ACCADEMICO 2009-2010

ai miei genitori e ai miei fratelli che hanno saputo sopportarmi in questi anni
e ad Alessia...

Indice

Sommario

È stato progettato il controllo di un manipolatore planare a tre gradi di libertà tramite il software *Simulink*©.
Il controllo avviene in tempo reale utilizzando Real-Time Windows Target per migliorare le prestazioni, la simulazione gira ad 1kHz.
Il programma è incentrato sulla realizzazione di una macchina a stati finiti che permette l' elaborazione dei riferimenti di posizione e la gestione delle emergenze.
Lo scambio degli Input e Output con il sistema reale avviene tramite la scheda di acquisizione dati Sensoray 626.
Le traiettorie desiderate per movimentare il robot sono pianificate da un altro calcolatore e comunicate al controllo tramite l' apertura di un socket, che utilizza il protocollo di trasmissione UDP. La frequenza di invio è 100 Hz. Il programma si occupa quindi di interpolare le posizioni ricevute e di mandarle ai motori.
Viene implementato un controllo sia di tipo decentralizzato sia centralizzato.

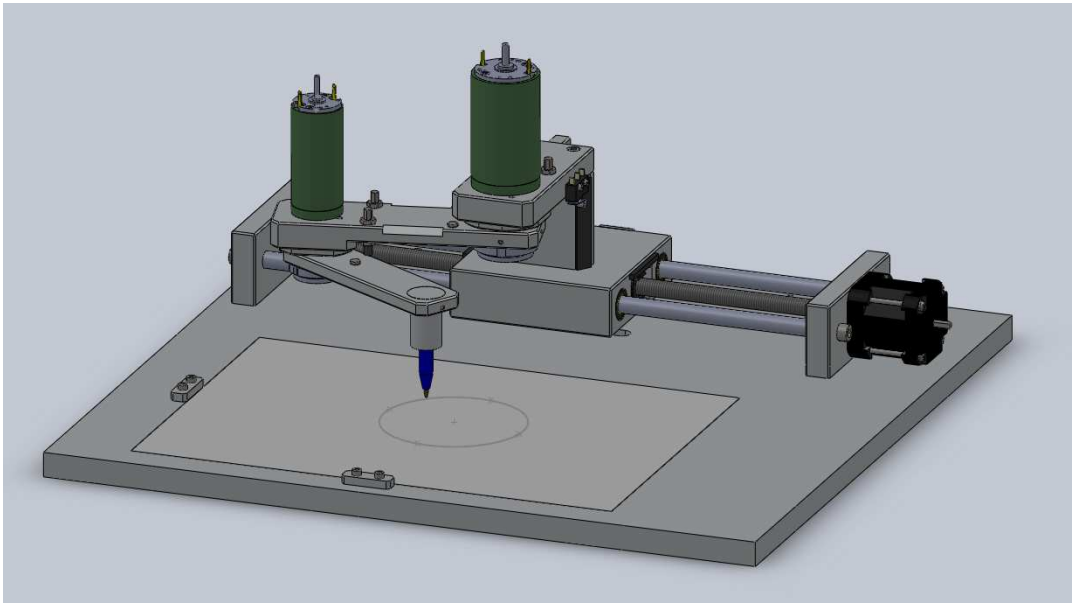

Figura 1:

Immagine del manipolatore realizzato

Introduzione

Il progetto che verrà spiegato nella seguente tesi, tratta della progettazione in *Simulink*© del controllo di un robot, lavoro che unito ad altri quattro progetti ne ha permesso la progettazione, e la realizzazione.
Il lavoro è stato realizzato da cinque studenti del corso di ingegneria dell' Automazione dell' Università di Paodva.
Il robot scelto è un manipolatore scara (Selective Compliant Assembly Robot Arm) a due bracci dotato in aggiunta di una base mobile che si muove lungo una guida lineare.
L' utilizzo di questa tipologia di manipolatore è molto sviluppato in ambito industriale, permette infatti qualsiasi movimento sul piano orizzontale.
La presenza della base mobile rende il robot a tre gradi di libertà, portando così una rindondanza nei movimenti che ne ha permesso lo studio per arrivare ad un' ottimazzazione.
L' obbiettivo del progetto è di realizzare il robot in modo che sia in grado di tracciare delle lettere o delle traiettorie attraverso dei punti predeterminati nello spazio operativo costituito da un foglio A4. A tale scopo è stato previsto anche il movimento verticale rigido di una matita, che potrà essere alzata o abbassata. La matita è l' end-effector del robot.
Ecco una foto rappresentativa del tipo di manipolatore:

Figura 2:
Foto di un manipolatore scara

Obiettivi

All' interno della realizzazione del robot, questo progetto di tesi ha come obbiettivo il controllo del movimento del robot, e la gestione delle emergenze.
Si vuole progettare una macchina a stati finiti che permetta di calcolare i riferimenti di posizione corretti durante il funzionamento normale, durante la calibrazione e durante le emergenze.
Per mantenere i motori controllati, si vuole implementare un controllo di posizione sia di tipo centralizzato sia di tipo decentralizzato.
Per ottenere questi obbiettivi importante è la gestione degli input e output del sistema reale, e la comunicazione con un altro calcolatore da cui vengono ricevute le traiettorie pianificate.

Organizzazione della tesi

La tesi è suddivisa in quattro capitoli.
Nel primo capitolo viene illustrato il robot, e come è stato organizzato e suddiviso il lavoro.

Nel secondo capitolo viene invece spiegata l' architettura del controllo, cioè qual è il procedimento generale per effettuare il controllo, come avviene la sua iterazione con la parte elettronica e meccanica del robot. E la teoria e le tecniche che sono state utilizzate per progettare il controllo sulla movimentazione.
Nel terzo capitolo viene spiegato nel dettaglio il programma Simulink. Quindi viene spiegato come sono state implementate nella pratica la teoria e le idee per lo sviluppo del controllo .
Nel quarto capitolo vengono descritti i test effettuati durante la progettazione.

Capitolo 1

Progetto

In questa capitolo verrà descritto il robot realizzato e l' idea generale del lavoro svolto per arrivare al suo funzionamento.
Descrivendo il lavoro svolto dagli altri quattro tesisti che hanno collaborato al progetto.

1.1 Descrizione del robot

Il manipolatore realizzato è uno SCARA, acronimo di Selective Compliant Assembly Robot Arm, a due membri, dotato in aggiunta di una base mobile, che permette il movimento del manipolatore lungo un' asse lineare.
Il manipolatore grazie all' aggiunta del terzo link è quindi a tre gradi di libertà.
L' ausilio della base mobile, porta ad una ridondanza nei movimenti, infatti è possibile raggiungere uno stesso punto dello spazio operativo con l' end effector in diversi modi. Questo ha portato ad un' analisi per l' ottimizzazione del movimento per raggiungere le posizioni desiderate utilizzando tutti e tre link. Inoltre viene superato il problema di alcune posizioni singolari che verrebbero a crearsi.
L' end-effector è una matita, che può essere alzata o abbassata e permette di scrivere nello spazio operativo.
Viene ora mostrata un' immagine che schematizza il manipolatore reale. Figura 1.1.
I giunti del manipolatore quindi sono:

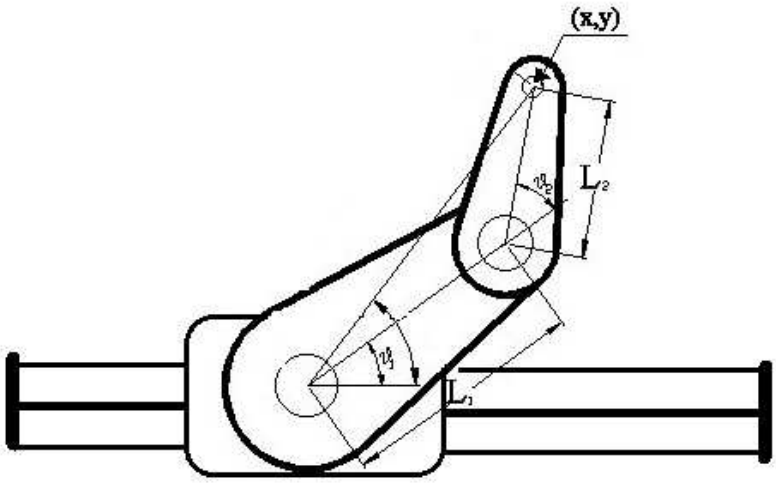

Figura 1.1:
Schematizzazione del robot

- il link0 che verrà anche chiamato "carrello" o "base mobile" a cui verrà associata una posizione lineare "x" che indicherà la posizione assoluta lungo la vite a ricircolo di sfere espressa in numero di passi o mm. Le dimensioni del link sono 100 mm * 100 mm.
  La lunghezza della vite verrà espressa o col numero di passi necessari alla base mobile per percorrere tutta la guida, considerando il finecorsa induttivo a sinistra il passo zero, e il fine corsa destro l' ultimo passo per un totale di 3000 passi, o in mm per un totale di 300 mm da induttivo a induttivo.

- il link1 a cui è associata la posizione angolare $\theta 1$ (vedere figura), positiva per una rotazione in senso antiorario e il cui zero corrisponde al link posto

lungo l' asse delle ascisse.

- il link2 a cui è associata la posizione angolare $\theta 2$ (vedere figura), positiva per una rotazione in senso antiorario ma è una posizione relativa,perché dipende anche dalla posizione del link1.

1.2 Descrizione dei cinque progetti

Per dare al lettore un' idea del lavoro generale necessario al completo funzionamento del manipolatore, vengono elencate e descritte brevemente qui di seguito tutte e cinque le parti in cui è il progetto è stato diviso, che corrispondono a cinque progetti di tesi.

Progettazione e costruzione del parte meccanica del manipolatore

Questo progetto di tesi consisteva nella scelta della componentistica di tutta la parte meccanica, dai motori al materiale da utilizzare per i link, nel disegno in tre dimensioni del robot utilizzando SolidWorks e nella costruzione della parte fisica.
Sono stati scelti due motori in continua per la movimentazione dei link 1 e 2 ed un motore passo per la movimentazione della base lineare.
La scelta si è effettuata principalmente considerando peso, dimensioni e la coppia erogabile dal motore.
Implementando un programma in grado di mostrare la coppia necessaria per effettuare i vari movimenti e considerando i'worst case', è stato possibile decidere quanta coppia devono poter erogare i motori.
Osservando così la coppia massima di picco, la coppia massima continua e la coppia di stallo è stato possibile scegliere i motori corretti per il progetto tra tutti quelli consultati.
Considerando sempre anche il prezzo dei motori per la scelta.
I motori sono stati posizionati all' inizio di ogni rispettivo link, si capisce quindi come la scelta del peso fosse essenziale soprattutto per quello del link2 che au-

menta l' inerzia da spostare. I motori inoltre lavorano in presa diretta non sono previsti quindi motoriduttori.
La lettura della posizione per il link 1 e 2 avviene tramite due encoder incrementali posti sui motori che fungono quindi da trasduttori di posizione.
Il link 0 lavora invece in catena aperta quindi non c'è nessuno strumento che faccia da trasduttore.
Per movimentarlo si è ricorso ad una vite a ricircolo di sfere, che trasformasse il movimento rotatorio del motore in un movimento lungo un unico asse orizzontale.
Si era pensato di utilizzare una guida lineare dove farlo scorrere, ma successivamente si è optato per l'impiego di due sbarre cilindriche.
Inizialmente si era pensato di realizzare i membri del robot in acrilonitrile-butadiene-stirene (ABS), ma successivamente si è deciso di utilizzare l'alluminio, poiché le sue caratteristiche di peso e resistenza si adattavano meglio, infatti essendo i motori posizionati sui link bisognava rendere i link più leggeri e resistenti possibili per compensare.
Successivamente si è passato alla progettazione con SolidWorks© del disegno 3D, in modo da determinare precisamente le dimensioni di ogni componente.
Infine è stato costrutito e montato ogni pezzo.

Progettazione e realizzazione dei driver dei motori in continua

Questo progetto prevede la progettazione e la realizzazione dei driver per i due motori in continua.
La progettazione della scheda elettronica inizia con la scelta delle componenti dalla lettura dei 'data sheet', che ha portato alla scelta dei seguenti integrati principali: microcontrollore dsPic30F4012 e LMD18200 integrato principale dello stadio in potenza.
Successivamente viene fatto lo schema logico utilizzando il software Power Logic, poi vengono progettate le piste con il software Power Pcb e programmato il microcontrollore. Infine viene realizzata la scheda fisicamente.
È stata realizzata una sola scheda elettronica per i driver di entrambi i motori, e pensata in modo che sia esattamente simmetrica, metà per un motore metà per l' altro.

I microcontrollori sono stati programmati in Simulink.
La scheda driver comunica attraverso appositi connettori con la scheda d' interfaccia e quindi con la scheda di acquisizione dati e con i motori e si occupa principalmente della gestione dei riferimenti di tensione per i due motori.

Progettazione e realizzazione del driver dei motore passo

Questo progetto prevede la progettazione e la realizzazione del driver per il motore passo, della scheda d' interfaccia e della realizzazione del quadro elettrico.
La progettazione delle due schede elettroniche segue il procedimento generale descritto per i driver dei motori in continua.
Il driver è stato realizzato in una scheda elettronica a struttura speculare. I principali integrati della scheda sono: microcontrollore dsPic30F4012, LMD18200 integrato principale dello stadio in potenza.
Il microcontrollore è stato programmato in un linguaggio che si basa sul linguaggio C ma è pensato apposta per i pic. Nel programma è stato pensato l' algoritmo per non perdere passi da fare, e per essere sincronizzato con i segnali ricevuti dalla scheda di acquisizione dati.
La scheda d' interfaccia comprende i connettori per le connessioni con i due driver, per la comunicazione con la scheda di acquisizione dati, e per gli encoder. Si può quindi considerare il centro dello scambio dei dati.
Il quadro elettrico contiene tutta la parte elettronica ed inoltre il circuito di emergenza. Nella parte esterna del quadro sono previsti gli ingressi per i connettori piatti della scheda di acquisizione dati e degli encoder, e una pulsantiera in cui sono previsti i pulsanti di arresto e messa in marcia.

Pianificazione delle traiettorie da inseguire e progettazione di una GUI per l' utente

Questo progetto di tesi consiste nella pianificazione delle traiettorie per tutti i movimenti del robot, e progettare un' interfaccia Matlab, GUI, per l' utente.
Il programma realizzato a tale fine viene fatto girare in calcolatore apposito che

chiameremo calcolatore A.
L' interfaccia è così strutturata:
Nella parte destra è rappresentato lo spazio operativo in cui portà agire il robot, ed è stata creata un' immagine del robot stesso, e l' utente potrà vederne i movimenti in tempo reale.
Infatti è stato previsto che la lettura degli encoder venga comunicata al programma e con questi dati viene rappresentata l' immagine del robot che si muoverà quindi nello spazio operativo rappresentato.
A sinistra sono presenti i menù dell' interfaccia dove è possibile selezionare quali operazioni far fare al robot.
Le possibilità consentite sono: far fare all' end-effector un moto punto punto o un percorso ad n punti, scegliere la configurazione con cui deve muoversi il manipolatore se con gomito sinistro o gomito destro, scegliere la modalità di controllo se centralizzato o decentralizzato e in questo caso anche se utilizzare o no la correzione dell' azione in avanti, decidere la modalità di funzionamento dei driver dei motori in continua se in tensione o in corrente, inoltre sono previste delle possibilità condizionate di mandare il robot in calibrazione e in emergenza.
Per effettuare i movimenti sono state pianificate delle traiettorie, ed il movimento pianificato viene temporizzato ed inviato al controllo attraverso l' utilizzo di un oggetto timer.
La pianificazione effettuata segue una legge con profilo trapezoidale di velocità, o a polinomio cubico, o a polinomio di quinto grado.
Importante è stata anche la scelta di come far muovere il robot, infatti l' introduzione della base con movimento lineare porta ad una ridondanza dei possibili movimenti per raggiungere uno stesso punto. Sono stati allora pensati degli algoritmi di ottimizzazione del movimento per decidere come effettuare il movimento.

Controllo di manipolatore a tre gradi di libertà

Questo progetto è quello che verrà descritto in questa tesi, e quindi lo vedremo nel dettaglio nei prossimi capitoli.
Consiste comunque nella progettazione di un programma utilizzando Simulink

per il controllo del manipolatore.
Quindi prevede la lettura di tutti i dati ricevuti dal sistema fisico, dalla posizione dei link alle possibili emergenze, nell' interpretazione e gestione di tutti i casi in cui si può trovare il robot, nell' invio di tutti i segnali necessari per il movimento sia comandati dall' utente tramite l' interfaccia, sia quelli previsti all' interno del programma stesso, e nella progettazione del controllo per l' inseguimento dei motori del movimento richiesto.
Il programma viene fatto girare in un calcolatore apposito che viene chiamato calcolatore B.

Capitolo 2

Architettura per il controllo

In questo capitolo verrà descritto come è stato deciso di realizzare il controllo del robot per capire cosa è stato utilizzato e comprendere il funzionamento generale.

Il controllo è stato programmato con il software *Simulink*© che bene si adatta alla realizzazione degli schemi a blocchi. Questo programma viene fatto girare in un primo calcolatore.
Il controllo serve a far inseguire determinate traiettorie ai motori, in modo da farli muovere il più simile possibile al movimento desiderato. La pianificazione di queste traiettorie avviene in un programma *Matlab*© che gira in un secondo calcolatore, dove inoltre è stata programmata una GUI, che rappresenta l' interfaccia per l' utente.
Da ora chiameremo il calcolatore in cui gira il programma *Matlab*© calcolatore A, e quello in cui gira il programma *Simulink*© calcolatore B.
Nel B è stata installata la scheda di acquisizione dati Sensoray 626, che permette l' invio dei dati dal calcolatore alla parte fisica e viceversa.
La Sensoray 626 è collegata tramite appositi connettori ad una scheda elettronica denominata interfaccia, che funge da collegamento con le schede elettroniche dei driver e i motori.
Lo schema di quanto descritto è il seguente. Figura 2.1.

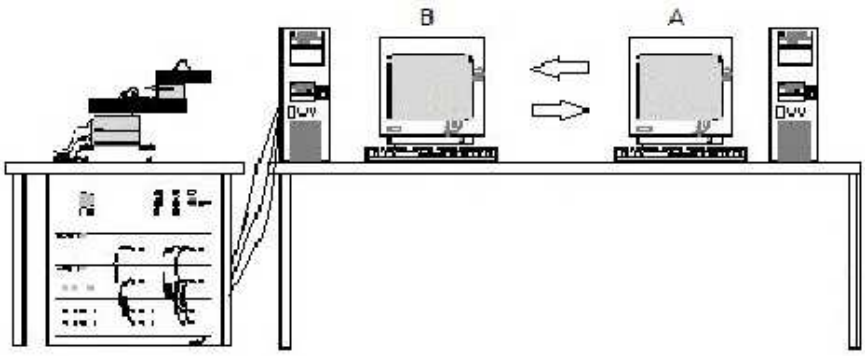

Figura 2.1: Schema dell' architettura del controllo. Nel calcolatore A gira il programma Matlab dove vengono pianificate le traiettorie. Nel calcolatore B gira il programma Simulink dove avvengono i calcoli del controllo.

2.1 Comunicazione Matlab Simulink

I due calcolatori collegati in rete, comunicano tra di loro per lo scambio dei dati utilizzando il protocollo UDP (User Datagram Protocol).
È stato preferito il protocollo UDP al TCP in quanto i dati trasmessi contemporaneamente sono di quantità ridotta, e il sistema UDP è più rapido e si adatta meglio a comunicazioni in tempo reale.
Il protocollo TCP è comunque più affidabile e garantisce l' arrivo dei dati al destinatario, non è così per l' UDP e si sono dovuti implementare delle verifiche di ricezione dei dati per compensare la perdita dei pacchetti.
Gli standard del protocollo di trasporto non specificano come questo debba interagire con gli applicativi. Avviene tramite un' interfaccia detta Socket in cui è necessario inserire l' indirizzo IP dell' host destinatario per inviare i dati e nel nostro caso siccome si vogliono sia inviare sia ricevere i dati in entrambi gli host in ognuno si inserirà l' indirizzo IP dell' altro.
Questo indirizzo serve per inviare i dati all' altro calcolatore, ma per arrivare alla

specifica applicazione è necessario inserire anche la porta di comunicazione, ogni programma avrà la propria che prende il nome di Local port che è quella in cui si ricevono i dati, la Local port dell' altro programma prende il nome di 'Remote port' ed è dove vengono inviati i dati.
I dati vengono inviati da Matlab a Simulink ad una frequenza di 100 Hz grazie alla creazione in Matlab di un oggetto 'timer' che temporizza i vettori da inviare, ed invia un elemento per ogni vettore ogni 0.01 secondi.
Purtroppo l' invio dei dati con quell' intervallo temporale non corrisponde ad una ricezione analoga, sia per i problemi del protocollo UDP accennati precedentemente sia perché l'oggetto timer scrive i dati in un buffer e capita che per problemi si accumulino e vengano inviati poi con delta temporali di appena 0.001 secondi.
L' invio dei dati da Simulink a Matlab avviene invece a 25 Hz, questo perché non è necessario ai fini dell' utilizzo che ne farà Matlab avere una frequenza maggiore. In questo caso l' invio avviene più semplicemente infatti come vedremo meglio nel prossimo capitolo, esiste già un blocchetto apposito per l' invio dei dati.
Il programma Matlab riceve i dati con una funzione recive che viene chiamata da un' altro oggetto timer alla frequenza di 25 Hz.
Dato che il calcolatore A comunica tramite la GUI con l' utente, e il calcolatore B tramite la Sensoray 626 al robot, la comunicazione tra i due è fondamentale, sia per far muovere correttamente il manipolatore sia come feedback per l' utente stesso.
Infatti tramite la GUI viene data la possibilità all' utente di far muovere il manipolatore, a seconda del movimento richiesto vengono pianificate le specifiche traiettorie e inviate al calcolatore B che invia i riferimenti di posizione ai driver che mandano i segnali di tensione ai motori per effettuare il movimento, tramite gli encoder posti sui motori stessi passando attraverso la Sensoray 626 arriva al controllo un feedback di posizione con cui può correggere l' errore di posizione. Questo feedback di posizione viene inviato anche a Matlab che con le posizioni così ottenute può rappresentare il manipolatore sulla GUI secondo i suoi movimenti reali, e per questo motivo se l' aggiornamento del disegno sulla GUI avviene ogni 0.04 secondi è più che sufficiente per far apparire il movimento continuo all'

occhio umano.

2.2 Layout generale

Il programma Simulink prevede dei blocchetti appositi per l' invio dei segnali voluti alla scheda di acquisizione, da cui escono dei segnali fisici di tensione, i blocchetti si chiamano 'Analog Output' e 'Digital Output'.
La scheda di acquisizione riceve poi i segnali del sistema reale, che sono leggibili nel programma tramite i blocchetti 'Analog Input', 'Digital Input' ed 'Encoder Input'.
Quindi questa scheda è il collegamento tra il sistema reale ed il sistema virtuale. Da questa partono due connettori piatti da 50 pin, uno per gli ingressi e le uscite analogiche e uno per gli ingressi e le uscite digitali, ed uno da 26 per la lettura encoder.
Questi connettori arrivano al quadro elettrico e vanno a collegarsi ad una scheda d' interfaccia, è questa scheda elettronica che permette poi l' indirizzamento dei segnali nei corretti punti di applicazione.
Queste due schede elettroniche sono quindi fondamentali per la gestione degli ingressi e delle uscite, e meritano di essere spiegate più nel dettaglio.

2.2.1 Sensoray 626

Questa scheda di acquisizione installata nel calcolatore B, è il mezzo che permette la comunicazione tra il modello reale del motore e il programma virtuale in *Simulink*©. Figura 2.2.
L' installazione è risultata essere molto semplice, è stata infatti solamente installata fisicamente la scheda all' interno del computer nell' apposita slot e fissata con una vite. Inoltre per il suo funzionamento non è stato necessario installare alcun driver aggiuntivo, una volta inserita la scheda era perfettamente leggibile e utilizzabile da Simulink.
Alla scheda sono stati collegati due connettori piatti da 50 pin, uno per gli ingressi

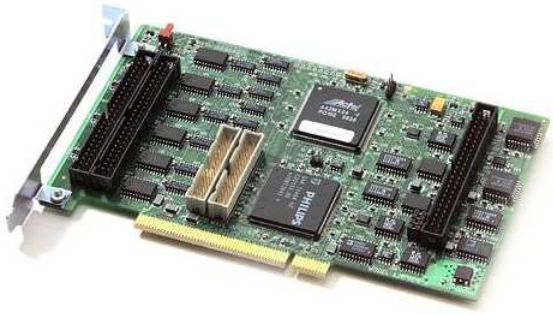

Figura 2.2: Foto della scheda di acquisizione dati, Sensoray626

e le uscite digitali, uno per gli ingressi e le uscite analogiche, ed un connettore piatto da 26 pin per la lettura degli encoder, secondo le caratteristiche della scheda questo connettore può leggere fino ad un massimo di tre encoder.
La scheda utillizzata è la revisione D, e ricordando che la versione Matlab utilizzata è la R2010a si è potuto constatare un ottima compatibilità. Anche il funzionamento in tempo reale del programma utilizzando Real Time Windows Target non ha causato nessun problema con l' utilizzo della scheda.
Le principali caratteristiche della scheda, tratte dal manuale di istruzioni ufficiale sono [1]:

- 48 canali I/O digitali;
- 20 dei canali I/O digitali hanno riconoscimento di margine e la possibilità di interruzione;
- 7 delle uscite digitali possono essere usate come contatori di overflow;
- gli I/O digitali hanno connettori di tipo standard per l' industria ;
- è dotata di un watchdog temporizzato con la possibilità di scelta di numerosi periodi di reset del PCI bus;
- 6 contatori up/down da 24 bit uniti in 3 coppie, con:
  ingressi che possono essere utilizzati in diversi modi (1x,2x, 4x) dagli ingres-

si degli encoder incrementali, dagli ingressi digitali, dalle coppie di contatori di overflow, dal clock del sistema o dal programma;

- controllo della carica della batteria di riserva Ni-Cad;
- 16 ingressi analogici differenziali con 14 bit di risoluzione, rate 15kHz, e ogni canale può essere programmato per ±5V o ±10V;
- 4 uscite analogiche con 13 bit di risoluazione, rate 20 kHz, programmati a ±10V;

2.2.2 Scheda elettronica d' interfaccia

Questa scheda elettronica raccoglie tutti i connettori collegati alla Sensoray 626, quelli che portano il segnale ai driver e quelli che arrivano dagli encoder, è quindi la scheda che permette il passaggio dei dati a tutte le corrette strutture.
Analizziamola osservando lo schema logico. Figura 2.3.

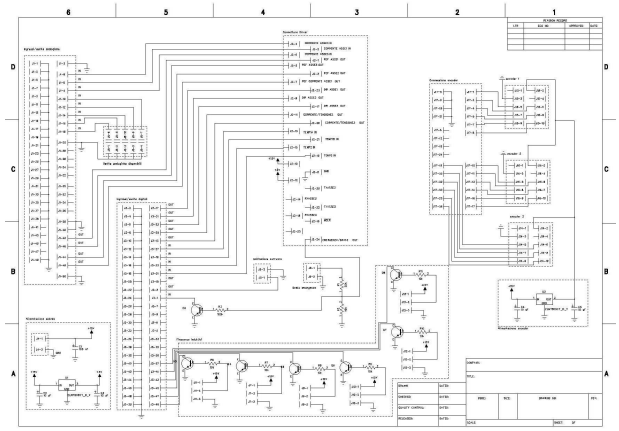

Figura 2.3: Schema elettrico della scheda d' interfaccia

La scheda viene alimentata a 15V, e viene collegata all' alimentatore con gli appositi connettori in posizione (6, A).
In posizione (6, D-C-B) è rappresentato dove viene collegato il connettore piatto da 50 pin degli ingressi e delle uscite analogiche della Sensoray 626, in posizione (5, C-B-A) il connettore piatto da 50 pin degli ingressi e delle uscite digitali.

Tutti gli ingressi e le uscite utilizzati vengono collegati tramite apposite piste ad un bus a 26 vie che fa comunicare la schede con le due schede driver, che è sempre un connettore piatto a 26 pin, posizione (4-3, D-C).
Le uscite analogiche disponibili perché non utilizzate per questo robot sono state portate ad un apposito connettore in posizione (5-6, C).
Il connettore da 26 pin degli encoder viene collegato in posizione (2, D-C). Tiene fino a tre encoder diversi che vengono collegati separati per comodità, e sono stati previsti i collegamenti per tutti e tre anche se ne vengono utilizzati solo 2.
Sotto a questi connettori è prevista l' alimentazione per gli encoder che viene ovviamente collegata a tutti e tre.
I finecorsa induttivi vengono letti in posizione (5-4-3, A-B) e i loro segnali vanno a pilotare la base di un transtor npn, il valore del segnale viene poi letto da collettore e collegati ai connettori della Sensoray 626 come uscite per la scheda d' interfaccia, ed ingressi per scheda di acquisizione. I transistor vengono alimentati dalla resistenza di pull up della Sensoray 626.
Per le emergenze generate dall' elettronica il segnale a 15V viene abbassato a 5V da una rete resistiva, posizione (3,B), e va direttamente a comandare il brake degli LMD che fanno frenare il robot, e va inoltre a pilotare la base di un transistor il cui collettore è collegato ad un ingresso digitale della Sensoray 626. Anche in questo caso l'alimentazione è presa da una resistenza di pull up della Sensoray 626.
L' uscita digitale per le emergenze generate dal controllo va ad un apposito connettore in posizione (4, B), che va a pilotare lo stesso relè delle emergenze volute dall' elettronica, danno quindi lo stesso effetto solo che uno è comandato dal programma Simulink e uno dall' elettronica.
Si può quindi vedere la scheda con questo semplice schema che dà una visione di tutti i suoi collegamenti. Figura 2.4.

Figura 2.4: Foto della scheda di acquisizione dati, Sensoray626

2.3 Teoria del controllo

Il programma per effettuare il controllo del robot, si deve occupare della gestione di tutti i casi in cui si può trovare e questo lo rende molto articolato, ma l' effetto finale che vuole ottenere è quello di garantire il corretto movimento dei motori.
In questo paragrafo vengono descritte la teoria, le tecniche e le idee utilizzate per l' inseguimento dei riferimenti di posizione.
Il link 0 non avendo né un encoder né un potenziometro né un qualsiasi strumento per avere un feedback sulla sua posizione lavora in catena aperta. Quindi vengono mandati i riferimenti al motore passo che eroga delle coppie, ma non si può correggere l' andamento del link e quindi aver certezza che sia stato fatto proprio il movimento desiderato.
L' unica azione di controllo prevista per questo link è quella di mandare un numero di passi da fargli fare inferiore al massimo numero di passi che il motore può effettivamente fare, e verificare sperimentalmente la frequenza di passi che può eseguire senza perderne un numero elevato e cercare di non superala.
Per gli altri due motori che lavorano in catena chiusa, sono stati implementati due tipi di controllo: il controllo decentralizzato e il controllo centralizzato.
Il controllo centralizzato è più complicato da realizzare, necessita di una buona conoscenza del modello dinamico del robot ed è più efficiente in termini di prestazioni.
Il controllo decentralizzato è più semplice da realizzare in quanto non richiede un' approfondita conoscenza del modello ma è anche meno efficace in quanto considera ogni motore a se stante come se non fosse parte del robot. Per migliorare

le sue prestazioni è stata implementata la tecnica dell' azione in avanti. Questa tecnica corrisponde all' aggiunta di una componente di feedforward e può essere a sua volta decentralizzata o centralizzata. Per il calcolo dell' azione in avanti centralizzata occorre però conoscere le coppie che vanno ad agire in ogni giunto, diventa quindi necessario implementare il modello dinamico anche per il controllo decentralizzato.
Prima di passare a descrivere nel dettaglio i due tipi di controllo viene allora effettuata l' analisi dinamica inversa del manipolatore.

2.3.1 Controllo Decentralizzato

L' idea del controllo decentralizzato è quella di controllare ogni link come se fosse un giunto indipendente staccato dal resto del robot.
L' effetto che può avere il movimento del resto del robot sul link in considerazione viene considerato come l' azione di una coppia resistente, come un disturbo quindi non si cerca di compensarne la causa ma solo l' effetto finale.
Un controllo decentralizzato è così strutturato. Figura 2.5 .
Questo schema rende l' idea di come avviene il controllo del robot intero.

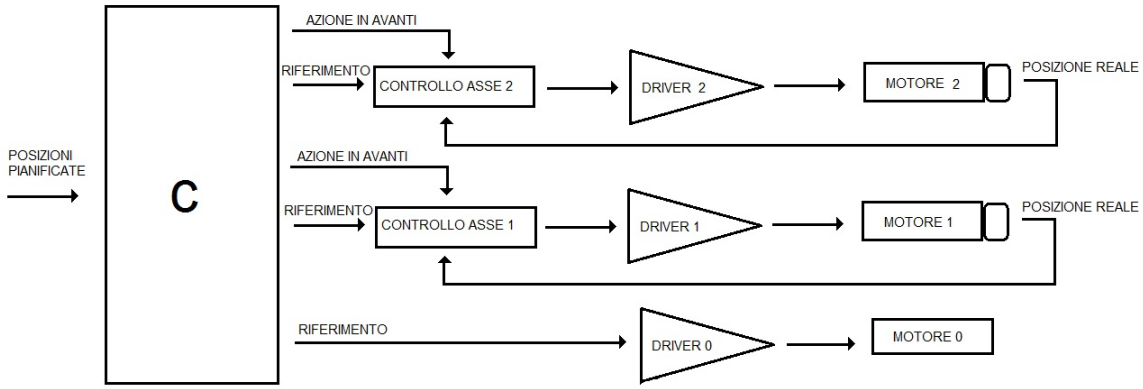

Figura 2.5: Schema di un controllo di tipo decentralizzato

Ogni asse ha il suo controllore specifico che agisce solo per quel membro.

Il motore passo lavora in catena aperta e come si può vedere dall' immagine non ha un feedback di posizione.
Entrando nel maggior dettaglio del controllo, le due retroazioni che si vedono nell' immagine hanno in realtà la seguente struttura generale.Figura 2.6 .
Si tratta di uno schema a retroazione negativa, il cui ingresso è il riferimento di

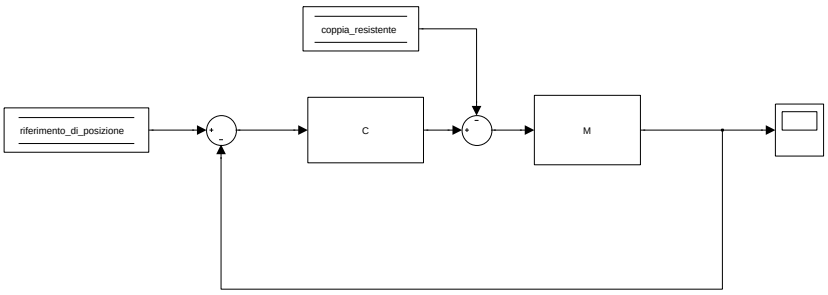

Figura 2.6: Schema generale di un controllo in retroazione

posizione desiderato, C è il controllore da progettare e M il processo da controllare che in questo caso sono i due motori, D è il disturbo che agisce sul sistema. L' uscita è la posizione reale a cui arriva il motore.
Lo scopo del controllore è quello di far corrispondere la posizione reale con quella desiderata, rendendo così l' errore di posizione nullo.
Sono previsti due schemi basati su questo principio nel programma, uno per motore.
Vengono quindi progettati due controllori ognuno specifico per un motore che devono essere in grado di far inseguire al motore il riferimento di posizione che riceve in ingresso.
I controllori utilizzati sono dei PID discreti, semplici da utilizzare in quanto si devono tarare tre parametri sperimentalmente: il guadagno proporzionale, il gua-

dagno derivativo e il guadagno integrativo.
Il problema di questo controllo è che il disturbo, che rappresenta la coppia resistente che agisce sul link, varia e quindi il controllore che non può cambiare dinamicamente può non controllare correttamente il movimento nei tempi desiderati.
Una tecnica per migliorare le prestazioni di questo tipo di controllo è quella di inserire un' azione in avanti.

Azione in avanti

Nel sistema di controllo visto, il controllore agisce solo a fronte di un errore d' inseguimento, altrimenti non produce tensione per prevenire l' errore. Questa tecnica si occupa di aggiungere una tensione che vada a compensare le causa di errore prevedibili.
Prende il nome di azione in avanti perché agisce dopo il controllore.
Può essere calcolata in modo decentralizzato o centralizzato e il calcolo varia a seconda se il driver utilizzato è in tensione o in corrente.
Analizziamo prima il caso con driver in tensione. Figura 2.7
con 'Gv' guadagno del driver in tensione.
Per il calcolo dell' azione in avanti decentralizzata si impone l' equazione:

$$V_{ref,a} \cdot G_v \cdot M(s) = \Theta_r \tag{2.1}$$

e sostituendo la funzione di trasferimento del motore si ottiene:

$$V_{ref,a} = \frac{Kv\dot{\Theta}_r + \frac{Ra \cdot I_{eq} \cdot \ddot{\Theta}_r}{Kt}}{Gv} \tag{2.2}$$

dove il primo termine serve a compensare la forza contro elettro motrice ed è infatti proporzionale alle velocità ed il secondo fornisce la tensione necessaria a far erogare la coppia per il movimento del motore.
Per il calcolo dell' azione in avanti centralizzata si aggiunge anche una componente dovuta alla coppia resistente generata appunto dal movimento degli altri link. L' azione in avanti diventa quindi:

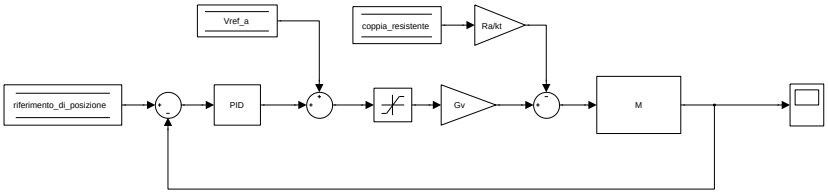

Figura 2.7: Schema di un controllo in retroazione con azione in avanti e driver in tensione

$$V_{ref,a} = \frac{Kv\dot{\Theta}_r + \frac{Ra \cdot I_{eq} \cdot \ddot{\Theta}_r}{Kt} + \frac{Ra \cdot Cr}{Kt}}{Gv} \tag{2.3}$$

Si fa presente che non sono stati considerati rapporti di riduzione e rendimenti perché i motori lavorano in presa diretta e quindi il valore di questi due parametri è 1.

I termini dipendenti da velocità ed accelerazione servono a correggere l' errore nel transitorio, il terzo è l' unico invece che va ad influire nella correzione sul punto finale.

Per i driver in corrente l' azione in avanti si ottiene con lo stesso metodo. Figura 2.8.

Allora l' azione in avanti decentralizzata diventa:

$$V_{ref,a} = \frac{Ki \cdot I_{eq} \cdot \ddot{\Theta}_r}{Kt} \tag{2.4}$$

e l' azione in avanti centralizzata:

$$V_{ref,a} = \frac{Ki(\cdot I_{eq} \cdot \ddot{\Theta}_r + C_r)}{Kt} \tag{2.5}$$

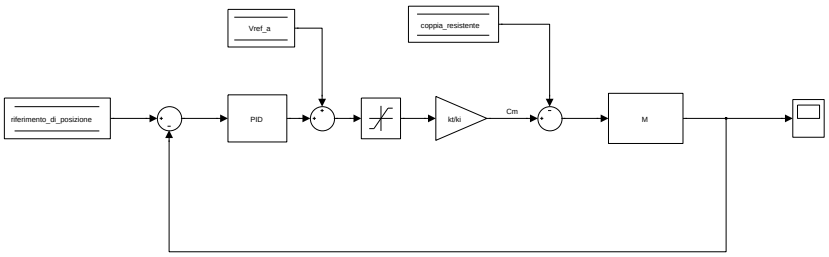

Figura 2.8: Schema di un controllo in retroazione con azione in avanti e driver in corrente

2.3.2 Controllo Centralizzato

Il controllo centralizzato si basa sul principio di controllare ogni link in base anche ai movimenti degli altri.
Infatti su robot a presa diretta come questo è opportuno non considerare dei semplici disturbi gli effetti di accoppiamento dei link, ma compensarli nell' erogazione della coppia del motore.
Per far questo è però necessaria una conoscenza maggiore del modello dinamico del robot. Figura 2.9.
Come si può vedere i feedback di posizione convergono tutti in unica struttura centralizzata, e non ognuno in un controllore decentralizzato, quindi i riferimenti vengono calcolati da una strategia unica che mira ad un effetto globale. In altri termini se ad esempio è previsto che un link stia fermo ma che gli altri si muovano, la coppia erogata su quel link non sarà nulla ma tiene conto degli effetti che gli altri link hanno.
I feedback a disposizione sono di posizione è risultato quindi conveniente realizzare un controllo a dinamica inversa.

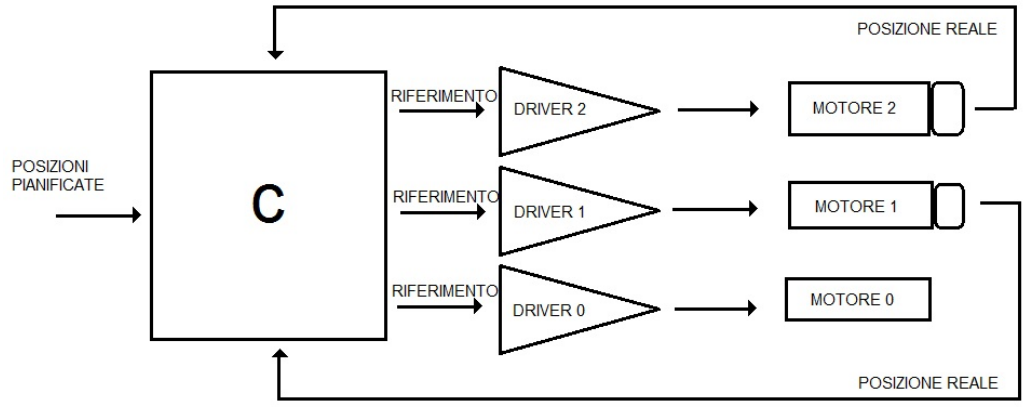

Figura 2.9: Schema del controllo centralizzato del robot

Questo metodo, tratto dallo studio nel libro di testo 'Robotica' [2], si basa sull' individuazione di una legge di controllo funzione dello stato del manipolatore che linearizzi l' intero sistema e che sia stabilizzante.
La legge di controllo diventa:

$$u = B(q) * \ddot{q} + n(q, \dot{q}) \tag{2.6}$$

con u vettore di controllo, B(q) matrice delle inerzie, $\ddot{q}$ vettore delle accelerazioni controllate e n(q,$\dot{q}$) vettore dei termini di Coriolis.
A tal fine è stato calcolato il vettore delle accelerazioni come:

$$\ddot{q} = \ddot{q_d} + K_D(\dot{q_d} - \dot{q}) + K_P(q_d - q) \tag{2.7}$$

con $\ddot{q_d}$ vettore delle accelerazioni desiderate per i motori, $\dot{q_d} - \dot{q}$ vettore degli errori di velocità dei motori, $q_d - q$ vettore degli errori di posizione dei motori e K_D, K_P matrici diagonali.
Le matrici di controllo diagonali rendono il sistema disaccoppiato e quindi ad un errore di posizione di un motore corrisponderà la correzione della componente di accelerazione del motore stesso. Per calcolare i valori dei guadagni di queste matrici si è partiti dai valori teorici:

$$K_D = diag\{2\xi\omega_{n1}, ..., 2\xi\omega_{nn}\} \qquad K_P = diag\{\omega_{n1}^2, ..., \omega_{nn}^2\} \tag{2.8}$$

e poi sono stati tarati sperimentalmente osservando i grafici dell' inseguimento di posizione.
Il vettore delle accelerazioni ottenuto è di dimensioni [2x1] in quanto i motori in catena chiusa da controllare sono 2, ma per il calcolo delle coppie viene aggiunta anche l' accelerazione del link 0 in quanto anche questa va ad influire nel modello dinamico. Non avendo nessun feedback di posizione viene utilizzata l' accelerazione desiderata del link.
Calcolando in modo dinamico grazie alla dinamica inversa le coppie da erogare con queste accelerazioni, e le posizioni e le velocità reali si ottiene un controllo che tiene conto dell' effetto di tutti gli accoppiamenti dei link.
Il problema di questo controllo è però che bisogna conoscere con grande accuratezza i parametri del modello dinamico perché sia un controllo efficiente.
Inoltre potrebbero sorgere problemi perché richiede molti calcoli eseguiti in modo dinamico, e quindi il software potrebbe non riuscire ad eseguirli alla frequenza richiesta. Anche per questo motivo il programma realizzato viene eseguito in tempo reale utilizzando Real-Time Windows Target.

Capitolo 3

Controllo del robot

In questo capitolo verrà spiegato come è stato programmato il controllo, cosa implica 'fare il controllo di un manipolatore' , qual è la logica utilizzata e il significato dei blocchi Simulink.

3.1 Introduzione Simulink

Il software Simulink si basa sull' utilizzo di funzioni rappresentate da dei blocchi già esistenti nelle librerie del software. La scrittura del codice per via grafica lo rende un programma particolarmente adatto alla rappresentazione degli schemi a blocchi.
Il programma realizzato è suddiviso in diversi livelli, infatti tramite i blocchetti 'subsystem' viene creato un nuovo livello in cui possono essere eseguite delle funzioni a parte. Il livello più alto è la schermata iniziale e via via si scende nei livelli inferiori.
In questa facciata sono presenti 8 subsystem del tipo 'Function-Call' dentro i quali saranno presenti altri subsystem e così via.
Il programma viene fatto girare in tempo reale, utilizzando Real-Time Windows Target, per permettere una maggiore velocità di esecuzione e quindi un controllo più veloce.
È stato scelto di far girare la simulazione alla frequenza di 1 kHz.
Per eseguire il programma in tempo reale è necessario impostare i seguenti para-

metri:
selezionare la modalità 'External' invece che 'Normal', nel menù 'Simulation' selezionare 'Configuration parameters' dentro cui nel menù 'Solver' impostare 'Fixed-Step type' al tempo di campionamento desiderato, 0.001 [s],e nel menù 'Real-Time Workshop' impostare 'System target file' a rtwintgt.tlc.
Per eseguire la simulazione è necessario impostare 'Simulation mode' ad 'External' e premere in ordine *Build*, *Connect to Target* e *Start*.
In tutti i blocchetti in cui era necessario impostare il tempo di campionamento è stato messo -1, questo dato indica che viene ereditato il tempo di campionamento dai blocchetti precedenti, se viene fatto così in tutti i blocchetti tutti quanti alla fine erediteranno il tempo della simulazione e quindi 0.001 secondi.
Oltre i blocchetti più comuni è stato necessario ricorrere a diversi altri blocchetti, e viene di seguito spiegato il significato di alcuni fondamentali:

- blocchetto: 'Data Store Memory'. Serve per creare una variabile e inizializzarla, è possibile impostare il nome della variabile e il suo valore iniziale. La variabile può essere poi richiamata con il blocchetto: 'Data Store Read', e sovrascritta con il blocchetto: 'Data Store Write'.

- blocchetto: 'Function-Call generator'. Viene utilizzato insieme al blocchetto 'demux' e al blocchetto 'Function-Call Subsystem' permette di dare un ordine a questi specifici subsystem. Infatti in generale disponendo dei blocchetti in un foglio di lavoro Simulink, non si può essere sicuri dell' ordine con cui verranno eseguiti dal programma. I vari subsystem verranno eseguiti alla massima velocità possibile per la simulazione.

- blocchetto: 'Switch-Case'. Viene utilizzato insieme al blocchetto 'Switch-Case Action Subsystem' ed è comandato da una variabile, quindi ad esempio un 'Data Store Read'. Al variare del valore della varibile questo blocco impone di eseguire il codice dentro ad uno specifico subsystem senza entrare negli altri.

Inoltre è stato realizzato un file Matlab in cui sono stati memorizzati tutti i dati costanti, utili al funzionamento.

Durante la lettura del codice si vedranno spesso delle costanti utilizzate, queste vengono lette dal workspace in cui vengono caricate eseguendo il file m.
Il file si chiama parametri e viene riportato nell' appendice B.
Il programma fatto è il risultato di numerosi tentativi e cambiamenti che hanno portato ad un programma efficiente e abbastanza intuitivo da capire.
Bisogna però premettere che funzionando con un continuo ciclo del codice scritto, spesso si incontreranno delle condizioni imposte per il corretto funzionamento e l' utilizzo di alcune variabili, non possibili da comprendere completamente sul momento, perché verranno spiegate e definite successivamente.
Si avrà quindi alla fine una corretta e completa visione del programma in tutte le sue parti.

3.2 Descrizione del programma

La schermata iniziale, mostra esplicitamente qual è la logica seguita. Figura 3.1.

Notiamo come prima cosa tutti blocchetti Data Store Memory nella parte destra, sono inizializzazioni delle variabili utilizzate nel programma e sono posizionati a questo livello per essere visibili anche in tutti i livelli inferiori, quindi in tutto il programma.
Si vede poi il blocchetto Function-Call Generator che chiama nell' ordine i principali subsystem del programma.
Questi sottosistemi rappresentano le operazioni principali del programma che vengono eseguite al loro interno, e ci danno quindi già un' idea dell logica seguita.
Verranno analizzati nell' ordine dall' alto verso il basso, che è l' ordine con cui vengono eseguiti da Simulink ad eccezione del subsystem 'Calcolo riferimento'. Parte cardine del programma, che se non fosse spiegata subito, renderebbe incomprensibili diverse parti di tutto il resto del programma.

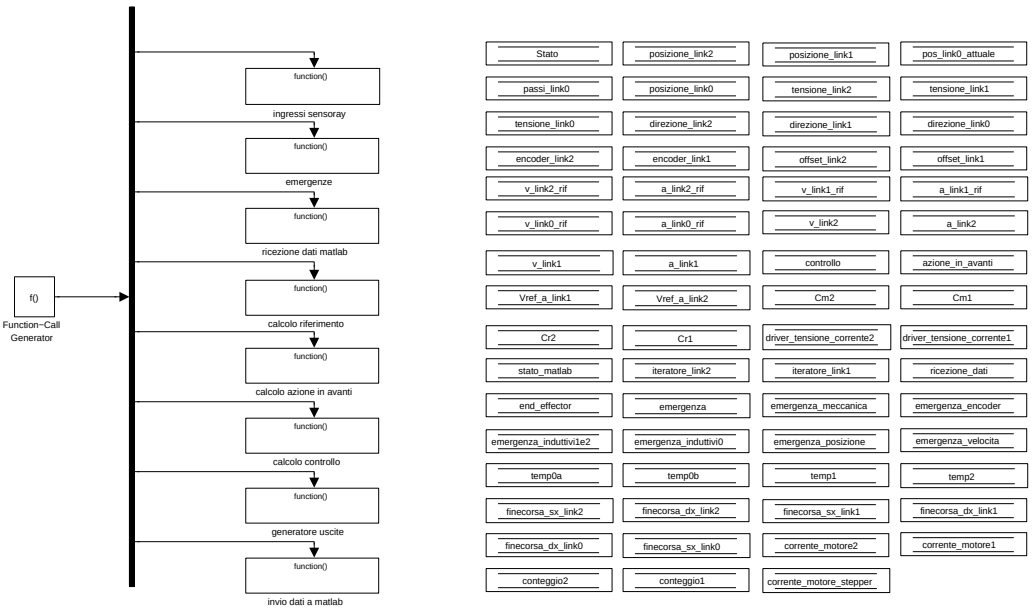

Figura 3.1: Schermata iniziale

3.2.1 Calcolo riferimento

In questo function-call subsystem, viene calcolata la posizione di riferimento per tutti e tre i link, sia nella fase di calibrazione sia in quella di funzionamento normale, che devono poi essere inseguite dal controllo.
Nel suo interno è quindi stata implementata la macchina a stati finiti, con l' utilizzo del blocchetto 'switch-case', già spiegata in via teorica nel capitolo precedente.Figura 3.2

La variabile 'Stato' comanda il passaggio da un case all' altro, diventando quindi una della variabili fondamentali del programma.
Come si può vedere dalla figura sono stati assegnati a dei valori della variabile gli stati previsti:

- Stato = 0 $\Rightarrow$ emergenza, il robot non è abilitato a muoversi;

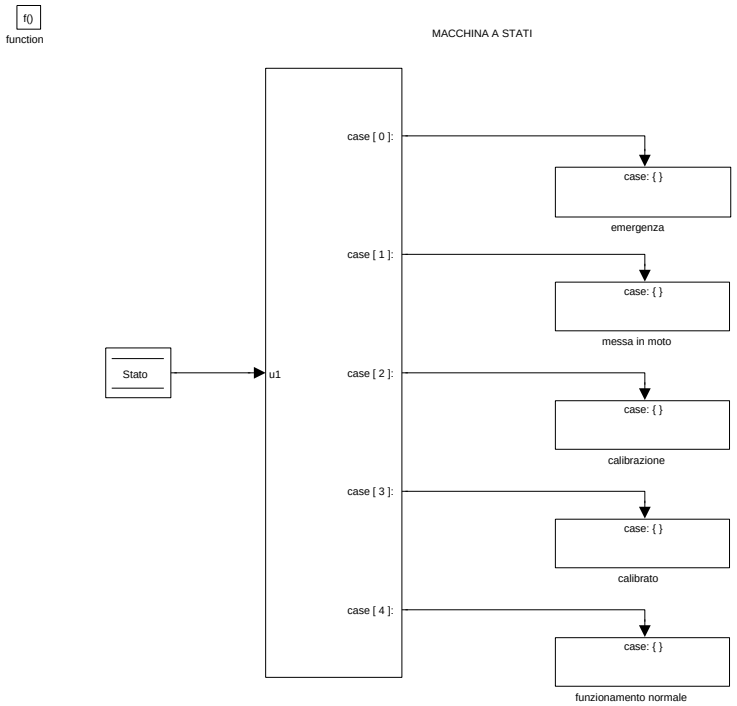

Figura 3.2: Subsystem: Calcolo riferimento. Utilizza una macchina a stati.

- Stato = 1 ⇒ messa in marcia, il robot è pronto per essere calibrato;
- Stato = 2 ⇒ calibrazione, vengono calibrati tutti e tre i link;
- Stato = 3 ⇒ calibrato, il robot ha finito la calibrazione adesso può entrare nel funzionamento normale;
- Stato = 4 ⇒ in funzione, il robot si muove a seconda dei comandi dell' utente;

Stato 0 : emergenza

Quando si verifica un problema nel funzionamento, che discuteremo nel 'Function-call subsystem: emergenze', lo stato viene posto a zero e si passa attraverso questo subsystem, che non modifica direttamente i riferimenti ma come vedremo più avanti impone che la tensione erogata ai motori sia zero e fa entrare l' elettronica

nel circuito d'emergenza.Figura 3.3

Al suo interno viene assegnato il valore 1 alla variabile 'emergenza', che è la

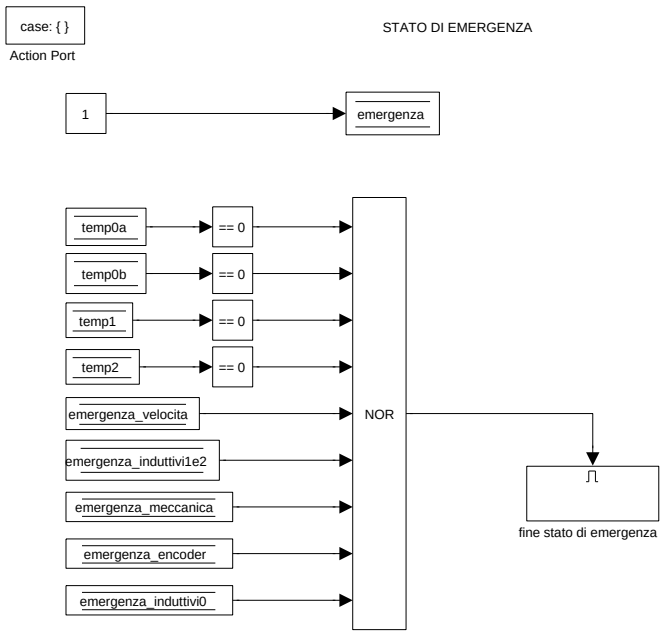

Figura 3.3: Subsystem: Calcolo riferimento ⇒ emergenza

variabile che verrà mandata all' elettronica e che quindi permetterà di entrare nel circuito d' emergenza.

Ed è presente una porta nor con in ingresso tutte le possibili emergenze, che va ad abilitare o no un altro subsystem, la porta garantisce che fino a quando una delle emergenze è attiva l' uscita è a zero quindi il subsystem 'fine stato di emergenza' non viene attivato e si rimane in stato di emergenza.

Durante questo stato, che viene comunicato sia all' elettronica e sia al programma Matlab, il manipolatore viene frenato dall' elettronica fino al bloccaggio e la GUI è disabilitata.

Quando invece non è presente nessuna emergenza ma siamo comunque in questo stato, significa che la fine dell' emergenza è stata letta nel cin questo ciclo di clock,

la porta nor dà come uscita true che convertita in double permette di andare ad abilitare il subsystem 'fine stato emergenza'. Figura 3.4

Viene assegnato il valore 0 alla variabile 'emergenza' per comunicare all' elet-

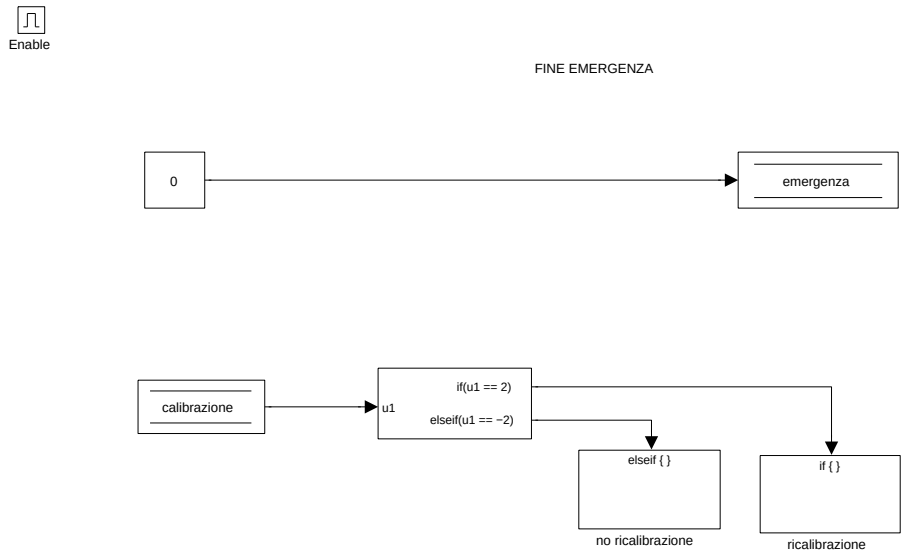

Figura 3.4: Subsystem: Calcolo riferimento ⇒ emergenza ⇒ fine stato di emergenza

tronica che l' emergenza è finita e che per il programma è possibile riprendere il funzionamento normale.

Inoltre si presentano due casi, che sono i due possibili effetti delle diverse emergenze, infatti alcune portano alla ricalibrazione totale del manipolatore, altre invece non alterano la lettura encoder e la posizione del motore passo e quindi una volta terminate si conosce la posizione di tutti e tre i link e non serve quindi ricalibrare. I due casi si distinguono grazie alla variabile 'calibrazione' che come vedremo successivamente 2 è il valore iniziale e significa che il robot deve essere calibrato, -2 che è il valore che indica che il robot è calibrato. L' assegnazione della variabile

avviene quando viene generata ogni emergenza e così si può decidere il suo effetto. Attraverso il blocco if se 'calibrazione' = 2 si passa nel sottosistema 'calibrazione' in cui viene assegnato 'Stato' = 1 e si passa quindi nello stato di attesa per la calibrazione, se 'calibrazione' = -2 si passa nel sottosistema 'no calibrazione' in cui viene assegnato 'Stato' = 3 e si passa quindi nello stato di attesa per il funzionamento normale.

Stato 1 : messa in moto

Questo è uno stato di attesa in cui il robot non è in emergenza ed è pronto per essere calibrato.
Non vengono quindi eseguiti calcoli o operazioni di alcun genere qui dentro. Per entrare nello stato calibrazione, non avviene un passaggio interno al programma Simulink ma comandato dal programma Matlab.
Come vedremo meglio nei Function-call subsystem: 'ricezione dati matlab', ed 'invio dati matlab' la variabile stato viene sempre inviata e ricevuta da Matlab. Quando viene inviato al calcolatore A 'Stato = 1' sull' interfaccia che era disabilitata, perché sicuramente lo stato precedente era quello di emergenza, viene abilitato un pulsante 'Calibrazione'. Quando l' utente decide di premerlo, vuol far partire la calibrazione, allora Matlab invece di inviare di nuovo 'Stato = 1'o invia 'Stato = 2'. Così la macchina a stati può entrare nello stato successivo.

Stato 2 : calibrazione

In questo stato avviene la calibrazione dell' intero robot.
Il problema principale consiste nel fatto che gli encoder quando il robot viene acceso non possono conoscere la posizione assoluta in cui si trova il motore, e segnano sempre 0 come valore iniziale indipendentemente dalla posizione in cui si trova.Figura 3.5
La soluzione utilizzata è stata quella di dare un piccolo incremento di posizione ai link fino ad arrivare ad un finecorsa induttivo, quando questo viene attivato si memorizza la posizione letta dall' encoder. Conoscendo già a priori qual è la posizione assoluta del finecorsa induttivo si ha così l' offset per riportare la posi-

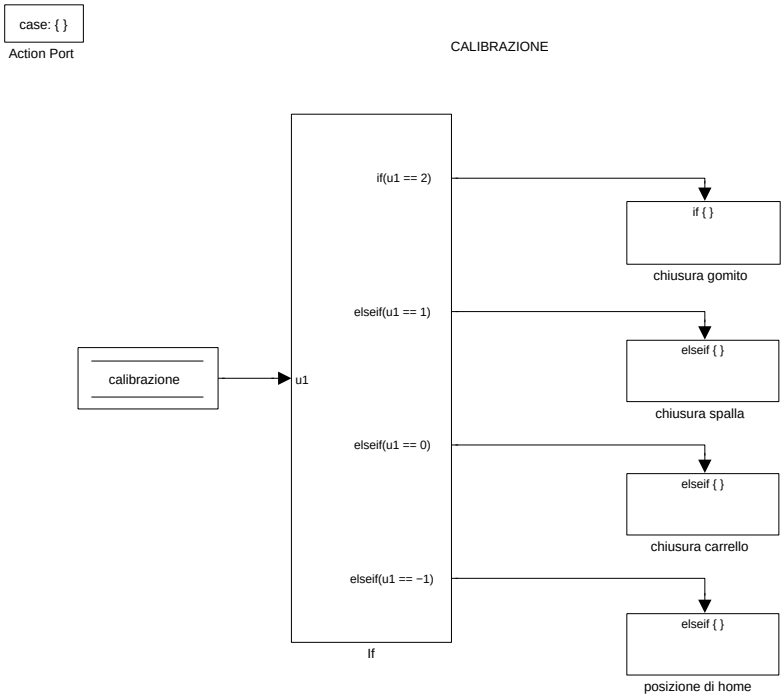

Figura 3.5: Subsystem: Calcolo riferimento ⇒ calibrazione

zione letta dagli encoder a quella assoluta.
Fatta questa operazione per tutti e tre i link, vengono portati contemporaneamente nella posizione di calibrazione detta: 'posizione di home'.
Viene utilizzata una nuova variabile 'calibrazione' che serve a passare dalla calibrazione di un link ad un altro.
Se 'calibrazione = 2' si sta calibrando il link 2. Figura 3.6

Viene dato un incremento a 'posizione_link2' di 0.02° convertito in radianti, rappresentato dalla costante 'incremento_calibrazione' e con i blocchi utilizzati questo incremento viene dato ogni volta alla posizione che è stata incrementata precedentemente, fornendo l' effetto desiderato. La posizione viene incrementata ogni ciclo di simulazione quindi ogni 0.001 [s].
L' incremento è volutamente piccolo perché questa fase non deve dare problemi di controllo e deve essere il più preciso possibile.

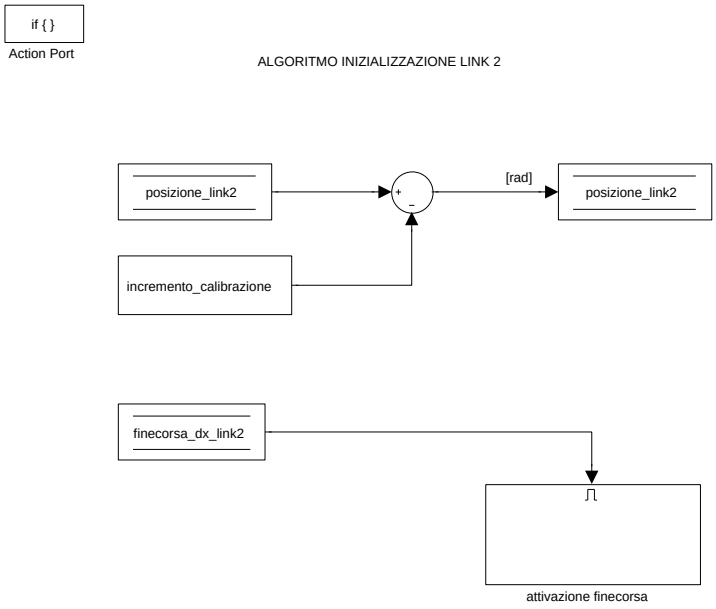

Figura 3.6: Subsystem: Calcolo riferimento ⇒ calibrazione ⇒ chiusura gomito

Come si può notare dalla figura l' incremento di posizione viene sottratto alla posizione, in quanto è stato scelto di chiudere il primo link verso il fine corsa induttivo destro, quindi in accordo con il riferimento di posizione scelto, e quindi con angoli che crescono positivamente in senso antiorario, la pozione deve diminuire.

La combinazione di blocchi successivi serve per fermare la calibrazione, infatti la variabile 'finecorsa_dx_link2' rileva quando il link2 arriva al finecorsa induttivo destro del link 2 e passando quindi da livello logico basso a livello logico alto abilita il subsystem 'attivazione finecorsa'. Figura 3.7.

Al suo interno viene imposto 'calibrazione' = 1 per passare nella fase di chiusura del link 1.

Come precedentemente accennato viene assegnato a 'posizione_link2' il valore noto in cui è stato posizionato il finecorsa induttivo,tramite la costante '-delta_fc_cal_link2'

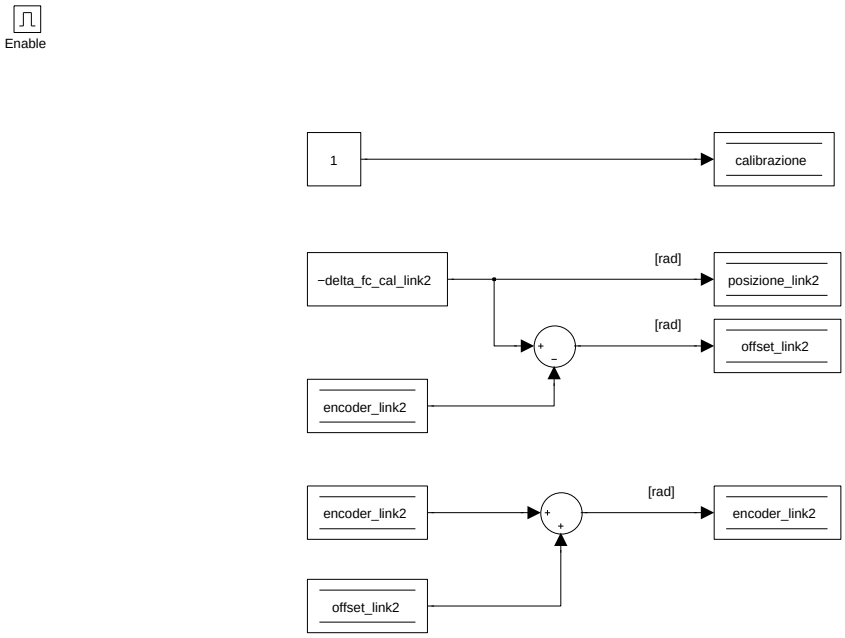

Figura 3.7: Subsystem: Calcolo riferimento ⇒ calibrazione ⇒ chiusura gomito ⇒ attivazione finecorsa

= 155 gradi converiti in radianti, caricata dal file Matlab.
A questa posizione viene sottratto la lettura encoder, 'encoder_link2', e la differenza viene memorizzata nella variabile 'offset_link2'. Questo valore indica quanto la lettura encoder è sfasata dalla posizione reale per il sistema di riferimento considerato.
Il riferimento di posizione ha così subito un gradino di ampiezza '-delta_fc_cal_link2' che non sarebbe accettabile in termini di controllo. Allora 'offset_link2' viene sommato a 'encoder_link2' così anche il valore della posizione reale ha subito numericamente questo incremento e l' errore di posizione, che è quello che viene controllato, rimane invariato.
La posizione del link torna quindi a coincidere con quella del sistema di riferimento considerato.
Terminata quindi questa prima fase si passa alla chiusura del link 1. Figura 3.8.

Il procedimento è analogo al precedente: viene dato lo stesso incremento di

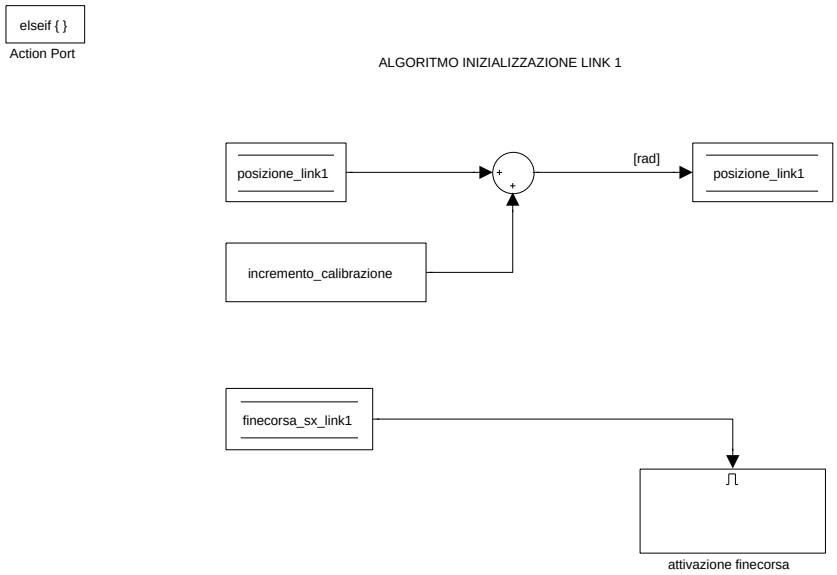

Figura 3.8: Subsystem: Calcolo riferimento ⇒ calibrazione ⇒ chiusura spalla

posizione al link1, ma questa volta viene sommato perché viene chiuso verso il finecorsa induttivo sinistro, viene assegnata la nuova posizione a 'posizione_link1', calcolato l' offset e sommato alla lettura encoder.
'calibrazione' viene impostata a 0.
Si è dunque scelto di far muovere il link 2 tenendo il link 1 chiuso e non portandolo già nella posizione di calibrazione, questo richiederebbe infatti più coppia per mantenere il link 2 in posizione.

Si passa quindi alla chiusura del link 0.
Anche per questo link il procedimento è analogo. Figura 3.9.
Questo link necessita della frequenza con cui fare i passi, e della direzione in cui farli: gli viene dato da fare 1 [passo/ms], quindi la costante 'passi_calibrazione'

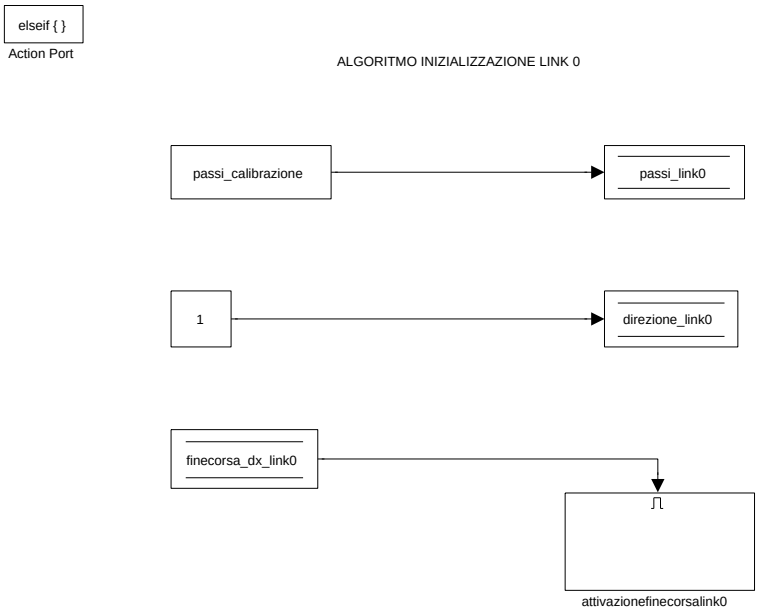

Figura 3.9: Subsystem: Calcolo riferimento ⇒ calibrazione ⇒ chiusura carrello

= 1. Quindi viene fatto muovere il più lentamente possibile per avere maggiore precisione, si perdono meno passi e si arriva esattamente nel primo passo che fa attivare il finecorsa induttivo.

Il sensore del finecorsa induttivo ha infatti ampiezza di circa 5 mm che corrispondono a 50 passi e quindi se i passi fossero maggiori ad 1 si rischierebbe di arrivare ad un passo, che fa attivare il finecorsa, ma non è quello iniziale e causerebbe uno sfasamento di passi considerarlo tale.

Viene inoltre assegnato a 'direzione_link0' il valore 1 che corrisponde ad una direzione positiva, in quanto si è deciso di farlo chiudere sul finecorsa induttivo destro.

Il motore passo ha infatti bisogno oltre alla frequenza di passi da fare anche la direzione in cui farli.

Quando il finecorsa si attiva, la variabili 'finecorsa_dx_link0' abilita il subsystem

enabled 'attivazione finecorsa'. Figura 3.10

non avendo il link 0 un encoder o un potenziometro e lavorando quindi in catena

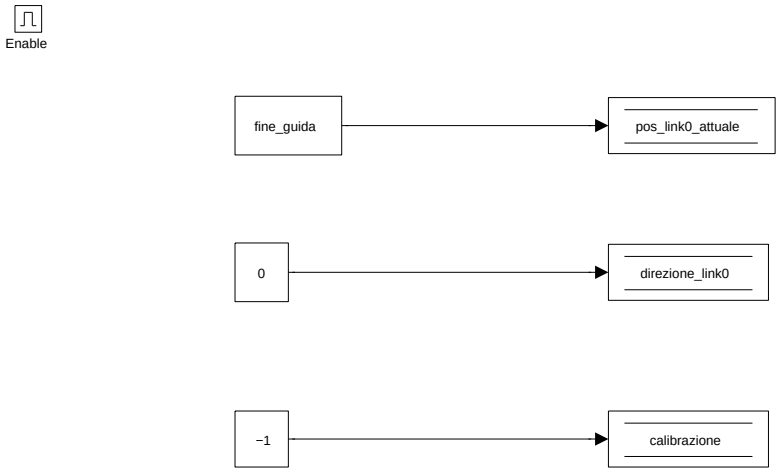

Figura 3.10: Subsystem: Calcolo riferimento ⇒ calibrazione ⇒ chiusura carrello ⇒ attivazione finecorsa

aperta, non esiste il problema che l' errore di posizione in ingresso al controllo non può ricevere un gradino troppo ampio.

Allora è sufficiente sovrascrivere la posizione del link imponendo quella del finecorsa che è salvata nella costante 'fine_guida', da ora in poi la posizione del carrello coinciderà con quella assoluta.

La direzione viene cambiata in 0, perché ora il carrello è arrivato alla posizione massima concessa a destra e dovrà tornare indietro.

A 'calibrazione' viene sovrascritto -1, si può passare alla fase di allineamento.

Tutti e tre i link sono sui finecorsa quindi in una posizione nota è possibile effettuare una traiettoria pianificata per portarli nella posizione di home.

La posizione di home scelta é:

- il carrello viene portato al centro della guida lineare, che corrsiponde al passo numero 1500 e cioè alla posizione lineare 15 cm;

- il link 1 viene portato in posizione perpendicolare alla guida lineare, θ 1 = 90°;

- il link 2 viene allineato con il link 1, θ 2 = 0°.

Tutti e tre i link vengono fatti muovere contemporaneamente. Figura 3.11
Il subsystem è composto di due parti principali, quella più in alto che si occupa

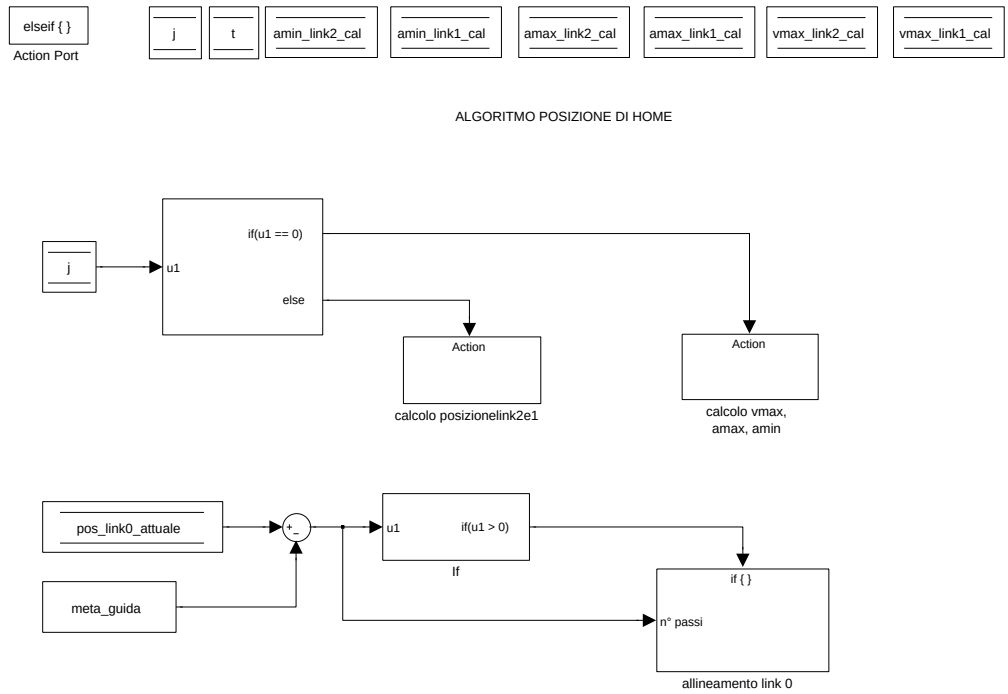

Figura 3.11: Subsystem: Calcolo riferimento ⇒ calibrazione ⇒ posizione di home

di portare nella posizione di home i link 1 e 2, la seconda più in basso che si occupa del link0. Si descrivono queste due parti in modo separato, ma ricordiamo

che vengono eseguite entrambe ad ogni ciclo di simulazione finché il valore di 'calibrazione' è uguale a -1.
Viene ora descitta la prima, quella riguardante la calibrazione dei due link 1 e 2. Si vede un primo subsystem if che dipende dalla varibile 'j'.
Serve per far eseguire il subsystem 'calcolo vmax, amax, amin' solo una volta: 'j' è inizializzata a 0, viene eseguito quindi il subsystem citato, alla fine del quale viene assegnato il valore 1 a 'j' così sempre durante lo stesso ciclo viene eseguito anche il subsystem ' calcolo posizione link 2 e 1 ' e il primo non verrà più eseguito; serve infatti solo a calcolare dei valori costanti che rimarranno invariati. Figura 3.12.
Per il calcolo della velocità massima, dell' accelerazione massima e dell' accele-

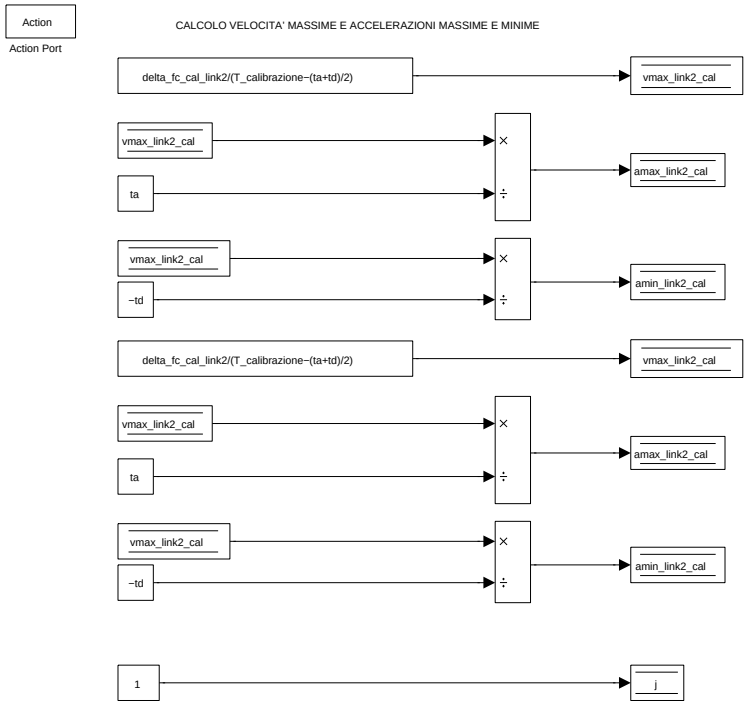

Figura 3.12: Subsystem: Calcolo riferimento ⇒ calibrazione ⇒ posizione di home ⇒ calcolo vmax, amax, amin

razione minima sono state utilizzate la seguenti equazioni:

$$\begin{cases} v_{max} = \frac{q_{fin}-q_{in}}{T_calibrazione-(ta+td)/2} \\ a_{max} = \frac{v_{max}}{ta} \\ a_{min} = \frac{-v_{max}}{td} \end{cases} \tag{3.1}$$

dove T_calibrazione è il tempo totale per cui è stato pianificato il movimento, ta è il tempo di accelerazione e td è il tempo di decelerazione, q_fin - q_in viene sostituito con la costante 'delta_fc_cal_link1' o 'delta_fc_cal_link2' rispettivamente per il primo e secondo membro, che esprime appunto la differenza tra la posizione di calibrazione e la posizione del finecorsa, quindi la posizione iniziale e finale della traiettoria in tensione. Queste variabili vengono caricate dal file Matlab.

I calcoli sono analoghi per i due membri, cambia solo il nome delle variabili utilizzate.

I risultati delle equazioni sono salvati nelle omonime variabili.

Calcolate le velocità massime e le accelerazione massime e minime è possibile pianificare la traiettoria per portare il manipolatore in posizione di home.

Conosciamo infatti la posizione iniziale e la posizione finale, quindi sarebbe stato inadeguato far arrivare i link in posizione di home ancora con dei semplici incrementi.

Il tipo di pianificazione scelto è a profilo trapezoidale di velocità.

Alla variabile 'j' viene assegnato il valore 1, quindi si passa nel subsystem 'calcolo posizione link 2 e 1'. Figura 3.13.

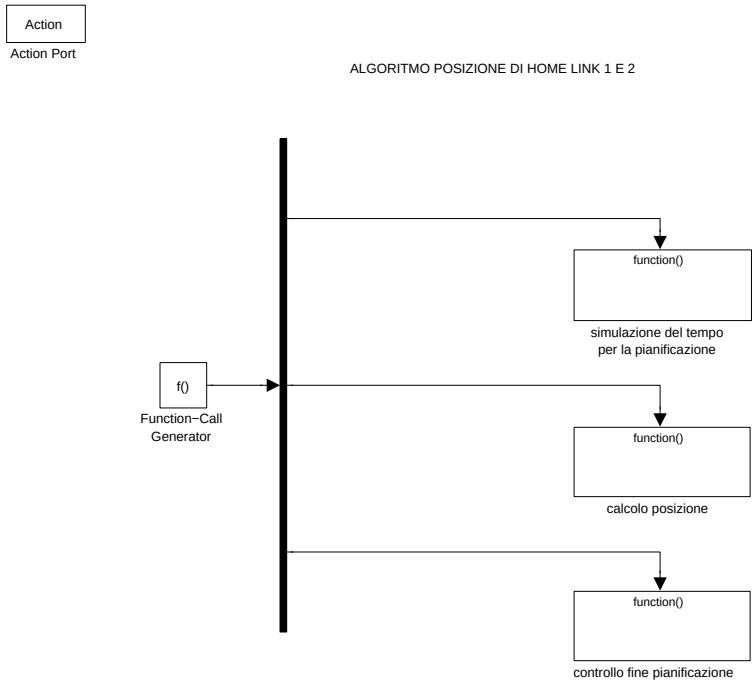

Figura 3.13: Subsystem: Calcolo riferimento ⇒ calibrazione ⇒ posizione di home ⇒ calcolo posizione link 2 e 1

Il sottosistema è caratterizzato da un funnction-call generator che ordina tre sottosistemi per effettuare il calcolo della pianificazione.
La pianificazione scelta è quella a profilo trapezoidale di velocità.
prevede un tempo totale 'T_calibrazione' = 2 [s], un tempo di accelerazione 'ta' = 0.6 [s], un tempo di decelerazione 'td' = 0.6 [s], questi valori vengono letti dal file Matlab.
Nel primo function-call subsystem 'simulazione del tempo per la pianificazione'. Figura 3.14.
viene incrementata la variabile 't' del valore corrispondente ad ogni ciclo di simulazione, quindi questa variabile serve a simulare il tempo che passa a partire da quando inizia la pianificazione, cioè come se la pianificazione partisse all' istante zero. Questo permette i calcoli corretti.

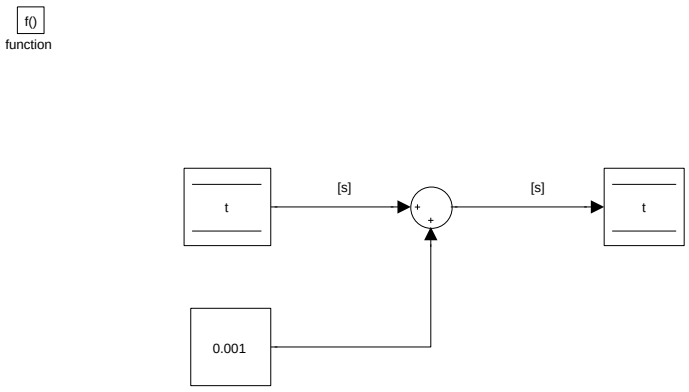

Figura 3.14: Subsystem: Calcolo riferimento ⇒ calibrazione ⇒ posizione di home ⇒ allineamento link2e1 ⇒ calcolo posizione link 2 e 1 ⇒ simulazione del tempo per la pianificazione

Nel secondo function-call subsystem 'calcolo posizione'.
Figura 3.15.
Viene calcolato il riferimento di posizione dei due link, che verrà utilizzato come riferimento per il controllo.
Il calcolo, come si vede, avviene per tre situazioni distinte: la fase di accelerazione, la fase a velocità costante e la fase di decelerazione.
Si entra in una di queste fasi a seconda del valore della variabile 't': se t è minore o uguale del tempo di accelerazione, 'ta' siamo nella prima fase, se 't' è maggiore del tempo di accelerazione ma minore o uguale del tempo totale per il movimento pianificato meno il tempo di decelerazione, '$T_{calibrazione}$' siamo nella fase a velocità

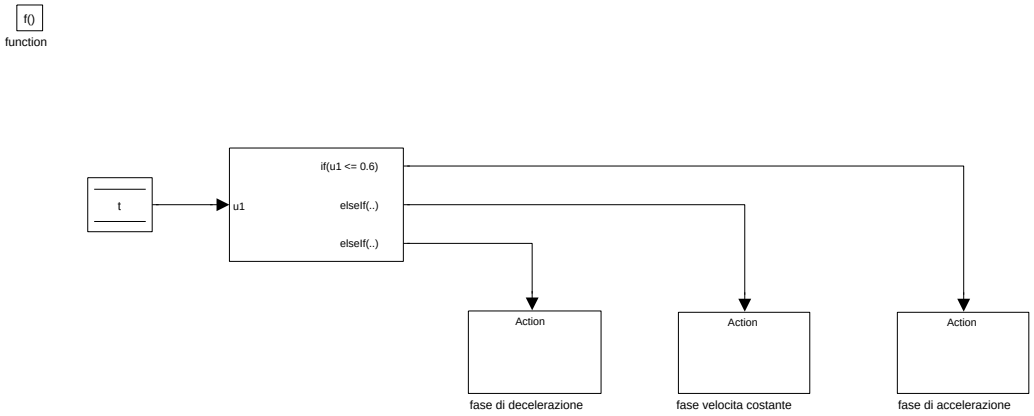

Figura 3.15: Subsystem: Calcolo riferimento ⇒ calibrazione ⇒ posizione di home ⇒ allineamento link2e1 ⇒ calcolo posizione link 2 e 1 ⇒ calcolo posizione

costante, se 't' è maggiore del tempo totale per il movimento pianificato meno il tempo di decelerazione siamo nella fase di decelerazione.
Il calcolo della posizione in fase di accelerazione.
Figura 3.16.
viene dato dall' equazione:

$$\theta(t) = \theta_{in} + 0.5 * a_{max} * t^2 \tag{3.2}$$

in cui θ_{in} corrisponde esattamente a quella del finecorsa induttivo che è il nostro punto di partenza e a_{max} è quella calcolata.
Questo è valido sia per il link 1 che per il link 2, e in questo sottosistema viene il calcolato il riferimento per entrambi.
Utilizzado le variabili corrette le equazioni diventano:

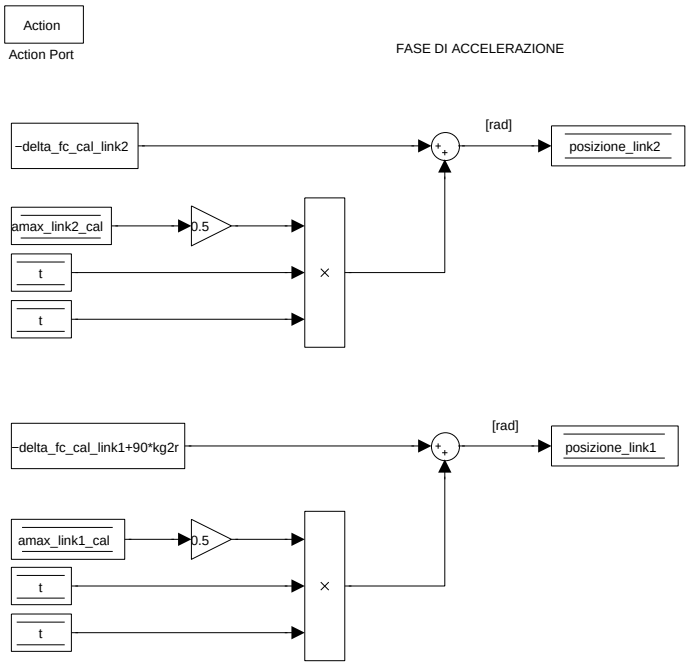

Figura 3.16: Subsystem: Calcolo riferimento ⇒ calibrazione ⇒ posizione di home ⇒ allineamento link2e1 ⇒ calcolo posizione link 2 e 1 ⇒ calcolo posizione ⇒ fase di accelerazione

- posizione_ link2 = -delta_fc_cal_link2 + 0.5*amax_link2_cal*t^2 ;
- posizione_ link1 = -delta_fc_cal_link1+90*kg2r + 0.5*amax_link1_cal*t^2 ;

Ricordiamo che queste equazioni valgono finché: $0 < t \leq ta$.

Quando il tempo diventa maggiore di ta si entra nel sottosistema del calcolo della posizione in fase a velocità costante. Figura 3.17.
viene dato dall' equazione:

$$\theta(t) = \theta_{in} + 0.5 * a_{max} * ta^2 + v_{max} * (t - ta) \tag{3.3}$$

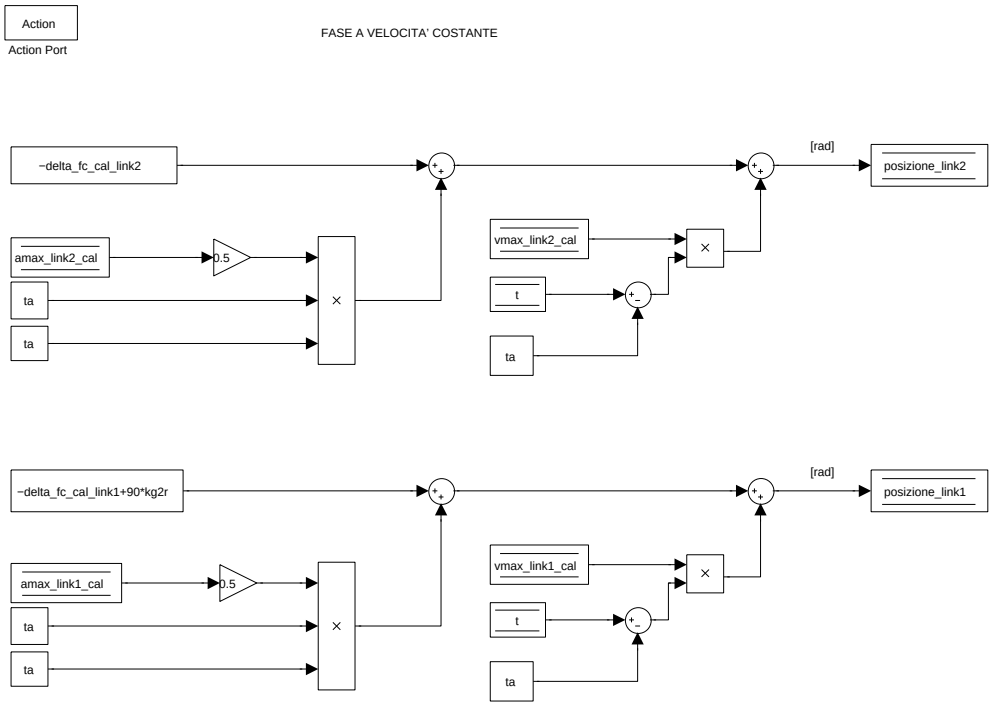

Figura 3.17: Subsystem: Calcolo riferimento ⇒ calibrazione ⇒ posizione di home ⇒ allineamento link2e1 ⇒ calcolo posizione link 2 e 1 ⇒ calcolo posizione ⇒ fase velocita costante

Il riferimento viene calcolato per entrambi i link in questo sottosistema.
θ_{in} e a_{max} sono i medesimi dei precedenti e:

- v_{max} = vmax_link1_cal per il link 1;
- v_{max} = vmax_link2_cal per il link 2;
- ta < t ≤ T_calibrazione - td.

Quando il tempo assume valori maggiori a T_calibrazione - td, si entra nel sottosistema del calcolo della posizione in fase di decelerazione. Figura 3.18
viene dato dall' equazione:

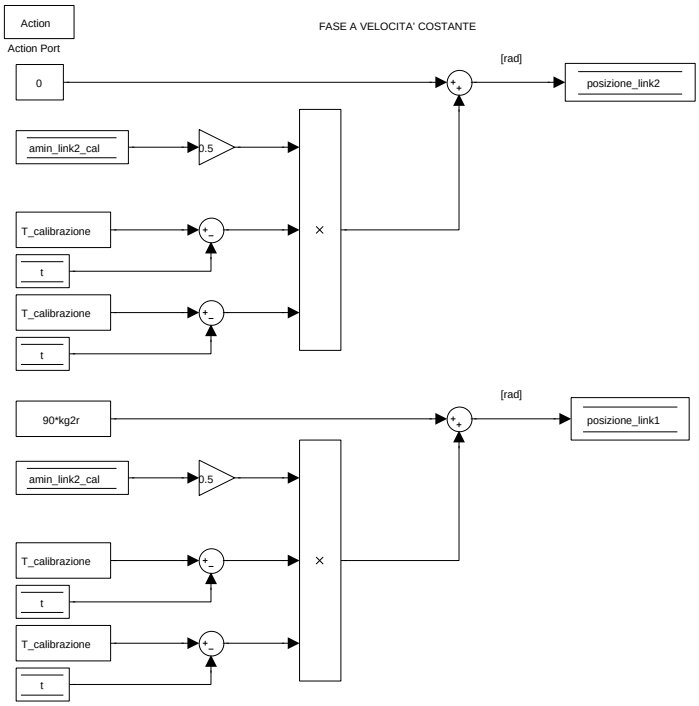

Figura 3.18: Subsystem: Calcolo riferimento ⇒ calibrazione ⇒ posizione di home ⇒ allineamento link2e1 ⇒ calcolo posizione link 2 e 1 ⇒ calcolo posizione ⇒ fase di decelerazione

$$\theta(t) = \theta_{fin} + 0.5 * a_{min} * (T - t)^2 \tag{3.4}$$

Il riferimento viene calcolato per entrambi i link in questo sottosistema.
I valori utilizzati sono:

- θ_{fin} per il link 1 = 90*kg2r che corrisponde alla posizione di home per il membro specifico;
- a_{min} = amin_link1_cal per il link 1;
- θ_{fin} per il link 2 = 0 la posizione di home prevede infatti il secondo link allineato con il primo;

- a_{min} = amin_link2_cal per il link 2;
- T - td < t ≤ T.

Nel terzo function-call subsystem 'controllo fine pianificazione'. Figura 3.19. viene controllato se la variabile 't' è maggiore del valore 2, cioè se il tempo della

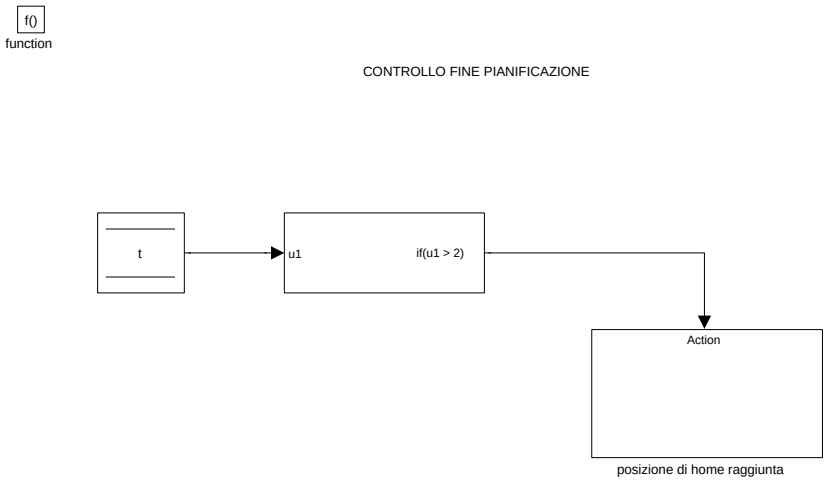

Figura 3.19: Subsystem: Calcolo riferimento ⇒ calibrazione ⇒ posizione di home ⇒ allineamento link2e1 ⇒ calcolo posizione link 2 e 1 ⇒ calcolo posizione ⇒ controllo fine pianificazione

pianificazione supera quello previsto.
Se questo avviene significa che la posizione di home è stata raggiunta, o almeno i riferimenti di posizione per i motori hanno raggiunto quella posizione.
Poi dipende dalla precisione di inseguimento del controllo, l' errore della posizione reale rispetto a quella desiderata.
Quando questa condizione è verificata, viene attivato il subsystem enabled: 'posizione di home raggiunta'. Figura 3.20.

giunti a questo punto il robot è completamente calibrato, possiamo dunque pas-

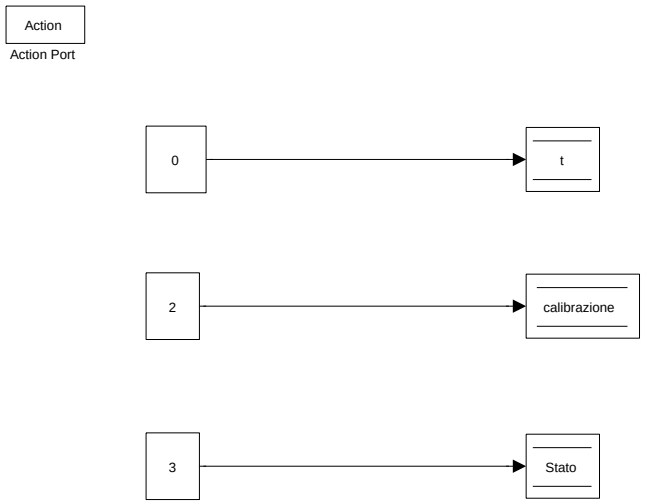

Figura 3.20: Subsystem: Calcolo riferimento ⇒ calibrazione ⇒ posizione di home ⇒ allineamento link2e1 ⇒ calcolo posizione link 2 e 1 ⇒ calcolo posizione ⇒ controllo fine pianificazione ⇒ posizione di home raggiunta

sare allo stato successivo della macchina a stati.
Viene azzerato il valore della variabile 't' nel caso dopo qualche eventuale emergenza si debba ricalibrare il robot.
La variabile 'calibrazione' viene portata al valore -2 che indica che il robot è calibrato.
Quindi le assegnazioni fatte sono:

- 'Stato' = 3;
- 't' = 0;

- 'calibrazione' = -2;

Ricordando che il subsystem 'posizione di home' era diviso in due parti, viene ora descritta la seconda parte quella che si occupa di portare il link 0 nella posizione di home. Si faccia riferimento alla figura 3.11.
Viene posta subito la condizione per controllare se il link 0 è già in posizione di home o no:
alla posizione attuale del link 0 si sottrae il numero di passi a cui corrisponde metà guida,rappresentato dalla costante 'metà_guida' = 1500, che è dove vogliamo far arrivare il carrello. Se la differenza è maggiore di zero vuole dire che la sua posizione è ancora tra il finecorsa induttivo destro e la posizione desiderata e deve essere corretta. Questa condizione continua fino al raggiungimento della posizione di home.
La correzione di posizione avviene all' interno del subsystem 'allineamento link 0', in cui si entra grazie ad un blocchetto if, solo quando la differenza precedente è strettamente maggiore di 0. Figura 3.21.

Viene inviato il numero che risulta dalla sottrazione come numero di passi da fare al controllo, che si occuperà di controllare se il numero di passi richiesti è fattibile dal motore passp in un solo ciclo di clock, e in caso contrario di suddividerli in modo corretto e di aggiornare la posizione del link 0.
Inoltre 'direzione_link0' è posta uguale a zero perché adesso i passi devono essere fatti in direzione negativa, in quanto il carrello deve per forza tornare indietro.
Alla fine della calibrazione dei link 1 e 2, si era detto che il robot era totalmente calibrato e si poteva passare allo stato successivo, affermazione che sembrerebbe non considerare il tempo di calibrazione del link 0.
Ma questo non è esatto.
Il raggiungimento della posizione di home del link 0 è sicuramente inferiore ai 2 sec pianificati per gli altri due link.
Infatti il carrello deve percorrere metà guida che corrisponde a 1500 passi che eseguendoli a 5 passi al millisecondo corrisponde ad un tempo totale di 0.3 sec, con ampi margini di sicurezza per eventuali modifiche temporali.

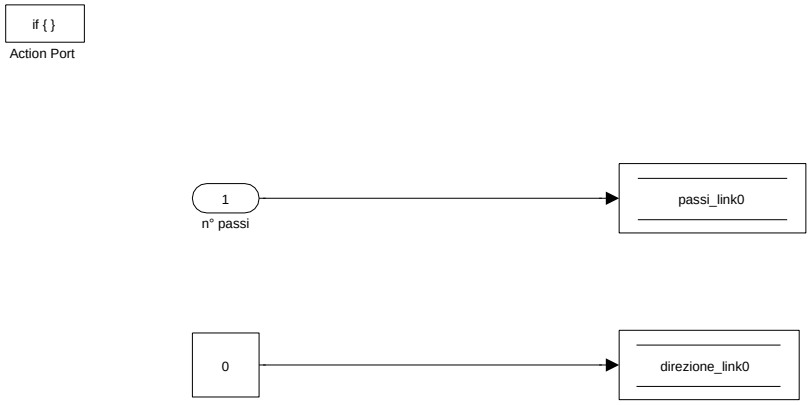

Figura 3.21: Subsystem: Calcolo riferimento ⇒ calibrazione ⇒ posizione di home ⇒ allineamento link 0

I movimenti dei link durante tutta la calibrazione, quindi sia in fase di allineamento ai finecorsa sia durante la posizione di home vengono rappresentati nei seguenti grafici. Figura 3.22-3.23-3.24.

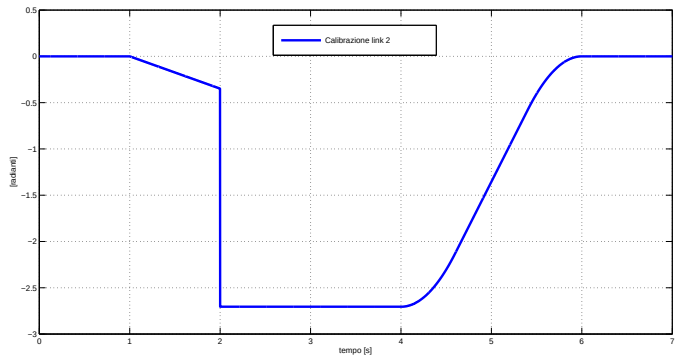

Figura 3.22: Grafico della posizione del link 2 durante la calibrazione

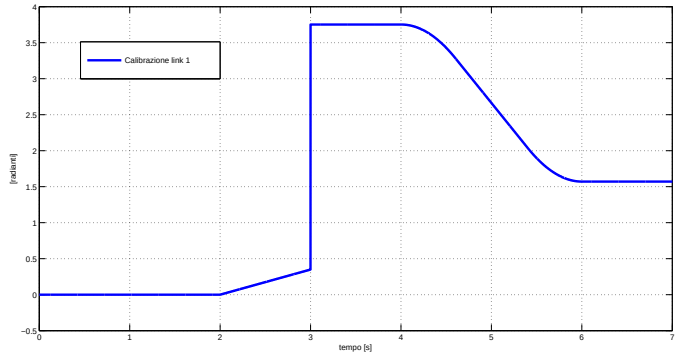

Figura 3.23: Grafico della posizione del link 1 durante la calibrazione

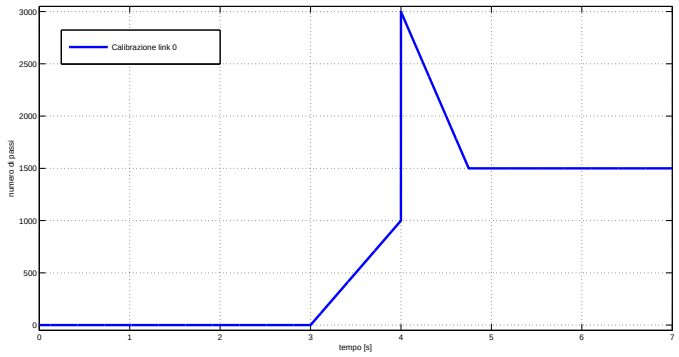

Figura 3.24: Grafico della posizione del link 0 durante la calibrazione

Stato 3 : calibrato

Il robot è calibrato e si entra allora in questo stato, che è uno stato di attesa, infatti non è presente alcun blocchetto al suo interno.
Attende che arrivi da Matlab il passaggio allo stato 4.
Quando infatti al calcolatore A viene inviato 'Stato' = 3, nell' interfaccia disabilitata compare la possibilità per l' utente di premere il pulsante 'start', la cui pressione abilita tutta l' interfaccia, Matlab legge le posizioni dei tre link che sono inviate da Simulink, che corrispondono a quelle di home e disegna il manipolatore con queste posizioni, e viene inviato 'Stato' = 4 al calcolatore B.
L' utente da ora ha la possibilità di far muovere il robot secondo le possibilità previste dal programma Matlab. Ad esempio premendo nello spazio operativo simulato nella GUI, il robot si muoverà per raggiungerlo.
Per quanto riguarda il programma Simulink, quando viene premuto il pulsante 'go' si riceve 'Stato' = 4 la macchina a stati entra nel suo ultimo stato possibile.

Stato 4 : funzionamento normale

Questo stato si può considerare il principale, in quanto viene calcolato il riferimento di posizione per tutti i movimenti voluti dall' utente in condizioni normali.
Il problema principale nasce dal fatto che, come vedremo meglio nel function-call subsystem 'ricezione dati matlab', i dati inviati da matlab vengono spediti a 100 Hz, quindi ogni centesimo di secondo, mentre la simulazione Simulink gira 1 kHz quindi dieci volte più veloce.
Diventa quindi necessario effettuare un' interpolazione tra le posizioni inviate da Matlab. Se così non fosse, si perderebbe il vantaggio di far girare la simulazione sotto Real-Time Windows Target a questa frequenza, inoltre ci sarebbe il rischio di dare come riferimento di posizione dei gradini di ampiezza eccessiva, che porterebbero a gravi problemi di controllo nell' inseguimento.
Inoltre bisogna considerare che non sempre i pacchetti dati inviati riescono ad essere inviati o ricevuti, ma alcuni vengono persi, e quindi bisogna considerare anche questa evenienza nell' algoritmo per l' interpolazione.
La comunicazione tra i due calcolatori c' è da sempre, ma i riferimenti di posizione

che vengono inviati dal programma Matlab, non vengono mai letti fino a quando 'Stato' non diventa = 4. Quando questo avviene vengono rinviate inizialmente a Simulink le posizione di home dei tre link, così il robot rimane nella posizione di calibrazione. Questo continua ad avvenire fino a quando l' utente non darà i comandi di movimentare il manipolatore, allora le posizione ricevute da Simulink cambieranno a seconda della traiettoria pianificata da Matlab, ci sarà l' interpolazione di posizione e verranno inviati i nuovi riferimenti ai motori che faranno muovere il robot.
Questo subsystem è strutturato con un function-call generator. Figura 3.25.

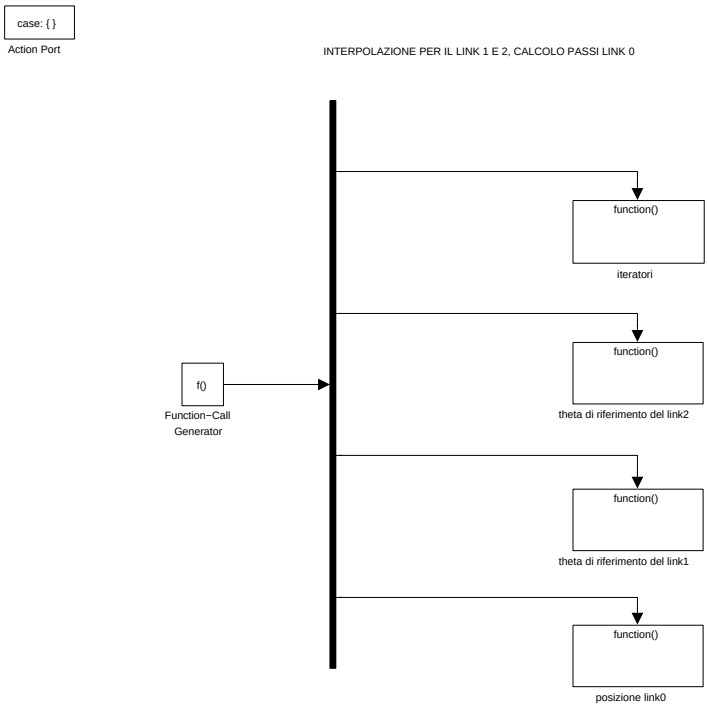

Figura 3.25: Subsystem: Calcolo riferimento ⇒ calibrazione ⇒ funzionamento normale

Nel primo function call subystem 'iteratori': Figura 3.26

vengono calcolati due iteratori, che non sono altro che due contatori, uno per

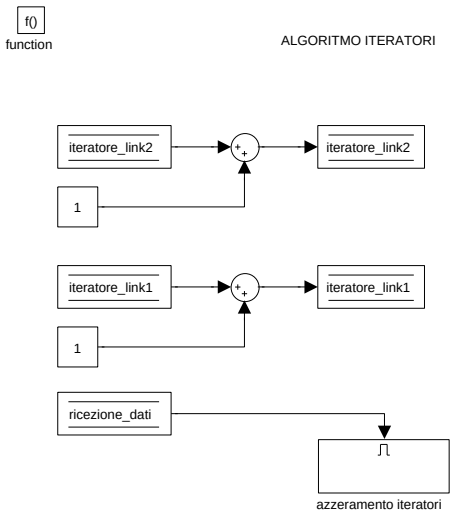

Figura 3.26: Subsystem: Calcolo riferimento ⇒ calibrazione ⇒ funzionamento normale ⇒ iteratori

il link 1, 'iteratore_link1' e uno per il link 2, 'iteratore_link2', che aumentano i loro valori ogni volta che si entra in questo sottosistema di 1.
Saranno utilizzati per l' interpolazione: siccome il programma Simulink gira 10 volte più velocemente di quanto vengono aggiornati i riferimenti di posizione da Matlab questi iteratori sono stati progettati per contare il numero di cicli che ha fatto Simulink dall' ultimo aggiornamento dei riferimenti.
Infatti quando viene ricevuto un pacchetto dati nuovo, 'ricezione_dati' diventa 1, e si attiva il subsystem enabled 'azzeramento iteratori', al cui interno si assegna il valore 1 alle variabili: 'iteratore_link2' ed 'iteratore_link1' e può ripartire così il conteggio. Il valore è posto ad 1 e non a zero perché si deve contare la volta in cui arriva il pacchetto nuovo di dati.
Nel secondo fucntion-call sibsystem 'theta di riferimento del link2'. Figura 3.27 viene calcolato il vero riferimento di posizione per il link 2, cioè quello ottenuto

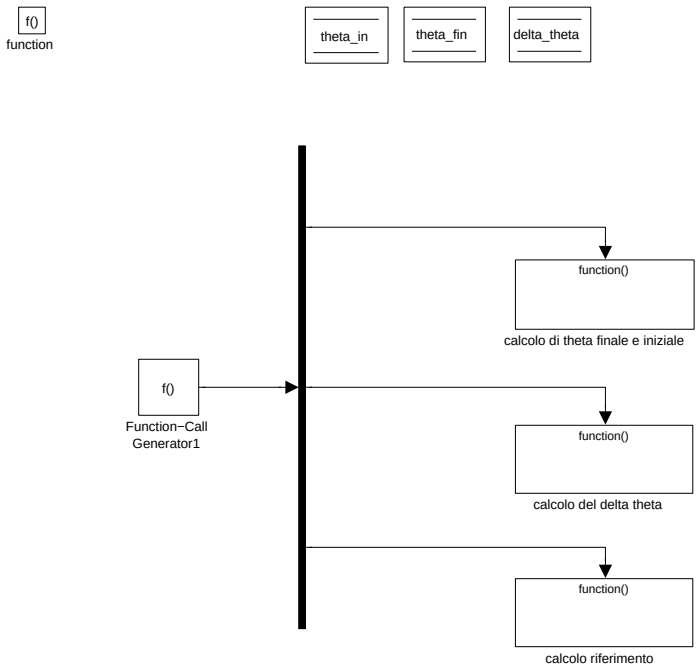

Figura 3.27: Subsystem: Calcolo riferimento ⇒ calibrazione ⇒ funzionamento normale ⇒ theta di riferimento del link2

dall' interpolazione.
$\grave{E}$ stato necessario strutturare il suo interno con un altro function-call generator, infatti anche se la logica è molto semplice il numero di blocchetti utilizzati, e scrivendoli ordinati dall' alto verso il basso non venivo eseguiti in ordine corretto da Simulink.
Questo viene specificato perché soprattutto all' inizio può essere fonte di molti errori.
Per effettuare l' interpolazione si considera la nuova posizione da raggiungere e la posizione di partenza in cui si trova il link.
Queste vengono memorizzate nel primo function-call subsystem 'calcolo di theta finale e iniziale'. Figura 3.28.
La nuova posizione è banalmente quella appena inviata da Matlab, quella attuale si ottiene inserendo 10 blocchi 'memory' in uscita dalla variabile 'posizione_link2':

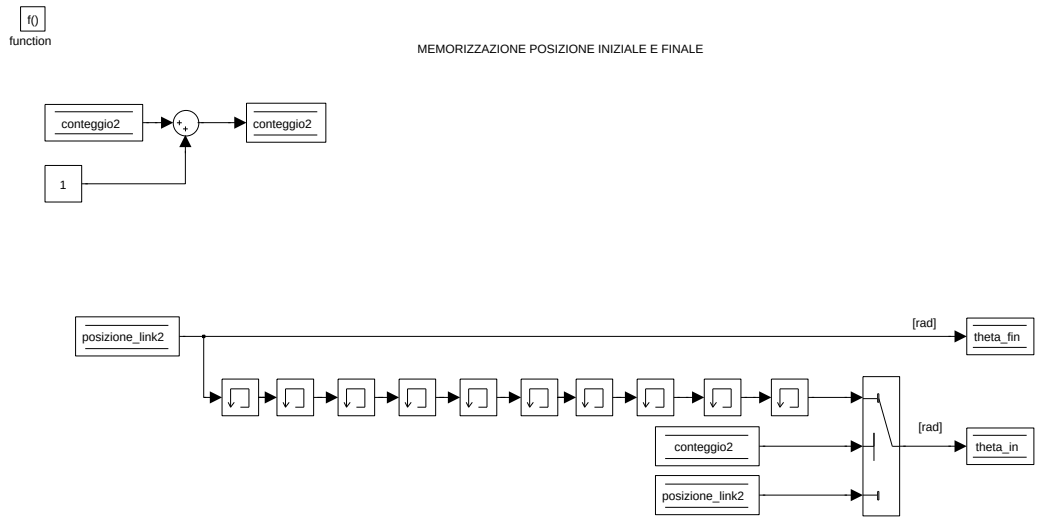

Figura 3.28: Subsystem: Calcolo riferimento ⇒ calibrazione ⇒ funzionamento normale ⇒ theta di riferimento del link2 ⇒ calcolo di theta finale e iniziale

ognuno di essi memorizza il valore della variabile per un ciclo di simulazione utilizzandone 10 si memorizza la posizione proprio per 0.01 sec, il tempo cioè necessario all' aggiornamento del riferimento di posizione, teoricamente. Così da conservare il valore della posizione precedente.
Vengono così memorizzate 'theta_fin' e 'theta_in'.
Però sorge il problema per il primo centesimo di secondo da quando si entra in questo stato, quindi per i primi 10 cicli: non arriva nessun valore a scrivere 'theta_in' a causa dei blocchi memory, e allora si farebbe l' interpolazione con il valore con cui la variabile è inizializzata, che ovviamente non può andare bene.
Si introduce allora la variabile 'conteggio2' che conta semplicemente il numero di volte che si entra in questo subsystem, e uno switch con la condizione che quando 'conteggio2' è maggiore di dieci allora 'theta_in' assume il valore che arriva dai blocchi memory, prima assume direttamente il valore di 'posizione_link2' e cioè

dove si trova il link realmente in quel primo periodo. Quindi per il primo centesimo di secondo la posizione finale e iniziale coincidono e questo garantisce che non si debbano mai interpolare posizione troppo lontane tra di loro, ma solo quello previste dalla pianificazione.
Nel secondo function-call subsystem 'calcolo del delta theta'. Figura 3.29.
viene fatta la sottrazione tra 'theta_fin' e 'theta_in' e il risultato diviso per 10,

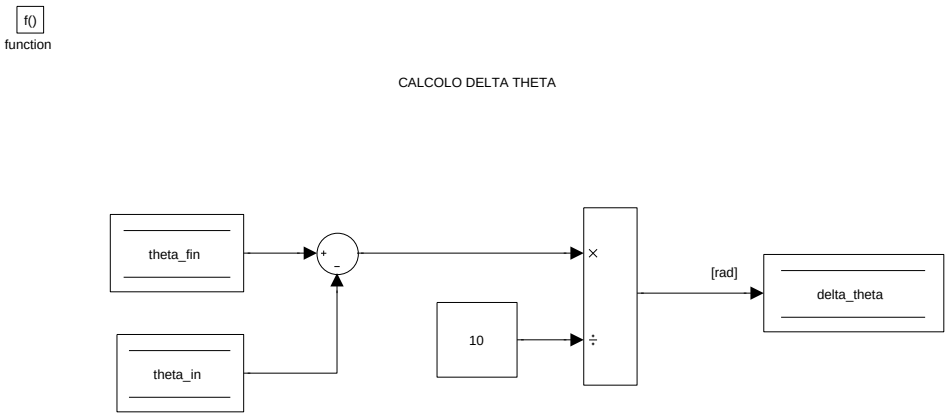

Figura 3.29: Subsystem: Calcolo riferimento ⇒ calibrazione ⇒ funzionamento normale ⇒ theta di riferimento del link2 ⇒ calcolo del delta theta

questo viene salvato nella variabile 'delta_theta'.
Così abbiamo trovato il valore costante con cui si può incrementare la posizione iniziale per raggiungere quella finale in 10 volte.
In pratica consiste nel rendere 10 volte più piccola l' ampiezza del gradino che verrebbe dato come riferimento al controllore.

Nel terzo function-call subsystem 'calcolo riferimento pid link2'. Figura 3.30

viene calcolato il riferimento di posizione per il controllore, secondo l' algoritmo:

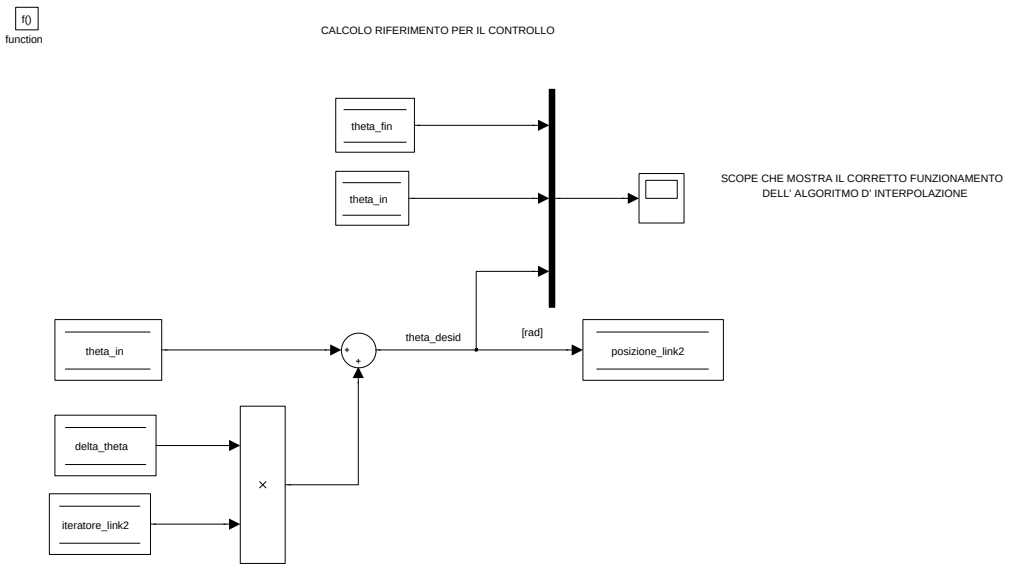

Figura 3.30: Subsystem: Calcolo riferimento ⇒ calibrazione ⇒ funzionamento normale ⇒ theta di riferimento del link2 ⇒ calcolo riferimento

posizione_link2 = theta_in + 'delta_theta'*'iteratore_link2'.

Diventa ora chiara l' utilità di aver creato gli iteratori, infatti andando a moltiplicare il valore del delta creano sempre il riferimento successivo e quando 'iteratore_link2' = 10, 'theta_in' +'delta'*'iteratore_link2' è proprio uguale a 'theta_fin'.
Si è detto però che si deve considerare anche il fatto che non sempre il pacchetto dati arriva a Simulink, quindi la posizione verrebbe aggiornata ad esempio dopo 0.02 sec, o 0.03 secondi o comunque dopo un tempo non conosciuto ma superiore al centesimo di secondo.
Ma l' algoritmo funziona lo stesso, infatti è vero che il valore degli iteratori supe-

rerebbe il valore 10 se non si riceve un pacchetto dati, e quindi si potrebbe pensare che 'theta_in' assumerebbe valori più grandi di 'theta_fin'(infatti inizialmente era stato pensato che il valore dell' iteratore tornasse ad 1 non solo se si ricevevano dati ma anche se l' iteratore assumeva il valore 11).
Ma questa condizione non serve, perché dopo 10 cicli se non si riceve nessun dato 'theta_fin' = 'theta_in' quindi 'delta_theta' = 0 che moltiplicato per un qualsiasi valore dell' iteratore da sempre zero e quindi il riferimento rimane fermo all' ultima posizione inviata da Matlab.
Esattamente analogo è quello che avviene per calcolare il riferimento per il link 1. Viene usata la stessa logica, e gli stessi blocchetti, cambiando nei nomi delle variabili il numero 2 con il numero 1.
Per il calcolo del riferimento nel link 0 si passa al quarto function-call subsystem 'posizione link 0'. Figura 3.31.

viene calcolata la differenza tra la posizione inviata da Matlab 'posizione_link0' e la posizione in cui si trova la base mobile 'posizione_link0_attuale', considerate sempre in numero di passi, così il risultato è proprio il numero di passi che deve fare il motore stepper.
Allora il modulo della differenza viene salvato nella variabile 'passi_link0' ed è il numero di passi che viene inviato al controllo che si occuperà di farli eseguire.
A seconda se era risultato un numero positivo o negativo, tramite un if viene assegnato alla variabile 'direzione_link0' rispettivamente 1 o 0.

Termina così la spiegazione del function-call subsystem 'calcolo riferimento'.
Ora vengono spiegati tutti gli altri sottosistemi della figura 3.1 in ordine dal più alto al più basso che è l' ordine secondo il quale vengono eseguiti da Simulink.

3.2.2 Ingressi Sensoray626

Il primo Function Call Subsystem prevede la lettura e la memorizzazione nelle variabili di tutti gli ingressi della Sensoray, che sono già stati elencati nel capitolo precedente. Figura 3.32.

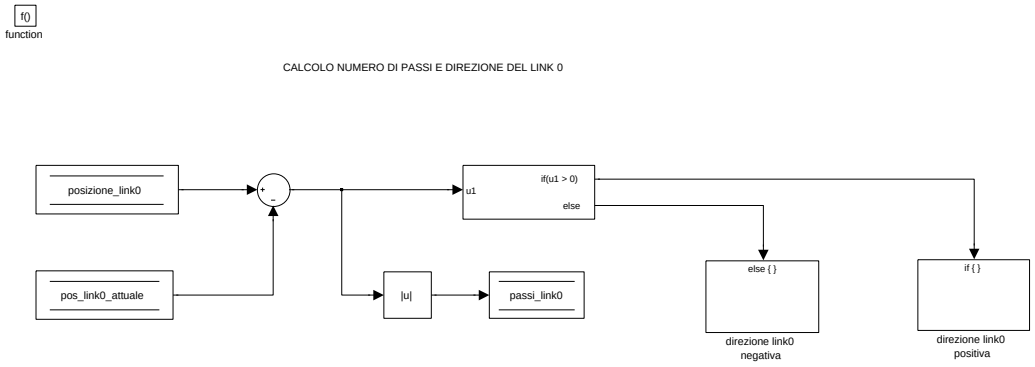

Figura 3.31: Subsystem: Calcolo riferimento ⇒ calibrazione ⇒ funzionamento normale ⇒ posizione link 0

È stato scelto di leggere tutti gli ingressi in un unico sottosistema perché questo permette una migliore leggibilità del programma, ed inoltre così tutti gli ingressi vengono letti esattamente nello stesso momento permettendo una maggiore precisione.
Per poter leggere i valori dalla Sensoray 626 sono stati utilizzati blocchetti 'Analog Input', 'Digital Input' ed 'Encoder Input', in cui impostando un specifico canale di lettura leggono i segnali in ingresso alla scheda di acquisizione.
Le variabili utilizzate per memorizzare i vari ingressi sono:

- 'encoder_link2' ed 'encoder_link1' per la posizione rispettivamente del link 2 e 1 letta dagli encoder, quindi la posizione reale dei membri.
  I canali di ingresso encoder utilizzati sono rispettivamente il 5 e il 4.
  Ingressi encoder x2.

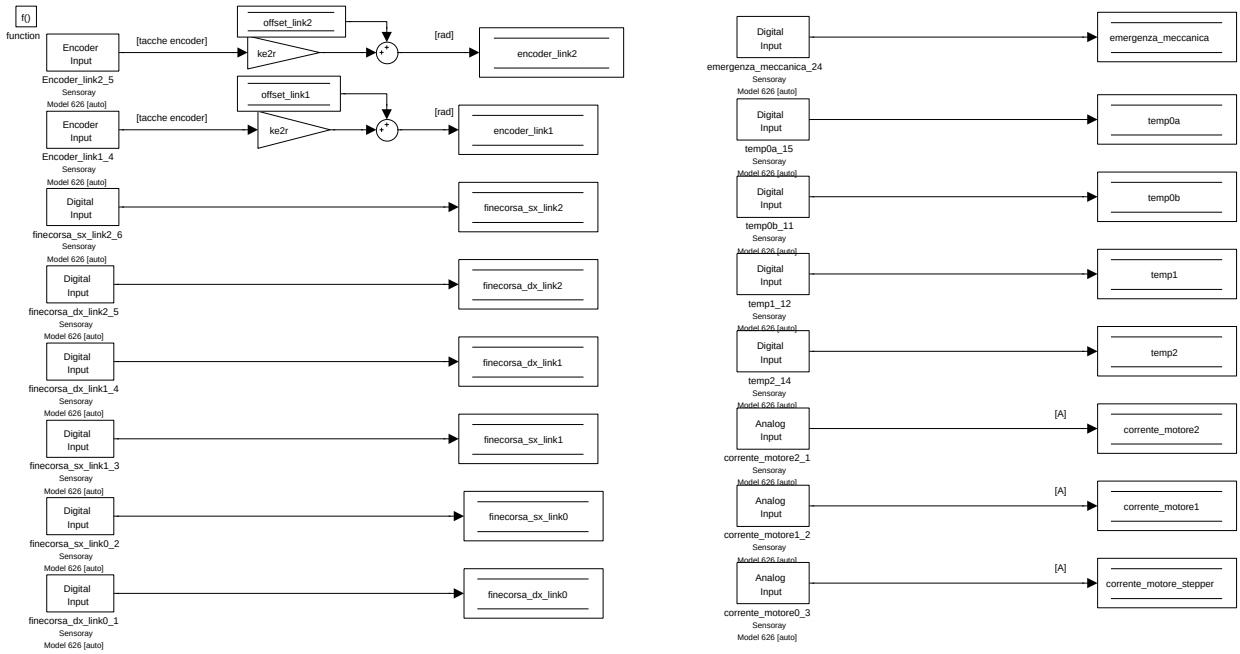

Figura 3.32: Subsystem in cui vengono memorizzati tutti gli ingressi della Sensoray

- 'finecorsa_sx_link2', 'finecorsa_dx_link2', 'finecorsa_dx_link1', 'finecorsa_sx_link1', 'finecorsa_sx_link0', 'finecorsa_dx_link0' per rilevare il passaggio sui finecorsa induttivi. Che segnano 0 se il link non è sopra il finecorsa induttivo, 1 quando lo fa attivare.
  I canali di ingressi digitali utilizzati sono rispettivamente 6,5,4,3,2,1. Ingressi digitali x6.

- 'emergenza_meccanica' corrisponde ad un relè che attiva il circuito d' emergenza, quindi serve per leggere se il sistema per qualche motivo è entrato in emergenza. La logica di funzionamento è 1 se siamo in emergenza, e quindi il manipolatore è bloccato dall' elettronica e non si può farlo muovere

via software, 0 se non ci sono emergenze. I possibili motivi per cui questo ingresso può passare allo stato logico alto sono:

- attivazione di uno qualsiasi dei finecorsa meccanici;
- presseione del pulsante di emergenza;
- non arriva l' alimentazione ai motori.

Il canale di ingresso digitale utilizzato è il 24.
Ingresso digitale x1.

- 'temp0a', 'temp0b', 'temp1', 'temp2' sono anche questi segnali d' emergenza. Si riferiscono alle temperature degli stadi in potenza dei tre motori, il motore stepper, relativo al link 0 utilizza due LMD e quindi ha bisogno da solo di due ingressi di temperatura che vengono chiamati a e b.
  Funzionano con la logica inversa, quando viene letto 0 vuol dire che la temperatura ha superato il massimo tollerabile, quando viene letto 1 la temperatura è dentro ai valori ammissibili.
  I canali di ingressi digitali utilizzati sono rispettivamente 15, 11, 12, 14.
  Ingressi digitali x4.

- 'corrente_motore1', 'corrente_motore2', 'corrente_motore_stepper', sono i monitor di corrente che passa attraverso gli LMD.
  I canali di ingresso analogici utilizzati sono rispettivamente 1, 2, 3.Ingressi analogici x3.

Questo è stato selezionato per essere il primo subsystem, in quanto essendo il collegamento con ciò che avviene nella realtà, è ovvio che da quello che leggiamo da qui, possiamo andare ad agire per modificare, correggere o mantenere l' azione del manipolatore.

3.2.3 Emergenze

Il secondo Function Call Subsystem serve a controllare eventuali stati di emergenza. Figura 3.33.

È suddiviso a sua volta in un function-call generator con sei subsystem, che

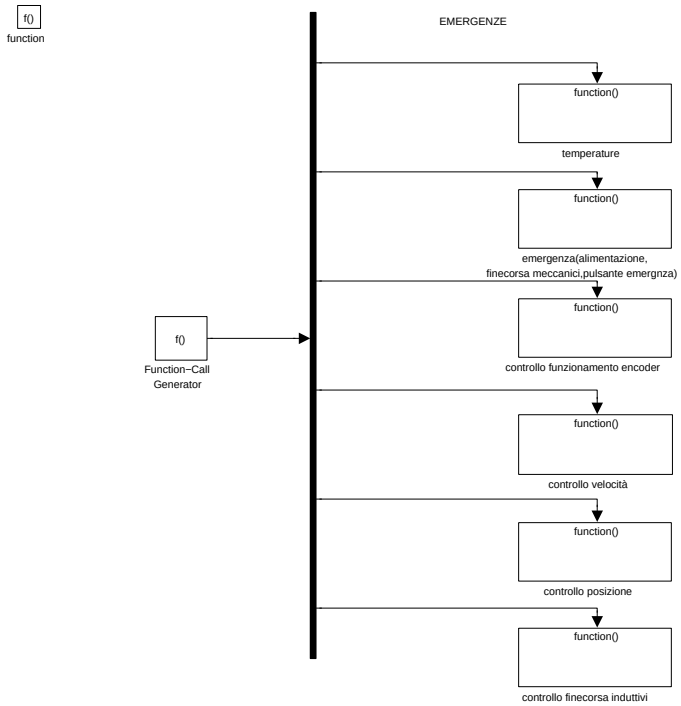

Figura 3.33: Subsystem: emergenze

rappresentano le sei categorie di emergenze che si possono attivare.
Tutti i tipi di emergenza sono stati distinti per rendere più leggibile in codice, e capire meglio quale emergenza è stata attivata.
Ma come è stato accennato nel paragrafo 'Calcolo riferimento' tutte le emergenze portano a far frenare i motori e si distinguono in due tipi, quelle che portano alla ricalibrazione di tutto il robot, e quelle invece che permettono di mantenere la lettura encoder e quindi non è necessario ricalibrare i membri al loro termine.

La prima categoria d' emergenza è considerata in 'temperature'. Figura 3.34. viene controllata la temperatura dei quattro stati di potenza.
Gli integrati LMD, gli integrti principali degli stadi di potenza, erogano la corrente ai motori. Avendo i motori una maggiore area di dissipazione del calore, avranno temperatura inferiore a quella degli integrati, quindi monitorando la corrente di

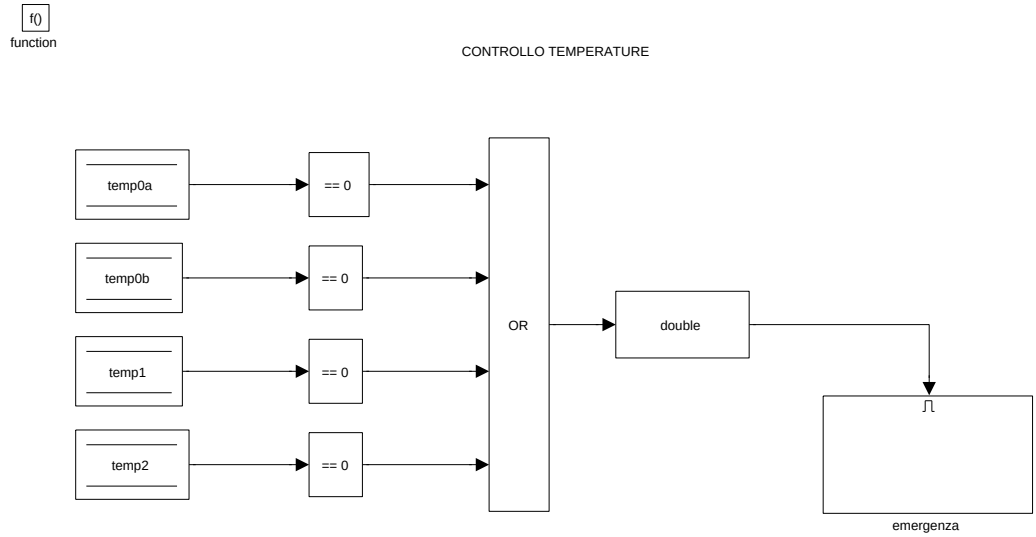

Figura 3.34: Subsystem: emergenze ⇒ temperature

quest' ultimi si può evitare anche di surriscaldare i motori.
I quattro segnali riferiti alle temperature sono digitali e indicano: se il livello logico è 1 che la temperatura è entro i limiti per il funzionamento, se il livello logico è 0 che è stata superata la temperatura limite, corrispondente circa a 170° (è stato misurato che il motore a questa temperatura arriva circa a 150°).
Per il loro funzionamento a logica inversa nello schema Simulink è stato previsto un blocchetto che uguaglia a zero questi ingresso digitali, se l' uguaglianza è verificata e siamo quindi in emergenza l' uscita del blocchetto diventa 1, se no è zero. Questo ci permette di considerarli come se funzionassero con la logica diretta.
Le uscite dei blocchetti comparatori sono gli ingressi di una porta OR che porta a false l' uscita se tutte le temperature non sono in emergenza e a true se una invece si attiva. La conversione in double permette l' attivazione del subsystem

'emergenza'. Figura 3.35.

Questa emergenza rientra nella tipologia che permette la lettura encoder e non

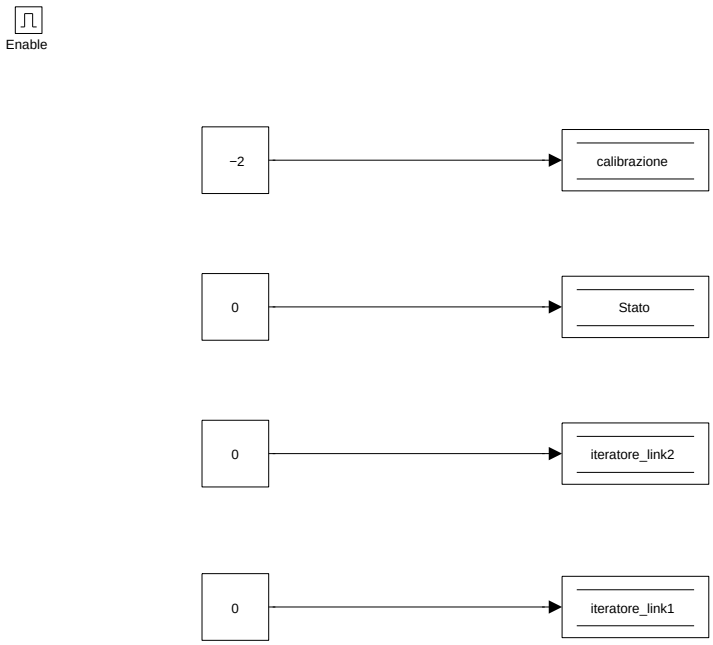

Figura 3.35: Subsystem: emergenze ⇒ temperature⇒ emergenza

bisogna ricalibrare il robot una volta terminata.

Quindi in questo sottosistema avvengono le seguenti assegnazioni:

- 'Stato' = 0, per entrare nello stato di emergenza;
- 'calibrazione' = -2, che è il valore che indica manipolatore calibrato;
- 'iteratore_link2' = 0, per far riniziare correttamente l' algoritmo di interpolazione della posizione del link 2;
- 'iteratore_link1' = 0, per far riniziare correttamente l' algoritmo di interpolazione della posizione del link 1;

Quando l' uscita della porta 'or' tornerà a zero l' emergenza sarà finita e si riprenderà il ciclo dell macchina a stati.

La seconda categoria di emergenza è considerata in 'emergenza(alimentazione, finecorsa meccanici,pulsante emergenza)'. Figura 3.36.
Si prende in considerazione una sola variabile 'emergenza_meccanica' e quindi non

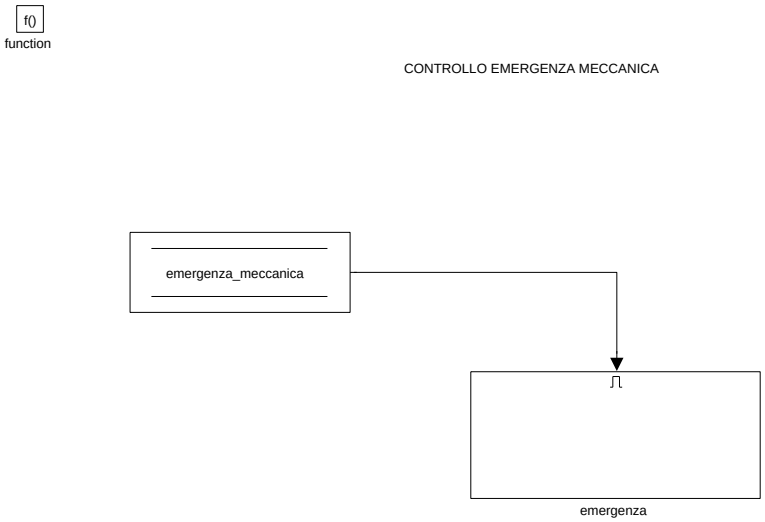

Figura 3.36: Subsystem: emergenze ⇒ emergenza (alimentazione, finecorsa meccanici,pulsante emergenza)

è presente la porta OR del caso precedente,ma la logica è la stessa, quando questa emergenza si attiva assume il valore 1 e va ad abilitare il subsystem 'emergenza'. Questa tipologia di emergenza porta invece alla ricalibrazione di tutto il robot, infatti se ad esempio il membro 0 è arrivato su un finecorsa meccanico, bisogna spegnere il quadro elettrico, ma allora si spengono anche i motori e gli encoder non leggono più la posizione.
L' interno del subsystem 'emergenza' è analogo a quello del precedente, ma cambia il valore assegnato a calibrazione. Figura 3.37.

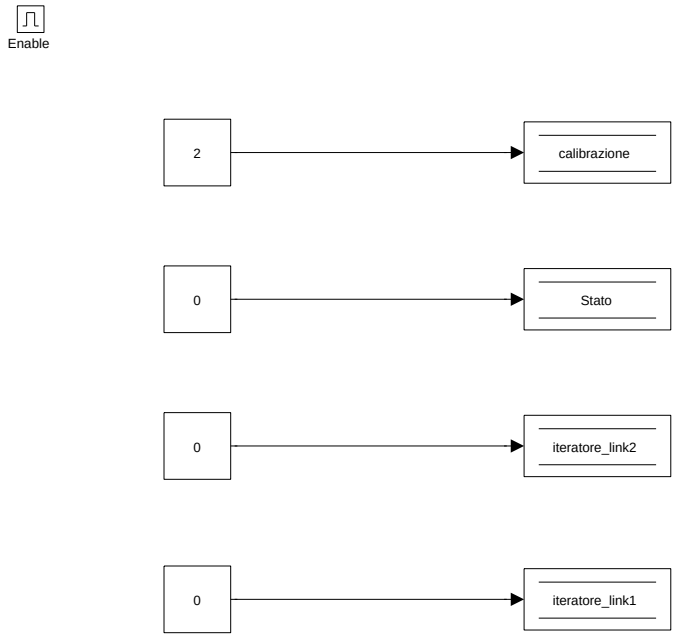

Figura 3.37: Subsystem: emergenze ⇒ temperature,emergenza(alimentazione, finecorsa meccanici,pulsante emergenza) ⇒ emergenza

- 'Stato' = 0, per entrare nello stato di emergenza;
- 'calibrazione' = 2, che è il valore iniziale della varibile e indica che il manipolatore deve essere calibrato;
- 'iteratore_link2' = 0, per far riniziare correttamente l' algoritmo di interpolazione della posizione del link 2;
- 'iteratore_link1' = 0, per far riniziare correttamente l' algoritmo di interpolazione della posizione del link 1;

Quando 'emergenza_meccanica' tornerà a zero l' emergenza sarà finita e si riprenderà il ciclo dell macchina a stati.
Queste due categorie di emergenze sono quelle che possono essere generate dall'

elettronica, ora analizziamo quelle che può generare il programma Simulink.

La terza categoria di emergenze è considerata in 'controllo funzionamento encoder'. Figura 3.38.
serve per controllare che arrivino i dati di lettura degli encoder, potrebbe infatti

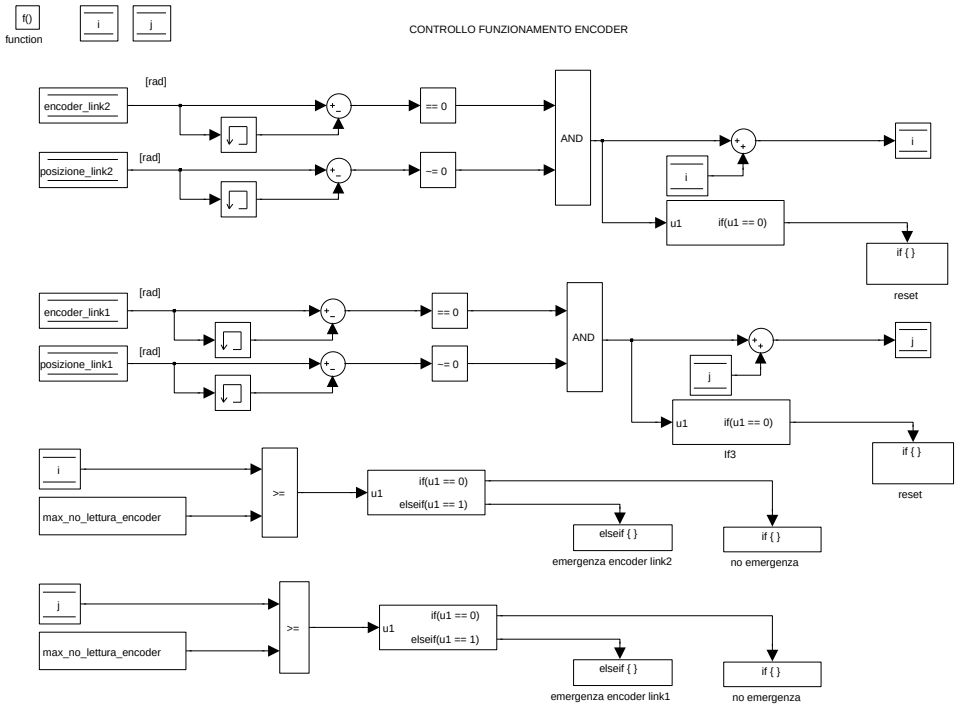

Figura 3.38: Subsystem: emergenze ⇒ controllo funzionamento encoder

succedere che si scolleghi il cavo di collegamento tra gli encoder e gli ingressi della Sensoray, o che venga danneggiato senza accorgersene.
Il procedimento come si può vedere dalla figura, è esattamente analogo per entrambi i link.
Per verificare questo si sottrae alla lettura encoder, la lettura precedente che si tiene in memoria con il blocchetto memory. Se la differenza è nulla vuol dire che la il dato encoder non è cambiato. La differenza viene messa in ingresso ad un blocchetto comparatore che la uguaglia a 0, se la condizione è verificata esce un

1, altrimenti uno 0, e questa uscita è l' ingresso di una porta 'and'.
L' altro ingresso della porta 'and' è dato dalla differenza tra la posizione di riferimento del link comparata con un blocchetto comparatore a se è diversa da 0. Quindi questo ingresso sarà 1 tutte le volte che il riferimento di posizione cambia valore.
Per tanto se il riferimento di posizione cambia valore e quello dell' encoder invece no, l' uscita della porta 'and' diventa 1 e va ad incrementare un contatore. Il contatore è 'i' per il link 2 e 'j' per il link 1.
Se l' uscita della porta AND è zero grazie ad un blocchetto if si entra nel subsystem 'reset' che azzera il contatore.
Il valore del contatore viene confrontato ogni volta con il valore massimo per cui questa situazione sia accettabile, che viene caricato dal file Matlab con il nome di 'max_no_lettura_encoder'. Il valore massimo è stato scelto essere 50, se il valore del contatore è minore, l' uscita dal blocchetto comparatore è uguale a 0 e allora grazie all' utilizzo del blocchetto if si entra nel subsystem 'no emergenza' in cui viene posta 'emergenza_encoder' = 0. Questa variabile serve ad indicare nello subsystem stato di emergenza se la lettura encoder genera emergenza o no. Quando invece il contatore supera il valore massimo, l' uscita del comparatore diventa 1 e si entra nel subsystem 'emergenza encoder link2' (o 'emergenza encoder link1' rispettivamente per i due link). Figura 3.39.
Questo tipo di emergenza porta alla ricalibrazione del robot indipendentemente da quale encoder manda in emergenza. Allora avvengono le seguenti assegnazioni:

- 'Stato' = 0, per entrare nello stato di emergenza;
- 'emergenza_simulink' = 1, per far attivare l' emergenza;
- 'calibrazione' = 2, che è il valore iniziale della variabile e indica che il robot deve essere calibrato;
- 'iteratore_link2' = 0, per poter reinterpolare correttamente la posizione del link 2;

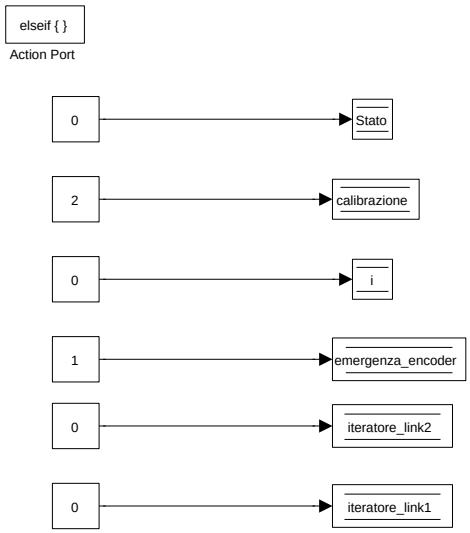

Figura 3.39: Subsystem: emergenze ⇒ controllo funzionamento encoder ⇒ emergenza

- 'iteratore_link1' = 0, per poter reinterpolare correttamente la posizione del link 1;
- 'i' = 0, per azzerare il contatore;

La terza categoria di emergenze è considerata nel subystem 'controllo velocità'. Figura 3.40.

qui non solo viene controllato che la velocità del link 1 e 2 non superi quella massima ammissibile, ma vengono anche calcolate le velocità e le accelerazioni. Avendo un feedback solo di posizione dei due link, ne viene fatta la derivata discreta per ottenere la velocità che viene memorizzata rispettivamente nelle variabili 'v_link1' e 'v_link2' e queste vengono derivate ulteriormente per ottenere le accelerazioni memorizzate nelle variabili 'a_link1' e 'a_link2'.

Le velocità vengono confrontate con la velocità massima ammissibile, che è stata

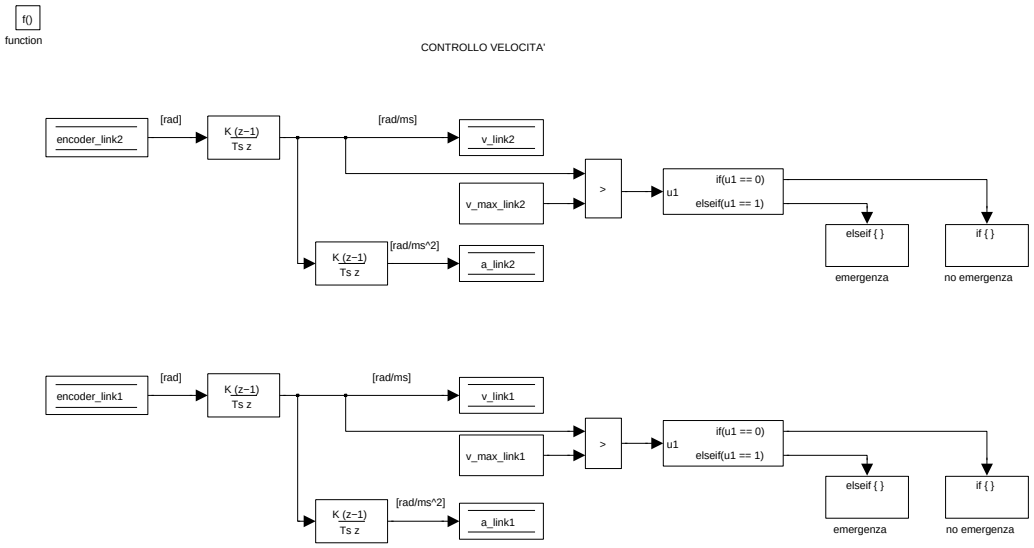

Figura 3.40: Subsystem: emergenze ⇒ controllo velocita

stimata a 4 [rad/s].

Finché la velocità non supera quella massima l' uscita del blocchetto 'relational operator' è zero e grazie al blocchetto if si entra nel subsystem 'no emergenza' in cui viene memorizzato 'emergenza_velocita' = 0, che se è appena finita l' emergenza permette il passaggio della macchina a stati dallo stato di emergenza al succesivo, durante il funzionamento normale invece conferma solo che va tutto bene.

Quando la velocità supera il limite massimo si entra nel subsystem 'emergenza'. Il subsystem ha struttura analoga a quelli precedenti.

Questa emergenza non altera la lettura encoder, e quindi non si deve ricalibrare il robot. Allora avvengono le seguenti assegnazioni nel sottosistema:

- 'Stato' = 0, per entrare nello stato di emergenza;
- 'emergenza_svelocita' = 1, per far attivare l' emergenza;

- 'calibrazione' = 2, che è il valore iniziale della variabile e verranno quindi calibrati tutti e tre i link;
- 'iteratore_link2' = 0, per poter reinterpolare correttamente la posizione del link 2;
- 'iteratore_link1' = 0, per poter reinterpolare correttamente la posizione del link 1;

La quarta categoria di emergenze è considerata in 'controllo posizione'. Figura 3.41.

Questa emergenza controlla che la posizione reale dei motori non sia troppo lon-

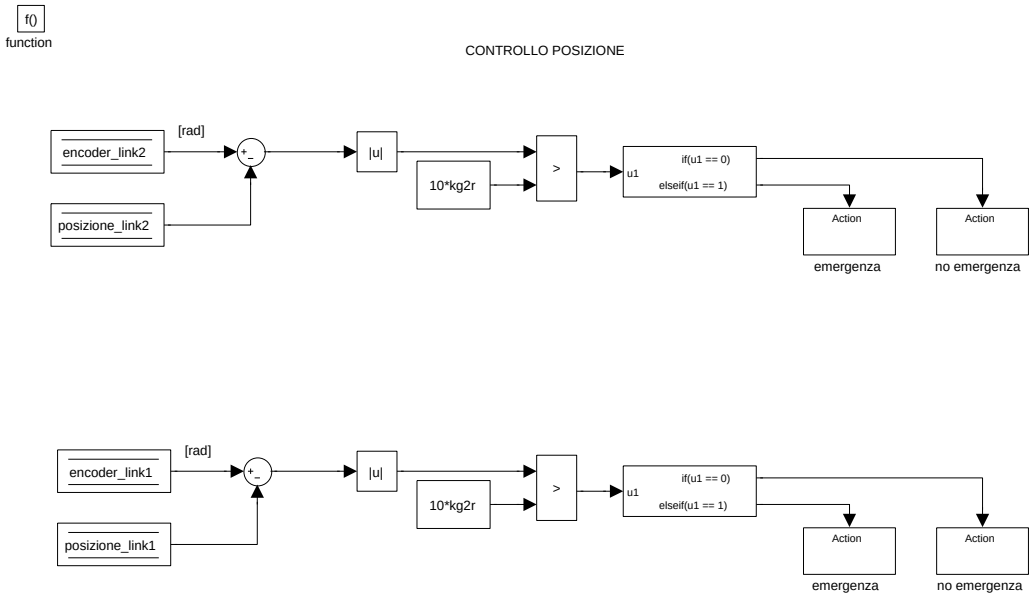

Figura 3.41: Subsystem: emergenze ⇒ controllo posizione

tana dalla posizione desiderata.

Se questo succede infatti significa che siamo andati fuori controllo e c'è il rischio di provocare danni al robot.

Per entrambi i link viene fatta la seguente operazione:
si sottrae la posizione desiderata a quella letta dagli encoder e ne viene calcolato il modulo se questo è superiore ad un certo valore scelto, il sistema entra in emergenza.
Il valore scelto oltre il quale l' errore di posizione non può andare è di 10 gradi.
Finché l' errore è inferiore a tale valore grazie ad un blocchetto 'if' si entra nel subsystem 'no emergenza' in cui viene assegnato il valore 0 alla variabile 'emergenza_posizione'.
Se l' errore invece supera tale soglia si entra nel subsystem 'emergenza'. Figura 3.42.
in cui vengono effettuate le seguenti assegnazioni:

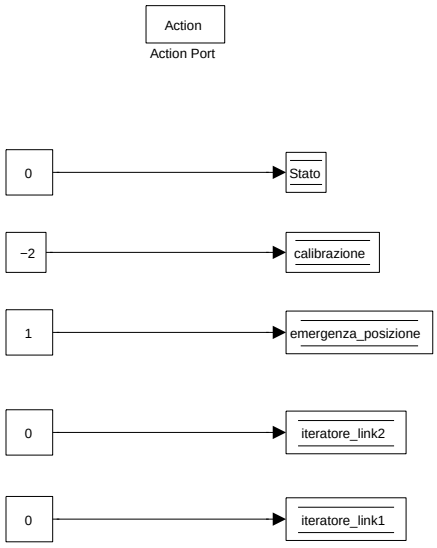

Figura 3.42: Subsystem: emergenze ⇒ controllo posizione ⇒ emergenza

'Stato' = 0 per entrare nello stato di emergenza;
'calibrazione' = -2 per indicare che il robot è già calibrato e ad emergenza finita

non bisognerà ricalibrare;
'emergenza_posizione' = 1 per indicare quale emergenza è stata generata;
'iteratore_link1' ed 'iteratore_link2' = 0 perché se si era nello stato = 4 e quindi si stava interpolando l' algoritmo deve essere resettato.
Quest' emergenza non prevede quindi la ricalibrazione, ma solo il bloccaggio del manipolatore fino al termine dell' emergenza.

La quinta categoria di emergenze è considerata in 'controllo finecorsa induttivi'. Figura 3.43.
Al suo interno troviamo un subsystem enabled che viene attivato solo se la va-

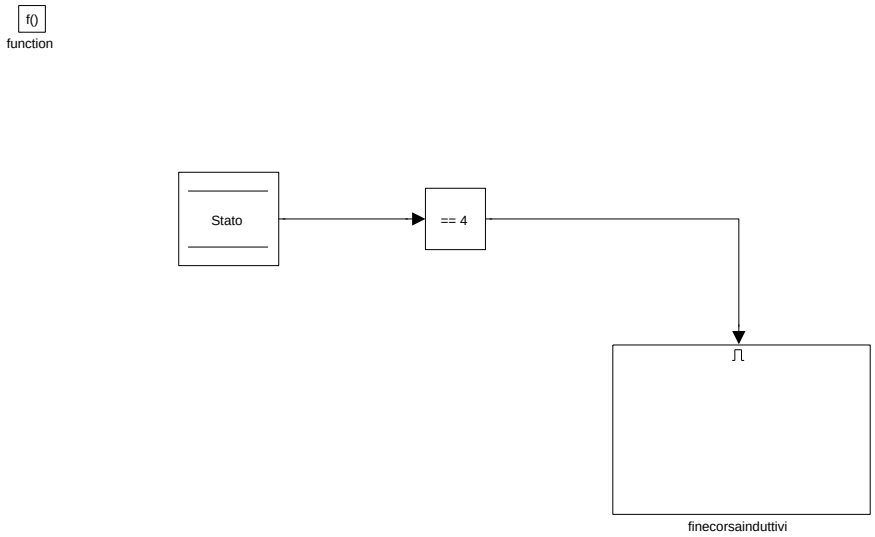

Figura 3.43: Subsystem: emergenze ⇒ controllo finecorsa induttivi

riabile 'Stato' = 4.
Questa condizione viene posta perché i finecorsa induttivi come è stato precedentemente spiegato vengono utilizzati dall' algoritmo di calibrazione e quindi fanno

attivare lo stato di emergenza solo se siamo nello stato di funzionamento normale. Se la condizione è verificata si entra nel subsystem 'finecorsa induttivi'. Figura 3.44.

Sono presenti due porte OR, che dividono il subsystem in due casi uno per i link

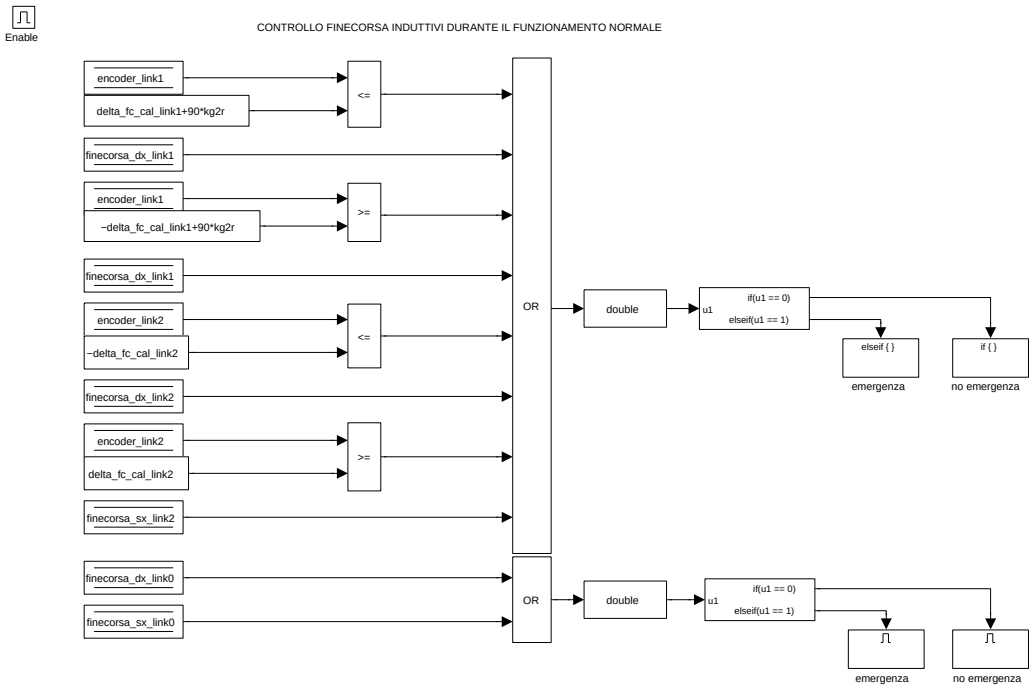

Figura 3.44: Subsystem: emergenze ⇒ controllo finecorsa induttivi ⇒ finecorsa induttivi

1 e 2, e uno per il link 0.

Sono divisi i due casi perché il primo porta allo stato di emergenza ma non deve essere ricalibrato tutto il robot, in quanto entrati nello stato di emergenza la tensione erogata dai motori in continua è nulla e i link possono essere spostati a mano per riportarli in condizioni di sicurezza con gli encoder che continuano a leggere la posizione.

Mentre nel secondo caso, quando si entra nello stato di emergenza il motore stepper continua ad erogare una coppia che continua a tenere bloccato dove è il carrello, l' unico modo per muoverlo è togliere l' alimentazione al quadro elettrico.

Questo corrisponde al fatto che gli encoder non leggono più la posizione mentre si sposta il manipolatore a mano e non si conosce più la posizione del link 0 non avendo un feedback, quindi è necessario ricalibrare tutto il robot.
Nella prima porta OR sono posti come ingressi tutti e quattro i finecorsa induttivi dei due link, e la verifica se la posizione letta dagli encoder non è oltre un finecorsa induttivo. Per fare questo viene controllato se la lettura encoder è inferiore alla posizione dei finecorsa destri e se è maggiore dei finecorsa sinistri. Queste condizioni sono necessarie perché tra il finecorsa induttivo e il finecorsa meccanico intercorre dello spazio in cui nessuno dei due tipi di finecorsa è attivo, se il membro arriva a quella posizione senza essere fermato deve attivarsi l' emergenza. L' emergenza avrebbe dovuto attivarsi già con l' induttivo, ma questa è un' ulteriore sicurezza.
La porta OR garantisce che se si attiva una qualsiasi delle condizioni l' uscita diventa 'true', e convertita in double fa attivare il subsystem 'emergenza'. Figura 3.45.
in cui vengono effettuate le seguenti assegnazioni:
'Stato' = 0 per entrare nello stato di emergenza;
'calibrazione' = -2 per indicare che il robot è già calibrato e ad emergenza finita non bisognerà ricalibrare;
'emergenza_induttivi1e2' = 1 per indicare quale emergenza è stata generata;
'iteratore_link1' ed 'iteratore_link2' = 0 perché l' algoritmo d' interpolazione deve essere resettato.
Se invece non si attiva nessuna condizione si entra nel subsystem 'no emergenza' in cui viene assegnato il valore 0 alla variabile 'emergenza_induttivi1e2'.
Nella seconda porta OR come ingressi è posta solo la lettura dei due finecorsa del link 0, dato che non si ha un ritorno di posizione quando uno di questi due si attiva, si entra nel subsystem 'emergenza'. Figura 3.46.
in cui vengono effettuate le stesse assegnazioni del caso precedente tranne 'calibrazione' = 2 che permette la ricalibrazione di tutto il robot.

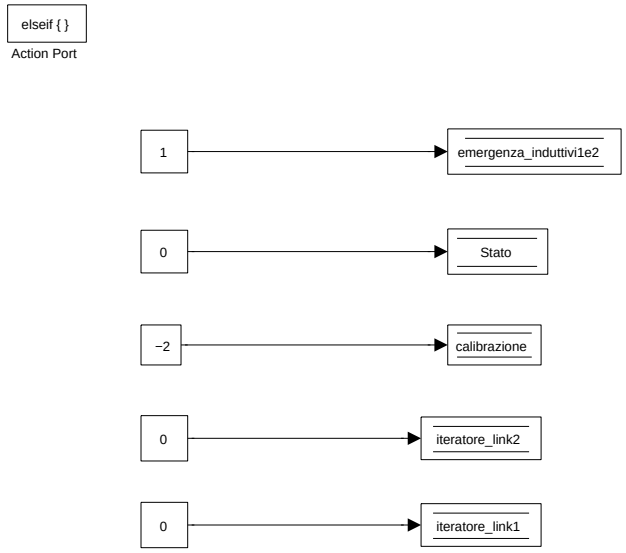

Figura 3.45: Subsystem: emergenze ⇒ controllo finecorsa induttivi ⇒ finecorsa induttivi ⇒ emergenza

3.2.4 Ricezione dati Matlab

In questo subsystem vengono ricevuti e memorizzati tutti i dati che vengono inviati dal calcolatore A. Figura 3.47.

I dati vengono ricevuti tramite il blocchetto 'Packet input' che appartiene proprio alla libreria 'Real Time Windows Target' quindi non ci sono problemi di incompatibilità e può ricevere i dati alla frequenza con cui gira la simulazione.

I parametri da inserire in questo blocchetto sono:

- l' indirizzo IP dell' altro computer, la 'Local port' e la 'Remote port', si installa una 'new board' con questi parametri che utilizzi il protocollo UDP e si crea così il Socket tramite cui avviene la comunicazione e si possono ricevere i dati;

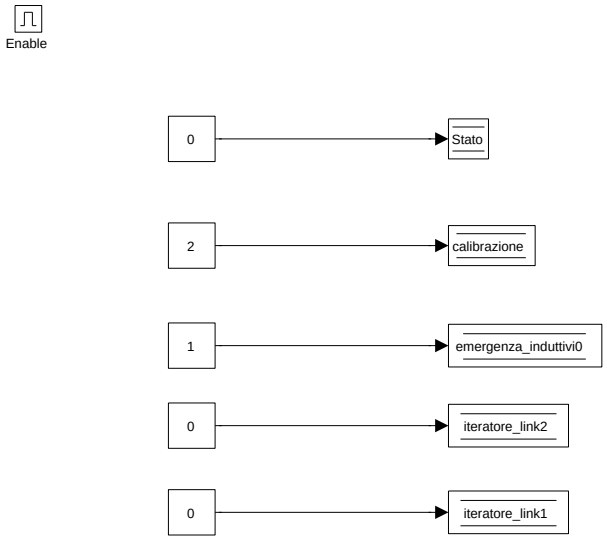

Figura 3.46: Subsystem: emergenze ⇒ controllo finecorsa induttivi ⇒ finecorsa induttivi ⇒ emergenza

- il tempo di campionamento che è stao posto anche questo a -1 quindi prova a ricevere i dati alla frequenza della simulazione;
- la dimensione del pacchetto in ricezione che essendo double si ottiene dal numero di dati ricevuti moltiplicati per 8, numero di bit per double;
- il numero di dati che vengono ricevuti che sono 15.

Fondamentale è la sincronizzazione tra la ricezione dei dati e il loro memorizzazione per l' utilizzo nel programma.
Come è stato descritto, nell' algoritmo di interpolazione gli iteratori si azzerano ogni volta che vengono ricevuti i dati, ed inoltre se si memorizzassero i dati anche negli istanti in cui non vengono ricevuti si rischierebbe di memorizzare dei valori errati come degli zeri.

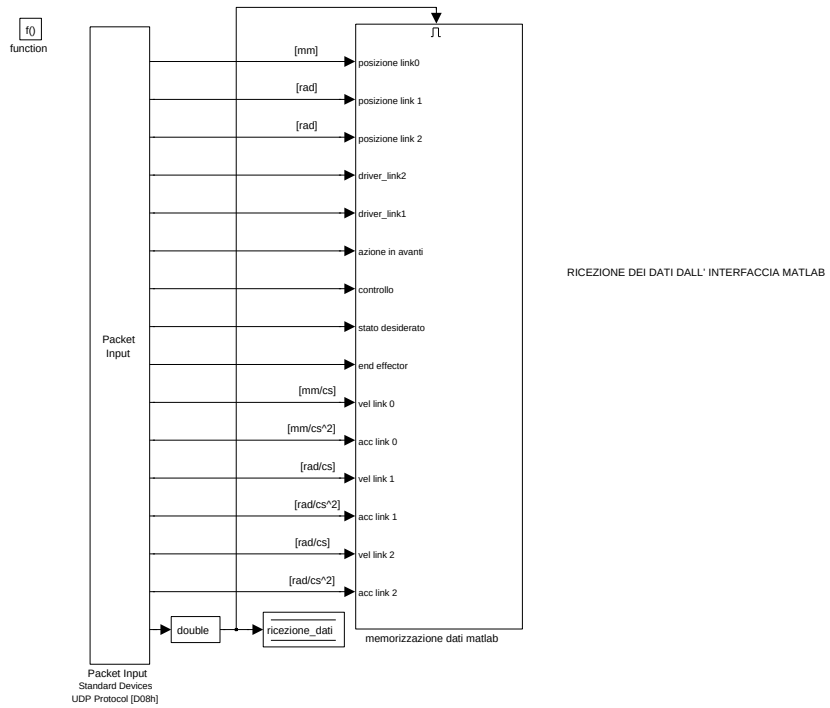

Figura 3.47: Subsystem: ricezione dati matlab

Come è già stato spiegato i dati vengono inviati da Matlab attraverso l' oggetto timer alla frequenza di 100 Hz, ma questo non corrisponde a ricevere i dati a quella frequenza. Infatti alcuni pacchetti dati vengono persi per i problemi di rete e relativi al protocollo di trasmissione utilizzato, altre volte si accumulano nel buffer e poi vengono letti ad intervalli di tempo anche minori al centesimo di secondo.

Per risolvere questi problemi il blocchetto 'Packet Input' ci permette di conoscere gli istanti esatti di ricezione dati: è sufficiente mettere la spunta sull' opzione 'data ready' del blocchetto, che gli aggiunge un' ulteriore uscita.

Questa uscita diventa 1 ogni volta che effettivamente viene ricevuto un pacchetto di dati nuovo e se no rimane a zero.

Questo dato, memorizzato nella variabile 'ricezione_dati', viene utilizzato per abilitare o no la lettura dei dati, infatti tutte le uscite del blocchetto entrano nel

subsystem enabled 'memorizzazione dati matlab' che viene abilitato da questa variabile. Figura 3.48.

I dati che vengono ricevuti, elencati in ordine dall' alto verso il basso, sono:

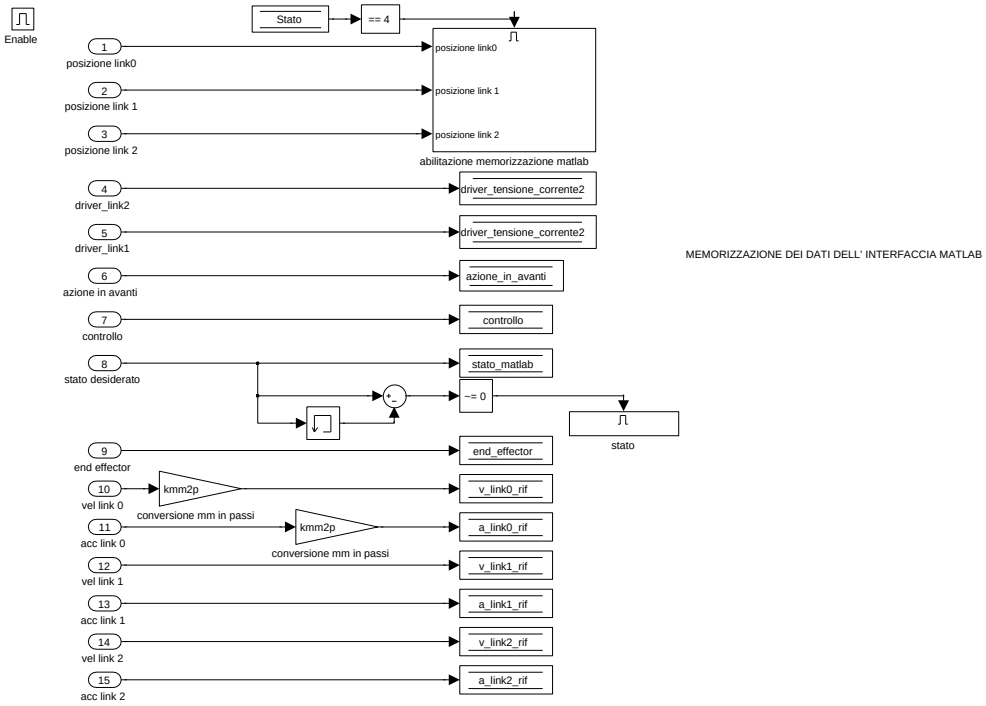

Figura 3.48: Subsystem: ricezione dati matlab ⇒ memorizzazione dati matlab

- i riferimenti di posizione per i tre link, ma questi entrano in un altro subsystem enabled che viene attivato solo quando 'Stato' = 4. Infatti negli stati precedenti i riferimenti di posizione vengono calcolati internamente al programma Simulink basta guardare l' algoritmo di calibrazione, se si sovrapponessero anche questi dati che sarebbero tutti zero perché nel programma Matlab non viene pianificata nessuna traiettoria si sbaglierebbero tutti i valori.

  Durante il funzionamento normale si entra quindi nel subsystem 'abilitazione memorizzazione posizioni'. Figura 3.49.

  le posizioni memorizzate sono:

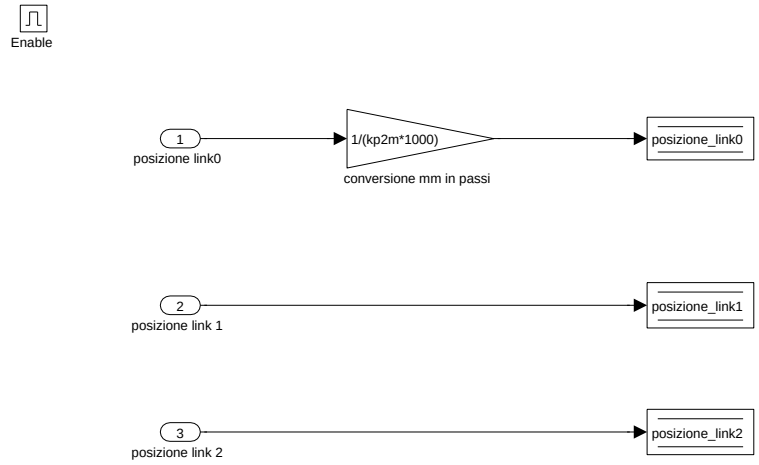

Figura 3.49: Subsystem: ricezione dati matlab ⇒ memorizzazione dati matlab ⇒ abilitazione memorizzazione posizioni

- il riferimento per la posizione del link 0, che indica la posizione successiva che deve raggiungere il carrello. Matlab la invia in mm, e allora deve essere convertita in numero di passi secondo il guadagno 'kmm2p' e viene salvata nella variabile 'posizione_link0';
- il riferimento per la posizione del link 1 espresso in radianti salvato nella variabili 'posizione_link1' ;
- il riferimento per la posizione del link 2 espresso in radianti salvato nella variabili 'posizione_link2'.

- il valore da inviare al driver del motore 2, per scegliere il funzionamento in tensione o corrente. Il primo corrisponde al valore 1 il secondo al valore 0. Questo dato viene memorizzato nella variabile 'driver_tensione_corrente2';

- il valore da inviare al driver del motore 1, per scegliere il funzionamento in tensione o corrente. Il primo corrisponde al valore 1 il secondo al valore 0. Questo dato viene memorizzato nella variabile 'driver_tensione_corrente1';

- il valore per utilizzare o no l' azione in avanti: 0 corrisponde a non utilizzare l' azione in avanti, 1 ad utilizzare l' azione in avanti decentralizzata, 2 ad utilizzare l' azione in avanti centralizzata. Questo dato viene memorizzato nella variabile 'azione_in_avanti';

- il valore per decidere che tipo di controllo utilizzare: 0 controllo decentralizzato, 1 controllo centralizzato. Questo dato viene memorizzato nella variabile 'controllo';

- il valore dello stato che il programma Matlab vorrebbe dare alla macchina a stati, viene memorizzato nella variabile 'stato_ matlab'.
  Questo dato è stato pensato non per seguire tutti i cambiamenti di 'Stato' ma solo per cambiarne il valore per entrare nello stato di calibrazione e di funzionamento normale. L' algoritmo pensato è stato che Matlab invii sempre lo stesso valore su questa variabile, quando il cambiamento di stato nella macchina dipende dalla GUI invia il valore corretto, quindi facendo la sottrazione tra il valore nuovo che arriva e il valore precedente, ottenuto grazie al blocchetto 'memory', se la differenza è diversa da zero vuol dire che l' ultimo dato inviato è quello che deve assumere la variabile 'Stato' e si entra in un subsystem in cui avviene l' assegnazione 'Stato' = 'stato_desiderato';
  Questo semplice algoritmo permette il passaggio dallo stato 1 allo stato 2, e dallo stato 3 allo stato 4 e il passaggio allo stato 0 da qualsiasi stato per generare l' emergenza;

- il valore che va a decidere se l' end effector, nel nostro caso la matita, deve appoggiarsi al foglio e quindi scrivere oppure no, rispettivamente 1 e 0;

- la velocità pianificata del link 0, memorizzata nella variabile 'v_link0_rif';

- l' accelerazione pianificata del link 0, memorizzata nella variabile 'a_link0_rif';

- la velocità pianificata del link 1, memorizzata nella variabile 'v_link1_rif';

- l' accelerazione pianificata del link 1, memorizzata nella variabile 'a_link1_rif';
- la velocità pianificata del link 2, memorizzata nella variabile 'v_link2_rif';
- l' accelerazione pianificata del link 2, memorizzata nella variabile 'a_link2_rif'.

3.2.5 Calcolo azione in avanti

La teoria di questa tecnica è stata spiegata nel capitolo precedente, in questa sezione vediamo come è stata implementata con Simulink.
Ricordiamo solamente che è una tecnica che va a compensare le componenti che causeranno errore di inseguimento, che esiste sia di tipo centralizzato che decentralizzato e che il suo valore cambia se si utilizza un driver controllato in tensione o in corrente. Figura 3.50
Il subsystem è diviso da un function call generator in due sottosistem' uno per il

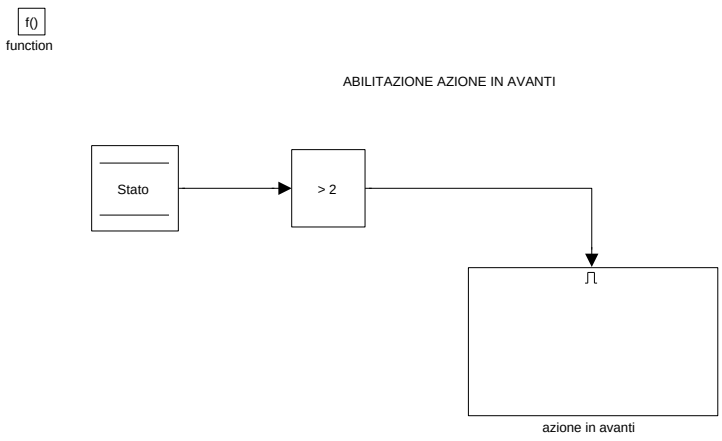

Figura 3.50: Subsystem: Calcolo azione in avanti

calcolo della coppia resistente, uno per lo specifico calcolo dell' azione in avanti, negli omonimi sottosistemi.
Il calcolo della coppia resistente viene posto in questo subsystem perché come è noto serve questo valore per il calcolo dell' azione in avanti centralizzata.
Poi i risultati ottenuti verrano utilizzati anche in altre parti del programma.
Il primo sottosistema ad essere eseguito è 'calcolo della coppia resistente'. Figura 3.51.
Vengono uniti in un unico vettore, attraverso il blocchetto 'Mux' che può ricevere

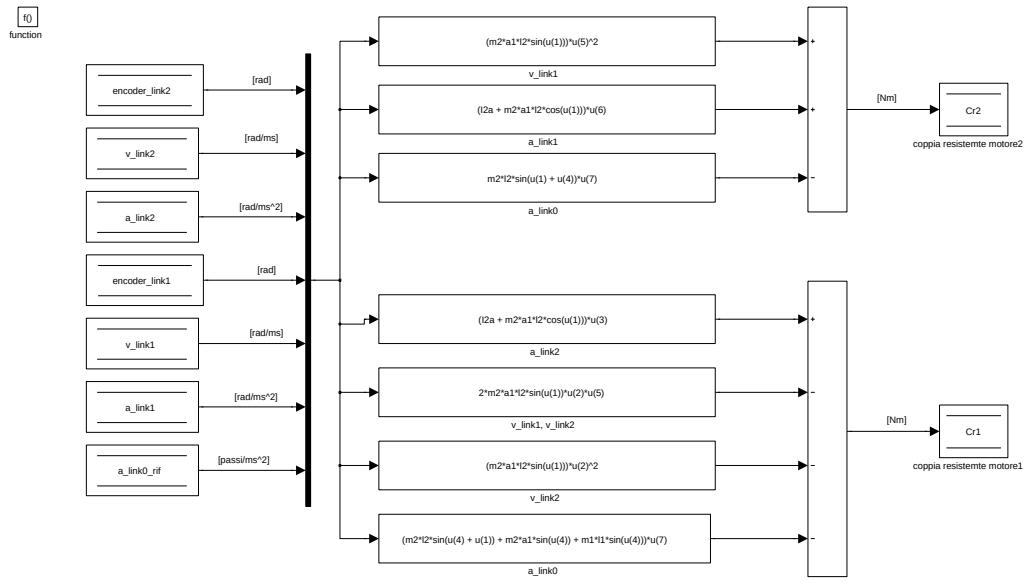

Figura 3.51: Subsystem: Calcolo azione in avanti ⇒ calcolo coppia resistente

n ingressi e li unisce in un' unica uscita, tutti gli ingressi necessari: le posizioni dei link 1 e 2 lette dagli encoder, le rispettive velocità ed accelerazioni e l' accelerazione del link 0.
L' uscita del 'Mux' viene posta in ingresso a dei blocchetti 'fcn' che permettono al loro interno di scrivere tutte le aperazioni matematiche desiderate e richiamare

i vari valori dell' ingresso chiamandoli come u(n° posizione nel vettore).
I calcoli implementati per i calcoli delle coppie resistenti sono dati dalle equazioni:

$$\begin{cases} C_{r2} = (I_{2,A} + m_2 a_1 l_2 cos(\Theta_2))\ddot{\Theta}_1 + m_2 a_1 l_2 sen(\Theta_2)\dot{\Theta}_1{}^2 - m_2 l_2 sen(\Theta_1 + \Theta_2)\ddot{x} \\ C_{r1} = (I_{2A} + m_2 a_1 l_2 (cos\Theta_2))\ddot{\Theta}_2 - 2m_2 a_1 l_2 sen(\Theta_2)\dot{\Theta}_1\dot{\Theta}_2 - m_2 a_1 l_2 sen(\Theta_2)\dot{\Theta}_2{}^2 \\ \qquad -m_2(l_2 sen(\Theta_1 + \Theta_2) + a_1 sin(\Theta_1))\ddot{x} - m_1 l_1 sin(\Theta_1)\ddot{x} \end{cases} \tag{3.5}$$

Memorizzate nelle omonime variabile.

Nel secondo function-call subsystem 'azione in avanti'. Figura 3.52.
vengono distinti il calcolo dell' azione in avanti per il motore 1 e per il motore

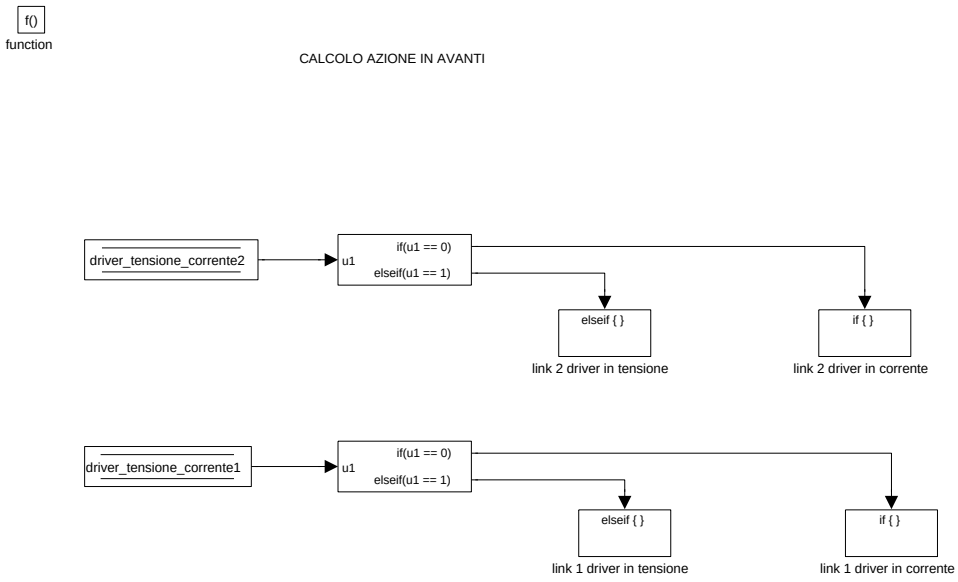

Figura 3.52: Subsystem: Calcolo azione in avanti ⇒ azione in avanti

2 e dalle variabili 'driver_tensione_corrente2' e 'driver_tensione_corrente1' inviate da Matlab un blocco if permette di entrare nel sottosistema per il calcolo dell' azione in avanti per il driver in corrente o in tensione.

Prendiamo in considerazione il calcolo per il motore 2.
Se 'driver_tensione_corrente2' = 1 si entra nel subsystem 'link 2 driver in tensione'. Figura 3.53.
al suo interno è prevista sia la possibilità di ottenere l'azione in avanti centraliz-

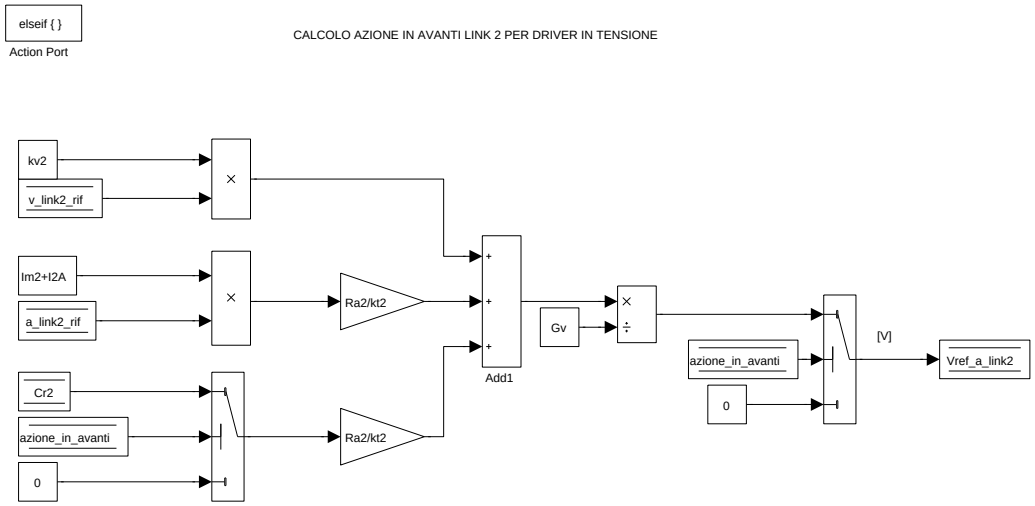

Figura 3.53: Subsystem: Calcolo azione in avanti ⇒ link 2 driver in tensione

zata, decentralizzata o di non usare l'azione in avanti.
Infatti nel primo switch, quello più a sinistra, è posta la condizione che se 'azione_in_avanti' = 2 allora viene letto il valore della coppia resistente, altrimenti questo termine è posto a zero. Questo equivale a scegliere tra centralizzata e decentralizzata infatti come è stato spiegato la prima tiene conto anche della coppia resistente e la seconda no.
Nel secondo switch è posta la condizione che se 'azione_in_avanti' = 0 allora il valore di 'Vref_a_link2' non è quello calcolato ma zero, quindi il controllo in questo caso non utilizza il contributo di nessuna azione in avanti.

L' equazione per il calcolo dell' azione in avanti decentralizzata è:

$$V_{ref,a} = \frac{Kv\dot{\Theta}_r + \frac{Ra \cdot I_{eq} \cdot \ddot{\Theta}_r}{Kt}}{Gv} \tag{3.6}$$

L' equazione per il calcolo dell' azione in avanti centralizzata è:

$$V_{ref,a} = \frac{Kv\dot{\Theta}_r + \frac{Ra \cdot I_{eq} \cdot \ddot{\Theta}_r}{Kt} + \frac{Ra \cdot C_{r2}}{Kt}}{Gv} \tag{3.7}$$

Se 'driver_tensione_corrente2' = 0 si entra nel subsystem 'link 2 driver in corrente'. Figura 3.54.

al suo interno è prevista sia la possibilità di ottenere l'azione in avanti centra-

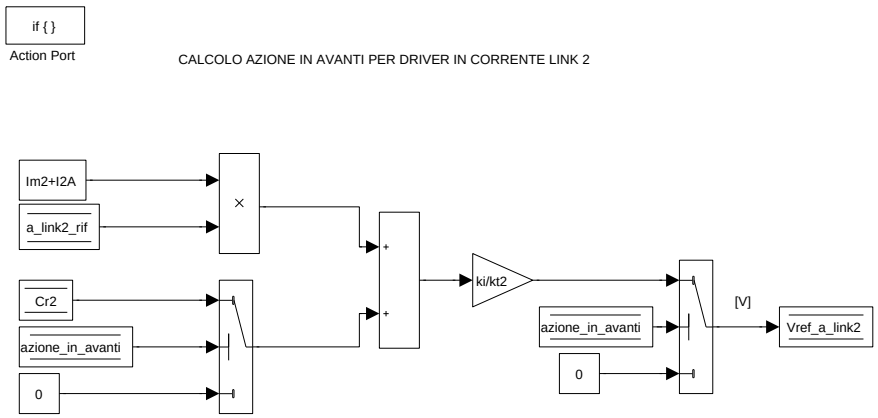

Figura 3.54: Subsystem: Calcolo azione in avanti ⇒ link 2 driver in corrente

lizzata, decentralizzata o di non usare l'azione in avanti analogamente al caso in tensione.

La tensione calcolata viene salvata sempre nella variabile 'Vref_a_link2', questo perché sarà questa la variabile che verrà presa in considerazione nel controllo, e

se si tratta di quella per il driver in tensione o in corrente questo dipende dalla scelta fatta dall' utente.

L' equazione per il calcolo dell' azione in avanti decentralizzata è:

$$V_{ref,a} = \frac{Ki \cdot I_{eq} \cdot \ddot{\Theta}_r}{Kt} \tag{3.8}$$

L' equazione per il calcolo dell' azione in avanti centralizzata è:

$$V_{ref,a} = \frac{Ki(\cdot I_{eq} \cdot \ddot{\Theta}_r + C_r)}{Kt} \tag{3.9}$$

Per il motore 1 il procedimento e la logica è esattamente lo stesso, vengono utilizzati gli stessi nomi della variabili cambiando il numero 2 in 1, la tensione dell' azione in avanti viene quindi memorizzata nella variabile 'Vref_a_link2'.

3.2.6 Calcolo controllo

In questo subsystem vengono implementati i due tipi di controllo visti per i link 1 e 2 e il controllo per il link 0. Figura 3.55.

All' interno del sotto sistema un altro function-call divide subito il controllo dei

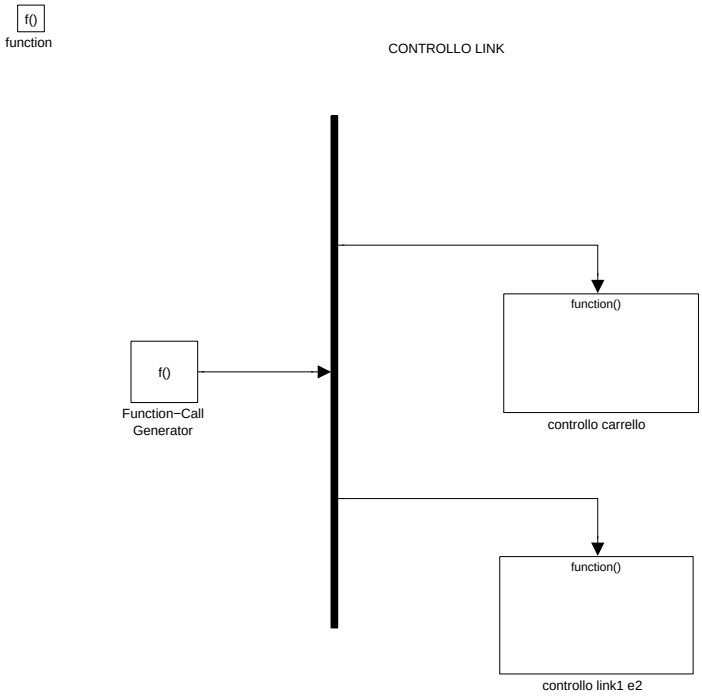

Figura 3.55: Subsystem: Calcolo controllo

vari membri: il link 0 verrà calcolato in un modo e gli altri due che vengono controllati in modo analogo in un altro.

Il primo function-call subsystem 'controllo carrello'. Figura 3.56.

si occupa di verificare che il numero di passi da far fare al motore passo-passo per quel clock di simulazione non sia superiore a quello massimo possibile, si effettua quindi in realtà un controllo sulla frequenza dei passi da inviare e si aggiorna la variabile della posizione del link.

Il numero di passi per millisecondo che viene consentito di fare è 5.

Fatta la differenza tra 'passi_link0' che sono il numero di passi che si vorrebbero

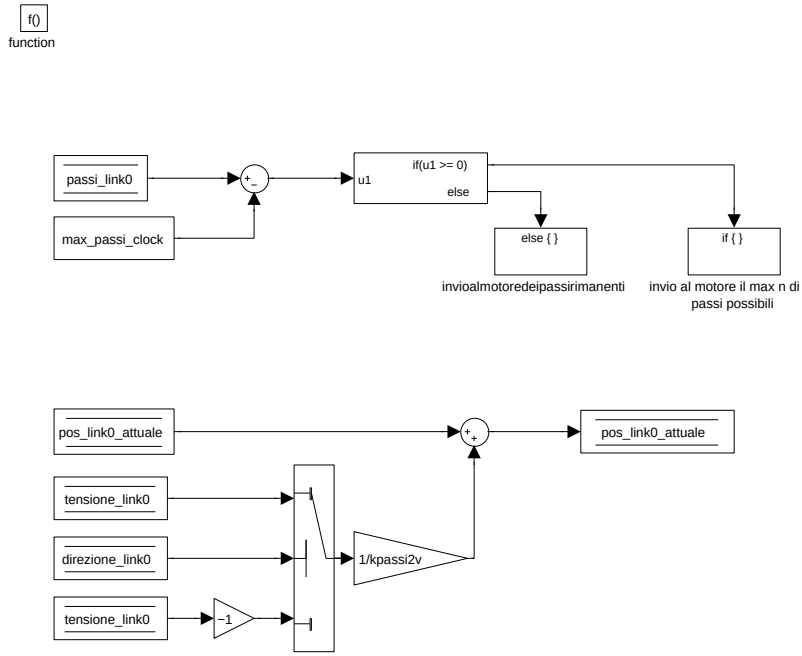

Figura 3.56: Subsystem: Calcolo controllo ⇒ controllo carrello

far fare e la costante 'max_passi_clock' caricata dal file Matlab, se il risultato è maggiore di zero vuol dire che i passi che si vorrebbero far fare sono eccessivi.
Allora grazie all' uso di un blocchetto 'if' quando questo succede si entra nel subsystem 'invio al motore del max n di passi possibili'. Figura 3.57.

al suo interno viene aggiornato il valore dei passi che diventa 'passi_link0' = 'passi_link0' - 'max_passi_clock' che rimangono da fare e viene memorizzato nella variabile che invierà i dati al motore il numero massimo di passi che può essere fatto 'tensione_link0' = 'max_passi_clock' convertito dal guadagno 'kpassi2v' in volt.

Inoltre è presente uno switch che legge la variabile 'Stato' e in cui è impostato che se la variabile è maggiore di 1 allora viene inviato la tensione del numero di passi, altrimenti la tensione è a zero.

Questo switch attua quindi l' effetto dello stato di emergenza, se siamo nello stato

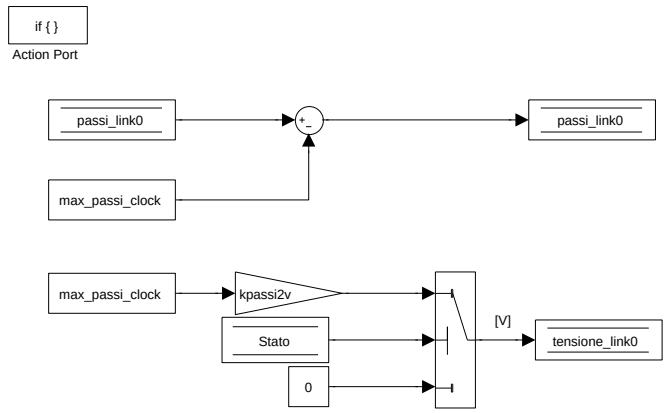

Figura 3.57: Subsystem: Calcolo controllo ⇒ controllo carrello ⇒ invio al motore del max n di passi possibili

di emergenza la tensione che deve erogare il motore passo è nulla. La tensione rimane nulla anche durante lo stato messa in marcia.
Se invece la differenza tra 'passi_link0' e 'max_passi_clock' risulta essere negativa vuol dire che i passi richiesti sono fattibili in un solo clock di simulazione e allora si entra nel subsystem 'invio al motore dei passi rimanenti'. Figura 3.58.
vengono memorizzati nella variabile che invierà la tensione al motore tutti i passi previsti convertiti in tensione dal guadagno 'kpassi2v'.
Anche qui è previsto lo switch che nello stato di emergenza impone la tensione da inviare al motore pari a zero.
Per quanto riguarda l' aggiornamento di posizione viene sommato o sottratto a 'posizione_link0_attuale' la tensione che deve essere inviata al motore riconvertita in numero di passi e considerando anche il segno.

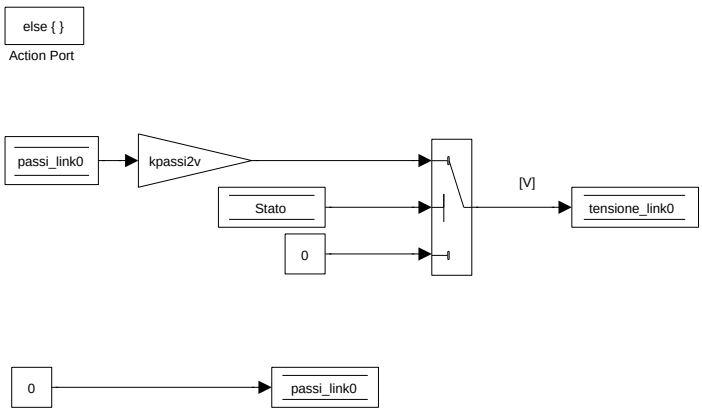

Figura 3.58: Subsystem: Calcolo controllo ⇒ controllo carrello ⇒ invio al motore dei passi rimanenti

L' aggiornamento di posizione permette di conoscere qual è la posizione del link ogni millesimo di secondo invece che ogni centesimo con i dati che vengono inviati da Matlab, e si può già aggiornare immediatamente senza aspettare di inviare realmente la tensione al motore perché lavorando questo motore in catena aperta si può solo tenere conto del riferimento di posizione voluto.
Si conclude così il controllo per la base.

Per il controllo per gli altri due link si entra nel secondo function-call subsystem 'controllo link1 e2'. Figura 3.59.
al suo interno si possono verificare due casi e dipendono dalla variabile 'controllo' inviata da Matlab.
I due casi possibili sono il controllo decentralizzato che corrisponde al valore 0

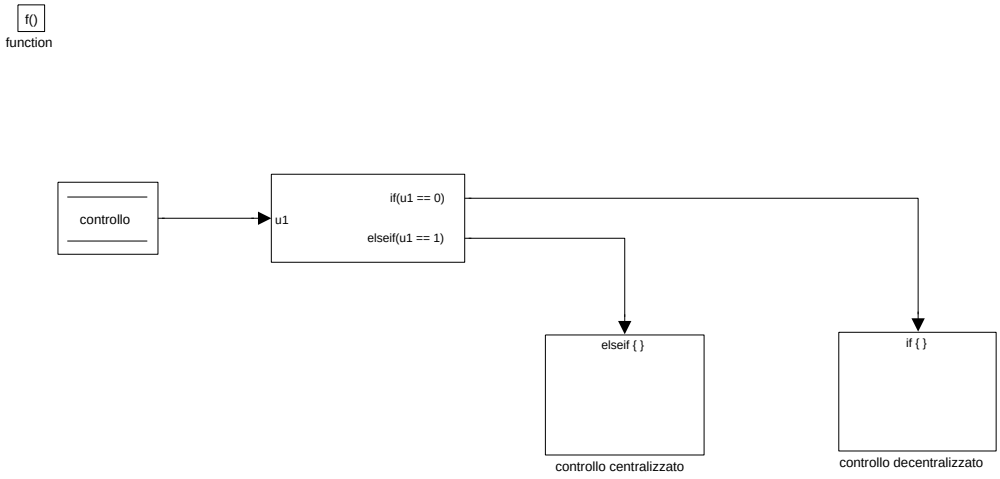

Figura 3.59: Subsystem: Calcolo controllo ⇒ controllo link1 e2

della variabile e il controllo centralizzato che corrisponde al valore 1.
La teoria che sta dietro a questi due tipi di controllo è stata spiegata nel capitolo precedente, ora viene mostrato come sono stati implementati in Simulink.
Se il tipo di controllo scelto è quello decentralizzato si entra nel subsystem 'controllo decentralizzato'. Figura 3.60.
dove si presentano due schemi retroazionati visti nella teoria.
Spieghiamo ora lo schema di retroazione per il controllo del motore 2.
L' ingresso è 'posizione_link2' che rappresenta il riferimento di posizione desiderato che è stato calcolato nel subsystem 'Calcolo riferimento' a questo viene tolto il feedback di posizione che è il valore letto dagli encoder 'encoder_link2'.
L' errore di posizione entra nel controllore, lo switch manuale permette la scelta di quale controllore utilizzare. Sono date due possibilità una funzione di trasferimento di un controllore PD, 'C'_{PD} progettata specificatamente per la funzione

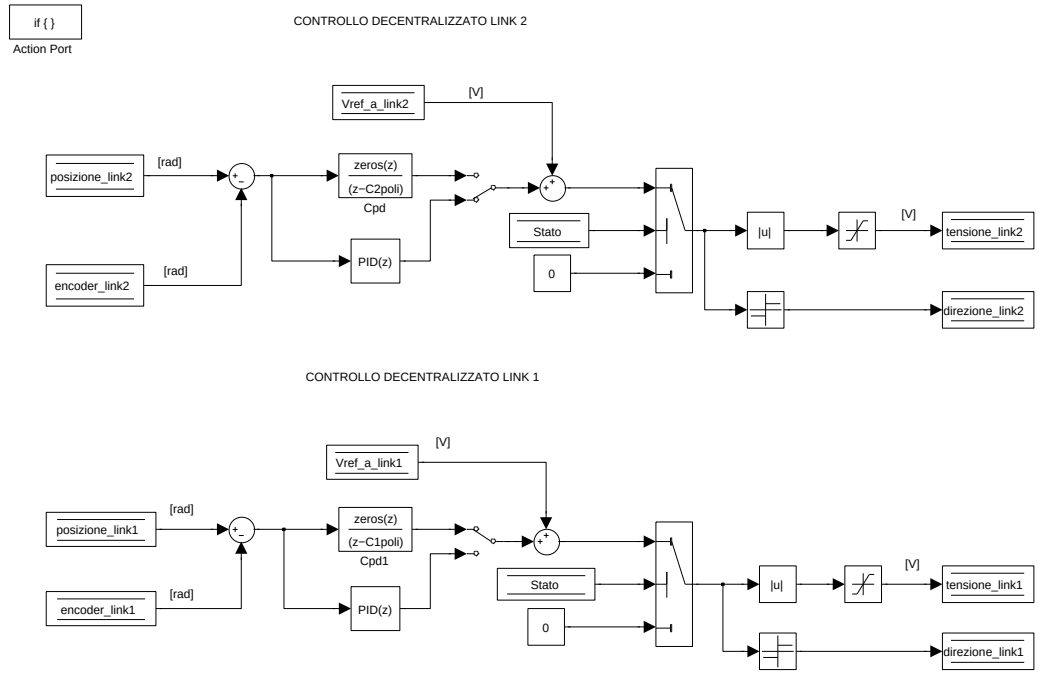

Figura 3.60: Subsystem: Calcolo controllo ⇒ controllo link1 e2 ⇒ controllo decentralizzato

di trasferimento del motore modellizzato, che insegue correttamente le traiettorie quando deve controllare la funzione che simula il motore ma soffre delle imprecisioni legate alla modellizzazione e quando viene utilizzata per il controllo del motore reale non garantisce più le stesse prestazioni.
Il controllore PID manuale permette di ritarare i suoi guadagni per tentativi e quindi osservando le risposte date dal motore fino al raggiungimento delle prestazioni volute.
Dopo il controllore viene aggiunta la tensione data dall' azione in avanti 'Vref_a_link2', la tensione che si ottiene da questa somma viene memorizzata in 'tensione_link2' ed è quella che verrà inviata al motore 2.
Viene posto uno switch, come già per il carrello, che controlla se 'Stato' 1 se la condizione non è verificata significa che si è nello stato di emergenza quindi la tensione da inviare al motore 2 è zero.

Per i motori in corrente continua una tensione pari a zero significa spegnere il motore, non mantenerlo fermo come avveniva nel motore passo. I link infatti non rimangono bloccati, ma sono non controllati. Ciò significa che possono essere spostati da forze esterne.
Questa caratteristica svantaggiosa, viene però sfruttata per quando l' emergenza viene data dall' attivazione di un finecorsa, infatti così è possibile spostare manualmente il link 2 e anche l' 1.
La tensione inviata è però divisa in modulo e segno, questo perché il micro controllore della scheda elettronica del driver è stato programmato per ricevere entrambi i dati. Inoltre viene posto un saturatore tra 0 e 5 Volt che sono i limiti di tensione accettabili per il micro controllore. Sperimentalmente si è comunque visto che l' attuazione necessaria a compiere i corretti movimenti non è mai superiore ai 5 Volt.
Lo schema si presenta analogo per il controllo del motore 1, viene solamente cambiato il nome delle variabili, il modulo della tensione è salvato nella variabile 'tensione_link1' e il rispettivo segno in 'direzione_link1'.

Se il tipo di controllo scelto è quello centralizzato si entra nel subsystem 'controllo centralizzato', in cui si trova un altro sottosistema abilitato dalla condizione 'Stato' 2. Il controllo, quindi, può avvenire in modo centralizzato solo durante il funzionamento normale, questo perché nella fase di calibrazione non si conoscono i riferimenti di velocità ed accelerazione, in quanto le traiettorie non vengono pianificate da Matlab.
Quando si è in 'Stato' = 4 e 'controllo' = 1 si accede al subsystem 'abilitazione controllo centralizzato'. Figura 3.61.
si vede chiaramente qual è il tipo di schema utilizzato: considerando posizioni, velocità ed accelerazioni reali e desiderate dei dei link 1 e 2 e l' accelerazione del link 0, tramite i calcoli di dinamica inversa si ottengono le coppie che devono erogare i due motori e convertite in tensioni vengono inviate ai driver.
Come spiegato nel capitolo precedente non vengono considerate la posizione e la velocità del link 0 perché non influiscono nel modello dinamico, e neanche le accelerazioni reali dei link 1 e 2 perché a partire da quelle desiderate e dagli errori

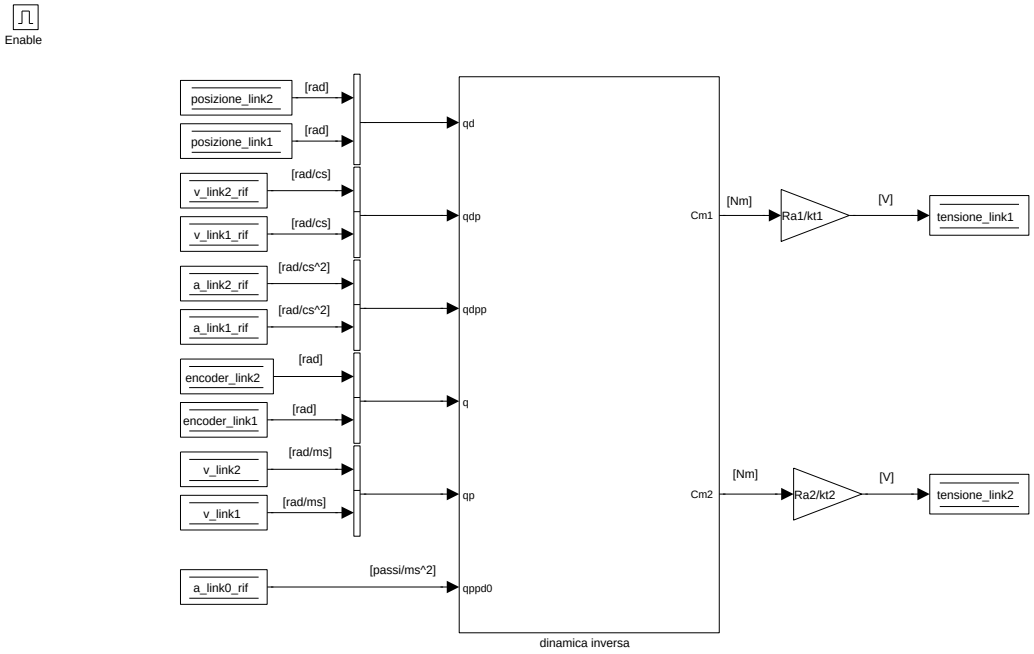

Figura 3.61: Subsystem: Calcolo controllo ⇒ controllo link1 e2 ⇒ controllo centralizzato ⇒ abilitazione controllo centralizzato

di velocità e posizione si ottengono le accelerazioni nuove da imporre.
Questi ingressi entrano nel subsystem 'dinamica inversa'. Figura 3.62.
al suo interno viene inizialmente calcolato il vettore delle accelerazioni di correzione $q\vec{p}p$, secondo l' equazione:

$$qpp = qpp_d + K_D(qp_d - qp) + K_P(q_d - q) \tag{3.10}$$

infine questo vettore delle accelerazioni insieme all' accelerazione del link 0, alle posizioni e velocità reali dei link 1 e 2 vengono inserite nel modello dinamico e forniscono le coppie che i motori devono erogare per inseguire la traiettoria, secondo le equazioni:

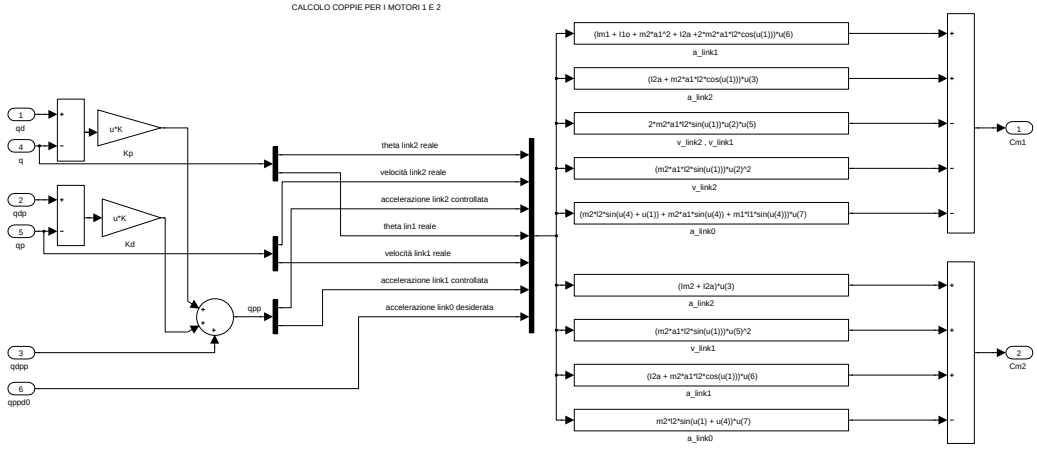

Figura 3.62: Subsystem: Calcolo controllo ⇒ controllo link1 e2 ⇒ controllo centralizzato ⇒ abilitazione controllo centralizzato ⇒ dinamica inversa

$$\begin{cases} C_{m2} = (I_{m2} + I_{2,A})\ddot{\Theta}_2 + (I_{2,A} + m_2 a_1 l_2 cos(\Theta_2))\ddot{\Theta}_1 + m_2 a_1 l_2 sen(\Theta_2)\dot{\Theta}_1{}^2 \\ \qquad -m_2 l_2 sen(\Theta_1 + \Theta_2)\ddot{x} \\ C_{m1} = (I_{m1} + I_{1,0} + m_2 a_1^2 + 2m_2 a_1 l_2 cos(\Theta_2) + I_{2,A})\ddot{\Theta}_1 + (I_{2A} + m_2 a_1 l_2 (cos\Theta_2))\ddot{\Theta}_2 \\ \qquad -2m_2 a_1 l_2 sen(\Theta_2)\dot{\Theta}_1\dot{\Theta}_2 - m_2 a_1 l_2 sen(\Theta_2)\dot{\Theta}_2{}^2 \\ \qquad -m_2(l_2 sen(\Theta_1 + \Theta_2) + a_1 sin(\Theta_1))\ddot{x} - m_1 l_1 sin(\Theta_1)\ddot{x} \end{cases} \tag{3.11}$$

3.2.7 Generatore Uscite

In questo function-call subsystem vengono inviate tutte le uscite calcolate alla scheda Sensoray 626.

Come per il primo function-call in cui si leggevano tutti gli ingressi della scheda, anche in questo caso è stato scelto di unire tutti i dati inviati, invece di inviarli

ogni volta che il segnale viene generato nei precedenti subsystem, per essere sicuri che l' invio avvenga nello stesso momento e per ottenere una maggiore leggibilità e comodità nel vedere le uscite inviate. Figura 3.63.

Le uscite inviate sono in tutto 10, 3 analogiche e 7 digitali, i blocchetti utilizzati

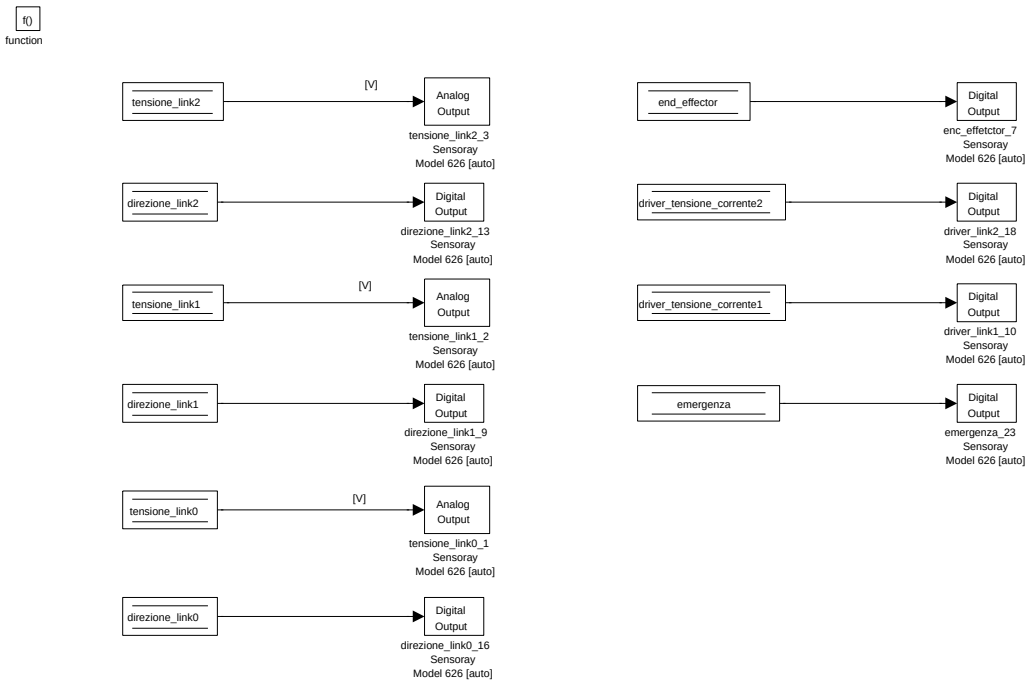

Figura 3.63: Subsystem: generatore uscite

per inviarle sono rispettivamente 'Analog output' e 'Digital output'.
I parametri da impostare in questi blocchetti sono:

- selezionare la scheda di acquisizione dati desiderata, Sensoray626;
- scegliere il 'Sample time',-1;
- scegliere l' 'Output channel' che varia da uscita ad uscita;
- per le uscite analogiche scegliere su 'block input signal' Volts, su quelle digitali 'channel mode' bit;

- impostare a zero il valore iniziale e finale onde evitare di dare segnali danneggianti in ingresso o in uscita.

Le uscite che vengono inviate sono:

- 'tensione_link2' che è il modulo della tensione per la movimentazione del motore 2. 'Output channel' = 3;
- 'direzione_link2' fornisce il segno della tensione per la movimentazione del motore 2. Segno positivo = 1, segno negativo = 0. 'Output channel' = 13;
- 'tensione_link1' che è il modulo della tensione per la movimentazione del motore 1. 'Output channel' = 2;
- 'direzione_link1' fornisce il segno della tensione per la movimentazione del motore 1. Segno positivo = 1, segno negativo = 0. 'Output channel' = 9;
- 'tensione_link0' che è il modulo della tensione per la movimentazione del motore 0. 'Output channel' = 1;
- 'direzione_link0' fornisce la direzione in cui il carrello deve muoversi. Direzione positiva = 1, direzione negativa = 0. 'Output channel' = 16;
- 'end_effector' comanda il movimento verticale della matita. La matita scrive sul foglio = 1, la matita non scrive sul foglio = 0. 'Output channel' = 7;
- 'driver_tensione_corrente2' decide se utilizzare il driver del motore 2 controllato in tensione = 1, o controllato in corrente = 0. 'Output channel' = 18;
- 'driver_tensione_corrente1' decide se utilizzare il driver del motore 1 controllato in tensione = 1, o controllato in corrente = 0. 'Output channel' = 10;
- 'emergenza' quando è = 1 fa azionare il circuito di emergenza, quando è = 0 il robot può funzionare. 'Output channel' = 23;

3.2.8 Comunicazione: invio dati matlab

In questo subsystem avviene l' invio dei dati verso il calcolatore A. Figura 3.64.
Come è stato spiegato precedentemente l' invio dei dati deve avvenire ad una

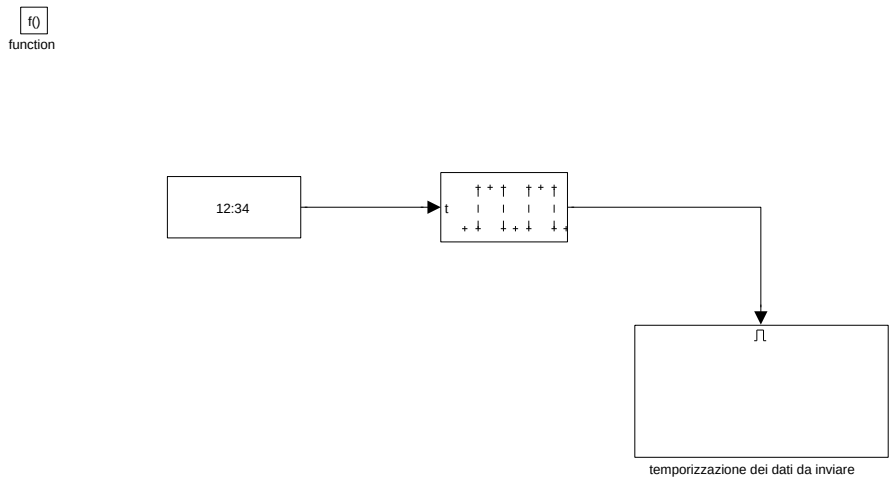

Figura 3.64: Subsystem: invio dati matlab

frequenza di 25 Hz, allora il sottosistema in cui avviene l' invio è abilitato da un' onda quadra il cui valore varia da 0 ad 1. Assume il valore 1 una volta ogni 40 cicli di simulazione e lo mantiene per un solo ciclo, considerando infatti che il clock della simulazione è 0.001 secondi si ottiene esattamente la frequenza desiderata. Tutte le volte che l' onda quadra assume il livello alto si entra nel subsystem 'temporizzazione dei dati inviati. Figura 3.65.

I dati vengono inviati tramite il pacchetto 'Packet Output' che è analogo al 'Packet Input' utilizzato per ricevere i dati, appartiene alla stessa libreria Simulink 'Real-Time Windows Target' e richiede i medesimi parametri.

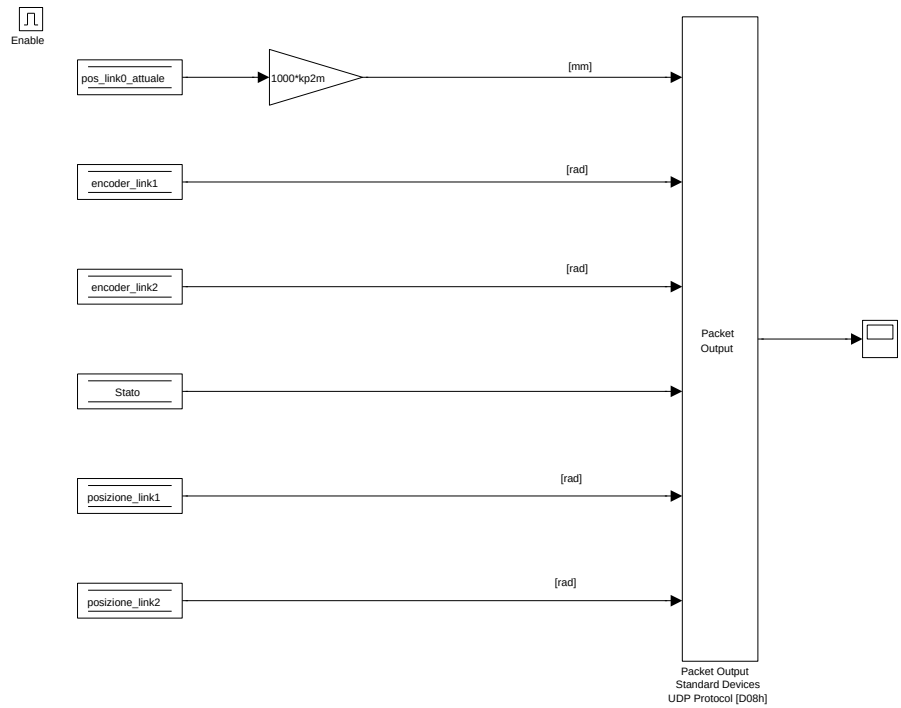

Figura 3.65: Subsystem: invio dati matlab ⇒ temporizzazione dei dati da inviare

Vengono inviati 6 dati, che in ordine sono:

- 'pos_link0_attuale'
- 'encoder_link1;
- 'encoder_link2;
- 'Stato';
- 'posizione_link1';
- 'posizione_link2';

con ovvio significato della variabili.
Vengono inviati i due riferimenti di posizione reali del link 1 e 2 e la posizione

del link 0, che sono i dati con i quali il programma Matlab va a rappresentare il manipolatore nell' interfaccia. Quindi all' utente appare il reale movimento e non quello pianificato.
Ovviamente però, il motore passo lavora in catena aperta e quindi non si ha il feedback di posizione e allora la posizione inviata e utilizzata per la rappresentazione è quella pianificata.
Si inviano inoltre le posizioni pianificate dei motori in corrente continua, per dare la possibilità al programma Matlab di tracciare nell' interfaccia la traiettoria desiderata, e la si può confrontare con il movimento reale dei link. Per il link 0 questo non ha senso che venga fatto.
Infine viene inviato lo stato della macchina a stati finiti, per permettere anche al programma Matlab di sapere in quale situazione si trova il robot.
L' invio di questo dato è indispensabile, in quanto fa parte degli algoritmi con cui viene modificata l' interfaccia.

Capitolo 4

Test

Per verificare il corretto funzionamento del programma, sono state previste diverse prove sia solo a livello software, sia solo con le schede elettroniche.
Non è stato possibile provare il programma in modo completo sul robot perché per problemi tecnici non è stato assemblato in tempo e non è stato testato il funzionamento del quadro elettrico con i motori.
Provare il programma direttamente sul robot costruito è comunque altamente sconsigliabile, in quanto si potrebbero generare problemi irreparabili sia per problemi del software che dell' elettronica.

4.1 simulazioni

Le prime simulazioni fatte sono state fatte a livello software quindi con la comunicazione dei due calcolatori, senza l' utilizzo dell' elettronica.
Si sono sostituiti tutti i blocchetti 'Analog input' e 'Digital input' presenti nel subsystem 'Ingressi Sensoray 626' per leggere gli ingressi della scheda di acquisizione con dei segnali fittizi che potessero simulare quelli reali dovuti ai comportamenti del robot.
Al posto di tutti gli ingressi digitali sono stati posti delle specie di impulsi di durata almeno 0.001 [s], affinché possano essere letti dalla simulazione, ottenuti tramite la differenza di gradini che si attivano in istanti temporali diversi. In que-

sto modo viene simulato il passaggio dallo stato logico basso allo stato logico alto o viceversa di tutti gli ingressi. Oppure nel caso in cui si volesse tenere un ingresso a livello logico solo alto o solo basso con delle costanti pari rispettivamente ad 1 o 0.

Al posto degli ingressi analogici per la lettura di posizione dei link 1 e 2, sono state poste le variabili: 'encoder_link2', 'encoder_link1', che corrispondono alle uscite delle retroazioni degli schemi di controllo in cui sono stati aggiunti la funzione di trasferimento del motore per poterlo simulare.

I tre ingressi analogici per leggere i monitor di corrente degli LMD dei tre motori dato che non sono stati utilizzati, era indifferente come venivano utilizzati. Figura 4.1.

La figura mostra solo un esempio delle diverse simulazioni che sono state prova-

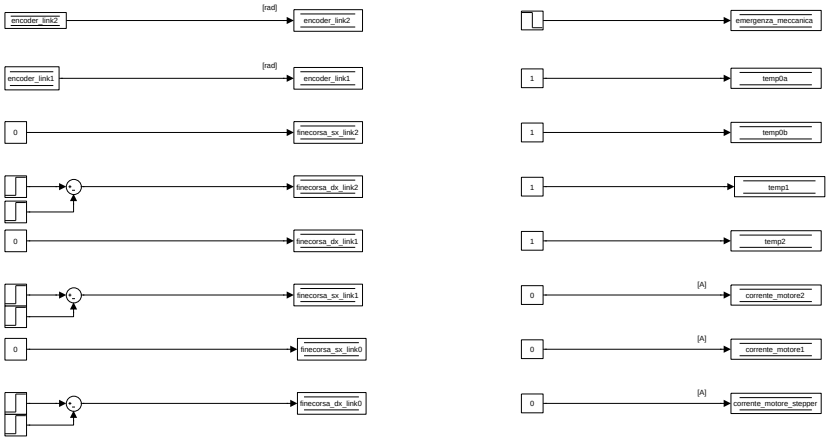

Figura 4.1: Subsystem: Ingressi Sensoray 626 per la simulazione

te.

Una prima verifica è stata osservare se lo logica della macchina funzionasse e quindi ci fosse il passaggio attraverso tutti gli stati.

Quindi sia i passaggi generati internamente al programma sia quelli comandati dal programma Matlab.

La verifica è in realtà visiva durante l' esecuzione per i cambiamenti che avvengono sull' interfaccia al cambiamento da uno stato all' altro.
Si mostra comunque il grafico della variabile 'Stato' esplicativo di quali possono essere i passaggi. Figura 4.2.

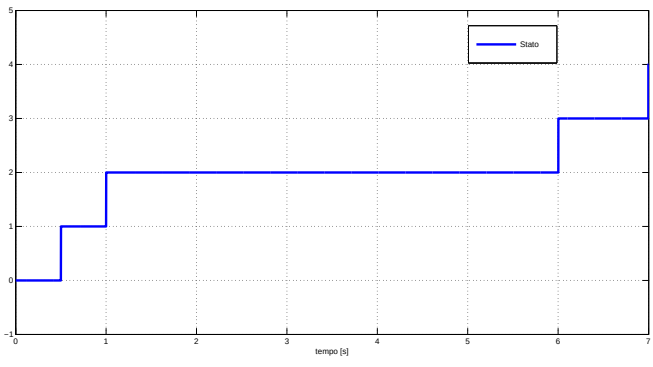

Figura 4.2: Grafico della variabile 'Stato'

Fondamentale è stato simulare l' attivazione dei finecorsa induttivi necessari per la calibrazione per poter passare dallo stato 2 allo stato 3.
A degli istanti temporali precisi l' attivazione fittizia di questi finecorsa permette tutti i passaggi della calibrazione.
Ecco il grafico dei passaggi della variabile calibrazione. Figura 4.3.
Per poter simulare il controllo sono stati modellizzati i motori e ne sono state

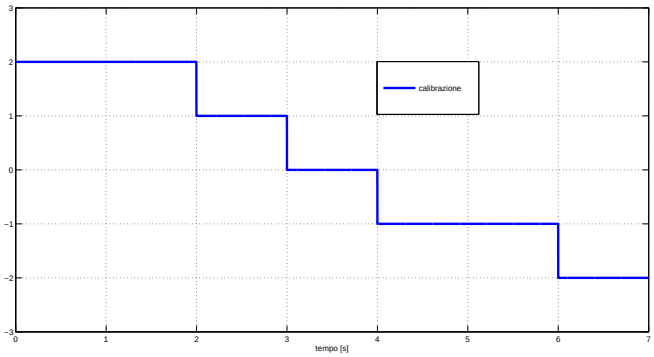

Figura 4.3: Grafico della variabile 'calibrazione'

calcolate le funzioni di trasferimento.
La funzione di trasferimento utilizzata è la seguente:

$$M(s) = \frac{Km}{1 + sTm} \cdot \frac{1}{s} \tag{4.1}$$

con Km = $\frac{1}{kv}$ e kv = back EMF constant di dimensioni [volt*s],
Tm = $\frac{RaIm}{ktkv}$ costante di tempo meccanica.
La funzione di trasferimento del motore 1 diventa:

$$M(s) = \frac{16.55}{0.004392s + 1} \cdot \frac{1}{s} \tag{4.2}$$

La funzione di trasferimento del motore 1 diventa:

$$M(s) = \frac{0.5883}{9.507e - 005s + 1} \cdot \frac{1}{s} \tag{4.3}$$

Viene sottolineato che per poter utilizzare queste funzioni di trasferimento è stato necessario discretizzarle, infatti il programma Simulink avendo un clock di esecuzione, diventa un sistema discreto.
Sono anche stati progettati dei controllori PD secondo il metodo dell' allocazione dei poli appositi per le funzioni di trasferimento dei motori, e poi discretizzata con il metodo della discretizzazione esatta.
Gli schemi di retroazione del controllo decentralizzato in questa fase di simulazione sono quindi diventati. Figura 4.4.

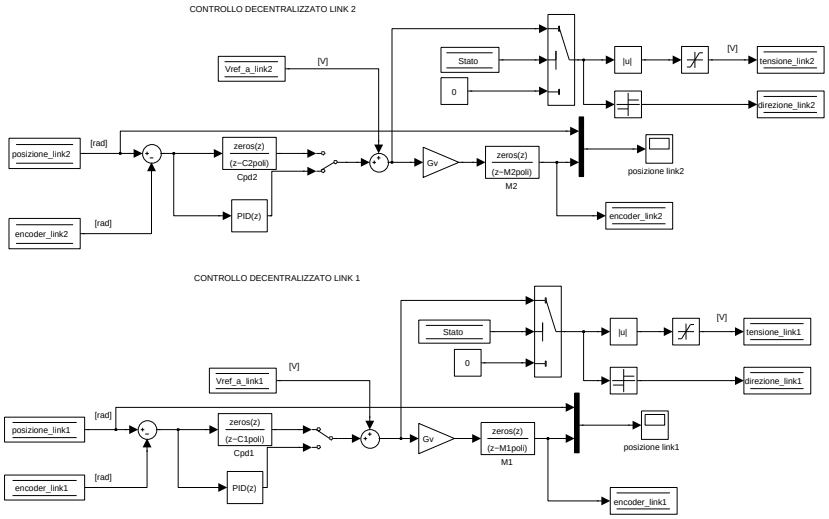

Figura 4.4: Schema del controllo decentralizzato in fase di simulazione

Durante la fase di calibrazione i risultati ottenuti dal controllo sono stato molto soddisfacenti si tratta pur sempre di una simulazione ma l' inseguimento di posizione è risultato pienamente soddisfacente.
Vengono ora mostrati i grafici delle posizione dei due link durante la fase di calibrazione mettendo a confronto la traiettoria pianificata e quella reale.
Posizione link 1. Figura 4.5.
Posizione link 2. Figura 4.6.

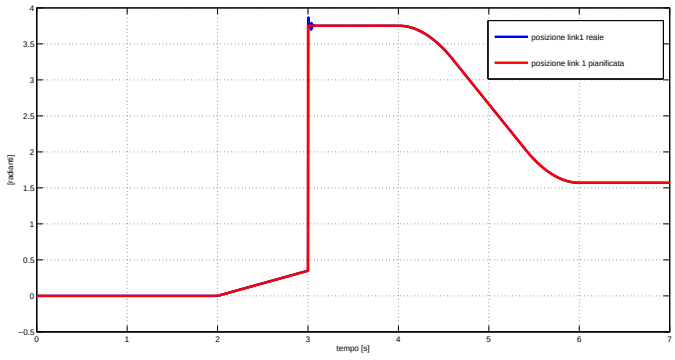

Figura 4.5: Grafico della posizione controllata del link 1 nella fase di calibrazione

Nel subsystem 'generatore uscite' sono stati sostituiti tutti i blocchetti 'Analog

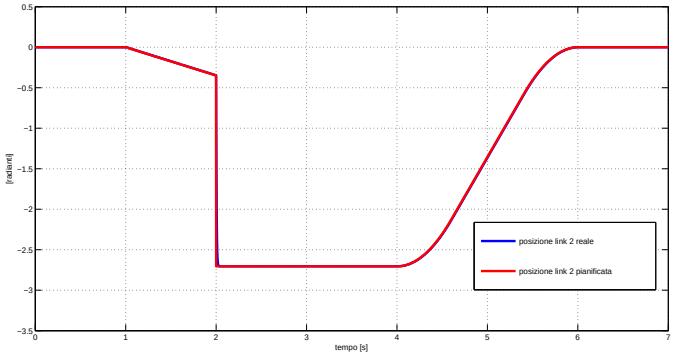

Figura 4.6: Grafico della posizione controllata del link 2 nella fase di calibrazione

Output' e 'Digtal Output' con degli semplici 'Scope' per poter osservare il comportamento delle uscite. Permettendo così di osservare i diversi casi e di andare a correggere il programma fino al corretto funzionamento.Figura 4.7.

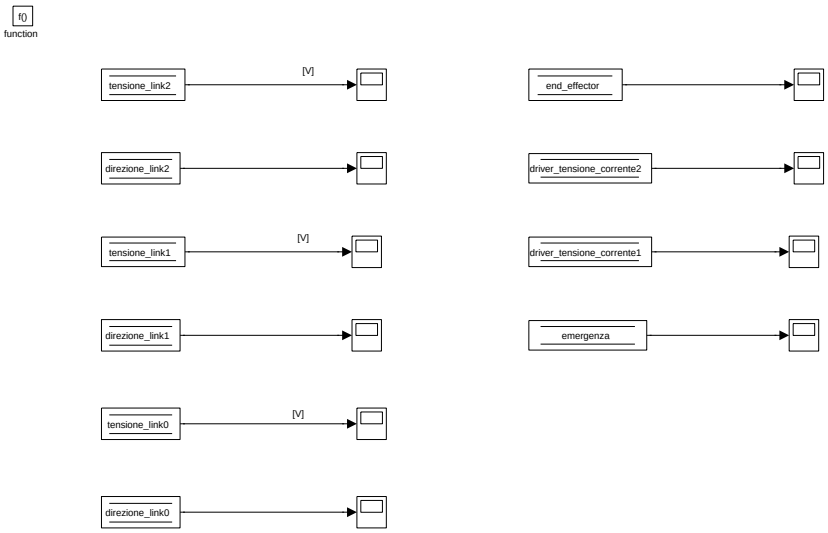

Figura 4.7: Subsystem: generatore uscite per la simulazione

Per testare la schede sono stati utilizzati dei file di prova che mandavano alle uscite analogiche e digitali dei segnali inizialmente molto semplici come delle co-

stanti, fino ad emulare delle traiettorie pianificate simili a quelle che il programma dovrà inviare.

I programmi di prova utilizzati sono come il seguente. Figura 4.8.

Nell' esempio riportato entrano delle costanti nelle uscite analogiche che pote-

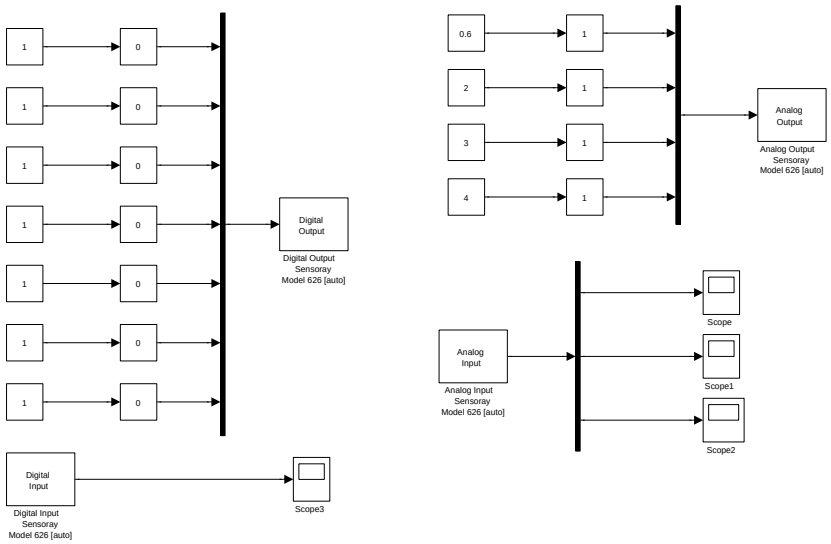

Figura 4.8: Programma utilizzato per testare le schede elettroniche e i motori

vano essere variate durante l' esecuzione della simulazione grazie all' inserimento del blocchetto 'slider gain'.

Si è potuto così osservare grazie all' utilizzo del tester se le schede elettroniche rispondevano correttamente.

Poi sono stati connessi anche i motori e si è arrivati a sostituire le costanti con delle traiettorie pianificate con profilo trapezoidale di velocità.

Si è arrivati a far rispondere i motori correttamente, a fargli inseguire quindi i riferimenti.

Non è stato però possibile testare il programma perché si è dovuto aspettare che venissero montate anche tutte le sicurezze, come i finecorsa, e inoltre è stato aggiunto all' ultimo momento il quadro e quindi è stato prima necessario testare il funzionamento di quello.

Conclusioni

In questo progetto di tesi è stato realizzato un programma per il controllo di manipolatore planare a tre gradi di libertà utilizzando il software *Simulink*©.
Il lavoro svolto comprendeva questi principali punti:

- progettazione degli schemi di controllo;
- gestione degli Input e Output col sistema reale;
- comunicazione dati con un altro calcolatore, tramite protocollo UDP, per la ricezione delle traiettorie pianificate.

I risultati ottenuti sono stati positivi.
La comunicazione è stata realizzata correttamente riuscendo a gestire i problemi causati dall' inaffidabilità del protocollo.
Le traiettorie ricevute vengono interpolate correttamente dal programma, la macchina a stati finiti cambia il suo stato correttamente sia per i passaggi interni al programma sia per quelli imposti via comunicazione.
L' invio delle posizioni controllate, rappresentative quindi di quelle reali, all' interfaccia Matlab vengono visualizzate correttamente e corrispondono al corretto funzionamento.
Mancano però le prove sperimentali del programma col robot.
Questo è dovuto a problemi tecnici che hanno portato all' assemblaggio del robot, e al collegamento del quadro elettrico in ritardo rendendo impossibile effettuare alcun test.
Sono stati comunque testati i motori singolarmente, simulando i riferimenti inviati dal programma ed hanno risposto correttamente.

Inoltre sono stati simulati tutti gli Input e Output e le risposte del programma sono state coerenti con quanto desiderato.
Questo non significa il raggiungimento del controllo dei motori, dato il carico inerziale mancante, e il sicuro funzionamento del programma per tutti le ulteriori complicazioni introdotte dal sistema reale completo.
Ma fa ben sperare per un proseguio del lavoro svolto.

Appendice A

Analisi dinamica inversa del manipolatore

Questa analisi ci permette note le posizioni, le velocità e le accelerazioni di calcolare le coppie che devono erogare i motori per mantenere le traiettorie desiderate. Consideriamo inizialmente i link 1 e 2 e successivamente studieremo anche il contributo dato dal link 0.
Per calcolare le coppie necessarie partiamo ipotizzando i due link separati:
dove:
$\vec{R_1}$ = forza di reazione vincolare che vincola il primo membro al carrello e si ripercuote opposta in quest' ultimo;
$\vec{R_2}$ = forza di reazione vincolare che vincola il secondo membro ad avere A nel membro 1, e questa forza si ripercuote opposta nel primo membro e prende il nome di azione mutua;
C_2 è la coppia che agisce sul membro 2 e si ripercuote opposta nel membro 1;
C_1 è la coppia che agisce sul membro 1 e si ripercuote opposta nel membro 0;
G_1 e G_2 sono i baricentri, e le masse dei membri vengono chiamate m1 ed m2.
Si possono così ottenere le equazioni dinamiche dei due membri:

$$\begin{cases} I_{1,0}\ddot{\Theta}_1 = C_1 - C_2 + [\vec{OA} \wedge (-\vec{R_2})] \times \vec{k} \\ I_{2,G_2}\ddot{\Theta}_{1,2} = C_2 + [\vec{G_2A} \wedge \vec{R_2}] \times \vec{k} \end{cases} \tag{A.1}$$

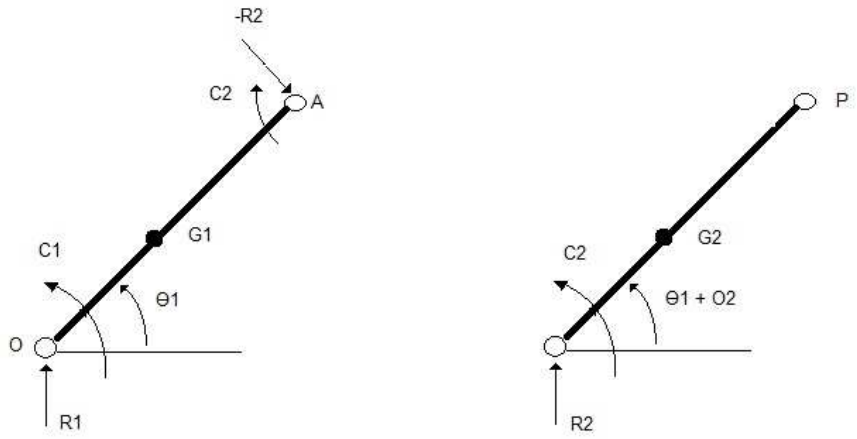

Figura A.1: Schema delle forze e delle coppie che agiscono sui link 1 e 2

dove:

$I_{1,0}$ = inerzia del membro 1 rispetto al punto O;

I_{2,G_2} = inerzia del membro 2 rispetto al suo baricentro.

Diventa così possibile calcolare le due coppie:

$$\begin{cases} C_1 = I_{1,0}\ddot{\Theta}_1 + C_2 - \vec{M}^O_{-R_2} \times \vec{k} \\ C_2 = I_{2,G_2}\ddot{\Theta}_{1,2} + \vec{M}^{G_2}_{R_2} \times \vec{k} \end{cases} \tag{A.2}$$

$\vec{M}^O_{-R_2}$ e $\vec{M}^{G_2}_{R_2}$ sono rispettivamente il momento che la forza $-R_2$ esercita nel punto O e il momento che la forza R_2 esercita su G_2.

Sapendo che:

$$m_2 \vec{a_{G_2}} = \vec{R_2} \tag{A.3}$$

calcolato $\vec{a_{G_2}}$ si può calcolare il momento di forza:

$$\vec{M}^{G_2}_{R_2} = m_2 \vec{G_2 A} \wedge [\vec{\omega_1} \wedge (\vec{\omega_1} \wedge \vec{OA})] + m_2 \vec{G_2 A} \wedge (\vec{\alpha_1} \wedge \vec{OA}) + m_2 \vec{G_2 A} \wedge [(\vec{\alpha_1} + \vec{\alpha_2}) \wedge \vec{G_2 A}] \tag{A.4}$$

Ora è possibile calcolare le coppie in funzione solo delle posizioni, velocità ed accelerazioni.

Il prodotto con il vettore $\vec{k}$ restituisce i moduli dei prodotti vettoriali in quanto siamo ortogonali al piano.
Le coppie ottenute sostituendo sono:

$$\begin{cases} C_2 = I_{2,A}\ddot{\Theta}_2 + (I_{2,A} + m_2 a_1 l_2 cos(\Theta_2))\ddot{\Theta}_1 + m_2 a_1 l_2 sen(\Theta_2)\dot{\Theta}_1{}^2 \\ C_1 = (I_{1,0} + m_2 a_1^2 + 2m_2 a_1 l_2 cos(\Theta_2) + I_{2,A})\ddot{\Theta}_1 + (I_{2A} + m_2 a_1 l_2 (cos\Theta_2))\ddot{\Theta}_2 \\ \quad -2m_2 a_1 l_2 sen(\Theta_2)\dot{\Theta}_1\dot{\Theta}_2 - m_2 a_1 l_2 sen(\Theta_2)\dot{\Theta}_2{}^2 \end{cases} \tag{A.5}$$

Queste sono le coppie che agiscono rispettivamente nel link 2 e nel link 1.
Per calcolare la coppia che devono erogare esattamente i due motori si utilizza il principio dei lavori virtuali:

$$\begin{cases} C_{m2}\delta\Theta_{m2} = I_{m2}\ddot{\Theta}_{m2}\delta\Theta_{m2} + C_2\frac{\delta\Theta_2}{\eta_2} \\ C_{m1}\delta\Theta_{m1} = I_{m1}\ddot{\Theta}_{m1}\delta\Theta_{m1} + C_1\frac{\delta\Theta_1}{\eta_1} \end{cases} \tag{A.6}$$

dividendo poi le due equazioni rispettivamente per $\delta\Theta_{m2}$ e $\delta\Theta_{m1}$ e considerando che i motori lavorano in presa diretta quindi sia i rapporti di trasmissione sia i rendimenti sono pari ad 1, si ottiene:

$$\begin{cases} C_{m2} = (I_{m2} + I_{2,A})\ddot{\Theta}_2 + C_2' \\ C_{m1} = (I_{m1} + I_{1,0} + m_2 a_1^2 + 2m_2 a_1 l_2 cos(\Theta_2) + I_{2,A})\ddot{\Theta}_1 + C_1' \end{cases} \tag{A.7}$$

Si fa presente che il secondo termine in entrambe le equazioni rappresenta la coppia resistente che agisce nel rispettivo motore.

Ora viene aggiunta l' influenza della base mobile.
La posizione e la velocità della base mobile non vanno ad influenzare la coppia da erogare, ma la sua accelerazione esercita una forza in direzione opposta su entrambi i membri. Queste forze sono rappresentate in figura come se agissero sul baricentro dei due membri.
Ricordando la prima legge di Newton:

F = m a

l' accelerazione è quella della base mobile, e la massa quella dei membri. La coppia è data dalla forza per il braccio, quindi il contributo del carrello in termini

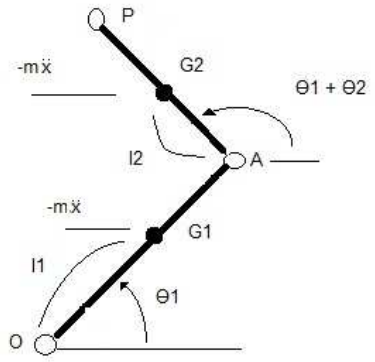

Figura A.2: Schema manipolatore con le forze agenti dovute all' accelerazione della base mobile

di coppia per i due motori é:

$$\begin{cases} C_{2,0} = -m_2 l_2 sen(\Theta_1 + \Theta_2)\ddot{x} \\ C_{1,0} = -m_2(l_2 sen(\Theta_1 + \Theta_2) + a_1 sin(\Theta_1))\ddot{x} - m_1 l_1 sin(\Theta_1)\ddot{x} \end{cases} \tag{A.8}$$

Sono state determinate tutte le componenti per il calcolo delle coppie, quindi le coppie che il motore 2 e il motore 1 devono erogare per inseguire le traiettorie sono:

$$\begin{cases} C_{m2} = (I_{m2} + I_{2,A})\ddot{\Theta}_2 + (I_{2,A} + m_2 a_1 l_2 cos(\Theta_2))\ddot{\Theta}_1 + m_2 a_1 l_2 sen(\Theta_2)\dot{\Theta}_1^{\,2} \\ \qquad -m_2 l_2 sen(\Theta_1 + \Theta_2)\ddot{x} \\ C_{m1} = (I_{m1} + I_{1,0} + m_2 a_1^2 + 2m_2 a_1 l_2 cos(\Theta_2) + I_{2,A})\ddot{\Theta}_1 + (I_{2A} + m_2 a_1 l_2 (cos\Theta_2))\ddot{\Theta}_2 \\ \qquad -2m_2 a_1 l_2 sen(\Theta_2)\dot{\Theta}_1\dot{\Theta}_2 - m_2 a_1 l_2 sen(\Theta_2)\dot{\Theta}_2^{\,2} \\ \qquad -m_2(l_2 sen(\Theta_1 + \Theta_2) + a_1 sin(\Theta_1))\ddot{x} - m_1 l_1 sin(\Theta_1)\ddot{x} \end{cases} \tag{A.9}$$

Appendice B

File Parametri

In questa appendice viene solo riportato il file Matlab in cui vengono definite delle varibili utilizzate nel programma.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%                         PARAMETRI                       %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc
clear all
close all

T = 0.001 ;                    % [s] periodo della simulazione,utilizzato anche per il compionamento della discretizzazione

% CONVERSIONI - A
kr2g = 180/pi;                 % conversione da radianti a gradi
kg2r = pi/180;                 % conversione da gradi a radianti

% DATI PER LA PIANIFICAZIONE DELLA TRAIETTORIA IN CALIBRAZIONE
T_calibrazione = 2             % [s] tempo del movimento
ta = 0.6                       % [s] tempo di accelerazione
td = 0.6;                      % [s] tempo di decelerazione

% CALIBRAZIONE
incremento_calibrazione = 0.02*kg2r ; % [rad] incremento angolare dato ai link2 e 1 nell' algoritmo di calibrazione
passi_calibrazione = 1 ;              % [passi/ms] incremento di passi al millisecondo per il motore passo in fase di calibrazione

% EMERGENZE
max_no_lettura_encoder = 50;   % n massimo di volte accettabile per cui la lettura encoder non cambia valore, ma i riferimenti di po
v_max_link2 = 4                % [rad/sec] velocità massima consentita al link2
v_max_link1 = 4                % [rad/sec] velocità massima consentita al link1

% DATI ROBOT
mtot = 2.360;                  % [kg] massa totale del robot

% GUIDA LINEARE
fine_guida = 3000;             % numero di passi da fine corsa sx a fine corsa dx della guida
meta_guida= fine_guida/2;      % numero di passi tra un fine corsa della guida lineare e il centro della guida
```

Figura B.1: Immagine file: 'Parametri' utilizzato per caricare le costanti utilizzate nel programma. Parte A.

```
% LINK 2
a2 = 0.1;                       % [m] lunghezza del link2
l2 = 0.078;                     % [m] distanza dal baricentro del link2 al giunto di colegamento con il link1
delta_fc_cal_link2 = 155*kg2r; % [rad] delta di posizione che c'è tra un fine corsa induttivo del link2 e la posizio

% LINK 1
a1 = 0.12;                      % [m] lunghezza del link1
l1 = 0.0003;                    % [m] distanza dal baricentro del link1 al giunto di colegamento con il link0
delta_fc_cal_link1 = -125*kg2r;% [rad] delta di posizione che c'è tra un fine corsa induttivo del link1 e la posizio

% DRIVER
Gv = 8 ;                        % guadagno driver in tensione
ki = 5/6 ;                      % guadagno nell' anello di corrente driver
f_max_passi = 5000;             % [passi/secondo] frequenza massima di passi che puo fare il motore stepper in un sec
max_passi_clock = f_max_passi/1000      % [passi/msecondo] massima frequenza di passi che puo fare il motore steppe

% PIC
ten_max_pic = 5;                % [V] tensione massima dei pic
ten_min_pic = 0;                % [V] tensione minima dei pic
kpassi2v = ten_max_pic/max_passi_clock;  % conversione dei passi che deve fare il motore stepper in tensione

% ENCODER
tacche_giro = 4000;             % numero di tacche dell' encoder per giro
ke2r = 2*pi/tacche_giro;        % conversione del numero dato dalla lettura degli encoder alla posizione in radianti

% DATI PER IL CALCOLO DELLE COPPIE PER IL CONTROLLO CENTRALIZZATO
I2a = 0.00112;                  % [kg*m^2] inerzia del link2 rispetto al giunto di unione con il link1
I1o = 0.00422;                  % [kg*m^2] inerzia del link1 rispetto al giunto di unione con il link0
I2g = I2a   m2*l2^2;

% CONVERSIONI - D
kr2v_link2 = 5/delta_fc_cal_link2;      % conversione da radianti a volt link2
kr2v_link1 = 5/delta_fc_cal_link1;      % conversione da radianti a volt link1
kg2v_link2 = kr2v_link2*kg2r;           % conversione da  gradi a volt link2
kg2v_link1 = kr2v_link1*kg2r;           % conversione da  gradi a volt link1
kp2m = 0.0001;                          % conversione da passi a metri
```

Figura B.2: Immagine file: 'Parametri' utilizzato per caricare le costanti utilizzate nel programma. Parte B.

```
% DATI PRIMO MOTORE
m1 = 0.495;                          % [kg] massa del motore 1
kt1 = 0.0603;                        % [Nm/A] costante di coppia
kv1 = 1000/158*60/(2*pi*1000);       % [volt*s] back_EMF constant
Ra1 = 1.16;                          % [ohm] terminal resistance
Im1 = 138*10^-7;                     % [kgm^2] inerzia del motore

km1 = 1/kv1;
Tm1 = Ra1*Im1/(kt1*kv1)              % [s] costante di tempo

M1 = tf(km1, [Tm1 1 0])              % funzione di trasferimento del motore1

M1d = c2d(M1,T,'tustin')             % funzione di trasferimento del motore1 discreta
M1poli = pole(M1d)                   % poli
[M1zeri, gainM1] = zero(M1d)         % zeri e guadagno

% DATI SECONDO MOTORE
m2 = 0.170;                          % [kg] massa del motore 2
kt2 = 0.0538 ;                       % [Nm/A] costante di coppia
kv2 = 178*60/(2*pi*1000);            % [volt*s] back_EMF constant
Ra2 = 2.52 ;                         % [ohm] terminal resistance
Im2 = 34.5*10^-7;                    % [kgm^2] inerzia del motore
km2 = 1/kv2;
Tm2 = Ra2*Im2/(kt2*kv2)              % [s] costante di tempo

M2 = tf(km2, [Tm2 1 0])              % funzione di trasferimento del motore
M2d =  c2d(M2,T,'tustin')            % funzione di trasferimento del motore discretizzata
M2poli = pole(M2d)                   % poli
[M2zeri, gainM2] = zero(M2d)         % zeri e guadagno
```

Figura B.3: Immagine file: 'Parametri' utilizzato per caricare le costanti utilizzate nel programma. Parte C.

Bibliografia

[1] S. Co., *Instruction Manual Sensoray Model 626*, January 2004.

[2] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotica*. Milano: McGraw-Hill, Terza Edizione.