

ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DIPARTIMENTO

Automation Engineering

TESI DI LAUREA MAGISTRALE

in

Industrial Robotics

Programmazione PLCopen di un robot Comau e di macchine automatiche

CANDIDATO

Roberto Badalamenti

RELATORE:

Prof. Claudio Melchiorri

Anno Accademico 2017/18

Sessione II

INDICE

INTRODUZIONE	4
1. CONFIGURAZIONE HARDWARE.....	6
1.1 Bus di campo POWERLINK	7
1.2 Azionamenti ACOPOS P3	10
1.3 Encoder EnDat 2.2 Heidenhain	15
2. CONFIGURAZIONE SOFTWARE	21
2.1 Definizione dei moduli hardware nel software	22
2.2 Configurazione hardware: unità, POWERLINK e vincoli.....	23
2.3 Variabili software e parametri	25
2.4 Programmi ST e architettura del movimento	26
2.5 PLCopen e lo standard IEC 61131	28
2.5.1 Introduzione allo standard IEC 61131-3	30
2.5.2 Elementi comuni nei linguaggi dello standard IEC 61131-3	31
2.5.3 Funzioni e blocchi funzione standardizzati.....	36
2.5.4 I linguaggi della norma IEC 61131-3	41
3. PIANIFICAZIONE DEL MOTO.....	49
3.1 Sincronizzazione nastro trasportatore-robot.....	50
3.2 Traiettoria continua	53
3.3 Pianificazione del movimento con più "camme elettroniche"	55
3.3.1 Cam profile automat.....	56
3.3.2 Traiettoria alternata	57
3.4 Confronto tra traiettorie e risultati di simulazione	59
3.5 Modello dinamico per la compensazione feedforward della coppia	64

4. APPLICAZIONE SAFETY	67
4.1 OpenSAFETY	69
4.2 Moduli Input/Output Safe	72
4.3 Controllore del moto Safety	76
4.4 Controllore “SafeLogic”	78
4.5 SafeDESIGNER	79
5. INTERFACCIA UOMO-MACCHINA IUM (HMI) E FUNZIONALITÀ .	82
5.1 Funzionalità di base	83
5.2 Funzionalità aggiuntive	87
5.3 Altre funzionalità e informazioni	89
BIBLIOGRAFIA	93

INTRODUZIONE

Lo scopo di questo lavoro è di presentare come un manipolatore possa essere utilizzato in un'applicazione industriale, quindi come possiamo programmarlo e quali problematiche potrebbero essere presenti nell'introduzione di un robot in una macchina automatica. L'applicazione industriale d'interesse consiste nel riempire, utilizzando un apposito end-effector per il robot, composto da sei "cannucce", dei flaconi che corrono lungo un nastro trasportatore, con ad esempio della soluzione medicinale. Questo delicato compito ci porterà ovviamente a forti vincoli dovuti a ragioni igieniche. Infatti, l'estremità delle cannucce del robot, al fine di evitare contaminazioni, non potranno toccare né il liquido né le pareti dei flaconi. Per questo motivo, il contatto tra cannucce e bottiglie è severamente vietato e ciò comporta quindi il bisogno di un perfetto sincronismo tra cannucce e flaconi, ovvero, tra i sei motori del robot e il motore che fa muovere il nastro trasportatore. Dopo una prima parte del lavoro dedicata a illustrare l'hardware necessario per l'applicazione, tra cui le principali componenti di questo modulo di linea industriale troviamo un manipolatore a sei gradi di libertà, "Comau Racer 5-080", affacciato ad un segmento di nastro trasportatore con un proprio motore sincrono, PLC di sicurezza e non, moduli di input e output, azionamenti elettrici per i motori, bus di campo e così via, siamo passati alla sua implementazione nel ambiente di sviluppo a cui seguirà una rapida spiegazione del protocollo Powerlink utilizzato nella comunicazione e infine una descrizione del software creato, illustrando le varie strutture e variabili necessarie per l'applicazione, nonché dando uno sguardo a quello che è lo standard di riferimento nella programmazione PLC, analizzando i vari linguaggi di programmazione e gli elementi della norma IEC 61131. Infatti, il software creato in "Automation Studio" (l'ambiente di sviluppo software di proprietà B&R.), che controlla il robot e il nastro trasportatore, è stato scritto in "testo strutturato" e per motivi di portabilità è stato scelto di utilizzare solamente funzioni che fanno parte dello standard PLCopen, così da poter in seguito riutilizzare il codice anche in qualche altro contesto con robot anche diversi.

Per quanto riguarda la pianificazione della traiettoria che deve eseguire il TCP (“Tool Central Point”) del robot, si è definita la traiettoria nello spazio per ciascuno dei tre assi (x-y-z) e delle tre rotazioni (attorno a x-y-z). Quindi da questi punti nello spazio verranno generati, a seguito, di una trasformazione cinematica inversa le posizioni corrispondenti dei sei motori del robot, che vengono azionati poi indipendentemente l'uno dall'altro. Infine si è testato il software sul sistema reale e corretto i vari bug incontrati. Uno dei principali problemi risolti è stata la sincronizzazione tra il nastro trasportatore e il TCP del robot, questa criticità è stata gestita analizzando la comunicazione e il trasferimento dei dati. Vale a dire, analizzando il bus di campo POWERLINK e il suo protocollo, utilizzato per la comunicazione tra PLC e azionamenti, e quindi riducendo il più possibile il percorso dei dati contenenti i target di posizione, regolando le differenze di ritardo nella ricezione dei target di posizione nei sette motori e utilizzando funzioni che prestano particolare attenzione alla gestione dei dati real-time, al fine di garantire così un perfetto sincronismo nel processo di riempimento. Infine, l'ultima parte dell'elaborato sarà dedicata all'applicazione di sicurezza sviluppata, in cui le principali funzionalità sono il controllo dei limiti di posizione, velocità, coppia e così via, su ciascun asse, il pulsante di sicurezza per il sistema spegnimento e introduzione di un'area di lavoro "Safe" in cui il robot può lavorare senza la possibilità di creare situazioni di pericolo. A cui seguirà una rapida descrizione della semplice interfaccia sviluppata sempre in Automation Studio, utilizzata per semplificare il controllo della macchina anche in fase di debug.

1. CONFIGURAZIONE HARDWARE

L'hardware principale per questa applicazione è costituito dal robot "Comau Racer 5-0.80" e dai dispositivi elettronici B&R, che sono principalmente il PLC, i moduli I/O e gli azionamenti elettrici. Entrando nel dettaglio dell'hardware utilizzato abbiamo: un motore a quattro poli sincrono, "8LSA35.DA030S300-3a" IP64, in grado di garantire una velocità nominale e una coppia rispettivamente di 3.000 rpm e 2.100 Nm utilizzato per far muovere il nastro trasportatore. Il robot "Comau Racer 5-0.80", un manipolatore a sei gradi di libertà, con estensione orizzontale massima di 809 mm e portata utile di 5 Kg, con grado di protezione ai liquidi e alle polveri IP 54 e con un range di movimento, ad esempio, per il primo giunto di ± 170 gradi con velocità massima di 360 deg/s equivalente a 60 rpm che può garantire una velocità lineare massima a 502 mm di distanza dalla base (ovvero l'attuale distanza fra la base del robot e il nastro trasportatore), di circa $3,15 \frac{m}{s}$. Il motore menzionato e i sei motori del robot sono controllati mediante tre azionamenti elettrici, in particolare tre servo azionamenti ACOPOS P3 B&R-8EI8X8MWT50-0700-1, collegati a un PLC B&R-5PC900_TS77_09 con scheda CPU Intel Celeron 1020E 2.2 GHz dual-core. Mentre per la gestione degli I/O abbiamo, due moduli di

ingresso digitali "Safe" B&R-X20SI4100 che assicurano secondo lo standard internazionale IEC-61131-2 una gestione sicura dei segnali di ingresso, oltre a questo un modulo di uscita "Safe": B&R-X20SO4110, affiancati ad altri due moduli di ingresso digitali non-Safe B&R-

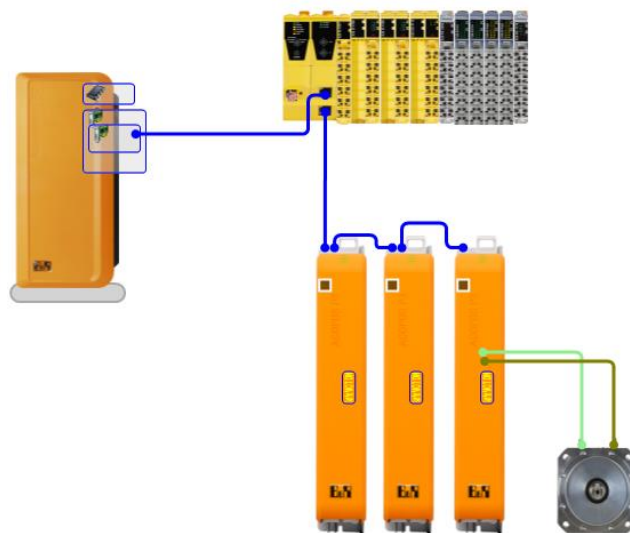


Figura 1.1 Configurazione hardware "physical view"
"Automation Studio"

X20DIF371 e altri due moduli di uscita digitali non "Safe" B&R-X20DOF322. Fra l'hardware utilizzato abbiamo inoltre, il modulo B&R-X20PS3300 per

l'alimentazione interna del bus di campo e i moduli di I/O. Infine abbiamo anche un modulo dedicato all'alimentazione e al monitoraggio dell'ingresso encoder B&R-X20DC1176 utilizzato per il motore del nastro trasportatore. La configurazione hardware risultante può essere apprezzata dalla "physical view" in "Automation Studio" visibile in figura 1.1. Dove la connessione in blu rappresenta il bus di campo POWERLINK utilizzato per la comunicazione delle informazioni. Questo ultimo basato su un protocollo di comunicazione open source deterministico in tempo reale, che sfrutta come supporto la connessione Ethernet ed è in grado di raggiungere una velocità di trasmissione di 100 Mbit e una precisione di sincronizzazione di ± 100 ns. [1] [2] [3]

1.1 Bus di campo POWERLINK

Powerlink è un protocollo real-time deterministico per il bus di campo B&R basato su Ethernet standard (IEE 802.3). Nelle condizioni operative, Powerlink utilizza l'intero frame dello standard Ethernet (Preambolo, indirizzo di destinazione, indirizzo di origine, tipo, dati e sequenza di controllo dei frame). Il protocollo supporta le varie topologie di rete come ad esempio ad anello chiuso albero, stella, daisy chain e così via. Nella rete powerlink ogni componente della rete ha il proprio

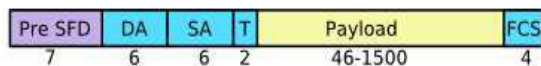


Figura 1.2 Frame Ethernet standard

Abbreviation	Name	Source
SoC	Start of Cycle	MN
PReq	Poll request	MN
PRes	Poll response	CN/MN
SoA	Start of Asynchronous	MN
ASnd	Asynchronous Send	CN/MN

Figura 1.3 Powerlink frame

"numero di nodo" che può essere assegnato sia manualmente che in modo dinamico, offrendo una maggiore flessibilità nella gestione del componente, funzionalità molto utile ad esempio in caso di sostituzione dei componenti. Il nodo nella rete può essere di due tipi: Managing Node o Controller Node (MN o CN). Il MN impone ai componenti della rete, il tempo di ciclo utilizzato e controlla la comunicazione del ciclo. Questo di

solito può essere un PLC o un PC industriale, mentre i CN sono il resto dei nodi della rete Powerlink (inclusa la CPU), come azionamenti, encoder, moduli di I/O e così via. Quindi una volta stabilita la topologia, all'inizio di ogni ciclo l'MN invia a tutti i componenti della rete un frame con un messaggio SoC (Start of Cycle) in multicast, ovvero contemporaneamente a tutti i nodi della rete, senza dati ma solo per scopi di sincronizzazione.

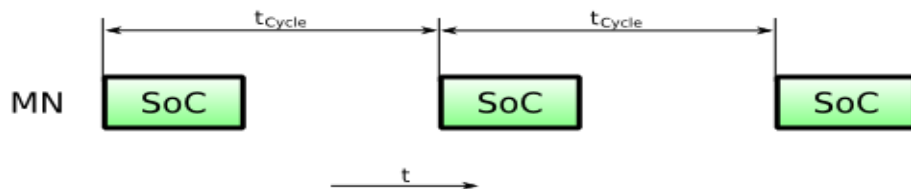


Figura 1.4 Fase iniziale della comunicazione

Una volta che il MN ha inviato un SoC a ciascuno dei CN nella rete, invia ad ogni CN un messaggio PReq (Poll request) in unicast, ovvero solo ad un singolo nodo, quindi ogni CN risponderà al MN in multicast con un messaggio di PRes (Poll response), in questo modo poiché il CN risponde solo dopo un messaggio di PReq da un MN evitiamo le possibili collisioni di dati all'interno del mezzo di comunicazione. Inoltre, questo metodo di comunicazione fornisce ad un nodo CN

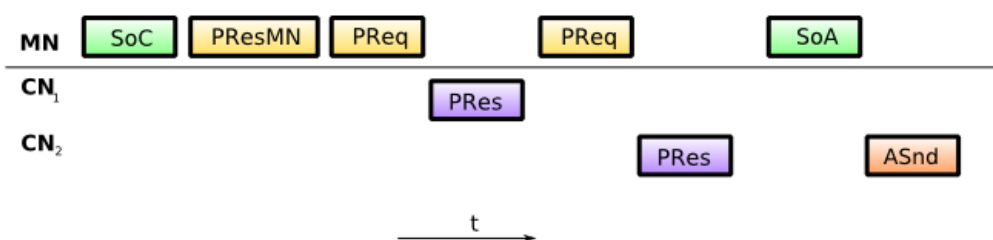


Figura 1.5 Trasferimento asincrono dei dati

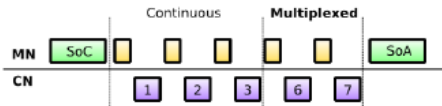
della rete, i mezzi necessari per una comunicazione tra i CN stessi (cross communication), ovvero, senza passare attraverso il Managing Node. Per quanto riguarda i dati scambiati nella comunicazione invece, se non sono time-critical, quindi non hanno particolari esigenze real-time, possono essere trasferiti in un frame Ethernet. Per fare ciò ogni CN può, con un PRes, informare il MN che vuole inviare dei dati asincroni. Quindi il MN decide quale dei CN può inviare i dati (solo un CN per periodo) e lo comunica al nodo in questione inviando ad esso un pacchetto SoA (Start of Asynchronous), quindi il CN ricevuto il SoA dal MN può

effettivamente inviare i dati asincroni in un pacchetto ASnd (Asynchronous send), questo quindi verrà trasferito nello slot di tempo dedicata alla comunicazione

- Ciclo i e $i+3$



- Ciclo $i+1$



- Ciclo $i+2$



Figura 1.6 Comunicazione con nodi "Continuous" e "Multiplexed"

asincrona di uno o più cicli.

Un'opportunità per il managing node di comunicare con i CN della rete è rappresentata dal pacchetto PResMN, questo permette di comunicare in contemporanea con tutti i vari CN della rete, utilizzando una comunicazione di tipo broadcast anziché sequenziale, che risulta essere un metodo indicato per il controllo centralizzato come nel caso dei

robot. Infatti, se volessimo una comunicazione di tipo sequenziale fra i nodi, nella configurazione della rete powerlink alcuni o tutti i CN possono essere impostati come "concatenati", in questo modo una volta che un CN invia un PRes, immediatamente dopo un nodo ad esso concatenato invia i propri pacchetti PRes al MN. In questo, permettiamo al master di gestire un elevato numero di slave in un unico ciclo powerlink sebbene questa metodologia sia consigliabile per un basso quantitativo di dati. Come già detto ogni componente hardware nella rete powerlink ha un proprio numero di nodo, questo può essere impostato sia manualmente sia automaticamente con la funzione DNA (Dynamic Node Allocation). Questa funzionalità molto utile permette una configurazione dei moduli hardware nella rete del bus di campo più rapida ed evita possibili errori nella configurazione della rete. Nell'impostazione del numero di nodo con l'allocazione dinamica (DNA) si settano prima tutti i "switch node" dei moduli hardware a zero e quindi si definisce quella che deve essere la "testa della linea" ovvero il Managing Node, MN. A questo punto il principio di allocazione dei numeri di nodo è il seguente, partendo dal MN, il "master" della comunicazione powerlink, si inviano dei pacchetti nella rete powerlink ai nodi successivi, tali nodi quindi potranno ricevere ed inoltrare il pacchetto al nodo successivo solo se esso stesso ha un numero di nodo assegnato e quindi in caso contrario, verrà assegnato un numero a tale nodo, il che farà attivare

l'hub interno e si potrà quindi continuare a inoltrare nella rete fino a che tutti i nodi avranno un proprio numero.

Ciascuno dei nodi può essere configurato come "Continuo" o "Multiplexed", nella prima configurazione ad ogni ciclo powerlink il nodo è raggiunto da un PReq e risponde con un PRes, mentre nella configurazione "Multiplexed", la comunicazione avviene in modo distribuito in un numero fissato di cicli. Per esempio, se i nodi 1, 2, 3 sono impostati come "Continuo", per consentire la comunicazione ad ogni ciclo, mentre i nodi da 4 a 8 sono invece settati come "Multiplexed". Lo scheduler distribuirà le stazioni powerlink nei vari cicli automaticamente in base al rapporto $(\text{stazione multiplexate}) / (\text{multiplexing prescale})$, quindi si arrotonda il risultato e si assegnano tale numero di stazioni multiplexate per ogni ciclo di rete. Alternativamente, possiamo forzare manualmente la scheduling con l'opzione "Forza il ciclo di rete di stazioni multiplexate" che risulta utile se si vuole garantire dello spazio in termini di tempo all'interno dei cicli powerlink ad una specifica stazione. Nel lavoro in questione ogni stazione della rete è stata impostata come continua e non concatenata, così da garantire la comunicazione ad ogni ciclo della posizione da raggiungere. [3] [4]

1.2 Azionamenti ACOPOS P3

Nella determinazione della qualità, precisione e capacità dinamiche di un processo è di fondamentale importanza la qualità del controllo della movimentazione da parte degli azionamenti elettrici e quindi il tipo di controllo effettuato dai drive e la loro configurazione. Sicuramente come ci si aspetta, il drive avrà un controllo ad anello chiuso, questo può essere facilmente immaginabile se si pensa al fatto, che lo scopo principale dell'azionamento è quello di raggiungere una data posizione il più velocemente e accuratamente possibile. Quindi di fatto la prima cosa che deve valutare il controllore è verso quale posizione si deve muovere, ovvero, quella che di solito si chiama setpoint di posizione, che diventa quindi il riferimento a cui tendere. Una volta noti i setpoint di posizione il controllore al fine di scegliere un'azione di comando ha la necessità di conoscere l'attuale posizione e velocità del

motore che viene quindi fornito tramite feedback dall'encoder che ricordiamo essere un dispositivo che può leggere la posizione e velocità attuale dei motori. Infine, da queste due informazioni e soprattutto dalla differenza dei due ovvero il "lag error" viene definita l'azione di comando, che tenterà a compensare tale differenza.

Il valore dell'azione di controllo nei servi azionamenti ACOPOS è di solito la risultante della somma di due componenti, ovvero, la parte proporzionale al "lag error" secondo un fattore di guadagno "kv" ed una integrale che dipenderà anch'essa da un altro fattore, chiamato in questo contesto "tempo di integrazione" e indicato con la sigla "tn". Infine, queste componenti vengono sommate insieme per diventare l'azione di comando e il risultato finale auspicabile sarà quello di aver compensato l'errore.

Analizzando più nel dettaglio il controllore interno all'azionamento ACOPOS,

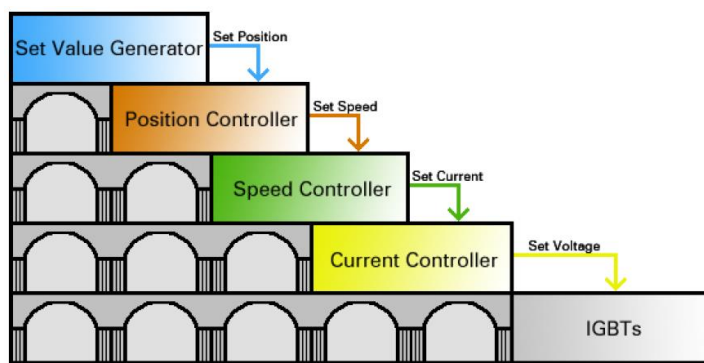


Figura 1.7 Struttura del controllore a cascata

possiamo notare che in realtà risulta un po' più complicato di un semplice controllore PI. Infatti, in realtà, il controllo risultata essere la cascata di controllori diversi, dove l'uscita di un controllore, quindi la

variabile controllata, diventa la variabile di riferimento per il controllore del livello inferiore e così via. Nella cascata di controllori come possiamo vedere da figura 1.7

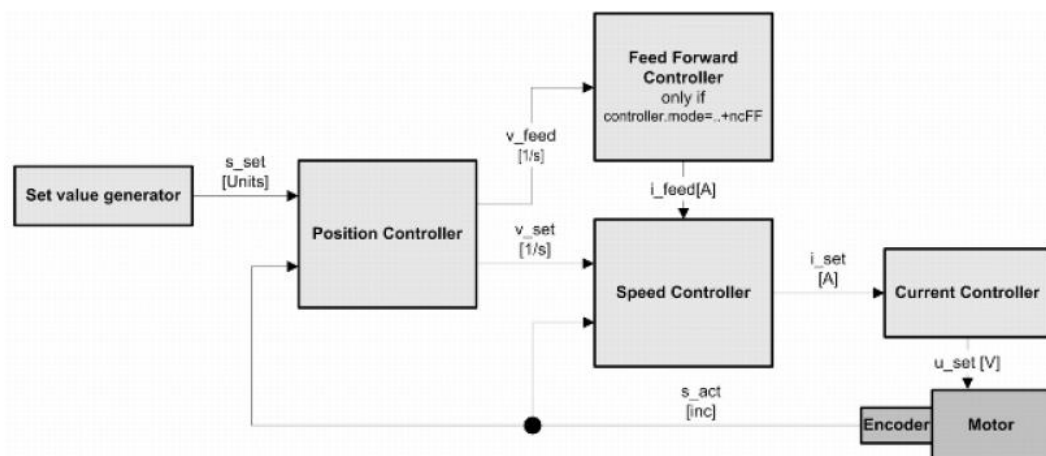


Figura 1.8 Diagramma a blocchi della cascata di controllori

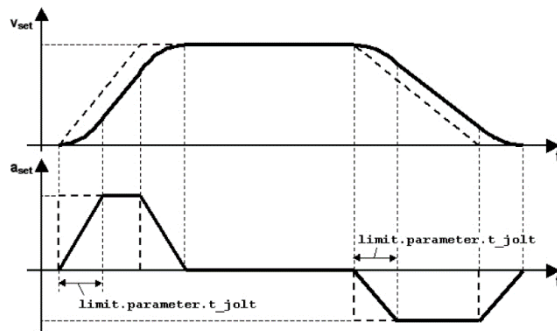


Figura 1.9 Profili di velocità e accelerazioni prima e dopo il limitatore di jerk

movimento, come distanza, velocità massima positiva e negativa e accelerazioni. Elemento caratteristico e presente nel generatore di setpoint è il “limitatore di jerk”, come sappiamo il “jerk” o “jolt” o “strappo” nella nostra lingua, rappresenta la derivazione nel tempo dell’accelerazione. Sostanzialmente quindi, se abbiamo una traiettoria in posizione a cui corrisponde un profilo di velocità trapezoidale, quindi con rampe di primo grado nella velocità durante la fase iniziale e finale della traiettoria, risulterà ovviamente, un profilo delle accelerazioni “a gradini”, questo di conseguenza comporterà grandi valori di coppia necessaria per il motore e inevitabilmente grandi vibrazioni nel sistema meccanico. Lo scopo del limitatore di jerk, quindi, è quello di modificare i gradini in accelerazione per ottenere dei trapezi, in modo tale da avere un minore “strappo”. Il parametro da settare per questa operazione è il tempo desiderato per fare evolvere l’accelerazione dal valore minimo al valore massimo che in Automation Studio è chiamato “t_jolt”, ed è possibile settarlo con un valore che va da 0 a 0.2. Infine, possiamo vedere il risultato di tale operazione in figura 1.9. Proseguendo l’analisi della cascata di controllori, all’uscita dal generatore di setpoint troviamo il controllore di posizione, del quale possiamo avere una rappresentazione più dettagliata in figura 1.10.

Da questa possiamo vedere che il controllore di posizione è stato implementato tramite un controllore PI con anti-windup e feed forward predittivo.

Analizzando questo controllore nel dettaglio vediamo, anzitutto, che la componente proporzionale al “lag error”, ovvero, la differenza fra s_set proveniente dal setpoint generator e s_act proveniente dall’encoder, con il fattore “kv” genera un setpoint di velocità che cambia istantaneamente al cambiare del “lag error”, mentre la componente integrale, come di norma, è utilizzata per compensare disturbi stazionari come ad esempio un carico sospeso e cambia il suo valore di uscita

e 1.8, il tutto inizia tramite il generatore di setpoint. Questo sostanzialmente fornisce ogni $400 \mu s$ la variabile di riferimento al primo dei controllori della cascata, ovvero, il controllore di posizione, al fine di creare un profilo di movimento rispettando quelle che sono i parametri base definiti per il

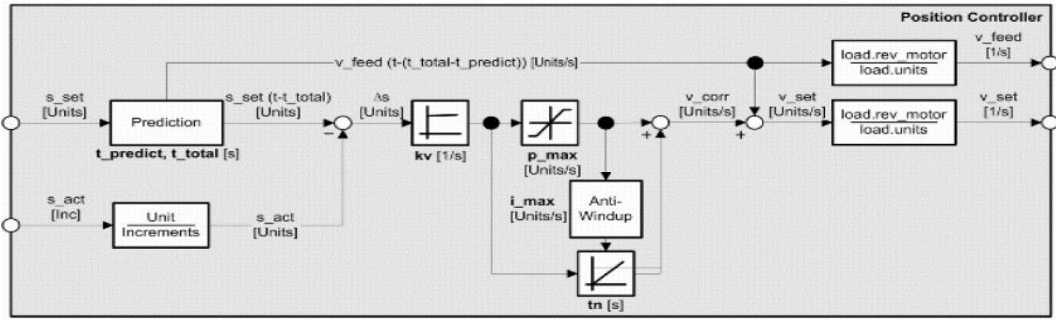


Figura 1.10 Diagramma a blocchi dettagliato del controllore di posizione

gradualmente. In questo schema vediamo che il sistema anti-windup per il controllore PI è effettuato dai due blocchi di saturazione p_max e i_max , questi agiscono, tramite una saturazione del termine in ingresso $k_v \Delta s$ in $-p_{max} \div p_{max}$, generando v_p , mentre il secondo blocco, i_max , limita la componente integrale al valore $(i_{limit} - v_p)$, infine la somma di v_p e v_i ci restituisce il valore di velocità v_corr . Mentre per quanto riguarda la componente di velocità per il feedforward “ v_feed ”, viene generata dalla derivazione nel tempo delle posizioni di setpoint opportunamente ritardate, quindi, questa componente viene sommata alla componente “ v_corr ” che a sua volta deriva dall’output del controllore PI con anti-windup con ingresso il “lag error” Δs , per generare finalmente il setpoint di velocità “ v_set ”. Questo viene prima trasformato dalle unit/s, nelle unità fisica delle rivoluzioni del motore rev/s, quindi questo valore viene usato in seguito nel controllore successivo della cascata come variabile di riferimento.

Proseguendo la cascata dei controllori come possiamo vedere in figura 1.8, il prossimo controllare che incontriamo è il “Feed Forward Controller”, questo nella pratica, semplicemente fornisce dei setpoint di velocità, coppia e corrente utili per migliorare l’inseguimento dei setpoint durante la movimentazione. Dopo questo controllore nella cascata, il prossimo che incontriamo è il controllore della velocità, come gl’altri nella cascata anche questo fornirà un setpoint per il controllore del

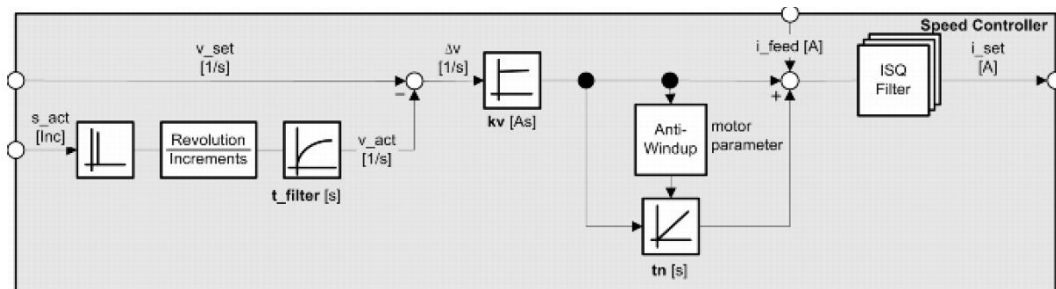


Figura 1.11 Diagramma a blocchi dettagliato del controllore di velocità

livello successivo e tale valore sarà anche questa volta l'output di un controllore PI con anti-windup. Analizzando più nel dettaglio questo controllore, notiamo che, l'uscita del controllore di posizione e quindi l'ingresso a questo controllore è prima interpolato, questo perché, mentre l'uscita del position controller è aggiornata ogni $400 \mu s$, il controllore della velocità calcola un nuovo set point ogni $200 \mu s$, quindi si ha la necessità di interpolare i vari setpoint così da avere il giusto numero di setpoint richiesti. L'altro ingresso di questo controllore invece è la posizione catturata dall'encoder la quale viene prima derivata nel tempo per ottenere la velocità attuale del motore e poi convertita da incr/s a rev/s . Infine, tale velocità prima di essere sottratta alla velocità di setpoint viene filtrata. Il risultato della sottrazione passa poi il controllore PI con anti-windup, infine l'uscita di tale controllore è filtrata utilizzando diversi setpoint di corrente e limitata usando un limitatore di coppia. Lo scopo principale di quest'ultimo passaggio è quello di assicurarsi che dall'azionamento non esca più corrente di quella che può supportare il motore. Il ISQ_FILTER infine è una cascata di filtri con lo scopo di filtrare la corrente di quadratura i_{sq} per evitare problemi come ad esempio interferenze nei segnali oppure per evitare o sopprime frequenze di risonanza, ovvero, il band stop filter, o ancora per filtrare accoppiamenti e disturbi provenienti dai percorsi di comunicazione, errori di quantizzazione nella conversione analogico-digitale e così via. Lo speed filter, invece, posto all'ingresso del controllore di velocità ha un comportamento di tipo passa-basso ed è utilizzato appunto per eliminare i disturbi ad alta frequenza e migliorare la qualità del controllo. Infine, l'ultimo dei filtri presentati è il "Band-stop filter", questo filtro risulta molto utile nell'eliminare la possibilità di una richiesta di corrente che stia vicino alla frequenza di risonanza del sistema, in pratica questo filtro costituito in sostanza da un filtro notch, consente di attenuare un stretto insieme di valori non voluti e quindi permette di aumentare i guadagni possibili nel controllore senza causare instabilità, da tenere in mente che l'utilizzo di questo filtro è consigliato per un sistema che abbia una frequenza di risonanza da 700 a 1500 Hz e oltre a questo esistono altre indicazioni nell'utilizzo di questo filtro, ovvero, questo filtro è indicato solo per sistemi meccanici rigidi quindi senza ingranaggi o cinghie e che abbiano un momento di inerzia sempre costante, da ricordare anche che la frequenza di risonanza di un sistema meccanico può cambiare nel tempo per l'invecchiamento delle componenti e quindi il filtro notch perderà di efficienza.

L'ultimo dei controllori della casca è il controllore della corrente, questo sostanzialmente con una struttura di tipo PI come il controllore della velocità e posizione, controlla i transistor IGBT al fine di generare un segnale di corrente PWM che sarà infine il segnale di comando per il motore. In questo caso la ciclicità del controllore di corrente dipende in modo inversamente proporzionale alla frequenza di commutazione, del quale un tipico esempio può essere 20 kHz, che quindi corrisponde ad un tempo di ciclo per il controllore di $50 \mu s$. Per finire introduciamo il tool interno di Automation Studio che permette un'auto calibrazione dei parametri dei controllori. Sostanzialmente si deve verificare che il drive, i freni meccanici e l'encoder funzionino correttamente e quindi selezionare nelle schedi di "init" degli assi prima la voce ncSpeed per calibrare il controllore di velocità e solo dopo, passare a ncPosition per parametrizzare il controllore della posizione, da tenere presente che in l'ordine della calibrazione svolge un ruolo importante. [5] [6]

1.3 Encoder EnDat 2.2 Heidenhain

Come già anticipato un elemento essenziale nel controllo dei motori da parte degli azionamenti è senz'altro la retroazione, questa nel modo più generale avviene tramite una lettura della posizione del motore da parte dell'encoder, quindi caratteristica essenziale di questo aspetto è senz'altro la velocità di trasferimento e l'affidabilità nella trasmissione, l'interfaccia encoder EnDat 2.2 di Heidenhain ha sicuramente queste caratteristiche. Sostanzialmente, questa interfaccia digitale e bidirezionale è capace di trasmettere valori di posizioni da encoder sia assoluti che incrementali, nonché informazioni memorizzate nell'encoder o di salvare nuove informazioni. Questo tipo di interfaccia utilizzando una comunicazione interamente di tipo seriale e sincrona con il segnale di clock proveniente dalla elettronica della destinazione è capace di sostenere l'intero traffico di dati con solo quattro linee di trasmissione, inoltre, questo tipo di interfaccia si presta bene ad applicazioni di sicurezza e può raggiungere livelli di integrità SIL 3. Infatti, ad esempio, è capace di trasmettere l'informazione della posizione in due canali indipendenti, così come i messaggi di errore ed altri tipi di informazioni riguardanti la sicurezza.

L'interfaccia EnDat 2.2 è compatibile con un gran parte dei tipi di encoder presenti nel mercato, ovvero, quelli lineari o rotativo, assoluti o incrementali, singolo giro o

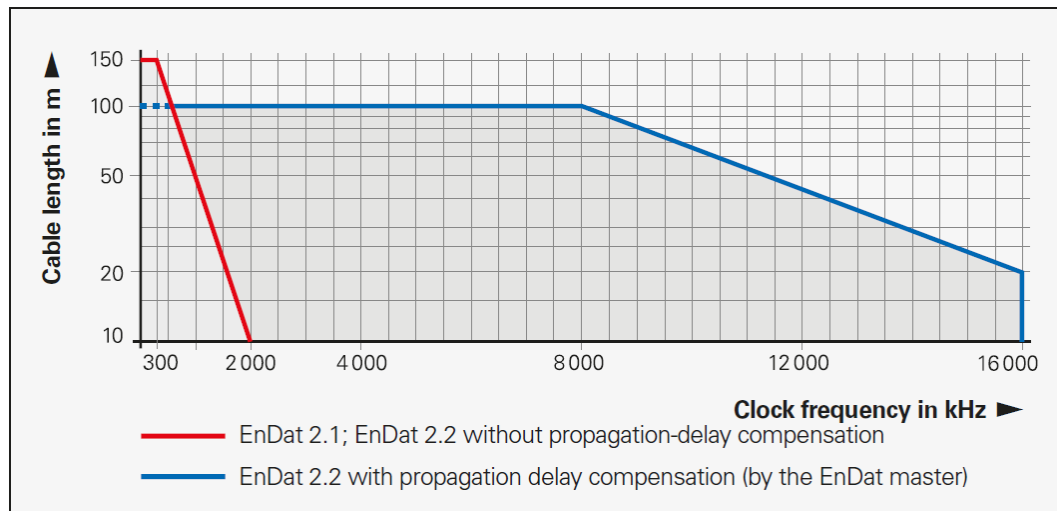


Figura 1.12 Relazione tra frequenza di clock e lunghezza del collegamento con EnDat 2.2

multi giro e così via. Un'altra caratteristica importante per questa interfaccia è la possibilità di trasferire dall'encoder oltre al valore di posizione altri dati addizionali, come ad esempio, timestamp, temperature, valori di sensori addizionali, limiti di posizione e così via.

Particolare attenzione va prestata anche per la frequenza del clock che a differenza degli altri tipi di interfaccia, qui può essere molto più elevata anche per lunghezze sostenute, come si può vedere da figura 1.12. Nel quale bisogna precisare, che la frequenza di clock è influenzata anche dal tipo di segnali usati, infatti per segnali incrementali si arriva ad una frequenza di clock di 2 MHz, mentre viceversa senza

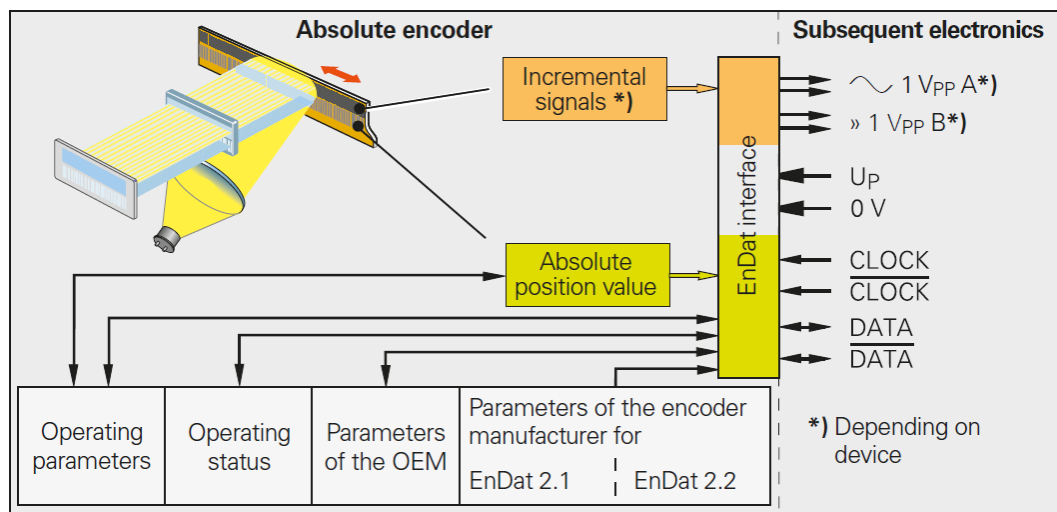


Figura 1.13 Schema di comunicazione EnDat 2.2

questi tipi di segnali fino a 16 MHz. Fra le caratteristiche più importanti abbiamo sicuramente, la possibilità di “appendere” nella comunicazione informazioni aggiuntive al valore di posizione, come parametri o messaggi di errore memorizzati in certe aree dell’encoder, dove comunque si ha un numero limitato di scritture e certe altre zone invece, per sicurezza invece possono essere protette dalla scrittura. In figura 1.13, possiamo vedere una rappresentazione dell’interfaccia EnDat 2.2 in cui possiamo notare i quattro segnali di comunicazione, ovvero, i due segnali di dati e i due segnali di clock, che rispettano lo standard EIA RS-485, che sostanzialmente specifica le caratteristiche elettriche di una connessione seriale, nel quale l’informazione in pratica è contenuta nella differenza dei due segnali trasmessi, che secondo la norma deve essere minimo di 0,2 V e può essere compresa tra +12 V e -7 V. Andremo ora ad analizzare come viene effettivamente propagata l’informazione della posizione, tramite anche l’aiuto di figura 1.14.

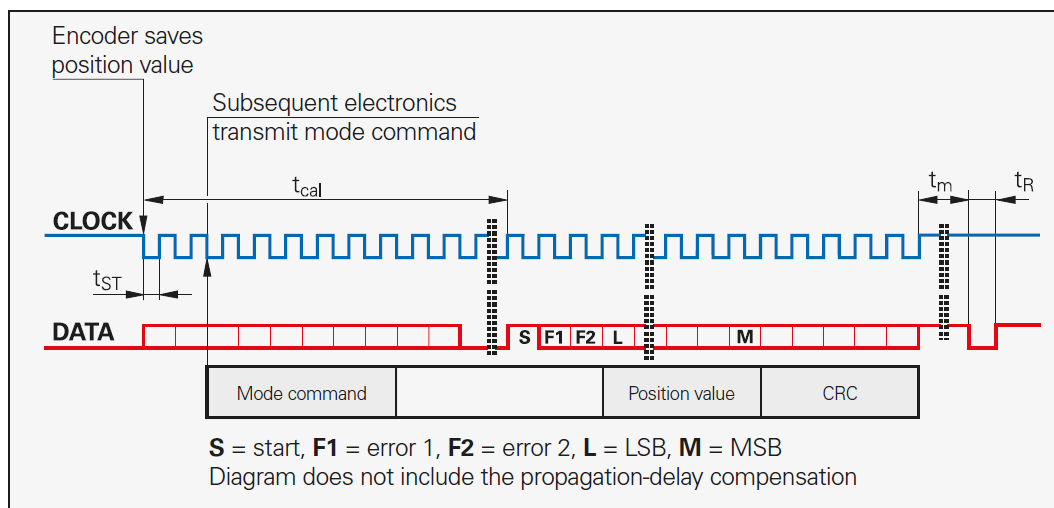


Figura 1.14 Pacchetto con valore di posizione senza dati aggiuntivi

Come si può vedere la trasmissione del pacchetto inizia con il primo fronte di discesa del clock e in quel istante l’encoder legge e memorizza il valore della posizione. Dopo due cicli il sistema a valle della comunicazione invia il segnale di “mode command”, che sostanzialmente sono tre bit inviati in maniera ridondante, invertiti o duplicati, che specificano il contenuto della trasmissione, che può essere, un valore di posizione, un valore di posizione con dati aggiuntivi, parametri da ricevere o da inviare dall’encoder e così via. Quindi, inviati i sei bit per specificare il tipo di messaggio e aspettato un tempo t_{cal} , che varia per valori di posizione o parametri e vale di solito $\leq 5 \mu s$ per un valore di posizione e massimo 12 ms per i

parametri, inizia la trasmissione dei dati dall'encoder all'elettronica di arrivo. I bit successivi a quello di start, indicano l'eventuale presenza di un malfunzionamento dell'encoder che avrebbe potuto causare delle letture errate della posizione. Questi due bit, quindi, generati indipendentemente, segnalano un errore la cui causa è memorizzata e può essere riletta nell'area di memoria del encoder chiamata "operating status" visibile anche in figura 1.12, alcuni degli errori possibili possono essere ad esempio, un malfunzionamento della unità di illuminazione, oppure un'ampiezza del segnale troppo bassa o troppo alta, l'eccessivo consumo di corrente e così via. Dopo questi due bit, può effettivamente iniziare la trasmissione della posizione, nel quale si invia per primo, il bit con minor valore (LSB) e continua per una lunghezza che dipende dal tipo di encoder usato e dalla sua

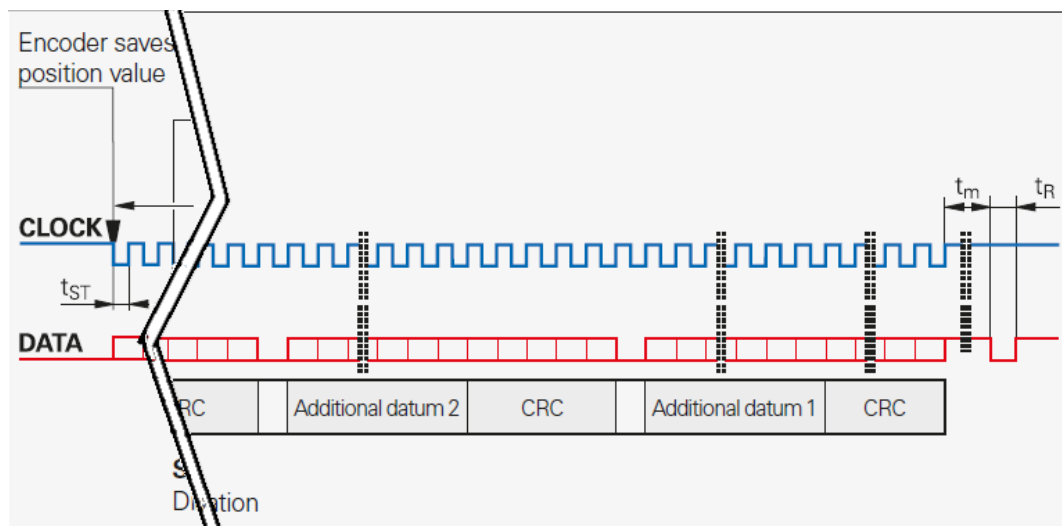


Figura 1.15 Pacchetto con valore di posizione e dati aggiuntivi

risoluzione, tale informazione è memorizzata nei parametri dell'encoder stesso. Una volta inviato anche il bit più significativo il messaggio è completato con il CRC. Inviato anche il CRC nel tipo di messaggio più classico quindi senza dati addizionali, si può terminare la comunicazione e questo viene segnalato dal clock che va alto per un tempo chiamato "recovery time", che per l'EnDat va da 10 a 30 μ s, quindi torna basso per meno di 500 ns e quando riscenderà di nuovo al valore basso inizierà la comunicazione di un altro valore di posizione. Un'alternativa a disposizione per l'EnDat 2.2 è quella di poter appendere al valore di posizione fino a due dati aggiuntivi, questi in pratica vengono segnalati selezionando l'area di memoria dell'encoder da occupare e il tipo di messaggio viene identificato con un numero da uno a sedici. Dopo l'invio del primo dato

addizionale ad ogni valore di posizione viene inviato anche il dato aggiuntivo finché non viene deselezionata questa opzione tramite un opportuno codice MRS.

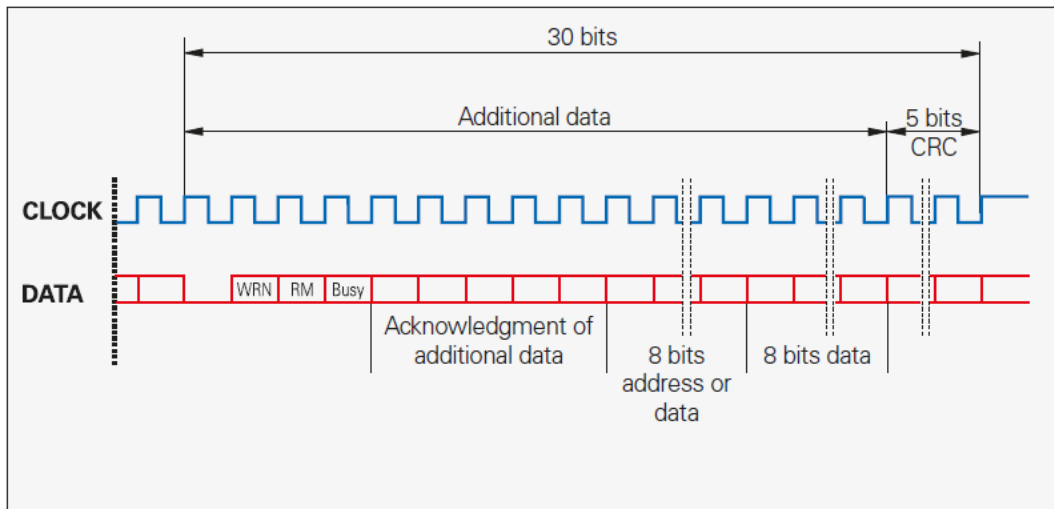


Figura 1.16 Contenuto dei dati aggiuntivi

Nell’analisi di questa interfaccia andremo ora ad analizzare nel dettaglio i possibili contenuti dei dati aggiunti che è possibile inviare con i valori di posizione con l’EnDat 2.2.

Come possiamo vedere la struttura dei dati aggiuntivi che possono essere uno o due, dipendentemente dal tipo di trasmissione selezionata con il codice MRS, è composta da 30 bit il primo dei quali con valore basso, di seguito a questo troviamo un bit denominato WRN che contiene l’informazione sui messaggi di warnings, attenzione, provenienti dall’encoder, tipo il raggiungimento o il superamento di certi limiti, la velocità rotazione o comunque di indicazioni che possono aver prodotto una lettura incorretta della posizione. A seguire abbiamo il bit RM (reference mark), che indica se è stata compiuta o meno l’operazione di referenziazione anche detta di homing che per gli encoder assoluti sarà sempre vero. Oltre a questo abbiamo il bit “Busy” che indica o meno se è stata già richiesta la lettura/scrittura

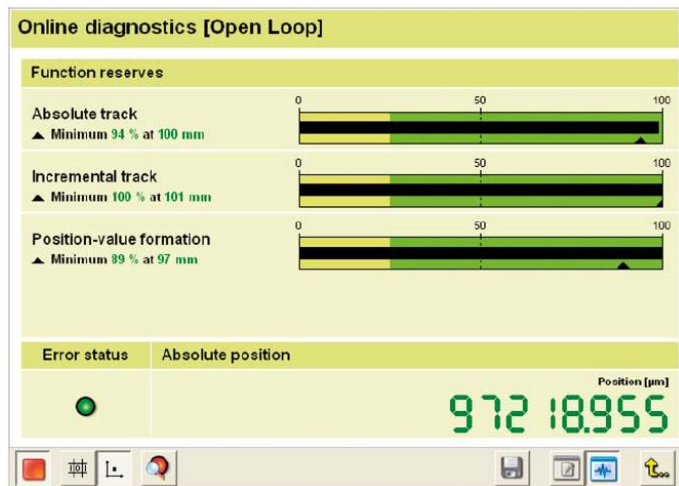


Figura 1.17 Esempio di diagnostica online

di un parametro o se è disponibile (valore basso). A seguire troveremo i cinque bit che identificano il tipo di contenuto e l'area di memoria a cui accedere e infine il dato addizionale che può essere ad esempio, diagnostica, il valore di posizione per la ridondanza, parametri di memoria, il codice MRS, accelerazione, posizioni limite e di "home", timestamp e così via.

Uno dei vantaggi nell'uso di questa interfaccia è senz'altro anche la possibilità di avere una diagnostica online così da poter controllare in maniera immediata l'andamento della macchina, e grazie ai dati memorizzati creare delle statiche sul utilizzo per fornire dei strumenti per i tecnici sul campo e in generale semplificare la risoluzione dei problemi. [7]

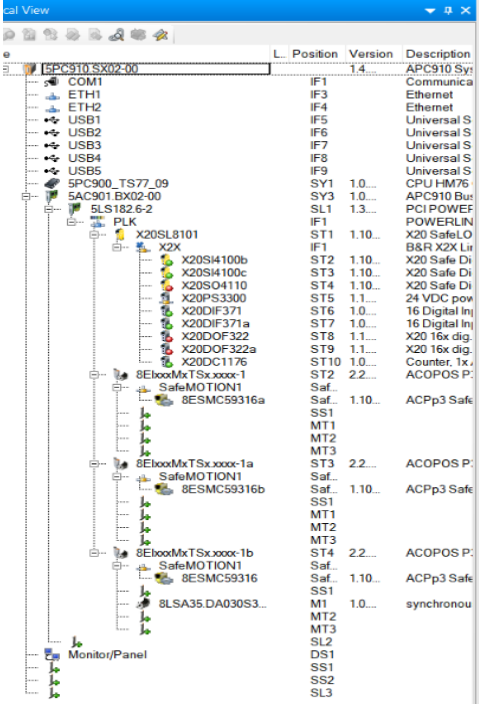
2. CONFIGURAZIONE SOFTWARE

Il software per questo progetto risulta principalmente diviso in tre o quattro blocchi, la parte "hardware" ("Physical View" in Automation Studio) che si occupa della gestione di tutti i vari moduli hardware presenti nella macchina automatica, ovvero in questa schermata si possono impostare i parametri inerenti a ciascuna versione specifica e tipo di hardware. Insieme a questo troviamo la parte "software" del progetto ("Logical View") che contiene i vari programmi scritti in testo strutturato nonché le variabili locali e globali, che rendono effettivamente operativa la macchina/robot. La parte di configurazione ("Configuration View") dove principalmente si configura lo scheduling del PLC assegnando i vari file ST nelle varie cicliche differenti, ognuna con il proprio periodo di ciclo e si possono fissare inoltre anche alcune parti di memoria come variabili permanenti, utili nei casi in cui serve dover recuperare delle informazioni fra una accensione ed un'altra, come nel nostro caso si è avuto bisogno per le cosiddette "endless position" che sono state utilizzate per recuperare la posizione degli encoder fra un'accensione ed un'altra così da poter riprendere il movimento in sicurezza conoscendo già l'attuale posizione del robot e della macchina. Infine, abbiamo la parte di "sicurezza" ("SafeDESIGNER"). In questo contesto si utilizzano solo componenti "Safe", ovvero componenti che soddisfano i requisiti di sicurezza come la ridondanza e così via, per ogni situazione in cui esiste un pericolo e di conseguenza, una possibile fonte di rischio per le persone, l'ambiente e le cose, in questa parte di software quindi utilizzando il linguaggio FBD (Function Block Diagram) o Ladder, nel nostro caso il primo dei due, si programma la gestione dei moduli di sicurezza quindi, implementando funzioni come lo stop di emergenza e il controllo dei limiti dinamici e di posizione.

2.1 Definizione dei moduli hardware nel software

Per l'implementazione dei moduli hardware nel programma si è partiti con la procedura guidata presente nella "Physical view". Definendo quindi l'hardware principale del progetto, ovvero definendo il tipo PC industriale o "system unit" da usare, quindi si passa alla definizione della "CPU board" da utilizzare all'interno

della "system unit" che ne garantirà una certa memoria, grafica e processore, ed infine si seleziona l'interfaccia bus. Quindi terminato il wizard iniziale, sempre nella "Physical view" si inserisce all'interno dell'interfaccia bus della scheda il modulo per l'interfaccia ethernet powerlink nonché contente anche funzioni di hub e ridondanza per la comunicazione, quindi adesso che abbiamo a disposizione una uscita powerlink dalla system unit possiamo collegarla ad altre unità tramite un cavo powerlink, quindi si definisce il collegamento fra la CPU ed un'altra definita CPU di sicurezza in cui è già presente un ingresso PLK (powerlink). Da quest'ultimo modulo, la cpu "safe", si collegano tramite X2X link, i moduli di input e output con specifiche di sicurezza e non, il modulo per la gestione dell'ingresso encoder e il modulo di alimentazione 24 VDC per i moduli I/O e il bus. Quindi infine collegati alla "CPU di sicurezza" si sono implementati i tre "servo azionamenti" ACOPOS P3 collegati con gli altri moduli della rete in una topologia di tipo "daisy chain". Questi azionamenti una volta aggiunti nella lista di hardware fanno partire automaticamente un wizard in cui si possono definire, i tre motori che andranno collegati al drive, anche se nel nostro caso non sono stati selezionati perché tale informazione è automaticamente disponibile ad Automation Studio se si utilizzano encoder di tipo ENDAT2.2 come è stato già anticipato, quindi il wizard continua con la scelta delle librerie per la movimentazione dei tre assi e quindi si passa poi alla creazione di tre strutture



	L.	Position	Version	Description
EPC910_SX02-00			1.4	APC910 Sy
COM1		IF1		Communica
ETH1		IF3		Ethernet
ETH2		IF4		Ethernet
USB1		IF5		Universal S
USB2		IF6		Universal S
USB3		IF7		Universal S
USB4		IF8		Universal S
USB5		IF9		Universal S
5PC900_TS77_09		SY1	1.0...	CPU HM76
5AC901_BX02-00		SY3	1.0...	APC910 Bus
5LS182.6-2		SL1	1.3...	PCI POWER
PLK		IF1		POWERLIN
X20SL8101		ST1	1.10...	X20 SafeLO
X2X		IF1		B&R X2X Lu
X20SH4100b		ST2	1.10...	X20 Safe Di
X20SH4100c		ST3	1.10...	X20 Safe Di
X20SH4110		ST4	1.10...	X20 Safe Di
X20PS3300		ST5	1.1...	24 VDC pow
X20DIF371		ST6	1.0...	16 Digital Inj
X20DIF371a		ST7	1.0...	16 Digital Inj
X20DOF322		ST8	1.1...	X20 16x dig.
X20DOF322a		ST9	1.1...	X20 16x dig.
X20DC1176		ST10	1.0...	Counter 1x
8EhoodMxTSxxxx-1		ST2	2.2...	ACOPOS P:
SafeMOTION1		Saf_		Saf_
8ESMC59316a		Saf_	1.10...	ACPp3 Safe
8EhoodMxTSxxxx-1a		ST3	2.2...	ACOPOS P:
SafeMOTION1		Saf_		Saf_
8ESMC59316b		Saf_	1.10...	ACPp3 Safe
8EhoodMxTSxxxx-1b		ST4	2.2...	ACOPOS P:
SafeMOTION1		Saf_		Saf_
8ESMC59316		Saf_	1.10...	ACPp3 Safe
8LSA35.DA030S3...		SS1	1.0...	synchronou
Monitor/Panel		MT1		
		MT2		
		MT3		
		SL2		
		DS1		
		SS1		
		SS2		
		SL3		

Figura 2.1 Lista hardware

presente un ingresso PLK (powerlink). Da quest'ultimo modulo, la cpu "safe", si collegano tramite X2X link, i moduli di input e output con specifiche di sicurezza e non, il modulo per la gestione dell'ingresso encoder e il modulo di alimentazione 24 VDC per i moduli I/O e il bus. Quindi infine collegati alla "CPU di sicurezza" si sono implementati i tre "servo azionamenti" ACOPOS P3 collegati con gli altri moduli della rete in una topologia di tipo "daisy chain". Questi azionamenti una volta aggiunti nella lista di hardware fanno partire automaticamente un wizard in cui si possono definire, i tre motori che andranno collegati al drive, anche se nel nostro caso non sono stati selezionati perché tale informazione è automaticamente disponibile ad Automation Studio se si utilizzano encoder di tipo ENDAT2.2 come è stato già anticipato, quindi il wizard continua con la scelta delle librerie per la movimentazione dei tre assi e quindi si passa poi alla creazione di tre strutture

rappresentanti i tre assi ed eventualmente altre tre per la creazione dei cosiddetti assi virtuali.

2.2 Configurazione hardware: unità, POWERLINK e vincoli

Una volta terminata la procedura di definizione dell'hardware nel programma. Si definiscono nella tabella "NC Mapping" i nomi degli assi "reali e virtuali" e si creano i collegamenti con la tabella di init "NC Init", i "parametri ACOPOS" e i "dati di addizionali" che includono il tipo di asse (periodico, ciclico o lineare) e la relazione tra "Unità" e le "Unità PLC", stabilita mediante il comando `PLCopen_ModPos = "<period>. <divisor>".` Pertanto, si sono fisati i parametri della "init parameter table". In cui, le principali modifiche includono, le opzioni di input digitale, limiti, controller, homing e parametri di base. Di seguito viene mostrato un esempio della tabella di init del primo e del secondo asse: Quindi

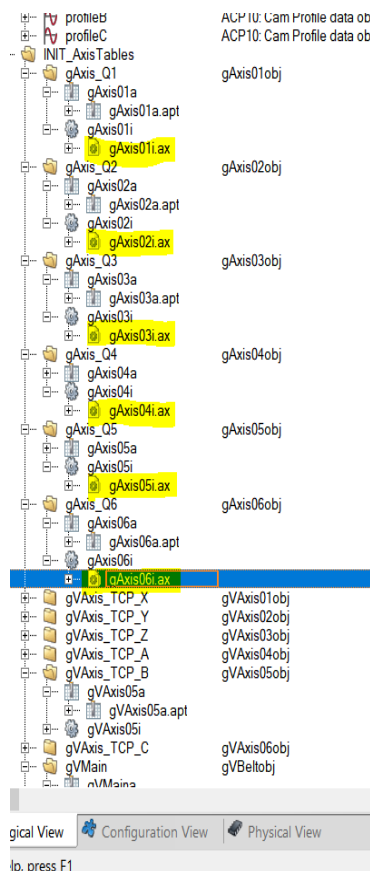


Figura 2.2 Init parameter table

utilizzando insieme il comando (`PLCopen_ModPos = "<period>. <divisor>"`) e i parametri della "tabella di init" si possono definire le "Units" e le "PLCUnits" utilizzati rispettivamente nel drive e nei blocchi funzione PLCopen. Il primo comando definisce il periodo (se esiste) dell'asse e la divisione da eseguire sul numero intero di "Unità" (utilizzato nell'azionamento), per ottenere le "PLCUnits" utilizzati invece nei blocchi funzione PLCopen. Mentre nei parametri della "init table" sotto il gruppo di variabili riguardanti l'interfaccia encoder "encoder_if" vengono definite le "Unità" ("units at the load") e i "rev_motor" (giri motore). Dove $rev_{mot} \cdot U_{mot} = unit\grave{a}$. In pratica, se ad esempio come nel caso del nastro trasportatore (nella tabella init) la variabile "unità" è uguale 200.000, rev_{motor} è uguale a 1 e (in NC Mapping) `PLCopen_ModPos = "1.800.000, 1.000"`, risulta che $T_{mot} = 1.800.000$

(periodo del motore in "PLCunits") con risoluzione 0,001 e 1 giro del motore corrispondono a 200.000 ("Unità") o equivalenti a 200 ("Unità PLC").

Name	ID	Value	Unit	Description
LIM_T1_POS	248	0.58	Nm	CTRL Torque limiter: Maximum acceleration torque in positive direction
LIM_T1_NEG	249	0.58	Nm	CTRL Torque limiter: Maximum acceleration torque in negative direction
LIM_T2_POS	348	0.58	Nm	CTRL Torque limiter: Maximum deceleration torque in positive direction
LIM_T2_NEG	349	0.58	Nm	CTRL Torque limiter: Maximum deceleration torque in negative direction
F_SWITCH	347	10000	cps	Power stage: Switch frequency
BRAKE_MODE	90	0		Motor holding brake: Mode
MOTOR_BRAKE_CU...	42	0	A	Motor holding brake: Rated current
MOTOR_BRAKE_TO...	43	0	Nm	Motor holding brake: Rated torque
MOTOR_BRAKE_ON...	44	0	s	Motor holding brake: Engaging delay
MOTOR_BRAKE_OF...	45	0	s	Motor holding brake: Release delay

Figura 2.3 Parametri degli azionamenti elettrici

Nella "Tabella parametri ACOPOS", si definiscono per ciascun asse i limiti di coppia, la frequenza di commutazione dell'inverter e i parametri per il freno di stazionamento del motore. Mentre nella finestra di configurazione powerlink possiamo modificare alcuni dei parametri della comunicazione powerlink come la durata del periodo ciclico e asincrono del PLK, la dimensione in OCTET del canale dedicato alla comunicazione o ad esempio il tipo di ottimizzazione per la comunicazione del quale si hanno due opzioni, ovvero, velocità effettiva dei dati o

Name	Value	Unit	Description
Operating mode	POWERLINK V2		
MTU size	300		
Baud rate	100 Mbit half duplex		
POWERLINK parameters			
Activate POWERLINK communication	on		
Device name	<InterfaceAddress>		
Host names			
Redundant parameter	Single CPU Project		
Host name	br-automation		
Cycle time	800	µs	
Multiplexing prescale	8		
Mode	managing node		
Advanced			
Broadcast channels			
Channel 1			
Name	ACP10_MC_BROADCAST		
Data type	OCTET[48]		
Direction	Output		
Channel 2			
Name	TODD		
Data type	BOOL		
Direction	Output		
Ethernet parameters			
Activate Ethernet communication	on		
Device name	<InterfaceAddress> ETH		
Redundant parameter	Single CPU Project		
Mode	enter IP address manually		
Host name	br-automation		
IP address			
Subnet Mask	255.0.0.0		
INA parameters			
Activate online communication	off		
Port number	11159		
INA device name	<InterfaceAddress>		
Redundant INA configuration	Single CPU		
INA node number	2		
ANSL parameters			
Activate online communication	off		
I/O-Bus parameters			

Figura 7 Parametri Powerlink

latenza minima. Inoltre possiamo fissare nella configurazione della CPU, la dimensione della memoria dedicata alle variabili permanenti PermanentPVs che in questo progetto è stato fissata a 280 byte, contenente la posizione corrente del motore allo scopo di ripristinare questa posizione dopo ad esempio, un riavvio del sistema o la perdita di alimentazione, tali variabili sono anche conosciute con il nome di "posizione senza fine" (endless position).

Anche nella configurazione della CPU di sicurezza è stato ampliato il numero di canali booleani "BOOL" per la comunicazione tra "SafeLogic" e CPU, poiché

questi canali booleani possono essere mappati con alcune variabili dell'applicazione standard tramite la scheda "NC Mapping" della CPU di sicurezza, così da poter essere utilizzati, con le giuste accortezze, nell'applicazione safe. Inoltre una mappatura simile può essere fatta anche con le variabili in entrata o in uscita dai moduli I/O.

2.3 Variabili software e parametri

Di seguito sono riportate le variabili e i parametri che, insieme ai programmi ST, sono necessari per la corretta esecuzione delle funzioni di sistema. Quindi, nella "Logical View", definiamo una traiettoria per ogni coordinata spaziale X, Y, Z, A, B, C, queste traiettorie vengono estratte da un CAD per il disegno meccanico dopo aver disegnato la traiettoria che l'end-effector dovrà eseguire, per soddisfare i requisiti di produttività e precisione. Nella "Configuration View": all'interno di "Cpu.per" definiamo una variabile per ogni asse di tipo "MC_ENDLESS_POSITION" che dovrebbe recuperare l'ultima posizione per ciascun asse. In "IoMap.iom" invece si creano le connessioni fra variabili di input e output con i moduli dedicati, ovvero si crea la connessione tra la variabile utilizzabile nell'applicazione ST e l'equivalente fisico, che potrebbe essere un pulsante premuto, una lampada accesa o una variabile da inviare o ricevere attraverso i moduli I/O, fino all'applicazione standard o di sicurezza. In "PVMMap.vvm" viene creato un collegamento per nome, una sorta di puntatore, alla struttura che definisce il modello cinematico e dinamico del robot. Inoltre, nella "physical view", si configura lo spazio dedicato alle variabili permanenti indicando tipo, dimensione e posizione, passando alla "CPU di sicurezza" si cambiamo anche qui il "numero di canali BOOL" in entrambe le direzioni da 8 a 32, ovvero quelle variabili booleane che possono essere scambiate fra applicazione standard e di sicurezza tramite la configurazione del file di "I/O Mapping" della cpu "safe" come è possibile vedere anche in immagine. Infine, aggiungiamo nel file "Cpu.sw", tutti i programmi ST, le "camme elettroniche" e le strutture dei parametri che si sono creati e che serviranno a far funzionare il sistema. Questo verrà quindi caricato nella CPU, il quale quindi processerà tali informazioni e in base alla ciclica scelta si avrà

Channel Name	Process Variable	Data Type	Task Class	Inverse	Simulate	Source File
SerialNumber		UDINT				
ModuleID		UDINT				
HardwareVariant		UDINT				
FirmwareVersion		UDINT				
SafeFirmwareVersion		UDINT				
LDID_low		UDINT				
LDID_high		UDINT				
BOOL101	-ResetButton	BOOL	Automatic	<input type="checkbox"/>	<input type="checkbox"/>	(PC900_T577_09)Map.iam
BOOL102		BOOL				
BOOL103		BOOL				
BOOL104		BOOL				
BOOL105		BOOL				
BOOL106		BOOL				
BOOL107		BOOL				
BOOL108		BOOL				
BOOL001	-SafetyBarrierOnError	BOOL	Automatic	<input type="checkbox"/>	<input type="checkbox"/>	(PC900_T577_09)Map.iam
BOOL002	-SafetyBarrierReady	BOOL	Automatic	<input type="checkbox"/>	<input type="checkbox"/>	(PC900_T577_09)Map.iam
BOOL003	-SafetyBarrierOpMode	BOOL	Automatic	<input type="checkbox"/>	<input type="checkbox"/>	(PC900_T577_09)Map.iam
BOOL004		BOOL				
BOOL005		BOOL				
BOOL006		BOOL				
BOOL007		BOOL				
BOOL008		BOOL				
F1S11ModuleOk		BOOL				
F1S11ModuleID		UDINT				
F1S11HardwareVariant		UDINT				
F1S11FirmwareVersion		UDINT				
F1S11StatusInput01		BOOL				
F1S11StatusInput02		BOOL				
ErrALoss		DINT				
ErrAOversize		DINT				
ErrACRCError		DINT				
ErrAOverflow		DINT				
ErrACollision		DINT				
ErrPhyLinkOk		BOOL				
ErrPhyLinkLoss		DINT				
ErrPhyZLinkOk		BOOL				
ErrPhyZLinkLoss		DINT				
Nettime		DINT				

Figura 8 Variabili booleani per la comunicazione tra PLC di sicurezza e non.

lo scheduling richiesto. In questo caso ad esempio fra i vari tempi di ciclo disponibili si è scelto il più rapido che corrisponde ad un periodo di 0,8 ms.

2.4 Programmi ST e architettura del movimento

Per la parte ST del software, si è utilizzato come modello i template B&R: camma, gear e asse singolo, che sono stati quindi adattati e modificati per la nostra applicazione. Entrando nel dettaglio, le principali funzioni aggiuntive non incluse nel modello utilizzato nell'applicazione sono TRF_inverse/TRF_direct, (per la cinematica diretta e inversa). MC_BR_ReadCyclicPosition (per leggere ciclicamente la posizione corrente dei giunti), MC_BR_MoveCyclicPositionExt (per alimentare ciclicamente con i set-point di posizione ogni asse) o MC_BR_CyclicWrite (che allo stesso modo alimenta ciclicamente con set-point di posizione ogni asse ma in più garantisce prestazioni in tempo reale), TRF_init_instance (che conterrà il modello cinematico, che nel caso del robot Comau usando Automation studio e già presente e può essere facilmente replicato) e MC_BR_InitEndlessPosition per ripristinare il valore dell'ultima posizione dei giunti.

Fondamentalmente quindi si sono creati dei "programmi ST" per la gestione di ciascun asse e uno per ogni generatore di traiettoria, chiamato anche camma elettronica in ambiente "Automation studio" e infine un programma principale che gestisce tutte le fasi necessarie di set-up, accensione e movimento. Nel init del main il programma principale, troviamo l'inizializzazione del modello cinematico del

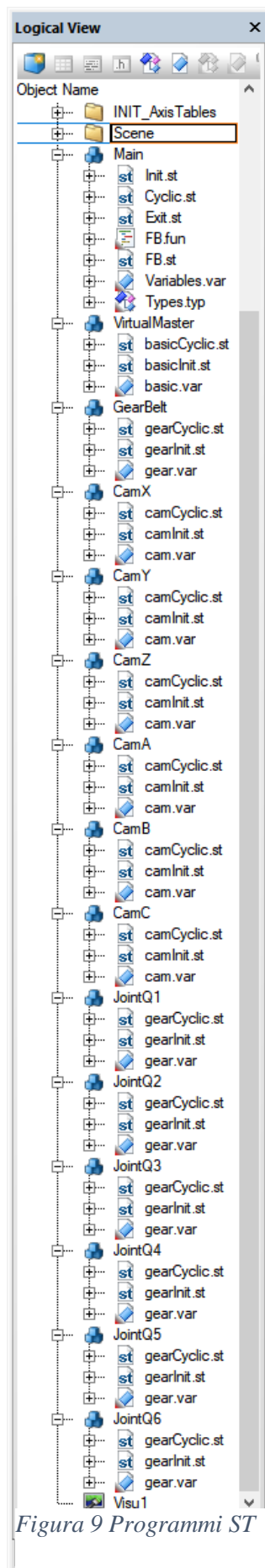


Figura 9 Programmi ST

robot Comau. Questo modello verrà utilizzato nella parte ciclica del programma principale per generare i set point desiderati in termini di posizione dei giunti dai set point di posizione nello spazio, dati dalla traiettoria originaria. L'idea di base dell'algoritmo della parte ciclica del programma consiste sostanzialmente in, per prima cosa, ripristinare l'ultima posizione assunta dal robot con il blocco "endless position" o se questa informazione non è presente per qualche motivo, come la prima accensione o il reset della memoria flash, si fa un movimento verso la posizione "zero" dei motori. Vale a dire, dove la posizione dell'angolo dei giunti è impostata su zero radianti, a questo punto si può eseguire una procedura di homing o referenziazione. Una volta eseguita tale procedura, il robot e il nastro trasportatore raggiungono la posizione di partenza della traiettoria. Questa posizione è ottenuta dal modello cinematico inverso, usando i valori iniziali delle traiettorie disegnate nello spazio per ciascun grado di libertà. Quindi una volta, che abbiamo dato il via all'esecuzione della traiettoria, possiamo prendere i set point di posizione dalle "camme elettroniche", ruotarle e traslarle per mezzo di una matrice di rototraslazione, per riportare il sistema di riferimento sincrono alla base e quindi trasformati nuovamente con la cinematica inversa per ottenere le posizioni dei giunti da scrivere nell'azionamento con funzioni speciali, che garantiscono prestazioni in tempo reale. Questo algoritmo si basa su diversi programmi, ognuno dei quali gestisce una parte del sistema. In particolare, come già detto abbiamo un programma per ciascun asse reale, che gestirà quindi le funzionalità dell'asse e sono quelli chiamati "JointQx" con "x" che va da uno a sei; questi riguardano quindi sia la gestione della macchina a stati di ciascun asse sia la scrittura della

successiva posizione desiderata nell'azionamento, proveniente dal programma principale. Tra i programmi che gestiscono gli assi reali abbiamo anche quello relativo all'asse del nastro trasportatore che a differenza degli altri svolge entrambe le funzioni, ovvero, legge la posizione dalla "camma elettronica" e quindi la scrive nell'azionamento dedicato a tale motore. Oltre a questi, abbiamo i programmi relativi alla gestione degli assi virtuali. Il compito principale di questi programmi è la gestione del legame master-slave degli assi virtuali rappresentanti le coordinate spaziali con l'asse virtuale master che sta a capo di tutta la motivazione, ovvero, il caricamento della "camma elettronica", così come la lettura della successiva posizione desiderata da inviare al programma principale, che si occuperà della trasformazione tramite la cinematica e del successivo trasferimento sotto forma di posizioni giunto agli assi reali. Infine, abbiamo il programma responsabile della gestione dell'asse master virtuale che è l'asse in testa a tutte le pianificazioni del movimento. Questo asse è semplicemente un asse periodico con periodo di 360 unità, che una volta avviato, girando a velocità costante e attraverso le apposite "camme elettroniche" dettano il movimento dell'intero sistema. In comune a tutti questi programmi, c'è la sezione di init di ciascun programma, che essenzialmente è un programma ST eseguito una sola volta e si occupa di definire il master e gli slave nel caso di accoppiamento e impostare i limiti di velocità e accelerazione per ciascun asse, così come la posizione di homing e il ridimensionamento della camma elettronica da eseguire. Inoltre, definiamo alcuni puntatori al controllore dell'asse e alla variabile contenente la posizione di giunto ottenuta dal programma principale, così da facilitare la scrittura della parte ciclica che può sostanzialmente essere ripetuta per ciascun asse andando a modificare solo il file di init. [8] [9] [10] [11] [12]

2.5 PLCopen e lo standard IEC 61131

PLCopen è una associazione internazionale indipendente fondata a seguito della pubblicazione dello standard IEC 61131-3, originariamente IEC 1131-3 nel 1992. L'introduzione di questo standard aveva lo scopo, per quando possibile, di cercare di uniformare i metodi di programmazione per i vari campi dell'ingegneria come motion control, applicazioni di sicurezza e così via, nonché per i vari tipi di

controllori nel mercato, al fine di ridurre i costi relativi alla programmazione, riducendo i tempi necessari allo sviluppo del software e cercando di armonizzare e adattare le varie interfacce per scambio di dati. La norma IEC 61131-3 è la terza parte della norma IEC 61131, ed è il segmento della norma dedicato esplicitamente ai linguaggi di programmazione, il resto delle parti in cui è divisa la norma si dedicherà invece ad altri aspetti della programmazione industriale come l'hardware, la comunicazione, le linee guida per l'applicazione e così via. Lo standard IEC 61131-3 è stato nella storia il primo standard a ricevere un consenso internazionale e deriva dalla modifica di altri standard, pubblicati in precedenza come IEC 848 pubblicato nel 1977, la prima bozza del IEC 61131 pubblicata nel 1979 o il DIN 19239 pubblicato in Germania nel 1983. Come anticipato lo standard, attualmente, è composto da otto sezioni diverse quindi dal 61131-1 fino al 61131-8 che si occupano dei vari aspetti della programmazione industriale e sono stati pubblicati e revisionati in momenti diversi, come ad esempio l'IEC 61131-3 pubblicato nel 1993, nel 2003 e l'ultima revisione nel 2013. Analizzando velocemente le otto parti della norma possiamo notare che questa è composta da, una prima parte dedicata alle informazioni generali quindi caratteristiche funzionali tipiche che distinguono un PLC da altri sistemi come ad esempio l'esecuzione ciclica dei programmi, una seconda parte che definisce le caratteristiche elettriche e meccaniche dei dispositivi nonché i test necessari per la qualificazione, una terza parte dedicata ai linguaggi di programmazione PLC dandone una descrizione ed alcuni esempi d'uso, una quarta parte dedicata alle linee guida per l'utente, per aiutarlo appunto nelle varie fasi del progetto. Una quinta parte dedicata alla comunicazione, ovvero, della comunicazione fra i PLC delle diverse case costruttrici con gli altri dispositivi, per permettere ai PLC di comunicare usando delle reti, in cooperazione con l'ISO 9506. La sesta parte dedicata alla sicurezza nei PLC con lo scopo di unificare e adattare i vari standard per la sicurezza già pubblicati in precedenza come l'IEC 61508 e l'IEC 62061, una settima dedicata ai linguaggi per il controllo Fuzzy, ovvero, un controllo effettuato con componenti analogiche anziché digitali, quindi senza la presenza di variabili logiche tipo zero e uno ma bensì una combinazione di questi, che prendono valori in modo continuo tra zero e uno ecco perché fuzzy, sfocato. Infine un'ultima parte dedicata alle linee guida delle applicazioni per PLC, ovvero, delle istruzioni utili sia all'utente finale che di assistenza in fase di programmazione. [8] [9] [10] [11] [12]

2.5.1 Introduzione allo standard IEC 61131-3

Andiamo ora ad analizzare più nel dettaglio la parte della norma dedicata ai linguaggi di programmazione PLC, ovvero l'IEC 61131-3. Nella fase iniziale di questa norma troviamo una descrizione degli elementi costitutivi dei vari linguaggi di programmazione, il primo dei quali sono i POU. I POU sono le unità software più piccole e indipendenti di un programma, è sono principalmente di tre tipi, dal più semplice il (FUN) function, al più strutturato il (PROG) program, passando per il (FB) function blocks. Oltre a questo, gli elementi fondamentali di un programma per memorizzare e processare le informazioni, sono le variabili. Per le variabili nella norma si specifica, che queste non devono essere memorizzate in zone precise della memoria, come avveniva in passato, ma bensì, si specifica che devono essere di un certo tipo, ovvero, uno dei quei tipi di dato definiti nella norma, ad esempio, intero, booleano, byte, oppure definiti dall'utente stesso, come strutture o arrays, e possono essere definite sia fuori dai POU, sia come parametri d'interfaccia di questi, oppure locali all'interno dei POU. In base a questa distinzione, se tali variabili sono usate all'interno dei POU, devono essere dichiarate contestualmente con la dichiarazione di questi e sono modificabili solo dal POU in questione. Infine tali variabili possono essere classificati come retain o meno se hanno appunto la proprietà di essere ancora disponibili dopo un spegnimento. A seguire della dichiarazione dei POU, in cui si definiscono in pratica le variabili di ingresso e uscita e quelle retain, si passa alle istruzioni che devono essere poi processate dal PLC. Queste istruzioni possono essere sia in forma testuale usando i linguaggi IL e ST, ovvero, instruction list e structured text, che in forma grafica con i linguaggi LD, FBD e SFC, ovvero, ladder, function block diagram e sequential function chart, in quest'ultimo ogni blocco del

diagramma può essere programmato con uno dei linguaggi messi a disposizione nella norma, ovvero, IL, ST, LD e FBD. Lo scopo principale dell'introduzione dei POU è quello di ridurre e unificare i

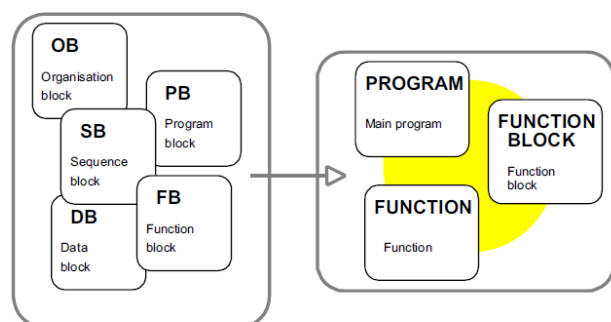


Figura 2.7 Evoluzione dei tipi di blocco prima e dopo la norma IEC 61131-3

vari tipi di blocchi funzione presenti prima della norma.

Procediamo adesso con la descrizione delle principali caratteristiche di questi tre tipi di blocco evidenziandone le maggiori differenze. Iniziando dal più semplice abbiamo il blocco function FUN, che sostanzialmente esegue una funzione a cui quindi ad ogni input restituisce lo stesso output e non ha variabili interne, anche detta senza memoria, come può essere una funzione aritmetica seno, coseno o una logica come AND, OR e così via. Il secondo blocco in ordine di complessità è il function block FB, questo a differenza del FUN possiede variabili interne chiamate stati che possono essere memorizzati e modificati durante le varie chiamate al blocco funzione e di conseguenza l'output di questo blocco dipenderà sia delle variabili interne, gli stati, che dalle variabili esterne, ad esempio possono essere degli algoritmi di controllo come un PI, oppure contatori, timer e così via. Infine, abbiamo il program PROG, nel quale si ha l'obbligo di definire ogni variabile nel programma che ha corrispettivo fisico, che può essere sia un input che un output e l'esecuzione della funzione dipenderà anche in questo caso sia dalle variabili interne che di quelle esterne, ad esempio il programma per un'intera linea. Per quando riguarda la chiamata dei POU's questo avviene tramite il nome, che quindi deve essere unico e conosciuto a tutti. Altro aspetto fondamentale ed esplicitamente indicato nella norma è l'impossibilità di richiamare ricorsivamente una funzione, come può avvenire invece in altri linguaggi di programmazione, ovvero, un blocco non può richiamare sé stesso. [8] [9] [10] [11] [12]

2.5.2 Elementi comuni nei linguaggi dello standard IEC 61131-3

Uno degli scopi dello standard, è quello di poter far cooperare i diversi linguaggi di programmazione. Al fine di ottenere ciò, quindi, si sono definite delle regole in comune per diversi aspetti della programmazione, fra questi abbiamo gli identifiers come indicato nella norma, ovvero, l'identificatore di un programma, una variabile, un blocco funzione e qualsiasi cosa abbia un nome all'interno del programma. Nei quali abbiamo delle regole che riguardano, l'unicità e la composizione del nome, ovvero, l'identificatore non può iniziare con un numero ma solo per lettere o trattino basso, non si possono avere due trattini bassi consecutivi, non si possono avere

spazi, non si può finire con un underscore e nel nome è indifferente l'uso del maiuscolo o minuscolo.

Altro elemento con regole in comune per tutti i linguaggi di programmazione della norma sono le keywords, le parole chiave, sostanzialmente sono delle parole che non possono essere utilizzate come identificatori perché vi è associata una data funzione. Queste parole come gli identificatori non tengono conto del maiuscolo e di solito vengono modificate automaticamente dall'ambiente di sviluppo per apparire maiuscole e non possono contenere spazi. Di queste parole chiave, anche se è presente una lista nella versione del 2013 della norma, comunque l'elenco completo dipende dalle case costruttrici. Nell'elenco delle parole chiave sicuramente saranno presenti l'inizio e la fine di elementi software come FUNCTION, END_FUNCTION, nomi dei tipi elementari, delle funzioni e blocchi funzione standard, FUN, FB, e i nomi delle variabili di ingresso e uscita, la variabile EN, ENO nei linguaggi grafici, gli elementi dei linguaggi ST e SFC e così via.

Proseguendo fra gli elementi comuni dei linguaggi della norma troviamo i literals, i literals in pratica sono i valori assegnati ad una variabile quindi sostanzialmente è la costante che identifica il valore di una variabile e quindi il range e il formato del literal dipenderà dalla natura della variabile. Nella norma si specificano tre tipi di literals, che sono, "numerical literals", "text strings" e infine "time literals". I letterali numerici, si possono distinguere principalmente per la loro base, che si può specificare inserendo prima del valore il numero della base senza il segno e seguito dal carattere cancelletto e dalla precisione e formato, con il quale quindi distinguiamo, i reali e gli interi tramite il punto decimale sul primo dei due tipi e infine, quelli con esponente indicati dalla lettera "E" maiuscola o minuscola, seguita dalla potenza con il segno. I "text strings" o "Character string literals", sono principalmente di due tipi a singolo byte o doppio byte, distinguibili dall'uso del apice singolo o doppio, oppure dal prefisso STRING# o WSTRING#, o ancora tramite il carattere speciale dollaro seguito da due o quattro caratteri esadecimali. In pratica con l'uso del singolo byte si va a restringere ancora l'insieme dei caratteri utilizzabili, lasciandone solo alcuni fra quelli elencati nella norma ISO/IEC 10646-1. Infine, abbiamo i letterali o "literals" del tempo, che si distinguono tra "durata" e "data e tempo del giorno", i primi definiti dal prefisso "t#" o "time#" oppure dalla versione con lettere maiuscole. Questi possono indicare il tempo scrivendo un valore seguito dall'unità che può essere giorni, ore, minuti, secondi o millisecondi,

quindi, d, h, m, s o ms, dove il meno significativo può avere un valore reale e si possono inserire qualsiasi combinazione di unità distanziandole con il trattino basso. Per il tempo e data del giorno invece si può usare uno tra i prefissi di seguito elencati, ovvero, DATE#, D#, TOD#, DT# e altri.

Proseguendo l'analisi degli elementi in comune menzioniamo i commenti, che viene stabilito dalla norma che possono essere posizionati ovunque nel codice si possa inserire uno spazio e devono essere delimitati con la sequenza di caratteri “(” e il corrispettivo in chiusura “)”, oppure, alternativamente per commenti su una sola riga con la sequenza “//”.

Altro elemento definito nella norma in comune a tutti i linguaggi di programmazione è la modalità di indirizzamento, ovvero, nella norma si specifica che ogni variabile usata deve essere dichiarata e contestualmente anche il suo tipo, in caso contrario il compilatore deve segnare un errore semantico nel codice.

Passiamo ora all'analisi dei tipi di dato definiti nella norma, innanzitutto specifichiamo che il tipo di dato è una proprietà della variabile che specifica quali valori può avere quella variabile, come ad esempio, il suo valore di default così come il range dei possibili valori e per finire il numero di bit necessari per la memorizzazione nel PLC. Uno dei vantaggi principali nell'introduzione dei tipi di

No.	Keyword	Data type	N ^a
1	BOOL	Boolean	1 ^h
2	SINT	Short integer	8 ^c
3	INT	Integer	16 ^c
4	DINT	Double integer	32 ^c
5	LINT	Long integer	64 ^c
6	USINT	Unsigned short integer	8 ^d
7	UINT	Unsigned integer	16 ^d
8	UDINT	Unsigned double integer	32 ^d
9	ULINT	Unsigned long integer	64 ^d
10	REAL	Real numbers	32 ^e
11	LREAL	Long reals	64 ^f
12	TIME	Duration	-- ^b
13	DATE	Date (only)	-- ^b
14	TIME_OF_DAY or TOD	Time of day (only)	-- ^b
15	DATE_AND_TIME or DT	Date and time of Day	-- ^b
16	STRING	Variable-length single-byte character string	8 ^{lq}
17	BYTE	Bit string of length 8	8 ^{lq}
18	WORD	Bit string of length 16	16 ^{lq}
19	DWORD	Bit string of length 32	32 ^{lq}
20	LWORD	Bit string of length 64	64 ^{lq}
21	WSTRING	Variable-length double-byte character string	16 ^{lq}

Figura 2.8 Tipi di dato elementari definiti nella norma IEC 61131-3

dato è sicuramente la possibilità di poter controllare in fase di compilazione se il valore della variabile è permesso dal tipo di dato. Nella norma vengono specificati un insieme di tipo di dati definiti elementari e comuni a tutti i

linguaggi e sono quelli riassunti nella figura 2.8. Dove insieme alla parola chiave che identifica il tipo, appare anche indicato, il nome e il numero di bit necessari per la memorizzazione in memoria. Oltre ai tipi elementari il programmatore può creare dei tipi di dato a sua scelta chiamati “derivati”, i quali possono essere usati nella dichiarazione delle variabili come quelli elementari e tale tipi di dato possono essere definiti dalle keywords TYPE e END_TYPE. Fra queste due parole chiave si andranno poi a definire delle variabili di un certo tipo e con certe proprietà, ovvero, ogni variabile può avere una delle proprietà seguenti, cioè, un valore iniziale, range, enumerativo, quindi la variabile può assumere uno dei nomi della lista, array e struttura. Dove per array si intende, un insieme di elementi consecutivi dello stesso tipo, mentre per struttura un insieme di elementi di tipi diversi, e questi due concetti possono essere nidificati a vicenda, creando le varie combinazioni possibili. In particolare se vogliamo creare un tipo derivato con dei limiti nei valori, dobbiamo inserire fra le due parole chiavi, l’identificativo della variabile seguito dai due punti, dal tipo del dato e tra le parentesi tonde il valore iniziale e quello finale distanziato da due punti, stessa cosa per gli enumerativi, ma al posto dei valori avremo dei nomi distanziati dalla virgola, oppure dei nomi associati a dei valori con il simbolo di assegnazione “:=”, e per finire gli array, definibili con la parola chiave “array”, le parentesi quadre l’entry iniziale, due punti e l’entry finale ed eventualmente la virgola per definire array a più dimensioni e infine la struttura, dichiarabile con le due parole chiavi STRUCT ed END_STRUCT, all’interno della definizione del tipo. Un’ulteriore alternativa è quello di creare dei riferimenti a delle variabili con la parola chiave REF_TO, seguita dal nome della variabile a cui vogliamo puntare o ancora si possono creare dei tipi di dato detti “direttamente derivati” che sono tipi che derivano direttamente da uno dei tipi elementari cambiandone ad esempio range o valore iniziale. Infine, abbiamo i cosiddetti “generic data type”, usati ad esempio nell’interfaccia di funzioni standard per indicare una famiglia di tipi elementari permessi in ingresso o uscita da quella funzione. Quindi ad esempio, per una funzione con ingresso un tipo qualsiasi rappresentante un numero avremo come tipo di dato in ingresso permesso ANY_NUM, lo stesso concetto è applicabile poi anche ai tipi di dato direttamente derivati, mentre l’utilizzo di tale tipo in POU definiti dal utente non è permesso dallo standard.

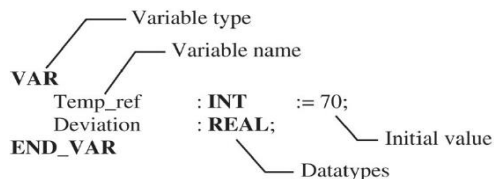


Figura 2.9 Dichiarazione di una variabile

Fra gli elementi comuni ai linguaggi non ci resta che analizzare le variabili e in particolare la dichiarazione all'interno di un programma. Sostanzialmente con la dichiarazione di una variabile si va a

definire un'area di memoria dove si può memorizzare e se necessario modificare un certo valore di un dato tipo, questa dichiarazione viene fatta sempre all'inizio di un POU e va a fissare delle proprietà della variabile dove per primo notiamo certamente il nome, ovvero, l'identificativo che si dà alla variabile a cui segue il tipo ed eventualmente il valore iniziale di tale variabile che va a sostituire il valore iniziale di default. In fase di dichiarazione come già detto si va a definire anche delle proprietà delle variabili, ad esempio, quella di figura 2.9 sarà una variabile di tipo locale rispetto al POU in cui è definito per via delle parole chiave utilizzate nella dichiarazione, un'alternativa sarebbe stata quella di utilizzare in fase di dichiarazione, che avverrà questa volta a livello di configurazione, con le parole chiave VAR_GLOBAL ed END_VAR, in questo modo però ancora non si permette l'accesso delle variabili da qualsiasi POU nel programma, per far ciò nella norma si specifica che le variabili devono essere definite anche all'interno dei POU in cui si vuole utilizzare questa volta con la keyword VAR_EXTERNAL, in modo da evitare al programmatore di utilizzare il nome di una variabile globale anche per una variabile locale al POU. Altre possibilità nella dichiarazione delle variabili può essere quella di definirle delle costanti, tramite la parola chiave CONSTANT accanto a VAR, questo in pratica andrà a fissare nel tempo il valore della variabile. Un'altra parola chiave che si può usare in fase di dichiarazione è RETAIN, questa keyword accanto a VAR, darà alla variabile la proprietà di memorizzare la variabile anche a seguito di un riavvio volontario o meno del sistema. Ancora oltre, abbiamo la possibilità di definire delle variabili di input e output per i POU, così che due o più POU possano scambiarsi delle variabili durante la chiamata ad una funzione, quindi le parole chiave utilizzate in questo caso sono VAR_INPUT, VAR_OUTPUT oppure VAR_IN_OUT. Per finire, esistono anche altre possibilità nella definizione delle proprietà delle variabili, come VAR_TEMP, che in pratica va a cancellare il valore della variabile all'interno del POU ogniqualvolta questo termina l'esecuzione e sarà quindi inizializzata nuovamente ad ogni richiesta del POU, e infine VAR_ACCESS che permetterà l'accesso diretto da parte di un altro

hardware. A queste proprietà si può aggiungere nella definizione delle variabili il

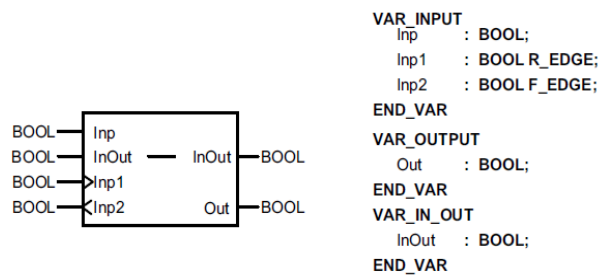


Figura 2.10 Dichiarazione grafica e testuale di variabili secondo la norma IEC 61131-3

qualificatore `READ_ONLY` o `READ_WRITE`, per permettere la sola scrittura o lettura e scrittura della variabile. Altri qualificatori esistenti sono ad esempio `R_EDGE` e `F_EDGE` che applicati ad una variabile booleana ne fa riconoscere il

fronte di salita e discesa, tenendo in considerazione che per la norma quest'ultimi due qualificatore possono essere utilizzati solo per variabili di tipo `VAR_INPUT`. Nella norma si indica anche la versione grafica per la dichiarazione delle variabili di cui mostriamo un esempio in figura 2.10. [8] [9] [10] [11] [12]

2.5.3 Funzioni e blocchi funzione standardizzati

Nella norma IEC 61131-3 oltre alle regole di sintassi appena descritte definisce anche un insieme di funzioni e blocchi funzione, i quali identificatori diventano quindi delle keywords. Queste funzioni, se vengono implementate dal costruttore e quindi offerte nella libreria dell'ambiente di sviluppo, di solito insieme a molte altre funzioni anche non standard, e possiedono lo stesso identificatore della norma, allora devono anche rispettare tutte le regole definite nella norma stessa. Andremo ora a mostrare un insieme di funzioni standardizzate nella norma, queste vengono raggruppate in otto gruppi che sono, le funzioni di conversione, numeriche, aritmetiche, bit-string (bit-shift e bitwise), selezione e comparazione, stringhe di caratteri, funzioni per il tempo e funzioni per tipi enumerativi. Una selezione di queste funzioni è mostrata in figura 2.11. Da questa lista, possiamo subito riconoscere le funzioni più elementari come la funzione di assegnazione indicato con il simbolo `“:=”` in ST, oppure con l'identificatore `MOVE` in LD e FBD, le funzioni logiche fra operatori booleani come `AND`, `OR`, `XOR`, `NOT` e altri ancora. Prima di descrivere alcune delle funzioni standardizzate andremo a descrivere le proprietà che possono avere queste funzioni, in particolare abbiamo la proprietà di `“overloaded”` che va ad indicare che una data funzione accetta in ingresso tipi

generici, quindi non ristretta ad un solo tipo ma può essere applicato anche a tipi diversi, nel quale però sia gli ingressi che l'output devono essere dello stesso tipo.

Nel caso invece si ha una restrizione dei tipi accettati allora si dice che la funzione è “typed” e il nome della funzione deve cambiare, facendo seguire all'identificatore distanziato da un trattino basso il tipo a cui si è ristretto gli ingressi.

Un'altra importante proprietà è l'estensibilità,

Function type	Function name	Comments
Arithmetic	ADD (+), MUL (*), SUB (-), DIV (/), MOD, EXPT (**), MOVE (:=)	MOVE is used for assignment in LD and FBD
Numerical	ABS, SQRT	General
	LN, LOG, EXP	Logarithmic
	SIN, COS, TAN, ASIN, ACOS, ATAN	Trigonometric
Bit-string operations	SHL, SHR, ROR, ROL	Bit shift operations
	AND (&), OR, XOR, NOT	Boolean operations
Selection	SEL, MUX, MAX, MIN, LIMIT	
Comparison	GT (>), GE (≥), EQ (=), LE (<), LT (≤), NE (≠)	
Type conversion	Syntax: Type1_TO_Type2 Examples: INT_TO_REAL, INT_TO_WORD, BOOL_TO_STRING, REAL_TO_INT, STRING_TO_TIME, DT_TO_STRING + many others	Which conversions are supported depends upon implementation
Text-string operations	LEN, LEFT, RIGHT, MID, CONCAT, INSERT, DELETE, REPLACE, FIND	

Figura 2.11 Alcune delle funzioni standardizzate nella norma

questa in pratica indica la possibilità di una data funzione standard (questa proprietà non è applicata a funzioni definite dall'utente) di accettare un numero variabile di ingressi. Sicuramente una delle famiglie di funzioni che più si adatta alla estensibilità anche per proprietà intrinseche delle operazioni, sono senz'altro le funzioni aritmetiche come ADD e MUL, ovvero, addizione, moltiplicazione, oppure le operazioni logiche già citate fra booleani o bit, o ancora le funzioni di selezione come MAX, MIN e MUX, le funzioni di comparazione maggiore, minore e così via, le funzioni fra stringhe tipo FIND, REPLACE, INSERT, CONCAT. Un aspetto da ricordare comunque nell'uso di queste funzioni e soprattutto quelle numeriche e aritmetiche, è sicuramente quello di accertarsi di non avere un overflow a seguito di un'operazione, visto che il tipo di dato in uscita è fisso e dipende tipicamente dai tipi di dato in ingresso. Altro aspetto da considerare nell'uso di queste funzione sono le priorità fra di esse, che vanno nel seguente modo, ovvero, abbiamo come priorità più alta come di solito le operazioni tra parentesi, seguono, le operazioni fatte con le funzioni numeriche quindi SIN, TAN, ABS, dopodiché abbiamo la negazione “-” e NOT, a cui seguono le operazioni aritmetiche con l'usuale priorità quindi prima la moltiplicazione, divisione e modulo quindi l'addizione e la sottrazione, le operazioni di comparazione e per finire AND, XOR e OR. Anche se è da specificare che nella norma comunque non si hanno limitazione nell'uso di parentesi. Completiamo l'analisi delle funzioni standard menzionando

alcune delle funzioni ancora non esplorate come le funzioni di bit-string, ovvero, SHL, SHR, ROL, ROR, che eseguono rispettivamente uno “shift” dei valori nei bit di N posizioni a sinistra per SHL e destra per SHR, mentre ROR e ROL eseguono una funzione simile ma a differenza dei primi due, ciò che viene fuori, a seguito si uno shift, a sinistra dalla stringa di bit viene inserito a destra in maniera circolare e viceversa per ROL. Infine abbiamo le funzioni dedicate alla conversione, che risultano il gruppo più popolato fra le funzioni standard. Queste funzioni risultano particolarmente delicate nell’utilizzo, infatti, si deve prestare particolare attenzione, soprattutto quando si va a fare una conversione fra tipi di dato numerici, perché, ciò può portarci a perdita di informazione o addirittura ad errori, da questa famiglia ne presentiamo due che hanno una funzione simile ma svolta in maniera leggermente diversa e sono REAL_TO_INT e TRUNC_TO_INT, entrambi fanno una conversione tra floating point e intero, con la differenza che il primo fa un arrotondamento all’intero più vicino mentre il secondo farà appunto un troncamento della parte decimale del numero in virgola mobile. Un’alternativa all’uso funzioni standardizzate è la possibilità di definire una nuova funzione, questo può avvenire tramite l’utilizzo delle apposite parole chiavi FUNCTION ed END_FUNCTION, dove accanto alla scritta function si inserirà il nome della funzione, seguito dai due punti e dal tipo di dato in uscita. All’interno del corpo della funzione ci saranno quindi le variabili di input, output o entrambi e infine le istruzioni della funzione. Da ricordarsi che il nome della funzione stessa funge da variabile per il dato di ritorno e le variabili di input possono essere utilizzate invece per la memorizzazione dei cosiddetti argomenti della funzione.

Passiamo adesso all’analisi dei blocchi funzione standardizzati dalla norma in

questione, tali blocchi funzione sono questa volta divisi in cinque categorie che contengono le funzionalità più importanti per la programmazione di un PLC e fra questi abbiamo, gli elementi bistabili, il

Name of std. FB with input parameter names	Names of output parameters	Short description
<i>Bistable elements</i>		
SR (S1, R)	Q1	Set dominant
RS (S, R1)	Q1	Reset dominant
<i>Edge detection</i>		
R_TRIG {>} (CLK)	Q	Rising edge detection
F_TRIG {<} (CLK)	Q	Falling edge detection
<i>Counters</i>		
CTU (CU, R, PV)	Q, CV	Up counter
CTD (CD, LD, PV)	Q, CV	Down counter
CTUD (CU, CD, R, LD, PV)	QU, QD, CV	Up/down counter
<i>Timers</i>		
TP (IN, PT)	Q, ET	Pulse
TON {T---0} (IN, PT)	Q, ET	On-delay
TOF {0---T} (IN, PT)	Q, ET	Off-delay
RTC (EN, PDT)	Q, CDT	Real-time clock
<i>Communication</i>		<i>See IEC 61131-5</i>

Figura 2.12 Blocchi funzione standardizzati

rilevamento dei fronti, contatori, timer e blocchi funzione dedicati alla

comunicazione. Una selezione di questi blocchi funzione è visibile in figura 2.12. Come già detto i function block hanno una memoria interna chiamati stati e pertanto ad ogni esecuzione della stessa funzione molto probabilmente si avrà un risultato diverso, un'altra conseguenza a questo fatto, sarà quindi la necessità di dover dichiarare un'istanza del blocco funzione all'interno del POU in cui viene chiamato tale blocco funzione. Analizziamo per primi i FB della famiglia "edge detection", questi tipi di blocchi funzione risultano molto utili nella programmazione, ad esempio, nel valutare quando una data condizione cambia il suo stato per attivare una qualche altra funzione. Quindi sostanzialmente sono delle funzioni che operano su variabili o espressioni

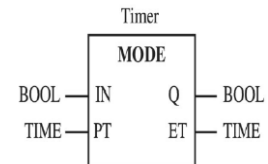


Figura 2.13 Timer standard

booleane e identificano il fronte di salita o discesa di questa variabile al cambio del suo valore, in pratica quindi le funzioni R_TRIG e F_TRIG danno in output una variabile con valore vero quando identificano rispettivamente un fronte di salita o un fronte di discesa nelle variabili di input, tale output quindi rimarrà vero per un tempo di ciclo del programma. La prossima famiglia che andremo ad analizzare è quella degli elementi bistabili anche chiamati flipflops, la cui funzione è quella di memorizzare un dato bit che può essere modificato o meno in base al valore dei due ingressi al function block, infatti nella sua forma grafica questo blocco funzione apparirà come un rettangolo con due ingressi, "S" ed "R", e un'uscita "Q". Sostanzialmente la sua funzione è quella di settare "Q" a 1, se "S" è uguale a 1, oppure, viceversa settare "Q" a 0, se "R" è uguale a 1, o ancora, conservare il valore dell'uscita se entrambi gli ingressi sono uguali 0. Infine, le due versioni specificate nella norma andranno ad evitare ambiguità nei casi in cui gli ingressi sono entrambi alti infatti in questo caso si adotterà la funzione set "S" se il flipflops è di tipo set dominant o viceversa la funzione di reset.

La prossima famiglia di function block che andremo ora ad analizzare è quella composta dai timer, di cui abbiamo una rappresentazione grafica a titolo di esempio in figura 2.13. Ovviamente, l'applicazione più d'interesse con l'uso di un timer è quella di poter cambiare il valore di una variabile dopo che si è aspettato il tempo desiderato. La norma specifica tre tipi di timer, il cui funzionamento può essere compreso anche dalla figura 2.14 e in cui le maggiori differenze si hanno nel valore della variabile di uscita "Q". Infatti, come si vede per il timer abbiamo due ingressi che sono la variabile booleana "IN" che fornisce il segnale di partenza al timer

tramite il suo fronte di salita e la variabile di tipi tempo “PT” che indica il tempo prestabilito da attendere. Mentre in uscita abbiamo la variabile booleana “Q” che indica lo stato del timer dipendentemente dal tipo scelto e infine la variabile di tipo tempo “ET” elapsed time, che indica appunto il tempo trascorso. Ora la differenza fra i vari tipi di timer, ovvero, TON, TOF e TP, sta nel fatto che con TON, la variabile di uscita diventa vera se il valore di ingresso IN resta vero per almeno il tempo PT, mentre nella versione TOF, l’uscita diventa subito vera appena IN è vera e appena diventa falso IN, ET può partire e appena raggiunge il valore di PT fa diventare l’uscita Q falsa, infine, ET si resettera dopo che IN torna vero.

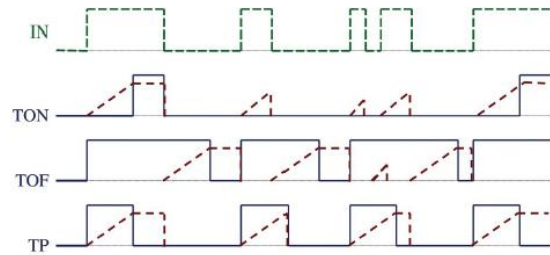


Figura 2.14 Differenze tra i vari tipi di timer in linea tratteggiata la variabile ET

Infine, l’ultimo tipo di timer è il TP e in questo caso abbiamo l’ET che incrementerà il suo valore appena IN è vero e l’uscita diventa vera se ET raggiunge PT, qualsiasi sia il valore di IN a differenza della versione TON.

Infine, l’ultima delle famiglie di function block che andremo ad analizzare è quella dei contatori, nel quale sono presenti tre versioni, ma di fatto uno dei tre racchiude le funzionalità degli altri due, in pratica quindi, abbiamo il contatore incrementale, quello decrementale e il terzo capace di entrambe le possibilità. Nel contatore detto “up/down counter” CTUD, abbiamo cinque ingressi, quattro dei quali booleani, identificati con le sigle CU, CD, R, LD, PV, che stanno rispettivamente per count up, count down, reset, load e preset value e tre uscite QU, QD, booleane e CV intera che stanno per, l’uscita per il counter-up, l’uscita per il counter-down e il valore corrente.

Passiamo ora alla definizione dei function block definiti dall’utente, questi come gli altri POU si dichiarano con l’uso delle parole chiave FUNCTION_BLOCK ed END_FUNCTION_BLOCK, all’ interno del quale si definisco le variabile di input, e output e infine il codice della funzione. Come i blocchi funzione anche i program PROG, possono essere definiti con le appropriate parole chiave e come già detto il programma è un’unità di più alto livello rispetto al resto dei POU presentati e quindi di conseguenza può contenere istanze sia delle funzioni che dei blocchi funzione o perfino chiamare altri programmi.

2.5.4 I linguaggi della norma IEC 61131-3

Andremo ora ad analizzare i linguaggi definiti dalla norma, che sono stati poi anche quelli usati in questo progetto per la programmazione della macchina automatica. Fra questi distinguiamo quelli in forma grafica come il function block diagram, FBD e il ladder, LD, con quelli in forma testuale come l'instruction list, IL e lo structured text, ST. Fra questi iniziamo la nostra analisi dal linguaggio ST, testo strutturato in quanto risulta quello più utilizzato in questo progetto e soprattutto nell'applicazione standard del controllo.

Il testo strutturato è uno dei linguaggi testuali della norma e senz'altro il più utilizzato nella pratica, infatti molti dei blocchi funzione forniti dalle case costruttrici sono scritte in ST. Il testo strutturato è un linguaggio di alto livello con una struttura molto simile a C, tale linguaggio come già detto può essere impiegato sia per la definizione di funzioni o blocchi funzioni, quindi in cooperazione con altri di linguaggi di programmazione, oppure, può essere usato per la definizione dell'intero programma con i propri pro e contro. Sicuramente i vantaggi del testo strutturato sono la compattezza rispetto agli altri linguaggi e la facilità nei calcoli aritmetici e numerici, la gestione delle strutture e la facilità nel uso di istruzioni di condizione e la gestione di loops. Per questi aspetti infatti, il testo strutturato possiede dei strumenti che facilitano tali operazione, come infatti sono presenti molti simboli di operazione che vanno a sostituire nell'utilizzo l'identificatore così da semplificare la comprensione del codice. Per le istruzioni condizionali e per i loops in ST abbiamo molte istruzioni che ne facilitano l'uso rispetto ad altri linguaggi della norma come ad esempio quelli grafici. Altro elemento distintivo con gli altri linguaggi è la modalità di assegnazione che come già detto avviene con il segno di uguaglianza preceduto dai due punti e questo tipo di assegnazione può essere estesa a più variabili. Per quando riguarda l'utilizzo e quindi la chiamata ai function block, come sappiamo questi devono essere prima dichiarati e contestualmente si dichiarano automaticamente anche le variabili di input e output del FB se presenti. Infatti, a seguito della chiamata di un blocco funzione questo non restituisce un valore come spesso avviene in C, ma bensì gli eventuali valori di uscita vanno a memorizzarsi nelle variabili di output, presenti nel argomento del blocco funzione. Successivamente tale variabile di uscita può essere recuperata con una sintassi che prevede, il nome dell'istanza del blocco funzione seguito da un

punto e di seguito il nome dell'argomento posto in corrispondenza della variabile di uscita. Oppure, alternativamente il valore della variabile di uscita può essere memorizzato in un'altra variabile, tramite un'istruzione scritta direttamente in fase di chiamata della funzione. In pratica in corrispondenza della variabile di uscita avremo l'argomento della funzione seguito dal simbolo " => " e infine la variabile di destinazione del valore. Per quanto riguarda gli aspetti di debolezza di questo linguaggio possiamo menzionare l'assenza dei blocchi per il rilevamento dei fronti e gli elementi bistabili, che sono comunque ben sostituiti dalle istruzioni condizionali come IF e CASE, grazie al quale si può molto semplicemente rimediare a queste mancanze. L'istruzione "IF" come nel linguaggio C può essere nidificato con l'uso della parola chiave ELSIF e può del tutto sostituire un flip-flop arrangiando nel modo corretto le condizioni e le istruzioni da eseguire. L'istruzione CASE invece è molto utile nei casi in cui si hanno molte comparazioni a cui corrispondono istruzioni diverse e anche in questo caso è possibile l'uso dell'ELSE. Un'importante funzionalità che ci permette l'uso dell'istruzione CASE, che è stata ampiamente usata nel progetto della macchina automatica oggetto di questo elaborato, è la possibilità di creare molto facilmente delle macchine a stati per il controllo del intero sistema. Infatti ad ogni opzione del CASE può corrispondere uno stato e le transizioni possono essere fatte semplicemente modificando il valore della variabile che seleziona l'opzione da eseguire. Altro elemento molto utile nel controllo di una macchina, sono le istruzioni per creare dei loop nel codice. In particolare nel testo strutturato abbiamo tre possibilità per creare dei loop e sono tramite le istruzioni, while, repeat e for. Con l'istruzione while, in particolare con l'uso delle parole chiave WHILE...DO...END_WHILE, si costruisce un loop dove ad ogni iterazione viene prima controllata la condizione o espressione booleana contenuta fra WHILE e DO e successivamente viene eseguito il codice fra DO ed END_WHILE. Tale operazione infine verrà ripetuta finché la condizione non assuma un valore logico vero, senza conoscere quindi a priori il numero di iterazioni che saranno necessarie per soddisfare la condizione, con lo svantaggio di avere una ripetizione infinita se non venisse modificata la condizione. In maniera simile si comporta anche la funzione repeat, che tramite le parole chiavi REPEAT...UNTIL...END_REPEAT, permette la creazione di un loop, dove il numero di iterazione non è direttamente indicato ma dipende dalla condizione posta fra le parole chiave UNTIL ed END_REPEAT. Quindi sostanzialmente a differenza

del loop while, qui la condizione viene valutata dopo aver eseguito le istruzioni del loop e quindi si assicura che viene eseguito almeno una volta il corpo del codice. L'ultima possibilità per creare dei loop è l'istruzione for, questa tramite le keywords FOR...TO...BY...DO...END_FOR, crea un loop in cui il numero di iterazioni questa volta è fissato.

Nel testo strutturato infine, abbiamo una istruzione che può essere utilizzata all'interno dei loop così da interrompere l'esecuzione di tale loop. Tale funzionalità si può avere tramite le parole chiave IF...THEN EXIT END_IF, in questo modo soddisfatta la condizione fra IF e THEN, l'esecuzione prosegue dall'istruzione successiva alla parola chiave che identifica la fine del loop all'interno del quale abbiamo l'istruzione di EXIT. Quindi nel caso di loop nidificati bisogna prevedere la presenza di istruzioni di exit ad ogni loop.

Passiamo ora all'analisi del secondo linguaggio di programmazione utilizzato nel progetto, ovvero parliamo del linguaggio FBD, function block diagram, utilizzato per l'applicazione di sicurezza del progetto, sviluppato tramite SafeDESIGNER. L'FBD è uno dei linguaggi grafici della norma IEC 61131-3, il quale utilizza simboli grafici anche concordi con un'altra norma la IEC 60617-12. L'FBD sostanzialmente si basa sulla connessione grafica di variabili, funzioni e blocchi funzione, con eventuali operatori logici fra l'una e l'altra connessione, questo linguaggio a differenza del LD non fa uso dei cosiddetti "binari", ovvero, di quelle linee parallele verticali a cui è associato un valore sempre vero a quella di sinistra e un valore neutro a quella di destra, ma qui la divisione del codice avviene tramite reti e sotto-reti e l'attivazione di una funzione avviene tramite l'ingresso EN sempre presente in tutti i blocchi di questo linguaggio e analogamente ci sarà una variabile di uscita ENO che avrà il valore logico vero appena l'uscita è pronta. Per quanto riguarda l'ordine di esecuzione di un programma FBD, questo va da destra verso sinistra e dall'alto verso il basso e la creazione di loop, quando è permesso dalla casa costruttrice, può avvenire sia esplicitamente tramite l'uso di connessione o tramite variabili, un'alternativa per modificare l'ordine di esecuzione di un programma è quello di usare la funzione di jump e le label, che permettono appunto la ripresa dell'esecuzione di un programma da un punto prestabilito.

Passiamo ora all'analisi dell'altro linguaggio di programmazione grafico stabilito nella norma ovvero il ladder LD, per poi andare a confrontare le varie differenze e similitudini con l'FBD. Nella descrizione del ladder, c'è sicuramente da ricordare,

che anche se, ad esempio in questo progetto non è stato usato, comunque il ladder vanta un grande impiego nella pratica della programmazione PLC ed è sicuramente da imputare alla facilità e comprensibilità nell'uso e soprattutto per essere stato di fatto il primo linguaggio in assoluto per la programmazione dei PLC. Uno degli elementi costitutivi di un programma ladder sono sicuramente le due linee verticali che racchiudono poi tutto il programma, queste due linee hanno un valore positivo quella di sinistra e un valore indefinito quella di destra, che dovrebbero rappresentare, in accordo anche con le analogie fatte ai tempi con i quadri elettrici, una l'alimentazione, la linea di sinistra, e l'altra la massa, la linea di destra, la cui presenza dipende esclusivamente dalla casa fornitrice dell'ambiente di sviluppo. Atra nomenclatura del linguaggio LD, sono i "Rung" che altro non sono che le varie righe di "codice" del programma, quello che poi in FBD chiamiamo reti e sotto-reti. Nella rappresentazione grafica del programma il ladder fa uso di alcuni simboli soprattutto per la gestione delle variabili booleane e fra questi troviamo i "contatti" che possono essere visti come variabili di ingresso booleane, ad esempio associate a condizioni provenienti da sensori o simili e le "bobine" che possono essere viste come la variabile di uscita di un'espressione booleana. Altri possibili simboli del linguaggio ladder sono ad esempio, i rilevatori di fronti che hanno lo stesso simbolo dei contatti ma con in mezzo una P o una N, rispettivamente per il fronte di salita o discesa, oppure potremmo avere, una bobina con in mezzo le lettere S e R che fungono da elemento bistabile. Oppure ancora, altro elemento caratteristico del linguaggio ladder è la funzionalità di "saltare" da un Rung (gradino della scala) ad un altro tramite l'opportuno elemento di azione che può essere condizionato o non. Questa funzionalità in pratica, tramite delle label che possono definire se presente il nome di ciascun rung del programma e può essere utilizzato come l'identificativo per saltare a quel specifico rung indicato dalla label, permette di saltare in un punto qualsiasi del programma. Infine, abbiamo la istruzione di RETURN che fa ritornare l'esecuzione al POU di partenza, ovvero se stiamo eseguendo il codice di un FB, chiamato da ad esempio da un altro FB, con l'istruzione RETURN possiamo ritornare al POU di partenza. Un elemento distintivo rispetto all'FBD ad esempio è la gestione delle variabili logiche, infatti ciò che viene fatto in FBD tramite funzioni o blocchi funzione come ad esempio le operazioni logiche, AND, OR e così via, in ladder ciò può essere gestito tramite le combinazioni dell'uso di contatti e linee di connessione orizzontali e verticali. Per quanto riguarda l'ordine di esecuzione di un

programma ladder, questo avviene, gradino per gradino dall'alto verso il basso e da sinistra verso destra esattamente come avviene anche per il function block diagram, quindi sostanzialmente il PLC valuterà i valori dei contatti, quindi i valori degli input/output associati o sensori e quant'altro e andrà di conseguenza a modificare il valore della bobina corrispondente. Un' aspetto importante da tenere in considerazione nella programmazione ladder, che può diversamente causare errori è l'impossibilità di usare la stessa bobina in gradini diversi della scala. Ovvero, una stessa variabile di uscita non può essere usata su due righe diverse del programma, altrimenti, molto probabilmente si avranno degli errori di conflitto.

Analizziamo ora l'ultimo dei linguaggi di programmazione testuali della norma rimasto, ovvero, l'instruction list, IL. Questo linguaggio risulta più di basso livello rispetto agli altri e infatti è un linguaggio di programmazione simile all'assembler, dove ad ogni riga corrisponde un comando eseguibile per il PLC. La struttura di base di un comando IL è composto da, una label seguita dai due punti, l'operatore o l'identificativo della funzione e la lista di operandi separati da almeno uno spazio, che saranno poi, le variabili di input della funzione. Anche in questo linguaggio la label servirà come identificatore per "saltare" da un'istruzione all'altra, mentre in contrasto con gli altri linguaggi di programmazione nell'IL non è permesso l'uso del punto e virgola. Similmente agli altri linguaggi assembler, anche IL fa uso di un elemento accumulatore, che però a differenza degli altri in questo non è composto da un hardware vero, ma piuttosto si tratta di un accumulatore virtuale, chiamato "Current result" CR e in pratica adattandosi all'ultimo tipo di dato gestito dalle istruzioni, memorizza il risultato delle operazioni. Da tenere presente che comunque una volta caricato un tipo di dato il successivo deve essere dello stesso tipo, almeno che l'istruzione corrente non è tale da modificare il tipo del CR e di solito queste funzioni sono quelle di load LD e generate GT, mentre store ST e jump JMP invece lo lasciano invariato, mentre l'istruzione call, chiamata al blocco funzione lascia il tipo del CR indefinito, ovvero dopo una chiamata con l'istruzione CAL possiamo fare solo un JMP, un LD , un CAL o un return RET, che sono le operazioni che non hanno bisogno di un CR con un tipo specifico. Oltre a queste possibilità anche se la norma non lo specifica direttamente, esistono dei modificatori per le operazioni già viste. In particolare abbiamo il modificatore di negazione N, che posto subito dopo l'identificatore della funzione ne nega il risultato, il modificatore per la nidificazione delle operazioni, le parentesi, che poste

subito dopo un identificatore, memorizza il valore e il tipo del CR e ne crea un nuovo per le operazioni interne alle parentesi, e una volta effettuate le operazioni interne e chiuse le parentesi il sistema recupera il valore e il tipo del CR e lo combina con quello creato. Il terzo e ultimo tipo di modificatore è quello “condizionale”, C, questo posto di fronte alle operazioni tipo JMP, ST, RET, CALC, eseguono l’operazione solo se il valore del CR è vero al momento dell’esecuzione di quella operazione. In IL esistono quindi due modi per chiamare una funzione e caricare i parametri, uno che fa uso delle parentesi e del simbolo di assegnazione (:=) e l’altro semplicemente scrivendo accanto al nome della funzione i parametri distanziati con la virgola considerando anche come parametro il valore del CR. Analogamente esistono tre modi per chiamare un blocco funzione, usando una chiamata con i parametri tra parentesi, inclusi input e output, un’alternativa è quella di caricare e salvare i parametri prima della chiamata e infine in maniera implicita, usando l’operatore come un parametro di input.

Andiamo ora ad analizzare un metodo grafico e strutturato per la scrittura di programmi, e in particolare un metodo adatto ai programmi perlopiù sequenziali ma anche paralleli e permette la semplificazione di programmi complessi in sottoprogrammi più piccoli facilmente gestibili, evidenziando nel programma le sequenze, gli stati e le condizioni per il passaggio da uno stato ad un altro. Questo metodo chiamato oggi SFC (Sequential Function Chart) deriva da uno standard francese il Grafset, che deriva a sua volta da un modello matematico chiamato Reti di Petri del 1962, poi internazionalmente standardizzato con la norma IEC 848 nel 1988 e infine introdotto con la norma in questione. Questo metodo in pratica usando uno dei quattro o anche tutti e quattro i linguaggi di programmazione della norma divide il compito del controllo in sotto parti. Gli elementi fondamentali di questo metodo sono gli step, le transizioni e le azioni, in cui gli step rappresentano in qualche modo lo stato in cui si trova la macchina, le transizioni indicano invece le condizioni da soddisfare affinché si passa da uno stato ad un altro e le azioni invece indicano le varie istruzioni che devono essere eseguite all’interno di uno stato o fase in cui si trova la macchina. Sostanzialmente, quindi in un programma SFC abbiamo degli step intervallati da transizioni, che graficamente appaiono come dei rettangoli intervallati da linee orizzontali che tagliano la linea che collega i rettangoli. Dove, i rettangoli rappresentano gli step del programma, ovvero, racchiudono una serie di azioni che il sistema dovrà eseguire in quel determinato stato mentre le linee

orizzontali rappresentano le transizioni tra uno stato e l'altro e di solito sono associate a variabili interne o ad esempio a valori di sensori o simili. In pratica quindi l'esecuzione di una rete SFC, inizia da un step speciale chiamato step di inizializzazione, che è il punto del programma in cui inizia l'esecuzione una volta che il PLC è andato nello stato di run, quindi si eseguono le istruzioni contenute all'interno dello step anche chiamate azioni e appena la transizione si completa ovvero la condizione a cui è associata diventa vera, lo step a monte della transizione si inattiva e si attiva quello a valle e quindi l'esecuzione passa al blocco successivo alla transizione. Oltre a questo tipo di esecuzione nelle reti SFC si possono avere altre alternative per scegliere il giusto flusso di operazione da fare, per questo si sono definite delle sequenze alternative. In queste sequenze riconosciamo, la sequenza singola, ovvero quella appena descritta, la sequenza divergente, ovvero, una ramificazione della singola sequenza dove ad ogni ramo di questa sequenza c'è un elemento di transizione e i diversi tipi di sequenze divergenti differiscono per l'ordine in cui vengono verificate le condizioni delle transizioni, ovvero, queste vengono verificate da sinistra verso destra, oppure, con una priorità scelta dall'utente. Alla sequenza divergente segue quella convergente, che serve in pratica a racchiudere tutti i vari flussi di esecuzione prima diramati e sostanzialmente si ha come nel caso della divergenza una condizione per ogni ramo prima del punto di convergenza e quando una di queste è verificata dopo che lo step a monte è stato eseguito il flusso dell'esecuzione può continuare nella sequenza singola. Analogamente abbiamo per l'esecuzione parallela la divergenza e la convergenza, con la differenza che in questo caso la condizione è posta prima della ramificazione nel caso della divergenza e dopo il punto di unione nella convergenza. Infine, nella lista di sequenze, abbiamo il cosiddetto ramo all'indietro di solito posto a valle di una divergenza, utile per la creazione di loop e per ultimo il "salta sequenza" che in pratica è un ramo vuoto posto fra una divergenza e una convergenza e serve appunto per evitare l'esecuzione di uno step. Un' altro elemento caratteristico delle reti SFC, è la dichiarazione implicita di due variabili per ogni step nella rete, accessibile facendo seguire al nome dello step un punto e la lettera X e T contenente l'informazione rispettivamente dello stato dello step e il tempo trascorso nell'esecuzione. Per finire passiamo all'analisi delle azioni e dei cosiddetti blocchi azione che possono essere presenti o meno ad ogni step della rete SFC, sostanzialmente questi blocchi sono formati da tre campi in cui distinguiamo, il tipo

dell'azione anche detto qualificatore, il secondo campo contenete l'istruzione o azione, che contiene o il nome della variabile modificata se questa è l'unica istruzione del blocco o il nome di un programma contenete varie istruzioni e infine il terzo campo contiene se necessario l'identificatore della variabile che è di solito il nome della variabile il cui valore viene modificato alla fine dell'esecuzione delle istruzioni. [8] [9] [10] [11] [12]

3. PIANIFICAZIONE DEL MOTO

Per pianificare il movimento dell'intero sistema composto dal robot e dal nastro trasportatore, si è pensato ad un'architettura con un asse master virtuale periodico, con un periodo di 360 unità collegato con altri sette assi slave. Tra cui sei di questi sono ancora assi virtuali, che abbiamo collegato, dopo una trasformazione, con sei assi reali, che quindi corrispondono a questo punto ai sei motori del robot. Dove il collegamento tra assi avviene tramite le cosiddette "camme elettroniche", ovvero una relazione precisa tra la posizione, velocità e accelerazione dell'asse master e quelle dell'asse slave.

In questo modo, seguendo il master virtuale possiamo eseguire, ciclicamente, il movimento desiderato nei tre orientamenti e rotazioni. Mentre per gli assi reali le posizioni da settare negli azionamenti vengono dal programma main, ovvero dalle sei posizioni degli assi virtuali, vengono generati tramite una matrice omogenea di trasformazione altre sei posizioni con un sistema di riferimento solidale alla base del robot e poi con la funzione cinematica inversa si ottengono le sei posizioni da inviare agli assi reali. Nel nostro caso, la lunghezza delle camme delle traiettorie

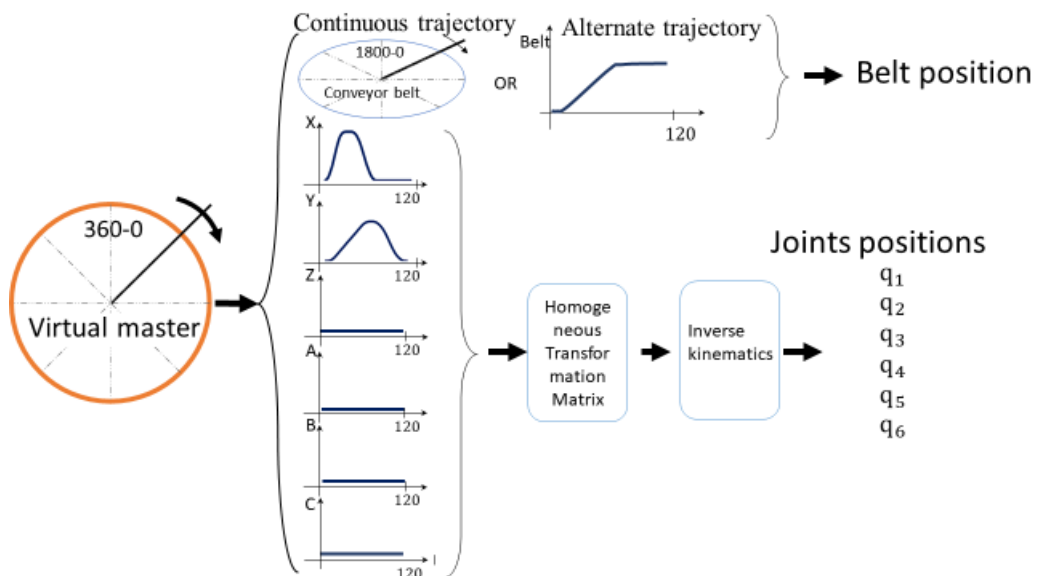


Figura 3.1 Architettura Motion

nello spazio è di 120 unità che corrisponde a un terzo del periodo del master e questo può darci l'idea della produttività. Pertanto, tenendo presente che 120 unità master corrispondono a un gruppo di solito di quattro o sei prodotti dipende dal tipo camma pianificata e dell'end-effector usato, possiamo utilizzare la velocità dell'asse master

per calcolare la produttività della macchina ovvero quanti prodotti al minuto riusciamo a processare. Quindi, dopo aver stabilito il collegamento tra il master virtuale con altri sei assi virtuali tramite camme elettroniche, rappresentanti le tre traslazioni e le rotazioni nello spazio. Le sei posizioni di questi slave vengono quindi, prima moltiplicate con una matrice di trasformazione omogenea. Trasferendo quindi il sistema di riferimento dall'end-effector ad uno sincrono con la base nel cosiddetto "world space" così poi da ottenere tramite la cinematica inversa la corrispondente posizione nello "spazio dei giunti". Con queste posizioni dei giunti possiamo quindi impostare efficacemente la posizione nei motori e spostarci verso la posizione desiderata per i robot, andando a scrivere con opportune funzione tali posizioni nei parametri dell'azionamento che provvederà ad effettuare lo spostamento desiderato. Per svolgere il compito proponiamo due diversi tipi di movimento, alternato e continuo, in questo modo, ad esempio, possiamo gestire o meno una ipotetica pesatura dei prodotti per il controllo statistico.

3.1 Sincronizzazione nastro trasportatore-robot

Per ottenere le migliori prestazioni in termini di sincronizzazione tra nastro trasportatore e robot, abbiamo considerato alcuni aspetti di interesse. Il primo dei quali riguarda la locazione degli assi master virtuali all'interno dell'architettura software, ovvero andiamo ad analizzare dove questa struttura viene definita e quindi

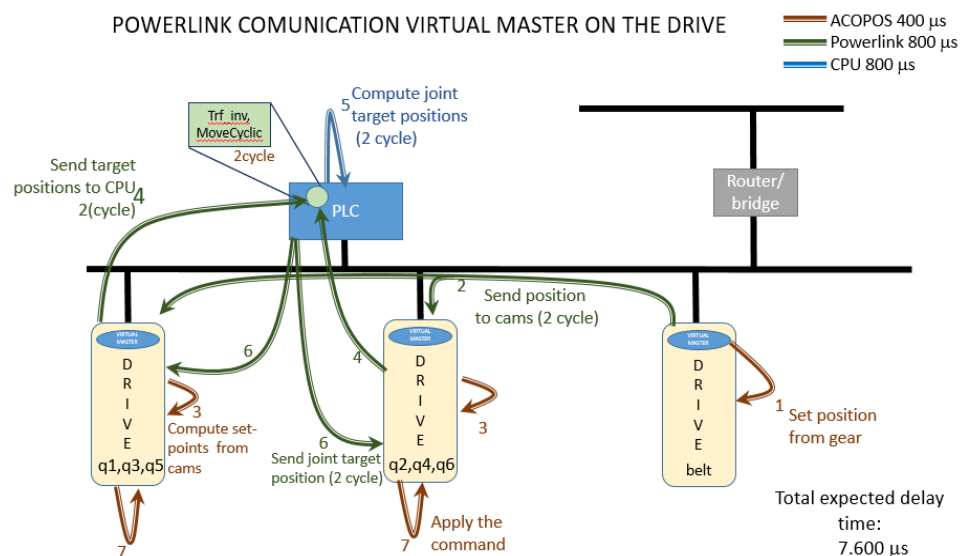


Figura 3.2 Comunicazione POWERLINK, master sull'azionamento

a quale locazione di memoria si deve raggiungere per utilizzare tale struttura nella parte ciclica dell'applicazione. Quindi il primo passo verso il miglioramento della sincronizzazione è stato quello di spostare, nel senso di definire, gli assi virtuali del

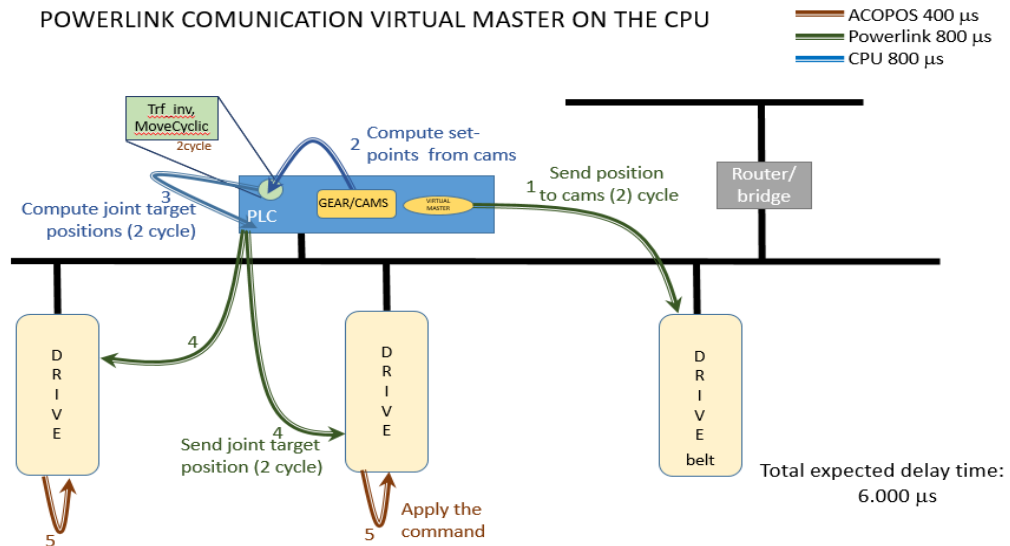
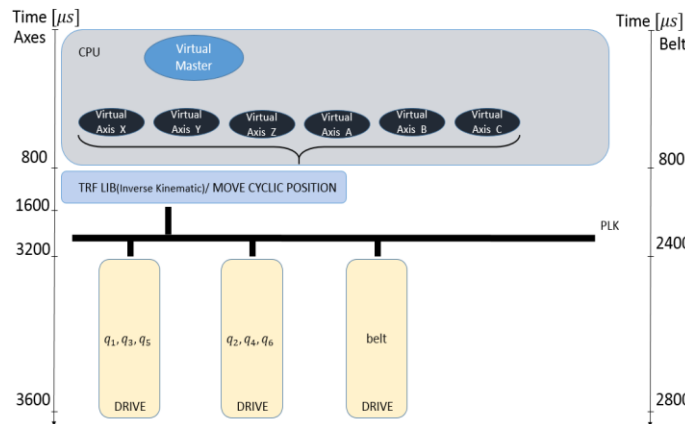


Figura 3.3 Comunicazione POWERLINK, master sulla CPU

robot, dagli azionamenti ACOPOS P3 alla CPU, questo infatti in qualche modo aiuta sicuramente a ridurre il tempo di ritardo della comunicazione dei set-point di posizione che vengono trasmessi ciclicamente dalla CPU. Per calcolare il tempo di comunicazione della cinghia e degli assi abbiamo considerato i seguenti contributi. Ovvero, un tempo di ciclo della CPU per l'elaborazione della nuova posizione del master e la posizione corrispondente nelle camme. Un altro tempo di ciclo della CPU (nel caso degli assi del robot) per l'elaborazione della cinematica inversa, poiché tale elaborazione necessita di informazioni dagli assi che vengono calcolate solo all'inizio del ciclo della CPU e due tempi di ciclo POWERLINK (decisi dal master della comunicazione) per informazioni sulla trasmissione dalla CPU tramite la rete POWERLINK e un ulteriore tempo di ciclo del driver per iniettare il comando degli azionamenti ai motori.

Come possiamo vedere con questa architettura software, si ha una piccola differenza nel tempo di comunicazione tra CPU e robot rispetto al tempo di



comunicazione tra CPU e nastro trasportatore, questo a causa della mancanza della trasformazione cinematica inversa per la posizione del nastro trasportatore. Quindi per gestire questa differenza e sincronizzare i due movimenti, è stato introdotto

Figura 3.410 Architettura del software

il parametro "t_total" (tempo di ritardo totale) che può essere modificato nella "tabella dei parametri init" del singolo asse. Infatti, impostando il giusto valore a "t_total" nell'asse, possiamo ritardarne la partenza del movimento, per sincronizzare il movimento dell'intero sistema. Un esempio di tale utilizzo è mostrato in figura 3.5 dove appunto si cerca di sincronizzare la partenza dei robot introducendo un

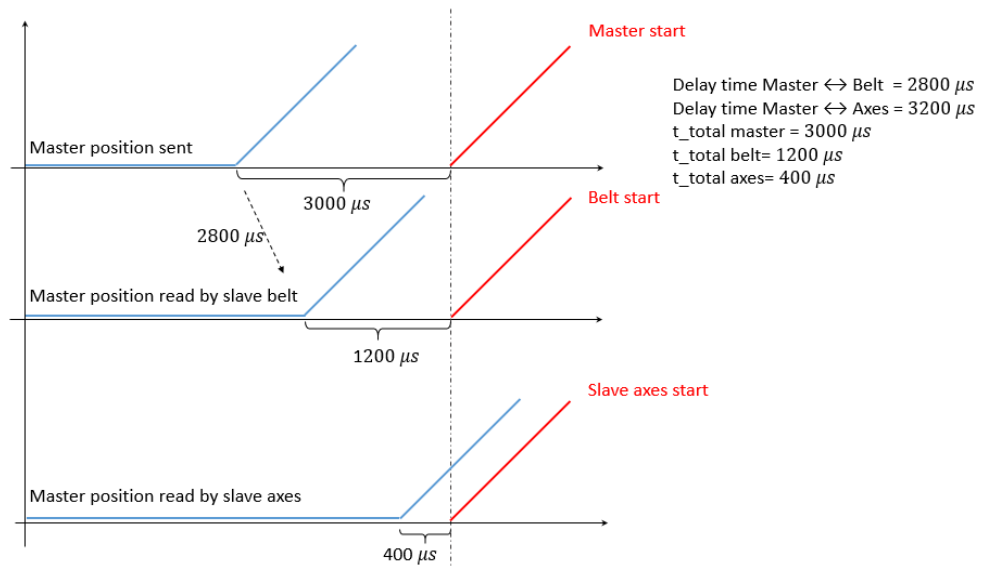


Figura 3.5 Applicazione del parametro t_total nella comunicazione della posizione

parametro t_total diverso per ogni gruppo di assi. Altre fonti di ritardo potrebbero essere il modo in cui estraiamo le informazioni sulla posizione corrente dei giunti, utile nel calcolo della cinematica inversa. Ovvero, possibili ritardi nella lettura di questa informazione possono derivare dal l'uso della funzione "MC_ReadStatus", infatti, questa estrae la posizione effettiva dalla struttura dell'asse globale (es.

GAxis.monitor.s) che ha lo svantaggio di non essere aggiornata in tempo reale. Un approccio alternativo a questo per la lettura della posizione, consiste nell'utilizzare la funzione "MC_BR_ReadCyclicPosition", che garantisce invece letture in tempo reale, dedicando uno spazio appropriato nel frame di comunicazione POWERLINK. D'altra parte, per quanto riguarda la scrittura delle variabili, abbiamo almeno due possibilità, che utilizzano rispettivamente i due blocchi funzione "MC_BR_MoveCyclicPositionExt" ed "MC_BR_CyclicWrite". L'idea principale dei due approcci è per il primo, adattare il comando dato se necessario, ai limiti di velocità e accelerazione predefiniti. Cioè, il valore dato all'azionamento viene primo saturato e interpolato per fare rispettare i limiti, seguendo il metodo di interpolazione scelto, e quindi trasferito all'azionamento. L'approccio alternativo consiste invece nello scrivere direttamente il comando nel drive lasciando all'utente (ovvero all'applicazione) il compito di dove inserire i giusti valori per l'azionamento rispetto i limiti fisici dell'asse, e in caso contrario sarà l'applicazione di sicurezza ad intervenire, che bloccherà i motori che hanno superato la velocità o coppia limite. [4]

3.2 Traiettoria continua

Per traiettoria continua in questo progetto si intende una traiettoria per il robot ed il nastro trasportare che non prevede soste, almeno per quanto riguarda il nastro trasportatore, che quindi scorrerà ad una velocità costante che corrisponderà ad un preciso rapporto di riduzione con il master virtuale. Per la creazione di tale traiettoria si è fatto uso di CAD meccanici, ovvero, si è disegnando nel CAD i punti

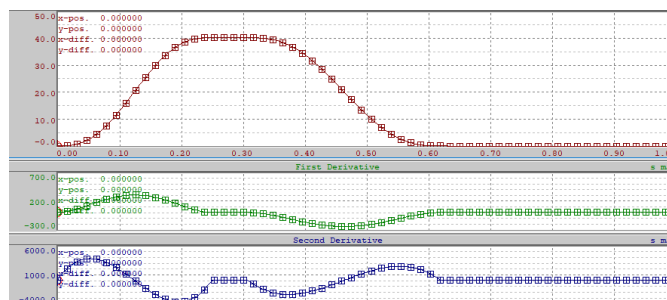


Figura 3.611 Traiettoria continua dell'asse x del robot

nello spazio i punti in cui l'end-effector del robot dovrebbe attraversare e le corrispondenti posizioni del nastro trasportatore istante per istante, quindi si sono creati i vettori di posizione, velocità e accelerazione per i

sei gradi di libertà nello spazio e i profili per il nastro trasportatore, del quale viene mostrato un esempio in figura 3.6 e 3.7 per gli assi x e y del robot.

La traiettoria continua, è il movimento di base progettato per il robot e il nastro trasportatore, ovvero la traiettoria in cui non abbiamo la funzionalità del valutare il peso. Per pianificare questa traiettoria, sono state quindi progettate sei "camme

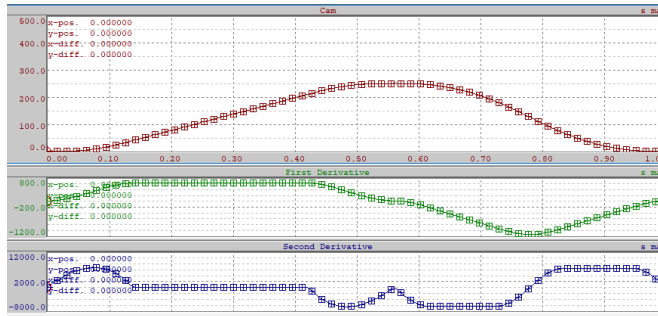


Figura 3.7 Traiettoria continua dell'asse y del robot

elettroniche" per i sei gradi di libertà del robot e in più una particolare "camma elettronica" per il nastro trasportatore, chiamata anche "electronic gear" in

Automation Studio. Questa ultima camma speciale ha il

compito di instaurare una semplice relazione tra il master virtuale e il motore del nastro trasportatore, ovvero, un rapporto di riduzione fra di essi, in modo tale che, ad ogni tot rivoluzioni del asse master virtuale corrisponda un giro del nastro trasportatore, che nel nostro caso corrisponde ad un rapporto 5:1. Per poter implementare questo rapporto si hanno principalmente due opzioni. Una è l'utilizzo della funzione conosciuta come "electronic gear" il che ci fa settare un rapporto preciso e fisso tra gli assi, la seconda opzione equivalente consiste invece nel utilizzare una camma elettronica opportunamente costruita tale da ricreare il rapporto voluto che quindi dovrà essere una relazione lineare. In questo progetto per motivi di semplicità del codice è stato scelto di utilizzare la seconda opzione e quindi si è creato una "camma elettronica" lineare con un'altezza 600 unità e una lunghezza di 120 unità di gradi master, con cui si ottiene il rapporto desiderato. La scelta di utilizzare la camma elettronica per il nastro trasportato è giustificata dal fatto che ci permette una più rapida transizione del sistema nella pianificazione del movimento con più camme, così che tutti gli assi abbiamo la stessa struttura e la modifica su un asse può essere immediatamente estesa a tutti gli assi necessari.

3.3 Pianificazione del movimento con più "camme elettroniche"

Per varie ragioni, potremmo voler pianificare diverse traiettorie per il robot e il nastro trasportatore, in modo da poter gestire il maggior numero di situazioni possibili. Per raggiungere questo obiettivo abbiamo diverse opportunità, dalla più semplice e immediata alla più complessa e strutturata. Di seguito mostreremo una delle soluzioni possibili che probabilmente risulta anche la soluzione più semplice,

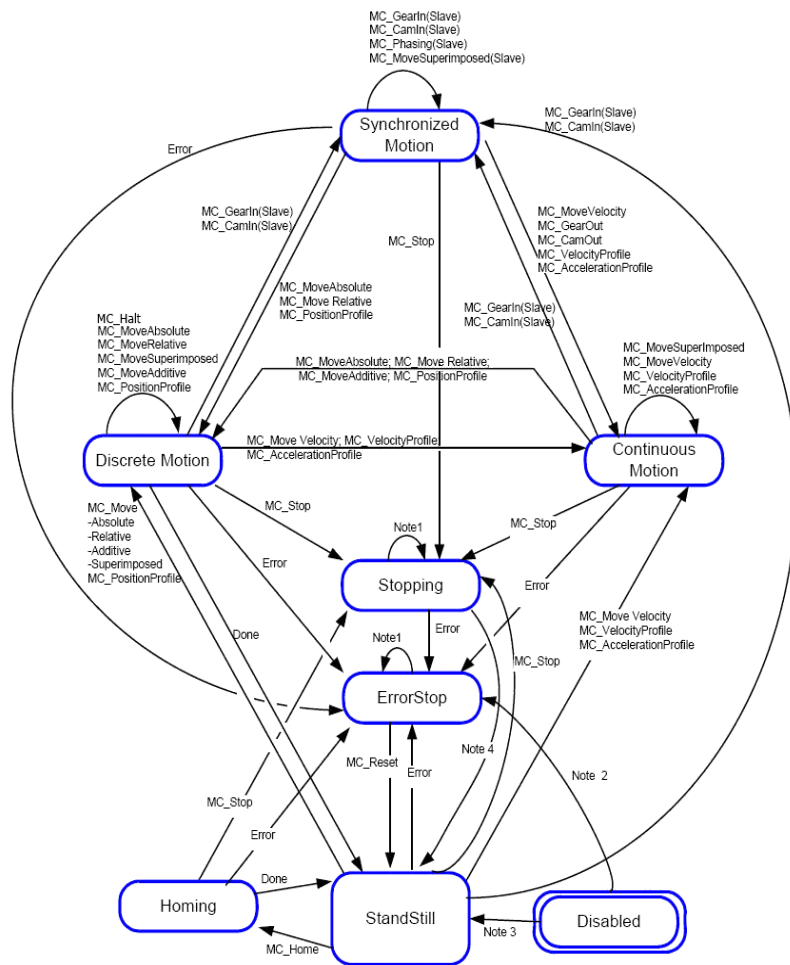


Figura 3.8 Diagramma della macchina a stati PLCopen

che verrà poi effettivamente utilizzata per ragioni di efficienza e parsimonia. Questa soluzione può essere meglio compresa con l'aiuto del diagramma della macchina a stati PLCopen in figura 3.8. In cui possiamo vedere che una volta stabilita quale "camme elettronica" eseguire, tramite la funzione "MC_CamIn", lo stato dell'azionamento per quell'asse diventa "Synchronized Motion". A questo punto, se volessimo cambiare la traiettoria, dovremmo eseguire nuovamente la funzione "MC_CamIn" con parametro la nuova "cam elettronica". Ma prima di questo dobbiamo riportare lo stato dell'asse allo stato "Standstill", questo avverrà in due passaggi il primo con la chiamata alla

funzione "MC_Stop" e il secondo step, quando l'uscita "execute" del blocco funzione "MC_Stop" torna a false. Pertanto, una volta che l'asse è tornato allo stato "Standstill", possiamo caricare la nuova "camma elettronica". Il principale svantaggio di questo metodo per la sostituzione della camma elettronica è sicuramente la necessità di dover fermare l'asse di interesse, aspetto che comunque può essere ignorato se non ci sono particolari requisiti real-time nell'esecuzione di tale operazione, come nel caso in questione, dove non si hanno vere e proprie composizioni dei movimenti per mezzo dell'unione di camme, ma semplicemente si ha un cambio della traiettoria eseguita per aggiungere nuove funzionalità. Quindi nell'operazione di sostituzione si può tranquillamente effettuare una pausa il che ci lascia liberi di utilizzare il metodo appena descritto.

Un metodo leggermente più complesso ma con vantaggi per l'esecuzione real-time della sostituzione della traiettoria consiste nell'uso del "cam profile automat".

3.3.1 Cam profile automat

Il metodo sicuramente più efficiente per caricare più "camme" è il "Cam Profile

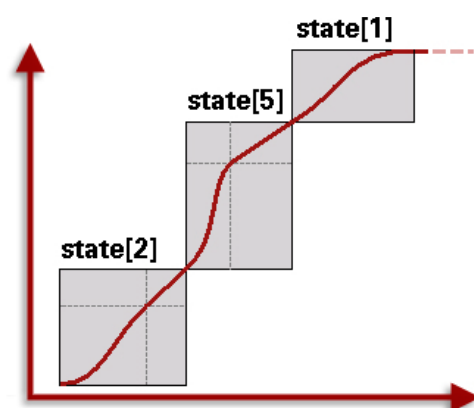


Figura 3.9 Sequenza di stati di "Cam Profile Automat"

Automat". Questo metodo consente di unire in modo preciso traiettorie diverse quando si verifica un evento. Più nel dettaglio, il "cam profile automat" è una macchina a stati con determinati parametri che consente di combinare più "camme elettroniche" compensando qualsiasi discontinuità nella posizione dell'asse slave e nella transizione tra una camma e l'altra. Il "cam profile automat" è una

struttura caricata nell'asse slave con un certo numero di stati, al massimo sei ed eventi definiti per la transizione tra uno stato e l'altro, in cui ogni stato è caratterizzato dalla propria traiettoria. Pertanto, ciò che accade in realtà è che le traiettorie sono caricate nell'azionamento dell'asse e il programma di controllo verificherà l'arrivo di un segnale di trigger, o ancora meglio, un metodo più

efficiente è quello di consentire all'azionamento stesso di decidere quando cambiare la camma tramite parametri e condizioni definite durante la creazione del "cam profile automat". Ciò consentirà una transizione più rapida, nonché una

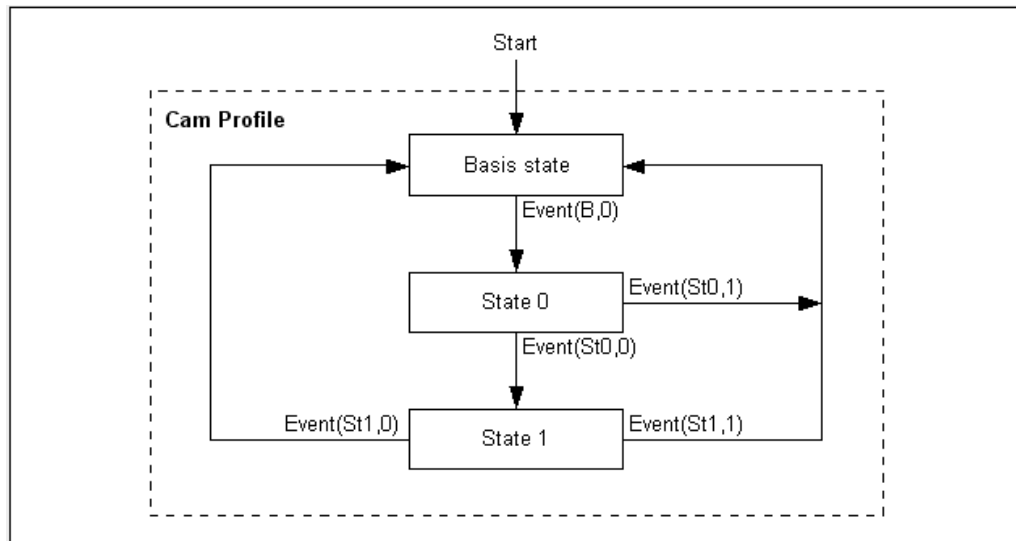


Figura 3.10 Esempio di "cam profile automat"

riduzione della complessità del programma di controllo e una risposta più reattiva. Il blocco funzione principale per la parametrizzazione e l'esecuzione del "cam profile automat" è il "MpAxisCamSequencer", grazie al quale è possibile definire gli stati e gli eventi per il passaggio da uno stato all'altro. Pertanto, una volta abilitato il blocco funzione, tutte le traiettorie verranno caricate nell'azionamento e il sistema automatico di camme sarà in esecuzione. [4] [13]

3.3.2 Traiettoria alternata

Per lasciare più funzionalità al robot, sono state pianificate diverse traiettorie, la "continua" e l'"alternata". Per la prima possiamo utilizzare "l'architettura del movimento" già vista. Mentre per la seconda, possiamo, lasciare l'architettura invariata e sostituire le camme, oppure in alternativa, possiamo interporre un blocco tra il master virtuale e gli assi rimanenti per programmare una pausa. Vale a dire, interporre un asse, tra il master e gli assi virtuali che rappresentano le tre traslazioni e rotazioni nello spazio: x, y, z, a, b, c, che farebbe fare all'intera macchina composta da robot e nastro trasportatore una pausa alla fine di il processo di riempimento, in modo da consentire la pesatura ai fini del controllo. Questo metodo presenta tuttavia

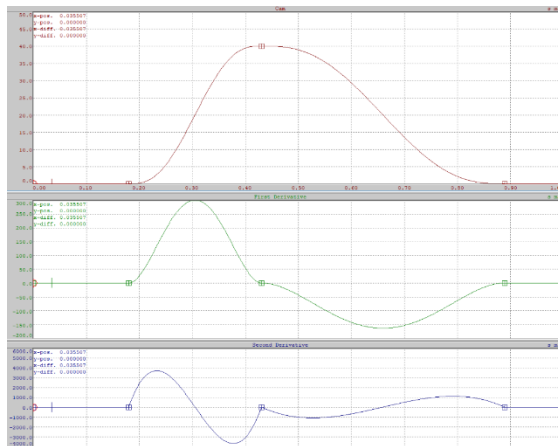


Figura 3.11 Traiettoria alternata dell'asse x del robot

è quello di lasciare invariata l'architettura del movimento e modificare invece le cosiddette "camme elettroniche", in modo che il nastro trasportatore si fermi al

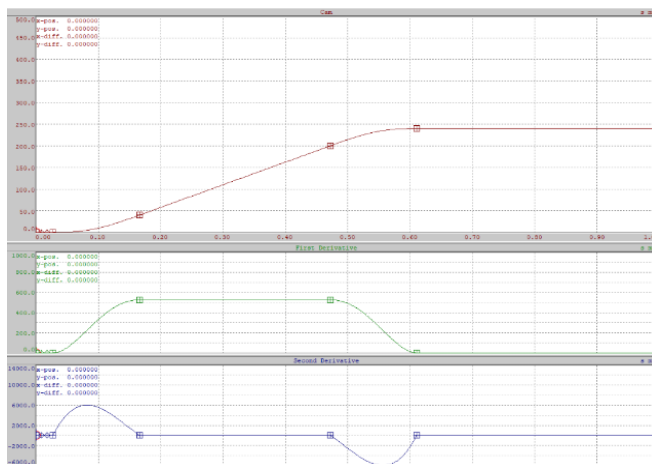


Figura 3.12 Traiettoria alternata del nastro trasportatore

Mentre per il secondo approccio (quello usato in pratica nel progetto), dobbiamo cambiare le camme per l'asse X e Y del robot e quello relativo al nastro

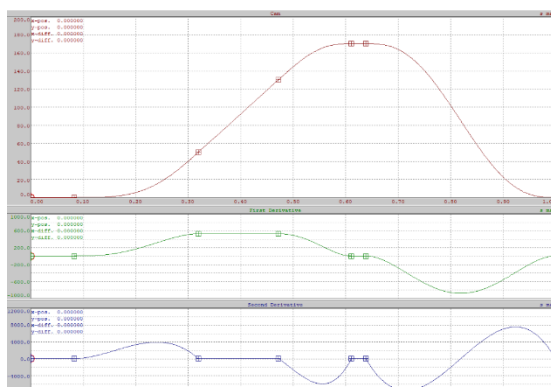


Figura 3.13 Traiettoria alternata dell'asse y del robot

alcuni svantaggi poiché in questo modo sia il nastro trasportatore che il robot sono portati a fermarsi e se per il nastro trasportatore questo può essere utile ad esempio per pesare i prodotti, per il robot che in questa situazione sarebbe libero di muoversi, facendolo fermare si introduce inevitabilmente dell'inefficienza nella traiettoria. Il metodo alternativo

è quello di lasciare invariata l'architettura del movimento e modificare invece le cosiddette "camme elettroniche", in modo che il nastro trasportatore si fermi al momento giusto, mentre il robot può completare la sua traiettoria, andando a mettersi in posizione per essere pronto al prossimo ciclo. In entrambe le opzioni, tuttavia, abbiamo bisogno di cambiare le "camme elettroniche". Vale a dire, per il primo metodo dobbiamo cambiare il blocco tra il master e gli altri assi.

trasportatore, per ottenere questo, si usa uno dei metodi precedentemente descritti per cambiare le "camme elettroniche". La traiettoria alternata, come le altre, sono state progettate con l'aiuto di un CAD meccanico e quindi trasferite in Automation Studio. Per trasferire la traiettoria, vengono memorizzati una serie di

punti della traiettoria presi dal CAD e quindi, una volta trasferiti in Automation Studio, viene eseguita l'interpolazione per un massimo di 64 punti. In ciascuna di queste traiettorie viene utilizzato un polinomio di quinto grado per interpolare i punti o alternativamente si può sostituire con altri supporti, quali spline, seni modificati, trapezi modificati, polinomi di sesto grado, e così via. Inoltre, oltre alla relazione fra posizione master-slave, dobbiamo impostare la relazione per le velocità e le accelerazioni.

Nel progettare questa traiettoria, occorre prestare particolare attenzione anche ai limiti di accelerazione e velocità, infatti, Automation Studio eseguirà l'arrotondamento di tutte le sezioni della traiettoria in cui si supererebbe tali limiti. Per quanto riguarda gli aspetti pratici di questa pianificazione possiamo dire che questa traiettoria è stata definita tra $0 \div 1$ per le posizioni master e quindi estesa tramite opportuni coefficienti fino a 120 unità, mentre è stato scelto di definire direttamente nella scala corretta quindi senza fattori di scale le posizioni degli slave. Come possiamo vedere in figura sono state rappresentate le traiettorie per il nastro e gli assi x e y del robot. Ciò che rimane da analizzare, sono le traiettorie per l'asse z del robot e le tre traiettorie per le tre rotazioni attorno a x-y-z, che in questo caso sono semplicemente delle linee rette con valore costante uguale a zero per fissare l'orientamento del end-effector nel "word space", che tuttavia comporterà ovviamente a piccoli movimenti nel "joint space".

3.4 Confronto tra traiettorie e risultati di simulazione

Al fine di validare il progetto software, abbiamo effettuato simulazioni utilizzando il simulatore interno di "Automation Studio" e successivamente testato sul sistema reale. Una volta verificata l'accuratezza delle traiettorie e compensate le piccole discrepanze tra CAD e sistema reale, è stata verificata la fattibilità delle traiettorie in relazione ai limiti di coppia e velocità. Questi test al fine di estrapolare quanti più dati utili possibili, sono stati effettuati alla velocità nominale della macchina, cioè alla velocità con cui la macchina automatica dovrebbe lavorare in condizioni normali, ovvero alla velocità per cui abbiamo una produttività di circa 420 prodotti al minuto. Questa produttività corrisponde a una velocità master di 140 unità/s.

Infatti, se consideriamo che un'intera traiettoria, sia continua che alternata, è lunga 120 unità master, alla velocità master di 140 unità/s corrispondono 1.1667 traiettorie/s che se ripetuto per 60 secondi equivale a 70 corse al minuto. Quindi per un end-effector con 6 cannuce corrisponde a 420 prodotti/min, a queste velocità si sono quindi effettuati dei test e i dati estrapolati sul sistema reale sono alcuni mostrati così per come sono ed altri post-elaborati su MATLAB e illustrati su grafici per motivi di semplicità. Questi dati sono stati estratti mediante la funzione "trace" di "Automation Studio", che durante il funzionamento del robot e del nastro trasportatore alla velocità master di 140 unità/s, memorizza momento per momento le variabili di interesse direttamente dall'azionamento precedentemente selezionato. In questa prima analisi, si è scelto di focalizzare

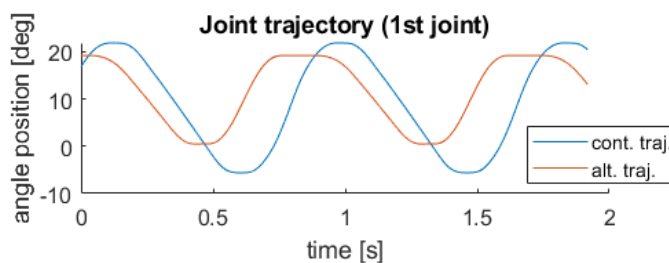


Figura 3.14 Traiettoria di posizione del primo giunto

l'attenzione sulla velocità e la coppia dei primi due giunti del robot perché considerati i più importanti, per via del loro fondamentale contributo al movimento per le due

traiettorie. Infatti possiamo affermare che per entrambe le traiettorie la maggior parte del movimento è dovuto ai primi due motori, ovvero, al primo motore per i movimenti lungo l'asse y (asse parallelo al nastro trasportare) e al secondo per i movimenti lungo l'asse x (l'asse verticale e perpendicolare al nastro trasportatore). Inoltre, per un'analisi più approfondita delle due traiettorie, sono state illustrate anche le posizioni del primo giunto nelle due camme e viene riportato lo spettro di frequenza dell'accelerazione delle due traiettorie al fine di studiarne le proprietà dinamiche attraverso il loro contenuto armonico. Il primo aspetto evidenziato in questo confronto è la traiettoria della prima articolazione nei due casi. Dal grafico possiamo avere un'idea di come saranno in seguito gli aspetti di interesse, quali velocità e coppia. Infatti, vediamo una maggiore escursione per la "camma continua", in un tempo comparabile, che probabilmente porterà a maggiori velocità e accelerazioni. Infatti, dall'analisi delle velocità dei primi due giunti. Possiamo, prima validare la loro fattibilità per quanto riguarda i limiti di velocità massima. Infatti, i limiti per il primo dei due giunti ricordiamo essere di 360 °/s e per il secondo giunto di 300 °/s, che risultano quindi essere superiori ai picchi massimi

di velocità assunti dai due giunti nelle due traiettorie. Quindi dal confronto delle

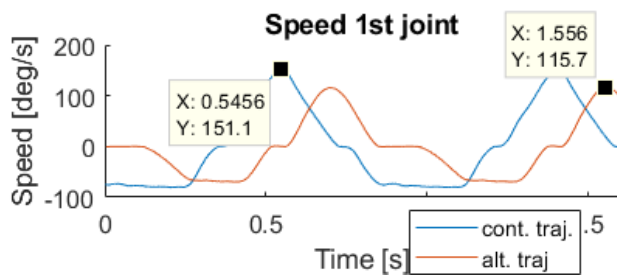


Figure 3.1512 Traiettoria di velocità del primo giunto

velocità nelle due traiettorie nei primi due giunti, possiamo confermare, come precedentemente ipotizzato. Infatti, per eseguire la traiettoria senza interruzioni, il primo giunto deve raggiungere una velocità assoluta più elevata, specialmente nella fase di ritorno alla posizione iniziale, dopo aver inseguito e inserito le cannuce dell'end-effector all'interno dei flaconi. Allo stesso modo, possiamo estendere l'analisi alle coppie necessarie per le traiettorie, che ci daranno poi informazioni sulle accelerazioni in gioco. Poiché tali accelerazioni non sono direttamente misurabili, ma solo derivabili dalla velocità e

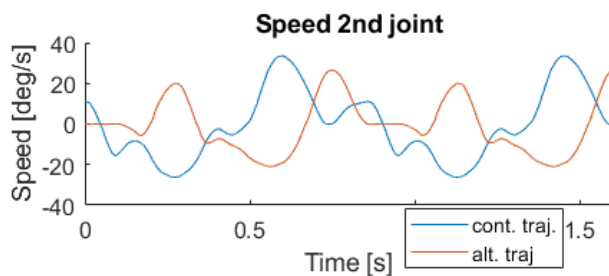


Figura 3.1613 Traiettoria di velocità del secondo giunto

quindi soggetti ad errori numerici. L'analisi delle coppie ci darà quindi un'idea anche dello stress che i motori del robot dovranno sostenere per una data traiettoria. Quindi una volta misurate le variabili di interesse, possiamo confermare l'effettiva fattibilità rispetto ai limiti di coppia, ricordando che questi limiti sono fissati a $2,42 \text{ N/m}$ per il primo giunto e $1,81 \text{ N/m}$ per il secondo, il che ci consente di affermare che questi limiti sono ampiamente soddisfatti per il secondo giunto e abbastanza al di sotto del limite per il primo giunto. Inoltre, da questi grafici, è possibile prevedere quale possa essere la traiettoria più critica e questo può essere ricercato e compreso nei picchi di coppia necessaria per i primi due giunti. In particolare, possiamo vedere che il picco con il valore assoluto più alto (in valore negativo) nella coppia, in entrambe le traiettorie è visto poco prima della fase di inserimento, ovvero, quando il robot una volta terminato il riempimento torna alla posizione iniziale alla massima velocità e poi deve fermarsi per ricominciare la traiettoria nel verso opposto, mentre il secondo picco più alto soprattutto nella traiettoria continua si ha quando il robot termina l'inseguimento, cioè, quando, una volta terminata la

velocità nelle due traiettorie nei primi due giunti, possiamo confermare, come precedentemente ipotizzato. Infatti, per eseguire la traiettoria senza interruzioni, il primo giunto deve raggiungere una

quindi soggetti ad errori numerici. L'analisi delle coppie ci darà quindi un'idea anche dello stress che i motori del robot dovranno sostenere per una data traiettoria. Quindi una

riempitura dei flaconi, l'end-effector viene tirato fuori dai flaconi, cambia direzione e il più velocemente possibile torna alla posizione iniziale della traiettoria. Questo secondo picco nella fase successiva all'inseguimento non è presente o comunque molto attenuato nella camma alternata, poiché questo in quest'ultimo caso il robot dovrà in confronto far un cambio di direzione meno brusco in quanto a differenza della camma continua, nella fase finale di riempimento il nastro trasportatore inizia a rallentare fino fermarsi per effettuare la pesatura dei prodotti. Confrontando quindi

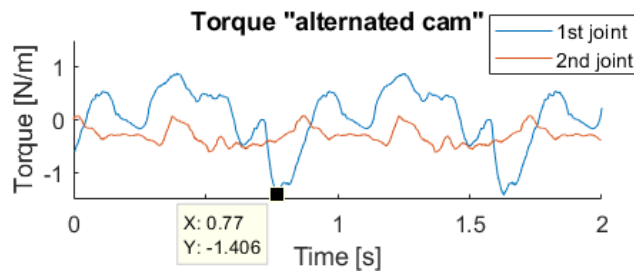


Figura 3.17 Coppie dei primi due giunti nella traiettoria alternata

Infatti nella "traiettoria continua" come abbiamo analizzato vi è un picco in valore positivo, non presente o comunque più attenuato nella traiettoria con pause, per via del cambio di direzione più brusco, dovuto all'inseguimento del nastro trasportatore che scorre ad una velocità costante, a differenza della "traiettoria alternata" nella quale il nastro alla fine della traiettoria rallenta fino a fermarsi. Quindi nel caso traiettoria continua l'end-effector una volta tirato fuori dai flaconi dovrà fermarsi il più presto possibile e cambiare direzione, mentre nella "camma alternata" poiché la pausa si verifica alla fine del processo di riempimento, il cambio di direzione sarà

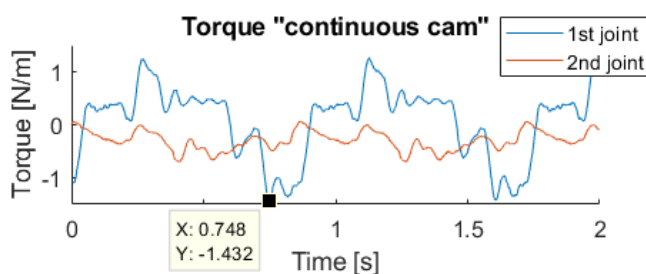


Figura 3.18 Coppia dei primi due giunti nella traiettoria continua

abbiamo un "picco" di valore negativo più accentuato nella camma continua per via della fase iniziale dell'inseguimento infatti nella traiettoria continua questa avviene con il nastro trasportatore a velocità costante quindi più critico rispetto ad una partenza con il nastro trasportatore fermo come nella traiettoria alternata. Un'idea invece dello stress meccanico del sistema si può ricavare dall'analisi delle traiettorie

le due traiettorie e le loro criticità per quanto riguarda i limiti di coppia possiamo affermare che la traiettoria senza soste risulta meno critica sotto questo punto di vista.

meno brusco poiché il nastro trasportatore avrà appena ripreso il movimento e quindi seguirà una velocità inferiore. Ma oltre a questo come si può vedere in figura 3.17 e 3.18

nel dominio della frequenza, in particolare dall'analisi di Fourier delle accelerazioni nelle due traiettorie. Pertanto, possiamo affermare che un sistema sarà più stressato,

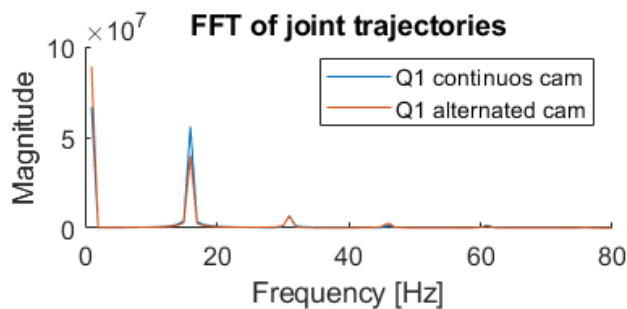


Figura 3.19 Trasformata veloce di Fourier (FFT) della traiettoria in posizione del primo giunto

quanto più alte sono le componenti frequenziali. Poiché le accelerazioni non sono direttamente misurabili dal motore, possiamo eseguire queste analisi o tramite la derivazione delle velocità, che tuttavia, introduce approssimazioni per il calcolo della derivata, oppure, richiamando le proprietà della trasformata di Fourier, in particolare della proprietà della derivazione nel tempo, infatti abbiamo che:

$$F \left\{ \frac{d^n}{dt^n} x(t) \right\} = (j2\pi f)^n X(f) \quad (3.4.1)$$

ciò implica che la trasformazione delle accelerazioni e delle posizioni differisce per un termine moltiplicativo $(-4 \pi^2 f^2)$, che può quindi essere ignorato nel confronto tra le due trasformate di Fourier, cioè tra la trasformazione della traiettoria continua e quella alternata. Pertanto, questa analisi ci dice infine che probabilmente la

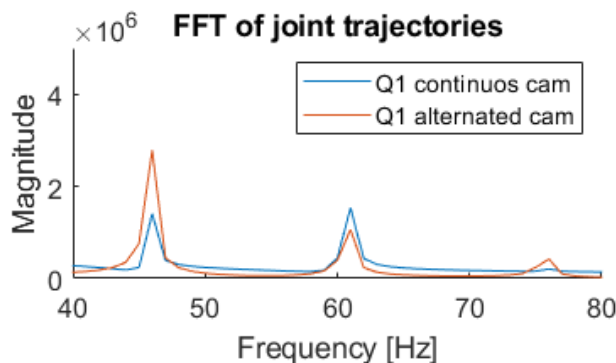


Figura 3.20 Zoom della Trasformata veloce di Fourier (FFT)

traiettoria che meccanicamente sollecita di più il sistema è la traiettoria alternata, a causa delle sue componenti ad alta frequenza. Infatti come possiamo vedere dai grafici, ovvero dai semi spettri in frequenza con le sole frequenze positive di figura 3.19 e 3.20 ottenuti a partire dalle traiettorie in posizione e trasformate con l'algoritmo FFT (Fast Fourier Transform) in matlab, la traiettoria con pause sembra sollecitare meccanicamente di più rispetto alla traiettoria continua, come possiamo infatti dedurre dalle maggiori componenti

tanto più alte sono le componenti frequenziali. Poiché le accelerazioni non sono direttamente misurabili dal motore, possiamo eseguire queste analisi o tramite la derivazione delle velocità, che tuttavia, introduce

traiettoria che meccanicamente sollecita di più il sistema è la traiettoria alternata, a causa delle sue componenti ad alta frequenza. Infatti come possiamo vedere dai grafici, ovvero dai semi spettri in frequenza con le sole

armoniche attorno ai 70/80 Hz non presenti invece nella traiettoria continua. [14]
[15]

3.5 Modello dinamico per la compensazione feedforward della coppia

Per aumentare le prestazioni, ovvero in questo caso la precisione del robot, possiamo considerare l'utilizzo del modello dinamico del robot. Il tipo di controllo che nel progetto eseguiamo per il robot consiste nell'impostare i set-point di posizione di ogni singolo motore e attivarlo per spostarsi verso di esso, in questo tipo di controllo ciò che principalmente ignoriamo sono gli effetti dinamici, o meglio, la relazione tra le forze generalizzate (forza e coppia) rispetto alla posizione, velocità e l'accelerazione del robot. Nel nostro caso questo modello viene utilizzato nella formulazione inversa del problema, in questo modo, dal modello, possiamo ottenere partendo dalle posizioni, velocità e accelerazioni dei giunti lungo la traiettoria, la coppia che potrebbe essere utilizzata per la compensazione feedforward dei motori. Nel nostro caso il modello del robot è stato ottenuto da uno tool MATLAB sviluppato da B&R, che permette, inserendo su di esso i giusti settaggi quali limiti, parametri cinematici e gli effetti di accoppiamento, di generare una traiettoria stimolante che consente di ottenere il modello dinamico finale. Come sappiamo dalla letteratura, i due principali approcci per la costruzione del modello dinamico sono, l'approccio di "Eulero-Lagrange" e l'approccio "Newton-Eulero". Il primo approccio utilizzando le coordinate generalizzate e la nozione di energia cinetica e potenziale, ci fornisce un metodo di calcolo numerico non ottimale. In particolare, la relazione fisica utilizzata è la seconda legge di Newton, il principio di D'Alembert, il principio delle reazioni vincolari e la funzione lagrangiana, che combinati insieme ci portano a creare il modello dinamico.

Un altro metodo più efficiente dal punto di vista del calcolo che andremo ora a analizzare è il cosiddetto approccio "Newton-Eulero", il quale, utilizzando il principio dell'equilibrio tra forze-coppie, crea un metodo ricorsivo per ottenere le forze generalizzate e il movimento del robot. Gli elementi di base per questo approccio sono:

$$f = ma \quad (4.4.1)$$

La seconda legge di Newton, di seguito "l'equazione di Eulero" anche conosciuta come "teorema del momento angolare", cioè:

$$T = \frac{d({}^0I {}^0\omega)}{dt} \quad (4.4.2)$$

In cui T è la somma di tutte le coppie che agiscono sul corpo e ${}^0I {}^0\omega$ sono il momento di inerzia e la velocità angolare espressa secondo un sistema di riferimento inerziale, con l'origine nel centro di massa del primo link. Quindi considerando un singolo "link" e sommando tutte le coppie presenti nel link corrente con quello precedente e successivo, nella forma di coppia pura oppure forza per braccio si ottiene la seguente formulazione.

$$f_i - f_{i+1} + m_i g = m_i \ddot{p}_{C_i} \quad (4.4.3)$$

Quindi l'equazione di Eulero può essere vista come:

$$v_i + f_i \times r_{i-1,C_i} - v_{i+1} - f_{i+1} \times r_{i,C_i} = \frac{d}{dt}(I_i \omega_i) \quad (4.4.4)$$

Il termine " $I\omega$ " per ragioni di semplicità è espresso in un sistema di riferimento allineato con il centro di massa del link corrente (per avere un momento di inerzia costante), quindi visto che il momento di inerzia dovrebbe essere pre- e post-moltiplicato per la matrice di rotazione. Eseguendo la derivata, possiamo ottenere qualcosa come:

$$v_i + f_i \times r_{i-1,C_i} - v_{i+1} - f_{i+1} \times r_{i,C_i} = I_i \dot{\omega}_i + \omega_i \times (I_i \omega_i) \quad (4.4.5)$$

Dove l'accelerazione del link può essere ottenuta dalla derivata della velocità che per un giunto rotazionale che corrisponde a:

$$\dot{p} = \dot{p}_{i-1} + \omega_i \times r_{i-1,i} \quad (4.4.6)$$

Quindi eseguendo la derivata di quest'ultima relazione otteniamo:

$$\ddot{p}_{C_i} = \ddot{p}_i + \dot{\omega}_i \times r_{r_i,C_i} + \omega_i \times (\omega_i \times r_{i,C_i}) \quad (4.4.7)$$

Pertanto, una volta che conosciamo la posizione, la velocità e l'accelerazione giunti, possiamo calcolare la forza e la coppia applicata ai link. In particolare, per ottenere la coppia, la formula ricorsiva computazionale parte dal link più esterno verso il più interno. Quindi partendo dalla forza applicata, massa e accelerazione dell'ultimo link viene calcolata la forza del link precedente dall'equazione (4.4.3), quindi con

la (4.4.5) otteniamo la coppia del giunto precedente e infine con la proiezione della forza lungo l'asse del link otteniamo la forza generalizzata del link precedente:

$$\tau_i = v_i^T z_{i-1} \quad (4.4.7)$$

Estendendo questa procedura per tutti i link, possiamo ottenere la coppia necessaria per una certa traiettoria nello spazio, al fine in questo caso, di una compensazione feedforward. [17]

4. APPLICAZIONE SAFETY

L'applicazione di sicurezza secondo le norme EN ISO 13849 e EN IEC 62061 è certamente un aspetto importante nel progetto di una macchina automatica. La definizione e l'implementazione dei requisiti di sicurezza per le macchine automatiche è un processo complesso e delicato, che coinvolge diverse fasi del progetto che possono essere viste nel ciclo di vita dell'implementazione di un'applicazione di sicurezza, che devono essere rispettate al fine di soddisfare lo standard.

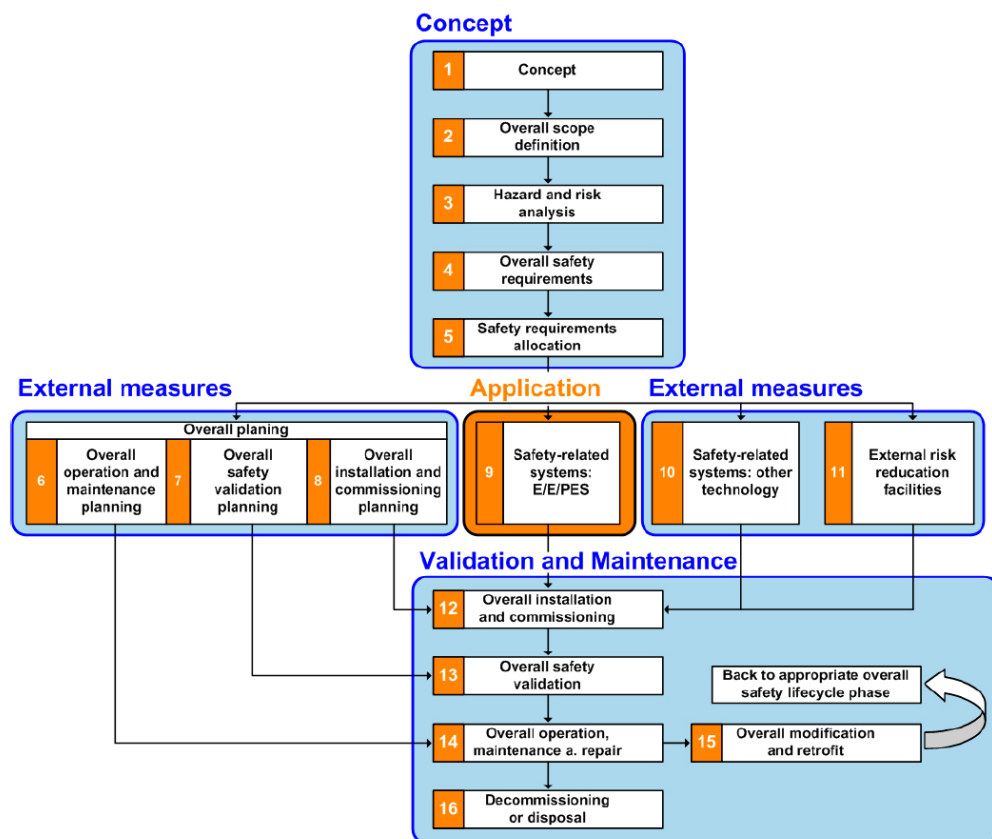


Figura 4.1 Ciclo di vita Safety secondo lo standard EN IEC 61508-1

Nel nostro caso, utilizziamo i sistemi di sicurezza integrati in B&R, che tramite moduli di sicurezza dedicati (I/O decentralizzati, controllo del movimento, protocolli di sicurezza) e applicazioni separate ci può portare a implementare una applicazione con un livello di integrità della sicurezza SIL 3 (Safety Integrity Level). Ovvero, la probabilità che la funzione di sicurezza venga eseguita una volta che ne viene richiesta. Per determinare il livello dell'integrità della sicurezza secondo la norma EN 62061 bisogna effettuare delle considerazioni sulla gravità

delle conseguenze provocate da un evento pericoloso, la frequenza e la durata all'esposizione al danno e così via, per far ciò ci si può servire di un tabella come quella di figura 4.2 dove viene mostrato un esempio di calcolo per una funzione di sicurezza SIL 2.

Probabilità che si verifichi un danno					
Fr		Pr		Av	
Frequenza, durata		Probabilità evento pericoloso		Possibilità di evitare il danno	
<= ora	5	Molto elevata	5		
> 1h <= giorno	5	Probabile	4		
> giorno <= 2 settimane	4	Possibile	3	Impossibile	5
> 2 settimane <= 1 anno	3	Raramente	2	Possibile	3
> 1 anno	2	Trascurabile	1	Probabile	1
Totale: 5 + 3 + 3 = 11					

Gravità del danno		Classe SIL				
Gra		Classe CI				
Conseguenze (gravità)		3-4	5-7	8-10	11-13	14-15
Morte, perdita di un occhio o di un braccio	4	SIL2	SIL2	SIL2	SIL3	SIL3
Permanente, perdita dita	3			SIL1	SIL2	SIL3
Reversibile, con cure mediche	2				SIL1	SIL2
Reversibile, solo primo soccorso	1	Altre misure				SIL1
È necessaria una funzione di sicurezza SIL2.						

Figura 4.2 Tabella di definizione del SIL

L'hardware principalmente utilizzato per l'applicazione di sicurezza include una safetyCPU, moduli I/O di sicurezza e il controllore del movimento "Safe" integrato nell'azionamento ACOPOS P3. Tutto ciò, connesso con il bus di campo Powerlink che ci dà la possibilità di utilizzare il protocollo openSafety. OpenSAFETY è un protocollo di comunicazione basato su un frame Ethernet, aperto e con specifiche focalizzate sulla sicurezza e quindi la trasmissione di informazioni riguardanti aspetti potenzialmente critici per possibilità di causare situazioni di pericolo. Infine, tutto ciò quindi viene unito e programmato nell'ambiente "SafeDESIGNER" in Automation Studio, ovvero, il software necessario per sviluppare l'applicazione di sicurezza, tramite blocchi funzioni PLCopen da librerie certificate come "Safe" e con un proprio livello di affidabilità. Le due applicazione che adesso gestiscono la macchina, ovvero, quella standard scritta in "testo strutturato" e quella di sicurezza scritta in FBD, sviluppate rispettivamente in Automation Studio e SafeDESIGNER anche se hanno una funzionalità del tutto indipendenti fra loro, è concesso l'accesso reciproco ai dati, vale a dire che, sia l'applicazione di sicurezza

che quella standard possono leggere alcune variabili booleane dal programma opposto. [18]

4.1 OpenSAFETY

OpenSAFETY è un protocollo per bus di campo, basato su Ethernet e ideato per applicazioni di sicurezza con caratteristiche che soddisfano i vincoli di applicazioni real-time, utilizzato principalmente per trasferire dati rilevanti per la sicurezza. In questa applicazione vedremo il protocollo openSafety utilizzato con il bus di campo POWERLINK, questa scelta è del tutto arbitraria visto che openSAFETY è definito

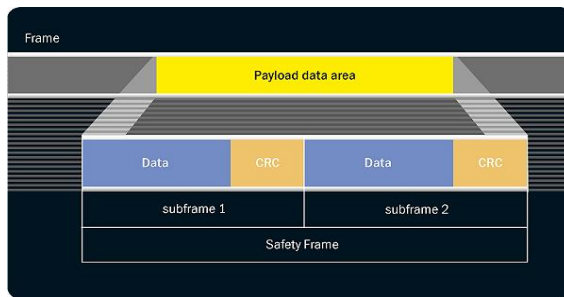


Figura 4.3 Illustrazione schematica di un telegramma openSAFETY

come un frame autonomo indipendente dal bus e in grado di trasferire insieme dati standard e di sicurezza. Il livello di sicurezza raggiungibile con questo protocollo è il SIL 3, con un tempo di ciclo nella comunicazione fino all'ordine dei microsecondi. I maggiori vantaggi

nel uso del protocollo OpenSAFETY deriva dalla grande configurabilità e l'adattabilità nel formato dei telegrammi, infatti il protocollo fa uso di frame uniformi indipendenti dal tipo di dato, configurazioni e tempi di sincronizzazione. La lunghezza del frame può essere determinata dall'applicazione stessa, ed è in grado di utilizzare pienamente un telegramma Ethernet e consente una larghezza di banda fino a 254 byte di dati. Inoltre, ogni nodo "Safety" della rete riconosce automaticamente il contenuto del telegramma, ovvero, il tipo e la lunghezza del frame che quindi possono non essere configurati.

In generale il principale rischio nel trasferimento di dati lungo una rete di comunicazione, è rappresentato dall'inoltro dei dati attraverso i gateway e le possibili interferenze elettromagnetiche presenti. Infatti quest'ultimi potrebbero come nel caso dei gateway, smarrire qualche pacchetto, alimentare la rete sbagliata, o ad esempio se i pacchetti sono divisi a causa della lunghezza dei dati, potrebbero arrivare misti, senza alcuni segmenti del pacchetto o ancora, una delle parti del

pacchetto potrebbe essere ritardata per via del traffico nella rete, oppure, nel caso di interferenze elettromagnetiche, si potrebbe verificare che qualche bit potrebbe essere capovolto o perfino distrutto.

Per ovviare a tutto ciò in OpenSafety sono stati definiti dei meccanismi per la prevenzione dei guasti, uno fra questi è il "timestamp", questo meccanismo



Figure 4.4 Tutti gli errori di trasmissione noti e i meccanismi di riconoscimento degli errori applicabili con openSAFETY

stampando su ogni pacchetto la data e l'ora corrente, evita ad esempio duplicazioni, mixaggi o ritardi eccessivi nella trasmissione dei telegrammi. Inoltre, nel frame sono presenti da 8 o 16 bit di identificazione che codificano parte dell'indirizzo e il tipo di telegramma. Il metodo più

affidabile per il controllo e l'identificazione delle modifiche al contenuto originale dei dati viene effettuata da un meccanismo chiamato CRC. Questa ultima procedura utilizza una chiave per generare un checksum per ogni set di dati che rappresenta una codifica dei dati contenuti nel set stesso, quindi in fase di trasmissione la chiave e checksum vengono allegati entrambi nel pacchetto insieme ai dati, quindi una volta che il pacchetto raggiunge il destinatario il set di dati originale viene calcolato utilizzando la chiave e il checksum e se non corrispondono i pacchetti vengono ignorati. Un altro metodo per la rilevazione degli errori consiste proprio nella struttura dei frame utilizzati nella comunicazione. Infatti i frame openSAFETY sono contenuti nell'area "payload data" del frame standard e sono costituiti da due sub-frame identici con lo stesso contenuto e con un proprio checksum, quindi risulta che la probabilità di avere entrambi i sub-frame modificati o distrutti è estremamente bassa. Comunque anche in questo caso il checksum può essere utile per la individuazione dei fault. La rete OpenSAFETY può contenere fino a 1023 dispositivi con l'opportunità di avere anche domini non omogenei rispetto a dispositivi safe e unsafe, così come, di applicare separazioni gerarchiche e stabilire aree di sicurezza separate nella rete.

In ogni dominio openSAFETY comunque ci sarà un Safety Configuration Manager SCM responsabile del controllo di tutti i nodi della rete. Come già detto, openSAFETY è indipendente dallo standard di comunicazione utilizzato e il frame openSAFETY è incapsulato nella comunicazione in un frame uniforme. Per quanto riguarda la sicurezza e l'integrità dei dati è garantita dalle misure di sicurezza di

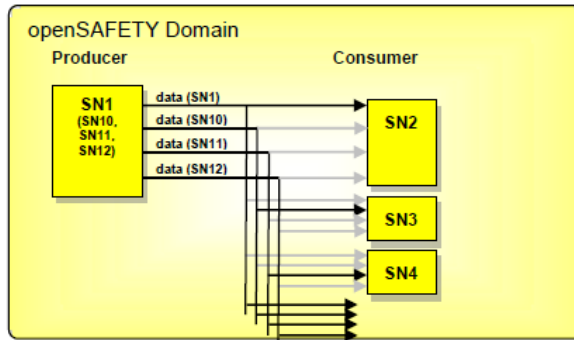


Figure 4.5 Comunicazione estesa Produttore - Consumatore

dello stesso dominio openSAFETY (SD), la comunicazione tra i diversi domini può essere effettuata tramite gli openSAFETY Domain Gateway. Tale comunicazione

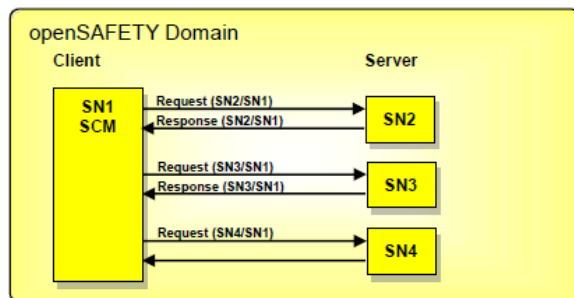


Figure 4.6 Comunicazione server-client

può avvenire ad esempio con il modello di comunicazione produttore/consumatore. In cui i nodi produttori inviano i dati con uno specifico Indirizzo di sicurezza (SADR) e ogni nodo nello stesso dominio può ricevere questi dati.

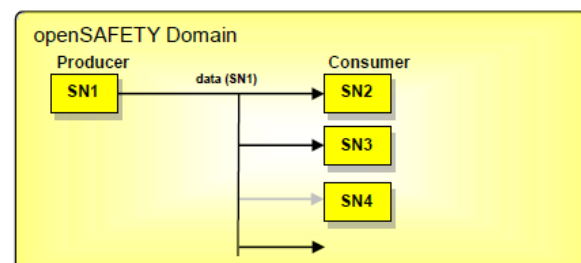


Figura 4.7 Comunicazione Produttore - Consumatore

oltre a questa modalità, c'è anche la possibilità di inviare pacchetti dati solo a specifici nodi del dominio, utilizzando ulteriori indirizzi SADR, che sarà compito del gestore delle configurazioni openSAFETY, ovvero, SCM di rendere questi indirizzi unici all'interno di ciascun dominio.

openSAFETY, mentre la protezione contro gli accessi non autorizzati dipendono dal protocollo di comunicazione utilizzato. In questo ultimo, le comunicazioni di sicurezza possono avvenire solo con i nodi openSAFETY (SN) all'interno

può avvenire ad esempio con il modello di comunicazione produttore/consumatore. In cui i nodi produttori inviano i dati con uno specifico Indirizzo di sicurezza (SADR) e ogni nodo nello stesso dominio può ricevere questi dati.

Oltre a questa modalità, c'è anche la possibilità di inviare pacchetti dati solo a specifici nodi del dominio, utilizzando ulteriori indirizzi SADR, che sarà compito del gestore delle configurazioni openSAFETY, ovvero, SCM di rendere questi indirizzi unici all'interno di ciascun dominio.

Infine, un ultimo possibile modello di comunicazione è la "comunicazione client del server" in cui il client invia una richiesta al server il quale risponde al client specificato. [19] [20] [21] [22]

4.2 Moduli Input/Output Safe

I moduli di input/output Safe sono tipicamente dispositivi connessi con sensori di sicurezza e attuatori, caratterizzati dalla loro multifunzionalità. Un esempio di

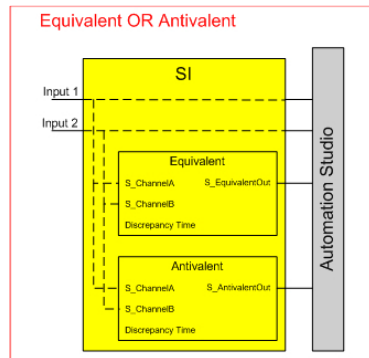


Figura 4.8 Valutazione a Due canali

queste funzionalità è la valutazione a due canali, in cui è possibile impostare una soglia di discrepanza per valutare la correttezza di alcuni input di sicurezza come ad esempio l'arresto di emergenza, barriere fotoelettriche, dispositivi operativi a due mani e così via. Inoltre tali moduli ci forniscono i segnali ad impulsi per la diagnosi della rete del sensore. Per distinguere tra i moduli I/O Safe, ogni dispositivo ha

il suo pattern diverso proveniente dal numero seriale

e dal numero del canale dell'impulso, che grazie al quale si riesce a riconoscere i cortocircuiti rispetto ai potenziali di alimentazione o ad altri canali. Il modulo

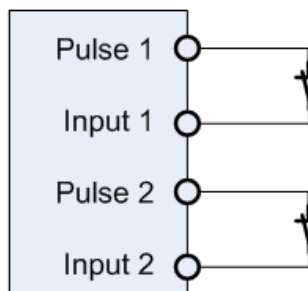


Figura 4.9 Connessione di sensori a un canale

digitale di sicurezza utilizzato è in grado di gestire quattro ingressi di sicurezza, con quattro uscite a impulsi a 24 VDC. Tenendo conto del fatto che il numero di ingressi che dovrebbero essere utilizzati nello stesso tempo dipende dalle condizioni operative e può essere estratto dai dati tecnici. Esistono diversi metodi di cablaggio per i dispositivi di input/output di sicurezza, ciascuno con caratteristiche di rilevamento degli errori

diverse. La connessione più semplice possibile consiste nel collegamento di sensori a un solo canale tramite contatti, in questo caso a ciascun canale di ingresso è assegnata un'uscita clock dedicata. Con questa connessione, il modulo soddisfa i requisiti di Categoria 3 in conformità con EN ISO 13849-1: 2015. Dove, la categoria 3 secondo la norma è caratterizzata dai seguenti comportamenti, ovvero, se un dispositivo rientra nella categoria 3 la funzione di sicurezza continua ad essere eseguita anche se si è verificato un singolo guasto, alcuni difetti, ma non tutti, vengono rilevati e la perdita della funzione di sicurezza è possibile a causa dell'accumulo di guasti non rilevati. Un'altra possibile connessione utilizza la valutazione a due canali direttamente dal modulo, tale tipo di connessione soddisfa

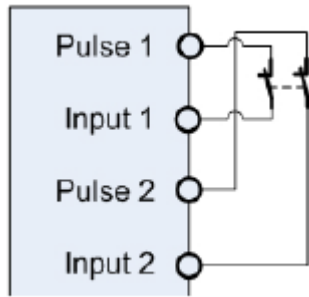


Figura 4.10 Connessione di sensori a due canali

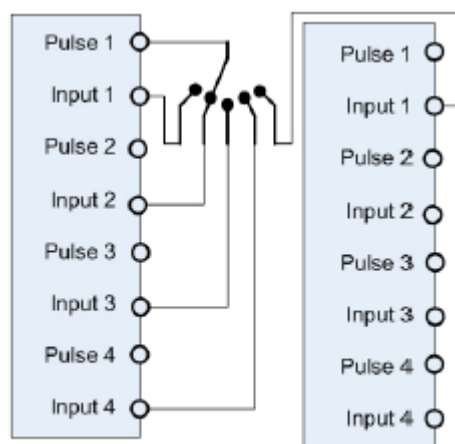


Figura 4.11 Attivazione di interruttori multi canale

possibilità consiste nel connettere i sensori attivi direttamente al modulo, questo tipo di connessione può portare al raggiungimento della categoria 3 con una

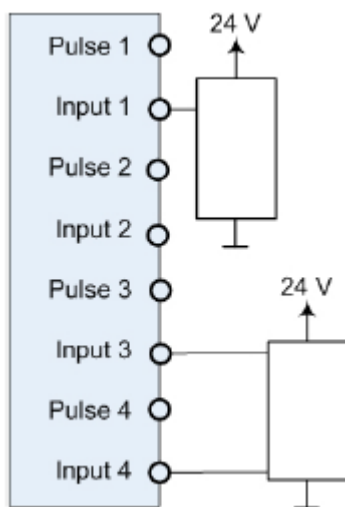


Figura 4.12 Attivazione di sensori attivi

la Categoria 4 della norma già menzionata. I dispositivi di quest'ultima categoria devono essere progettati in modo tale che un singolo guasto non porti alla perdita della funzione di sicurezza e il singolo guasto debba essere rilevato prima o alla richiesta successiva della funzione di sicurezza, cioè immediatamente all'inizio o alla fine di ogni ciclo di funzionamento della macchina. Tuttavia, se il rilevamento dei guasti non è possibile, l'accumulo di

guasti non rilevati non deve comportare la perdita della funzione di sicurezza. Un'altra alternativa per la connessione di sensori è costituita dalla connessione multi-canale con contatti, nella quale, dipendentemente da dove viene fatta la valutazione degli input il segnale d'impulso deve essere diverso per tutti i canali se la valutazione è fatta internamente al modulo, oppure, in caso contrario lo stesso impulso deve essere noto a tutti i moduli. Infine un'ultima

connessione a singolo cavo e fino alla categoria 4 con cablaggio a due canali. Un aspetto importante per i moduli di I/O Safe è sicuramente la comunicazione. Una delle caratteristiche incluse in questo aspetto è il tempo minimo di ciclo, utile ad esempio, per sapere fino a quanto si può ridurre il tempo di ciclo del bus senza errori di comunicazione. Nel nostro caso tale tempo minimo di ciclo corrisponde a $200\ \mu\text{s}$ mentre il tempo necessario per generare un campione va da $400\ \mu\text{s}$ fino a $1750\ \mu\text{s}$, più il tempo di filtraggio. Ogni modulo di ingresso digitale sicuro è dotato di filtri configurabili separatamente. La valutazione a

due canali del segnale viene eseguita da una versione estesa della funzione PLCopen equivalente e antivalente mostrata sotto.

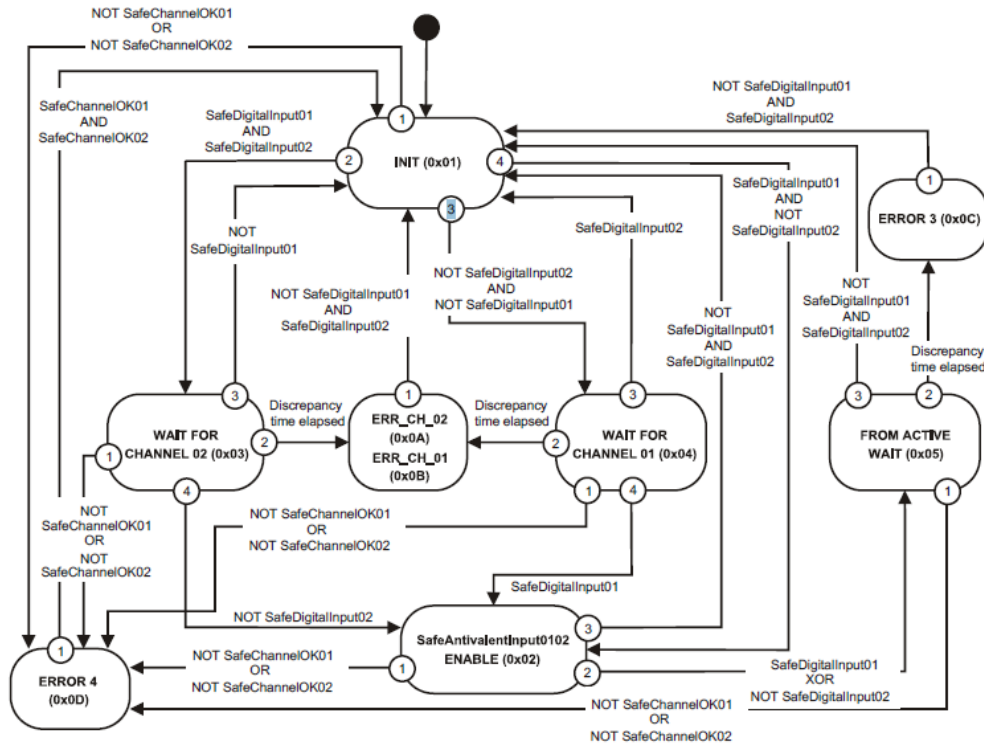


Figura 4.13 Diagramma a Stati del blocco funzione "Antivalent"

Infine, dopo aver analizzato tutti i passaggi necessari nella comunicazione tra PLC e SafeIO, ci soffermiamo adesso sugli aspetti riguardanti i moduli di output "Safe". Principalmente i moduli di output di sicurezza vengono utilizzati per i controlli di attuatori. Esistono due tipi di uscita, una di tipo "A" senza potenziale di terra che può essere quindi utilizzata con attuatori e dispositivi che non hanno potenziale di terra come valvole, relè o altro e quella di tipo B utilizzabile invece con attuatori che prevedono la presenza del potenziale di terra, come ad esempio inverter e così via. Inoltre, i canali di uscita digitali sicuri offrono come azione principale la protezione contro il riavvio indesiderato e la combinazione tra controllo relativo alla sicurezza e il controllo standard è organizzato in modo tale che l'esecuzione di una richiesta di interruzione abbia sempre la massima priorità. L'output di un modulo di sicurezza può essere configurato principalmente in due modalità: l'abilitazione diretta e l'abilitazione via "SafeLogic".

Nel primo caso l'output viene considerato come non-safe nella applicazione standard e in questo metodo si permette quindi di utilizzare l'uscita safe sia nell'applicazione standard che in quella di sicurezza. In questo caso il segnale viene portato come output di sicurezza se questo viene abilitato come tale, sia

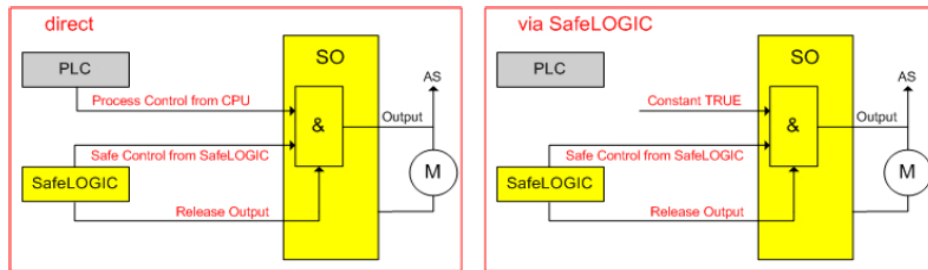


Figura 4.14 Principio di abilitazione per i moduli di output "Safe"

dall'applicazione standard che da quella di sicurezza, mentre nel secondo caso sarà solo il controllore di sicurezza a poter abilitare o meno l'uscita.

Per gestire il riavvio, ogni modulo di uscita ha il proprio blocco di riavvio interno e il proprio canale di riletture per valutare lo stato dell'uscita corrente. Sostanzialmente quindi, si deve eseguire una data sequenza di operazioni per riabilitare il canale ogni volta che si è verificato un errore nei moduli, nella rete e/o ogni volta che viene eseguita una funzione safety. Ovvero, si devono correggere gli errori presenti nei moduli o canali, riabilitare il segnale safety per quel canale, aspettare un tempo di ciclo di rete per essere sicuri della lettura e attendere il fronte di salita del canale. Esistono principalmente due diversi tipi di uscite per le varie applicazioni: l'uscita "Relay" e l'uscita "OSSD" (output switching sensor device). L'uscita di tipo Relay non prevede l'utilizzo di un master clock, quindi il modulo non ha l'opportunità del controllo degli errori e tale uscita permette al sistema di

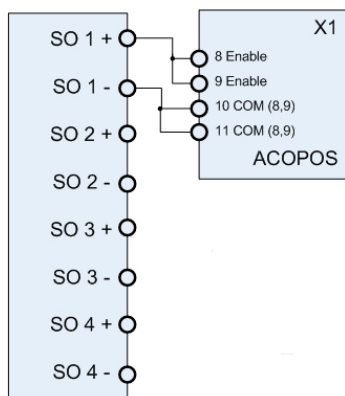


Figura 4.15 Connessione ACOPOS (SIL 2, CAT 3)

spegnersi in modo indipendente dagli altri moduli. L'uscita di tipo OSSD invece, offre la possibilità di controllare il valore di un'uscita spegnendo per un breve lasso di tempo un'uscita attiva. Alcuni esempi di collegamento tra attuatori e uscite di sicurezza sono quelli tra ACOPOS e ACOPOSmulti con i relativi moduli di output, i quali tramite una connessione diretta, si riesce a soddisfare i requisiti di Categoria 3 e SIL 2 per la connessione con i drive ACOPS e fino a Categoria 4 e SIL 3 con gli ACOPOSmulti. Tenendo

presente che come già detto, con la sigla SIL si intende Safety Integrity Level ed è una statistica dell'affidabilità della funzione di sicurezza, ovvero la riduzione del rischio fornita dalla contromisura. [23] [24] [25] [26]

4.3 Controllore del moto Safety

In questo progetto, per il controllo dei sette motori utilizzati per la movimentazione del robot e del nastro si è fatto uso di tre azionamenti ACOPOS P3. Quest'ultimi sono drive a tre assi con SafeMotion integrato e con interfacce per vari tipi di encoder tra cui l'EnData 2.2, che di fatto sarà poi quello utilizzato nel progetto. Per quanto riguarda quindi il controllo di sicurezza dei motori questo azionamento fornisce una serie di funzioni rilevanti per la sicurezza, secondo la norma IEC 61800-5-2. Tra queste funzioni notiamo: il Safe Torque Off (STO / STO1) che sarà poi la funzione utilizzata per lo stop di emergenza insieme al freno di stazionamento dei motori. Il Safe Stop (SS1 / SS2) il quale una volta attivo decelera gli assi in un tempo dato, non preoccupandosi della eventuale sincronia fra assi diversi e ne disabilita la coppia con la necessità di un freno di emergenza meccanico nel caso di pesi sospesi. Il Safe Operating Stop (SOS) che invece si occupa di fermare un dato asse in una finestra di posizioni possibili, il Safe Limited Speed (SLS) che una volta richiesto monitora fino a quattro velocità per assicurarsi che stiano entro i limiti e in caso contrario decelerare o accelerare l'asse. Il Safe Maximum Speed (SMS) che a differenza del SLS non può essere richiesto durante l'applicazione, ma solo in fase di configurazione e quindi se attivo monitora costantemente l'asse d'interesse. Il Safe Direction (SDI), che monitora il verso di rotazione del motore in caso sia permesso solo un verso, il Safe Limited Increment (SLI) il quale monitora la posizione degli assi per far sì che l'incremento di posizione si mantenga in un dato range. Il Safe Limited Position (SLP), che vien utilizzato per monitorare costantemente la posizione di un'asse, proprio come è stato fatto, ad esempio, in questo progetto per il controllo di certi assi che avrebbero potuto causare dei danni nel caso in cui si fossero superati certi limiti. Il Safe Limited Acceleration (SLA) per monitorare i limiti di accelerazione e decelerazione, il Safe Maximum Position (SMP) che analogamente a SMS può essere solo configurato e non richiesto

attivamente durante l'esecuzione e ha lo stesso scopo della funzione SLP. La Safe Brake Control (SBC), ovvero la funzione per il controllo del freno motore e così via. Solitamente il controllo effettivo di questo drive viene eseguito dall'applicazione standard, mentre le funzioni relative alla sicurezza vengono gestite dall'applicazione proveniente dal SafeDESIGNER e costituita da componenti aggiuntivi SafeMOTION, che monitorando i relativi limiti quali: posizione, velocità, accelerazione, coppia e così via, attivano immediatamente, se necessario, l'impulso disabilitante che provoca lo spegnimento del motore e l'attivazione dei freni di stazionamento del motore. Un aspetto comune a queste funzioni di sicurezza è senz'altro il modo in cui devono essere richieste, infatti, per attivare una di queste funzioni, l'ingresso del controller deve ricevere uno zero logico. Questa caratteristica rispetta il principio generale del fail-safe, ovvero, il principio per il quale si cerca, in caso di guasti, di portare il sistema verso la configurazione più sicura possibile, tale convenzione è anche conosciuta con il nome di "idle current principle". Di seguito descriviamo una tipica azione dopo una richiesta di una funzione di sicurezza, come ad esempio la funzione STO, che una volta richiesta, esegue l'impulso di sicurezza disabilitante, che agendo sul pattern di modulazione

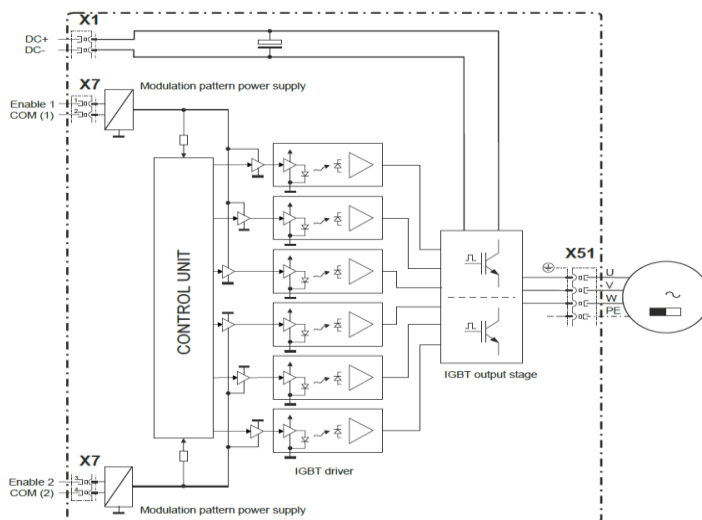


Figura 4.16 Diagramma a blocchi per il "safe pulse disabling"

del convertitore DC-DC che pilota i transistor IGBT, si interrompe il modello di modulazione generato, che deve essere trasferito agli azionamenti per generare effettivamente il campo rotante per il motore, il che fa cessare la produzione di coppia da parte del motore. Inoltre, se il motore si trovava in movimento o se erano presenti carichi sospesi prima dell'attivazione dell'impulso di sicurezza

dei driver IGBT nell'ACOPOS P3, interrompe il campo rotante sulle uscite IGBT e ciò di conseguenza interrompe anche l'alimentazione di potenza al motore. Studiando con maggiore attenzione

l'azionamento, vediamo che agendo sull'ingresso

disabilitante a seguito di tale richiesta i motori si troveranno senza coppia, quindi per ottenere il principio di fail-safe bisogna che vengano attivati i freni di stazionamento. Le funzioni principalmente utilizzate in questo progetto per i motori del robot e il nastro trasportatore sono, la funzione STO Safe Torque Off utilizzata per agire dopo la specifica richiesta, ovvero la pressione del pulsante di emergenza e la funzione SLA Safe Limited Acceleration, che monitora l'accelerazione al fine di mantenerla all'interno di un intervallo predefinito, ciascuna di queste funzioni è quindi ripetuta per ogni motore del sistema. [23] [26] [28]

4.4 Controllore “SafeLogic”

SafeLogic è un controllore di sicurezza con un sistema basato su un singolo task, in grado di raggiungere un tempo di ciclo inferiore a 1 ms. Questo controllore è anche responsabile del coordinamento della comunicazione relativa alla sicurezza di tutti i moduli nell'applicazione. Inoltre, SafeLogic controlla la configurazione di tutti i moduli, in particolare il tipo di modulo e la versione del firmware, che vengono riconosciuti automaticamente se sono compatibili con il sistema. Inoltre i parametri di tutti i moduli collegati sono controllati e in caso scaricati automaticamente, così che si garantisce senza sforzo la coerenza dei parametri anche in caso di sostituzione dell'hardware o per i nuovi dispositivi. Queste informazioni quindi per essere più accessibili in caso di manutenzione sono memorizzate in un supporto di memoria separato denominato SafeKey. SafeLogic, come operazione principale esegue in sicurezza l'applicazione progettata in SafeDESIGNER in grado di raggiungere fino ad un livello di integrità della funzione, SIL 3. Le principali differenze dagli altri controller sono: la possibilità di gestire fino a cento nodi safe e la possibilità di comunicare con altri controllori di

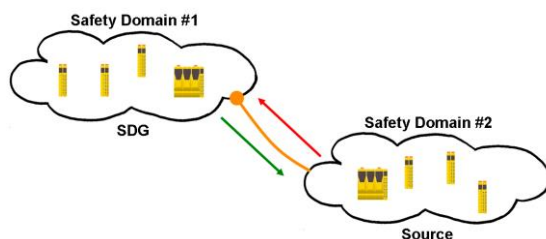


Figura 4.17 Comunicazione tra controllori “SafeLogic”

sicurezza al di fuori di uno stesso dominio safety. La comunicazione tra controllori safety avviene indirizzando il dominio di sicurezza con un SafeLogic ID univoco. Questa comunicazione può essere utilizzata

per funzionalità globali come un arresto di emergenza esteso a più di una macchina interconnessa o per diversi assi dipendenti. In questo modo, possiamo consentire una raccolta centralizzata di tutte le informazioni rilevanti per la sicurezza, così da poter controllare al meglio l'intera macchina. Per facilitare la comunicazione, il controllore SafeLogic fornisce un gateway, che gestisce l'inoltro dei dati da una rete all'altra, denominati Safety Domain Gateway (SDG) che consentono la comunicazione appunto con diversi domini. La comunicazione fra controllori di sicurezza può essere impostata sia nel controllori sorgente che in quello ricevitore, così da poter estendere la comunicazione in cascata e definire diversi possibili layout. Tale comunicazione è quindi impostata nel progetto SafeDESIGNER e viene eseguita mediante le appropriate variabili booleane e con i parametri del tempo di ciclo desiderati, al fine di definire il tempo di trasmissione. [23] [29]

4.5 SafeDESIGNER

SafeDESIGNER è il software dedicato alla programmazione delle componenti rilevanti per la sicurezza del progetto ed è completamente indipendente dall'applicazione standard. L'unico modo per poter comunicare tra un'applicazione e l'altra avviene tramite predefiniti canali booleani in cui si possono scambiare alcune variabili tra le due applicazioni, ad esempio una variabile non di sicurezza dall'applicazione standard può raggiungere l'applicazione Safe e viceversa. I linguaggi di programmazione utilizzabili in SafeDESIGNER sono il FBD (Function

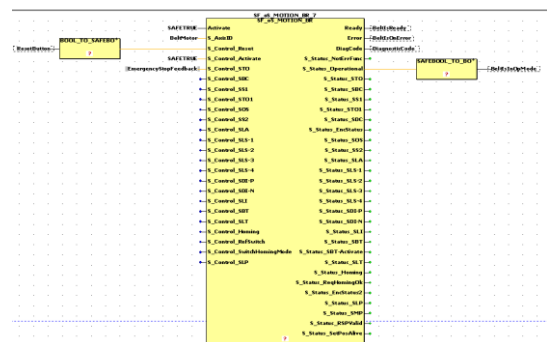


Figura 4.18 vista SafeDESIGNER

Block Diagram) e il Ladder, che possono essere liberamente utilizzati entrambi anche nello stesso codice. In figura 4.18 possiamo vedere ad esempio una vista del SafeDESIGNER in cui si vede utilizzare un blocco funzione utilizzando il linguaggio FBD, del quale vediamo infatti i blocchi grafici collegati a variabili o altri blocchi funzione. In questo progetto quindi sono state implementate alcune delle funzioni PLCopen di

sicurezza disponibili in questo ambiente. In particolare è stato implementato SF_EmergencyStop che connesso con il solito pulsante di sicurezza tramite un modulo di input Safe, esegue la fermata dell'intero sistema e con gli opportuni parametri aziona di conseguenza il freno di stazionamento. Freno che nel nostro caso risulta essenziale a causa della presenza del manipolatore che per sua natura è

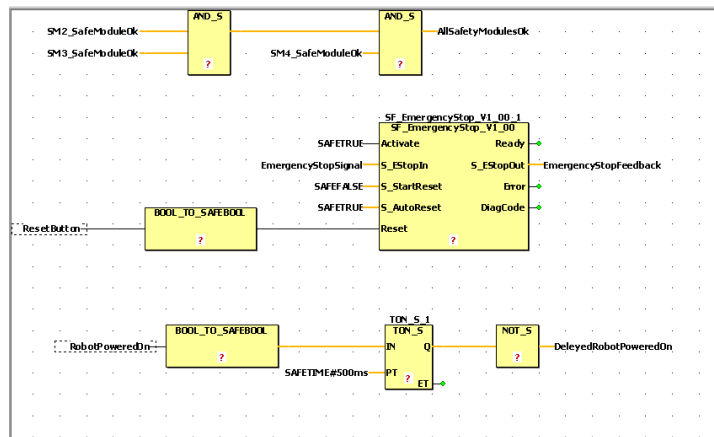


Figura 4.19 Gestione del segnale del blocco di emergenza in SafeDESIGNER

interessato dalla gravità e tenderà a collassare quando le coppie sul motore non saranno più presenti. Oltre a ciò, l'applicazione di sicurezza monitorerà ciascun asse in base alle informazioni riportate nella tabella informativa dei driver,

tra cui troveremo le variabili relative alla velocità, l'accelerazione, coppia massima positiva e negativa, il range di posizioni possibili dei motori, così come sono definite le rampe di decelerazione e più in generale tutte le contromisure da adottare una volta che uno dei limiti è stato violato.

Un'altra delle funzionalità più avanzate che possiamo sfruttare, fa uso di una libreria B&R chiamata safeRobotics, che per mezzo del modello cinematico è in grado di ricostruire in ogni momento la posizione esatta dell'end-effector del robot,

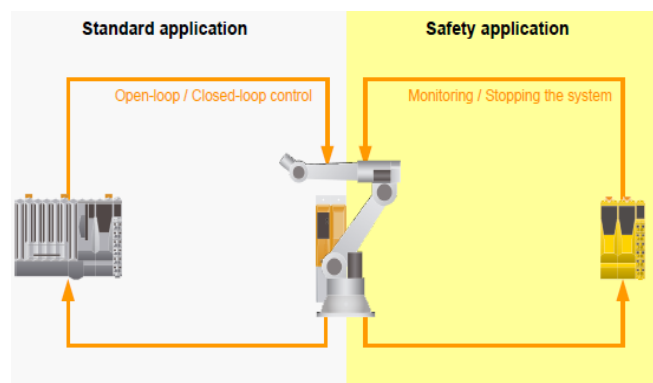


Figura 4.20 Applicazione standard vs. Applicazione "safety"

che quindi ne possiamo ad esempio consentire il movimento solo in posizioni predefinite. Questo tipo di funzionalità può essere molto utile in caso di inserimento del robot in una macchina automatica, per evitare ad esempio il contatto

indesiderato o la collisione con parti della meccanica circostante, o ancora in caso di cooperazione di più di un robot per fissare l'area di lavoro di ciascun robot al fine

di evitare l'impatto tra i due. Per implementare queste funzionalità abbiamo avuto bisogno di diversi blocchi funzione, ognuno dei quali si occuperà di un dato aspetto che poi verranno valutati insieme per effettuare la funzione volute. Sostanzialmente ci saranno quindi, blocchi funzione che si occuperanno di estrarre la posizione dei sei assi, altri responsabili della costruzione del modello cinematico, identificando prima i tipi di giunti presenti nel robot come lineare o rotazionale. Quindi,

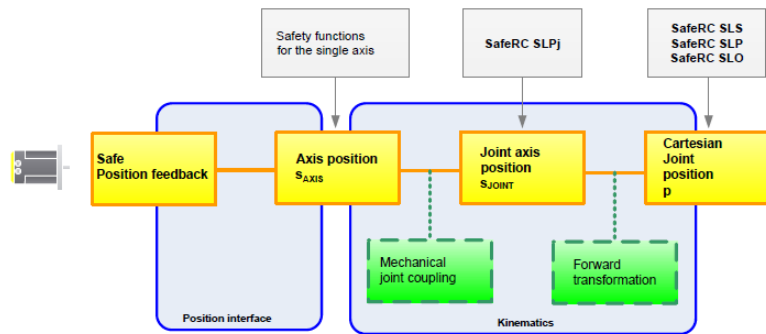


Figura 4.21 elaborazione di informazioni

per mezzo di una tabella vengono definite le specifiche geometriche del robot come la lunghezza dei link, la direzione di rotazione degli assi e i possibili legami di accoppiamento tra alcuni giunti. E infine sarà il blocco funzione appropriato che eseguirà la funzione SLP, che prendendo le informazioni del modello cinematico e dalla tabella che definisce lo spazio a disposizione del robot, fermandolo se i limiti nello spazio verranno violati. [30] [31]

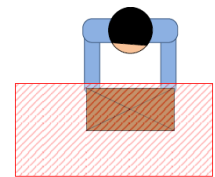


Figura 4.22 Esempio di utilizzo di safeRobotics

5. INTERFACCIA UOMO-MACCHINA IUM (HMI) E FUNZIONALITÀ

L'interfaccia uomo-macchina anche conosciuta nella pratica con la sigla HMI (Human-Machine Interface) costituisce il mezzo con cui un operatore può interagire con la macchina. Sostanzialmente quindi l'HMI rappresenta il software e l'hardware necessario per monitorare e controllare da remoto la macchina automatica. In questo progetto si è scelto quindi, continuando ad usare Automation Studio come ambiente di sviluppo, di implementare una semplice interfaccia così da poter controllare più facilmente la macchina anche in fase di test. Infatti, proprio per definizione di interfaccia uomo-macchina questa ci fornisce dei semplici comandi per una certa funzione della macchina, nascondendo le varie operazioni di più basso livello, che il software dovrà eseguire al fine di poter soddisfare la richiesta. Gli elementi essenziali quindi per una HMI devono comprendere sicuramente un flusso di informazioni e serie di possibili azioni di controllo al fine di poter comandare la funzionalità voluta. I concetti principali da rispettare secondo la letteratura per una buona interfaccia sono senz'altro l'accessibilità e l'usabilità. Quest'ultima in pratica definita anche dell'ISO (International Organization for Standardization) risulta essere una misura della capacità e facilità, quindi il grado di soddisfazione nel raggiungimento di un dato obiettivo, che in questo caso corrisponde all'utilizzo di una interfaccia uomo-macchina da parte di un operatore, le maggiori problematiche legate all'usabilità, si riscontrano quando il modello, quindi, l'idea della macchina e del suo funzionamento del progettista, non rispecchiano il modello della macchina secondo l'utilizzatore e quindi possiamo affermare che un'HMI, sarà tanto più usabile tanto più sono simili il modello della macchina secondo il progettista e l'utente finale. Le conseguenti caratteristiche di un sistema usabile risultano essere quindi ad esempio, l'efficacia nel raggiungere certi obiettivi, intendendo, l'accuratezza e la completezza con cui tali obiettivi sono raggiunti, fra queste caratteristiche troviamo anche l'efficienza, ovvero, le risorse spese per raggiungere una voluta accuratezza e completezza dell'obiettivo. La soddisfazione, quindi il comfort nell'utilizzo, la facilità di apprendimento, la facilità di memorizzazione e la sicurezza e robustezza all'errore che di fatto deve avere un impatto tanto minore quanto più alta è la probabilità di manifestarsi. Altro

concetto fondamentale nella creazione di un'interfaccia è rappresentato dall'accessibilità, che in questo contesto corrisponde alla versatilità dell'interfaccia, ovvero, la possibilità dell'interfaccia di essere facilmente utilizzata da diverse tipologie di utenti. In questo specifico contesto ad esempio, potrebbero essere i soggetti che si potrebbero occupare di prove sulla macchina, del suo collaudo e per finire dall'utente finale.

Nell'implementazione dell'interfaccia in questo progetto si è quindi cercato di rispecchiare per quanto possibile i concetti appena descritti e di fornire i mezzi necessari per il controllo della macchina automatica al fine di poter fornire sia le funzionalità di base della macchina, ovvero, l'evoluzione delle due differenti traiettorie e quindi il modo per passare da un tipo movimentazione all'altra e sia alcune funzionalità aggiuntive, come può essere la movimentazione in "Jog" della macchina, ovvero la movimentazione manuale, utile ad esempio nella creazione di nuove traiettoria o nel test di funzioni di sicurezza come possono essere ad esempio i limiti di posizione del robot, nonché di altre funzionalità ancora, introdotte a puro scopo didattico, come ad esempio la funzionalità implementata per l'esecuzione di nuove traiettorie che possono essere definite scegliendo punti qualsiasi nell'area di lavoro del robot durante la movimentazione manuale dello stesso. [32] [33]

5.1 Funzionalità di base

Di seguito dopo aver illustrato le caratteristiche di base di un'interfaccia uomo-macchina, andremo ad analizzare più nel dettaglio le varie funzioni di più basso livello che vengono svolte dal software una volta che viene richiesta una funzionalità dall'HMI. Sostanzialmente quindi andremo ad analizzare l'impatto che avrà il cambio di valore di una certa variabile dovuto a funzionalità richieste dalla macchina. Come possiamo vedere dalla figura 5.1, nella schermata di home dell'HMI, possiamo notare una sezione dedicata ai comandi di base che saranno le prime delle funzionalità che andremo ad analizzare. Andando in ordine quindi possiamo vedere che il primo pulsante di queste funzionalità è il tasto di "Power on" che ovviamente sarà dedicato all'accensione della macchina, ovvero, entrando nel dettaglio su quello che succede nella pratica, sostanzialmente una volta premuto

il pulsante “Power on”, una corrispettiva variabile booleana nel codice cambierà il suo valore, a questo punto il codice della macchina attenderà il via da parte di un'altra variabile che controlla lo stato della comunicazione powerlink della rete, una volta che si è raggiunto l'and delle due variabili, ovvero, entrambe sono vere il software può passare all'abilitazione delle strutture rappresentanti gli assi reali e virtuali della macchina, quindi in pratica delle sei strutture di tipo "gear_typ" per i

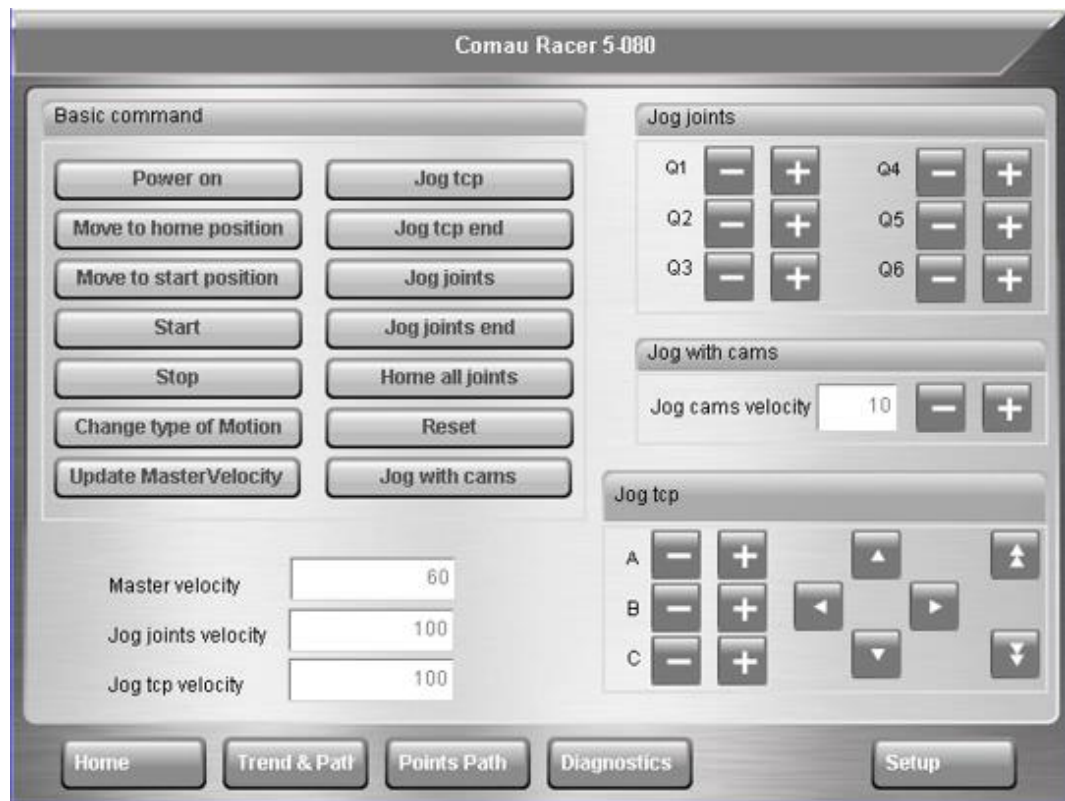


Figura 5.1 Schermata di home dell'HMI

sei assi reali del robot, una struttura di tipo “basic_typ” per l'asse master virtuale e i restanti sette assi, i sei virtuali del robot che rappresentano le coordinate spaziale e quello del nastro trasportatore di tipo “cam_typ”. A questo punto una volta abilitate le strutture degli assi, il software passa al controllo delle attuali posizioni andando a leggere se presenti le variabili che sono in carico della memorizzazione dell'ultima posizione dell'encoder prima dello spegnimento della macchina. Una volta effettuato questo controllo il prossimo step necessario per rendere operativa la macchina è la richiesta della seconda funzione dell'HMI, ovvero, “Move to home position”. Questa essenzialmente si occupa di portare il robot da qualsiasi configurazione abbia, per quando riguarda la posizione dei motori, alla posizione per cui ogni motore del robot si trova nella posizione con angolo “zero”. Il principale vantaggio di questo step, è la possibilità di poter verificare che

effettivamente la posizione zero del motore secondo il software corrisponda poi veramente con la posizione zero fisica, che si può verificare facilmente e immediatamente tramite un semplice strumento meccanico in dotazione con il robot. In queste prime movimentazioni in cui il robot si muove da un punto ad un altro, senza vincoli intermedi si è scelto per rendere il più fluido possibile il movimento, di impostare diverse velocità per ogni singolo asse del robot. Quello che si vuole ottenere sostanzialmente è che l'arrivo dell'end-effector nella posizione finale coincida con la fine del movimento di ogni singolo asse, per fare ciò banalmente si è pensato di fissare un tempo voluto in cui il robot dovrà effettuare il movimento e assegnare ad ogni asse la velocità risultante dal rapporto fra la distanza da percorrere e il tempo fissato, andando a saturare eventualmente le velocità per cui si superano i limiti dei motori. Il prossimo passo ora, è quello del "Move to start position", questa funzione sostanzialmente fa una operazione simile a quella di prima ma con uno scopo diverso, ovvero, una volta richiesta questa funzionalità viene impostata la posizione di target nella struttura di ogni singolo asse, che però questa volta non sarà quella di home, ma bensì, quella risultante della rototraslazione della posa da un sistema di riferimento solidale all'end-effector ad uno solidale alla base e quindi successivamente al risultato della cinematica inversa applicata su quella configurazione nello spazio conosciuto come punto di inizio di una delle due traiettorie. Anche in questo caso la regolazione delle velocità per ogni singolo asse è effettuata in modo da dare più fluidità possibile al movimento. Una volta arrivati a destinazione il robot si troverà nella posizione di inizio di una delle due traiettorie e aspetterà lo start dall'interfaccia per poter partire con l'evoluzione della traiettoria. Quindi una volta impostato il movimento da effettuare il software, controlla che effettivamente il robot sia arrivato in posizione, quindi va a configurare gli assi virtuali in modo tale da leggere la prossima destinazione dal valore corrispondente nella camma elettronica, quindi sostanzialmente avremo valori diversi al variare della posizione del master virtuale che ovviamente parte da zero ed ha un periodo di 360 unità. Quindi "agganciate" tutte le camme elettroniche, secondo il linguaggio di Automation Studio, degli assi virtuali e del nastro trasportatore al master virtuale, siamo pronti per la partenza della macchina, quindi premendo il tasto di "Start", il software in pratica fa partire in modalità movimento continuo con velocità costante il master che quindi fa generare ad ogni ciclo un set di posizioni x,y,z,a,b,c, che vengono elaborate, tramite, prima una matrice di

trasformazione omogenea per cambiarne il sistema di riferimento e quindi con la cinematica inversa al fine di ricavare le posizioni per ogni singolo asse che vengono poi passate con opportune funzioni con vincoli real-time già sopra descritti, negli assi reali del robot, che si occuperanno quindi di passare l'informazione al driver incaricato di azionare i motori. Una volta in questo stato, una delle funzionalità di base possibile, consiste nel poter cambiare la velocità dell'asse master che di conseguenza cambierà in automatico la velocità di tutta la macchina costituita da robot e nastro trasportare. Questo può essere effettuato cliccando sulla casella accanto "master velocity", digitando il nuovo valore nel pad numerico che compare e confermando schiacciando il tasto "Update master velocity". Quindi una volta richiesta verrà effettuata una modifica nella struttura di tipo "basic_typ" del master virtuale, ovvero, andrà a essere modificato il parametro velocità. Un'altra funzionalità possibile, una volta che la macchina è in movimento è quella che ci permette di cambiare il tipo di traiettoria, questa può essere richiesta dall'interfaccia con il tasto "change type of motion". L'effetto di quest'ultima richiesta nel software viene affrontata in maniera non immediata infatti, quello che succede effettivamente nel software è che, l'automa o macchina a stati che controlla il sistema, va in uno stato dove ci si aspetta l'and di tre variabili, ovvero, ci si aspetta che, tutti i motori del robot siano al meno di una tolleranza vicino alla posizione di inizio traiettoria, il nastro trasportatore anch'esso vicino alla posizione di inizio traiettoria, quindi, vicino la posizione zero e come controllo ridondante che l'end-effector del robot non sia al di sotto di una certa quota che rappresenta l'altezza per cui le cannucce dovrebbero stare all'interno dei flaconi. Quindi nell'istante in cui si verificano queste condizioni il software può effettivamente inviare un segnale di "Halt" a tutti gli assi del sistema, che provocheranno quindi uno "stop in fase", ovvero uno stop controllato in cui si è prefissata la decelerazione da effettuare così da non perdere la sincronia fra i motori. Fermato il sistema si passa a "sganciare" le camme dagli assi e successivamente "riagganciare" le camme relative alla nuova traiettoria da effettuare, dove con i termini "agganciare e sganciare", si intende in Automation Studio, come già detto, il passaggio secondo la macchina a stati dello standard PLCopen tra uno degli stati fra "StandStill", "Continuous Motion e Discrete Motion" verso lo stato "Synchronized motion" tramite la funzione MC_CamIn o MC_GearIn e viceversa. Quindi una volta "riagganciate" le camme della nuova traiettoria si ripetono le operazioni fatte in fase iniziale per lo start-up

e automaticamente una volta pronto, il robot eseguirà la nuova traiettoria. Per finire, tra le funzionalità di base troviamo, ovviamente, il pulsante che effettuerà uno stop della macchina non di emergenza quindi usando segnali come quelli usati in precedenza per il cambio della traiettoria, in questo modo si può fermare la macchina senza modificare la sincronia nei motori, funzionalità che può essere utile ad esempio in caso di test.

5.2 Funzionalità aggiuntive

Nel progetto dell'interfaccia si è pensato, oltre alle funzionalità di base come quelle appena descritte, di introdurre funzionalità che possiamo definire aggiuntive, in quanto non complementariamente indispensabili, ma comunque di interesse in certi casi, come può essere ad esempio, la verifica della correttezza di una funzione di sicurezza o la messa a punto della posizione dei motori persa per qualche motivo, ad esempio, a seguito di un riavvio improvviso senza la memorizzazione automatica dell'ultima posizione, che quindi di conseguenza resetterà la nuova posizione di zero e così via. In comune a tutti questi casi quindi potrebbe essere utile una movimentazione manuale del robot, così da poter ad esempio: muovere un asse fino a controllare quali siano i suoi limiti di posizione senza causare danni, come nel caso del quinto giunto in questo progetto, oppure ancora per referenziare un asse dopo un riavvio, oppure, per muovere l'end-effector fuori dall'area di lavoro per controllare se la funzione di sicurezza effettivamente agisce per bloccarlo, oppure ancora per controllare la precisione di una traiettoria e così via. Per tutti questi casi quindi si è pensato a due tipi diversi di movimentazione manuale, che possono essere utili nei vari casi di interesse. Questi tipi movimentazioni manuali vengono chiamate nella pratica, "Jog joint", la movimentazione manuale che si ottiene agendo su ogni asse e la movimentazione "Jog tcp", che si ottiene muovendo nello spazio il tcp (tool central point) il punto centrale dello strumento. Queste due funzionalità come vediamo dalla figura 5.1 possono essere richieste direttamente dall'interfaccia e avranno un effetto immediato nel sistema. Ovviamente, le due funzioni differiranno per le operazioni preliminari da effettuare affinché il robot possa essere comandato manualmente. Il primo che analizziamo è il "Jog tcp", come

già detto il “jog tcp” è caratterizzato da una movimentazione diretta dell’end-effector nello spazio, quindi, come ci si aspetta nell’architettura del movimento ci deve essere ancora la presenza attiva degli assi virtuali rappresentanti le coordinate spaziali. Infatti una volta richiesta la funzionalità, in pratica avviene che, si dà un comando di “Halt”, quindi stop in fase, a tutti gli assi che svolgono la funzione di camma elettronica, ovvero, i sei assi virtuali del robot, il master virtuale e l’asse del nastro trasportatore, quindi una volta fermi si alza e si abbassa volontariamente il segnale di stop agli stessi assi così da poter cambiare lo stato degli assi da “Synchronized Motion” a “StandStill”, come è possibile vedere da figura 3.8, dove con “note 4” si intende appunto, $MC_Stop.Done = TRUE$ e $MC_Stop.Execute = FALSE$. Quindi una volta che si sono “sganciate” sostanzialmente tutte le camme elettroniche dal master virtuale, possiamo andare comandare ogni singolo asse virtuale del robot, tramite i parametri “MoveJogPos” e “MoveJogNeg”, all’interno del campo “command” delle strutture di tipo “cam_typ”. Quindi automaticamente cambiando la posizione degli assi virtuali nel software si valutano le nuove coordinate spaziali, quindi tali coordinate vengono elaborate tramite la solita matrice di rototraslazione e la cinematica inversa così da fornire le nuove posizioni degli assi reali, le quali una volta inviati agli azionamenti fanno muovere effettivamente il manipolatore. Una volta terminata l’operazione di movimentazione in “jog tcp” per il robot, si può riconfigurare la macchina alla configurazione iniziale premendo il tasto “Jog tcp end”, in questo modo quindi chiedendo nuovamente le operazioni di “move to home e start position”, si riporta la macchina alla configurazione per cui si è pronti per la partenza dell’attività base, ovvero, l’esecuzione della traiettoria. Alternativamente, la seconda modalità di movimentazione manuale prevede, la possibilità di controllare manualmente ogni motore del robot, quindi, in questo caso sostanzialmente, una volta richiesta la funzionalità di “Jog joint”, il software, opera similmente al caso del “jog tcp” e quindi ferma e porta allo stato di “StandStill” tutti gli assi del sistema, infine quindi per eseguire il controllo dei motori, questo avviene agendo sui parametri “MoveJogPos” e il corrispettivo negativo delle strutture assi degli assi reali, ovvero, i sei assi dei sei giunti del manipolatore. Anche in questo caso, esiste poi una procedura per il recupero delle funzionalità del robot che passa per le richieste nell’interfaccia delle funzioni di “Jog joint end”, “Move to home” e “Move to start”, al termine del quale il robot è pronto a ripartire per l’esecuzione della traiettoria.

Infine, l'ultima della modalità di movimentazione manuale implementata può essere utile ad esempio nei primi test di una nuova traiettoria, infatti, grazie a questa funzionalità è possibile eseguire una determinata traiettoria, andando a muovere manualmente il master virtuale a cui sono “agganciate” le varie camme, anziché andare di velocità costante, come viene fatto in condizioni normali per l'esecuzione della traiettoria. Quindi le operazioni effettuate una volta richiesta la funzione chiamata di “Jog with cams”, consiste banalmente nel inviare un segnale di “Halt” al master virtuale e quindi comandare quest'ultimo asse tramite i soliti comandi di “MoveJogPos” e “MoveJogNeg” del campo “command” della struttura. In questo modo si può quindi provare, avanzando, millimetro per millimetro la nuova traiettoria al fine di controllarne la fattibilità e la precisione. Fra le opzioni selezionabili nell'HMI, ci resta da analizzare la funzione di “Home all joints” e la funzione di “reset”, quest'ultime due funzionalità come si può prevedere non saranno strettamente necessarie all'utente finale ma potranno servire al progettista in fase di “debug” a svolgere, funzioni ripetitive in maniera rapida, come può essere ad esempio l'operazione di dover “referenziare” (eseguire la procedura di homing) di ogni asse andando effettivamente a selezionare ciascun asse e forzare la variabile direttamente dal software, o similmente, fare degli acknowledgment ai vari errori che si presentano negli azionamenti per tutti gli assi presenti, che si può invece eseguire rapidamente con il tasto “reset” messo a disposizione nell'interfaccia.

5.3 Altre funzionalità e informazioni

Nel progetto di questa interfaccia, si è pensato di implementare per il robot, anche funzionalità diverse da quelle prettamente industriali, con lo scopo di acquisire più familiarità possibile con il controllo della macchina ed esplorare le varie applicazioni possibili utilizzando il manipolatore e le funzioni a disposizione. Fra le applicazioni di questo tipo, si è scelto di implementare una funzione chiamata nell'interfaccia “Start learning path”, sostanzialmente questa funzionalità permette al robot di memorizzare un numero prestabilito di punti e lasciare che il robot si muova liberamente da un punto all'altro della traiettoria senza vincoli e in maniera ciclica. Al fine di poter utilizzare questa funzionalità, quindi iniziare a memorizzare

dei punti nello spazio, si ha la necessità di dover avere la macchina in una della modalità di movimentazione manuale di tcp o joint, così che, si è liberi di muovere il robot nello spazio e si possono selezionare i punti di interesse della nuova traiettoria che vogliamo affrontare, quindi una volta selezionata la modalità



Figura 5.2 Schermata Trend & Path, HMI

manuale possiamo passare alla selezione dei punti. Per far ciò quindi, si passa alla seconda schermata dell'interfaccia premendo sul tasto in basso "Trend & Path" e una volta in questa schermata possiamo notare che è stato riportato per comodità i comandi di movimentazione manuale tcp e joint, ed è presente il tasto "Get position". Questo tasto una volta premuto memorizza le sei posizioni dei motori e si prepara per la memorizzazione del punto successivo. Nel riquadro sotto "get position" troviamo un visualizzatore contenente l'informazione sul numero di punti acquisiti. Quanto si sono raccolti abbastanza punti per la traiettoria, possiamo dare il via a questa movimentazione tramite il comando "Start learning path". A questo punto il software riprende i punti memorizzati e ciclo per ciclo si occupa di caricare i nuovi punti nell'azionamento, quindi una volta nota la distanza da coprire per ogni asse, si assegna la velocità ad ogni asse secondo il criterio già descritto per cui si garantisce fluidità al movimento. Una volta avviato il movimento, il software resta

in attesa che il robot arrivi nella posizione richiesta e una volta in quella configurazione si carica nuovamente il prossimo punto in cui passare e così in modo ciclico per tutti i punti, fino a che non si richiede lo stop di questa traiettoria con il comando “End learning path”, che fa resettare i punti e riporta il sistema nello stato di fine movimentazione manuale, nel quale a seguito delle procedure di calibrazione può ripartire con le funzioni di base.

Nel progetto di questa semplice interfaccia si è pensato di visualizzare anche delle informazioni che potessero essere utile nelle varie fasi del processo in corso, qui di seguito, ad esempio, possiamo vedere la terza schermata dell’interfaccia uomo-macchina. In questa schermata si è pensato di visualizzare i vari punti memorizzati durante la movimentazione manuale così da poterne valutare la fattibilità rispetto alla vicinanza dei limiti di posizione nei motori.

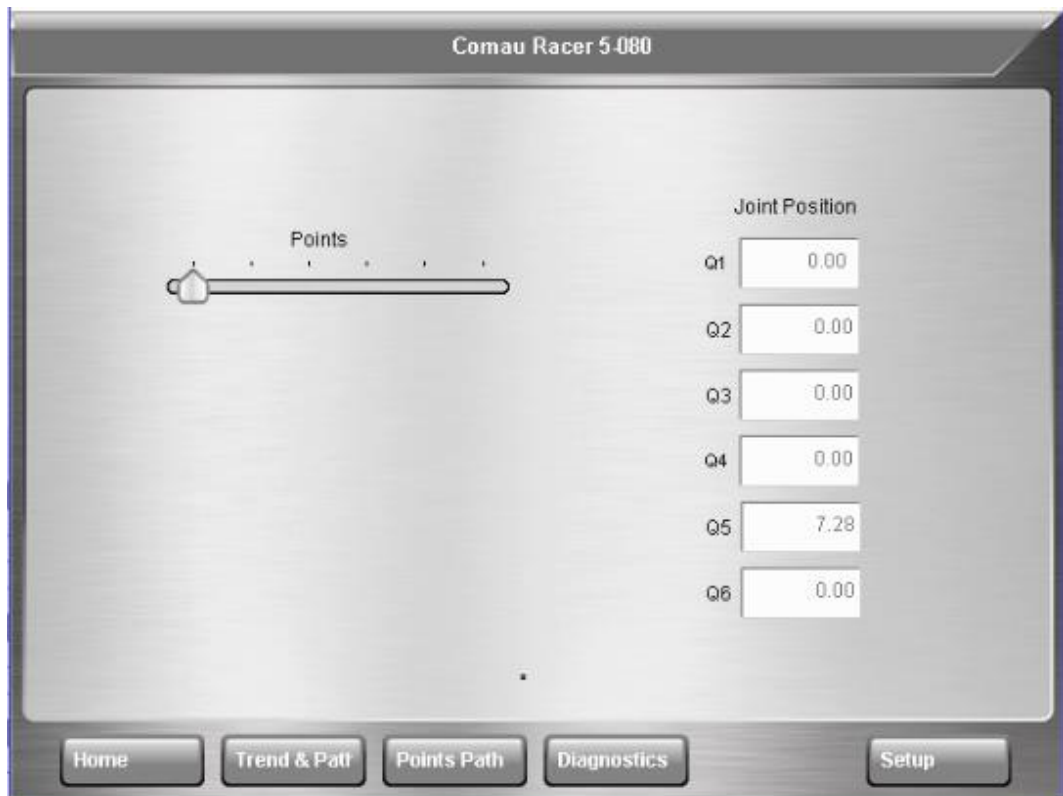


Figura 5.3 Schermata Points path, HMI

Altre informazioni acquisibili dall’interfaccia sono presenti nella seconda schermata ovvero quella dedicata a “Trend & Path”, visibile in figura 5.2, dove in alto troviamo l’informazione sulla massima velocità registrata nei motori di ogni singolo asse del robot, questa informazione può essere utile, ad esempio in fase di valutazione della produttività di una data traiettoria a velocità master diverse, così da tenere facilmente sotto controllo i limiti di velocità. Altre informazioni presenti

sono la produttività stimata e quella effettiva degli ultimi quattro minuti. Sostanzialmente per la produttività stimata si è fatto uso della velocità attuale del master, il quale valore nominale è di $140 \frac{\text{unità}}{\text{s}}$, quindi si è diviso per la lunghezza delle camme, che quindi in questo caso è di 120 *unità* e il loro rapporto ci dà il numero di traiettorie al secondo, quindi moltiplicato per 60 secondi ci darà il numero di traiettorie al minuto, che moltiplicato per il numero di flaconi riempiti contemporaneamente nel nostro caso sei, si ottiene la produttività al minuto che con questi valori risulta essere di 420 prodotti/minuto. Per la produttività effettiva durante il funzionamento invece, si è tenuto conto della distanza percorsa dal master, in pratica nel software, si è fatto in modo da, far patire un timer che scandisce sessanta secondi alla volta e contemporaneamente parte un integrale della distanza percorsa dal master, che una volta finito il minuto viene similmente a quanto fatto per quella stimata, diviso per la lunghezza delle camme e moltiplicato per il numero di cannucce dell'end-effector, quindi per il numero di flaconi riempiti contemporaneamente.

BIBLIOGRAFIA

- [1] Comau Robotics Product Range, Edition – 02/17 - Turin
- [2] www.br-automation.com/en-gb/products/
- [3] EPSG Draft Standard 301Ethernet POWERLINK Communication Profile Specification Version 1.2.0 © EPSG (Ethernet POWERLINK Standardisation Group) 2013
- [4] Automation Studio – B&R Help Explorer Version 4.1.2.34841
- [5] TM450 Motion Control Concept and Configuration1.1.0.1 ©2017/09/21, B&R
- [6] ACOPOS P3 User's manual, 1.10(November 2017), Model no.:MAACPP3-ENG
- [7] Technical InformationEnDat 2.2 – Bidirectional Interface for Position Encoders, HEIDENHAIN, Traunreut Germany 09/2017
- [8] Karl-Heinz John, Michael Tiegelkamp, IEC 61131-3: Programming Industrial Automation, Concepts and Programming Languages, Requirements for Programming Systems,Decision-Making Aids, Second Edition, Springer-Verlag Berlin Heidelberg 2001, 2 010
- [9] IEC 61131-3 International Standard, Programmable controllers – Part 3: Programming languages, Edition 3.0, IEC, Geneva, Switzerland, 2013
- [10] IEC 61131-3 Programmable controllers – Part 3: Programming languages, Second edition, IEC Geneva 20, Switzerland, 2003
- [11] Programmable Logic Controllers, A Practical Approach to IEC 61131-3 Using CODESYS, First edition, John Wiley & Sons, West Sussex, PO19 8SQ, United Kingdom, 2015
- [12] Practical Industrial Programming using IEC 61131-3 for PLC, IDC technologies Technology Training that Works
- [13] TM441 – Motion control: Electronic gears and cam profiles, V1.1.0.2 ©2017/09/26 by B&R,
- [14] L.Biagiotti, C. Melchiorri, Trajectory planning for automatic machines and robots, Springer-Verlag Berlin Heidelberg, 2008
- [15] M.Luise,G.M. Vitetta, Teoria dei segnali, McGraw-Hill,07/2012
- [16] Ethernet Facts <http://www.ethernet-powerlink.org> Automation Studio Hel

- [17] Claudio Melchiorri, Corso di robotica industriale”, Università degli studi di Bologna, LAR laboratorio di automazione e robotica, 1993-94
- [18] Sicurezza e sicurezza funzionale.Guida generale, Copyright 02/2011 ABB
- [19] <https://www.open-safety.org/index.php?id=5431&L=jkbocgztdssd>
- [20] B&R TM500 “Introduction to integrated safety tecnologia”
- [21] openSAFETY Safety Profile Specification EPSG Working Draft Proposal 304 V1.5.0 EPSG
- [22] Powerlink Fact, The Magazine for the Industrial Ethernet Standard, volume 5, Issue 1, April 2010
- [23] I sistemi di comando delle macchine secondo le norme EN ISO 3849-1 e EN ISO 13849-2, F.Pera, G. L. Amicucci, 2017 Inail, Milano, aprile 2017
- [24] Data sheet V1.120 X20(c)SIx1x0, Bernecker + Rainer B&R, Austria
- [25] TM500 - Introduction to Integrated Safety Technology, V1.3.0.6 ©2018/01/08 by B&R,
- [26] Data sheet V1.120 X20(c) SOx1x0, B&R, Austria
- [27] ACOPOS P3 user's manual V1.00, MAACPP3-ENG, Austria, February 2016
- [28] SafeMOTION User´s Manual V 4.4, MAACPMSAFEMC-ENG, 2017-11-13
- [29] Data sheet V1.120 X20(c) SL81xx, B&R,
- [30] TM510 - Working with SafeDESIGNER, V3.0.0.4 ©2017/05/22 by B&R,
- [31] TM550 – SafeROBOTICS, V1.3 ©2017/01/24 by B&R.
- [32] Marco Infussi, Chi: interfacce creative, underscored science ed.ltd
- [33] Jakob Nielsen, "*Usability Engineering*", Academic Press, 1993.