



UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea in Ingegneria Informatica

Elaborato d'esame

Tecnologie Informatiche Per L'Automazione Industriale

Progettazione e sviluppo di controllori PLC e PID

Anno Accademico 2019/2020

Professore

Prof. Adriano Mele

Studenti

Alberto Forlenza

Paolo Russo

Vittorio Ruggiero

Indice

Sommario.....	
Capitolo 1: PAV WAREHOUSE.....	
1.1 OpenPLC.....	7
1.1.1 Panoramica OpenPLC.....	7
1.1.2 Le variabili.....	9
1.1.3 Gli SFC.....	15
1.2 Modbus TCP/IP.....	26
1.3 Factory IO.....	27
1.4 ScadaBR.....	29
Capitolo 2: Il PID.....	
1.1 Introduzione e Modello Matematico.....	33
1.2 Progettazione del Pid.....	36
1.3 Digitalizzazione	39
Conclusioni.....	

Tabella delle figure

Illustrazione 1: una vista della PAV WAREHOUSE in FactoryIO.....	6
Illustrazione 2: panoramica di OPENPLC EDITOR.....	7
Illustrazione 3: struttura del progetto.....	8
Illustrazione 4: azioni create utilizzando linguaggio ST.....	8
Illustrazione 5: esempi di variabili.....	9
Illustrazione 6: file di configurazione.....	14
Illustrazione 7: SFC start_button.....	15
Illustrazione 8: SFC carrelli entrata.....	16
Illustrazione 9: carrello di scarico.....	17
Illustrazione 10: check spazio libero magazzino.....	17
Illustrazione 11: auto e manual mode.....	18
Illustrazione 12: ricerca primo scaffale libero.....	19
Illustrazione 13: fase finale dello store.....	20
Illustrazione 14: SFC azione retrieve.....	21
Illustrazione 15: azione check_retrieve_position in linguaggio ST.....	21
Illustrazione 16: SFC azione forward.....	22
Illustrazione 17: livelli d'altezza.....	23
Illustrazione 18: contatori pallet da smistare.....	23
Illustrazione 19: carrelli smistatore.....	24
Illustrazione 20: gestione carrelli smistatore.....	24
Illustrazione 21: piastra smistatrice.....	25
Illustrazione 22: ramo sinistro.....	25
Illustrazione 23: ramo centrale.....	25
Illustrazione 24: settaggio driver.....	27
Illustrazione 25: configurazione ip/port.....	28
Illustrazione 26: configurazione I/O points.....	28
Illustrazione 27: configurazione slave device.....	28
Illustrazione 28: ScadaBR come macchina virtuale	29

Illustrazione 29: creazione DataSource.....	29
Illustrazione 30: proprietà della DataSource.....	30
Illustrazione 31: creazione del Data Point.....	30
Illustrazione 32: Modbus Address Mapping.....	30
Illustrazione 33: i data points.....	30
Illustrazione 34: Vista della HMI creata in ScadaBR.....	31
Illustrazione 35: schema di un motore DC.....	33
Illustrazione 36: FDT del Motore.....	36
Illustrazione 37: Risposta del motore alla tensione nominale.....	37
Illustrazione 38: Risposta del Sistema a CC.....	38
Illustrazione 39: Schema Simulink.....	38
Illustrazione 40: sovralongazione > 20%.....	39
Illustrazione 41: Confronto uscite.....	40
Illustrazione 42: Schema Simulink.....	40

Sommario

Il nostro progetto è diviso in due parti. Nella prima parte abbiamo affrontato la progettazione e lo sviluppo di un controllore a logica programmabile, PLC, utilizzando gli strumenti teorici e pratici forniti durante il corso. Tale progettazione e sviluppo è stata resa possibile grazie all'utilizzo di strumenti software quali OPENPLC, FACTORY IO e ScadaBR che hanno costituito l'ambiente di sviluppo per il nostro SOFT PLC.

Risultato di tale progettazione è il PAV WAREHOUSE; un impianto automatico di stoccaggio pallet di cui troverete un'analisi approfondita nelle pagine seguenti.

La seconda parte del progetto ci ha visto alle prese con la progettazione di un controllore PI con lo scopo di controllare la velocità di un motore a corrente continua.

Partendo dall'analisi delle equazioni fisiche descrittive del sistema, siamo giunti ad un suo modello matematico ed attraverso l'ausilio del software Matlab abbiamo sintetizzato il controllore.

Dopo aver eseguito le varie simulazioni anche tramite l'ausilio di Simulink abbiamo inoltre fornito una versione digitalizzata di tale controllore.

Capitolo 1: PAV WAREHOUSE

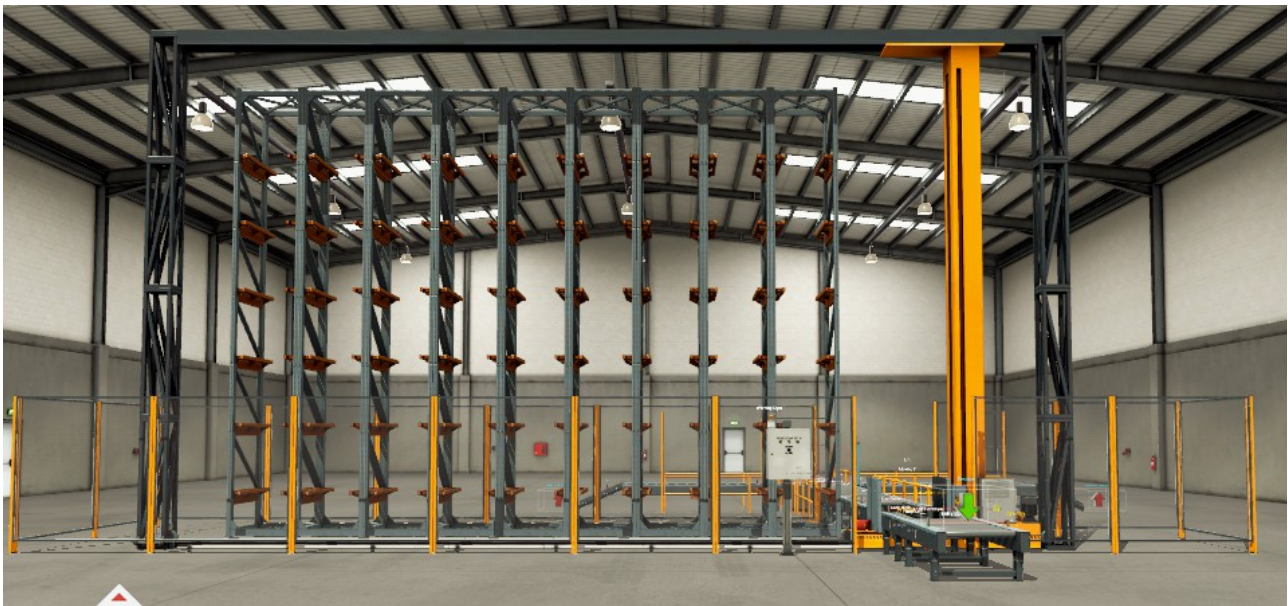


Illustrazione 1: una vista della PAV WAREHOUSE in FactoryIO

Il PAV warehouse è il magazzino per il quale abbiamo progettato ed implementato un controllore PLC che ha permesso di automatizzare le operazioni dell'impianto.

In particolare il nostro magazzino supporta le tipiche operazioni di una warehouse:

1. Storage dei Pallet: i pallet possono essere posizionati nel magazzino secondo una modalità automatica o manuale. Nel primo caso i pallet saranno memorizzati con una politica sequenziale che permette l'ubicazione del pallet nel primo scaffale libero. Invece, con la modalità manuale l'operatore può selezionare l'ubicazione all'interno della scaffalatura.
2. Retrieve dei Pallet: l'operatore può prelevare un pallet dalla scaffalatura indicandone la relativa posizione. Esso verrà poi sottoposto alla successiva operazione di forward.
3. Smistamento dei Pallet: con questa operazione i pallet vengono inoltrati su un carrello centrale ed in base ad un criterio SORT BY HEIGHT verranno smistati in una apposita direzione (Left, Right, Center) .

1.1 OpenPLC

1.1.1 Panoramica OpenPLC

OpenPLC è un progetto open-source ideato e sviluppato da Thiago Alves, scaricabile dal sito ufficiale <https://www.openplcproject.com/>.

Grazie all'utilizzo di diversi software, quali un editor (OpenPLC Editor) ed un ambiente d'esecuzione fornito da OpenPLC Runtime, esso permette di progettare e testare PLC sia in software che in hardware. La facilità d'utilizzo del pacchetto di software e l'essere open-source conferiscono ad OpenPLC la caratteristica di rappresentare una soluzione low cost nell'ambito dell'automazione e della ricerca e per tale motivo esso viene usato non solo nei settori dell'industria automatizzata ma anche per piccoli progetti homemade.

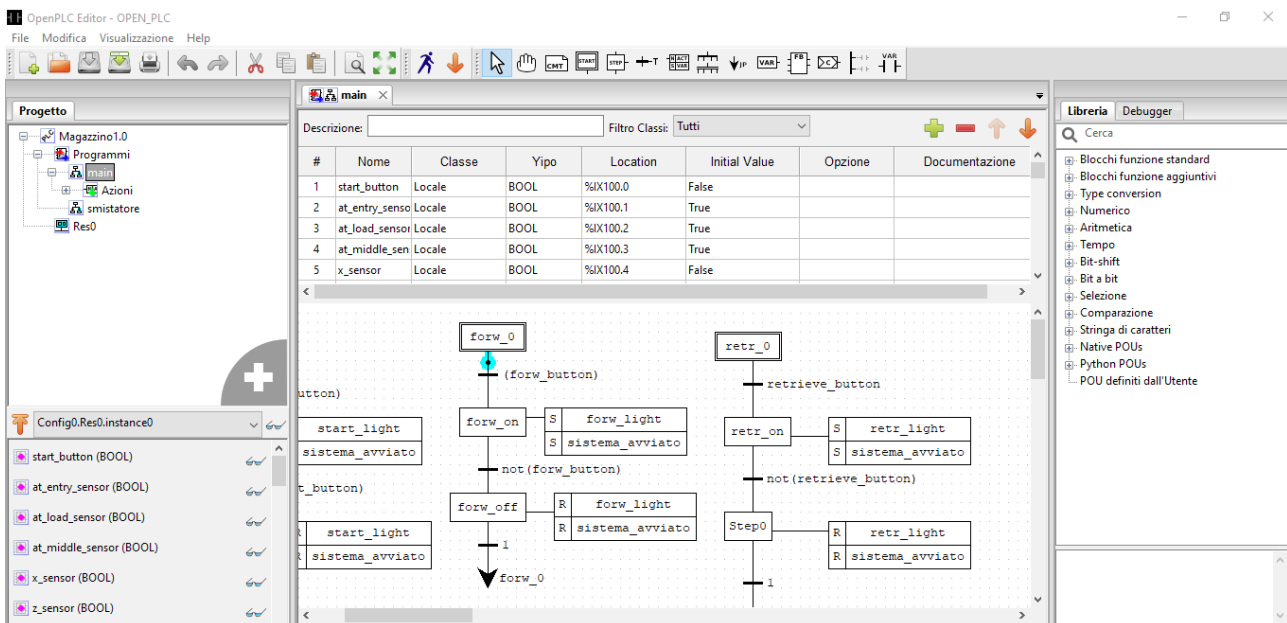


Illustrazione 2: panoramica di OPENPLC EDITOR

Attraverso OpenPLC Editor è possibile creare delle POU ,quali programmi, funzioni o blocchi funzionali utilizzando i principali linguaggi per la programmazione dei PLC.

In particolar modo esso consente di utilizzare:

1. Instruction List language
2. Structured Text language
3. Ladder Diagram language
4. Function Block Diagram language
5. Sequential Function Chart language

Nel nostro progetto abbiamo deciso di creare due POU “programma”: main e smistatore.

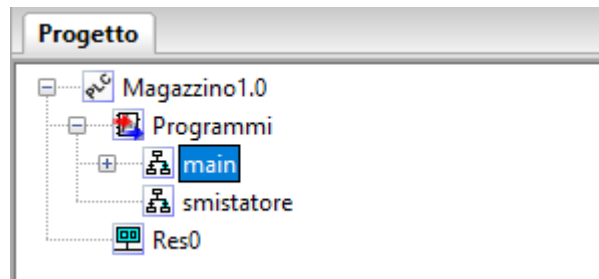


Illustrazione 3: struttura del progetto

Il **main** contiene diversi SFC che gestiscono i vari automatismi della parte anteriore del magazzino; in particolare abbiamo SFC per il controllo dei carrelli trasportatori, SFC per il controllo di bottoni e luci ed infine SFC per le azioni di store, forward e retrieve dei pallet.

Lo **smistatore** contiene invece gli SFC che permettono , in base all'altezza dei singoli pallet, di smistarli su degli opportuni carrelli.

Inoltre volendo utilizzare diversi linguaggi abbiamo deciso di implementare anche alcune funzionalità del nostro magazzino creando delle azioni in linguaggio ST.

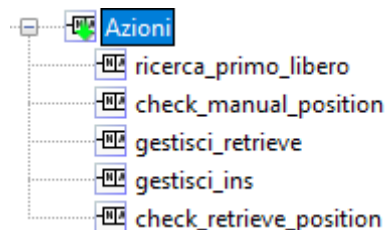


Illustrazione 4: azioni create utilizzando linguaggio ST

1.1.2 Le variabili

OpenPLC permette di creare diverse variabili, assegnando loro:

1. **un nome**
2. **una classe:** rappresenta lo scope della variabile e possiamo avere variabili locali, external, input, output, inout ed temp. Nel nostro progetto sono state utilizzate maggiormente variabili locali ed esterne.
3. **un tipo:** bool, int, real, time etc.
4. **una location:** costituisce l'indirizzo di memoria utilizzato per il PLC addressing. Gli indirizzi PLC sono costruiti come un unione di quattro parti ordinate:
 - il simbolo identificativo %
 - una classe di memorizzazione I (inputs), Q (outputs), M (memory)
 - data size indication (X, B, W, D, L)
 - un indirizzo gerarchico
5. **un valore iniziale**
6. **opzioni e documentazione**

#	Nome	Classe	Yipo	Location	Initial Value
20	errore_0	Locale	BOOL	%QX100.7	
21	carrello_uscita	Locale	BOOL	%QX101.0	False
22	carrello_scarico	Locale	BOOL	%QX101.1	False
23	retr_light	Locale	BOOL	%QX101.2	False
24	position	Locale	INT	%QW100	0

Illustrazione 5: esempi di variabili

Maggiori informazioni sul PLC addressing possono essere trovate consultando la documentazione ufficiale di OPENPLC direttamente dal sito: <https://www.openplcproject.com/reference/plc-addressing/>.

Le variabili utilizzate nel nostro progetto sono riportate nelle successive figure.

#	Nome	Classe	Tipo	Location	Val. iniziale	Descrizione
1	start_button	LOCALE	BOOL	%IX100.0	FALSE	determina l'avvio del processo di storage
2	at_entry_sensor	LOCALE	BOOL	%IX100.1	TRUE	determina la presenza del pallet sull'estremità iniziale del carrello di entrata
3	at_load_sensor	LOCALE	BOOL	%IX100.2	TRUE	determina la presenza del pallet sull'estremità finale del carrello di entrata
4	at_middle_sens	LOCALE	BOOL	%IX100.3	TRUE	determina la presenza del pallet sul braccio meccanico
5	x_sensor	LOCALE	BOOL	%IX100.4	FALSE	coordina il movimento sull'asse X (orizzontale) del braccio meccanico parallelamente al magazzino
6	z_sensor	LOCALE	BOOL	%IX100.5	FALSE	coordina il movimento sull'asse Z (verticale) del braccio meccanico parallelamente al magazzino
7	manual	LOCALE	BOOL	%IX100.6	FALSE	definisce la modalità manuale di storage all'interno del magazzino
8	auto	LOCALE	BOOL	%IX100.7	FALSE	definisce la modalità automatica di storage all'interno del magazzino
9	forw_button	LOCALE	BOOL	%IX101.0	FALSE	determina l'avvio del processo di forward
10	at_unload_sens	GLOBALE	BOOL	%IX101.1	FALSE	determina l'avvio del processo di smistamento
11	retrieve_button	LOCALE	BOOL	%IX101.3	FALSE	determina l'avvio del processo di retrieve
12	turn_entry	LOCALE	BOOL	%IX101.5	FALSE	determina l'avviamento di loaded_turn ed il reset degli attuatori enterconveyor e enterconveyor1 . Inoltre, ogni volta che si attiva decrementa la variabile cont
13	at_front	LOCALE	BOOL	%IX101.6	FALSE	permette di fermare il pallet sullo smistatore,pronto per essere processato

#	Nome	Classe	Tipo	Location	Val. iniziale	Descrizione
14	pallet_sensor	LOCALE	BOOL	%IX101.7	FALSE	fa parte di una rete di sensori per determinare l'altezza di un generico pacco
15	low	LOCALE	BOOL	%IX102.0	FALSE	fa parte di una rete di sensori per determinare l'altezza di un generico pacco
16	medium	LOCALE	BOOL	%IX102.1	FALSE	fa parte di una rete di sensori per determinare l'altezza di un generico pacco
17	hight	LOCALE	BOOL	%IX102.2	FALSE	fa parte di una rete di sensori per determinare l'altezza di un generico pacco
18	at_right	LOCALE	BOOL	%IX102.3	FALSE	determina il reset degli attuatori loaded_turn_right e turn e della variabile right_ok
19	at_right_exit	LOCALE	BOOL	%IX102.4	FALSE	determina il reset dell'attuatore rightconveyor
20	at_left	LOCALE	BOOL	%IX102.5	FALSE	determina il reset degli attuatori loaded_turn e turn e della variabile left_ok
21	at_left_exit	LOCALE	BOOL	%IX102.6	FALSE	determina il reset dell' attuatore leftconveyor
22	at_fwd	LOCALE	BOOL	%IX102.7	FALSE	determina il reset dell' attuatore loaded_turn
23	at_fwd_exit	LOCALE	BOOL	%IX103.0	FALSE	determina il reset dell' attuatore frconveyor
24	carrello_entrata	LOCALE	BOOL	%QX100.0	FALSE	determina lo stato del primo carrello di entrata
25	carrello_carico	LOCALE	BOOL	%QX100.1	FALSE	determina lo stato del secondo carrello di entrata
26	start_light	LOCALE	BOOL	%QX100.2	FALSE	determina lo stato della luce rispettiva del pulsante di storage
27	carica_pallet	LOCALE	BOOL	%QX100.3	FALSE	determina lo stato dell'attuatore incaricato di prelevare il pallet
28	lift	LOCALE	BOOL	%QX100.4	FALSE	determina lo stato di elevazione del braccio meccanico

#	Nome	Classe	Tipo	Location	Val. iniziale	Descrizione
29	scarica_pallet	LOCALE	BOOL	%QX100.5	FALSE	determina lo stato dell'attuatore incaricato di scaricare il pallet
30	forw_light	LOCALE	BOOL	%QX100.6	FALSE	determina lo stato della luce rispettiva del pulsante di forward
31	errore_0	LOCALE	BOOL	%QX100.7	FALSE	determina lo stato di allarme del sistema
32	carrello_uscita	LOCALE	BOOL	%QX101.0	FALSE	determina lo stato del primo carrello di uscita,posto dopo il braccio meccanico
33	retr_light	LOCALE	BOOL	%QX101.2	FALSE	Determina lo stato della luce rispettiva del pulsante di retrieve
34	enterconveyor	LOCALE	BOOL	%QX101.3	FALSE	determina lo stato del secondo carrello di uscita posto dopo carrello_uscita
35	enterconveyor1	LOCALE	BOOL	%QX101.4	FALSE	determina lo stato del terzo carrello di uscita,posto dopo enterconveyor
36	loaded_turn	LOCALE	BOOL	%QX101.5	FALSE	rappresenta lo stato dell'attuatore presente all'interno dello smistatore per caricare e smistare a sinistra ed al centro i vari pacchi
37	rightconveyor	LOCALE	BOOL	%QX101.6	FALSE	determina lo stato dell'attuatore associato al carrello di destra
38	turn	LOCALE	BOOL	%QX101.7	FALSE	determina lo stato dell'attuatore attraverso il quale è possibile ruotare lo smistatore per smistare i vari pacchi
39	loaded_turn_right	LOCALE	BOOL	%QX102.0	FALSE	rappresenta lo stato dell'attuatore presente all'interno dello smistatore per smistare destra i vari pacchi
40	leftconveyor	LOCALE	BOOL	%QX102.1	FALSE	determina lo stato dell'attuatore associato al carrello di sinistra
41	frconveyor	LOCALE	BOOL	%QX102.2	FALSE	determina lo stato dell'attuatore associato al carrello centrale

#	Nome	Classe	Tipo	Location	Val. iniziale	Descrizione
42	manual_positio n	LOCALE	BOOL	%QW0	FALSE	determina la posizione manuale definita dall' operatore per effettuare l'operazione di storage o retrieve
43	counter	LOCALE	INT		0	determina la prima posizione libera all'interno del magazzino
44	sync0	LOCALE	BOOL		FALSE	variabile di sincronizzazione
45	sync1	LOCALE	BOOL		FALSE	variabile di sincronizzazione
46	sync2	LOCALE	BOOL		FALSE	variabile di sincronizzazione
47	sync3	LOCALE	BOOL		FALSE	variabile di sincronizzazione
48	sync4	LOCALE	BOOL		FALSE	variabile di sincronizzazione
49	magazzino	LOCALE	ARRA Y[1..5 4]		0	array di tipo strutturato che determina lo stato delle varie celle di immagazzinamento.
50	i	LOCALE	INT		0	usata come indice per la ricerca del primo elemento libero
51	trovato	LOCALE	BOOL		FALSE	variabile di ricerca
52	pos	LOCALE	INT		0	variabile di ricerca
53	max_dim	LOCALE	INT		54	dimensione massima di storage del magazzino
54	cont_mag	LOCALE	INT		0	numero di elementi presenti nel magazzino
55	ok_retrieve	LOCALE	BOOL		FALSE	variabile per la gestione del retrieve
56	ok_ins	LOCALE	BOOL		TRUE	variabile per la gestione dell'inserimento di un pallet
57	ok_ret_pos	LOCALE	BOOL		FALSE	variabile di controllo sulla posizione
58	sistema_avviato	LOCALE	BOOL		FALSE	variabile di stato del sistema
59	store_avviato	LOCALE	BOOL		FALSE	variabile di stato del sistema
60	retrieve_avviato	LOCALE	BOOL		FALSE	variabile di stato del sistema
61	forward_avviato	LOCALE	BOOL		FALSE	variabile di stato del sistema
62	wait	LOCALE	BOOL		FALSE	variabile di sincronizzazione
63	wait1	LOCALE	BOOL		FALSE	variabile di sincronizzazione
64	wait2	LOCALE	BOOL		FALSE	variabile di sincronizzazione

#	Nome	Classe	Tipo	Location	Val. iniziale	Descrizione
65	wait3	LOCALE	BOOL		FALSE	variabile di sincronizzazione
66	left_ok	LOCALE	BOOL		FALSE	variabile di direzione
67	right_ok	LOCALE	BOOL		FALSE	variabile di direzione
68	fwd_ok	LOCALE	BOOL		FALSE	variabile di direzione
69	cont	LOCALE	INT		0	variabile di gestione per l'azione di forward

La definizione delle variabili globali in OPENPLC è effettuata in un particolare file, detto file di configurazione.

Oltre ad essa, in tale file possiamo creare anche diversi task ognuno dei quali può eseguire delle istanze dei programmi che abbiamo sviluppato. Nel nostro caso abbiamo un unico task denominato task0 il quale in maniera ciclica, ad intervalli di 20 ms, va ad eseguire le istanze dei programmi main e smistatore.

#	Nome	Classe	Yipo	Location	Initial Value	Opzione	Documentazione
1	at_unload_sen	Global	BOOL	%IX101.1	True		

Tasks:

Nome	Triggering	Singolo	Interval	Priorità
task0	Ciclico		T#20ms	0

Instances:

Nome	Yipo	Task
instance0	main	task0
instance1	smistatore	task0

Illustrazione 6: file di configurazione

1.1.3 Gli SFC

Per la **gestione dei pulsanti** dell'impianto abbiamo implementato diversi SFC ognuno dei quali soddisfa il medesimo pattern. Per tale motivo riportiamo in figura solamente il diagramma del pulsante start.

Esso costituisce un SFC molto semplice, in quanto alla pressione del pulsante viene attivata la fase di start_on nella quale viene effettuato il set della luce del pulsante ed il set dello stato del sistema. Qualora il pulsante venga ripremuto verrà eseguita una transizione che porterà il pulsante nello stato di start_off in cui vengono resettati i valori precedentemente settati.

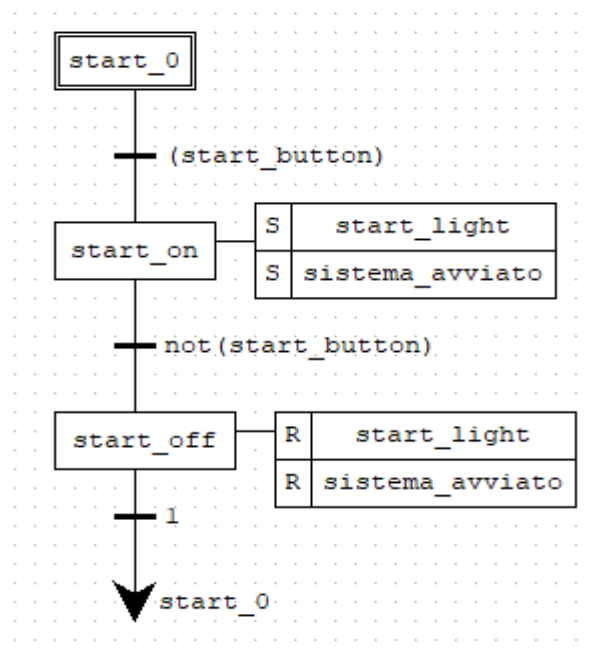


Illustrazione 7: SFC start_button

Le operazioni di store e forward coinvolgono i **carrelli d'entrata**, associati alle variabili `carrello_carico` e `carrello_entrata`. Essi vengono attivati qualora il programma rilevi la presenza di un pallet da immagazzinare o da inoltrare sulla linea e vengono resettati una volta che il pallet è stato caricato dal braccio meccanico. Inoltre un'ulteriore condizione per far muovere i carrelli è che il pallet precedente sia stato caricato (`not(at_load_sensor) = TRUE`).

Infine vogliamo rendere noto che poiché OPENPLC non permette di accedere e gestire i marker di fase (`nome_fase.X` e `nome_fase.T`) abbiamo temporizzato le fasi attraverso l'utilizzo di una variabile di sincronizzazione la quale è settata TRUE dopo un certo tempo.

Tale operazione è resa possibile dall'utilizzo del qualificatore **DELAYED** che ci consente di effettuare una transizione di fase dopo t secondi.

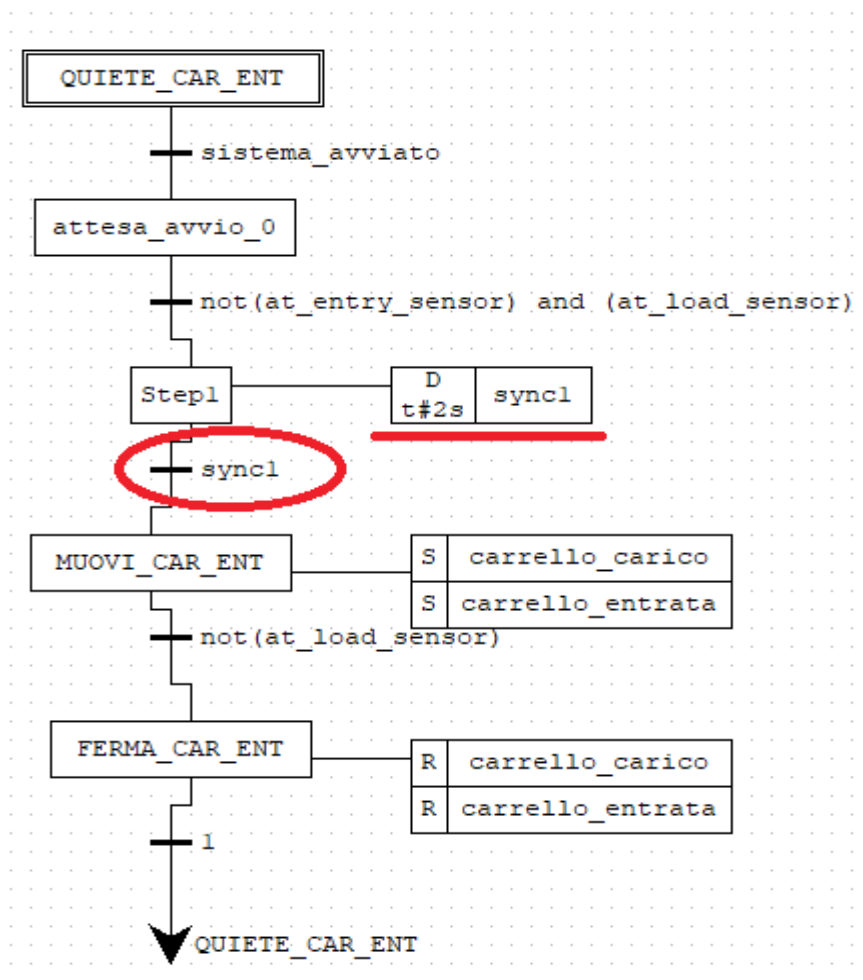


Illustrazione 8: SFC carrelli entrata

Le operazioni di forward e retrieve coinvolgono vari carrelli d'uscita. Fra questi, in particolare, troviamo il carrello che abbiamo denominato **carrello_scarico**. Esso rappresenta la congiunzione fra il programma main e quello smistatore in quanto è incaricato di inoltrare i pacchi verso gli attuatori utilizzati, appunto, dallo smistatore. Rilevata dal sensore **at_unload_sensor** la presenza di un pallet, verrà settata la variabile legata al carrello ed il conseguente avvio di questo.

Una volta che il pallet supererà il sensore, il carrello si fermerà.

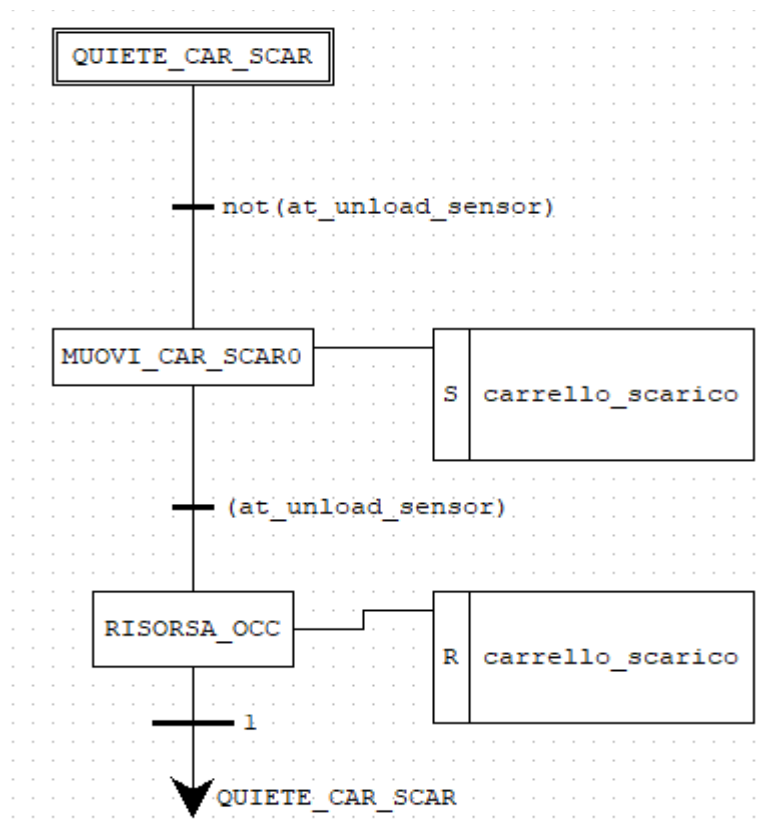


Illustrazione 9: carrello di scarico

L'operazione di **storage** consentirà all'operatore di immagazzinare il pallet, posto all'ingresso, nella prima posizione libera o in una posizione da lui selezionata.

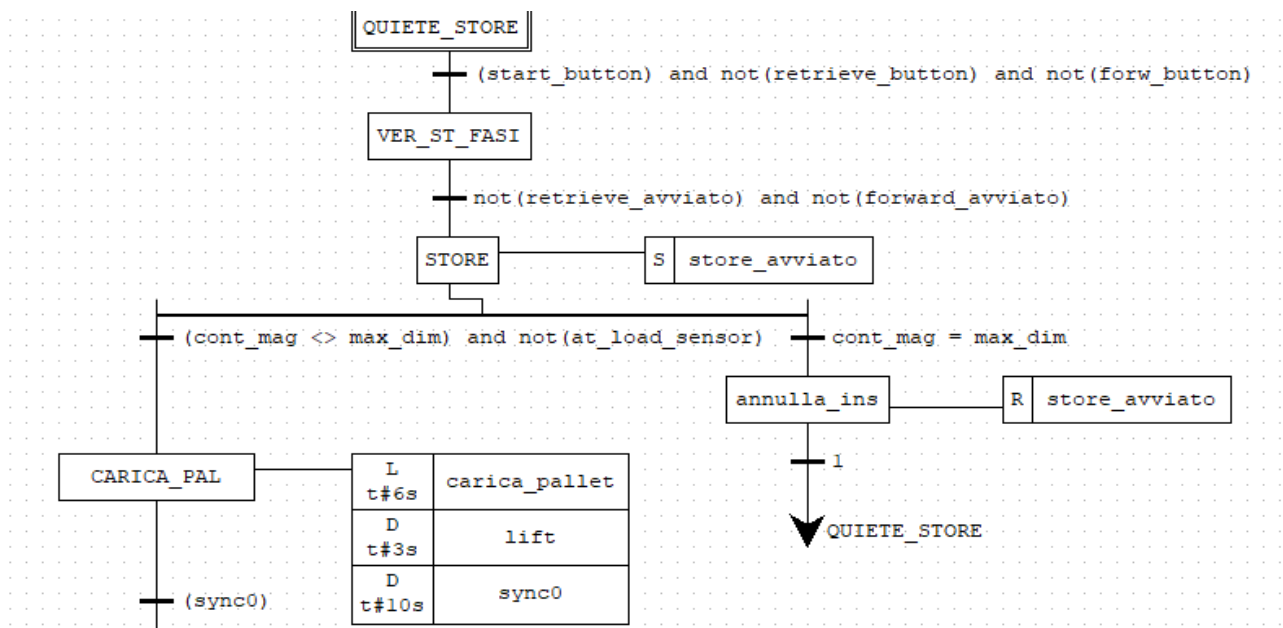


Illustrazione 10: check spazio libero magazzino

Dopo aver gestito la mutua esclusione tra i pulsanti di store, forward e retrieve, la fase di STORE viene avviata.

In OPENPLC abbiamo gestito il nostro magazzino attraverso diverse variabili:

1. magazzino, ossia un **array** da 54 elementi
2. max_dim, dimensione massima del magazzino rappresentata con un **intero** di valore 54
3. cont_mag, tiene traccia del numero di elementi memorizzati nel magazzino

Prima di procedere al caricamento del pallet sul braccio meccanico, il programma verificherà la disponibilità dello spazio all'interno del magazzino (cont_mag < > max_dim).

Qualora vi sia spazio disponibile, il pallet verrà caricato sul braccio; altrimenti verrà eseguito un salto a store_quiete, con conseguente reset della fase di store, per dare la possibilità all'operatore di eseguire operazioni di forward o retrieve.

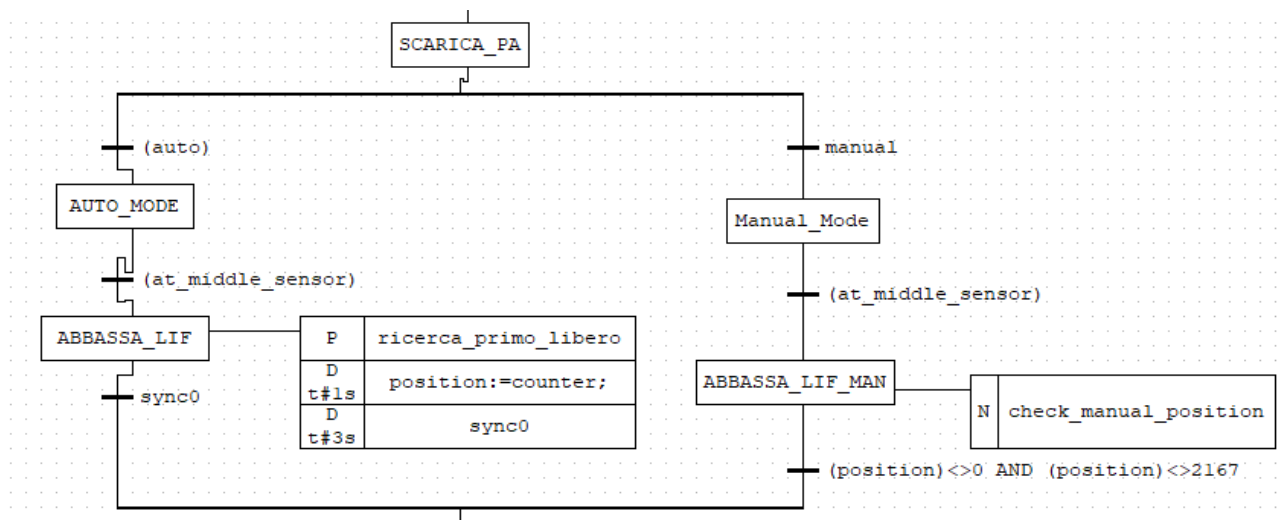


Illustrazione 11: auto e manual mode

Per quanto riguarda l'inserimento automatico, una volta che il pallet viene caricato sul braccio (at_middle_sensor = FALSE), la fase di ABBASSA_LIF permette di ricercare il primo elemento libero attraverso l'azione con qualificatore P, ricerca primo libero.

Tale azione è stata implementata utilizzando il linguaggio STRUCTURED TEXT come riportato nella illustrazione successiva.

```

i := 1;
trovato := FALSE;
pos := -1;

while (not(trovato) AND (i<=max_dim)) do
    if (magazzino[i]=False) then
        pos := i;
        trovato := TRUE;
    else
        i := i+1;
    end_if;
end_while;

magazzino[pos]:=True;
counter := pos;

```

Illustrazione 12: ricerca primo scaffale libero

Individuata la posizione, essa verrà memorizzata in counter e successivamente assegnata a position. Ciò produrrà lo spostamento del braccio meccanico nella posizione individuata.

Nel caso di inserimento manuale viene eseguita la fase ABBASSA_LIF_MAN in cui viene eseguita l'azione **check_manual_position**.

```

if( (manual_position<1) OR (manual_position>max_dim) ) then
    position:=0;
    errore_0:=1;

else
    if ( (magazzino[manual_position]=TRUE)) then
        position:=0;
        errore_0:=1;
    else
        position:=manual_position;
        magazzino[manual_position]:=TRUE;
        errore_0:=0;
    end_if;
end_if;

```

Tale azione ha come obiettivo quello di verificare che la posizione inserita dall'operatore sia valida:

1. $\text{manual_position} \in [1, \text{max_dim}]$
2. $\text{magazzino}[\text{manual_position}] \neq \text{TRUE}$

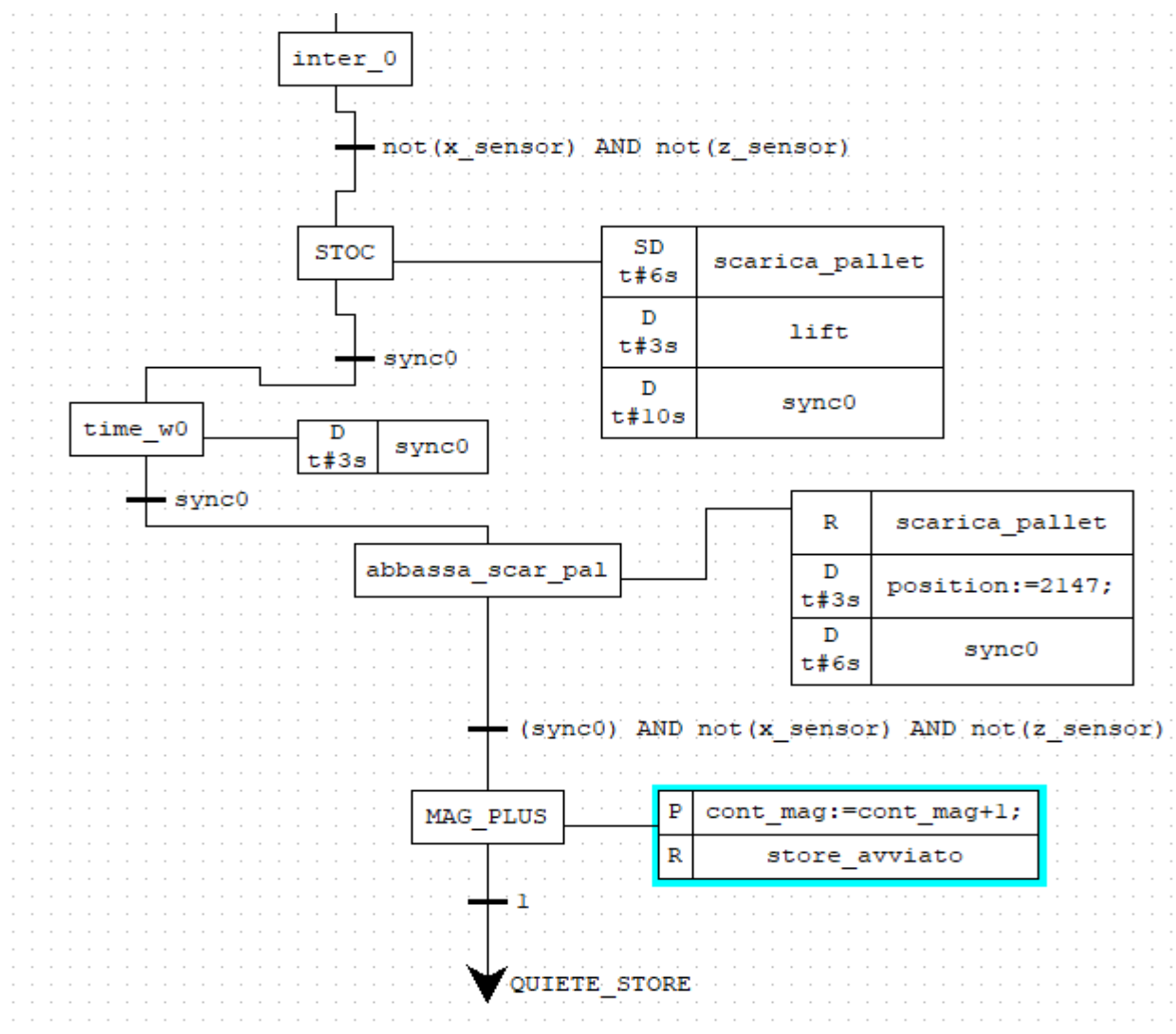


Illustrazione 13: fase finale dello store

Quando il braccio giunge in posizione viene eseguito lo scarico del pallet nello scaffale. Infine la posizione del braccio viene ripristinata e la variabile `cont_mag` è incrementata di un'unità. La gestione dei movimenti del braccio, nella fase di scarico del pallet, è stata realizzata mediante l'utilizzo dei qualificatori **SD**, **D** i quali hanno permesso di gestire al meglio i tempi di movimento dello stesso analizzati nella fase di simulazione tramite FACTORYIO.

L'operazione di **retrieve** consentirà all'operatore di prelevare un pallet , posto all'interno del magazzino, da una posizione da lui selezionata.

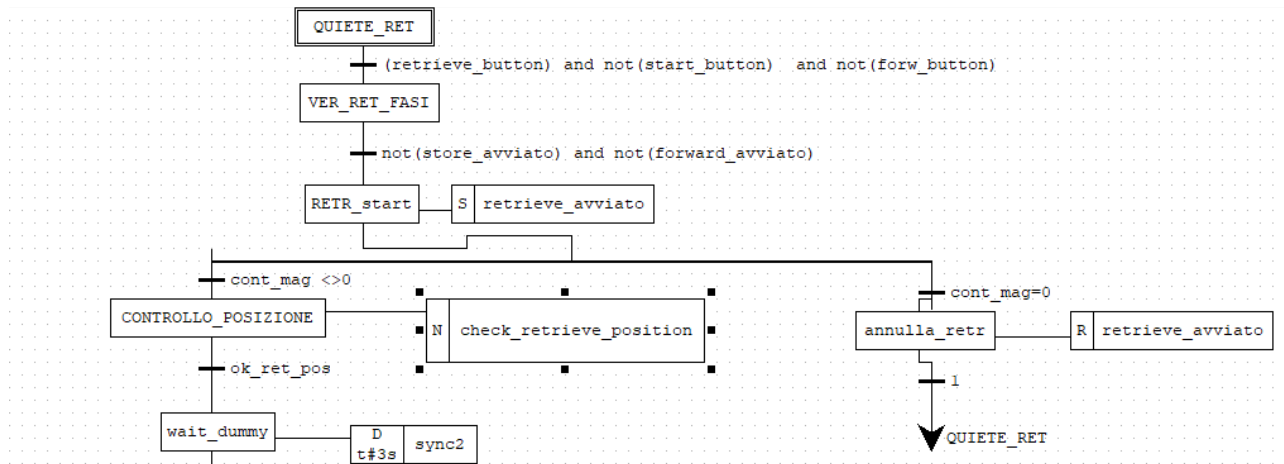


Illustrazione 14: SFC azione retrieve

In maniera molto simile all'azione di storage viene effettuato un controllo atto a verificare che all'interno del magazzino sia presente almeno un elemento; inoltre viene effettuato un controllo sulla posizione, inserita dall'operatore, del pallet da prelevare.

```

if( (manual_position<1) OR (manual_position>max_dim) ) then
    ok_ret_pos:=False;
    errore_0:=1;
else
    if ( (magazzino[manual_position]=False)) then
        ok_ret_pos:=False;
        errore_0:=1;
    else
        position:=manual_position;
        magazzino[manual_position]:=False;
        ok_ret_pos:=True;
        errore_0:=0;
    end_if;
end_if;
  
```

Illustrazione 15: azione check_retrieve_position in linguaggio ST

La restante parte del SFC segue, in maniera analoga, i passi già visti nell'SFC dell'azione di storage con l'unica differenza che alla fine delle operazioni la capacità del magazzino viene decrementata di un'unità.

Infine nel programma main possiamo trovare l'ultimo SFC di interesse: il **forward**.

L'azione di forward permette di inoltrare pacchi dai carrelli d'entrata a quelli d'uscita affinché questi siano sottoposti all'attività di smistamento senza il coinvolgimento di alcuna operazione di storage.

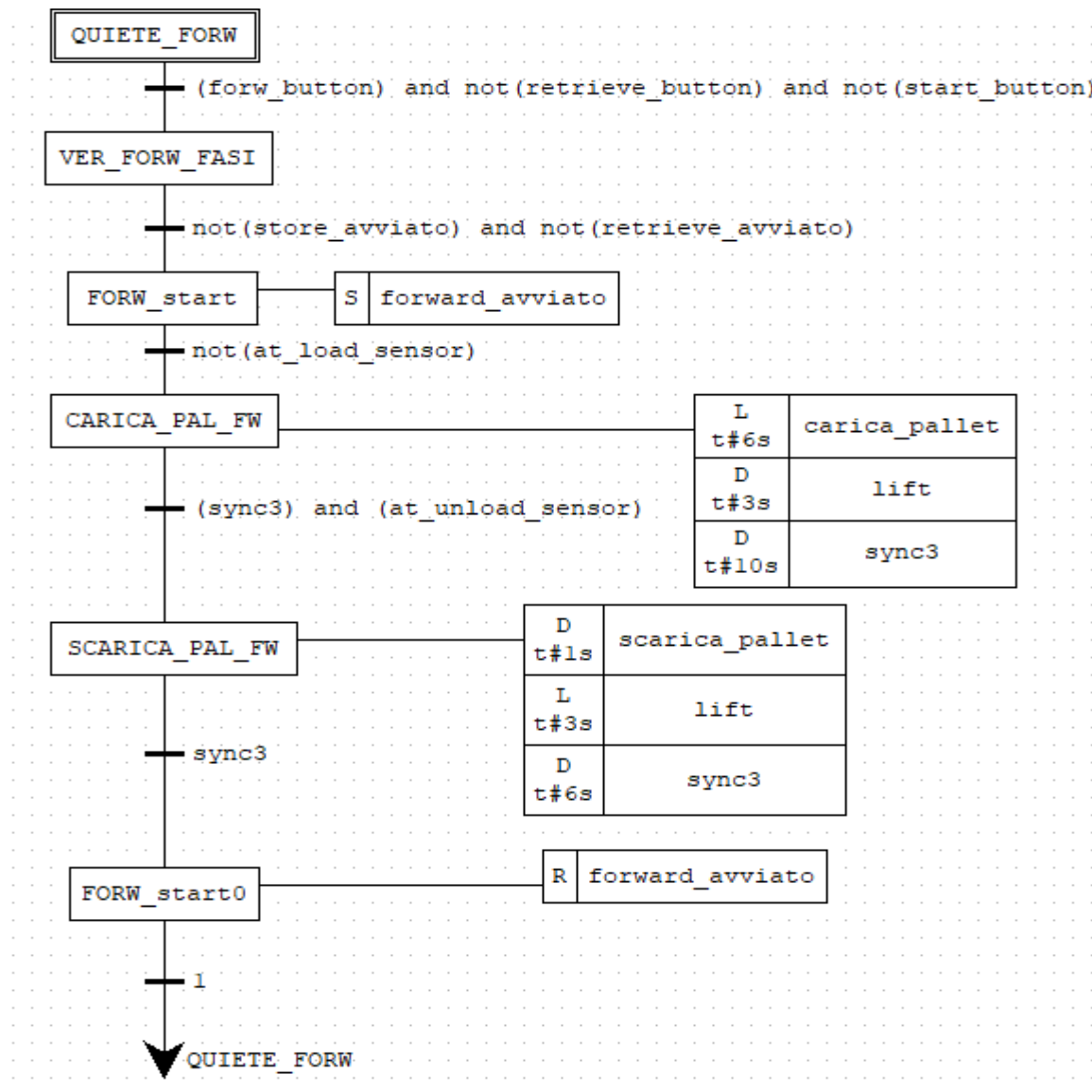


Illustrazione 16: SFC azione forward

L' SFC non presenta particolari criticità. Alla pressione del pulsante di forward, assicuratosi che gli altri due pulsanti siano spenti, il sistema verrà posto nello stato di forward_avviato ed il pallet transiterà oltre il braccio meccanico pronto ad essere smistato.

Nel programma smistatore sono contenuti invece gli SFC che consentono al nostro impianto di smistare i pallet in base alla rispettiva altezza su diversi carrelli.

Infatti sulla base di tre livelli di altezza (basso, medio, alto) essi verranno così ripartiti:

1. pacchi alti → carrello di sinistra
2. pacchi medi → carrello centrale
3. pacchi bassi → carrello di destra



Illustrazione 17: livelli d'altezza

Per conoscere il numero di pallet coinvolti nella operazione di smistamento abbiamo implementato un “contatore di coda”. Esso ci permette di ottimizzare l'accensione/spegnimento degli attuatori coinvolti in tale operazione. Ogni volta che un pallet arriverà sulla linea di smistamento il contatore sarà incrementato; finita la fase di smistamento esso verrà decrementato.

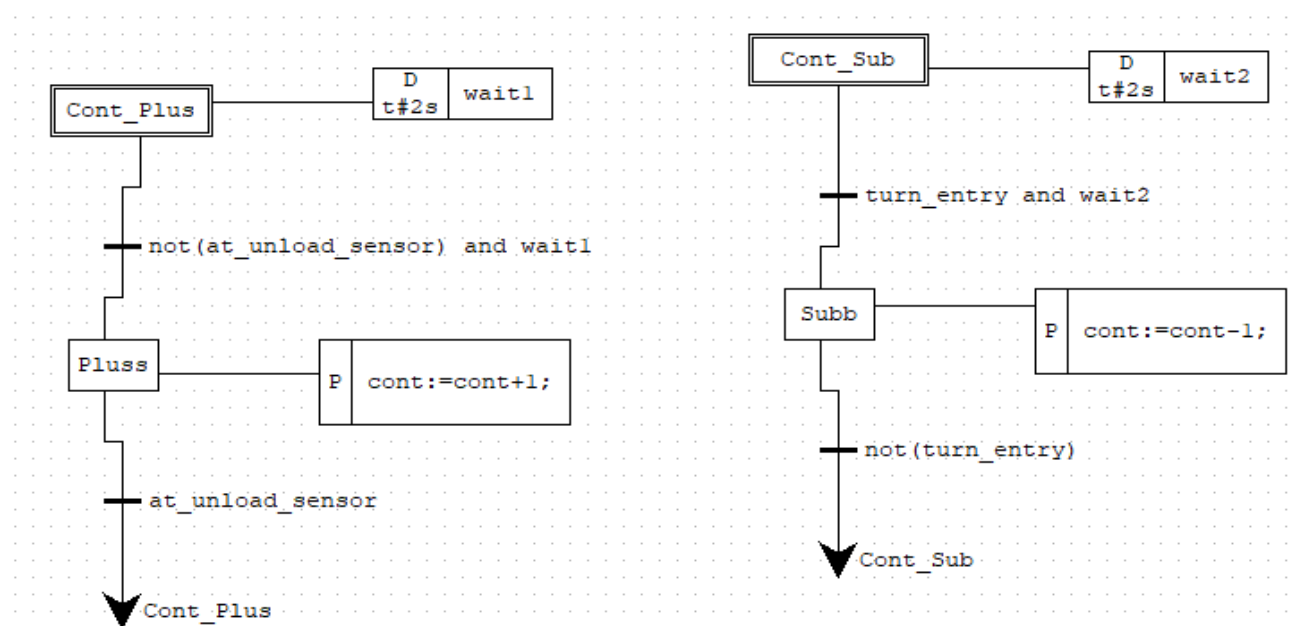


Illustrazione 18: contatori pallet da smistare

La linea di ingresso allo smistatore sarà costituita da tre carrelli distinti; il primo è gestito attraverso il programma main mentre i restanti vengono gestiti con il seguente diagramma.

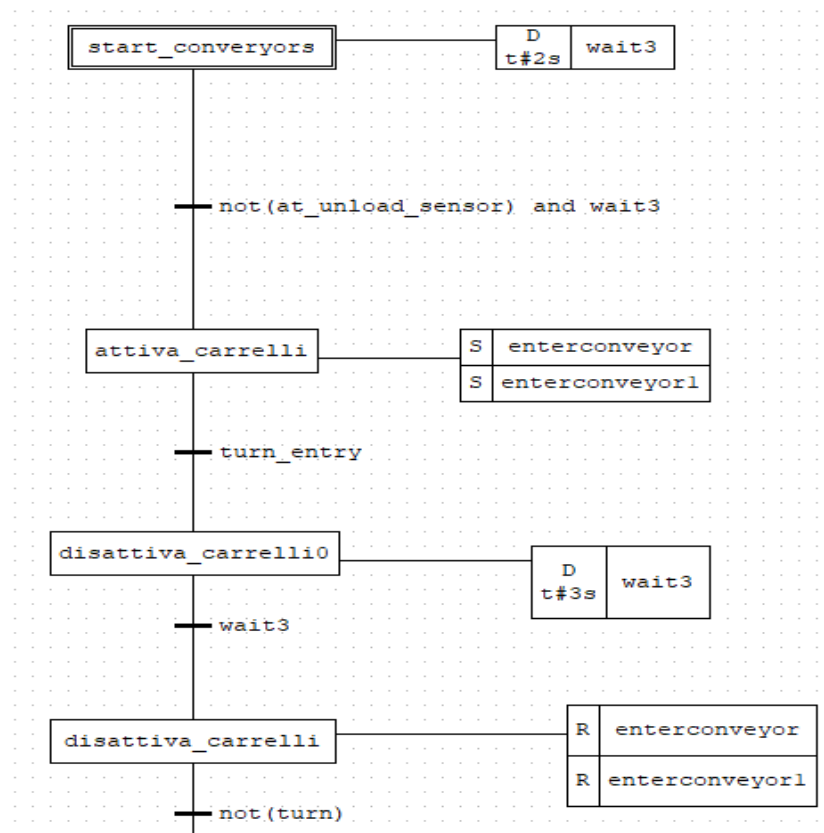
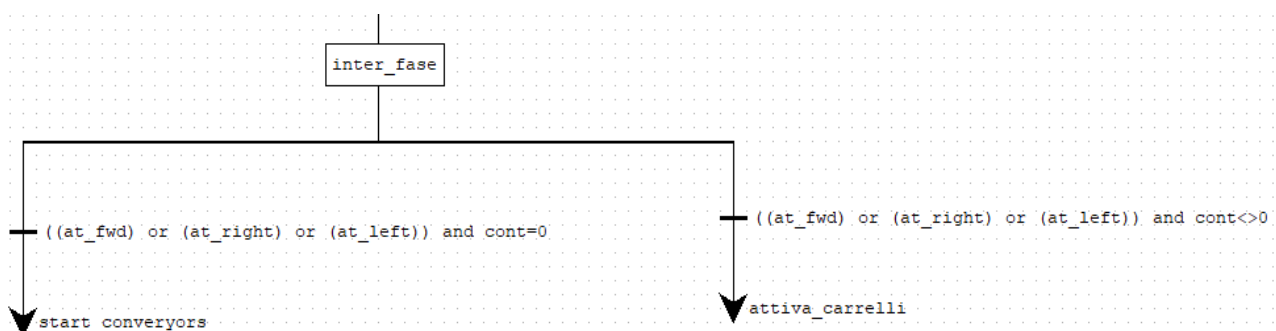


Illustrazione 19: carrelli smistatore

Una volta che il pallet è posizionato sulla piastra “smistatrice” viene valutata la presenza di ulteriori pacchi da smistare. Se tale numero di pacchi è nullo allora i carrelli dello smistatore dovranno essere “definitivamente” spenti altrimenti verrà rieseguita la fase di attivazione dei carrelli.



razione 20: gestione carrelli smistatore

Illust

La vera scelta di smistamento viene effettuata grazie all'ausilio di sensori che permettono di valutare l'altezza dei pacchi ed una piastra rotante.

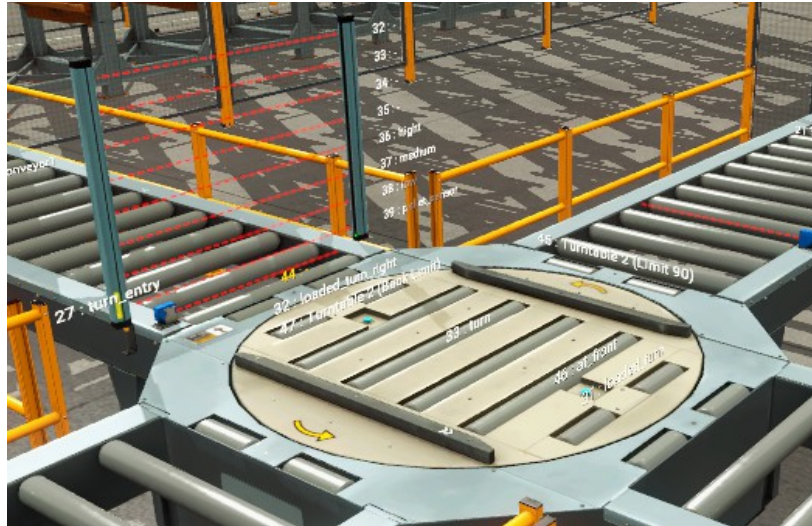


Illustrazione 21: piastra smistatrice

Il diagramma da noi realizzato per la gestione dello smistamento è costituito da una divergenza iniziale composta da tre ramificazioni. Ciascuna verrà percorsa in base ai rilevamenti dei sensori d'altezza. Ad esempio: quando i primi due sensori saranno veri ed il terzo sarà falso, verrà percorsa la ramificazione implementata per i pacchi medi.

Dal punto di vista strutturale le divergenze sono molto simili tra loro e differiscono per l'attivazione di una specifica variabile (left_ok , right_ok, fwd_ok) che permette di agire sulla rotazione della piastra e di gestire i relativi carrelli d'uscita che porteranno il pallet a destinazione.

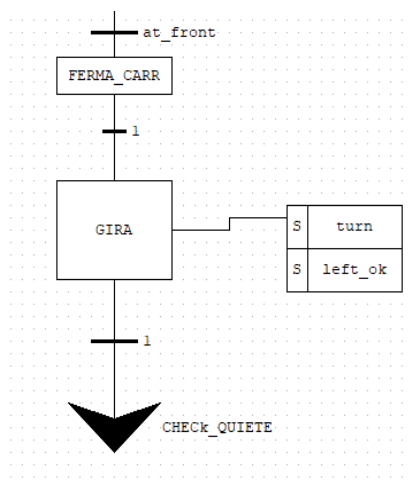


Illustrazione 23: ramo centrale

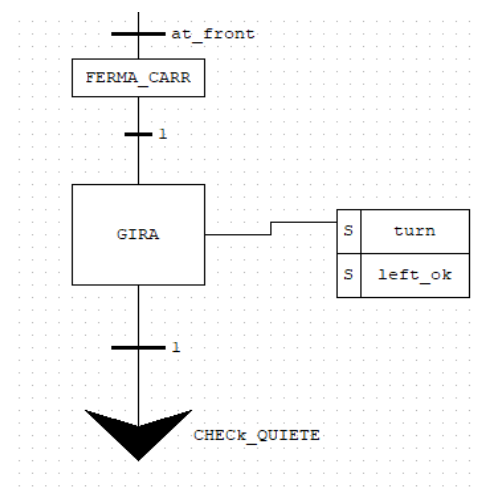


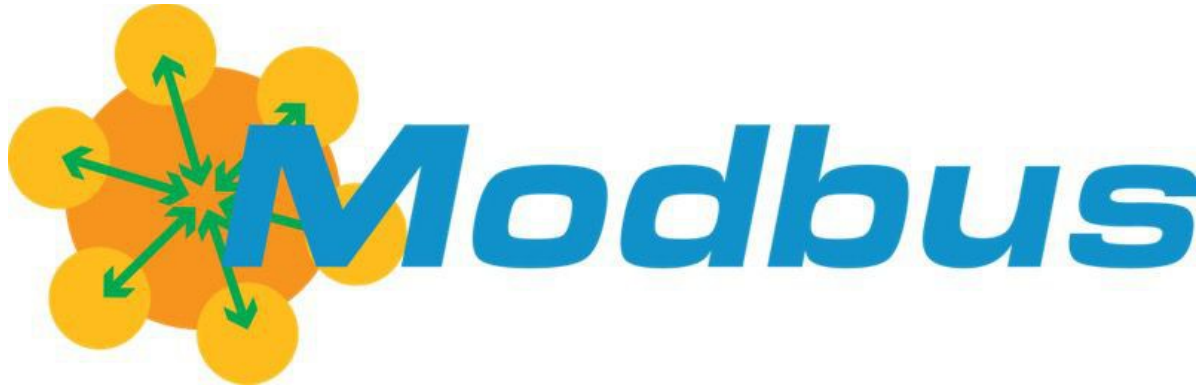
Illustrazione 22: ramo sinistro

1.2 Modbus TCP/IP

Il Modbus è un protocollo di comunicazione appartenente allo strato APPLICAZIONE (lv. 7) dello stack ISO/OSI che consente di comunicare con un PLC. Nato come un protocollo di comunicazione seriale, oggi si adatta allo standard TCP/IP.

Tale protocollo consente una comunicazione CLIENT/SERVER o MASTER/SLAVE tra dispositivi connessi su tipi di reti e bus diversi. Nel nostro caso esso ci ha fornito la possibilità di collegare al nostro soft PLC una HMI, realizzata in SCADABR, attraverso la quale abbiamo monitorato e gestito il comportamento del sistema. Inoltre ci ha consentito di utilizzare un ambiente di simulazione quale quello offerto da FACTORY IO.

Attraverso lo scambio dei messaggi i client possono interagire con l'entità server ed utilizzando i comandi modbus possono agire sui registri dell'unità remota.



1.3 Factory IO

Factory IO è un software che permette di progettare e simulare applicazioni industriali

<https://factoryio.com/features> .

Esso fornisce una libreria di pezzi industriali quali carrelli, elevatori, sensori etc. attraverso i quali è possibile creare una scena rappresentante un impianto. Inoltre esso fornisce un completo supporto al protocollo modbus e permette dunque di interfacciarsi con un PLC.

Per poter connettere FACTORY IO al soft PLC realizzato attraverso OPENPLC per prima cosa abbiamo dovuto creare una nuova configurazione.

Tale azione può essere eseguita recandosi alla voce **Menù** → **Device** in Factory IO.

Dal menù a tendina abbiamo selezionato infine la voce TCP/IP Server.

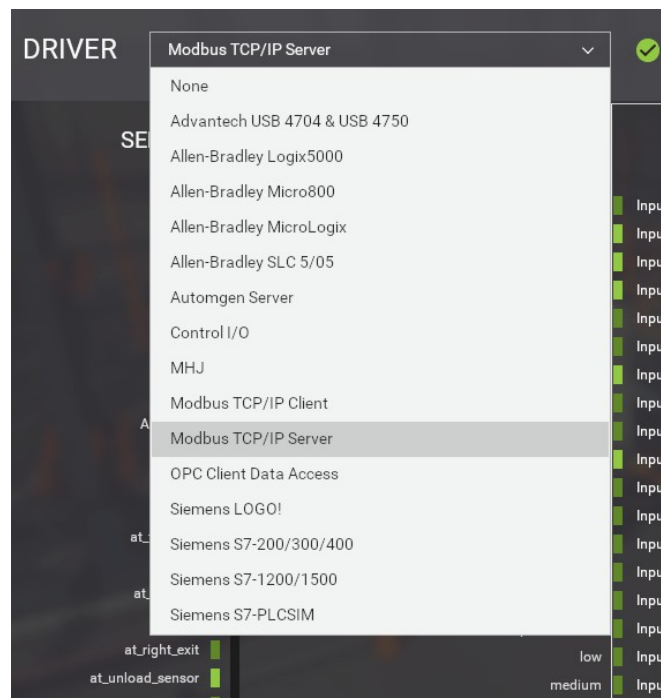


Illustrazione 24: settaggio driver

Recandoci alla voce configurazione abbiamo settato l'indirizzo ip della macchina sulla quale si trova il soft plc e come numero di porta 503.

Sebbene il protocollo modbus lavori sulla porta di default 502 , per evitare conflitti con OpenPLC, lo stesso Thiago ci ha suggerito di utilizzare tale numero di porta.

Possiamo anche settare il numero di digital input/output con il quale lavorerà l'impianto.

Il passo finale è stato quello di configurare attraverso l'interfaccia web di OpenPLC runtime, accessibile tramite localhost:8080, uno slave device.

Server

☒ Auto start

Host

192.168.56.1

Port

503

Illustrazione 25: configurazione ip/port

I/O Points

	Offset	Count
Digital Inputs	0	40
Digital Outputs	0	40
Register Inputs	0	0
Register Outputs	0	1

Illustrazione 26: configurazione I/O points

Edit slave device

Device Name

FactoryIO

Device Type

Generic Modbus TCP Device

Slave ID

1

IP Address

192.168.1.136

IP Port

503

Discrete Inputs (%IX100.0)

Start Address: 0 Size: 100

Coils (%QX100.0)

Start Address: 0 Size: 100

Input Registers (%IW100)

Start Address: 0 Size: 100

Holding Registers - Read (%IW100)

Illustrazione 27: configurazione slave device

Qualora tutti i settaggi siano stati impostati correttamente , nel Runtime Logs verranno riportati dei messaggi di connessione stabilita tra Factory IO ed il nostro PLC.

1.4 ScadaBR

ScadaBR è un sistema SCADA sviluppato come software open-source. Per poterlo utilizzare abbiamo adottato una macchina virtuale linux-based e attraverso l'IP fornito è stato possibile accedere alla web app che consente di operare sul sistema Scada.

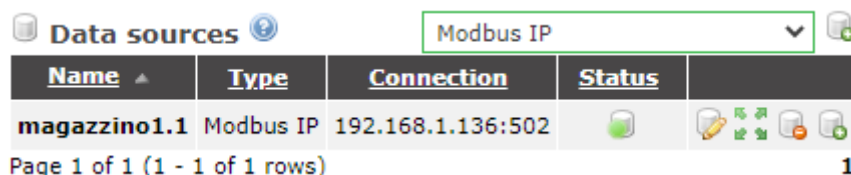
```
Welcome to ScadaBR! To use ScadaBR, open your browser and navigate to:
http://192.168.1.106:8080/ScadaBR
Have Fun!
scadabr login:
```

Illustrazione 28: ScadaBR come macchina virtuale

Un software SCADA, Supervisory Control and Data Acquisition, consente all'operatore di interfacciarsi con il processo controllato.

Per poter analizzare lo stato dell'impianto, il sistema deve acquisire tramite opportuni driver di comunicazione i dati dello stesso. Questi dati vengono raccolti in un database, il nucleo del sistema scada, che in ScadaBR è rappresentato dalla **DataSource**.

In particolare ScadaBR all'atto della creazione di una nuova DataSource ne consente di stabilire il tipo, esso va ad influire sulla tipologia di protocollo che verrà utilizzato per la comunicazione con il PLC. Nel nostro caso è stato selezionato come type il modbus IP.



Name	Type	Connection	Status
magazzino1.1	Modbus IP	192.168.1.136:502	

Page 1 of 1 (1 - 1 of 1 rows)

Illustrazione 29: creazione DataSource

La configurazione del DataSource segue ormai il noto settaggio dell'IP della macchina sul quale il PLC è in esecuzione, il numero di porta ed altri valori come l'update period, timeout etc...

Configurata la DataSource bisognerà aggiungere i data points.

Per ogni data point è necessario inserire :

1. un nome
2. un register range: tipologia del data point (input status, coil status, HRegister, Iregister)
3. un offset

Modbus IP properties

Name: magazzino1.1

Export ID (XID): DS_727403

Update period: 1 millisecond(ms)

Quantize: ☐

Timeout (ms): 500

Retries: 2

Contiguous batches only: ☐

Create slave monitor points: ☐

Max read bit count: 2000

Max read register count: 125

Max write register count: 120

Transport type: TCP with keep-alive

Host: 192.168.1.136

Port: 502

Encapsulated: ☐

Illustrazione 30: proprietà della DataSource

Point details

Name: auto_mode

Export ID (XID): DP_364824

Slave id: 1

Register range: Input status

Modbus data type: Binary

Offset (0-based): 807

Bit: 0

Number of registers: 0

Character encoding: ASCII

Settable: ☐

Multiplier: 1

Additive: 0

Illustrazione 31: creazione del Data Point

L'offset ci consente di mappare gli indirizzi del modbus con gli indirizzi dei registri del PLC. In particolare in OpenPLC il mapping segue le regole contenute nella seguente tabella.

Register Type	Usage	PLC Address	Modbus Address	Register Size	Value Range	Access
Coil Registers	Digital Outputs	%QX0.0 - %QX99.7	0 - 799	1 bit	0 or 1 OFF or ON	read and write
Discrete Input Registers	Digital Inputs	%IX0.0 - %IX99.7	0 - 799	1 bit	0 or 1 OFF or ON	read only
Input Registers	Analog Inputs	%IW0 - %IW99	0 - 1023	16 bits	0 to 65,535	read only
Holding Registers	Analog Outputs	%QW0 - %QW99	0 - 1023	16 bits	0 to 65,535	read and write

Illustrazione 32: Modbus Address Mapping

Una volta che il DataSource ed i Data Points sono stati configurati ed abilitati è possibile supervisionare il processo ed intervenire su di esso qualora ve ne fosse la necessità.

Points

Name	Data type	Status	Slave	Range	Offset (0-based)
auto_mode	Binary		1	Input status	807
carrello_carico	Binary		1	Coil status	801
carrello_entrata	Binary		1	Coil status	800
carrello_uscita	Binary		1	Coil status	808
carrello_uscita_1	Binary		1	Coil status	811

Illustrazione 33: i data points

ScadaBR offre, inoltre, la possibilità di implementare un HMI (Human Machine Interface) attraverso la quale l'operatore può ricevere informazioni dall'impianto ed eventualmente agire su di esso. Tale HMI può essere realizzata tramite la graphical view che permette di aggiungere svariati Widget collegabili ai Data Points.

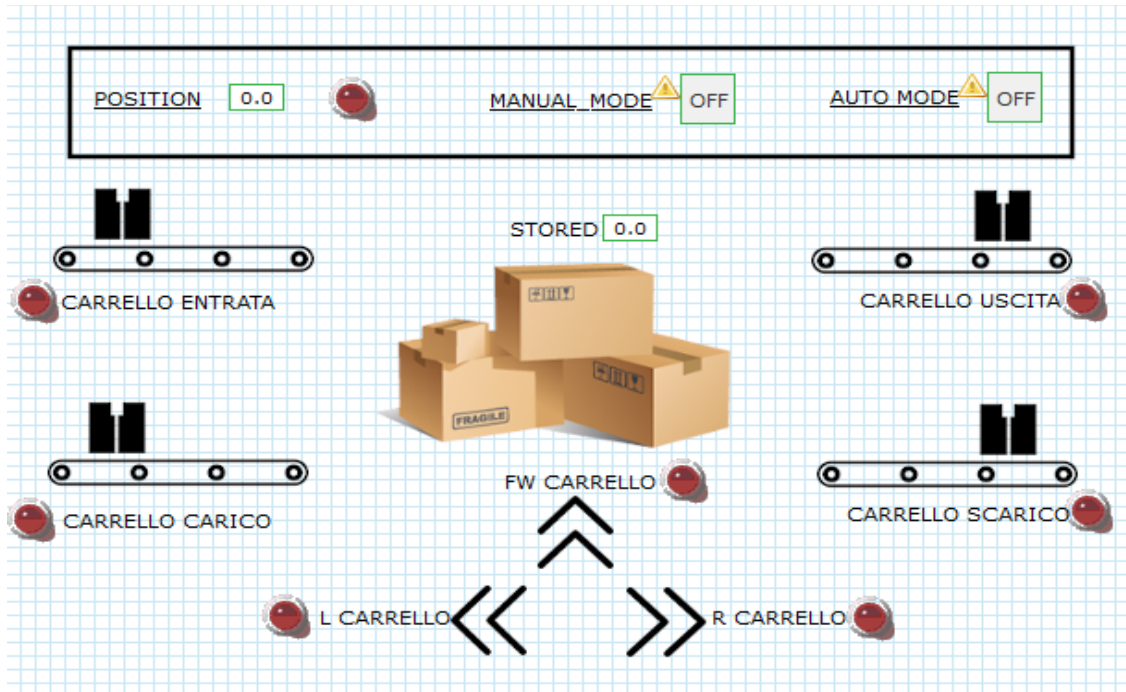


Illustrazione 34: Vista della HMI creata in ScadaBR

Un' ulteriore funzionalità che ScadaBR offre è la **gestione degli allarmi**. Esso, infatti, permette di impostare e gestire allarmi al verificarsi di un determinato evento.

Nel nostro caso abbiamo configurato un allarme in grado di avvertire l'operatore qualora la posizione inserita per le operazioni di store e retrieve non sia valida. Ciò può essere ottenuto configurando un **event detector** nelle point properties ed associando ad esso un livello d'allarme.

Point properties Data source: magazzino1.1 Point name: error	Event detectors Type: Change Type: State detector Export ID (XID): PED_276326 Alias: Invalid Position Alarm level: Urgent State: One Duration: 0 second(s)
Logging properties Logging type: When point value changes Purge: After 1 year(s) Default cache size: 1 Reset cache	
Purge now Purge data older than 1 year(s)	

Capitolo 2: Il PID

I controllori PID (Proportional – Integral – Derivative) sono una sottocategoria di controllori largamente utilizzati nell'ambito industriale. Essi offrono supporto al controllo modulante, ossia quella tipologia di controllo in cui i vari dispositivi fanno in modo che le variabili controllate del processo inseguano dei valori desiderati.

L'azione di controllo generata da un controllore PID è data dalla somma di tre termini e la sua espressione nel dominio del tempo è la seguente:

$$u(t) = K_p * e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{d}{dt} e(t)$$

Mentre nel dominio di Laplace essa è rappresentata dalla seguente formula:

$$U(s) = K_p * E(s) + \frac{K_i}{s} E(s) + s K_d * E(s)$$

Sempre in tale dominio il controllore PID può essere espresso mediante la FDT:

$$K(s) = \frac{K_p * (1 + T_i s + T_i T_d * s^2)}{T_i s} \quad \text{con} \quad \frac{1}{T_i} = \frac{K_i}{K_p} \quad T_d = \frac{K_d}{K_p} \quad \text{tempo integrale, tempo derivativo}$$

Vogliamo far notare che il controllore così descritto non è realizzabile in quanto risulta essere un sistema improprio. (n > d con n grado del numeratore e d grado del denominatore)

Quindi per poter rendere il PID fisicamente realizzabile si è soliti inserire un polo in alta frequenza al fine di rendere il sistema almeno proprio.

Nel nostro progetto abbiamo realizzato un controllore PI che consente il controllo della velocità di un motore elettrico alimentato in corrente continua.

Per fare ciò abbiamo imposto la natura del controllore ed abbiamo effettuato una sintesi nel dominio della frequenza.

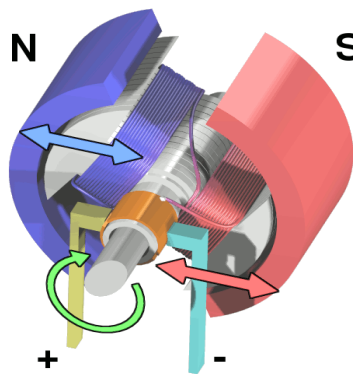
1.1 Introduzione e Modello Matematico

Un motore elettrico è una macchina elettrica che consente di trasformare una tensione continua in ingresso in una coppia motrice in uscita, ossia esso converte energia elettrica in energia meccanica resa disponibile sull'asse del motore.

Il motore elettrico è costituito da due parti: il rotore e lo statore.

Lo statore costituisce la parte esterna della macchina, l'involucro e genera un flusso magnetico costante, il cui valore dipende dall'intensità di corrente che scorre nelle sue spire interne.

All'interno dello statore c'è il rotore. Esso è realizzato in ferro laminato e presenta delle cavità attraverso le quali vengono fatti passare degli avvolgimenti in cui circola la corrente.



Tale corrente prende il nome di corrente d'armatura e il suo passaggio in tale avvolgimento di spire, nel rotore, genera un campo elettromagnetico. Tale campo è immerso a sua volta nel campo magnetico generato dallo statore, caratterizzato dalla presenza di due o più magneti.

Il rotore inizia a ruotare per effetto della forza di Lorentz generata dal passaggio di corrente nei suoi avvolgimenti e sotto l'azione del campo magnetico prodotto dallo statore.

Il motore DC può essere illustrato in maniera schematica come segue.

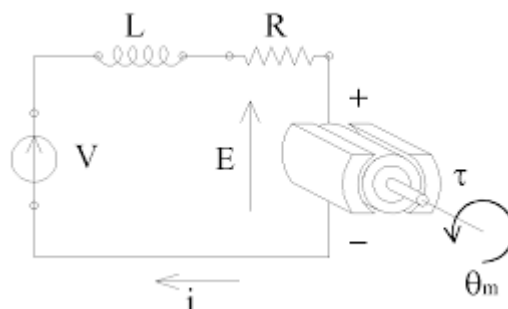


Illustrazione 35: schema di un motore DC

Le equazioni differenziali, descrittive del sistema, possono essere ottenute come illustrato di seguito.

Applicando l'equazione di Kirchhoff alle maglie al circuito in figura otteniamo:

$$-v + Ri + \frac{Ldi}{dt} + e = 0 \quad \text{con}$$

v = Tensione applicata al motore

R = Resistenza

L = Induttanza

ω = Velocità angolare

e = forza elettromotrice indotta

Per poter consentire la rotazione la forza applicata sul motore (T) deve superare la forza di attrito e la forza di inerzia.

$$Fa + Fb - T = 0$$

Fa = forza di inerzia

Fb = forza di attrito

T = forza del motore

Notando che:

$$e = K_c * \omega$$

$$T = K_i * i$$

$$Fa = -J * \left(\frac{d\omega}{dt} \right)$$

$$Fb = -B * \omega$$

Le equazioni caratteristiche tenendo conto delle costanti legate al motore saranno:

$$L \frac{di}{dt} + Ri = v - K_c * \omega$$

$$J \frac{d\omega}{dt} + B\omega = K_i * i$$

K_c = costante di proporzionalità della velocità angolare del motore.

K_i = costante di coppia.

Applicando la trasformata di Laplace il sistema diventa:

$$L(s * i(s) - i(0)) + R * i(s) = V(s) - K_c * \omega(s)$$

$$J(s * \omega(s) - \omega(0)) + B * \omega(s) = K_i * I(s)$$

Considerando nulle le condizioni iniziali otteniamo:

$$V(s) = (R + L * s) * I(s) + K_c * \omega(s) \quad [1]$$

$$(J * s + B) * \omega(s) = K_i * I(s) \quad [2]$$

La prima equazione [1] possiamo scriverla in funzione di I(s):

$$I(s) = \frac{(V(s) - e(s))}{(R + L*s)} \quad \text{con} \quad e(s) = K_c * \omega(s)$$

Poiché abbiamo che $T(s) = K_i * I(s)$ e per la [2], sostituendo l'espressione di I(s) abbiamo :

$$(J*s + B) * \omega(s) = K_i * \left(\frac{(V(s) - K_c * \omega(s))}{(R + L*s)} \right)$$

Effettuando il rapporto tra $\omega(s)$ ed $V(s)$, abbiamo la f.d.t che descrive il nostro impianto:

$$G(s) = \frac{(\omega(s))}{(V(s))} = \frac{(K_i)}{((R + L*s) * (J*s + B) + K_i * K_c)}$$

I parametri elettrici e meccanici del motore utilizzati durante la fase di progettazione e simulazione sono stati reperiti attraverso un datasheet consultabile sul sito dell'azienda [Siboni SRL](#) .

In particolar modo abbiamo considerato il motore DC 16P Series.

Simbolo	Descrizione	Valore
R	Resistenza Totale	7.9 Ω
L	Induttanza	4.16 mH
J	Momento di Inerzia	3.2*10 ⁻⁶ Kg*m ²
B	Attrito	8.1*10 ⁻⁶
K _i	Costante di Coppia	0,032 N*m/A
K _c	Costante di Tensione	3.95*10 ⁻³ V/Rpm

Il coefficiente d'attrito può essere calcolato come:

$$\frac{Coppia\ nominale}{Velocità\ nominale} = \frac{0,03}{3700} = 8,1 * 10^{-6}$$

1.2 Progettazione del Pid

Una volta ricavati i parametri caratteristici del motore, abbiamo implementato il nostro controllore PI attraverso l'ausilio del software Matlab. La strategia adottata è stata quella di imporre la struttura del nostro controllore affinché vengano rispettate le seguenti specifiche:

- errore a regime nullo per $r(t)=1(t)$
- $s \leq 20\%$
- $ta_{5\%} < 0.1 \text{ s}$

La funzione di trasferimento del motore, sostituendo i parametri numerici è la seguente:

```
motorG =  
  
          0.032  
-----  
1.331e-08 s^2 + 2.531e-05 s + 0.0001904  
  
Continuous-time transfer function.
```

Illustrazione 36: FDT del Motore

Ottenibile attraverso i seguenti comandi Matlab:

```
syms s;  
  
numG=Ki;  
denGs=(L*s+R)*(J*s+B)+Ki*Kc;  
denG=sym2poly(denGs);  
motorG=tf(numG,denG);  
opt= stepDataOptions('StepAmplitude',24);  
step(motorG,opt);
```

Tracciando la risposta indiciale del sistema, abbiamo notato che la velocità d'uscita si assesta a valori molto prossimi alla velocità nominale indicata nel datasheet del motore, 3700 rpm.

Il passo successivo è stato quello di tradurre le specifiche di progetto in caratteristiche che la $F(s)$, ossia la funzione ad anello aperto, deve possedere.

In particolar modo:

- errore nullo a regime \rightarrow presenza di un integratore che non essendo presente nella fdt dell'impianto sarà aggiunto nella fdt del nostro controllore

- $s \leq 20\% \rightarrow \zeta \geq \frac{\left| \ln \frac{20}{100} \right|}{\sqrt{\pi^2 + \ln^2\left(\frac{20}{100}\right)}} \rightarrow f_m \geq 45^\circ$

- $t_{as} < 0.1 \rightarrow \frac{3}{0,45 * \omega_c} \leq 0.1 \rightarrow \omega_c \geq 66$ abbiamo considerato $\omega_c = 70$ rad/s

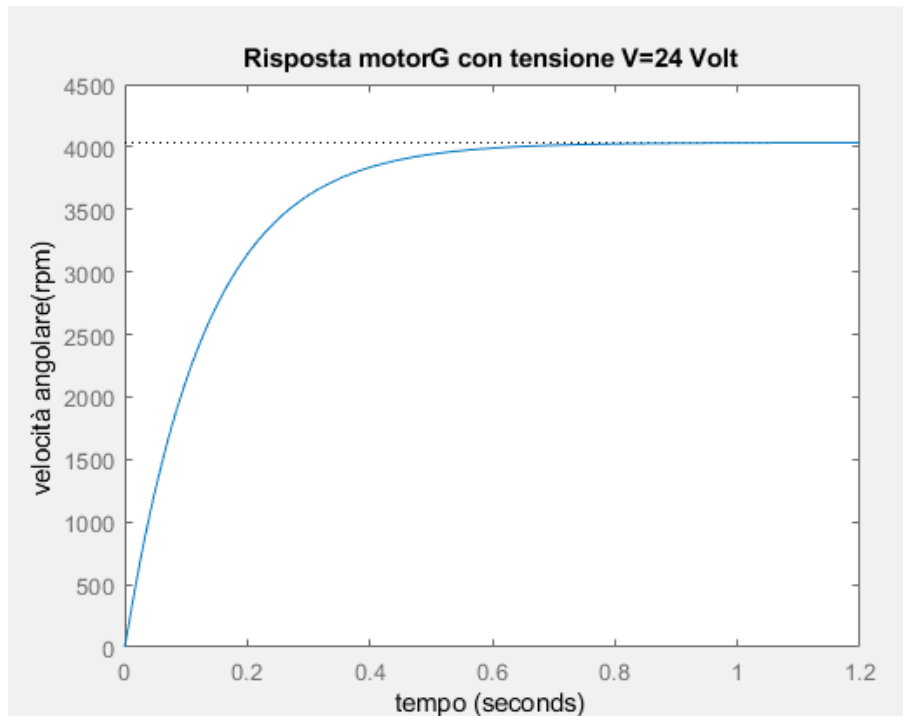


Illustrazione 37: Risposta del motore alla tensione nominale

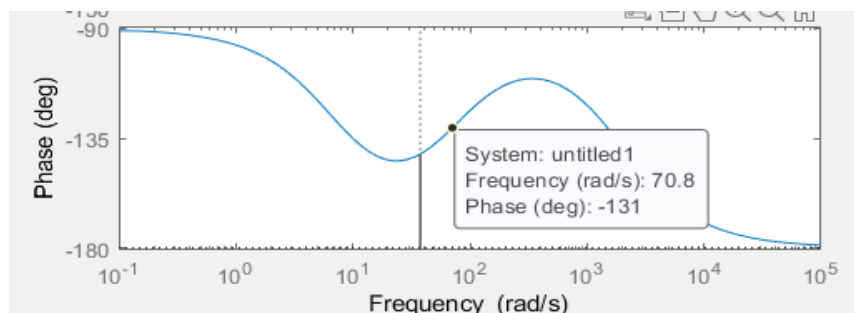
Avendo imposto come struttura del $K(s)$ un controllore PI, esso sarà della forma:

$$K(s) = \frac{K_p}{T_i} * \left(\frac{1 + T_i * s}{s} \right)$$

La presenza del polo nell'origine ci garantisce un errore nullo a regime, inoltre analizzando i diagrammi di Bode per la funzione ad anello aperto $F(s) = K1(s) * G(s)$ con $K1(s) = 1/s$, notiamo che alla pulsazione 70 rad/s dobbiamo recuperare il modulo di circa 10dB e la fase deve essere anticipata di 40° .

L'anticipo della fase può essere ottenuto sfruttando e piazzando in maniera opportuna lo zero del controllore PI. Nel nostro caso abbiamo piazzato tale valore alla pulsazione $\omega = 70$ rad/s ottenendo così il seguente controllore:

$$K2(s) = \frac{1 + 0.015s}{s}$$



L'ultimo passo è stato quello di bilanciare il rapporto $K=K_p/T_i$ affinché avessimo un aumento di 10dB alla pulsazione $\omega=70$ rad/s. Ciò è stato fatto moltiplicando la struttura del nostro controllore per $K=2.5$ e successivamente, volendo abbassare ancora un po' il valore della sovraelongazione percentuale, abbiamo scelto un K finale pari a 10.

$$K(s) = 10 * \left(\frac{1 + \frac{s}{70}}{s} \right)$$

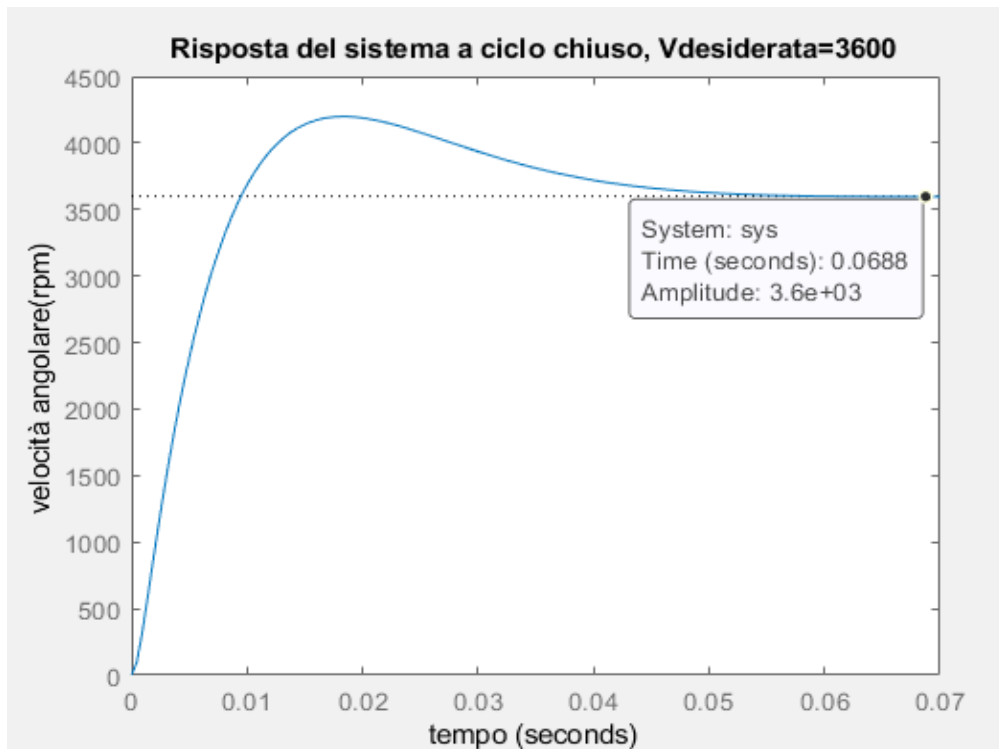


Illustrazione 38: Risposta del Sistema a CC

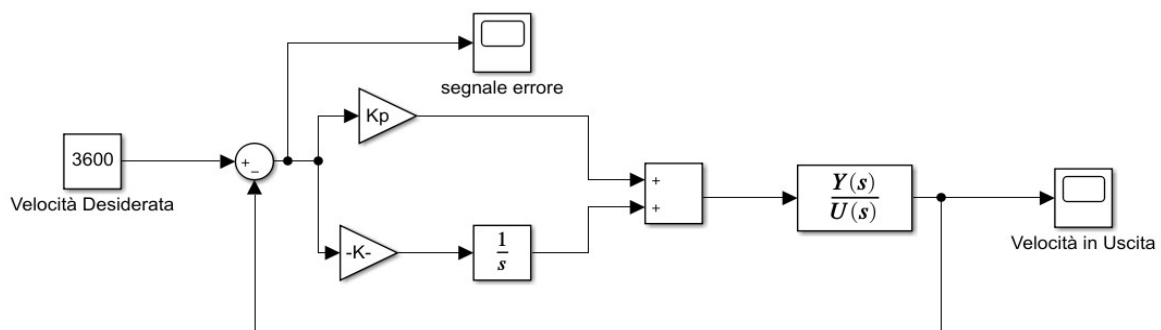


Illustrazione 39: Schema Simulink

1.3 Digitalizzazione

I controllori a supporto del controllo modulante furono originariamente sviluppati utilizzando svariate tecnologie analogiche. Oggi vengono implementati attraverso dispositivi di controllo digitali come ad esempio i microprocessori.

La sintesi di un controllore digitale può essere condotta seguendo due approcci:

1. Progettazione del controllore $K(s)$ continuo e successiva discretizzazione con ricavo di un controllore $K(z)$ avente un comportamento simile al suo analogo a tempo continuo
2. Discretizzazione dell'impianto o processo da controllare, $G(s) \rightarrow G(z)$, e successiva progettazione del controllore $K(z)$ a tempo discreto

Nel nostro progetto abbiamo utilizzato il primo approccio, in particolare come metodo di discretizzazione abbiamo utilizzato il **metodo di Tustin**, anche noto come metodo dei trapezi.

Tale metodo permette di digitalizzare il controllore, nota la sua fdt a tempo continuo, effettuando la sostituzione:

$$s = \frac{2}{T} * \left(\frac{z-1}{z+1} \right) \quad \text{con } T = \text{periodo di campionamento}$$

Nel nostro caso, essendo $\omega_{\max} = 167 \text{ rad/s}$ la massima pulsazione di interesse, abbiamo fissato $T = 2\pi / 10 * \omega_{\max}$. Sebbene tale scelta abbia prodotto l'uscita desiderata, abbiamo notato un valore di sovraelongazione $> 20\%$, dovuto al peggioramento del margine di fase introdotto dalla presenza dello ZOH.

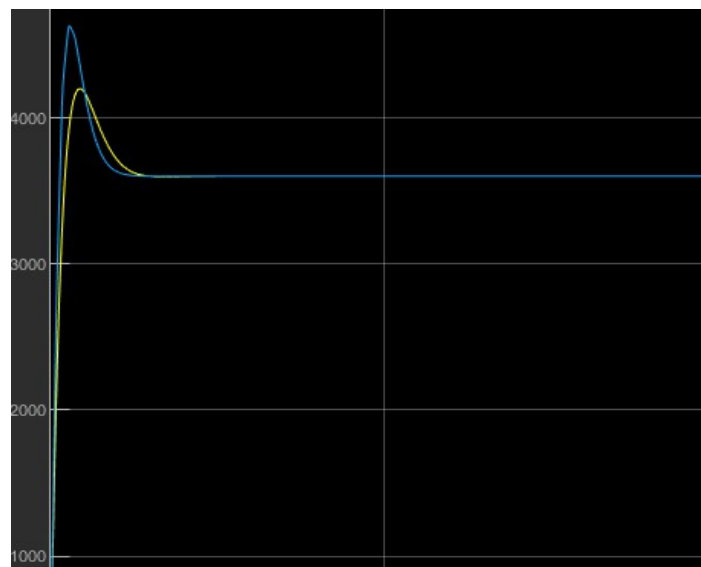


Illustrazione 40: sovraelongazione $> 20\%$

Per tale motivo abbiamo optato per un aumento della frequenza di campionamento all'ordine del KHz ottenendo l'uscita riportata in figura.



Illustrazione 41: Confronto uscite

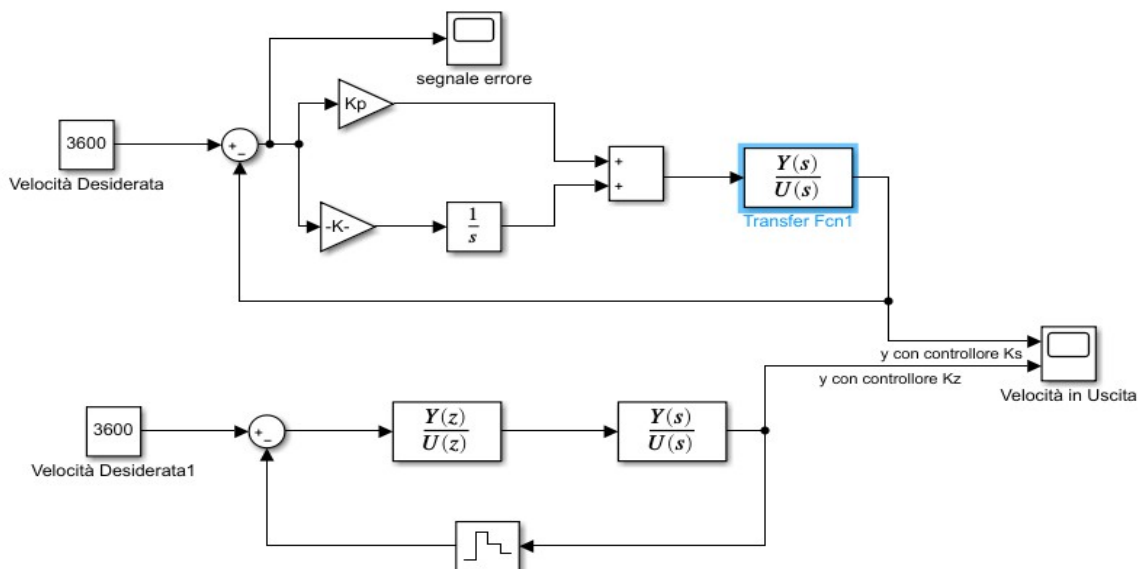


Illustrazione 42: Schema Simulink

Le procedure matlab utilizzate e lo schema simulink sono reperibile nei file: pid_elaborato.slx e pi_programma.m .

Conclusioni

Nella stesura della documentazione abbiamo cercato di illustrare in dettaglio le varie funzionalità e potenzialità dei software utilizzati; auguriamo al lettore di poterne trarre beneficio.

Questo progetto ci ha consentito di avvicinarci ad un campo, per noi, molto interessante ed ampio quale quello della automazione industriale. Il poter progettare e in seguito realizzare “praticamente” un PLC per un sistema semi-reale, il poter approfondire un sistema molto utilizzato nell'automazione come il motore DC hanno rappresentato per noi un'esperienza entusiasmante e formativa.

Vogliamo infine ringraziare il professore Adriano Mele che con grande disponibilità e sensibilità ha supervisionato il nostro operato.