

Progetto del corso di **Tecnologie Informatiche per l'Automazione Industriale** – TIAI  
(a.a. 2019/2020)

*“Ascensore”*

Il progetto d'esame consiste nello sviluppo di un software per gestire e controllare un ascensore, al fine di utilizzo di utenti qualsiasi.

Esso è stato sviluppato utilizzando tre diversi linguaggi di programmazione, per lo sviluppo del software da caricare su un PLC industriale. In particolare è stato utilizzato:

- Sequential Function Chart (SFC)
- Ladder
- Structured Text (ST)

Sono stati realizzati due programmi equivalenti, uno sviluppato interamente in SFC e ST (per la scrittura di funzioni un po' più dettagliate) e uno sviluppato in Ladder e ST (analogo motivo).

I due programmi anche se equivalenti presentano delle differenze per mettere in gioco le potenzialità di ogni linguaggio. La descrizione dettagliata verrà mostrata in seguito.

In più è stato scelto anche di progettare un controllore PID per gestire il riscaldamento/raffreddamento della cabina e per mantenere al loro agio gli utenti finì all'utilizzo. Anch'esso verrà descritto dettagliatamente in seguito

Giuseppe Francesco Di Cecio N46004000  
Giuseppe Frate N46004109

*Professore Adriano Mele*

## LADDER

Il sistema su cui si è deciso di implementare un controllo logico-sequenziale mediante PLC è un ascensore servente 4 piani di un complesso.

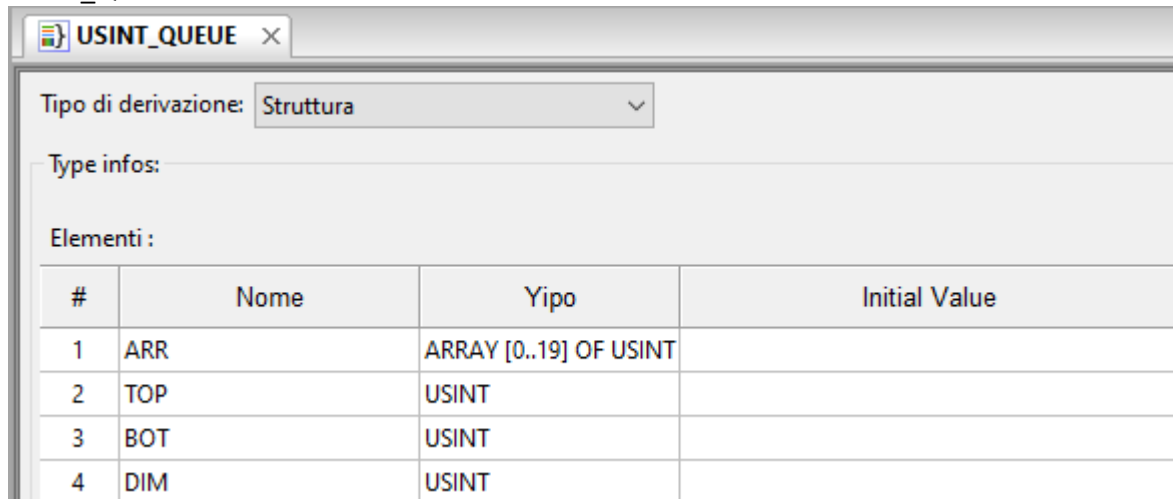
In questa sezione si esamina lo sviluppo della logica di controllo con il linguaggio ladder che ben si presta per la descrizione di un processo che evolve a eventi discreti.

## Configurazione

Il progetto è stato realizzato con Open-PLC, ovvero un soft PLC open source, su una macchina Windows. Dunque, la simulazione delle funzionalità non ha pretese di tempo reale, ma solamente di verificare la correttezza del controllore. A questo livello si specifica il numero di risorse, che nel caso in esame risulta essere una in quanto il PLC non prevede di agire su più impianti.

Inoltre, è possibile dichiarare tipi personalizzati.

USINT\_QUEUE:



#	Nome	Yipo	Initial Value
1	ARR	ARRAY [0..19] OF USINT	
2	TOP	USINT	
3	BOT	USINT	
4	DIM	USINT	

ARR: array di 20 USINT per la memorizzazione delle richieste.


TOP: testa della coda.

BOT: coda della coda.

DIM: numero di elementi in coda (utile per eventuali cicli su di essa).

Trattasi di una coda circolare con una taglia tale da consentire eventuali comportamenti indesiderati da parte degli utenti in fase di prenotazione.

Ora si definiscono i task, cioè quei flussi di esecuzione agenti su dati, ai quali vanno assegnati le istruzioni per realizzare il controllo. Inoltre, è qui che bisogna specificare eventuali variabili globali visibili alle POU interne.


Config0.Res0
×

Filtro Classi: Tutti ▼

#	Nome	Classe	Yipo	Location	Initial Value	Opzione	Documentazione
1	QUEUE	Global	USINT_QUEUE				
2	STATE	Global	USINT				
3	DIR	Global	BOOL				

Tasks:

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

Instances:

Nome	Yipo	Task
instance0	program0	task0

QUEUE: USINT\_QUEUE che gestisce le prenotazioni per l'utilizzo dell'ascensore.

STATE: USINT che memorizza la posizione corrente dell'ascensore.

DIR: BOOL che tiene traccia della direzione dell'ultimo movimento dell'ascensore (TRUE se è salito, FALSE se è sceso).

Il PLC avrà memorizzato un solo programma eseguito dall'unico task applicativo, periodico di periodo 20ms e a priorità massima. La scelta di non inserire multiprogrammazione è stata principalmente adottata per non disperdere troppo la logica di controllo trattandosi alla fine di un progetto non eccessivamente esteso.

## Blocchi funzionali

Affinché il programma sia modulare sono stati inseriti dei blocchi funzionali che agiscono su variabili interne e sulle variabili globali al livello di risorsa.

IN\_USINT\_QUEUE:

ST IN\_USINT\_QUEUE

Descrizione:  Filtro Classi: Tutti

#	Nome	Classe	Yipo	Initial Value	Opzione	Documentazione
1	QUEUE	External	USINT_QUEUE			
2	STATE	External	USINT			
3	DIR	External	BOOL			
4	I	Locale	USINT			
5	J	Locale	USINT			
6	H	Locale	USINT			
7	IN	Input	BOOL			
8	DATA	Input	USINT			
9	OUT	Output	BOOL			

```
1 IF IN THEN
2   IF DATA > STATE THEN
3     I := QUEUE.BOT;
4     FOR H := 0 TO QUEUE.DIM - 1 DO
5       IF DATA < QUEUE.ARR[I] OR (QUEUE.ARR[I] < STATE AND DIR = 1) THEN
6         EXIT;
7       END_IF;
8       I := (I + 1) MOD 20;
9     END_FOR;
10  ELSIF DATA < STATE THEN
11    I := QUEUE.BOT;
12    FOR H := 0 TO QUEUE.DIM - 1 DO
13      IF DATA > QUEUE.ARR[I] OR (QUEUE.ARR[I] > STATE AND DIR = 0) THEN
14        EXIT;
15      END_IF;
16      I := (I + 1) MOD 20;
17    END_FOR;
18  END_IF;
19
20  FOR J := QUEUE.TOP TO I + 1 DO
21    QUEUE.ARR[J] := QUEUE.ARR[J - 1];
22  END_FOR;
23
24  QUEUE.ARR[I] := DATA;
25  QUEUE.TOP := (QUEUE.TOP + 1) MOD 20;
26  QUEUE.DIM := QUEUE.DIM + 1;
27 END_IF;
28 OUT := QUEUE.DIM > 0;
```

Questo FB gestisce l'inserimento ordinato in coda. L'ordine è tale da minimizzare le oscillazioni dell'ascensore: esso servirà prima le richieste che lo portano a muoversi nell'ultima direzione registrata fino a che non si rende necessaria l'inversione.

Un ascensore così progettato non ha bisogno che venga specificata la direzione all'atto della richiesta di un piano.

## OUT\_USINT\_QUEUE:

**OUT\_USINT\_QUEUE** ×

Descrizione:  Filtro Classi: Tutti

#	Nome	Classe	Yipo	Initial Value	Opzione	Documentazione
1	QUEUE	External	USINT_QUEUE			
2	IN	Input	BOOL			
3	DATA	Output	USINT			

```

1 IF IN THEN
2   DATA := QUEUE.ARR[QUEUE.BOT];
3   QUEUE.BOT := (QUEUE.BOT + 1) MOD 20;
4   QUEUE.DIM := QUEUE.DIM - 1;
5 END_IF;
    
```

Questo FB è un semplice prelievo in coda da QUEUE che permette di passare alla prossima richiesta.

## EN\_USINT\_STATE:

**OUT\_USINT\_QUEUE** **EN\_USINT\_STATE** ×

Descrizione:  Filtro Classi: Tutti

#	Nome	Classe	Yipo	Initial Value	Opzione	Documentazione
1	STATE	External	USINT			
2	IN	Input	BOOL			
3	DATA	Input	USINT			

```

1 IF IN THEN
2   STATE := DATA;
3 END_IF;
    
```

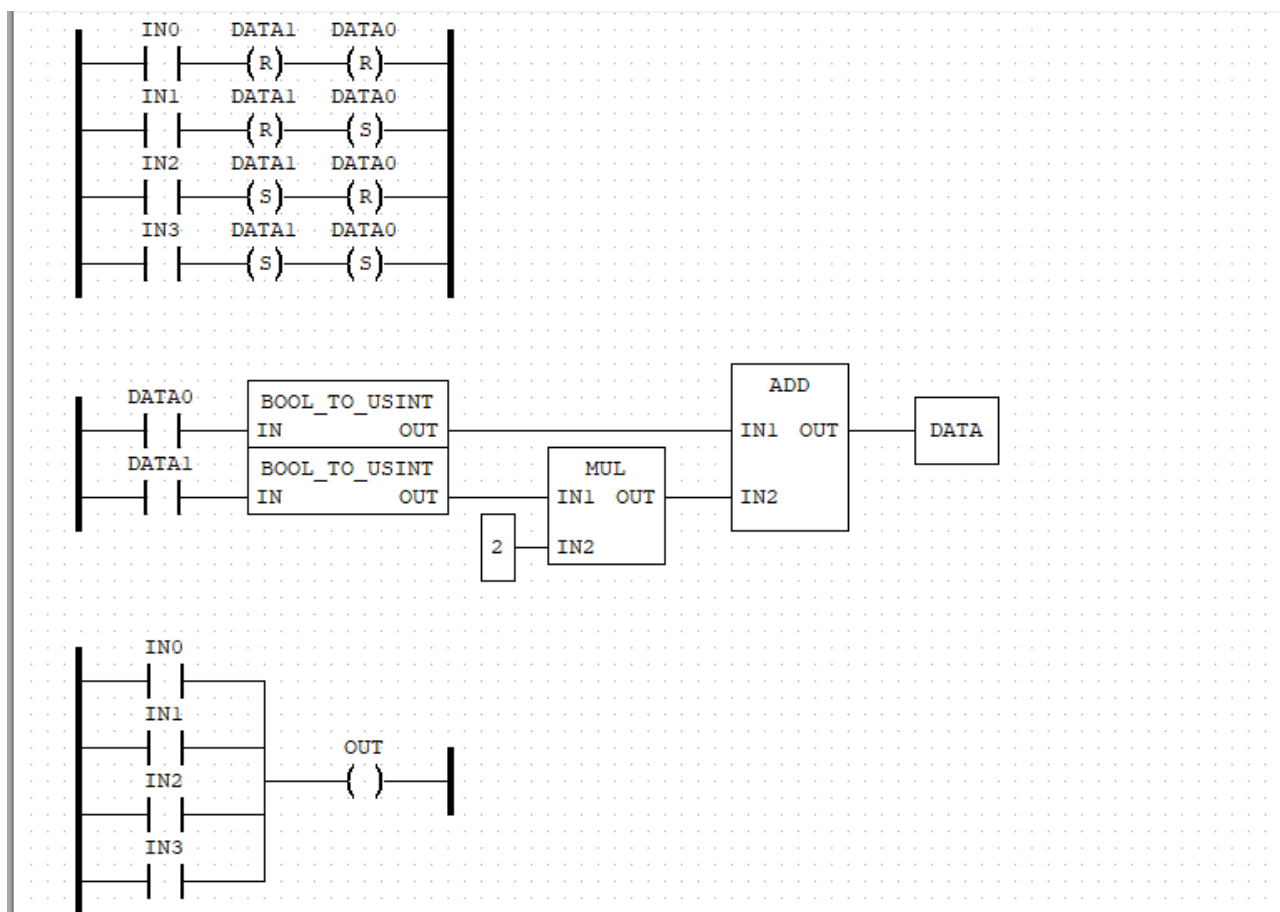
Questo FB non fa altro che assegnare a STATE una posizione.

## ENCODER:

**ENCODER** ×

Descrizione:  Filtro Classi: Tutti

#	Nome	Classe	Yipo	Initial Value	Opzione	Documentazione
1	IN0	Input	BOOL			
2	IN1	Input	BOOL			
3	IN2	Input	BOOL			
4	IN3	Input	BOOL			
5	DATA0	Locale	BOOL			
6	DATA1	Locale	BOOL			
7	OUT	Output	BOOL			
8	DATA	Output	USINT			



A differenza dei precedenti FB quest'ultimo è scritto in ladder.

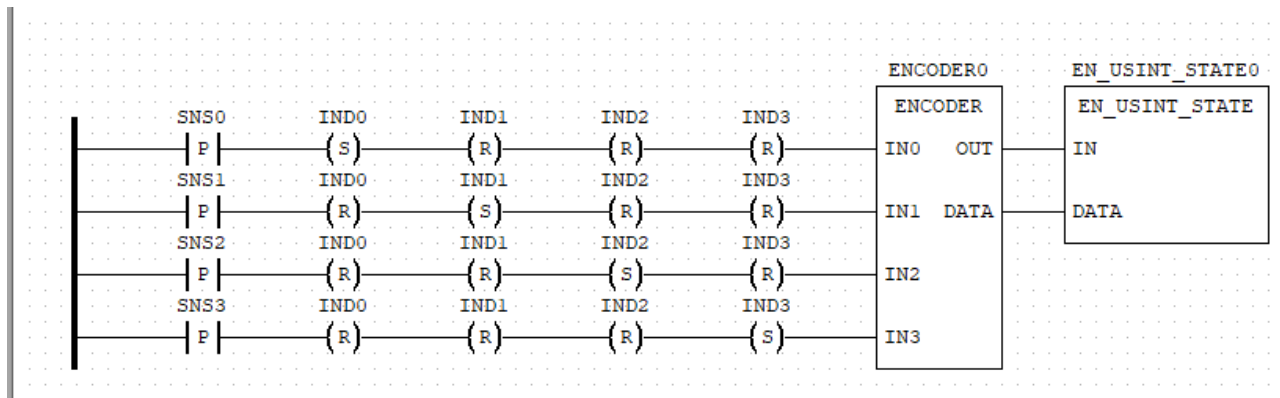
Il suo scopo è quello di assegnare a INi con  $i = 1, 2, 3, 4$  il valore di  $i$ . In questo modo, per esempio, un eventuale segnale alto su IN0 produrrà in uscita 0.

Non è strettamente necessario a livello logico però è utile per evitare ripetizione di altri blocchi all'interno del programma.

## Programma

Il programma principale deputato all'interfacciamento I/O è suddiviso in tre parti.

La prima parte riguarda i sensori SNSi che hanno il compito di rilevare il piano corrente dell'ascensore e modificare le variabili indicatrici INDi.



The diagram illustrates a 4-channel encoder interface. On the left, four input channels are shown, each with a normally open contact (IND0, IND1, IND2, IND3) and a normally closed contact (IND0, IND1, IND2, IND3) in parallel with a normally open contact (REQ0, REQ1, REQ2, REQ3). These are connected to the ENCODER1 block. The ENCODER1 block has inputs IN0, IN1, IN2, IN3 and outputs OUT, DATA. The OUT signal is connected to the IN input of the IN\_USINT\_QUEUE0 block. The DATA signal is connected to the DATA input of the IN\_USINT\_QUEUE0 block. The IN\_USINT\_QUEUE0 block has a READY output signal.

Da notare che lo schema non è perfettamente simmetrico poiché il piano terra ha maggiore priorità rispetto agli altri piani. Infatti, IND0 ha valore iniziale TRUE e se non ci sono richieste pendenti il sistema inserisce automaticamente la richiesta per scendere (contatto READY chiuso sul fronte di discesa).

Quando, invece, risulta uguale vengono resettate sia UP che DOWN, mentre viene settata OPEN. Da questo momento parte un TON che garantisce 20s di intervallo tra una richiesta e un'altra.

## SFC

Il principio di funzionamento si basa su due code, una per gestire le prenotazioni durante la fase di salita, e un'altra per gestire le prenotazioni durante la fase di discesa. Lo scopo è quello di soddisfare solo le prenotazioni in salita, quando l'ascensore sale, fino a raggiungere l'ultimo piano in richiesta, e viceversa con le prenotazioni in discesa.

### Variabili globali

Contengono informazioni comuni a tutti i task

#	Nome	Classe	Yipo	Location	Initial Value	Opzione	
1	<u>Richieste_Salita</u>	Global	Richieste				
2	<u>Richieste_Discesa</u>	Global	Richieste				
3	<u>Display</u>	Global	Tastierino				
4	<u>Info</u>	Global	GestionePiani				
5	<u>Flag</u>	Global	BOOL		0		

- Richieste\_Salita (di tipo Richieste), struttura con la coda delle prenotazioni per salire ad un piano superiore
- Richieste\_Discesa (di tipo Richieste), struttura con la coda delle prenotazioni per scendere ad un piano inferiore
- Display (di tipo Tastierino), struttura che contiene i pulsanti del tastierino numerico all'interno dell'ascensore
- Info (di tipo GestionePiani), struttura che contiene le informazioni riguardanti la direzione dell'ascensore e il piano corrente
- Flag, variabile booleana che attiva il task per permettere di prenotare piani all'interno della cabina

### **Richieste**

#	Nome	Yipo	Initial Value
1	TOP	USINT	0
2	DIM	UINT	0
3	BOT	USINT	0
4	QUEUE	ARRAY [0..19] OF SINT	

### **GestionePiani**

#	Nome	Yipo	Initial Value
1	DIR	SINT	0
2	PIANO_CORRENTE	SINT	0

### **Tastierino**

#	Nome	Yipo	Initial Value
1	PULS_0	BOOL	0
2	PULS_1	BOOL	0
3	PULS_2	BOOL	0
4	PULS_3	BOOL	0



Funzioni che agiscono sui dati globali per permettere una gestione univoca degli stessi

### BoubleSort\_Crescente

Riordina in modo crescente una coda in ingresso

#	Nome	Classe	Yipo	Initial Value	Opzione	Documentazione
1	CODA	Input	Richieste			
2	I	Locale	USINT			
3	DUMMY	Locale	SINT			
4	CAMBIO	Locale	BOOL	1		

```

1 IF CODA.DIM > 1 THEN
2   WHILE CAMBIO DO
3     CAMBIO := 0;
4     FOR I:=CODA.BOT TO CODA.TOP-1 BY 1 MOD 20 DO
5       IF CODA.QUEUE[I] > CODA.QUEUE[I+1] THEN
6         DUMMY := CODA.QUEUE[I];
7         CODA.QUEUE[I] := CODA.QUEUE[I+1];
8         CODA.QUEUE[I+1] := DUMMY;
9         CAMBIO := 1;
10      END_IF;
11    END_FOR;
12  END_WHILE;
13  BOUBLESORT_CRESCENTE := CODA;
14 END_IF;
15 RETURN;
16

```

### BoubleSort\_Decrescente

Riordina in modo decrescente una coda in ingresso

#	Nome	Classe	Yipo	Initial Value	Opzione	Documentazione
1	CODA	Input	Richieste			
2	I	Locale	USINT			
3	DUMMY	Locale	SINT			
4	CAMBIO	Locale	BOOL	1		

```

1 IF CODA.DIM > 1 THEN
2   WHILE CAMBIO DO
3     CAMBIO := 0;
4     FOR I:=CODA.BOT TO CODA.TOP-1 BY 1 MOD 20 DO
5       IF CODA.QUEUE[I] < CODA.QUEUE[I+1] THEN
6         DUMMY := CODA.QUEUE[I];
7         CODA.QUEUE[I] := CODA.QUEUE[I+1];
8         CODA.QUEUE[I+1] := DUMMY;
9         CAMBIO := 1;
10      END_IF;
11    END_FOR;
12  END_WHILE;
13  BOUBLESORT_DECRESCENTE := CODA;
14 END_IF;
15 RETURN;
16

```

### Inserisci\_Piano\_Salita

Inserisce un valore nella coda per le prenotazioni in salita

#	Nome	Classe	Yipo	Initial Value	Opzione	Documentazione
1	CODA	Input	Richieste			
2	PIANO	Input	SINT			

```
1 CODA.QUEUE[CODA.TOP] := PIANO;
2 IF CODA.DIM < 20 THEN
3   CODA.TOP := (CODA.TOP + 1) MOD 20;
4   CODA.DIM := CODA.DIM + 1;
5   CODA := BOUBLESORT_CRESCENTE(CODA);
6 END_IF;
7 INSERISCI_PIANO_SALITA := CODA;
8 RETURN;
9
10
```

### Inserisci\_Piano\_Discesa

Inserisce un valore nella coda per le prenotazioni in discesa

#	Nome	Classe	Yipo	Initial Value	Opzione	Documentazione
1	CODA	Input	Richieste			
2	PIANO	Input	SINT			

```
1 CODA.QUEUE[CODA.TOP] := PIANO;
2 IF CODA.DIM < 20 THEN
3   CODA.TOP := (CODA.TOP + 1) MOD 20;
4   CODA.DIM := CODA.DIM + 1;
5   CODA := BOUBLESORT_DECRESCENTE(CODA);
6 END_IF;
7 INSERISCI_PIANO_DISCESA := CODA;
8 RETURN;
```

### Preleva\_Piano

Preleva un piano da una qualsiasi coda

#	Nome	Classe	Yipo	Initial Value	Opzione	Documentazione
1	CODA	Input	Richieste			

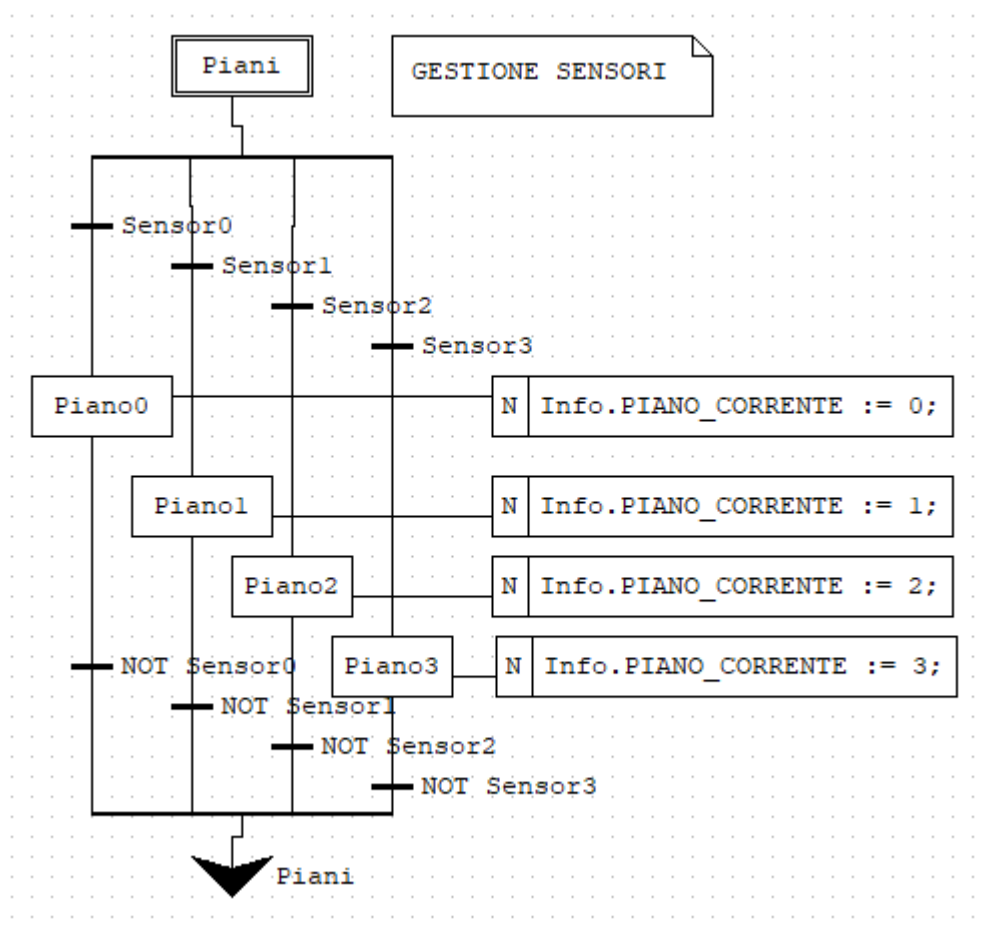
```
1 CODA.DIM := CODA.DIM - 1;
2 CODA.BOT := (CODA.BOT + 1) MOD 20;
3 PRELEVA_PIANO := CODA;
4 RETURN;
```

### Sensori\_Piani

Task periodico che si attiva ogni 20ms.

Si occupa di settare una variabile globale che mantiene memorizzato il piano a cui l'ascensore si trova (o l'ultimo a cui si è fermato, se in movimento).

#	Nome	Classe	Yipo	Location	Initial Value	Opzione	Documentaz
1	Info	External	GestionePiani				Informazioni
2	Display	External	Tastierino				Display interno
3	Sensor0	Locale	BOOL		1		Piano 0
4	Sensor1	Locale	BOOL		0		Piano 1
5	Sensor2	Locale	BOOL		0		Piano 2
6	Sensor3	Locale	BOOL		0		Piano 3



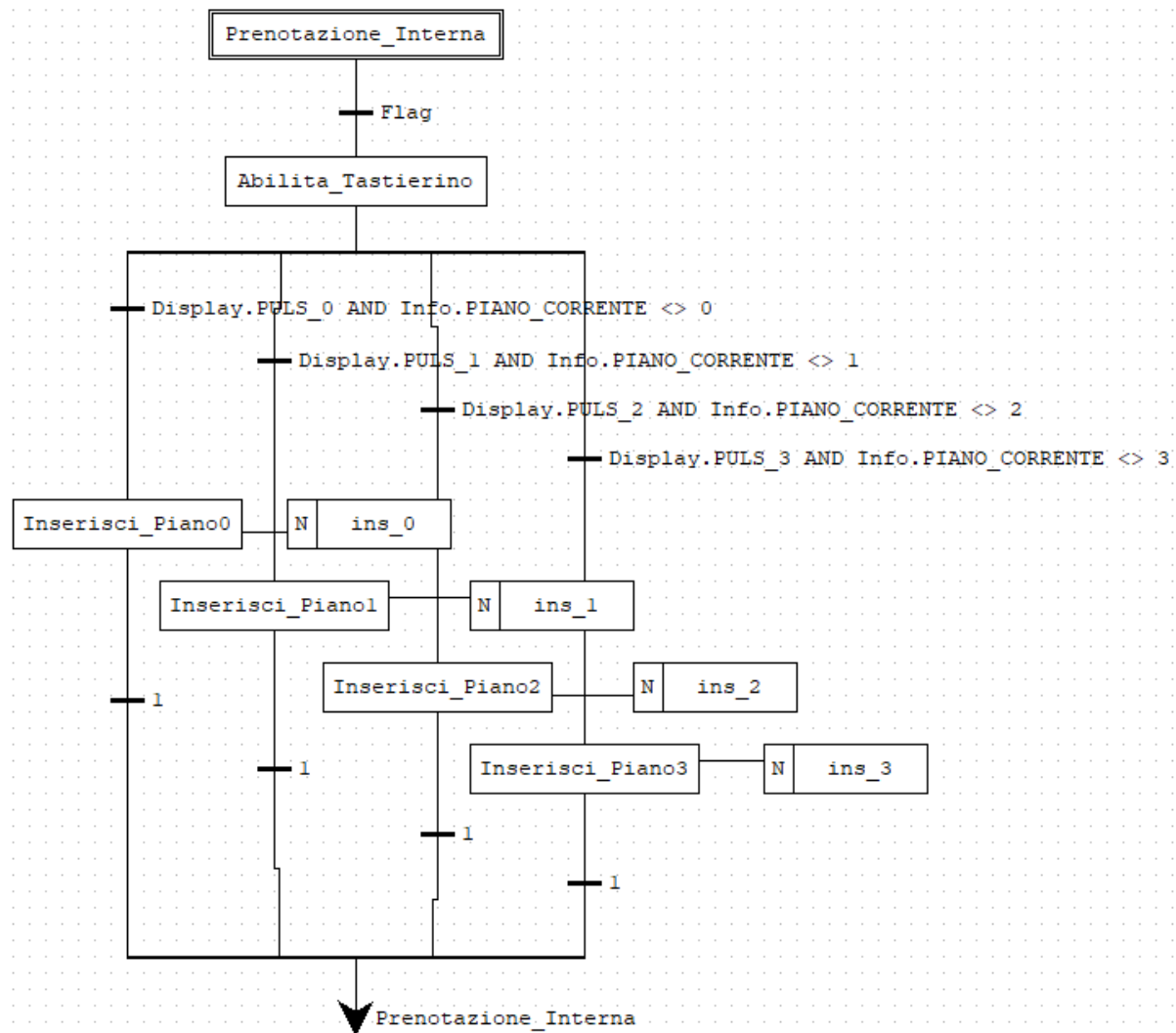
### Prenotazioni\_Interne

Task periodico che si attiva ogni 20ms.

Si occupa di inserire un piano all'interno della coda delle prenotazioni quando l'ascensore è fermo ad un piano e si digita un numero sul tastierino. L'algoritmo prevede di aggiungere un piano alla coda di salita se il piano da raggiungere è superiore al piano corrente e viceversa. All'atto dell'inserimento la coda viene anche ordinata in modo da raggiungere prima il piano più vicino e poi quello più lontano, durante il suo movimento.

Utilizza solo le variabili globali mostrate in precedenza.

#	Nome	Classe	Yipo	Location	Initial Value	Opzione	Docum
1	Flag	External	BOOL				
2	Richieste_Salita	External	Richieste				
3	Richieste_Discisa	External	Richieste				
4	Display	External	Tastierino				
5	Info	External	GestionePiani				



Ins\_0

```

1 Richieste_Discesa := INSERISCI_PIANO_DISCESA(Richieste_Discesa,0);
2 |

```

Ins\_1

```

1 IF Info.PIANO_CORRENTE < 1 THEN
2   Richieste_Salita := INSERISCI_PIANO_SALITA(Richieste_Salita,1);
3 ELSE
4   Richieste_Discesa := INSERISCI_PIANO_DISCESA(Richieste_Discesa,1);
5 END_IF;

```

Ins\_2

```

1 IF Info.PIANO_CORRENTE < 2 THEN
2   Richieste_Salita := INSERISCI_PIANO_SALITA(Richieste_Salita,2);
3 ELSE
4   Richieste_Discesa := INSERISCI_PIANO_DISCESA(Richieste_Discesa,2);
5 END_IF;

```

Ins\_3

```

1 Richieste_Salita := INSERISCI_PIANO_SALITA(Richieste_Salita,3);

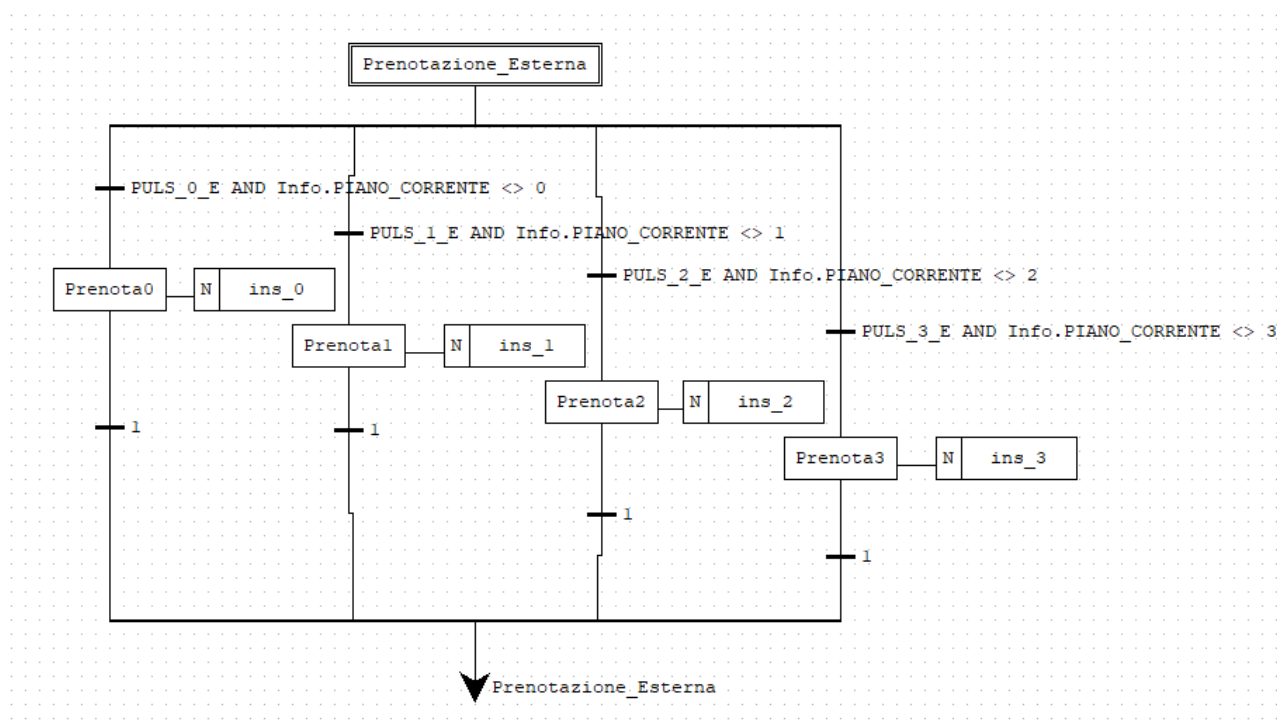
```

## Prenotazioni\_Esterne

Task periodico che si attiva ogni 20ms.

Si occupa di inserire un piano all'interno delle code premendo pulsanti dall'esterno. Se l'ascensore è in salita e si vuole raggiungere un piano superiore a dove ci si trova, allora il piano in questione viene aggiunto alla coda delle salite, altrimenti alla coda delle discese.

#	Nome	Classe	Yipo	Location	Initial Value	Opzione	Do
1	PULS_0_E	Locale	BOOL				
2	Info	External	GestionePiani				
3	Richieste_Salita	External	Richieste				
4	Richieste_Discesa	External	Richieste				
5	PULS_1_E	Locale	BOOL				
6	PULS_2_E	Locale	BOOL				
7	PULS_3_E	Locale	BOOL				



*Ins\_0*

```
1 Richieste_Discesa := INSERISCI_PIANO_DISCESA(Richieste_Discesa,0);
2 |
```

*Ins\_1*

```
1 IF Info.PIANO_CORRENTE < 1 THEN
2 Richieste_Salita := INSERISCI_PIANO_SALITA(Richieste_Salita,1);
3 ELSE
4 Richieste_Discesa := INSERISCI_PIANO_DISCESA(Richieste_Discesa,1);
5 END_IF;
```

*Ins\_2*

```
1 IF Info.PIANO_CORRENTE < 2 THEN
2 Richieste_Salita := INSERISCI_PIANO_SALITA(Richieste_Salita,2);
3 ELSE
4 Richieste_Discesa := INSERISCI_PIANO_DISCESA(Richieste_Discesa,2);
5 END_IF;
```

*Ins\_3*

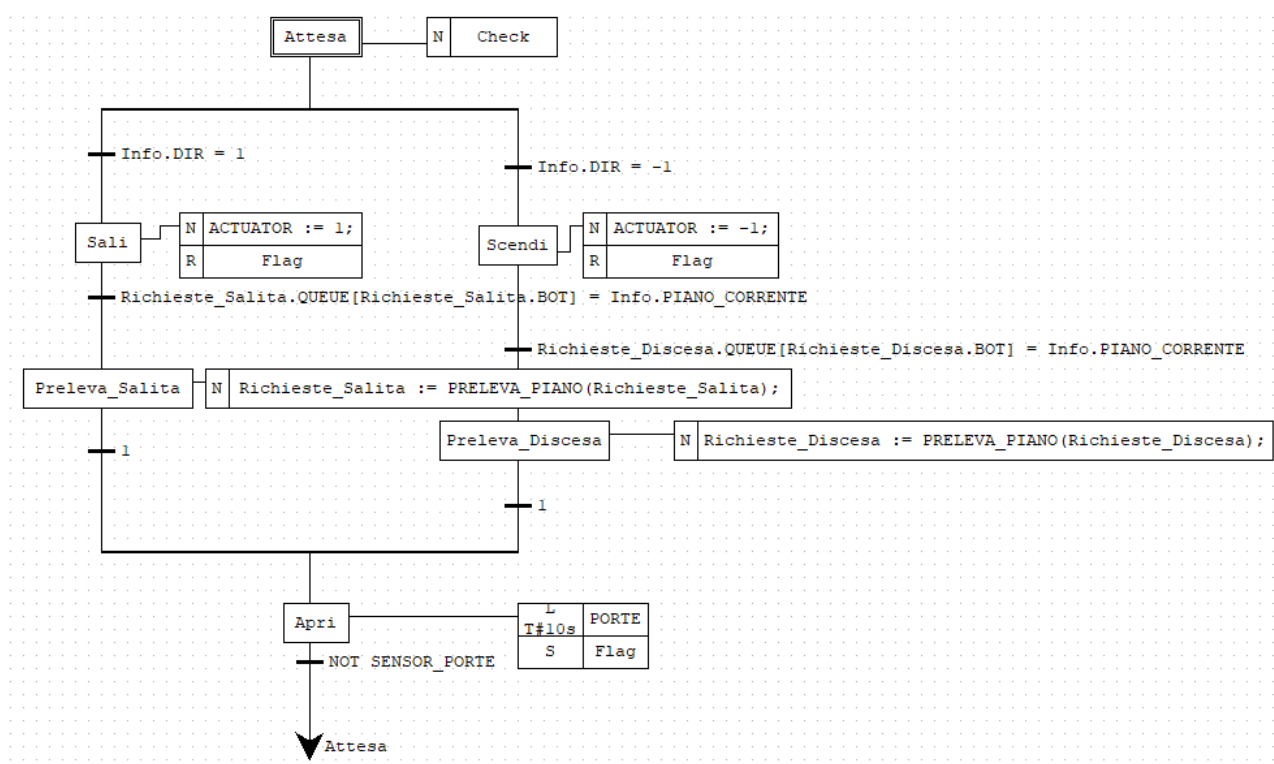
```
1 Richieste_Salita := INSERISCI_PIANO_SALITA(Richieste_Salita,3);
```

## Gestione\_Motore

Task periodico che si attiva ogni 20ms.

Si occupa di selezionare la rotazione del motore, per far salire o scendere l'ascensore. Quando viene eseguito controlla, attraverso un'azione di *check* la rotazione del motore. Una volta selezionata la rotazione, raggiunge il primo piano utile in una delle due code di gestione. Successivamente preleva il piano della coda e così via, fin quando non vengono soddisfatte tutte le richieste.

#	Nome	Classe	Yipo	Location	Initial Value	Opzione	Docu
1	ACTUATOR	Locale	SINT		0		
2	PORTE	Locale	BOOL		0		
3	SENSOR_PORTE	Locale	BOOL		0		
4	Flag	External	BOOL				
5	Info	External	GestionePiani				
6	Richieste_Salita	External	Richieste				
7	Richieste_Discesa	External	Richieste				



## Check

```

1 WHILE Info.DIR = 0 DO
2   IF Info.DIR = 1 AND Richieste_Salita.DIM <> 0 THEN
3     Info.DIR := 1;
4   ELSIF Info.DIR = -1 AND Richieste_Discesa.DIM <> 0 THEN
5     Info.DIR := -1;
6   ELSIF Richieste_Salita.DIM <> 0 THEN
7     Info.DIR := 1;
8   ELSIF Richieste_Discesa.DIM <> 0 THEN
9     Info.DIR := -1;
10  ELSE
11    Info.DIR := 0;
12  END_IF;
13 END_WHILE;
  
```

## PID

Per la parte di programma dedicata al controllo modulante è stato scelto un regolatore PID con taratura basata sulle formule di inversione.

## Modellistica

L'impianto selezionato è una semplice superficie chiusa (nel caso specifico la cabina di un ascensore) di cui si vuole regolare la temperatura.

L'equazione differenziale lineare a coefficienti costanti per ricavare il modello ISU è quella standard per un sistema che scambia calore con l'ambiente esterno.

$$C \dot{T} = q - k (T - T_u)$$

Tenendo conto delle costanti tipiche dell'aria e sapendo che la cabina ha dimensioni 2.00x2.00x3.00 m<sup>3</sup> è facile ricavare i parametri C (capacità termica) e k (coefficiente di scambio termico) dell'equazione. Ovviamente lo scambio di calore avviene su una parete laterale per qualche secondo.

```
C_th = 15000; %Capacità termica aria in una cabina
K_th = 600;   %Coefficiente di scambio termico

A = -K_th/C_th;
B = [1 K_th/C_th];
C = 1;
D = [0 0];
```

In MATLAB dalle precedenti considerazioni ricaviamo il sistema ISU e la funzione di trasferimento rispetto all'ingresso controllabile.

```
Gs = tf(ss(A,B,C,0));
Gs = Gs(1);
```

## Controllo

A questo punto bisogna chiudere in retroazione l'impianto sfruttando la conoscenza dell'uscita, ovvero la temperatura interna.

Avendo a disposizione il modello matematico del sistema è possibile ricorrere alla taratura con le formule d'inversione. Pertanto, si impone il passaggio della funzione ad anello aperto rispetto alla circonferenza di raggio unitario per una pulsazione desiderata e con un certo margine di fase. Il sistema è di tipo passabasso, quindi è certo che la pulsazione Wg sarà unica.

```
Wg = 1;
PM = 120;
Mg = 1/abs(evalfr(Gs, 1i*Wg));
Pg = PM - 180 - rad2deg(angle(evalfr(Gs, 1i*Wg)));
```

Mg e Pg rappresentano modulo e fase della risposta armonica del PID in Wg, quindi è banale dedurre che essi serviranno per ricavare i guadagni delle tre azioni di controllo. Bisogna notare inoltre che una taratura del genere lascia un grado di libertà, ciò vuol dire che o Ki o Kd vanno assegnate in base a risultati su prove ripetute. Una buona scelta consiste nel prendere Ki almeno un ordine di grandezza inferiore a Kp e imporre un margine di fase consistente.

```
Kp = Mg*cosd(Pg);
Ki = Kp/25;
Kd = (Ki+Mg*sind(Pg)*Wg)/Wg^2;

s = tf('s');
Ks = Kp + Ki/s + Kd*s;
Fs = Ks*Gs;
margin(Fs);
```

La battitura del comando margin è mirata alla verifica del passaggio imposto che risulta corretto in quanto le formule di inversione a differenza di altri metodi di taratura sono analitici per definizione.

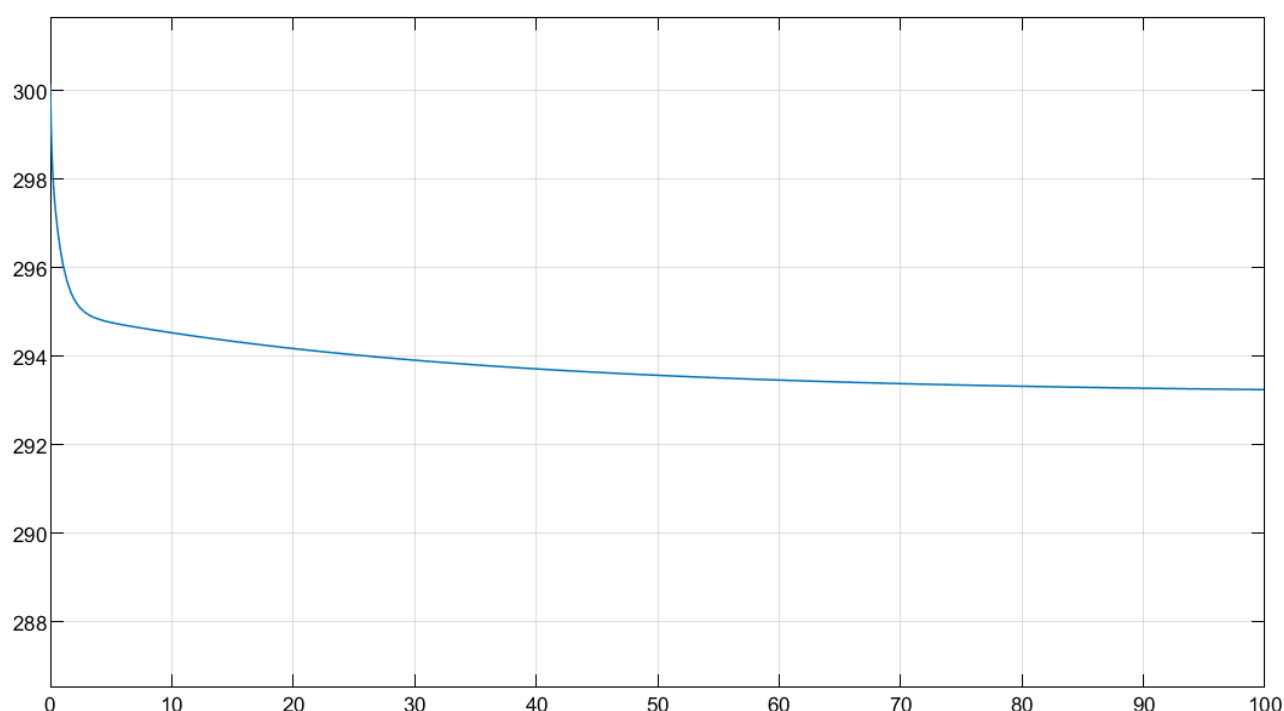
Il sistema controllato ora va simulato tramite il programma Simulink.



La temperatura esterna influisce sul sistema in maniera ovviamente non controllabile. In verità essa viene pesata tramite il vettore B per un coefficiente che la rende abbastanza trascurabile per il controllo SISO (in questo progetto si è assunto che la cabina rimanga aperta per pochi secondi e le dimensioni siano inferiori rispetto a un tipico ambiente per una persona).

Per la simulazione si è scelto di usare un set-point a gradino di 293.15 K, una condizione iniziale di 300.15 K e si è assunta una temperatura esterna pari a quest'ultima.

Per verificare l'efficacia del controllo si valuta la risposta del sistema.



Come si può osservare il sistema evolve lentamente trattandosi di un sistema termico e il controllo non è molto spinto. Infatti, il tempo di assestamento è dell'ordine dei minuti e l'andamento è abbastanza dolce (non è preferibile avere una risposta più rapida ma anche più oscillatoria in quanto escursioni termiche troppo pronunciate risulterebbero banalmente pericolose).

Un'ultima considerazione sulla taratura: la scelta delle formule di inversione si è resa necessaria poiché il sistema è del primo ordine. Un' eventuale approccio Z&N a ciclo chiuso non è considerabile, mentre per quello ad anello aperto è necessario usare un espediente non ragionevole: bisogna introdurre un ritardo matematico infinitesimo per poter usufruire delle tabelle a disposizione.