POLITECNICO DI TORINO

Facoltà di Ingegneria Meccanica

Corso di Laurea Magistrale in Ingegneria Meccanica
Indirizzo Automazione

Tesi di Laurea Magistrale

Pendolo inverso con controllo Fuzzy implementato su scheda Arduino

Relatore:
Prof. Terenziano Raparelli
Co-relatori:
Prof. Luigi Mazza, Prof. Federico Colombo

Candidato:
Marco Ghia

Luglio 2020

Indice

Indice I
Indice figure III
Indice tabelle VII
Introduzione 1
1 Controllo Fuzzy 2
1.1 Schema controllo Fuzzy 3
1.1.1 Fuzzificazione 3
1.1.2 Regole 4
1.1.3 Defuzzificazione 7
1.2 Funzione di approssimazione fuzzy 7
1.3 Metodo di interferenza di Sugeno 8
1.4 Vantaggi e limiti 8
2 Descrizione Attrezzatura 9
2.1 L'attuatore pneumatico 9
2.2 Valvola 10
2.3 Trasduttore angolare 13
2.4 Trasduttore di posizione LVDT 13
2.5 Guida lineare a ricircolo di sfere 13
2.6 Gruppo carrello-pendolo 14
2.7 Arduino Mega 2560 15
2.8 PCB 17
3 Modello Analitico 18
4 Modello Simulink 23
5 Controlli Fuzzy 30
5.1 Controllo fuzzy lineare e non lineare 30
5.2 Taratura controllo fuzzy PID 35
6 Controlli Fuzzy PID 36
6.1 PD+I Fuzzy 37
6.1.1 Taratura del controllore 38
6.1.2 Tuning dei guadagni 39
6.2 PID-like Fuzzy 46
6.2.1 Taratura del controllore 47

6.2.2 Tuning dei guadagni ... 48
6.3 PD+PI Fuzzy ... 54
6.3.1 Taratura del controllore ... 55
6.3.2 Tuning dei guadagni ... 56
6.4 Confronto controllori fuzzy ... 60
6.5 Confronto controlli fuzzy ... 63
7 Simulazione scheda Arduino ... 66
7.1 Nozioni base di Arduino ... 70
7.2 Implementazione libreria Fuzzy su Arduino ... 72
7.3 Verifica libreria fuzzy ... 76
7.4 Simulazione della libreria agendo manualmente sugli input ... 80
7.5 Programmazione scheda Arduino ... 87
7.5.1 Controllore PD+I ... 87
7.5.2 Controllore PID-like ... 89
7.5.3 Controllore PD+PI ... 90
8 Comunicazione Arduino - Matlab & Simulink ... 92
8.1 Conversione Fuzzy-Tool Matlab ... 96
8.2 S-Function Builder ... 100
8.3 Controllore PD+I con S-Function Builder ... 108
9 Pendolo inverso in Simscape-Multibody ... 113
10 Conclusioni ... 123
11 Appendice ... 125
11.1 Codice Arduino verifica libreria fuzzy ... 125
11.2 Codice Arduino per agire manualmente sugli input ... 128
11.3 codice Matlab per simulazione PD+I ... 130
11.4 codice Matlab per simulazione PD+PI ... 133
11.5 codice Matlab per simulazione PID-like ... 133
11.6 codice Matlab per confronto controllori ... 134
11.7 codice Arduino PD+I ... 135
11.8 codice Arduino PID-like ... 139
11.9 codice Arduino PD+PI ... 143
11.10 codice Matlab per il controllo PD+I da caricare sulla scheda Arduino ... 147
11.11 File Solid in Simscape-Multibody con sistemi riferimento ... 149
12 Bibliografia ... 151

Indice figure

Figura 1-1: forma funzioni di appartenenza 2
Figura 1-2: nucleo e supporto fuzzy set trapezoidale 2
Figura 1-3: fuzzy set delle variabili 3
Figura 1-4: regole fuzzy 4
Figura 1-5: clipping e scaling 5
Figura 1-8: superficie di controllo fuzzy 7
Figura 2-1: schema sistema 9
Figura 2-2: attuatore Camozzi 24N2A16A500 9
Figura 2-3: velocità massime pistone 10
Figura 2-4: disegni valvola 11
Figura 2-5: valvola Camozzi AP-7211-LR2-U7 e schema ISO 11
Figura 2-6: portata massima valvola in funzione pressione taratura 12
Figura 2-7: conversione PWM 12
Figura 2-8: Minipattino (sx) e Minirotaia (dx) 14
Figura 2-9: cinematismi 14
Figura 2-10: pendolo e struttura del carrello 15
Figura 2-11: Arduino Mega 2560 16
Figura 3-1: versi convenzionali variabili cinematiche 18
Figura 3-2: diagrammi di corpo libero 18
Figura 3-3: riferimenti camere attuatore 19
Figura 3-4: pressioni e forze sull'attuatore 19
Figura 3-5: convenzione pressione nelle valvole 21
Figura 3-6: portata nella valvola ISO 6358 21
Figura 4-1: schema sistema 23
Figura 4-2: schema pneumatico 23
Figura 4-3: schema valvole 24
Figura 4-4: collegamenti valvole-attuatore 24
Figura 4-5: flusso nella valvola 25
Figura 4-6: schema attuatore pneumatico 26
Figura 4-7: continuità nella camera A 26
Figura 4-8: continuità nella camera B 27
Figura 4-9:reazione stantuffo-camera 28
Figura 4-10: equilibrio orizzontale 29
Figura 4-11: equilibrio alla rotazione 29
Figura 5-1: Fuzzy Logic Controller 30
Figura 5-2: fuzzy set input lineare 31
Figura 5-3: superficie fuzzy lineare 31
Figura 5-4: fuzzy set output 32
Figura 5-5: verifica linearità 33
Figura 5-6: linearità con “de”=0 33
Figura 5-7: linearità con “e”=0 34
Figura 5-8: linearità con “e” e “de” diversi da zero 34
Figura 5-9: es. fuzzy PID 35
Figura 6-1: PD+I 37
Figura 6-2: blocco controllo PD+I 37

Figura 6-3: controllo PD+I sull'errore posizione 37
Figura 6-4: controllo PD+I sull'angolo teta 38
Figura 6-5: guadagno proporzionale, PD+I 39
Figura 6-6: guadagno derivativo, PD+I 40
Figura 6-7: zoom guadagno derivativo 40
Figura 6-8: set e fb, PD+I 41
Figura 6-9: pressione e forze, PD+I 42
Figura 6-10: flussi in massa, PD+I 42
Figura 6-11: altro gradino, PD+I 43
Figura 6-12: sinusoide, PD+I 43
Figura 6-13, flussi con sinusoide, PD+I 44
Figura 6-14: p e F con sinusoide, PD+I 44
Figura 6-15: disturbo, PD+I 45
Figura 6-16: PID-like 46
Figura 6-17: blocco controllo PD+I 46
Figura 6-18: controllo PID-like sull'errore di posizione 46
Figura 6-19: controllo PID-like sull'angolo teta 47
Figura 6-20: guadagno proporzionale, PD+PI 49
Figura 6-21: zoom guadagno derivativo 49
Figura 6-22: guadagno derivativo, PD+PI 50
Figura 6-23: set e fb, PD+PI 51
Figura 6-24: altro gradino PD+PI 51
Figura 6-25: sinusoide, PD+PI 52
Figura 6-26: disturbo, PD+PI 53
Figura 6-27: PD+PI 54
Figura 6-28: blocco controllo PD+PI 54
Figura 6-29: controllo PD+PI sull'errore di posizione 54
Figura 6-30: controllo PD+PI sull'angolo teta 55
Figura 6-31: set e fb, PID like 57
Figura 6-32: sinusoide, PID like 58
Figura 6-33: altro gradino, PID like 58
Figura 6-34: disturbo PID like 59
Figura 6-35: confronto gradino 0.1m 61
Figura 6-36: confronto gradino 0.05m 61
Figura 6-37; confronto gradino 0.05 m 64
Figura 6-38: confronto gradino 0.1 m 64
Figura 6-39: confronto sin 65
Figura 7-1: aggiungere Device 66
Figura 7-2: Virtual Terminal 67
Figura 7-3:confronto schede Arduino Uno 67
Figura 7-4: mostra Compilazione Arduino 68
Figura 7-5: Edit Component scheda Arduino 68
Figura 7-6: Virtual Terminal 69
Figura 7-7: acquisizione dati 69
Figura 7-8: sezioni codici Arduino 70
Figura 7-9: comandi Arduino IDE 71
Figura 7-10: scheda e porta di comunicazione 71
Figura 7-11: contenuto libreria Fuzzy.h 72

Figura 7-12: caricamento libreria 72
Figura 7-13: es. funzioni trapezoidali 73
Figura 7-14: es. intestazione dati 77
Figura 7-15: sim. Variando: e 78
Figura 7-16: sim. variando: de 78
Figura 7-17: confronto libreria Arduino con tool Matlab 79
Figura 7-18: sim. variando: e, de 79
Figura 7-19: simulazione input e output 80
Figura 7-20:generatore Dc 80
Figura 7-21: voltmetro 81
Figura 7-22: posizionamenti del potenziometro 81
Figura 7-23: potenziometro 82
Figura 7-24: pin output digitali abilitati al PWM 83
Figura 7-25: time1 84
Figura 7-26: time3 85
Figura 7-27: time2 85
Figura 7-28: fuzzy tool e=70.09 de=-30.01 86
Figura 7-29: time3 86
Figura 7-30:controllo PD+I sull'errore posizione 87
Figura 7-31: controllo PID-like sull'errore di posizione 89
Figura 7-32: controllo PD+PI sull'errore di posizione 90
Figura 8-1: estensioni Arduino Hardware 92
Figura 8-2: solutore a passo fisso 93
Figura 8-3: Hardware Implementation 94
Figura 8-4: Data Import/Export 94
Figura 8-5: Host-board connection 95
Figura 8-6: Deploy to Hardware 95
Figura 8-7: sim Fuzzy-Tool Matlab 96
Figura 8-8: file iniziali 96
Figura 8-9: code generation report 97
Figura 8-10: file creati dalla conversione 97
Figura 8-11: confronto sim su PC e su scheda Arduino 98
Figura 8-12: controllo Fuzzy x e teta 98
Figura 8-13: errore saturazione memoria 99
Figura 8-14: librerie fuzzy 100
Figura 8-15: S-Function Builder set 1 101
Figura 8-16: S-Function Builder set 2 101
Figura 8-17: S-Function Builder set 3 102
Figura 8-18: S-Function Builder set 4 102
Figura 8-19: Includes 103
Figura 8-20: S-Function Builder set 5 103
Figura 8-21: conversione void setup ( ) 104
Figura 8-22: S-Function Builder set 6 104
Figura 8-23: S-Function Builder set 7 105
Figura 8-24: file creati da S-Function Builder 105
Figura 8-25: aggiunta extern "C" 106
Figura 8-26: file Simulink controllo 106
Figura 8-27: memoria occupata con S-Function Builder 107

Figura 8-28: S-Function Builder vs Fuzzy-Tool Matlab 107
Figura 8-29: controllo PD+I 108
Figura 8-30: controllo x 108
Figura 8-31: controllo teta 108
Figura 8-32: memoria occupata dal controllo PD+I 109
Figura 8-33: libreria Simulink Support Package for Arduino Hardware 109
Figura 8-34: input analogico 110
Figura 8-35: output PWM 110
Figura 8-36: mappa dei pin 110
Figura 8-37: modello implementazione sul banco 111
Figura 8-38: conversione sensore x 111
Figura 8-39: conversione sensore teta 112
Figura 8-40: suddivisione del segnale tra le valvole 112
Figura 9-1: sottosistema meccanico Simscape-Multibody 113
Figura 9-2: blocchi di partenza 113
Figura 9-3: assegnazione frame. 114
Figura 9-4: File Solid, Carrello 114
Figura 9-5: n. collegamenti= n. frame 115
Figura 9-6: collegamento rigido 115
Figura 9-7: collegamento con guida prismatica 115
Figura 9-8: Rigid Trasform 115
Figura 9-9: giunto prismatico 116
Figura 9-10: conversione Simulink-Simscape 116
Figura 9-11: giunto rotoidale 117
Figura 9-12: rotazione e traslazione 117
Figura 9-13: Mechanics Explorer 118
Figura 9-14: gradino 0.05m 119
Figura 9-15: gradino 0.1m 119
Figura 9-16: disturbo 120
Figura 9-17: sinusoide 120
Figura 9-18: frame video 121
Figura 9-19: eq. pendolo 122
Figura 9-20: teorema Huygens Steiner 122

Indice tabelle

Tabella 1-1: regole fuzzy 5
Tabella 2-1: parametri attuatore 10
Tabella 2-2: parametri valvola 12
Tabella 2-3: parametri trasduttore angolare 13
Tabella 2-4: parametri trasduttore LVDT 13
Tabella 2-5: parametri Arduino Mega 2560 16
Tabella 3-1: valori parametri 22
Tabella 5-1: regole fuzzy lineare 31
Tabella 5-2: regole fuzzy non lineari 32
Tabella 6-1: controllori PID 36
Tabella 6-2: parametri controllore PD+I 39
Tabella 6-3: tuning PD+I 41
Tabella 6-4: parametri controllore PID-like 48
Tabella 6-5: tuning PD+PI 50
Tabella 6-6: parametri controllore PD+PI 56
Tabella 6-7: tuning, PID like 57
Tabella 6-8: fuzzy set fc2_4 63
Tabella 6-9: fuzzy rule fc2_4 63

Introduzione

Lo scopo principale di questa tesi è il controllo del pendolo inverso azionato mediante un attuatore pneumatico, ovvero un sistema altamente instabile. Per il controllo del sistema si è scelto di usare una scheda Arduino, ovvero un controllore a basso costo, implementando una logica fuzzy.

Alla luce dell'emergenza sanitaria, che ha portato alla chiusura dei laboratori didattici del Politecnico di Torino, per lo sviluppo di un controllo si è sfruttato un modello Simulink del banco prova fornito da un precedente tesista [1], in cui si sono provati a sviluppare tre controllori fuzzy differenti.

Per il controllo del pendolo si è scelta l'implementazione della logica fuzzy (la cui traduzione letterale è "sfocata"), che non è adatta a problemi di elevata precisione, mentre funziona bene in presenza di margini d'incertezza. Va evidenziato che in molte applicazioni, come nel pendolo inverso, non è necessaria un'elevata precisione, viene infatti sfruttato in controlli di automobili, elettrodomestici, messa a fuoco fotocamere, ovvero in quelle applicazioni dove non è presente una logica binaria, ma si sfrutta una logica più graduale "sfocata" per l'attuazione [2].

Completata la taratura dei controllori fuzzy in Simulink, quindi simulando anche il controllore della scheda Arduino, si è proceduto con lo sviluppo dei programmi dei soli controllori in Arduino IDE per una possibile futura implementazione sulla scheda del banco prova.

Per valutare l'effettivo funzionamento dei programmi sviluppati per la scheda Arduino si è proceduto utilizzando Proteus, un software di simulazione di circuiti elettronici che, mediante la implementazione di librerie esterne Arduino, consente la simulazione dei programmi scritti in Arduino IDE, il cui funzionamento non era prima verificabile in assenza della scheda.

Si è scelto successivamente di acquistare la scheda Arduino Mega 2560, la medesima del banco, per verificare il corretto funzionamento del simulatore su Proteus. La motivazione principale dell'acquisto è stata la possibilità di programmare la scheda Arduino sfruttando le estensioni di Matlab: MATLAB Support Package for Arduino Hardware e Simulink Support Package for Arduino Hardware, che consentono di sfruttare i vantaggi di un linguaggio di alto livello e di un post processing più semplice e veloce rispetto all'utilizzo di Arduino IDE.

In conclusione, si è usato Simscape-Multibody per simulare le equazioni di equilibrio del pendolo inverso e per ridurre il grado di approssimazione del modello utilizzato.

1 Controllo Fuzzy

In Matlab è disponibile un tool per lavorare in modo intuitivo con la logica fuzzy: Fuzzy Logic Designer, tramite il quale verranno realizzate le seguenti analisi. Le principali nozioni di questa logica di funzionamento sono state apprese da [1] e [2], del quale vengono di seguito riportati i principi fondamentali.

La traduzione di fuzzy è sfumata/sfocata, ovvero è una logica non binaria. Come obiettivo si pone quello di eliminare le discontinuità tipiche di un controllo binario, tramite i fuzzy set o classi di appartenenza, il cui grado di appartenenza o membership μ può variare tra 0 e 1. La forma della funzione di appartenenza determina la membership μ e tra le forme più comuni ci sono quelle: triangolari, trapezoidali e gaussiane.

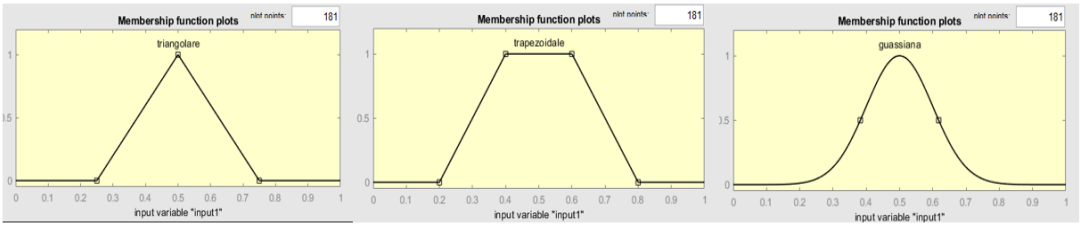


Figura 1-1: forma funzioni di appartenenza

Per quanto riguarda le funzioni di appartenenza trapezoidali si possono individuare un nucleo ed un supporto. La specificità è in funzione della estensione del nucleo rispetto al supporto, mentre la precisione è legata alla pendenza del lato obliquo, maggiore è la pendenza maggiore è la precisione.

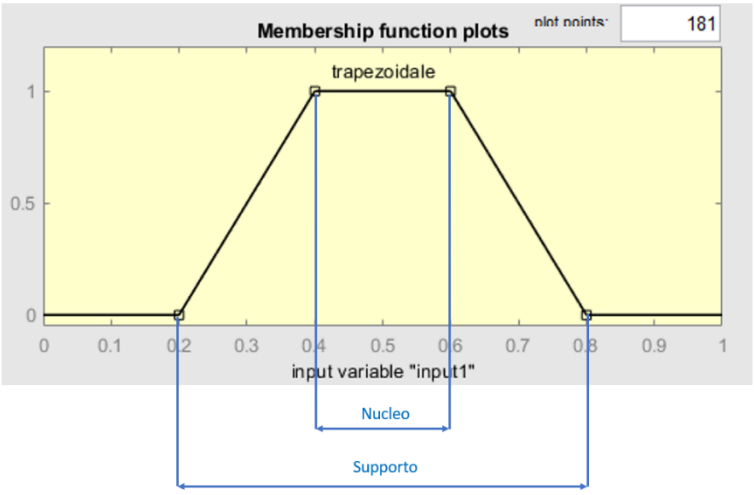


Figura 1-2: nucleo e supporto fuzzy set trapezoidale

1.1 Schema controllo Fuzzy

Lo schema di un controllo fuzzy è costituito da tre fasi: fuzzificazione, applicazione delle regole e defazzificazione. In figura viene riportata la schermata principale del Fuzzy tool di Mattlab.

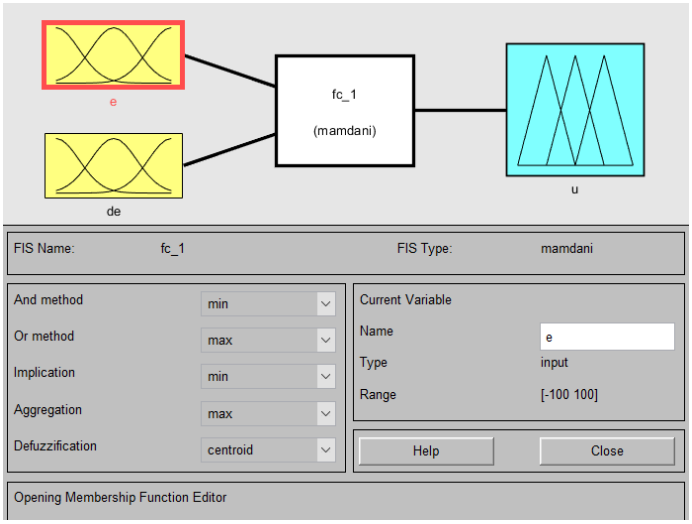


Figura 1-3: es. di un controllo fuzzy con fuzzy tool

1.1.1 Fuzzificazione

Nella fase iniziale di fuzzificazione delle variabili, come primo passo si adimensionalizzano gli ingressi. È necessario assegnare un fuzzy set alle variabili, un esempio comune può essere NN (molto negativo), N (negativo), Z (zero), P(positivo), PP (molto positivo).

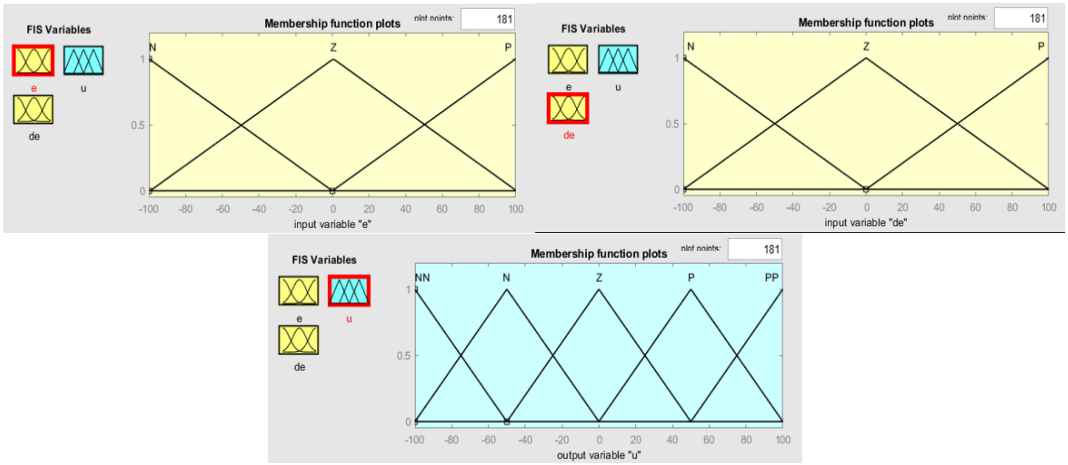


Figura 1-3: fuzzy set delle variabili

È opportuno che le funzioni si sovrappongano, in modo che per ogni valore di ingresso vengano attivate almeno due regole. Quindi, si definisce la completezza di un fuzzy set, l'ordinata dell'intersezioni maggiore. Affinché esista sempre una funzione prevalente, viene suggerito un valore di circa 0.5, come mostrato da [3].

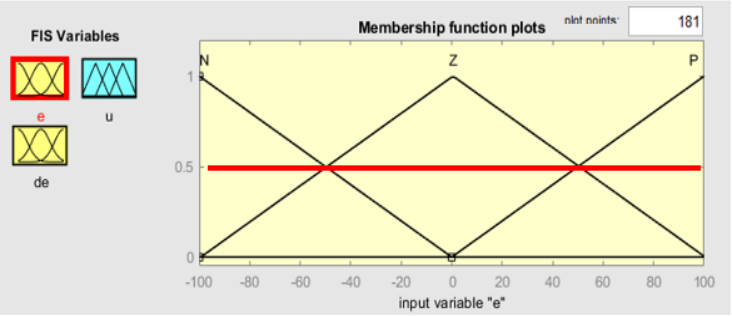


Figura 1-4: es. completezza=0.5

1.1.2 Regole

È necessario definire un set di regole che leghino tra loro ingressi e uscite, con la seguente struttura: SE (condizione) ALLORA (azione). Il metodo comunemente usato è quello di Mamdani. Viene mostrato in figura un esempio di applicazione di regole nel tool di Matlab.

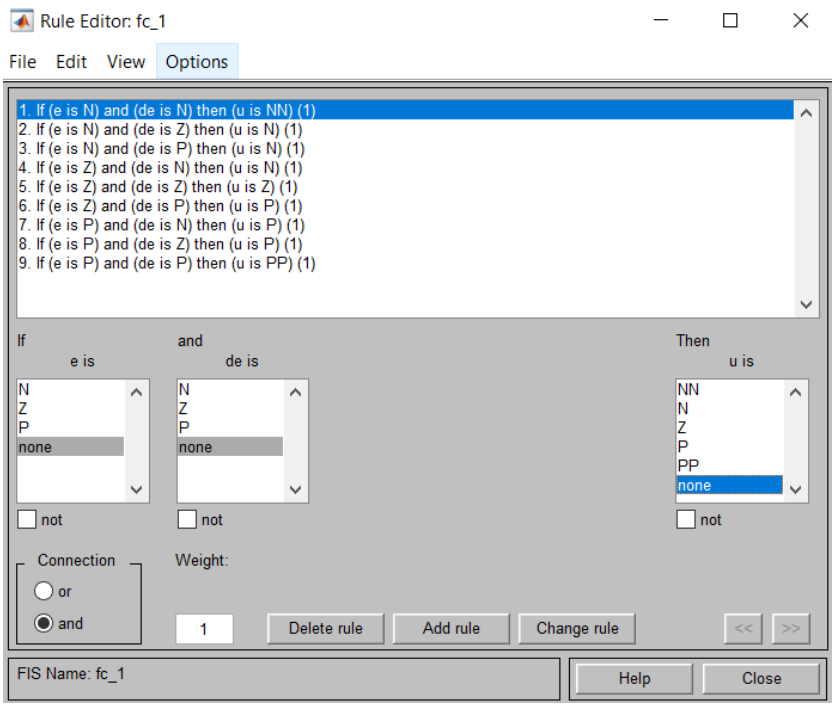


Figura 1-4: regole fuzzy

Un'altra rappresentazione delle regole nel caso di due ingressi e un'uscita può essere con la matrice delle regole di Mac Vicar-Whelan.

fc_1		de		
		N	Z	P
e	N	NN	N	N
	Z	N	Z	P
	P	P	P	PP

Tabella 1-1: regole fuzzy

Come calcolare i gradi di attivazione dei fuzzy set di output:

1. Si calcolano i gradi di attivazione degli input
2. Si fa l'intersezione di insiemi fuzzy per ogni cella della matrice generata:

$$\mu_z(e_i, \dot{e}_j) = min\left(\mu_i(e), \mu_j(\dot{e})\right)$$

3. Si fa l'unione di insiemi fuzzy per calcolare il grado di attivazione di ogni classe d'uscita, ovvero prendo il massimo a parità di funzione in tutta la matrice:

$$\mu(u_k) = max\left(\mu_z(e_i, \dot{e}_j)\right) = max\left(min\left(\mu_i(e), \mu_j(\dot{e})\right)\right)$$

Quindi, per determinare il valore di μ in ogni cella della matrice, valuto il minimo tra i due input corrispondenti, successivamente per determinare il grado di attivazione della classe di uscita di tutte le celle che presentano la medesima classe, scelgo quella che ha il valore massimo.

Per rappresentare l'uscita *u* ci sono due metodi:

1. Clipping o alpha-cut: è una limitazione della funzione di appartenenza al grado di attivazione (in figura 0.2), questo metodo è più diffuso per la semplicità, ma perdo un'informazione nel taglio sulla forma della funzione (es. passaggio da una funzione triangolare ad una trapezoidale).
2. Scaling: è una riduzione della funzione di appartenenza al grado di attivazione.

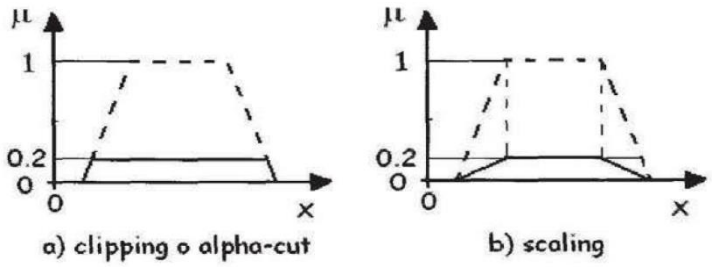


Figura 1-5: clipping e scaling

In figura abbiamo due esempi del metodo di clipping usato nel tool di Matlab:

- Es.1: se e = 0 e de = - 80 allora u = - 37.9
- Es.2: se e = 70 e de= - 80 allora u = 13.4

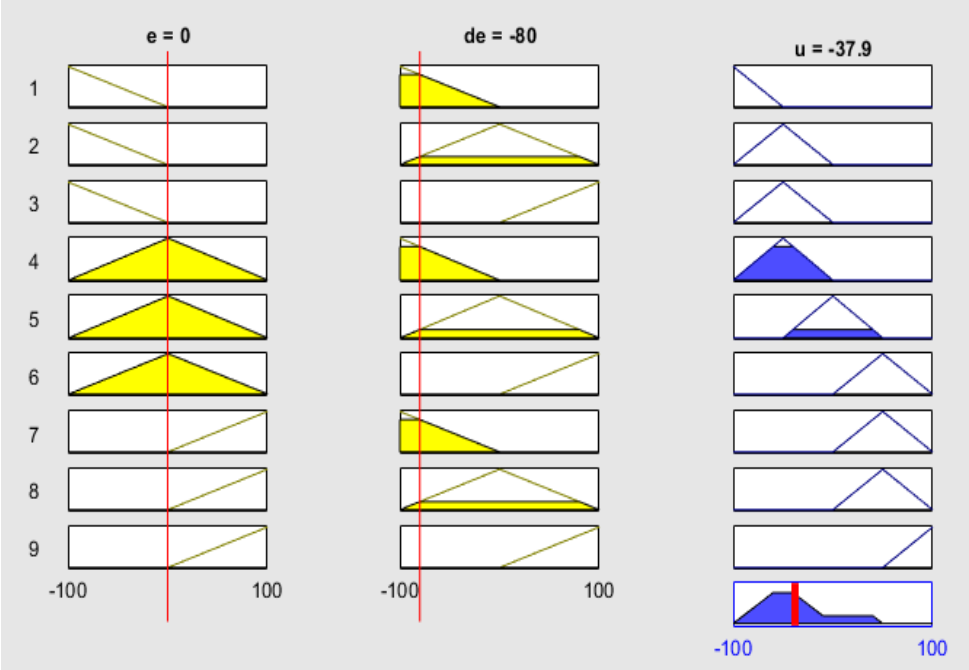


Figura 1-6: es.1 clipping

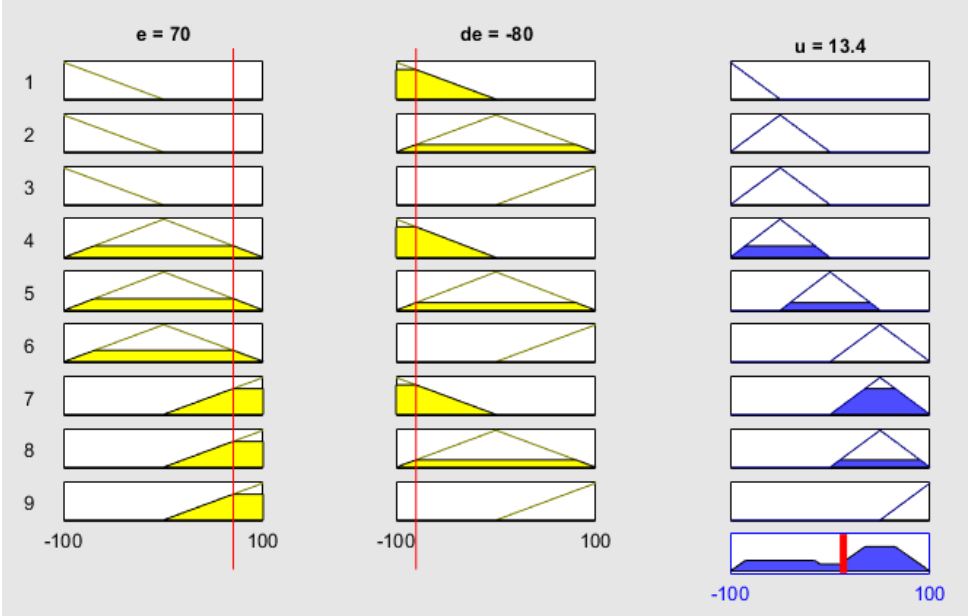


Figura 1-7: es.2 clipping

In figura si può notare come ci siano 9 righe, una per ogni fuzzy rule, in ogni riquadro si può osservare il grado di attivazione della corrispettiva funzione, mentre in basso a destra è rappresentata l'uscita u valutata con il metodo del centro di gravità, che viene illustrato in seguito.

1.1.3 Defuzzificazione

La defuzzificazione è la conversione del fuzzy set di uscita in un valore numerico. La conversione può avvenire con metodi differenti:

- Metodo del centro di gravità o centro di massa:

$$u_0 = \frac{\int u\,\mu(u)\,du}{\int \mu(u)\,du}$$

- Metodo dei massimi che a sua volta ha tre varianti:
  - FOM (first of maximum): u_0 è la prima ascissa del massimo di u
  - LOM (last of maximum): u_0 è l'ultima ascissa del massimo di u
  - MOM (middle of maximum): u_0 è la media di FOM e LOM

Come già sottolineato in precedenza, nel fuzzy tool viene implementato il metodo del centro di gravità per ricavare l'output.

1.2 Funzione di approssimazione fuzzy

La funzione di approssimazione fuzzy varia al variare della forma delle funzioni di appartenenza, della posizione dei massimi e delle regole.
Nel caso di due variabili si può rappresentare come una superficie, allora si parla di superficie di controllo fuzzy. Un esempio è rappresentato in figura tramite il tool di Matlab.

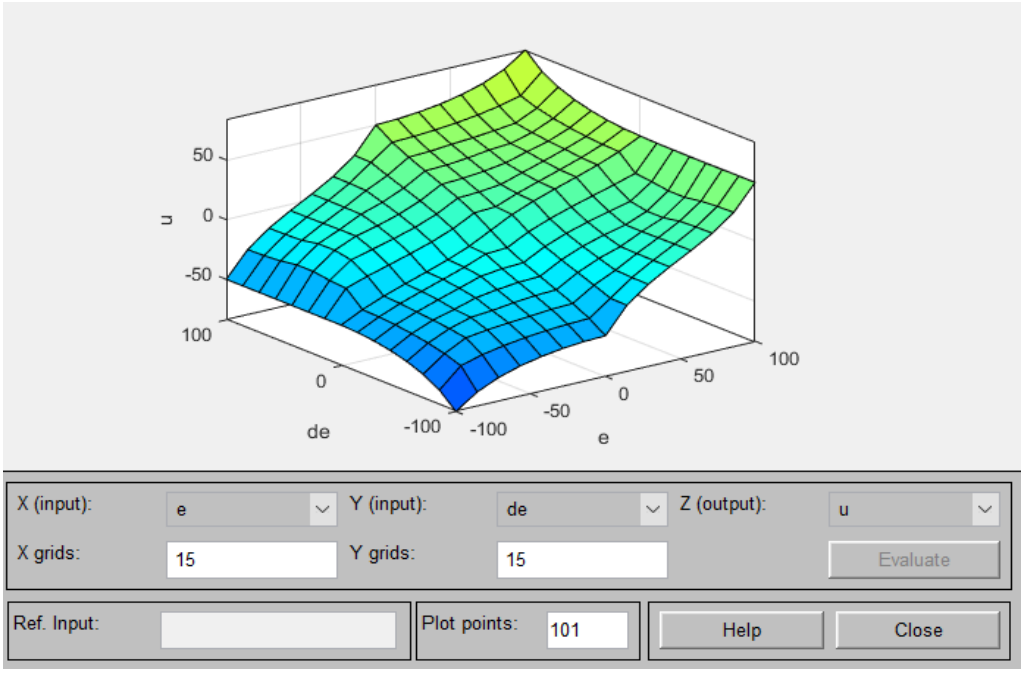


Figura 1-6: superficie di controllo fuzzy

1.3 Metodo di interferenza di Sugeno

Il metodo di Sugeno è un'alternativa a Mamdani e a differenza del precedente le funzioni di uscita sono singoli valori (come dei picchi). È anch'esso implementabile nel tool, ma viene sconsigliato in quanto riduce la gradualità di questo controllo, che è la caratteristica più importante.

1.4 Vantaggi e limiti

La robustezza dei controllori deve ancora essere rigorosamente mostrata, ma è messa in evidenza sperimentalmente in letteratura. Oltretutto sono sistemi molto semplici da realizzare: sono sufficienti infatti schede a basso costo e sensori semplici, quindi non molto precisi, ma robusti ed economici.

Sono comunque presenti dei limiti. Le regole fuzzy vanno fornite da esperti nel settore, ma se la progettazione è su una nuova macchina magari non si ha sufficiente base di conoscenza per implementare le regole. Non sono adatti a problemi di elevata precisione, mentre lo sono in presenza di margini d'incertezza. Va sottolineato che in molte applicazioni non è necessaria un'elevata precisione. Si usano in controlli di: automobili (es. ABS ed iniezione), elettrodomestici, messa a fuoco fotocamere, impianti per il controllo della temperatura.

2 Descrizione Attrezzatura

Il sistema fisico è stato realizzato per studiare un pendolo inverso, essendo un ottimo esempio di sistema instabile sul quale sviluppare dei controllori. Uno schema generale del sistema è riportato in figura.

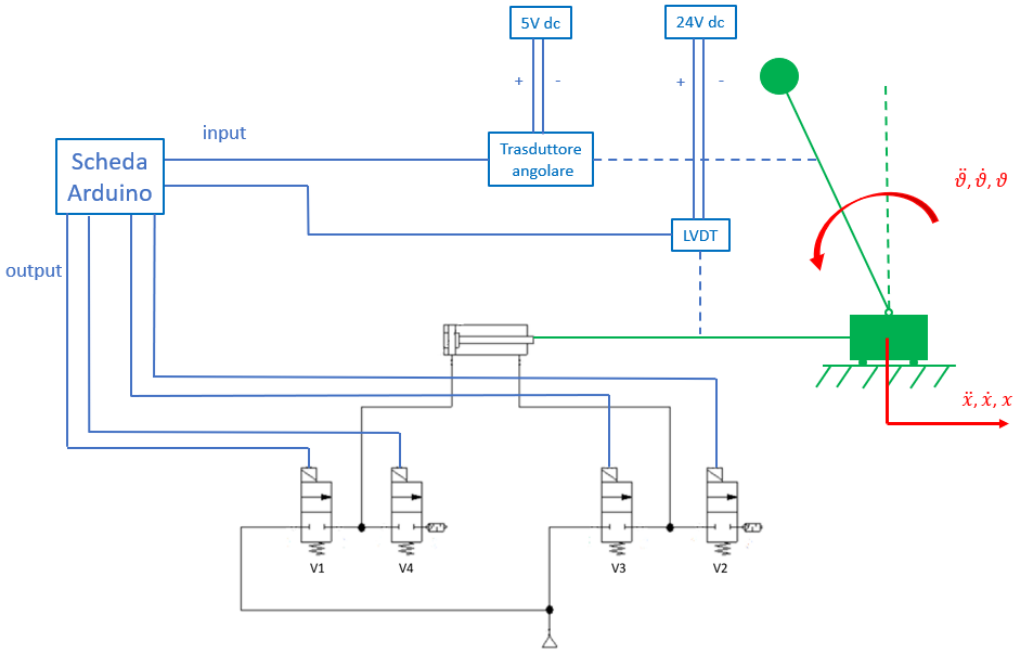


Figura 2-1: schema sistema

In presenza di un segnale positivo dalla scheda Arduino verranno attivate le valvole 1 e 2 consentendo la fuoriuscita dello stelo, mentre con un segnale negativo il segnale (in modulo) viene mandato alle valvole 3 e 4 che consentono il rientro dello stelo.
Il seguente sistema è stato realizzato nel DIMEAS per lo sviluppo iniziale delle tesi [4] e [5] dove è stato studiato inizialmente mediante PLC e Arduino, in seguito si sono cercate diverse implementazioni [6]. Nei paragrafi successivi viene ripresa la loro analisi per inquadrare bene il problema e mostrare l'attrezzatura usata, per una migliore comprensione del problema.

2.1 L'attuatore pneumatico

L'attuatore pneumatico rappresentato in figura è un Camozzi con codice 24N2A16A500.

Figura 2-2: attuatore Camozzi 24N2A16A500

È un attuatore a doppio effetto senza smorzatori. Viene collegato alla struttura principale attraverso un giunto rotante e mentre al carrello tramite giunto sferico, per compensare disallineamenti e far sì che la forza sia trasmessa lungo l'asse dell'asta. In tabella vengono evidenziati i principali parametri dell'attuatore, mentre in figura la velocità massima raggiungibile in relazione al carico.

Parametri attuatore idraulico	
alesaggio	16 mm
diametro stelo	6 mm
corsa	500 mm
temperature lavoro	0 ÷ 80 °C
pressione d'esercizio	0 ÷ 10 bar

Tabella 2-1: parametri attuatore

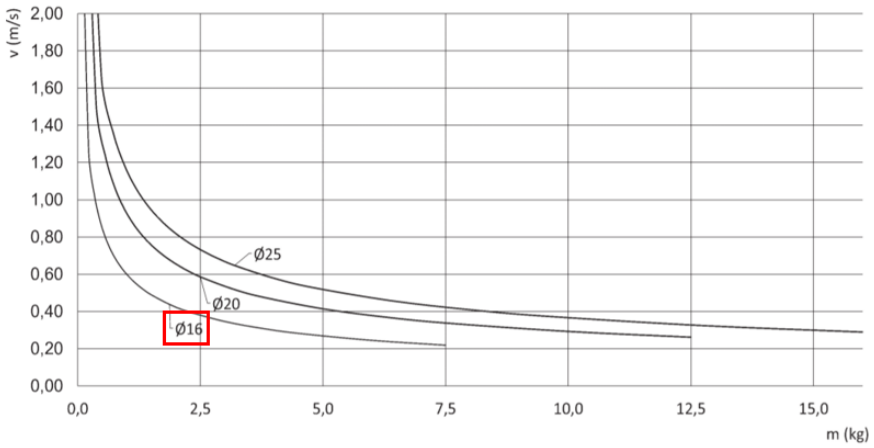


Figura 2-3: velocità massime pistone

2.2 Valvola

Le quattro valvole sono delle elettrovalvole proporzionali 2/2 normalmente chiuse. Il driver può essere controllato con una tensione compresa tra 0 e 10 V dc o con una corrente compresa tra 0 e 10 mA; la tensione di alimentazione è di 24Vdc. In particolare, il driver funziona con un segnale PWM ad una frequenza di 500Hz. In questo caso il driver sarà controllato con tensione poiché l'uscita della scheda Arduino fornisce un segnale PWM nell'intervallo tra 0 e 5V. Quindi dopo un amplificatore di tensione che raddoppia la tensione fornita dalla scheda Arduino, il segnale viene inviato al driver: una tensione pari a 0 V dc corrisponde a un ciclo di lavoro dello 0% e quindi la valvola è totalmente chiusa, altrimenti una tensione pari a 10 V dc corrisponde con un ciclo di lavoro del 100% e quindi la valvola è totalmente aperta. In figura abbiamo una rappresentazione della valvola, con schema ISO e disegni, viene anche riportata la portata massima della valvola in funzione della pressione di taratura. In tabella sono riportate le caratteristiche principali della valvola.

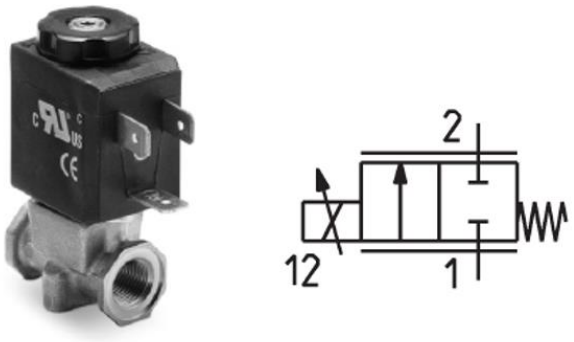


Figura 2-5: valvola Camozzi AP-7211-LR2-U7 e schema ISO

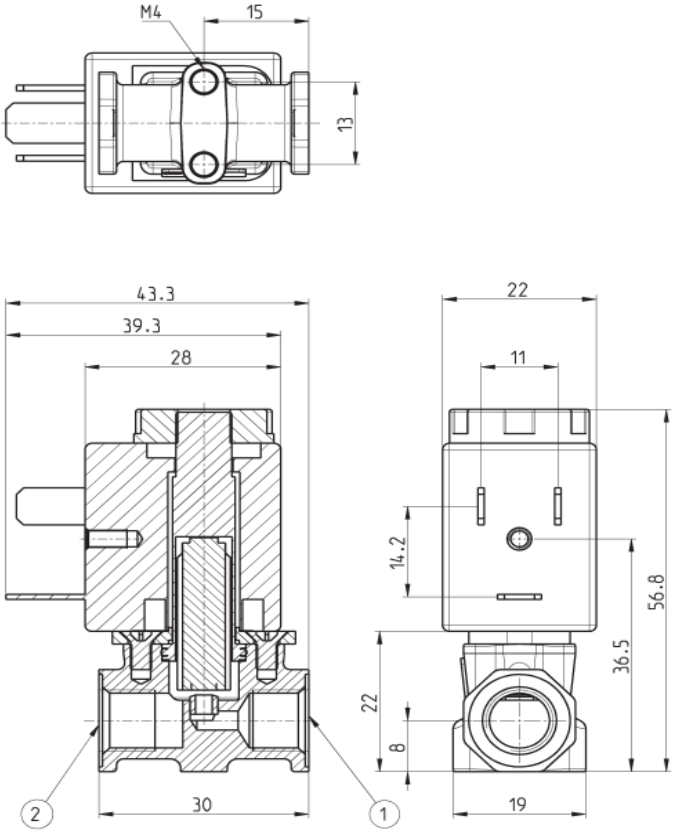


Figura 2-4: disegni valvola

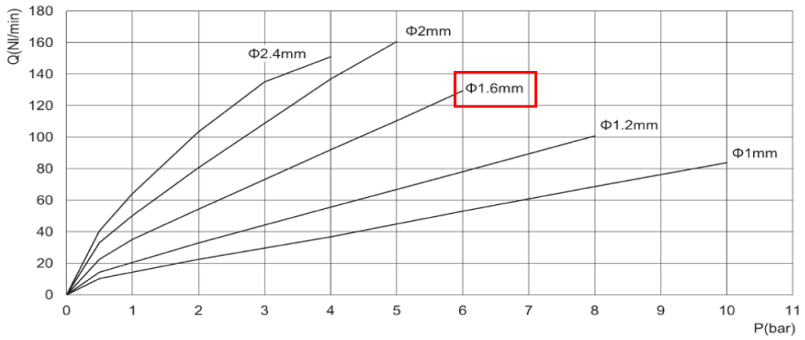


Figura 2-6: portata massima valvola in funzione pressione taratura

Parametri valvola	
dimensione	22 mm
diametro nominale	1,6 mm
massima pressione alimentazione	6 bar
portata massima	110 Nl/min
Kv	1,2 l/min
connettore	G1/8
voltaggio di alimentazione	24 Vdc
funzione	2/2 NC

Tabella 2-2: parametri valvola

Per una migliore comprensione viene mostrato il funzionamento della generazione dell'output in PWM partendo da un segnale continuo, utilizzando la definizione di duty cycle.

$$d.c = \frac{t_{on}}{T}\ 100$$

In figura viene mostrato un esempio di conversione ripreso da [7].

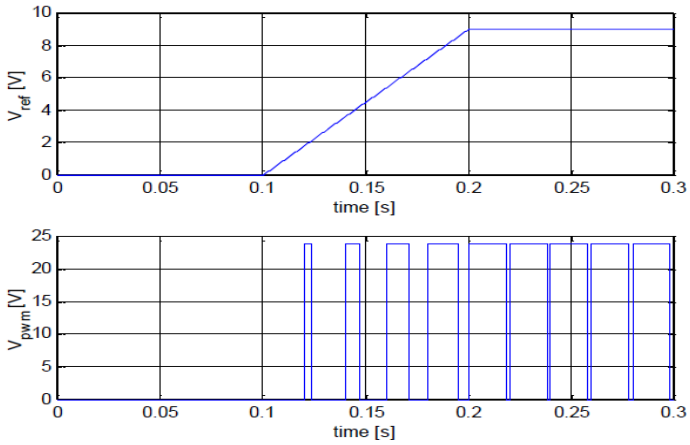


Figura 2-7: conversione PWM

2.3 Trasduttore angolare

Il trasduttore angolare serve per misurare la posizione angolare del pendolo e trasmetterla ad Arduino. La misurazione viene fatta tramite un segnale in tensione. Il trasduttore usato sfrutta l'effetto Hall ed è prodotto da Penny Giles con il codice SRH280- DP/090/180/A1/3/D/68/P5. L'uscita in tensione è 0.5÷4.5 V dc e il campo di misurazione è selezionabile tra 0÷90° (con il canale CH1) oppure 0÷180° (con il canale CH2). Essendo il segnale tra 0 e 5V non è necessaria una conversione per la scheda Arduino. I parametri più importanti sono evidenziati in tabella.

Parametri trasduttore angolare	
alimentazione	5 V dc
uscita	0,5 ÷ 4,5 V dc
campo misura CH1	0° ÷ 90°
campo misura CH2	0° ÷ 180°
risoluzione	0,025% FS (12 bit)
non linearità	< 0,4% FS

Tabella 2-3: parametri trasduttore angolare

2.4 Trasduttore di posizione LVDT

Il trasduttore di posizione serve a misurare la posizione x del carrello ed è fissato all'attrezzatura tramite due giunti rotoidali. La corsa misurabile corrisponde a quella dell'attuatore idraulico, ovvero di 500 mm. La tensione in uscita può assumere valori da 0 a 10 V dc. Non è possibile collegarlo direttamente alla scheda Arduino perché accetta valori soltanto fino a 5 V dc. I principali parametri del trasduttore sono evidenziati in figura.

Parametri trasduttore LVDT	
alimentazione	24 V dc
uscita	0,5 ÷ 10 V dc
campo misura	0 ÷ 500 mm
temperatura di lavoro	0° ÷ 70°
banda morta	200 Hz
non linearità	< 0,5 FS

Tabella 2-4: parametri trasduttore LVDT

2.5 Guida lineare a ricircolo di sfere

Guida lineare a ricircolo di sfere del costruttore Rexroth. Il codice per il minipattino a sfere è R044209401, mentre il codice per la minirotaia a sfere è R044500431620. La rotaia può essere fissata al banco tramite i fori presenti sulla medesima (visibili in figura), mentre il collegamento tra pattino e rotaia avviene tramite quattro viti mordenti o bulloni.

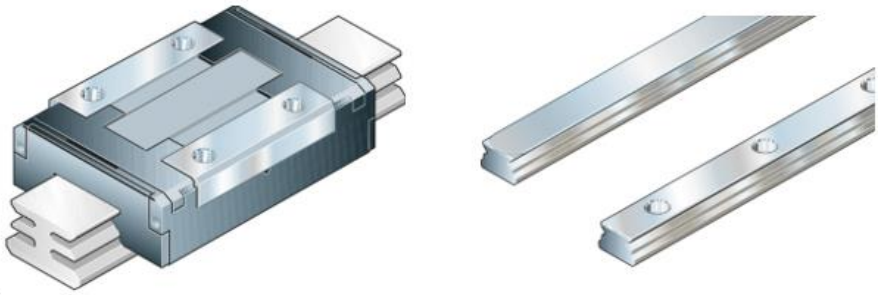


Figura 2-8: Minipattino (sx) e Minirotaia (dx)

2.6 Gruppo carrello-pendolo

Il gruppo è composta da un carrello e da un pendolo. Il carrello è collegato alla rotaia tramite una guida prismatica, mentre il pendolo è collegato al medesimo tramite una cerniera.

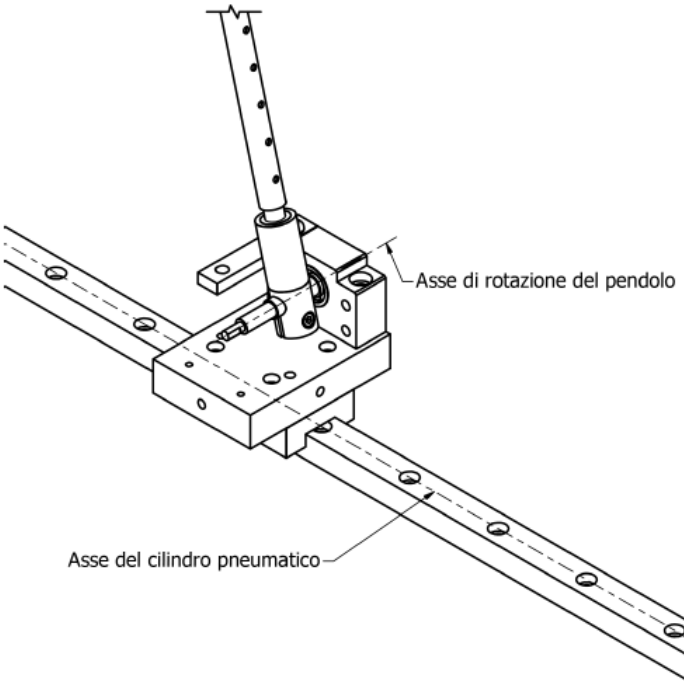


Figura 2-9: cinematismi

Il pendolo è stato costruito in modo tale da consentirne la variazione della lunghezza tra 400 e 700 mm, per l'analisi effettuata è stata scelta una lunghezza di 500 mm. Un'altra possibile variazione è la massa del pendolo, che può essere variata sostituendo il disco installato con uno di peso diverso.

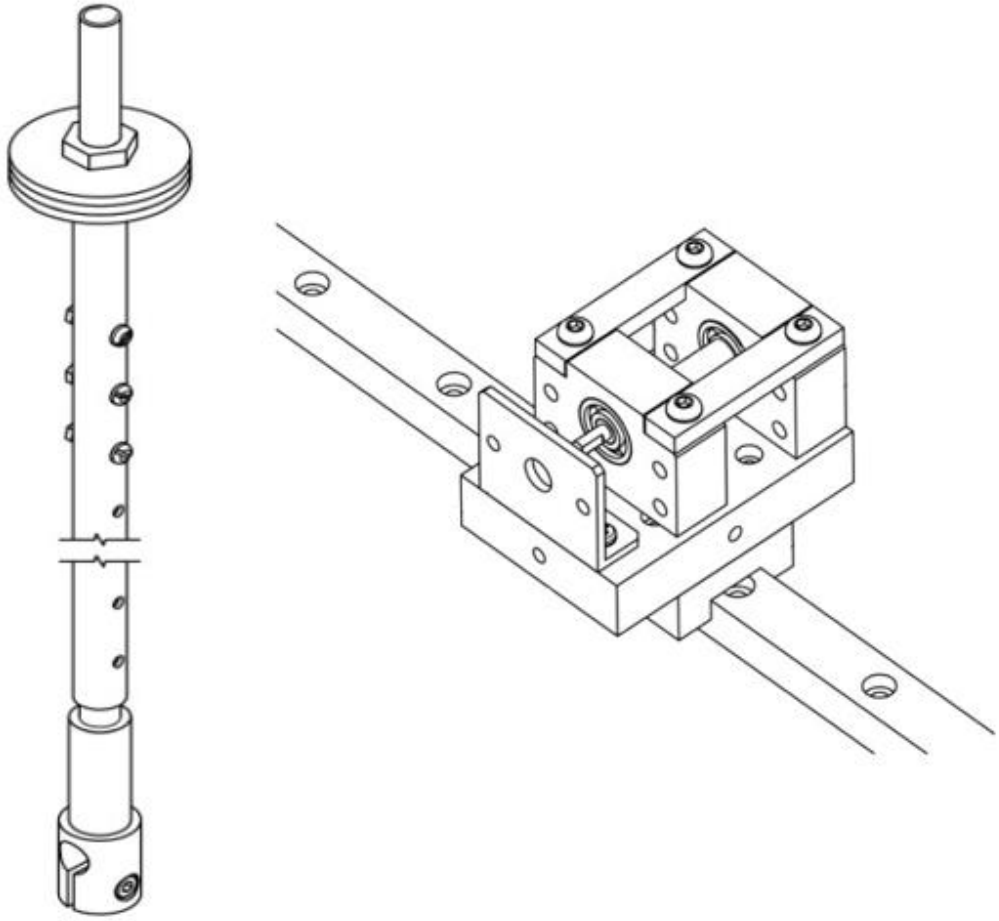


Figura 2-10: pendolo e struttura del carrello

Ad eccezione del disco (rappresentato come una massa concentrata) tutti i componenti del pendolo sono realizzati in alluminio, poiché la barra risulta molto leggera il suo momento di inerzia viene trascurato nella prima analisi del modello.
Sulla struttura sono presenti dei fori per collegare il sensore LVDT, sono anche presenti due barre laterali per evitare che il pendolo urti le guide, limitandone l'oscillazione tra -20° e +20°.

2.7 Arduino Mega 2560

La scheda Arduino Mega 2560 provvede al controllo del sistema.
Arduino Mega 2560 è una scheda microcontrollore basata su ATmega2560. Dispone di 54 pin di ingresso / uscita digitali (di cui 15 utilizzabili come uscite PWM), 16 ingressi analogici, 4 UARTs (porte seriali hardware), un oscillatore a cristallo da 16 MHz, una connessione USB, un jack di alimentazione, un header ICSP e un pulsante di ripristino. Contiene tutto il necessario per supportare il microcontrollore; basta collegarlo a un computer con un cavo USB o alimentarlo con un adattatore AC-DC o una batteria per utilizzarlo. Il Mega 2560 è un aggiornamento di Arduino Mega.

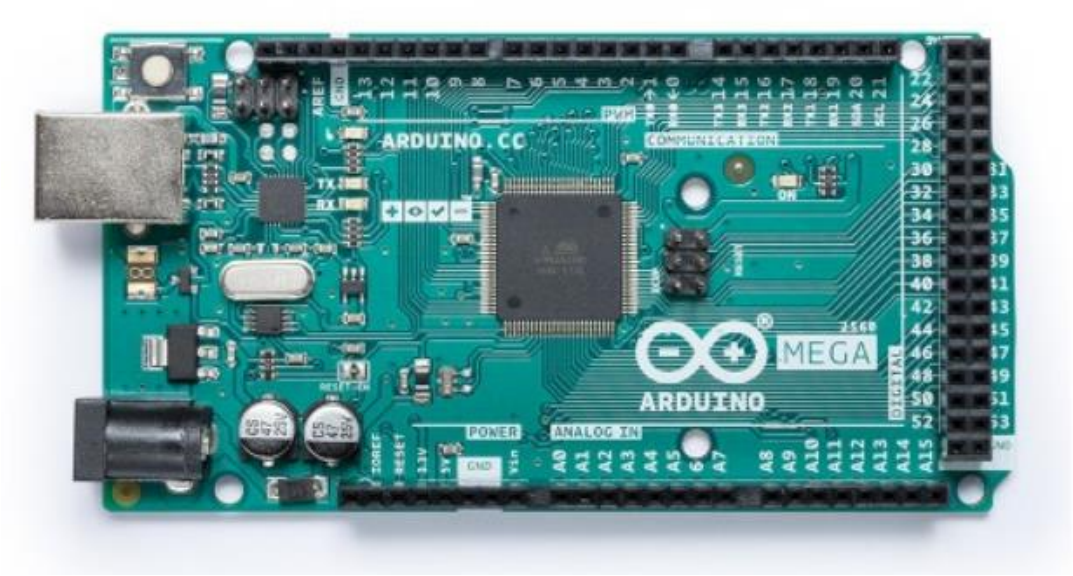


Figura 2-11: Arduino Mega 2560

La scheda può essere programmata con il software Arduino (IDE) usando il linguaggio di programmazione C di alto livello. Le principali caratteristiche della scheda sono evidenziate in tabella, reperite da [8].

Parametri Arduino Mega 2560	
voltaggio operativo	5 V
voltaggio input (raccomandato)	7 -12 V
voltaggio input (limite)	6 -20 V
Pin I/O digitali	54 (di cui 15 con out PWM)
Pin I/O analogici	16
corrente dc per Pin I/O	20 mA
memoria flesh	256 KB (di cui 8 KB usati dal boot loader)
SRAM	8 KB
EEPROM	4 KB
clock speed	16 MHz
led incorporato	13
lunghezza	101,52 mm
larghezza	53,3 mm
peso	37 g

Tabella 2-5: parametri Arduino Mega 2560

2.8 PCB

Il PSB è un'interfaccia tra "sensori – Arduino" e tra "driver delle valvole – Arduino". È un apparato necessario visto che Arduino accetta come input e genera come output solo segnali tra 0 – 5 V. I driver delle valvole funzionano in un range tra 0 – 10 V, è quindi necessario un circuito che raddoppi la tensione in uscita dalla scheda Arduino e, poiché le valvole sono controllate in coppia, servono due circuiti. Il sensore di posizione LVDT genera una tensione tra 0 – 10 V, pertanto il suo valore va dimezzato per essere accettato dalla scheda Arduino, di conseguenza è possibile usare un semplice partitore di tensione che la dimezzi. Solo il sensore angolare fornisce una tensione tra 0 -5 V, allora può essere collegato direttamente alla scheda.

3 Modello Analitico

Modello meccanico del pendolo e del carrello: continuando l'analisi di [6], è rappresentato in figura, dove sono indicati i versi convenzionali della variabile x e ϑ.

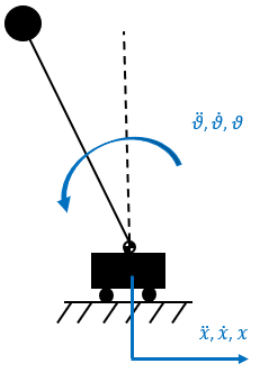


Figura 3-1: versi convenzionali variabili cinematiche

Di seguito vengono rappresentati i diagrammi di corpo libero dei medesimi corpi.

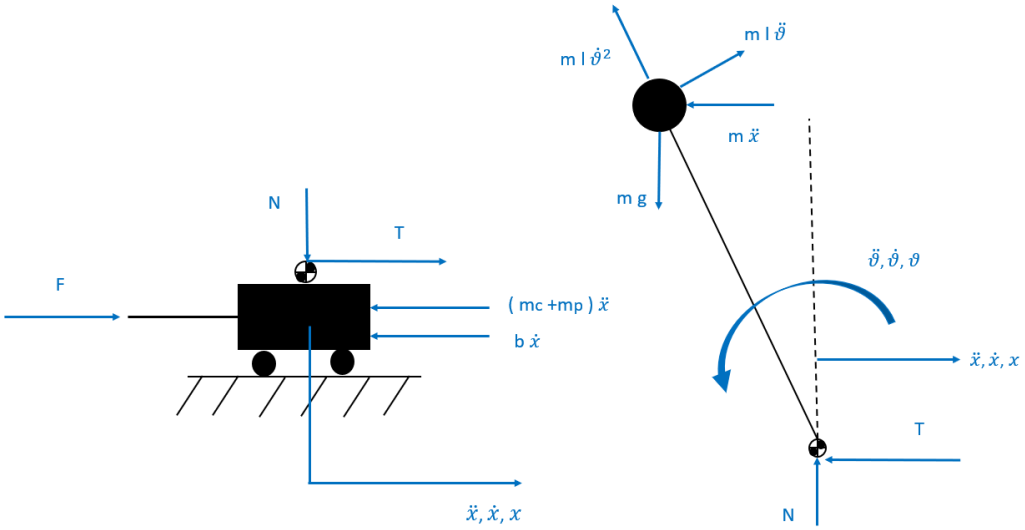


Figura 3-2: diagrammi di corpo libero

Nella trattazione viene trascurato il momento di inerzia dell'asta del pendolo, visto che è realizzata in alluminio e la massa concentrata posta in monte genera forze di entità molto maggiori.
L'equilibrio orizzontale pendolo:

$$m\, l\, \ddot{\vartheta}\, \cos\vartheta \; - \; m\, \ddot{x} \; - m\, l\, \dot{\vartheta}^2\, \sin\vartheta \; - \; T \; = 0$$

L'equilibrio alla rotazione del pendolo sulla cerniera:

$$m\,g\,\sin\vartheta\;l\;+\;m\,\ddot{x}\,\cos\vartheta\;l\;-\;m\,l\,\ddot{\vartheta}\,l\;=0$$

L'equilibrio orizzontale carrello:

$$F\;+\;T\;-\;(m_C+m_P)\,\ddot{x}\;-\;b\,\dot{x}\;=0$$

Dove:

- m_C: la massa del carrello
- m_P: la massa del pistone
- m: la massa concentrata del pendolo
- l: la lunghezza dell'asta
- b: il coefficiente di attrito viscoso, che include sia l'attrito del carrello sia l'attrito nel cilindro pneumatico
- F: è la forza generata dall'attuatore idraulico

Si può osservare come tenendo conto a questo livello dell'inerzia generata dalla massa del pistone, non venga quindi impostata nell'equilibrio del pistone.
In figura viene rappresentato un modello del cilindro a doppio effetto usato per movimentare il carrello e tenere in equilibrio il pendolo.

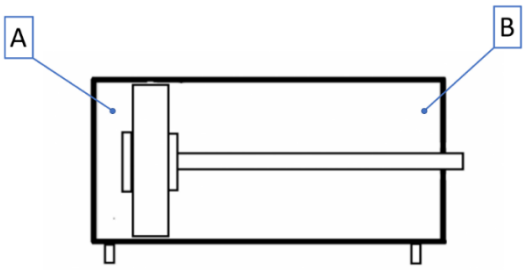


Figura 3-3: riferimenti camere attuatore

Vengono evidenziate, a sinistra le pressioni (assolute) che agiscono su pistone e stelo mentre a destra il diagramma di corpo libero.
L'equilibrio orizzontale dell'attuatore idraulico:

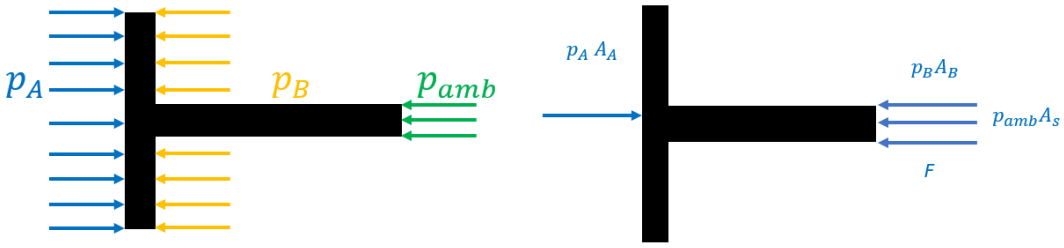


Figura 3-4: pressioni e forze sull'attuatore

$$p_A\,A_A\;-\;p_B\,A_B\;-\;p_{amb}\,A_s\;-\;F\;=0$$

$$A_A\;=\;\pi\,\frac{D^2}{4}$$

$$A_s = \pi \frac{d^2}{4}$$
$$A_B = A_A - A_s$$

Dove:

- D: è l'alesaggio del cilindro
- d: è il diametro dello stelo

Per una trattazione più completa viene considerata la forza di reazione della camera quando il cilindro arriva a fondo corsa. Essendo una deformazione viene considerata come una forza di tipo elastico, con una rigidezza molto elevata. È molto importante tenere in conto che il verso della forza varia a seconda di quale finecorsa viene raggiunto.

$$F_{reazione\ camera} = sign(x)\left(|x| - \frac{corsa}{2}\right) k_{attuatore}\ , se\ |x| \geq \frac{corsa}{2}$$

$$F = p_A\, A_A - p_B\, A_B - p_{amb}\, A_s - F_{reazione\ camera}$$

Ipotesi di gas ideale:

$$p\, v = R\, T \Leftrightarrow p\, \frac{V}{m} = R\, T \Leftrightarrow m = \frac{p\, V}{R\, T}$$

L'equazione di conservazione della massa, con ipotesi di trasformazione isoterma ($T = cost$):

$$\dot{m} = \frac{dm}{dt} = \frac{\dot{p}\, V}{R\, T} + \frac{p\, \dot{V}}{R\, T}$$

Nella camera A dell'attuatore:

$$\dot{m}_{\ in,A} - \dot{m}_{out,A} = \frac{dm_A}{dt} = \frac{\dot{p}_{\ A} V_A}{R\, T} + \frac{p_A\, \dot{V}_A}{R\, T}$$

Nella camera B dell'attuatore:

$$\dot{m}_{\ in,B} - \dot{m}_{out,B} = \frac{dm_B}{dt} = \frac{\dot{p}_{\ B} V_B}{R\, T} + \frac{p_B\, \dot{V}_B}{R\, T}$$

Volume nella camera A:

$$V_A = V_{A0} + \left(\frac{1}{2}c + x\right) A_A$$

Volume nella camera B:

$$V_B = V_{B0} + \left(\frac{1}{2}c - x\right) A_B$$

Dove:

- V_{A0}: è il volume morto nella camera posteriore
- V_{B0}: è il volume morto nella camera anteriore
- c : è la corsa dello stantuffo

Derivate temporali dei volumi in camera:

$$\dot{V}_A = \dot{x}\, A_A$$
$$\dot{V}_B = \dot{x}\, A_B$$

Equazione di portata attraverso una valvola, ISO 6358:

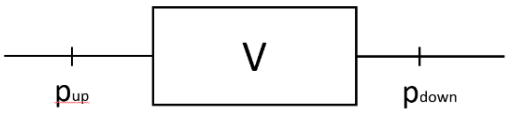


Figura 3-5: convenzione pressione nelle valvole

$$Q = C_v\, p_{up} \sqrt{1 - \left(\frac{r - b_v}{1 - b_v}\right)^2} \sqrt{\frac{T_N}{T_{up}}} \quad , se\ b_v < \mathrm{r} \leq 1\ (flusso\ sub - sonico)$$

$$Q = Q_{cr} = C_v p_{up} \sqrt{\frac{T_N}{T_{up}}}, se\ 0 \leq r \leq b_v\ (flusso\ sonico)$$

$$r = \frac{p_{down}}{p_{up}}$$

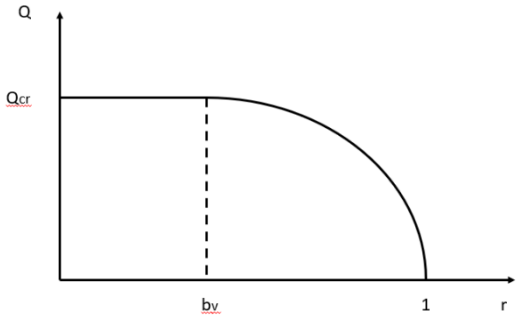


Figura 3-6: portata nella valvola ISO 6358

dove:

- C_v : è la conduttanza della valvola
- b_v: è il rapporto critico della valvola

Relazione tra il flusso di massa e flusso in volume:

$$\dot{m} = Q\, \rho$$

$$\dot{m} = C_v\, p_{up}\, \rho \sqrt{1 - \left(\frac{r - b_v}{1 - b_v}\right)^2} \sqrt{\frac{T_N}{T_{up}}} \quad , se\ b_v < r \leq 1\ (flusso\ sub - sonico)$$

$$\dot{m} = \dot{m}_{cr} = C_v\, p_{up}\, \rho \sqrt{\frac{T_N}{T_{up}}} \quad , se\ 0 \leq r \leq b_v\ (flusso\ sonico)$$

La valvola proporzionale è governata da un comando u, proveniente dal controllo:

$$-1 \leq u \leq 1$$

$u > 0$ ottengo la fuoriuscita dello stelo
$u < 0$ ottengo il rientro dello stelo

$$\dot{m} = |u| \, C_v \, p_{up} \, \rho \sqrt{1 - \left(\frac{r - b_v}{1 - b_v}\right)^2} \sqrt{\frac{T_N}{T_{up}}} \quad , se \; b_v < r \leq 1 \; (flusso \; sub - sonico)$$

$$\dot{m} = \dot{m}_{cr} = |u| \, C_v \, p_{up} \, \rho \sqrt{\frac{T_N}{T_{up}}} \quad , se \; 0 \leq r \leq b_v \; (flusso \; sonico)$$

Le seguenti equazioni vengono implementate in un file Simulink per la simulazione del banco prova sperimentale.
Per una lettura immediata vengono riportate le variabili con valori in tabella

variabile	descrizione	valore	unità
t_in	tempo inizio sim.	0	s
t_fin	tempo fine sim.	40	s
Tsim	tempo campionamento	1,00E-03	s
dh_zone	dead zone	1E-10	
m	massa pendolo	0,2	kg
mc	massa carrello	1,8	kg
mp	massa pistone	0,4	kg
g	acc. Gravità	9,81	m/s^2
b	attrito viscoso	20,83	Ns/m
l	lunghezza asta	0,5	m
R	costante dei gas (aria)	287,05	J/kg K
T	temp. camera cilindro	293	K
T_N	temp. Normale	293	K
c	corsa cilindro	0,5	m
V_A0	vol. morto camera post.	1,00E-06	m^3
V_B0	vol. morto camera ant.	1,00E-06	m^3
D	diametro pistone	1,60E-02	m
d	diametro stelo	6,00E-03	m
A_A	Area spinta camera post.	pi*D^2/4	m^2
A_s	area stelo	pi*d^2/4	m^2
A_B	Area spinta camera ant.	A_A-A_s	m^2
p_amb	pressione ambiente	1,00E+05	Pa
p_supply	pressione alimentazione	6,00E+05	Pa
bv	rapp. critico valvola	0,3	
Cv	conduttanza valvola	2,40E-09	m^3(ANR)/s Pa
rho	densità aria nella valvola	1,225	kg/m^3

Tabella 3-1: valori parametri

4 Modello Simulink

Modello a blocchi del sistema: continuando l'analisi di [6], si presenta la possibilità di selezionare due differenti tipi di set a gradino e una sinusoide e di implementare un disturbo esterno.

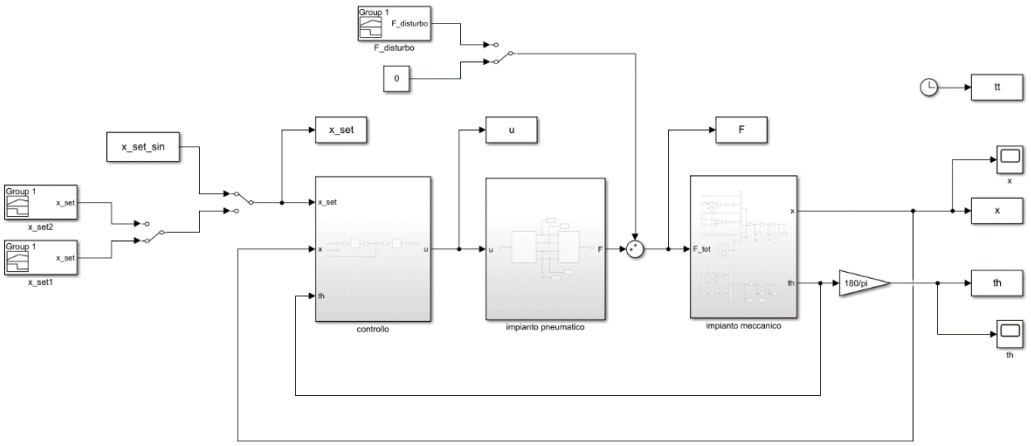


Figura 4-1: schema sistema

Nel sottosistema impianto pneumatico troviamo a sua volta due sottosistemi.

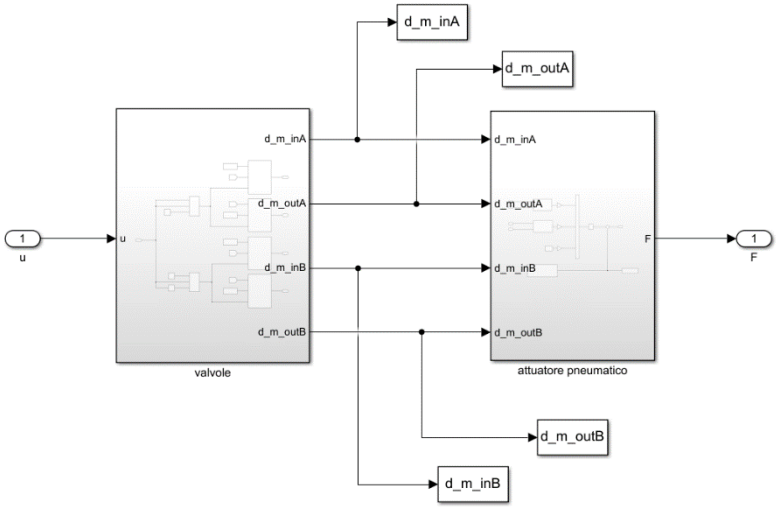


Figura 4-2: schema pneumatico

Nel blocco valvole viene rappresentato come il comando u uscente dal blocco di controllo vada a governare le varie valvole a seconda del segno.

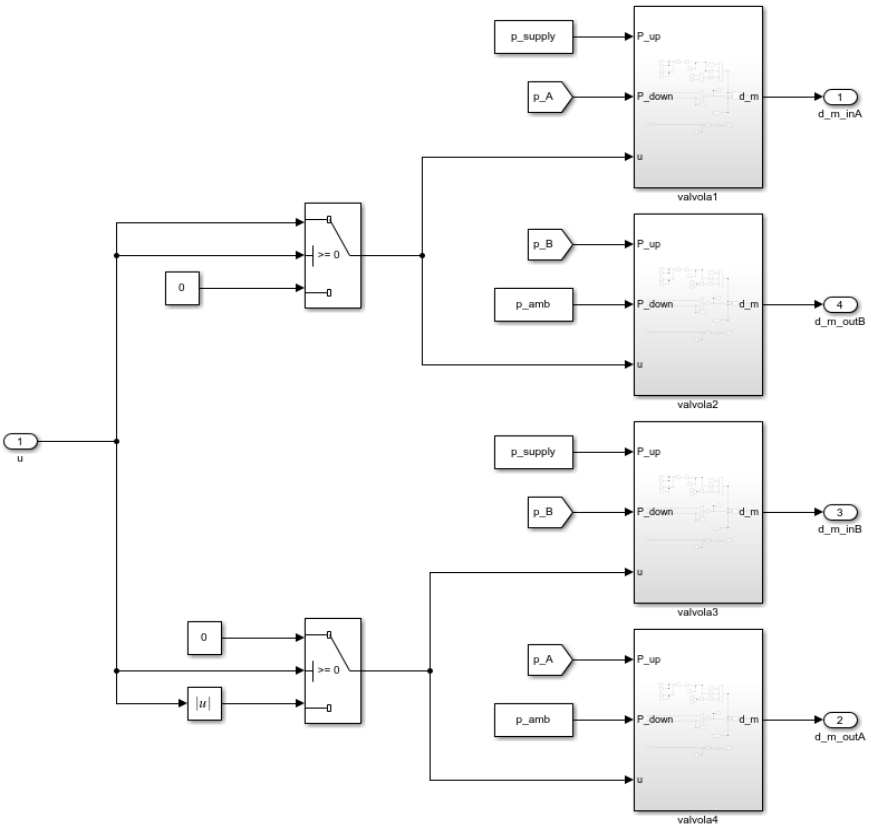


Figura 4-3: schema valvole

Per una migliore comprensione dello schema pneumatico è stato realizzato il seguente schema.

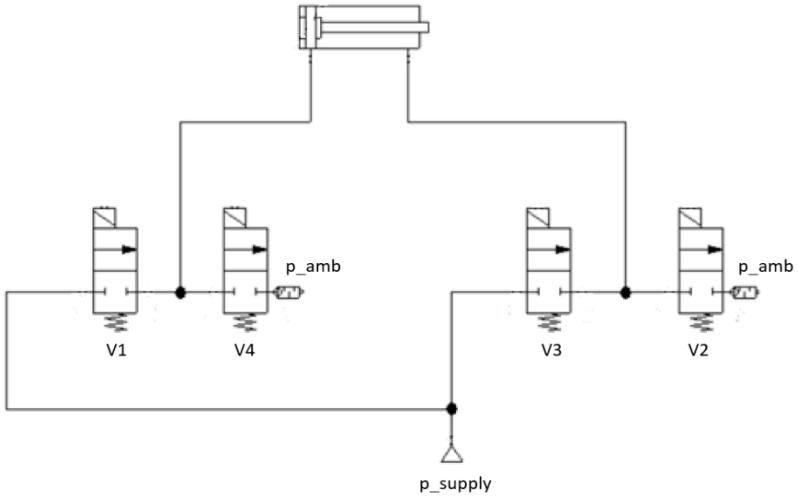


Figura 4-4: collegamenti valvole-attuatore

Dentro ogni corrispettivo blocco valvole viene implementata la normativa ISO 6358 per il calcolo della portata. Nello switch il valore 0.3 fa riferimento al rapporto critico della valvola in prova.

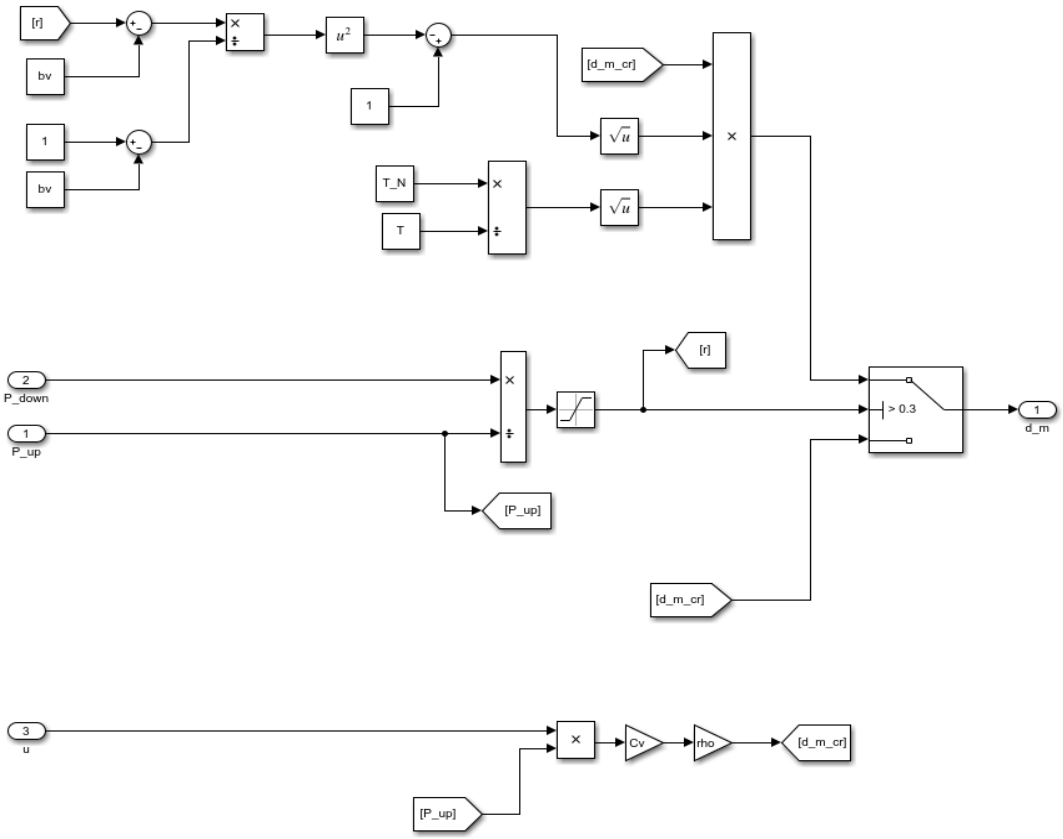


Figura 4-5: flusso nella valvola

Dentro il blocco dell'attuatore pneumatico viene implementato l'equilibrio orizzontale dello stantuffo (non consideriamo ancora a questo livello l'inerzia, che verrà implementata in seguito).

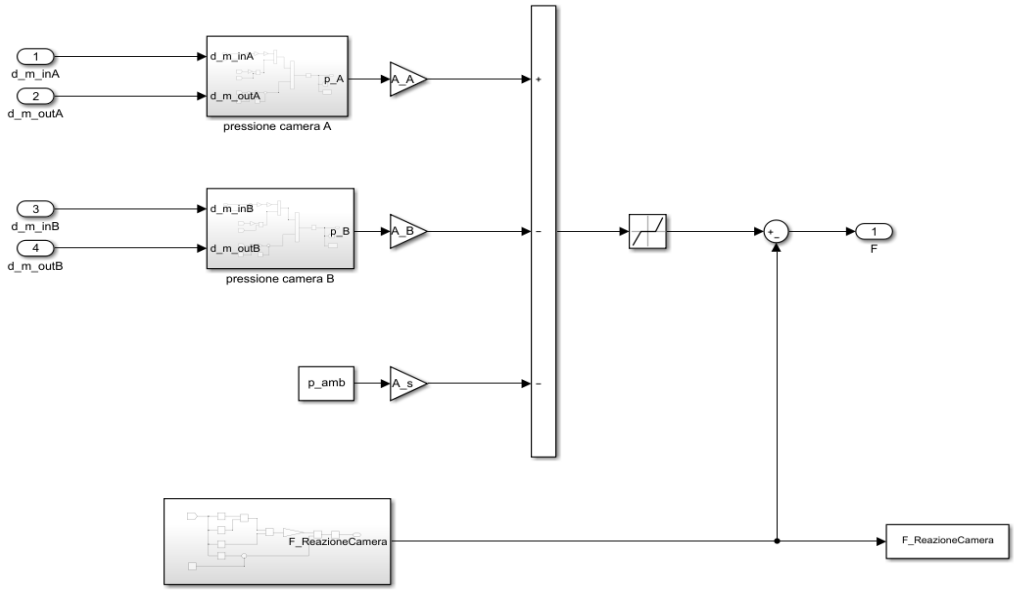


Figura 4-6: schema attuatore pneumatico

Nei blocchi pressione camera viene implementata l'equazione di conservazione della massa nella camera A e B dell'attuatore. Il blocchetto dell'integrale presenta limiti superiore e inferiore, visto che: $p_{amb} \leq p \leq p_{supply}$

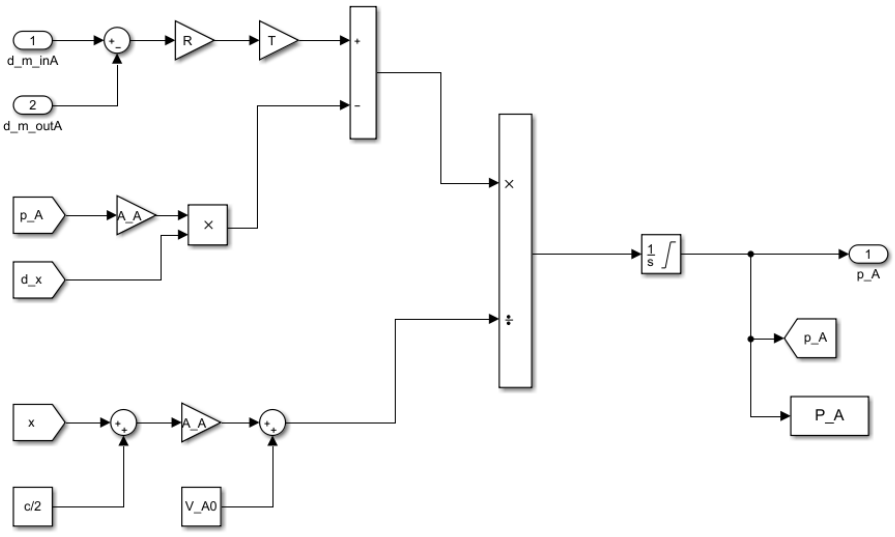


Figura 4-7: continuità nella camera A

Nella camera A, si evidenzia la derivata della pressione in camera nell'equazione della portata:

$$\dot{m}_{in,A} - \dot{m}_{out,A} = \frac{dm_A}{dt} = \frac{\dot{p}_A V_A}{R\,T} + \frac{p_A \dot{V}_A}{R\,T}$$

$$\dot{p}_A = \frac{1}{\left(x + \frac{c}{2}\right) A_A + V_{A0}} \left(R\,T \left(\dot{m}_{in,A} - \dot{m}_{out,A}\right) - p_A\, \dot{x}\, A_A \right)$$

Nella camera B, si evidenzia la derivata della pressione in camera nell'equazione della portata:

$$\dot{m}_{in,B} - \dot{m}_{out,B} = \frac{dm_B}{dt} = \frac{\dot{p}_B V_B}{R\,T} + \frac{p_B \dot{V}_B}{R\,T}$$

$$\dot{p}_B = \frac{1}{\left(-x + \frac{c}{2}\right) A_B + V_{B0}} \left(R\,T \left(\dot{m}_{in,B} - \dot{m}_{out,B}\right) + p_B\, \dot{x}\, A_B \right)$$

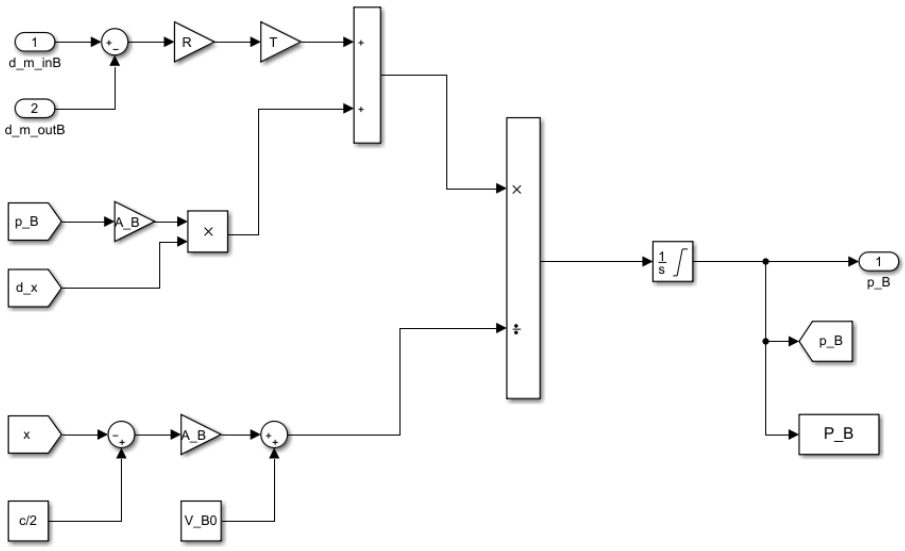


Figura 4-8: continuità nella camera B

Per completare l'equilibrio dello stantuffo è stata calcolata la forza di reazione in camera nel caso in cui vengano raggiunti e urtati i finecorsa.

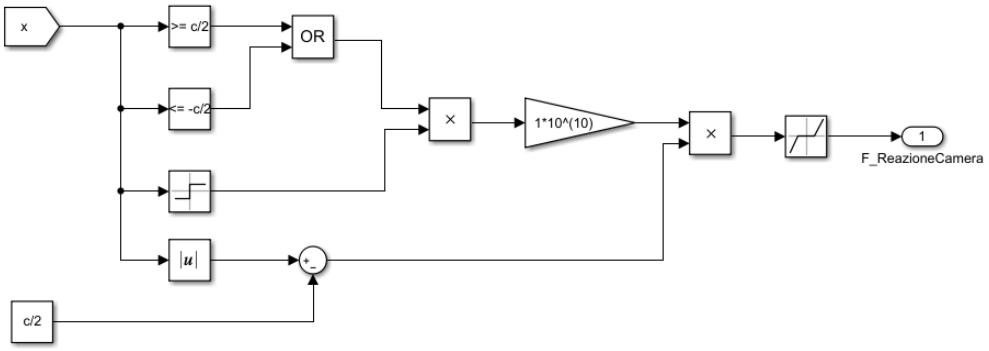


Figura 4-9:reazione stantuffo-camera

Nel blocco dell'impianto meccanico vengono implementate le equazioni di equilibrio necessarie per ricavare le variali cinematiche di pendolo e carrello.
L'equilibrio orizzontale pendolo:

$$m\,l\,\ddot{\vartheta}\,\cos\vartheta\,-\,m\,\ddot{x}\,-m\,l\,\dot{\vartheta}^2\,\sin\vartheta\,-\,T\,=0$$

L'equilibrio alla rotazione del pendolo sulla cerniera:

$$m\,g\,\sin\vartheta\,\,l\,+\,m\,\ddot{x}\,\cos\vartheta\,\,l\,-\,m\,l\,\ddot{\vartheta}\,l\,=0$$

L'equilibrio orizzontale carrello:

$$F\,+\,\,T\,-\,(m_C+m_P)\,\ddot{x}\,-\,b\,\dot{x}\,=0$$

Dalle tre precedenti equazioni riusciamo a ricavare $\ddot{\vartheta}$ e $\ddot{x}$:

$$\ddot{\vartheta}=\frac{1}{l}(\ddot{x}\,cos\,\vartheta+g\,sin\,\vartheta)$$

$$\ddot{x}\,=\frac{1}{m+m_C+\,m_P+\,m\,cos^2\vartheta}\,(\,F\,+\,mg\,sin\,\vartheta\,cos\,\vartheta\,-\,b\,\dot{x}\,-\,m\,l\,\dot{\vartheta}^2\,\,sin\,\vartheta\,)$$

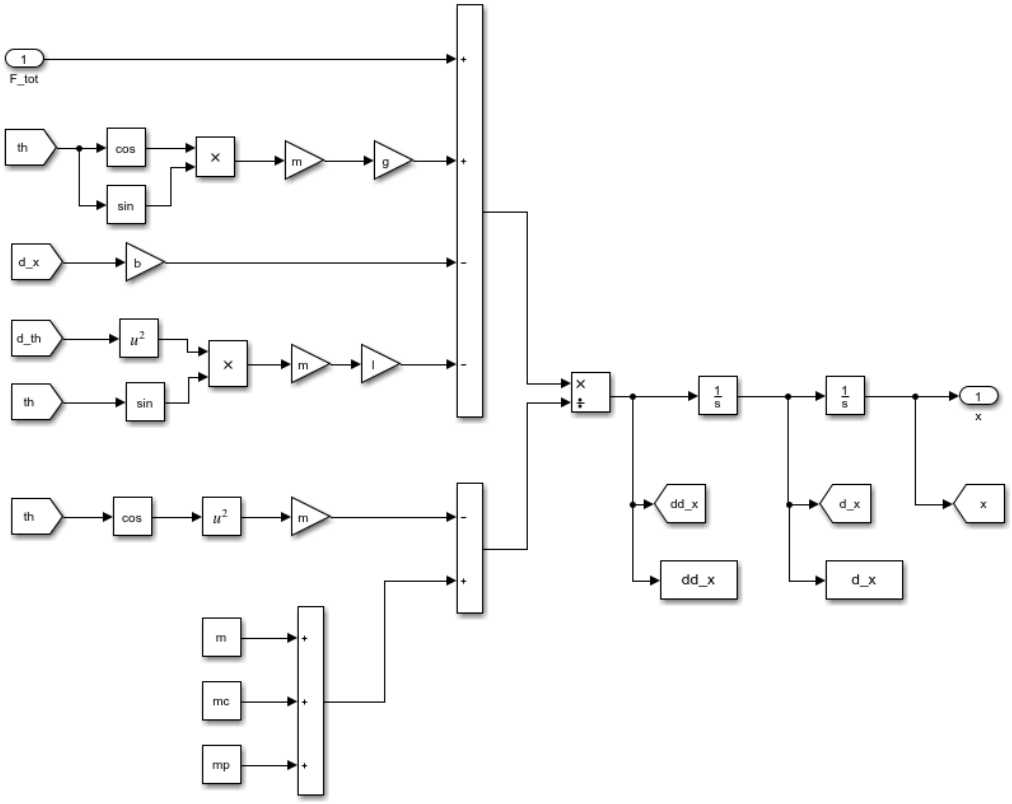


Figura 4-10: equilibrio orizzontale

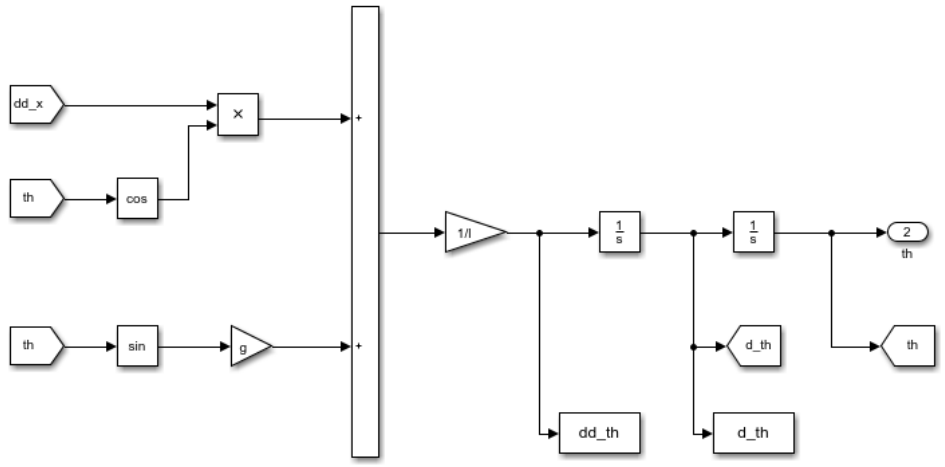


Figura 4-11: equilibrio alla rotazione

5 Controlli Fuzzy

Nel modello Simulink viene implementato il blocchetto Fuzzy Logic Controller che consente di implementare un controllo fuzzy mediante un file di estensione ". fis" precedentemente creato con il fuzzy tool di Matlab.

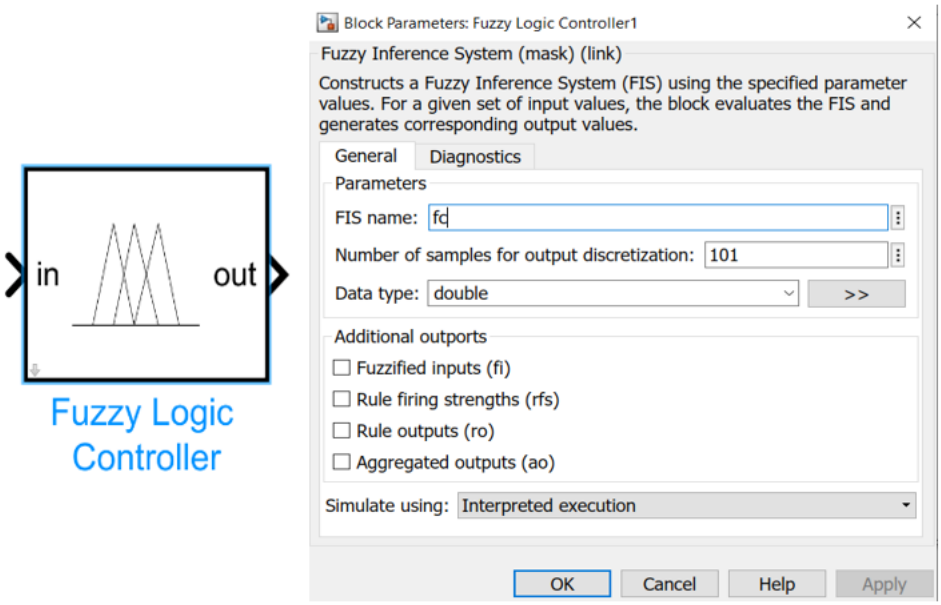


Figura 5-1: Fuzzy Logic Controller

Nel listato Matlab viene inserito il comando:

```
fc=readfis('fc_1');
```

la cui funzione è quella di caricare il file fc_1.fis (nell'esempio riportato) nella variabile fc che è il controllo fuzzy nel workspace, per essere poi richiamato nel file Simulink con la medesima variabile fc. In questo modo è sufficiente variare il nome nel file Matlab per poter implementare in modo agevole diversi controllori fuzzy e testarne la loro efficacia.

5.1 Controllo fuzzy lineare e non lineare

A seconda della legge di controllo, un controllore fuzzy può essere lineare e verificabile visualizzando la superficie di controllo che dovrà essere un piano, oppure non lineare qualora non si verifichi tale condizione.
È dimostrabile che se:

- Sono usate per gli ingressi funzioni triangolari con completezza pari a 0.5:
- Le funzioni dei due fuzzy set di ingresso, con intervallo tra -100 e 100 devono essere così definite:
    - N: centrata in -100
    - Z: centrata in 0
    - P: centrata in 100

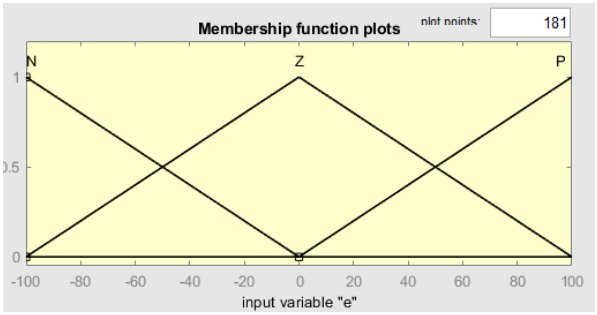


Figura 5-2: fuzzy set input lineare

- Si usa il metodo di Sugeno per l'interferenza
- Per il fuzzy set di uscita, con intervallo tra -200 e 200, i singleton devono essere così definiti:
  - NN: centrato in -200
  - N: centrato in -100
  - Z: centrato in 0
  - P: centrato in 100
  - PP: centrato in 200
- Si sceglie il centro di massa per la defuzzificazione
- Si utilizza una base di regole come quella evidenziata in tabella

fc_0		de		
		N	Z	P
e	N	NN	N	Z
	Z	N	Z	P
	P	Z	P	PP

Tabella 5-1: regole fuzzy lineare

Allora il generico PID è circa lineare, come giustificato dalla superficie delle regole che è un piano.

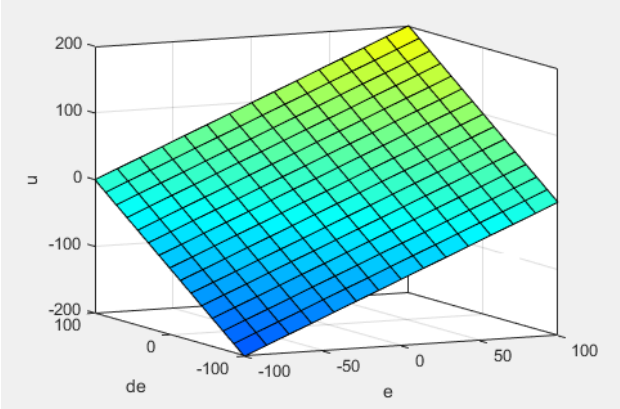


Figura 5-3: superficie fuzzy lineare

Tra le sorgenti di non linearità ci sono: la forma delle funzioni di appartenenza e la base delle regole usate.

Nel caso dell'applicazione al pendolo inverso sono state impiegate delle sorgenti di non linearità per corrispondere meglio alle esigenze di stabilità del controllo. Di seguito verrà analizzato quanto questo controllo si discosti dalla linearità.
Come regole fuzzy è stata implementata la seguente matrice fornita da [6] le cui regole evidenziate in giallo sono quelle variate dal caso precedente.

fc_1		de		
		N	Z	P
e	N	NN	N	N
	Z	N	Z	P
	P	P	P	PP

Tabella 5-2: regole fuzzy non lineari

I fuzzy set di input non subiscono variazione, mentre varia quello di output, infatti passiamo al metodo Mamdani con funzioni triangolari in un intervallo tra -100 e 100.

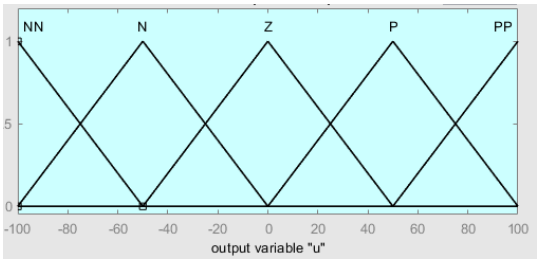
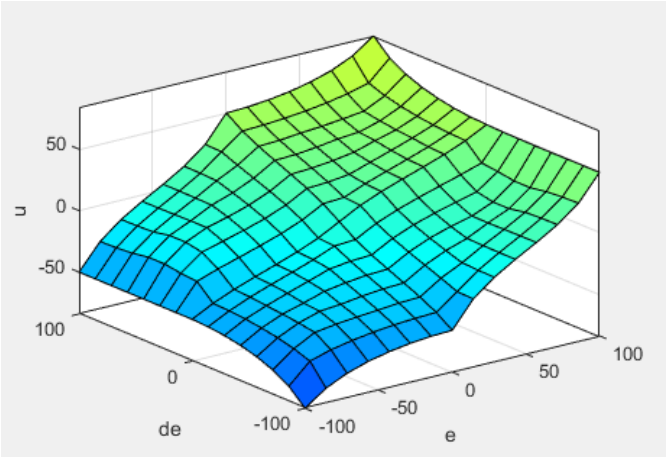


Figura 5-4: fuzzy set output

La superficie ottenuta da questo controllore non è propriamente un piano come si evince in figura e la componente di non linearità non è molto elevata.



Per una migliore analisi sulla non linearità del controllore fuzzy proposto sono state eseguite delle simulazioni su Matlab per consentire una migliore manipolazione dei dati, visto che il tool per implementare la logica fuzzy consente solo una visualizzazione dei dati.

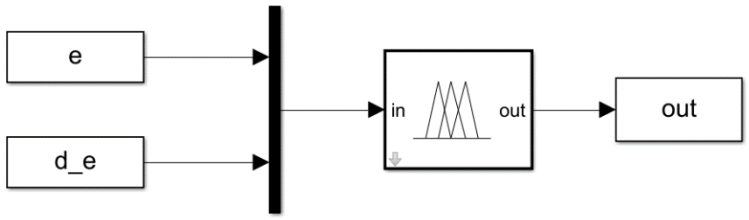


Figura 5-5: verifica linearità

Sono state effettuate tre simulazioni differenti:

1. Nella prima simulazione è stato posto il vettore della derivata dell'errore pari a zero.

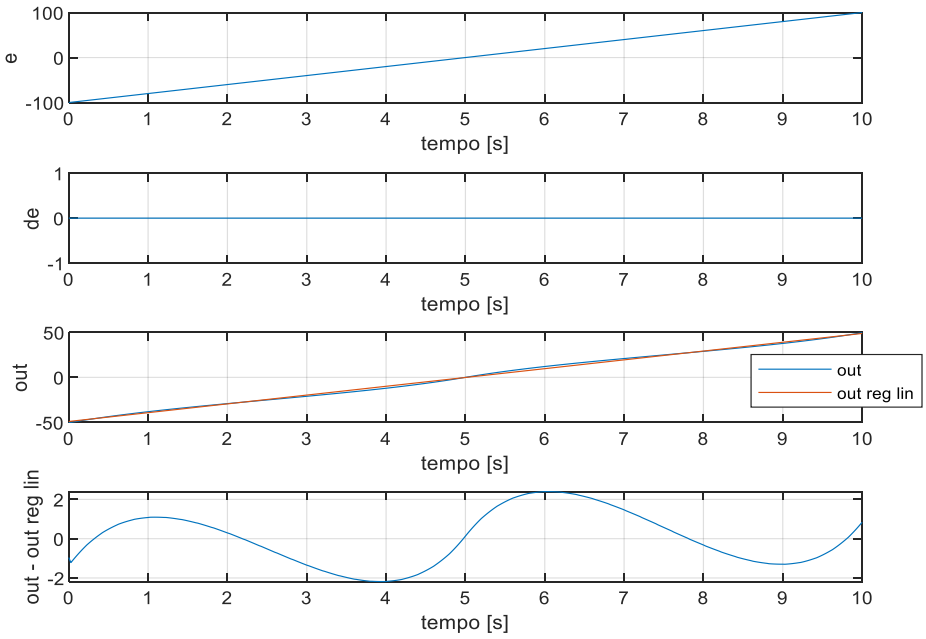


Figura 5-6: linearità con "de"=0

Nel terzo subplot è stato rappresentato l'output della simulazione in blu, mentre in rosso è stata costruita una retta con il metodo dei minimi quadrati. Nel quarto subplot invece si è messa in evidenza la differenza tra i due termini precedenti.

2. Nella seconda simulazione è stato posto il vettore dell'errore pari a zero

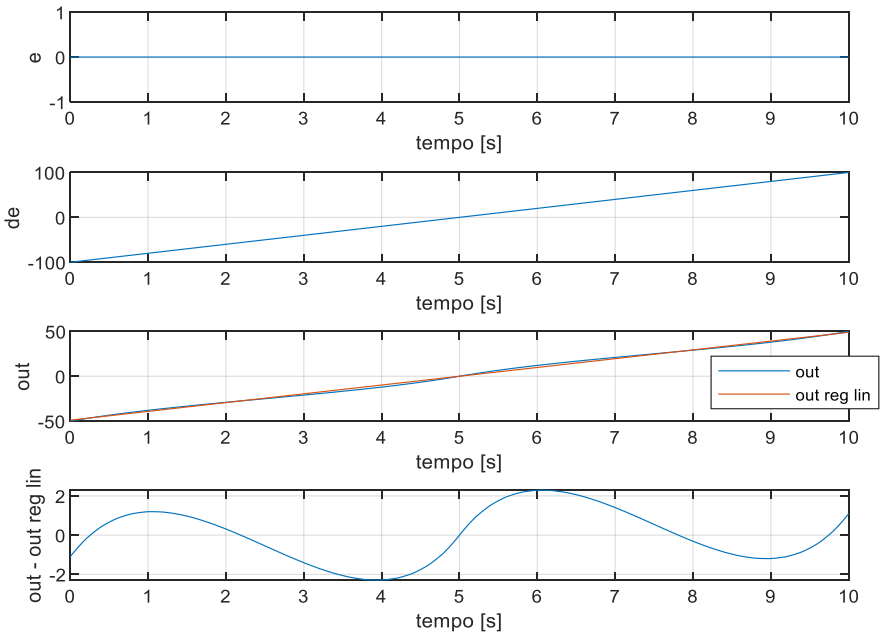


Figura 5-7: linearità con "e"=0

3. Nella terza simulazione sono invece stati fatti variare linearmente sia l'errore sia la derivata dell'errore.

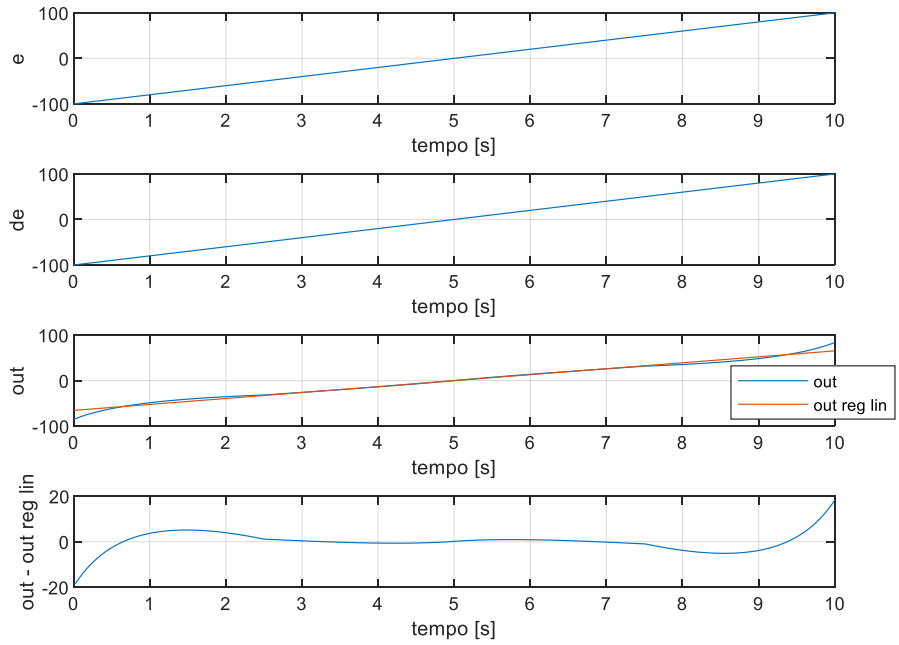


Figura 5-8: linearità con "e" e "de" diversi da zero

Dell'analisi dei tre casi si può osservare come nel campo di lavoro centrale, che è quello che dovrebbe essere maggiormente sfruttato dal controllo, la caratteristica dell'uscita può essere approssimata come lineare, mentre agli estremi del campo di lavoro questa ipotesi è meno veritiera.

5.2 Taratura controllo fuzzy PID

Processo di taratura del controllore generico fuzzy PID, come suggerito da [3]:

1. Nell'anello di controllo viene implementato un controllore PID, il quale viene tarato tramite procedure note in letteratura.
2. Il controllore PID viene sostituito con un fuzzy PID lineare.
3. I guadagni del fuzzy PID vengono scelti tramite equivalenza con quelli del PID.
4. Si rende il fuzzy PID non lineare
5. Si ricalibrano i guadagni, ricordando che:
    a. GE: è legato al tempo di salita e all' overshoot.
    b. GCE: la sua funzione è di ridurre e smorzare l'overshoot.
    c. GIU: per limitare e rimuovere l'errore di regime.
    d. GU: severe a rendere il regolatore più o meno sensibile.

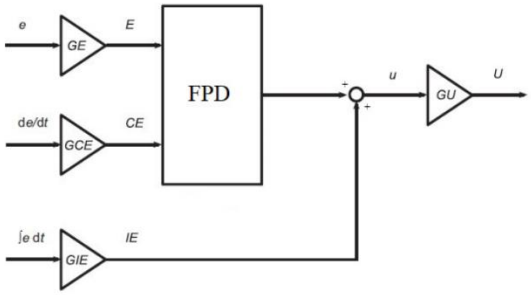


Figura 5-9: es. fuzzy PID

Un'alternativa al metodo proposto può essere una taratura manuale del controllore:

1. GE: viene scelto in modo da coprire l'intero range della variabile considerata.
2. Si impostano GIU=GCE=0 e si calibra GU in modo da ottenere l'andamento desiderato, senza considerare un possibile errore a regime.
3. Si aumenta GU (che è come un guadagno complessivo) e si calibra GCE per ridurre l'overshoot.
4. Si tara GIU per eliminare un possibile errore a regime.
5. Finché GU non presenta il valore massimo possibile la procedura va ripetuta.

È molto importante considerare che non è presente un metodo univoco per la taratura di un controllore fuzzy, in particolare per un sistema altamente instabile, come il pendolo inverso. Nel seguito infatti si è considerato un metodo ibrido tra i due per cercare di tarare i controllori.

6 Controlli Fuzzy PID

Di seguito verranno proposti tre diversi controlli per stabilizzare il pendolo inverso e consentire il raggiungimento della posizione di set voluta:

1. PD+I
2. PD+PI
3. PID-like

Questi tre controllori vengono tarati sfruttando due controllori PID. La generale funzione di trasferimento per un controllore PID è la seguente:

$$G(s) = k_p + \frac{1}{s}k_i + \frac{s\,k_d}{T_f\,s + 1}$$

I controllori per la posizione del carrello e per l'angolo del pendolo risultano essere:

$$G(s)_x = k_{p\_x} + \frac{1}{s}k_{i\_x} + \frac{s\,k_{d\_x}}{T_{f\_x}\,s + 1}$$

$$G(s)_{th} = k_{p\_th} + \frac{1}{s}k_{i\_th} + \frac{s\,k_{d\_th}}{T_{f\_th}\,s + 1}$$

controllore PID x		controllore PID th	
kp_x	-0,000802	kp_th	2,09
Ki_x	-4,68E-06	Ki_th	6,31
Kd_x	-0,0306	Kd_th	0,103
Tf_x	0,00875	Tf_th	0,000875

Tabella 6-1: controllori PID

I valori delle due funzioni di trasferimento PID sono stati reperiti da [5].

6.1 PD+I Fuzzy

In figura, reperita da [3], viene rappresentato il controllo PD+I.

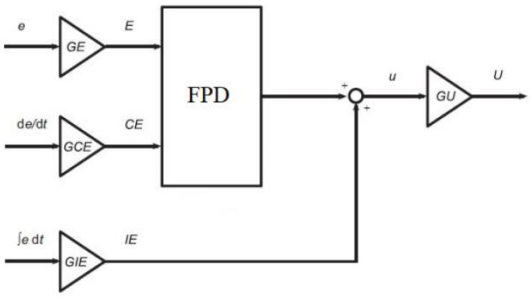


Figura 6-1: PD+I

In Simulink viene implementato nei blocchi di controllo del sistema

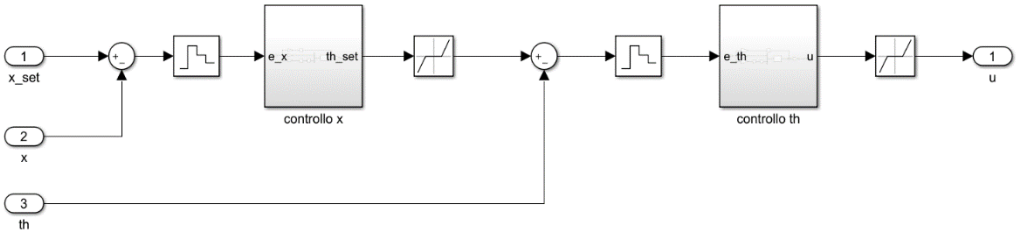


Figura 6-2: blocco controllo PD+I

Il blocchetto Zero-Order Hold all'ingresso dei controlli viene inserito per simulare la lettura della scheda Arduino dai sensori, mentre il blocchetto Dead Zone viene inserito per limitare gli errori numerici, che potrebbero sopraggiungere in fase di simulazione.

Vengono implementati due controlli in serie:

1. Sull'errore di posizione

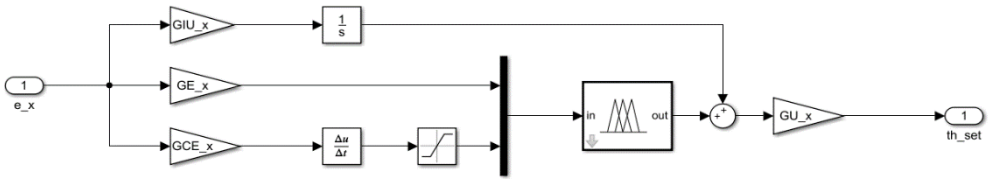


Figura 6-3: controllo PD+I sull'errore posizione

2. Sull'angolo teta

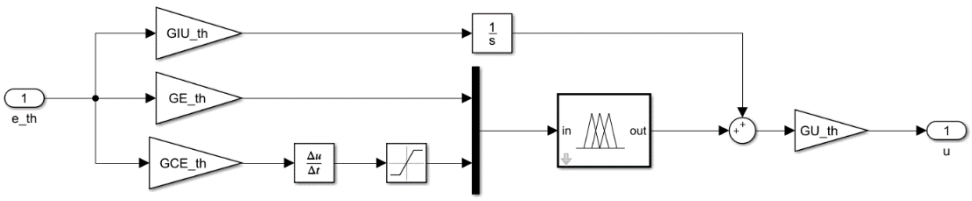


Figura 6-4: controllo PD+I sull'angolo teta

6.1.1 Taratura del controllore

Partendo dall'espressione generica del PD+I fuzzy:

$$u = GU\left(f(e\ GE, \dot{e}\ GCE) + \int e\ GIU\right)$$

Ipotizzando che il controllo fuzzy possa essere considerato lineare:

$$f(e\ GE, \dot{e}\ GCE) \approx e\ GE + \dot{e}\ GCE\ \Rightarrow\ u = GU\left(e\ GE + \dot{e}\ GCE + \int e\ GIU\right)$$

Applicando la trasformata di Laplace:

$$u = GU\left(e\ GE + s\ e\ GCE + \frac{1}{s} e\ GIU\right)$$

$$u = (GU\ GE)\ e + (GU\ GCE)\ s\ e + (GU\ GIU)\ \frac{1}{s}\ e$$

$$G(s) = \frac{u}{e} = (GU\ GE) + (GU\ GCE)\ s + (GU\ GIU)\frac{1}{s}$$

Confrontandola con la funzione di trasferimento del PID:

$$k_p = GU\ GE$$
$$k_i = GU\ GIU$$
$$k_d = GU\ GCE$$

Il parametro GE può essere facilmente ricavato dalla relazione:

$$GE = \frac{fuzzy_{max}}{errore_{max}}$$

dove $fuzzy_{max}$ rappresenta il massimo valore di input del fuzzy set dell'errore, mentre $errore_{max}$ è il massimo valore di errore, in questo modo si riesce a coprire tutto il campo operativo. Per la variabile errore di x: $fuzzy_{max} = 100$ e $errore_{max} = corsa/2 = 0.25$, mentre per l'angolo teta: $fuzzy_{max} = 100$ e $errore_{max} = 20°$ che è la corsa limitata dalla struttura del pendolo.
Riusciamo quindi a ricavare i restanti guadagni:

$$GU = \frac{k_p}{GE}$$

$$GIU = \frac{k_i}{GU}$$
$$GCE = \frac{k_d}{GU}$$

Vengono riportati in tabella i valori dei guadagni ottenuti.

controllore PD+I x		controllore PD+I th	
GE_x	4,000000E+02	GE_th	2,864789E+02
GU_x	-2,005000E-06	GU_th	7,300000E-03
GIU_x	2,334200E+00	GIU_th	8,649195E+02
GCE_x	1,526200E+04	GCE_th	1,411830E+01

Tabella 6-2: parametri controllore PD+I

6.1.2 Tuning dei guadagni

Con i seguenti valori il controllo non risulta ancora essere ottimale perciò, come suggerito precedentemente, si interviene manualmente in ordine sul guadagno: proporzionale, derivativo e integrativo imponendo come set un gradino rappresentativo di 0.1 m all'istante t = 1 s.
In prima istanza si agisce sul controllo proporzionale.

```
GU_x=Kp_x/GE_x*k_prop;
```

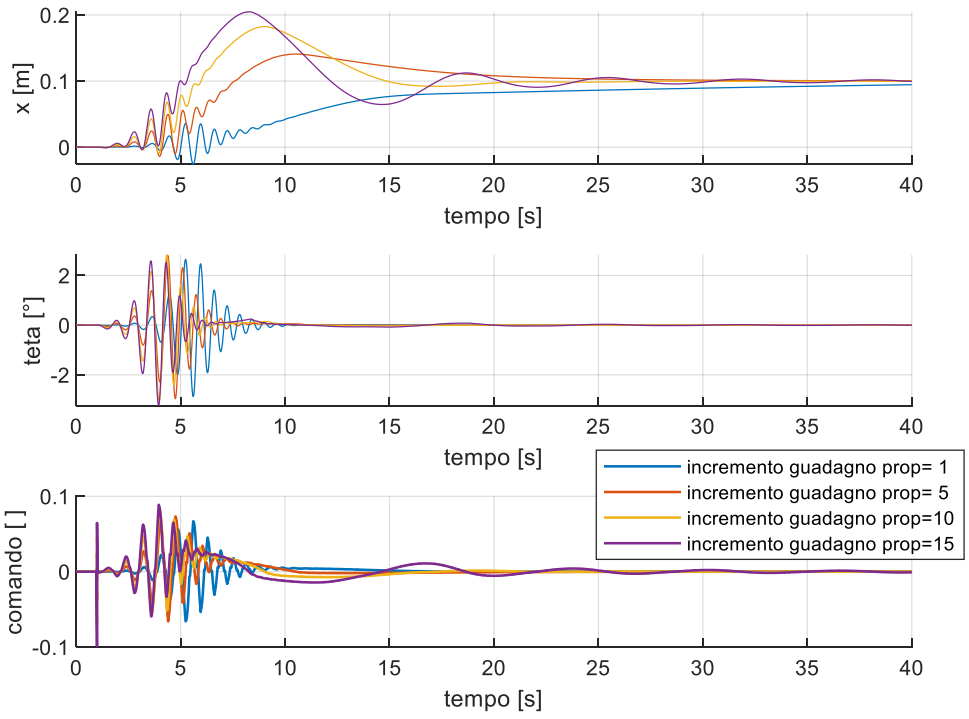


Figura 6-5: guadagno proporzionale, PD+I

Viene quindi scelto un valore di k_prop=10 riducendo in questo modo il tempo necessario ad arrivare a regime senza un eccessivo overshoot.
Per ridurre il valore dell'overshoot si va invece ad agire sul guadagno derivativo.

```
GCE_x=(Kd_x/GU_x)*k_der;
```

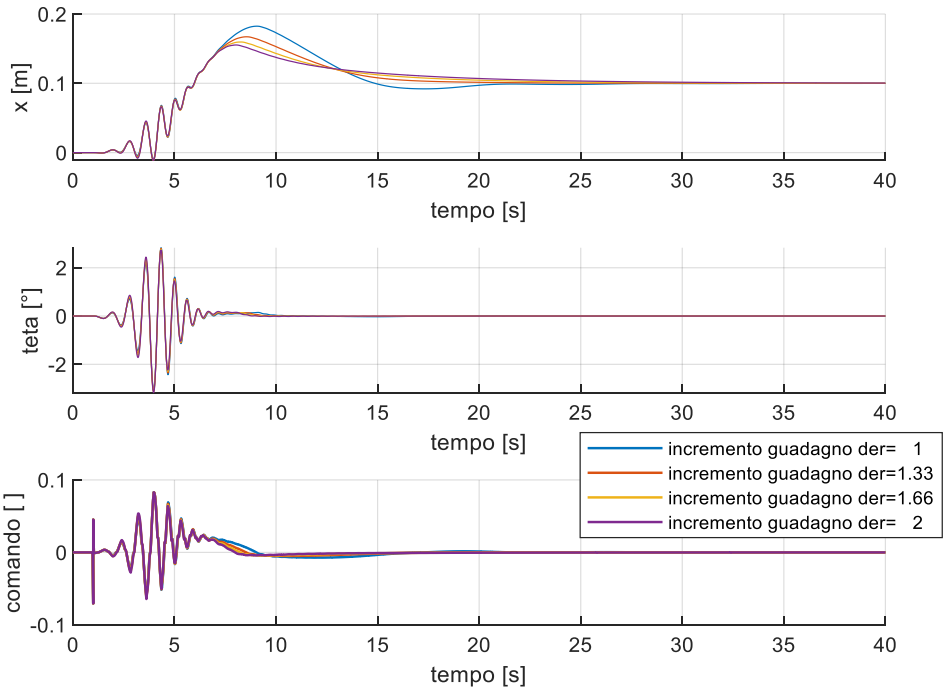


Figura 6-6: guadagno derivativo, PD+I

Viene quindi scelto un valore di k_der=1.33 riducendo in questo modo viene diminuito l'overshoot. Nella figura sottostante viene mostrato uno zoom per giustificare meglio la scelta.

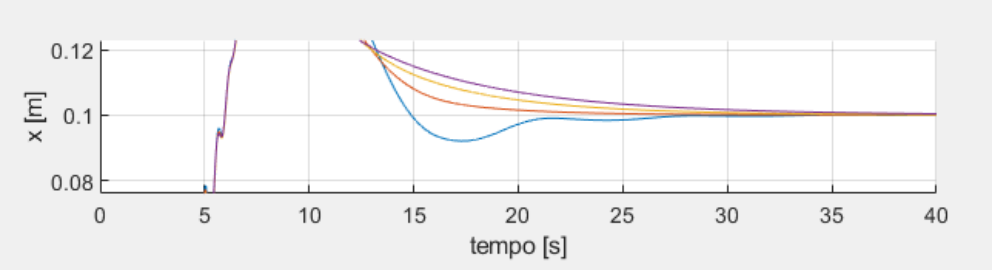


Figura 6-7: zoom guadagno derivativo

Non viene modificato il guadagno integrativo visto che il sistema a regime presenta già un ottimo comportamento.

Vengono riportati in tabella i valori finali del controllo.

controllore PD+I x		controllore PD+I th	
GE_x	4,000000E+02	GE_th	2,864789E+02
GU_x	-2,005000E-05	GU_th	7,300000E-03
GIU_x	2,334000E-01	GIU_th	8,649195E+02
GCE_x	2,029800E+03	GCE_th	1,411830E+01

Tabella 6-3: tuning PD+I

In Matlab vengono implementate le seguenti righe di codice per aggiornare i parametri del controllore rispetto al caso di partenza ottenuto con la linearizzazione:

```
k_prop=10;
k_der=1.33;
k_int=1;
x_max=0.25;
GE_x=100/x_max;
GU_x=(Kp_x/GE_x)*k_prop;
GCE_x=(Kd_x/GU_x)*k_der;
GIU_x=(Ki_x/GU_x)*k_int;
```

In figura è rappresentato l'andamento del feedback con il set con il tuning dei parametri ora ottenuti.

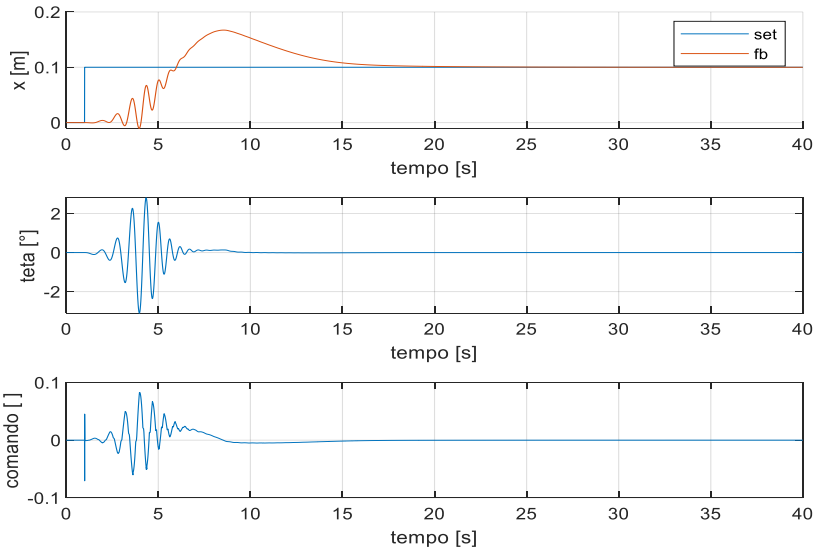


Figura 6-8: set e fb, PD+I

Per ogni prova è possibile anche ottenere i seguenti grafici:

- le pressioni nelle camere
- la forza inviata al carrello e quella che si sviluppa a seguito di urti con i fine corsa.
- I flussi in massa entranti e uscenti nelle varie camere

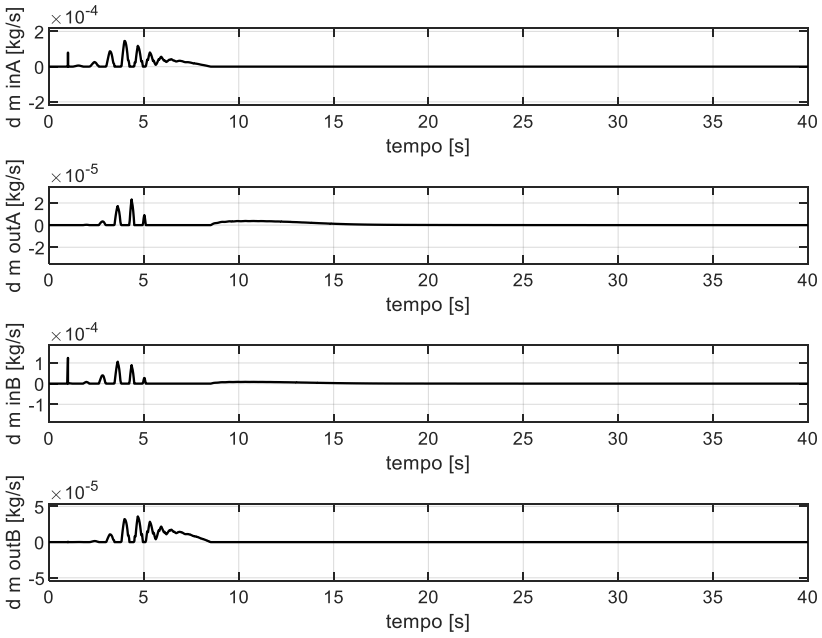


Figura 6-10: flussi in massa, PD+I

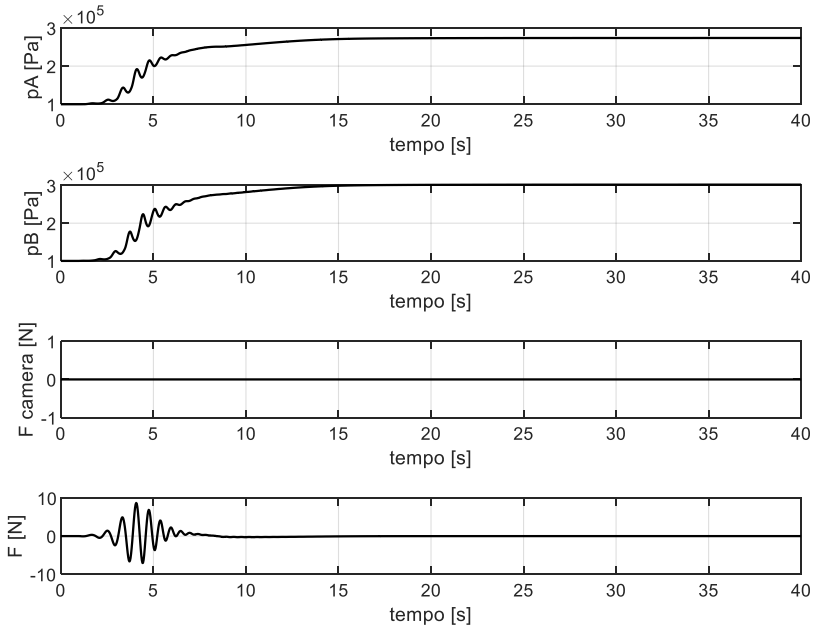


Figura 6-9: pressione e forze, PD+I

Dall'analisi dei grafici il sistema raggiunge la stabilità intorno ai 17 secondi.

Di seguito vengono anche plottati gli andamenti nel caso di un gradino con escursione di 0.05 m, e di un set sinusoidale con ampiezza 0.05 m e frequenza 0.05 Hz.

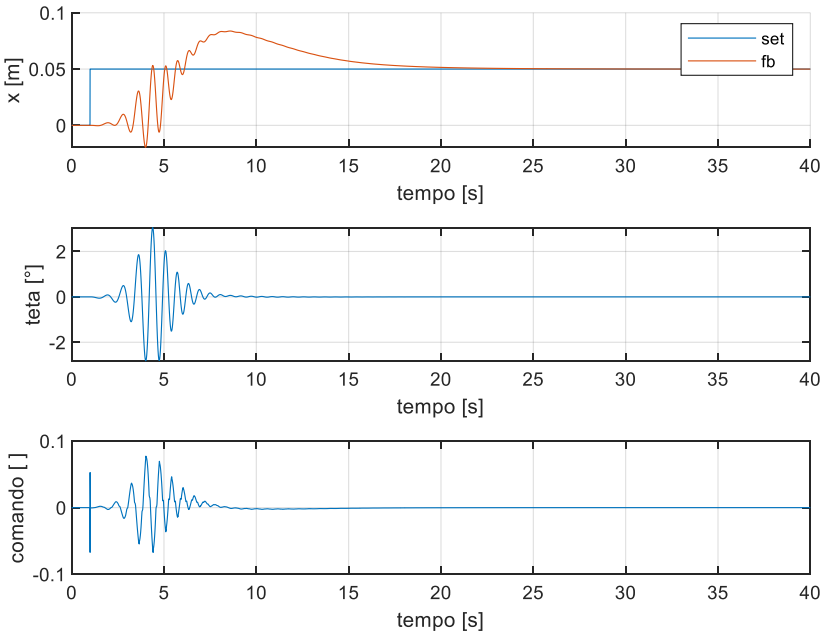


Figura 6-11: altro gradino, PD+I

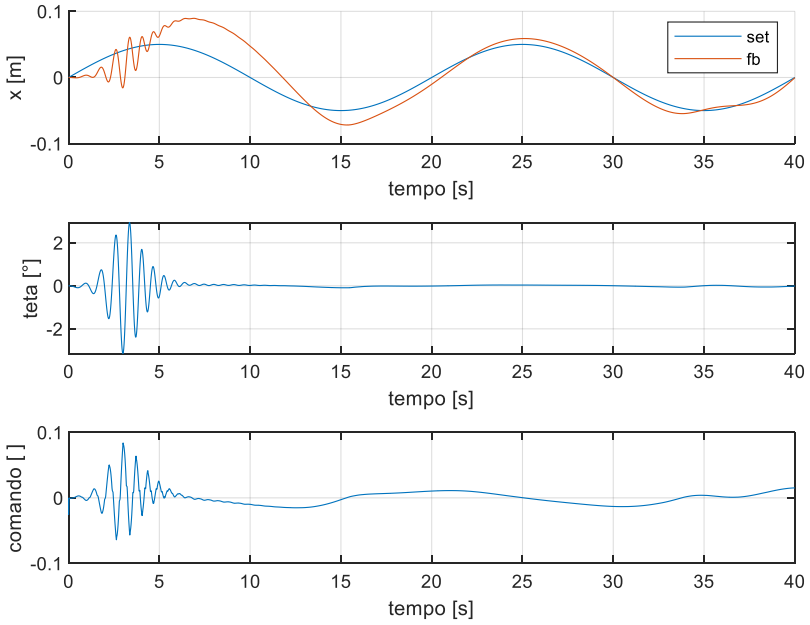


Figura 6-12: sinusoide, PD+I

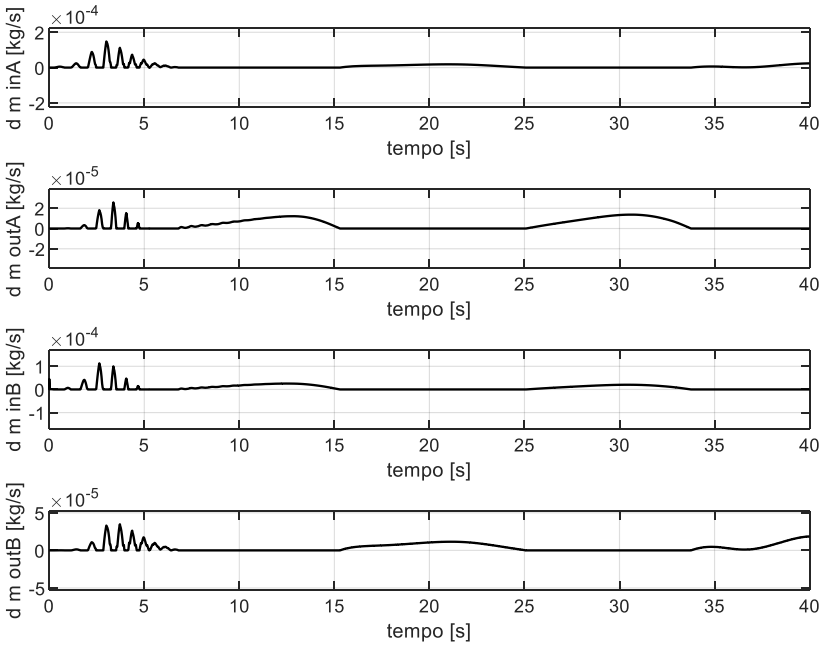


Figura 6-13, flussi con sinusoide, PD+I

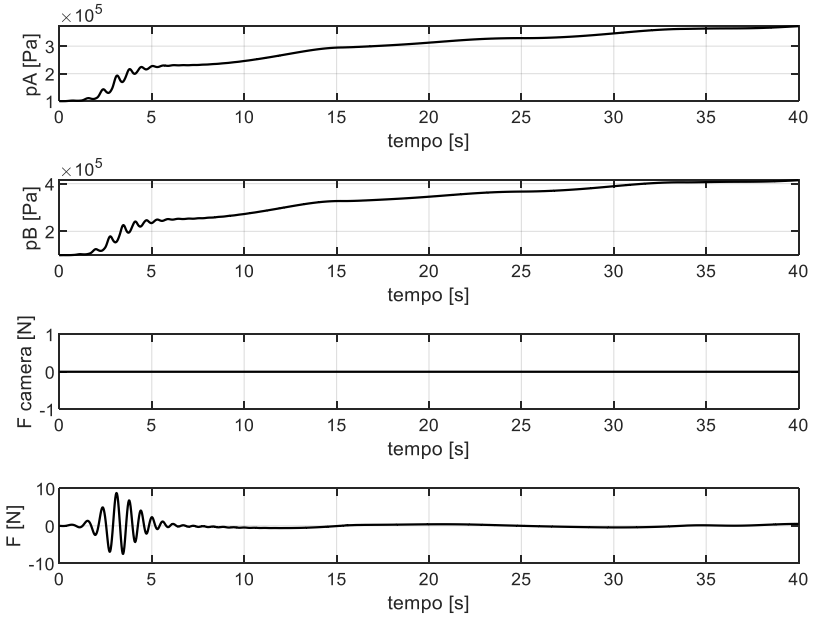


Figura 6-14: p e F con sinusoide, PD+I

A fronte di una maggiore difficolta iniziale il feedback segue bene l'andamento del set dopo circa un periodo, ovvero verso i 20 s. Oltretutto può essere interessante osservare l'andamento dei flussi nelle varie camere del cilindro e quello della forza, che seguono circa un andamento sinusoidale.

Mediante il programma è anche possibile simulare un disturbo esterno a gradino esterno (persistente dopo l'applicazione). Nel caso in esame viene posto un gradino di 5 N al tempo di 25 s quando il sistema risulta essere stabile.

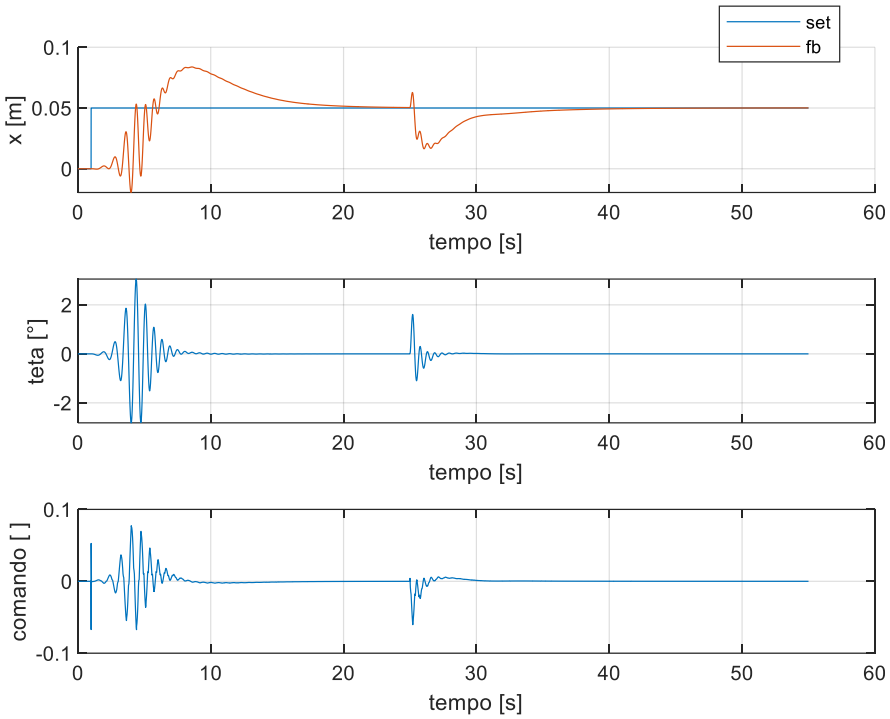


Figura 6-15: disturbo, PD+I

Nonostante l'applicazione del carico esterno il sistema impiega circa quindici secondi per raggiungere nuovamente la stabilità.
Il listato Matlab per ottenere i seguenti grafici è riportato in appendice.

6.2 PID-like Fuzzy

In figura, reperita da [1], viene rappresentato il controllo PDI-like.

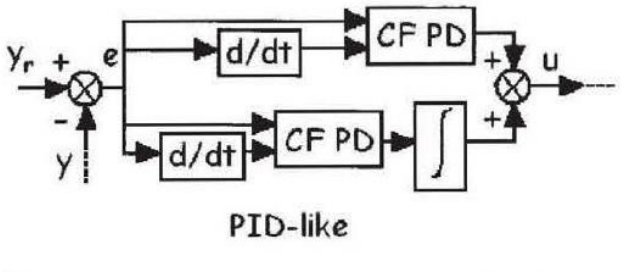


Figura 6-16: PID-like

In Simulink viene implementato nei blocchi di controllo del sistema

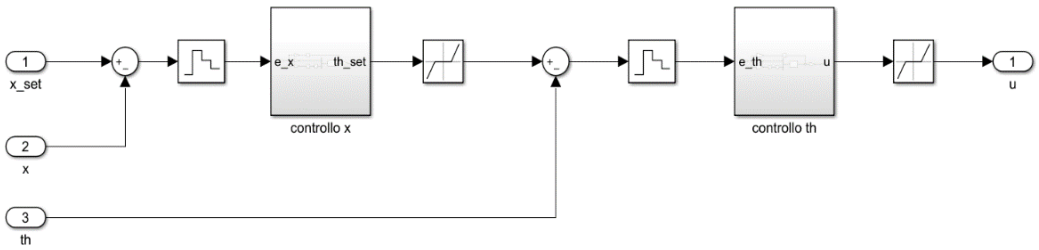


Figura 6-17: blocco controllo PD+I

Il blocchetto Zero-Order Hold all'ingresso dei controlli viene inserito per simulare la lettura della scheda Arduino dai sensori, mentre il blocchetto Dead Zone viene inserito per limitare gli errori numerici, che potrebbero sopraggiungere in fase di simulazione.

Vengono implementati due controlli in serie:

1. Sull'errore di posizione

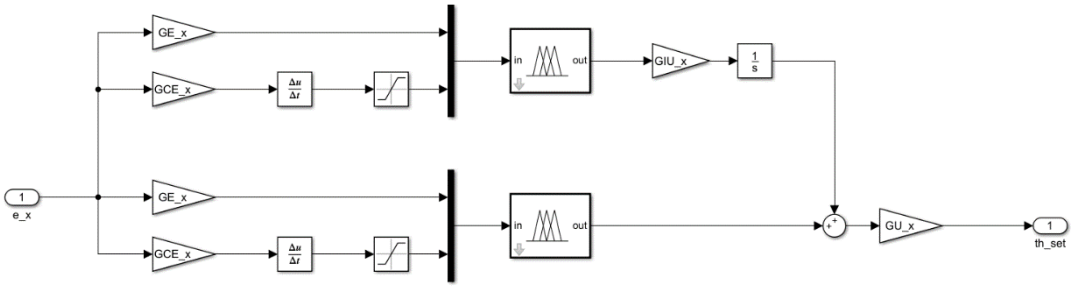


Figura 6-18: controllo PID-like sull'errore di posizione

2. Sull'angolo teta

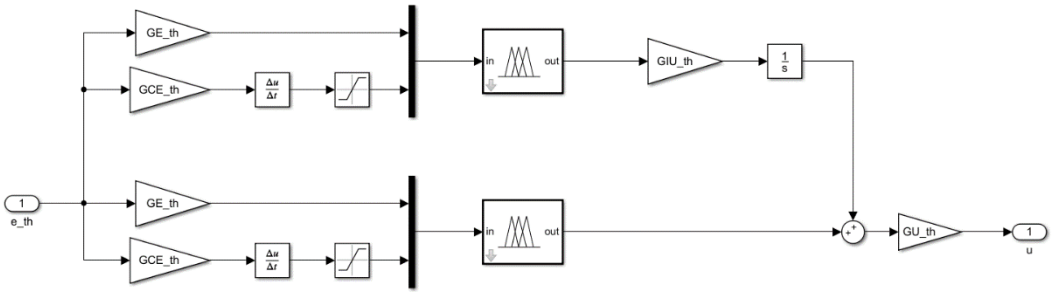


Figura 6-19: controllo PID-like sull'angolo teta

6.2.1 Taratura del controllore

Partendo dall'espressione generica del PD+I fuzzy:

$$u = GU\left(f(e\ GE, \dot{e}\ GCE) + f(e\ GE, \dot{e}\ GCE)\int e\ GIU\right)$$

Ipotizzando che il controllo fuzzy possa essere considerato lineare:

$$f(e\ GE, \dot{e}\ GCE) \approx e\ GE + \dot{e}\ GCE\ \Rightarrow\ u = GU\left(e\ GE + \dot{e}\ GCE + (e\ GE + \dot{e}\ GCE)\int e\ GIU\right)$$

Applicando la trasformata di Laplace:

$$u = GU\left(e\ GE + s\ e\ GCE + (e\ GE + s\ e\ GCE)\ \frac{1}{s}\ e\ GIU\right)$$

$$u = GU\left((GE + GCE\ GIU)e + (GCE)se + (GE\ GIU)\frac{1}{s}e\right)$$

$$G(s) = \frac{u}{e} = (GU\ GE + GU\ GCE\ GIU)\ + (GU\ GCE)s + (GU\ GE\ GIU)\frac{1}{s}$$

Confrontandola con la funzione di trasferimento del PID:

$$k_p = GU\ GE + GU\ GCE\ GIU$$

$$k_d = GU\ GCE$$

$$k_i = GU\ GE\ GIU$$

Il parametro GE può essere facilmente ricavato dalla relazione:

$$GE = \frac{fuzzy_{max}}{errore_{max}}$$

dove $fuzzy_{max}$ rappresenta il massimo valore di input del fuzzy set dell'errore, mentre $errore_{max}$ è il massimo valore di errore, in questo modo si riesce a coprire tutto il campo operativo. Per la variabile errore di x: $fuzzy_{max} = 100$ e $errore_{max} = corsa/2 = 0.25$, mentre per l'angolo teta: $fuzzy_{max} = 100$ e $errore_{max} = 20°$ che è la corsa limitata dalla struttura del pendolo.
Per ricavare i tre guadagni è necessario risolvere una equazione di secondo grado in GU.

$$GU^2\ (GE^2) - GU\left(k_p\ GE\right) + (k_d\ k_i) = 0$$

$$GU = \frac{-\left(-k_p GE\right) \pm \sqrt{\left(k_p\ GE\right)^2 - 4(GE^2)(k_d k_i)}}{2\ (GE^2)}$$

Delle due soluzioni viene scelta la soluzione con segno meno in quanto presenta una migliore risposta. Dal suo valore posso ricavare gli altri guadagni

$$GCE = \frac{k_d}{GU}$$

$$GIU = \frac{k_i}{GU\ GE}$$

Vengono riportati in tabella i valori dei guadagni ottenuti.

controllore PID-like x		controllore PID-like th	
GE_x	4,000000E+02	GE_th	2,864789E+02
GU_x	-1,334100E-06	GU_th	6,000000E-03
GIU_x	8,800000E-03	GIU_th	3,690300E+00
GCE_x	2,293700E+04	GCE_th	1,725670E+01

Tabella 6-4: parametri controllore PID-like

6.2.2 Tuning dei guadagni

Con i seguenti valori il controllo non risulta ancora essere ottimale perciò, come suggerito precedentemente, si interviene manualmente in ordine sul guadagno: proporzionale, derivativo e integrativo imponendo come set un gradino rappresentativo di 0.1 m all'istante t = 1 s.
In prima istanza si agisce sul controllo proporzionale.

```
GU_x=Kp_x/(2*GE_x)*k_prop;
```

Viene quindi scelto un valore di k_prop=8 e si riduce in questo modo il tempo necessario ad arrivare a regime senza un eccessivo overshoot.

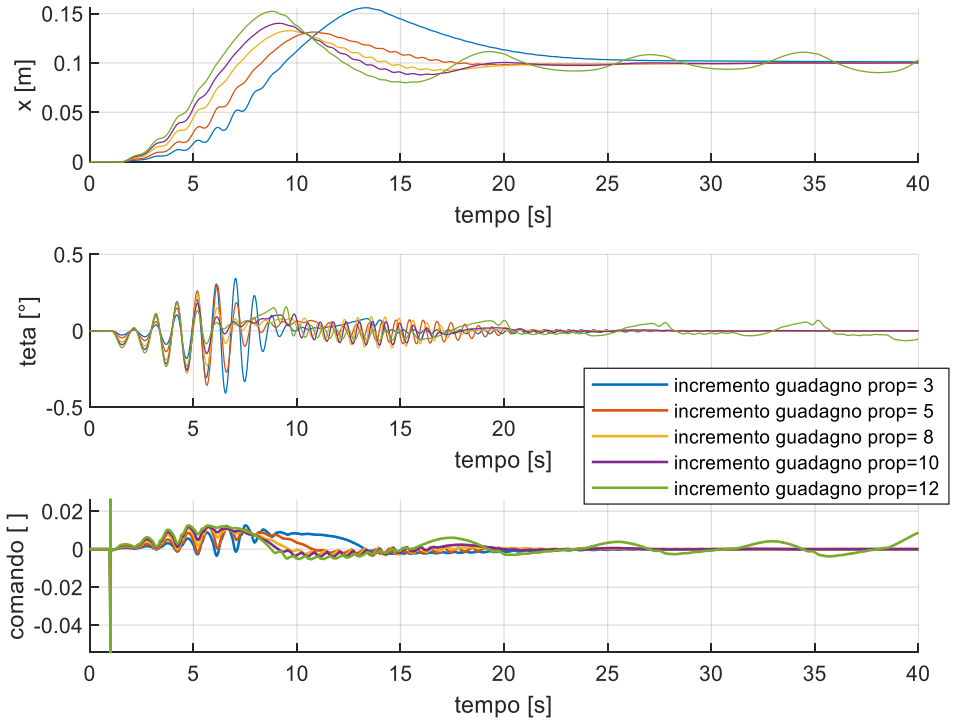


Figura 6-20: guadagno proporzionale, PD+PI

Per ridurre il valore dell'overshoot si va invece ad agire sul guadagno derivativo.

Viene riportata l'espressione del guadagno derivativo:

```
GCE_x=( Kd_x/GU_x )*k_der;
```

Viene quindi scelto un valore di k_der=1.66 e riducendo in questo modo viene diminuito l'overshoot. Nella figura sottostante viene mostrato uno zoom per giustificare meglio la scelta.

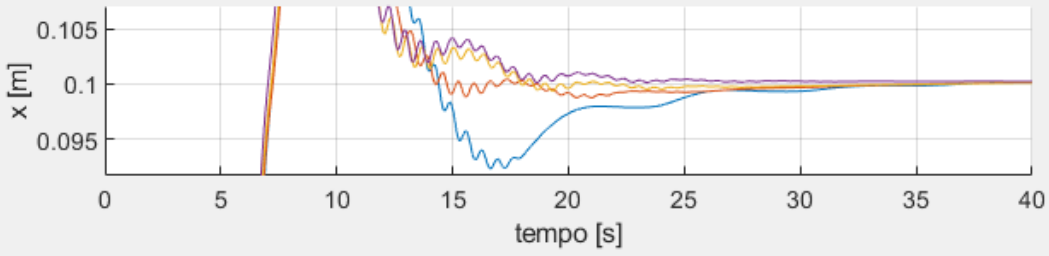


Figura 6-21: zoom guadagno derivativo

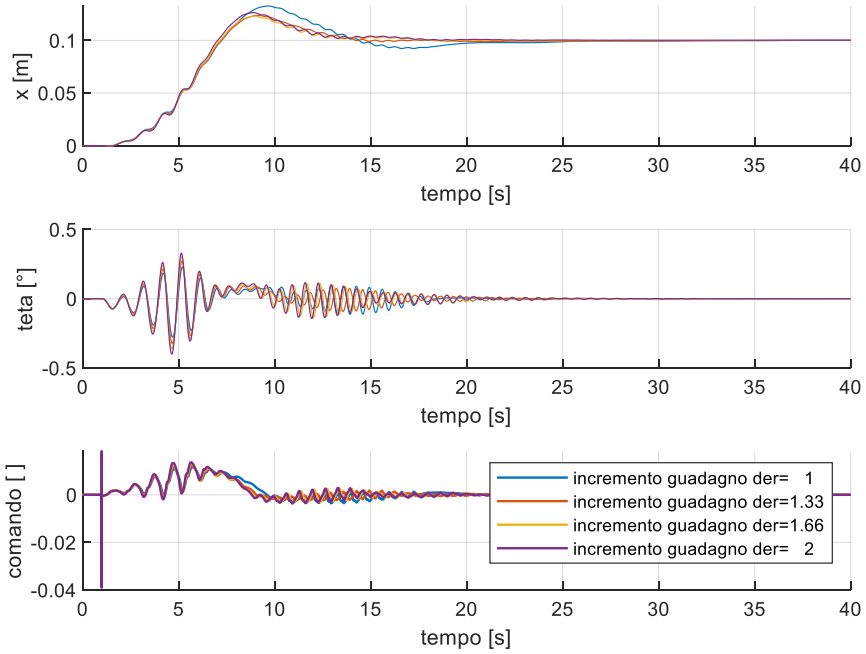


Figura 6-22: guadagno derivativo, PD+PI

Non viene modificato il guadagno integrativo visto che il sistema a regime presenta già un ottimo comportamento.

Vengono riportati in tabella i valori finali del controllo.

controllore PD+I x		**controllore PD+I th**	
GE_x	4,000000E+02	GE_th	2,864789E+02
GU_x	-8,020000E-06	GU_th	6,000000E-03
GIU_x	5,835000E-01	GIU_th	3,690300E+00
GCE_x	6,333700E+03	GCE_th	1,725670E+01

Tabella 6-5: tuning PD+PI

In Matlab vengono implementate le seguenti righe di codice per aggiornare i parametri del controllore, rispetto al caso di partenza ottenuto con la linearizzazione:

```
k_prop=8;
k_der=1.66;
k_int=1;
x_max=0.25;
GE_x=100/x_max;
GU_x=Kp_x/(2*GE_x)*k_prop;
GCE_x=(Kd_x/GU_x)*k_der;
GIU_x=(Ki_x/GU_x)*k_int;
```

In figura è rappresentato l'andamento del confronto con il set e con il tuning dei parametri ora ottenuto.

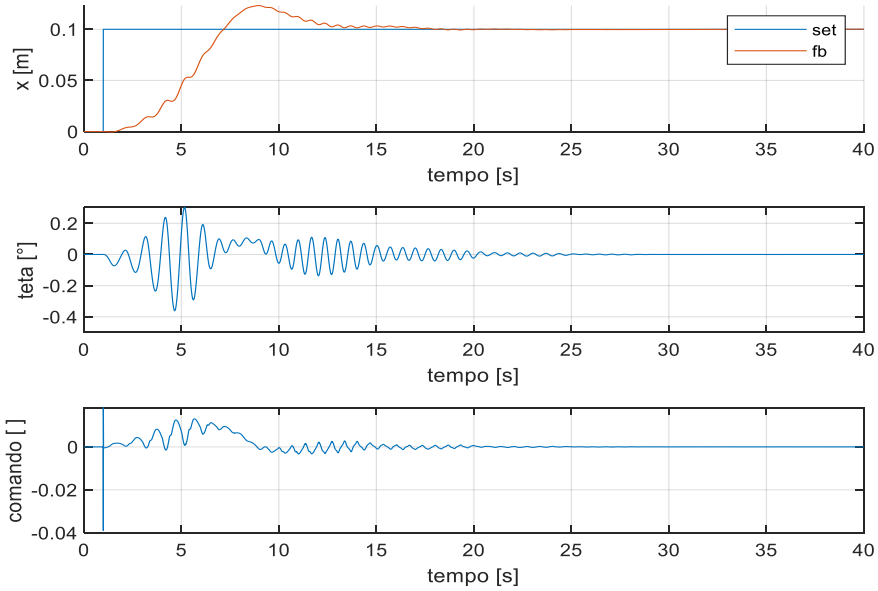


Figura 6-23: set e fb, PD+PI

Dall'analisi dei grafici il sistema raggiunge la stabilità intorno ai 25 secondi.
Vengono plottatati gli andamenti nel caso di un gradino con ampiezza di 0.05 m.

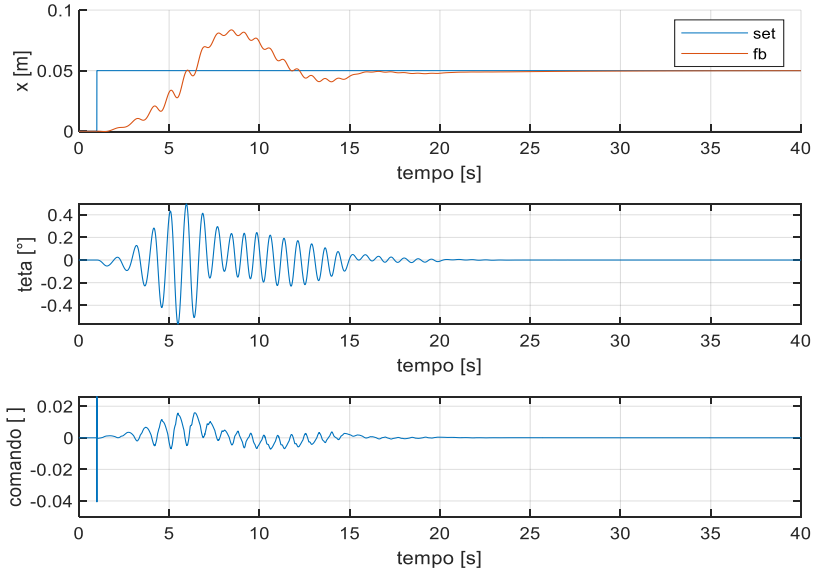


Figura 6-24: altro gradino PD+PI

Viene mostrato l'andamento anche con un set sinusoidale con ampiezza 0.03 m e frequenza 0.05 Hz. La sinusoide rispetto al caso PD+I presenta un'ampiezza minore, dato che con 0.05 m il sistema non riusciva a seguire il set diventando instabile dopo circa tre periodi.

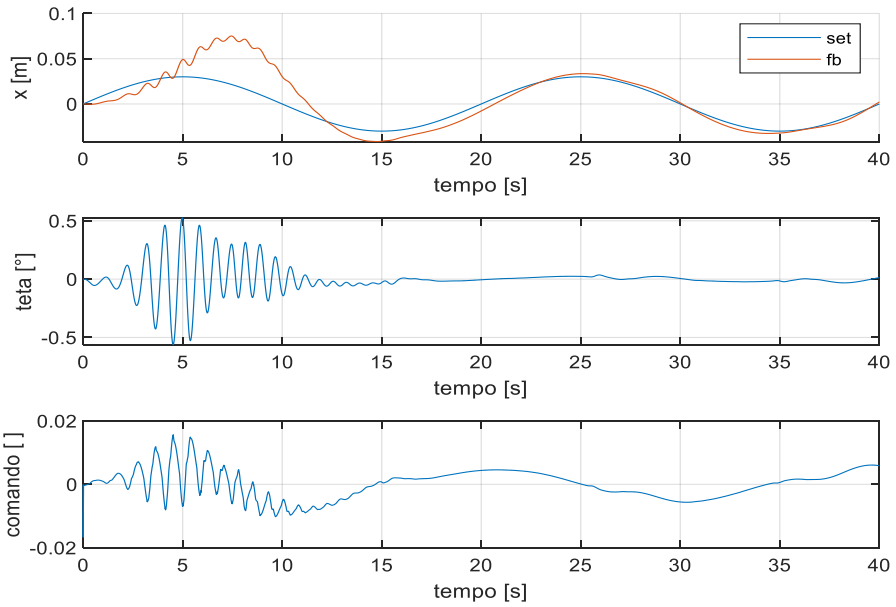


Figura 6-25: sinusoide, PD+PI

A fronte di una maggiore difficolta iniziale, il feedback segue bene l'andamento del set dopo circa un periodo, ovvero verso i 20 s.

Mediante il programma è anche possibile simulare un disturbo esterno a gradino esterno (persistente dopo l'applicazione). Nel caso in esame viene posto un gradino di 2 N al tempo di 25 s quando il sistema risulta essere stabile. Rispetto al controllore PD+I è stato necessario abbassare l'ampiezza del gradino, in quanto il sistema diventava instabile con un disturbo di 5N. Con l'applicazione del carico esterno il sistema impiega circa quindici secondi per raggiungere nuovamente la stabilità.

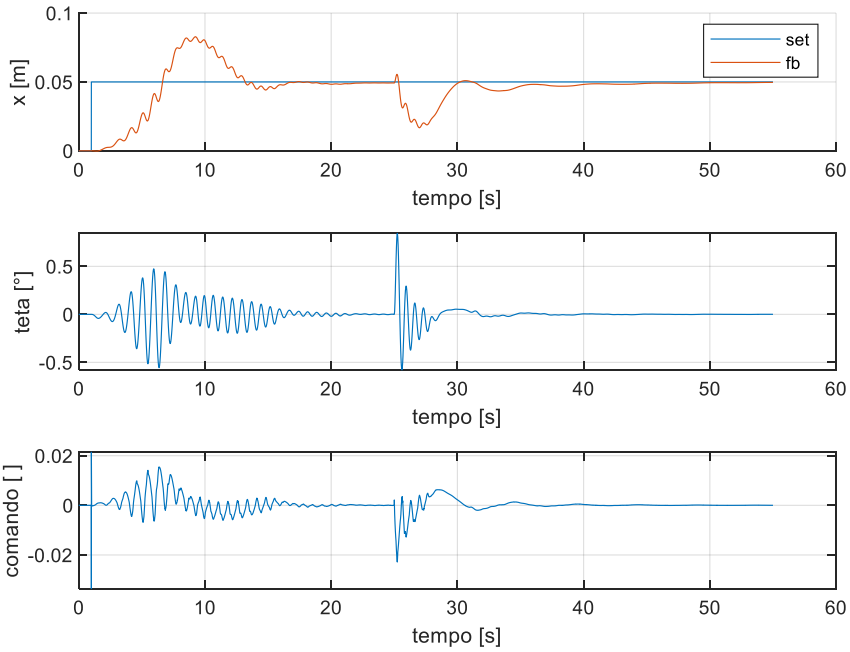


Figura 6-26: disturbo, PD+PI

Il listato Matlab per ottenere i seguenti grafici è riportato in appendice.

6.3 PD+PI Fuzzy

In figura, reperita da [1], viene rappresentato il controllo PD+PI.

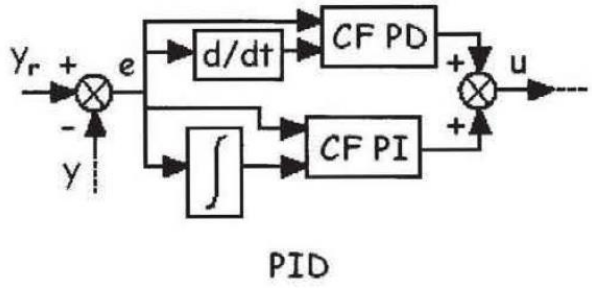


Figura 6-27: PD+PI

In Simulink viene implementato nei blocchi di controllo del sistema

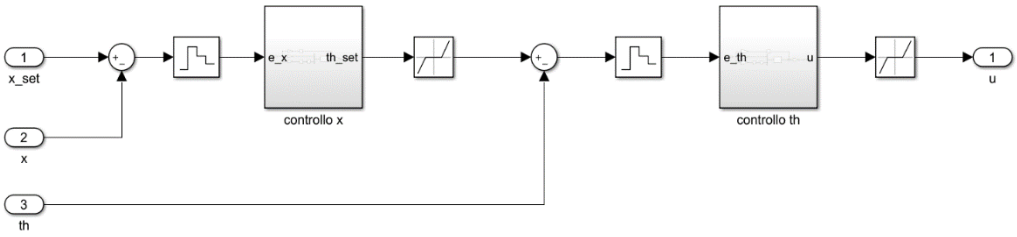


Figura 6-28: blocco controllo PD+PI

Il blocchetto Zero-Order Hold all'ingresso dei controlli viene inserito per simulare la lettura della scheda Arduino dai sensori, mentre il blocchetto Dead Zone viene inserito per limitare gli errori numerici, che potrebbero sopraggiungere in fase di simulazione.

Vengono implementati due controlli in serie:

1. Sull'errore di posizione

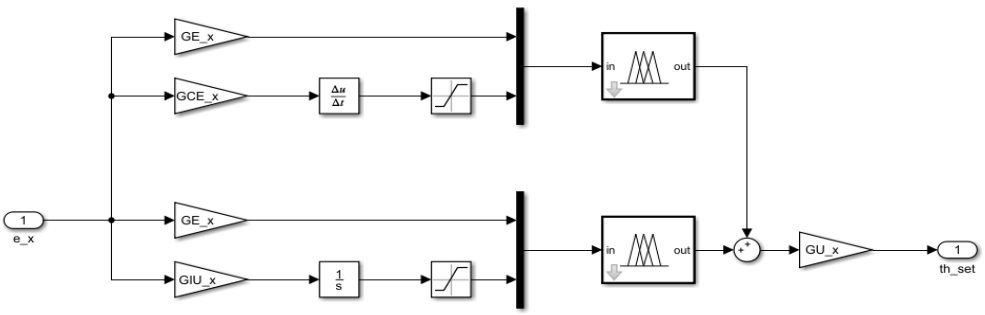


Figura 6-29: controllo PD+PI sull'errore di posizione

2. Sull'angolo teta

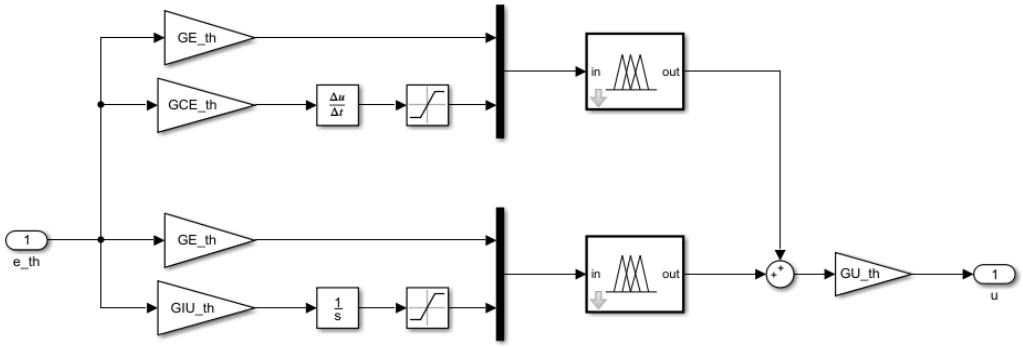


Figura 6-30: controllo PD+PI sull'angolo teta

6.3.1 Taratura del controllore

Partendo dall'espressione generica del PD+I fuzzy:

$$u = GU\left(f(e\ GE, \dot{e}\ GCE) + f\left(e\ GE, \int e\ GIU\right)\right)$$

Ipotizzando che il controllo fuzzy possa essere considerato lineare:

$$f(e\ GE, \dot{e}\ GCE) \approx e\ GE + \dot{e}\ GCE\ \Rightarrow\ u = GU\left(e\ GE + \dot{e}\ GCE + \int e\ GIU\right)$$

$$f(e\ GE, \dot{e}\ GCE) \approx e\ GE + \dot{e}\ GCE$$

$$f\left(e\ GE, \int e\ GIU\right) \approx e\ GE + \int e\ GIU$$

da cui

$$u = GU\left(e\ GE + \dot{e}\ GCE + e\ GE + \int e\ GIU\right)$$

Applicando la trasformata di Laplace:

$$u = GU\left(e\ GE + s\ e\ GCE + e\ GE + \frac{1}{s}\ e\ GIU\right)$$

$$u = (2\ GU\ GE)\ e + (GU\ GCE)\ s\ e + (GU\ GIU)\ \frac{1}{s}\ e$$

$$G(s) = \frac{u}{e} = (2\ GU\ GE) + (GU\ GCE)\ s + (GU\ GIU)\ \frac{1}{s}$$

Confrontandola con la funzione di trasferimento del PID:

$$k_p = 2\ GU\ GE$$

$$k_i = GU\ GIU$$

$$k_d = GU\ GCE$$

Il parametro GE può essere facilmente ricavato dalla relazione:

$$GE = \frac{fuzzy_{max}}{errore_{max}}$$

dove $fuzzy_{max}$ rappresenta il massimo valore di input del fuzzy set dell'errore, mentre $errore_{max}$ è il massimo valore di errore, in questo modo si riesce a coprire tutto il campo operativo. Per la variabile errore di x: $fuzzy_{max} = 100$ e $errore_{max} = corsa/2 = 0.25$, mentre per l'angolo teta: $fuzzy_{max} = 100$ e $errore_{max} = 20°$ che è la corsa limitata dalla struttura del pendolo.
Riusciamo quindi a ricavare i restanti guadagni:

$$GU = \frac{k_p}{2\,GE}$$
$$GIU = \frac{k_i}{GU}$$
$$GCE = \frac{k_d}{GU}$$

Vengono riportati in tabella i valori dei guadagni ottenuti.

controllore PD+PI x		controllore PD+PI th	
GE_x	4,000000E+02	GE_th	2,864789E+02
GU_x	-1,002500E-06	GU_th	3,600000E-03
GIU_x	4,668300E+00	GIU_th	1,729800E+03
GCE_x	3,052400E+04	GCE_th	2,823670E+01

Tabella 6-6: parametri controllore PD+PI

6.3.2 Tuning dei guadagni

Con i seguenti valori il controllo non risulta ancora essere ottimale perciò, come suggerito precedentemente, si interviene manualmente in ordine sul guadagno: proporzionale, derivativo e integrativo imponendo come set un gradino rappresentativo di 0.1 m all'istante t = 1 s.

Con questo tipo di controllo l'analisi è risultata essere molto più complessa rispetto agli altri due casi, necessitando di un aggiornamento ciclico dei vari guadagni, con un'analisi comunque molto simile alla precedente. In Matlab vengono implementate le seguenti righe di codice per aggiornare i parametri del controllore rispetto al caso di partenza ottenuto con la linearizzazione.

```
k_prop=25;
k_der=1.66;
k_int=1/5;
x_max=0.25;
GE_x=100/x_max;
GU_x=k_prop*( -(-Kp_x*GE_x) - sqrt( (Kp_x*GE_x)^2-4*GE_x^2*Kd_x*Ki_x ) )/ ( 2*GE_x^2 ) ;
GCE_x=k_der*( Kd_x/GU_x );
GIU_x=k_int*( Ki_x/( GU_x*GE_x ) );
```

Vengono riportati in tabella i valori finali del controllo.

controllore PD+I x		controllore PD+I th	
GE_x	4,000000E+02	GE_th	2,864789E+02
GU_x	-3,335200E-05	GU_th	3,600000E-03
GIU_x	7,016000E-01	GIU_th	1,729800E+03
GCE_x	1,523000E+03	GCE_th	2,823670E+01

Tabella 6-7: tuning, PID like

In figura è rappresentato l'andamento del confrontato con il set e con il tuning dei parametri ora ottenuto.

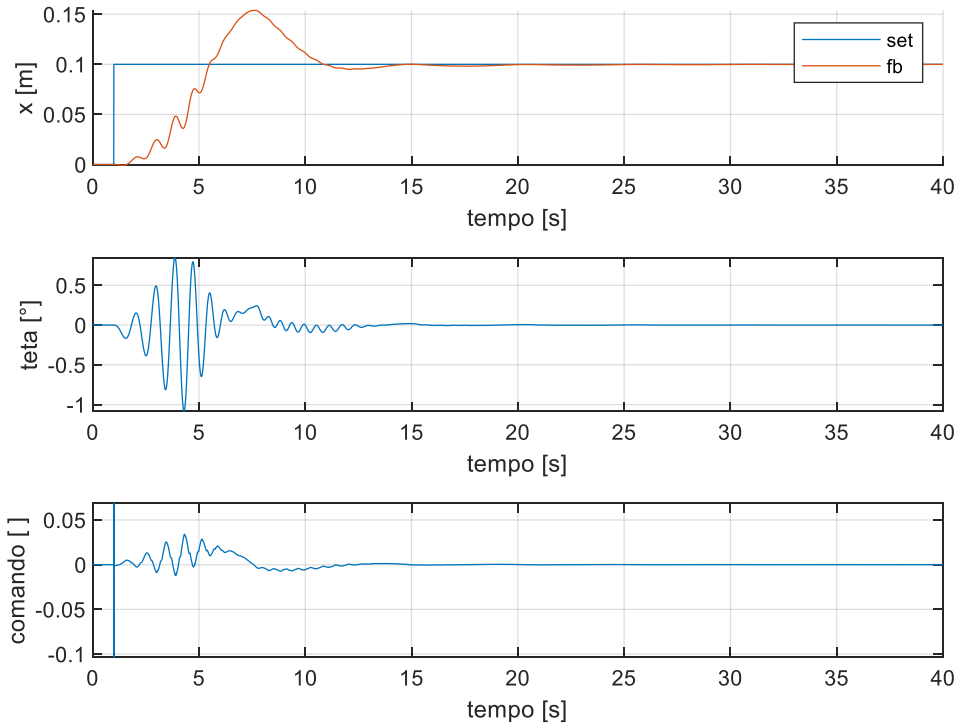


Figura 6-31: set e fb, PID like

Dall'analisi dei grafici il sistema raggiunge la stabilità intorno ai 15 secondi.
Di seguito vengono anche plottati gli andamenti nel caso di un gradino con escursione di 0.05 m, e di un set sinusoidale con ampiezza 0.03 m e frequenza 0.05 Hz, come nel caso del controllore PD+PI visto che il controllore non risultava stabile con un'ampiezza di 0.05 m.

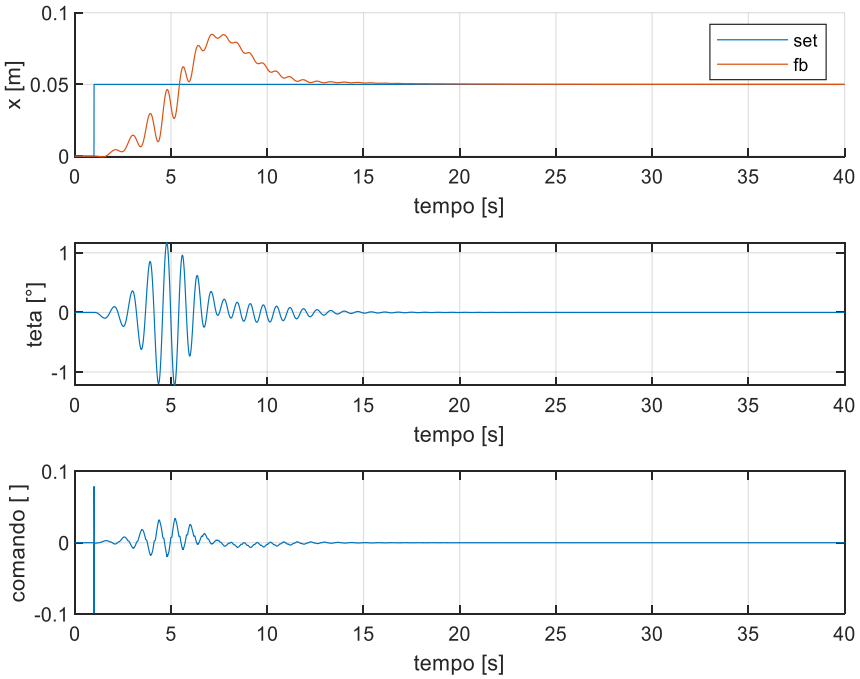


Figura 6-33: altro gradino, PID like

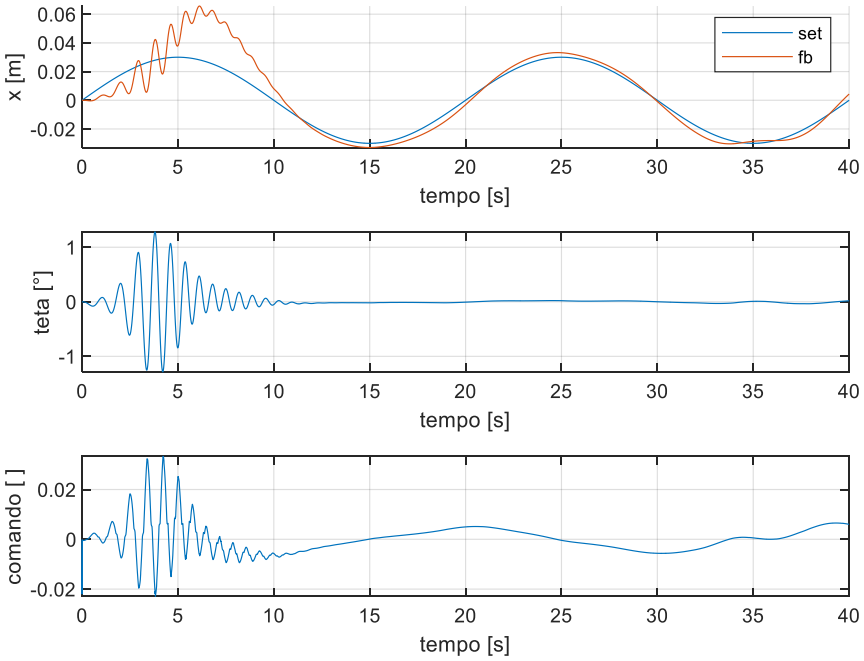


Figura 6-32: sinusoide, PID like

A fronte di una maggiore difficolta iniziale il feedback segue bene l'andamento del set dopo circa mezzo periodo, ovvero verso i 10 s.
Mediante il programma è anche possibile simulare un disturbo esterno a gradino esterno (persistente dopo l'applicazione). Nel caso in esame viene posto un gradino di 3 N al tempo di 25 s quando il sistema risulta essere stabile. Rispetto al controllore PD+I è stato necessario abbassare il livello del gradino come per il PD+PI, in quando il sistema diventava instabile con un disturbo di 5N.

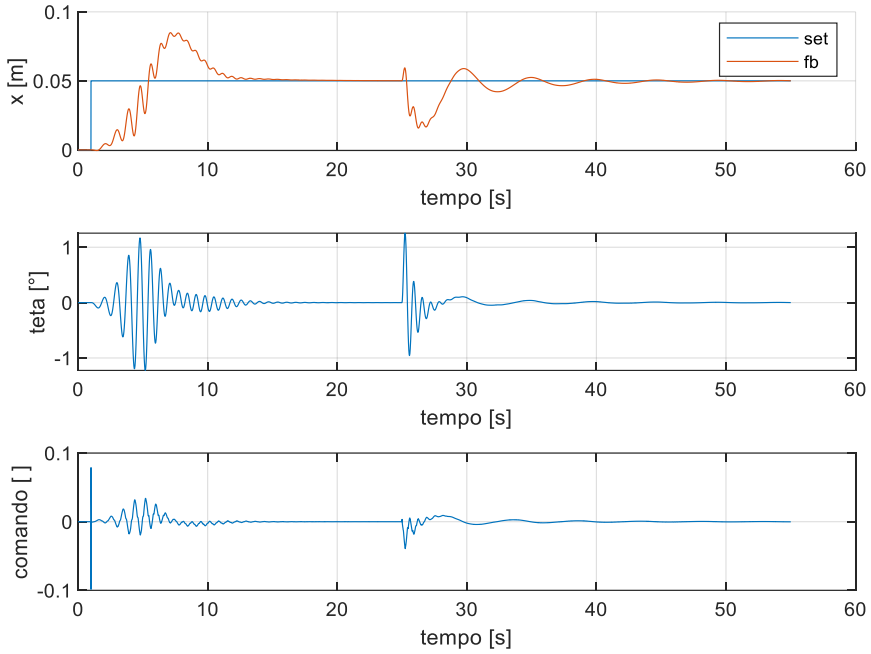


Figura 6-34: disturbo PID like

Nonostante l'applicazione del carico esterno il sistema impiega circa venticinque secondi per raggiungere nuovamente la stabilità.

Il listato Matlab per ottenere i seguenti grafici è riportato in appendice.

6.4 Confronto controllori fuzzy

Viene effettuato un confronto tra i controllori fuzzy precedentemente analizzati, per stabilire quale sia conveniente implementare dal punto di vista teorico sulla scheda Arduino.

Tramite il comando "save" di Matlab vengono creati dei file di estensione .m che contengono le variabili usate per il confronto: x, teta, e u. Questo comando è posizionato al fondo del codice rappresentativo di ogni controllore, dopo aver svolto la simulazione su file Simulink. Viene riportato un comando di salvataggio a titolo di esempio.

```
save(['sim_save',filesep,'PD_plus_I_gradino1.mat'],'x','th','u','tt','x_set');
save(['sim_save',filesep,'PD_plus_I_gradino2.mat'],'x','th','u','tt','x_set');
```

sim_save è la cartella dove viene salvato il file, mentre 'filesep' serve a definire successivamente il nome del file di salvataggio. Vengono effettuati due differenti salvataggi per ogni file, uno per il gradino di ampiezza 0.05 m e uno per quello di 0.1 m.

Nel file di confronto per caricare le variabili salvate nel workspace si usa il comando "load" di Matlab. Il contenuto delle variabili viene poi caricato all'interno delle matrici, dove saranno caricati anche i valori degli altri controllori, per poi essere plottati mediante un ciclo for. Vengono riportate le righe di codice per il controllore PD+I.

```
load(['sim_save',filesep,'PD_plus_I_gradino1.mat']);
% controllo: PD+I con fc 1
cont=cont+1;
% salvo valori nelle matrici
uu(:,cont)=u;
xx(:,cont)=x;
tth(:,cont)=th;
t(:,cont)=tt;
legenda(cont,1)="PD+I";
legenda1(cont,1)="set";
legenda1(cont+1,1)="PD+I";
```

Il codice completo per il confronto viene riportato in appendice.

In figura è mostrato il confronto sul gradino di ampiezza 0.1 m e 0.05 m.

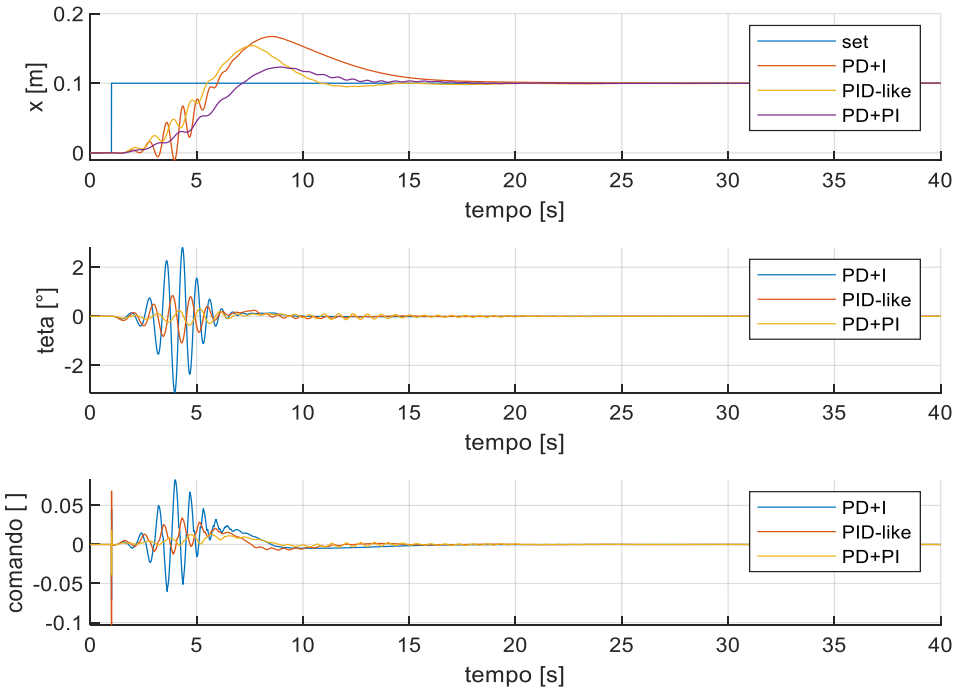


Figura 6-35: confronto gradino 0.1m

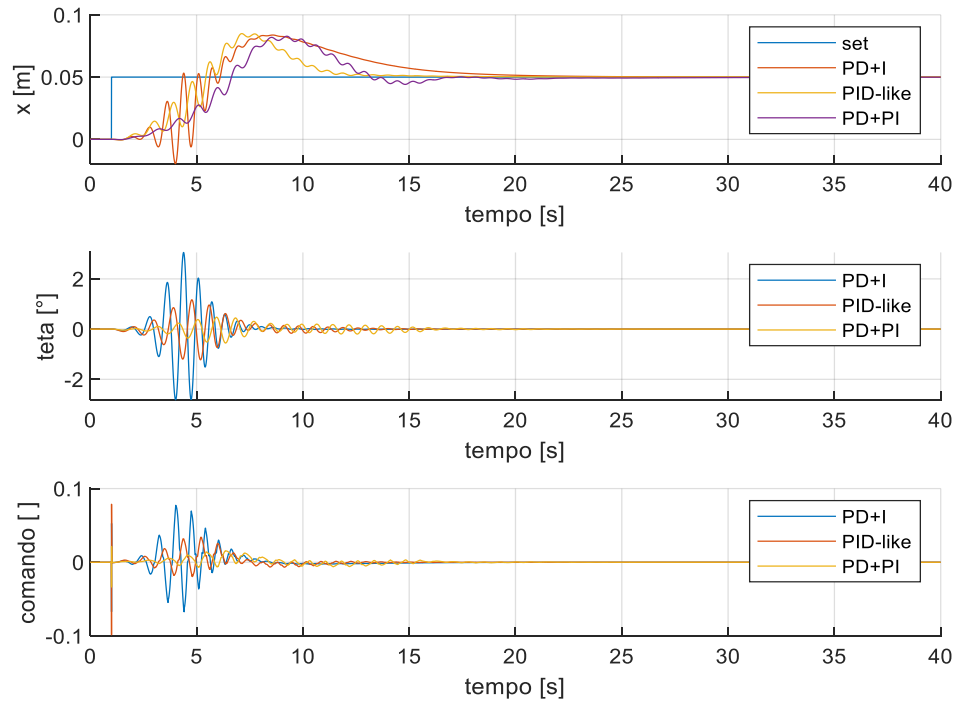


Figura 6-36: confronto gradino 0.05m

Il PD+I è quello che genera comandi u di maggiore entità, di conseguenza angoli teta maggiori, oltretutto è il controllo che presenta una maggiore sovraelongazione e un tempo di assestamento leggermente maggiore degli altri due controlli. Mentre, come vantaggio presenta in entrambi i gradini un segnale di posizione molto più pulito “liscio”, essendo un controllo più stabile.

Non va comunque scordato che il PD+I è il controllore che consente di seguire onde sinusoidali di maggiore ampiezza, ovvero taglia dopo in frequenza (essendo infatti controllo non lineare conta anche l’ampiezza del segnale oltre che la frequenza) e riesce a resistere a disturbi impulsivi di entità maggiore rispetto agli altri due controllori.

Alla luce di queste osservazioni il controllore PD+I può essere considerato il controllore con caratteristiche migliori. Non escludo comunque la possibilità che con un tuning dei parametri differente o con un altro controllo fuzzy, possa risultare più vantaggioso usare un altro controllore.

6.5 Confronto controlli fuzzy

Viene di seguito proposto un confronto variando il controllore fuzzy all'interno del controllo PD+I.

Oltre al controllo fino ad ora implementato fc_1, si è scelto di provare ad ampliare il fuzzy set delle variabili di ingresso e uscita, cercando di discretizzare maggiormente le variabili. Viene così proposto il controllo fuzzy fc2_4 (il codice numerico assegnato è stato scelto solo per tenere traccia dei cambiamenti tra una prova e l'altra), dopo una campagna di test. Del controllo vengono mostrati il fuzzy set e le regole.

fc2_4		de				
		NN	N	Z	P	PP
e	NN	NNN	NN	NN	N	N
	N	NN	NN	N	Z	P
	Z	NN	N	Z	P	PP
	P	N	Z	P	PP	PP
	PP	P	P	PP	PP	PPP

Tabella 6-9: fuzzy rule fc2_4

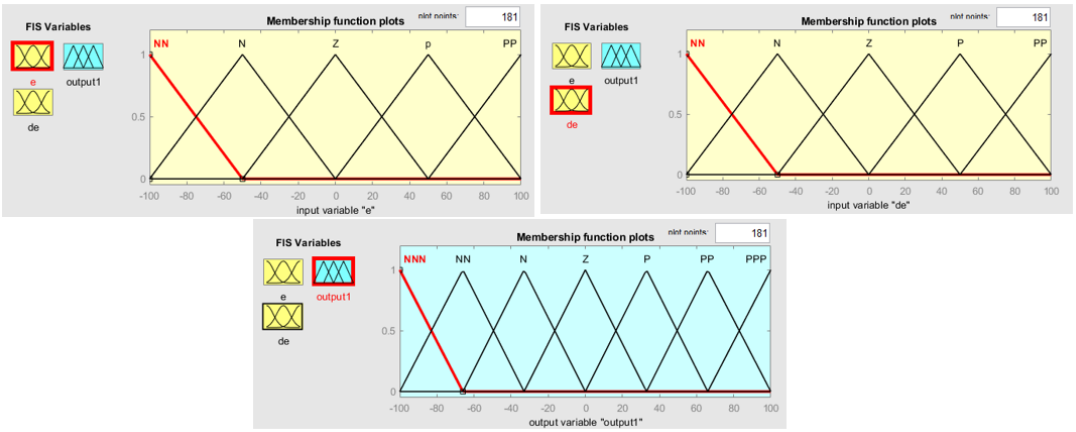


Tabella 6-8: fuzzy set fc2_4

Vengono di seguito mostrati dei confronti con il controllo fuzzy fc_1, con due set a gradino, di ampiezza 0.1m e 0.05 m, e con un set sinusoidale di ampiezza 0.05 e frequenza 0.05 Hz

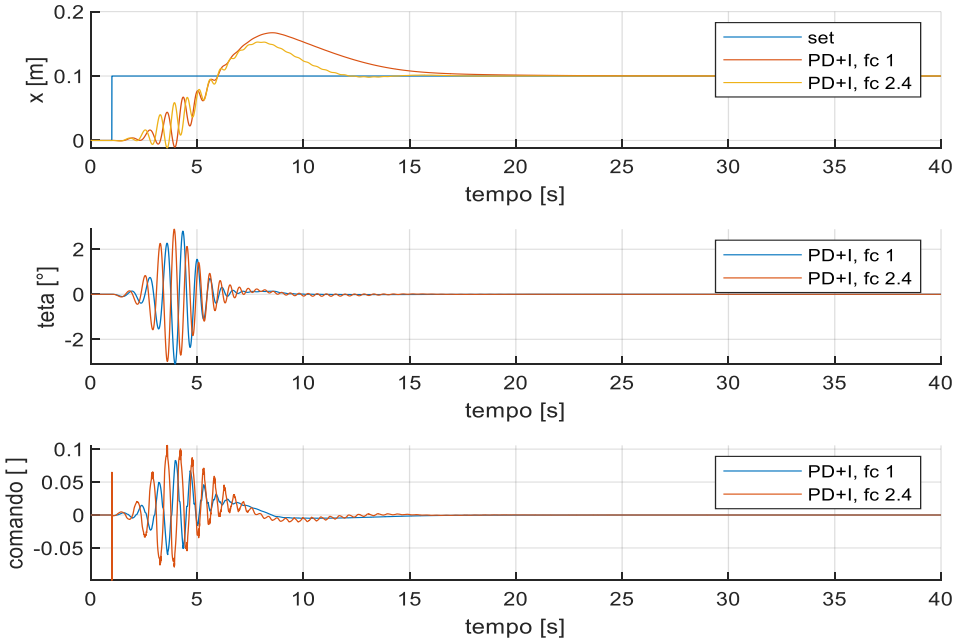


Figura 6-38: confronto gradino 0.1 m

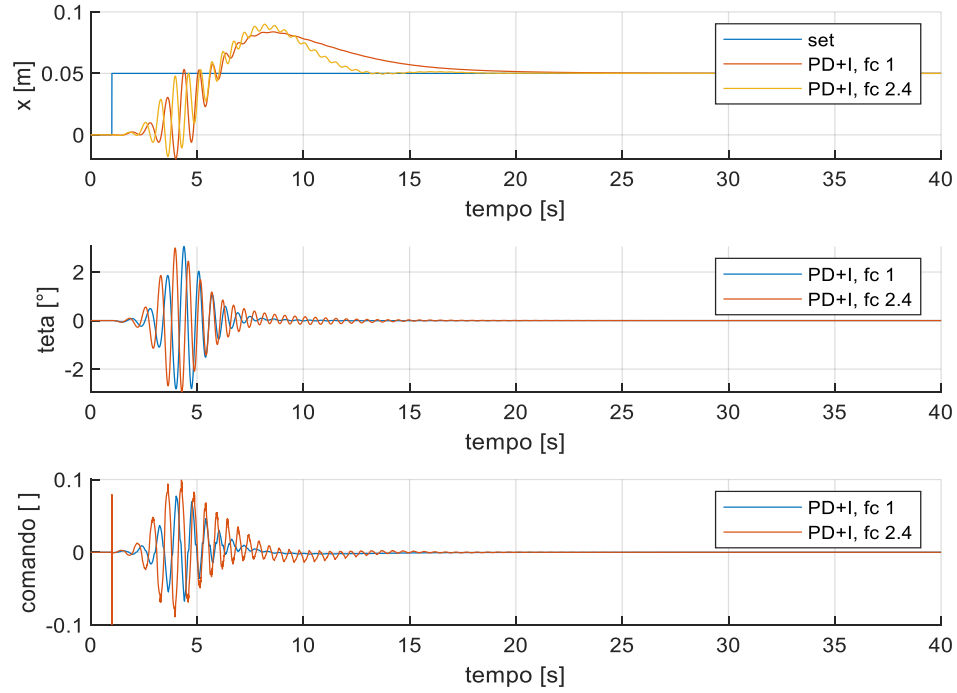


Figura 6-37; confronto gradino 0.05 m

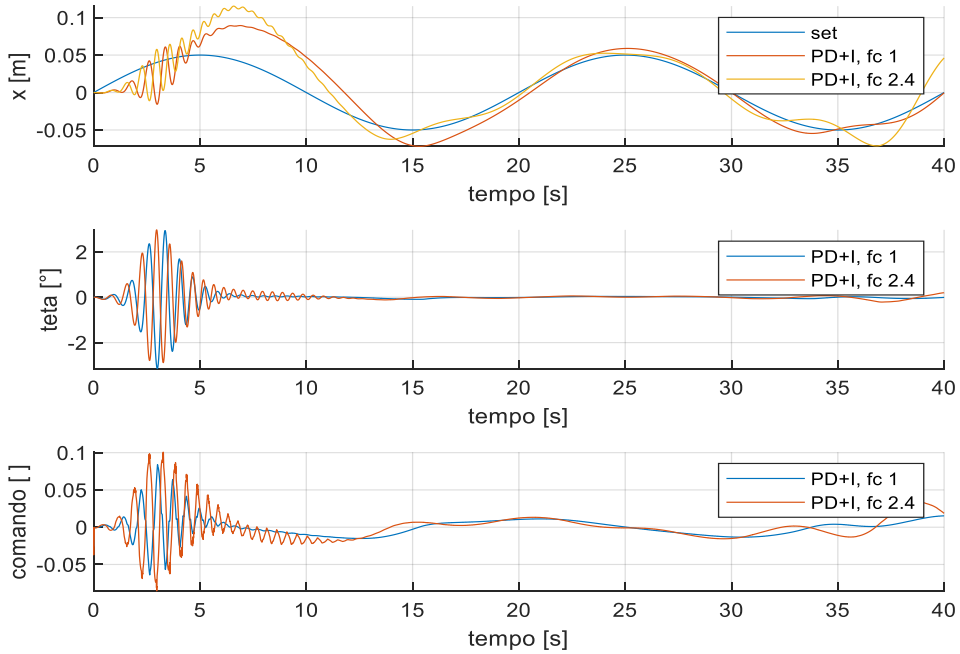


Figura 6-39: confronto sin

Dall'analisi dei seguenti grafici è possibile affermare che con un set a gradino il controllo fc2_4 risponde meglio, dato che a parità di overshoot raggiunge il set in un tempo inferiore, evidenziando però nel caso del gradino con ampiezza minore una maggiore instabilità del controllo, mostrando un segnale meno "liscio". Mentre un risultato diverso si ottiene con un set sinusoidale, dove invece il controllo fc2_4 diventa instabile nel secondo periodo, infatti pochi istanti dopo la simulazione (non riportati, visto che la precedente parte non risulterebbe più leggibile) si perde il controllo del pendolo.

Questo mostra come aumentare la sensibilità del fuzzy set non comporti necessariamente un miglioramento delle performance del controllo come ci si potrebbe aspettare intuitivamente.

7 Simulazione scheda Arduino

Per testare il funzionamento del codice sulla scheda Arduino si è scelto di usare un simulatore. Il software scelto tra i vari disponibili in commercio è Proteus Professional.

Proteus Virtual System Modeling (VSM) fonde la simulazione SPICE in modalità mista con la simulazione di microcontrollori. Consente la prototipazione rapida di progetti hardware e firmware, nel software stesso. Consente di progettare, testare ed eseguire il debug dei progetti incorporati nel simulatore di circuiti elettronici Proteus, prima di ordinare un prototipo fisico. Nel caso in esame è stato necessario aggiungere alle librerie di sistema dei file esterni al programma per simulare una scheda Arduino.

Tra le varie componenti da implementare nella libreria che sono riuscito a reperire, avevo pensato alla scheda mega 2560 in analisi, tuttavia presentava una serie di errori anche per le istruzioni più semplici; si è scelto quindi di passare alla scheda Arduino Uno, la più economica disponibile in commercio e con minori performance rispetto alla precedente, ma il cui modello implementato nel software non presentava errori, essendo la più diffusa.

Creando un nuovo progetto si ottiene la seguente schermata iniziale, dalla quale andiamo ad aggiunge la scheda Arduino uno premendo il tab Componente Mode e successivamente Pick Device, nella keyword inseriamo Arduino come parola chiava e selezioniamo il dispositivo Genuino Uno, che è risultato essere il Device più affidabile compatibile con l'aggiunta di librerie esterne.

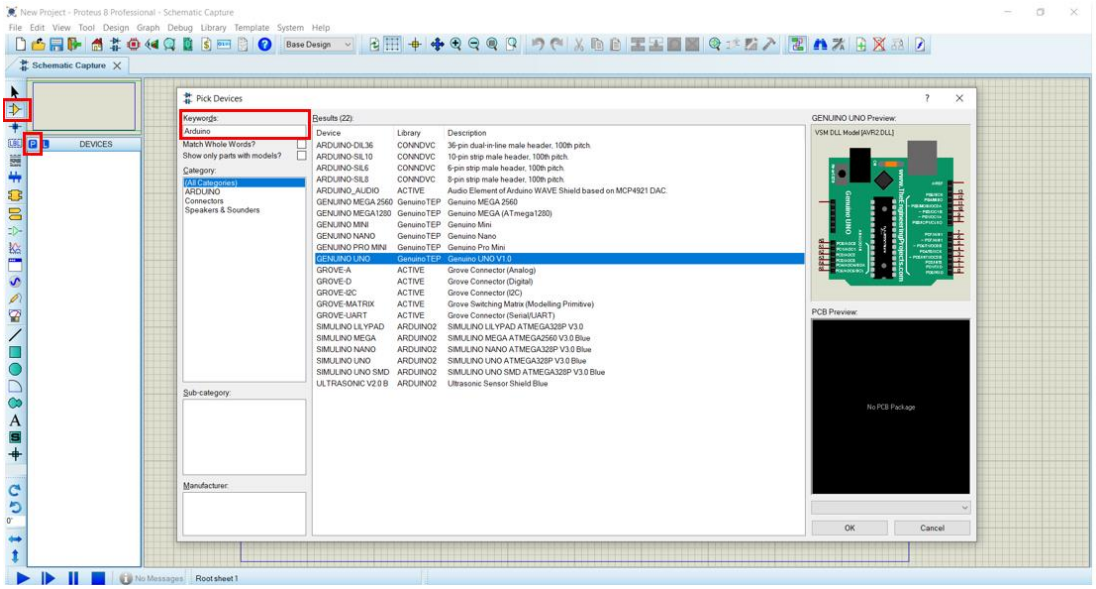


Figura 7-1: aggiungere Device

Infine, si posiziona il componente all'interno della finestra Schematic Capture. A titolo illustrativo viene evidenziato in figura l'uguaglianza tra la scheda Genuino Uno e quella reale, per simulare al meglio i circuiti coi vari pin. Ricordiamo che i pin con la tilde sono quelli abilitati all'output Analogico in PWM.

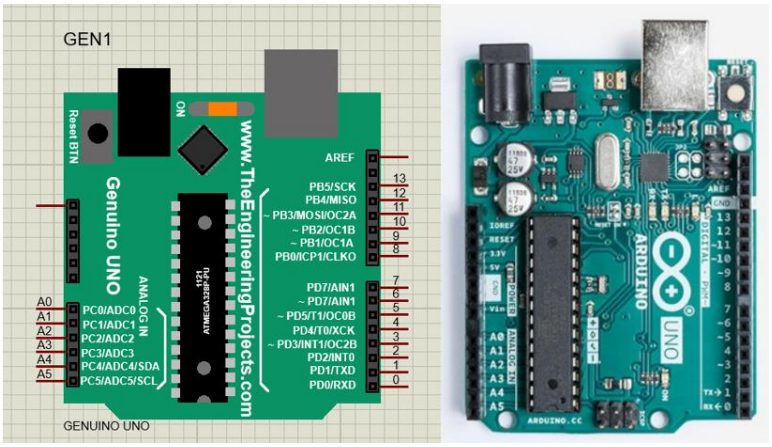


Figura 7-3:confronto schede Arduino Uno

Viene anche aggiunto un Virtual Terminal, per simulare il Monitor Seriale di Arduino. Il pin TXD della scheda va collegato col pin RXD del monitor, mentre il pin RXD della scheda va collegato col pin TXD del monitor per consentire la lettura dei dati.

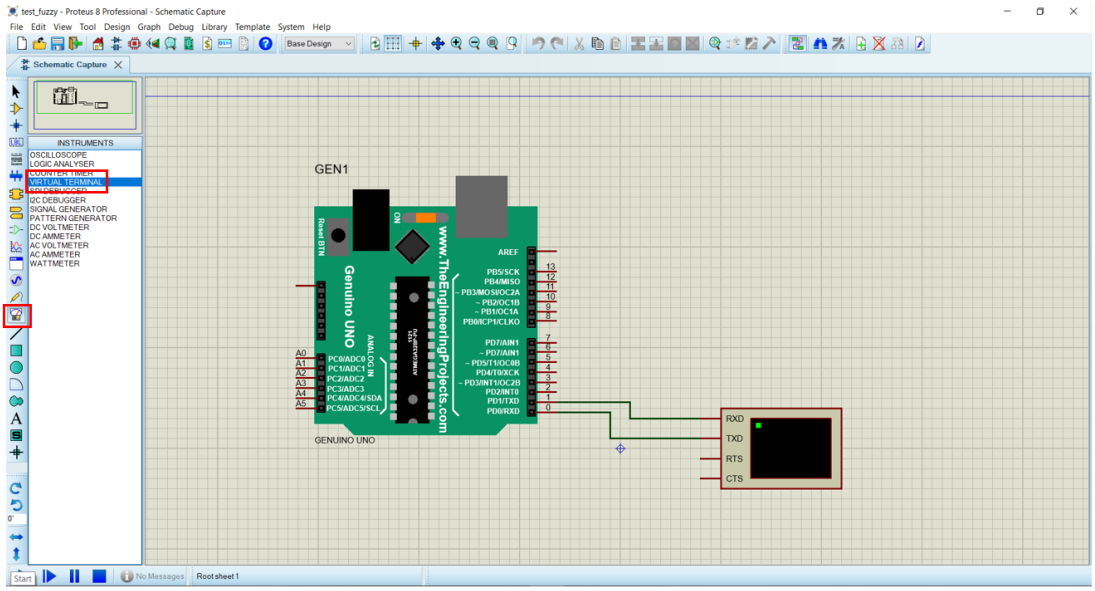


Figura 7-2: Virtual Terminal

Per caricare il file generato per Arduino sul simulatore è necessario seguire questa procedura:

1. Nel programma Arduino IDE in file impostazioni è necessario attivare la spunta “Compilazione” in “Mostra un output dettagliato durante” per visualizzare durante la verifica del file la directory dove viene salvato il file temporaneo (cancellato una volta chiusa l’applicazione Arduino IDE) con estensione “.hex”.

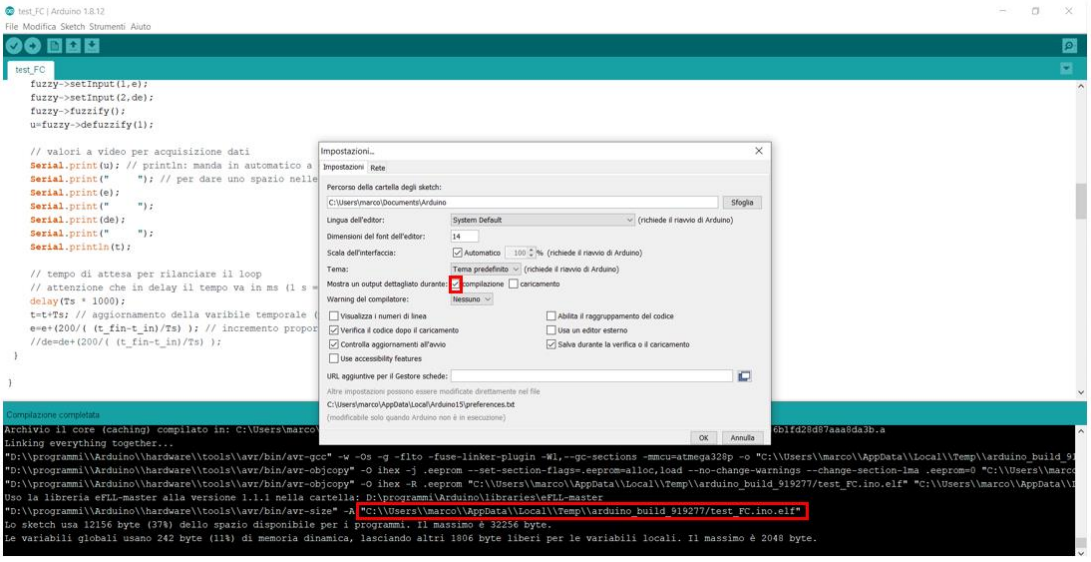


Figura 7-4: mostra Compilazione Arduino

2. in Proteus fare doppio clic col tasto sinistro del mouse sulla scheda Arduino, nella finestra Edit Component in Program File bisogna puntare alla directory del file creato dalla verifica di Arduino e selezionare il file: “NomeFile_ino.hex” (non quello preceduto dal campo with_bootloader).

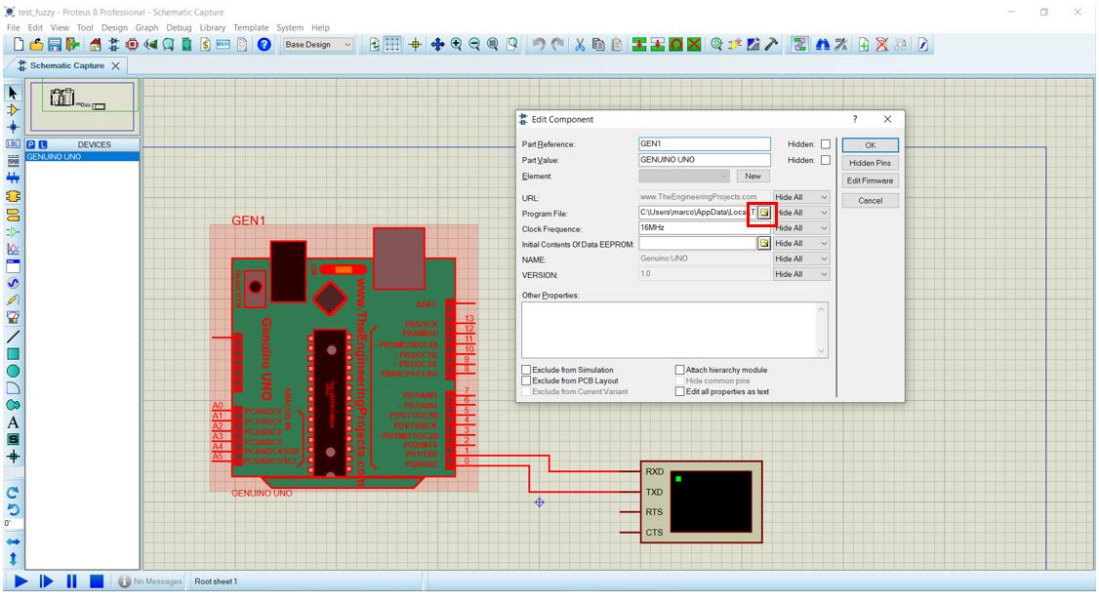


Figura 7-5: Edit Component scheda Arduino

3. Si fa iniziare la simulazione agendo sul tasto Run the Simulation. È probabile a questo punto che sia necessario far comparire il Virtual Terminal manualmente (se non compare in automatico) andando in Debug e cliccando sulla spunta del Virtual Terminal.

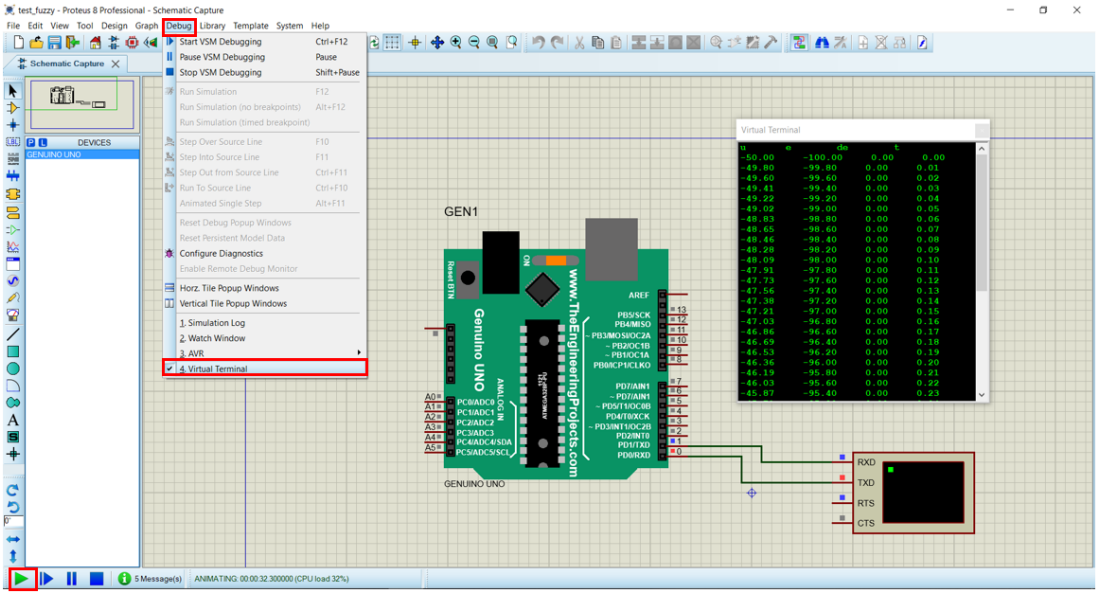


Figura 7-6: Virtual Terminal

4. è importante sapere che il Virtual Terminal presenta uno spazio limitato (come quello in Arduino IDE), quindi ogni tanto è necessario mettere in pausa l'esecuzione e pulire lo schermo: in caso di un'acquisizione dati a questo punto risulta consono copiare il contenuto prima dell'eliminazione.

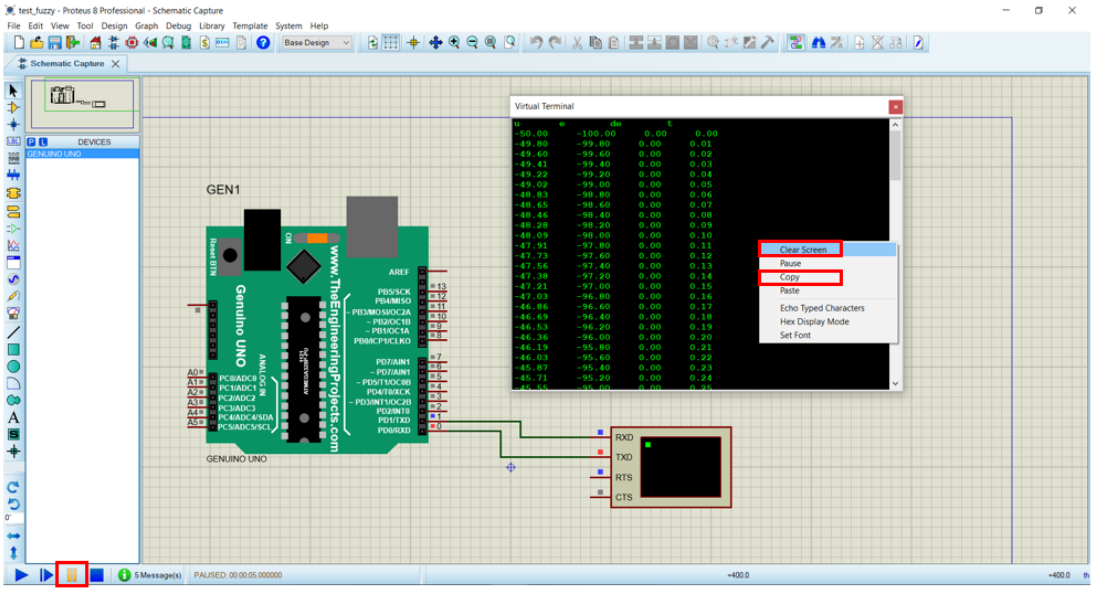


Figura 7-7: acquisizione dati

7.1 Nozioni base di Arduino

Prima di spiegare il funzionamento della libreria Fuzzi.h conviene riportare nelle nozioni base di Arduino per una migliore comprensione dei listati.

Nei listati Arduino possono essere identificate tre sezioni, che vengono evidenziate in rosso in figura:

- void setup(): le istruzioni presenti al suo interno vengono eseguite una sola volta, qualunque variabile definita al suo interno è una variabile locale. Tra le funzioni più importanti da inserire al suo interno c'è Serial.begin (9600), la quale implica che quando connettiamo la scheda alla porta usb lui dovrà trasmettere dati con una velocità pari a 9600 byte che sono sufficienti per trasmettere parole e numeri
- void loop(): le istruzioni presenti al suo interno vengono eseguite ciclicamente. Per dare stabilità al codice è frequente inserire al fondo un ritardo delay() espresso in millisecondi.
- Un'area esterna alle due precedenti: all'interno della quale vengono frequentemente dichiarate le variabili, che risulteranno essere quindi usabili sia nel void setup() sia nel void loop(), e richiamate le librerie tramite la stringa #include< >.

Figura 7-8: sezioni codici Arduino

I tab fondamentali per l'utilizzo di Arduino IDE sono:

- Verifica: per controllare se sono presenti errori di scrittura, o utilizzi non appropriati delle funzioni.
- Carica: per caricare il programma all'interno della scheda.

- Salva: per salvare il file, che deve risiedere in una cartella che presenta lo stesso nome, altrimenti il file non verrà riaperto in seguito
- Monitor seriale: disponibile solo quando connettiamo la scheda al pc.

Figura 7-9: comandi Arduino IDE

Due operazioni essenziali prima di caricare il programma nella scheda sono:

- verificare che sia selezionata la scheda corretta nel menu strumenti, in quanto sono presenti diversi controllori in commercio con diverse potenze di calcolo ad esempio.
- controllare che la porta selezionata sia quella alla quale stiamo comunicando con il controllore (es. COM4).

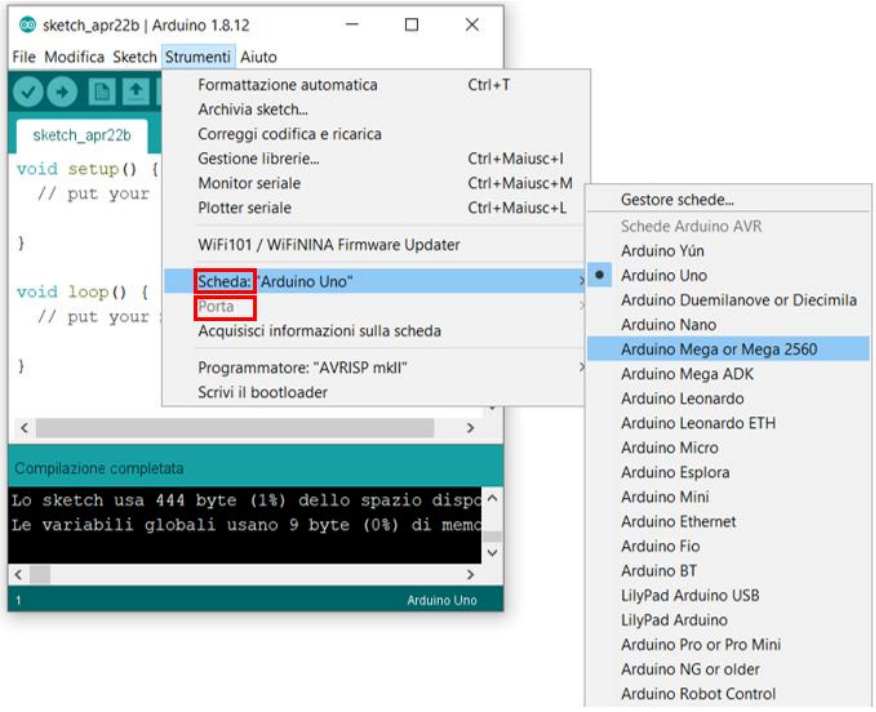


Figura 7-10: scheda e porta di comunicazione

7.2 Implementazione libreria Fuzzy su Arduino

Di seguito verranno commentate alcune righe di codice necessarie per una maggiore comprensione della libreria fuzzy di Arduino.

È innanzitutto necessario reperire la libreria fuzzy per Arduino, disponibile al link [9] , dove è possibile scaricare una cartella di nome “eFLL-master” di cui viene illustrato il contenuto.

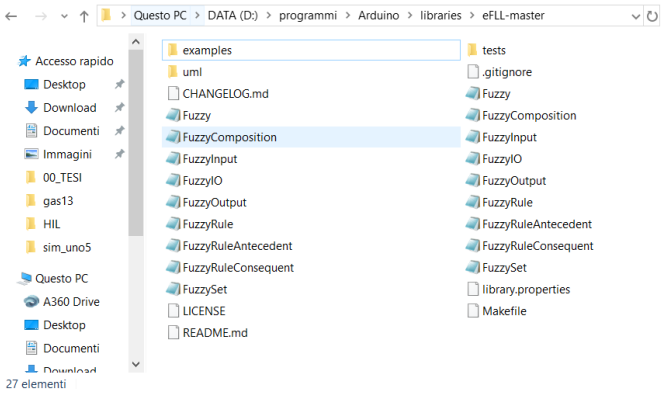


Figura 7-11: contenuto libreria Fuzzy.h

La seguente cartella dovrà poi essere posizionata nella directory delle librerie a cui fa riferimento Arduino, come evidenziato in figura.

Per caricare il contenuto della libreria in un programma si segue la seguente procedura: sketch, #include libreria, eFLL

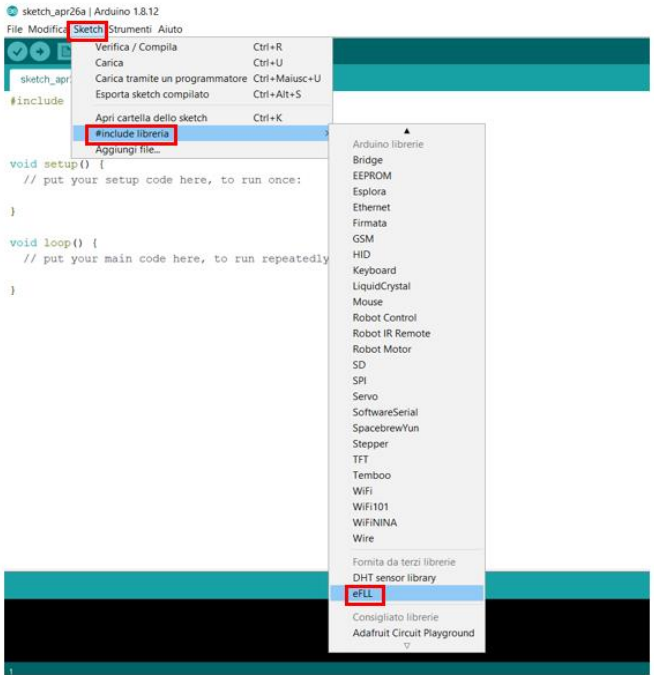


Figura 7-12: caricamento libreria

Così facendo comparirà la seguente riga di codice a conferma di avvenuto caricamento:
#include <Fuzzy.h>

Per l'apprendimento del funzionamento della libreria si è fatto riferimento a [10].
È necessario creare una funzione che contenga il controllore, in questo caso fuzzy:
Fuzzy *fc_1 = new Fuzzy();

All'interno del void setup viene inserita una funzione Set_LogicaFuzzy(); per richiamare il caricamento dei fuzzy set e delle regole.

Inizialmente vanno definite le funzioni di appartenenza dei fuzzy set di input e output, che nel caso in esame sono errore e derivata dell'errore per l'input e out per l'output. I quattro valori numerici proposti fanno riferimento ai vertici di una funzione trapezoidale. Ad esempio, la funzione FuzzySet *trapezio = new FuzzySet(-75, -25, 25, 75) , ciò non deve essere visto come una limitazione alla funzione trapezoidale, visto che con la medesima si possono anche rappresentare volendo funzioni triangolari FuzzySet *triangolo = new FuzzySet(-75, 0, 0, 75) o singleton FuzzySet *singleton = new FuzzySet(0,0,0,0)

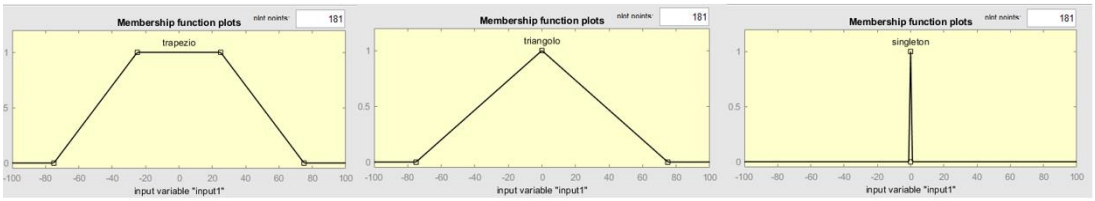


Figura 7-13: es. funzioni trapezoidali

```
// set FuzzyInput
FuzzySet *N_e = new FuzzySet(-100, -100, -100, 0);
FuzzySet *Z_e = new FuzzySet(-100, 0, 0, 100);
FuzzySet *P_e = new FuzzySet(0, 100, 100, 100);

FuzzySet *N_de = new FuzzySet(-100, -100, -100, 0);
FuzzySet *Z_de = new FuzzySet(-100, 0, 0, 100);
FuzzySet *P_de = new FuzzySet(0, 100, 100, 100);

// set FuzzyOutput
FuzzySet *NN_out = new FuzzySet(-100, -100, -100, -50);
FuzzySet *N_out = new FuzzySet(-100, -50, -50, 0);
FuzzySet *Z_out = new FuzzySet(-50, 0, 0, 50);
FuzzySet *P_out = new FuzzySet(0, 50, 50, 100);
FuzzySet *PP_out = new FuzzySet(50, 100, 100, 100);
```

Di seguito vanno invece aggregate le varie funzione per costruire i fuzzy set. Tramite le funzioni addFuzzySet si aggregano le varie funzioni e va sottolineato che una funzione può essere assegnata una sola volta. Con la funzione addFuzzyInput si costruisce un fuzzy set di input, mentre con la funzione addFuzzyOutput si costruisce un fuzzy set di output.

```
// FuzzySetInput: 1) errore 2) derivata dell'errore
FuzzyInput *err = new FuzzyInput(1);
err->addFuzzySet(N_e);
err->addFuzzySet(Z_e);
err->addFuzzySet(P_e);
fc_1->addFuzzyInput(err);

FuzzyInput *d_err = new FuzzyInput(2);
d_err->addFuzzySet(N_de);
d_err->addFuzzySet(Z_de);
d_err->addFuzzySet(P_de);
fc_1->addFuzzyInput(d_err);

// FuzzySetOutput: 1) out_fc
FuzzyOutput *out_fc = new FuzzyOutput(1);
out_fc->addFuzzySet(NN_out);
out_fc->addFuzzySet(N_out);
out_fc->addFuzzySet(Z_out);
out_fc->addFuzzySet(P_out);
out_fc->addFuzzySet(PP_out);
fc_1->addFuzzyOutput(out_fc);
```

Infine, si procede con la stesura delle regole. Con la funzione FuzzyRuleAntecedent() assegno il nome all'identità a sinistra, ovvero al "se", tramite joinWithAND() e joinWithOR() è possibile articolare un costrutto logico da caricare nell'identità "se". Invece usando la funzione FuzzyRuleConsequent() assegno il nome all'identità a destra, ovvero ad "allora", con addOutput() assegno la funzione corrispettiva. Per concludere con la funzione FuzzyRule() assegno il nome alla regola assegnandogli le condizione logiche corrispettive, e sfruttando addFuzzyRule() carico la regola nel controllore fuzzy, nel caso in esame fc_1. Va sottolineato che i nomi usati non possono essere ripetuti, perciò si usano numeri progressivi per le condizioni logiche e le regole.

```
// se err=N e d_err=N ==> out=NN
FuzzyRuleAntecedent *if_err_N_AND_d_err_N = new FuzzyRuleAntecedent();
if_err_N_AND_d_err_N->joinWithAND(N_e, N_de);
FuzzyRuleConsequent *then_out_NN1 = new FuzzyRuleConsequent();
then_out_NN1->addOutput(NN_out);
FuzzyRule *fc_1Rule1 = new FuzzyRule(1, if_err_N_AND_d_err_N, then_out_NN1);
fc_1->addFuzzyRule(fc_1Rule1);

// se err=N e d_err=Z ==> out=N
FuzzyRuleAntecedent *if_err_N_AND_d_err_Z = new FuzzyRuleAntecedent();
if_err_N_AND_d_err_Z->joinWithAND(N_e, Z_de);
FuzzyRuleConsequent *then_out_N1 = new FuzzyRuleConsequent();
then_out_N1->addOutput(N_out);
FuzzyRule *fc_1Rule2 = new FuzzyRule(2, if_err_N_AND_d_err_Z, then_out_N1);
fc_1->addFuzzyRule(fc_1Rule2);

// se err=N e d_err=P ==> out=N
```

```
FuzzyRuleAntecedent *if_err_N_AND_d_err_P = new FuzzyRuleAntecedent();
if_err_N_AND_d_err_P->joinWithAND(N_e, P_de);
FuzzyRuleConsequent *then_out_N2 = new FuzzyRuleConsequent();
then_out_N2->addOutput(N_out);
FuzzyRule *fc_1Rule3 = new FuzzyRule(3, if_err_N_AND_d_err_P, then_out_N2);
fc_1->addFuzzyRule(fc_1Rule3);

// se err=Z e d_err=N ==> out=N
FuzzyRuleAntecedent *if_err_Z_AND_d_err_N = new FuzzyRuleAntecedent();
if_err_Z_AND_d_err_N->joinWithAND(Z_e, N_de);
FuzzyRuleConsequent *then_out_N3 = new FuzzyRuleConsequent();
then_out_N3->addOutput(N_out);
FuzzyRule *fc_1Rule4 = new FuzzyRule(4, if_err_Z_AND_d_err_N, then_out_N3);
fc_1->addFuzzyRule(fc_1Rule4);

// se err=Z e d_err=Z ==> out=Z
FuzzyRuleAntecedent *if_err_Z_AND_d_err_Z = new FuzzyRuleAntecedent();
if_err_Z_AND_d_err_Z->joinWithAND(Z_e, Z_de);
FuzzyRuleConsequent *then_out_Z1 = new FuzzyRuleConsequent();
then_out_Z1->addOutput(Z_out);
FuzzyRule *fc_1Rule5 = new FuzzyRule(5, if_err_Z_AND_d_err_Z, then_out_Z1);
fc_1->addFuzzyRule(fc_1Rule5);

// se err=Z e d_err=P ==> out=P
FuzzyRuleAntecedent *if_err_Z_AND_d_err_P = new FuzzyRuleAntecedent();
if_err_Z_AND_d_err_P->joinWithAND(Z_e, P_de);
FuzzyRuleConsequent *then_out_P1 = new FuzzyRuleConsequent();
then_out_P1->addOutput(P_out);
FuzzyRule *fc_1Rule6 = new FuzzyRule(6, if_err_Z_AND_d_err_P, then_out_P1);
fc_1->addFuzzyRule(fc_1Rule6);

// se err=P e d_err=N ==> out=P
FuzzyRuleAntecedent *if_err_P_AND_d_err_N = new FuzzyRuleAntecedent();
if_err_P_AND_d_err_N->joinWithAND(P_e, N_de);
FuzzyRuleConsequent *then_out_P2 = new FuzzyRuleConsequent();
then_out_P2->addOutput(P_out);
FuzzyRule *fc_1Rule7 = new FuzzyRule(7, if_err_P_AND_d_err_N, then_out_P2);
fc_1->addFuzzyRule(fc_1Rule7);

// se err=P e d_err=Z ==> out=P
FuzzyRuleAntecedent *if_err_P_AND_d_err_Z = new FuzzyRuleAntecedent();
if_err_P_AND_d_err_Z->joinWithAND(P_e, Z_de);
FuzzyRuleConsequent *then_out_P3 = new FuzzyRuleConsequent();
then_out_P3->addOutput(P_out);
FuzzyRule *fc_1Rule8 = new FuzzyRule(8, if_err_P_AND_d_err_Z, then_out_P3);
fc_1->addFuzzyRule(fc_1Rule8);

// se err=P e d_err=Z ==> out=PP
FuzzyRuleAntecedent *if_err_P_AND_d_err_P = new FuzzyRuleAntecedent();
```

```
if_err_P_AND_d_err_P->joinWithAND(P_e, P_de);
FuzzyRuleConsequent *then_out_PP1 = new FuzzyRuleConsequent();
then_out_PP1->addOutput(PP_out);
FuzzyRule *fc_1Rule9 = new FuzzyRule(9, if_err_P_AND_d_err_P, then_out_PP1);
fc_1->addFuzzyRule(fc_1Rule9);
```

Per utilizzare nel void loop() il controllore fuzzy, si usa la funzione setInput() per caricare le variabili di input, fuzzify() per attivare il processo di fuzzificazione e defuzzify() per quello di defuzzificazione.

```
fc_1->setInput(1,e);
fc_1->setInput(2,de);
fc_1->fuzzify();
u=fc_1->defuzzify(1);
```

7.3 Verifica libreria fuzzy

Per verificare il corretto funzionamento della libreria Fuzzy su Arduino si è scelto di confrontare l'andamento della prova di linearità con i risultati ottenuti da Simulink. Vengono quindi simulate le tre prove:

- Variando linearmente l'errore e lasciando a zero la sua derivata
- Variando linearmente la derivata dell'errore e lasciando a zero l'errore
- Variando linearmente entrambi

Nell'intestazione del programma vengono quindi riportate le seguenti istruzioni che verranno eseguite una alla volta, commentando le restanti nelle successive simulazioni.

```
// variabili di errore: definite a seconda del caso analizzato
// caso1: var_e
float e = -100;
float de = 0;
// caso2: var_de
float e = 0;
float de = -100;
// caso3: var_e_de
float e = -100;
float de = -100;
```

Mentre nella void loop() viene implementato un codice per leggere i valori delle variabili e riportarli a video, in modo tale da consentirne un'acquisizione su Proteus (cosa impossibile senza un simulatore, in quanto Arduino IDE per lanciare un programma necessita di una scheda collegata, sul quale caricarlo), per accedere al Monitor Seriale su Arduino si possono usare due funzioni: Serial.print() che stampa semplicemente a video e Serial.println() che oltre a stampare a video manda anche a capo. Mettendo tra le parentesi una variabile questa verrà visualizzata a video (convertendola ovviamente in formato char), ma è anche possibile riportare delle stringhe di testo usando i doppi apici.
Sempre all'interno del void loop() è necessario aggiornare la variabile "e" o "de", oppure entrambe, tramite un incremento proporzionale al tempo di simulazione.

```
Serial.print(u);
Serial.print("    ");
Serial.print(e);
Serial.print("    ");
Serial.print(de);
Serial.print("    ");
Serial.println(t);
t=t+Ts;
e=e+(200/( (t_fin-t_in)/Ts) );
de=de+(200/( (t_fin-t_in)/Ts) );
```

Per una migliore lettura dei risultati da monitor seriale, anche per un futuro utente, vengono inserite nel void setup() le seguenti righe di codice per generare una prima riga che funga come intestazione per le varie colonne di dati, consentendo quindi una migliore comprensione in fase di esecuzione.

```
Serial.print("u      ");
Serial.print("e      ");
Serial.print("de      ");
Serial.println("t");
```

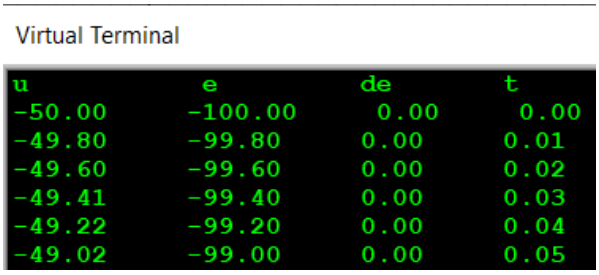


Figura 7-14: es. intestazione dati

Una volta terminata l'esecuzione del programma il contenuto del Virtual Terminal viene copiato in un file di testo (.txt) per essere successivamente elaborato in Matlab.

Nella prima simulazione è stato variato linearmente l'errore, lasciando nulla la sua derivata.

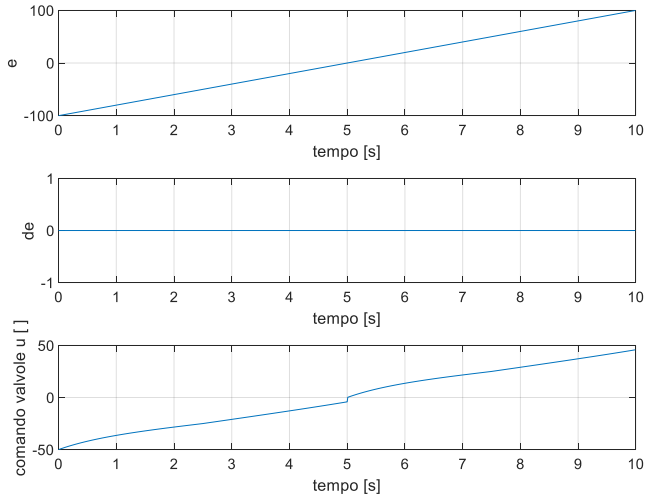


Figura 7-15: sim. Variando: e

Nella seconda simulazione è stata variata linearmente la derivata dell'errore "de", lasciando nullo l'errore.

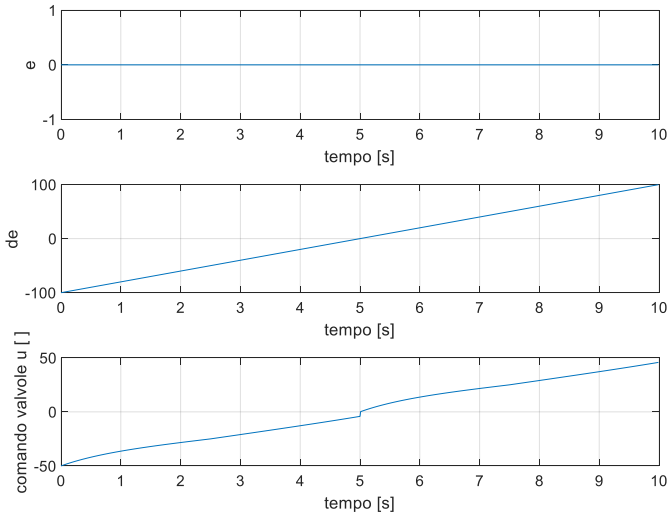


Figura 7-16: sim. variando: de

Nella terza simulazione sono stati fatti variare linearmente entrambi.

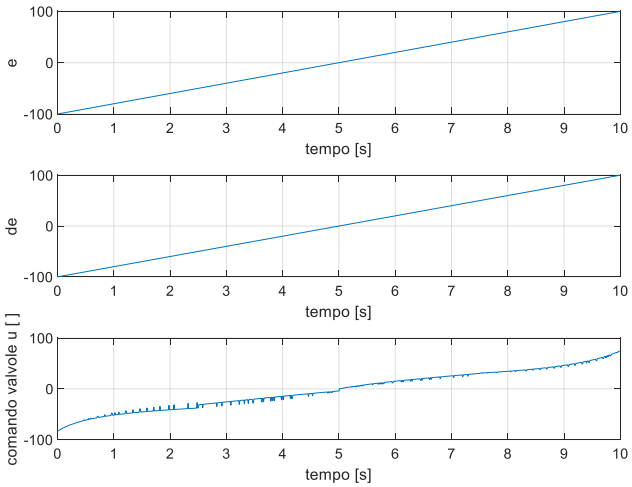


Figura 7-18: sim. variando: e, de

Di seguito viene riportato in figura il confronto con il fuzzy tool di Matlab. Nei tre subplot vengono rappresentati i segnali di uscita u diretti alle valvole nei tre casi prima analizzati.

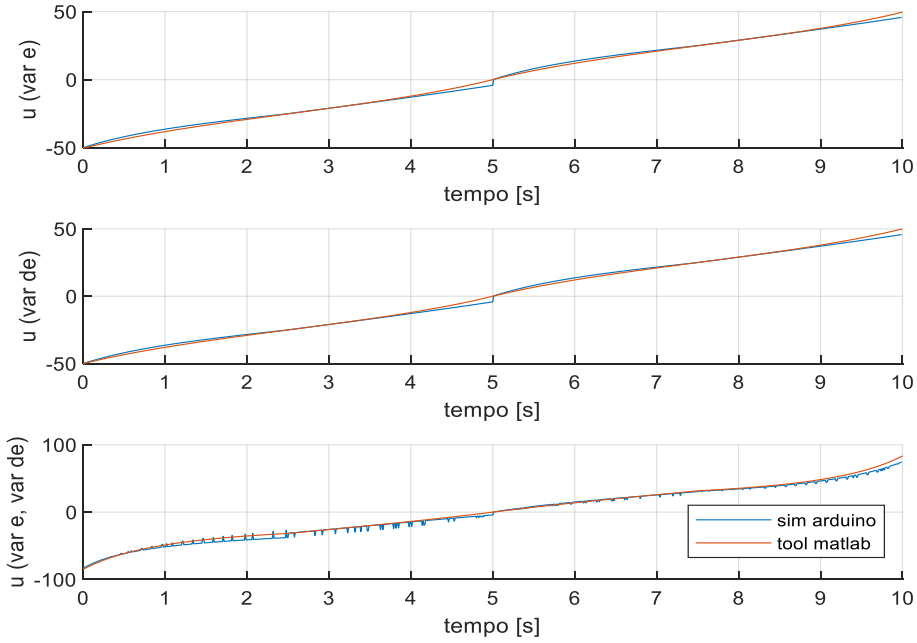


Figura 7-17: confronto libreria Arduino con tool Matlab

Dall'analisi si può evincere che il comportamento della libreria fuzzy sul simulatore di Arduino su Proteus funziona correttamente, in quanto rispecchia il comportamento del fuzzy tool di Matlab.

7.4 Simulazione della libreria agendo manualmente sugli input

Per un test più accurato sul funzionamento della libreria viene costruito un altro circuito, per permette all'utilizzatore di testare in tempo reale l'uscita corrispondente agli input impartiti e acquistare familiarità con le risposte del controllo fuzzy.

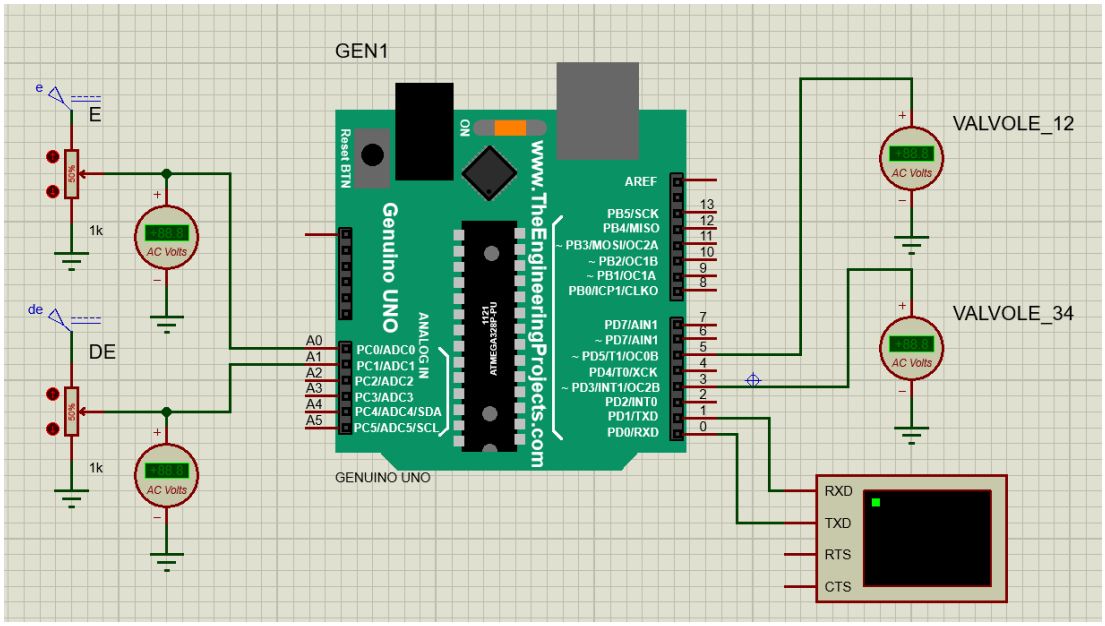


Figura 7-19: simulazione input e output

Per realizzare il circuito è stato necessario aggiungere ulteriori componenti rispetto al caso precedente:

- Due generatori di corrente continua (DC Generators) per simulare l'errore e la sua derivata. Il valore di tensione è stato impostato a 5 V che è il massimo valore leggibile dalla scheda Arduino. Tra le impostazioni è possibile anche definire un nome rappresentativo.

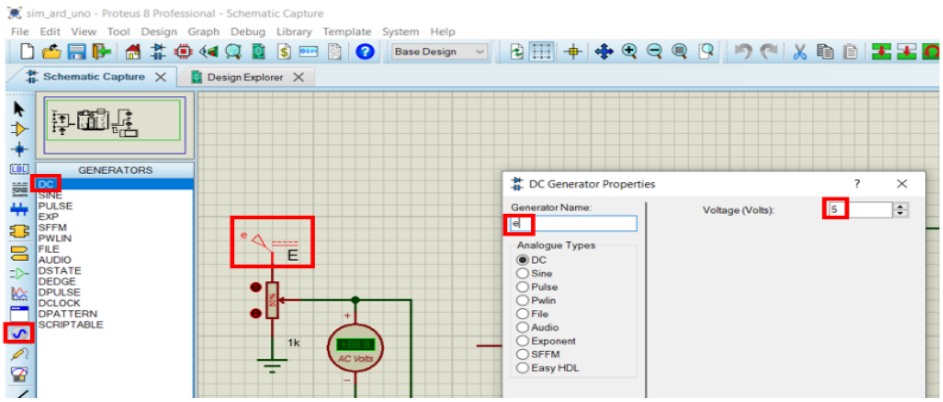


Figura 7-20:generatore Dc

- Quattro voltmetri, due per la lettura dei valori di tensione in ingresso simulando valori di errore e della sua derivata, che potrebbero anche rappresentare segnali provenienti dal LVDT o dal traduttore angolare, mentre gli altri due per la lettura dei valori in uscita destinati al comando delle valvole. Come per il generatore di tensione è possibile assegnare un nome simbolico

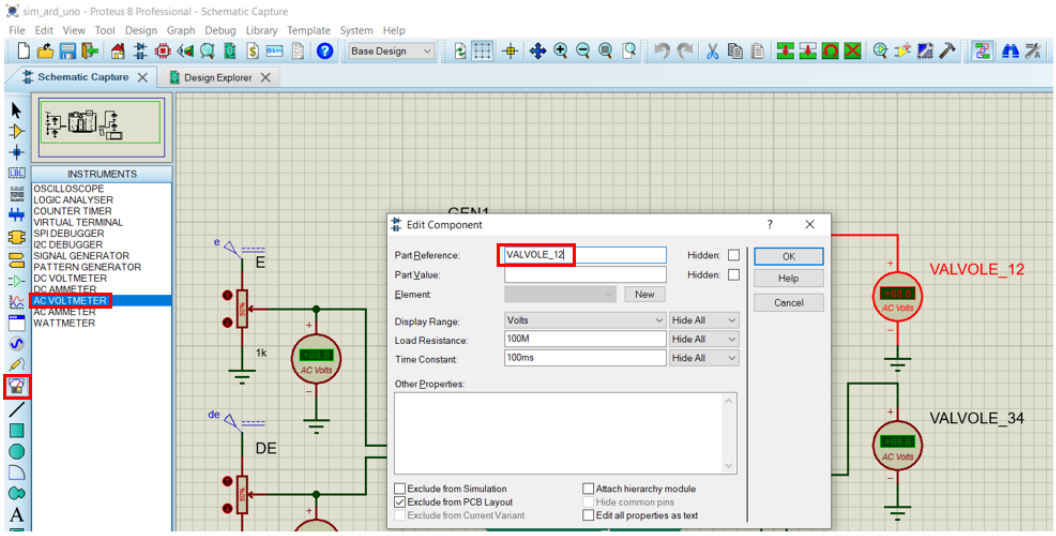


Figura 7-21: voltmetro

- Due potenziometri in modo tale da consentire di variare in tempo reale i valori di tensione ingresso nella scheda dell'errore e della sua derivata. Durante la simulazione è possibile agire sul potenziometro attraverso le frecce a lato, oppure trascinando il corpo centrale. Assume il valore del 100% se viene lasciata passare tutta la tensione, mentre 0% se non viene fatta passare la tensione. In figura sono rappresentate alcune possibili posizioni assumibili dal potenziometro.

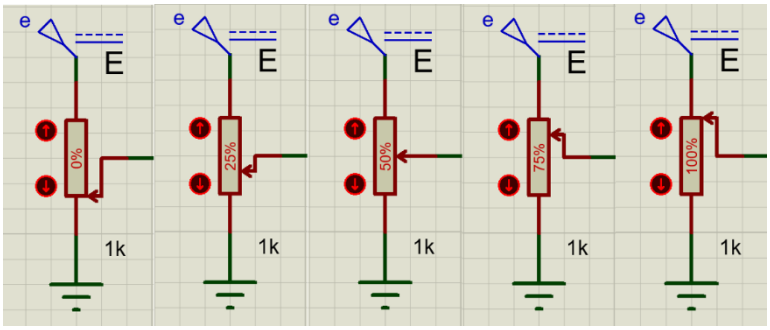


Figura 7-22: posizionamenti del potenziometro

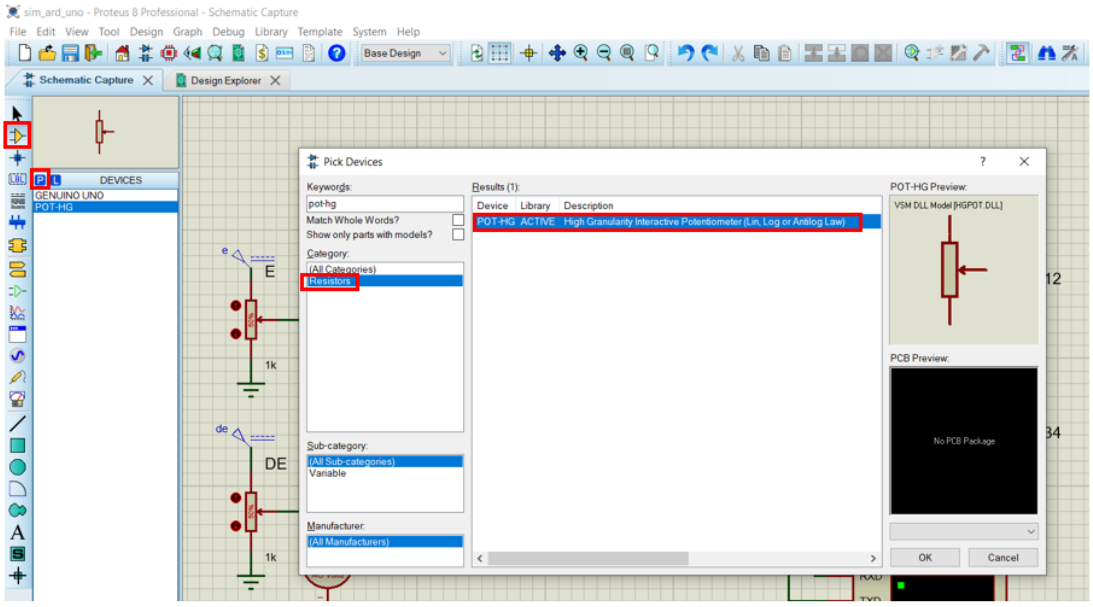


Figura 7-23: potenziometro

È stato necessario aggiornare il codice per consentire la lettura dei valori dai pin di analogici input e la scrittura sugli output analogici (PWM).
Prima del void setup() sono state dichiarate le variabili che fanno riferimento ai vari pin:

```
const int e_pin = A0;
const int de_pin = A1;
const int valv_12_pin = 3; // comanda valvola 1 e 2
const int valv_34_pin = 5; // comanda valvola 3 e 4
```

Tramite la funzione pinMode() è possibile dichiarare un pin come input o come output. Per dare una maggiore stabilità al codice si è scelto di dare come comando iniziale alle valvole un valore nullo. Nel void setup() vengono inseriti i seguenti comandi:

```
// PIN lettura input
pinMode(e_pin, INPUT);
pinMode(de_pin, INPUT);
// PIN analogici (PWM) sui quali scriviamo i valori (OUTPUT)
pinMode(valv_12_pin, OUTPUT);
pinMode(valv_34_pin, OUTPUT);
// come valore iniziale impostiamo lo zero (nessun comando)
analogWrite(valv_12_pin, 0);
analogWrite(valv_34_pin, 0);
```

Tramite la funzione analogRead() è possibile leggere un valore analogico in ingresso da un pin, mentre il comando (float) serve per evidenziare alla scheda Arduino di trattare il valore letto come un decimale. analogRead fornisce valori tra 0 e 1023, dato che vengono usati 10 bit, di conseguenza $2^{10} = 2014$ possibili valori. Con la funzione analogWrite() è possibile scrivere sui pin digitali in PWM simulando così un segnale analogico. Questa operazione può essere effettuata su tutti quei pin che presentano la tilde, che indica l'abilitazione all'output in PWM, come quelli evidenziati in figura.

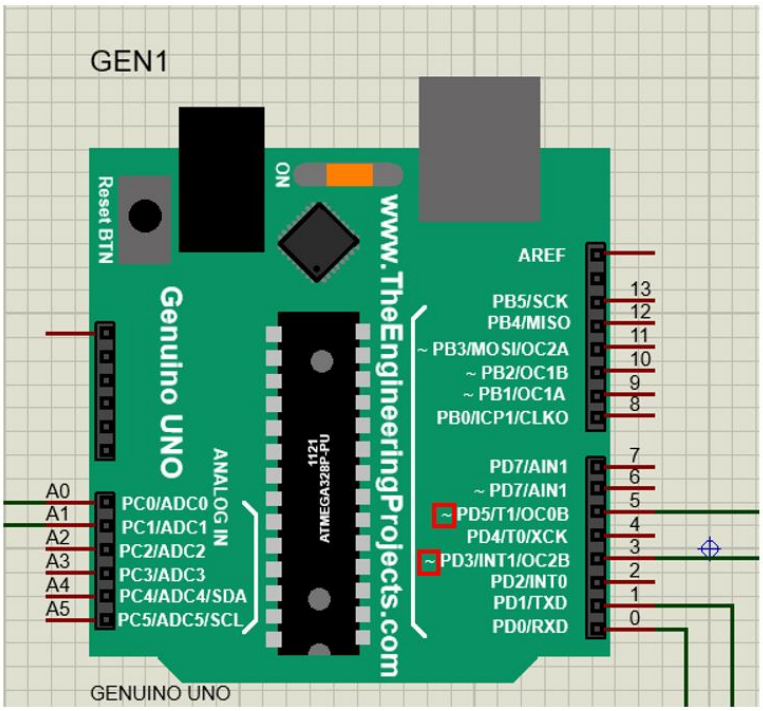


Figura 7-24: pin output digitali abilitati al PWM

A differenza della funzione analogRead la analogWrite lavora su 8 bit, di conseguenza $2^8 = 256$ possibili valori, ovvero da 0 a 255. Per questo la variabile u_scal, che non è altro che la variabile u espressa tra -1 e 1 anziché tra -100 e 100, viene quindi moltiplicata per il valore di 255 e mandata come comando al pin di uscita corrispondente.
Prima di mandare il segnale in uscita è importante capire a quali valvole mandare i segnali, per questo viene implementato un if() che verifica il segno della funzione u_scal.
Nel void loop() vengono quindi inserite le seguenti istruzioni:

```
e= ( (float)analogRead(e_pin) *200 / 1023 - 100 );
de=( (float)analogRead(de_pin) * 200 / 1023.0 - 100 );

u_scal=(float)u/100;
// invio comandi alle valvole
if (u_scal == 0) {
  analogWrite(valv_12_pin, 0);
  analogWrite(valv_34_pin, 0);
}
if (u_scal > 0) {
```

```
  analogWrite(valv_12_pin, u_scal * 255);
  analogWrite(valv_34_pin, 0);
 }
 if (u_scal < 0) {
  analogWrite(valv_12_pin, 0);
  analogWrite(valv_34_pin, abs(u_scal) * 255);
 }
```

Per evidenziare l'efficacia del codice vengono riportati alcuni fotogrammi di una simulazione.

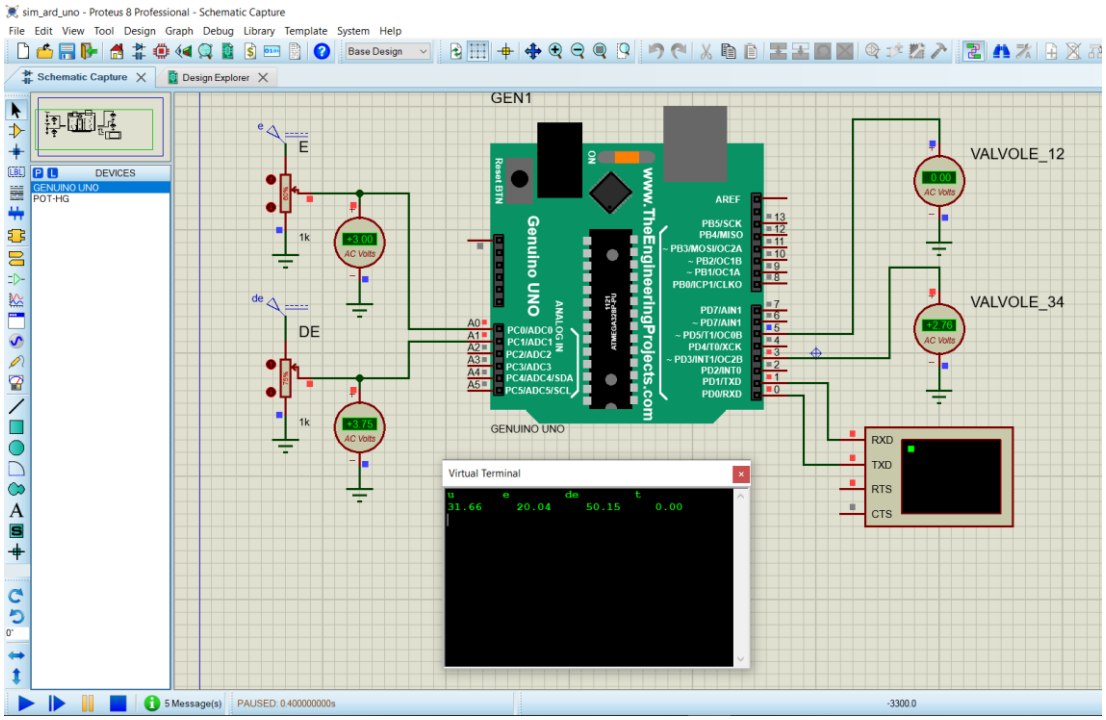


Figura 7-25: time1

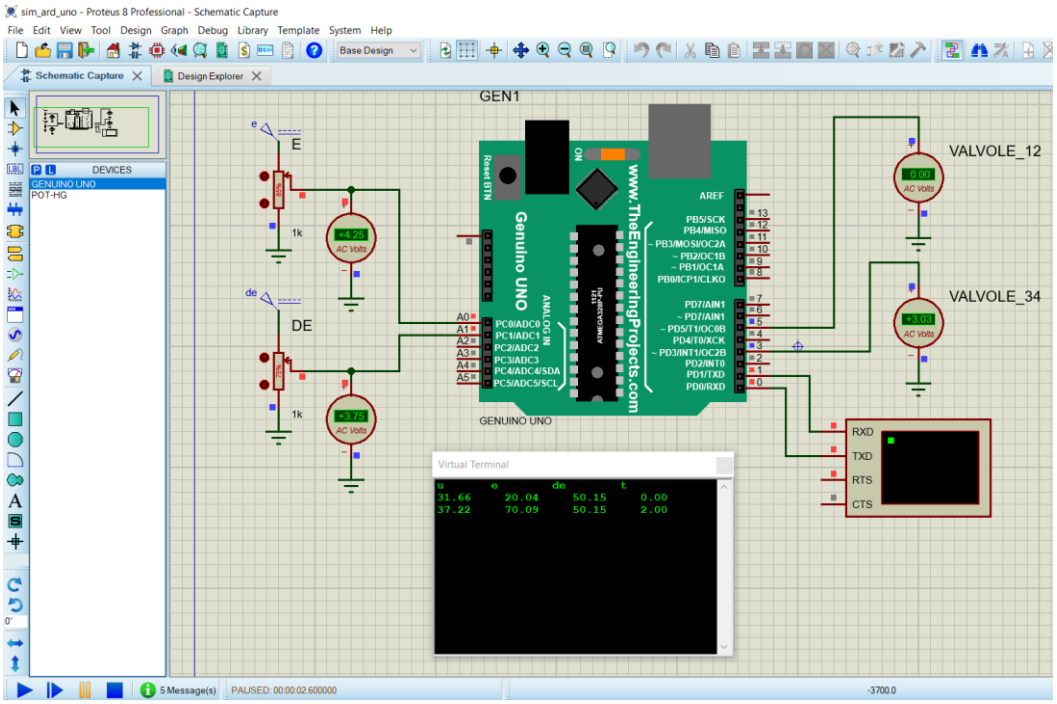


Figura 7-27: time2

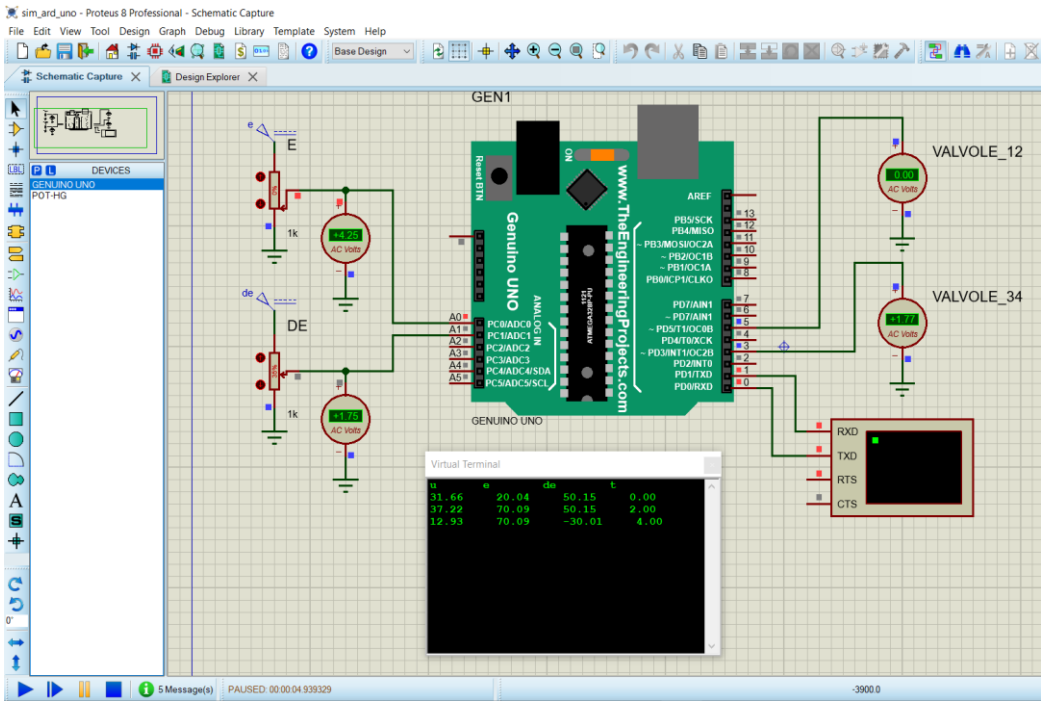


Figura 7-26: time3

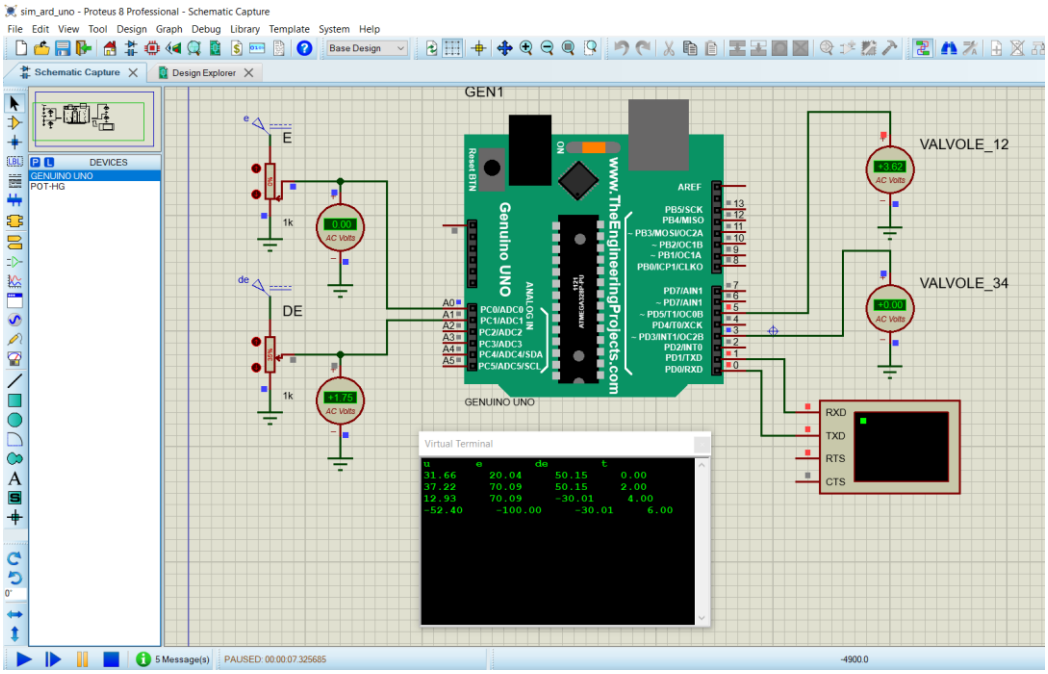


Figura 7-29: time3

Come ulteriore verifica del corretto funzionamento si può osservare come al time3 con un errore di 70.09 e la derivata di -30.01 otteniamo un'uscita di 12.93: il risultato è molto simile a quello ottenuto con il fuzzy tool che è invece di 13.2. Questa differenza può essere dovuta ad errori di approssimazione all'interno della scheda Arduino Uno o all'implementazione della libreria "Genuino"(delle schede Arduino) in Proteus.

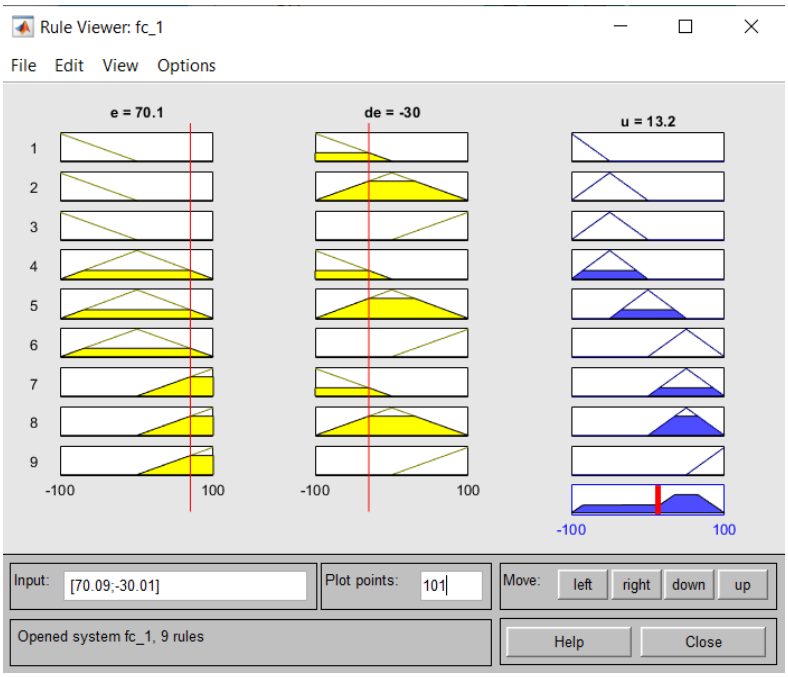


Figura 7-28: fuzzy tool e=70.09 de=-30.01

7.5 Programmazione scheda Arduino

Una volta verificato il corretto funzionamento delle librerie implementate si è passati alla stesura dei codici in Arduino IDE per i tre controllori precedentemente sviluppati.

Il codice completo di tutti i controllori viene riportato in appendice e nei successivi paragrafi vengono evidenziati i passaggi più significativi.

La lettura dei sensori viene effettuata mediante i pin di input analogici con il comando analogRead(pin). Le due equazioni riportate servono per una conversione del valore letto.

```
// lettura input da sensori e conversione
 x = -1.0 * ( (double)analogRead(x_pin) * corsa / 1023.0 - corsa / 2 );
 th = -1.0 * ( (double)analogRead(th_pin) * 1.57 / 1023.0 - 0.785 ) - 0.291;
```

Equazione lettura x:

- (double): serve per evidenziare il formato derivante dalle operazioni. Questo comando se non implementato può portare la scheda Arduino a volte a trattare numeri decimali come se fossero interi, per questo viene riportato nelle operazioni di moltiplicazione che prevedono un'assegnazione ad una variabile successiva
- *corsa/1023: per ottenere un valore tra 0 e corsa.
- -corsa/2: perché il segnale di x che vogliamo trattare deve variare tra -corsa/2 e +corsa/2.
- -1.0*: per come è montato il sensore.

Equazione lettura teta:

- *1.57/1023: per ottenere un valore tra 0° e 90° (1.57 rad= 90°).
- -0.785: perché il segnale di teta che vogliamo trattare deve variare tra -45° e +45°.
- -0.291: questo valore deriva da una lettura effettuata dal sensore quando l'asta è verticale, ovvero a 0°.

7.5.1 Controllore PD+I

Viene riportata un'immagine del controllo fuzzy sull'errore di posizione, per una migliore comprensione del listato.

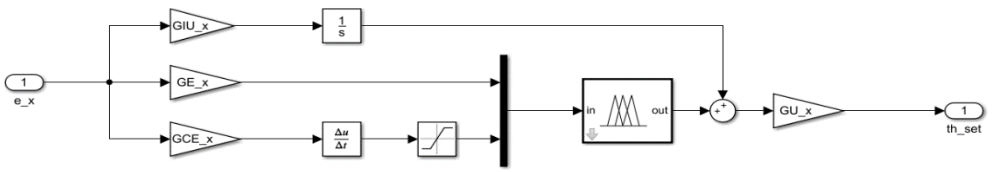


Figura 7-30:controllo PD+I sull'errore posizione

Per trattare derivate ed integrali è necessario tenere memoria delle variabili di errore del ciclo precedente.

```
// salvataggio valori di errore del ciclo precedente
// operazione necessaria per eseguire derivate e integrali
e_x_prec = e_x;
e_th_prec = e_th;
```

Con queste informazioni è quindi possibile ricavare il valore di derivata ed integrale. Si osserva che la derivata viene già moltiplicata per il relativo guadagno, per verificare se non è in saturazione prima di eseguire una fuzzificazione. Il valore dell'integrale viene approssimato mediante un trapezio.

```
// errore ciclo attuale
e_x = x_set - x;

// derivata errore con saturazione tra -100 e 100
d_e_x = (double)( (e_x - e_x_prec) / Ts ) * GCE_x;
if (d_e_x > 100)
  d_e_x = 100;
if (d_e_x < -100)
  d_e_x = -100;

// integrale dell'errore (con approssimazione trapezio)
i_e_x = (double)(e_x + e_x_prec) / Ts;
```

È possibile ora eseguire il controllo fuzzy sulla x. Ricordiamo che alla derivata è già stato applicato il corrispettivo guadagno, mentre alla variabile di errore no.

```
// richiamo funzioni fc_1
fc_1->setInput(1, (double)e_x * GE_x);
fc_1->setInput(2, d_e_x);
fc_1->fuzzify();
out1 = fc_1->defuzzify(1);
th_set = (double)( (i_e_x * GIU_x) + out1 ) * GU_x;
```

La medesima procedura viene ripetuta con la variabile teta, variando ovviamente i guadagni.

```
// errore ciclo attuale
e_th = th_set - th;

// derivata errore con saturazione tra -100 e 100
d_e_th = (double)( (e_th - e_th_prec) / Ts ) * GCE_th;
if (d_e_th > 100)
  d_e_th = 100;
if (d_e_th < -100)
  d_e_th = -100;

// integrale dell'errore (con approssimazione trapezio)
```

```
i_e_th = (double)(e_th + e_th_prec) / Ts;

// richiamo funzioni fc_1
fc_1->setInput(1, (double)e_th * GE_th);
fc_1->setInput(2, d_e_th);
fc_1->fuzzify();
out2 = fc_1->defuzzify(1);
u = (double)( (i_e_th * GIU_th) + out2 ) * GU_th;
```

Viene infine eseguita una verifica affinché la variabile "u" ottenuta sia compresa tra -1 e 1, e successivamente sono inviati i comandi alle valvole, sfruttando la variazione di segno di u. Se u>0 il comando viene quindi mandato alle valvole 1 e 2, e viceversa alle valvole 3 e 4 se u<0.

```
if (u > 1)
  u = 1;
if (u < -1)
  u = -1;

// invio comandi alle valvole
if (u == 0) {
  analogWrite(valv_12_pin, 0);
  analogWrite(valv_34_pin, 0);
}
if (u > 0) {
  analogWrite(valv_12_pin, (double)u * 255);
  analogWrite(valv_34_pin, 0);
}
if (u < 0) {
  analogWrite(valv_12_pin, 0);
  analogWrite(valv_34_pin, (double)abs(u) * 255);
```

7.5.2 Controllore PID-like

Viene riportata un'immagine del controllo fuzzy sull'errore di posizione, per una migliore comprensione del listato.

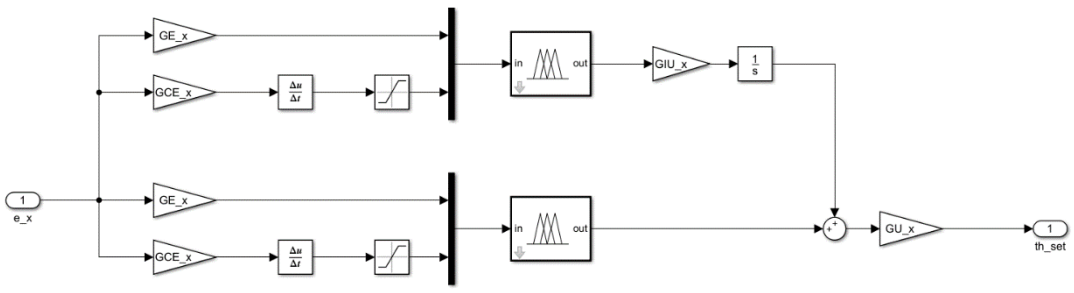


Figura 7-31: controllo PID-like sull'errore di posizione

Rispetto al caso PD+I vengono riportate solo le variazioni derivanti dal diverso controllo.

Rispetto al controllo PD+I si evidenzia che il controllo fuzzy è necessario due volte nel solo controllo della posizione. Il richiamo viene effettuato una sola volta in realtà, in quanto si genera il medesimo output, ma il secondo deve anche affrontare una integrazione prima di sommarsi col precedente (come è visibile nella figura del controllo riportata). Infatti, al calcolo di teta set la variabile out1 (uscita del fuzzy) viene richiamata due volte.

```
// richiamo funzioni fuzzy
fuzzy->setInput(1,(double)e_x*GE_x);
fuzzy->setInput(2,d_e_x);
fuzzy->fuzzify();
out1_prec=out1; // salvo il valore precedente
out1=fuzzy->defuzzify(1);

// integrale della funzione fc (con approssimazione trapezio)
i_e_fc_x=(double)(out1 + out1_prec)/ Ts;
th_set = (double)( out1*(i_e_fc_x*GIU_x) + out1 )*GU_x;
```

7.5.3 Controllore PD+PI

Viene riportata un'immagine del controllo fuzzy sull'errore di posizione, per una migliore comprensione del listato.

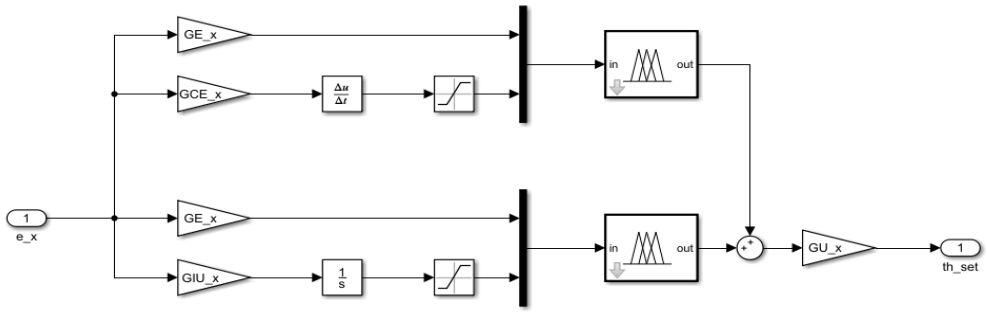


Figura 7-32: controllo PD+PI sull'errore di posizione

Rispetto al caso PD+I vengono riportate solo le variazioni derivanti dal diverso controllo.

È necessario in questo caso verificare che anche l'integrale, moltiplicato per il corrispettivo guadagni, sia compreso tra -100 e 100.

```
// integrale dell'errore (con approssimazione trapezio)
i_e_x = (double)( (e_x + e_x_prec) / Ts ) * GIU_x;
if (i_e_x > 100)
  i_e_x = 100;
```

```
if (i_e_x < -100)
  i_e_x = -100;
```

Come nel caso PID-like anche nel PD+PI è necessario richiamare il controllo fuzzy due volte all'interno del solo controllo di posizione, a differenza del caso precedente però si ottengono output differenti che vengono slavati nelle variabili out1 e out2.

```
// richiamo funzioni fuzzy
fuzzy->setInput(1, (double)e_x * GE_x);
fuzzy->setInput(2, d_e_x);
fuzzy->fuzzify();
out1 = fuzzy->defuzzify(1);

fuzzy->setInput(1, (double)e_x * GE_x);
fuzzy->setInput(2, i_e_x);
fuzzy->fuzzify();
out2 = fuzzy->defuzzify(1);
th_set = (double)( out1 + out2 ) * GU_x;
```

8 Comunicazione Arduino - Matlab & Simulink

Per una verifica dei programmi ottenuti è stata acquistata una scheda Arduino Mega 2560 rev3. I risultati sono stati soddisfacenti in quanto i programmi simulati con Proteus hanno dato i medesimi risultati implementandoli sulla scheda tramite Arduino IDE. Si è dimostrato come le librerie venissero implementate in modo esauriente all'interno del programma e come l'hardware sia stato implementato in modo corretto.

Il motivo principale dell'acquisto della scheda è stato però un altro. Si è infatti pensato di programmare la medesima non mediante Arduino IDE che è un linguaggio di programmazione di basso livello, ma mediante un linguaggio di alto livello come Matlab e Simulink. Questa scelta è giustificata dalla possibilità di poter usare funzioni più complesse come derivate, integrali, bande morte, o più semplicemente per un controllo in real time del processo senza la necessità di salvare dati da un monitor seriare, per un successivo post-processamento meno dispendioso in termini di tempo.

Per fare ciò è stato necessario aggiungere delle estensioni a Matlab tramite Get Hardware Support Package: MATLAB Support Package for Arduino Hardware e Simulink Support Package for Arduino Hardware.

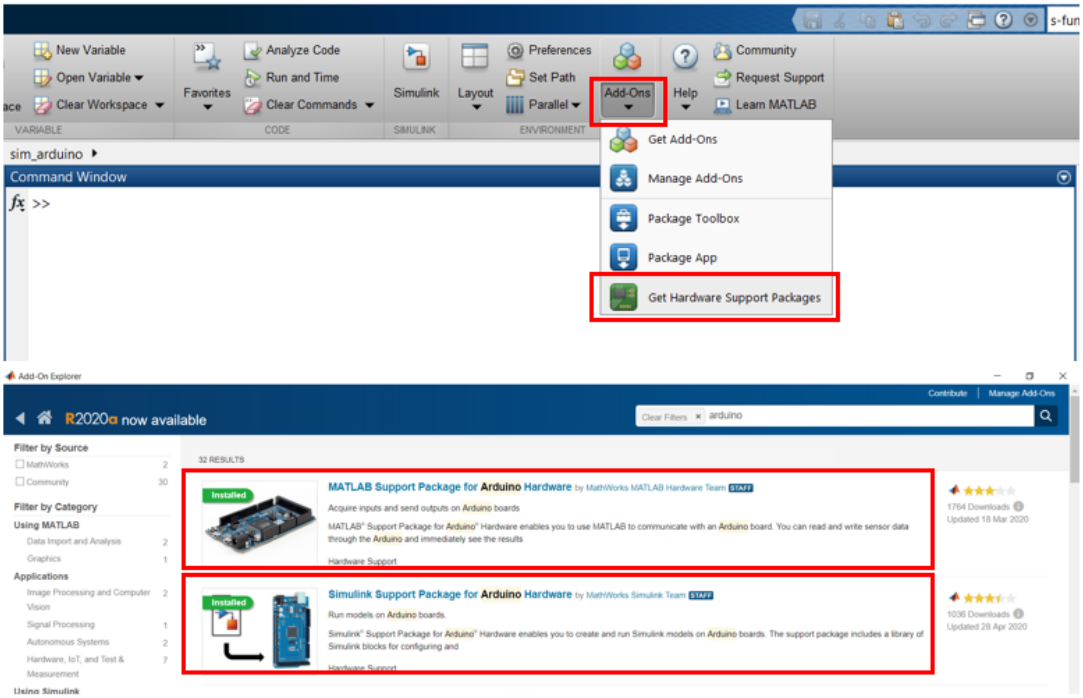


Figura 8-1: estensioni Arduino Hardware

Una volta installate le estensioni è possibile collegare la scheda Arduino al pc tramite una USB 2.0 tipo A/B e compariranno nella Command Window le seguenti frasi:

```
Arduino Mega 2560 detected.
```

This device is ready for use with MATLAB Support Package for Arduino Hardware. Get started with examples and other documentation.
This device is ready for use with Simulink Support Package for Arduino Hardware. Get started with examples and other documentation.

Per un'ulteriore verifica della comunicazione seriale e, per rilevare le proprietà della comunicazione, le librerie preinstallate, il nome identificativo della scheda, informazioni sui pin e porta di accesso si usa il comando:

```
a=arduino() ;
```

che restituisce le seguenti proprietà:

```
a =
  arduino with properties:
                Port: 'COM4'
               Board: 'Mega2560'
      AvailablePins: {'D2-D53', 'A0-A15'}
 AvailableDigitalPins: {'D2-D53', 'A0-A15'}
    AvailablePWMPins: {'D2-D13', 'D44-D46'}
  AvailableAnalogPins: {'A0-A15'}
   AvailableI2CBusIDs: [0]
          Libraries: {'I2C', 'SPI', 'Servo'}
```

Conviene ora pulire la variabile "a" per evitare errori durante l'esecuzione dei file Simulink. A questo punto è necessario settare correttamente il file Simulink per consentine il caricamento sulla scheda. Nel "Configuration Parameters", nella sezione "Solver" è necessario selezionare un solutore a passo fisso.

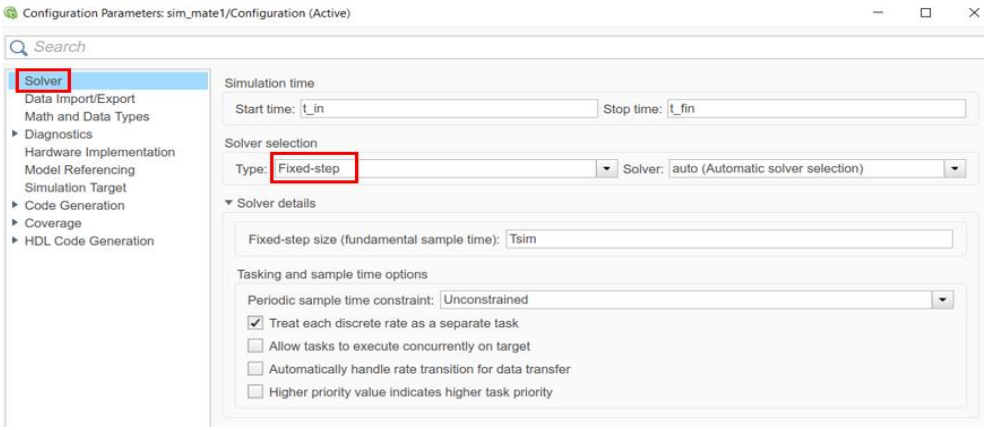


Figura 8-2: solutore a passo fisso

Per evitare possibili errori durante la conversione dei file, conviene nella sezione "Data Import/Export" deselezionare tutte le spunte, anche per non riempire la memoria della scheda con dati che non usiamo.

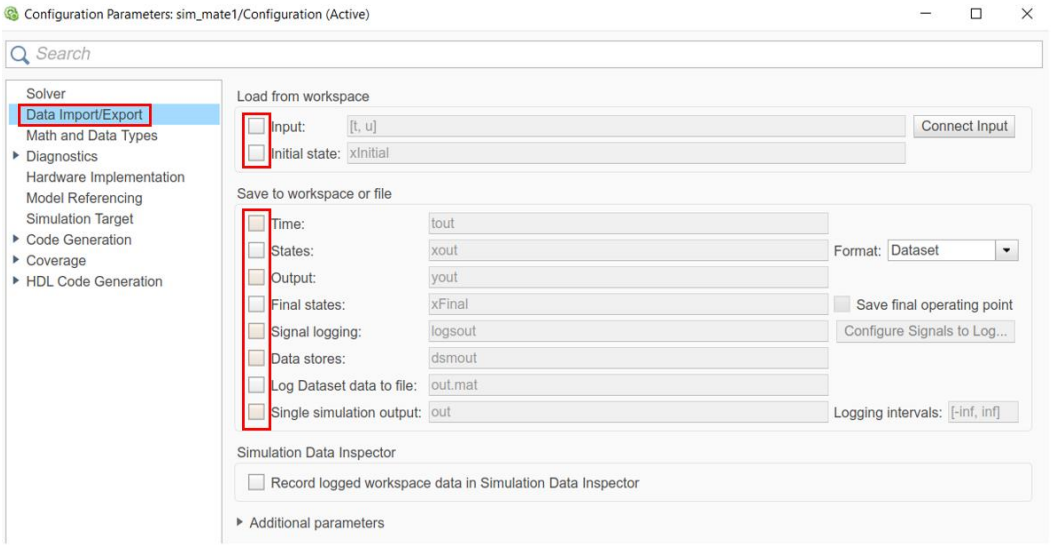


Figura 8-4: Data Import/Export

Nella sezione “Hardware Implementation” in “Hardware board” bisogna selezionare la scheda Arduino di riferimento, nel caso considerato la Mega 2560.

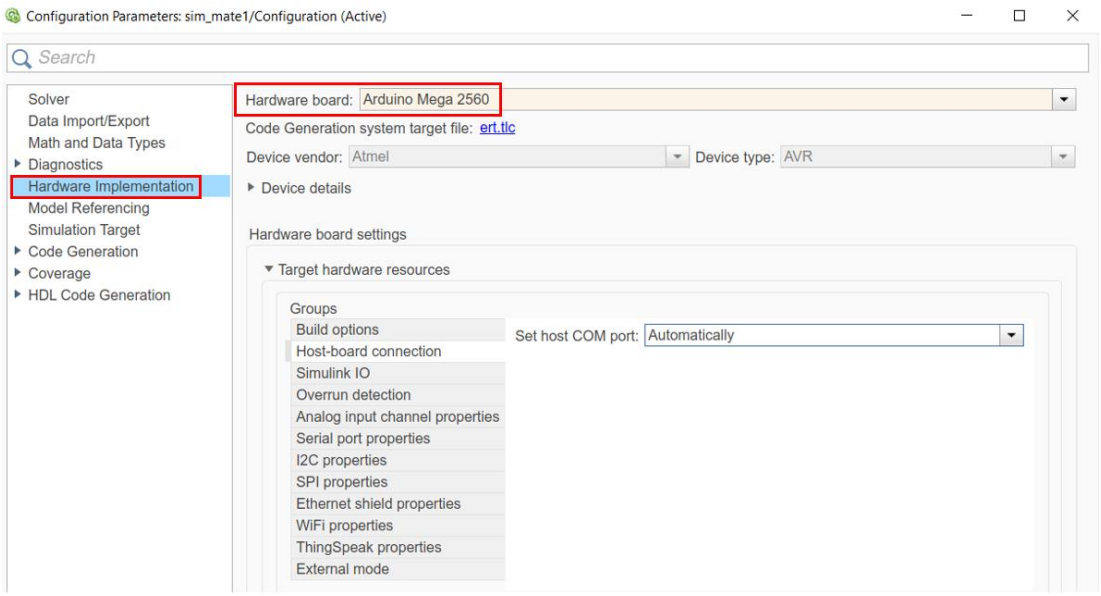


Figura 8-3: Hardware Implementation

Va sottolineato che potrebbe accadere che Simulink non riesca a leggere correttamente la porta alla quale è collegata la scheda, quindi è possibile inserirla manualmente escludendo il riconoscimento automatico in “Host-board connection” (il riconoscimento della porta era stato fatto prima manualmente nella Command Window tramite a=arduino( )).

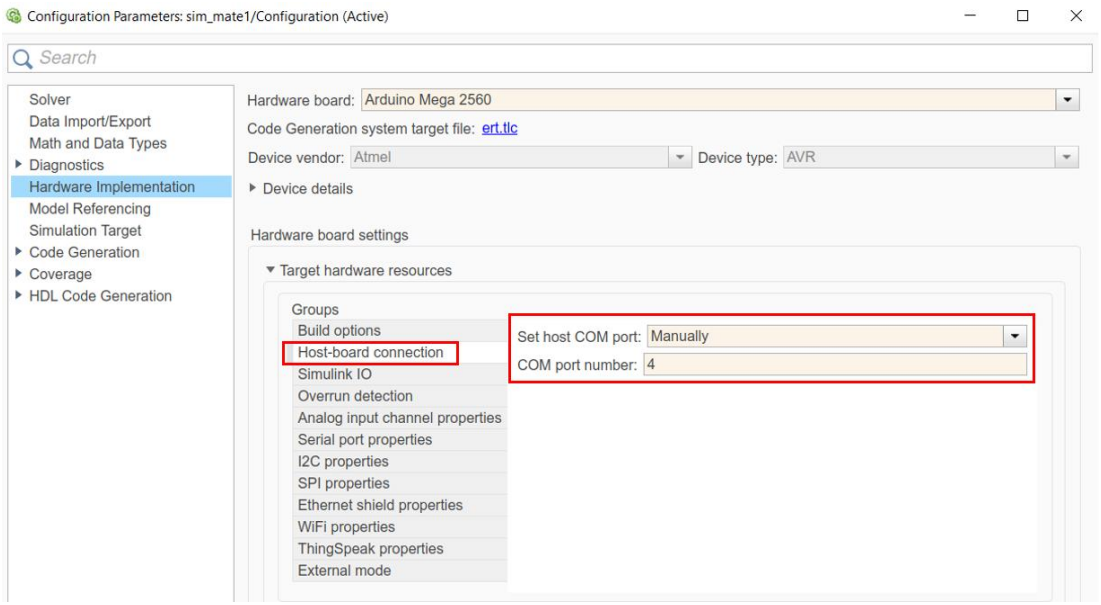


Figura 8-5: Host-board connection

La “Simulation Mode” va invece spostata dal tab Normal ad External per far in modo che il programma in Simulink venga eseguito dalla scheda Arduino e non dal pc. In questa modalità si ha la possibilità di eseguire un controllo in real time, ad esempio per un possibile aggiornamento dei guadagni più efficace. Oltretutto è possibile salvare le variabili all’interno di vettori senza saturare la memoria della scheda Arduino, visto che ad ogni loop di codice vengono inviati a Matlab tramite seriale, mantenendoli quindi salvati nella workspace per un efficace post processing.

Vi è inoltre la possibilità di caricare il file sulla scheda, in modo tale da poterla sconnettere dal pc (mantenendola ovviamente alimentata) per una possibile implementazione in laboratorio dove magari non è più necessario monitorare i parametri in real time, visto che il programma è già consolidato e la scheda svolge semplicemente il ruolo di controllore. Per farlo è necessario attivare il comando “Deploy to Hardware” e conviene impostare il tempo di esecuzione come infinito (inf) per non interrompere l’esecuzione dopo un tempo prefissato.

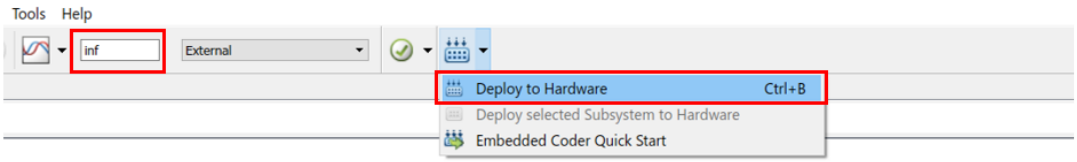


Figura 8-6: Deploy to Hardware

8.1 Conversione Fuzzy-Tool Matlab

Viene implementato il seguente schema Simulink già sviluppato precedentemente.

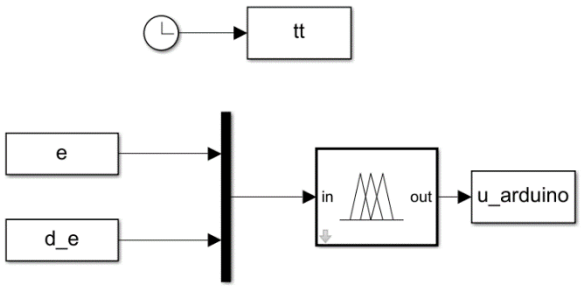


Figura 8-7: sim Fuzzy-Tool Matlab

Nella cartella sono inizialmente presenti questi file:

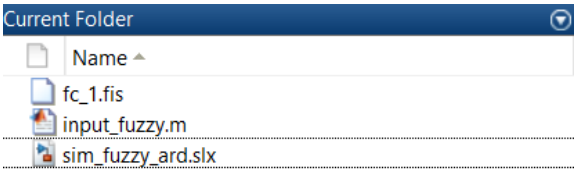


Figura 8-8: file iniziali

che sono rispettivamente:

- fc_1.fis: le regole del controllo fuzzy
- input_fuzzy.m: è un file da lanciare prima dell'esecuzione del file Simulink per caricare tutte le variabili al suo interno
- sim_fuzzy_ard.slx: è il file Simulink

è importante sottolineare che essendo il file Simulink eseguito su Hardware esterno non può essere lanciato dal file Matlab (.m), ma deve essere lanciato dal comando Run del file Simulink.

La prima volta che il file viene eseguito necessita un di tempo maggiore, in quanto vengono creati in modo automatico file e cartelle di conversione dal file Simulink in file leggibili dalla scheda Arduino, che vengono caricati sulla medesima. Viene generato il seguente report finita la compilazione, in caso di conversione riuscita.

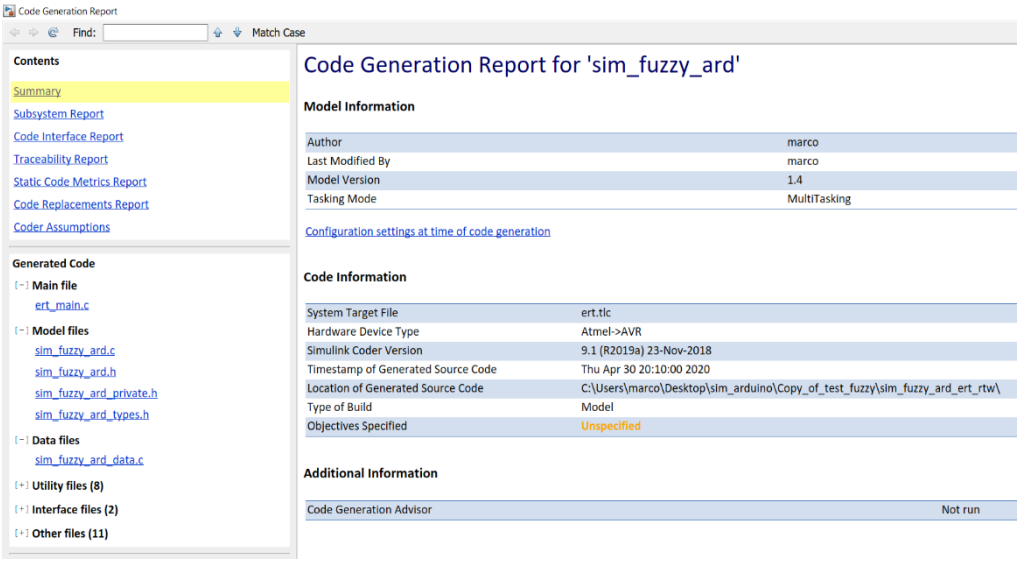


Figura 8-9: code generation report

In seguito, vengono creati nella cartella ulteriori file e cartelle, come mostrato in figura.

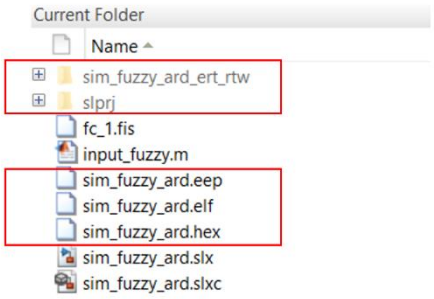


Figura 8-10: file creati dalla conversione

In modo molto intuitivo è possibile quindi programmare sulla scheda Arduino tramite un linguaggio di programmazione di alto livello.

Come verifica del corretto funzionamento del codice si sono confrontate le curve proposte per la verifica dalla linearità come valori significativi, lanciando il codice prima sul pc, poi sulla scheda Arduino. Le due curve di output risultano praticamente sovrapposte, a verifica della corretta conversione del file.

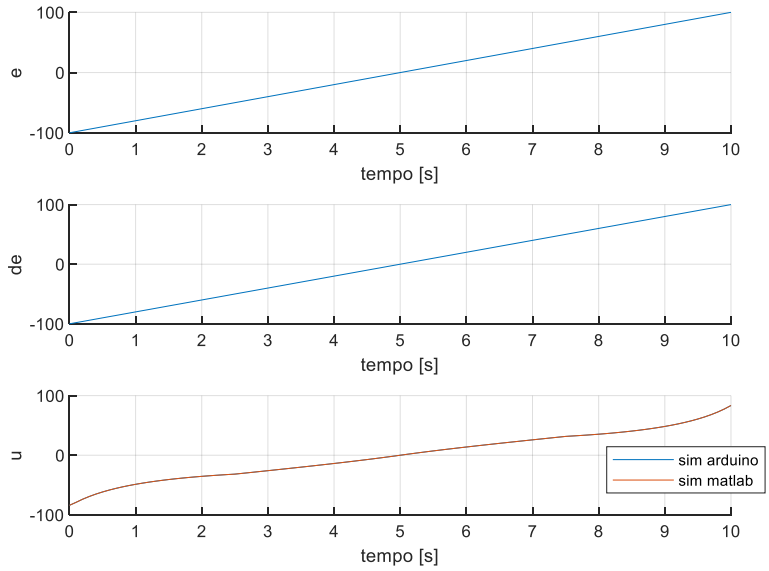


Figura 8-11: confronto sim su PC e su scheda Arduino

L'unico problema derivato da questo tipo di conversione è nella memoria occupata. Infatti, di seguito viene proposto un file Simulink che contiene tutto il controllore PD+I, dove sono necessari due controllori fuzzy più tutti i vari guadagni proporzionali, derivativi e integrativi.

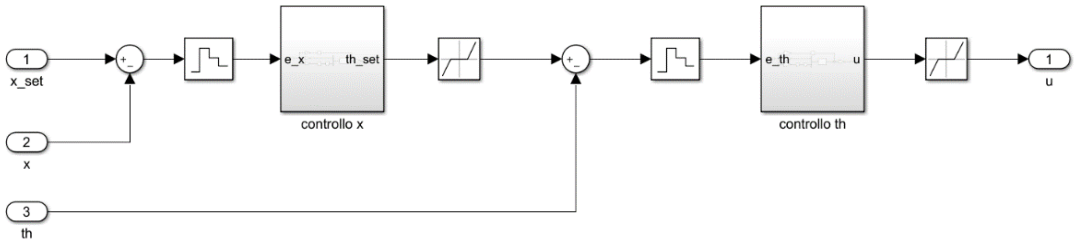


Figura 8-12: controllo Fuzzy x e teta

In questo caso lanciando la simulazione sulla scheda il Diagnostic Viewer restituisce un errore indicando che la memoria dati è stata occupata al 100.2% mentre quella del programma solo del 11.4%, questo è indice che la memoria occupata dalla conversione del Fuzzy-Tool è molto elevata. La saturazione della memoria rende impossibile caricare, e di conseguenza eseguire il codice su questa scheda.

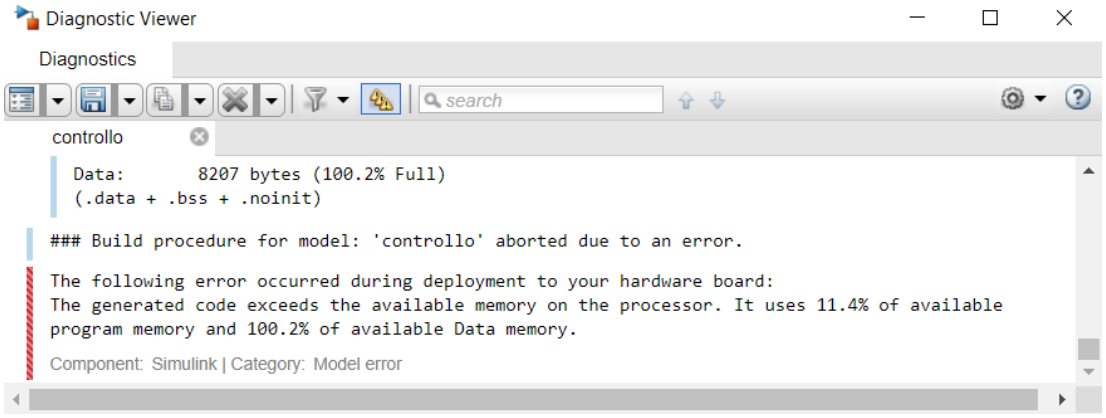


Figura 8-13: errore saturazione memoria

Con una scheda Arduino dotata di maggiore memoria potrebbe essere infatti possibile eseguire il codice.

8.2 S-Function Builder

Una possibile soluzione a questo problema di saturazione della memoria delle variabili può essere l'utilizzo del S-Function Builder, che consente una conversione di codice da un linguaggio di basso livello, come Arduino IDE che è scritto in C++, ad uno di alto livello come in Matlab&Simulink, visto che l'unica funzione che occupa una percentuale elevata di memoria è comunque già stata sviluppata in Arduino IDE.

Di seguito viene spiegato come settare correttamente questa funzione e come caricare le librerie necessarie per l'esecuzione del controllo fuzzy.

Inizialmente è necessario accedere alla directory dove si è salvata la libreria fuzzy in Arduino, ovvero la eFLL-master. Accedendo alla cartella bisogna copiare tutte le librerie, ovvero i file con estensione ".h" e ".cpp" per poi copiarli nella cartella di lavoro di Matlab. In figura per una maggiore chiarezza vengono mostrati i file iniziali della cartella ed evidenziati i file da libreria.

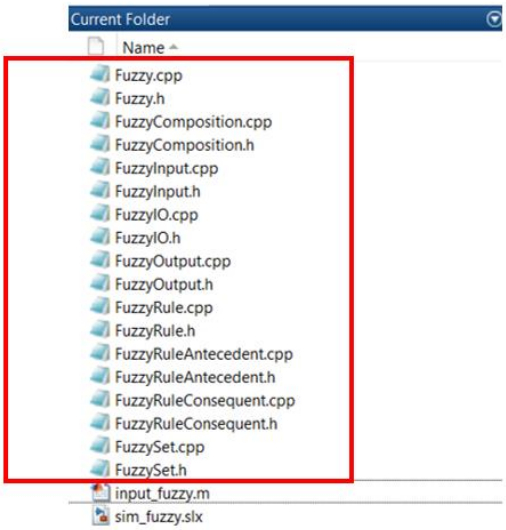


Figura 8-14: librerie fuzzy

Nel file Simulink si aggiunge il blocco S-Function Builder, che andiamo ad inizializzare correttamente. Assegnamo un nome simbolico alla funzione, ad esempio "controllo_fuzzy". Cambiamo il linguaggio da "Inherit from model settings" in "C++" dato che può capitare che a volte non riesca a riconoscere correttamente il linguaggio proposto, ad esempio nel mio caso veniva riconosciuto come un codice C e non C++.

Nel tab "Inizialization" il numero di stati discreti va portato da 0 ad un valore di 1.

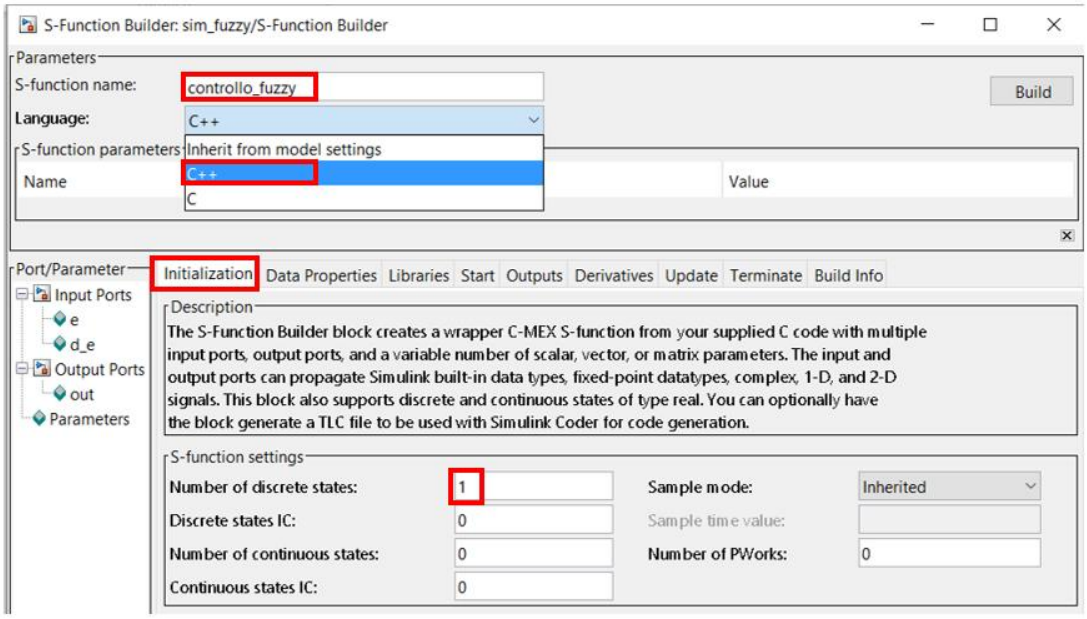


Figura 8-15: S-Function Builder set 1

Nel tab “Data Properties” in “Input ports” si creano le variabili di input, è possibile assegnare un nome simbolico, cancellarle se create per errore, riordinarle. È importante verificare che la dimensione sia 1-D e che siano reali. In alto a sinistra è presente un visualizzatore per poter verificare in ogni scheda il nome che si è attribuito alle variabili per una corretta scrittura del codice.

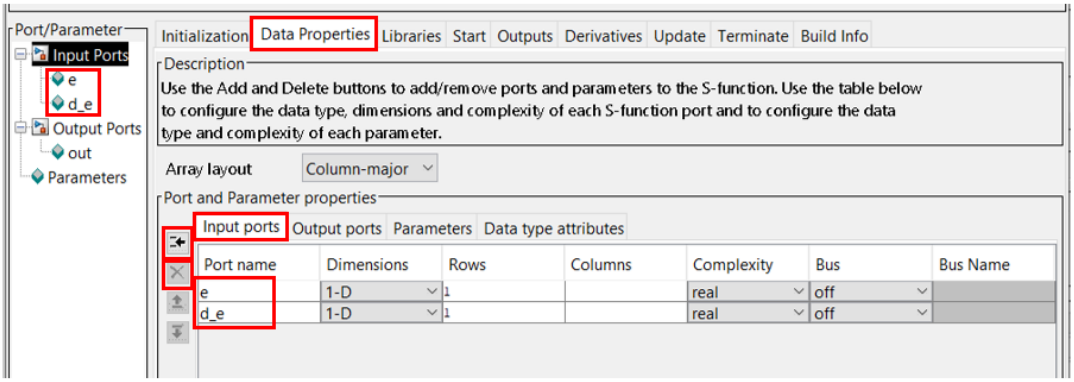


Figura 8-16: S-Function Builder set 2

Lo stesso procedimento in “Output port” per le variabili di output. In “Parameters” si ha invece la possibilità di definire le costanti da usare nel codice (in questo caso non necessarie) mediante la medesima procedura.

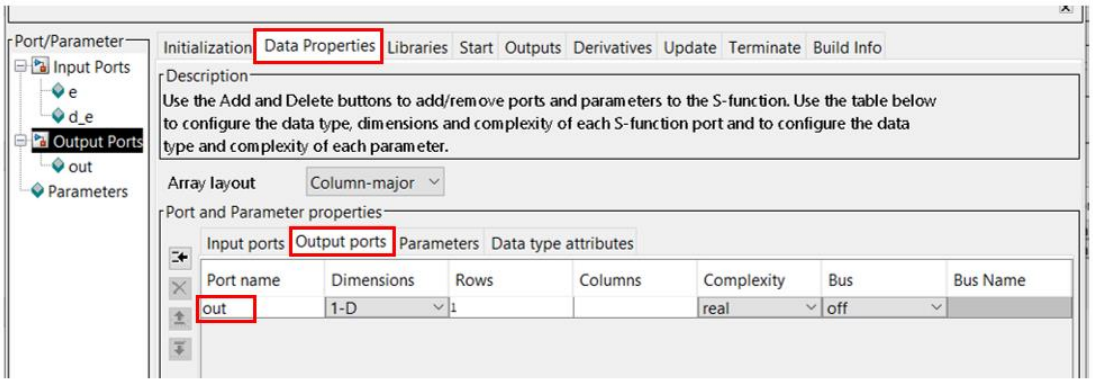


Figura 8-17: S-Function Builder set 3

Nel tab “Libraries” vengono caricate le librerie che devono essere nella stessa directory del file Simulink. Nella casella “Includes” tramite il comando #include si caricano le librerie.

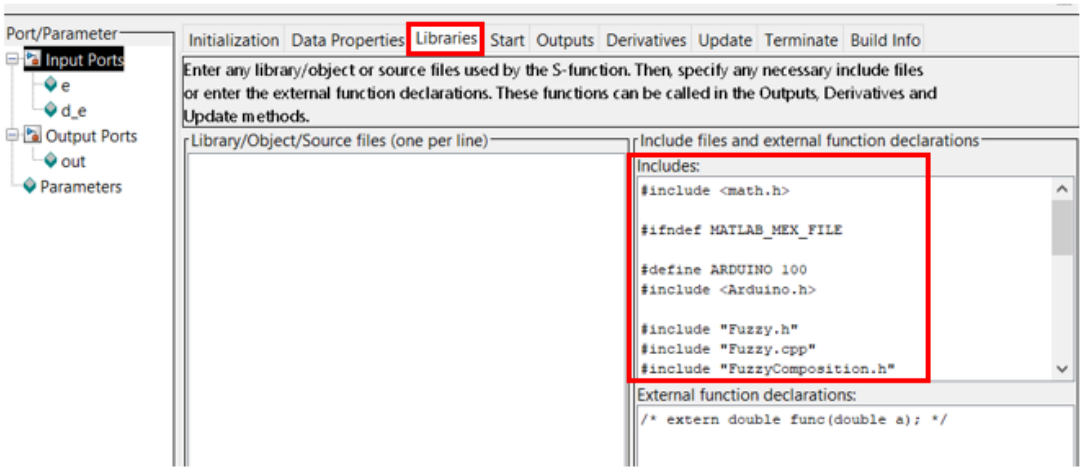


Figura 8-18: S-Function Builder set 4

Vengono in seguito rappresentate tutte le righe di codice da aggiungere per un corretto funzionamento. La libreria math.h è di default e non va eliminata. Va sottolineato che le librerie interne già predisposte in Matlab vanno aggiunte tramite la dicitura #include <libreria.h> come math.h e Arduino.h, infatti di queste non è necessario aggiungere il file di sistema nella cartella di lavoro. Mentre per le librerie esterne si usa #include “libreria.h” e #include “libreria.cpp” ed è necessario che risiedano nella medesima directory del file Simulink.

Tra le stringhe #ifndef MATLAB_MEX_FILE e #endif vanno caricati tutti gli elementi per la comunicazione con Arduino, non scordarsi di inserire #define ARDUINO 100 e #include <Arduino.h>, che sono di vitale importanza per le funzioni base di Arduino. Oltre alla librerie in questa sezione è importante anche inserire l’elemento che contiene il controllore fuzzy: Fuzzy *fc_1 = new Fuzzy(), più in generale definire tutti quegli elementi che in Arduino IDE erano esterni sia al void setup() che al void loop().

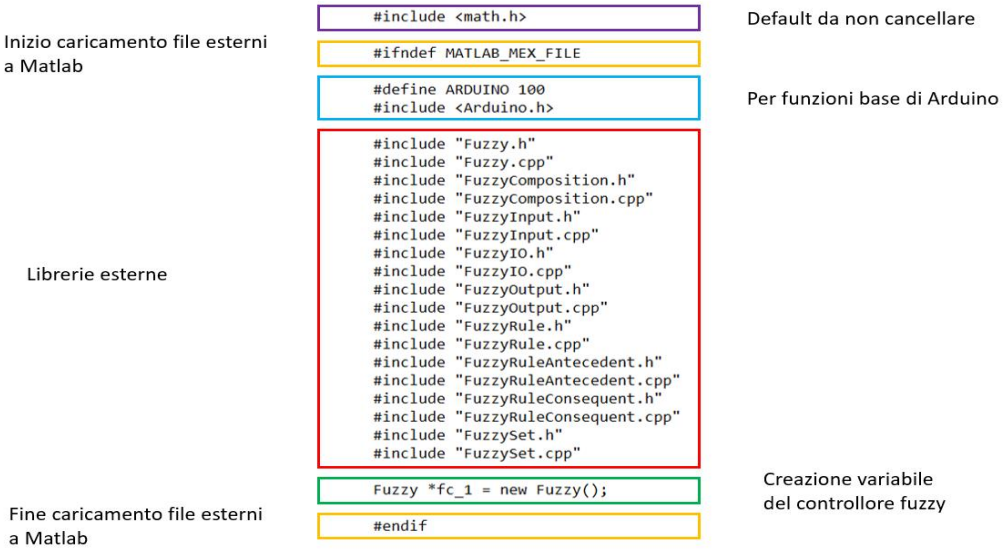


Figura 8-19: Includes

Nel tab "Update" viene invece inserito il contenuto della funzione void setup() del codice Arduino IDE. Nel nostro caso l'unico elemento che è presente è la funzione Set_LogicaFuzzy() che contiene le creazione del controllo fuzzy. Per evitare errori di compilazione non viene riportata la funzione, ma direttamente il contenuto della medesima.

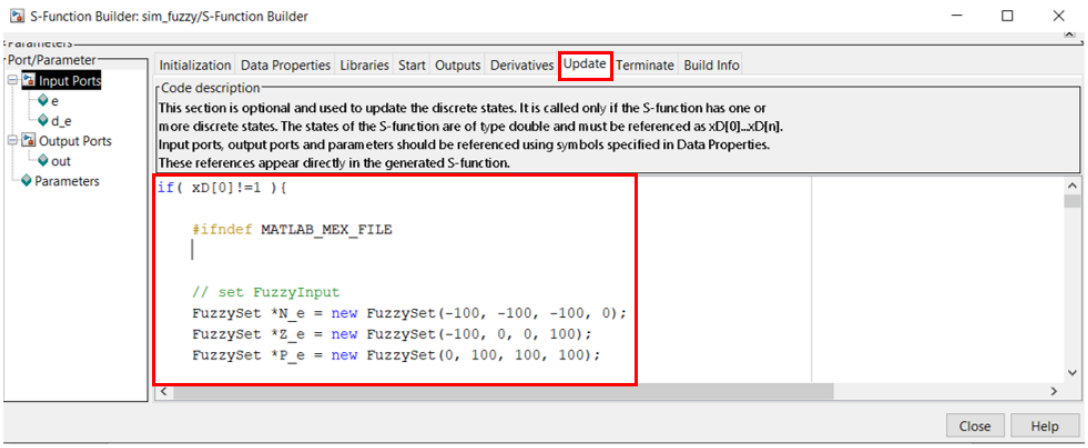


Figura 8-20: S-Function Builder set 5

Come nel tab delle librerie tra le stringhe #ifndef MATLAB_MEX_FILE e #endif vanno inserite le righe di codice provenienti dal listato di Arduino. Tutto il codice è sviluppato all'interno di un if(), che verrà eseguito come il void setup() una sola volta, quando la variabile xD[0] che rappresenta lo stato discreto vale zero; infatti a fondo codice gli viene poi assegnato il valore unitario per accedere, come vedremo, al void loop(). In figura è rappresentato uno schema del codice.

```
if( xD[0]!=1 ){
    #ifndef MATLAB_MEX_FILE
    ..... Set_LogicaFuzzy(); .....
    #endif
    xD[0]=1;
}
```

Figura 8-21: conversione void setup ( )

Nel tab "Output" viene inserito il contenuto della funzione void loop() all'interno di un if() la cui condizione è che lo stato discreto xD[0] sia pari ad uno, in modo tale da eseguirlo dopo il void setup(). Una differenza di scrittura rispetto al file in Arduino IDE è che le variabili vengono richiamate nel seguente modo: nome_variabile[0], ovvero bisogna aggiungere la [0] dato che facciamo riferimento a variabili 1-D e vogliamo accedere alla loro unica cella di memoria disponibile (la numero 0).

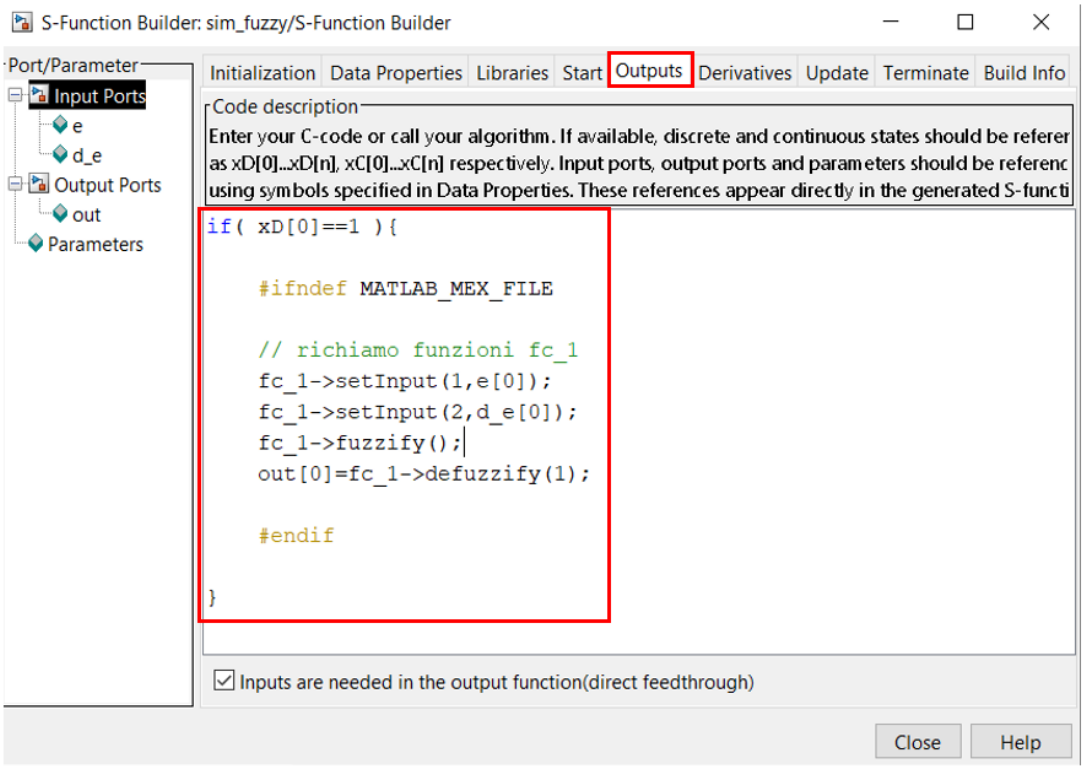


Figura 8-22: S-Function Builder set 6

Una volta compilati tutti i campi è possibile col tasto "Build" creare la funzione. In "Build info" è possibile osservare la corretta creazione dei file inerenti alla funzione in esame.

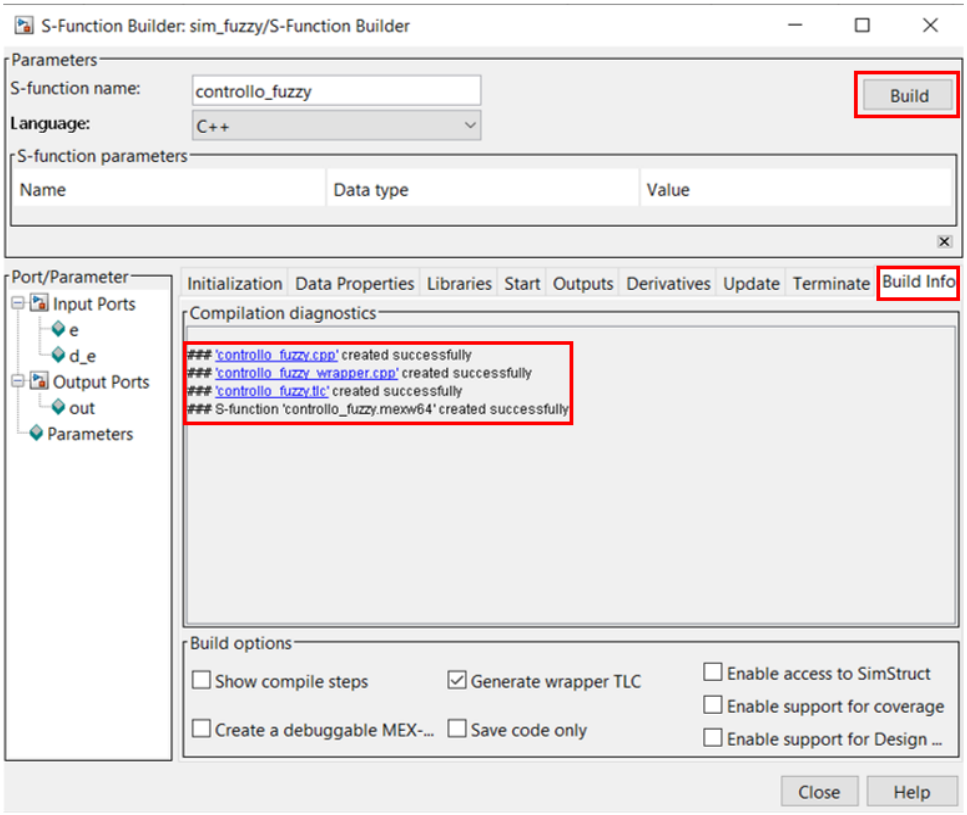


Figura 8-23: S-Function Builder set 7

Nella directory di lavoro vengono creati i seguenti file in seguito alla costruzione della S-Function Builder.

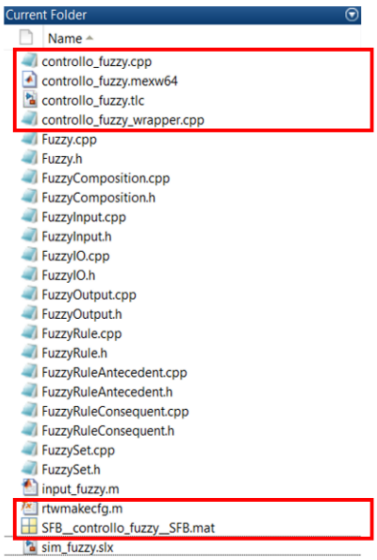


Figura 8-24: file creati da S-Function Builder

Per garantire un corretto caricamento delle variabili all'interno del file Simulink bisogna aprire il file controllo_fuzzy_wrapped.cpp (in generale il file che presenta la dicitura "_wrapped.cpp") ed aggiungere prima delle due funzioni void, contenute al suo interno, la stringa: extern "C". In seguito, bisogna salvare il file e ricordarsi che ogni qualvolta si ricostruisce la S-Function Builder questo file andrà sempre riaggiornato, altrimenti il file Simulink restituisce un errore. Un esempio di modifica delle funzioni void è riportato in figura.

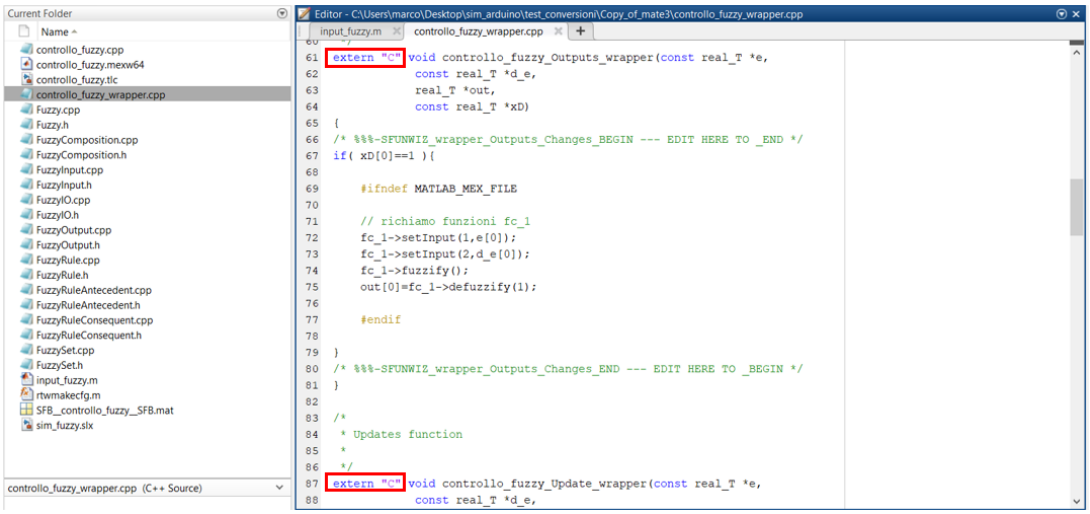


Figura 8-25: aggiunta extern "C"

Il modello Simulink implementato è quindi il seguente.

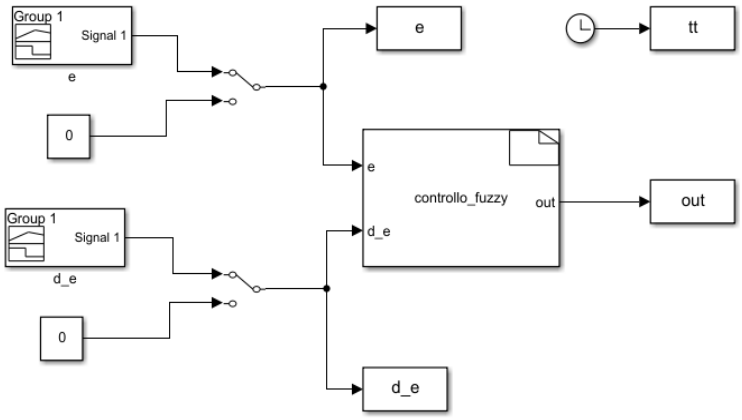


Figura 8-26: file Simulink controllo

Lanciando il programma e successivamente aprendo il Diagnostic Viewer è possibile notare come il programma circa dopo occupi 11.9% della memoria, come nel caso della conversione del fuzzy tool di Matlab in cui si occupava 11.4%, mentre la memoria dati occupa solo il 18.6% della totale con S-Function Builder e non più il 100.2%.

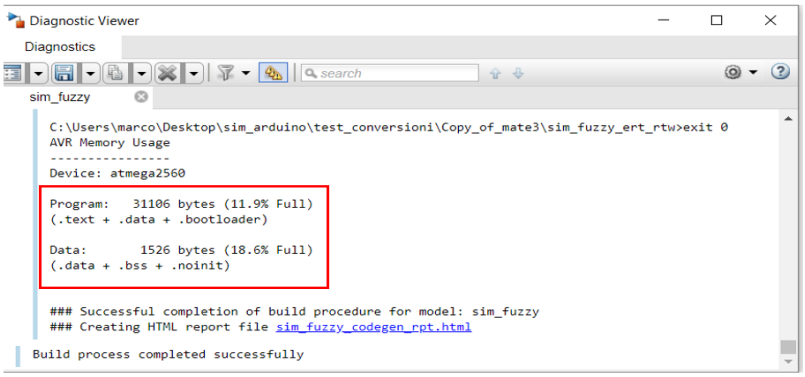


Figura 8-27: memoria occupata con S-Function Builder

Allegato in appendice viene riportato per completezza il file Arduino a cui fare riferimento per la conversione, nella sezione "Codice Arduino per verifica libreria Fuzzy". Si ricorda inoltre che alcune funzioni presenti nel codice sono implementati esternamente alla funzione S-Function Builder nel file Simulink, come le scritte e video o la comunicazione seriale.

Viene rappresentato un confronto, sempre sfruttando l'analisi di linearità come parametro, tra il fuzzy tool di Matlab eseguito su pc e del S-Function Builder eseguito su scheda Arduino. Dall'analisi del grafico si evidenzino minime differenze, che si erano già riscontrate in modo analogo lanciando il file da Arduino IDE su Proteus. Si evince quindi che il file lavora correttamente.

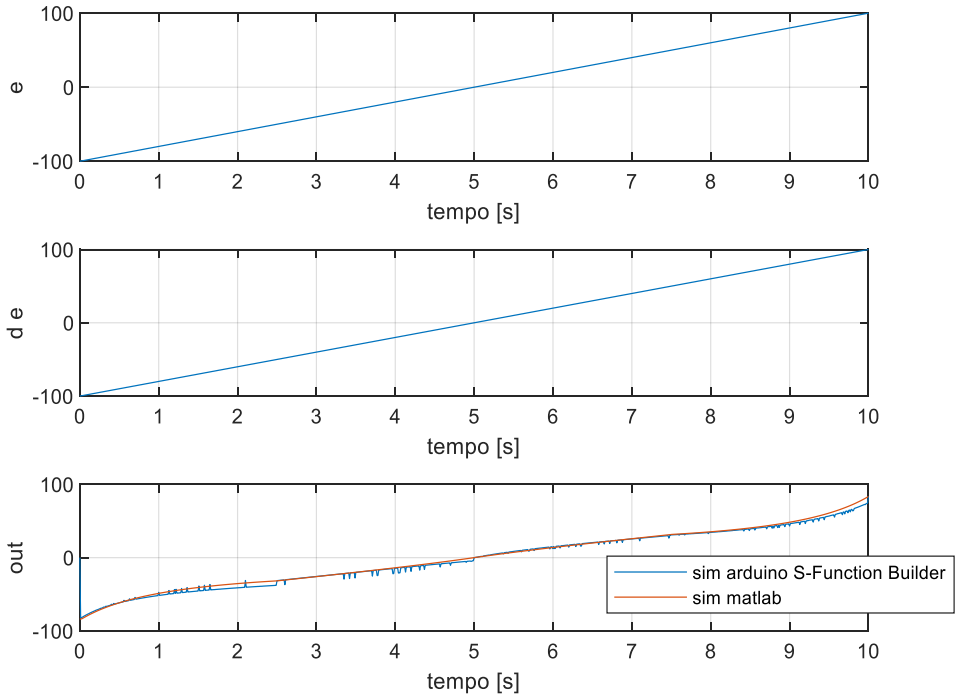


Figura 8-28: S-Function Builder vs Fuzzy-Tool Matlab

8.3 Controllore PD+I con S-Function Builder

È possibile a questo punto verificare che ci sia sufficiente spazio in memoria per ospitare tutto il controllore. Viene perciò proposto il seguente modello.

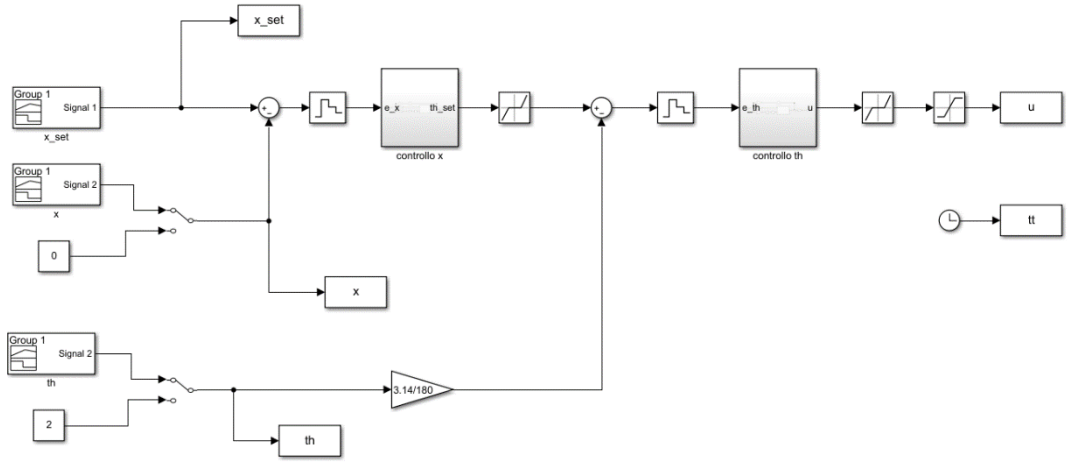


Figura 8-29: controllo PD+I

Controllo x è:

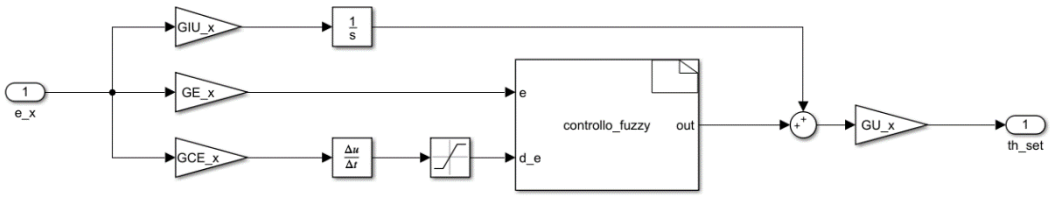


Figura 8-30: controllo x

Controllo teta è:

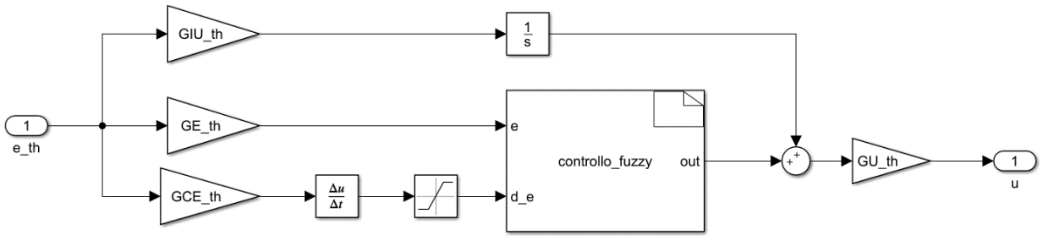


Figura 8-31: controllo teta

La S-Function Builder è quella illustrata nel precedente paragrafo.

La memoria occupata dal programma è del 13.5% mentre la memoria dati del 23.3% ovvero la scheda è in grado di immagazzinare completamente il programma.

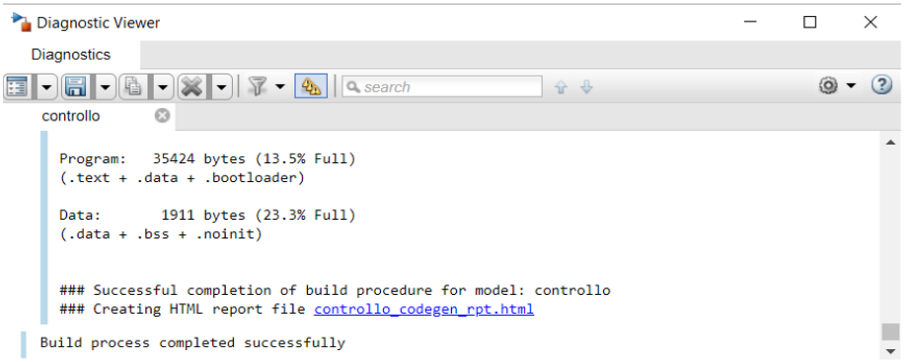


Figura 8-32: memoria occupata dal controllo PD+I

Lo scopo del seguente listato era quello di verificare il caricamento del controllore, infatti come input sono stati presi dei valori simbolici per simulare il funzionamento.

Una funzionalità molto comoda del Simulink Support Package for Arduino Hardware, non ancora mostrata, è la possibilità di interfacciarsi comodamente ai pin per la lettura di segnali o per l'invio. Nelle librerie di Simulink infatti abbiamo una ulteriore estensione mostrata in figura.

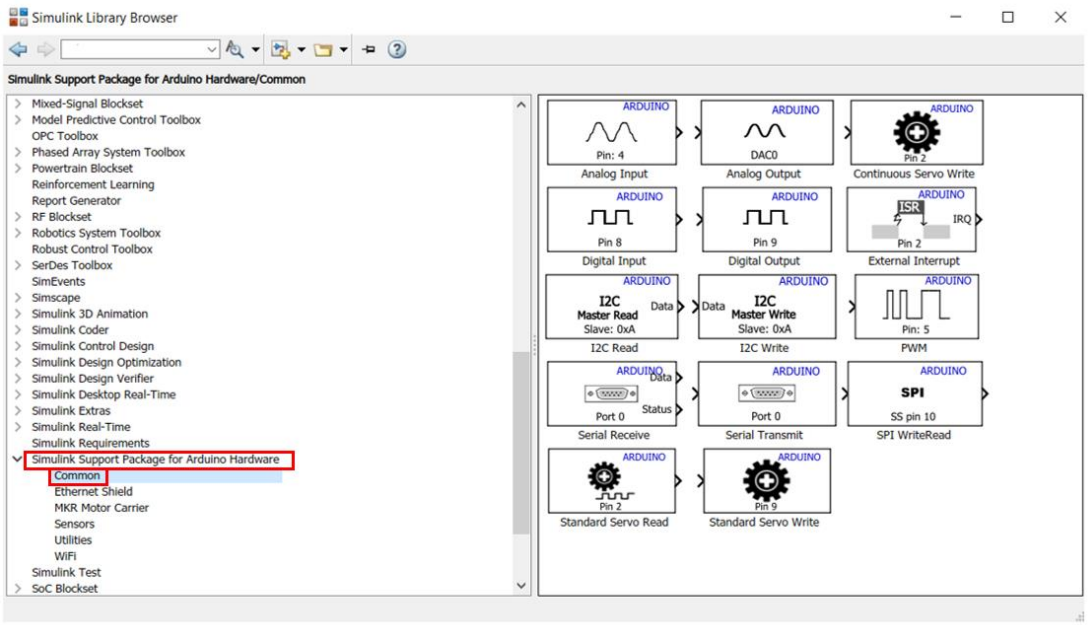


Figura 8-33: libreria Simulink Support Package for Arduino Hardware

Nell'applicazione implementata serve usare l'input analogico e l'output in PWM. La funzione input analogico permette di selezionare la porta da usare e richiede il tempo di campionamento.

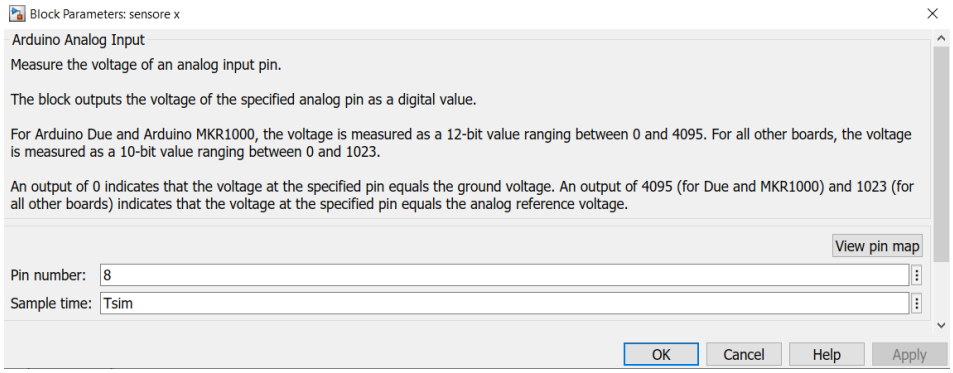


Figura 8-34: input analogico

La funzione output in PWM richiede invece soltanto il pin.

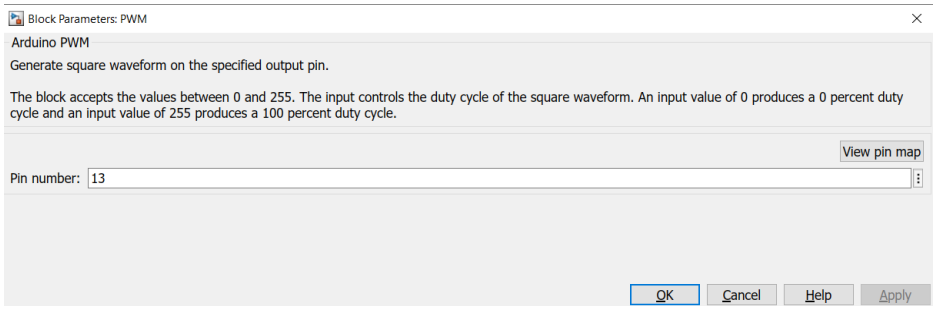


Figura 8-35: output PWM

Per entrambe le funzioni è possibile visualizzare una mappa dei pin delle schede principali, risultando molto comodo nel caso la scheda non sia facilmente accessibile o reperibile.

arduinoPinMapping.html

	Blocks	Target Hardware							
		Uno	Nano 3.0	Leonardo	Micro	Mega 2560/ADK	Due	Robot Control Board	Robot Motor Board
User Selectable Pins	Digital Input	0-13	0-13	0-19	0-19	0-53	0-53	0-3,5-17,19-22	0-19
	Digital Output	0-13	0-13	0-19	0-19	0-53	0-53	0-3,5-17,19-22	0-19
	Analog Input	0-5	0-7	0-11	0-11	0-15	0-11	0,1,2,3,4,5,6,7,11	0,1,2,3,4,5,6,11
	Analog Output	N/A	N/A	N/A	N/A	N/A	DAC0, DAC1	N/A	N/A
	PWM	3,5,6,9,10,11	3,5,6,9,10,11	3,5,6,9,10,11,13	3,5,6,9,10,11,13	2-13, 44-46	2-13	N/A	N/A
	Standard Servo Read	0-13	0-13	0-19	0-19	0-53	0-53	19,20,21,22	4,12,18,19
	Standard Servo Write	0-13	0-13	0-19	0-19	0-53	0-53	19,20,21,22	4,12,18,19
	Continuous Servo Write	0-13	0-13	0-19	0-19	0-53	0-53	19,20,21,22	4,12,18,19
	External Interrupt	2,3	2,3	0,1,2,3,7	0,1,2,3,7	2,3,18-21	0-53	0,1,2,3,7	0,1,2,3,7
	SPI Slave Select (SS)	0-10	N/A	0-19	0-19	0-49, 53	4,10,52	0-19	0-19

Figura 8-36: mappa dei pin

Viene quindi costruito un modello del controllore sfruttando questa nuova libreria, per una implementazione nel banco prova. Il modello viene mostrato in figura.

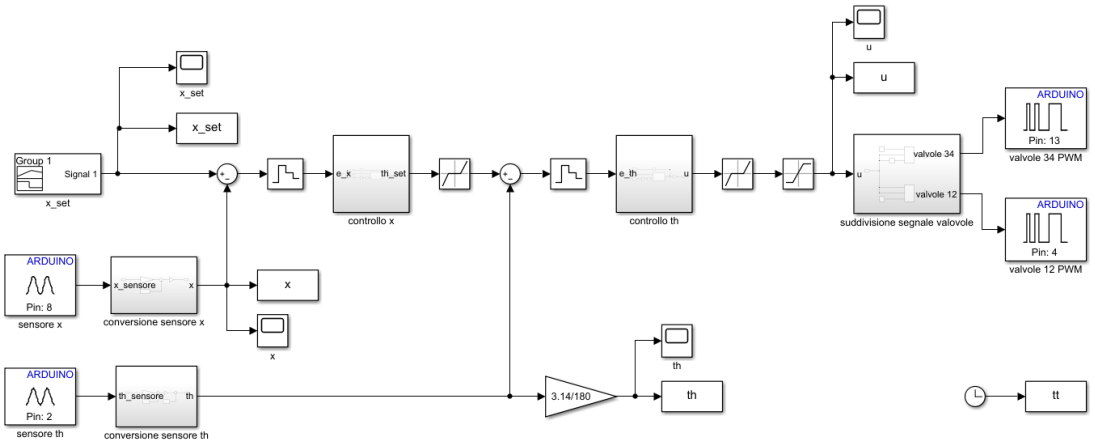


Figura 8-37: modello implementazione sul banco

Come era stato evidenziato nel file scritto in Arduino IDE vengono usati i seguenti pin:

- Pin A8: per la lettura della posizione x, tramite LVDT
- Pin A2: per la lettura della posizione angolare teta, tramite traduttore angolare
- Pin 13: per il comando alle valvole 3 e 4
- Pin 4: per il comando alle valvole 1 e 2

Tramite i blocchetti To Workspace è possibile fare facilmente un'analisi dei dati a fine processo, mentre tramite i blocchetti scope è possibile fare un'analisi in real time del controllore.

All'interno del blocco conversione sensore x, viene fatta una conversione da una lettura analogica, che fornisce 1024 valori ovvero da 0 a 1023, del sensore x dandogli anche un offset, visto che la lettura del sensore va da 0 a 0.5, con questa conversione forniamo in output valori tra -0.25 e 0.25.

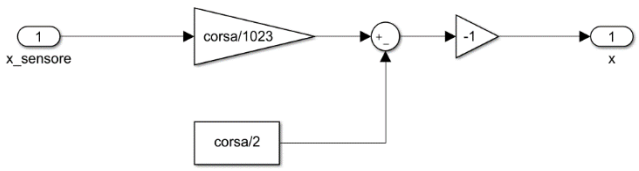


Figura 8-38: conversione sensore x

All'interno del blocco conversione sensore teta, viene fatta una conversione da una lettura analogica, che fornisce 1024 valori ovvero da 0 a 1023, del sensore teta dandogli anche un

offset, visto che la lettura del sensore va da 0° a 90°, oltretutto per come viene montato il sensore tale valore va ancora corretto con th_pos.

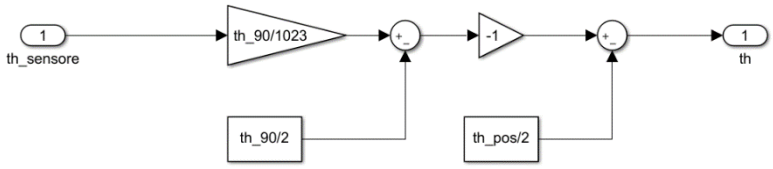


Figura 8-39: conversione sensore teta

Per la suddivisione del segnale tra le valvole viene implementato il seguente blocco, tenendo conto che se il segnale è positivo va alle valvole 1 e 2, mentre se negativo, il suo valore assoluto va alle valvole 3 e 4. All'uscita è inoltre presente un guadagno visto che deve essere inviato al pin di output un segnale in PWM che prevede 256 valori, da 0 a 255.

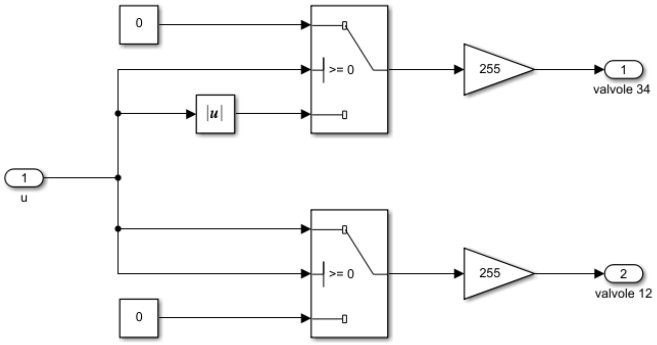


Figura 8-40: suddivisione del segnale tra le valvole

Un ulteriore vantaggio non ancora analizzato è la possibilità di generare fuzzy set con funzioni complesse tramite Matlab o Simulink come ad esempio gradini ad un determinato istante o funzioni triangolari e trapezoidali, facilmente implementabili con il Signal Builder.

In allegato viene riportato il codice che è necessario lanciare prima dell'esecuzione del file Simulink.

Il medesimo procedimento può essere ripetuto anche per i controllori PID-like e PD+PI.

9 Pendolo inverso in Simscape-Multibody

Per ridurre il grado di approssimazione della simulazione del banco prova viene proposto di modellare le equazioni di equilibrio meccanico del pendono inverso con Simscape-Multibody, invece che mediante equazioni in solo Simulink, che presentano approssimazioni, tra le quali trascurare l'inerzia dell'asta del pendolo.

Viene quindi modificato il sottosistema meccanico del modello come mostrato in figura.

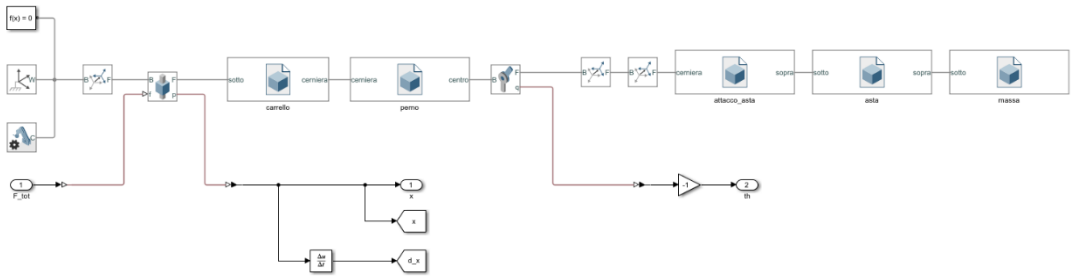


Figura 9-1: sottosistema meccanico Simscape-Multibody

I blocchi inizialmente presenti sono il "World Frame" che imposta un sistema di riferimento di base per i corpi, il "Mechanism Configuration" che consente di impostare un set di forze esterne agenti su tutti i corpi, come ad esempio la forza di gravità nel caso considerato (porre molta attenzione su quale dei tre assi impostare la forza), ed il "Solver Configuration" di Simscape-Multibody.

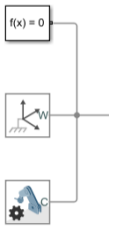


Figura 9-2: blocchi di partenza

Si è quindi proceduto modellando l'assieme di corpi, che costituiscono il sistema del pendolo inverso in Solidworks e, successivamente, esportando le varie parti in formato ". step", si è reso leggibile in ambiente Simscape. Si è scelto di modellare in Solidworks e non con il modellatore nativo di Simscape, in quanto quest'ultimo non essendo propriamente un CAD necessiterebbe di creare moltissime componenti e collegarle tra loro, anche solo per creare una parte molto semplice come il carrello

Per importare i file step all'interno di Simscape si sfrutta il blocco "File Solid". Importando il file vengono perse le informazioni sul peso che possono però essere indicate in seguito. Del modello è anche possibile variare l'estetica con colore (espresso in RGB o da palette) e l'opacità.

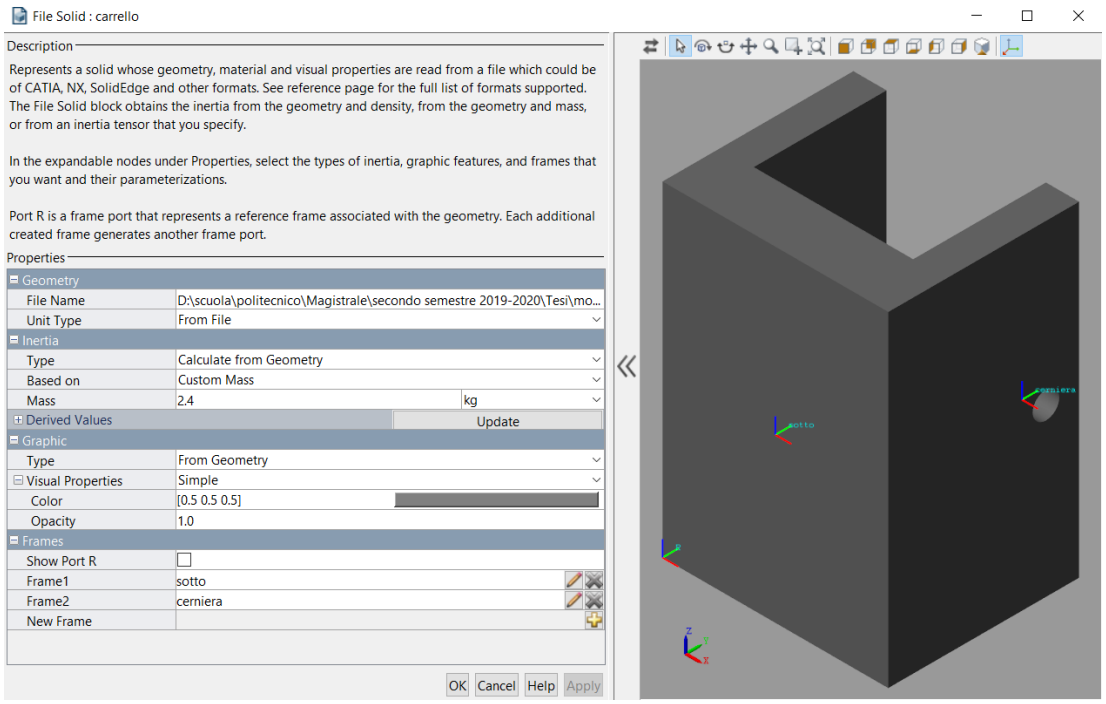


Figura 9-3: File Solid, Carrello

L'impostazione più importante è inserire i riferimenti (frame) precisi per poter collegare in seguito tra loro le parti correttamente. La modalità di assegnazione più comoda per i riferimenti è "Based on Geometric Feature" che consente, selezionando una feature, come ad esempio un

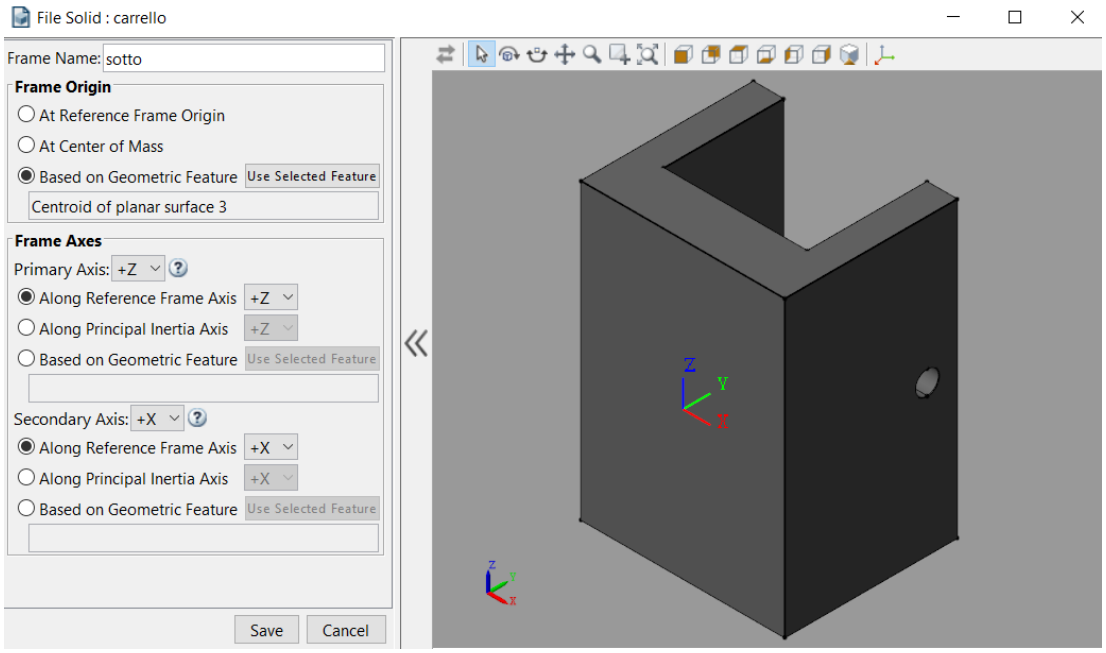


Figura 9-4: assegnazione frame.

foro o una superficie, di assegnargli un frame, eventualmente variando l'orientazione di essi se necessario, rendendo il successivo accoppiamento intuitivo come in Solidworks.
Il blocco esterno presenta un numero di collegamenti pari al numero di frame assegnati, nella parte "carrello" in esempio "sotto" e "cerniera" che erano stati precedentemente assegnati.

Figura 9-5: n. collegamenti= n. frame

È possibile collegare tra loro le parti rigidamente, come ad esempio il "carrello" e il "perno", o "l'attacco dell'asta" e "l'asta" semplicemente collegandoli tra loro.

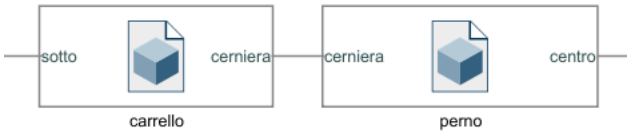


Figura 9-6: collegamento rigido

Tra il sistema di riferimento e il carrello viene inserita una guida prismatica per simulare lo spostamento del carrello lungo l'asse x.

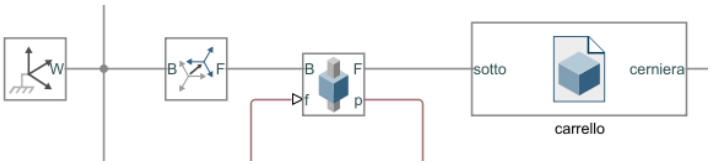


Figura 9-7: collegamento con guida prismatica

Prima della guida prismatica viene inserito un blocco "Rigid Trasform" per effettuare una rotazione rigida (o volendo anche una traslazione, mediante lo stesso blocco) tra un sistema di riferimento e l'altro, per orientare correttamente il sistema del carrello con quello di base.

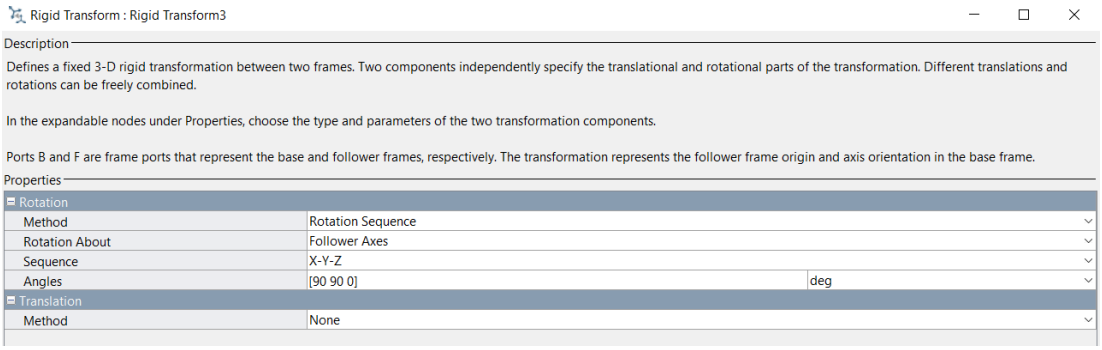


Figura 9-8: Rigid Trasform

Nel giunto prismatico viene installato il trasduttore di posizione LVDT come in laboratorio, quindi si seleziona tra i sensori quello di posizione, attivando la porta “p”. Sulla guida viene anche applicata una forza, quella proveniente dall’attuatore idraulico e dai disturbi esterni e, per implementarla, è necessario nella sezione “”Actuation” selezionare in “Force” la dicitura “Provided by Input”. Bisogna anche assegnare il coefficiente di smorzamento, come avevamo previsto anche con l’analisi precedente, di b=20.83 N/(m/s).

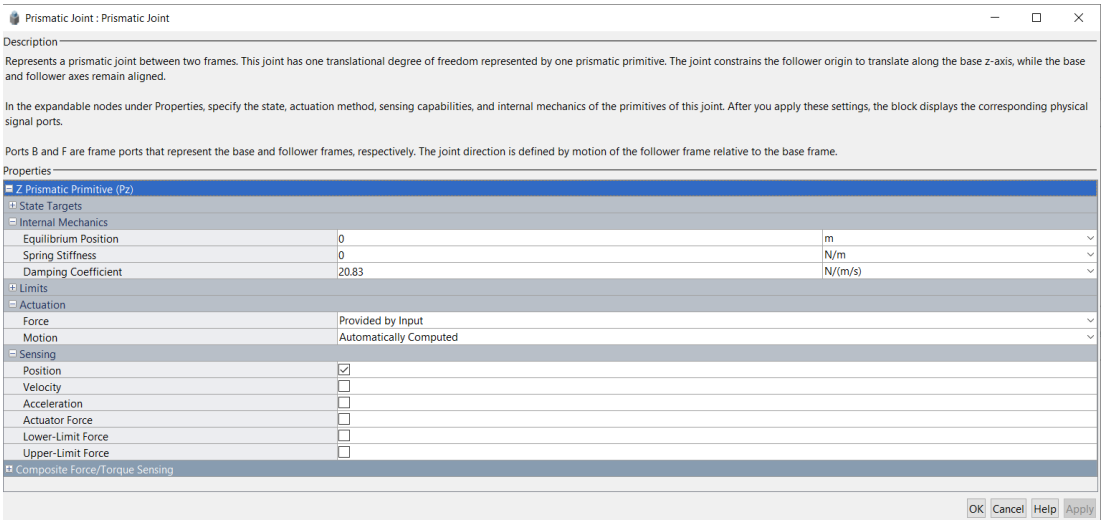


Figura 9-9: giunto prismatico

Per far comunicare tra loro gli elementi dell’ambiente Simulink e Simscape si usano il “Simulink-PS” per convertire da Simulink a Simscape, come ad esempio la forza in input nel sistema meccanico. Nelle impostazioni è necessario selezionare l’unità di misura del segnale in input, mentre si usa “PS-Simulink” per convertire da Simscape a Simulink, come lo spostamento x; anche in questo caso è necessario selezionare l’unità di misura del segnale di uscita. Si può osservare come nei due ambienti i collegamenti abbiano colori differenti, oltretutto Matlab non consente il loro collegamento se non mediante i blocchi di conversione.

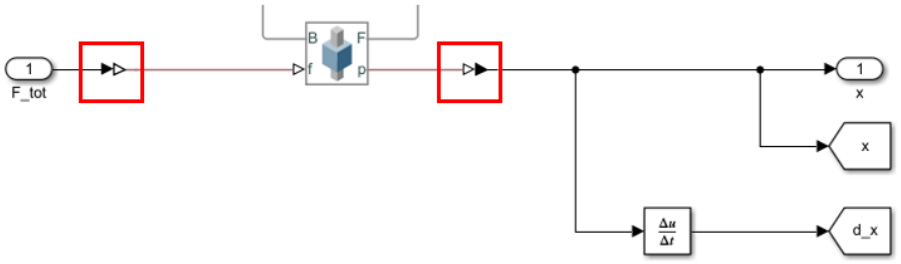


Figura 9-10: conversione Simulink-Simscape

Nel giunto rotoidale, che collega il “perno” con “l’attacco dell’asta”, è stato installato un sensore di posizione angolare, come in laboratorio viene quindi spuntato tra i vari sensori quello di posizione, attivando la porta “q”. La posizione angolare viene restituita in radianti ma con verso opposto a quello considerato dal sensore di laboratorio, per questo viene posta una inversione di segno.

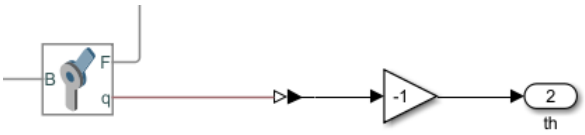


Figura 9-11: giunto rotoidale

Si può osservare che tra il sistema di riferimento di “perno” e “attacco dell’asta” è necessario applicare una rotazione con la cerniera , per simulare il giunto, che prevede rotazioni attorno all’asse z, e una traslazione (mantenute separate per essere sicuri del corretto ordine di applicazione) in quanto la faccia di accoppiamento del perno con l’asta non veniva riconosciuta correttamente.

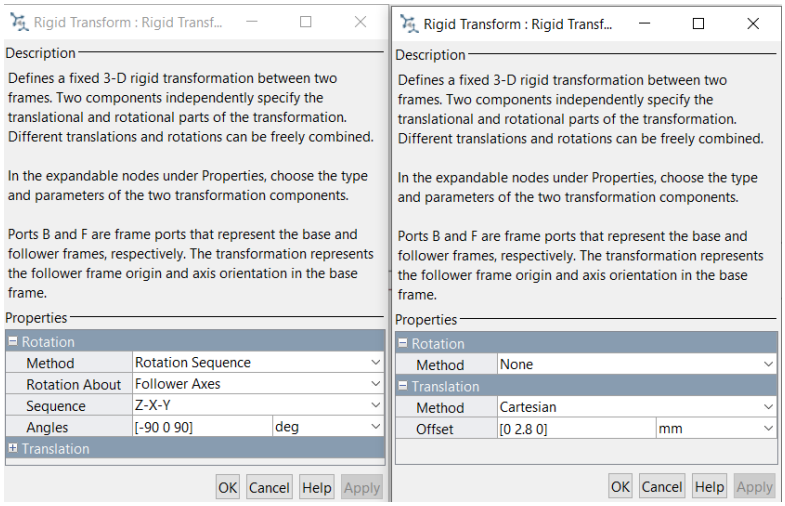


Figura 9-12: rotazione e traslazione

Per una migliore comprensione del sistema in appendice vengono riportati i “File Solid” delle restanti parti, per poter visualizzare i collegamenti tra i vari Frame.

Una volta salvato il file, tramite la schermata “Mechanics Explorer”, è possibile visualizzare il meccanismo completo. Per non ottenere errori di visualizzazione è importante, se si spostano i file “.step” dei componenti, aggiornare la nuova directory nei “File Solid” di Simscape.

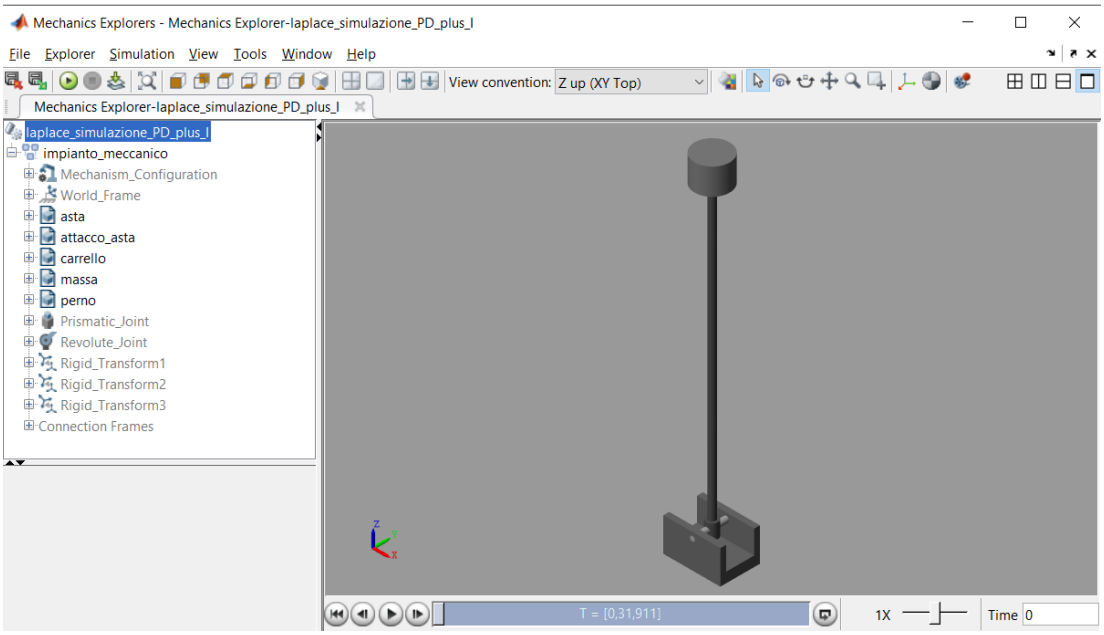


Figura 9-13: Mechanics Explorer

Implementato il modello nel file Simulink il controllore PD+I funzionava, ma non in modo ottimale, è stato quindi necessario effettuare nuovamente un leggero tuning dei parametri di partenza sul controllo della posizione x. Vengono rappresentati e commentati i precedenti valori.

```
k_prop=9.5; %k_prop=10;
k_der=2; %k_der=1.33;
k_int=1.5; %k_int=1;
x_max=0.25;
GE_x=100/x_max;
GU_x=(Kp_x/GE_x)*k_prop;
GCE_x=(Kd_x/GU_x)*k_der;
GIU_x=(Ki_x/GU_x)*k_int;
```

Nelle seguenti figure viene mostrato un confronto, a parità di set imposti, tra il precedente modello in solo Simulink e l'attuale modello in Simulink e Simscape-Multibody. Per il confronto vengono usati: due set a gradino di ampiezza 0.1 e 0.05 m, un set sinusoidale di ampiezza 0.05 m e frequenza 0.05 Hz, e un disturbo esterno di 5 N sul set a gradino di 0.05m a 25 s.

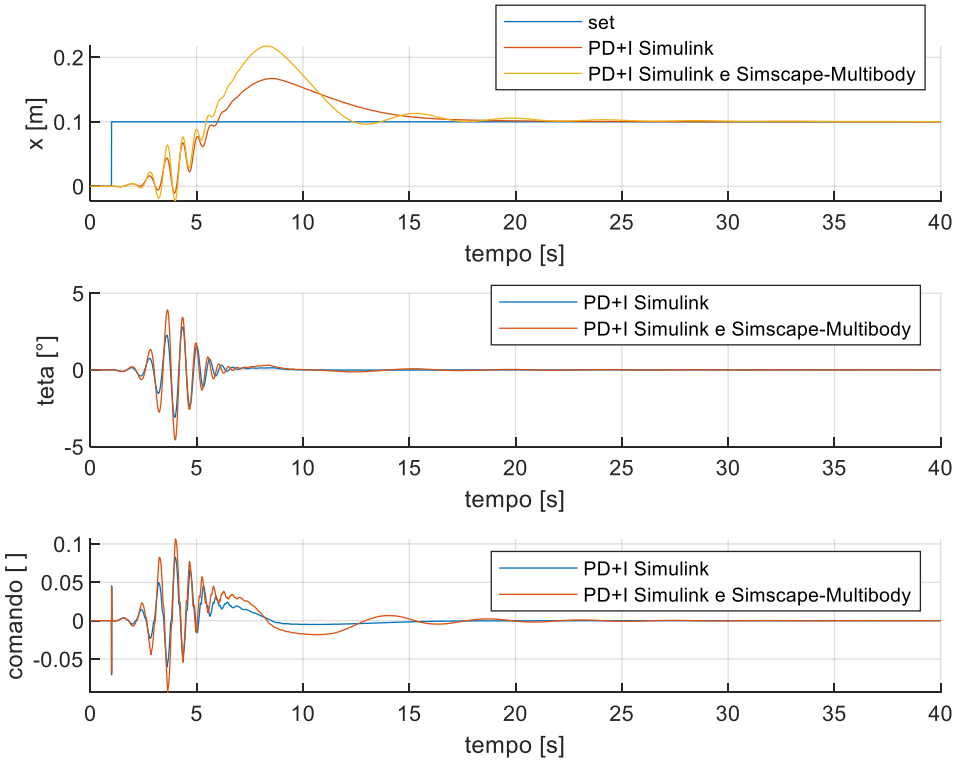


Figura 9-15: gradino 0.1m

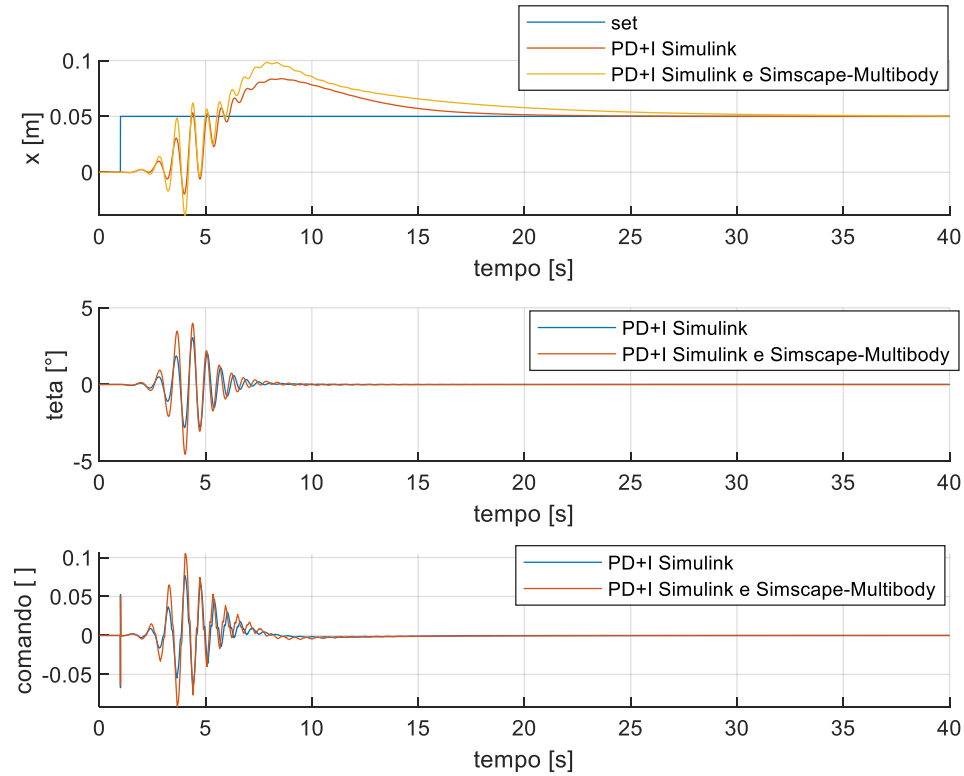


Figura 9-14: gradino 0.05m

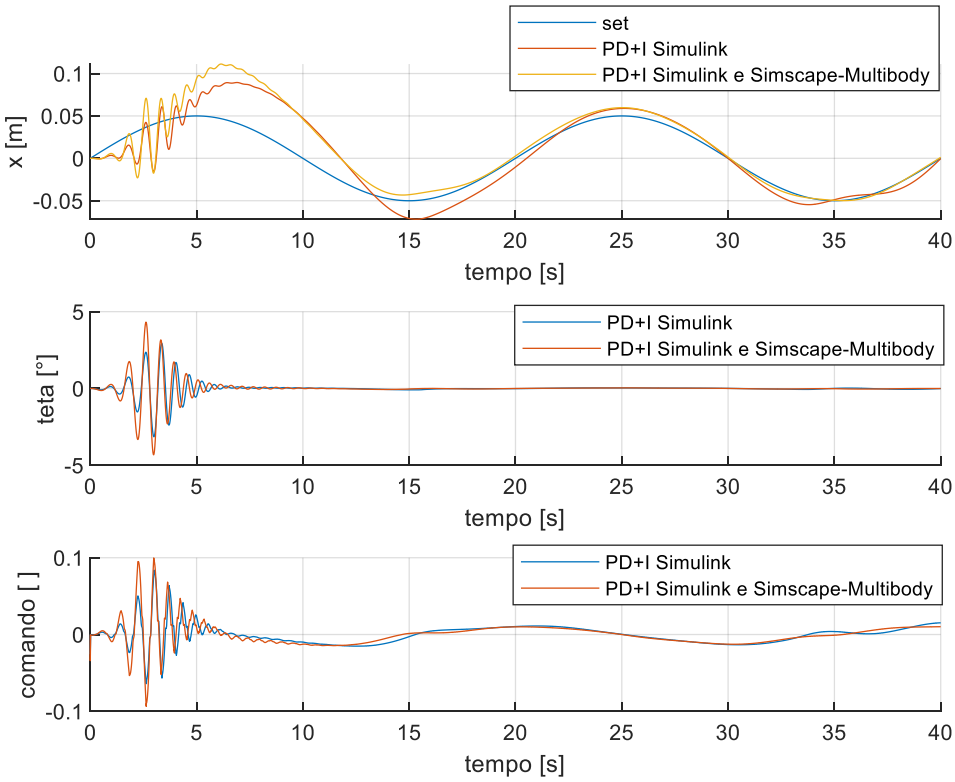


Figura 9-17: sinusoide

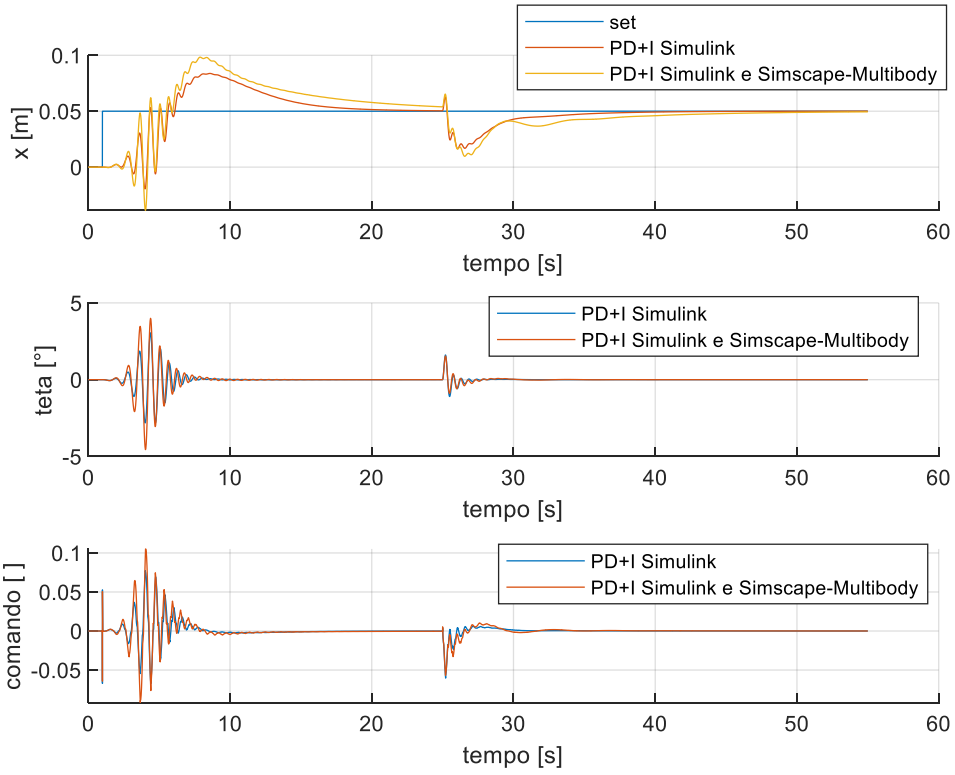


Figura 9-16: disturbo

Si può osservare come il modello Simscape-Multibody presenti una difficoltà maggiore a seguire il set. Questo risultato era atteso, in quanto considerando l'inerzia dei corpi si ottiene una risposta peggiore del sistema a parità di funzione di set imposta.
Tra i due modelli non si evidenziano grosse differenze di riposta ai set imposti esternamente, questo è indice che il modello in solo Simulink dell'equilibrio meccanico non si discosta molto dal modello più completo realizzato in Simscape-Multibody e Simulink, infatti non è stato necessario variare di molto i valori dei guadagni del controllo.

Di seguito vengono rappresentati alcuni frame del video che è possibile salvare dalla simulazione del gradino di ampiezza 0.1 m.

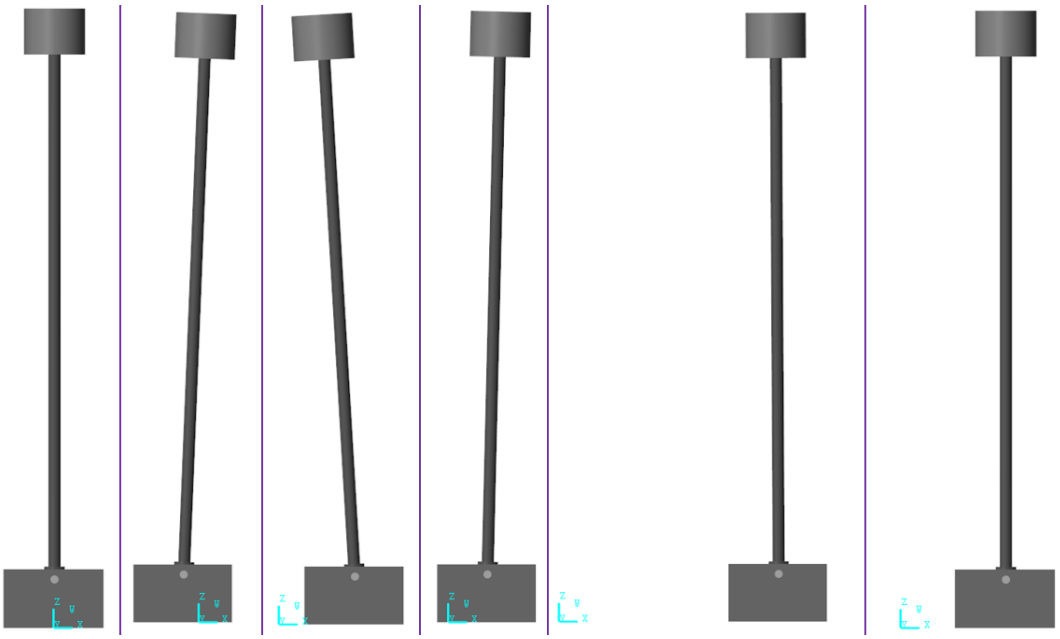


Figura 9-18: frame video

In figura è possibile osservare il meccanismo secondo il quale il pendolo per spostarsi a destra, prima si sposta a sinistra facendo inclinare il pendolo verso destra, consentendogli quindi di spostarsi verso destra senza perdere il controllo sulla posizione angolare teta. La posizione di zero è rappresentata del sistema di riferimento evidenziato in azzurro, il world frame. Le oscillazioni non sono molto marcate, infatti il massimo è inferiore ai 5°.

Viene infine proposta una considerazione sull'inerzia introdotta dell'asta anche a livello del solo modello Simulink. L'inerzia di un'asta sottile rispetto al baricentro può essere espressa come:

$$I_{cm} = \frac{1}{12} m l^2$$

dove ricordiamo l=500mm è la lunghezza dell'asta, mentre m=0.152kg la massa. Secondo il teorema di Huygens Steiner:

$$I = I_{cm} + m\,(d)^2 = I_{cm} + m\left(\frac{l}{2}\right)^2 = \frac{1}{3} m l^2$$

Per una migliore interpretazione viene riportata un'immagine ripresa da [11].

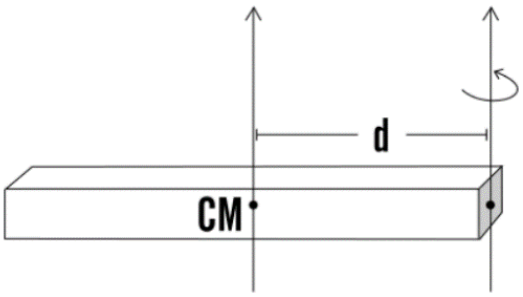


Figura 9-20: teorema Huygens Steiner

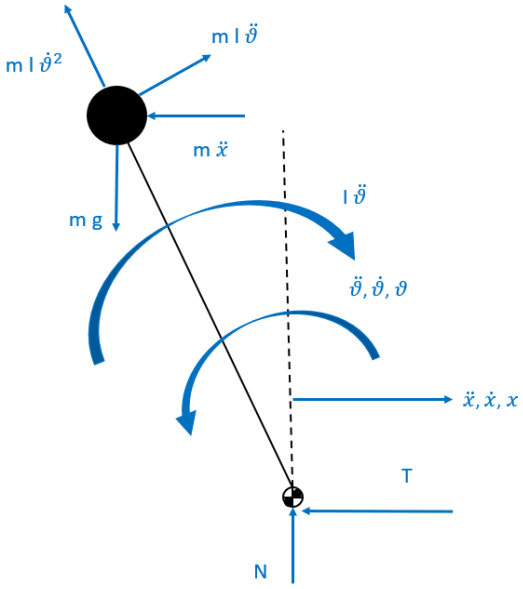


Figura 9-19: eq. pendolo

Viene pertanto aggiornato l'equilibrio alla rotazione del pendolo sulla cerniera introducendo l'inerzia dell'asta:

$$m\,g\ \sin\vartheta\ l\ +\ m\,\ddot{x}\ \cos\vartheta\ l\ -\ m\,l\,\ddot{\vartheta}\,l\ -I\,\ddot{\vartheta}\ =0$$

La trattazione in solo Simulink può considerarsi così aggiornata.

10 Conclusioni

L'idea principale del lavoro svolto è stata la realizzazione di un controllore fuzzy per stabilizzare un pendolo inverso pneumatico. Si è sfruttato un doppio controllo, uno sulla posizione x del carrello e uno sull'angolo teta del pendolo.

Tutti e tre i controllori fuzzy sviluppati in Simulink sono riusciti a stabilizzare il pendolo inverso, con una maggiore efficacia del PD+I, che presenta un taglio a frequenze maggiori con funzioni di set sinusoidali e una maggiore capacità di reagire ai disturbi.

La simulazione in Proteus del solo controllo fuzzy ha dato risultanti soddisfacenti, sovrapponibili a quelli del fuzzy tool di Matlab, evidenziando come la libreria fuzzy di Arduino lavori correttamente. Per una maggiore efficacia nel trasmettere il funzionamento del controllo si è anche scelto di simulare gli input utilizzando dei potenziometri per farli variare manualmente durante l'esecuzione della simulazione.

I risultati ottenuti dalla successiva implementazione del controllo fuzzy all'interno della scheda Arduino fisica con software Arduino IDE hanno dato i medesimi risultati del simulatore Proteus mostrandone l'efficacia.

Tramite MATLAB Support Package for Arduino Hardware e Simulink Support Package for Arduino Hardware si è proceduto ad implementare i controllori Arduino sfruttando come ambiente di programmazione non più Arduino IDE, che presenta un linguaggio di programmazione di basso livello (C++), ma tramite Simulink, sfruttando quindi la libreria aggiuntiva per interfacciarsi con i vari pin e funzioni di Arduino.

Tra i vantaggi di questo tipo di programmazione tramite Simulink Support Package for Arduino Hardware ci sono:

- La possibilità di programmare sfruttando funzioni come integrali, derivate, bande morte (ed altre) che richiederebbero molte righe di codice e approssimazioni maggiori per essere implementate in C++, oltre che una profonda conoscenza del linguaggio di programmazione.
- La possibilità di un controllo in real time delle variabili.
- Un post processing più veloce, senza la necessita di copiare stringhe dalla Command Window all'interno di file di testo per poi esportarle in Matlab per un'analisi.
- La possibilità di avere due tipi differenti di esecuzione: il primo tramite il comando Run selezionando la modalità External che consente il controllo in real time e il post processing più efficacie, e la seconda che prevede di caricare il programma direttamente sulla scheda tramite il comando Deploy To Hardware, consentendo di sconnettere la scheda dal pc (continuando comunque a fornirgli potenza tramite un alimentatore) consentendo una volta consolidato il programma di posizionare la scheda sull'attrezzatura di prova e farla funzionare autonomamente.

Infine, modellando le equazioni di equilibrio meccanico del pendono inverso con Simscape-Multibody è stato possibile tenere conto di alcuni parametri prima trascurati (come l'inerzia del pendolo). La simulazione, in seguito ad una lieve ricalibrazione dei parametri del controllore, ha dato risultati paragonabili a quelli del precedente modello, mostrando come le

ipotesi fatte fossero plausibili. È stato possibile anche ricavare dei video della simulazione del pendolo con il Mechanics Explorer

Tra gli sviluppi futuri prevedo la possibilità di verificare il corretto funzionamento dei programmi proposti in tesi nei vari linguaggi di programmazione usati, per poi eseguire un confronto e vedere quale codice riesca effettivamente a stabilizzare meglio il pendolo e anche quale dei tre controlli sviluppati reagisca meglio. Tra le varie proposte potrebbe anche esserci la possibilità di realizzare un Hardware il the Loop, caricando il programma del controllore nella scheda Arduino e simulando il banco in Simulink, in modo che la comunicazione seriale tra i due sistemi possa avvenire tramite USB. Infine, potrebbe essere possibile modellare anche l'impianto pneumatico con Simscape, riducendo così il grado di approssimazione del sistema.

11 Appendice

11.1 Codice Arduino verifica libreria fuzzy

```
// controllo libreria fc_1
#include <Fuzzy.h>

// fc_1: è la variabile fc_1
Fuzzy *fc_1 = new Fuzzy();

// variabili di errore: definite a seconda del caso analizzato
// caso1: var_e
float e = -100; // e : errore
float de = 0; // d : derivata

// caso2: var_de
//float e = 0; // e : errore
//float de = -100; // d : derivata

// caso3: var_e_de
//float e = -100; // e : errore
//float de = -100; // d : derivata

// variabile di uscita
float u = 0;

// tempo di campionamento [s]
//float Ts = 0.05;
float Ts = 0.01;

float t_in=0.00; // tempo inizio simulazione [s]
float t_fin=10.00; // tempo fine simulazione [s]

// in e out funzione fc_1
float err;
float d_err;
float out_fc;

// variabile temporale
float t=t_in;

void setup() {

  // Set the Serial output
  Serial.begin(9600);

  // richiamo la funzione che crea le regole
  Set_LogicaFuzzy();

  // creo delle colonne per fare poi un acquisizione dati
  Serial.print("u          ");
  Serial.print("e        ");
  Serial.print("de      ");
  Serial.println("t");
}

void loop() {
 if(t>=t_in & t<=t_fin){

    // richiamo funzioni fc_1
```

```
  fc_1->setInput(1,e);
  fc_1->setInput(2,de);
  fc_1->fuzzify();
  u=fc_1->defuzzify(1);

  // valori a video per acquisizione dati
  Serial.print(u); // println: manda in automatico a capo
  Serial.print("    "); // per dare uno spazio nelle acquisizioni
  Serial.print(e);
  Serial.print("    ");
  Serial.print(de);
  Serial.print("    ");
  Serial.println(t);

  // tempo di attesa per rilanciare il loop
  // attenzione che in delay il tempo va in ms (1 s = 1000 ms)
  delay(Ts * 1000);
  t=t+Ts; // aggiornamento della varibile temporale (per tracciare poi i grafici)
  e=e+(200/( (t_fin-t_in)/Ts) ); // incremento proporzionale col tempo
  //de=de+(200/( (t_fin-t_in)/Ts) );
 }
}

// fc_1 set 1
void  Set_LogicaFuzzy () {

 // set FuzzyInput
 FuzzySet *N_e = new FuzzySet(-100, -100, -100, 0);
 FuzzySet *Z_e = new FuzzySet(-100, 0, 0, 100);
 FuzzySet *P_e = new FuzzySet(0, 100, 100, 100);

 FuzzySet *N_de = new FuzzySet(-100, -100, -100, 0);
 FuzzySet *Z_de = new FuzzySet(-100, 0, 0, 100);
 FuzzySet *P_de = new FuzzySet(0, 100, 100, 100);

 // set FuzzyOutput
 FuzzySet *NN_out = new FuzzySet(-100, -100, -100, -50);
 FuzzySet *N_out = new FuzzySet(-100, -50, -50, 0);
 FuzzySet *Z_out = new FuzzySet(-50, 0, 0, 50);
 FuzzySet *P_out = new FuzzySet(0, 50, 50, 100);
 FuzzySet *PP_out = new FuzzySet(50, 100, 100, 100);

 // FuzzySetInput: 1) errore 2) derivata dell'errore
 FuzzyInput *err = new FuzzyInput(1);
 err->addFuzzySet(N_e);
 err->addFuzzySet(Z_e);
 err->addFuzzySet(P_e);
 fc_1->addFuzzyInput(err);

 FuzzyInput *d_err = new FuzzyInput(2);
 d_err->addFuzzySet(N_de);
 d_err->addFuzzySet(Z_de);
 d_err->addFuzzySet(P_de);
 fc_1->addFuzzyInput(d_err);

 // FuzzySetOutput: 1) out_fc
 FuzzyOutput *out_fc = new FuzzyOutput(1);
 out_fc->addFuzzySet(NN_out);
 out_fc->addFuzzySet(N_out);
 out_fc->addFuzzySet(Z_out);
 out_fc->addFuzzySet(P_out);
 out_fc->addFuzzySet(PP_out);
```

```
fc_1->addFuzzyOutput(out_fc);

// regole fc_1

// se err=N e d_err=N ==> out=NN
FuzzyRuleAntecedent *if_err_N_AND_d_err_N = new FuzzyRuleAntecedent();
if_err_N_AND_d_err_N->joinWithAND(N_e, N_de);
FuzzyRuleConsequent *then_out_NN1 = new FuzzyRuleConsequent();
then_out_NN1->addOutput(NN_out);
FuzzyRule *fc_1Rule1 = new FuzzyRule(1, if_err_N_AND_d_err_N, then_out_NN1); // cambiare il nome alla regola e incrementare l'indice
fc_1->addFuzzyRule(fc_1Rule1);

// se err=N e d_err=Z ==> out=N
FuzzyRuleAntecedent *if_err_N_AND_d_err_Z = new FuzzyRuleAntecedent();
if_err_N_AND_d_err_Z->joinWithAND(N_e, Z_de);
FuzzyRuleConsequent *then_out_N1 = new FuzzyRuleConsequent();
then_out_N1->addOutput(N_out);
FuzzyRule *fc_1Rule2 = new FuzzyRule(2, if_err_N_AND_d_err_Z, then_out_N1);
fc_1->addFuzzyRule(fc_1Rule2);

// se err=N e d_err=P ==> out=N
FuzzyRuleAntecedent *if_err_N_AND_d_err_P = new FuzzyRuleAntecedent();
if_err_N_AND_d_err_P->joinWithAND(N_e, P_de);
FuzzyRuleConsequent *then_out_N2 = new FuzzyRuleConsequent(); // then_u_N1: metto 1 per non ridefinire la stessa variabile
then_out_N2->addOutput(N_out);
FuzzyRule *fc_1Rule3 = new FuzzyRule(3, if_err_N_AND_d_err_P, then_out_N2);
fc_1->addFuzzyRule(fc_1Rule3);

// se err=Z e d_err=N ==> out=N
FuzzyRuleAntecedent *if_err_Z_AND_d_err_N = new FuzzyRuleAntecedent();
if_err_Z_AND_d_err_N->joinWithAND(Z_e, N_de);
FuzzyRuleConsequent *then_out_N3 = new FuzzyRuleConsequent();
then_out_N3->addOutput(N_out);
FuzzyRule *fc_1Rule4 = new FuzzyRule(4, if_err_Z_AND_d_err_N, then_out_N3);
fc_1->addFuzzyRule(fc_1Rule4);

// se err=Z e d_err=Z ==> out=Z
FuzzyRuleAntecedent *if_err_Z_AND_d_err_Z = new FuzzyRuleAntecedent();
if_err_Z_AND_d_err_Z->joinWithAND(Z_e, Z_de);
FuzzyRuleConsequent *then_out_Z1 = new FuzzyRuleConsequent();
then_out_Z1->addOutput(Z_out);
FuzzyRule *fc_1Rule5 = new FuzzyRule(5, if_err_Z_AND_d_err_Z, then_out_Z1);
fc_1->addFuzzyRule(fc_1Rule5);

// se err=Z e d_err=P ==> out=P
FuzzyRuleAntecedent *if_err_Z_AND_d_err_P = new FuzzyRuleAntecedent();
if_err_Z_AND_d_err_P->joinWithAND(Z_e, P_de);
FuzzyRuleConsequent *then_out_P1 = new FuzzyRuleConsequent();
then_out_P1->addOutput(P_out);
FuzzyRule *fc_1Rule6 = new FuzzyRule(6, if_err_Z_AND_d_err_P, then_out_P1);
fc_1->addFuzzyRule(fc_1Rule6);

// se err=P e d_err=N ==> out=P
FuzzyRuleAntecedent *if_err_P_AND_d_err_N = new FuzzyRuleAntecedent();
if_err_P_AND_d_err_N->joinWithAND(P_e, N_de);
FuzzyRuleConsequent *then_out_P2 = new FuzzyRuleConsequent();
then_out_P2->addOutput(P_out);
FuzzyRule *fc_1Rule7 = new FuzzyRule(7, if_err_P_AND_d_err_N, then_out_P2);
fc_1->addFuzzyRule(fc_1Rule7);

// se err=P e d_err=Z ==> out=P
```

```
  FuzzyRuleAntecedent *if_err_P_AND_d_err_Z = new FuzzyRuleAntecedent();
  if_err_P_AND_d_err_Z->joinWithAND(P_e, Z_de);
  FuzzyRuleConsequent *then_out_P3 = new FuzzyRuleConsequent();
  then_out_P3->addOutput(P_out);
  FuzzyRule *fc_1Rule8 = new FuzzyRule(8, if_err_P_AND_d_err_Z, then_out_P3);
  fc_1->addFuzzyRule(fc_1Rule8);

  // se err=P e d_err=Z ==> out=PP
  FuzzyRuleAntecedent *if_err_P_AND_d_err_P = new FuzzyRuleAntecedent();
  if_err_P_AND_d_err_P->joinWithAND(P_e, P_de);
  FuzzyRuleConsequent *then_out_PP1 = new FuzzyRuleConsequent();
  then_out_PP1->addOutput(PP_out);
  FuzzyRule *fc_1Rule9 = new FuzzyRule(9, if_err_P_AND_d_err_P, then_out_PP1);
  fc_1->addFuzzyRule(fc_1Rule9);

}
```

11.2 Codice Arduino per agire manualmente sugli input

Non viene ricopiato il void Set_LogicaFuzzy () in quanto identico al caso precedente.

```
#include <Fuzzy.h>

// fc_1: è la variabile fc_1
Fuzzy *fc_1 = new Fuzzy();

// assegnazione dei PIN
const int e_pin = A0;
const int de_pin = A1;
const int valv_12_pin = 3; // comanda valvola 1 e 2
const int valv_34_pin = 5; // comanda valvola 3 e 4

// variabili di errore
float e = 0; // e : errore
float de = 0; // d : derivata

// variabile di uscita
float u ;
float u_scal;

// tempo di campionamento [s]
//float Ts = (float)0.01;
float Ts = (float)2;

// in e out funzione fc_1
float err;
float d_err;
float out_fc;

// variabile temporale
float t=0;

void setup() {

  // Set the Serial output
  Serial.begin(9600);

  // PIN lettura input
  pinMode(e_pin, INPUT);
```

```
 pinMode(de_pin, INPUT);

 // PIN analogici (PWM) sui quali scriviamo i valori (OUTPUT)
 pinMode(valv_12_pin, OUTPUT); // PIN analogico può assumere valori da 0 a 255
 pinMode(valv_34_pin, OUTPUT);
 // come valore iniziale impostiamo lo zero (nessun comando)
 analogWrite(valv_12_pin, 0);
 analogWrite(valv_34_pin, 0);

 // richiamo la funzione che crea le regole
 Set_LogicaFuzzy();

 // creo delle colonne per fare poi un'acquisizione dati
 Serial.print("u      ");
 Serial.print("e       ");
 Serial.print("de       ");
 Serial.println("t");

}

void loop() {

 e= ( (float)analogRead(e_pin) *200 / 1023 - 100 );
 de=( (float)analogRead(de_pin) * 200 / 1023.0 - 100 );

 // richiamo funzioni fc_1
 fc_1->setInput(1,e);
 fc_1->setInput(2,de);
 fc_1->fuzzify();
 u=fc_1->defuzzify(1);

 u_scal=(float)u/100;
 // invio comandi alle valvole
 if (u_scal == 0) {
  analogWrite(valv_12_pin, 0);
  analogWrite(valv_34_pin, 0);
 }
 if (u_scal > 0) {
  analogWrite(valv_12_pin, u_scal * 255);
  analogWrite(valv_34_pin, 0);
 }
 if (u_scal < 0) {
  analogWrite(valv_12_pin, 0);
  analogWrite(valv_34_pin, abs(u_scal) * 255);
 }

 // valori a video per acquisizione dati
 Serial.print(u); // println: manda in automatico a capo
 Serial.print("    "); // per dare uno spazio nelle acquisizioni
 Serial.print(e);
 Serial.print("    ");
 Serial.print(de);
 Serial.print("    ");
 Serial.println(t);

 // tempo di attesa per rilanciare il loop
 // attenzione che in delay il tempo va in ms (1 s = 1000 ms)
 delay(Ts * 1000);
 t=t+Ts; // aggiornamento della varibile temporale (per tracciare poi i grafici)
}
```

11.3 codice Matlab per simulazione PD+I

```
clear;
close all;
clc;

%% inizializzazione variabili
t_in=0; % tempo inizio simulazione [s]
t_fin=40; % tempo fine simulazione [s]
Ts=0.001; % tempo di campionamento [s] , per controllori
Tsim=0.001; % fixed step size [s] , della simulazione
dh_zone=10^(-10); % dead zone (per evitare errori numerici)

m=0.2; % massa pendolo [kg]
mc=1.8; % massa carrello [kg]
mp=0.4; % massa pistone [kg]
g=9.81; % accelerazione di gravità
b=20.83; % attrito viscoso nel carrello [Ns/m]
l=0.5; % lunghezza asta pendolo [m]
R=287.05; % costante dei gas [J/kgK]
T=293; % temperatura in camera cilindro [K]
T_N=293; % temperatura normale [K]
c=500*10^(-3); % corsa del cilindro [m]
% A: camera posteriore
% B: camera anteriore
V_A0=1*10^(-6); % volume morto camera A [m^3]
V_B0=1*10^(-6); % volume morto camera B [m^3]
D=16*10^(-3); % diametro pistone [m]
d=6*10^(-3); % diametro asta [m]
A_A=pi*D^2/4; % area di spinta camera A [m^2]
A_s=pi*d^2/4; % area stelo[m^2]
A_B=A_A-A_s; % area di spinta camera B [m^2]
p_amb=1*10^5; % pressione relativa ambiente [Pa]
p_supply=6*10^5; % pressione relativa di alimentazione [Pa]
bv=0.3; % rapporto critico valvola
Cv=2.4*10^(-9); % conduttanza della valvola [m^3(ANR)/sPa]
rho=1.225; % densità aria nella valvola [kg/m^3]

%creazione di un set sinusoidale
tt_in=[t_in : Tsim : t_fin];
x_set_sin(:,1)=tt_in;
ampiezza=0.05;
frequenza=0.05;
x_set_sin(:,2)=ampiezza*sin(2*pi*frequenza*tt_in);

% carico i workspace le regole fuzzy
fc=readfis('fc_1');
```

```
% guadagni PID
Kp_x=-0.000802;
Ki_x=-4.68e-6;
Kd_x=-0.0306;
Tf_x=0.00875;

Kp_th=2.09;
Ki_th=6.31;
Kd_th=0.103;
Tf_th=0.000875;

% guadagni Fuzzy
k_prop=10;
k_der=1.33;
k_int=1;
x_max=0.25;
GE_x=100/x_max;
GU_x=(Kp_x/GE_x)*k_prop;
GCE_x=(Kd_x/GU_x)*k_der;
GIU_x=(Ki_x/GU_x)*k_int;

th_max=pi*20/180;
GE_th=100/th_max;
GU_th=Kp_th/GE_th;
GCE_th=Kd_th/GU_th;
GIU_th=Ki_th/GU_th;

%% lancio simulazione
opz = simset('SrcWorkspace','current','DstWorkspace','current');
sim('laplace_simulazione_PD_plus_I.slx',[],opz);

%% analisi grafici
cont=1;
close all;

% variabili cinematiche e comando valvola
cont=cont+1;
figure(cont)
subplot(3,1,1)
hold on
plot(tt,x_set),grid on
plot(tt,x),grid on
xlabel('tempo [s]'),ylabel('x [m]')
legend('set','fb')
subplot(3,1,2)
plot(tt,th),grid on
xlabel('tempo [s]'),ylabel('teta [°]')
subplot(3,1,3)
```

```
plot(tt,u),grid on
xlabel('tempo [s]'),ylabel('comando [ ]')

% flussi in massa
cont=cont+1;
figure(cont)
subplot(4,1,1)
plot(tt,d_m_inA,'k','Linewidth',1),grid on
ylim([-1.5*max(abs(d_m_inA+10^(-20))), 1.5*max(abs(d_m_inA+10^(-20)))])
xlabel('tempo [s]'),ylabel('d m inA [kg/s]')
subplot(4,1,2)
plot(tt,d_m_outA,'k','Linewidth',1),grid on
ylim([-1.5*max(abs(d_m_outA+10^(-20))), 1.5*max(abs(d_m_outA+10^(-20)))])
xlabel('tempo [s]'),ylabel('d m outA [kg/s]')
subplot(4,1,3)
plot(tt,d_m_inB,'k','Linewidth',1),grid on
ylim([-1.5*max(abs(d_m_inB+10^(-20))), 1.5*max(abs(d_m_inB+10^(-20)))])
xlabel('tempo [s]'),ylabel('d m inB [kg/s]')
subplot(4,1,4)
plot(tt,d_m_outB,'k','Linewidth',1),grid on
ylim([-1.5*max(abs(d_m_outB+10^(-20))), 1.5*max(abs(d_m_outB+10^(-20)))])
xlabel('tempo [s]'),ylabel('d m outB [kg/s]')

% andamento pressioni e forze
cont=cont+1;
figure(cont)
subplot(4,1,1)
plot(tt,P_A,'k','Linewidth',1),grid on
xlabel('tempo [s]'),ylabel('pA [Pa]')
subplot(4,1,2)
plot(tt,P_B,'k','Linewidth',1),grid on
xlabel('tempo [s]'),ylabel('pB [Pa]')
subplot(4,1,3)
plot(tt,F_ReazioneCamera,'k','Linewidth',1),grid on
xlabel('tempo [s]'),ylabel('F camera [N]')
subplot(4,1,4)
plot(tt,F,'k','Linewidth',1),grid on
xlabel('tempo [s]'),ylabel('F [N]')

%% salvataggio valori

%save(['sim_save',filesep,'PD_plus_I_gradino1.mat'],'x','th','u','tt','x_set');
%save(['sim_save',filesep,'PD_plus_I_gradino2.mat'],'x','th','u','tt','x_set');
```

11.4 codice Matlab per simulazione PD+PI

rispetto al listato precedente vengono solo riportate le variazioni.

```
% guadagni Fuzzy
k_prop=8;
k_der=1.66;
k_int=1;
x_max=0.25;
GE_x=100/x_max;
GU_x=Kp_x/(2*GE_x)*k_prop;
GCE_x=(Kd_x/GU_x)*k_der;
GIU_x=(Ki_x/GU_x)*k_int;

th_max=pi*20/180;
GE_th=100/th_max;
GU_th=Kp_th/(2*GE_th);
GCE_th=Kd_th/GU_th;
GIU_th=Ki_th/GU_th;

%% lancio simulazione
opz = simset('SrcWorkspace','current','DstWorkspace','current');
sim('laplace_simulazione_PD_plus_PI.slx',[],opz);

%% salvataggio valori

%save(['sim_save',filesep,'PD_plus_PI_gradino1.mat'],'x','th','u','tt','x_set');
%save(['sim_save',filesep,'PD_plus_PI_gradino2.mat'],'x','th','u','tt','x_set');
```

11.5 codice Matlab per simulazione PID-like

rispetto al listato precedente vengono solo riportate le variazioni.

```
% guadagni Fuzzy
k_prop=25;
k_der=1.66;
k_int=1/5;
x_max=0.25;
GE_x=100/x_max;
GU_x=k_prop*( -(-Kp_x*GE_x) - sqrt( (Kp_x*GE_x)^2-4*GE_x^2*Kd_x*Ki_x ) )/ ( 2*GE_x^2 ) ;
GCE_x=k_der*( Kd_x/GU_x );
GIU_x=k_int*( Ki_x/( GU_x*GE_x ) );

th_max=pi*20/180;
GE_th=100/th_max;
GU_th= ( -(-Kp_th*GE_th) + sqrt( (Kp_th*GE_th)^2-4*GE_th^2*Kd_th*Ki_th ) )/ ( 2*GE_th^2 ) ;
GCE_th=Kd_th/GU_th;
```

```
GIU_th=Ki_th/( GU_th*GE_th );

%% lancio simulazione
opz = simset('SrcWorkspace','current','DstWorkspace','current');
sim('laplace_simulazione_PID_like.slx',[],opz);

%% salvataggio valori
%save(['sim_save',filesep,'PID_like_gradino1.mat'],'x','th','u','tt','x_set');
%save(['sim_save',filesep,'PID_like_gradino2.mat'],'x','th','u','tt','x_set');
```

11.6 codice Matlab per confronto controllori

il medesimo codice è applicato anche nel caso del gradino 2 variano i file caricati.

```
%% acquisizioni dati da simulazioni
cont=0;

load(['sim_save',filesep,'PD_plus_I_gradino1.mat']);
% controllo: PD+I con fc 1
cont=cont+1;
% salvo valori nelle matrici
uu(:,cont)=u;
xx(:,cont)=x;
tth(:,cont)=th;
t(:,cont)=tt;
legenda(cont,1)="PD+I";
legenda1(cont,1)="set";
legenda1(cont+1,1)="PD+I";

load(['sim_save',filesep,'PID_like_gradino1.mat']);
% controllo: PID like con fc 1
cont=cont+1;
% salvo valori nelle matrici
uu(:,cont)=u;
xx(:,cont)=x;
tth(:,cont)=th;
t(:,cont)=tt;
legenda(cont,1)="PID-like";
legenda1(cont+1,1)="PID-like";

load(['sim_save',filesep,'PD_plus_PI_gradino1.mat']);
% controllo: PD+PI con fc 1
cont=cont+1;
% salvo valori nelle matrici
uu(:,cont)=u;
xx(:,cont)=x;
tth(:,cont)=th;
```

```
t(:,cont)=tt;
legenda(cont,1)="PD+PI";
legenda1(cont+1,1)="PD+PI";

%% analisi grafici
figure(1)
subplot(3,1,1)
hold on
plot(tt,x_set),grid on
xlabel('tempo [s]'),ylabel('x [m]')

for i=1:cont
   u=uu(:,i);
   x=xx(:,i);
   th=tth(:,i);
   tt=t(:,i);
   % analisi grafici
   figure(1)
   subplot(3,1,1)
   hold on
   plot(tt,x),grid on
   xlabel('tempo [s]'),ylabel('x [m]')
   legend(legenda1)
   subplot(3,1,2)
   hold on
   plot(tt,th),grid on
   xlabel('tempo [s]'),ylabel('teta [°]')
   legend(legenda)
   subplot(3,1,3)
   hold on
   plot(tt,u),grid on
   xlabel('tempo [s]'),ylabel('comando [ ]')
   legend(legenda)
end
```

11.7 codice Arduino PD+I

la funzione Set_LogicaFuzzy() è già definita in un codice soprastante e non viene di conseguenza riportata.

```
#include <Fuzzy.h>

// fc_1: è la variabile fc_1
Fuzzy *fc_1 = new Fuzzy();

// definisco pigreco
const double pi = 3.14159265358979;
```

```
// assegnazione dei PIN
const int x_pin = A8;
const int th_pin = A2;
const int valv_12_pin = 4; // comanda valvola 1 e 2
const int valv_34_pin = 13; // comanda valvola 3 e 4

// condizioni iniziali variabili cinematiche
double x = 0;
double th = 0;

// variabili di errore
double e_x = 0; // e : errore
double e_th = 0;
double d_e_x = 0; // d : derivata
double d_e_th = 0;
double i_e_x = 0; // i : integrale
double i_e_th = 0;
// errori cicli precedenti
double e_x_prec = 0;
double e_th_prec = 0;

// variabili di set
double x_set = 0; // variabile da modificare
double th_set = 0;

// variabile di comando valvole
double u = 0;

// tempo di campionamento
double Ts = 0.005;

// costanti geometriche
double corsa = 0.5;

// guadagni PID
double Kp_th = 2.09;
double Ki_th = 6.31;
double Kd_th = 0.103;

double Kp_x = -0.000802;
double Ki_x = -4.68e-6;
double Kd_x = -0.0306;

// guadagni Fuzzy
double k_prop=10;
double k_der=1.33;
double k_int=1;
double x_max=0.25;
double GE_x=100/x_max;
```

```
double GU_x=(Kp_x/GE_x)*k_prop;
double GCE_x=(Kd_x/GU_x)*k_der;
double GIU_x=(Ki_x/GU_x)*k_int;


double th_max = pi * 20 / 180;
double GE_th = 100 / th_max;
double GU_th = Kp_th / GE_th;
double GCE_th = Kd_th / GU_th;
double GIU_th = Ki_th / GU_th;

// output funzioni fc_1
double out1=0;
double out2=0;

// in e out funzione fc_1
double err;
double d_err;
double out_fc;

// variabile temporale
double t = 0;

void setup() {

 // Set the Serial output
 Serial.begin(9600);

 // PIN lettura input
 pinMode(x_pin, INPUT);
 pinMode(th_pin, INPUT);

 // PIN analogici (PWM) sui quali scriviamo i valori (OUTPUT)
 pinMode(valv_12_pin, OUTPUT); // PIN analogico può assumere valori da 0 a 255
 pinMode(valv_34_pin, OUTPUT);
 // come valore iniziale impostiamo lo zero (nessun comando)
 analogWrite(valv_12_pin, 0);
 analogWrite(valv_34_pin, 0);

 // richiamo la funzione che crea le regole
 Set_LogicaFuzzy();

 // creo delle colonne per fare poi un'acquisizione dati
 Serial.print("u ");
 Serial.print("x ");
 Serial.print("th ");
 Serial.print("t ");

}
```

```
void loop() {

 // valori a video per acquisizione dati
 Serial.println(u); // println: manda in automatico a capo
 Serial.print(" "); // per dare uno spazio nelle acquisizioni
 Serial.print(x);
 Serial.print(" ");
 Serial.print(th);
 Serial.print(" ");
 Serial.print(t);

 // lettuara input da sensori e conversione
 x = -1.0 * ( (double)analogRead(x_pin) * corsa / 1023.0 - corsa / 2 );
 th = -1.0 * ( (double)analogRead(th_pin) * 1.57 / 1023.0 - 0.785 ) - 0.291;

 // salvataggio valori di errore del ciclo precedente
 // operazione necessaria per eseguire derivate e integrali
 e_x_prec = e_x;
 e_th_prec = e_th;

 // errore ciclo attuale
 e_x = x_set - x;

 // derivata errore con saturazione tra -100 e 100
 d_e_x = (double)( (e_x - e_x_prec) / Ts ) * GCE_x;
 if (d_e_x > 100)
   d_e_x = 100;
 if (d_e_x < -100)
   d_e_x = -100;

 // integrale dell'errore (con approssimazione trapezio)
 i_e_x = (double)(e_x + e_x_prec) / Ts;

 // richiamo funzioni fc_1
 fc_1->setInput(1, (double)e_x * GE_x);
 fc_1->setInput(2, d_e_x);
 fc_1->fuzzify();
 out1 = fc_1->defuzzify(1);
 th_set = (double)( (i_e_x * GIU_x) + out1 ) * GU_x;

 // errore ciclo attuale
 e_th = th_set - th;

 // derivata errore con saturazione tra -100 e 100
 d_e_th = (double)( (e_th - e_th_prec) / Ts ) * GCE_th;
 if (d_e_th > 100)
   d_e_th = 100;
 if (d_e_th < -100)
```

```
  d_e_th = -100;

 // integrale dell'errore (con approssimazione trapezio)
 i_e_th = (double)(e_th + e_th_prec) / Ts;

 // richiamo funzioni fc_1
 fc_1->setInput(1, (double)e_th * GE_th);
 fc_1->setInput(2, d_e_th);
 fc_1->fuzzify();
 out2 = fc_1->defuzzify(1);
 u = (double)( (i_e_th * GIU_th) + out2 ) * GU_th;

 if (u > 1)
  u = 1;
 if (u < -1)
  u = -1;

 // invio comandi alle valvole
 if (u == 0) {
  analogWrite(valv_12_pin, 0);
  analogWrite(valv_34_pin, 0);
 }
 if (u > 0) {
  analogWrite(valv_12_pin, (double)u * 255); // ricordiamo che -1<=u<=1
  analogWrite(valv_34_pin, 0);
 }
 if (u < 0) {
  analogWrite(valv_12_pin, 0); // ricordiamo che -1<=u<=1
  analogWrite(valv_34_pin, (double)abs(u) * 255);
 }

 // tempo di attesa per rilanciare il loop
 // attenzione che in delay il tempo va in ms (1 s = 1000 ms)
 delay(Ts * 1000);
 t = t + Ts; // aggiornamento della varibile temporale (per tracciare poi i grafici)
}
```

11.8 codice Arduino PID-like

la funzione Set_LogicaFuzzy() è già definita in un codice soprastante e non viene di conseguenza riportata. Anche il voidSetup non è riportato in quanto identico al caso PD+I

```
#include <Fuzzy.h>

// fuzzy: è la variabile fuzzy
Fuzzy *fuzzy = new Fuzzy();
```

```
// definisco pigreco
const float pi = 3.14159265358979;

// assegnazione dei PIN
const int x_pin = A8;
const int th_pin = A2;
const int valv_12_pin = 4; // comanda valvola 1 e 2
const int valv_34_pin = 13; // comanda valvola 3 e 4

// condizioni iniziali variabili cinematiche
float x = 0;
float th = 0;

// variabili di errore
float e_x = 0; // e : errore
float e_th = 0;
float d_e_x = 0; // d : derivata
float d_e_th = 0;
float i_e_fc_x = 0; // i : integrale errore controllo fuzzy
float i_e_fc_th = 0;
// errori cicli precedenti
float e_x_prec = 0;
float e_th_prec = 0;

// variabili di set
float x_set = 0; // variabile da modificare
float th_set = 0;

// variabile di comando valvole
float u = 0;

// tempo di campionamento
float Ts = 0.005;

// costanti geometriche
float corsa = 0.5;

// guadagni PID
float Kp_th = 2.09;
float Ki_th = 6.31;
float Kd_th = 0.103;

float Kp_x = -0.000802;
float Ki_x = -4.68e-6;
float Kd_x = -0.0306;

// guadagni Fuzzy
float k_prop=25;
float k_der=1.66;
```

```
float k_int=1/5;
float x_max=corsa/2;
float GE_x=100/x_max;
float GU_x=k_prop * ( -(-Kp_x*GE_x) - sqrt( pow( (Kp_x*GE_x) , 2 )-4*GE_x*GE_x*Kd_x*Ki_x ) ) / (
2*GE_x*GE_x ) ; // pow(base,esponente)
float GCE_x=k_der*( Kd_x/GU_x );
float GIU_x=k_int*( Ki_x/( GU_x*GE_x ) );

float th_max=pi*20/180;
float GE_th=100/th_max;
float GU_th= ( -(-Kp_th*GE_th) + sqrt( pow( (Kp_th*GE_th) , 2 )-4*GE_th*GE_th*Kd_th*Ki_th ) ) / (
2*GE_th*GE_th ) ;
float GCE_th=Kd_th/GU_th;
float GIU_th=Ki_th/( GU_th*GE_th );

// output funzioni fuzzy e valori precedenti (per fare integrale)
float out1=0;
float out2=0;
float out1_prec=0;
float out2_prec=0;

// in e out funzione fuzzy
float err;
float d_err;
float out_fc;

// variabile temporale
float t=0;

void loop() {

 // valori a video per acquisizione dati
 Serial.println(u); // println: manda in automatico a capo
 Serial.print(" "); // per dare uno spazio nelle acquisizioni
 Serial.print(x);
 Serial.print(" ");
 Serial.print(th);
 Serial.print(" ");
 Serial.print(t);

 // lettuara input da sensori e conversione
 x = -1.0 * ( (double)analogRead(x_pin) * corsa / 1023.0 - corsa / 2 );
 th = -1.0 * ( (double)analogRead(th_pin) * 1.57 / 1023.0 - 1.57/2 ) - 0.291;

 // salvataggio valori di errore del ciclo precedente
 // operazione necessaria per eseguire derivate e integrali
 e_x_prec = e_x;
 e_th_prec = e_th;
```

```
// errore ciclo attuale
e_x = x_set - x;

// derivata errore con saturazione tra -100 e 100
d_e_x = (double)( (e_x - e_x_prec)/Ts )*GCE_x;
if (d_e_x > 100)
  d_e_x = 100;
if (d_e_x < -100)
  d_e_x = -100;

// richiamo funzioni fuzzy
fuzzy->setInput(1,(double)e_x*GE_x);
fuzzy->setInput(2,d_e_x);
fuzzy->fuzzify();
out1_prec=out1; // salvo il valore precedente
out1=fuzzy->defuzzify(1);

// integrale della funzione fc (con approssimazione trapezio)
i_e_fc_x=(double)(out1 + out1_prec)/ Ts;

th_set = (double)( out1*(i_e_fc_x*GIU_x) + out1 )*GU_x;

// errore ciclo attuale
e_th = th_set - th;

// derivata errore con saturazione tra -100 e 100
d_e_th = (double)( (e_th - e_th_prec)/Ts )*GCE_th;
if (d_e_th > 100)
  d_e_th = 100;
if (d_e_th < -100)
  d_e_th = -100;

// richiamo funzioni fuzzy
fuzzy->setInput(1,(double)e_th*GE_th);
fuzzy->setInput(2,d_e_th);
fuzzy->fuzzify();
out2_prec=out2; // salvo il valore precedente
out2=fuzzy->defuzzify(1);

// integrale dell'errore (con approssimazione trapezio)
i_e_fc_th=(double)(out2 + out2_prec)/ Ts;
u = (double)( out2*(i_e_fc_th*GIU_th) + out2 )*GU_th;

if (u > 1)
  u = 1;
if (u < -1)
  u = -1;

// invio comandi alle valvole
```

```
 if (u == 0) {
   analogWrite(valv_12_pin, 0);
   analogWrite(valv_34_pin, 0);
 }
 if (u > 0) {
   analogWrite(valv_12_pin, (double)u * 255); // ricordiamo che -1<=u<=1
   analogWrite(valv_34_pin, 0);
 }
 if (u < 0) {
   analogWrite(valv_12_pin, 0); // ricordiamo che -1<=u<=1
   analogWrite(valv_34_pin, (double)abs(u) * 255);
 }

 // tempo di attesa per rilanciare il loop
 // attenzione che in delay il tempo va in ms (1 s = 1000 ms)
 delay(Ts * 1000);
 t=t+Ts; // aggiornamento della varibile temporale (per tracciare poi i grafici)
}
```

11.9 codice Arduino PD+PI

la funzione Set_LogicaFuzzy() è già definita in un codice soprastante e non viene di conseguenza riportata. Anche il voidSetup non è riportato in quanto identico al caso PD+I

```
#include <Fuzzy.h>

// fuzzy: è la variabile fuzzy
Fuzzy *fuzzy = new Fuzzy();

// definisco pigreco
const float pi = 3.14159265358979;

// assegnazione dei PIN
const int x_pin = A8;
const int th_pin = A2;
const int valv_12_pin = 4; // comanda valvola 1 e 2
const int valv_34_pin = 13; // comanda valvola 3 e 4

// condizioni iniziali variabili cinematiche
float x = 0;
float th = 0;

// variabili di errore
float e_x = 0; // e : errore
float e_th = 0;
float d_e_x = 0; // d : derivata
float d_e_th = 0;
float i_e_x = 0; // i : integrale
```

```
float i_e_th = 0;
// errori cicli precedenti
float e_x_prec = 0;
float e_th_prec = 0;

// variabili di set
float x_set = 0; // variabile da modificare
float th_set = 0;

// variabile di comando valvole
float u = 0;

// tempo di campionamento
float Ts = 0.005;

// costanti geometriche
float corsa = 0.5;

// guadagni PID
float Kp_th = 2.09;
float Ki_th = 6.31;
float Kd_th = 0.103;

float Kp_x = -0.000802;
float Ki_x = -4.68e-6;
float Kd_x = -0.0306;

// guadagni Fuzzy
float k_prop=8;
float k_der=1.66;
float k_int=1;
float x_max=0.25;
float GE_x=100/x_max;
float GU_x=Kp_x/(2*GE_x)*k_prop;
float GCE_x=(Kd_x/GU_x)*k_der;
float GIU_x=(Ki_x/GU_x)*k_int;

float th_max=pi*20/180;
float GE_th=100/th_max;
float GU_th=Kp_th/(2*GE_th);
float GCE_th=Kd_th/GU_th;
float GIU_th=Ki_th/GU_th;

// output funzioni fuzzy
float out1=0;
float out2=0;
float out3=0;
float out4=0;
```

```
// in e out funzione fuzzy
float err;
float d_err;
float out_fc;

// variabile temporale
float t = 0;

void loop() {

 // valori a video per acquisizione dati
 Serial.println(u); // println: manda in automatico a capo
 Serial.print(" "); // per dare uno spazio nelle acquisizioni
 Serial.print(x);
 Serial.print(" ");
 Serial.print(th);
 Serial.print(" ");
 Serial.print(t);

 // lettuara input da sensori e conversione
 x = -1.0 * ( (double)analogRead(x_pin) * corsa / 1023.0 - corsa / 2 );
 th = -1.0 * ( (double)analogRead(th_pin) * 1.57 / 1023.0 - 0.785 ) - 0.291;

 // salvataggio valori di errore del ciclo precedente
 // operazione necessaria per eseguire derivate e integrali
 e_x_prec = e_x;
 e_th_prec = e_th;

 // errore ciclo attuale
 e_x = x_set - x;

 // derivata errore con saturazione tra -100 e 100
 d_e_x = (double)( (e_x - e_x_prec) / Ts ) * GCE_x;
 if (d_e_x > 100)
   d_e_x = 100;
 if (d_e_x < -100)
   d_e_x = -100;

 // integrale dell'errore (con approssimazione trapezio)
 i_e_x = (double)( (e_x + e_x_prec) / Ts ) * GIU_x;
 if (i_e_x > 100)
   i_e_x = 100;
 if (i_e_x < -100)
   i_e_x = -100;

 // richiamo funzioni fuzzy
 fuzzy->setInput(1, (double)e_x * GE_x);
 fuzzy->setInput(2, d_e_x);
 fuzzy->fuzzify();
```

```
out1 = fuzzy->defuzzify(1);

fuzzy->setInput(1, (double)e_x * GE_x);
fuzzy->setInput(2, i_e_x);
fuzzy->fuzzify();
out2 = fuzzy->defuzzify(1);
th_set = (double)( out1 + out2 ) * GU_x;

// errore ciclo attuale
e_th = th_set - th;

// derivata errore con saturazione tra -100 e 100
d_e_th = (double)( (e_th - e_th_prec) / Ts ) * GCE_th;
if (d_e_th > 100)
  d_e_th = 100;
if (d_e_th < -100)
  d_e_th = -100;

// integrale dell'errore (con approssimazione trapezio)
i_e_th = (double)( (e_th + e_th_prec) / Ts ) * GIU_th;
if (i_e_th > 100)
  i_e_th = 100;
if (i_e_th < -100)
  i_e_th = -100;

// richiamo funzioni fuzzy
fuzzy->setInput(1, (double)e_th * GE_th);
fuzzy->setInput(2, d_e_th);
fuzzy->fuzzify();
out3 = fuzzy->defuzzify(1);

fuzzy->setInput(1, (double)e_th * GE_th);
fuzzy->setInput(2, i_e_th);
fuzzy->fuzzify();
out4 = fuzzy->defuzzify(1);
u = (double)( out3 + out4 ) * GU_th;

if (u > 1)
  u = 1;
if (u < -1)
  u = -1;

// invio comandi alle valvole
if (u == 0) {
  analogWrite(valv_12_pin, 0);
  analogWrite(valv_34_pin, 0);
}
if (u > 0) {
  analogWrite(valv_12_pin, (double)u * 255); // ricordiamo che -1<=u<=1
```

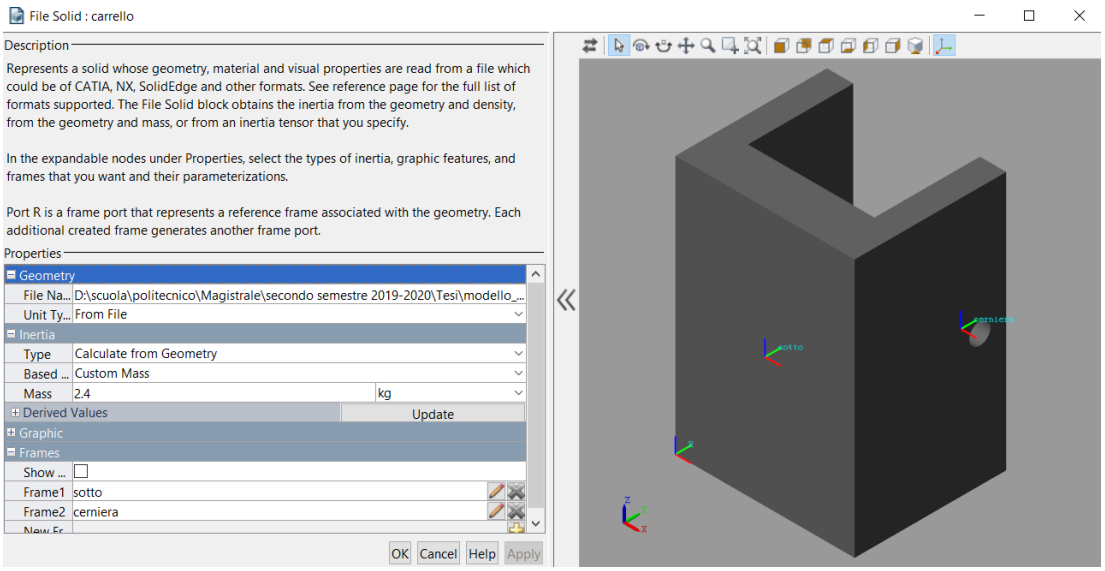
```
  analogWrite(valv_34_pin, 0);
 }
 if (u < 0) {
  analogWrite(valv_12_pin, 0); // ricordiamo che -1<=u<=1
  analogWrite(valv_34_pin, (double)abs(u) * 255);
 }

 // tempo di attesa per rilanciare il loop
 // attenzione che in delay il tempo va in ms (1 s = 1000 ms)
 delay(Ts * 1000);
 t = t + Ts; // aggiornamento della varibile temporale (per tracciare poi i grafici)
}
```

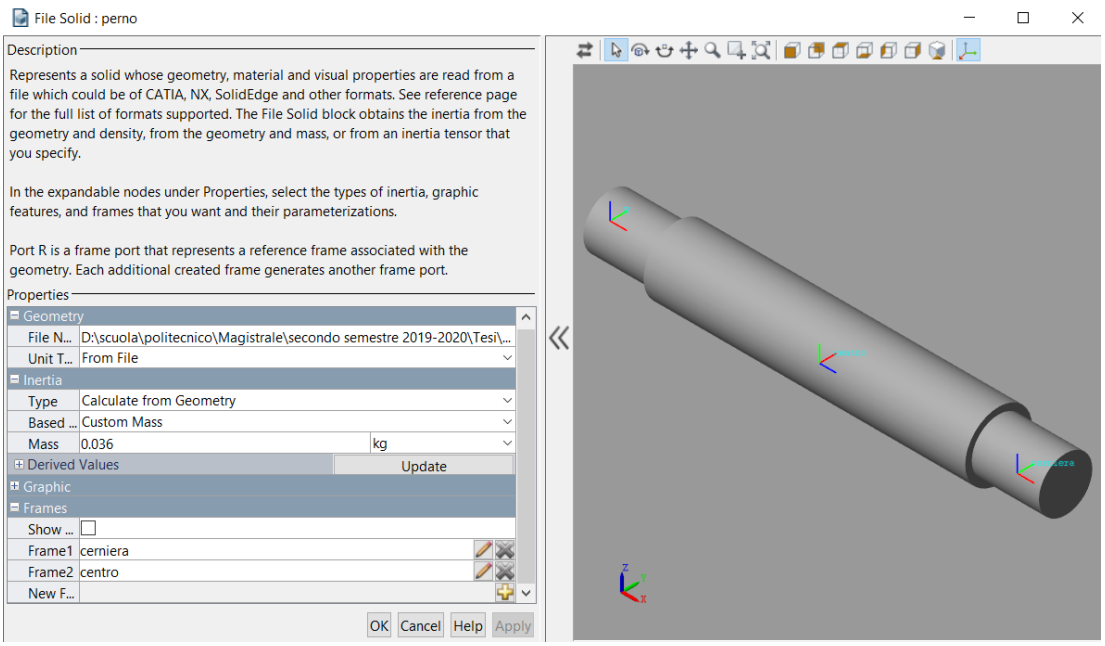
11.10 codice Matlab per il controllo PD+I da caricare sulla scheda Arduino

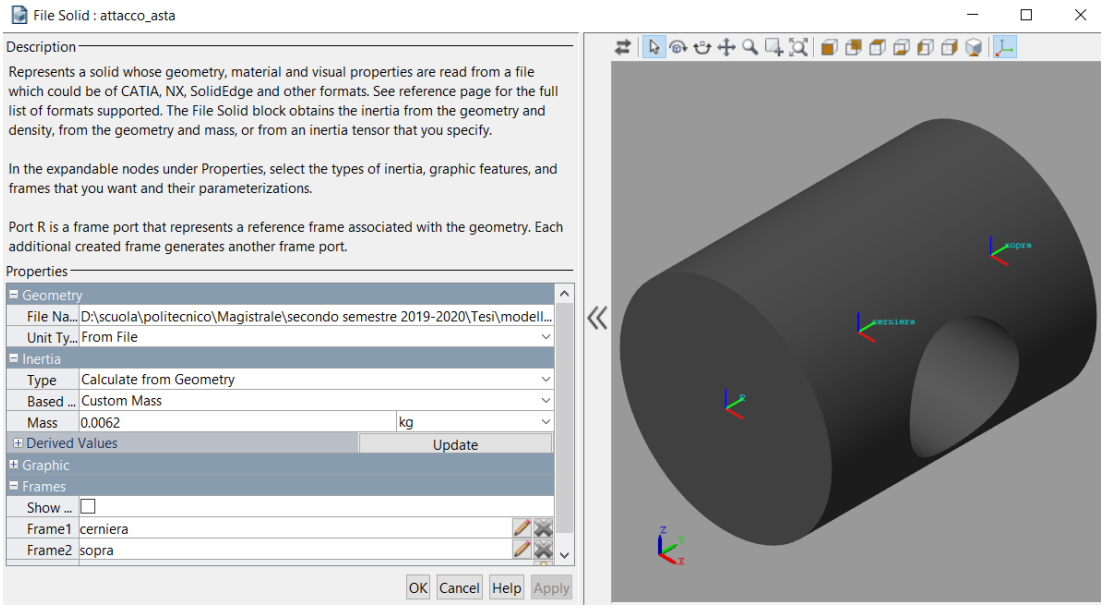
```
%% caricamento dati

Tsim=0.01; % tempo di campionamento [s]
t_in=0; % tempo inizio simulazione [s]
t_fin=10; % tempo fine simulazione [s]
dh_zone=10^(-10); % dead zone (per evitare errori numerici)

corsa=500*10^(-3); % corsa del cilindro [m]
th_90=deg2rad(90); % per come lavora il sensore
th_pos=0.291; % dovuto ad installazione sensore


% guadagni PID
Kp_x=-0.000802;
Ki_x=-4.68e-6;
Kd_x=-0.0306;
Tf_x=0.00875;

Kp_th=2.09;
Ki_th=6.31;
Kd_th=0.103;
Tf_th=0.000875;

% guadagni Fuzzy
k_prop=10;
k_der=1.33;
k_int=1;
x_max=0.25;
GE_x=100/x_max;
GU_x=(Kp_x/GE_x)*k_prop;
GCE_x=(Kd_x/GU_x)*k_der;
GIU_x=(Ki_x/GU_x)*k_int;

th_max=pi*20/180;
GE_th=100/th_max;
GU_th=Kp_th/GE_th;
```

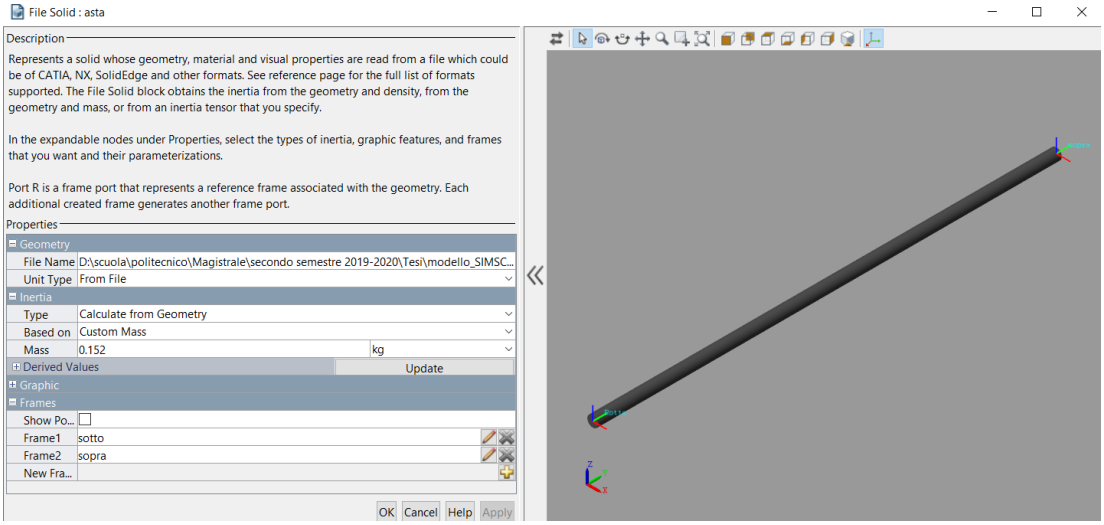
```
GCE_th=Kd_th/GU_th;
GIU_th=Ki_th/GU_th;


%% post processing (da eseguire terminata la simulazione)

figure(1)
subplot(3,1,1)
hold on
plot(tt,x_set),grid on
plot(tt,x),grid on
xlabel('tempo [s]'),ylabel('x')
legend('x set','x fb')
subplot(3,1,2)
plot(tt,th),grid on
xlabel('tempo [s]'),ylabel('th')
subplot(3,1,3)
plot(tt,u),grid on
xlabel('tempo [s]'),ylabel('u')
```

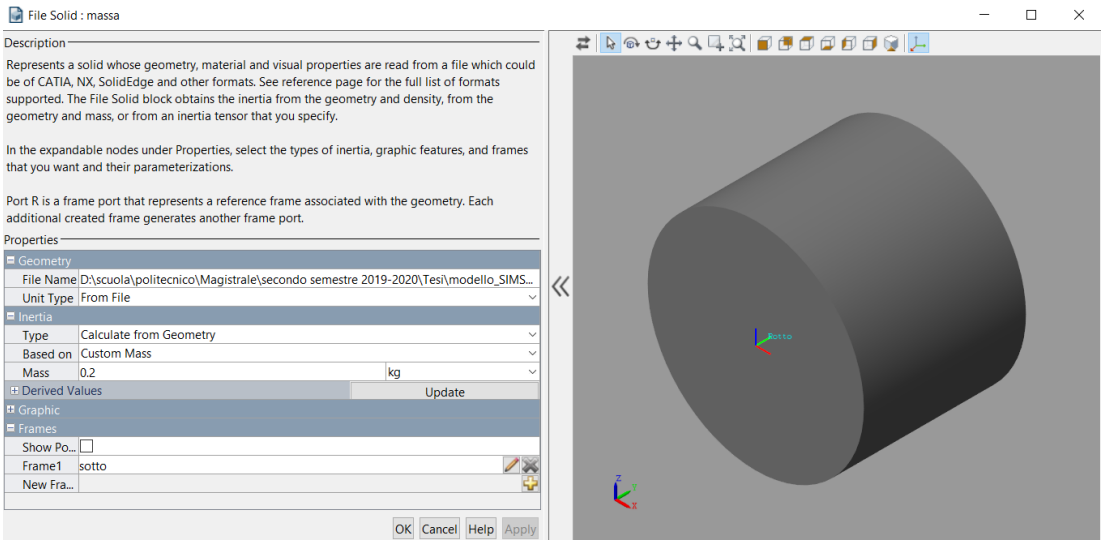
11.11 File Solid in Simscape-Multibody con sistemi riferimento






File Solid : attacco_asta
Description
Represents a solid whose geometry, material and visual properties are read from a file which could be of CATIA, NX, SolidEdge and other formats. See reference page for the full list of formats supported. The File Solid block obtains the inertia from the geometry and density, from the geometry and mass, or from an inertia tensor that you specify.
In the expandable nodes under Properties, select the types of inertia, graphic features, and frames that you want and their parameterizations.
Port R is a frame port that represents a reference frame associated with the geometry. Each additional created frame generates another frame port.
Properties
Geometry
File Na... D:\scuola\politecnico\Magistrale\secondo semestre 2019-2020\Tesi\modell...
Unit Ty... From File
Inertia
Type Calculate from Geometry
Based ... Custom Mass
Mass 0.0062 kg
Derived Values Update
Graphic
Frames
Show ...
Frame1 cerniera
Frame2 sopra
OK Cancel Help Apply
sopra
cerniera


File Solid : asta
Description
Represents a solid whose geometry, material and visual properties are read from a file which could be of CATIA, NX, SolidEdge and other formats. See reference page for the full list of formats supported. The File Solid block obtains the inertia from the geometry and density, from the geometry and mass, or from an inertia tensor that you specify.
In the expandable nodes under Properties, select the types of inertia, graphic features, and frames that you want and their parameterizations.
Port R is a frame port that represents a reference frame associated with the geometry. Each additional created frame generates another frame port.
Properties
Geometry
File Name D:\scuola\politecnico\Magistrale\secondo semestre 2019-2020\Tesi\modello_SIMSC...
Unit Type From File
Inertia
Type Calculate from Geometry
Based on Custom Mass
Mass 0.152 kg
Derived Values Update
Graphic
Frames
Show Po...
Frame1 sotto
Frame2 sopra
New Fra...
OK Cancel Help Apply


File Solid : massa
Description
Represents a solid whose geometry, material and visual properties are read from a file which could be of CATIA, NX, SolidEdge and other formats. See reference page for the full list of formats supported. The File Solid block obtains the inertia from the geometry and density, from the geometry and mass, or from an inertia tensor that you specify.
In the expandable nodes under Properties, select the types of inertia, graphic features, and frames that you want and their parameterizations.
Port R is a frame port that represents a reference frame associated with the geometry. Each additional created frame generates another frame port.
Properties
Geometry
File Name D:\scuola\politecnico\Magistrale\secondo semestre 2019-2020\Tesi\modello_SIMS...
Unit Type From File
Inertia
Type Calculate from Geometry
Based on Custom Mass
Mass 0.2 kg
Derived Values Update
Graphic
Frames
Show Po...
Frame1 sotto
New Fra...
OK Cancel Help Apply
sotto

12 Bibliografia

[1] S. D. Natale, Model-based design of a fuzzy logic controller for an inverse pendulum, 2018/2019.

[2] V. V. e F. Colombo, Automazione dei sistemi meccanici, corso di base.

[3] J. Jantzen, Foundations of fuzzy control, a practical approach, wiley.

[4] I. M.Dotoli, Controllo fuzzy di un pendolo inverso.

[5] A. Marino, Realizzazione di un banco a pendolo inverso con attuatore pneumatico e controllo mediante PLC e Arduino.

[6] M. Pontin, Modellazione, realizzazione e controllo mediante PLC di un sistema a pendolo inverso ad attuazione pnematica, 2018.

[7] M. Sorli, PDF lezioni di Meccatronica, Politecnico, 2020.

[8] https://store.arduino.cc/arduino-mega-2560-rev3.

[9] «zerokol / eFLL,» [Online]. Available: https://github.com/zerokol/eFLL.

[10] «eFLL - A Fuzzy Library for Arduino and Embedded Systems,» [Online]. Available: https://blog.zerokol.com/2012/09/arduinofuzzy-fuzzy-library-for-arduino.html.

[11] https://www.youmath.it/lezioni/fisica/dinamica/3013-teorema-di-steiner.html.