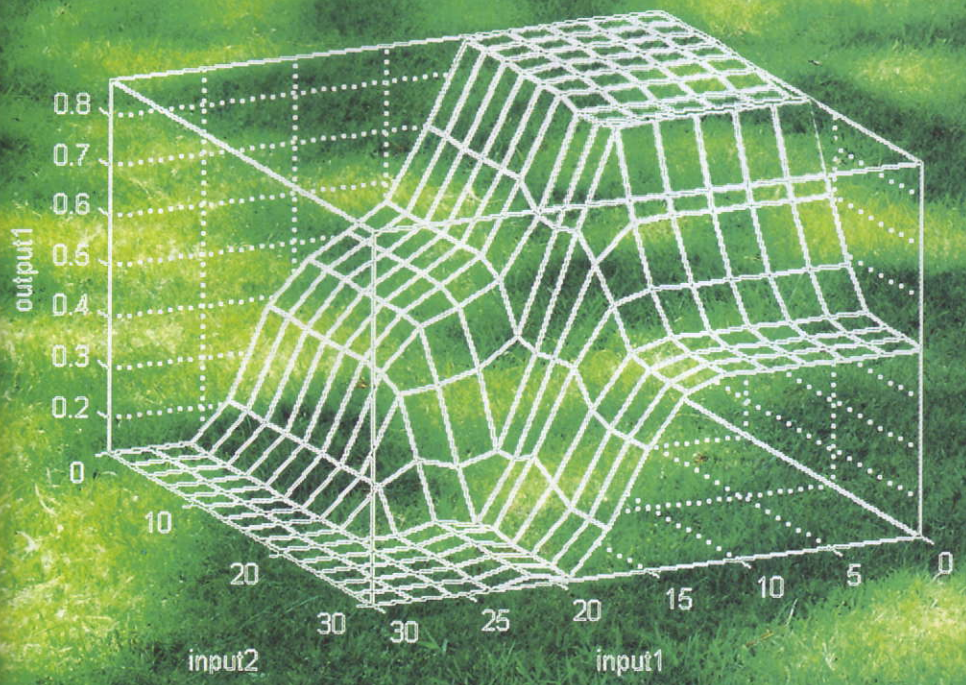


Per la logica binaria un bicchiere o è pieno o è vuoto ed un prato o è al sole o in ombra; approfondiamo un approccio logico che permette di programmare un computer in modo da riconoscere le mille sfumature tra pieno, vuoto, sole ed ombra.

LOGICA FUZZY CON ARDUINO

di DANIELE DENARO

Il bicchiere è pieno o vuoto? Il prato è al sole o in ombra? Usando la logica Booleana, non c'è scampo: il bicchiere è da considerare pieno o è da considerare vuoto. Pieno a metà, ovvero 0.5 non è contemplato: solo 0 oppure 1. Questo approccio provoca talvolta rigidità di funzionamento inaccettabili. Come nei processi a regole (es. sistemi esperti). Cioè nei processi che prendono in considerazione una serie di condizioni: per esempio una serie di istruzioni IF, da cui conseguire una certa azione (**Fig. 1**).

Fin dagli anni 60 a Berkeley il professore Zadeh si era reso conto che si poteva introdurre una nuova logica non più a due stati, ma per così dire, più "dispersa","sfumata": "fuzzy", appunto. Lo stato di una variabile non è più 0 o 1 ma un qualunque valore tra questi estremi. Valore che rappresenta la "vicinanza" alla verità 1.

È evidente, allora, che vanno ridefinite le funzioni logiche AND, OR e NOT. Usualmente si utilizzano le funzioni minimo per AND, massimo per OR e complemento a 1 per NOT. Per esempio se A=0.8 e B=0.3: C=(A AND B)

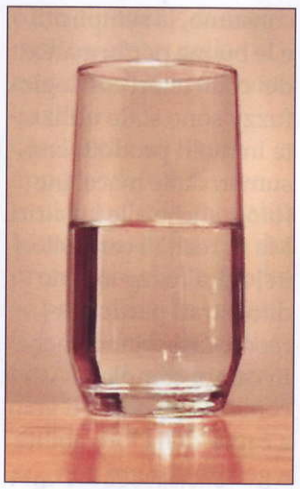


***Fig. 1** - Pieno o vuoto?*

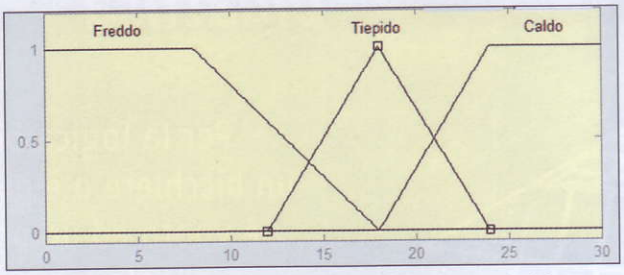


***Fig. 2** - Categorie di Freddo, Tiepido e Caldo.*

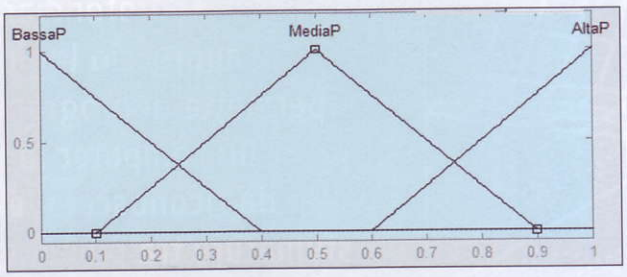


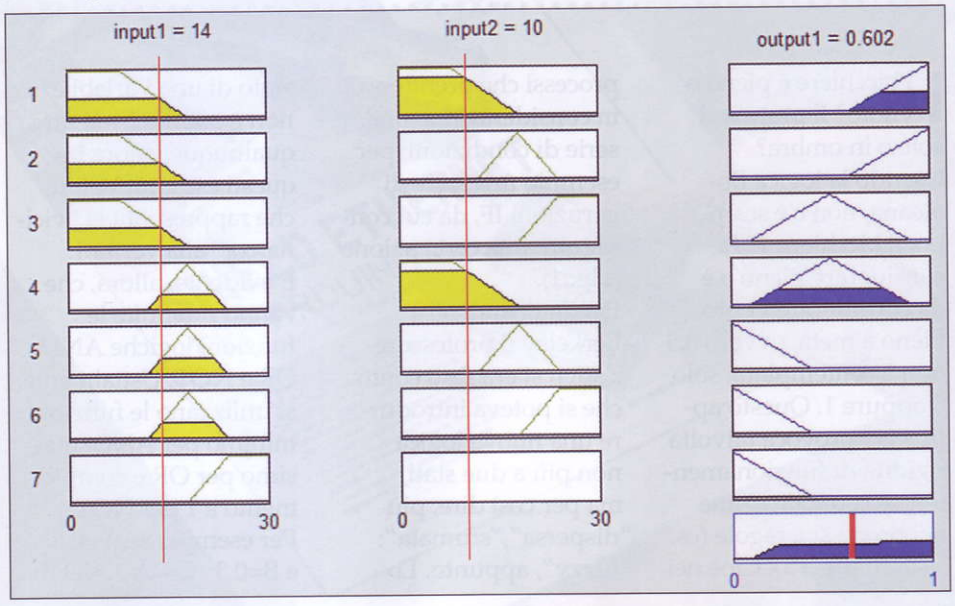
***Fig. 3** - Categorie di potenze.*

nel calcolo fuzzy sarà pari a C=0.3. Ovvero C, che deve contemplare la veridicità contemporanea di A e B, non può che essere un sottoinsieme comune a tutti e due, ovvero 0.3. Analogamente C=(A OR B)=0.8 e C= (NOT A)=0.2.
Successivamente negli anni '70 e poi '80, la logica Fuzzy ha cominciato ad essere utilizzata nel controllo automatico con ottimo successo. Successivamente, con la pervasività dei microprocessori nei prodotti di consumo, la semplicità e le buone performance dei controllori con logica fuzzy sono state utilizzate in molti prodotti consumer: dalle macchine fotografiche alle lavatrici. Ma in realtà i controllori in logica fuzzy si sono dimostrati particolarmente importanti anche in campi complessi come lo spazio ed il volo a causa della loro abilità a gestire sistemi non lineari. Non si capisce allora perché non utilizzare la logica fuzzy in un hardware come Arduino destinato ad una utenza non solo di tecnici e ingegneri. Ciò perché un'altra importante caratteristica della logica fuzzy è quella di essere facilmente descritta da regole di "buon senso".
Per capire il funzionamento di un PID è necessario conoscere i rudimenti della teoria del controllo automatico, pur rimanendo l'uso del PID una attività di consistente contenuto euristico (i.e. per tentativi). Mentre un controller fuzzy si comporta come farebbe un essere umano .
Vediamo un esempio. Supponiamo di voler controllare un termoventilatore avendo a disposizione la temperatura dell'ambiente da riscaldare e la temperatura dell'esterno.
In questo caso la prima cosa da fare è definire dei concetti di variabili fuzzy per questi due input. Per esempio potremo definire, per entrambi, delle temperature tipo, come: "Freddo", "Tiepido" e "Caldo". A questi concetti (chiamati "set" nella logica Fuzzy) vanno attribuiti dei range di temperature in cui essi sono validi (**Fig. 2**).
Per l'output possiamo considerare tre gradi di potenza : "BassaP", "MediaP", AltaP" (**Fig. 3**).
Queste definizioni corrispondono alla fase cosiddetta di "fuzzificazione", in cui vengono organizzate delle variabili "fuzzy" collegate al mondo reale dalle "membership

***Fig. 4** - Regole di comportamento.*

Se interno è Freddo ed esterno è Freddo allora AltaP
Se interno è Freddo ed esterno è Tiepido allora AltaP
Se interno è Freddo ed esterno è Caldo allora MediaP
Se interno è Tiepido ed esterno è Freddo allora MediaP
Se interno è Tiepido ed esterno è Tiepido allora BassaP
Se interno è Tiepido ed esterno è Caldo allora BassaP
Se interno è Caldo allora BassaP

***Fig. 5** - Esecuzione delle regole.*



functions"; ovvero funzioni del grado di verità rispetto alla grandezza rappresentata. Queste funzioni sono decise dall'utente utilizzando combinazioni lineari come triangoli e trapezi, ma anche funzioni non lineari come gaussiane od altro.
A questo punto si tratta di immaginare cosa farebbe un essere intelligente (e non troppo freddoloso) se comandasse il termoventilatore. In **Fig. 4** è descritto un possibile comportamento.
Poiché alla fine si tratta di comandare la potenza del termoventilatore (per esempio con PWM), bisogna trasformare l'insieme di queste regole in un valore preciso di potenza.
Questo viene fatto dal processo di "inferenza" e dal processo di "defuzzificazione". Il processo di inferenza acquisisce un valore per ogni input e usando le regole determina il valore fuzzy di output. Per esempio se la temperatura interna fosse di 14 gradi e quella esterna di 10 gradi, allora il valore di output sarebbe 0.6 (**Fig. 5**).
Infatti il valore fuzzy di output sarebbe fornito da un basso valore di verità per la potenza alta ma anche da un basso valore di verità per la potenza media.
Per passare da questo insieme ad un valore preciso si usa in genere il metodo del baricentro (defuzzificazione) che fornisce appunto 0.606.
Per tutti i valori dei due input ne risulta una funzione superficie non lineare come in **Fig. 6**.
Tutte queste non linearità , la possibilità di spezzare il problema in sottoinsiemi, la possibilità di usare in combinazione più input (ma anche più output) e la possibilità di fare affidamento sull'approccio a regole, permette ai controlli fuzzy di essere molto potenti nella gestione di sistemi non lineari.
Quello che abbiamo appena visto è un controller fuzzy in modalità Mamdani, dal nome del professore che lo propose per primo. In realtà il processo di defuzzificazione è abbastanza dispendioso per un

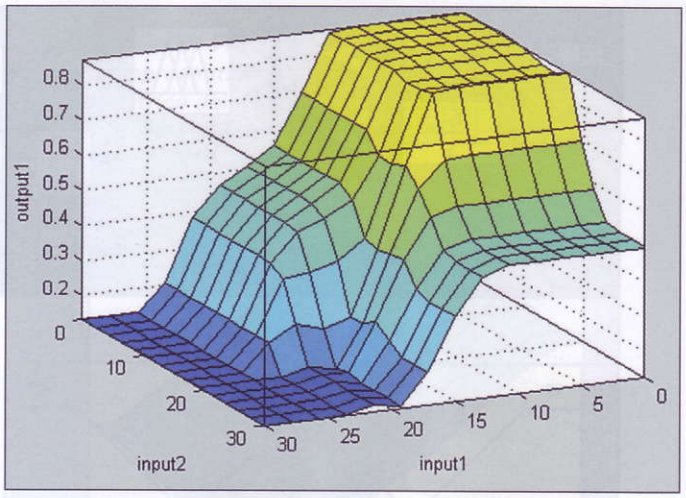


***Fig. 6** - Output in funzione dei possibile valori di input.*

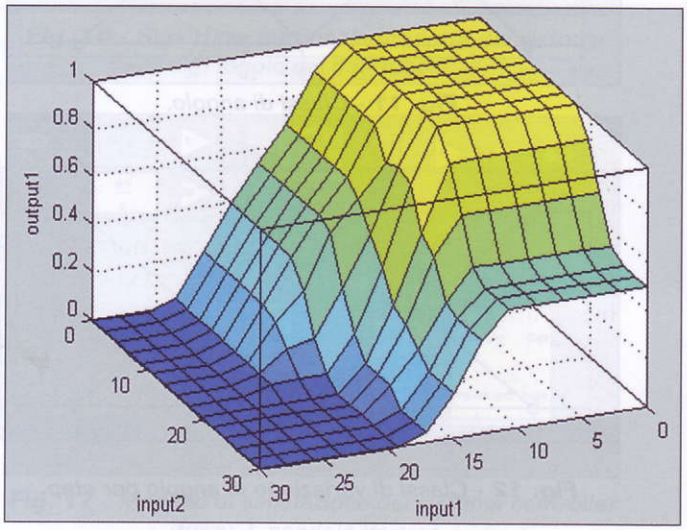


***Fig. 8** - Output in funzione dei possibili valori di input in modalità "Sugeno".*

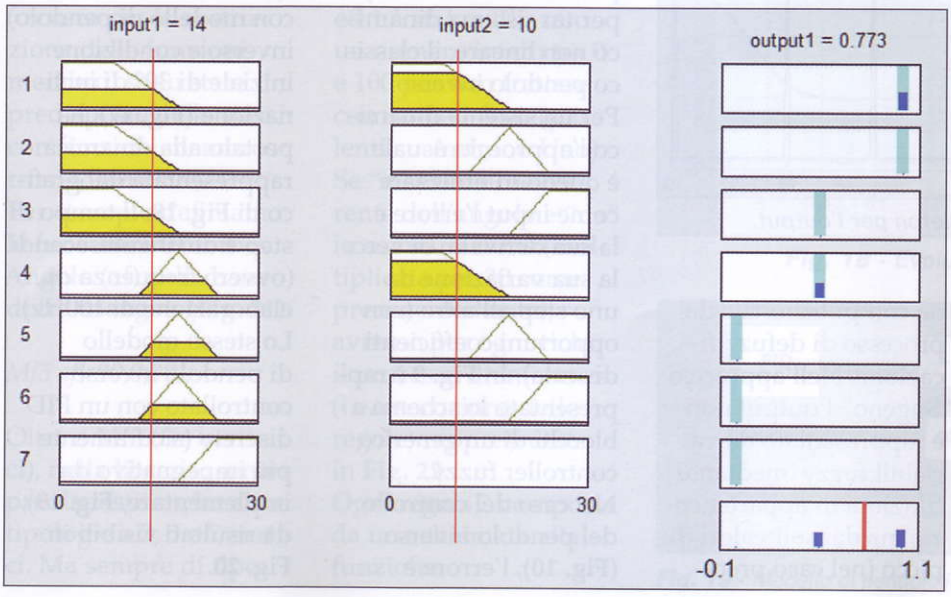


***Fig. 7** - Esecuzione delle regole in modalità "Sugeno".*

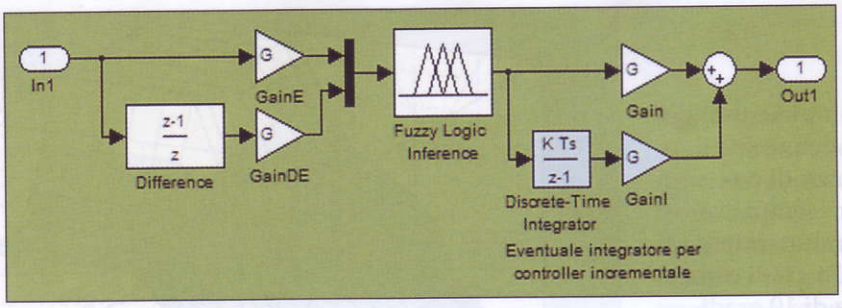


Fig. 9 *Controller fuzzy per sistemi dinamici.*

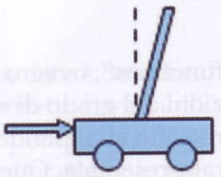


Fig. 10 - *Pendolo inverso.*

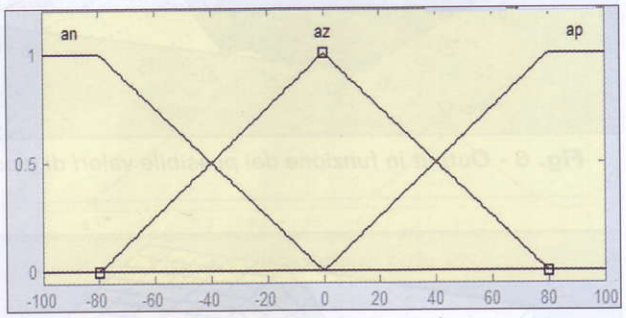


Fig. 11 - *Classi di angolo.*

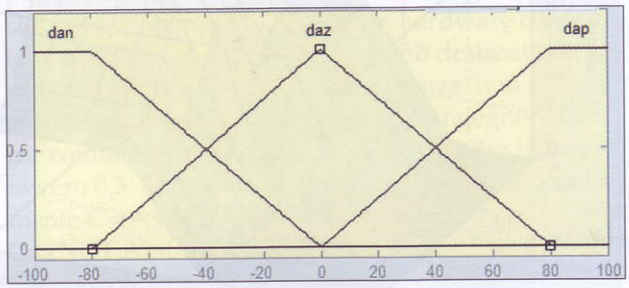


Fig. 12 - *Classi di variazione di angolo per step.*

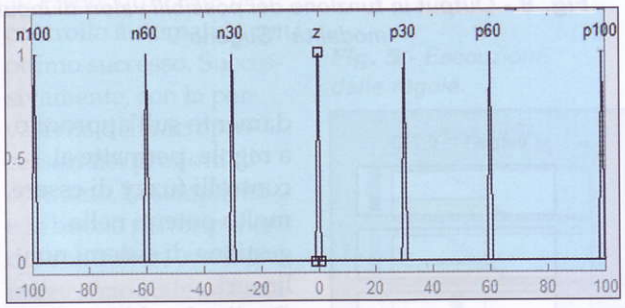


Fig. 13 - *Valori singleton per l'output.*

microcontroller (calcolo del baricentro dell'inviluppo risultante). In questi casi si usa alternativamente l'approccio Sugeno (dal nome del professore che lo propose) che consiste in una notevole semplificazione computazionale del processo di defuzzificazione. Nell'approccio "Sugeno" l'output non è rappresentato da variabili fuzzy mediante funzioni di appartenenza, ma dai soli valori di picco (nel caso precedente 0, 0.5 ,1), chiamati singleton, ed in **Fig. 7** potete vedere come la definizione dell'output è cambiata.

Il calcolo dell'output è fatto dalla semplice media pesata dei singleton. Come si vede, il risultato è un po' diverso rispetto a prima, ma a questo si può ovviare ridefinendo meglio le funzioni di appartenenza e soprattutto i singleton. Anche la superficie funzionale è abbastanza simile, come si può vedere in **Fig. 8.**

La logica fuzzy-Sugeno è stata inserita anche nell'assembler di alcuni microcontroller come la serie ST Five 508 (STMicroelectronics).

Vediamo ora un esempio di controller fuzzy per un sistema dinamico non lineare: il classico pendolo inverso.

Per un sistema dinamico l'approccio usuale è quello di utilizzare come input l'errore e la sua derivata, ovvero la sua variazione da uno step all'altro (con opportuni coefficienti di scala). In **Fig. 9** è rappresentato lo schema a blocchi di un generico controller fuzzy.

Nel caso del controllo del pendolo inverso (**Fig. 10**), l'errore è l'angolo rispetto alla verticale e l'output è la forza di spinta o di trazione del carrello. L'obiettivo è mantenerlo verticale.

Avendo definito per l'errore di angolo tre set (**Fig. 11**), per la sua variazione ancora tre set (ed anche definiti nello stesso modo: **Fig. 12**), per l'output sette singleton (**Fig. 13**)., le regole potrebbero essere quelle illustrate in **Fig. 14**.

O in modo più sintetico, utilizzando una rappresentazione a matrice per i valori di output, come rappresentato dalla figura **Fig. 15**.

Ne risulta la superficie funzionale rappresentata in **Fig. 16**.

La simulazione in ambiente Matlab-Simulink , con modello di pendolo inverso e condizione iniziale di 30° di inclinazione (**Fig. 17**), ha portato alla dinamica rappresentata dal grafico di **Fig. 18**. Il tempo di step è di 10 millisecondi (ovvero frequenza di elaborazione di 100Hz).

Lo stesso modello di pendolo inverso controllato con un PID discreto (sicuramente più impegnativo da implementare, **Fig. 19**), dà risultati visibili in **Fig. 20**.

Fig. 14 *Regole per il controller del pendolo inverso.*

Se a negativo e da negativa allora out n100
Se a negativo e da zero allora out n60
Se a negativo e da positiva allora out n30
Se a zero e da negativa allora out n30
Se a zero e da zero allora out zero
Se a zero e da positiva allora out p30
Se a positivo e da negativa allora out p30
Se a positivo e da zero allora out p60
Se a positivo e da positiva allora out p100

Fig. 15 *Rappresentazione a matrice dei valori di output.*

	dan	daz	dap
an	n100	n60	n30
az	n30	zero	p30
ap	p30	p60	p100

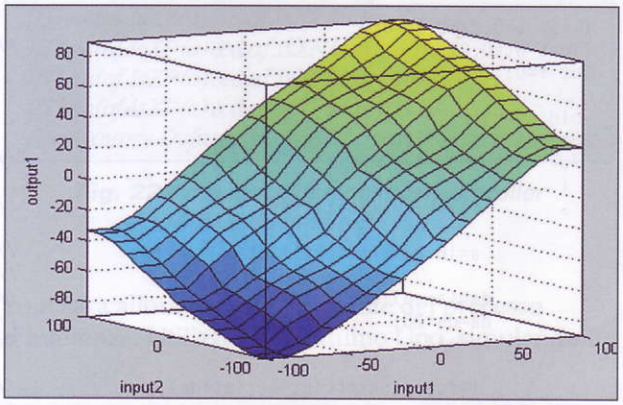


Fig. 16 - *Superficie funzionale prodotta dal sistema di regole del controller.*

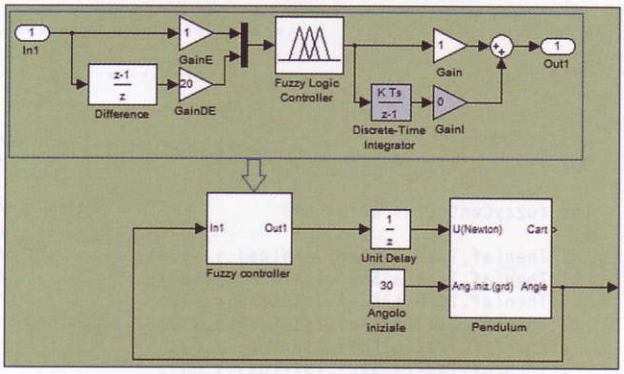


Fig. 17 - *Modello di simulazione del sistema controller fuzzy + pendolo inverso.*

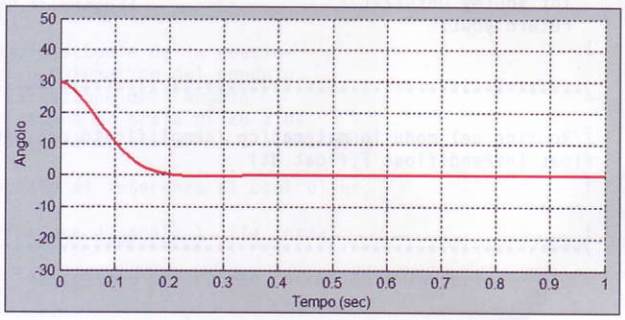


Fig. 18 - *Evoluzione dell'angolo.*

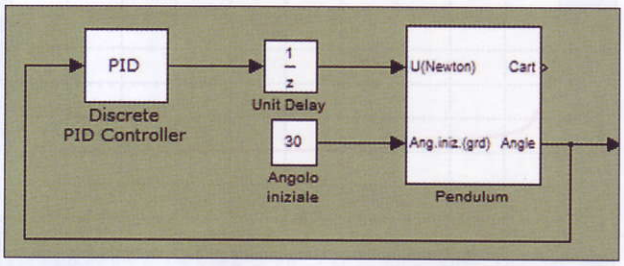


Fig. 19 - *Modello di pendolo inverso controllato da un PID discreto.*

Come si è visto l'approccio fuzzy, lungi da essere più semplicistico è in realtà molto efficace e nel contempo più intuitivo.
Inoltre la logica fuzzy può essere composta anche in modo complesso, con diversi precedenti e diversi conseguenti (termini delle regole).
In Arduino queste strutture possono essere rese abbastanza leggere e semplici da usare. La libreria Fuzzy proposta utilizza valori interi per velocizzare i calcoli.
Vediamo l'esempio del controller per pendolo inverso realizzato con la libreria Fuzzy per Arduino. Per la definizione di variabili Fuzzy (ovvero per la costruzione delle funzioni membro) sono state predisposte delle classi con funzioni membro triangolari.
Per esempio definiamo le funzioni membro Angolo (af) sulla base di tre triangoli (**Fig. 21**).

Mf3 af(-90,0,90);

Oltre a Mf3 (tre vertici), nella libreria, sono predisposte anche altre tipologie a 2, 5 e 7 vertici. Ma sempre di tipo triangolare e consecutivi , per semplificarne la definizione e l'elaborazione. Utilizzeremo lo stesso insieme anche per la variazione dell'angolo.
A questo punto si tratta di definire le regole.
Per le condizioni possiamo utilizzare le funzioni valore di appartenenza (IsX...) , definite nelle classi Mf , e possiamo utilizzare gli operatori logici ridefiniti opportunamente per queste classi: '&&' per AND (con il metodo del minimo), ' || ' per OR (con il metodo del massimo) e '!' per NOT (con il metodo del complemento).
Come valore risultante per la variabile fuzzy , invece del valore tra 0 ed 1 è stato utilizzato un numero intero tra 0 e 100 per non trattare con numeri float (più lenti su Arduino Uno).
Se "a" è il valore corrente dell'angolo e "da" la sua variazione (moltiplicata per 20 come precedentemente), avremo per ogni ciclo il processo di inferenza (i.e. esecuzione delle regole) rappresentato in **Fig. 22**.
Ogni regola è composta da una chiamata alla funzione:

Listato 1

```
#include <Fuzzy.h>

/*****************************************************************************/
/* Esempio di utilizzo della libreria per controllare un pendolo inverso (simulato)  */
/*****************************************************************************/

float a=0,da=0,ap=0;
float dt=0.01;                                  //Intervallo di temporizzazione (sec)
int f=0;

void setup() {
 Serial.begin(9600);
 for (int i=0; i<100;i++)                       //Numero di cicli di simulazione
  {
         float t=(float)i*dt;                   // ciascuno di dt secondi

         a=InvPend((float)f,dt);                //riceve l'angolo attuale dalla routine
                                                //con il modello matematico del pendolo
        da=(a-ap);ap=a;                         //calcola la variazione dell'angolo
        da=da*20;                               //applica un fattore di scala di 20 a da
         a=a*1;                                 //applica un fattore di scala  di 1 all'angolo

         f=fuzzyContr((int)a,(int)da);          //riceve la forza da applicare dalla
                                                //routine che esegue l'inferenza fuzzy
         f=f*1;                                 //applica un fattore di scala 1 alla forza
         Serial.println(a);
         delay(100);                            //ritardo utilizzato solo per non sovraccaricare la seriale
  }
}

void loop() {
}

Mf3 af(-90,0,90);                               //Fuzzy sets (3 set )
Out y;                                          //Definizione dell'unica variabile di output

int fuzzyContr(int a,int da)                    //Routine che implementa il controller Fuzzy
{                                               //Processo di inferenza
  IfThen(af.IsXN(a)&&af.IsXN(da),y,-100);       //Regole
  IfThen(af.IsXN(a)&&af.IsXZ(da),y,-60);        //    :
  IfThen(af.IsXN(a)&&af.IsXP(da),y,-30);        //    :
  IfThen(af.IsXZ(a)&&af.IsXN(da),y,-30);
  IfThen(af.IsXZ(a)&&af.IsXZ(da),y,0);
  IfThen(af.IsXZ(a)&&af.IsXP(da),y,30);
  IfThen(af.IsXP(a)&&af.IsXN(da),y,30);
  IfThen(af.IsXP(a)&&af.IsXZ(da),y,60);
  IfThen(af.IsXP(a)&&af.IsXP(da),y,100);
  int yout=y.Defuzzy();                         //"Defuzzificazione" della variabile di output
  return yout;                                  //Fine del processo di inferenza
}

/*****************************************************************************/

//Routine del modello matematico (semplificato del pendolo inverso.
float InvPend(float F,float dt)
{
    :
}
/*****************************************************************************/
```

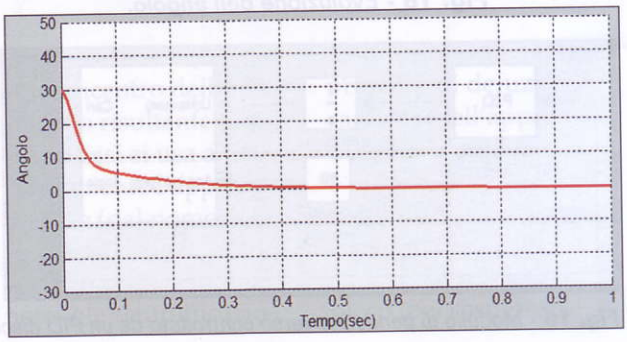


Fig. 20 *Evoluzione dell'angolo nel caso del controllo con PID discreto.*

IfThen(espressione logica fuzzy, variabile di uscita, valore singleton)

Cioè :

- antecedente: espressione logica o variabile fuzzy;
- conseguente: variabile di uscita associata;
- conseguente: valore singleton da attribuirle.

Bisogna, quindi aver definito prima oggetti che rappresentano le variabili di uscita mediante l'uso della classe Out:

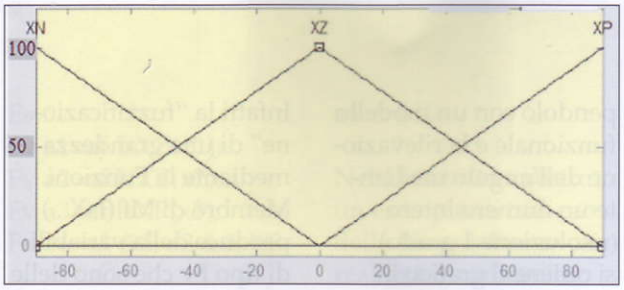


***Fig. 21** - Set di angoli per il controller, con la libreria Fuzzy.*

```
Out y;                          //inizializza la variabile di uscita
IfThen(af.IsXN(a)&&af.IsXN(da),y,-100);
IfThen(af.IsXN(a)&&af.IsXZ(da),y,-60);
IfThen(af.IsXN(a)&&af.IsXP(da),y,-30);
IfThen(af.IsXZ(a)&&af.IsXN(da),y,-30);
IfThen(af.IsXZ(a)&&af.IsXZ(da),y,0);
IfThen(af.IsXZ(a)&&af.IsXP(da),y,30);
IfThen(af.IsXP(a)&&af.IsXN(da),y,30);
IfThen(af.IsXP(a)&&af.IsXZ(da),y,60);
IfThen(af.IsXP(a)&&af.IsXP(da),y,100);
float yout=y.Defuzzy();         //defuzzificazione
```

***Fig. 22** - Inferenza di regole del controller utilizzando la libreria Fuzzy.*

Out y;

Alla fine dell'esecuzione delle regole si chiama, su questo oggetto, la funzione di "defuzzificazione":

float yout=y.Defuzzy();

Yout sarà allora l'azione risultante del processo d'inferenza. Utilizzando Arduino Uno, simulando il

Listato 2

```
#include <Fuzzy.h>

/*****************************************************************************/
/*  Controllo del pendolo inverso con la funzione FuzzyController(...)                                  */
/*****************************************************************************/

float a=0,da=0,ap=0;
float dt=0.01;                          //Intervallo di temporizzazione (sec)
int f=0;

Mf3 af(-90,0,90);                       //Fuzzy sets
int mtx[3][3]={{-100,-60,-30},          //Matrice dei valori Singleton
                          {-30,0,30},
                         {30,60,100}};

FuzzyControllerMf3 FzC(af,mtx,0);       //Fuzzy controller con set af e con matrice mtx

void setup() {
 Serial.begin(9600);
 for (int i=0; i<100;i++)               //numero di cicli di simulazione
  {
        float t=(float)i*dt;            // ciascuno di dt secondi

         a=InvPend((float)f,dt);        //riceve l'angolo attuale dalla routine
                                        //con il modello matematico del pendolo
        da=(a-ap);ap=a;                 //calcola la variazione dell'angolo
        da=da*20;                       //applica un fattore di scala di 20 a da
         a=a*1;                         //applica un fattore di scala  di 1 all'angolo

         f=FzC.Inference((int)a,(int)da); //applica processo di inferenza al controller

         f=f*1;                         //applica un fattore di scala 1 alla forza
         Serial.println(a);
         delay(100);                    //ritardo utilizzato solo per non sovraccaricare la seriale
  }
}

void loop() {
}


/*****************************************************************************/

//Routine del modello matematico (semplificato del pendolo inverso.
float InvPend(float F,float dt)
{
    :
}

/*****************************************************************************/
```

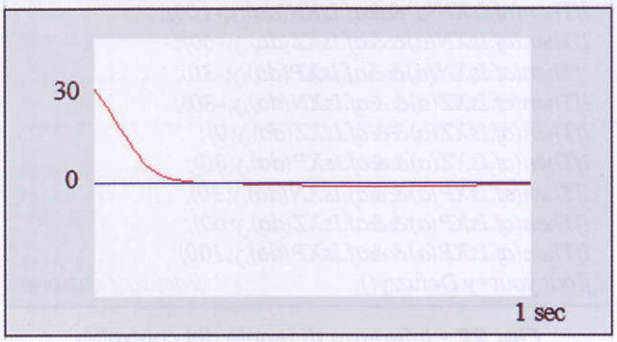


***Fig. 23** - Evoluzione dell'angolo nella simulazione su Arduino Uno.*

pendolo con un modello funzionale e la rilevazione dell'angolo mediante un numero intero (risoluzione 1 grado) si ottiene il grafico in **Fig. 23**. La libreria può essere utilizzata per un ragionamento Fuzzy utilizzando delle variabili fuzzy intermedie.

Infatti la "fuzzificazione" di una grandezza mediante le Funzioni Membro di Mf (IsX...) produce della variabili di tipo Fv che sono delle variabili fuzzy (valore da 0 a 100) che possono essere combinate fra loro con gli operatori logici: &&,||,!

La libreria **Fuzzy**

Come si è detto la libreria Fuzzy lavora con numeri interi per velocizzare l'elaborazione. Per cui, per esempio, il valore di verità è espresso in percentuale (0-100).
La libreria è composta dai seguenti elementi:

- Classi c++ che definiscono i fuzzy set. Ovvero le classi logiche in cui sono suddivise le variabili reali.
- Il tipo variabile fuzzy (con valore di verità in percentuale)
- Funzioni di appartenenza che da un valore reale restituiscono una variabile fuzzy (con il suo grado di appartenenza ovvero di verità).
- Overload degli operatori && (And), || (Or), ! (Not) per poterli applicare alle variabili fuzzy.
- Il tipo variabile di output. Con la relativa funzione di "defuzzificazione".
- La funzione che definisce una regola del tipo If Then.

Le classi c++ che definiscono i fuzzy set sono del tipo:
Mfx(v1,v2,..)
Dove x è il numero di set (2,3,5,7) . Per ogni set è previsto un parametro intero che definisce il vertice del triangolo. Infatti le funzioni di appartenenza sono tutte dei triangoli, parzialmente sovrapposti.
Es.: Mf5 mf(-150,-100,0,100,150) rappresenta un fuzzy set di 5 classi (**Fig. 24**)

Come si vede nell'esempio, i triangoli non devono essere per forza isosceli, ma saranno comunque sempre parzialmente sovrapposti. Nel caso di Mf5 i set si chiamano XN2,XN1,XZ,XP1,XP2.
Altre definizioni di set:
Mf2(XN,XP)
Mf3(XN,XZ,XP)
Mf7(XN3,XN2,XN1,XZ,XP1,XP2,XP3)

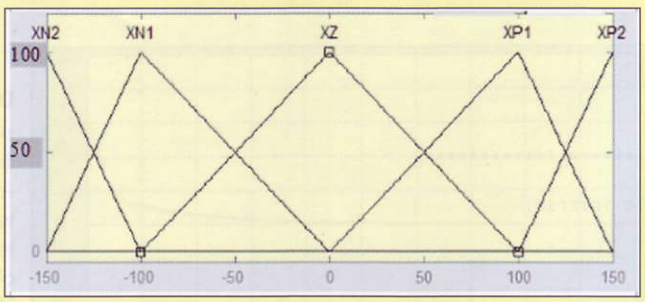


***Fig. 24** - Esempio di definizione di un set di cinque classi.*

Per comodità sono state inserite definizioni di set positivi, identici ai corrispondenti di doppio segno, ma con i nomi dei set più coerenti alla realtà di un dominio tutto positivo.
MfP3(XZ,XP1,XP2)
MfP5(XZ,XP1,XP2,XP3)

Ognuna di queste classi contiene funzioni di appartenenza il cui nome fa riferimento al set. Per esempio per Mf5 abbiamo disponibili le seguente funzioni di appartenenza:

IsXN1(x), IsXN2(x), IsXZ(x), IsXP1(x), IsXP2(x)

Queste funzioni restituiscono una variabile fuzzy rappresentata dalla classe Fv di libreria.
La generica regola è implementata dalla funzione:

IfThen(espressionelogica,variabileout,valoresingleton)

Ovvero: il valore di verità dell'espressione logica è applicato come peso al valore (intero) singleton ed il risultato è sommato alla variabile di output. Bisogna aver, quindi, definito in precedenza una variabile tramite la classe Out di libreria. (Ovviamente le variabili di out possono essere più di una)
Es.:
Out y;
IfThen(IsXN1(x1)&&IsXZ(x2),y,128);

Alla fine dell'elenco di regole, per tirare le somme, va lanciata sull'oggetto y (di tipo Out) la funzione Defuzzy().
Es.:
int yout=y.Defuzzy();

Questa funzione applica la media pesata dei singleton (metodo

Es.:
Fv a1=af.IsXZ(x1);
Fv a2=af.IsXZ(x2);
Fv a3=af.IsXP(x3);
Fv a4=a1 && a2 && !a3;

se x1=35, x2= -10, x3= -5

allora il valore
di verità di
a4 (a4.v) è:
a4.v=61%

Nel **Listato 1** è mostrato un esempio di utilizzo della libreria Fuzzy che realizza un controller per un pendolo inverso, simulato con un modello matematico semplificato, senza attriti. Il programma completo lo si può trovare come sketch incluso nella libreria con nome: "FuzzyTest1". Nel **Listato 2** è mostrato invece lo stesso controller che utilizza direttamente la funzione *FuzzyController(...)* contenuta nella libreria.
Il programma completo lo si può trovare come sketch incluso nella libreria con nome: "FuzzyTest2".

Il software di libreria (con le istruzioni per il suo utilizzo) può essere scaricato dal nostro sito www.elettronicain.it/

```
Mf3 af(-90,0,90);          //definizioni
Out y;
while(true)                //ciclo di inferenza
{
  //dati i valori di a e da
  IfThen(af.IsXN(a)&&af.IsXN(da),y,-100);
  IfThen(af.IsXN(a)&&af.IsXZ(da),y,-60);
  IfThen(af.IsXN(a)&&af.IsXP(da),y,-30);
                :
  float yout=y.Defuzzy(); //output utile
}
```

***Fig. 25** - Esempio di processo di inferenza.*

Sugeno) e resetta la variabile di output per una nuova inferenza.
Da notare che con questa libreria non c'è bisogno di definire preventivamente un set di singleton, in quanto ad ogni regola può essere passato come parametro un qualunque intero come valore singleton.
In **Fig. 25** è rappresentato un esempio di processo complessivo.
Nella libreria sono inoltre predisposti 2 tipi di fuzzy controller completi: uno con 3 set e uno con 5 set .
Essi utilizzano due variabili, uno per l'errore ed uno per la sua variazione (o la derivata). È necessario considerare eventuali fattori di scala (moltiplicatori) in particolare per la variazione dell'errore

```
Mf3 ef(-90,0,90);              //Definisce il fuzzy set
int mtx[3][3]={{-100,-80,-40}, //Definisce la matrice
               {-40,0,40},     //dei singleton
               {40,80,100}};
FuzzyControllerMf3 FzC(ef,mtx,0);//definisce il contr.
while(true)
{
  ...data una funzione esterna che fornisce e e de
  e=e*k1;de=de*k2;
  f=FzC.Inference(e,de);       //f: output utile
}
```

***Fig. 26** - Esempio di utilizzo di un controller fuzzy di libreria.*

***Fig. 27** Matrice dei valori singleton passata al controller.*

E \ dE	XN	XZ	XP
XN	-100	-80	-40
XZ	-40	0	40
XP	40	80	100

(o per la derivata). In quanto ambedue le grandezze utilizzano lo stesso fuzzy set.
L'utilizzo di queste strutture prevede solo due fasi:

- La definizione, con il passaggio del fuzzy set e della matrice dei singleton.
- L'esecuzione ripetuta della funzione "Inference".

In **Fig. 26** è rappresentato un esempio di utilizzo di un controller fuzzy con tre set per gli input.
La matrice dei singleton è la matrice dei valori per classe di errore (riga) e classe di variazione errore (colonna), come probabilmente appare più chiaramente nella **Fig. 27**.
Il terzo parametro nella definizione del controller è un moltiplicatore per il funzionamento incrementale. Se questo valore è zero non viene restituito nessuna frazione di azione incrementale. Il calcolo del valore incrementale utilizza un float , ma se è 0 non viene effettuata nessuna moltiplicazione in virgola mobile.
L'utilizzo del contributo incrementale può permettere un eventuale azzeramento dell'errore a regime, ma introduce delle possibili oscillazioni.
Questa libreria può essere utilizzata tranquillamente su hardware diverso da Arduino, come qualunque altro tipo di microcontroller. Infatti pesa pochissimo ed è abbastanza veloce. Da alcuni test è risultata una velocità di circa
0.3 millisecondi
per un ciclo di inferenza di 9 regole più funzione di "defuzzificazione" (su Arduino Uno)
Ovvero circa **30 microsecondi** per regola con AND fra due funzioni di appartenenza.
La libreria va posizionata (con la sua cartella) nella cartella "libraries" dell'ambiente di sviluppo di Arduino e contiene anche un paio di esempi e il file per l'evidenziazione delle keyword.