Lecture 34

April 20, 2012

Introduction to Fuzzy Logic

PID Motor Control

How do you control the speed of a motor? Let S_D be the desired speed, S_M be the measured speed, and PWM be the PWM value. Define the error as

$$e(t) = S_D - S_M$$

1. Proportional control:

$$\text{PWM}_P = k_P e(t)$$

2. Integral control:

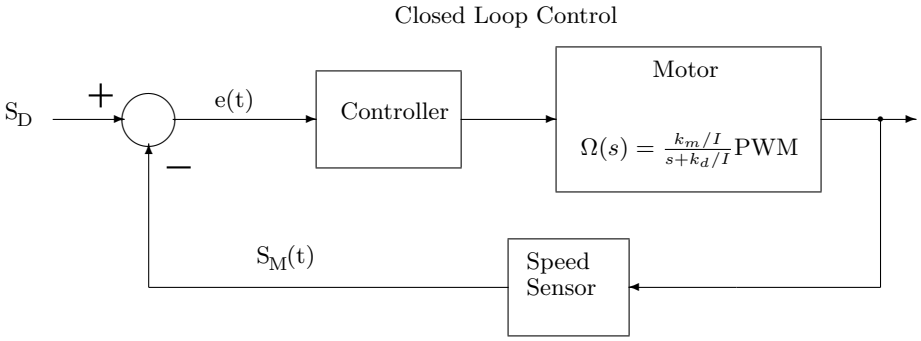
$$\text{PWM}_I = k_I \int_0^t e(\tau) d\tau$$

3. Differential control:

$$\text{PWM}_D = k_D \frac{de(t)}{dt}$$

Then $\text{PWM} = \text{PWM}_P + \text{PWM}_I + \text{PWM}_D$

To design a controller, you need to find k_P, k_I and k_D such that motor responds in a "good" way. Control theory shows how to find "good" values for k_P, k_I and k_D, based on the characteristics of the system. (In the figure below, k_m and k_d are parameters of the motor; they determine how a motor responds to an input current I.)



Type	Controller
Proportional	k_P
Integral	k_I/s
Differential	sk_D

k_d is the viscous friction of the motor

k_m relates torque to PWM

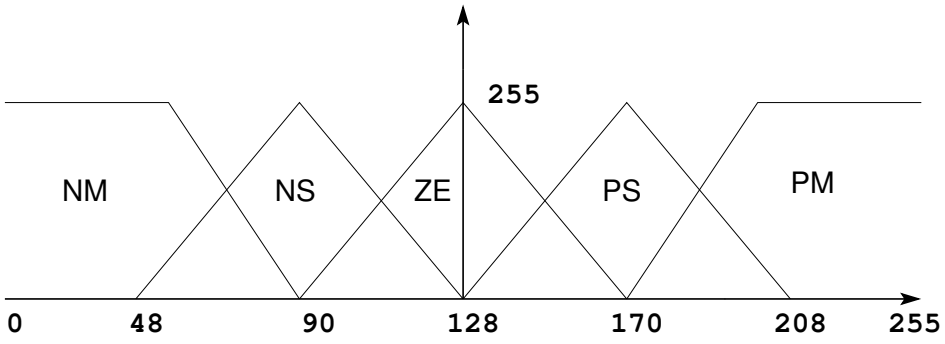
Fuzzy Logic Motor Control

Traditional control theory requires one to make a complex mathematical model of the plant to be controlled, then mathematically analyze the closed-loop control system to find a "good" controller. Newer control methods have been developed which try to control a system in an intuitive way, like a human or an animal might. Three methods often used are `neural networks`, `genetic algorithms` and `fuzzy logic`. The MC9S12 has instructions specifically for fuzzy logic control.

References: `http://focus.ti.com/lit/an/slaa235/slaa235.pdf`, `http://www.fuzzy-logic.com/ch3.htm`

To implement fuzzy logic control in an MC9S12, you need to define the input membership functions, the output membership functions and the rules which relate the output to the input.

- Input membership functions: These are always a trapezoid. You define these in the MC9S12 by listing a leftmost value, the rightmost value, the left slope and the right slope. In the example from TI, the range of input values is from -0xC00 to +0xC00. For the MC9S12, the range of input values must be an eight-bit unsigned number from 0x00 to 0xFF. In the example from TI, the membership value is a number from 0x000 (input is not a member) to 0x400 (membership is maximum). For the MC9S12, the membership value is an unsigned character, from 0 to 0xFF. Thus, we must redraw the membership plot to look like this:



For the zero (ZE) membership function, the left value is 90, the right value is 170, the left slope 6 (255/(128-90)), and the right slope is 6. Something on one of the ends of the x-axis has a slope of zero. The membership functions for the example from TI would be defined as:

```
; Fuzzy input membership function definitions for speed error
E_Pos_Medium:     DC.B    170, 255,   6,   0
E_Pos_Small:      DC.B    128, 208,   6,   6
E_Zero:           DC.B     90, 170,   6,   6
E_Neg_Small:      DC.B     48, 128,   6,   6
```

```
E_Neg_Medium:     DC.B      0,  80,   0,   6

; Fuzzy input membership function definitions for differential speed error
dE_Pos_Medium:     DC.B   170, 255,   6,   0
dE_Pos_Small:      DC.B   128, 208,   6,   6
dE_Zero:           DC.B    90, 170,   6,   6
dE_Neg_Small:      DC.B    48, 128,   6,   6
dE_Neg_Medium:     DC.B     0,  80,   0,   6
```

- Output membership functions: One value for each, the output value to achieve the desired result. The output is the value we add to the PWM to achieve the desired result. The example from TI used positive and negative numbers. For the MC9S12, the output membership functions need to be unsigned 8-bit numbers. We will use the mid-point (0x80, or 128_{10}) to represent zero change. Numbers higher than 128 will represent a increase in duty cycle, and numbers less than 128 will represent a decrease in duty cycle. The change in duty cycle can be found by subtracting 128 from the final answer.

```
; Fuzzy output membership function definition
PM_Output:         DC.B    192
PS_Output:         DC.B    160
ZE_Output:         DC.B    128
NS_Output:         DC.B     96
NM_Output:         DC.B     64
```

- Rules: Let a unique number represent each of the input memberships and each of the output memberships. List the input combinations, followed by the marker `$FE$`, then list the output value followed by the marker `$FE`. After the final rule, use the marker `$FF` instead of `$FE`. Here are the rules for the example from TI:

```
; Offset values for input and output membership functions
; Used for defining the rules
E_PM               EQU         0  ; Positive medium error
E_PS               EQU         1  ; Positive small error
E_ZE               EQU         2  ; Zero error
E_NS               EQU         3  ; Negative small error
E_NM               EQU         4  ; Negative medium error
dE_PM              EQU         5  ; Positive medium differential error
dE_PS              EQU         6  ; Positive small differential error
dE_ZE              EQU         7  ; Zero differential error
dE_NS              EQU         8  ; Negative small differential error
dE_NM              EQU         9  ; Negative medium differential error
O_PM               EQU        10  ; Positive medium output
O_PS               EQU        11  ; Positive small
O_ZE               EQU        12  ; Zero output
O_NS               EQU        13  ; Negative small
```

```
O_NM            EQU       14  ; Negative medium output
MARKER          EQU      $FE  ; Rule separator
END_MARKER      EQU      $FF  ; End of Rule marker

; Rule Definitions
Rule_Start:     DC.B     E_PM,dE_PM,MARKER,O_NM,MARKER
                DC.B     E_PM,dE_PS,MARKER,O_NM,MARKER
                DC.B     E_PM,dE_ZE,MARKER,O_NM,MARKER
                DC.B     E_PM,dE_NS,MARKER,O_NS,MARKER
                DC.B     E_PM,dE_NM,MARKER,O_ZE,MARKER

                DC.B     E_PS,dE_PM,MARKER,O_NM,MARKER
                DC.B     E_PS,dE_PS,MARKER,O_NM,MARKER
                DC.B     E_PS,dE_ZE,MARKER,O_NS,MARKER
                DC.B     E_PS,dE_NS,MARKER,O_ZE,MARKER
                DC.B     E_PS,dE_NM,MARKER,O_PS,MARKER

                DC.B     E_ZE,dE_PM,MARKER,O_NM,MARKER
                DC.B     E_ZE,dE_PS,MARKER,O_NS,MARKER
                DC.B     E_ZE,dE_ZE,MARKER,O_ZE,MARKER
                DC.B     E_ZE,dE_NS,MARKER,O_PS,MARKER
                DC.B     E_ZE,dE_NM,MARKER,O_PM,MARKER

                DC.B     E_NS,dE_PM,MARKER,O_NS,MARKER
                DC.B     E_NS,dE_PS,MARKER,O_ZE,MARKER
                DC.B     E_NS,dE_ZE,MARKER,O_PS,MARKER
                DC.B     E_NS,dE_NS,MARKER,O_PM,MARKER
                DC.B     E_NS,dE_NM,MARKER,O_PM,MARKER

                DC.B     E_NM,dE_PM,MARKER,O_ZE,MARKER
                DC.B     E_NM,dE_PS,MARKER,O_PS,MARKER
                DC.B     E_NM,dE_ZE,MARKER,O_PM,MARKER
                DC.B     E_NM,dE_NS,MARKER,O_PM,MARKER
                DC.B     E_NM,dE_NM,MARKER,O_PM,END_MARKER
```

- Fuzzification: Set aside memory to store how much the current measurement fits into the membership functions. Load the X register with the address of the start of the `ERROR` membership functions, the Y register with the start of the `ERROR` membership results, and the A register with the value of the `ERROR` measurement. Then execute the `MEM` instruction five times (for the five membership functions). Repeat for the `D_ERROR` measurement:

```
; Locations for the fuzzy input membership values
I_E_PM          DS.B     1
I_E_PS          DS.B     1
I_E_ZE          DS.B     1
```

```
I_E_NS          DS.B    1
I_E_NM          DS.B    1

I_dE_PM         DS.B    1
I_dE_PS         DS.B    1
I_dE_ZE         DS.B    1
I_dE_NS         DS.B    1
I_dE_NM         DS.B    1

; Output fuzzy membership values - initialize to zero
M_PM            DC.B    0
M_PS            DC.B    0
M_ZE            DC.B    0
M_NS            DC.B    0
M_NM            DC.B    0

; Fuzzification
                LDX     #E_Pos_Medium ; Start of Input Mem func
                LDY     #I_E_PM       ; Start of Fuzzy Mem values
                LDAA    ERROR         ; Get ERROR value
                LDAB    #5            ; Number of iterations
Loop_E:         MEM                   ; Assign mem value
                DBNE    B,Loop_E      ; Do all five iterations
                LDAA    D_ERROR       ; Get D_ERROR value
                LDAB    #5            ; Number of iterations
Loop_dE:        MEM                   ; Assign mem value
                DBNE    B,Loop_dE     ; Do all five iterations
```

- Rule Evaluation: Assign truth values to fuzzy output membership functions based on the truth values for fuzzy input membership functions. This is done with the MC9S12 instruction `REV` (or `REVW`). To use the `REV` instruction, load the X register with the address of the first 8-bit element in the rules list, load the Y register with the base address of the fuzzy inputs and fuzzy outputs, put `$FF` into A, and clear the V bit of the CCR. (The last two are accomplished with the `LDAA $FF` instruction). Also, the output membership values need to be zero before executing the `REV` instruction. Here is MC9S12 code for rule evaluation:

```
; Process rules
            LDX     #M_PM         ; Clear output membership values
            LDAB    #5
Loopc:      CLR     1,X+
            DBNE    B,Loopc

            LDX     #Rule_Start   ; Address of rule list -> X
            LDY     #I_E_PM       ; Address of input membership list -> Y
            LDAA    #$FF          ; FF -> A, clear V bit of CCR
            REV                   ; Rule evaluation
```

- Defuzzification: Calculate the output signal from the results of the rule evaluation:

$$OutputSignal = \frac{\sum_{i=1}^{N} S_i F_i}{\sum_{i=1} N F_i}$$

where S_i is the output value for the i^{th} output function. For this example, S_i = (192, 150, 128, 96, 64). F_i is i^{th} value from the rule evaluation. The MC9S12 instruction `WAV` (for Weighted AVerage) calculates the two sums. Load the X register with the base address of the output membership functions and load the Y register with the base address of the output membership values. The `WAV` instruction puts the 32-bit numerator into the `{Y:D}` registers, and the 16-bit denominator into the `X` register. To do the division, use the `EDIV` instruction. The quotient will be in the Y register, and the remainder will be in the D register. Transfer the contents of the Y register to the D register. Then the value to add to the PWM will be in the B register.

```
; Defuzzification
              LDX       #PM_Output     ; Address of output functions -> X
              LDY       #M_PM          ; Address of output membership values -> Y
              LDAB      #$05           ; Number of iterations
              WAV                      ; Defuzzify
              EDIV                     ; Divide
              TFR       Y,D            ; Quotient to D; B now from 0 to 255
              SUBB      #128           ; Subtract offset
              STAB      PWM_Change     ; Save answer
```

Here is the fuzzy logic program:

```
                ORG       $1000

; Offset values for input and output membership functions
; Used for defining the rules
E_PM            EQU         0  ; Positive medium error
E_PS            EQU         1  ; Positive small error
E_ZE            EQU         2  ; Zero error
E_NS            EQU         3  ; Negative small error
E_NM            EQU         4  ; Negative medium error
dE_PM           EQU         5  ; Positive medium differential error
dE_PS           EQU         6  ; Positive small differential error
dE_ZE           EQU         7  ; Zero differential error
dE_NS           EQU         8  ; Negative small differential error
dE_NM           EQU         9  ; Negative medium differential error
O_PM            EQU        10  ; Positive medium output
O_PS            EQU        11  ; Positive small
O_ZE            EQU        12  ; Zero output
O_NS            EQU        13  ; Negative small
O_NM            EQU        14  ; Negative medium output
MARKER          EQU       $FE  ; Rule separator
END_MARKER      EQU       $FF  ; End of Rule marker

ERROR:          DS.B      1    ; Space for speed error value
d_ERROR:        DS.B      1    ; Space for differential speed error value
PWM_Change:     DS.B      1    ; Space for change in PWM value

; Fuzzy input membership function definitions for speed error
E_Pos_Medium:     DC.B    170, 255,   6,   0
E_Pos_Small:      DC.B    128, 208,   6,   6
E_Zero:           DC.B     90, 170,   6,   6
E_Neg_Small:      DC.B     48, 128,   6,   6
E_Neg_Medium:     DC.B      0,  80,   0,   6

; Fuzzy input membership function definitions for differential speed error
dE_Pos_Medium:     DC.B   170, 255,   6,   0
dE_Pos_Small:      DC.B   128, 208,   6,   6
dE_Zero:           DC.B    90, 170,   6,   6
dE_Neg_Small:      DC.B    48, 128,   6,   6
dE_Neg_Medium:     DC.B     0,  80,   0,   6

; Fuzzy output membership function definition
PM_Output:        DC.B    192
PS_Output:        DC.B    160
ZE_Output:        DC.B    128
```

```
NS_Output:        DC.B     96
NM_Output:        DC.B     64

; Locations for the fuzzy input membership values
I_E_PM            DS.B      1
I_E_PS            DS.B      1
I_E_ZE            DS.B      1
I_E_NS            DS.B      1
I_E_NM            DS.B      1

I_dE_PM           DS.B      1
I_dE_PS           DS.B      1
I_dE_ZE           DS.B      1
I_dE_NS           DS.B      1
I_dE_NM           DS.B      1

; Output fuzzy membership values - initialize to zero
M_PM              DC.B      0
M_PS              DC.B      0
M_ZE              DC.B      0
M_NS              DC.B      0
M_NM              DC.B      0

; Rule Definitions
Rule_Start:       DC.B      E_PM,dE_PM,MARKER,O_NM,MARKER
                  DC.B      E_PM,dE_PS,MARKER,O_NM,MARKER
                  DC.B      E_PM,dE_ZE,MARKER,O_NM,MARKER
                  DC.B      E_PM,dE_NS,MARKER,O_NS,MARKER
                  DC.B      E_PM,dE_NM,MARKER,O_ZE,MARKER

                  DC.B      E_PS,dE_PM,MARKER,O_NM,MARKER
                  DC.B      E_PS,dE_PS,MARKER,O_NM,MARKER
                  DC.B      E_PS,dE_ZE,MARKER,O_NS,MARKER
                  DC.B      E_PS,dE_NS,MARKER,O_ZE,MARKER
                  DC.B      E_PS,dE_NM,MARKER,O_PS,MARKER

                  DC.B      E_ZE,dE_PM,MARKER,O_NM,MARKER
                  DC.B      E_ZE,dE_PS,MARKER,O_NS,MARKER
                  DC.B      E_ZE,dE_ZE,MARKER,O_ZE,MARKER
                  DC.B      E_ZE,dE_NS,MARKER,O_PS,MARKER
                  DC.B      E_ZE,dE_NM,MARKER,O_PM,MARKER

                  DC.B      E_NS,dE_PM,MARKER,O_NS,MARKER
                  DC.B      E_NS,dE_PS,MARKER,O_ZE,MARKER
                  DC.B      E_NS,dE_ZE,MARKER,O_PS,MARKER
                  DC.B      E_NS,dE_NS,MARKER,O_PM,MARKER
```

```
                DC.B    E_NS,dE_NM,MARKER,O_PM,MARKER

                DC.B    E_NM,dE_PM,MARKER,O_ZE,MARKER
                DC.B    E_NM,dE_PS,MARKER,O_PS,MARKER
                DC.B    E_NM,dE_ZE,MARKER,O_PM,MARKER
                DC.B    E_NM,dE_NS,MARKER,O_PM,MARKER
                DC.B    E_NM,dE_NM,MARKER,O_PM,END_MARKER

; Main program
                ORG     $2000

; Fuzzification
                LDX     #E_Pos_Medium  ; Start of Input Mem func
                LDY     #I_E_PM        ; Start of Fuzzy Mem values
                LDAA    ERROR          ; Get ERROR value
                LDAB    #5             ; Number of iterations
Loop_E:         MEM                    ; Assign mem value
                DBNE    B,Loop_E       ; Do all five iterations
                LDAA    D_ERROR        ; Get D_ERROR value
                LDAB    #5             ; Number of iterations
Loop_dE:        MEM                    ; Assign mem value
                DBNE    B,Loop_dE      ; Do all five iterations

; Process rules
                LDX     #M_PM          ; Clear output membership values
                LDAB    #5
Loopc           CLR     1,X+
                DBNE    B,Loopc

                LDX     #Rule_Start    ; Address of rule list -> X
                LDY     #I_E_PM        ; Address of input membership list -> Y
                LDAA    #$FF           ; FF -> A, clear V bit of CCR
                REV                    ; Rule evaluation

; Defuzzification
                LDX     #PM_Output     ; Address of output functions -> X
                LDY     #M_PM          ; Address of output membership values -> Y
                LDAB    #$05           ; Number of iterations
                WAV                    ; Defuzzify
                EDIV                   ; Divide
                TFR     Y,D            ; Quotient to D; B now from 0 to 255
                SUBB    #128           ; Subtract offset
                STAB    PWM_Change     ; Save answer

                SWI
```