Fuzzy Logic Control

- Simpler than PID controllers and can be more robust.
- \textit{Physical plant} has \textit{real state variables}.
- DAS monitors them and creates \textit{estimated state variables}.
- \textit{Preprocessor} calculates \textit{crisp inputs}.
- \textit{Fuzzification} converts into \textit{input fuzzy membership sets}.
- The \textit{fuzzy rules} calculate \textit{output fuzzy membership sets}.
- \textit{Defuzzification} converts them into \textit{crisp outputs}.
- \textit{Postprocessor} modifies them into more convenient format.
- \textit{Actuator system} affects physical plant based on outputs.

DAC, ADC Fuzzy Controller

- Two control inputs:
  - $S^*$: desired motor speed, rpm
  - $S'$: current estimated motor speed, rpm
- One control output:
  - $N$: digital value that we write to the DAC
- To use 8-bit math, change units to $1000/256 = 3.9$ rpm
  - $T^* = (256 \cdot S^*)/1000$: desired motor speed, 3.9 rpm
  - $T' = (256 \cdot S')/1000$: current motor speed, 3.9 rpm
- Two crisp inputs:
  - $E = T^* - T'$: error in motor speed, 3.9 rpm
  - $D = T'(n) - T'(n - 1)$: change in speed, 3.9 rpm
Subtraction with Overflow/Underflow Checking

```c
char Subtract(unsigned char N, unsigned char M){
    /* returns N-M */
    unsigned int N16,M16;
    int Result16;
    N16=N;  // Promote N,M */
    M16=M;
    Result16=N16-M16;  /* -255Result16+255 */
    if(Result16<-128) Result16 = -128;
    if(Result16>127) Result16 = 127;
    return(Result16);
}
```

Crisp Inputs

```c
unsigned char Ts;  /* Desired Speed */
unsigned char T;  /* Current Speed */
unsigned char Told;  /* Previous Speed */
char D;  /* Change in Speed */
char E;  /* Error in Speed */

void CrispInput(void){
    E=Subtract(Ts,T);
    D=Subtract(T,Told);
    Told=T;  /* Set up Told for next time */
}
```

Fuzzy Membership Sets

- Input fuzzy membership sets:
  - Slow - True if motor is spinning too slow
  - OK - True if motor is spinning at proper speed
  - Fast - True if motor is spinning too fast
  - Up - True if motor speed is getting larger
  - Constant - True if motor speed is remaining the same
  - Down - True if motor speed is getting smaller

- Output fuzzy membership sets:
  - Decrease - True if motor speed should be decreased
  - Same - True if motor speed should remain the same
  - Increase - True if motor speed should be increased

Fuzzification

[Diagram showing fuzzy membership sets for Crisp Input E and Crisp Input D.]
Calculation of the Fuzzy Membership Variables

```c
#define TE 20
unsigned char Fast, OK, Slow, Down, Constant, Up;
#define TD 20
unsigned char Increase, Same, Decrease;
#define TN 20

void InputMembership(void) {
    if (E <= -TE) { /* E <= -TE */
        Fast = 255; OK = 0; Slow = 0;
    } else if (E < 0) { /* -TE < E < 0 */
        Fast = (255 * (-E)) / TE; OK = 255 - Fast; Slow = 0;
    } else if (E < TE) { /* 0 < E < TE */
        Fast = 0; Slow = (255 * E) / TE; OK = 255 - Slow;
    } else { /* +TE <= E */
        Fast = 0; OK = 0; Slow = 255;
    }
}
```

Calculation of the Output Fuzzy Membership

```c
unsigned char min(unsigned char u1, unsigned char u2) {
    if (u1 > u2) return (u2);
    else return (u1);
}

unsigned char max(unsigned char u1, unsigned char u2) {
    if (u1 < u2) return (u2);
    else return (u1);
}

void OutputMembership(void) {
    Same = min(OK, Constant);
    Decrease = min(OK, Up);
    Decrease = max(Decrease, min(Fast, Constant));
    Decrease = max(Decrease, min(Fast, Up));
    Increase = min(OK, Down);
    Increase = max(Increase, min(Slow, Constant));
    Increase = max(Increase, min(Slow, Down));
}
```

Fuzzy Logic Rules

<table>
<thead>
<tr>
<th>E</th>
<th>Down</th>
<th>Constant</th>
<th>Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slow</td>
<td>Increase</td>
<td>Increase</td>
<td></td>
</tr>
<tr>
<td>OK</td>
<td>Increase</td>
<td>Same</td>
<td>Decrease</td>
</tr>
<tr>
<td>Fast</td>
<td>Decrease</td>
<td>Decrease</td>
<td></td>
</tr>
</tbody>
</table>

Calculation of the Fuzzy Membership Var (cont)

```c
if (D <= -TD) { /* D <= -TD */
    Down = 255; Constant = 0; Up = 0;
} else if (D < 0) { /* -TD < D < 0 */
    Down = (255 * (-D)) / TD; Constant = 255 - Down; Up = 0;
} else if (D < TD) { /* 0 < D < TD */
    Down = 0; Up = (255 * D) / TD; Constant = 255 - Up;
} else { /* +TD <= D */
    Down = 0; Constant = 0; Up = 255;
}
```
Slide 13: Defuzzification

<table>
<thead>
<tr>
<th>Fuzzy Membership Value</th>
<th>Crisp Output, $N$ (change in D/A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>255</td>
<td>Definitely true</td>
</tr>
<tr>
<td>Decrease</td>
<td>Definitely false</td>
</tr>
<tr>
<td>Same</td>
<td>Half true</td>
</tr>
<tr>
<td>Increase</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>-128</td>
</tr>
<tr>
<td>-TN</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>+TN</td>
</tr>
<tr>
<td>+127</td>
<td>Increase</td>
</tr>
</tbody>
</table>

Slide 14: Calculation of the Crisp Output

```c
char dN;
void CrispOutput(void){
    dN=(TN*(Increase-Decrease)) / (Decrease+Same+Increase);
}
```

Slide 15: Main Program for Fuzzy Logic Control

```c
unsigned int Time;
#define rate 2000
void Initialize(void){
    OPTION=0x80;    /* Turn on A/D */
    PORTB=0;
    N=0;            /* Initial Actuator */
    Told=0;
    Ts=128;        /* 500 rpm */
    #define CCF 0x80
    unsigned char Sample(unsigned char channel){
        ADCTL=channel; /* Start A/D */
        while((ADCTL&CCF)==0); /* Wait for CCF */
        return(ADR1);
    }
}
```

Slide 16: Main Program for Fuzzy Logic Control (cont)

```c
void Main(void){
    int dT;
    Initialize(); /* Turn on A/D init globals */
    Time=TCNT+rate; /* First TCNT value */
    while(1){
        while((dT=Time-TCNT)>0){}; Time=Time+rate; /* Next TCNT value */
        T=Sample(0); /* Sample A/D and set T */
        CrispInput(); /* Calculate E,D and new Told*/
        InputMembership();/*Fast,OK,Slow,Down,Cons,Up*/
        OutputMembership();/*Inc,Same,Dec*/
        CrispOutput(); /* dN */
        N=max(0,min(N+dN,255));
        PORTB=N;       /* Set Actuator */
    }
}
```