HW #1. FLC Design

- Design a fuzzy logic controller for the system in the source program, to have an output response of rising time 20.0 sec, 5.0 %OS, and 2 % settling time 70.0 sec for the unit step input.

- Submit a report which includes the description on what you have modified in the given source program, along with the FLC source code.

- Compare the performance of the FLC with that of the PID controller.

Due date: Sept. 29, 2019

To: https://klms.kaist.ac.kr

- To compile in Ubuntu
  - Using g++
    ① Open terminal
    ② Go to the flc folder
    ③ Compile the program using g++ with '–std=c+11' option
    $ g++ -std=c++11 main.cpp

  - Using cmake
    ① Open terminal
    ② Go to the flc folder
    ③ Build with cmake
    $ cmake .
    ④ Make
    $ make

main.hpp

- Tune the parameters, **Ke, Kce, Ku** for FLC.
- Tune the parameters, **Kp, Ki, Kd** for PID controller.

```
double Ke = 1.886226, Kce = 1.8875, Ku = 1.051013;
// Ke, Kce, Ku are parameters for FLC.

double Kp = 0.927961, Ki = 0.000024, Kd = 1.130647;
// Kp, Ki, Kd are parameters for PID controller.

double yout = 0.0;   // Initial output
double target = 1.0; // Target value
control.setTarget(target);  // Set a target value in the system

int maxTimeStep = 500;
for (int t=0; t<maxTimeStep; t++) {
//    control.constantK(1);         // with gain controller
//    control.PID(Kp, Ki, Kd);      // with PID controller
      control.FLC(Ke, Kce, Ku);     // with FLC

      yout = control.motor();
      control.delay(yout);
}
```

- **control.hpp**
  - Fuzzy logic controller

```
void FLC(double Ke, double Kce, double Ku, int method=1) {
double error, d_error;
error = Ke*e[0]; d_error = Kce*(e[0]-e[1])/timeStep;

rule[0] = Fuzzy::strength(error, Fuzzy::NB, d_error, Fuzzy::ZO); out[0] = Fuzzy::NB;
rule[1] = Fuzzy::strength(error, Fuzzy::NB, d_error, Fuzzy::PS); out[1] = Fuzzy::NM;
rule[2] = Fuzzy::strength(error, Fuzzy::NM, d_error, Fuzzy::ZO); out[2] = Fuzzy::NM;
...
rule[18]= Fuzzy::strength(error, Fuzzy::PB, d_error, Fuzzy::ZO); out[18]= Fuzzy::PB;

m[0]=Ku*Fuzzy::defuzzy(rule, out, method);
}
```

  - PID controller

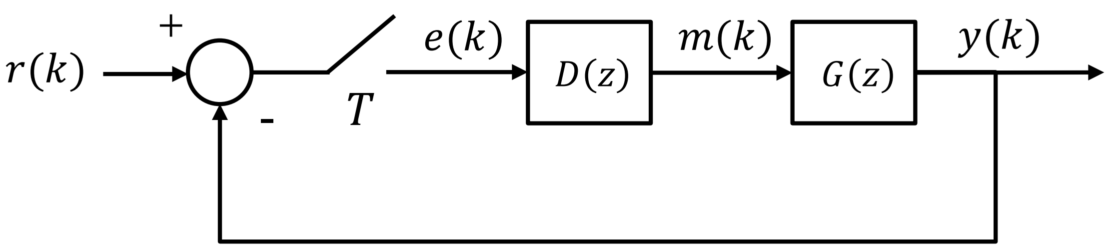
```
void PID(double Kp, double Ki, double Kd) {
    double a = Kp + Ki*timeStep/2.0+Kd/timeStep;
    double b = Ki*timeStep/2.0 - Kp - 2.0*Kd/timeStep;
    double c = Kd/timeStep;
    m[0] = m[1] + a*e[0] + b*e[1] + c*e[2];
}
```

- The nth-order system's difference equation is

$$a[0]y[k] + \cdots + a[n-1]y[k-n+1] = b[0]m[k] + \cdots + b[n-1]m[k-n+1]$$

- System function: $G(z)$

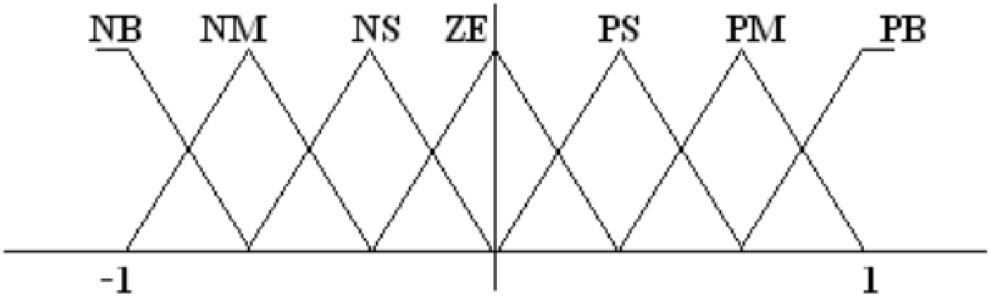
```
double motor( ) {
     double mtotal=0.0,ytotal=0.0;
     for(int i=0;i <order; i++){
         mtotal += b[i]*m[i];
         if(i > 0) ytotal += a[i]*y[i];
     }
    y[0] = mtotal-ytotal;
    return y[0];
}
```



Motor Control Block Diagram

- **fuzzy.hpp**

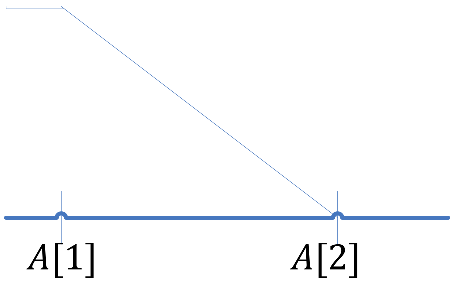
```
// Define Rule and corresponding output.
typedef std::vector<double> Rule;
typedef std::vector<std::vector<double>> Out;

// Define membership functions
 std::vector<double> NB = {-INF,   -1,   -2/3.}; // Define membership functions
 std::vector<double> NM = {-1,   -2/3., -1/3.};  // → Triangular MFs
 std::vector<double> NS = {-2/3., -1/3.,  0};      // (left base value, center value, right base value)
 std::vector<double> ZO = {-1/3.,  0,    1/3.};
 std::vector<double> PS = { 0,    1/3.,  2/3.};
 std::vector<double> PM = { 1/3.,  2/3., 1};
 std::vector<double> PB = { 2/3.,  1,    INF};
```

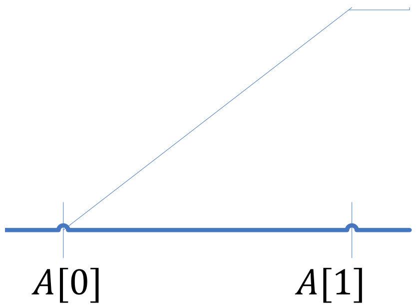


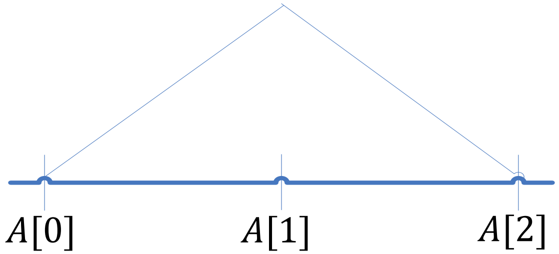
- **fuzzy.hpp**

```
double membership(double x, std::vector<double>& A) {
    double a, b; // y = ax+b

    if (A[0]==-INF && x<A[1])  return 1;
    if (A[2]== INF && x>=A[1]) return 1;

    if (x>=A[0] && x<A[1]) {
        a = 1/(A[1]-A[0]);
        b = -A[0]/(A[1]-A[0]);
        return a*x+b;
    }

    if (x>=A[1] && x<A[2]) {
        a = -1/(A[2]-A[1]);
        b = A[2]/(A[2]-A[1]);
        return a*x+b;
    }
    return 0;
}
```

// 3 types of the membership function







- **fuzzy.hpp**

```
// Defuzzification method 1
case 1:
    for (int i=0; i< rule.size(); i++) {
        tmp1+=rule[i]*out[i][1];
        tmp2+=rule[i];
    }
    if (tmp2==0) return 0;
    return tmp1/tmp2;
```

/* **Height defuzzification** */

$$Z_H = \frac{\sum_{k=1}^{n} c_{(k)} f_k}{\sum_{k=1}^{n} f_k}$$

```
// Defuzzification method 2
case 2:
for (int i=0; i<rule.size(); i++) {
    for (int j=0; j<member.size(); j++) {
        tmp1 = membership(-1.0+2.0*j/(member.size()-1.0),
        out[i]);
        tmp1 = std::max(tmp1, rule[i]);
        member[j] = std::max(member[j], tmp1);
    }
}
tmp1=0.0; tmp2=0.0;
for (int i=0; i<member.size(); i++) {
    tmp1 += member[i]*(-1.0+2.0*i/member.size()-1.0);
    tmp2 += member[i];
}
if (tmp2=0.0) return 0.0;
return tmp1/tmp2;
```

/* **Center Of Area** */

$$Z_{COA} = \frac{\int_z \mu_A(z) z dz}{\int_z \mu_A(z) dz}$$

- Output response