Fuzzy Based PID Motor Control

ASHISH DAGA
EMBEDDED TEAM, KRSSG

Motor Control

Typical Motor Control

1. Direct Voltage
2. Pulsating Voltage

Pulsating Voltage has a lot of advantages over Direct Voltage Application.

Typical Motor Response

On application of a pulsating voltage difference the most usual response of motor speed against voltage applied can be approximated into two linear plots.

For precise control we require feedback.

This feedback can be provided by an encoder. An encoder will provide us with pulses, the total count of which is proportional to the amount of rotation of the motor shaft.

Feedback

We have the actual value of rotation observed corresponding to the applied voltage.

From this feedback it is possible to obtain 3 important Parameters

1. Current Error: Desired Value – Obtained Value

2. Cumulative Error: Sum of all previous errors

3. Differential Error: Current Error – Previous Error

PID Control

The basic equation

Set Value += Kp * Current Error + Ki * Cumulative Error + Kd * Differential Error

The 3 important constants

1. Kp – Proportional

2. Ki – Integral

3. Kd – Derivative

Each form of error has something to contribute as a feedback, these constants can be assumed to be the weights of all these inputs.

PID Control

The Pros

1. Theoretically Stable
2. Easy to formulate

The Cons

1. We are dealing with a discrete time domain, not continuous
2. Value of constants highly dependent on physical conditions

Fuzzy Logic

Fuzzy logic can be intuitively understood by the classical example of hot water – cold water mixing in the shower.

The Algorithm to apply fuzzy logic to any system can be divided into 4 major steps

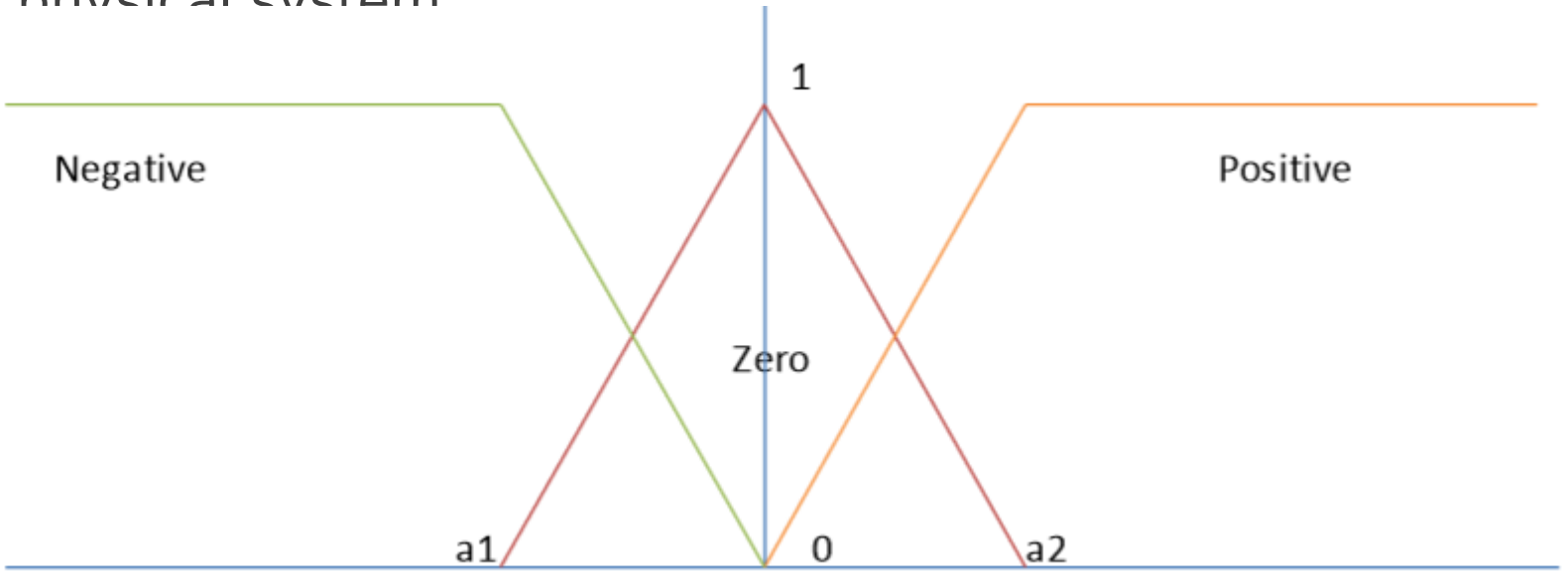
1. Error Fuzzification
2. Fuzzy Matrix Formulation
3. Rule Set Mapping
4. De-fuzzification

Error Fuzzification

1. Decide on no. of fuzzy levels you require for the error

2. Follow an appropriate scheme to implement the same

You may use a linear relationship, trapezoidal, Gaussian etc depending on your physical system



Creating Linguistic Rules

Based on the inputs from the error parameters we decide the relative weightage of an output.

Example

If error is negative and Δerror is negative use Kp medium.

If error is zero and Δerror is negative use Kp medium.

If error is positive and Δerror is negative use Kp large.

Defining the Rule Set Matrix

These rules can be written down in a matrix

Kp/ Δerror error	Negative	Zero	Positive
Negative	Medium	Medium	Large
Zero	Medium	Small	Medium
Positive	Large	Medium	Medium

Kd/ Δerror error	Negative	Zero	Positive
Negative	Medium	Medium	Small
Zero	Large	Small	Large
Positive	Small	Medium	Medium

Implementation of Rule Set

We can choose the weights which need to represented in Matrix form according to suitable physical applications and testing.

The Rule Set may be comprising of a minimum value, mean value(arithmetic or geometric) or any such weighted combination.

The Code for a Minimum Based Approach is

```
Fuzzy_Matrix[i][0][0] = Negative_D_Error[i] < Negative_Error[i] ? Negative_D_Error[i] : Negative_Error[i];
Fuzzy_Matrix[i][0][1] = Zero_D_Error[i] < Negative_Error[i] ? Zero_D_Error[i] : Negative_Error[i];
Fuzzy_Matrix[i][0][2] = Positive_D_Error[i] < Negative_Error[i] ? Positive_D_Error[i] : Negative_Error[i];
Fuzzy_Matrix[i][1][0] = Negative_D_Error[i] < Zero_Error[i] ? Negative_D_Error[i] : Zero_Error[i];
Fuzzy_Matrix[i][1][1] = Zero_D_Error[i] < Zero_Error[i] ? Zero_D_Error[i] : Zero_Error[i];
Fuzzy_Matrix[i][1][2] = Positive_D_Error[i] < Zero_Error[i] ? Positive_D_Error[i] : Zero_Error[i];
Fuzzy_Matrix[i][2][0] = Negative_D_Error[i] < Positive_Error[i] ? Negative_D_Error[i] : Positive_Error[i];
Fuzzy_Matrix[i][2][1] = Zero_D_Error[i] < Positive_Error[i] ? Zero_D_Error[i] : Positive_Error[i];
Fuzzy_Matrix[i][2][2] = Positive_D_Error[i] < Positive_Error[i] ? Positive_D_Error[i] : Positive_Error[i];
```

Rule Set Mapping

Now to finally come up with the actual weights for the constants, we may again use a scheme such as Maximum, RMS value etc to obtain the actual weight for each constant

Shown here is a code section to determine the weight for Kp

```
Kp_Small[i] = Fuzzy_Matrix[i][1][1];

Kp_Medium[i] = Fuzzy_Matrix[i][1][0] > Fuzzy_Matrix[i][0][1] ? Fuzzy_Matrix[i][1][0] : Fuzzy_Matrix[i][0][1];
Kp_Medium[i] = Kp_Medium[i] > Fuzzy_Matrix[i][2][1] ? Kp_Medium[i] : Fuzzy_Matrix[i][2][1];
Kp_Medium[i] = Kp_Medium[i] > Fuzzy_Matrix[i][1][2] ? Kp_Medium[i] : Fuzzy_Matrix[i][1][2];
Kp_Medium[i] = Kp_Medium[i] > Fuzzy_Matrix[i][0][0] ? Kp_Medium[i] : Fuzzy_Matrix[i][0][0];
Kp_Medium[i] = Kp_Medium[i] > Fuzzy_Matrix[i][2][2] ? Kp_Medium[i] : Fuzzy_Matrix[i][2][2];


Kp_Large[i] =  Fuzzy_Matrix[i][0][2] > Fuzzy_Matrix[i][2][0] ? Fuzzy_Matrix[i][0][2] : Fuzzy_Matrix[i][2][0];
```

Defuzzification

We multiply the weights obtained with the guess values for the constant and obtain an actual value of Kp, Kd used for Multiplication.

```
Kp_Multiplier[i] = (Kp_Small[i] * Kp_Small_Value) + (Kp_Medium[i] * Kp_Medium_Value) + (Kp_Large[i] * Kp_Large_Value);
Kd_Multiplier[i] = (Kd_Small[i] * Kd_Small_Value) + (Kd_Medium[i] * Kd_Medium_Value) + (Kd_Large[i] * Kd_Large_Value);

Kp_Divider[i] = (Kp_Small[i] + Kp_Medium[i] + Kp_Large[i]);
Kd_Divider[i] = (Kd_Small[i] + Kd_Medium[i] + Kd_Large[i]);
```

Advantages of Fuzzy Logic

1. It is adaptive
2. Does not require a rigorous analysis
3. High tolerance for error as it is based on everything being “fuzzy”

Thank You