Fuzzy-PID

STM Code implementing Fuzzy Logic for PID to drive motor.

Involves UART Communication between STM32F407VG and FPGA.

Workflow:

- Receives the number of Ticks as input from FPGA.
- Computes the appropriate PWM Value using Fuzzy Logic in STM.
- Transmits the computed PWM to FPGA which in turn drives the Motor.

Abstract

The traditional Mamdani approach of Fuzzy Logic is computationally expensive and highly inefficient when it comes to controlling velocity and direction of motion of bots which need to have fast response as well as acquire accurate target states with minimal error. Added to this, it is necessary to take into consideration the means of communication, which in our case is via a wireless network. In this paper we propose a modified version of the Mamdani approach and then proceed to integrate it with a PID control function in an attempt to achieve better results.

Introduction

DC motors in general do not have a linear rpm-pwm relationship. The reasons for this include friction in the axle and back emf as well as external influences like resistive torque from the load or fluctuating voltages. But more often than not, we would like the motor to run at the same speed or to provide the same torque irrespective of the conditions it is run in or the load it is required to carry. It is imperative that given a target state, the motor may require different PWM values to reach that state subject to different conditions. Therefore, an automated non-linear correction function is essential to provide optimal outputs in any arbitrary condition.

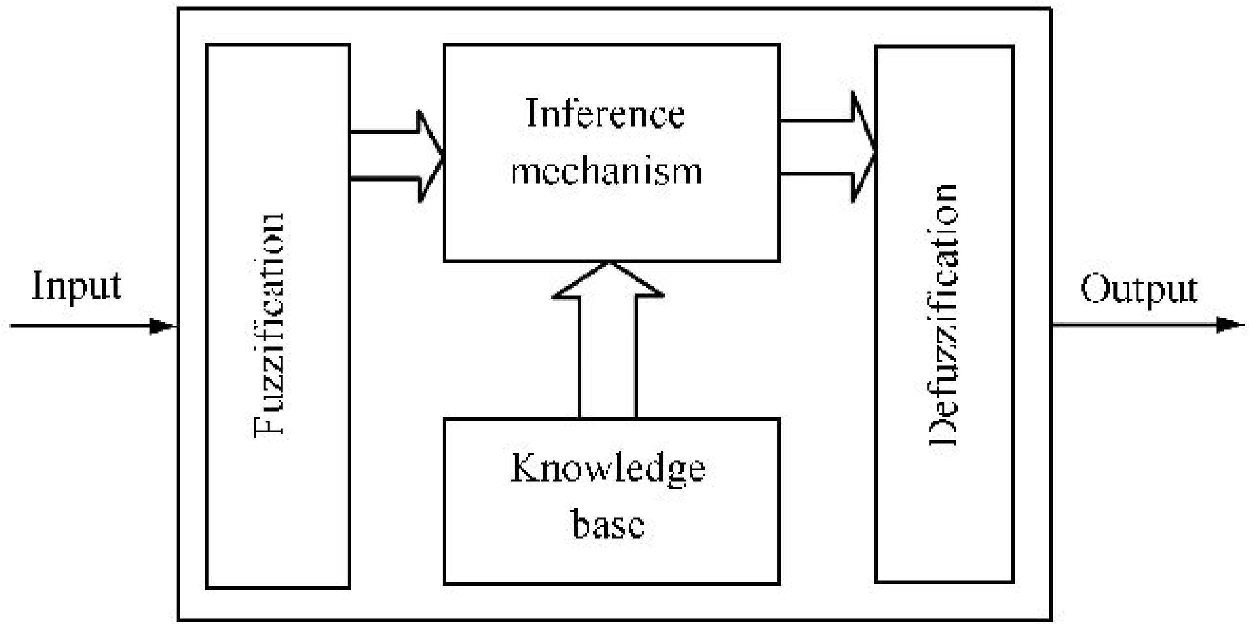
One of the most popular industrial control systems used is the PID control logic.

$$Y(t) = K_p e(t) + \int K_i e(t)dt + K_d de/dt$$

where e(t) is the error at sampling time time t and de is the change in error.
Kp, Ki and Kd can be tuned to provide optimal performance. One widely used method is the Zeigler-Nicholas method.

Unfortunately, tuning the PID controller for best performance is quite time consuming. That is why, an automated control function which bypasses this tuning process and automatically optimizes Kp, Ki and Kd would be very helpful.

In our case, we implemented a PD controller with fuzzy logic for running our motors.



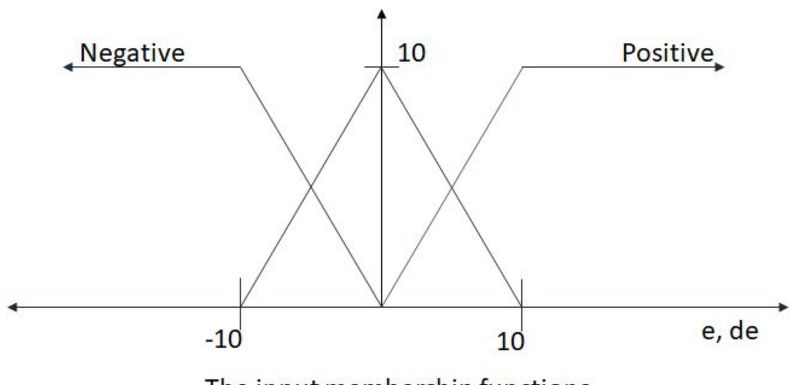
First, the membership functions and linguistic definitions are determined to capture the desired dynamics. Once these are determined, the implementation of the controller is achieved by directly applying existing techniques. However, the determination of these rules and definitions are not obvious for complex systems, but they are critical to the performance of the controller.

As shown above, a general fuzzy logic controller consists of four parts:

- Fuzzification which involves taking inputs from the sensors and fuzzifying them for further processing. The fuzzified sets express measurement uncertainties.
- Knowledge base where the rules are formed using linguistic variables.
- The fuzzified measurements are then used by the inference engine to evaluate the rules stored in the fuzzy rule base and a fuzzified output is determined.
- The fuzzified output is then converted to a single crisp value. This conversion is called defuzzification. The defuzzified values represent the actions to be taken by the FLC in controlling the process.

Our fuzzy controller takes as input the error and change in error and outputs Kp and Kd. These new values are fed into a PD function which computes the required PWM.

Fuzzification:



The input membership functions

PE, ZE, NE, PDE, NDE and ZDE were initialized to zero.
Then the values were updated according to the input membership functions shown above. Here we used 10 as an indication of full membership in order to reduce computational complexity.

Creating the Fuzzy Matrix (Rule Base):

3x3 matrix used:
For simplification, a 3*3 Rule Base Matrix was implemented for each of the Kp Controller and Kd Controller.

	NE	ZE	PE
NDE	min(NDE, NE)	min(NDE, ZE)	min(NDE, PE)
ZDE	min(ZDE, NE)	min(ZDE, ZE)	min(ZDE, PE)
PDE	min(PDE, NE)	min(PDE, ZE)	min(PDE, PE)

Defuzzification:

The Defuzzification includes setting the following parameters:

- Kp_small has significant value if and only if the Zero Error and Change in Zero Error have a significant membership value.
- Similarly, Kp_Large has a significant value if there is a “high Negative Error and High Positive Change in Error” or “high Positive Error and High Negative Change in Error”.
- And the rest of the elements in the fuzzy matrix help in computing the value for Kp_Medium.
- Using similar logic, Kd_small is set when there is “High Negative Error and High Positive Change in Error” or “High Zero Error and High Change in Zero error” or “High Positive Error and High Negative Change in Error”
- Kd_Large has significant value when there is “High Zero Error and High Negative Change in Error” or “High Zero Error and High Positive Change in Error”
- And Kp_Medium is set for the rest of the possible combinations in Error and Change in Error.

Finally, Kp is computed by:

Kp = (Kp_Small*Kp_Small_Val + Kp_Medium*Kp_Medium_Val + Kp_Large*Kp_Large_Val)/(Kp_Small + Kp_Medium + Kp_Large)

and Kd is computed by:

Kd = (Kd_Small*Kd_Small_Val + Kd_Medium*Kd_Medium_Val + Kd_Large*Kd_Large_Val)/(Kd_Small + Kd_Medium + Kd_Large)

PD-Control:

The preliminary PD value is calculated using the general rules.

Fault = error * Kp_computed + d_error * Kd_computed.

After this, the value is added to Expected_PWM, which has already been assigned the previous Fuzzy_PWM output.

We applied bounding values of 0 and 127 to the output PWM in order to ensure that it does not produce any invalid outputs. So if the required supply PWM exceeds 127, the code ensures that the final output PWM is 127. Similarly for 0. The reason why we did this is that we are using 8-bit UART communication for receiving the ticks and in return computing and sending the PWM. The MSB is used for controlling direction, i.e 1 for anticlockwise direction (say) and 0 for clockwise direction. Our Fuzzy_PID only controls speed in one direction. Hence the bounds.

Variable Initializations:

The constants for the proportional, integral and derivative are initialized to some values:
Kp_Small_Val = 1, Kp_Medium_Val = 2, Kp_Large_Val = 3,
Kd_Small_Val = 3, Kd_Medium_Val = 6, Kd_Large_Val = 9;
All other variables are initialized to zero.

Functions Used:

- void Initialise_LED()
    - This is the basic function for the STM's LED initialization
- void Initialise_UART()
    - This function defines the specifications of the UART used in the STM for communication
    - UART4 was used
    - Size of data transmitted was 8 bit words
- void Initialise_NVIC()
    - Interrupt handler
- void Compute_Error()
    - Computes the error and difference of previous error
- void Error_Fuzzification()
    - This function fuzzifies error and d_error into the three linguistic variables NE, ZE and PE and NDE, ZDE and PDE respectively.
- void Create_Fuzzy_Matrix()
    - This function assigns values to the 3x3 fuzzy matrix using the NE, ZE, PE, PDE, ZDE and NDE values computed in the previous function.
- void Defuzzification()
    - This function is responsible for computing the crisp values of Kp and Kd from the created fuzzy matrix.
- volatile int Fuzzy_PID()
    - This function is responsible for calculating the resultant PWM
    - Currently suited for controlling motion in one direction only.
- void UART4_IRQHandler()
    - Receives ticks in the range(0-127) and outputs PWM(0-127)
    - 127 PWM corresponds to full PWM to be supplied to the motor driver.
- main(void)
    - Here the initialization functions are called once
    - First PWM data is sent and the next iteration happens when a value is received by the interrupt handler which in turn calls the fuzzy functions.