A Set of Expert Systems for PID Tuning: Simple Design with Inference Rules

Nelson Campos [1] and Antonio Marcus Nogueira Lima[1] and Saulo Oliveira Dornellas Luiz [1]

***Abstract*— This paper describes inference rules for tuning PID controllers for process control. Fuzzy logic is being applied to determine the controller parameters. The Mamdani method with 6 rules is being used here for tuning the PID controllers. The same method is implemented with the Sugeno Approach. Further, the Clips Expert System is being used to build a model based on rules for Tunning the same System previous modeled with Fuzzy Logic.**

I. INTRODUCTION

The Proportional-Integral-Derivative (PID) Controllers are so far the most widely used in the industry because of its simplicity and large range of operations. There are many techniques for PID design, but in this present work we will constraint our approach for the Ziegler & Nichols method for tuning the coefficients of the controller. [1]

The transfer function of a PID controller has the following form:

$$G_c(s) = K_p + \frac{K_i}{s} + K_d s \tag{1}$$

where Kc, Ki, and Kd are the proportional, integral, and derivative gains, respectively. The equation (1) could be written as:

$$G_c(s) = K_p(1 + \frac{1}{T_i s} + T_d s) \tag{2}$$

Tunning rules would be applied to determine the parameters in the equations (1) and (2). According to [2], the following assertions give information about the PID parameters:

1) Increasing K_c decreases period and vice versa
2) Increasing K_c increases overshoot and vice versa
3) Increasing K_c decreases rise time and vice versa
4) Increasing K_c increases damping and vice versa
5) Decreasing T_i increases overshoot ratio and vice versa
6) Decreasing T_i increases damping and vice versa
7) Decreasing T_i decreases stability and vice versa
8) Increasing T_i decreases overshoot and vice versa
9) Increasing T_d increases stabilty and vice versa
10) Increasing T_d decreases rise time and vice versa.

All the features are self-explanatory except for the damping that is defined as follows:

$$damping = \frac{SecondPeak - FirstValley}{FirstPeak - FirstValley} \tag{3}$$

TABLE I
ZIEGLER & NICHOLS METHOD

Controller	Kp	Ti	Td
P	0.5Ku	Infinity	0
PI	0.45Ku	Pu/1.2	0
PID	0.6Ku	Pu/2	Pu/8

II. FUZZY LOGIC FOR PID TUNNING: MAMDANI METHOD

In order to design a PID Controller based on inference rules, the process of the equation (4) was used as a case of study.

$$H(s) = \frac{e^{-s}}{(10s+1)(10s+1)} \tag{4}$$

This is a second order process with a lag of one second. The closed-loop output of (4) is showed in Figure 1.

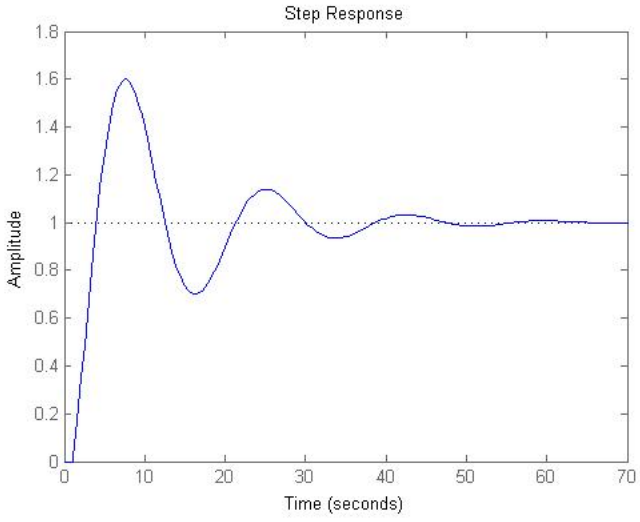

Fig. 1. Closed-loop output of the (4)

A traditional PID controller was used in order to compare the step response in the closed loop of the system with the proportional gain Kc = 15, Ti = 6.75 and Td = 1.69.

Table I shows the Ziegler & Nichols tuning method for the parameters based on the critic gain and the period where the system starts its oscillation.

Figure 2 shows the proposed Fuzzy Gain Scheduler. The approach here takes advantage of the rules 1 to 10 and

some reasoning skills are used to calculate the controller parameters.

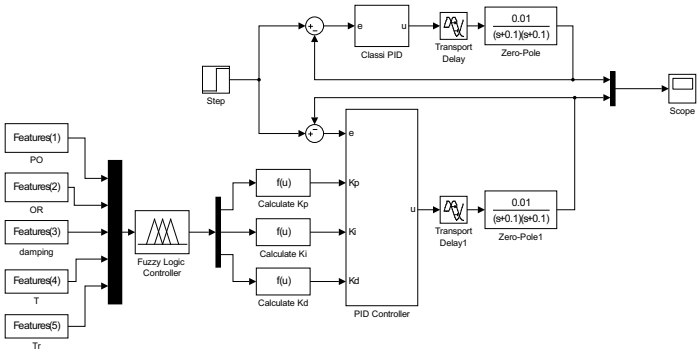

Fig. 2. The proposed Fuzzy Gain Scheduler

Fuzzification is the process of changing a real scalar value into a fuzzy value. This is achieved with the different types of fuzzifiers (membership functions).

Defuzzification is the process of producing a quantifiable result in fuzzy logic, given fuzzy sets and corresponding membership degrees. It is typically needed in fuzzy control systems.

The most commonly method for fuzzy inference is the Mamdani methodology. Mamdani's method was among the first control systems built using fuzzy set theory.

The Mamdani inference expects the output membership functions to be fuzzy sets. After the aggregation process, there is a fuzzy set for each output variable that needs defuzzification.

The controller gain scheduler proposed here is Mamdani-type and has 5 inputs: Percentual of Overshoot (PO), Overshoot Ratio (OR), damping, period (T) and rise time (Tr). These features are extracted from the matlab function calculateFeatures in which expects a transfer function and its step response and returns the desired features. These features are defined in the fuzzy toolbox as gaussian member funcitons with two ranges: Small and Big. As described before, in the Mamdani-type method the outputs are also member functions. The ouputs are the gain parameters of the PID are their member functions are also of gaussian type. The inputs and outputs of the proposed scheduler can be seen in Figure 3. Figure 4 shows the PO member functions and Figure 5 shows output Ti member functions.

The rules 1 to 10 were defined in the Fuzzy Rule Editor. These rules infer the behavior of the output variables in order to the laws dictated by them. Figure 6 shows the Rule Editor of the Fuzzy toolbox.

The rules can also be verified graphically as can be seen in Figure 7. Figures 8 and 9 show the generated surface of

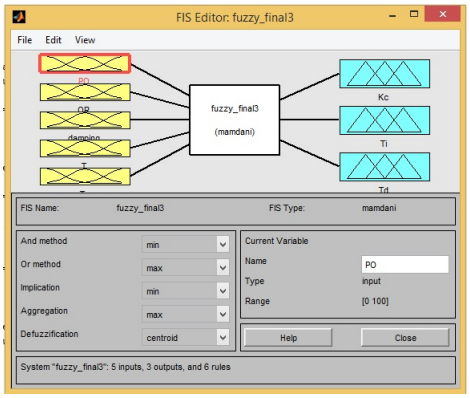

Fig. 3. The Fuzzy Inputs and Outputs

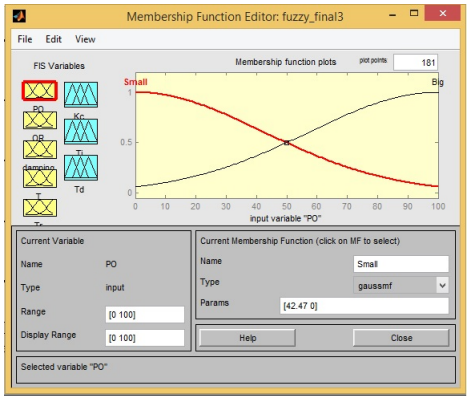

Fig. 4. PO Member Funtions

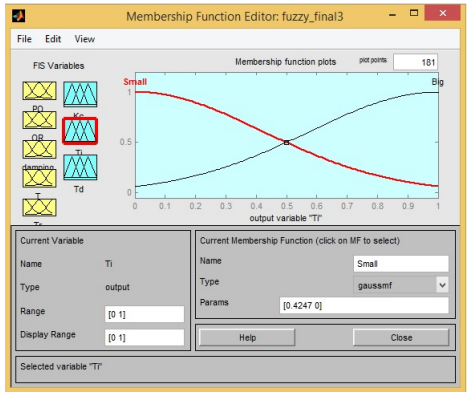

Fig. 5. Ti Member Functions

the gain Kc in function of OR and PO, and the gain Ti in function of the damping and T, respectively.

The result of the two outputs: the classic PID (in yellow) and the Fuzzy Gain Scheduler (in purple) are showed in Figure 10;

The range of the outputs Kc, Ti and Td are defined to be between 0 and 1 in the Fuzzy Toolbox. However it may not cover all the desired scenarios in the PID controller. An adjustment inspired in [3] is done as follows:

$$Kc = (Kc_{max} - Kc_{min})Kc^{fuzzy} + Kc_{min} \qquad (5)$$

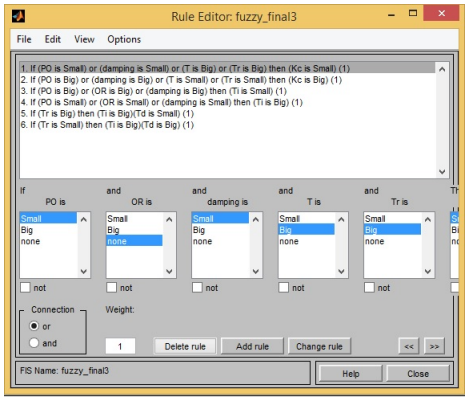

Fig. 6. The Fuzzy Rule Editor

Fig. 7. The Fuzzy Rule Viewer

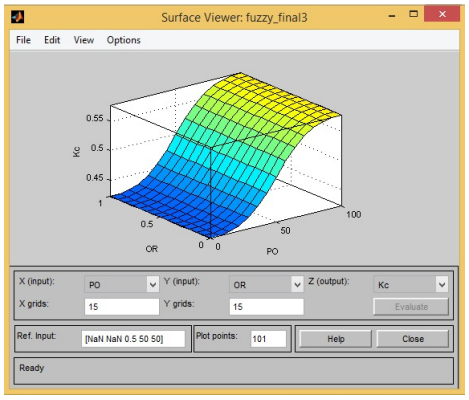

Fig. 8. The Surface Viewer for KcxPOxOR

$$Kd = (Kd_{max} - Kd_{min})Kd^{fuzzy} + Kd_{min} \quad (6)$$

Different of [3], we also define the third output as the integrative gain as follows:

$$Ki = (Ki_{max} - Ki_{min})Ki^{fuzzy} + Ki_{min} \quad (7)$$

The parameters Kc^{fuzzy}, Ki^{fuzzy} and Kd^{fuzzy} are in the range [0,1] and are normalized with equations (5) to (7). The minimum and maximum values of each parameters are constrained with the Table I. For example, according to

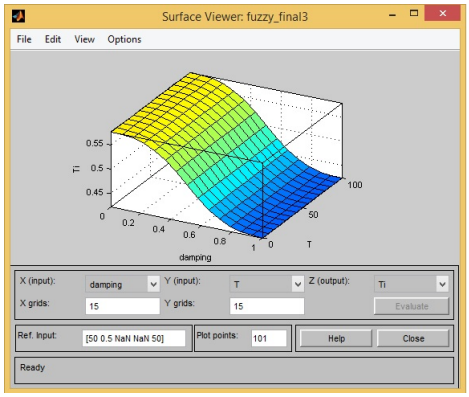

Fig. 9. The Surface Viewer for TixTxdamping

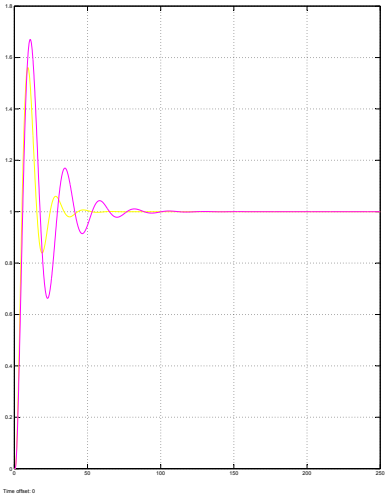

Fig. 10. The two outputs of the system

the Ziegler & Nichols method, $Ki = 2 * Ku/Tu$. Thus, $Ki_{min} = 1.8 * Ku/Tu$ and $Ki_{max} = 2 * Ku/Tu$. This reasoning is analog to ther others parameters.

III. Fuzzy Logic for PID Tunning: Sugeno Method

The Takagi-Sugeno-Kang fuzzy method of inference is similar to the Mamdani method in many respects.The first two parts of the fuzzy inference process, fuzzifying the inputs and applying the fuzzy operator, are the same. The main difference between Mamdani and Sugeno is that the Sugeno output membership functions are either linear or constant. Figure 11 shows the inputs and outputs of the proposed scheduler, now with the Sugeno method of inference.

Similar to the Mamdani type, the Sugeno inputs are defined with a set of member functions as can be seen in Figure 12. However, the ouputs of the Sugeno are discrete points unlike the mamdani as can be seen in Figure 13.

The fuzzy rules of the Sugeno have not changed as showed in Figure 14. Figure 15 shows these rules graphically. Figure 16 shows a surface of the output Kc in function of the features Percentage of Overshoot (PO) and Overshoot Ratio (OR).

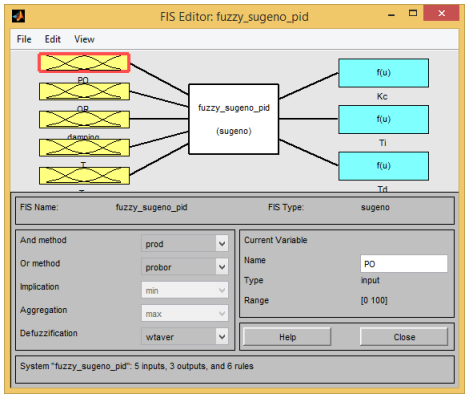

Fig. 11. The Sugeno Inputs and Outputs

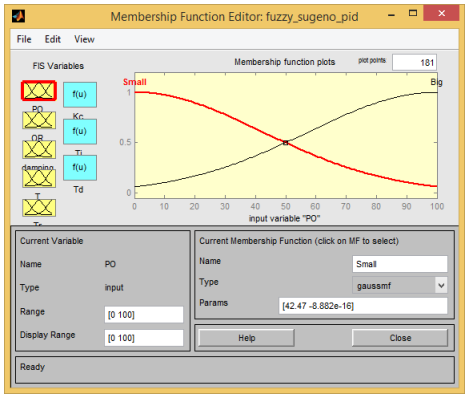

Fig. 12. The Sugeno Inputs Member Functions

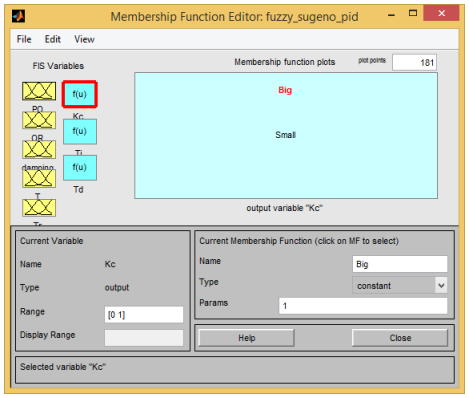

Fig. 13. The Sugeno Output Member Functions

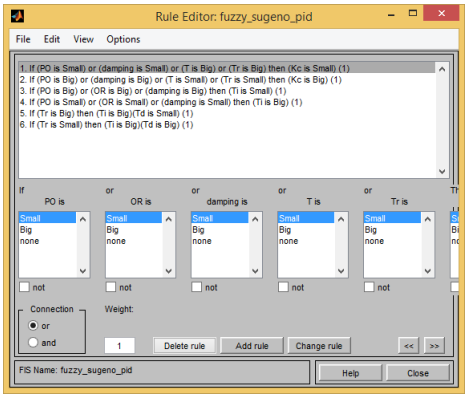

Fig. 14. The Sugeno Fuzzy Rules Editor

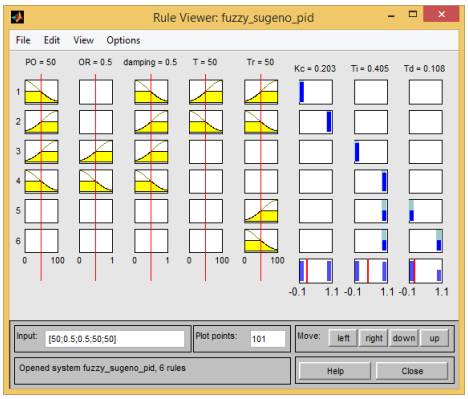

Fig. 15. The Sugeno Fuzzy Rule viewer

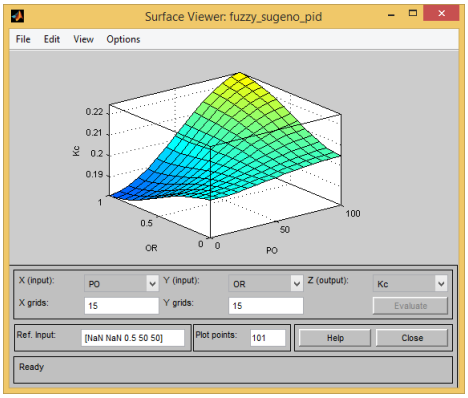

Fig. 16. The Sugeno Surface of KcxPOxOR

The project of the proposed scheduler with the Sugeno inference was implemented with minimal changes compared to the Mamdani type. The output of the closed loop system is showed in Figure 17.

IV. CLIPS: A Tool for Building Expert Systems

An expert system is a program which is specifically intended to model human expertise or knowledge.

Developed at NASA's Johnson Space Center from 1985 to 1996, the C Language Integrated Production System (CLIPS) is a rule-based programming language useful for creating expert systems and other programs where a heuristic solution is easier to implement and maintain than an algorithmic solution [5]. Written in C for portability, CLIPS can be installed and used on a wide variety of platforms. Since 1996, CLIPS has been available as public domain software.

The CLIPS shell provides the basic elements of an expert system:

1) *fact-list* and *instance-list*: Global memory for data
2) *knowledge-base*: Contains all the rules, the rule base
3) *inference engine*: Controls overall execution of rules

Similar to the Fuzzy Approach, the Rules 1 to 10 were described in the CLIPS Environment as it could be seen in the

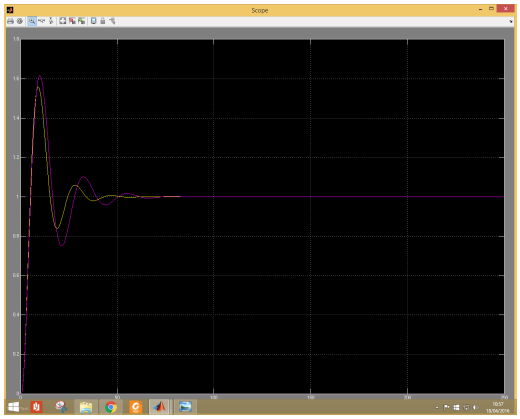

Fig. 17. The Sugeno closed loop output

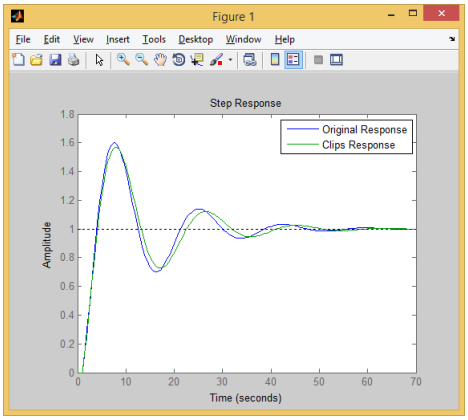

Fig. 19. CLIPS: The Output System

rules.clp source code. These rules should read the Features provided by the function calculateFeatures. To accomplish that, the rule fileOP defined in the file fileOP.clp reads a csv file that provides the features that will be the inputs of the rules. The data coming from the data.csv will set the values of the variables that will adjust the gains of the PID controllers with the rules in the get-gain.clp. To automate all these processes, a batch file run.bat runs all the commands needed for the Expert System Algorithm. Finally, once the gains are defined bu the Clips Environment, these values are exported to the Matlab Workspace and based on the rules, the new PID controller will be defined.

Figure 18 shows the CLIPS Environment and Figure 19 shows the output response of the System that interacts with the CLIPS rules.

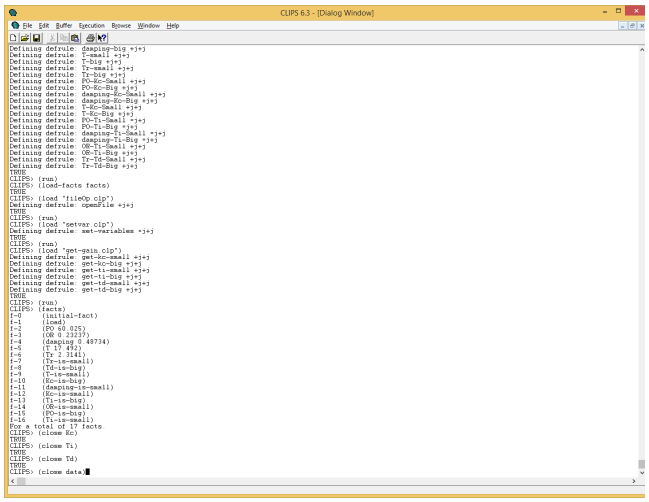

Fig. 18. The CLIPS Environment

V. CONCLUSIONS

In this work was presented a simple Fuzzy PID gain scheduler. Although there are controversies about the Fuzzy logic applications [4], when there is no interest in build analytic and mathematical models to a given process, common sense and some rules plus reasoning skills may be applied to design a feasible application and Fuzzy logic sound good in this situation. Furthermore, the same system implemented with Fuzzy Logic were implemented in a declarative programming language based on C (CLIPS).

VI. APPENDIX

```
function Features = calculateFeatures(F, y, t)
N = size(t,1);
dy = zeros(N,1);
info = stepinfo(F);
zero_cross = 0;
t_cross = 0;
index = 0;


for i=2:N
  dy(i) = y(i)-y(i-1);
end


for i=1:N
    if abs(y(i)-1) < abs(dy(i))/2
        t_cross(i) = t(i);
        index(i) = i;
        zero_cross = zero_cross+1;
    end
end

%Remove all the elements which values are 0
t_cross = t_cross(t_cross ~=0);
index = index(index ~=0);

P1 = max(y);
PO = 100*(P1-1);
riseTime = info.RiseTime;

if size(index,2) == 2 || size(index,2) == 3
    T = NaN;
    P2 = 0;
    OR = 0;
    damping = 0;
    disp('System without oscillation')

else if size(index,2) > 3 % If the system has oscillation
  T = t_cross(3)-t_cross(1);
  t0 = index(1);
  t1 = index(3);
  size(index)
  t2 = index(4);

  V1 = min(y(t0:t1));
  P2 = max(y(t1:t2));
  OR = (P2-1)/(P1-1);
  damping = (P2-V1)/(P1-V1);

  if OR < 1e-6
      disp('System Unstable')
  else disp('System with oscillation')
  end

else
    T = NaN;
    PO = NaN;
    OR = NaN;
    damping = NaN;
    riseTime = NaN;
    disp('Unstable System')
end
end
Features = [PO; OR; damping; T; riseTime];
csvwrite('data.csv', Features);
```

```
;; If PO is Small then Kc is Small
(defrule PO-Kc-Small
(PO-is-small)
=>
(assert (Kc-is-small)))
;; If PO is Big then Kc is Big
(defrule PO-Kc-Big
(PO-is-big)
=>
(assert (Kc-is-big)))
;; If damping is Small then Kc is Small
(defrule damping-Kc-Small
(damping-is-small)
=>
(assert (Kc-is-small)))
;; If damping is Big then Kc is Big
(defrule damping-Kc-Big
(damping-is-big)
=>
(assert (Kc-is-big)))
;; If T is Big then Kc is Small
(defrule T-Kc-Small
(T-is-big)
=>
(assert (Kc-is-small)))
;; If T is Small then Kc is Big
(defrule T-Kc-Big
(T-is-small)
=>
(assert (Kc-is-big)))
;; If PO is Big then Ti is Small
(defrule PO-Ti-Small
(PO-is-big)
=>
(assert (Ti-is-small)))
;; If PO is Small then Ti is Big
(defrule PO-Ti-Big
(PO-is-small)
=>
(assert (Ti-is-big)))
;; If damping is Big then Ti is Small
(defrule damping-Ti-Small

(damping-is-big)
=>
(assert (Ti-is-small)))
;; If damping is Small then Ti is Big
(defrule damping-Ti-Big
(damping-is-small)
=>
(assert (Ti-is-big)))

;; If OR is Big then Ti is Small
(defrule OR-Ti-Small
(OR-is-big)
=>
(assert (Ti-is-small)))

;; If OR is Small then Ti is Big
(defrule OR-Ti-Big
(OR-is-small)
=>
(assert (Ti-is-big)))
;; If Tr is Big then Td is Small
(defrule Tr-Td-Small
(Tr-is-big)
=>
(assert (Td-is-small)))
;; If Tr is Small then Td is Big
(defrule Tr-Td-Big
(Tr-is-small)
=>
(assert (Td-is-big)))
```

```
(defrule openFile
(load)
=>
(open "data.csv" data "r")
(open "Kc.txt" Kc "w")
(open "Ti.txt" Ti "w")
(open "Td.txt" Td "w"))
```

```
(defrule set-variables
(load)
=>
(bind ?PO (read data))
(assert (PO ?PO))
(bind ?OR (read data))
(assert (OR ?OR))
(bind ?damping (read data))
(assert (damping ?damping))
(bind ?T (read data))
(assert (T ?T))
(bind ?Tr (read data))
(assert (Tr ?Tr)))
```

```
(defrule get-kc-small
(Kc-is-small)
=>
(printout Kc 0 crlf))

(defrule get-kc-big
(Kc-is-big)
=>
(printout Kc 1 crlf))

(defrule get-ti-small
(Ti-is-small)
=>
(printout Ti 0 crlf))

(defrule get-ti-big
(Ti-is-big)
=>
(printout Ti 1 crlf))

(defrule get-td-small
(Td-is-small)
=>
(printout Td 0 crlf))

(defrule get-td-big
(Td-is-big)
=>
(printout Td 1 crlf))
```

```
(clear)
(load "rules.clp")
(run)
(load-facts facts)
(load "fileOp.clp")
(run)
(load "setvar.clp")
(run)
(load "get-gain.clp")
(run)
(facts)
(close Kc)
(close Ti)
(close Td)
(close data)
```

```
% Nelson Campos

clear all
close all

s = tf('s');

H = tf(1, [100 20 1], 'InputDelay', 1);

Kc = 15;
Ti = 6.75;
Td = 1.69;

PID = [Kc Ti Td];
Gc = PID(1)*(1+1/(PID(2)*s)+PID(3)*s);

disp('Features = [PO; OR; damping; T; riseTime]')
F = feedback(Gc*H,1);
[y,t] = step(F);

Features = calculateFeatures(F,y,t);

% From the Ziegler-Nichols Tuning Rule
Ku = 25;
Tu = 13.5;

Kpmin = 0.55*Ku;
Kpmax = 0.65*Ku;
Kdmin = 0.070*Ku*Tu;
Kdmax = 0.075*Ku*Tu;
Kimin = 1.15*Ku/Tu;
Kimax = 1.30*Ku/Tu;

Kc_clp = load('Kc.txt');
Ti_clp = load('Ti.txt');
Td_clp = load('Td.txt');

Kc2 = Kc_clp(length(Kc_clp));
Ti2 = Ti_clp(length(Ti_clp));
Td2 = Td_clp(length(Td_clp));

Kp_new = Kpmin + (Kpmax-Kpmin)*Kc2;
Ti_new = Kimin + (Kimax-Kimin)*Ti2;
Td_new = Kdmin + (Kdmax-Kdmin)*Td2;

PID_new = [Kp_new Kp_new/Ti_new Td_new/Kp_new];
Gc_new = PID_new(1)*(1+1/(PID_new(2)*s)+PID_new(3)*s);
F_new = feedback(Gc_new*H, 1);

figure(1), step(F, F_new), legend('Original Response', 'Clips Response')
```

REFERENCES

[1] Ziegler, John G., and Nathaniel B. Nichols. "Optimum settings for automatic controllers." trans. ASME 64.11 (1942).

[2] Litt, Jonathan. ”An expert system to perform on-line controller tuning.” Control Systems, IEEE 11.3 (1991): 18-23.
[3] Zhao, Zhen-Yu, Masayoshi Tomizuka, and Satoru Isaka. ”Fuzzy gain scheduling of PID controllers.” Control Applications, 1992., First IEEE Conference on. IEEE, 1992.
[4] Zadeh, Lotfi A. ”Is there a need for fuzzy logic?.” Information sciences 178.13 (2008): 2751-2779.
[5] Riley, Gary. ”Clips: An expert system building tool.” (1991).