TECHNICAL UNIVERSITY OF ATHENS
DEPARTMENT OF MECHANICAL ENGINEERING
MSC IN AUTONOMOUS CONTROL SYSTEMS AND ROBOTICS

ADVANCED MANUFACTURING SYSTEMS
(CIM - INDUSTRY 4.0)

IMPLEMENTATION OF A FUZZY CONTROL ALGORITHM FOR THE PREDICTION OF TENSION VARIATIONS IN HOT ROLLING PROCESS IN PYTHON

Ronaldo Tsela - 02124203
Fall 2024-2025

Contents

List of Figures

1. Purpose

The purpose of this project is to develop the fuzzy logic system presented by Jong-Yeob Jung and Yong-Taek Im in their study, "*Fuzzy control algorithm for the prediction of tension variations in hot rolling*" [1]. This work forms part of the task in the "*Advanced Manufacturing Systems - (CIM - Industry 4.0)*" lecture series for the MSc program in "*Autonomous Control Systems and Robotics*" at NTUA.

2. Overview

This document outlines the design, usage, and evaluation of the fuzzy logic system introduced in the original paper, *"Fuzzy Control Algorithm for the Prediction of Tension Variations in Hot Rolling"* by Jong-Yeob Jung and Yong-Taek Im [1]. The implementation consists of a Python package designed to predict variations in front and back tension to optimize the hot rolling process of steel strips and achieve fine thickness uniformity at steady-state operation. The program features an easy-to-use API for predicting these values, handling tasks such as input pre-processing, system configuration, and fuzzy logic-based output computation, with minimal manual intervention. The system is evaluated using the same data samples provided in the referenced paper. Additionally, example programs are included as templates for interested users to explore and test the system.

3. Paper Overview

The paper presents the study and development of a controller for the hot rolling manufacturing process using the fuzzy logic theory. Hot rolling involves producing thin steel sheets with uniform thickness by rolling heated steel strips. As the authors note, the quality of the final product primarily depends on achieving uniform thickness distribution. To ensure high-accuracy thickness uniformity, thickness control systems such as *Finishing Mill Set-Up (*referred to as FMU*)* and *Automatic Gauge Control (*referred to as AGC) are typically employed. However, these classical controllers suffer from inaccuracies based on inherent improper modeling, leading to cumulative errors in thickness uniformity. When these issues arise, manual intervention by experienced operators is required. Each adjustment however may fix the uniformity deviations but introduces delays, which degrade overall performance.

To address this problem, the authors propose a fuzzy logic-based algorithm for controlling the process automatically. Specifically, they present a controller scheme that uses the knowledge of an experienced operator through a set of fuzzy logic rules, to automatically adjust the front and back tension. They emphasize that tension is one of the most critical parameters in hot rolling processes that affects the accuracy of the results and by guiding in that way the rolling process the desirable outcome can be achieved. The algorithm is developed based on production data and simulations that model materials using finite elements. Through statistical analysis of production data, they identify the key control variables to be the entry thickness and the carbon equivalent of the input steel strip.

In this project, we focus on implementing this fuzzy logic system, targeting to reproduce the results presented in the study to evaluate the operability of the method. Note that in the paper two fuzzy logic systems are presented, however here only the steady-state prediction algorithm (first part) is implemented since the second part, which consists of modifying the steady-state precomputed values triggered by the tracking error, is not completely described and tested in the paper. Therefore it is really hard to implement that and for this reason it is omitted (probably for future work…).

4. Development Methodology

4.1. The Fuzzy Logic System

In this section, we describe the proposed fuzzy logic system outlined in [1] for predicting the variations in front tension ($\Delta\sigma_f$) and back tension ($\Delta\sigma_b$) (first part). As previously mentioned, the system's input parameters (crisp sets) include the entry thickness of the steel strip (h_{in}) and the carbon equivalent of the material ($C\%$). These inputs are used to infer the variations in front and back tension under steady-state conditions.

First these input parameters are discretized into six and four levels, respectively, as specified in Tables 4 and 7 of the paper. Subsequently these quantized values are fuzzified. To transform these crisp values into fuzzy variables, the system at hand employs triangular membership functions. Therefore each discretized level subsequently corresponds to membership degrees for six and four linguistic variables respectively, as given by Tables 5 and 6 in the paper.

To create the fuzzy control logic between the input and output parameters, 16 rules in total are utilized, as shown in Table 8 of the paper. The rules are formulated in the following general structure:

$$R_i: \; if \; h_{in} \; is \; X^i{}_1 \; and \; C\% \; is \; X^i{}_2 \; then \; \Delta\sigma_f \; is \; Y^i{}_1 \; and \; \Delta\sigma_b \; is \; Y^i{}_2 \quad (1)$$

where $X^i{}_1$ and $X^i{}_2$ represent the linguistic variables for the entry thickness and the carbon equivalent respectively and $Y^i{}_1$ and $Y_2{}^i$ represent the output linguistic variables for the front and back tensions variations respectively for the rule R_i, with $i = 1, 2, \ldots n$ for n applied rules.

The output is classified into 12 singletons, each associated with 12 linguistic variables spanning a range from 9.00% to 33.00%, as given by Table 9 of the paper. During the inference process, the strength of each rule is determined using *Mamdani's* minimum operation, as expressed by the following formula:

$$g^i = min\{X^i{}_1(h_{in}),\; X^i{}_2(C\%)\} \quad (2)$$

where g^i is the weight of the truth value of the i-th rule.

Defuzzification is performed using the center-of-area method, with the output crisp values computed using the following formula:

$$\Delta\sigma_k = \frac{\sum_{i=1}^{n} g^i \cdot Y_k{}^i}{\sum_{i=1}^{n} g^i} \quad (3)$$

where $\Delta\sigma_k$ with $k = f \mid b$ is the real variable that represents the front and back tension respectively.

Finally, the defuzzified values must be expressed as actual values with physical meaning. Although the exact implementation details are not provided in the paper, the process of deriving the actual values for the tension variations is based on class interpolation, as defined by the following formula:

$$[l,\, u] \;=\; (1 - w) \cdot [a_1,\, b_1] \;+\; w \cdot [a_2,\, b_2] \quad (4)$$

Here,$[l, u]$ represents the continuous output range with $l < u$, while $[a_1,\, b_1]$ and $[a_2,\, b_2]$ with $a_i < b_i$, are the two input continuous ranges of values. Moreover these ranges satisfy the conditions $a_1 \leq l \leq a_2$ and $b_1 \leq u \leq b_2$. The weight w is computed using the following equation:

$$w = (c_{12} - c_1) / (c_2 - c_1) \quad (5)$$

where c_{12} is a class index lying between the first class index c_1 and the second class index c_2. Typically, c_{12} is not an integer.

From the resulting range values, the lower value (i.e., the first value in the range) is retained and assigned as the final crisp value result from the fuzzy system. The underlying concept is to create a new class with values that lie between two existing classes. By interpolating the class indices and mapping the corresponding values from the two classes that define the intermediate class, we can derive a range of values characterized by the combined class or index. All the steps of this algorithm as described above in a high-level representation are depicted in **Figure 1**.

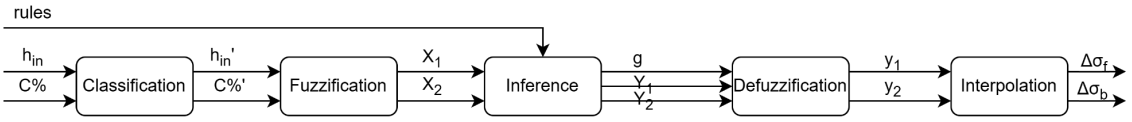


Figure 1. The hot rolling algorithm data flow block diagram.

4.2. Selected Programming Environment

Although existing packages in various programming environments, such as MATLAB and Python, provide extensive support for fuzzy logic systems, this project focuses on implementing the entire fuzzy logic system from scratch as described in the paper. Instead of using ready-made packages or tools (e.g. APIs), in this project the system is built using fundamental data structures. This approach was chosen in order to obtain better understanding of fuzzy logic concepts ("*learn by doing*") and to ensure that each step aligns precisely with the methodology described in the original paper.

Here as programming environment Python is selected and the goal set is the development of a Python package with the task of predicting the crisp values of variations in front and back tension for the manufacturing process of hot rolling given the data samples from the paper.

4.3. Data Structures and User Interface

The whole system is implemented as a Python class *HotRollingFuzzyControl*, which contains methods for implementing the fuzzy logic system as aforementioned.

Continuous input ranges are discretized using constant dictionaries, such as *h_in_range* and *C_range* attributes, which map ranges to numerical strings (e.g., "1", "2" and so on). Similarly, levels are mapped to linguistic variables through membership functions (*h_in_mf*, *C_mf*) and assigned specified membership values and output values are mapped to singleton values and linguistic variables respectively (*d_sigma_range*, *d_sigma_singleton*). Rules are defined as a list (*rules*) of constant dictionaries, formatted as specified by **Equation (1)**.

Input ranges have adjustable parameters for minimum, maximum, and arithmetic accuracy. These can be dynamically set using the following setters member functions:

- *set_h_in_min_val*(): Sets the minimum permissible entry thickness value.
- *set_h_in_max_val*(): Sets the maximum permissible entry thickness value.
- *set_C_perc_min_val*(): Sets the minimum permissible carbon equivalent value.
- *set_C_perc_max_val*(): Sets the maximum permissible carbon equivalent value.
- *set_arithm_accuracy*(): Sets the arithmetic precision (decimal places).

The underlying fuzzy system implements the processes shown in **Figure 1**. Thus users are not required to manually perform these steps. By calling *compute_steady_state*() with the crisp input values for entry thickness and carbon equivalent, the system computes and stores the front and back tension variations (*d_sigma_f*, *d_sigma_b*) within the class (as an attribute). These values can be retrieved using the getter methods *get_d_sigma_f*() and *get_d_sigma_b*().

The results are stored internally rather than directly returned by *compute_steady_state*() to allow for implementation of update rules defined by the second fuzzy system, which could not be completed due to insufficient details in the paper at the moment.

The main internal functions are:

- *classify():* Takes a real input value x and a dictionary R of the form {class: range} and returns the class to which x belongs or *None* if it is outside the permissible range.
- *steady_state_fuzzification():* Converts a crisp input x into a fuzzy representation using a membership function mf and produces a dictionary of the form {linguistic_variable: membership}.
- *steady_state_inference():* Evaluates the defined rules and computes the rule weight using Mamdani's minimum method as given by **Equation (2)** and assigns singleton values to output linguistic variables. Also it performs the actual defuzzification, applying these weights and generating output values for variations in front and back tension based on **Equation (3)**.
- *steady_state_defuzzification():* Although actual defuzzification was made by *steady_state_inference()*, the values produced are not real, and without physical meaning but rather they represent a class or better an index that shows the combination of different classes. This member function processes the fuzzy results from the inference step **Equation (4)** and converts the class/index representation into physically meaningful values using the interpolation scheme given by **Equation (4)**. Here c_{12} is the actual value exported by the defuzzifier, c_1 is the integer part of it and $c_2 = c_1 + 1$.
- *interpolate():*Implements the interpolation logic described by **Equation (4)** and **Equation (5)**.

While these functions can be called outside of *compute_steady_state*(), it is not recommended.

4.4. Package Installation

To keep this package separate from other Python packages on your system, it is best practice to use a virtual environment. To create and activate a virtual environment, on a terminal emulator run within the project folder the following:

```
$ python3 -m venv hot_rolling_fuzzy
$ source ./hot_rolling_fuzzy/bin/activate
```

Navigate to the *./hot_rolling_fuzzy_package* directory and run the following command to install the package locally on your computer (within the virtual environment you just created).

```
$ pip3 install -e .
```

Upon successful installation of the package and the required internal packages needed for the implementation, the package is ready for use.

4.5. Program Set-up and Execution

Using the program is straightforward and intuitive. It comes pre-configured with the data provided in the paper, and the fuzzy system is ready for immediate use. To get started, simply import the package

into your project and instantiate a *HotRollingFuzzyControl()* object. Then, feed your input values h_{in} and $C\%$ into the *.compute()* method. To retrieve the results, call the *.get_d_sigma_f()* and *.get_d_sigma_b()* methods.

The simplest way to use the program is to predict the tension variation ratios for a single configuration, as demonstrated in the following code snippet which serves as a template.

```
# import the package
from hot_rolling_fuzzy_package.hot_rolling_fuzzy import *

 if __name__ == "__main__":

    # Create an object
    ctrl = HotRollingFuzzyControl()

    # Define the crisp input values
    h_in_test = 3.7
    C_test = 10.7

    # Make the prediction
    ctrl.compute_steady_state(h_in_test, C_test)

    # Read the predicted values
    d_sigma_f = ctrl.get_d_sigma_f()
    d_sigma_b = ctrl.get_d_sigma_b()

    # Print the results
    print(f"d_sigma_f = { d_sigma_f }")
    print(f"d_sigma_b = { d_sigma_b }")
```

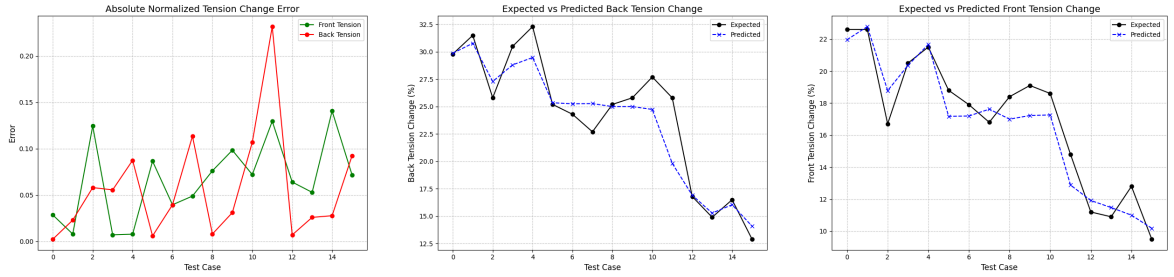
To run the script described earlier, which is located in the deliverables folder (or project repository) as *simple_template.py*, execute the following command in your terminal: *$ python3 simple_template.py*

Using the input values provided in the paper's example (*4. Results and Discussion*)—an entry thickness of 3.7 and a carbon equivalent of 10.7% (10^{-3})—the program produces output values of 17.1% and 25.3%. These results are very close to the values reported in the paper (17.0% and 25.4%, respectively).

5. Experimentation and Results

In addition to *simple_template.py*, another ready-to-use script, *steady_state_evaluation.py*, is included as part of the deliverables. This script is specifically designed to evaluate the functionality of the implementation. It uses the test data from Table 10 in the paper, which includes values for entry thickness and carbon equivalence. These inputs are evaluated against the corresponding variations in front and back tension provided in the same table.

For the evaluation, the script computes the absolute error between the expected and predicted values, normalized against the expected values. The evaluation results are presented in **Figure 2(a)**, while **Figure 2(b)** and **Figure 2(c)** illustrate the predicted and expected values for the variations in front and back tensions, respectively. It is obvious that the designed system serves its purpose.



(a) (b) (c)

Figure 2 - The validation results for the implementation.

6. References

[1] Jong-Yeob Jung, Yong-Taek Im.
Fuzzy control algorithm for the prediction of tension variations in hot rolling.
1998

[2] Franck Dernoncourt.
Introduction to fuzzy logic.
2013

[3] Helios.
Προηγμένα Συστήματα Κατεργασιών (CIM-INDUSTRY 4.0)
Παρουσιάσεις, Σημειώσεις και Παραδείγματα.
2024.

[4] M. Hellmann.
Fuzzy Logic Introduction.
2001.