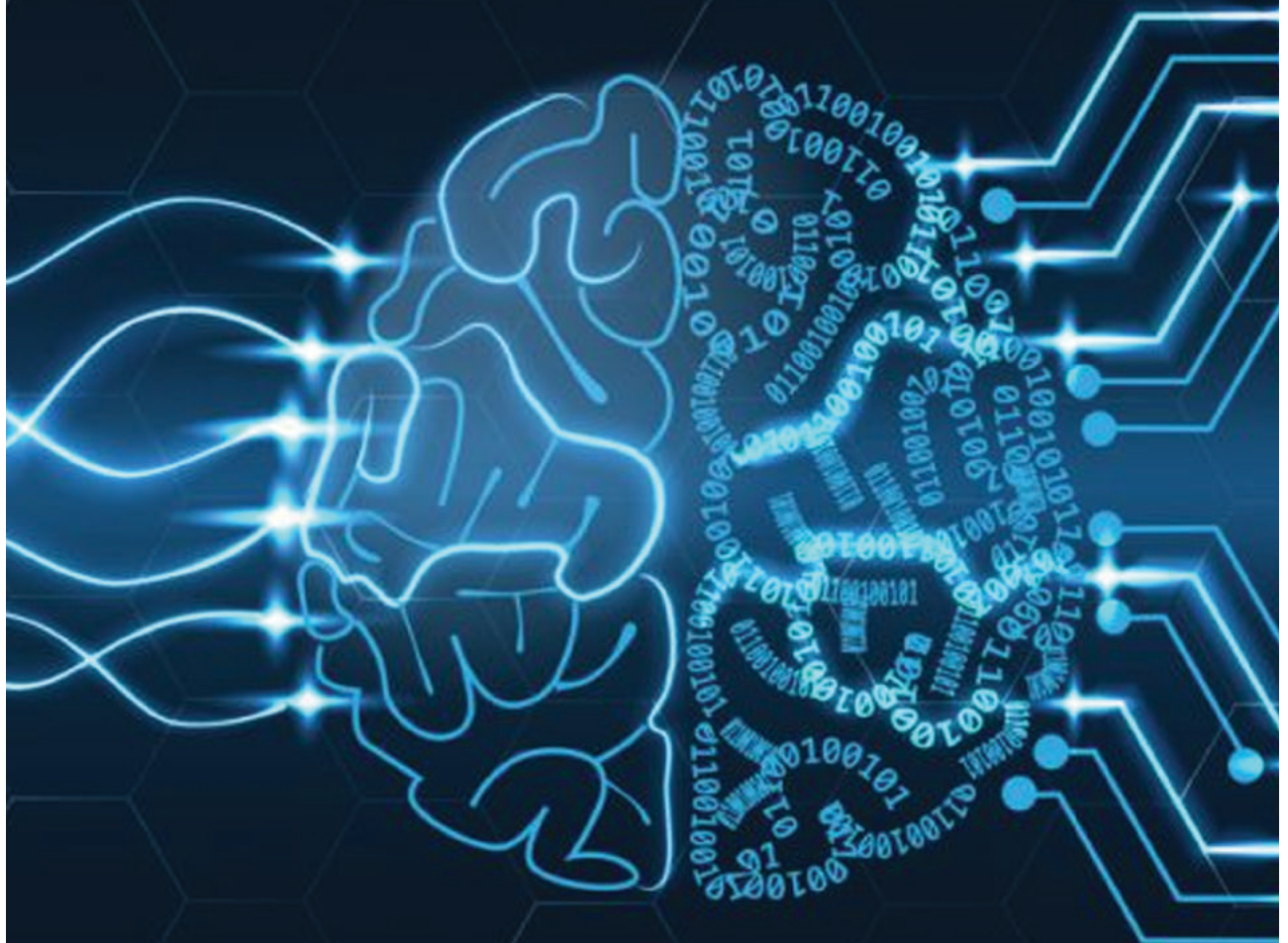


# PEMROGRAMAN FUZZY DAN JARINGAN SYARAF TIRUAN UNTUK SISTEM KENDALI CERDAS

Dilengkapi dengan Contoh Pemrograman Python



# PEMROGRAMAN FUZZY DAN JARINGAN SYARAF TIRUAN UNTUK SISTEM KENDALI CERDAS

*Dilengkapi dengan Contoh Pemrograman Python*

Basuki Rahmat

Budi Nugroho



Edisi Asli

Hak Cipta © 2019, pada penulis

Griya Kebonagung 2, Blok I2, No. 14

Kebonagung, Sukodono, Sidoarjo

Telp. : 0812-3250-3457

Website : [www.indomediapustaka.com](http://www.indomediapustaka.com)

E-mail : [indomediapustaka.sby@gmail.com](mailto:indomediapustaka.sby@gmail.com)

**Hak cipta dilindungi undang-undang.** Dilarang memperbanyak sebagian atau seluruh isi buku ini dalam bentuk apa pun, baik secara elektronik maupun mekanik, termasuk memfotokopi, merekam, atau dengan menggunakan sistem penyimpanan lainnya, tanpa izin tertulis dari Penulis.

## UNDANG-UNDANG NOMOR 19 TAHUN 2002 TENTANG HAK CIPTA

1. Barang siapa dengan sengaja dan tanpa hak mengumumkan atau memperbanyak suatu ciptaan atau memberi izin untuk itu, dipidana dengan pidana penjara paling lama **7 (tujuh) tahun** dan/atau denda paling banyak **Rp 5.000.000.000,00 (lima miliar rupiah)**.
2. Barang siapa dengan sengaja menyiarkan, memamerkan, mengedarkan, atau menjual kepada umum suatu ciptaan atau barang hasil pelanggaran Hak Cipta atau Hak Terkait sebagaimana dimaksud pada ayat (1), dipidana dengan pidana penjara paling lama **5 (lima) tahun** dan/atau denda paling banyak **Rp 500.000.000,00 (lima ratus juta rupiah)**.

---

Rahmat, Basuki

Nugroho, Budi

Pemrograman Fuzzy dan Jaringan Syaraf Tiruan untuk Sistem Kendali Cerdas/

Basuki Rahmat, Budi Nugroho

Edisi Pertama

—Sidoarjo: Indomedia Pustaka, 2018

Anggota IKAPI No. 195/JTI/2018

1 jil., 17 × 24 cm, 114 hal.

ISBN: 978-602-6417-94-7

I. Teknik

2. Pemrograman Fuzzy dan Jaringan Syaraf Tiruan  
untuk Sistem Kendali Cerdas

I. Judul

II. Basuki Rahmat, Budi Nugroho



# Kata Pengantar

Segala puji dan syukur kehadiran Allah S.W.T yang telah melimpahkan rahmat, hidayah dan ridho-Nya sehingga kami dapat menyelesaikan penyusunan buku yang berjudul “Pemrograman Fuzzy dan Jaringan Syaraf Tiruan Untuk Sistem Kendali Cerdas”. Semoga buku ini bisa menjadi salah satu rujukan Buku Teks untuk para mahasiswa Teknik Informatika, Ilmu Komputer, Sistem Komputer, Teknik Elektro dan para mahasiswa teknik lainnya yang mencari literatur pemrograman sistem cerdas. Dengan selesainya buku ini, kami mengucapkan banyak terimakasih kepada:

1. Teman-teman di Jurusan Sistem Komputer Institut Bisnis dan Informatika STIKOM Surabaya dan di Jurusan Teknik Informatika UPN “Veteran” Jawa Timur.
2. LPPM dan UPN “Veteran” Jawa Timur yang telah mengusahakan dan membiayai penerbitan buku ini, dalam Program Penelitian Mandiri Skim Peningkatan Mutu Pembelajaran (PMP) UPN “Veteran” Jawa Timur Tahun Anggaran 2018.
3. Dan semua pihak yang telah membantu terselesainya buku ini.

Kami menyadari bahwa buku ini masih banyak kekurangan, oleh karena itu kami mengharapkan saran dan kritik untuk perbaikan selanjutnya.

Surabaya, Januari 2019  
Tim Penulis



# Daftar Isi

<b>Kata Pengantar .....</b>	<b>i</b>
<b>Daftar Isi .....</b>	<b>ii</b>

## **Bagian 1**

### **Teori dan Aplikasi Sistem Fuzzy**

<b>Bab 1. Teori Fuzzy .....</b>	<b>1</b>
1.1. Teori Himpunan Fuzzy .....	1
1.2. Fungsi Keanggotaan Fuzzy .....	2
1.3. Operasi Himpunan Fuzzy .....	4
1.4. Basis Aturan .....	4
1.5. Logika Fuzzy .....	4
1.6. Sistem Inferensi Fuzzy.....	5
1.7. Sistem Fuzzy Model Mamdani.....	5
1.8. Sistem Fuzzy Model Sugeno .....	8
1.9. Center Average Defuzzifier .....	9
 <b>Bab 2. Pemrograman Fuzzy .....</b>	 <b>11</b>
2.1. Pemrograman Fuzzy dengan Python .....	11

2.1.1. Instalasi Anaconda .....	11
2.1.2. Python Menggunakan Jupyter Notebook .....	16
2.1.3. Python Menggunakan Spyder .....	18
2.1.4. Instalasi Scikit-fuzzy .....	20
2.2. Fuzzifikasi dengan Python.....	24
2.2.1. Fungsi Keanggotaan Segitiga .....	24
2.2.2. Fungsi Keanggotaan Trapesium.....	25
2.2.3. Fungsi Keanggotaan Gaussian .....	25
2.2.4. Fungsi Keanggotaan Bell .....	26
2.3. Defuzzifikasi dengan Python .....	27
2.4. Contoh Simulator Fuzzy dengan Python .....	29
<b>Bab 3. Sistem Kendali Cerdas Berbasis PID-Fuzzy .....</b>	<b>35</b>
3.1. Sistem Kendali PID.....	35
3.2. Sistem Kendali PID-Fuzzy .....	37
3.3. Pemrograman Sistem Kendali PID .....	39
3.4. Pemrograman Sistem Kendali PID-Fuzzy.....	41
3.5. Pemrograman TCLab.....	47
3.5.1. Pengaturan dan Instalasi.....	48
3.5.2. Pemodelan Dinamis Sistem Pemanas.....	52
3.5.2.1. Pemodelan Dinamis Keseimbangan Energi.....	52
3.5.2.2. Pemodelan Dinamis Orde Satu Plus Waktu-Tunda .....	55
3.5.2.3. Pemodelan Dinamis Orde Dua Plus Waktu-Tunda .....	58
3.5.4. Pengujian Model Pemanas .....	61
 <b>Bagian2</b>	
<b>Teori dan Aplikasi Jaringan Syaraf Tiruan</b>	
<b>Bab 4. Teori Jaringan Syaraf Tiruan .....</b>	<b>73</b>
4.1. Konsep Dasar.....	73
4.2. Arsitektur Jaringan Syaraf Tiruan .....	74
4.3. Pelatihan Jaringan Saraf Tiruan .....	75
4.3.1. Backpropagation.....	75
4.3.2. Dynamic Tunneling Technique.....	77
 <b>Bab 5. Pemrograman Jaringan Syaraf Tiruan.....</b>	<b>81</b>
5.1. Pemrograman Jaringan Syaraf Tiruan dengan Python.....	81
5.2. Prediksi XOR dengan JST .....	84
5.3. Prediksi XOR dengan JST-Keras .....	94

<b>Bab 6. Sistem Kendali Cerdas Berbasis PID-JST .....</b>	<b>97</b>
6.1. Sistem Kendali PID-JST.....	97
6.2. Pemrograman Sistem Kendali PID-JST .....	99
<b>Bab 7. Penutup .....</b>	<b>103</b>
7.1. Kesimpulan.....	103
7.2. Ucapan Terimakasih .....	103
<b>DAFTAR PUSTAKA .....</b>	<b>105</b>





# BAB 1

## Teori Fuzzy

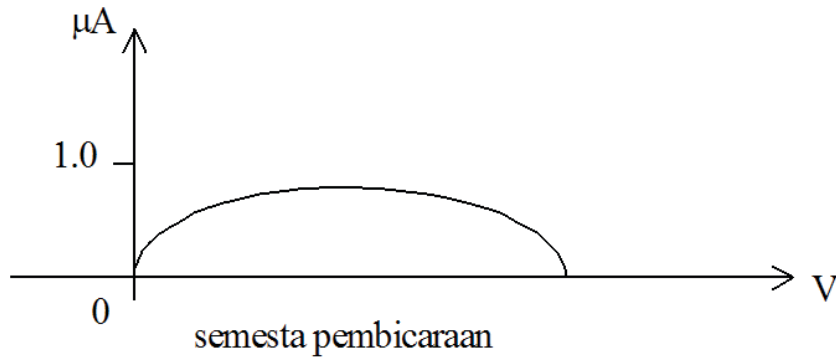
---

### 1.1. Teori Himpunan Fuzzy

Teori himpunan fuzzy ini didasarkan pada logika fuzzy (Kosko, B. 1992). Terdapat nilai logika antara 0 dan 1 yang menyatakan tingkat kebenaran. Misalkan  $V$  adalah kumpulan obyek yang secara umum dinyatakan dengan  $\{v\}$ , yang bisa berharga diskrit atau kontinyu.  $V$  disebut semesta pembicaraan (*universe of discourse*), dan  $v$  mewakili elemen-elemen  $V$ . Suatu himpunan fuzzy  $A$  dalam semesta pembicaraan  $V$  dapat dinyatakan oleh suatu fungsi keanggotaan  $\mu_A$  (*membership function*) yang mewakili nilai dalam interval nilai logika  $[0,1]$  untuk setiap  $v$  dalam  $V$  dan dinyatakan sebagai:

$$\mu_A = V \rightarrow [0,1] \quad (1.1)$$

Yang dapat digambarkan dalam bentuk seperti terlihat pada Gambar 1.1.



Gambar 1.1. Himpunan fuzzy dan fungsi keanggotaan

Himpunan fuzzy A dalam himpunan semesta V dapat dinyatakan sebagai pasangan antara elemen v dan tingkat fungsi keanggotaan, atau:

$$A = \{(v, \mu_A(v)) / v \in V\} \quad (1.2)$$

Semua elemen v dalam V memberikan nilai  $\mu_A > 0$  disebut sebagai penyokong (*support*) dari himpunan fuzzy yang bersangkutan, jika  $\mu_A = 0.5$  maka v disebut sebagai titik silang (*crossover*) dan himpunan fuzzy dimana penyokongnya bernilai 1.0 disebut sebagai fuzzy tunggal (*singleton*).

## 1.2. Fungsi Keanggotaan Fuzzy

Fungsi Keanggotaan yang sering digunakan adalah sebagai berikut :

### 1). Fungsi keanggotaan Segitiga

Fungsi keanggotaan yang mempunyai parameter a,b,c dengan formulasi:  
segitiga  $(x;a,b,c) =$

$$\max \left[ \min \left( \left( \frac{x-a}{b-a} \right), \left( \frac{c-x}{c-b} \right) \right), 0 \right] \quad (1.3)$$

### 2). Fungsi keanggotaan Trapesium

Fungsi keanggotaan yang mempunyai parameter a,b,c,d dengan formulasi:  
trapesium  $(x;a,b,c,d) =$

$$\max \left[ \min \left( \min \left( \left( \frac{x-a}{b-a} \right), 1 \right), \left( \frac{d-x}{d-c} \right) \right), 0 \right] \quad (1.4)$$

3). Fungsi keanggotaan Gaussian

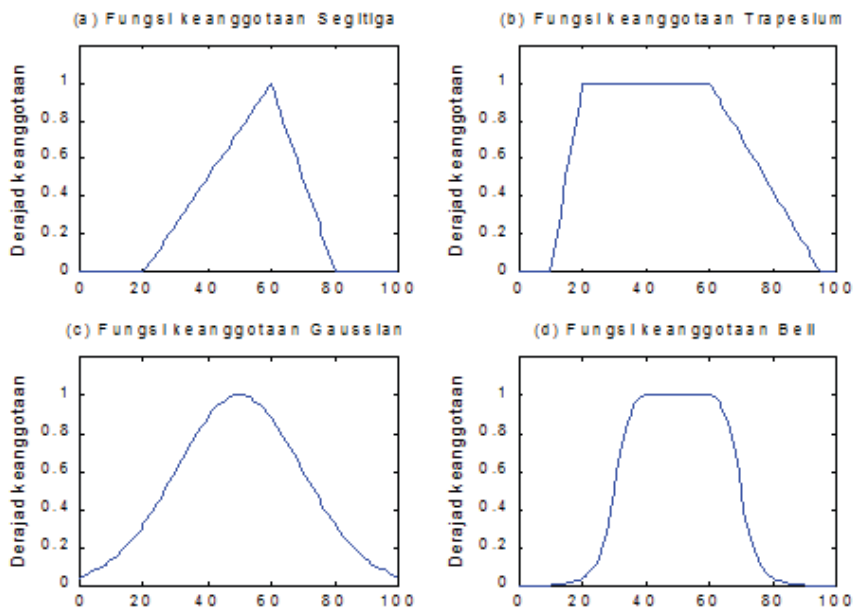
Fungsi yang mempunyai parameter  $a$ ,  $\sigma$  dengan formulasi:

$$\text{gaussian}(x; \sigma, a) = \exp\left[-\frac{1}{2}\left(\frac{x-c}{\sigma}\right)^2\right] \quad (1.5)$$

4). Fungsi keanggotaan Bell yang diperluas. Fungsi Keanggotaan yang mempunyai parameter  $a, b, c$  dengan formulasi:

$$\text{bell}(x; a, b, c) = \frac{1}{\left(1 + \left|\frac{(x-c)^{2b}}{a}\right|\right)} \quad (1.6)$$

dengan  $b$  positif. Jika  $b$  negatif fungsi keanggotaan menjadi fungsi keanggotaan bell terbalik. Ilustrasi dari keempat fungsi keanggotaan diatas diperlihatkan pada Gambar 1.2.



Gambar 1.2. Jenis fungsi keanggotaan fuzzy

---

### 1.3. Operasi Himpunan Fuzzy

Misalkan A dan B adalah dua himpunan fuzzy dalam himpunan semesta V dengan fungsi keanggotaan masing-masing  $\mu_A$  dan  $\mu_B$ . Beberapa operasi yang dipakai dalam himpunan fuzzy, misalnya:

1. Komplemen dari A ( $A'$ ), dimana  $\mu_{A'}(v) = 1 - \mu_A(v)$ .
2. Irisan dari A dan B ( $A \cap B$ ), dimana  $\mu(A \cap B) = \min(\mu_A(v), \mu_B(v))$ .
3. Gabungan dari A dan B ( $A \cup B$ ), dimana  $\mu(A \cup B) = \max(\mu_A(v), \mu_B(v))$ .

---

### 1.4. Basis Aturan

Basis aturan merupakan inti dari sistem fuzzy karena pada bagian ini berada sekumpulan aturan dalam bentuk

$$\begin{aligned} &\text{if } x_1(k) \text{ is } A_1' \text{ and } \dots \text{ and } x_n(k) \text{ is } A_n' \text{ and } x_1(k+1) \text{ is } B_1' \text{ and } \dots \\ &\text{and } x_n(k+1) \text{ is } B_n' \text{ Then } u' = a_1' x_1(k) + \dots + a_n' x_n(k) \\ &+ b_1' x_1(k+1) + \dots + b_n' x_n(k+1) + c'. \end{aligned} \quad (1.7)$$

dengan  $l = 1, 2, \dots, M$ ,  $A_i^j, B_i^j$  merupakan himpunan fuzzy yang dibentuk, sedangkan  $a_i^j, b_i^j, c_i^j$  merupakan parameter-parameter, M jumlah aturan, dan x, u merupakan masukan dan keluaran pada sistem fuzzy.

Aturan *if-then* pada logika fuzzy adalah pernyataan dari bentuk jika A maka B, dimana A dan B adalah himpunan fuzzy. Karena keringkasannya bentuk ini sering digunakan untuk mewakili kemampuan manusia untuk mengambil keputusan atas suatu kondisi yang penuh dengan ketidakpastian dan ketidaktepatan. Contohnya: jika tekanan tinggi, maka volume kecil, dimana tinggi dan kecil adalah besaran kualitatif yang dijelaskan dalam fungsi keanggotaan.

---

### 1.5. Logika Fuzzy

Dalam logika fuzzy ada dua kaidah atur simpulan fuzzy yang penting, yaitu GMP (*Generalized Modus Ponens*) dan GMT (*Generalized Modus Tollens*), yang di definisikan sebagai berikut :

GMP :

premis 1 : x is  $A'$

premis 2 : if x is A then y is B

konsekuen : y is  $B'$

Jika terdapat  $B = A \circ R$  maka berlaku  $B' = A' \circ R$ .

GMT :

premis 1 :  $y$  is  $B'$

premis 2 : if  $x$  is  $A$  then  $y$  is  $B$

konsekuen :  $x$  is  $A'$ .

Jika terdapat  $B = A \circ R$  maka berlaku  $A' = R \circ B'$ .

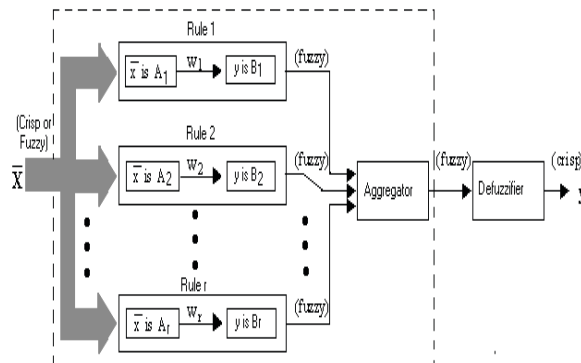
$A, A', B$  dan  $B'$  adalah himpunan fuzzy, sedangkan  $x$  dan  $y$  variabel linguistik.

## 1.6. Sistem Inferensi Fuzzy

Sistem Inferensi Fuzzy adalah sistem kerja komputer yang didasarkan pada konsep teori fuzzy, aturan fuzzy *if-then*, dan logika fuzzy. Struktur dasar dari sistem Inferensi Fuzzy terdiri dari:

1. Basis aturan yang berisi aturan *if-then*.
2. Basis data yang mendefinisikan fungsi keanggotaan dari himpunan fuzzy.
3. Unit pengambilan keputusan yang menyatakan operasi inferensi aturan-aturan.
4. Fuzzifikasi yang mentransformasi masukan himpunan klasik (*crisp*) ke derajat tertentu yang sesuai dengan aturan besaran fungsi keanggotaan
5. Defuzzifikasi yang mentransformasi hasil fuzzy ke bentuk keluaran yang *crisp*.

Blok diagram Sistem Inferensi Fuzzy terlihat pada gambar dibawah ini



Gambar 1.3. Sistem Inferensi Fuzzy

## 1.7. Sistem Fuzzy Model Mamdani

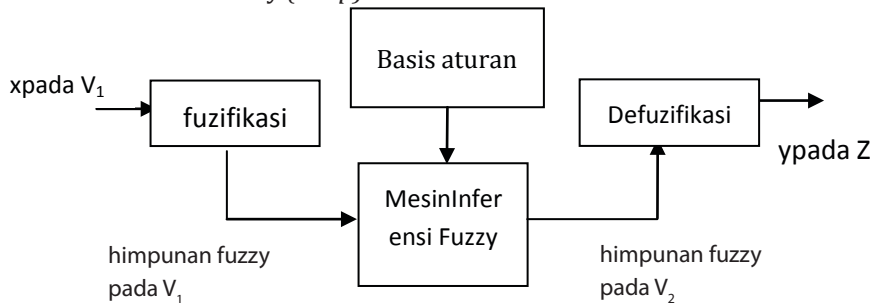
Sistem fungsi Mamdani merupakan bentuk aturan seperti

$$\mathcal{R}^i : \text{if } x_1 \text{ is } F_1^j \text{ and } \dots \text{ and } x_n \text{ is } F_n^i \text{ then } y \text{ is } G^i \quad (1.8)$$

Dengan konfigurasi sistem seperti terlihat pada Gambar 1.4 pada sistem fuzzy model Mamdani terdapat banyak kebebasan dalam memilih fuzzifikasi, mesin

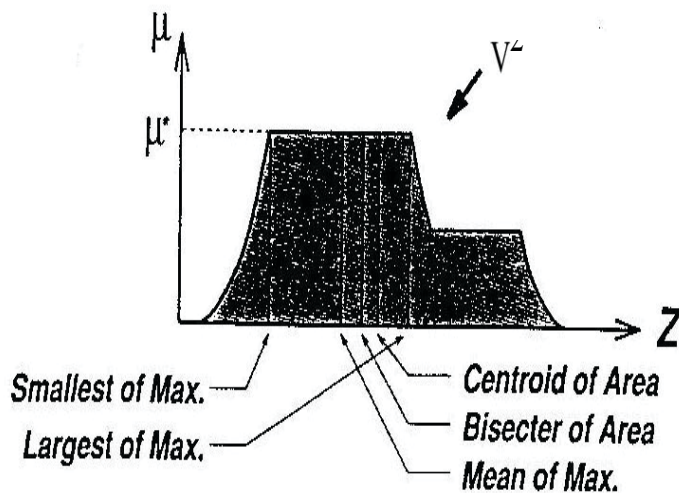
inferensi fuzzy, dan defuzifikasi, sehingga dapat dirancang sistem fuzzy yang tepat untuk suatu persoalan tertentu.

Fuzzifikasi berfungsi memetakan titik crisp  $x \in V_1$  ke dalam himpunan fuzzy A pada  $V_1$ . Mesin inferensi fuzzy memetakan himpunan fuzzy A pada  $V_1$  kepada himpunan fuzzy A pada  $V_2$ . Defuzifikasi berfungsi menerjemahkan besaran fuzzy ke dalam besaran bukan fuzzy (*crisp*).



Gambar 1.4. Sistem fuzzy model Mamdani

Pada bagian defuzifikasi terdapat lima metode untuk mendefuzifikasi himpunan fuzzy  $V_2$  menjadi besaran bukan fuzzy (*crisp*) pada Z seperti terlihat pada Gambar 1.5 berikut.



Gambar 1.5. Metode defuzifikasi dari fuzzy model mamdani

Dimana perumusan yang digunakan masing-masing sebagai berikut (Jang, J.-S.R, Sun, C.-T & Mizutani, E. 1996):

a). Centroid of area  $z_{COA}$  :

$$z_{COA} = \frac{\int_z \mu_A(z) z \, d}{\int_z \mu_A(z) \, d} \quad (1.9)$$

b). Bisector of area  $z_{BOA}$  :

$$\int_{\alpha}^z \mu_A(z) \, d = \int_{z_{BOA}}^{\beta} \mu_A(z) \, d \quad (1.10)$$

c). Mean of maximum  $z_{MOM}$  :

$$z_{MOM} = \frac{\int_{z'} z \, d}{\int_{z'} d} \quad (1.11)$$

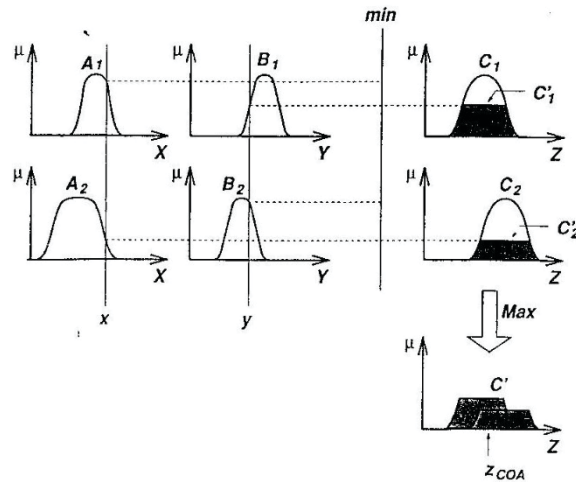
d). Smallest of maximum  $z_{SOM}$  :

$z_{MOM}$  adalah nilai minimum dari  $z$ .

e). Largest of maximum  $z_{LOM}$  :

$z_{LOM}$  adalah nilai maksimum dari  $z$ .

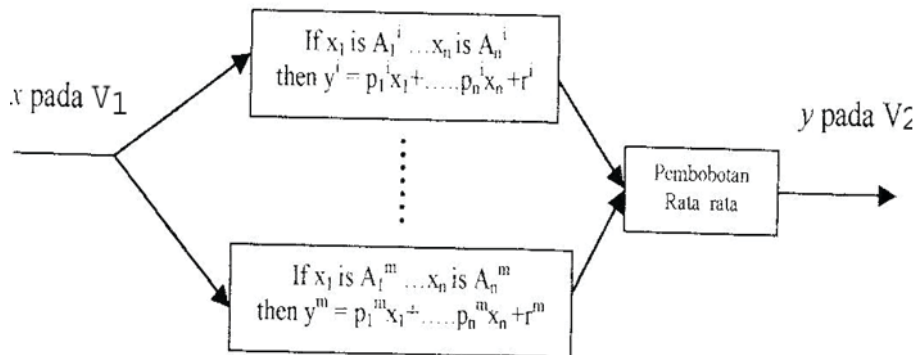
Untuk bisa memahami mekanisme sistem inferensi fuzzy model mamdani salah satunya yang menggunakan minimum dan maksimum, seperti terlihat pada Gambar 1.6.



Gambar 1.6. Mekanisme sistem inferensi fuzzy model mamdani

## 1.8. Sistem Fuzzy Model Sugeno

Sistem fuzzy model Sugeno juga dikenal dengan sistem fuzzy model TSK pertama diperkenalkan oleh Takagi, Sugeno dan Kang dalam usaha membentuk aturan fuzzy dari himpunan data masukan dan keluaran. Konfigurasi sistem fuzzy model Sugeno dapat dilihat pada Gambar 1.7.



Gambar 1.7. Sistem fuzzy model Sugeno orde satu



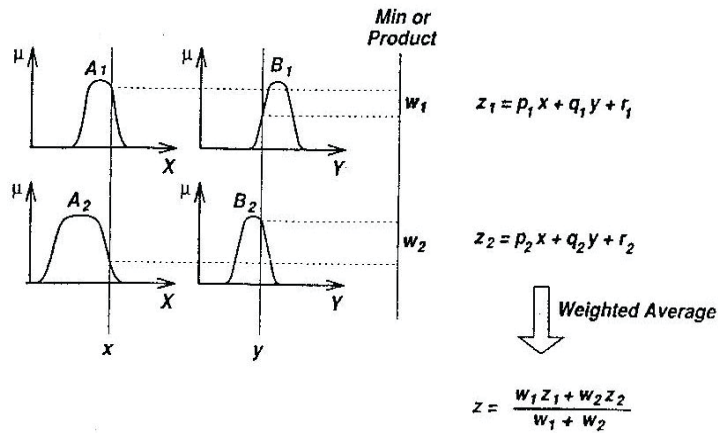
Keluaran dari sistem fuzzy Sugeno merupakan pembobotan rata-rata, yang dinyatakan dengan

$$y(x) = \frac{\sum_{l=1}^M W^l y^l}{\sum_{l=1}^M W^l} \quad (1.12)$$

dengan bobot  $W^l$  merupakan nilai kebenaran dari aturan,  $W^l$  dapat dihitung dengan

$$W^l = \prod_{i=1}^n \mu_{F_i^l}(X) \quad (1.13)$$

Mekanisme sistem inferensi fuzzy model Sugeno diperlihatkan seperti pada Gambar 1.8.



Gambar 1.8. Mekanisme sistem inferensi fuzzy model Sugeno

Kelebihan dari sistem Fuzzy Sugeno adalah cara perhitungan yang lebih sederhana, transparan dan efisien serta lebih memudahkan dalam pekerjaan komputasi dibandingkan dengan sistem fuzzy model Mamdani.

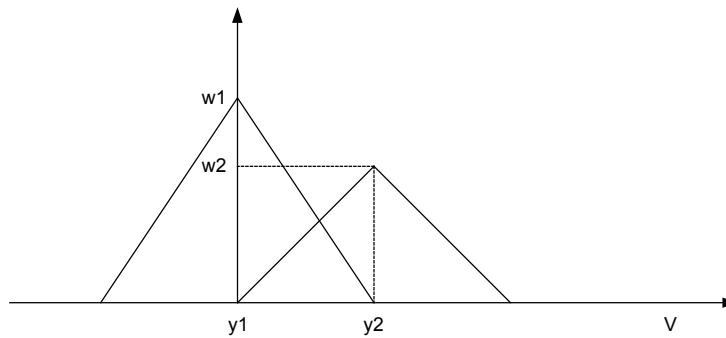
## 1.9. Center Average Defuzzifier

Selain mekanisme sistem inferensi fuzzy model Mamdani dan Sugeno, salah satu metode yang sering digunakan karena mudah penggunaannya, yaitu digunakan *Center Average Defuzzifier*. Misalkan diperoleh keluaran berupa dua fungsi keanggotaan fuzzy seperti pada Gambar 1.9. Dimana  $y_1$  dan  $y_2$  adalah nilai tengah dari masing-

masing fungsi keanggotaan fuzzy, dan  $w_1$  dan  $w_2$  adalah tinggi. Maka *Center Average defuzzifier*  $y^*$  bisa dihitung seperti pada Persamaan (1.14) (Li-Xin Wang, 1997).

$$y^* = \frac{\sum_{l=1}^2 y_l w_l}{\sum_{l=1}^2 w_l} \quad (1.14)$$

Gambarannya sebagai berikut.



Gambar 1.9. Contoh center average defuzzifier

# BAB 2

## Pemrograman Fuzzy

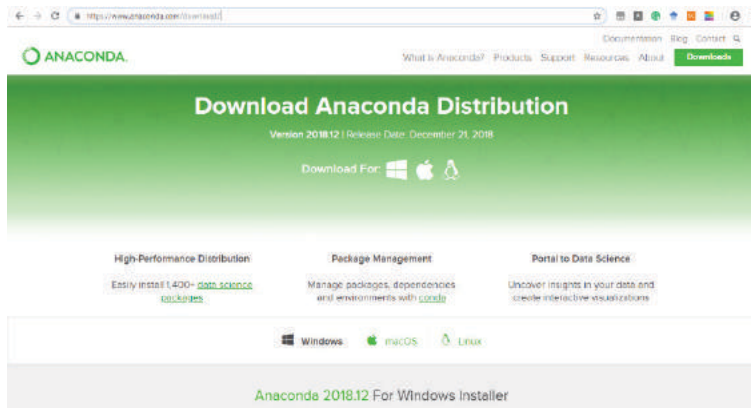
---

### 2.1. Pemrograman Fuzzy dengan Python

#### 2.1.1. Instalasi Anaconda

Belajar sebuah bahasa pemrograman bukan hanya sekadar mempelajari sintaknya saja. Kadang kita harus belajar tentang bagaimana lingkungan bahasa itu bekerja, mengelola paket, pilihan editor dan terlebih lagi apabila kita perlu bahasa itu untuk mencapai tujuan lainnya. Seperti halnya cara mudah menjalankan python dapat melalui lingkungan anaconda. Melalui anaconda, python dapat dijalankan dengan dua cara, yaitu berbasis web melalui Jupyter Notebook dan berbasis desktop melalui Spyder.

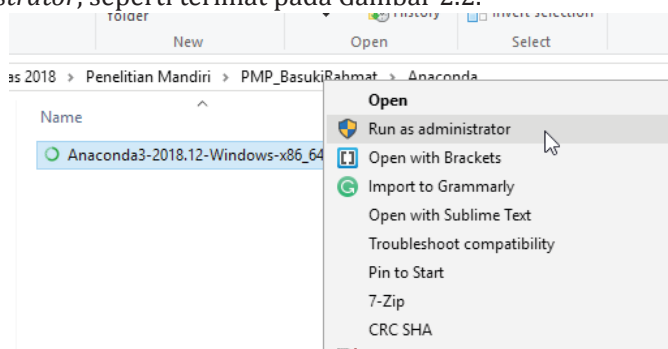
Anaconda ini gratis, memiliki banyak paket dan tool, siap digunakan, sudah termasuk *Python distribution* di dalamnya. Untuk mendapatkan anaconda, silahkan download di alamat <https://www.anaconda.com/download>, seperti terlihat pada Gambar 2.1.



Gambar 2.1. Download anaconda

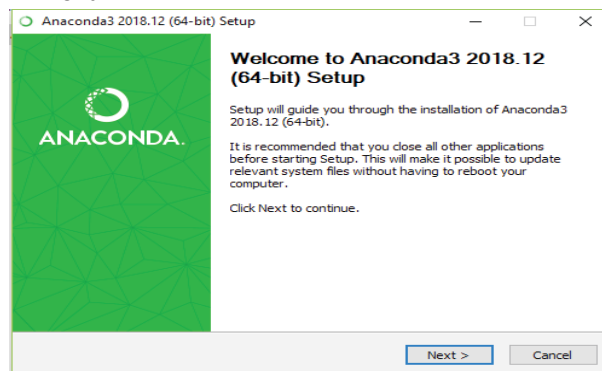
Selanjutnya, silahkan dilakukan instalasi anaconda. Proses instalasi anaconda di Windows 10, sesuai langkah-langkah berikut ini.

1. Setelah download anaconda, silahkan dilakukan proses instalasi dengan *run as administrator*, seperti terlihat pada Gambar 2.2.



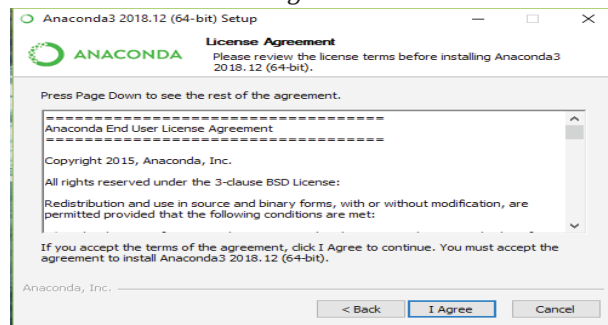
Gambar 2.2. Instal anaconda

- 2, Silahkan klik Next.



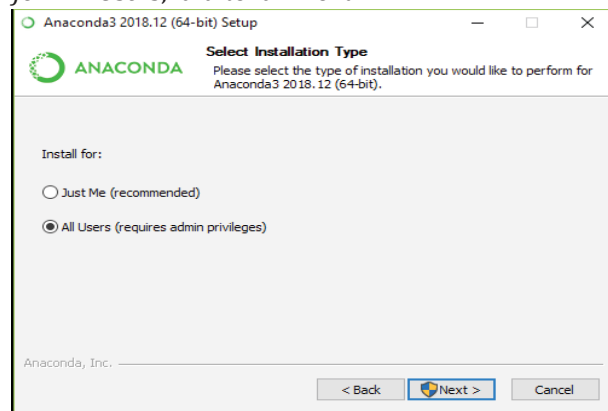
Gambar 2.3. Tekan Next

3. Kemudian baca lisensi dan klik *I Agree*.



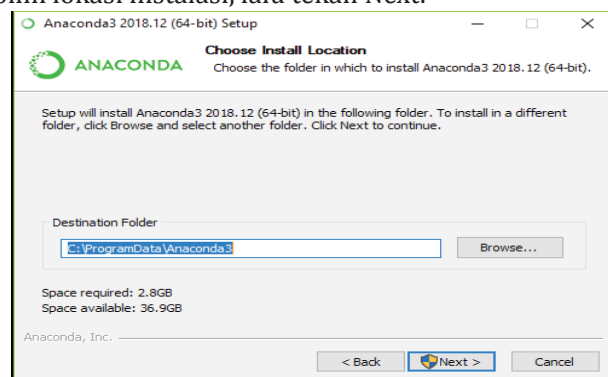
Gambar 2.4. Lisensi anaconda

4. Pilih *Intall for: All Users*, lalu tekan Next.



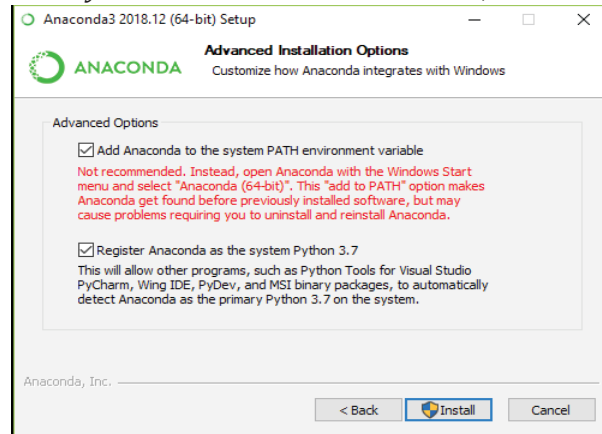
Gambar 2.5. Pilihan jenis instalasi

5. Silahkan pilih lokasi instalasi, lalu tekan Next.



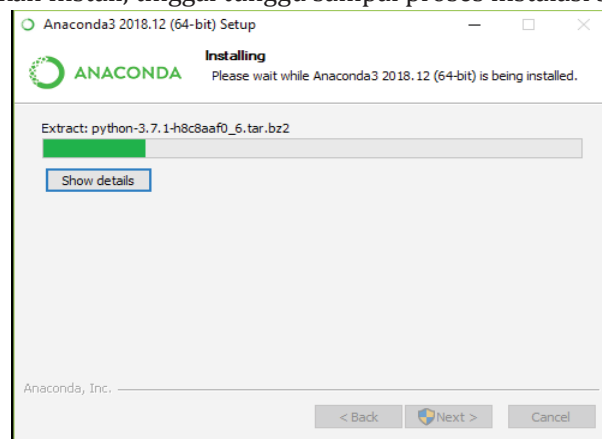
Gambar 2.6. Pilihan lokasi instalasi

6. Selanjutnya pilihan integrasi anaconda dengan lingkungan windows, pilih *Add Anaconda to the system PATH environment variable*, lalu tekan Install.



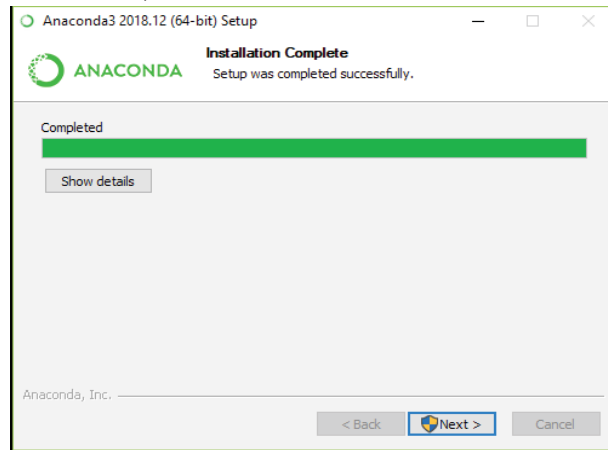
Gambar 2.7. Pilihan integrasi dengan windows

7. Setelah tekan Install, tinggal tunggu sampai proses instalasi selesai.



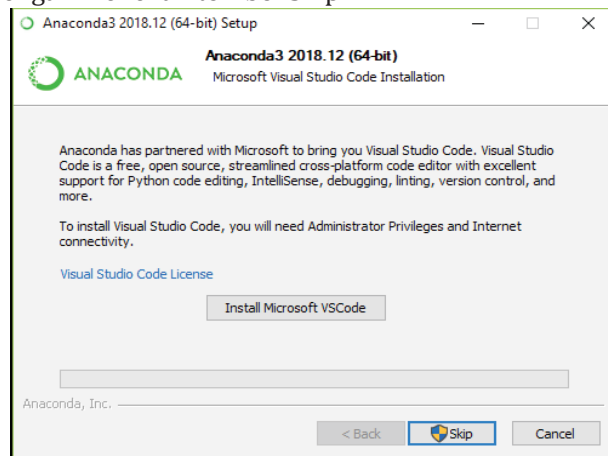
Gambar 2.8. Proses instalasi

8. Proses instalasi selesai, tekan Next.



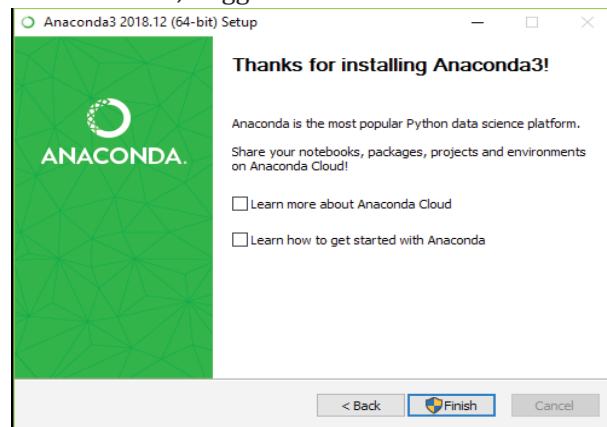
Gambar 2.9. Instalasi selesai

9. Selanjutnya, ada pilihan instal Microsoft Visual Studio Code (VSCode), dapat dilewati dengan menekan tombol Skip.



Gambar 2.10. Pilihan instal VSCode

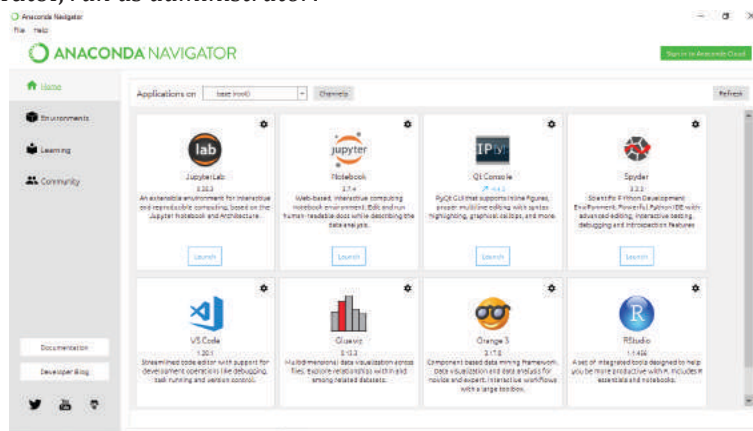
10. Instalasi anaconda selesai, tinggal tekan Finish.



Gambar 2.11. Instalasi anaconda selesai

### 2.1.2. Python Menggunakan Jupyter Notebook

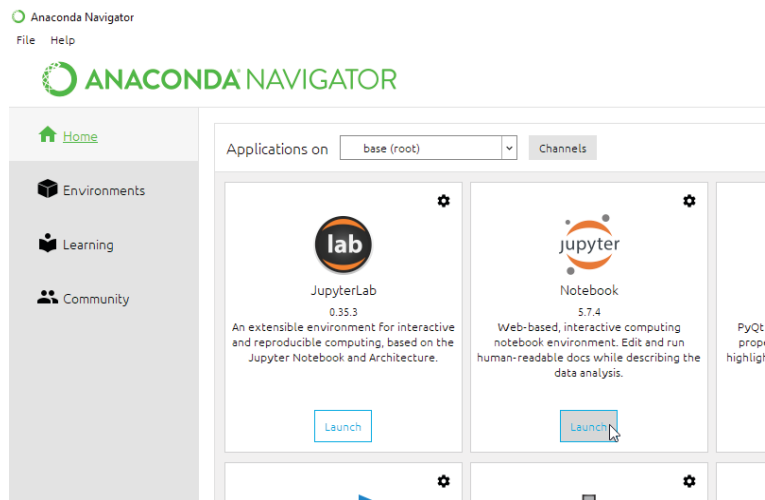
Selanjutnya, untuk memulai pemrograman python melalui lingkungan anaconda, silahkan dijalankan anaconda di windows. Jalankan Anaconda Navigator sebagai administrator, *run as administrator*.



Gambar 2.12. Tampilan awal anaconda

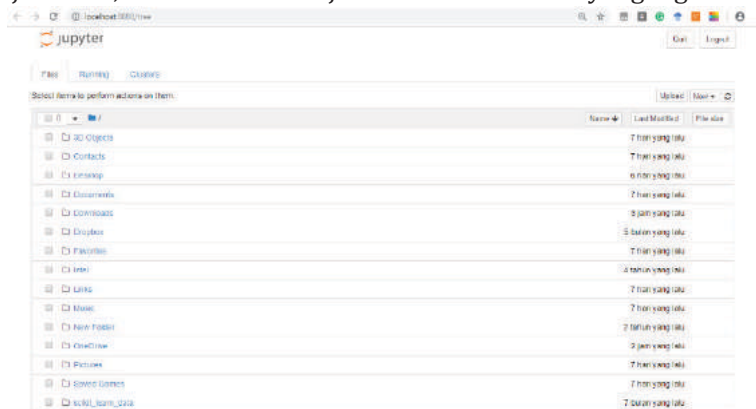


Untuk memulai python menggunakan Jupyter Notebook, silahkan tekan Jupyter Notebook.



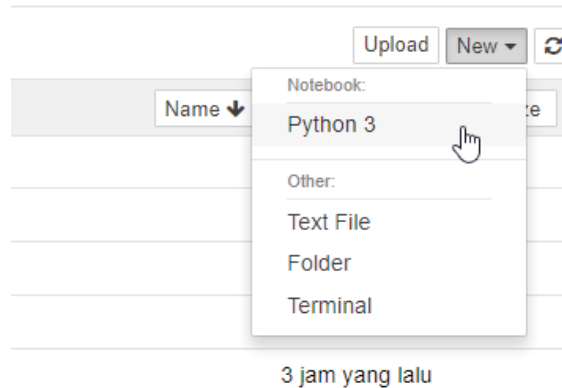
Gambar 2.13. Menjalankan Jupyter Notebook

Setelah dijalankan, maka akan menuju ke default browser yang digunakan.



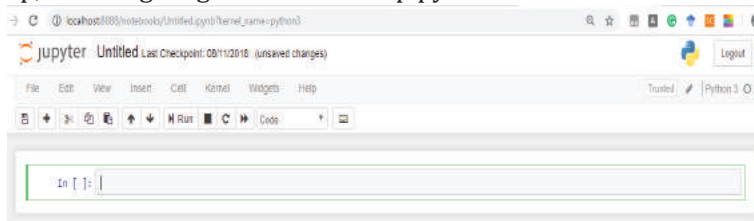
Gambar 2.14. Tampilan Jupyter Notebook di browser

Untuk mencoba menjalankan skrip python, silahkan klik tombol New dan arahkan ke python, seperti terlihat pada Gambar 2.15.



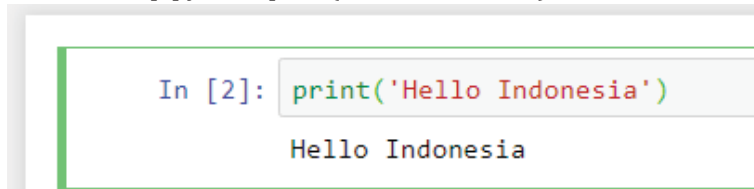
Gambar 2.15. Menjalankan python dari Jupyter Notebook

Setelah siap, bisa langsung diketikkan skrip python.



Gambar 2.16. Python Jupyter Notebook siap digunakan

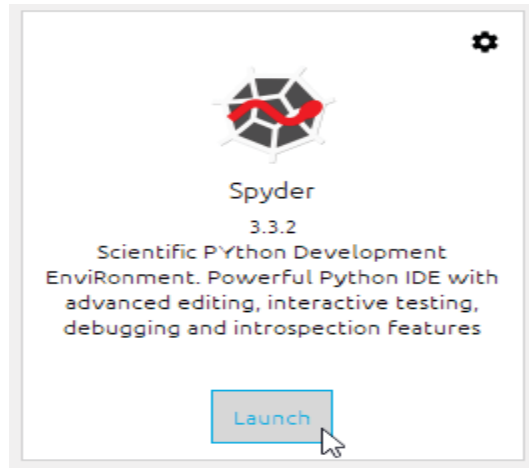
Silahkan tuliskan skrip python, print('Hello Indonesia').



Gambar 2.17. Contoh hasil skrip python

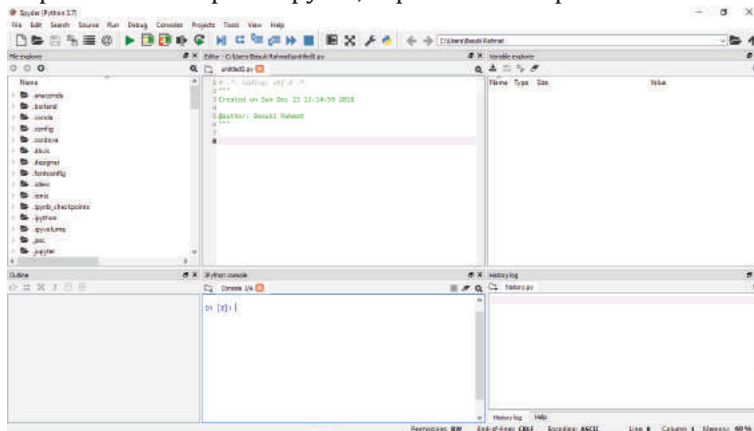
### 2.1.3. Python Menggunakan Spyder

Cara lain menjalankan python melalui anaconda, dapat digunakan Spyder. Silahkan tekan Spyder.



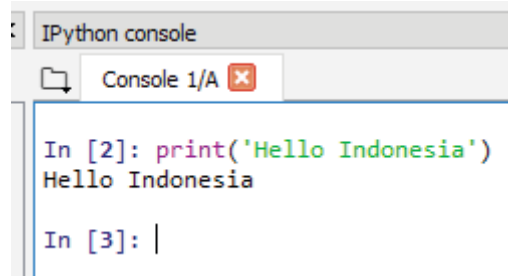
Gambar 2.18. Menjalankan Spyder melalui anaconda

Tunggu sampai keluar tampilan Spyder, seperti terlihat pada Gambar 2.19.



Gambar 2.19. Tampilan Spyder

Untuk mencoba, silahkan tuliskan skrip python, `print('Hello Indonesia')`.



Gambar 2.20. Tampilan hasil menjalankan skrip python

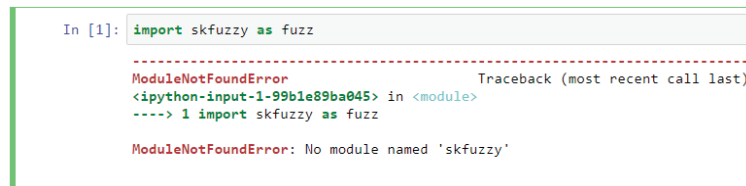
Selanjutnya, untuk keperluan contoh penulisan buku ini, python dijalankan melalui Jupyter Notebook.

#### 2.1.4. Instalasi Scikit-fuzzy

Pemrograman Fuzzy dengan Python memanfaatkan *toolkit* scikit-fuzzy (skfuzzy). Scikit-fuzzy adalah *toolkit* logika fuzzy untuk SciPy. Berupa kumpulan algoritma logika fuzzy yang dimaksudkan untuk digunakan dalam SciPy *Stack*, yang ditulis dalam bahasa komputasi Python.

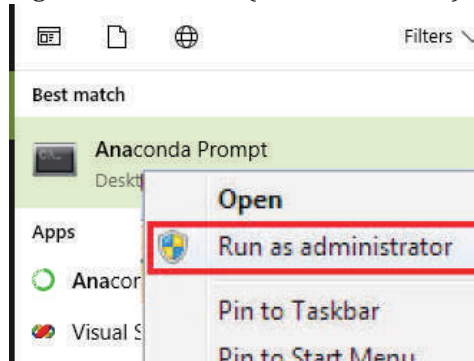
Ketikkan skrip import skfuzzy, atau jika nantinya ingin digunakan secara singkat sebagai fuzz, ketikkan import skfuzzy as fuzz. Jika belum pernah diinstal sebelumnya, seharusnya akan muncul error.

```
>> import skfuzzy as fuzz
```



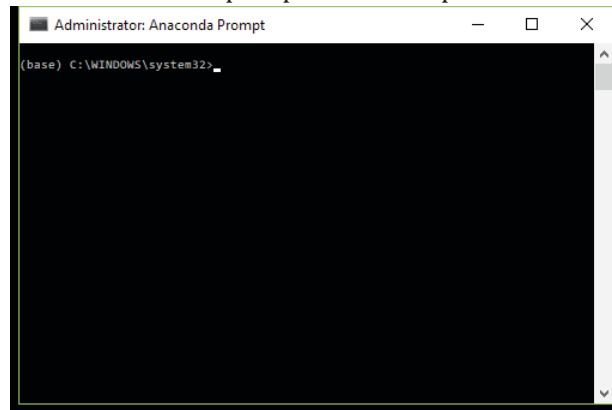
Gambar 2.21. Import skfuzzy error

Untuk instal modul skfuzzy melalui panel Anaconda Prompt, silahkan jalankan Anaconda Prompt sebagai administrator (*as administrator*).



Gambar 2.22. Menjalankan Anaconda Prompt sebagai administrator

Sehingga muncul Anaconda Prompt seperti terlihat pada Gambar 2.23.

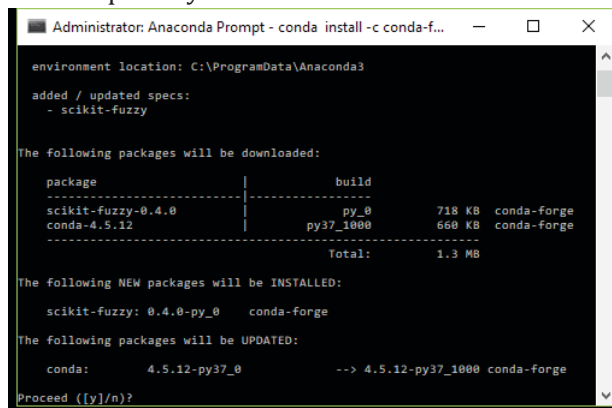


Gambar 2.23. Anaconda Prompt

Untuk instal modul scikit-fuzzy menggunakan conda, bisa menggunakan salah satu dari dua perintah berikut ini:

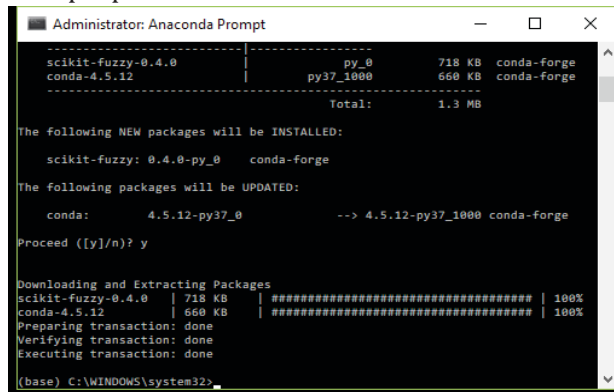
```
>> conda install -c conda-forge scikit-fuzzy  
>> conda install -c conda-forge/label/gcc7 scikit-fuzzy
```

Misalnya digunakan perintah yang pertama, maka akan muncul seperti terlihat pada Gambar 2.24. Ada pertanyaan *Proceed?* Silahkan tekan Y.



Gambar 2.24. Paket scikit-fuzzy

Tekan Y, tunggu sampai proses instalasi modul selesai.



```
Administrator: Anaconda Prompt

-----
scikit-fuzzy-0.4.0 | py_0 | 718 KB | conda-forge
conda-4.5.12 | py37_1000 | 660 KB | conda-forge
-----
Total: 1.3 MB

The following NEW packages will be INSTALLED:

  scikit-fuzzy: 0.4.0-py_0 conda-forge

The following packages will be UPDATED:

  conda: 4.5.12-py37_0 --> 4.5.12-py37_1000 conda-forge

Proceed ([y]/n)? y

Downloading and Extracting Packages
scikit-fuzzy-0.4.0 | 718 KB | | 100%
conda-4.5.12 | 660 KB | | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done

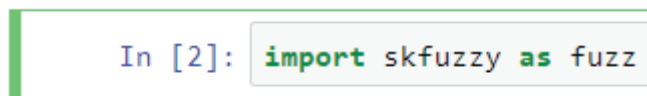
(base) C:\WINDOWS\system32>
```

Gambar 2.25. Proses instalasi modul scikit-fuzzy sukses

Silahkan dicoba lagi, jalankan lagi skrip import skfuzzy.

```
>> import skfuzzy as fuzz
```

Jika berhasil, tidak muncul pesan error:



```
In [2]: import skfuzzy as fuzz
```

Gambar 2.26. Modul skfuzzy berhasil diimport

Modul lain yang sering dibutuhkan adalah modul Numpy. NumPy adalah paket dasar untuk komputasi ilmiah dengan Python. Numpy berisi antara lain:

1. objek array N-dimensi yang kuat.
2. fungsi yang canggih.
3. alat untuk mengintegrasikan kode C/C++ dan Fortran.
3. aljabar linier yang sangat berguna, untuk transformasi Fourier, dan kemampuan angka acak, dll.

Selain penggunaan ilmiahnya yang jelas, NumPy juga dapat digunakan sebagai wadah data generik multi dimensi yang efisien. Tipe data sebarang dapat didefinisikan. Ini memungkinkan NumPy untuk berintegrasi dengan cepat dengan berbagai macam basis data.

Untuk instal modul Numpy menggunakan conda, bisa menggunakan perintah berikut ini:

```
>> conda install -c anaconda numpy
```

Ketikkan skrip import numpy, atau jika nantinya ingin digunakan secara singkat sebagai np, ketikkan import numpy as np.

```
>> import numpy as np
```



```
In [2]: import numpy as np
```

Gambar 2.27. Modul numpy berhasil diimport

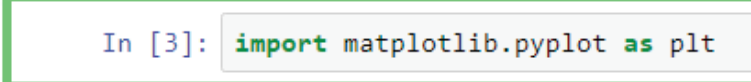
Modul lain yang sering dibutuhkan adalah modul Matplotlib. Modul matplotlib dibutuhkan untuk keperluan pengeplotan grafik di python. Matplotlib ini digunakan untuk visualisasi data Python menggunakan: plot, pyplot, pylab, dan lain-lain. Dengan modul matplotlib dapat dibuat plot, histogram, spektrum daya, diagram batang, diagram galat, plot sebar, dll.

Untuk instal modul matplotlib menggunakan conda, bisa menggunakan salah satu dari perintah-perintah berikut ini:

```
>> conda install -c conda-forge matplotlib
>> conda install -c conda-forge/label/testing matplotlib
>> conda install -c conda-forge/label/testing/gcc7 matplotlib
>> conda install -c conda-forge/label/gcc7 matplotlib
>> conda install -c conda-forge/label/broken matplotlib
>> conda install -c conda-forge/label/rc matplotlib
```

Ketikkan skrip import matplotlib.pyplot, atau jika nantinya ingin digunakan secara singkat sebagai plt, ketikkan import matplotlib.pyplot as plt.

```
>> import matplotlib.pyplot as plt
```



```
In [3]: import matplotlib.pyplot as plt
```

Gambar 2.28. Modul matplotlib berhasil diimport

## 2.2. Fuzzifikasi dengan Python

Seperti apa yang telah dijabarkan di Bab 1, beberapa fungsi keanggotaan Fuzzy yang sering digunakan antara lain: segitiga, trapesium, gaussian dan bell. Untuk pemrograman fuzzifikasi atau mengubah besaran krisp ke fuzzy, dapat dijabarkan pada uraian berikut.

### 2.2.1. Fungsi Keanggotaan Segitiga

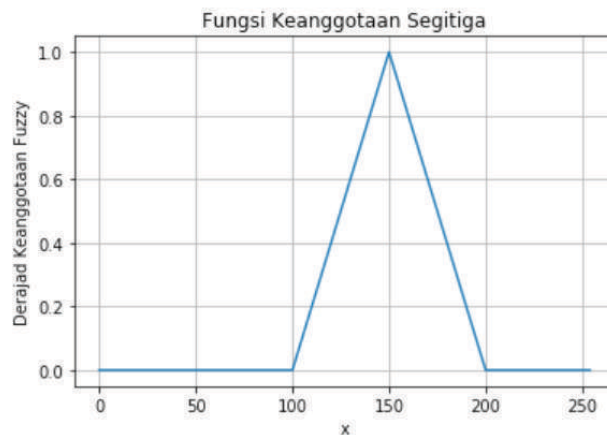
Contoh pembuatan fungsi keanggotaan segitiga dengan python, dengan nilai variabel  $x = 0-255$ , dan parameter  $a, b, c$  masing-masing 100, 150, 200, seperti terlihat pada skrip berikut ini.

```
import numpy as np
import matplotlib.pyplot as plt
import skfuzzy as fuzz

x = np.arange(0, 255, 1)
a = 100
b = 150
c = 200
y = fuzz.trimf(x, [a, b, c])

plt.plot(y)
plt.xlabel('x')
plt.ylabel('Derajat Keanggotaan Fuzzy')
plt.title('Fungsi Keanggotaan Segitiga')
plt.grid(True)
```

Jika dijalankan, hasilnya seperti terlihat pada Gambar 2.29.



Gambar 2.29. Fungsi Keanggotaan Segitiga



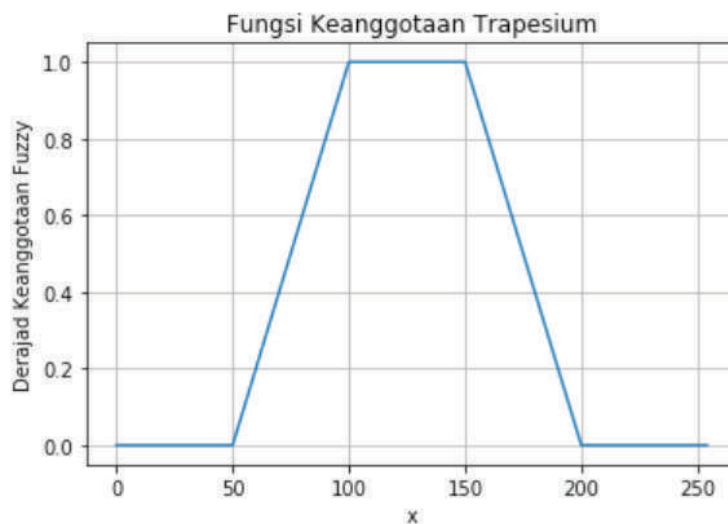
### 2.2.2. Fungsi Keanggotaan Trapesium

Contoh pembuatan fungsi keanggotaan trapesium dengan python, dengan nilai variabel  $x = 0-255$ , dan parameter  $a, b, c, d$  masing-masing 50, 100, 150, 200, seperti terlihat pada skrip berikut ini.

```
x = np.arange(0, 255, 1)
a = 50
b = 100
c = 150
d = 200
y = fuzz.trapmf(x, [a, b, c, d])

plt.plot(y)
plt.xlabel('x')
plt.ylabel('Derajat Keanggotaan Fuzzy')
plt.title('Fungsi Keanggotaan Trapesium')
plt.grid(True)
```

Jika dijalankan, hasilnya seperti terlihat pada Gambar 2.30.



Gambar 2.30. Fungsi Keanggotaan Trapesium

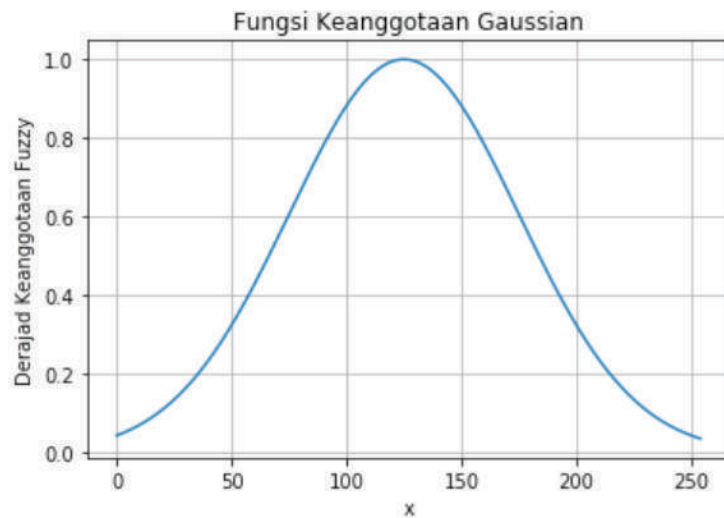
### 2.2.3. Fungsi Keanggotaan Gaussian

Contoh pembuatan fungsi keanggotaan gaussian dengan python, dengan nilai variabel  $x = 0-255$ , dan parameter  $a, b$  masing-masing 125, 50, seperti terlihat pada skrip berikut ini.

```
x = np.arange(0, 255, 1)
a = 125
b = 50
y = fuzz.gaussmf(x, a, b)
```

```
plt.plot(y)
plt.xlabel('x')
plt.ylabel('Derajat Keanggotaan Fuzzy')
plt.title('Fungsi Keanggotaan Gaussian')
plt.grid(True)
```

Jika dijalankan, hasilnya seperti terlihat pada Gambar 2.31.



Gambar 2.31. Fungsi Keanggotaan Gaussian

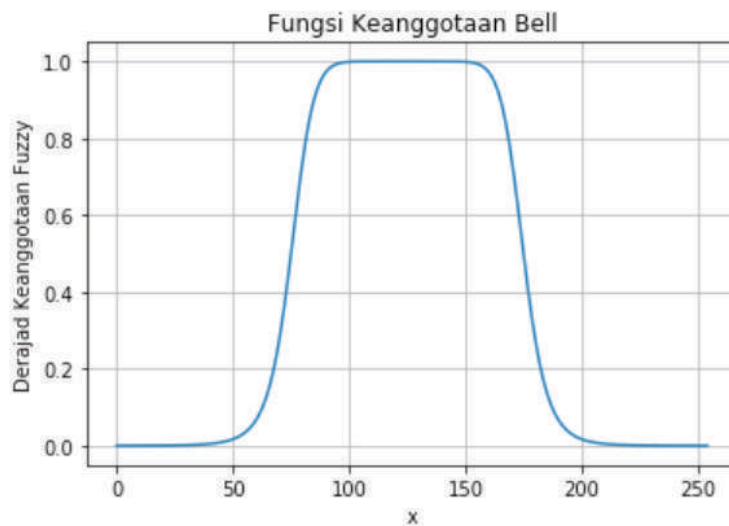
#### 2.2.4. Fungsi Keanggotaan Bell

Contoh pembuatan fungsi keanggotaan bell dengan python, dengan nilai variabel  $x = 0-255$ , dan parameter  $a, b, c$ , masing-masing 50, 5, 125, seperti terlihat pada skrip berikut ini.

```
x = np.arange(0, 255, 1)
a = 50
b = 5
c = 125
y = fuzz.gbellmf(x, a, b, c)
```

```
plt.plot(y)
plt.xlabel('x')
plt.ylabel('Derajat Keanggotaan Fuzzy')
plt.title('Fungsi Keanggotaan Bell')
plt.grid(True)
```

Jika dijalankan, hasilnya seperti terlihat pada Gambar 2.32.



Gambar 2.32. Fungsi Keanggotaan Bell

## 2.3. Defuzzifikasi dengan Python

Seperti apa yang telah dijabarkan di Bab 1, beberapa defuzzifikasi yang sering digunakan antara lain: *centroid*, *bisector*, *mean of maximum*, *smallest of maximum*, *largest of maximum*. Untuk pemrograman defuzzifikasi atau mengembalikan dari nilai fuzzy ke krisp, menggunakan python, silahkan diketikkan skrip berikut.

```

import numpy as np
import matplotlib.pyplot as plt
import skfuzzy as fuzz

# Hasilkan fungsi keanggotaan trapesium pada rentang [0, 1]
x = np.arange(0, 5.05, 0.1)
mfx = fuzz.trapmf(x, [2, 2.5, 3, 4.5])

# Defuzzifikasi menggunakan lima cara
defuzz_centroid = fuzz.defuzz(x, mfx, 'centroid')
defuzz_bisector = fuzz.defuzz(x, mfx, 'bisector')
defuzz_mom = fuzz.defuzz(x, mfx, 'mom')
defuzz_som = fuzz.defuzz(x, mfx, 'som')
defuzz_lom = fuzz.defuzz(x, mfx, 'lom')

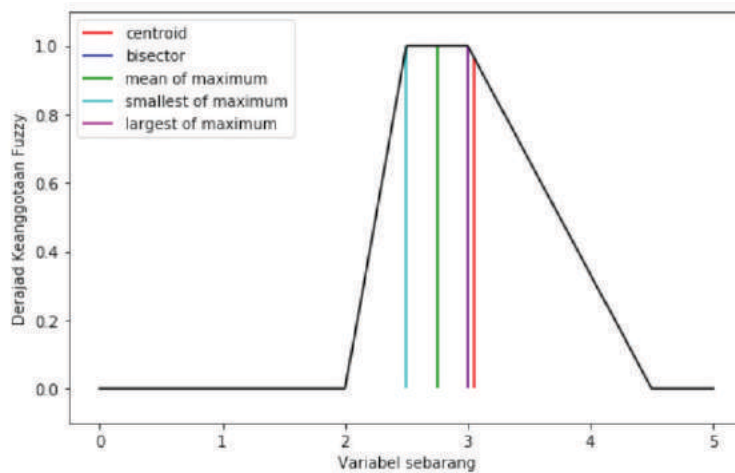
# Kumpulkan info untuk garis vertikal
labels = ['centroid', 'bisector', 'mean of maximum', 'smallest of maximum',
          'largest of maximum']
xvals = [defuzz_centroid,
          defuzz_bisector,
          defuzz_mom,
          defuzz_som,
          defuzz_lom]
colors = ['r', 'b', 'g', 'c', 'm']
ymax = [fuzz.interp_membership(x, mfx, i) for i in xvals]

# Tampilkan dan bandingkan hasil defuzzifikasi
plt.figure(figsize=(8, 5))
plt.plot(x, mfx, 'k')

for xv, y, label, color in zip(xvals, ymax, labels, colors):
    plt.vlines(xv, 0, y, label=label, color=color)
plt.ylabel('Derajat Keanggotaan Fuzzy')
plt.xlabel('Variabel sebarang')
plt.ylim(-0.1, 1.1)
plt.legend(loc=2)
plt.show()

```

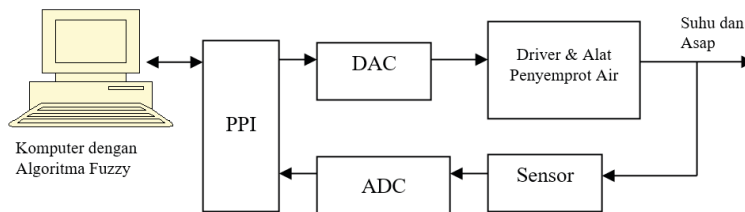
Jika dijalankan, hasilnya seperti terlihat pada Gambar 2.33.



Gambar 2.33. Macam-macam defuzzifikasi

## 2.4. Contoh Simulator Fuzzy dengan Python

Berikut ini contoh pemrograman fuzzy untuk Simulator Pemadam Kebakaran Berbasis Fuzzy. Simulator ini dirancang seperti Gambar 2.34.



Gambar 2.34. Contoh simulator fuzzy

Dari Gambar 2.34 dapat dijelaskan sebagai berikut. Sensor suhu dan sensor asap akan mengubah besaran suhu dan asap menjadi tegangan. Tegangan akan diubah menjadi digital oleh rangkaian *Analog to Digital Converter* (ADC). Komputer akan membaca sinyal tegangan tersebut melalui port masukan dari *Programmable Pheriperal Interface* (PPI) 8255. Dari nilai yang terbaca ini digunakan sebagai masukan dari sistem fuzzy. Selanjutnya dengan algoritma fuzzy diputuskan seberapa besar tegangan (dalam digital) yang harus diumpankan oleh PPI melalui port keluaran ke rangkaian *Digital to Analog Converter* (DAC). Selanjutnya keluaran DAC akan menyalakan penggerak (driver) dari alat penyemprot air.

Penyelesaian Simulator Pemadam Kebakaran Berbasis Fuzzy ini, terdiri dari tiga tahap, yaitu:

- 1). Fuzzifikasi
- 2). Pembuatan Basis Aturan, dan
- 3). Defuzzifikasi.

Masing-masing dapat dijelaskan sebagai berikut:

Program fuzzifikasi digunakan untuk mengubah besaran masukan berupa suhu dan asap dalam digital (hasil pembacaan ADC) kedalam besaran fuzzy. Serta besaran keluaran berupa banyak sedikitnya air hasil keluaran alat semprot setelah mendapatkan tegangan dari rangkaian DAC.

Fungsi keanggotaan Suhu, besaran fuzzy yang digunakan adalah Suhu Rendah, dan Suhu Tinggi. Fungsi keanggotaan Asap, besaran fuzzy yang digunakan adalah Asap Sedikit, dan Asap Banyak. Dan fungsi keanggotaan Semprotan, besaran fuzzy yang digunakan adalah Semprotan Kecil, dan Semprotan Besar.

Basis Aturan inferensi fuzzy yang digunakan, yaitu:

- IF Suhu IS Tinggi AND Asap IS Banyak THEN Semprotan IS Besar
- IF Suhu IS Tinggi AND Asap IS Sedikit THEN Semprotan IS Besar
- IF Suhu IS Rendah AND Asap IS Banyak THEN Semprotan IS Kecil
- IF Suhu IS Rendah AND Asap IS Sedikit THEN Semprotan IS Kecil

Jika Defuzzifikasi yang digunakan adalah centroid, maka penyelesaian Simulator Pemadam Kebakaran Berbasis Fuzzy ini, dituliskan dalam skrip python sebagai berikut.

```
import skfuzzy as fuzz
import numpy as np
import matplotlib.pyplot as plt

# Fungsi Keanggotaan Input Output
Suhu = np.arange(0, 255, 1)
Asap = np.arange(0, 255, 1)
Semprotan = np.arange(0, 255, 1)

# Fungsi Keanggotaan Suhu
t_Suhu_Rendah = fuzz.trapmf(Suhu, [-1, 0, 100, 255])
t_Suhu_Tinggi = fuzz.trapmf(Suhu, [0, 155, 255, 256])

# Fungsi Keanggotaan Asap
t_Asap_Sedikit = fuzz.trapmf(Asap, [-1, 0, 100, 255])
t_Asap_Banyak = fuzz.trapmf(Asap, [0, 155, 255, 256])
```

```

# Fungsi Keanggotaan Semprotan
t_Semprotan_Kecil = fuzz.trapmf(Semprotan, [-1, 0, 100, 255])
t_Semprotan_Besar = fuzz.trapmf(Semprotan, [0, 155, 255, 256])

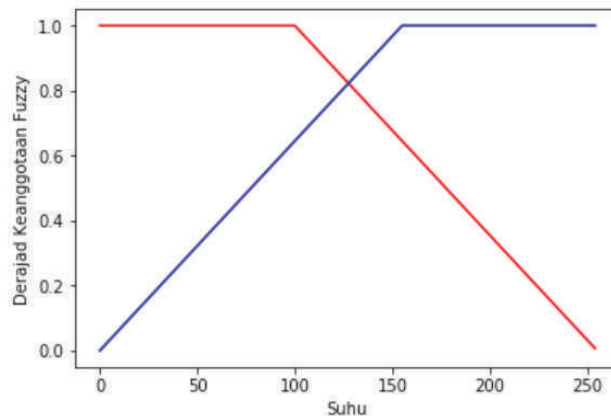
# Visualisasi Fungsi Keanggotaan Suhu
fig, ax = plt.subplots()
ax.plot(Suhu, t_Suhu_Rendah, 'r', Suhu, t_Suhu_Tinggi, 'b')
ax.set_ylabel('Derajat Keanggotaan Fuzzy')
ax.set_xlabel('Suhu')
ax.set_ylim(-0.05, 1.05);

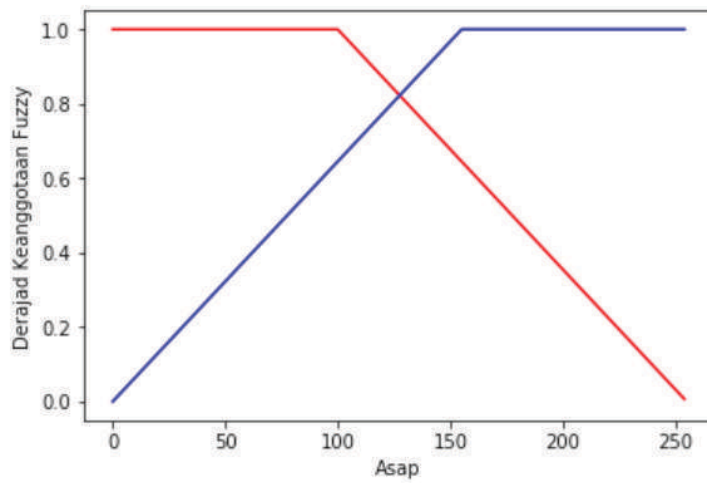
# Visualisasi Fungsi Keanggotaan Asap
fig, ax = plt.subplots()
ax.plot(Asap, t_Asap_Sedikit, 'r', Asap, t_Asap_Banyak, 'b')
ax.set_ylabel('Derajat Keanggotaan Fuzzy')
ax.set_xlabel('Asap')
ax.set_ylim(-0.05, 1.05);

# Visualisasi Fungsi Keanggotaan Semprotan
fig, ax = plt.subplots()
ax.plot(Semprotan, t_Semprotan_Kecil, 'r', Semprotan, t_Semprotan_Besar, 'b')
ax.set_ylabel('Derajat Keanggotaan Fuzzy')
ax.set_xlabel('Semprotan')
ax.set_ylim(-0.05, 1.05);

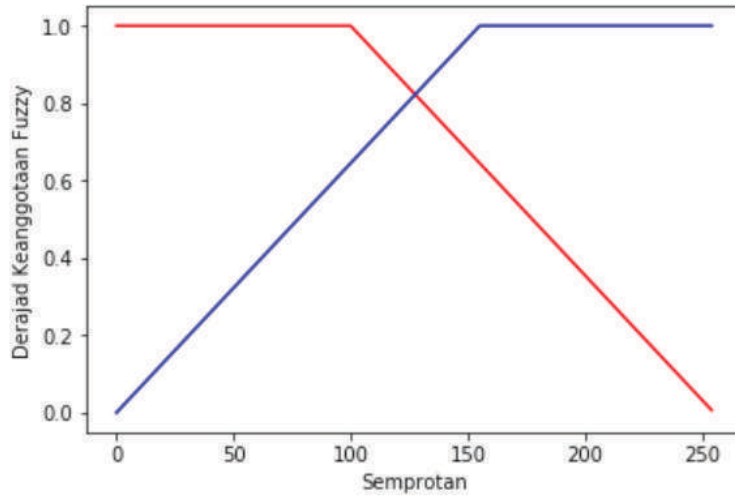
```

Jika dijalankan, hasilnya seperti terlihat pada Gambar 2.35 dan Gambar 2.36.





Gambar 2.35. Visualisasi Fungsi Keanggotaan Fuzzy Input



Gambar 2.36. Visualisasi Fungsi Keanggotaan Fuzzy Output



Kelanjutan skripnya sebagai berikut:

```
def Suhu_category(Suhu_in = 1):
    Suhu_cat_Rendah = fuzz.interp_membership(Suhu, t_Suhu_Rendah, Suhu_in)
    Suhu_cat_Tinggi = fuzz.interp_membership(Suhu, t_Suhu_Tinggi, Suhu_in)
    return dict(Rendah = Suhu_cat_Rendah, Tinggi = Suhu_cat_Tinggi)

def Asap_category(Asap_in = 2):
    Asap_cat_Sedikit = fuzz.interp_membership(Asap, t_Asap_Sedikit, Asap_in)
    Asap_cat_Banyak = fuzz.interp_membership(Asap, t_Asap_Banyak, Asap_in)
    return dict(Sedikit = Asap_cat_Sedikit, Banyak = Asap_cat_Banyak)

# Contoh input
Suhu_in = Suhu_category(100)
Asap_in = Asap_category(100)
print ("Suhu terukur"), Suhu_in
print ("Asap terukur"), Asap_in
```

Jika dijalankan, hasilnya seperti terlihat pada Gambar 2.37.

```
Suhu terukur
Asap terukur

Out[23]: (None, {'Sedikit': 1.0, 'Banyak': 0.6451612903225806})
```

Gambar 2.37. Keluaran ujicoba Suhu dan asap terukur

Selanjutnya, basis aturan fuzzy dan proses defuzzifikasi untuk mencari nilai DAC yang akan dikirimkan ke Driver Semprotan dituliskan skripnya sebagai berikut:

```

rule1 = np.fmin(Suhu_in['Tinggi'], Asap_in['Banyak'])
rule2 = np.fmin(Suhu_in['Tinggi'], Asap_in['Sedikit'])
rule3 = np.fmin(Suhu_in['Rendah'], Asap_in['Banyak'])
rule4 = np.fmin(Suhu_in['Rendah'], Asap_in['Sedikit'])

imp1 = np.fmin(rule1, t_Semprotan_Besar)
imp2 = np.fmin(rule2, t_Semprotan_Besar)
imp3 = np.fmin(rule3, t_Semprotan_Kecil)
imp4 = np.fmin(rule4, t_Semprotan_Kecil)

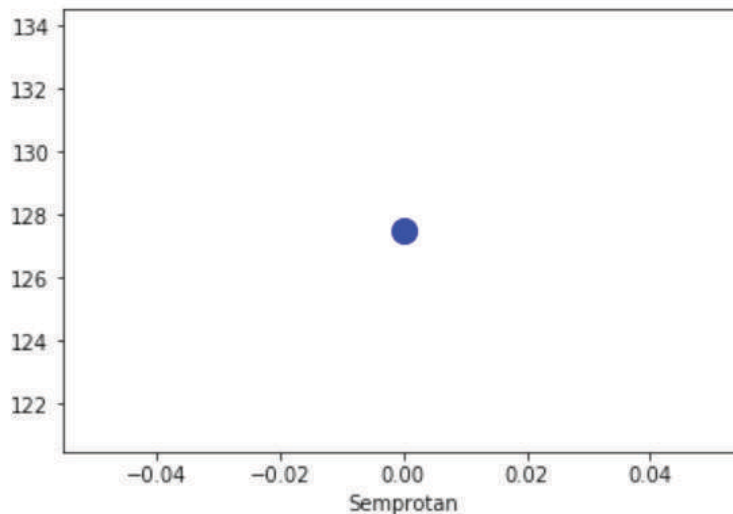
# Hitung agregat semua output
aggregate_membership = np.fmin(imp1, np.fmin(imp2, np.fmin(imp3,imp4)))

# Hitung Defuzzifikasi menggunakan centroid
result_Semprotan = fuzz.defuzz(Semprotan, aggregate_membership , 'centroid')
print (result_Semprotan)

# Visualisasi
plt.plot(result_Semprotan, 'bo', linewidth=2, markersize=12)
plt.xlabel('Semprotan');

```

Jika dijalankan, diperoleh nilai DAC yang akan dikirimkan ke Driver Semprotan, sebesar 127.49590847016553. Bentuk visualisasinya seperti terlihat pada Gambar 2.38.



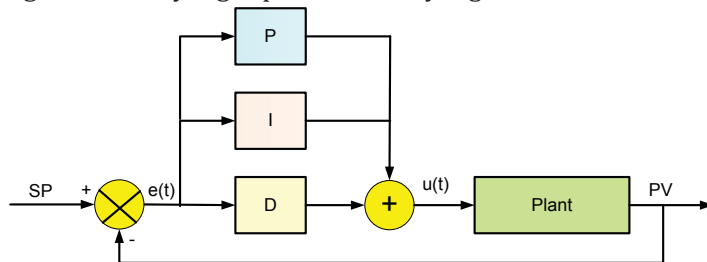
Gambar 2.38. Visualisasi keluaran Fuzzy

# BAB 3

## Sistem Kendali Cerdas Berbasis PID-Fuzzy

### 3.1. Sistem Kendali PID

Sistem Kendali di industri yang paling terkenal adalah Sistem Kendali Proporsional Integral dan Derivatif (PID). Diagram blok Sistem Kendali PID seperti diperlihatkan pada Gambar 3.1. PID menggabungkan tiga aksi kendali proporsional, integral dan derivatif. Masing-masing aksi kendali ini mempunyai keunggulan-keunggulan tertentu, dimana aksi kendali proporsional mempunyai keunggulan *rise time* yang sangat cepat, aksi kendali integral mempunyai keunggulan untuk memperkecil *error*, dan aksi kendali derivatif mempunyai keunggulan untuk memperkecil *error* atau meredam *overshoot*. Tujuan penggabungan ketiga aksi kendali ini agar dihasilkan keluaran dengan *risetime* yang cepat dan *error* yang kecil.



Gambar 3.1. Sistem Kendali PID

Pengendali PID dalam kerjanya secara otomatis menyesuaikan keluaran kendali berdasarkan perbedaan antara *Setpoint* (SP) dan variabel proses yang terukur (PV), sebagai *error* pengendalian  $e(t)$ . Nilai keluaran Pengendali  $u(t)$  ditransfer sebagai masukan sistem. Masing-masing hubungan yang digunakan seperti terlihat pada Persamaan (3.1) dan (3.2) (BYU, 2018a).

$$e(t) = SP - PV \quad (3.1)$$

$$u(t) = u_{bias} + K_c e(t) + \frac{K_c}{\tau_I} \int_0^t e(t) dt - K_c \tau_D \frac{d(PV)}{dt} \quad (3.2)$$

Istilah  $u_{bias}$  adalah konstanta yang biasanya diatur ke nilai  $u(t)$  ketika pengendali pertama beralih dari mode manual ke mode otomatis. Ini memberikan transfer “*bumpless*” jika kesalahannya nol ketika pengendali dihidupkan. Ketiga nilai penalaan atau *tuning* untuk pengendali PID adalah gain  $K_c$ , konstanta waktu integral  $\tau_I$ , dan konstanta waktu derivatif  $\tau_D$ . Nilai  $K_c$  adalah penguat *error* proporsional dan istilah integral membuat pengendali lebih agresif dalam menanggapi *error* dari *Setpoint*. Konstanta waktu integral  $\tau_I$  (juga dikenal sebagai waktu reset integral) harus positif dan memiliki satuan waktu. Karena  $\tau_I$  semakin kecil, istilah integralnya lebih besar karena  $\tau_I$  berada di penyebut. Konstanta waktu derivatif  $\tau_D$  juga memiliki satuan waktu dan harus positif.

*Setpoint* (SP) adalah nilai target, dan *variabel proses* (PV) adalah nilai terukur yang mungkin menyimpang dari nilai yang diinginkan. Kesalahan dari setpoint adalah perbedaan antara SP dan PV dan didefinisikan sebagai *error*,  $e(t) = SP - PV$ .

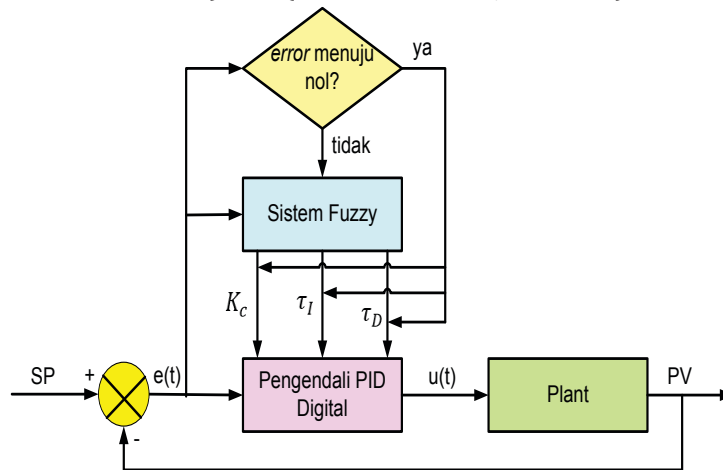
Selanjutnya, untuk keperluan implementasi digunakan Pengendali PID Diskrit. Pengendali digital diimplementasikan dengan periode *sampling diskrit*. Bentuk terpisah dari persamaan PID diperlukan untuk memperkirakan integral dan derivatif dari *error*. Modifikasi ini menggantikan bentuk kontinu integral dengan penjumlahan *error* dan menggunakan  $\Delta t$  sebagai waktu antara contoh pengambilan sampel dan  $n_t$  sebagai jumlah contoh pengambilan sampel. Ini juga menggantikan derivatif dengan versi turunannya atau metode lain yang disaring untuk memperkirakan kemiringan instan (PV). Persamaan (3.2) jika dinyatakan kedalam bentuk digital seperti diperlihatkan pada Persamaan (3.3) (BYU, 2018a).

$$u(t) = u_{bias} + K_c e(t) + \frac{K_c}{\tau_I} \sum_{i=1}^{n_t} e_i(t) \Delta t - K_c \tau_D \frac{PV_{n_t} - PV_{n_t-1}}{\Delta t} \quad (3.3)$$

Dari Persamaan (3.3) dapat dilihat tiga parameter penentu keberhasilan proses kendali, yaitu gain  $K_c$ , konstanta waktu integral  $\tau_I$  dan konstanta waktu derivatif  $\tau_D$ . Proses pencarian atau pengaturan atau penalaan agar diperoleh nilai  $K_c$ ,  $\tau_I$  dan  $\tau_D$  terbaik ini umumnya disebut dengan proses penalaan atau *tuning*.

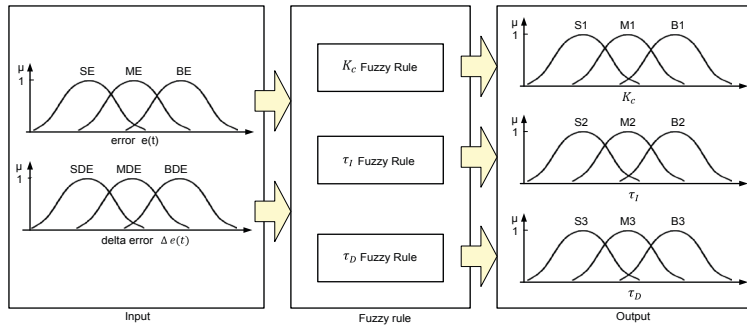
## 3.2. Sistem Kendali PID-Fuzzy

Proses penalaan nilai  $K_c$ ,  $\tau_i$  dan  $\tau_d$  pada pengendali PID memberikan peluang pengaturan secara manual maupun pengaturan secara terprogram menggunakan bantuan algoritma sistem cerdas. Salah satu algoritma sistem cerdas yang memungkinkan untuk kebutuhan ini, yaitu menggunakan sistem fuzzy. Proses penalaan nilai  $K_c$ ,  $\tau_i$  dan  $\tau_d$  pada pengendali PID menggunakan sistem fuzzy salah satu yang memungkinkan yaitu dengan cara seperti diperlihatkan pada Gambar 3.2. Ide penggabungan PID dengan Fuzzy yang lain, dapat dibaca pada literatur-literatur berikut: (Dettori, Iannino, Colla, & Signorini, 2018), (Fan, Yu, Yan, & Hong, 2018), (Nayak, Shaw, & Sahu, 2018), dan (Verma, Manik, & Jain, 2018).



Gambar 3.2. Proses penalaan nilai  $K_c$ ,  $\tau_i$  dan  $\tau_d$  pada pengendali PID menggunakan sistem fuzzy

Proses penalaan nilai  $K_c$ ,  $\tau_i$  dan  $\tau_d$  pada pengendali PID menggunakan sistem *fuzzy* seperti pada Gambar 3.2, membutuhkan mekanisme penyesuaian nilai  $K_c$ ,  $\tau_i$  dan  $\tau_d$ , berdasarkan pembacaan *error*  $e(t)$  dan *delta\_error*  $\Delta e(t)$ . Pembacaan *error* dan *delta\_error* digunakan untuk memutuskan apakah perlu dilakukan perubahan nilai  $K_c$ ,  $\tau_i$  dan  $\tau_d$  atau tidak. Jika *error* sudah konvergen ke arah nol, maka nilai  $K_c$ ,  $\tau_i$  dan  $\tau_d$  yang sudah ada dipertahankan. Namun jika masih jauh dari konvergen ke arah nol, maka perlu dilakukan proses penalaan mengikuti aturan (*rule*) yang sudah dibuat. Sistem fuzzy untuk keperluan ini diperlihatkan pada Gambar 3.3.



Gambar 3.3. Sistem Fuzzy untuk penalaan nilai  $K_c$ ,  $\tau_I$  dan  $\tau_D$

Dari Gambar 3.3, istilah-istilah yang digunakan diuraikan sebagai berikut. *Masukan error  $e(t)$* , terdiri dari *Small Error (SE)*, *Medium Error (ME)*, dan *Big Error (BE)*. *Masukan delta\_error  $\Delta e(t)$* , terdiri dari *Small Delta Error (SDE)*, *Medium Delta Error (MDE)*, dan *Big Delta Error (BDE)*. Keluaran nilai  $K_c$ ,  $\tau_I$  dan  $\tau_D$  masing-masing untuk kecil (*Small*), sedang (*Medium*) dan besar (*Big*) dinyatakan dalam S1, M1, B1, S2, M2, B2, S3, M3, dan B3. Basis Aturan sistem fuzzy untuk penalaan nilai  $K_c$ ,  $\tau_I$  dan  $\tau_D$  diperlihatkan pada Tabel 3.1.

Tabel 3.1. Basis Aturan

Rule	Input		Output		
	error	delta error	$K_c$	$\tau_I$	$\tau_D$
1	SE	SDE	S1	S2	S3
2	SE	MDE	S1	S2	S3
3	SE	BDE	S1	S2	S3
4	ME	SDE	M1	M2	M3
5	ME	MDE	M1	M2	M3
6	ME	BDE	M1	M2	M3
7	BE	SDE	B1	B2	B3
8	BE	MDE	B1	B2	B3
9	BE	BDE	B1	B2	B3

### 3.3. Pemrograman Sistem Kendali PID

Untuk merealisasikan bagaimana proses penalaan nilai  $K_c$ ,  $\tau_I$  dan  $\tau_D$  dari pengendali PID, terlebih dahulu dibuat simulasi sistem kendali PID. Proses penalaan nilai  $K_c$ ,  $\tau_I$  dan  $\tau_D$  dilakukan secara manual. Pemrograman Python untuk simulasi pengendalian

menggunakan pengendali PID dengan proses penalaan nilai  $K_c$ ,  $\tau_i$  dan  $\tau_d$  secara manual diperlihatkan pada skrip program berikut.

```
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
from scipy.integrate import odeint
import ipywidgets as wg
from IPython.display import display

n = 100 # time points to plot
tf = 50.0 # final time
SP_start = 2.0 # time of set point change

def process(y,t,u):
    Kp = 4.0
    taup = 3.0
    thetap = 1.0
    if t<(thetap+SP_start):
        dydt = 0.0 # time delay
    else:
        dydt = (1.0/taup) * (-y + Kp * u)
    return dydt

def pidPlot(Kc,taul,tauD):
    t = np.linspace(0,tf,n) # create time vector
    P= np.zeros(n) # initialize proportional term
    I = np.zeros(n) # initialize integral term
    D = np.zeros(n) # initialize derivative term
    e = np.zeros(n) # initialize error
    OP = np.zeros(n) # initialize controller output
    PV = np.zeros(n) # initialize process variable
    SP = np.zeros(n) # initialize setpoint
    SP_step = int(SP_start/(tf/(n-1))+1) # setpoint start
    SP[0:SP_step] = 0.0 # define setpoint
    SP[SP_step:n] = 4.0 # step up
    y0 = 0.0 # initial condition
```

```

# loop through all time steps
for i in range(1,n):
    # simulate process for one time step
    ts = [t[i-1],t[i]]      # time interval
    y = odeint(process,y0,ts,args=(OP[i-1],)) # compute next step
    y0 = y[1]               # record new initial condition
    # calculate new OP with PID
    PV[i] = y[1]            # record PV
    e[i] = SP[i] - PV[i]    # calculate error = SP - PV
    dt = t[i] - t[i-1]      # calculate time step
    P[i] = Kc * e[i]         # calculate proportional term
    I[i] = I[i-1] + (Kc/taul) * e[i] * dt # calculate integral term
    D[i] = -Kc * tauD * (PV[i]-PV[i-1])/dt # calculate derivative term
    OP[i] = P[i] + I[i] + D[i] # calculate new controller output

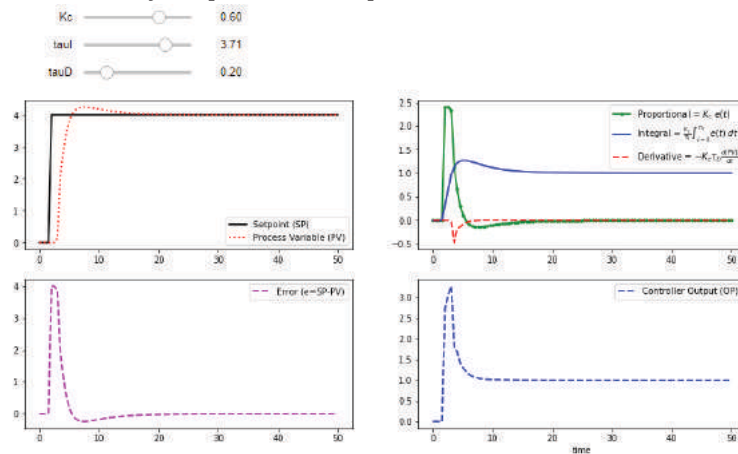
# plot PID response
plt.figure(1,figsize=(15,7))
plt.subplot(2,2,1)
plt.plot(t,SP,'k-',linewidth=2,label='Setpoint (SP)')
plt.plot(t,PV,'r:',linewidth=2,label='Process Variable (PV)')
plt.legend(loc='best')
plt.subplot(2,2,2)
plt.plot(t,P,'g.-',linewidth=2,label=r'Proportional = $K_c \cdot e(t)$')
plt.plot(t,I,'b-',linewidth=2,label=r'Integral = $\frac{K_c}{\tau_I} \int_{t=0}^{t_n} e(t) \cdot dt$')
plt.plot(t,D,'r--',linewidth=2,label=r'Derivative = $-K_c \tau_D \frac{d(PV)}{dt}$')
plt.legend(loc='best')
plt.subplot(2,2,3)
plt.plot(t,e,'m--',linewidth=2,label='Error (e=SP-PV)')
plt.legend(loc='best')
plt.subplot(2,2,4)
plt.plot(t,OP,'b-',linewidth=2,label='Controller Output (OP)')
plt.legend(loc='best')
plt.xlabel('time')

Kc_slide = wg.FloatSlider(value=0.1,min=-0.2,max=1.0,step=0.05)
taul_slide = wg.FloatSlider(value=4.0,min=0.01,max=5.0,step=0.1)
tauD_slide = wg.FloatSlider(value=0.0,min=0.0,max=1.0,step=0.1)
wg.interact(pidPlot, Kc=Kc_slide, taul=taul_slide, tauD=tauD_slide)

```



Jika dijalankan, hasilnya seperti terlihat pada Gambar 3.4.



Gambar 3.4. Hasil proses penalaan nilai  $K_c$ ,  $\tau_i$  dan  $\tau_d$  pengendali PID terhadap keluaran sistem

### 3.4. Pemrograman Sistem Kendali PID-Fuzzy

Tujuan pemanfaatan sistem fuzzy, adalah bagaimana melakukan proses penalaan nilai  $K_c$ ,  $\tau_i$  dan  $\tau_d$  pada pengendali PID, agar diperoleh hasil pengendalian yang terbaik. Hasil pengendalian yang baik ditunjukkan dengan penurunan error yang signifikan. Contoh implementasi pemrograman python untuk proses penalaan nilai  $K_c$ ,  $\tau_i$  dan  $\tau_d$  pada pengendali PID menggunakan sistem fuzzy pada Gambar 3.3 dan Basis Aturan seperti pada Tabel 3.1, diperlihatkan pada skrip program berikut ini.

```
# Import NumPy and scikit-fuzzy
import numpy as np
import skfuzzy as fuzz

# Generate universe functions
ERROR = np.arange(0, 5, 0.01)
DELTA_ERROR = np.arange(0, 5, 0.01)

Kc = np.arange(-0.2, 1.0, 0.01)
tauI = np.arange(0.01, 5.0, 0.01)
tauD = np.arange(0.0, 1.0, 0.01)

# Membership functions for ERROR
SE = fuzz.gaussmf(ERROR, 0.01, 0.8495)
ME = fuzz.gaussmf(ERROR, 2.5, 0.8495)
BE = fuzz.gaussmf(ERROR, 5, 0.8495)
```

```

# Membership functions for DELTA_EROR
SDE = fuzz.gaussmf(DELTA_ERROR, 0.01, 0.8495)
MDE = fuzz.gaussmf(DELTA_ERROR, 2.5, 0.8495)
BDE = fuzz.gaussmf(DELTA_ERROR, 5, 0.8495)

# Membership functions for OUTPUT_Kc, OUTPUT_tauI, OUTPUT_tauD
S1 = fuzz.gaussmf(Kc, -0.2, 0.2039)
M1 = fuzz.gaussmf(Kc, 0.4, 0.2039)
B1 = fuzz.gaussmf(Kc, 1, 0.2039)

S2 = fuzz.gaussmf(tauI, 0.01, 0.8476)
M2 = fuzz.gaussmf(tauI, 2.505, 0.8476)
B2 = fuzz.gaussmf(tauI, 5, 0.8476)

S3 = fuzz.gaussmf(tauD, 6.939e-18, 0.1699)
M3 = fuzz.gaussmf(tauD, 0.5, 0.1699)
B3 = fuzz.gaussmf(tauD, 1, 0.1699)

def ERROR_category(ERROR_in = 5):
    ERROR_cat_SMALL = fuzz.interp_membership(ERROR, SE, ERROR_in)
    ERROR_cat_MEDIUM = fuzz.interp_membership(ERROR, ME, ERROR_in)
    ERROR_cat_BIG = fuzz.interp_membership(ERROR, BE, ERROR_in)
    return dict(SMALL_ERROR = ERROR_cat_SMALL, MEDIUM_ERROR = ERROR_cat_MEDIUM, BIG_ERROR = ERROR_cat_BIG)

def DELTA_ERROR_category(DELTA_ERROR_in = 5):
    DELTA_ERROR_cat_SMALL = fuzz.interp_membership(DELTA_ERROR, SDE, DELTA_ERROR_in)
    DELTA_ERROR_cat_MEDIUM = fuzz.interp_membership(DELTA_ERROR, MDE, DELTA_ERROR_in)
    DELTA_ERROR_cat_BIG = fuzz.interp_membership(DELTA_ERROR, BDE, DELTA_ERROR_in)
    return dict(SMALL_DELTA_ERROR = DELTA_ERROR_cat_SMALL, MEDIUM_DELTA_ERROR = DELTA_ERROR_cat_MEDIUM, BIG_DELTA_ERROR = DELTA_ERROR_cat_BIG)

```

```

# RULE for OUTPUT_Kc
rule1 = np.fmax(ERROR_in['SMALL_ERROR'], DELTA_ERROR_in['SMALL_DELTA_ERROR'])
rule2 = np.fmax(ERROR_in['SMALL_ERROR'], DELTA_ERROR_in['MEDIUM_DELTA_ERROR'])
rule3 = np.fmax(ERROR_in['SMALL_ERROR'], DELTA_ERROR_in['BIG_DELTA_ERROR'])
rule4 = np.fmax(ERROR_in['MEDIUM_ERROR'], DELTA_ERROR_in['SMALL_DELTA_ERROR'])
rule5 = np.fmax(ERROR_in['MEDIUM_ERROR'], DELTA_ERROR_in['MEDIUM_DELTA_ERROR'])
rule6 = np.fmax(ERROR_in['MEDIUM_ERROR'], DELTA_ERROR_in['BIG_DELTA_ERROR'])
rule7 = np.fmax(ERROR_in['BIG_ERROR'], DELTA_ERROR_in['SMALL_DELTA_ERROR'])
rule8 = np.fmax(ERROR_in['BIG_ERROR'], DELTA_ERROR_in['MEDIUM_DELTA_ERROR'])
rule9 = np.fmax(ERROR_in['BIG_ERROR'], DELTA_ERROR_in['BIG_DELTA_ERROR'])

# RULE for tau_I
rule10 = np.fmax(ERROR_in['SMALL_ERROR'], DELTA_ERROR_in['SMALL_DELTA_ERROR'])
rule11 = np.fmax(ERROR_in['SMALL_ERROR'], DELTA_ERROR_in['MEDIUM_DELTA_ERROR'])
rule12 = np.fmax(ERROR_in['SMALL_ERROR'], DELTA_ERROR_in['BIG_DELTA_ERROR'])
rule13 = np.fmax(ERROR_in['MEDIUM_ERROR'], DELTA_ERROR_in['SMALL_DELTA_ERROR'])
rule14 = np.fmax(ERROR_in['MEDIUM_ERROR'], DELTA_ERROR_in['MEDIUM_DELTA_ERROR'])
rule15 = np.fmax(ERROR_in['MEDIUM_ERROR'], DELTA_ERROR_in['BIG_DELTA_ERROR'])
rule16 = np.fmax(ERROR_in['BIG_ERROR'], DELTA_ERROR_in['SMALL_DELTA_ERROR'])
rule17 = np.fmax(ERROR_in['BIG_ERROR'], DELTA_ERROR_in['MEDIUM_DELTA_ERROR'])
rule18 = np.fmax(ERROR_in['BIG_ERROR'], DELTA_ERROR_in['BIG_DELTA_ERROR'])

```

```

rule16 = np.fmax(ERROR_in['BIG_ERROR'], DELTA_ERROR_in['SMALL_DELTA_
ERROR'])
rule17 = np.fmax(ERROR_in['BIG_ERROR'], DELTA_ERROR_in['MEDIUM_DELTA_
ERROR'])
rule18 = np.fmax(ERROR_in['BIG_ERROR'], DELTA_ERROR_in['BIG_DELTA_
ERROR'])

# RULE for tau_D
rule19 = np.fmax(ERROR_in['SMALL_ERROR'], DELTA_ERROR_in['SMALL_DELTA_
ERROR'])
rule20 = np.fmax(ERROR_in['SMALL_ERROR'], DELTA_ERROR_in['MEDIUM_
DELTA_ERROR'])
rule21 = np.fmax(ERROR_in['SMALL_ERROR'], DELTA_ERROR_in['BIG_DELTA_
ERROR'])
rule22 = np.fmax(ERROR_in['MEDIUM_ERROR'], DELTA_ERROR_in['SMALL_
DELTA_ERROR'])
rule23 = np.fmax(ERROR_in['MEDIUM_ERROR'], DELTA_ERROR_in['MEDIUM_
DELTA_ERROR'])
rule24 = np.fmax(ERROR_in['MEDIUM_ERROR'], DELTA_ERROR_in['BIG_DELTA_
ERROR'])
rule25 = np.fmax(ERROR_in['BIG_ERROR'], DELTA_ERROR_in['SMALL_DELTA_
ERROR'])
rule26 = np.fmax(ERROR_in['BIG_ERROR'], DELTA_ERROR_in['MEDIUM_DELTA_
ERROR'])
rule27 = np.fmax(ERROR_in['BIG_ERROR'], DELTA_ERROR_in['BIG_DELTA_
ERROR'])

# IMPLIKASI for Kc
imp1 = np.fmax(rule1, S1)
imp2 = np.fmax(rule2, S1)
imp3 = np.fmax(rule3, S1)
imp4 = np.fmax(rule4, M1)
imp5 = np.fmax(rule5, M1)
imp6 = np.fmax(rule6, M1)
imp7 = np.fmax(rule7, B1)
imp8 = np.fmax(rule7, B1)
imp9 = np.fmax(rule7, B1)

```

```

# IMPLIKASI for tauI
imp10 = np.fmax(rule10, S2)
imp11 = np.fmax(rule11, S2)
imp12 = np.fmax(rule12, S2)
imp13 = np.fmax(rule13, M2)

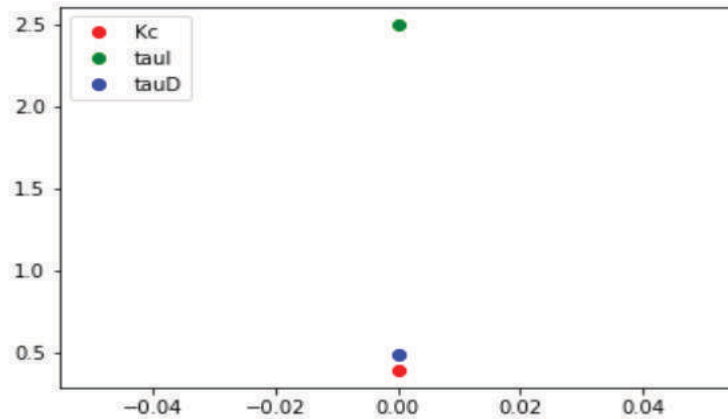
imp14 = np.fmax(rule14, M2)
imp15 = np.fmax(rule15, M2)
imp16 = np.fmax(rule16, B2)
imp17 = np.fmax(rule17, B2)
imp18 = np.fmax(rule18, B2)

# IMPLIKASI for tauD
imp19 = np.fmax(rule19, S3)
imp20 = np.fmax(rule20, S3)
imp21 = np.fmax(rule21, S3)
imp22 = np.fmax(rule22, M3)
imp23 = np.fmax(rule23, M3)
imp24 = np.fmax(rule24, M3)
imp25 = np.fmax(rule25, B3)
imp26 = np.fmax(rule26, B3)
imp27 = np.fmax(rule27, B3)

# Aggregate all output - min
aggregate_membership1 = np.fmax(imp1, np.fmax(imp2, np.fmax(imp3,
np.fmax(imp4, np.fmax(imp5, np.fmax(imp6, np.fmax(imp7,
np.fmax(imp8, imp9))))))))
aggregate_membership2 = np.fmax(imp10, np.fmax(imp11, np.fmax(imp12,
np.fmax(imp13, np.fmax(imp14, np.fmax(imp15, np.fmax(imp16,
np.fmax(imp17, imp18))))))))
aggregate_membership3 = np.fmax(imp19, np.fmax(imp20, np.fmax(imp21,
np.fmax(imp22, np.fmax(imp23, np.fmax(imp24, np.fmax(imp25,
np.fmax(imp26, imp27))))))))
# Defuzzification
result_Kc = fuzz.defuzz(Kc, aggregate_membership1 , 'centroid')
result_tauI = fuzz.defuzz(tauI, aggregate_membership2 , 'centroid')
result_tauD = fuzz.defuzz(tauD, aggregate_membership3 , 'centroid')
print (result_Kc)
print (result_tauI)
print (result_tauD)

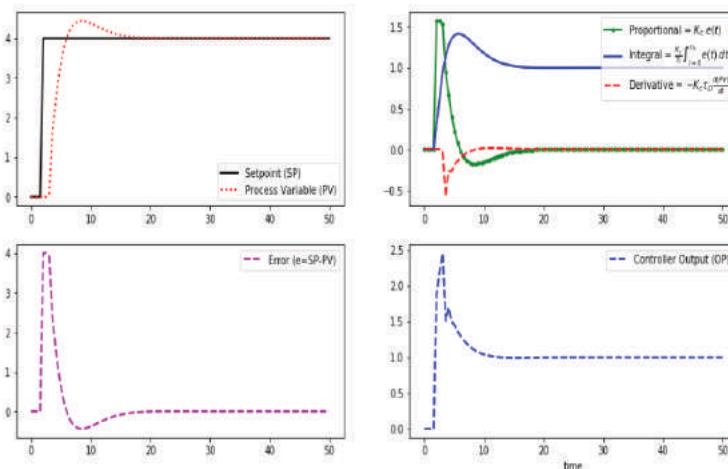
```

Dari contoh proses penalaan nilai  $K_c$ ,  $\tau_i$  dan  $\tau_d$  pada pengendali PID berbasis sistem fuzzy ini, diperoleh hasil  $K_c = 0.39331459212203296$ ,  $\tau_i = 2.4983242909359484$ , dan  $\tau_d = 0.49331226163799125$ , seperti terlihat pada Gambar 3.5.



Gambar 3.5. Hasil proses penalaan nilai  $K_c$ ,  $\tau_i$  dan  $\tau_d$  berbasis sistem fuzzy

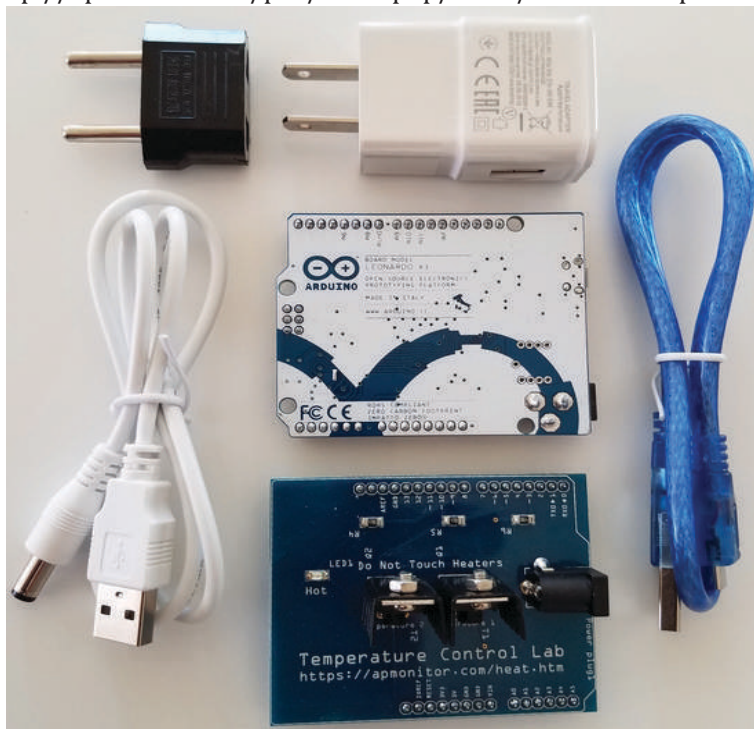
Nilai-nilai  $K_c$ ,  $\tau_i$  dan  $\tau_d$  hasil proses penalaan berbasis sistem fuzzy, selanjutnya dilihat pengaruhnya terhadap keberhasilan pengendalian. Hasil proses pengendalian menggunakan pengendali PID melalui proses penalaan nilai  $K_c$ ,  $\tau_i$  dan  $\tau_d$  berbasis sistem fuzzy diperlihatkan pada Gambar 3.6. Terlihat sistem kendali PID-Fuzzy bekerja dengan baik, ditunjukkan dengan error menuju nol.



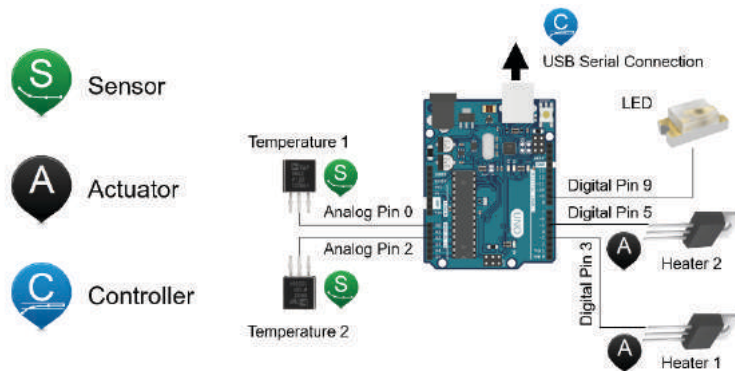
Gambar 3.6. Hasil proses penalaan nilai  $K_c$ ,  $\tau_i$  dan  $\tau_d$  pada pengendali PID berbasis sistem fuzzy terhadap keluaran sistem

### 3.5. Pemrograman TCLab

Selanjutnya, untuk keperluan uji-coba sistem kendali menggunakan plant riil, digunakan Lab Kit TCLab. TCLab adalah modul hardware kendali suhu umpan balik dengan Arduino, LED, dua pemanas, dan dua sensor suhu. Keluaran daya pemanas disesuaikan untuk mempertahankan *setpoint* suhu yang diinginkan. Energi panas dari pemanas ditransfer melalui konduksi, konveksi, dan radiasi ke sensor suhu. Panas juga dipindahkan dari perangkat ke lingkungan. TCLab juga dapat digunakan untuk identifikasi model dan pengembangan pengendali. Bentuk TCLab seperti diperlihatkan pada Gambar 3.7 berikut. Gambar skematiknya diperlihatkan pada Gambar 3.8. Informasi lebih lanjut mengenai Lab Kit TCLab ini dapat diakses pada alamat: <http://apmonitor.com/pdc/index.php/Main/ArduinoTemperatureControl>.



Gambar 3.7. TCLab



Gambar 3.8. Skematik TCLab

### 3.5.1. Pengaturan dan Instalasi

TCLab dapat bekerja dengan Bahasa Pemrograman MATLAB maupun Python. Paket tambahannya harus diinstal terlebih dahulu agar TCLab dapat digunakan. MATLAB membutuhkan paket dukungan Arduino. Python membutuhkan paket tambahan seperti paket tclab yang dapat diinstal dengan *pip* melalui Anaconda Prompt.

```
>> pip install tclab
```

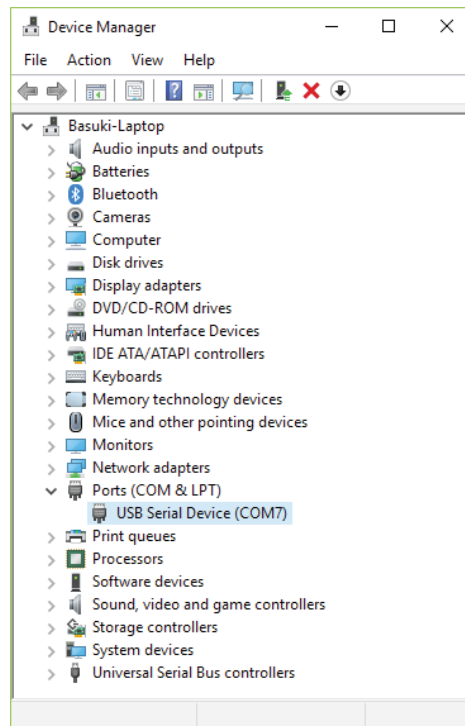
Untuk menguji apakah Lab Kit TCLab dapat bekerja dengan baik, silahkan hardware TCLab dihubungkan ke komputer (laptop). Power untuk pemanas juga dapat dicolokkan ke listrik. Seperti diperlihatkan pada Gambar 3.9.



Gambar 3.9. Hubungan TCLab ke komputer



Setelah hardware TCLab ditancapkan ke port USB komputer, silahkan cek di *Device Manager*, apakah port USB Serial Device (COM berapa) sudah muncul di Ports (COM & LPT). Seperti diperlihatkan pada Gambar 3.10, pada contoh ini muncul di COM 7.



Gambar 3.10. Tampilan USB Serial pada Device Manager

Selanjutnya silahkan diuji dengan menjalankan skrip python berikut.

```
import tclab
import time
# Connect to Arduino
a = tclab.TCLab()
print('LED On')
a.LED(100)
# Pause for 1 second
time.sleep(1.0)
print('LED Off')
a.LED(0)
a.close()
```

Jika *library* tclab belum diinstal akan muncul error seperti terlihat pada Gambar 3.11.

```

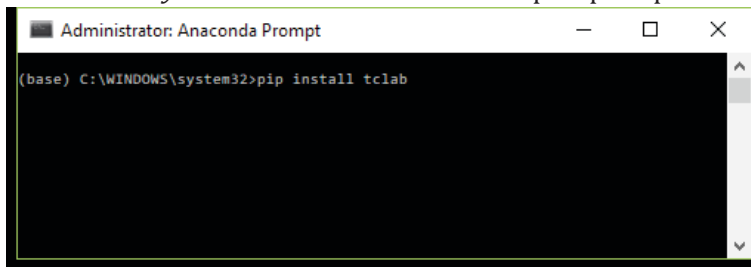
-----
AttributeError                                Traceback (most recent call last)
<ipython-input-3-611ab67d351a> in <module>
      2 import time
      3 # Connect to Arduino
----> 4 a = tclab.TCLab()
      5 print('LED On')
      6 a.LED(100)

AttributeError: module 'tclab' has no attribute 'TCLab'

```

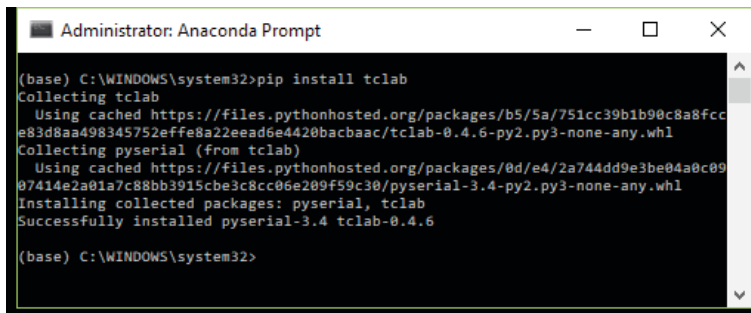
Gambar 3.11. Error jika tclab belum terinstal

Silahkan diinstal *library* tclab melalui Anaconda Prompt seperti pada Gambar 3.12.



Gambar 3.12. Instalasi library tclab

Jika tclab berhasil diinstal, maka akan muncul pesan sukses seperti terlihat pada Gambar 3.13.



Gambar 3.13. TCLab berhasil diinstal

Selanjutnya, jalankan lagi skrip di atas. Jika berhasil akan tampil seperti terlihat pada Gambar 3.14.

```

TCLab version 0.4.6
Arduino Leonardo connected on port COM7 at 115200 baud.
TCLab Firmware Version 1.01.
LED On
LED Off
TCLab disconnected successfully.

```

Gambar 3.14. Pengujian TCLab berhasil

Untuk menguji pemanasnya, ada dua pemanas (Q1 dan Q2) silahkan ketikkan skrip python berikut ini.

```
import tclab
import numpy as np
import time
# Connect to Arduino
a = tclab.TCLab()
# Temperatures
print('Temperatures')
print('Temperature 1: ' + str(a.T1) + ' degC')
print('Temperature 2: ' + str(a.T2) + ' degC')
# Turn LED on
print('LED On')
a.LED(100)
# Turn on Heaters (0-100%)
print('Turn On Heaters (Q1=90%, Q2=80%)')
a.Q1(90.0)
a.Q2(80.0)
# Sleep (sec)
time.sleep(30.0)
# Turn Off Heaters
print('Turn Off Heaters')
a.Q1(0.0)
a.Q2(0.0)
# Temperatures
print('Temperatures')
print('Temperature 1: ' + str(a.T1) + ' degC')
print('Temperature 2: ' + str(a.T2) + ' degC')
a.close()
```

Jika dijalankan, akan dihasilkan keluaran seperti terlihat pada Gambar 3.15.

```
TCLab version 0.4.6
Arduino Leonardo connected on port COM7 at 115200 baud.
TCLab Firmware Version 1.01.
Temperatures
Temperature 1: 31.86 degC
Temperature 2: 31.82 degC
LED On
Turn On Heaters (Q1=90%, Q2=80%)
Turn Off Heaters
Temperatures
Temperature 1: 37.21 degC
Temperature 2: 34.11 degC
TCLab disconnected successfully.
```

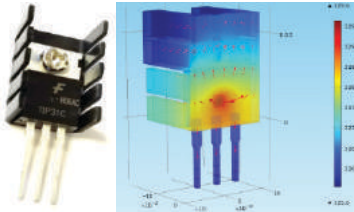
Gambar 3.15. Keluaran pengujian pemanas

### 3.5.2. Pemodelan Dinamis Sistem Pemanas

Tujuan: Menurunkan model transien nonlinear berdasarkan hukum keseimbangan energi (*energy balance*) dan dicocokkan dengan sistem orde satu linier atau orde dua linier. Kemudian hasil prediksi model dibandingkan dengan data transien riil keluaran dari hardware TCLab.

#### 3.5.2.1. Pemodelan Dinamis Keseimbangan Energi

Pemodelan pertama adalah menurunkan model dinamis sistem dengan menggunakan nilai perkiraan parameter. Tiga elemen penting untuk kendali lup tertutup TCLab yaitu perangkat pengukuran (sensor suhu termistor), aktuator (transistor), dan kemampuan untuk melakukan kendali terkomputerisasi (dengan antarmuka USB). Pada keluaran maksimum, transistor melepas daya 1 Watt pada keluaran pemanas 100%. Massa transistor dan *heat sink* dengan sirip adalah 4 gram. Diperlihatkan pada Gambar 3.16.



Gambar 3.16. Pemanas TCLab

Pemanas TCLab memiliki kapasitas panas 500 J/kgK. Luas permukaan pemanas dan sensor sekitar 12 cm<sup>2</sup>. Koefisien perpindahan panas konvektif untuk udara diam adalah sekitar 10 W/m<sup>2</sup>K. Perpindahan panas yang dihasilkan oleh transistor melalui perangkat, utamanya disebabkan oleh faktor konveksi, selain faktor radiasi. Perpindahan panas radiatif dapat dimasukkan kedalam model untuk menentukan fraksi panas yang hilang akibat konveksi dan radiasi panas. Perpindahan panas ditingkatkan dengan kopling termal yang menghubungkan dua komponen. Secara lengkap karakteristik pemanas TCLab seperti yang terdata pada Tabel 3.2.

Tabel 3.2. Karakteristik pemanas TCLab

Quantity	Value
Initial temperature ( $T_0$ )	296.15 K (23°C)
Ambient temperature ( $T_\infty$ )	296.15 K (23°C)
Heater output ( $Q$ )	0 to 1 W
Heater factor ( $\alpha$ )	0.01 W/(% heater)
Heat capacity ( $C_p$ )	500 J/kg-K

Quantity	Value
Surface Area (A)	$1.2 \times 10^{-3} \text{ m}^2$ (12 cm <sup>2</sup> )
Mass (m)	0.004 kg (4 gm)
Overall Heat Transfer Coefficient (U)	10 W/m <sup>2</sup> -K
Emissivity (ε)	0.9
Stefan Boltzmann Constant (σ)	$5.67 \times 10^{-8} \text{ W/m}^2\text{-K}^4$

Selanjutnya dilakukan pemodelan dinamis, respons dinamis antara daya masukan ke transistor dan suhu yang dirasakan oleh termistor. Digunakan keseimbangan energi untuk memulai penurunan, seperti pada Persamaan (3.4) (BYU, 2018b).

$$mc_p \frac{dT}{dt} = \sum \dot{h}_{in} - \sum \dot{h}_{out} + Q \quad (3.4)$$

Istilah-istilah yang diperlukan untuk keperluan ini diperluas atau disederhanakan. Keseimbangan energi penuh didalamnya sudah termasuk istilah konveksi dan radiasi. Persamaan (3.4) diperluas menjadi Persamaan (3.5) (BYU, 2018b).

$$mc_p \frac{dT}{dt} = UA(T_{\infty} - T) + \epsilon \sigma A(T_{\infty}^4 - T^4) + \alpha Q \quad (3.5)$$

Dimana m adalah massa,  $c_p$  adalah kapasitas panas, T adalah suhu, U adalah koefisien perpindahan panas, A adalah area,  $T_{\infty}$  adalah suhu sekitar,  $\epsilon = 0.9$  adalah emisivitas,  $\sigma = 5.67 \times 10^{-8} \text{ W/m}^2\text{K}^4$  adalah konstanta Stefan-Boltzmann, dan Q adalah persentase keluaran pemanas. Parameter  $\alpha$  adalah faktor yang menghubungkan keluaran pemanas (0-100%) dengan daya yang didisipasi oleh transistor dalam Watt. Persamaan ini dapat dikembangkan untuk simulasi respons suhu yang dinamis karena adanya dorongan (mati, hidup, mati) pada keluaran pemanas. Biarkan pemanas hidup untuk waktu yang cukup untuk mengamati kondisi yang hampir stabil.

Pemrograman python untuk menguji keluaran model keseimbangan energi, silahkan ketikkan skrip berikut.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

# define energy balance model
def heat(x,t,Q):
    # Parameters
    Ta = 23 + 273.15 # K
    U = 10.0 # W/m^2-K
    m = 4.0/1000.0 # kg
    Cp = 0.5 * 1000.0 # J/kg-K
    A = 12.0 / 100.0**2 # Area in m^2
    alpha = 0.01 # W / % heater
    eps = 0.9 # Emissivity
    sigma = 5.67e-8 # Stefan-Boltzman

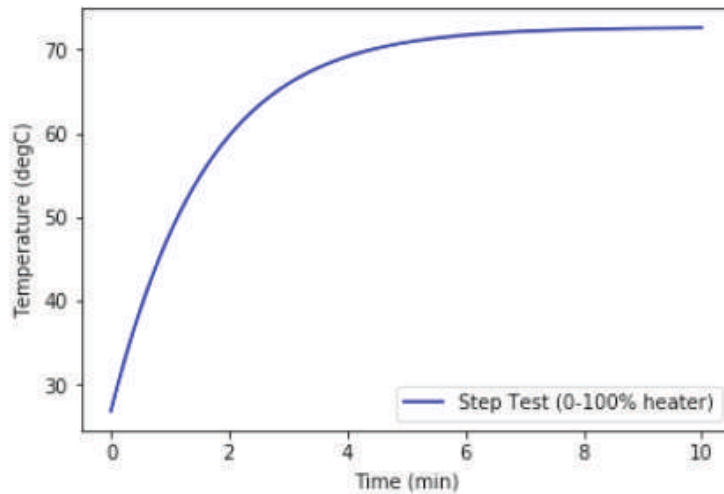
    # Temperature State
    T = x[0]

    # Nonlinear Energy Balance
    dTdt = (1.0/(m*Cp))*(U*A*(Ta-T) \
        + eps * sigma * A * (Ta**4 - T**4) \
        + alpha*Q)
    return dTdt

Q = 100.0 # Percent Heater (0-100%)
T0 = 23.0 + 273.15 # Initial temperature
n = 60*10+1 # Number of second time points (10min)
time = np.linspace(0,n-1,n) # Time vector
T = odeint(heat,300.0,time,args=(Q,)) # Integrate ODE

# Plot results
plt.figure(1)
plt.plot(time/60.0,T-273.15,'b-')
plt.ylabel('Temperature (degC)')
plt.xlabel('Time (min)')
plt.legend(['Step Test (0-100% heater)'])
plt.show()
```

Jika dijalankan, hasilnya seperti terlihat pada Gambar 3.17.



Gambar 3.17. Respon step model keseimbangan energi

Dari keluaran model keseimbangan energi berupa transien nonlinear dapat diamati apakah pengaruh perpindahan radiasi panas cukup signifikan. Nantinya apakah respon suhu ini sesuai dengan keluaran model orde satu atau orde dua, berdasarkan nilai-nilai parameter yang tidak pasti pada model berbasis fisis. Tujuan akhirnya nanti apakah model-model ini cukup membantu prediksi suhu jika dibandingkan dengan data riil keluaran langsung dari hardware TCLab.

### 3.5.2.2. Pemodelan Dinamis Orde Satu Plus Waktu-Tunda

Pemodelan kedua sistem pemanas TCLab ini dicoba didekati dengan sistem linear Model Orde Satu Plus Waktu-Tunda atau *First-Order Plus Dead-Time* (FOPDT). Sistem linear orde satu dengan waktu tunda ini adalah deskripsi empiris umum dari banyak proses dinamis yang stabil. FOPDT digunakan untuk mendapatkan konstanta penalaan pengendali awal.

Skrip python dengan *widget* interaktif berikut menunjukkan efek dari tiga parameter yang dapat ditala atau disesuaikan dalam persamaan FOPDT. Tiga parameter FOPDT yang dimaksud adalah gain  $K_p$ , konstanta waktu  $\tau_p$ , dan waktu tunda  $\theta_p$ .

```

import ipywidgets as wg
from IPython.display import display
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt

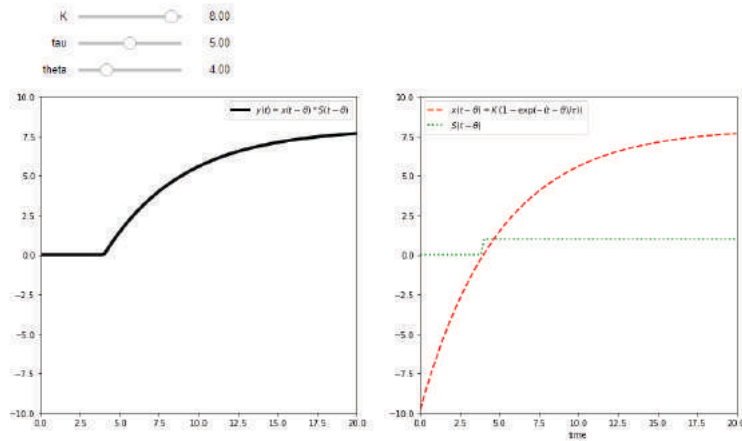
def fopdtPlot(K,tau,theta):
    n = 100 # time points to plot
    t = np.linspace(0,20,100) # create time vector
    # create 0 -> 1 step at t=theta
    delay = np.empty_like(t)
    for i in range(n):
        if t[i] < theta:
            delay[i] = 0.0
        else:
            delay[i] = 1.0
    # calculate response to step input
    x = K * (1.0-np.exp(-(t-theta)/tau))
    y = x * delay
    # plot response
    plt.figure(1,figsize=(15,7))
    plt.subplot(1,2,1)
    plt.plot(t,y,'k-',linewidth=4,label=r'$y(t)=x(t-\backslash theta)*S(t-\backslash theta)$')
    plt.legend(loc='best')
    plt.ylim([-10,10])
    plt.xlim([0,20])
    plt.subplot(1,2,2)
    plt.plot(t,x,'r--',linewidth=2,label=r'$x(t-\backslash theta)=K\;(1-\backslash exp(-(t-\backslash theta)/\backslash tau))$')
    plt.plot(t,delay,'g-',linewidth=2,label=r'$S(t-\backslash theta)$')
    plt.xlabel('time')
    plt.legend(loc='best')
    plt.ylim([-10,10])
    plt.xlim([0,20])

K_slide = wg.FloatSlider(value=8.0,min=-10.0,max=10.0,step=0.1)
tau_slide = wg.FloatSlider(value=5.0,min=0.1,max=10.0,step=0.1)
theta_slide = wg.FloatSlider(value=4.0,min=0.1,max=15.0,step=0.1)
wg.interact(fopdtPlot, K=K_slide, tau=tau_slide, theta=theta_slide)

```



Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 3.18.



Gambar 3.18. Pengaruh perubahan parameter FOPDT

Setelah mendapatkan pemahaman intuitif tentang bagaimana parameter-parameter gain  $K_p$ , konstanta waktu  $\tau_p$ , dan waktu tunda  $\theta_p$  mempengaruhi respon step, penting untuk memahami persamaan matematis FOPDT. Persamaannya diuraikan pada penjelasan berikut ini.

Persamaan FOPDT dinyatakan dalam bentuk (BYU, 2018b):

$$\tau_p \frac{dy(t)}{dt} = -y(t) + K_p u(t - \theta_p) \quad (3.6)$$

Persamaan (3.6) memiliki variabel keluaran  $y(t)$  dan masukan  $u(t)$  dan tiga parameter yang tidak diketahui yaitu gain proses  $K_p$ , konstanta waktu  $\tau_p$ , dan waktu tunda  $\theta_p$ .

### 1. Gain proses $K_p$ .

Gain proses adalah perubahan keluaran  $y$  yang diinduksi oleh perubahan satuan pada masukan  $u$ . Gain proses dihitung dengan cara mengevaluasi perubahan dalam  $y(t)$  dibagi dengan perubahan  $u(t)$  pada kondisi awal dan akhir kondisi tunak (*steady state*) (BYU, 2018b).

$$K_p = \frac{\Delta y}{\Delta u} = \frac{y_{ss2} - y_{ss1}}{u_{ss2} - u_{ss1}} \quad (3.7)$$

Gain proses mempengaruhi besarnya respon, terlepas dari kecepatan respon.

## 2. Konstanta waktu $\tau_p$ .

Diberikan perubahan  $u(t) = \Delta u$ , solusi untuk diferensial orde satu linier (tanpa waktu tunda) menjadi (BYU, 2018b):

$$y(t) = \left(e^{-\frac{t}{\tau_p}}\right)y(0) + \left(1 - e^{-\frac{t}{\tau_p}}\right)K_p\Delta u \quad (3.8)$$

Jika kondisi awal  $y(0)=0$  dan pada  $t=\tau_p$ , maka solusinya disederhanakan sebagai berikut (BYU, 2018b):

$$y(\tau_p) = \left(1 - e^{-\frac{\tau_p}{\tau_p}}\right)K_p\Delta u = (1 - e^{-1})K_p\Delta u = 0.632K_p\Delta u \quad (3.9)$$

Maka konstanta waktu proses adalah jumlah waktu yang diperlukan untuk mencapai keluaran  $(1 - \exp(-1))$  atau 63.2% menuju kondisi tunak. Konstanta waktu proses mempengaruhi kecepatan respon.

## 3. Waktu tunda $\theta_p$ .

Waktu tunda dinyatakan sebagai pergeseran waktu yang terdapat didalam variabel input  $u(t)$  (BYU, 2018b).

$$u(t - \theta_p) \quad (3.10)$$

Misalkan sinyal masukan itu adalah berupa fungsi step yang biasanya berubah dari 0 ke 1 pada waktu  $t=0$  tetapi pergeseran ini tertunda 5 detik. Fungsi masukan  $u(t)$  dan fungsi keluaran  $y(t)$  digeser waktunya 5 detik. Solusi untuk persamaan diferensial orde satu dengan waktu tunda diperoleh dengan mengganti semua variabel  $t$  dengan  $t - \theta_p$  dan menerapkan hasil kondisional berdasarkan waktu tunda  $\theta_p$  (BYU, 2018b).

$$y(t < \theta_p) = y(0) \quad (3.11)$$

$$y(t \geq \theta_p) = \left(e^{-\frac{(t-\theta_p)}{\tau_p}}\right)y(0) + \left(1 - e^{-\frac{(t-\theta_p)}{\tau_p}}\right)K_p\Delta u \quad (3.12)$$

### 3.5.2.3. Pemodelan Dinamis Orde Dua Plus Waktu-Tunda

Pemodelan ketiga sistem pemanas TCLab ini dicoba didekati dengan sistem linear Model Orde Dua Plus Waktu-Tunda atau *Second-Order Plus Dead-Time* (SOPDT). Sistem linear orde dua dengan waktu tunda ini adalah deskripsi empiris dari proses dinamis yang berpotensi berosilasi.

Persamaannya (BYU, 2018b):

$$\tau_s^2 \frac{d^2 y}{dt^2} + 2\zeta \tau_s \frac{dy}{dt} + y = K_p u(t - \theta_p) \quad (3.13)$$

Persamaan (3.13) memiliki keluaran  $y(t)$  dan masukan  $u(t)$  dan empat parameter yang tidak diketahui. Keempat parameter tersebut adalah gain  $K_p$ , faktor redaman  $\zeta$ , konstanta waktu orde dua  $\tau_s$ , dan waktu tunda  $\theta_p$ .

Alternatif untuk pendekatan penyesuaian grafik respon model adalah dengan penggunaan optimasi agar keluaran model SOPDT sesuai dengan data riil. Tujuannya untuk meminimalkan jumlah kesalahan kuadrat yang menyebabkan penyimpangan model SOPDT dari data. Algoritma optimasi mengubah parameter untuk mencocokkan data pada titik waktu yang ditentukan.

Berikut ini skrip python untuk membangkitkan data simulasi keluaran dari Model SOPDT.

```
# Generate process data as data.txt
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

# define process model (to generate process data)
def process(y,t,n,u,Kp,taup):
    # arguments
    # y[n] = outputs
    # t = time
    # n = order of the system
    # u = input value
    # Kp = process gain
    # taup = process time constant

    # equations for higher order system
    dydt = np.zeros(n)
    # calculate derivative
    dydt[0] = (-y[0] + Kp * u)/(taup/n)
    for i in range(1,n):
        dydt[i] = (-y[i] + y[i-1])/(taup/n)
    return dydt
```

```

# specify number of steps
ns = 50
# define time points
t = np.linspace(0,40,ns+1)
delta_t = t[1]-t[0]
# define input vector
u = np.zeros(ns+1)
u[5:20] = 1.0
u[20:30] = 0.1
u[30:] = 0.5

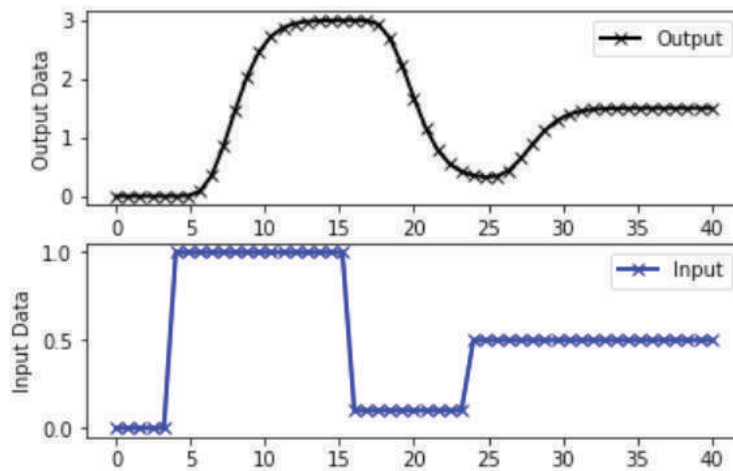
# use this function or replace yp with real process data
def sim_process_data():
    # higher order process
    n=10    # order
    Kp=3.0  # gain
    taup=5.0 # time constant
    # storage for predictions or data
    yp = np.zeros(ns+1) # process
    for i in range(1,ns+1):
        if i==1:
            yp0 = np.zeros(n)
            ts = [delta_t*(i-1),delta_t*i]
            y = odeint(process,yp0,ts,args=(n,u[i],Kp,taup))
            yp0 = y[-1]
            yp[i] = y[1][n-1]
    return yp
yp = sim_process_data()

# Construct results and save data file
# Column 1 = time
# Column 2 = input
# Column 3 = output
data = np.vstack((t,u,yp)) # vertical stack
data = data.T             # transpose data
np.savetxt('data.txt',data,delimiter=',')

```

```
# plot results
plt.figure()
plt.subplot(2,1,1)
plt.plot(t,yp,'kx-',linewidth=2,label='Output')
plt.ylabel('Output Data')
plt.legend(loc='best')
plt.subplot(2,1,2)
plt.plot(t,u,'bx-',linewidth=2)
plt.legend(['Input'],loc='best')
plt.ylabel('Input Data')
plt.show()
```

Jika dijalankan, hasilnya seperti terlihat pada Gambar 3.19.



Gambar 3.19. Keluaran Model SOPDT

### 3.5.4. Pengujian Model Pemanas

Selanjutnya untuk menguji model-model pemanas yang telah dijelaskan, dibandingkan dengan data riil keluaran dari hardware TCLab. Model-model yang dimaksud yaitu model keseimbangan energi, model orde satu linier dengan waktu tunda, dan model orde dua linier dengan waktu tunda. Hasil prediksi model selanjutnya dibandingkan dengan data transien riil keluaran dari hardware TCLab.

Berikut ini skrip python untuk membandingkan keluaran model keseimbangan energi (*energy balance*) dan orde satu linier dengan waktu tunda (FOPDT), dengan data riil keluaran dari hardware TCLab.

```
import tclab
import numpy as np
import time
import matplotlib.pyplot as plt
from scipy.integrate import odeint

# FOPDT model
Kp = 0.5    # degC/%
tauP = 120.0 # seconds
thetaP = 10 # seconds (integer)
Tss = 23    # degC (ambient temperature)
Qss = 0     # % heater

# define energy balance model
def heat(x,t,Q):
    # Parameters
    Ta = 23 + 273.15 # K
    U = 10.0         # W/m^2-K
    m = 4.0/1000.0   # kg
    Cp = 0.5 * 1000.0 # J/kg-K
    A = 12.0 / 100.0**2 # Area in m^2
    alpha = 0.01     # W / % heater
    eps = 0.9        # Emissivity
    sigma = 5.67e-8  # Stefan-Boltzman

    # Temperature State
    T = x[0]

    # Nonlinear Energy Balance
    dTdt = (1.0/(m*Cp))*(U*A*(Ta-T) \
        + eps * sigma * A * (Ta**4 - T**4) \
        + alpha*Q)
    return dTdt

# Connect to Arduino
a = tclab.TCLab()
```

```

# Turn LED on
print('LED On')
a.LED(100)

# Run time in minutes
run_time = 10.0
# Number of cycles
loops = int(60.0*run_time)
tm = np.zeros(loops)

# Temperature (K)
Tsp1 = np.ones(loops) * 23.0 # set point (degC)
T1 = np.ones(loops) * a.T1 # measured T (degC)

Tsp2 = np.ones(loops) * 23.0 # set point (degC)
T2 = np.ones(loops) * a.T2 # measured T (degC)

# Predictions
Tp = np.ones(loops) * a.T1
error_eb = np.zeros(loops)
Tpl = np.ones(loops) * a.T1
error_fopdt = np.zeros(loops)

# impulse tests (0 - 100%)
Q1 = np.ones(loops) * 0.0
Q2 = np.ones(loops) * 0.0
Q1[10:110] = 50.0 # step up for 100 sec
Q1[200:300] = 90.0 # step up for 100 sec
Q1[400:500] = 70.0 # step up for 100 sec

print('Running Main Loop. Ctrl-C to end.')
print(' Time  Q1   Q2   T1   T2')
print('{:6.1f} {:6.2f} {:6.2f} {:6.2f} {:6.2f}'.format(tm[0], \
                                                    Q1[0], \
                                                    Q2[0], \
                                                    T1[0], \
                                                    T2[0]))

```

```

# Create plot
plt.figure(figsize=(10,7))
plt.ion()
plt.show()

# Main Loop
start_time = time.time()
prev_time = start_time
try:
    for i in range(1,loops):
        # Sleep time
        sleep_max = 1.0
        sleep = sleep_max - (time.time() - prev_time)
        if sleep>=0.01:
            time.sleep(sleep-0.01)
        else:
            time.sleep(0.01)
            # Record time and change in time
            t = time.time()
            dt = t - prev_time
            prev_time = t
            tm[i] = t - start_time

# Read temperatures in Kelvin
T1[i] = a.T1
T2[i] = a.T2

# Simulate one time step with Energy Balance
Tnext = odeint(heat,Tp[i-1]+273.15,[0,dt],args=(Q1[i-1],))
Tp[i] = Tnext[1]-273.15
error_eb[i] = error_eb[i-1] + abs(Tp[i]-T1[i])

# Simulate one time step with FOPDT model
z = np.exp(-dt/tauP)
Tpl[i] = (Tpl[i-1]-Tss) * z \
        + (Q1[max(0,i-int(thetaP)-1)]-Qss)*(1-z)*Kp \
        + Tss
error_fopdt[i] = error_fopdt[i-1] + abs(Tpl[i]-T1[i])

```



```

# Write output (0-100)
a.Q1(Q1[i])
a.Q2(Q2[i])

# Print line of data
print('{:6.1f} {:6.2f} {:6.2f} {:6.2f} {:6.2f}'.format(tm[i], \
                                                    Q1[i], \
                                                    Q2[i], \
                                                    T1[i], \
                                                    T2[i]))

# Plot
plt.clf()
ax=plt.subplot(3,1,1)
ax.grid()
plt.plot(tm[0:i],T1[0:i], 'ro',label=r'$T_1$ measured')
plt.plot(tm[0:i],Tp[0:i], 'k-',label=r'$T_1$ energy balance')
plt.plot(tm[0:i],Tpl[0:i], 'g:',label=r'$T_1$ FOPDT')
plt.plot(tm[0:i],T2[0:i], 'bx',label=r'$T_2$ measured')
plt.ylabel('Temperature (degC)')
plt.legend(loc=2)
ax=plt.subplot(3,1,2)
ax.grid()
plt.plot(tm[0:i],error_eb[0:i], 'k-',label='Energy Balance')
plt.plot(tm[0:i],error_fopdt[0:i], 'g:',label='Linear')
plt.ylabel('Cumulative Error')
plt.legend(loc='best')
ax=plt.subplot(3,1,3)
ax.grid()
plt.plot(tm[0:i],Q1[0:i], 'r-',label=r'$Q_1$')
plt.plot(tm[0:i],Q2[0:i], 'b:',label=r'$Q_2$')
plt.ylabel('Heaters')
plt.xlabel('Time (sec)')
plt.legend(loc='best')
plt.draw()
plt.pause(0.05)
plt.ylabel('Heaters')
plt.xlabel('Time (sec)')
plt.legend(loc='best')
plt.draw()
plt.pause(0.05)

```

```

# Turn off heaters
a.Q1(0)
a.Q2(0)
# Save text file
a.save_txt(tm[0:i],Q1[0:i],Q2[0:i],T1[0:i],T2[0:i],Tsp1[0:i],Tsp2[0:i])

    # Save figure
    plt.savefig('test_Models.png')

# Allow user to end loop with Ctrl-C
except KeyboardInterrupt:
    # Disconnect from Arduino
    a.Q1(0)
    a.Q2(0)
    print('Shutting down')
    a.close()
    a.save_txt(tm[0:i],Q1[0:i],Q2[0:i],T1[0:i],T2[0:i],Tsp1[0:i],Tsp2[0:i])
    plt.savefig('test_Models.png')

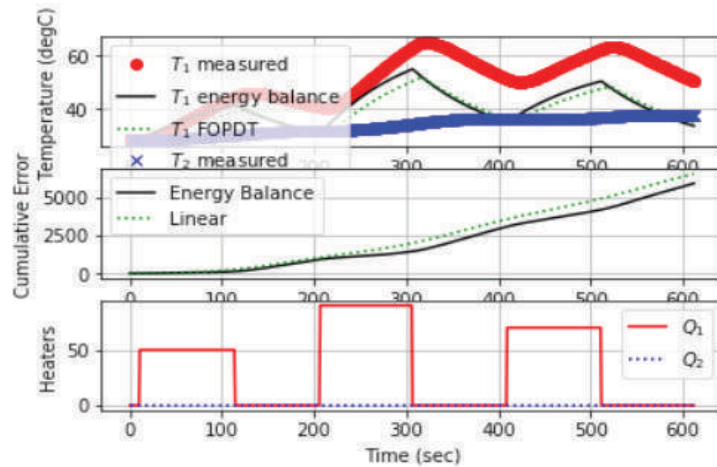
# Make sure serial connection still closes when there's an error
except:

    # Disconnect from Arduino
    a.Q1(0)
    a.Q2(0)
    print('Error: Shutting down')
    a.close()

    a.save_txt(tm[0:i],Q1[0:i],Q2[0:i],T1[0:i],T2[0:i],Tsp1[0:i],Tsp2[0:i])
    plt.savefig('test_Models.png')
    raise

```

Jika dijalankan maka hasilnya seperti terlihat pada Gambar 3.20.



Gambar 3.20. Perbandingan keluaran model keseimbangan energi, FOPDT dan hardware TCLab

Dan berikut ini skrip python untuk membandingkan keluaran model orde dua linier dengan waktu tunda (SOPDT) dengan data riil keluaran dari hardware TCLab.

```
import tclab
import numpy as np
import time
import matplotlib.pyplot as plt
from scipy.optimize import minimize
import random

# Second order model of TCLab
# initial parameter guesses
Kp = 0.2
taus = 50.0
zeta = 1.2
# magnitude of step
M = 80
# overdamped 2nd order step response
def model(y0,t,M,Kp,taus,zeta):
    # y0 = initial y
    # t = time
    # M = magnitude of the step
    # Kp = gain
```

```

# taus = second order time constant
# zeta = damping factor (zeta>1 for overdamped)
a = np.exp(-zeta*t/taus)
b = np.sqrt(zeta**2-1.0)
c = (t/taus)*b
y = Kp * M * (1.0 - a * (np.cosh(c)+(zeta/b)*np.sinh(c))) + y0
return y

# define objective for optimizer
def objective(p,tm,y meas):
    # p = optimization parameters
    Kp = p[0]
    taus = p[1]
    zeta = p[2]
    # tm = time points
    # y meas = measurements
    # y pred = predicted values
    n = np.size(tm)
    y pred = np.ones(n)*y meas[0]
    for i in range(1,n):
        y pred[i] = model(y meas[0],tm[i],M,Kp,taus,zeta)
    sse = sum((y meas-y pred)**2)
    # penalize bound violation
    if taus<10.0:
        sse = sse + 100.0 * (10.0-taus)**2
    if taus>200.0:
        sse = sse + 100.0 * (200.0-taus)**2
    if zeta<=1.1:
        sse = sse + 1e6 * (1.0-zeta)**2
    if zeta>=5.0:
        sse = sse + 1e6 * (5.0-zeta)**2
    return sse

# Connect to Arduino
a = tclab.TCLab()
# Get Version
print(a.version)
# Turn LED on
print('LED On')
a.LED(100)

```

```

# Run time in minutes
run_time = 5.0

# Number of cycles
loops = int(60.0*run_time)
tm = np.zeros(loops)
z = np.zeros(loops)

# Temperature (K)
T1 = np.ones(loops) * a.T1 # measured T (degC)
T1p = np.ones(loops) * a.T1 # predicted T (degC)

# step test (0 - 100%)
Q1 = np.ones(loops) * 0.0
Q1[1:] = M # magnitude of the step
print('Running Main Loop. Ctrl-C to end.')
print(' Time Kp  taus  zeta')
print('{:6.1f} {:6.2f} {:6.2f} {:6.2f}'.format(tm[0],Kp,taus,zeta))

# Create plot
plt.figure(figsize=(10,7))
plt.ion()
plt.show()

# Main Loop
start_time = time.time()
prev_time = start_time
try:
    for i in range(1,loops):
        # Sleep time
        sleep_max = 1.0
        sleep = sleep_max - (time.time() - prev_time)
        if sleep>=0.01:
            time.sleep(sleep)
        else:
            time.sleep(0.01)

        # Record time and change in time
        t = time.time()

```

```

dt = t - prev_time
prev_time = t
tm[i] = t - start_time

# Read temperatures in Kelvin
T1[i] = a.T1

# Estimate parameters after 15 cycles and every 3 steps
if i>=15 and (np.mod(i,3)==0):
    # randomize guess values
    r = random.random()-0.5 # random number -0.5 to 0.5
    Kp = Kp + r*0.05
    taus = taus + r*1.0
    zeta = zeta + r*0.01
    p0=[Kp,taus,zeta] # initial parameters
    solution = minimize(objective,p0,args=(tm[0:i+1],T1[0:i+1]))
    p = solutio
n.x
    Kp = p[0]
    taus = max(10.0,min(200.0,p[1])) # clip to >10, <=200
    zeta = max(1.1,min(5.0,p[2])) # clip to >=1.1, <=5

# Update 2nd order prediction
for j in range(1,i+1):
    T1p[j] = model(T1p[0],tm[j],M,Kp,taus,zeta)

# Write output (0-100)
a.Q1(Q1[i])

# Print line of data
print('{:6.1f} {:6.2f} {:6.2f} {:6.2f}'.format(tm[i],Kp,taus,zeta))

# Plot
plt.clf()
ax=plt.subplot(2,1,1)
plt.ylabel('Temperature (degC)')
plt.legend(loc=2)
ax=plt.subplot(2,1,2)
ax.grid()
plt.plot(tm[0:i],Q1[0:i],'b-',label=r'$Q_1$')
plt.ylabel('Heaters')

```

```

plt.xlabel('Time (sec)')
plt.legend(loc='best')
plt.draw()
plt.pause(0.05)

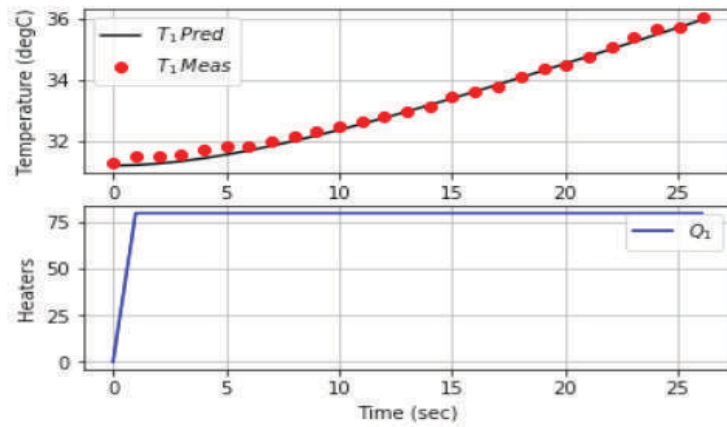
# Turn off heater
S
a.Q1(0)
a.Q2(0)
# Save text file
a.save_txt(tm[0:i],Q1[0:i],z[0:i],T1[0:i],T1e[0:i],z[0:i],z[0:i])
# Save figure
plt.savefig('test_Second_Order.png')

# Allow user to end loop with Ctrl-C
except KeyboardInterrupt:
    # Disconnect from Arduino
    a.Q1(0)
    a.Q2(0)
    print('Shutting down')
    a.close()
    a.save_txt(tm[0:i],Q1[0:i],z[0:i],T1[0:i],z[0:i],z[0:i],z[0:i])
    plt.savefig('test_Heaters.png')

# Make sure serial connection still closes when there's an error
except:
    # Disconnect from Arduino
    a.Q1(0)
    a.Q2(0)
    print('Error: Shutting down')
    a.close()
    a.save_txt(tm[0:i],Q1[0:i],z[0:i],T1[0:i],z[0:i],z[0:i],z[0:i])
    plt.savefig('test_Second_Order.png')
    raise

```

Jika dijalankan maka hasilnya seperti terlihat pada Gambar 3.21.



Gambar 3.21. Perbandingan keluaran model SOPDT dan hardware TCLab

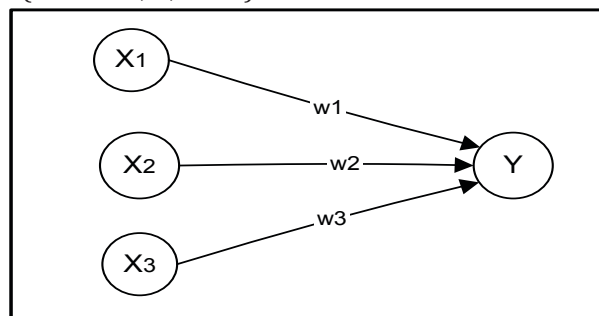


# BAB 4

## Teori Jaringan Syaraf Tiruan

### 4.1. Konsep Dasar

Jaringan Syaraf Tiruan (JST) atau *Artificial Neural Network* adalah model pemrosesan komputasi yang diinspirasi dari cara kerja sistem jaringan syaraf manusia. Pada JST juga terdapat istilah *neuron* atau sering disebut *node*. Setiap *neuron* terhubung dengan *neuron* yang lain melalui *layer* dengan bobot tertentu. Sedangkan setiap *neuron* mempunyai internal state yang disebut aktivasi. Aktivasi tersebut merupakan fungsi dari masukan yang diterima. Suatu *neuron* mengirimkan sinyal ke *neuron-neuron* yang lain (Setiawan, K, 2003).



Gambar 4.1. Ilustrasi Jaringan Saraf Tiruan

Misalnya sebuah *neuron* Y pada Gambar 4.1 menerima masukan dari *neuron* X1, X2, X3. Aktivasi atau keluaran sinyal adalah  $Y=f(y)$ . Bobot yang menghubungkan *neuron* X1, X2, X3 ke *neuron* Y adalah  $w_1, w_2, w_3$ . Maka masukan jaringan  $y_{in} = \sum X_i w_i$  pada *neuron* Y adalah jumlah dari sinyal bobot dan masing-masing *neuron* X1, X2, X3, yang kemudian di aktivasi dengan fungsi aktivasi berikut ini.

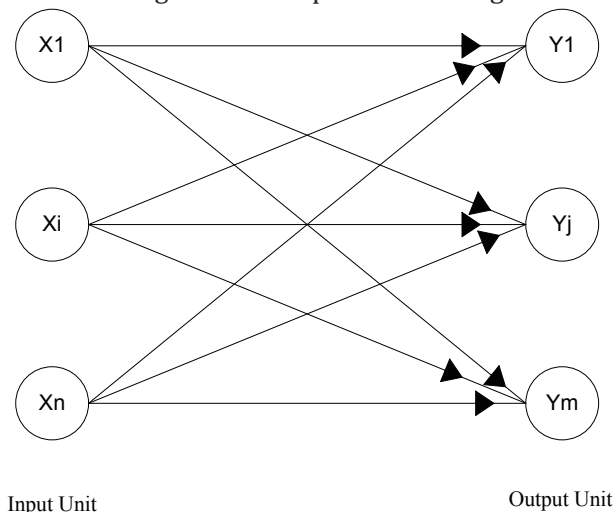
$$f(y) = \frac{1}{1 + \exp(-y_{in})} \quad (4.1)$$

## 4.2. Arsitektur Jaringan Syaraf Tiruan

Jaringan Syaraf Tiruan dapat diklasifikasikan menjadi dua jenis yaitu *single layer* dan *multilayer*.

### 1. *Single Layer*

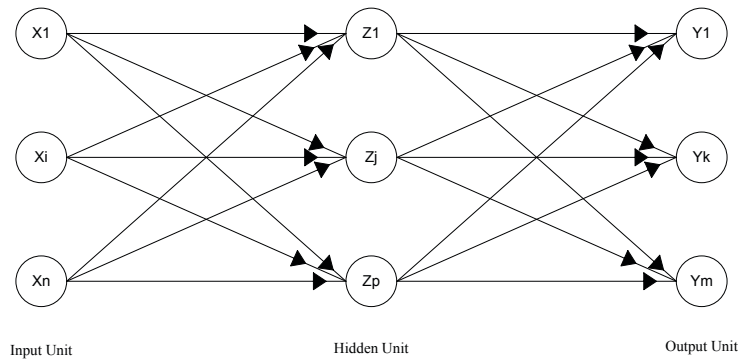
*Neuron-neuron* dikelompokkan menjadi dua bagian yaitu unit-unit masukan dan unit-unit keluaran. Unit-unit masukan menerima masukan dari luar sedangkan unit-unit keluaran menghasilkan respon sesuai dengan masukannya.



Gambar 4.2. Arsitektur Jaringan Syaraf Tiruan *Single Layer*

### 2. *Multilayer*

Pada jaringan multilayer, selain terdapat unit-unit masukan dan unit-unit keluaran, juga terdapat *hidden unit*. Jumlah *hidden unit* dan jumlah lapisannya (*layer*) berbanding lurus dengan kompleksitas jaringan. Jaringan multilayer digunakan untuk menyelesaikan persoalan yang lebih rumit.



Gambar 4.3. Arsitektur Jaringan Saraf Tiruan *Multilayer*

### 4.3. Pelatihan Jaringan Saraf Tiruan

Pelatihan JST bertujuan untuk mencari bobot-bobot yang terdapat pada setiap *layer*. Ada dua jenis pelatihan dalam sistem JST (Setiawan, K, 2003), yaitu:

#### 1. *Supervised Learning*

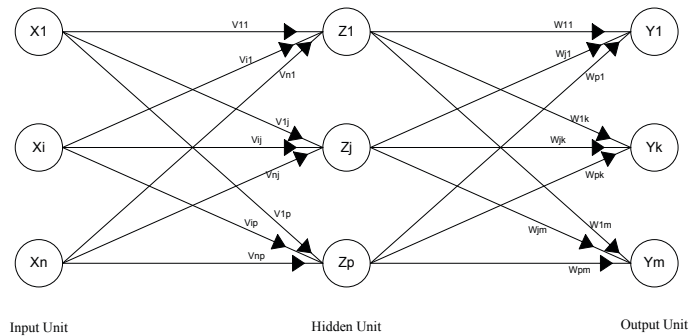
Dalam proses pelatihan ini, jaringan dilatih dengan cara diberikan data-data yang disebut data latih (*dataset*) yang terdiri atas pasangan data masukan-keluaran yang diharapkan. JST memiliki *associative memory*. Setelah jaringan dilatih, *associative memory* dapat mengingat suatu pola. Jika jaringan diberi masukan baru, jaringan dapat menghasilkan keluaran seperti yang diharapkan berdasarkan pola yang sudah ada.

#### 2. *Unsupervised Learning*

Dalam proses pelatihan ini, jaringan dilatih hanya dengan diberi data masukan yang memiliki kesamaan sifat tanpa disertai keluaran.

#### 4.3.1. Backpropagation

*Backpropagation* merupakan salah satu metode pelatihan dari JST. *Backpropagation* menggunakan arsitektur *multilayer* dengan metode pelatihan *supervised* (Setiawan, K, 2003).



Gambar 4.4. Arsitektur JST dengan Metode Backpropagation

Keterangan:

$X_i$  Unit masukan ke-i

$Z_j$  *Hidden unit* ke-j

$Y_k$  Unit keluaran ke-k

$V_{ij}$  Bobot antara unit masukan ke-i dengan *hidden unit* ke-j

$W_{jk}$  Bobot antara *hidden unit* ke-j dengan unit keluaran ke-k

### Tahapan Proses Metode *Backpropagation*

Pada dasarnya, pembelajaran atau pelatihan dengan metode *backpropagation* terdiri atas tiga langkah utama, yaitu:

1. Data dimasukkan ke unit masukan (*feedforward*).
2. Perhitungan dan propagasi balik dari *error*.
3. Pembaharuan (*adjustment*) bobot.

Untuk selengkapnya, langkah-langkah pelatihan adalah sebagai berikut :

Langkah 1            Inisialisasi nilai bobot  
                           Nilai bobot dapat diset dengan sembarang angka (acak) antara -0.5 dan 0.5

Umpan maju (*feedforward*)

Langkah 2            Setiap unit masukan ( $X_i, i = 1, \dots, n$ ) menerima sinyal masukan dan menyebarkannya pada seluruh *hidden unit* melalui  $V_{ij}$ .

Langkah 3            Setiap *hidden unit* ( $Z_j, j = 1, \dots, p$ ) menghitung sinyal-sinyal masukan yang sudah berbobot dan menggunakan fungsi aktivasi yang telah ditentukan untuk menentukan sinyal keluaran dari *hidden unit* tersebut.

Langkah 4            Setiap unit keluaran ( $Y_k$ ,  $k = 1, \dots, m$ ) menghitung sinyal-sinyal masukan yang sudah berbobot dan menggunakan fungsi aktivasi yang telah ditentukan untuk menentukan sinyal keluaran dari unit keluaran tersebut lalu mengirim sinyal keluaran ini ke seluruh unit pada unit keluaran.

Propagasi error (*backpropagation of error*)

Langkah 5            Setiap unit keluaran ( $Y_k$ ,  $k = 1, \dots, m$ ) memiliki target keluaran yang sesuai dengan masukan pelatihan. Hitung kesalahan antara target keluaran dengan keluaran yang dihasilkan jaringan sehingga menghasilkan faktor koreksi error ( $\delta_k$ ). Faktor koreksi error digunakan untuk menghitung koreksi error ( $\Delta W_{jk}$ ) untuk memperbaharui  $W_{jk}$  dan dikirimkan ke *hidden unit*.

Langkah 6            Setiap *hidden unit* ( $Z_j$ ,  $j = 1, \dots, p$ ) menghitung bobot yang dikirimkan ke unit keluaran dan menghasilkan faktor koreksi error ( $\delta_j$ ). Faktor koreksi error digunakan untuk menghitung koreksi error ( $\Delta V_{ij}$ ) untuk memperbaharui  $V_{ij}$  dan dikirimkan ke unit masukan.

Pembaharuan bobot (*adjustment*)

Langkah 7            Setiap unit keluaran ( $Y_k$ ,  $k = 1, \dots, m$ ) memperbaharui bobotnya dari setiap *hidden unit*

$$W_{jk}(\text{baru}) = W_{jk}(\text{lama}) + \Delta W_{jk} \quad (4.2)$$

Demikian pula setiap *hidden unit* ( $Z_j$ ,  $j = 1, \dots, p$ ) memperbaharui bobotnya dari setiap unit masukan

$$V_{ij}(\text{baru}) = V_{ij}(\text{lama}) + \Delta V_{ij} \quad (4.3)$$

Langkah 8            Memeriksa *stop condition*

Untuk menentukan *stopping condition* dengan cara membatasi *error*. Digunakan metode *Mean Square Error* (MSE) dan *Root Mean Square Error* (RMSE) untuk menghitung rata-rata *error* antara keluaran yang dikehendaki (target) dengan keluaran yang dihasilkan (prediksi) oleh jaringan.

Setelah pelatihan selesai, jaringan akan dapat mengenali pola-pola data yang dilatihkan. Cara mendapatkan keluaran adalah dengan cara melakukan proses umpan maju (langkah 2 sampai langkah 4).

### 4.3.2. Dynamic Tunneling Technique

Kelemahan pembelajaran JST dengan menggunakan *backpropagation* biasanya sering terjebak di *error* lokal minimal. Untuk mengatasi hal ini salah satunya dengan

cara menambahkan algoritma *Dynamic Tunneling Technique* (DTT). DTT adalah penerapan dari metode pencarian langsung dan itu dapat diperlakukan sebagai metode pencarian pola Hooke-Jeeves (RoyChowdhury, P., Singh, Y. P. dan Chansarkar, R. A., 1999). Titik kesetimbangan  $u_{eq}$  dari sistem dinamis:

$$\frac{du}{dt} = g(u) \quad (4.4)$$

diistilahkan dengan atraktor (*repeller*) jika tidak (paling tidak satu) nilai eigen dari matriks A

$$A = \frac{\partial g(u_{eq})}{\partial u} \quad (4.5)$$

memiliki bagian riil positif. Secara khas, sistem dinamis seperti pada Persamaan (4.5) memenuhi kondisi Lipschitz

$$\left| \frac{\partial g(u_{eq})}{\partial u} \right| < \infty \quad (4.6)$$

yang menggaransi keberadaan dari solusi unik untuk tiap kondisi awal  $u_0$ . Biasanya beberapa sistem memiliki waktu relaksasi tak hingga untuk sebuah atraktor dan waktu lepas dari *repeller*. Berdasarkan violasi dari kondisi Lipschitz pada titik kesetimbangan, yang mana menginduksi solusi tunggal sedemikian hingga tiap-tiap solusi mendekati sebuah atraktor, atau melepaskan dari *repeller* dalam waktu berhingga.

Untuk menunjukkan pernyataan di atas, pikirkan sebuah sistem yang didefinisikan dengan:

$$\frac{du}{dt} = -u^{1/3} \quad (4.7)$$

Sistem yang dinyatakan oleh Persamaan (4.7) memiliki titik kesetimbangan pada  $u = 0$ , yang mengganggu kondisi Lipschitz pada  $u = 0$ , selama:

$$\left| \frac{d}{du} \left( \frac{du}{dt} \right) \right| = \left| -\frac{1}{3} u^{-2/3} \right| \rightarrow \infty, \quad \text{seperti } u \rightarrow 0 \quad (4.8)$$

Titik kesetimbangan dari sistem yang telah disebutkan di atas diistilahkan dengan titik kesetimbangan pengatraktifan (*attracting equilibrium point*), selama dari nilai kondisi awal sembarang  $u_0 \neq 0$ , sistem dinamis pada Persamaan (4.7) mencapai titik kesetimbangan  $u = 0$  pada waktu berhingga  $t_1$  yang diberikan dengan:

$$t_1 = - \int u^{-1/3} du = \frac{3}{2} u_0^{2/3} \quad (4.9)$$

Dengan cara yang sama, sistem dinamis:

$$\frac{du}{dt} = u^{-1/3} \quad (4.10)$$

memiliki titik kesetimbangan tidak stabil perepelan (*repelling unstable equilibrium point*) pada  $u = 0$  yang mengganggu kondisi Lipschitz. Kondisi awal sembarang yang secara infinitesimal mendekati titik perepelan (*repelling point*)  $u = 0$  akan melepas *repeller*, untuk mencapai titik  $u_0$  pada waktu berhingga yang diberikan dengan:

$$t_1 = \int u^{-1/3} du = \frac{3}{2} u_0^{2/3} \quad (4.11)$$

Konsep dari algoritma DTT didasarkan pada gangguan pada kondisi Lipschitz pada titik kesetimbangan, yang mana diatur oleh kenyataan bahwa partikel sembarang yang diletakkan pada gangguan kecil dari titik kesetimbangan akan berpindah dari titik dimana ia tinggal ke tempat lain didalam ruang waktu yang telah dibahas sebelumnya. Terobosan (*tunneling*) diterapkan dengan cara menyelesaikan persamaan diferensial berikut:

$$\frac{dw_{j,k}^l}{dt} = \rho (w_{j,k}^l - w_{j,k}^{l*})^{1/3} \quad (4.12)$$

Disini  $\rho$  menyatakan kekuatan pembelajaran (*strength of learning*),  $w_{j,k}^{l*}$  menyatakan minimum lokal terakhir untuk  $w_{j,k}^l$ . Jelaslah dari Persamaan (4.12) bahwa titik minimum lokal  $W^*$  juga merupakan titik kesetimbangan dari sistem terobosan (*tunneling system*). Nilai dari  $w_{j,k}^l$  adalah  $w_{j,k}^{l*} + \varepsilon_{j,k}^l$ , dimana  $|\varepsilon_{j,k}^l| < 1$  dan Persamaan (4.12) diintegrasikan untuk waktu tertentu ( $t$ ), dengan  $\Delta t$  (*time-step*) yang kecil. Setelah semua  $\Delta t$ ,  $mse_w$  dihitung dengan nilai baru dari  $w_{j,k}^l$  menjaga komponen sisa  $W$  sama seperti  $w_{j,k}^{l*}$ . Terobosan digunakan untuk menghentikan ketika  $mse_w \leq mse_{llm}$  dimana  $llm$  menunjukkan minimum lokal terakhir (kondisi penurunan), dan memulai penurunan gradien berikutnya. Jika kondisi penurunan ini tidak memuaskan, maka kemudian proses diulang dengan semua komponen  $w_{j,k}^{l*}$  sampai kondisi penurunan di atas tercapai. Jika tidak ada nilai  $w_{j,k}^l$  dari kondisi di atas yang memuaskan, maka minimum lokal yang terakhir menjadi titik minimum global. Dengan cara pengulangan penurunan gradien dan terobosan dalam ruang bobot ini, maka dapat diperoleh titik minimum global. Dalam pekerjaan ini, untuk menghindari kondisi saturasi dari neuron, maka digunakan koreksi yang memenuhi hubungan rumus berikut:

$$f'(\alpha) = f'(\alpha) + 0.1 \quad (4.13)$$

Akhirnya, ungkapan umum untuk sistem dinamis yang dibahas secara terpisah dalam dua tahap (EBP dan DTT) dapat dituliskan sebagai berikut:

$$\begin{aligned} \frac{dw_{j,k}^l}{dt} = & -\frac{\eta \partial E(W)}{\partial w_{j,k}^l} \Theta[1 - \Theta[mse(W) - mse(W^*)]] \\ & + \rho (w_{j,k}^l - w_{j,k}^{ls})^{1/3} \Theta[mse(W) - mse(W^*)] \end{aligned} \quad (4.14)$$

dimana  $\Theta$  adalah fungsi *step* yang didefinisikan dengan

$$\Theta[x] = \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases}$$



# BAB 5

## Pemrograman Jaringan Syaraf Tiruan

---

### 5.1. Pemrograman Jaringan Syaraf Tiruan dengan Python

Belajar Jaringan Syaraf Tiruan (JST) untuk menyelesaikan permasalahan di dunia nyata dengan Bahasa Pemrograman Python akan lebih mudah dipahami jika dimulai dari penyelesaian kasus sederhana. Sebelum JST digunakan untuk menyelesaikan permasalahan yang kompleks di sistem kendali cerdas. Contoh kasus sederhana yang ingin diselesaikan dalam buku ini, yaitu JST akan diperankan sebagai mesin gerbang XOR. Pada saat proses pelatihan, JST akan mempelajari bagaimana pola masukan keluaran berdasarkan *dataset* Gerbang Logika XOR. Selanjutnya, pada saat pengujian, JST diberikan masukan sebarang, dia akan memprediksi keluarannya.

Terdapat dua cara pemrograman JST menggunakan Bahasa pemrograman Python yang dilakukan di buku ini. Pertama memanfaatkan skrip python standar, yang kedua dengan memanfaatkan *library Keras*.

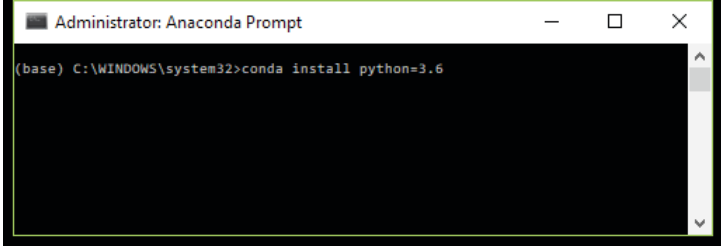
*Keras* adalah *library* Python yang sangat bagus dan mudah digunakan untuk mengembangkan dan mengevaluasi model JST dan *Deep Learning*. *Keras* adalah *Application Program Interface* (API) JST level tinggi, ditulis dengan bahasa pemrograman Python dan mampu berjalan di atas TensorFlow, CNTK, atau Theano. *Keras* memungkinkan pekerjaan eksperimen menjadi lebih cepat, dibandingkan

tanpa menggunakan *Keras*. *Keras* mampu mengubah ide menjadi kenyataan dengan sesedikit mungkin penundaan, ini kunci pengerjaan penelitian yang baik. *Keras* memungkinkan pembuatan prototipe yang mudah dan cepat (melalui keramahan pengguna, modularitas, dan ekstensibilitas). *Keras* juga mendukung *convolutional networks* dan *recurrent networks*, serta kombinasi keduanya.

*Library Keras* membutuhkan *library TensorFlow* sebagai *backend*. Oleh karena itu sebelum instal *Keras* terlebih dulu diinstal *library* ini. Sayangnya, *Tensorflow* sampai buku ini ditulis, belum *support* untuk python 3.7. Jika kita instal python satu paket dengan *anaconda* terbaru, defaultnya python 3.7, maka perlu dilakukan *downgrade* ke python 3.6. Caranya melalui *Anaconda Prompt*, ketikkan perintah berikut:

```
>> conda install python=3.6
```

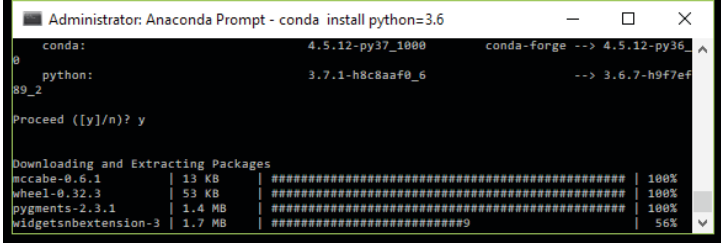
Seperti yang ditunjukkan pada Gambar 5.1.



```
Administrator: Anaconda Prompt
(base) C:\WINDOWS\system32>conda install python=3.6
```

Gambar 5.1. Downgrade ke python 3.6

Setelah menekan tombol *enter* dan menekan tombol 'y', silahkan tunggu sampai proses instalasi selesai, seperti ditunjukkan pada Gambar 5.2.



```
Administrator: Anaconda Prompt - conda install python=3.6
conda: 4.5.12-py37_1000 conda-forge --> 4.5.12-py36_
python: 3.7.1-h8c8aaf0_6 --> 3.6.7-h9f7ef
Proceed ([y]/n)? y

Downloading and Extracting Packages
mccabe-0.6.1 | 13 KB | 100%
wheel-0.32.3 | 53 KB | 100%
pygments-2.3.1 | 1.4 MB | 100%
widgetsnbextension-3 | 1.7 MB | 56%
```

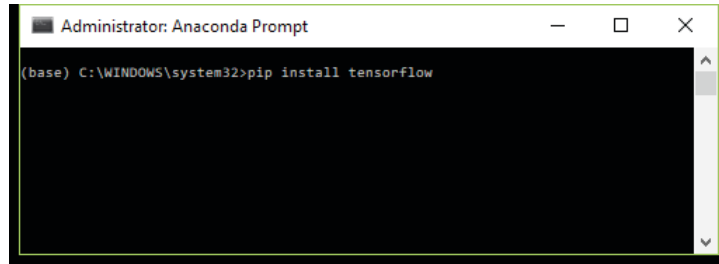
Gambar 5.2. Proses instalasi python 3.6

Setelah itu, lakukan instal *Tensorflow*. Cara instalasinya, silahkan jalankan *Anaconda Prompt*, dan ketikkan salah satu perintah berikut:

```
>> pip install tensorflow
```

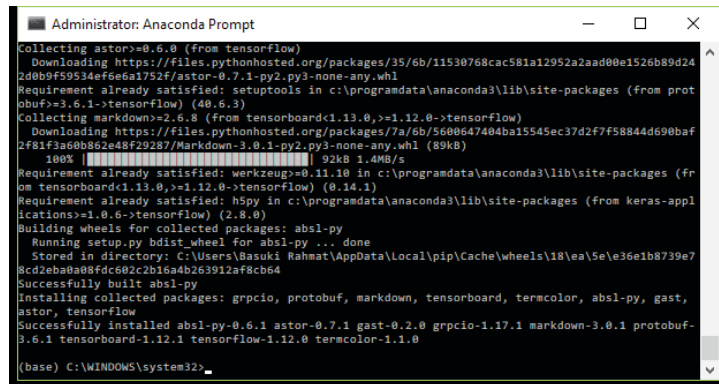
```
>> conda install -c conda-forge tensorflow
```

Seperti yang ditunjukkan pada Gambar 5.3.



Gambar 5.3. Instalasi library TensorFlow

Setelah menekan tombol *enter*, silahkan tunggu sampai proses instalasi selesai, seperti ditunjukkan pada Gambar 5.4.



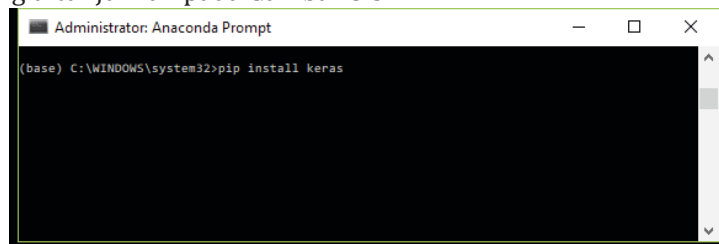
Gambar 5.4. Proses instalasi TensorFlow berhasil

Selanjutnya, untuk melakukan instalasi library *Keras* silahkan jalankan *Anaconda Prompt*, dan ketikkan salah satu dari perintah berikut:

```
>> pip install keras
```

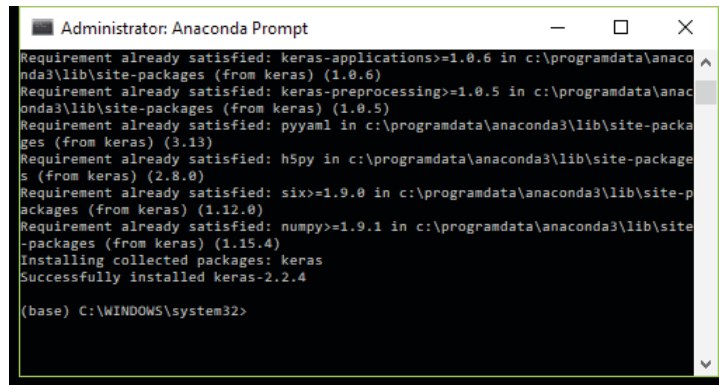
```
>> conda install -c conda-forge keras
```

Seperti yang ditunjukkan pada Gambar 5.5.



Gambar 5.5. Instalasi library Keras

Setelah menekan tombol *enter*, silahkan tunggu sampai proses instalasi selesai, seperti ditunjukkan pada Gambar 5.6.



```
Administrator: Anaconda Prompt
Requirement already satisfied: keras-applications>=1.0.6 in c:\programdata\anaco
nda3\lib\site-packages (from keras) (1.0.6)
Requirement already satisfied: keras-preprocessing>=1.0.5 in c:\programdata\anac
onda3\lib\site-packages (from keras) (1.0.5)
Requirement already satisfied: pyyaml in c:\programdata\anaconda3\lib\site-packa
ges (from keras) (3.13)
Requirement already satisfied: h5py in c:\programdata\anaconda3\lib\site-package
s (from keras) (2.8.0)
Requirement already satisfied: six>=1.9.0 in c:\programdata\anaconda3\lib\site-p
ackages (from keras) (1.12.0)
Requirement already satisfied: numpy>=1.9.1 in c:\programdata\anaconda3\lib\site
-packages (from keras) (1.15.4)
Installing collected packages: keras
Successfully installed keras-2.2.4

(base) C:\WINDOWS\system32>
```

Gambar 5.6. Proses instalasi Keras berhasil

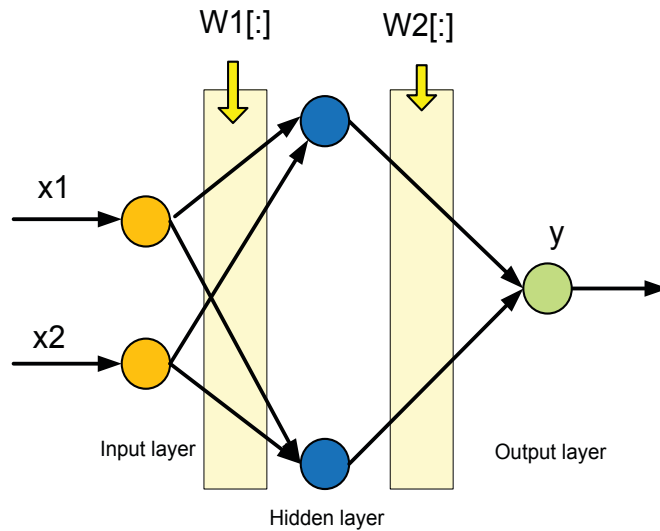
## 5.2. Prediksi XOR dengan JST

Sudah disebutkan sebelumnya, akan dijelaskan contoh pemrograman JST dengan bahasa pemrograman python. Dimana studi kasusnya, JST akan menirukan cara kerja mesin Gerbang Logika XOR. Pada saat proses pelatihan, JST akan mempelajari bagaimana pola masukan keluaran berdasarkan *dataset* Gerbang Logika XOR. Selanjutnya, pada saat pengujian, JST diberikan masukan sebarang, dia akan memprediksi keluarannya. *Dataset* pelatihan Gerbang Logika XOR, seperti terlihat pada Tabel 5.1.

Tabel 5.1. Dataset Gerbang Logika XOR

x1	x2	y
0	1	1
1	0	1
1	1	0
0	0	0

Selanjutnya, untuk menyelesaikan permasalahan ini, dirancang arsitektur JST dengan dua masukan dan satu keluaran sesuai dengan *dataset* yang digunakan. Rancangan arsitektur JST diperlihatkan pada Gambar 5.7.



Gambar 5.7. Arsitektur JST

Pemrograman python untuk menyelesaikan permasalahan Gerbang Logika XOR ini, seperti dijabarkan pada langkah-langkah skrip berikut.

```
import numpy as np # For matrix math
import matplotlib.pyplot as plt # For plotting
import sys # For printing

# The training data:

X = np.array([
    [0, 1],
    [1, 0],
    [1, 1],
    [0, 0]
])
# The labels for the training data.
y = np.array([
    [1],
    [1],
    [0],
    [0]
])
outNN = np.zeros(4) # For NN output arrays
```

Skrip penulisan arsitektur JST sebagai berikut:

```
num_i_units = 2 # Number of Input units
num_h_units = 2 # Number of Hidden units
num_o_units = 1 # Number of Output units
```

Skrip penulisan parameter JST sebagai berikut:

```
# The learning rate for Gradient Descent.
learning_rate = 0.01

# The parameter to help with overfitting.
reg_param = 0

# Maximum iterations for Gradient Descent.
max_iter = 50

# Number of training examples
m = 4
```

Skrip penulisan bobot-bobot awal (W1, W2) dan bias (B1, B2) sebagai berikut:

```
np.random.seed(1)
W1 = np.random.normal(0, 1, (num_h_units, num_i_units)) # 2x2
W2 = np.random.normal(0, 1, (num_o_units, num_h_units)) # 1x2

B1 = np.random.random((num_h_units, 1)) # 2x1
B2 = np.random.random((num_o_units, 1)) # 1x1
```

Fungsi aktivasi yang digunakan, fungsi aktivasi sigmoid, skrip fungsinya sebagai berikut:

```
def sigmoid(z, derv=False):
    if derv: return z * (1 - z)
    return 1 / (1 + np.exp(-z))
```

Fungsi tahap maju (*forward pass*):

```
def forward(x, predict=False):
    a1 = x.reshape(x.shape[0], 1) # Getting the training example as a column vector.

    z2 = W1.dot(a1) + B1 # 2x2 * 2x1 + 2x1 = 2x1
    a2 = sigmoid(z2) # 2x1

    z3 = W2.dot(a2) + B2 # 1x2 * 2x1 + 1x1 = 1x1
    a3 = sigmoid(z3)

    if predict: return a3
    return (a1, a2, a3)
```

Selanjutnya untuk menampung *update* penurunan gradien bobot dan bias dinyatakan dalam  $dW1$ ,  $dW2$ ,  $dB1$ ,  $dB2$ . Variabel-variabel ini akan berisi gradien bobot dan bias yang akan digunakan untuk memperbarui bobot dan bias. Juga, disiapkan vektor untuk menyimpan nilai-nilai biaya (*cost*) untuk setiap iterasi berupa penurunan gradien *error* dari target untuk membantu memvisualisasikan biaya saat bobot dan bias diperbarui.

```
dW1 = 0 # Gradient for W1
dW2 = 0 # Gradient for W2

dB1 = 0 # Gradient for B1
dB2 = 0 # Gradient for B2

cost = np.zeros((max_iter, 1))
```

Selanjutnya untuk proses pelatihan, digunakan fungsi *train*, seperti pada skrip berikut.

```
def train(_W1, _W2, _B1, _B2): # The arguments are to bypass UnboundLocalError
    error
    for i in range(max_iter):
        c = 0
        dW1 = 0
        dW2 = 0
        dB1 = 0
        dB2 = 0

        for j in range(m):
            sys.stdout.write("\rIteration: {} and {}".format(i + 1, j + 1))

            # Forward Prop.
            a0 = X[j].reshape(X[j].shape[0], 1) # 2x1

            z1 = _W1.dot(a0) + _B1 # 2x2 * 2x1 + 2x1 = 2x1
            a1 = sigmoid(z1) # 2x1
            z2 = _W2.dot(a1) + _B2 # 1x2 * 2x1 + 1x1 = 1x1
            a2 = sigmoid(z2) # 1x1

            # Back prop.
            dz2 = a2 - y[j] # 1x1
            dW2 += dz2 * a1.T # 1x1 .* 1x2 = 1x2

            dz1 = np.multiply((_W2.T * dz2), sigmoid(a1, derv=True)) # (2x1 * 1x1)
            . * 2x1 = 2x1
```

```

dW1 += dz1.dot(a0.T) # 2x1 * 1x2 = 2x2

dB1 += dz1 # 2x1
dB2 += dz2 # 1x1

c = c + (-(y[j] * np.log(a2)) - ((1 - y[j]) * np.log(1 - a2)))
sys.stdout.flush() # Updating the text.

_W1 = _W1 - learning_rate * (dW1 / m) + ( (reg_param / m) * _W1)
_W2 = _W2 - learning_rate * (dW2 / m) + ( (reg_param / m) * _W2)

_B1 = _B1 - learning_rate * (dB1 / m)
_B2 = _B2 - learning_rate * (dB2 / m)
cost[i] = (c / m) + (
    (reg_param / (2 * m)) *
    (
        np.sum(np.power(_W1, 2)) +
        np.sum(np.power(_W2, 2))
    )
)
return (_W1, _W2, _B1, _B2)

```

Untuk menjalankan proses pelatihan dengan fungsi *train* diatas, caranya ketikkan skrip berikut.

```
W1, W2, B1, B2 = train(W1, W2, B1, B2)
```

Jika semua skrip di atas dijalankan, contoh hasilnya seperti terlihat pada Gambar 5.8.

```

In [17]: W1, W2, B1, B2 = train(W1, W2, B1, B2)

Iteration: 500 and 4

In [18]: W1
Out[18]: array([[ 1.57970137, -0.56772981],
                [-0.5654201 , -1.10156296]])

In [19]: W2
Out[19]: array([[ 0.86820662, -2.19599663]])

In [20]: B1
Out[20]: array([[0.429051 ],
                [0.52062069]])

In [21]: B2
Out[21]: array([[0.32673699]])

```

Gambar 5.8. Keluaran bobot dan bias hasil pelatihan

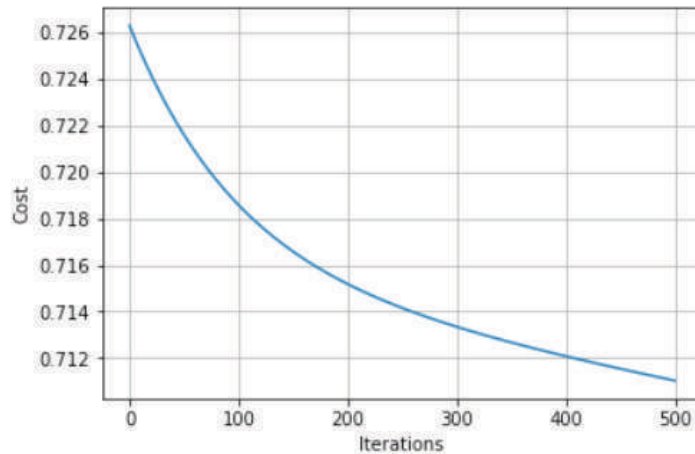


Selanjutnya untuk pengeplotan grafik *cost* berupa penurunan gradien *error* tiap iterasi digunakan skrip berikut ini.

```
# Assigning the axes to the different elements.
plt.plot(range(max_iter), cost)
# Labelling the x axis as the iterations axis.
plt.xlabel("Iterations")
# Labelling the y axis as the cost axis.
plt.ylabel("Cost")

# Showing the plot.
plt.grid(True)
plt.show()
```

Jika dijalankan, hasilnya berupa grafik *cost* yaitu penurunan gradien error tiap iterasi seperti terlihat pada Gambar 5.9.



Gambar 5.9. Grafik penurunan gradien error tiap iterasi hasil proses pelatihan

Selanjutnya dilakukan pengujian JST dengan diberikan nilai masukan sebarang, apakah bisa memprediksi dengan tepat sesuai keluaran yang diinginkan Gerbang Logika XOR. Misalkan diberikan data masukan sebarang, dengan nama variabel *coba*, seperti skrip berikut.

```
coba = np.array([
    [0, 1],
    [1, 0],
    [1, 1],
    [0, 0]
])
```

Pengujian JST tahap maju (umpan maju), untuk mendapatkan prediksi JST yang dibandingkan dengan target yaitu keluaran Gerbang Logika XOR yang diinginkan, caranya silahkan ketikkan skrip berikut ini.

```
# Forward Prop.
for j in range(4):
    a0 = coba[j].reshape(coba[j].shape[0], 1) # 2x1

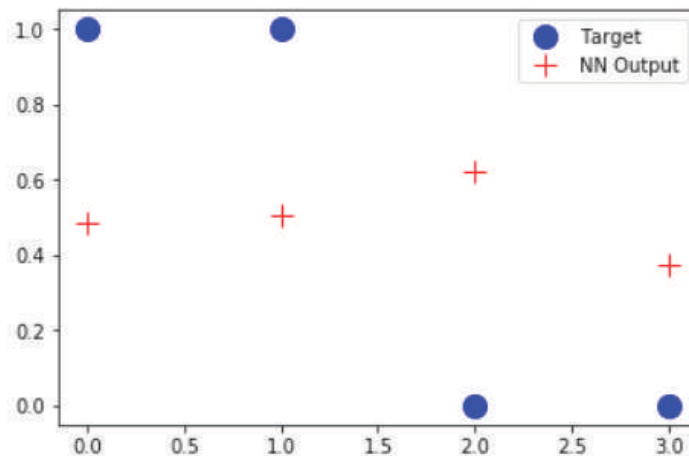
    z1 = W1.dot(a0) + B1 # 2x2 * 2x1 + 2x1 = 2x1
    a1 = sigmoid(z1) # 2x1

    z2 = W2.dot(a1) + B2 # 1x2 * 2x1 + 1x1 = 1x1
    outNN[j] = sigmoid(z2) # 1x1
```

Selanjutnya untuk pengeplotan keluaran hasil prediksi JST dibandingkan dengan target berupa keluaran yang diinginkan dari gerbang Logika XOR, silahkan ketikkan skrip berikut ini.

```
plt.plot(y, 'bo', label='Target', linewidth=2, markersize=12)
plt.plot(outNN, 'r+', label='NN Output', linewidth=2, markersize=12)
plt.legend(loc='upper right')
plt.show()
```

Jika dijalankan, hasilnya seperti terlihat pada Gambar 5.10.



Gambar 5.10. Perbandingan prediksi JST dan target

Selanjutnya untuk menghitung kesalahan prediksi, digunakan *Mean Square Error* (MSE) dan *Root Mean Square Error* (RMSE). Berikut ini skrip pythonnya (memanfaatkan *library* scikit-learn).

```
from sklearn.metrics import mean_squared_error
from math import sqrt
mse1 = mean_squared_error(y, outNN)
rmse1 = sqrt(mean_squared_error(y, outNN))
```

Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 5.11.

```
In [30]: mse1
```

```
Out[30]: 0.25878169771977794
```

```
In [31]: rmse1
```

```
Out[31]: 0.5087059049389715
```

Gambar 5.11. Kesalahan prediksi JST

### 5.3. Prediksi XOR dengan JST-Keras

Seperti telah disebutkan sebelumnya, terdapat dua cara pemrograman JST menggunakan Bahasa pemrograman Python yang dilakukan di buku ini. Pertama memanfaatkan skrip python standar seperti telah dijelaskan pada Sub Bab 5.2, yang kedua dengan cara memanfaatkan *library Keras*. Setelah semua library yang dibutuhkan sudah terinstal seperti Tensorflow dan *Keras*, maka python sudah siap untuk menyelesaikan kasus Gerbang Logika XOR. Selanjutnya hasilnya bisa dibandingkan dengan cara skrip sebelumnya.

Silahkan ketikkan skrip python berikut, sesuai arsitektur JST pada Gambar 5.7 (2 masukan dan 1 keluaran).

```
# Import `Sequential` from `keras.models`
from keras.models import Sequential

# Import `Dense` from `keras.layers`
from keras.layers import Dense

# Initialize the constructor
model = Sequential()
```

```
# Add an input layer
model.add(Dense(2, activation='sigmoid', input_shape=(2,)))

# Add one hidden layer
model.add(Dense(2, activation='sigmoid'))

# Add an output layer
model.add(Dense(1, activation='sigmoid'))
```

Ketikkan skrip berikut ini, untuk model JST-nya, dan dapatkan bobot-bobot dan bias awal.

```
# Model output shape
model.output_shape

# Model summary
model.summary()

# Model config
model.get_config()

# List all weight tensors
model.get_weights()
```

Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 5.12.

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 2)	6
dense_2 (Dense)	(None, 2)	6
dense_3 (Dense)	(None, 1)	3

=====  
 Total params: 15  
 Trainable params: 15  
 Non-trainable params: 0  
 =====

```
Out[31]: [array([[ -0.6369038 ,  0.7741929 ],
           [ 0.89861023,  0.49025428]], dtype=float32),
          array([0., 0.], dtype=float32),
          array([[ -1.125      , -0.36791164],
           [ 0.8911623 , -0.09824407]], dtype=float32),
          array([0., 0.], dtype=float32),
          array([[0.6184999],
           [1.4075023]], dtype=float32),
          array([0.], dtype=float32)]
```

Gambar 5.12. Model, bobot dan bias awal

Untuk pelatihan JST silahkan ketikkan skrip berikut.

```
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

model.fit(X_train, y_train, epochs=20, batch_size=1, verbose=1)
```

Jika dijalankan, hasilnya seperti terlihat pada Gambar 5.13.

```
Epoch 13/20
4/4 [=====] - 0s 4ms/step - loss: 0.7819 - acc: 0.5000
Epoch 14/20
4/4 [=====] - 0s 2ms/step - loss: 0.7808 - acc: 0.5000
Epoch 15/20
4/4 [=====] - 0s 3ms/step - loss: 0.7801 - acc: 0.5000
Epoch 16/20
4/4 [=====] - 0s 4ms/step - loss: 0.7798 - acc: 0.5000
Epoch 17/20
4/4 [=====] - 0s 3ms/step - loss: 0.7787 - acc: 0.5000
Epoch 18/20
4/4 [=====] - 0s 5ms/step - loss: 0.7769 - acc: 0.5000
Epoch 19/20
4/4 [=====] - 0s 5ms/step - loss: 0.7762 - acc: 0.5000
Epoch 20/20
4/4 [=====] - 0s 3ms/step - loss: 0.7754 - acc: 0.5000

Out[32]: <keras.callbacks.History at 0x23f3d4a99e8>
```

Gambar 5.13. Proses pelatihan JST menggunakan Keras

Untuk menguji coba JST hasil *Keras* ini, silahkan diberikan masukan sebarang, misalkan sama seperti pada cara skrip sebelumnya. Silahkan ketikkan skrip berikut.

```
y_pred = model.predict(coba)
```

Jika dijalankan, hasilnya seperti terlihat pada Gambar 5.14.

```
In [34]: y_pred

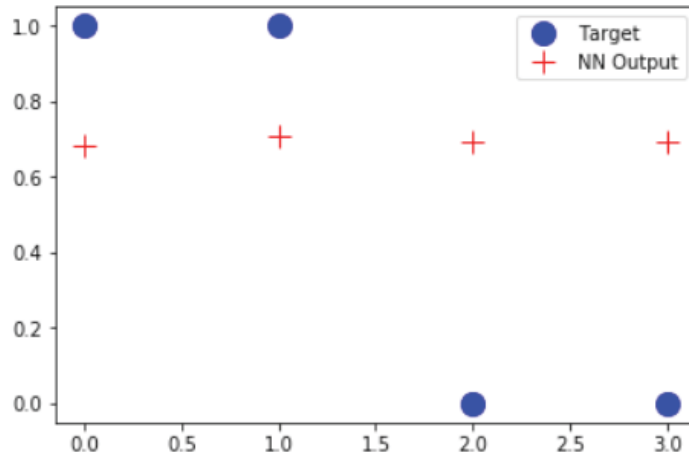
Out[34]: array([[0.6830085 ],
                [0.707375  ],
                [0.6946924 ],
                [0.69468486]], dtype=float32)
```

Gambar 5.14. Hasil ujicoba prediksi JST Keras

Selanjutnya untuk pengeplotan keluaran hasil prediksi JST *Keras* dibandingkan dengan target berupa keluaran yang diinginkan dari gerbang Logika XOR, silahkan ketikkan skrip berikut ini.

```
plt.plot(y, 'bo', label='Target', linewidth=2, markersize=12)
plt.plot(y_pred, 'r+', label='NN Output', linewidth=2, markersize=12)
plt.legend(loc='upper right')
plt.show()
```

Jika dijalankan, hasilnya seperti terlihat pada Gambar 5.15.



Gambar 5.15. Perbandingan prediksi JST Keras dan target

Selanjutnya untuk menghitung kesalahan prediksi JST Keras, digunakan *Mean Square Error* (MSE) dan *Root Mean Square Error* (RMSE). Berikut ini skrip pythonnya.

```
from sklearn.metrics import mean_squared_error
from math import sqrt
mse2 = mean_squared_error(y, y_pred)
rmse2 = sqrt(mean_squared_error(y, y_pred))
```

Jika dijalankan, maka hasilnya seperti terlihat pada Gambar 5.16.

```
In [37]: mse2
Out[37]: 0.28782439128215564

In [38]: rmse2
Out[38]: 0.5364926758886421
```

Gambar 5.16. Kesalahan prediksi JST Keras

Jika dibandingkan, hasil kedua cara pemrograman JST diatas, antara menggunakan skrip standar (JST) dan menggunakan *library Keras* (JST Keras), dapat ditabelkan seperti pada Tabel 5.2.

Tabel 5.2. Perbandingan prediksi JST

Ukuran Kesalahan Prediksi	JST	JST Keras
mse	0.2587817	0.2878244
rmse	0.5087060	0.5364927

Dari hasil perbandingan pada Tabel 5.2, terlihat hasil menggunakan skrip sendiri lebih baik dibandingkan menggunakan library *Keras*, ditunjukkan dengan mse dan rmse yang lebih kecil. Namun demikian, penggunaan *Keras*, tetap sangat membantu, terutama jika dikembangkan untuk arsitektur JST yang lebih kompleks.





# BAB 6

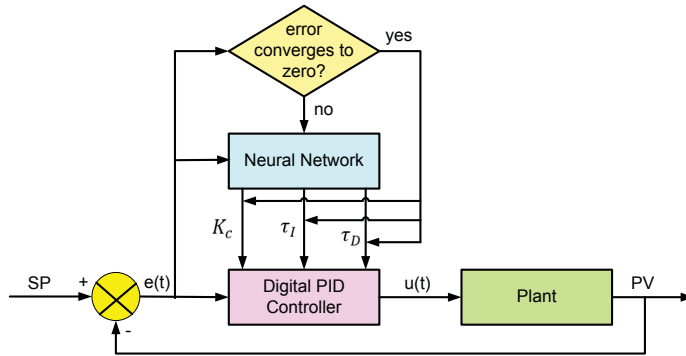
## Sistem Kendali Cerdas Berbasis PID-JST

---

### 6.1. Sistem Kendali PID-JST

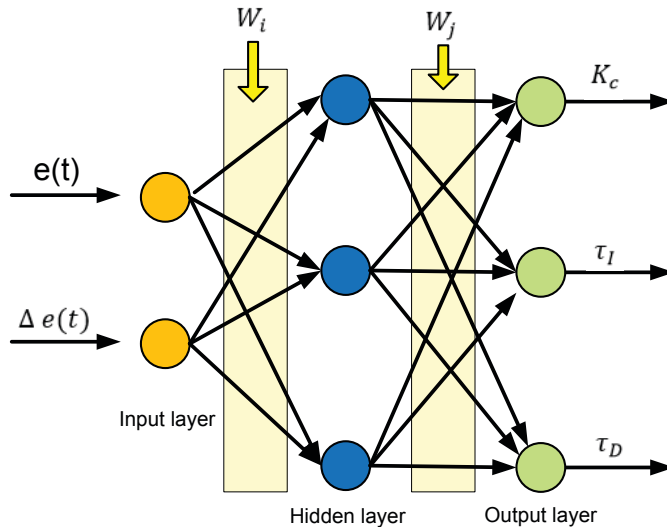
Sistem kendali Proporsional Integral dan Derivatif (PID), sudah dijelaskan di Bab 3. Termasuk peluang penggunaan sistem cerdas fuzzy untuk proses penalaan nilai  $K_c$ ,  $\tau_i$  dan  $\tau_d$  pada pengendali PID. Pada Bab ini akan dijelaskan sistem yang sama, namun menggunakan algoritma penalaan yang berbeda. Sistem fuzzy digantikan dengan Jaringan Syaraf Tiruan (JST).

Seperti telah dijelaskan di bab 3, proses penalaan nilai  $K_c$ ,  $\tau_i$  dan  $\tau_d$  pada pengendali PID memberikan peluang pengaturan secara terprogram menggunakan algoritma sistem cerdas. Kali ini algoritma sistem cerdas yang digunakan yaitu algoritma JST. Dimana salah satu cara penalaan nilai  $K_c$ ,  $\tau_i$  dan  $\tau_d$  yang masuk akal seperti diperlihatkan pada Gambar 6.1. Ide penggabungan PID dengan JST yang lain, dapat dibaca pada literatur-literatur berikut: (Chen & Lin, 2013), (JIA, BAI, SHAN, CUI, & XU, 2014), (Ohnishi & Yamamoto, 2004), (Sharma, Kumar, Gaur, & Mittal, 2016), dan (Zeng, Xie, Chen, & Weng, 2018).



Gambar 6.1. Proses penalaan nilai  $K_c$ ,  $\tau_I$  dan  $\tau_D$  pada pengendali PID menggunakan JST

Proses penalaan nilai  $K_c$ ,  $\tau_I$  dan  $\tau_D$  pada pengendali PID menggunakan JST seperti pada Gambar 6.1, membutuhkan mekanisme bagaimana melakukan penyesuaian nilai  $K_c$ ,  $\tau_I$  dan  $\tau_D$ , berdasarkan pembacaan *error*  $e(t)$  dan *delta\_error*  $\Delta e(t)$ . Pembacaan *error* dan *delta\_error* digunakan untuk memutuskan apakah perlu dilakukan pengubahan nilai  $K_c$ ,  $\tau_I$  dan  $\tau_D$  atau tidak. Jika *error* sudah konvergen ke arah nol, maka nilai  $K_c$ ,  $\tau_I$  dan  $\tau_D$  yang sudah ada dipertahankan. Namun jika masih jauh dari konvergen ke arah nol, maka perlu dijalankan pengubahan nilai  $K_c$ ,  $\tau_I$  dan  $\tau_D$  dengan cara *update* bobot JST. Proses diulang-ulang sampai sistem pengendalian selesai. Arsitektur JST untuk keperluan ini diperlihatkan pada Gambar 6.2.



Gambar 6.2. Arsitektur JST untuk penyesuaian nilai  $K_c$ ,  $\tau_I$  dan  $\tau_D$

## 6.2. Pemrograman Sistem Kendali PID-JST

Seperti telah disebutkan, tujuan pemanfaatan JST adalah bagaimana melakukan proses penyesuaian atau penalaan nilai  $K_c$ ,  $\tau_i$  dan  $\tau_d$ , agar diperoleh hasil pengendalian yang terbaik. Hasil pengendalian yang baik ditunjukkan dengan penurunan error yang signifikan. Contoh implementasi pemrograman penalaan nilai  $K_c$ ,  $\tau_i$  dan  $\tau_d$  berbasis JST menggunakan bahasa pemrograman Python diperlihatkan pada skrip program berikut ini.

```
import numpy as np # For matrix math
import matplotlib.pyplot as plt # For plotting
import sys # For printing

num_i_units = 2 # Number of Input units
num_h_units = 3 # Number of Hidden units
num_o_units = 3 # Number of Output units

# The learning rate for Gradient Descent.
learning_rate = 0.01
# The parameter to help with overfitting.
reg_param = 0
# Maximum iterations for Gradient Descent.
max_iter = 100
# Number of training examples
m = 4

#Generating the Weights and Biases
np.random.seed(1)
W1 = np.random.normal(0, 1, (num_h_units, num_i_units)) # 3x2
W2 = np.random.normal(0, 1, (num_o_units, num_h_units)) # 3x3

B1 = np.random.random((num_h_units, 1)) # 3x1
B2 = np.random.random((num_o_units, 1)) # 3x1

def train(_W1, _W2, _B1, _B2): # Neural Network Training
    for i in range(max_iter):
        c = 0

        dW1 = 0
        dW2 = 0
```

```

dB1 = 0
dB2 = 0

for j in range(m):
    sys.stdout.write("\rIteration: {} and {}".format(i + 1, j + 1))

    # Forward Prop.
    a0 = X[j].reshape(X[j].shape[0], 1) # 2x1

    z1 = _W1.dot(a0) + _B1 # 3x2 * 2x1 + 3x1 = 3x1
    a1 = sigmoid(z1) # 3x1

    z2 = _W2.dot(a1) + _B2 # 3x2 * 2x1 + 3x1 = 3x1
    a2 = sigmoid(z2) # 3x1

    # Back prop.
    dz2 = a2 - y[j].reshape(y[j].shape[0], 1) # 3x1
    dW2 += dz2 * a1.T # 3x1 .* 1x3 = 3x3

    dz1 = np.multiply((_W2.T).dot(dz2), sigmoid(a1, deriv=True)) # (3x3 *
3x1) .* 3x1 = 3x1

    dB1 += dz1 # 3x1
    dB2 += dz2 # 3x1

    c = c + (-(y[j].reshape(y[j].shape[0], 1) * np.log(a2)) - ((1 - y[j].reshape(y[j].
shape[0], 1)) * np.log(1 - a2)))
    dW1 += dz1.dot(a0.T) # 3x1 * 1x2 = 3x2
    sys.stdout.flush() # Updating the text.

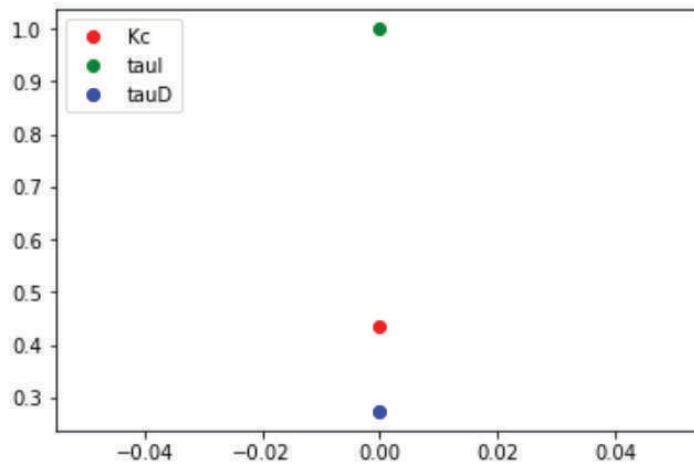
    _W1 = _W1 - learning_rate * (dW1 / m) + ( (reg_param / m) * _W1)
    _W2 = _W2 - learning_rate * (dW2 / m) + ( (reg_param / m) * _W2)
    _B1 = _B1 - learning_rate * (dB1 / m)
    _B2 = _B2 - learning_rate * (dB2 / m)

return (_W1, _W2, _B1, _B2)

```

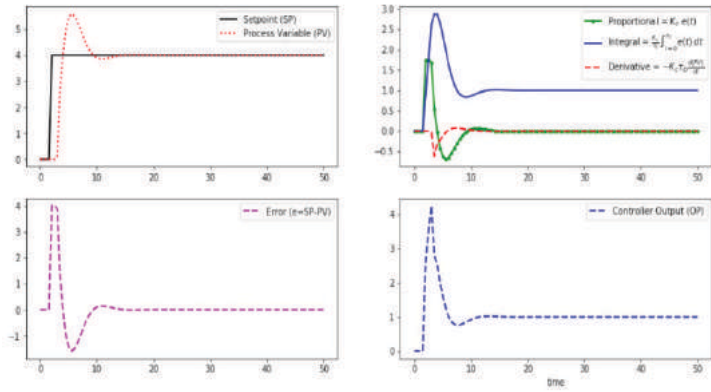
```
# Testing
for j in range(1):
    a0 = coba[j].reshape(coba[j].shape[0], 1) # 2x1
    z1 = W1.dot(a0) + B1 # 3x2 * 2x1 + 3x1 = 3x1
    a1 = sigmoid(z1) # 3x1
    z2 = W2.dot(a1) + B2 # 3x2 * 2x1 + 3x1 = 3x1
    outNN = sigmoid(z2) # 3x1
```

Dari contoh proses penalaan nilai  $K_c$ ,  $\tau_i$  dan  $\tau_d$  pada pengendali PID berbasis JST ini, diperoleh hasil  $K_c = 0.43504464$ ,  $\tau_i = 0.99994792$ , dan  $\tau_d = 0.27455905$ , seperti terlihat pada Gambar 6.3.



Gambar 6.3. Hasil proses penalaan nilai  $K_c$ ,  $\tau_i$  dan  $\tau_d$  berbasis JST

Nilai-nilai  $K_c$ ,  $\tau_i$  dan  $\tau_d$  hasil proses penalaan berbasis JST ini, selanjutnya dilihat pengaruhnya terhadap keberhasilan pengendalian. Hasil proses pengendalian menggunakan pengendali PID melalui proses penalaan nilai  $K_c$ ,  $\tau_i$  dan  $\tau_d$  berbasis JST diperlihatkan pada Gambar 6.4. Terlihat sistem kendali PID-JST bekerja dengan baik, ditunjukkan dengan error menuju nol.



Gambar 6.4. Hasil proses penalaan nilai  $K_c$ ,  $\tau_i$  dan  $\tau_d$  pada pengendali PID berbasis JST terhadap keluaran sistem

# BAB 7

## Penutup

---

### 7.1. Kesimpulan

Dari hasil pengujian yang telah dilakukan, dapat ditunjukkan bahwa bahasa pemrograman Python dapat digunakan sebagai alternatif untuk mewujudkan simulasi sistem kendali cerdas berbasis Fuzzy dan Jaringan Syaraf Tiruan (JST). Pada sebagian besar sistem kendali di industri saat ini dimana Pengendali Proportional Integral Derivatif (PID) memegang peranan sangat penting, maka fuzzy dan JST dapat digunakan sebagai alternatif untuk proses penalaan nilai  $K_c$ ,  $\tau_i$  dan  $\tau_d$ . Hasil simulasi menunjukkan efek perubahan nilai  $K_c$ ,  $\tau_i$  dan  $\tau_d$  oleh sistem fuzzy dan JST pada pengendali PID mempengaruhi keberhasilan sistem kendali.

---

### 7.2. Ucapan Terimakasih

Terimakasih kepada Kementerian Riset, Teknologi dan Pendidikan Tinggi Universitas Pembangunan Nasional “Veteran” Jawa Timur yang telah memberikan dana penelitian sesuai dengan Surat Perjanjian Penugasan dalam rangka pelaksanaan Program Penelitian Mandiri Skim Peningkatan Mutu Pembelajaran (PMP) UPN “Veteran” Jawa Timur Tahun Anggaran 2018, Nomor: SPP/3/UN63.8/LT/IV/ 2018.





## Daftar Pustaka

- BYU, B. Y. U. (2018a). Proportional Integral Derivative (PID) Control. *Apmonitor.com*. Retrieved from [http:// apmonitor.com/pdc/index.php/Main/ProportionalIntegralDerivative](http://apmonitor.com/pdc/index.php/Main/ProportionalIntegralDerivative)
- BYU, B.Y.U.(2018b). Temperature Control Lab. *Apmonitor.com*. Retrieved from [http:// apmonitor.com/pdc/ index.php/Main/ArduinoTemperatureControl](http://apmonitor.com/pdc/index.php/Main/ArduinoTemperatureControl)
- Chen, S.-Y., & Lin, F.-J. (2013). Decentralized PID neural network control for five degree-of-freedom active magneticbearing. *Engineering Applications of Artificial Intelligence*, 26(3), 962–973. [http:// doi.org/https://doi.org/10.1016/j.engappai.2012.11.002](http://doi.org/https://doi.org/10.1016/j.engappai.2012.11.002)
- Dettori, S., Iannino, V., Colla, V., & Signorini, A. (2018). An adaptive Fuzzy logic-based approach to PID control of steam turbines in solar applications. *Applied Energy*, 227, 655–664. [http://doi.org/https://doi.org/ 10.1016/j.apenergy.2017.08.145](http://doi.org/https://doi.org/10.1016/j.apenergy.2017.08.145)
- Fan, Z., Yu, X., Yan, M., & Hong, C. (2018). Oxygen Excess Ratio Control of PEM Fuel Cell Based on Self-adaptive Fuzzy PID. This work is supported by the National Nature Science Foundation of China No. 61520106008, the National Key Research and Development Program of China 2017YFB0102800 and the industrial innovation special funds of Jilin Province 2018C035-2. *IFAC-PapersOnLine*, 51(31), 15–20. [http://doi.org/ https://doi.org/10.1016/j.ifacol.2018.10.004](http://doi.org/https://doi.org/10.1016/j.ifacol.2018.10.004)

- Jang, J.-S.R, Sun, C.-T & Mizutani, E. (1996). *Neuro-Fuzzy and Soft Computing*. London: Prentice Hall Inc.
- JIA, C., BAI, T., SHAN, X., CUI, F., & XU, S. (2014). Cloud Neural Fuzzy PID Hybrid Integrated Algorithm of Flatness Control. *Journal of Iron and Steel Research, International*, 21(6), 559–564. <http://doi.org/> [https://doi.org/10.1016/S1006-706X\(14\)60087-X](https://doi.org/10.1016/S1006-706X(14)60087-X)
- Kosko, B. (1992). *Neural Networks and Fuzzy Systems*. London: Prentice Hall Inc.
- Nayak, J. R., Shaw, B., & Sahu, B. K. (2018). Application of adaptive-SOS (ASOS) algorithm based interval type-2 fuzzy-PID controller with derivative filter for automatic generation control of an interconnected power system. *Engineering Science and Technology, an International Journal*, 21(3), 465–485. <http://doi.org/> <https://doi.org/10.1016/j.jestch.2018.03.010>
- Ohnishi, Y., & Yamamoto, T. (2004). A design of nonlinear PID control systems with parameter compensated neural network. *IFAC Proceedings Volumes*, 37(12), 505–509. <http://doi.org/> [https://doi.org/10.1016/S1474-6670\(17\)31519-7](https://doi.org/10.1016/S1474-6670(17)31519-7)
- RoyChowdhury, P., Singh, Y. P., dan Chansarkar, R. A. (1999). Dynamic Tunneling Technique for Efficient Training of Multilayer Perceptrons. *IEEE Transactions on Neural Networks*. Vol.10, No.1. January.
- Setiawan, K. (2003). *Paradigma Sistem Cerdas*, Bayumedia Publishing. Malang.
- Sharma, R., Kumar, V., Gaur, P., & Mittal, A. P. (2016). An adaptive PID like controller using mix locally recurrent neural network for robotic manipulator with variable payload. *ISA Transactions*, 62, 258–267. <http://doi.org/> <https://doi.org/10.1016/j.isatra.2016.01.016>
- Verma, O. P., Manik, G., & Jain, V. K. (2018). Simulation and control of a complex nonlinear dynamic behavior of multi-stage evaporator using PID and Fuzzy-PID controllers. *Journal of Computational Science*, 25, 238–251. <http://doi.org/> <https://doi.org/10.1016/j.jocs.2017.04.001>
- Wang, L. (1997). *A Course in Fuzzy Systems and Control*. London: Prentice-Hall.
- Zeng, G.-Q., Xie, X.-Q., Chen, M.-R., & Weng, J. (2018). Adaptive population extremal optimization-based PID neural network for multivariable nonlinear control systems. *Swarm and Evolutionary Computation*. <http://doi.org/> <https://doi.org/10.1016/j.swevo.2018.04.008>