

BAB II DASAR TEORI

2.1. Tinjauan Pustaka

Pada bab ini dijelaskan tinjauan pustaka dan dasar teori yang digunakan sebagai acuan dalam merancang sistem pengendalian laju aliran air berbasis OpenPLC pada ESP32 menggunakan metode PID.

Tabel 2.1 Tinjauan Pustaka

No	Judul Penelitian	Keterangan	Perbedaan
1.	Pengaturan Kecepatan Pompa Air pada Media Tanam Hidroponik Berbasis IoT (Mahdafiqia et al., 2026)	Penelitian ini membahas pengaturan kecepatan pompa air menggunakan esp32 dan sensor <i>flow</i> untuk mengukur laju aliran air. Sistem menggunakan Ziegler–Nichols. Pengujian dilakukan untuk mengetahui kemampuan sistem dalam mempertahankan laju aliran sesuai nilai yang ditentukan.	Penelitian ini menggunakan OpenPLC berbasis ESP32 sebagai pengendali utama untuk mengatur laju aliran air menggunakan PID dan sensor <i>flow</i> sebagai umpan balik.
2.	Implementasi Sistem Kendali PID untuk Pengaturan	Penelitian ini membahas sistem pengendalian debit aliran air	Penelitian ini menggunakan OpenPLC berbasis ESP32 sebagai

	Debit Aliran Air pada <i>Mini Plant</i> Berbasis PLC dan HMI (sutiadi et al., 2026)	menggunakan PLC dan HMI. Sensor aliran digunakan sebagai masukan <i>feedback</i> untuk mengetahui kondisi debit aktual, sedangkan pompa digunakan sebagai aktuator. Pengendalian dilakukan menggunakan metode PID dan parameter PID diperoleh menggunakan metode Ziegler Nichols.	alternatif PLC untuk menjalankan kontrol PID laju aliran air.
3.	Design and Implementation of PID Based Flow Rate Control Using PLC (Shaikh et al., 2023)	Penelitian ini membahas perancangan sistem pengendalian laju aliran air menggunakan PLC dan kontrol PID. Beberapa jenis pengendalian dibandingkan, yaitu P, PI, PD, dan PID.	Penelitian ini menerapkan blok CTRL_PID pada OpenPLC <i>Runtime</i> berbasis ESP32, sedangkan penelitian sebelumnya menggunakan PLC dan membandingkan beberapa metode kontrol.

		Parameter PID diperoleh menggunakan Ziegler–Nichols dan sistem diuji berdasarkan kemampuan mempertahankan laju aliran terhadap nilai setpoint.	
4.	Sistem Pengendalian Aliran Air Laut pada Alat Pembuat Garam Menggunakan Kontrol PID (Dharma Putri & Fitri, 2024)	Penelitian ini menggunakan ESP32 sebagai pengendali sistem dan sensor <i>flow</i> untuk mengukur laju aliran air, data dari sensor digunakan sebagai feedback dalam pengendalian PID untuk mempertahankan aliran air pada kondisi yang diinginkan. Penelitian ini juga melakukan pengujian terhadap kemampuan sensor dalam membaca laju aliran	Penelitian ini menggunakan OpenPLC Runtime pada ESP32 sebagai pengendali PID, dengan sensor <i>flow</i> sebagai umpan balik dan PWM sebagai keluaran kendali pompa.

5.	OpenPLC: An IEC 61131-3 Compliant Open Source Industrial Controller for Cyber Security Research (Alves & Moris, 2018)	Penelitian ini membahas OpenPLC sebagai <i>platform PLC open-source</i> yang dikembangkan berdasarkan standar IEC 61131-3. OpenPLC dapat digunakan untuk menjalankan program kendali PLC dan mendukung implementasi sistem kendali pada perangkat dengan sumber daya yang relatif rendah.	Penelitian ini menerapkan OpenPLC pada ESP32 secara langsung untuk pengendalian laju aliran air berbasis PID, bukan hanya membahas OpenPLC sebagai platform PLC <i>open-source</i> .
----	---	---	--

Berdasarkan beberapa penelitian terdahulu yang telah dikaji, pengendalian laju aliran air umumnya dilakukan menggunakan mikrokontroler maupun PLC dengan sensor *flow* sebagai umpan balik dan PID sebagai metode pengendalian. Beberapa penelitian juga telah menggunakan metode Ziegler–Nichols dalam menentukan parameter awal PID. Selain itu, OpenPLC telah dikembangkan sebagai platform PLC *opensource* yang mendukung pemrograman sesuai standar IEC 61131-3. Namun, implementasi OpenPLC *Runtime* pada ESP32 sebagai platform pengendali utama yang menjalankan fungsi kontrol PID pada sistem laju aliran air masih dapat dikaji lebih lanjut.

Oleh karena itu, penelitian ini difokuskan pada implementasi dan evaluasi OpenPLC berbasis ESP32 sebagai platform pengendali pada sistem laju aliran air. Sistem laju aliran air digunakan sebagai *plant* pengujian, sedangkan metode PID

digunakan sebagai algoritma kontrol yang dijalankan pada OpenPLC. Pengujian dilakukan melalui kondisi *open loop* untuk mengetahui karakteristik sistem dan memperoleh parameter awal PID menggunakan metode Ziegler–Nichols, kemudian dilanjutkan dengan pengujian *closed-loop* untuk mengevaluasi kinerja sistem berdasarkan *rise time*, *overshoot*, *settling time*, dan *steady state error*.

2.2. Sistem Kontrol

Sistem kontrol merupakan suatu sistem yang digunakan untuk mengatur, mengendalikan, dan menjaga suatu variabel agar tetap berada pada nilai yang diinginkan (*setpoint*). Sistem kontrol terdiri dari beberapa komponen utama yaitu input, proses, output, dan *feedback* yang bekerja dalam pengarahannya suatu kecerdasan mesin dengan tujuan untuk melakukan pengaturan atau pengendalian suatu proses untuk mendapatkan suatu besaran yang diinginkan [10]. Secara umum, sistem kontrol dibagi menjadi dua jenis, yaitu sistem kontrol Loop Terbuka dan sistem kontrol Loop Tertutup.

2.2.1. Sistem Kontrol Loop Terbuka (*Open-Loop*)

Sistem kontrol Loop Terbuka merupakan sistem yang tidak mempunyai umpan balik, dimana output yang dihasilkan tidak mempengaruhi inputan dari sistem tersebut. Kelebihan dari sistem ini adalah memiliki struktur yang lebih sederhana dengan biaya implementasi yang rendah [11]. Meskipun memiliki kelebihan struktur yang lebih sederhana, sistem ini juga memiliki kekurangan yaitu tidak mampu mengoreksi error dan tidak akurat.

2.2.2. Sistem Kontrol Loop Tertutup (*Closed-Loop*)

Sistem kontrol Loop Tertutup merupakan sistem yang mempunyai umpan balik atau *feedback*, dimana *feedback* tersebut nantinya digunakan untuk membandingkan output yang dihasilkan dengan *setpoint* [12]. Apabila terjadi *error*, maka sistem akan secara otomatis melakukan koreksi. Sistem kontrol loop tertutup atau *closed-loop* adalah sistem kontrol yang sinyal keluarannya mempunyai pengaruh langsung pada pengontrolan. Sinyal kesalahan penggerak, yang merupakan selisih antara sinyal masukan dan sinyal umpan balik [13].

Diumpangkan ke kontroler untuk memperkecil kesalahan dan membuat agar keluaran sistem mendekati harga yang diinginkan. Dengan kata lain, istilah “loop

tertutup” berarti menggunakan aksi umpan balik untuk memperkecil kesalahan sistem.

Pada penelitian ini, sistem yang digunakan adalah sistem dengan kontrol Loop Tertutup, karena diperlukan pengaturan laju aliran air yang stabil.

2.3. Laju Aliran Fluida (*Flow Rate*)

Laju Aliran Fluida merupakan jumlah volume fluida yang melewati suatu penampang dalam interval waktu tertentu. Pada sistem pengendalian aliran, laju aliran digunakan sebagai salah satu variabel proses yang dapat diukur dan dikendalikan. Secara umum, laju aliran volumetrik dapat dinyatakan dengan persamaan:

$$Q = \frac{V}{t} \quad (2.1)$$

dengan:

- Q : Laju aliran volumetrik
- V : Volume Fluida
- t : Waktu

Persamaan tersebut menunjukkan bahwa laju aliran merupakan banyaknya volume fluida yang melewati suatu titik dalam interval waktu tertentu. Pada penelitian ini, laju aliran dinyatakan dalam mL/min karena variabel tersebut digunakan sebagai process variable pada sistem pengendalian PID.”

2.4. Sensor *Water Flow*

Sensor *water flow* merupakan sensor yang digunakan untuk mengukur laju aliran air yang melewati suatu pipa atau saluran tertentu. Sensor *water flow* digunakan untuk mengetahui estimasi jumlah debit air, laju aliran air dan jumlah air yang mengalir dalam satuan mililiter atau liter. Sensor ini umumnya bekerja berdasarkan prinsip efek *hall*, dimana putaran turbin di dalam sensor menghasilkan *pulse signal* yang sebanding dengan kecepatan aliran air.



Gambar 2.1 Sensor *Water Flow* YF-S201

Efect hall sensors dapat menghasilkan sebuah tegangan dengan resistensi medan magnet yang diterima oleh sistem tersebut dan partikel yang melewatinya [14]. Sensor YF-S201 menghasilkan pulsa dengan frekuensi yang berubah mengikuti laju aliran air. Hubungan nominal antara frekuensi dan laju aliran dapat dinyatakan sebagai:

$$f = Kq$$

Dengan f sebagai frekuensi pulsa dalam Hz, Q sebagai laju aliran dalam L/min, dan k sebagai konstanta sensor. Untuk YF-S201, nilai nominal yang digunakan pada penelitian ini adalah $k = 7,5$, sehingga:

$$Q_{L/min} = \frac{f}{7,5} \quad (2.2)$$

Apabila laju aliran dinyatakan dalam mL/min:

$$Q_{mL/min} = \frac{f}{7,5} \times 1000 \quad (2.3)$$

Apabila pulsa dihitung sebanyak N dalam interval Δt sekon, persamaan dapat dituliskan:

$$Q_{mL/min} = \frac{N}{7,5\Delta t} \times 1000 \quad (2.4)$$

Hubungan tersebut digunakan sebagai dasar perhitungan pada ESP32 pendukung. Jumlah pulsa YF-S201 dihitung selama interval sekitar satu detik, kemudian dikonversi menjadi laju aliran dalam mL/min sebelum dikirim menuju ESP32 OpenPLC. Pada interval penghitungan 1 s, perubahan satu pulsa ekuivalen dengan perubahan sekitar 133,33 mL/min.

Sensor ini digunakan sebagai feedback dalam sistem kontrol untuk mengetahui kondisi aktual laju aliran air. Secara umum, sensor menghasilkan pulse signal dengan frekuensi yang sebanding dengan laju aliran. Semakin besar debit air, semakin tinggi frekuensi pulsa yang dihasilkan [15]. Aliran air memutar rotor yang memiliki magnet permanen; saat rotor berputar, magnet melewati Hall sensor sehingga menghasilkan pulsa listrik yang kemudian dihitung oleh mikrokontroler.

Tabel 2.2 Spesifikasi Sensor Water Flow

Parameter	Spesifikasi
Tegangan Kerja	5 V DC
Arus Kerja	15mA pada 5v
Output	Digital Pulse
Range Aliran	1 – 30 L/menit
Akurasi	10%

2.5. ESP32

ESP32 merupakan mikrokontroler yang memiliki kemampuan tinggi dengan berbagai fitur seperti WiFi, Bluetooth, *Dual-core processor*, GPIO yang banyak, dan harga yang relatif terjangkau [18]. pada sistem otomasi berbasis perangkat lunak, ESP32 juga dapat digunakan sebagai perangkat keras untuk menjalankan *OpenPLC Runtime*. Dengan implementasi tersebut, program pengendalian yang dibuat menggunakan *OpenPLC Editor* dapat dijalankan pada ESP32 sehingga mikrokontroler dapat melakukan fungsi pengendalian berdasarkan logika program PLC yang telah dirancang.



Gambar 2.2 Mikrokontroler ESP32

Dalam penelitian ini digunakan dua buah mikrokontroler ESP32 dengan fungsi yang berbeda. ESP32 OpenPLC menjalankan OpenPLC *Runtime*, program FBD, dan *Function Block* CTRL_PID. ESP32 pendukung menangani akuisisi sensor serta penerapan sinyal kendali pada perangkat keras. Pertukaran nilai antarkedua ESP32 dilakukan melalui komunikasi *pin-to-pin* berbasis PWM–RC–ADC sesuai rancangan sistem.

Tabel 2.3 Spesifikasi ESP32

Parameter	Spesifikasi
<i>Frekuensi Clock</i>	Hingga 240 MHz
Tegangan operasi	3.0 – 3.6V
Tegangan input	5V
Memori RAM	520 KB SRAM
Jumlah GPIO	34 Pin
ADC (<i>Analog Input</i>)	12-bit (18 <i>Channel</i>)
<i>PWM</i>	16 <i>Channel</i>
I2C	2

2.6. Kontrol PID (*Proportional Integral Derivative*)

Kontrol *Proportional-Integral-Derivative* (PID) merupakan salah satu metode kontrol umpan balik yang banyak digunakan karena memiliki struktur yang relatif sederhana serta dapat digunakan untuk memperoleh respons sistem yang sesuai melalui pengaturan parameter kontrolnya [19]. Kontrol PID bekerja berdasarkan nilai kesalahan (*error*) antara nilai *setpoint* dan nilai aktual variabel proses. Aksi pengendalian diperoleh melalui kombinasi komponen *proportional*, *integral*, dan *derivative*.

Pada penelitian ini, metode PID digunakan sebagai algoritma pengendalian yang dijalankan pada OpenPLC berbasis ESP32. PID digunakan untuk mengolah selisih antara nilai *setpoint* laju aliran air dan nilai aktual yang diperoleh dari sensor, kemudian menghasilkan keluaran kendali yang digunakan sebagai dasar pengaturan PWM pompa. Dengan demikian, PID digunakan sebagai metode kontrol untuk mengevaluasi implementasi fungsi pengendalian pada OpenPLC.

Dalam penelitian ini, evaluasi pengendali PID difokuskan pada kemampuan process variable dalam mengikuti nilai *setpoint* yang diberikan. Sementara itu, kesesuaian process variable terhadap nilai laju aliran fisik dievaluasi melalui pengujian sensor dan pengukuran volumetrik. Pemisahan kedua pengujian tersebut dilakukan agar karakteristik pengendali tidak tercampur dengan karakteristik akurasi sistem pengukuran.

2.6.1. Proportional (P)

Kontrol *proportional* merupakan bagian dari pengendali PID yang menghasilkan aksi kendali sebanding dengan nilai *error* pada saat itu. Besarnya aksi *proportional* ditentukan oleh *proportional* gain K_p . Secara matematis, aksi *proportional* dapat dinyatakan sebagai:

$$u_p(t) = K_p e(t) \quad (2.5)$$

Nilai K_p yang lebih besar dapat meningkatkan respons pengendali terhadap error, tetapi nilai yang terlalu besar dapat menyebabkan respons menjadi agresif dan meningkatkan kecenderungan *overshoot*. Sebaliknya, nilai K_p yang terlalu kecil dapat menghasilkan respons yang lebih lambat. Aksi *proportional* dapat mempercepat respons sistem, tetapi tidak selalu mampu menghilangkan *steady-state error* secara mandiri.

2.6.2. Integral (I)

Kontrol integral bekerja dengan mengakumulasi *error* terhadap waktu. Selama masih terdapat *error*, aksi integral akan terus memberikan koreksi untuk mendorong *process variable* mendekati *setpoint*. Pada bentuk PID ideal, aksi integral dapat dinyatakan sebagai:

$$u_I(t) = \frac{K_p}{T_i} \int_0^t e(\tau) d\tau \quad (2.6)$$

dengan T_i sebagai *integral time*. Aksi *integral* berperan dalam mengurangi *steady-state error*. Namun, aksi integral yang terlalu kuat dapat meningkatkan *overshoot* dan menyebabkan respons sistem menjadi lebih berosilasi.

2.6.3. Derivative (D)

Kontrol *derivative* bekerja berdasarkan laju perubahan *error* terhadap waktu. Aksi ini memberikan respons terhadap seberapa cepat *error* berubah

sehingga dapat membantu meredam perubahan respons yang terlalu cepat. Pada bentuk PID ideal, aksi *derivative* dapat dinyatakan sebagai:

$$u_D(t) = K_p T_d \frac{de(t)}{dt} \quad (2.7)$$

dengan T_d sebagai *derivative time*. Aksi *derivative* dapat membantu mengurangi kecenderungan *overshoot* dan meningkatkan peredaman respons. Namun, komponen *derivative* juga sensitif terhadap *noise* pengukuran karena perubahan kecil pada sinyal dapat menghasilkan nilai turunan yang besar.

2.6.4. Persamaan PID

Metode *Proportional-Integral-Derivative* merupakan metode pengendalian umpan balik yang bekerja berdasarkan nilai kesalahan (*error*) antara nilai acuan (*setpoint*) dan nilai proses atau *process variable*. Pada implementasi penelitian ini, nilai acuan direpresentasikan oleh masukan SET, sedangkan nilai proses aktual direpresentasikan oleh masukan ACT pada *Function Block* CTRL_PID OpenPLC. Selisih antara kedua nilai tersebut menghasilkan sinyal kesalahan yang digunakan oleh pengendali untuk menentukan aksi kontrol terhadap pompa.

Secara matematis, nilai kesalahan dapat dinyatakan sebagai:

$$e(t) = \text{SET}(t) - \text{ACT}(t)$$

Sinyal keluaran pengendali PID dapat dinyatakan dalam bentuk gain proporsional, integral, dan derivatif sebagai berikut:

$$u(t) = K_p e(t) + K_i \int e(t) dt + K_d \frac{de(t)}{dt} \quad (2.8)$$

Dimana:

- $u(t)$: sinyal keluaran kontrol
- $e(t)$: nilai *error*, yaitu selisih antara *setpoint* dan *process variable*
- K_p : konstanta proporsional
- T_i : waktu integral
- T_d : waktu derivatif

Persamaan tersebut menunjukkan bahwa keluaran pengendali merupakan kombinasi aksi proporsional, integral, dan derivatif. Pada penelitian ini, parameter pengendali yang digunakan dalam proses analisis dan tuning dinyatakan dalam bentuk K_p , K_i , dan K_d . Penyesuaian parameter tersebut bertujuan memperoleh

respons laju aliran yang mampu mengikuti nilai setpoint dengan karakteristik transien dan kondisi steady-state yang sesuai.

Pada penelitian ini, parameter yang digunakan secara langsung pada Function Block CTRL_PID adalah KP, TN, dan TV. Nilai SET pada blok tersebut berfungsi sebagai nilai acuan, sedangkan ACT merupakan nilai aktual laju aliran hasil pengukuran sensor YF-S201 yang telah melalui proses akuisisi dan pengolahan data. Keluaran CTRL_PID selanjutnya digunakan sebagai dasar pembentukan sinyal kendali menuju driver BTS7960 untuk mengatur kerja pompa sehingga laju aliran dapat dipertahankan mendekati nilai acuan.

2.6.5. Metode *Tuning Ziegler–Nichols Open-Loop*

LRMetode tuning Ziegler–Nichols *open-loop* atau *process reaction curve* dilakukan dengan memberikan perubahan masukan tetap pada sistem dan mengamati respons keluaran terhadap waktu. Respons digunakan untuk memperoleh karakteristik proses berupa *dead time* (L) dan kemiringan maksimum respons (R). Kemiringan maksimum dapat dinyatakan sebagai $R = \Delta Q / \Delta t$ pada bagian kurva dengan gradien terbesar. Parameter tersebut selanjutnya digunakan untuk memperoleh parameter awal pengendali.

Pada penelitian ini metode Ziegler–Nichols *open-loop* digunakan untuk menentukan parameter awal pengendali PID berdasarkan respons laju aliran air. Sistem diberikan nilai PWM secara konstan dan perubahan laju aliran diamati hingga mencapai kondisi relatif stabil. Data respons tersebut kemudian digunakan untuk menentukan parameter kontrol.

Metode Ziegler–Nichols yang digunakan pada penelitian ini adalah metode *open-loop process reaction curve*, bukan metode *closed-loop ultimate gain*. Pengujian dilakukan dengan memberikan perubahan masukan step pada *plant* dan merekam respons laju aliran terhadap waktu.

Perubahan masukan dinyatakan sebagai:

$$\Delta u = u_1 - u_0$$

Kemiringan maksimum respons diperoleh dari:

$$R = \frac{\Delta Q}{\Delta t} \quad (2.11)$$

Parameter awal PID kemudian dihitung menggunakan:

$$K_p = \frac{1.2\Delta u}{RL} \quad (2.12)$$

$$T_i = 2L \quad (2.13)$$

$$T_d = 0.5L \quad (2.14)$$

Parameter T_i dan T_d merupakan parameter waktu yang diperoleh dari metode Ziegler–Nichols. Agar hasil tuning dinyatakan dalam bentuk gain PID yang digunakan pada penelitian ini, nilai tersebut selanjutnya dikonversi menjadi K_i dan K_d menggunakan:

$$K_i = \frac{K_p}{T_i} \quad (2.15)$$

$$K_d = K_p T_d \quad (2.16)$$

2.6.6. Implementasi PID pada OpenPLC

Pada penelitian ini, algoritma PID diimplementasikan pada OpenPLC menggunakan *Function Block* CTRL_PID. Blok CTRL_PID merupakan blok kontrol PID yang memiliki masukan berupa nilai aktual (*actual value*), nilai *setpoint*, parameter kontrol, serta beberapa parameter pendukung lainnya. Parameter utama kontrol PID pada blok tersebut dinyatakan dalam bentuk KP, TN, dan TV, Meskipun *Function Block* CTRL_PID menggunakan nama masukan KP, TN, dan TV, parameter pengendali dalam pembahasan dan proses tuning penelitian ini dinyatakan dalam bentuk gain K_p , K_i , dan K_d . Nilai K_i dan K_d kemudian direpresentasikan ke parameter internal CTRL_PID menggunakan hubungan:

$$TN = \frac{K_p}{K_i} \quad (2.17)$$

$$TV = \frac{K_d}{K_p} \quad (2.18)$$

Dengan demikian, TN dan TV hanya digunakan sebagai parameter implementasi internal *Function Block* CTRL_PID, sedangkan parameter hasil tuning dan analisis tetap dinyatakan sebagai K_p , K_i , dan K_d . sedangkan nilai keluaran kontrol dihasilkan melalui variabel Y [33].

Berdasarkan dokumentasi CTRL_PID pada OSCAT Basic, nilai kesalahan pengendalian dinyatakan sebagai selisih antara nilai SET dan ACT [33]. Hubungan tersebut dinyatakan sebagai:

$$\mathbf{DIFF = SET - ACT} \quad (2.7)$$

Dengan:

SET : nilai setpoint

ACT : nilai aktual variabel proses

DIFF : selisih antara *setpoint* dan nilai aktual

Berdasarkan dokumentasi OSCAT Basic, prinsip pengendalian pada *function block* CTRL_PID menggunakan parameter KP sebagai penguatan proporsional, TN sebagai waktu integral, dan TV sebagai waktu derivatif. Persamaan pengendalian CTRL_PID dapat dinyatakan sebagai [33]:

$$Y = KP \left(\text{DIFF} + \frac{1}{TN} \int \text{DIFF} dt + TV \frac{d(\text{DIFF})}{dt} \right) + \text{OFS} \quad (2.15)$$

dengan:

Y : keluaran kontrol,

KP : penguatan proporsional,

TN : waktu integral,

TV : waktu derivatif,

OFS : nilai *offset* keluaran.

Berdasarkan hubungan parameter yang digunakan pada CTRL_PID, parameter TN dan TV berkaitan dengan gain integral dan derivatif melalui $TN = KP/KI$ dan $TV = Kd/Kp$ [33]. Dengan melakukan penyusunan ulang secara aljabar, hubungan tersebut dapat dinyatakan sebagai:

$$K_I = \frac{K_P}{T_N} \quad (2.16)$$

dan

$$K_D = K_P \times T_V \quad (2.17)$$

Pada persamaan (2.16) dan persamaan (2.17) merupakan bentuk ekuivalen apabila parameter PID dinyatakan dalam bentuk gain konvensional. Pada implementasi penelitian ini, parameter yang dimasukkan secara langsung ke function block CTRL_PID tetap berupa KP, TN, dan TV.

Struktur dan parameter CTRL_PID juga diverifikasi melalui *source code* CTRL_PID.EXP OSCAT [34]. *Source code* tersebut menunjukkan penggunaan masukan ACT dan SET serta parameter KP, TN, TV, OFS, LL, dan LH pada struktur pengendali.

Struktur serta parameter CTRL_PID tersebut juga dapat dilihat secara langsung pada *source code* blok, dimana masukan yang digunakan antara lain ACT, SET, SUP, OFS, M_I, MAN, RST, KP, TN, TV, LL, dan LH, sedangkan keluarannya meliputi Y, DIFF, dan LIM [34].

Pada penelitian ini, nilai laju aliran air yang telah melalui proses *scaling* dan *filtering* digunakan sebagai ACT, sedangkan setpoint laju aliran digunakan sebagai SET. *Function Block* CTRL_PID memproses selisih kedua nilai tersebut berdasarkan parameter KP, TN, dan TV untuk menghasilkan keluaran kendali Y. Keluaran tersebut dikirim melalui jalur *pin-to-pin* menuju ESP32 pendukung, kemudian diterapkan sebagai Motor PWM pada *driver* BTS7960 untuk mengatur kecepatan pompa.

Program pengendalian pada OpenPLC dieksekusi secara berulang melalui mekanisme tugas siklik (*cyclic task*). Pada setiap siklus eksekusi, program membaca atau memperbarui nilai variabel proses, menjalankan logika pengendalian, kemudian menghasilkan nilai keluaran sesuai dengan program yang telah dirancang [32]. Mekanisme tugas siklik merupakan bentuk eksekusi yang umum digunakan pada aplikasi kontrol PLC.

Dalam penelitian ini, penggunaan PID tidak menjadi fokus utama penelitian, tetapi digunakan sebagai algoritma kontrol untuk mengevaluasi implementasi OpenPLC berbasis ESP32 dalam menjalankan fungsi pengendalian secara *closed-loop*. Oleh karena itu, parameter PID yang telah diperoleh melalui metode *tuning* kemudian diterapkan pada blok CTRL_PID, sedangkan kinerja

implementasi OpenPLC dievaluasi berdasarkan respons laju aliran air yang dihasilkan.

2.6.7. *Function Block* CTRL_PID

Function Block CTRL_PID merupakan blok fungsi yang digunakan pada penelitian ini untuk mengimplementasikan algoritma *Proportional-Integral-Derivative* pada OpenPLC. *Function Block* tersebut menerima nilai aktual proses melalui masukan ACT, nilai referensi melalui masukan SET, serta parameter pengendali berupa KP, TN, dan TV. Keluaran pengendali dihasilkan melalui variabel Y.

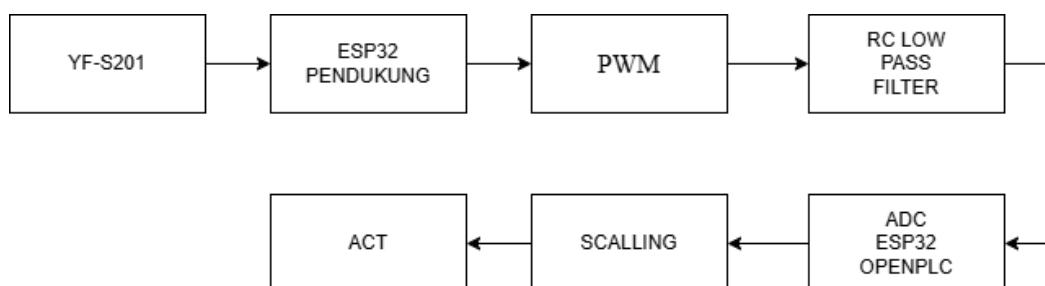
Parameter KP merupakan penguatan proporsional, TN merupakan waktu integral, sedangkan TV merupakan waktu derivatif. Pada penelitian ini, nilai laju aliran yang telah diproses digunakan sebagai ACT, sedangkan nilai *setpoint* laju aliran digunakan sebagai SET. Keluaran Y selanjutnya digunakan sebagai dasar pembentukan sinyal kendali pompa.

Penggunaan *Function Block* CTRL_PID memungkinkan algoritma PID dijalankan secara siklik oleh OpenPLC *Runtime* tanpa membangun persamaan PID secara manual di dalam program utama.

2.7. Hubungan Antar Sistem dalam Penelitian

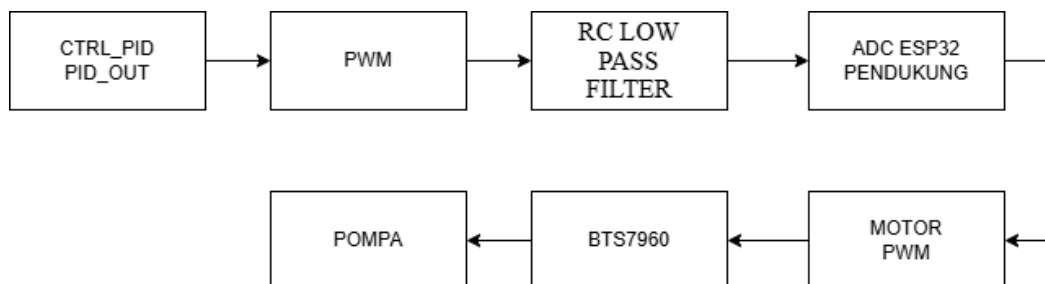
Pada penelitian ini digunakan dua ESP32 dengan fungsi yang berbeda. ESP32 pendukung menangani pembacaan sensor YF-S201 dan penerapan sinyal kendali terhadap aktuator, sedangkan ESP32 OpenPLC menjalankan OpenPLC *Runtime*, program FBD, dan *Function Block* CTRL_PID.

Jalur *feedback* laju aliran berlangsung melalui:



Gambar 2.3 Diagram Blok Feedback Laju Aliran

Setelah nilai ACT dibandingkan dengan SET, *Function Block* CTRL_PID menghasilkan keluaran Y yang disimpan sebagai PID_OUT. Jalur keluaran kendali berlangsung melalui:

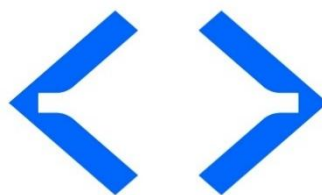


Gambar 2.4 Diagram Blok Jalur Keluaran Kendali

Kedua jalur tersebut membentuk sistem closed-loop. Sensor JSN-SR04T tidak digunakan sebagai process variable PID, tetapi sebagai sensor level dan *interlock* untuk menghentikan pompa ketika level air mencapai batas yang ditentukan.

2.8. OpenPLC Editor

OpenPLC Editor merupakan sebuah *software open-source* yang digunakan untuk membuat, mengedit, dan mengompilasi program berbasis standar IEC 61131-3. *Software* ini dirancang untuk memprogram sistem kendali berbasis PLC secara fleksibel dan gratis. OpenPLC editor ini berfungsi untuk membuat logika kontrol dengan berbagai bahasa pemrograman seperti *Ladder Diagram*, *Function Block Diagram*, *Structured Text*, dan *Instruction List* [5].



Gambar 2.5 OpenPLC Editor

Hasil program dari OpenPLC Editor ini bisa di upload ke berbagai *platform* seperti ESP32, Arduino, Raspberry Pi, maupun OpenPLC *Runtime*. OpenPLC Editor mengacu pada standar *internasional* IEC 61131-3, yaitu standar untuk bahasa pemrograman PLC [23]. Standar ini bertujuan untuk menyediakan struktur pemrograman yang seragam, memudahkan integrasi antar sistem, dan

meningkatkan portabilitas program. Secara umum, sistem OpenPLC Editor terdiri dari tiga bagian utama, yaitu:

a. *OpenPLC Editor*

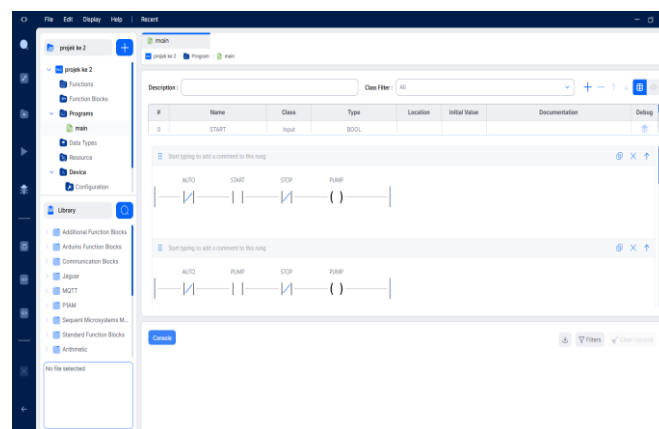
Digunakan untuk membuat program kontrol. Pada penelitian ini bahasa pemrograman utama yang digunakan adalah *Function Block Diagram* (FBD).

b. *OpenPLC Runtime*

Berfungsi untuk eksekutor program yang telah dikompilasi

c. *Hardware Target*

Perangkat fisik seperti ESP32, Arduino, atau Raspberry Pi.



Gambar 2.6 Tampilan Pada OpenPLC Editor

OpenPLC Editor memiliki beberapa fitur seperti *multi-platform*, mendukung berbagai *hardware*, *open source*, dan integrasi dengan modbus TCP/IP. Selain memiliki fitur seperti yang telah disebutkan, aplikasi ini juga memiliki kekurangan seperti performa tergantung *hardware* yang digunakan, dan dokumentasi masih terbatas dibanding PLC komersial [24].[25]

2.8.1. OpenPLC Runtime

OpenPLC Runtime adalah suatu komponen yang berfungsi untuk mengeksekusi program yang telah dibuat di OpenPLC editor. Program pengendalian pada penelitian ini dirancang menggunakan *Function Block Diagram* (FBD) pada OpenPLC Editor. Program tersebut selanjutnya dikompilasi sebelum

dijalankan pada perangkat target ESP32.. Dengan demikian, OpenPLC *runtime* berfungsi sebagai pengeksekusi yang menghubungkan logika program dengan perangkat input dan output pada sistem kendali.

Pada penelitian ini, OpenPLC *Runtime* diimplementasikan pada ESP32 sebagai *soft programmable logic controller*. Data laju aliran tidak dibaca secara langsung dari YF-S201 oleh ESP32 OpenPLC, tetapi diterima dari ESP32 pendukung melalui jalur PWM–RC low-pass filter–ADC. Data tersebut diproses menggunakan program *Function Block Diagram* (FBD), kemudian CTRL_PID menghasilkan keluaran kendali yang dikirim kembali menuju ESP32 pendukung untuk diterapkan pada *driver* BTS7960 dan pompa.

2.8.2. OpenPLC

OpenPLC merupakan sistem *programmable logic controller open-souce* yang dirancang untuk menjalankan logika kontrol menggunakan perangkat keras berbasis mikrokontroler. Berbeda dengan PLC konvensional, OpenPLC ini bersifat gratis, dan fleksibel yang dimana dapat dijalankan di berbagai platform seperti ESP32, arduino, dan raspberry Pi [17]. Arsitektur OpenPLC terdiri dari beberapa bagian berikut:

- a. *Programming Layer* menggunakan OpenPLC Editor, aplikasi ini mendukung bahasa IEC 61131-3 seperti *ladder diagram*, *structured Text*, dan *function block diagram*.
- b. *Compilation Layer*, dimana nantinya program dikompilasi menjadi kode c menggunakan *tool MatIEC compiler*
- c. *Runtime Layer*, OpenPLC Runtime mengeksekusi program yang berjalan di Mikrokontroler.
- d. *Hardware layer* meliputi sensor sebagai input, aktuator sebagai output, dan komunikasi menggunakan *Modbus* dll

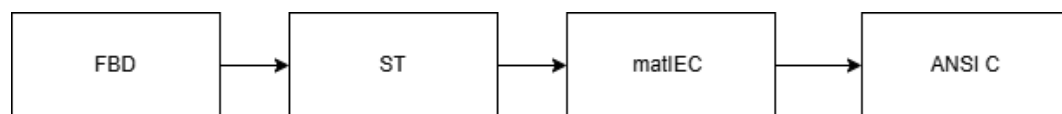
Cara kerja OpenPLC pada ESP32 dimulai dari pembuatan program menggunakan OpenPLC Editor, kemudian program dikompilasi dan diunggah ke perangkat target. Pada penelitian ini ESP32 OpenPLC menjalankan OpenPLC *Runtime* dan program FBD sebagai pengendali utama. Data sensor *flow* diterima dari ESP32 pendukung melalui jalur PWM–RC–ADC, sedangkan keluaran kendali

dikirim kembali ke ESP32 pendukung untuk diterapkan pada aktuator. OpenPLC juga mendukung beberapa mekanisme komunikasi lain, seperti *Modbus*, tetapi protokol tersebut bukan jalur utama komunikasi kontrol pada penelitian ini.

2.8.3. MatIEC Compiler

MatIEC merupakan *compiler* atau *code translator* sumber terbuka yang digunakan untuk menerjemahkan program berbasis bahasa IEC 61131-3 ke dalam kode yang dapat diproses lebih lanjut oleh sistem komputasi. MatIEC dikembangkan untuk mendukung beberapa bahasa tekstual IEC 61131-3, yaitu *Structured Text* (ST), *Instruction List* (IL), dan bentuk tekstual *Sequential Function Chart* (SFC) [35]. MatIEC menyediakan program *iec2c* yang berfungsi menghasilkan kode ANSI C yang ekuivalen dengan program IEC 61131-3 yang diberikan sebagai masukan.

Dalam lingkungan OpenPLC, program pengendalian dapat dibuat menggunakan bahasa grafis seperti *Ladder Diagram* (LD) maupun *Function Block Diagram* (FBD). Program grafis tersebut tidak langsung diproses oleh MatIEC. OpenPLC Editor terlebih dahulu mengubah representasi program menjadi bentuk tekstual *Structured Text*, kemudian kode ST tersebut diteruskan ke *iec2c* untuk diterjemahkan menjadi kode C [36]. Mekanisme tersebut memungkinkan program yang dirancang secara grafis tetap dapat diproses menggunakan toolchain MatIEC. Secara umum proses kompilasi dapat digambarkan sebagai:



Gambar 2.7 Proses Kompilasi

MatIEC melakukan proses penerjemahan melalui beberapa tahap, yaitu analisis leksikal, analisis sintaksis, analisis semantik, dan pembangkitan kode. Pada tahap akhir, *code generator* menghasilkan kode C yang merepresentasikan logika program PLC [35], [37]. Struktur tersebut memungkinkan kode hasil penerjemahan digunakan pada berbagai platform selama tersedia *compiler* C dan lapisan perangkat keras yang sesuai.

Pada OpenPLC, kode hasil MatIEC tidak berdiri sendiri. Kode logika PLC yang dihasilkan selanjutnya digabungkan dengan komponen OpenPLC lainnya, seperti pemetaan input/output, *hardware layer*, tabel memori, serta fungsi runtime yang diperlukan agar program dapat berinteraksi dengan perangkat keras [38]. Setelah seluruh bagian tersebut digabungkan, kode dikompilasi menggunakan toolchain perangkat target sehingga dapat dijalankan pada perangkat keras yang digunakan.

Pada penelitian ini, program kontrol yang dibuat pada OpenPLC Editor, termasuk penggunaan Function Block CTRL_PID, menjadi bagian dari logika program PLC yang diproses dalam proses kompilasi. Logika tersebut kemudian diterjemahkan menjadi kode C dan diintegrasikan dengan OpenPLC Runtime sehingga dapat dieksekusi pada mikrokontroler ESP32. Dengan mekanisme tersebut, pengguna tetap dapat merancang logika kontrol menggunakan bahasa PLC tanpa harus menulis seluruh algoritma kendali langsung dalam bahasa C.

2.8.4. Scan Cycle OpenPLC

Programmable Logic Controller bekerja menggunakan mekanisme yang dikenal sebagai *scan cycle*, yaitu proses eksekusi program yang dilakukan secara berulang selama PLC berada dalam kondisi aktif. Pada setiap siklus, PLC akan membaca seluruh sinyal masukan, menjalankan logika program yang telah dibuat, memperbarui kondisi keluaran, kemudian melakukan proses internal sebelum kembali mengulangi siklus berikutnya. Waktu yang dibutuhkan untuk menyelesaikan satu siklus disebut sebagai *scan time*, yang umumnya berada pada orde milidetik bergantung pada kompleksitas program dan spesifikasi perangkat keras plc. *Scan cycle* memungkinkan PLC memberikan respons yang cepat terhadap perubahan kondisi pada sistem sehingga banyak digunakan pada aplikasi otomasi industri yang memerlukan operasi secara *real-time*.

Tahap pertama dalam *scan cycle* adalah *input scan*, yaitu proses pembacaan seluruh kondisi masukan dari sensor atau perangkat input yang terhubung ke PLC. Data hasil pembacaan tersebut kemudian disimpan ke dalam memori sehingga digunakan sebagai acuan selama proses eksekusi program. Setelah seluruh data masukan diperoleh, PLC akan mengeksekusi logika kontrol sesuai program yang

telah dibuat menggunakan bahasa pemrograman IEC 61131-3, seperti *Ladder Diagram* maupun *Function Block Diagram*.

Tahap berikutnya adalah *output update*, yaitu proses pengiriman hasil eksekusi program menuju perangkat keluaran, seperti motor, pompa, maupun aktuator lainnya. Setelah seluruh keluaran diperbarui, PLC akan melakukan proses internal (*housekeeping*) yang meliputi komunikasi data, diagnostik sistem, serta pengelolaan memori sebelum kembali memulai siklus berikutnya. Karena seluruh tahapan tersebut berlangsung secara berulang dengan waktu yang sangat singkat, PLC mampu mempertahankan kestabilan sistem kontrol meskipun terjadi perubahan kondisi proses secara terus-menerus.

Pada penelitian ini, mekanisme scan cycle diterapkan pada ESP32 OpenPLC. Setiap siklus memperbarui process variable flow yang diterima dari ESP32 pendukung melalui jalur PWM–RC–ADC, menjalankan logika FBD dan Function Block CTRL_PID, kemudian memperbarui PID_OUT. Keluaran tersebut dikirim kembali ke ESP32 pendukung untuk dibentuk menjadi Motor PWM pada BTS7960. Siklus berlangsung terus-menerus selama sistem aktif sehingga koreksi closed-loop dapat dilakukan secara periodik.

2.8.5. Function Block Diagram Pada OpenPLC

Function Block Diagram (FBD) merupakan salah satu bahasa pemrograman standar yang ditetapkan dalam standar IEC 61131-3 untuk pemrograman *Programmable Logic Controller* (PLC). Berbeda dengan *Ladder Diagram* yang menyerupai rangkaian relay, *Function Block Diagram* menggunakan blok-blok fungsi yang saling terhubung melalui jalur aliran data (*data flow*). Setiap blok memiliki masukan (*input*) dan keluaran (*output*) yang digunakan untuk melakukan operasi logika, aritmatika, perbandingan, maupun fungsi kontrol lainnya.

Pada OpenPLC, *Function Block Diagram* banyak digunakan karena memiliki tampilan yang lebih sederhana dan mudah dipahami, terutama pada implementasi sistem kendali yang melibatkan proses perhitungan seperti konversi data sensor, pengolahan sinyal, serta algoritma kontrol PID. Selain itu, penggunaan *Function Block Diagram* mempermudah proses pemrograman karena setiap fungsi

direpresentasikan dalam bentuk blok yang memiliki karakteristik tertentu sesuai dengan operasi yang dilakukan.

Dalam penelitian ini, *Function Block Diagram* (FBD) digunakan sebagai bahasa pemrograman utama pada OpenPLC untuk melakukan scaling dan filtering data flow, membentuk logika interlock level, serta menjalankan Function Block CTRL_PID. Blok yang digunakan meliputi operasi aritmatika, perbandingan, konversi tipe data, pemindahan data, dan blok pengendali PID.

2.8.6. *Function Block* yang Digunakan pada Penelitian

Pada penelitian ini digunakan beberapa *function block* bawaan OpenPLC untuk melakukan pengolahan data sensor dan pengendalian aktuator. *Function block* tersebut meliputi DIV, MUL, ADD, SUB, MOVE, LT, GE, TO_REAL, serta PID. Masing-masing *function block* memiliki fungsi yang berbeda sesuai dengan kebutuhan sistem pengendalian.

a. *Function Block* DIV

Function block DIV (*Division*) digunakan untuk melakukan operasi pembagian antara dua buah data numerik. Nilai pada masukan pertama (IN1) akan dibagi dengan nilai pada masukan kedua (IN2) sehingga menghasilkan nilai keluaran (OUT).

Secara matematis operasi pembagian tersebut dinyatakan sebagai

$$OUT = \frac{IN1}{IN2} \quad (2.18)$$

Function block DIV umumnya digunakan untuk proses penskalaan data sensor, normalisasi data, maupun proses konversi nilai agar sesuai dengan kebutuhan sistem pengendalian.

b. *Function Block* MUL (*Multiply*)

Function block MUL (*Multiply*) digunakan untuk melakukan operasi perkalian antara dua buah data numerik. Persamaan yang digunakan yaitu

$$OUT = IN1 \times IN2$$

Function block ini banyak digunakan untuk proses penguatan sinyal, proses kalibrasi data sensor, maupun perhitungan matematis lainnya.

c. Function Block ADD (Addition)

Function Block ADD digunakan untuk menjumlahkan dua buah data numerik sehingga menghasilkan nilai keluaran. Persamaan yang digunakan yaitu

$$OUT = IN1 + IN2$$

Function blok ADD umumnya digunakan untuk memberikan nilai *offset* maupun melakukan penjumlahan beberapa parameter pada proses pengendalian.

d. Function Block SUB (Subtraction)

Function blok SUB digunakan untuk menghitung selisih antara dua buah nilai numerik. Persamaan yang digunakan yaitu

$$OUT = IN1 - IN2$$

Function blok SUB banyak digunakan dalam proses kalibrasi sensor, perhitungan error, maupun pengolahan data hasil pengukuran.

e. Function Block MOVE

Function blok MOVE digunakan untuk memindahkan nilai dari variabel sumber menuju variabel tujuan tanpa mengubah nilai tersebut. Nilai masukan (*IN*) akan langsung disalin ke variabel keluaran (*OUT*).

Persamaan yang digunakan yaitu

$$OUT = IN$$

Blok ini umumnya digunakan untuk proses inisialisasi nilai, pemindahan data antar variabel, serta penyimpanan nilai hasil perhitungan.

f. Function Block LT (Less Than)

Function blok LT (*Less Than*) merupakan blok pembandingan yang digunakan untuk mengetahui apakah nilai pada masukan pertama lebih kecil dibandingkan dengan nilai pada masukan kedua.

Secara logika dituliskan sebagai berikut

$$OUT = \begin{cases} \text{TRUE}, & IN1 < IN2, \\ \text{FALSE}, & IN1 \geq IN2. \end{cases} \quad (2.19)$$

Function block LT digunakan pada sistem kendali untuk menghasilkan keputusan berdasarkan kondisi tertentu.

g. ***Function Block GE (Greater Than or Equal)***

Function block GE (*Greater Than or Equal*) digunakan untuk membandingkan dua buah nilai numerik. Apabila nilai masukan pertama lebih besar atau sama dengan masukan kedua, maka keluaran bernilai *TRUE*. Sebaliknya apabila lebih kecil, maka keluaran bernilai *FALSE*. Secara logika, blok GE menghasilkan *TRUE* apabila $IN1 \geq IN2$ dan *FALSE* apabila syarat tersebut tidak terpenuhi.

$$OUT = \begin{cases} TRUE, & IN1 \geq IN2, \\ FALSE, & IN1 < IN2. \end{cases} \quad (2.20)$$

Function block ini banyak digunakan sebagai logika keputusan dalam sistem otomatis.

h. ***Function Block TO_REAL***

Function block TO_REAL digunakan untuk mengubah tipe data numerik menjadi tipe data REAL. Konversi ini diperlukan ketika suatu nilai akan digunakan pada proses perhitungan yang memerlukan bilangan pecahan, seperti algoritma PID.

Secara umum proses konversi dapat dituliskan sebagai

$$OUT = REAL(IN)$$

Penggunaan tipe data *REAL* menghasilkan ketelitian yang lebih baik dibandingkan tipe data integer, terutama pada proses kontrol.

i. ***Function Block CTRL PID***

Function Block CTRL_PID merupakan salah satu function block bawaan OpenPLC yang digunakan untuk mengimplementasikan algoritma Proportional Integral Derivative (PID) pada sistem pengendalian. Function block ini menerima nilai setpoint (SP) dan process variable (PV) sebagai masukan, kemudian menghasilkan sinyal kendali (CV) berdasarkan selisih antara kedua nilai tersebut atau yang disebut sebagai error.

CTRL_PID bekerja dengan menggabungkan tiga aksi pengendalian, yaitu aksi proporsional (*Proportional*), aksi integral (*Integral*), dan aksi diferensial (*Derivative*). Kombinasi ketiga aksi tersebut bertujuan menghasilkan respon sistem yang cepat, stabil, serta mampu meminimalkan kesalahan terhadap nilai setpoint.

Persamaan umum PID telah dijelaskan pada Subbab 2.6.4. Pada bagian ini penjelasan difokuskan pada fungsi dan parameter Function Block CTRL_PID yang digunakan pada implementasi OpenPLC.

Pada OpenPLC, *Function Block* CTRL_PID menerima ACT sebagai nilai aktual, SET sebagai setpoint, serta parameter KP, TN, dan TV. Keluaran Y digunakan sebagai sinyal kendali, sedangkan LL dan LH membatasi rentang keluaran sesuai kebutuhan sistem.

Dalam penelitian ini, *Function Block* CTRL_PID digunakan untuk mengendalikan laju aliran air. Nilai setpoint menjadi masukan SET, sedangkan process variable hasil pengolahan sensor flow menjadi masukan ACT. Keluaran CTRL_PID tidak langsung diberikan ke BTS7960, tetapi diteruskan melalui jalur komunikasi pin-to-pin menuju ESP32 pendukung, kemudian digunakan sebagai Motor PWM untuk mengendalikan driver BTS7960 dan pompa.

2.9. Power Supply

Power Supply merupakan komponen yang digunakan untuk mengubah energi listrik dari sumber utama menjadi energi listrik lain sesuai dengan kebutuhan beban. Dalam sistem elektronika, *power supply* umumnya digunakan untuk mengubah tegangan AC dari jaringan listrik PLN menjadi DC yang stabil agar dapat digunakan oleh perangkat seperti mikrokontroler, sensor, maupun aktuator [25].



Gambar 2.8 Power Supply 24V

Pada umumnya, *power supply* terdiri dari beberapa bagian utama, yaitu transformator (pada tipe tertentu), penyearah (*rectifier*), filter, dan regulator tegangan [26]. Proses dimulai dari penurunan atau penyesuaian tegangan AC, kemudian disearahkan menjadi tegangan DC menggunakan dioda, setelah itu diratakan menggunakan kapasitor filter, dan akhirnya distabilkan oleh rangkaian regulator agar menghasilkan tegangan keluaran yang konstan.

Pada penelitian ini digunakan *power supply* 24V DC, berikut spesifikasi umum dari *power supply* 24V DC.

Tabel 2.4 Spesifikasi Power Supply

Parameter	Spesifikasi
Tegangan Input	110 – 220 V AC
Frekuensi input	50/60 Hz
Tegangan Output	24 V DC
Arus output maksimum	40 A

2.10. Buck Converter

Buck converter merupakan salah satu jenis DC-DC *converter* yang berfungsi untuk menurunkan tegangan DC dari nilai yang lebih tinggi menjadi lebih rendah dengan efisiensi tinggi. Berbeda dengan regulator linear, *buck converter* bekerja menggunakan prinsip *switching* dengan memanfaatkan komponen utama seperti transistor (*switch*), dioda, induktor, dan kapasitor [27].



Gambar 2.9 Buck Converter

Prinsip kerja *buck converter* dimulai saat saklar (*switch*) dalam kondisi *ON*, di mana arus mengalir melalui induktor dan menyimpan energi dalam bentuk medan magnet. Ketika saklar *OFF*, energi yang tersimpan pada induktor dilepaskan melalui dioda ke beban [28]. Proses *switching* ini terjadi secara cepat (frekuensi tinggi) sehingga menghasilkan tegangan keluaran yang stabil setelah difilter oleh kapasitor.

Hubungan antara tegangan input dan output pada *buck converter* dapat dinyatakan sebagai:

$$V_{\text{out}} = D \times V_{\text{in}} \quad (2.21)$$

di mana:

- V_{out} = tegangan output
- V_{in} = tegangan input
- D = *duty cycle* (0–1)

Dengan prinsip ini, *buck converter* mampu menghasilkan tegangan yang lebih rendah dengan efisiensi yang umumnya mencapai 80–95%, sehingga sangat cocok digunakan pada sistem berbasis mikrokontroler dan aktuator [27]. Pada penelitian ini digunakan dua buah *buck converter*, yaitu:

- a. *Step-down* 24V ke 12V
- b. *Step-down* 12V ke 5V

Tabel 2.5. Spesifikasi *Buck Converter* 24V ke 12V

Parameter	Spesifikasi
Tegangan Input	18 – 36 V DC
Daya Maksimum	Hingga 240 Watt
Tegangan Output	12 V DC
Arus maksimum	10 – 20 A
Frekuensi <i>Switching</i>	150 kHz

Tabel 2.6. Spesifikasi *Buck Converter* 12V ke 5V

Parameter	Spesifikasi
Tegangan Input	7 – 24 V DC
Daya Maksimum	Hingga 25 Watt
Tegangan Output	5V DC
Arus maksimum	2 – 5 A

2.11. Pompa 12V

Pompa 12V merupakan aktuator utama dalam sistem pengendalian laju aliran air yang berfungsi untuk mengalirkan fluida dari satu titik ke titik lainnya dalam rangkaian sistem [29]. Pompa ini bekerja menggunakan sumber tegangan DC sebesar 12 Volt dan memiliki konsumsi arus yang bergantung pada spesifikasi serta beban kerja yang diberikan. Dalam penelitian ini, pompa digunakan untuk menghasilkan dan mengatur laju aliran air yang akan dikendalikan menggunakan metode PID.

Pengendalian pompa dilakukan melalui sinyal *Pulse Width Modulation* (PWM) yang dihasilkan oleh mikrokontroler ESP32 dan diteruskan ke rangkaian driver [30]. Dengan metode ini, kecepatan putaran pompa dapat diatur secara variatif sesuai dengan output dari kontrol PID, sehingga laju aliran air dapat disesuaikan dengan nilai setpoint yang diinginkan. Semakin besar nilai *duty cycle* PWM, maka semakin tinggi kecepatan pompa dan debit aliran air yang dihasilkan.



Gambar 2.10 Pompa 12V

Pompa 12V yang digunakan umumnya memiliki karakteristik berupa tegangan kerja 12 V DC, arus kerja sekitar 1–2 A, serta daya berkisar antara 12 hingga 24 Watt. Pompa ini juga memiliki keunggulan dalam hal kemudahan pengendalian, ukuran yang relatif kecil, serta kompatibilitas yang baik dengan sistem berbasis mikrokontroler. Dengan demikian, pompa 12V sangat sesuai digunakan sebagai aktuator dalam sistem pengendalian laju aliran air berbasis ESP32 dan OpenPLC.

2.12. Sensor Ultrasonik JSN-SR04T

Sensor ultrasonik JSN-SR04T digunakan untuk mengukur jarak antara posisi sensor dan permukaan air. Sensor bekerja dengan memancarkan gelombang ultrasonik sekitar 40 kHz melalui transmitter dan menerima pantulannya melalui receiver. Karena waktu yang diukur merupakan waktu tempuh pergi-pulang gelombang, jarak dihitung menggunakan setengah dari hasil perkalian kecepatan suara dengan waktu tempuh pulsa.



Gambar 2.8 Sensor Ultrasonik JSN-SR04T

Untuk menghitung jarak dari sensor ke permukaan air digunakan persamaan

$$d = \frac{v \cdot t}{2} \quad (2.22)$$

Dimana:

d = jarak (cm)

v = kecepatan suara di udara

t = waktu tempuh pulsa (detik)

kemudian dibagi 2 karena gelombang bolak-balik

untuk perhitungan selisih pembacaan dihitung menggunakan persamaan

$E = d_{\text{sensor}} - d_{\text{referensi}}$

Dengan:

E : selisih pembacaan (cm)

d_{sensor} : jarak hasil pembacaan JSN-SR04T (cm)

$d_{\text{referensi}}$: jarak hasil pengukuran referensi (cm)

Untuk mengetahui besar selisih pembacaan tanpa mempertimbangkan arah kesalahan, digunakan nilai absolut:

$$\Delta d = |d_{\text{sensor}} - d_{\text{referensi}}| \quad (2.23)$$

Dengan:

Δd : selisih absolut pembacaan (cm)

d_{sensor} : jarak hasil pembacaan JSN-SR04T (cm)

$d_{\text{referensi}}$: jarak hasil pengukuran referensi (cm)

Persentase error kemudian dihitung menggunakan:

$$Error = \frac{\Delta d}{d_{\text{referensi}}} \times 100\% \quad (2.24)$$

Perhitungan dilakukan pada setiap kondisi pengujian untuk mengetahui besar penyimpangan pembacaan sensor terhadap nilai referensi.

Dalam aplikasi pengukuran level air, sensor JSN-SR04T tidak secara langsung mengukur tinggi air, tetapi mengukur jarak antara sensor dan permukaan air. Apabila posisi sensor terhadap dasar tangki dinyatakan sebagai H , sedangkan jarak antara sensor dan permukaan air dinyatakan sebagai d , maka tinggi permukaan air dapat dihitung menggunakan:

$$h = H - d$$

Hubungan $h = H - d$ digunakan untuk memperoleh tinggi permukaan air dari jarak yang dibaca sensor. Nilai H aktual yang digunakan pada prototipe dan contoh perhitungannya dijelaskan pada Bab III.

Sensor JSN-SR04T memiliki keterbatasan berupa jarak ukur minimum atau zona buta. Berdasarkan spesifikasi sensor yang digunakan, jarak minimum pengukuran berada pada kisaran 20 cm. Apabila permukaan air terlalu dekat dengan sensor dan masuk ke daerah tersebut, pembacaan dapat menjadi kurang representatif. Pada prototipe penelitian ini, sensor dipasang pada bagian atas tangki dengan jarak referensi sekitar 40 cm terhadap dasar tangki. Oleh karena itu, posisi pemasangan sensor dan rentang level pengisian perlu diperhatikan agar permukaan air tetap berada dalam rentang pembacaan sensor yang dapat digunakan.

Tabel 2.7 Spesifikasi Sensor Ultrasonik JSN-SR04T

Parameter	Spesifikasi
Tegangan kerja	5 V DC
Arus Kerja	30mA
Jarak Ukur	20 – 600 cm
Output	Digital (Trig & Echo)
Frekuensi Ultrasonik	40 kHz

2.13. *Pulse Width Modulation (PWM)*

Pulse Width Modulation (PWM) merupakan teknik pengendalian dengan mengubah lebar pulsa sinyal digital tanpa mengubah amplitudo logikanya. Besarnya *duty cycle* menunjukkan perbandingan waktu kondisi *HIGH* terhadap satu periode sinyal. Semakin besar *duty cycle*, semakin besar nilai rata-rata sinyal yang diteruskan. Sebaliknya, semakin kecil *duty cycle*, semakin kecil nilai rata-rata sinyal.

Secara matematis, nilai *duty cycle* dapat dinyatakan menggunakan persamaan berikut.

$$D = \frac{T_{ON}}{T} \times 100\% \quad (2.25)$$

Keterangan:

D = *Duty Cycle* (%)

T_{ON} = waktu sinyal berada pada kondisi HIGH (s)

T = periode satu siklus sinyal PWM (s)

Selain *duty cycle*, parameter penting lainnya pada sinyal PWM adalah frekuensi. Frekuensi menunjukkan jumlah siklus sinyal yang dihasilkan setiap detik dan dinyatakan dalam satuan *Hertz* (Hz). Pemilihan frekuensi PWM harus disesuaikan dengan karakteristik aktuator yang digunakan agar sistem dapat bekerja secara stabil dan menghasilkan respon yang optimal. Pada sistem berbasis mikrokontroler, sinyal PWM umumnya dihasilkan menggunakan modul *timer* internal sehingga frekuensi maupun *duty cycle* dapat diatur secara fleksibel sesuai kebutuhan aplikasi.

Pada penelitian ini, sinyal PWM digunakan pada dua fungsi, yaitu sebagai media representasi nilai pada komunikasi pin-to-pin antar ESP32 dan sebagai sinyal pengaturan pompa. Keluaran CTRL_PID pada ESP32 OpenPLC dikirim menuju ESP32 pendukung melalui jalur PWM-RC-ADC, kemudian ESP32 pendukung menghasilkan Motor PWM yang diterapkan pada driver BTS7960 untuk mengatur kecepatan pompa.

2.14. Driver Motor BTS7960

Driver Motor BTS7960 merupakan modul pengendali untuk motor DC. Modul ini dirancang untuk mengendalikan motor DC melalui konfigurasi *H-Bridge*, sehingga arah putaran maupun kecepatan motor dapat diukur menggunakan sinyal logika dan PWM. Pada modul BTS7960 tersedia dua buah masukan PWM, yaitu RPWM (*Right PWM*) dan LPWM (*Left PWM*). Kedua masukan tersebut digunakan untuk menentukan arah putaran motor sekaligus mengatur besar kecilnya kecepatan putaran melalui perubahan nilai *duty cycle*. Apabila hanya pin RPWM yang diberikan sinyal PWM sedangkan LPWM berada pada kondisi logika rendah, maka motor akan berputar pada satu arah. Sebaliknya, apabila sinyal PWM diberikan pada LPWM dan RPWM berada pada kondisi rendah, maka motor akan berputar ke arah sebaliknya. Dengan prinsip tersebut, BTS7960 mampu mengendalikan motor secara fleksibel sesuai kebutuhan sistem pengendalian.

Pada penelitian ini, driver BTS7960 digunakan sebagai antarmuka antara ESP32 pendukung dan pompa DC 12 V. Sinyal Motor PWM diberikan pada kanal RPWM, sedangkan LPWM dipertahankan pada kondisi rendah karena pompa dioperasikan pada satu arah. Besar *duty cycle* Motor PWM menentukan tingkat aktuasi pompa dan memengaruhi laju aliran air.

2.15. Komunikasi Antar ESP32 Menggunakan Pin-to-Pin

Komunikasi antar mikrokontroler merupakan proses pertukaran informasi antara dua atau lebih perangkat untuk mendukung sistem kendali yang terintegrasi. Selain menggunakan protokol seperti UART, I²C, dan SPI, pertukaran sinyal dapat dilakukan langsung melalui GPIO atau pin-to-pin. Pada penelitian ini, nilai numerik tidak dikirim hanya sebagai status HIGH/LOW, tetapi direpresentasikan oleh *duty cycle* PWM. Sinyal PWM kemudian diratakan menggunakan RC low-pass filter dan dibaca kembali sebagai tegangan analog oleh ADC pada ESP32 penerima.

Dibandingkan dengan komunikasi serial, metode *pin-to-pin* memiliki keunggulan berupa implementasi yang lebih sederhana, tidak memerlukan konfigurasi protokol komunikasi, serta memiliki latensi yang rendah untuk pertukaran sinyal digital. Namun demikian, metode ini memiliki keterbatasan dalam jumlah data yang dapat dikirimkan karena setiap sinyal umumnya

memerlukan satu jalur GPIO tersendiri. Oleh karena itu, komunikasi *pin-to-pin* lebih sesuai digunakan pada sistem yang hanya memerlukan pertukaran beberapa sinyal kontrol atau status antarmikrokontroler.

Pada penelitian ini digunakan dua ESP32 dengan tugas berbeda. ESP32 OpenPLC menjalankan OpenPLC *Runtime* dan *Function Block* CTRL_PID, sedangkan ESP32 pendukung menangani akuisisi sensor serta penerapan keluaran kendali pada perangkat keras. Komunikasi berlangsung dua arah. Pada jalur feedback, data YF-S201 diproses oleh ESP32 pendukung, direpresentasikan sebagai duty cycle PWM, diratakan menggunakan RC low-pass filter, kemudian dibaca oleh ADC ESP32 OpenPLC. Pada jalur kendali, keluaran CTRL_PID dikirim dari ESP32 OpenPLC melalui mekanisme PWM–RC–ADC menuju ESP32 pendukung sebelum diterapkan pada BTS7960 dan pompa.

2.16. RC Low-Pass Filter

RC *Low-Pass Filter* merupakan rangkaian elektronika pasif yang tersusun atas sebuah resistor (R) dan kapasitor (C) yang berfungsi untuk melewatkan sinyal berfrekuensi rendah serta meredam komponen sinyal berfrekuensi tinggi. Rangkaian ini banyak digunakan pada sistem instrumentasi dan kendali sebagai penyaring (*filter*) untuk mengurangi gangguan (*noise*) pada sinyal analog sehingga data yang diterima oleh sistem menjadi lebih stabil. Prinsip kerja RC *Low-Pass Filter* didasarkan pada karakteristik kapasitor yang memiliki reaktansi tinggi terhadap sinyal berfrekuensi rendah dan reaktansi rendah terhadap sinyal berfrekuensi tinggi, sehingga komponen frekuensi tinggi akan diredam sebelum mencapai rangkaian berikutnya. Karakteristik RC *Low-Pass Filter* ditentukan oleh konstanta waktu (*time constant*) yang dinyatakan sebagai

$$\tau = RC \quad (2.26)$$

dengan:

τ : Konstanta waktu

R: Resistansi

C: Kapasitansi

Selain konstanta waktu, parameter penting lainnya adalah frekuensi potong (*cut-off frequency*), yaitu batas frekuensi ketika amplitudo sinyal keluaran turun menjadi sekitar 70,7% dari amplitudo sinyal masukan. Nilai frekuensi potong dirumuskan sebagai

$$f_c = \frac{1}{2\pi RC} \quad (2.27)$$

Dengan:

F_c: Frekuensi potong

R: resistansi

C: Kapasitansi

Semakin besar nilai resistor maupun kapasitor yang digunakan, maka nilai frekuensi potong akan semakin kecil sehingga kemampuan filter dalam meredam gangguan berfrekuensi tinggi menjadi semakin baik. Sebaliknya, apabila nilai resistor atau kapasitor diperkecil, maka frekuensi potong akan meningkat sehingga lebih banyak komponen frekuensi yang dapat dilewatkan.

Pada penelitian ini, RC low-pass filter digunakan pada kedua jalur komunikasi PWM antarmikrokontroler untuk mengubah duty cycle menjadi tegangan rata-rata yang dapat dibaca oleh ADC ESP32 penerima. Pemilihan nilai resistor dan kapasitor serta perhitungan konstanta waktu dan frekuensi potong untuk rangkaian aktual dijelaskan pada Bab III.

2.17. Bahasa Pemrograman PLC Berdasarkan IEC 61131-3

Bahasa pemrograman PLC merupakan metode yang digunakan untuk menyusun logika pengendalian pada sistem otomasi. Berdasarkan standar IEC 61131-3, terdapat beberapa bahasa pemrograman yang dapat digunakan, seperti *Ladder Diagram* (LD), *Function Block Diagram* (FBD), *Structured Text* (ST), dan *Sequential Function Chart* (SFC). Pada penelitian ini bahasa yang digunakan adalah *Function Block Diagram* karena lebih sesuai untuk implementasi algoritma PID dan pengolahan data numerik dibandingkan *Ladder Diagram*.

2.18. Penerapan Function Block Diagram pada Penelitian

Pada penelitian ini, *Function Block Diagram* (FBD) digunakan sebagai bahasa pemrograman utama untuk menghubungkan proses pembacaan dan penskalaan sensor, filtering, logika interlock, serta Function Block CTRL_PID. Penjelasan umum mengenai FBD pada lingkungan OpenPLC telah diberikan pada Subbab 2.8.5.

Struktur FBD memudahkan penelusuran aliran data dari process variable menuju pengendali dan dari keluaran pengendali menuju aktuator karena setiap operasi direpresentasikan sebagai blok yang saling terhubung.

Blok yang digunakan pada bagian laju aliran antara lain DIV, TO_REAL, MUL, ADD, MOVE, GE, dan CTRL_PID. Rangkaian blok tersebut diimplementasikan pada ESP32 yang menjalankan OpenPLC Runtime.

2.19. Tipe Data PLC

Tipe data merupakan klasifikasi data yang menentukan jenis nilai yang dapat disimpan oleh suatu variabel pada PLC. Pemilihan tipe data yang sesuai akan mempengaruhi penggunaan memori, kecepatan pemrosesan, serta ketelitian hasil perhitungan. Beberapa tipe data yang umum digunakan pada OpenPLC meliputi sebagai berikut:

a. BOOL

Bool digunakan untuk menyimpan data logika dengan dua kondisi, yaitu *TRUE* dan *FALSE*.

b. INT

Int digunakan untuk menyimpan bilangan bulat bertanda dengan rentang -32.768 hingga 32.767.

c. UINT

Uint digunakan untuk menyimpan bilangan bulat tanpa tanda dengan rentang 0 hingga 65.535

d. REAL

Real digunakan untuk menyimpan bilangan pecahan (*floating point*) sehingga sesuai untuk data hasil pengukuran sensor maupun perhitungan algoritma PID.

Dalam penelitian ini, tipe data Bool digunakan pada status digital seperti pompa dan relay, Int digunakan pada data pembacaan sensor yang masih berupa bilangan bulat, sedangkan Real digunakan pada proses perhitungan PID karena memerlukan tingkat presisi yang lebih tinggi.

2.20. Addressing PLC

Addressing PLC merupakan metode pemberian alamat memori pada setiap variabel yang digunakan dalam program. Alamat tersebut digunakan sebagai media komunikasi antara perangkat keras dengan program PLC sehingga data dari sensor maupun aktuator dapat diakses secara langsung selama proses scan cycle berlangsung. Pada OpenPLC, beberapa jenis alamat memori yang umum digunakan meliputi:

- a. %IX sebagai alamat *Digital Input*
- b. %QX sebagai alamat *Digital Output*
- c. %IW sebagai alamat *Input Word*, yang umumnya digunakan untuk data analog atau data numerik dari sensor.
- d. %QW sebagai alamat *Output Word*, yang digunakan untuk mengirimkan data numerik ke aktuator.
- e. %MW sebagai *Memory Word*, yaitu memori internal PLC untuk menyimpan hasil perhitungan sementara.

%MD sebagai *Memory Double Word*, yang pada penelitian ini digunakan untuk menyimpan variabel numerik tipe REAL seperti process variable, setpoint, dan parameter pengendalian.

Pemilihan alamat memori yang tepat mempermudah pengembangan program, komunikasi dengan perangkat keras, serta proses monitoring dan debugging selama sistem beroperasi.

2.21. Parameter Evaluasi Respons Sistem

Evaluasi respons sistem dilakukan untuk mengetahui karakteristik dan performa sistem pengendalian berdasarkan data hasil pengujian. Parameter evaluasi yang digunakan meliputi nilai rata-rata *process variable*, *rise time*, *maximum overshoot*, *settling time*, *steady-state error*, *Mean Absolute Error* (MAE), *Root*

Mean Square Error (RMSE), standar deviasi, koefisien variasi, dan persentase data yang berada dalam pita toleransi. Persamaan pada subbab ini digunakan sebagai dasar pengolahan data pada Bab III, sedangkan hasil perhitungannya disajikan pada Bab IV.

a. Nilai Rata-Rata Process Variable

Nilai rata-rata *process variable* digunakan untuk mengetahui nilai representatif respons sistem pada kondisi *steady-state*. Nilai tersebut dihitung menggunakan:

$$\overline{PV} = \frac{1}{N} \sum_{i=1}^N PV_i$$

Dengan:

$\overline{PV}$: nilai rata-rata *process variable*

PV_i : nilai *process variable* pada sampel ke- i

N : jumlah data yang dianalisis.

b. Rise Time

Rise time merupakan waktu yang diperlukan respons sistem untuk meningkat dari 10% hingga 90% dari nilai acuan. *Rise time* dihitung menggunakan:

$$t_r = t_{90} - t_{10}$$

Dengan:

t_r : rise time

t_{10} : waktu ketika respons pertama kali mencapai 10% nilai acuan

t_{90} : waktu ketika respons pertama kali mencapai 90% nilai acuan.

Pada pengujian *closed-loop*, nilai acuan yang digunakan adalah nilai *setpoint*.

c. Maximum Overshoot Open-Loop

Pada pengujian *open-loop*, *maximum overshoot* menunjukkan besarnya nilai puncak respons terhadap nilai kondisi tunak. Persentase *maximum overshoot* dihitung menggunakan:

$$M_p = \frac{Q_{\text{peak}} - Q_{ss}}{Q_{ss}} \times 100\%$$

d. *Maximum Overshoot Closed-Loop*

Pada pengujian *closed-loop*, nilai *maximum overshoot* dihitung terhadap nilai *setpoint*. Persamaan yang digunakan adalah:

$$M_p = \frac{PV_{\text{peak}} - SP}{SP} \times 100\%$$

e. *Settling Time*

Settling time merupakan waktu yang diperlukan respons sistem untuk masuk ke dalam pita toleransi tertentu dan tetap berada di dalam pita tersebut hingga akhir interval analisis.

Batas atas pita toleransi dapat dinyatakan menggunakan:

$$PV_{\text{upper}} = SP(1 + \delta)$$

sedangkan batas bawah dinyatakan menggunakan:

$$PV_{\text{lower}} = SP(1 - \delta)$$

f. *Steady-State Error*

Steady-state error menunjukkan selisih antara nilai *setpoint* dengan rata-rata *process variable* pada kondisi tunak. Nilai error dihitung menggunakan:

$$e_{ss} = SP - \overline{PV}$$

persentase *steady-state error* terhadap *setpoint* dihitung menggunakan:

$$e_{ss}(\%) = \frac{|SP - \overline{PV}|}{SP} \times 100\%$$

Persamaan persentase tersebut lebih mudah digunakan dalam laporan karena dapat membandingkan error pada beberapa nilai *setpoint* yang berbeda.

g. *Mean Absolute Error (MAE)*

Mean Absolute Error digunakan untuk mengetahui rata-rata besar penyimpangan *process variable* terhadap *setpoint* tanpa memperhatikan arah error. MAE dihitung menggunakan:

$$MAE = \frac{1}{N} \sum_{i=1}^N |PV_i - SP|$$

Semakin kecil nilai MAE, semakin dekat nilai *process variable* terhadap nilai *setpoint*.

h. *Root Mean Square Error* (RMSE)

Root Mean Square Error memberikan bobot yang lebih besar terhadap error yang memiliki nilai tinggi karena error dikuadratkan sebelum dirata-ratakan. RMSE dihitung menggunakan:

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^N (PV_i - SP)^2}$$

Nilai RMSE yang semakin kecil menunjukkan bahwa penyimpangan respons terhadap *setpoint* semakin kecil.