



SKRIPSI

**KENDALI KECEPATAN PADA MOTOR DC
CONVEYOR MENGGUNAKAN METODE PID
ZIEGLER-NICHOLS BERBASIS IOT**

MOCH HAWIN HAMAMI

NPM 18081010038

DOSEN PEMBIMBING

Dr. Basuki Rahmat, S.Si., M.T.

Henni Endah Wahanani, S.T., M.Kom.

**KEMENTERIAN PENDIDIKAN TINGGI, SAINS, DAN TEKNOLOGI
UNIVERSITAS PEMBANGUNAN NASIONAL VETERAN JAWA TIMUR
FAKULTAS ILMU KOMPUTER
PROGRAM STUDI INFORMATIKA
SURABAYA
2025**

Halaman ini sengaja dikosongkan



SKRIPSI

KENDALI KECEPATAN PADA MOTOR DC CONVEYOR MENGGUNAKAN METODE PID ZIEGLER-NICHOLS BERBASIS IOT

MOCH HAWIN HAMAMI

NPM 18081010038

DOSEN PEMBIMBING

Dr. Basuki Rahmat, S.Si., M.T.

Henni Endah Wahanani, S.T., M.Kom.

**KEMENTERIAN PENDIDIKAN TINGGI, SAINS, DAN TEKNOLOGI
UNIVERSITAS PEMBANGUNAN NASIONAL VETERAN JAWA TIMUR
FAKULTAS ILMU KOMPUTER
PROGRAM STUDI INFORMATIKA
SURABAYA
2025**

Halaman ini sengaja dikosongkan

LEMBAR PENGESAHAN

KENDALI KECEPATAN PADA MOTOR DC CONVEYOR MENGGUNAKAN METODE PID ZIEGLER-NICHOLS BERBASIS IOT

Oleh:

MOCH HAWIN HAMAMI

NPM. 18081010038

Telah dipertahankan di hadapan dan diterima oleh Tim Pengaji Skripsi Prodi Informatika Fakultas Ilmu Komputer Universitas Pembangunan Nasional Veteran Jawa Timur Pada hari Jum'at, tanggal 16 Mei 2025.

Menyetujui

Dr. Basuki Rahmat, S.Si., M.T.

NIP. 19690723 202121 1 002

(Pembimbing I)

Henni Endah Wahanani, S.T., M.Kom.

NIP. 19780922 202121 2 005

(Pembimbing II)

Fawwaz Ali Akbar, S.Kom., M.Kom.

NIP. 19920317 201803 1 002

(Ketua Pengaji)

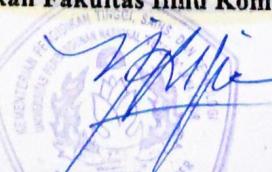
Firza Prima Aditiawan., S.Kom., M.TI.

NIP. 19860523 202121 1 003

(Anggota Pengaji)

Mengetahui,

Dekan Fakultas Ilmu Komputer



Prof. Dr. Ir. Novirina Hendrasarie, M.T.

NIP. 19681126 199403 2 001

Halaman ini sengaja dikosongkan

LEMBAR PERSETUJUAN

KENDALI KECEPATAN PADA MOTOR DC **CONVEYOR** MENGGUNAKAN METODE
PID ZIEGLER-NICHOLS BERBASIS IOT

Oleh:

MOCH HAWIN HAMAMI

NPM. 18081010038



Menyetujui,

Koordinator Program Studi Informatika

Fakultas Ilmu Komputer

A handwritten signature in black ink, appearing to read "Fetty Tri Anggraeny", is placed over the text above it.

Fetty Tri Anggraeny, S.Kom., M.Kom.

NIP. 19820211 202121 2 005

Halaman ini sengaja dikosongkan

SURAT PERNYATAAN BEBAS PLAGIASI

Saya yang bertanda tangan dibawah ini :

Nama Mahasiswa : Moch Hawin Hamami
NPM : 18081010038
Program : Sarjana (S1)
Program Studi : Informatika
Fakultas : Ilmu Komputer

Menyatakan bahwa dalam dokumen ilmiah Tugas Skripsi ini tidak terdapat bagian dari karya ilmiah lain yang telah diajukan untuk memperoleh gelar akademik di suatu lembaga Pendidikan Tinggi, dan juga tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang/lembaga lain, kecuali yang secara tertulis disitasi dalam dokumen ini dan disebutkan secara lengkap dalam daftar pustaka.

Dan saya menyatakan bahwa dokumen ilmiah ini bebas dari unsur-unsur plagiasi. Apabila dikemudian hari ditemukan indikasi plagiat pada Skripsi ini, saya bersedia menerima sanksi sesuai dengan peraturan perundang-undangan yang berlaku.

Demikian surat pernyataan ini saya buat dengan sesungguhnya tanpa ada paksaan dari siapapun juga dan untuk dipergunakan sebagaimana mestinya.



Surabaya, 16 Juni 2025

Yang Membuat Pernyataan



Moch Hawin Hamami

NPM. 18081010038

Halaman ini sengaja dikosongkan

ABSTRAK

Nama Mahasiswa / NPM	:	Moch Hawin Hamami / 18081010038.
Judul Skripsi	:	Kendali Kecepatan Pada Motor DC <i>Conveyor</i> Menggunakan Metode PID <i>Ziegler-Nichols</i> Berbasis IoT.
Dosen Pembimbing	:	1. Dr. Basuki Rahmat, S.Si., M.T. 2. Henni Endah Wahanani, S.T., M.Kom.

Motor DC pada sistem *Conveyor* industri kerap mengalami penurunan kecepatan (RPM) akibat beban berlebih, yang berdampak pada ketidakstabilan proses produksi. Untuk mengatasi permasalahan tersebut, penelitian ini mengembangkan sistem kendali kecepatan *Conveyor* berbasis *Internet of Things* (IoT) dengan menerapkan metode kendali PID yang di *tuning* menggunakan pendekatan *Ziegler-Nichols* Tipe 2. Mikrokontroler ESP32 digunakan sebagai unit kendali utama untuk membaca data dari *Encoder* dan mengatur kecepatan motor DC melalui *Motor Driver* L298N. Sistem ini terintegrasi dengan platform Ubidots menggunakan protokol komunikasi MQTT, sehingga memungkinkan pemantauan kecepatan (RPM) serta kendali *On / Off* dan arah rotasi motor (maju / mundur) secara *real-time* melalui koneksi internet. Selain itu, sistem ini juga dapat dioperasikan secara *offline*. Meskipun pemanfaatan IoT memberikan kemudahan yang signifikan, sistem masih memiliki keterbatasan, salah satunya adalah penggunaan potensiometer fisik untuk mengatur kecepatan (RPM), yang masih memerlukan intervensi langsung dari operator di lokasi. Penelitian ini menggunakan rasio *gear* yang tidak terlalu besar untuk memastikan daya angkat *Conveyor* tetap tercapai, sekaligus menjaga kecepatan (RPM) agar tidak terlalu rendah. Sistem ini dikembangkan menggunakan metodologi *Rapid Application Development* (RAD) dan diuji melalui pendekatan eksperimental. Hasil pengujian menunjukkan bahwa sistem berhasil mengimplementasikan fungsi kendali *On / Off* dan juga kendali arah rotasi pada motor DC, baik dalam jarak dekat maupun jarak jauh. Metode PID *Ziegler-Nichols* Tipe 2 dengan parameter *tuning* akhir $K_p=7.2$, $K_i=48$, dan $K_d=0,27$ mampu menjaga kecepatan motor DC tetap stabil mendekati nilai *Setpoint* meskipun terjadi perubahan beban. Kedepannya, sistem ini diharapkan dapat dikembangkan lebih lanjut melalui integrasi *multi-platform* serta fitur IoT yang lebih canggih, responsif, dan adaptif terhadap kebutuhan industri modern.

Kata kunci : ESP32, PID, *Ziegler-Nichols*, RPM, Ubidots, MQTT.

Halaman ini sengaja dikosongkan

ABSTRACT

<i>Student Name / NPM</i>	: <i>Moch Hawin Hamami / 18081010038.</i>
<i>Thesis Title</i>	: <i>Speed Control on DC Conveyor Motor Using IoT-Based PID Ziegler-Nichols Method.</i>
<i>Advisors</i>	: 1. <i>Dr. Basuki Rahmat, S.Si., M.T.</i> 2. <i>Henni Endah Wahanani, S.T., M.Kom.</i>

DC motors in industrial conveyor systems often experience a decrease in speed (RPM) due to overload, which has an impact on the instability of the production process. To overcome these problems, this research develops an Internet of Things (IoT)-based Conveyor speed control system by applying a PID control method tuned using the Ziegler-Nichols Type 2 approach. The ESP32 microcontroller is used as the main control unit to read data from the Encoder and regulate the speed of the DC motor through the L298N Motor Driver. The system is integrated with Ubidots platform using MQTT communication protocol, thus enabling speed monitoring (RPM) as well as On / Off control and motor rotation direction (forward / backward) in real-time via internet connection. In addition, the system can also be operated offline. Although the utilization of IoT provides significant convenience, the system still has limitations, one of which is the use of a physical potentiometer to adjust the speed (RPM), which still requires direct intervention from the operator on site. This research uses a gear ratio that is not too large to ensure the conveyor's carrying capacity is still achieved, while keeping the speed (RPM) from being too low. The system was developed using Rapid Application Development (RAD) methodology and tested through an experimental approach. The test results show that the system successfully implements the On / Off control function and also controls the direction of rotation on the DC motor, both at close range and at long range. The Ziegler-Nichols Type 2 PID method with final tuning parameters $K_p=7.2$, $K_i=48$, dan $K_d=0.27$ can keep the DC motor speed stable near the Setpoint value despite load changes. In the future, this system is expected to be further developed through multi-platform integration and the addition of more sophisticated, responsive, and adaptive IoT features that cater to the needs of modern industry.

Keywords : ESP32, PID, Ziegler-Nichols, RPM, Ubidots, MQTT.

Halaman ini sengaja dikosongkan

KATA PENGANTAR

Puji syukur ke hadirat Allah SWT atas segala rahmat, hidayah dan karunia-Nya kepada penulis sehingga skripsi dengan judul **“Kendali Kecepatan Pada Motor DC Conveyor Menggunakan Metode PID Ziegler-Nichols Berbasis IoT”** dapat terselesaikan dengan baik.

Penulis mengucapkan terima kasih kepada Bapak Dr. Basuki Rahmat, S.Si., M.T., selaku Dosen Pembimbing utama yang telah meluangkan waktunya untuk memberikan bimbingan, nasehat serta motivasi kepada penulis. Dan penulis juga banyak menerima bantuan dari berbagai pihak, baik itu berupa moril, spiritual maupun materiil. Untuk itu penulis mengucapkan terima kasih kepada:

1. Bapak Prof. Dr. Ir. Ahmad Fauzi, M.MT., IPU., selaku Rektor Universitas Pembangunan Nasional “Veteran” Jawa Timur.
2. Ibu Prof. Dr. Ir. Novirina Hendrasarie, M.T., selaku Dekan Fakultas Ilmu Komputer Universitas Pembangunan Nasional “Veteran” Jawa Timur.
3. Ibu Fetty Tri Anggraeny, S.Kom., M.Kom selaku Koordinator Program Studi Informatika Fakultas Ilmu Komputer Universitas Pembangunan Nasional “Veteran” Jawa Timur.
4. Ibu Henni Endah Wahanani, S.T., M.Kom., selaku Dosen Pembimbing Kedua yang selalu memberikan dukungan berupa koreksi dan saran dalam proses penyelesaian skripsi ini.
5. Dosen-dosen Program Studi Informatika Fakultas Ilmu Komputer Universitas Pembangunan Nasional “Veteran” Jawa Timur.
6. Terima kasih yang tak terhingga kepada keluarga, ayah, ibu dan juga adik, yang telah memberikan semangat, doa, dan segala dukungannya hingga saat ini, dan tidak pernah lelah memberikan motivasi kepada penulis.
7. Rekan satu Angkatan 2018, Devan Cakra Mudra Wijaya, yang mau menyempatkan waktu di samping kesibukan untuk membantu penulis dalam bertukar pikiran terkait pengembangan tugas akhir ini.
8. Seluruh teman senasib seperjuangan, kakak tingkat, serta adik tingkat yang selama ini tetap saling membantu di samping kesibukan.
9. Seluruh pihak yang tidak bisa disebutkan satu per satu.

Penulis menyadari bahwa di dalam penyusunan skripsi ini banyak terdapat kekurangan. Untuk itu kritik dan saran yang membangun dari semua pihak sangat diharapkan demi kesempurnaan penulisan skripsi ini. Akhirnya, dengan segala keterbatasan yang penulis miliki semoga laporan ini dapat bermanfaat bagi semua pihak umumnya dan penulis pada khususnya.

Surabaya, 16 Juni 2025

Moch Hawin Hamami

DAFTAR ISI

LEMBAR JUDUL SKRIPSI	i
LEMBAR PENGESAHAN.....	ii
LEMBAR PERSETUJUAN	iii
SURAT PERNYATAAN BEBAS PLAGIASI	iv
ABSTRAK.....	v
<i>ABSTRACT</i>	vi
KATA PENGANTAR	vii
DAFTAR ISI	ix
DAFTAR TABEL.....	xii
DAFTAR GAMBAR.....	xiii
DAFTAR PERSAMAAN	xvi
DAFTAR <i>SOURCE CODE</i>	xvii
BAB I PENDAHULUAN	1
1.1. Latar Belakang	1
1.2. Rumusan Masalah.....	3
1.3. Tujuan.....	3
1.4. Batasan Masalah.....	4
1.5. Manfaat	4
BAB II TINJAUAN PUSTAKA	5
2.1. Penelitian Terdahulu	5
2.2. ESP32 DOIT DevKit-C V1	6
2.3. Motor DC JGA25-370 12V <i>Encoder</i>	8
2.3.1. Prinsip Kerja Motor DC	9
2.3.2. Elemen - Elemen Motor DC	9
2.4. <i>Motor Driver L298N</i>	10

2.5. LCD (<i>Liquid Crystal Display</i>) I2C	11
2.6. Potensiometer.....	12
2.7. <i>Push Button</i>	14
2.8. <i>Breadboard</i>	14
2.9. <i>Digital Tachometer</i>	15
2.10. Kendali PID (<i>Proportional-Integral-Derivative</i>).....	16
2.10.1. Sejarah Kendali PID	16
2.10.2. Komponen Utama dalam Kendali PID.....	16
2.10.3. Prinsip Kerja Kendali PID	17
2.10.4. Karakteristik Sistem Kendali PID	17
2.10.5. Metode <i>Tuning: Ziegler-Nichols</i>	18
2.10.6. Pembagian Metode <i>Tuning Ziegler-Nichols</i>	18
2.11. PWM (<i>Pulse Width Modulation</i>).....	22
2.12. IoT (<i>Internet of Things</i>).....	22
2.13. IoT <i>Platform</i> : Ubidots	24
2.14. Diagram Blok.....	25
2.15. Diagram <i>Wiring</i>	26
BAB III METODOLOGI PENELITIAN	27
3.1. Prasyarat (<i>Requirement</i>)	27
3.1.1. Studi Literatur	28
3.1.2. Analisis Masalah	28
3.1.3. Analisis Solusi	28
3.1.4. Analisis Kebutuhan	28
3.1.5. Analisis Sistem	31
3.2. Perancangan Prototipe (<i>Prototyping</i>).....	31
3.2.1. Skema Perancangan <i>Software</i>	31
3.2.2. Skema Perancangan <i>Hardware</i>	33
3.2.3. Skema Kalibrasi CPR	40

3.2.4. Skema Implementasi PID <i>Ziegler-Nichols</i>	42
3.3. Hasil (<i>Output</i>).....	45
3.4. Pengujian (<i>Testing</i>)	45
BAB IV HASIL DAN PEMBAHASAN	47
4.1. Hasil Penelitian	47
4.1.1. Implementasi Rancangan <i>Hardware</i>	47
4.1.2. Implementasi Rancangan <i>Software</i>	47
4.2. Hasil Pengujian.....	73
4.2.1. Pengujian Konektivitas <i>Wi-Fi</i>	73
4.2.2. Pengujian Konektivitas IoT	75
4.2.3. Pengujian <i>Publish–Subscribe</i> pada MQTT	76
4.2.4. Pengujian LCD I2C	77
4.2.5. Pengujian Kecepatan Putaran (RPM) Motor DC pada Platform Ubidots	79
4.2.6. Pengujian Kecepatan Putaran (RPM) Motor DC pada Arduino IDE	80
4.2.7. Pengujian Kecepatan Putaran (RPM) Motor DC dengan <i>Digital Tachometer</i> ..	81
4.2.8. Pengujian Tombol Fisik dan Non-Fisik	82
4.2.9. Pengujian Efek Bobot Barang terhadap Waktu Tempuh <i>Conveyor</i>	83
BAB V KESIMPULAN DAN SARAN.....	85
5.1. Kesimpulan	85
5.2. Saran.....	86
LAMPIRAN	87
DAFTAR PUSTAKA	89

Halaman ini sengaja dikosongkan

DAFTAR TABEL

Tabel 2. 1. Formula <i>Ziegler-Nichols</i> Metode 1.....	20
Tabel 2. 2. Formula <i>Ziegler-Nichols</i> Metode 2.....	21
Tabel 3. 1. Kebutuhan Perangkat Keras	29
Tabel 3. 2. Kebutuhan Perangkat Lunak.....	29
Tabel 3. 3. Kebutuhan Alat dan Bahan	30
Tabel 3. 4. Skenario Otomatisasi Sistem	31
Tabel 3. 5. Skenario Kendali Sistem.....	32
Tabel 3. 6. Perancangan <i>Conveyor</i>	34
Tabel 3. 7. Detail Pengkabelan <i>Hardware</i>	39
Tabel 3. 8. Langkah – Langkah Menentukan <i>Ku</i> dan <i>Pu</i>	43
Tabel 3. 9. Langkah – Langkah Penyelesaian Studi Kasus	44
Tabel 3. 10. Formula Diterapkan Pada Studi Kasus	44
Tabel 3. 11. Hasil Yang Diharapkan	45
Tabel 3. 12. Skenario Uji Coba (<i>by manual</i>)	45
Tabel 3. 13. Skenario Uji Coba (<i>by system</i>)	46
Tabel 4. 1. <i>Monitoring</i> Kecepatan Putaran (RPM) Motor DC pada Platform Ubidots	79
Tabel 4. 2. Hasil Pengukuran <i>Digital Tachometer</i>	82
Tabel 4. 3. Hasil Pengujian Tombol.....	83
Tabel 4. 4. Hasil Pengujian Waktu Tempuh Barang	84

Halaman ini sengaja dikosongkan

DAFTAR GAMBAR

Gambar 2. 1. ESP32 DOIT DevKit-C V1	7
Gambar 2. 2. Motor DC JGA25-370 12V <i>Encoder Pinout</i>	8
Gambar 2. 3. Elemen Motor DC.....	9
Gambar 2. 4. Modul <i>Motor Driver</i> L298N.....	11
Gambar 2. 5. LCD I2C	12
Gambar 2. 6. Potensiometer	13
Gambar 2. 7. <i>Push Button</i>	14
Gambar 2. 8. <i>Medium Breadboard</i>	15
Gambar 2. 9. <i>Digital Tachometer</i>	15
Gambar 2. 10. Kurva Respons Dengan <i>Overshoot</i> Maksimum 25%	18
Gambar 2. 11. Respons Sistem Terhadap Masukan Fungsi <i>Step</i>	19
Gambar 2. 12. Respons Sistem Berbentuk Kurva S	19
Gambar 2. 13. Sistem <i>Loop</i> Tertutup dengan Pengendali Proporsional	21
Gambar 2. 14. Manipulasi PWM.....	22
Gambar 2. 15. Pengiriman Data MQTT	23
Gambar 2. 16. Tampilan IoT <i>Platform</i> : Ubidots	24
Gambar 3. 1. Tahapan Penelitian.....	27
Gambar 3. 2. <i>Workflow</i> Sistem	32
Gambar 3. 3. Perancangan <i>Conveyor</i> Tampak Samping	34
Gambar 3. 4. Perancangan <i>Conveyor</i> Tampak Depan	35
Gambar 3. 5. <i>Blueprint</i> Kerangka Bagian Samping dan <i>Belt Conveyor</i>	36
Gambar 3. 6. <i>Blueprint</i> Kerangka Bagian Depan dan <i>Roller Conveyor</i>	36
Gambar 3. 7. <i>Blueprint</i> Papan Panel.....	37
Gambar 3. 8. Diagram Blok Perancangan <i>Hardware</i>	37
Gambar 3. 9. Diagram <i>Wiring</i> Perancangan <i>Hardware</i>	38
Gambar 3. 10. <i>Flowchart</i> Implementasi PID	42
Gambar 4. 1. Rancangan <i>Hardware</i> keseluruhan	47
Gambar 4. 2. <i>Download</i> CP210X <i>Driver</i>	48
Gambar 4. 3. Instalasi CP210X <i>Driver</i>	48
Gambar 4. 4. <i>Additional Board Manager URL</i> ESP32	49
Gambar 4. 5. Instalasi ESP32 <i>Board</i>	49
Gambar 4. 6. Pengaturan ESP32 <i>Board</i>	50

Gambar 4. 7. Pengaturan <i>Port</i> (COM) ESP32 di Arduino IDE.....	50
Gambar 4. 8. Mengecek <i>Port</i> (COM) ESP32 di <i>Device Manager</i>	51
Gambar 4. 9. Instalasi <i>Library</i> di <i>Library Manager</i>	51
Gambar 4. 10. Instalasi <i>Library</i> Secara Manual	52
Gambar 4. 11. <i>Library</i> Berhasil Ditambahkan pada Arduino IDE	52
Gambar 4. 12. Kalibrasi CPR Secara Manual	53
Gambar 4. 13. <i>Login</i> Ubidots	65
Gambar 4. 14. Menu <i>User</i> Ubidots	66
Gambar 4. 15. Token Ubidots.....	66
Gambar 4. 16. Menambah Dasbor Baru pada Ubidots.....	66
Gambar 4. 17. Mengganti Beberapa Opsi pada Menu <i>Dashboards</i>	67
Gambar 4. 18. Pilih Perangkat.....	67
Gambar 4. 19. Menambahkan <i>Widget</i> Baru pada Ubidots	68
Gambar 4. 20. Pilihan <i>Widget</i> pada Ubidots.....	68
Gambar 4. 21. Opsi pada <i>Widget</i>	68
Gambar 4. 22. Memilih Variabel	69
Gambar 4. 23. Tampilan <i>Widget</i> : <i>Line Chart</i>	69
Gambar 4. 24. Hasil Pembuatan Variabel.....	70
Gambar 4. 25. Memilih Variabel pada <i>Widget</i> : <i>Switch 1</i>	70
Gambar 4. 26. Hasil <i>Duplicate</i>	71
Gambar 4. 27. Memilih Variabel pada <i>Widget</i> : <i>Switch 2</i>	71
Gambar 4. 28. Memilih Variabel pada <i>Widget</i> : Indikator 1.....	72
Gambar 4. 29. Memilih Variabel pada <i>Widget</i> : Indikator 2.....	72
Gambar 4. 30. Tampilan <i>Widget</i> : <i>Switch</i> Ketika Dinyalakan	73
Gambar 4. 31. Tampilan <i>Widget</i> : <i>Switch</i> Ketika Dimatikan.....	73
Gambar 4. 32. Status <i>Wi-Fi</i> Berhasil Terhubung	74
Gambar 4. 33. <i>Wi-Fi</i> Mencoba Menghubungkan Kembali	74
Gambar 4. 34. <i>Wi-Fi</i> Gagal Terhubung	74
Gambar 4. 35. IoT Berhasil Tersambung.....	75
Gambar 4. 36. IoT Gagal Tersambung dengan Kode <i>Error -2</i>	75
Gambar 4. 37. IoT Gagal Tersambung dengan Kode <i>Error -4</i>	76
Gambar 4. 38. IoT Gagal Tersambung dengan Kode <i>Error 5</i>	76
Gambar 4. 39. <i>Publish</i> dan <i>Subscribe</i>	77
Gambar 4. 40. Tampilan Animasi Awal	77
Gambar 4. 41. Tampilan <i>Loading</i>	78

Gambar 4. 42. Tampilan <i>STANDBY</i>	78
Gambar 4. 43. Tampilan <i>OFF</i>	78
Gambar 4. 44. Tampilan Kecepatan Putaran (RPM) Motor DC secara aktual pada LCD	78
Gambar 4. 45. Tampilan Kecepatan Putaran (RPM) Motor DC pada <i>Line Charts</i>	80
Gambar 4. 46. Tampilan Kecepatan Putaran (RPM) Motor DC pada Arduino IDE	80
Gambar 4. 47. Grafik Kendali PID ZN-2 terhadap Kecepatan Putaran (RPM) Motor DC.....	81
Gambar 4. 48. Pengujian dengan <i>Digital Tachometer</i>	81

Halaman ini sengaja dikosongkan

DAFTAR PERSAMAAN

Persamaan 2. 1	20
Persamaan 2. 2	20
Persamaan 2. 3	21
Persamaan 3. 1	40

Halaman ini sengaja dikosongkan

DAFTAR *SOURCE CODE*

<i>Source Code 3. 1. Kalibrasi CPR.....</i>	41
<i>Source Code 4. 1. Main.ino</i>	54
<i>Source Code 4. 2. Connection.h</i>	55
<i>Source Code 4. 3. WiFi_Connection.ino</i>	56
<i>Source Code 4. 4. Ubidots_Connection.ino.....</i>	57
<i>Source Code 4. 5. Ubidots_PubSub.ino.....</i>	58
<i>Source Code 4. 6. IO_Device.h</i>	60
<i>Source Code 4. 7. IO_Device.ino</i>	61
<i>Source Code 4. 8. PID_Control.ino</i>	64

Halaman ini sengaja dikosongkan

BAB I

PENDAHULUAN

1.1. Latar Belakang

Seiring dengan dinamika perkembangan zaman, kemajuan teknologi telah menjadi bagian penting dalam berbagai aktivitas manusia [1]. Transformasi digital yang berkembang dengan sangat pesat ditandai oleh hadirnya berbagai perangkat elektronik yang semakin canggih dan inovatif, yang secara signifikan mendukung efisiensi dan kemudahan dalam berbagai aspek kehidupan [2], [3], [4], [5]. Dalam dunia industri, motor DC pada sistem *Conveyor* sering kali berperan sebagai tulang punggung dalam proses produksi, karena kemampuannya dalam menggerakkan material dari satu titik ke titik lainnya secara cepat dan efisien [6], [7]. Namun, dalam praktiknya, sistem ini kerap menghadapi tantangan berupa kesalahan *steady-state* yang cukup signifikan, terutama ketika menghadapi beban angkut yang berlebihan [8]. Kondisi ini dapat menyebabkan penurunan kecepatan putaran (RPM) motor DC, yang pada akhirnya berdampak pada ketidakstabilan alur produksi [9].

Untuk mengatasi permasalahan performa motor akibat perubahan beban, diperlukan sebuah sistem pengendali yang mampu mengompensasi kekurangan tersebut dan memberikan kinerja yang lebih optimal. Salah satu metode pengendalian yang paling umum dan efektif digunakan, baik dalam industri maupun bidang robotika, adalah pengendali PID (*Proportional-Integral-Derivative*) [10], [11]. Meskipun terdapat berbagai metode kontrol lain seperti *Fuzzy Logic*, *Neural Network Control*, dan *Model Predictive Control* (MPC), pengendali PID tetap menjadi pilihan utama dalam sistem kendali industri. Hal ini disebabkan oleh kemudahan implementasinya, efektivitas biaya, serta kemampuannya dalam memberikan performa yang memadai di berbagai kondisi operasi [12]. Dengan menggabungkan aksi *Proportional*, *Integral*, dan *Derivative*, kontroler PID mampu meningkatkan respons sistem secara keseluruhan, sehingga menjadikannya lebih cepat, akurat dalam mencapai *setpoint*, dan stabil terhadap gangguan atau fluktuasi beban [8].

Metode *tuning* PID yang umum digunakan antara lain : *Ziegler-Nichols (ZN)*, *Cohen-Coon*, *Manual Tuning*, *Software-based Autotuning*, dan *Relay Feedback* [13], [14], [15], [16], [17]. Khusus untuk metode *Ziegler-Nichols*, pendekatan ini dibagi lagi menjadi dua, yaitu metode pertama dan metode kedua. PID *Ziegler-Nichols* metode 1 membahas tentang penalaan berdasarkan respons terhadap *input* langkah (*step response*). Metode ini dikenal sebagai *open-loop tuning* atau *reaction curve method*, di mana parameter PID ditentukan berdasarkan kurva

tanggapan sistem terhadap lonjakan masukan (biasanya berupa perubahan mendadak pada *input kontrol*). Parameter seperti waktu tunda (L) dan konstanta waktu (T) digunakan untuk menghitung nilai P, I, dan D. Sementara itu, PID *Ziegler-Nichols* metode 2 membahas tentang penalaan berdasarkan osilasi berkelanjutan pada sistem tertutup (*closed-loop tuning*). Metode ini disebut juga sebagai *ultimate gain method*, di mana pengguna menaikkan nilai gain proporsional (K_p) secara bertahap hingga sistem menunjukkan osilasi terus-menerus [18], [19].

Penelitian oleh Zaidir Jamal menunjukkan bahwa penerapan pengendali PID pada miniatur *Hopper-Conveyor Plant* mampu menjaga aras bahan baku padat tetap konstan dengan akurasi tinggi. Hasil yang diperoleh meliputi: *overshoot* sebesar 1 cm (0,05%), *rise time* 280 detik, dan nilai IAE rata-rata 3.1733. Waktu respons tersebut tergolong lambat, terutama jika sistem diterapkan pada industri yang membutuhkan respons cepat terhadap perubahan kondisi [20].

Penelitian oleh Alifa Restu Janwar Wiriahan dan Andry Irawan menunjukkan hasil positif setelah menerapkan metode kontrol PID, kecepatan putaran motor DC menunjukkan peningkatan yang signifikan dengan mencapai nilai *set point* yang ditentukan dengan stabilitas yang lebih baik. Saat mengalami gangguan, grafik keluaran akan menunjukkan penyimpangan sesaat dari nilai *set point*, sebelum akhirnya sistem secara efektif mengembalikan kecepatan putaran ke titik stabil semula. Namun untuk penalaan PID masih dilakukan secara manual melalui metode osilasi (*trial and error*) [21].

Menurut penelitian yang dilakukan oleh Rizky Hansza dan Subuh Isnur Haryudo, dengan adanya IoT, pengguna dapat mengelola sejumlah perangkat elektronik melalui internet. Berkait sensor *voice recognition*, perintah untuk mengatur kecepatan Motor DC menjadi sangat mudah dilakukan hanya dengan suara dari jarak jauh, tetapi tidak cukup baik jika digunakan dalam keadaan tidak ideal [22].

Selain itu, ada juga penelitian dari David Setiawan yang menyatakan bahwa kontrol kecepatan motor DC bisa dilakukan melalui koneksi *Bluetooth* berbasis Android untuk meningkatkan kenyamanan dan kemampuan kendali jarak jauh. Namun, penelitian ini memiliki keterbatasan pada jangkauan kontrolnya, sehingga perlu pengembangan alternatif lebih lanjut [23].

Penelitian yang dilakukan oleh penulis ini akan jauh berbeda dengan penelitian-penelitian sebelumnya, karena menggunakan algoritma PID *Ziegler-Nichols* yang telah diintegrasikan dengan *Internet of Things* (IoT) untuk mengendalikan dan memantau kecepatan putaran (RPM) motor DC pada *Conveyor*. Selain itu, sistem ini dilengkapi dengan fitur pengendalian seperti perintah *On / Off*, serta pengaturan arah putaran pada motor DC (maju atau mundur), yang tentunya dapat dioperasikan dari jarak yang dekat maupun dari jarak yang jauh. Pengendalian jarak

dekat menggunakan Potensiometer dan *Push Button*, sedangkan pengendalian jarak jauh menggunakan Ubidots sebagai Platform IoT.

Untuk dapat membuat sistem tersebut, maka dibutuhkan sebuah metodologi yang sesuai. Penulis menetapkan RAD (*Rapid Application Development*) sebagai metodologi di penelitian ini. RAD adalah model pengembangan perangkat lunak yang dilakukan secara bertahap dengan pendekatan yang terstruktur, khususnya untuk proyek dengan waktu pengerjaan yang singkat. Pengembangan sistem dimulai dari tahap *requirement*, lalu *prototyping*, kemudian *output*, dan yang terakhir adalah *testing* [24].

1.2. Rumusan Masalah

Berdasarkan pemaparan yang ada di latar belakang, adapun rumusan masalah yang dapat dipetik dari pembahasan dan pelaksanaan penelitian ini, yaitu sebagai berikut :

- a. Bagaimana caranya mengimplementasikan metode PID *Ziegler-Nichols* pada sistem pengendalian kecepatan putaran (RPM) motor DC pada *Conveyor* ?
- b. Bagaimana caranya mengendalikan *On / Off* dan arah putaran motor DC pada *Conveyor* dari jarak jauh maupun jarak dekat ?
- c. Bagaimana proses implementasi dan integrasi protokol MQTT pada ESP32 ?
- d. Bagaimana hasil pengendalian kecepatan putaran (RPM) motor DC pada *Conveyor* saat menggunakan metode PID *Ziegler-Nichols* ?
- e. Bagaimana efektivitas kendali *On / Off* dan arah putaran motor DC pada sistem *Conveyor* saat dioperasikan dari jarak jauh maupun jarak dekat ?

1.3. Tujuan

Berdasarkan permasalahan yang diteliti, adapun tujuan dari pelaksanaan penelitian ini yaitu sebagai berikut :

- a. Untuk mengimplementasikan metode PID *Ziegler-Nichols* pada sistem kendali kecepatan putaran (RPM) motor DC pada *Conveyor*.
- b. Untuk mengimplementasikan kendali *On / Off* dan arah putaran motor DC pada *Conveyor* dari jarak jauh maupun jarak dekat.
- c. Untuk mengetahui cara kerja dan penggunaan protokol MQTT pada ESP32.

- d. Untuk mengetahui hasil pengendalian kecepatan putaran (RPM) motor DC secara *real-time* pada *Conveyor*.
- e. Untuk mengetahui efektivitas kendali *On / Off* dan arah putaran motor DC pada *Conveyor* dari jarak jauh maupun jarak dekat.

1.4. Batasan Masalah

Berdasarkan apa yang telah dijelaskan dalam rumusan masalah di atas, maka batasan masalah perlu ditetapkan oleh penulis, yaitu sebagai berikut :

- a. ESP32 merupakan *Controller* yang digunakan oleh penulis untuk mengendalikan *On* atau *Off*, arah putaran (Maju atau Mundur), dan kecepatan putaran (RPM) motor DC pada *Conveyor*.
- b. Protokol komunikasi IoT yang digunakan adalah MQTT.
- c. Kendali PID menggunakan metode *Ziegler-Nichols* dengan *Loop Tertutup*.
- d. Ubidots sebagai platform IoT digunakan untuk memantau nilai kecepatan putaran (RPM) motor DC pada *Conveyor*, sekaligus untuk mengendalikan *On / Off* dan juga arah putaran dari motor DC tersebut.
- e. *Digital Laser Photo Tachometer* digunakan oleh penulis untuk mengukur kecepatan putaran (RPM) motor DC pada *Conveyor* guna memperoleh data yang lebih akurat dan meyakinkan.
- f. RAD (*Rapid Application Development*) digunakan sebagai metodologi dalam penelitian ini. Sedangkan untuk pengujian sistem, menggunakan pendekatan eksperimental.

1.5. Manfaat

Berdasarkan penelitian ini, manfaat yang diperoleh antara lain :

- a. Bagi penulis, hal ini dapat dijadikan sebagai ajang pembuktian kemampuan dalam menciptakan suatu inovasi yang bermanfaat.
- b. Bagi pembaca, dapat dijadikan sebagai acuan dalam mempelajari ilmu *Internet of Things* (IoT) dan juga kendali PID dengan metode *Ziegler-Nichols*.

BAB II

TINJAUAN PUSTAKA

2.1. Penelitian Terdahulu

Landasan teori dan temuan dari penelitian sebelumnya merupakan hal yang sangat penting dalam membangun kerangka berpikir pada penelitian ini. Penelitian-penelitian terdahulu yang relevan dengan permasalahan yang dikaji akan memberikan gambaran yang lebih jelas tentang perkembangan pengetahuan di bidang ini serta menjadi acuan dalam merumuskan hipotesis dan metodologi penelitian. Untuk itu, penulis akan mengkaji secara mendalam berbagai jurnal ilmiah yang relevan sebagai sumber acuan utama dalam penulisan tugas akhir ini. Berikut ini akan dipaparkan hasil penelitian terdahulu yang digunakan oleh penulis sebagai referensi, antara lain :

Penelitian yang dilakukan oleh David Setiawan (2017), berjudul “Sistem Kontrol Motor DC Menggunakan PWM Arduino Berbasis *Android System*” ini membahas tentang sistem kontrol motor DC yang menggunakan sinyal PWM pada *board* Arduino Uno yang dikendalikan melalui aplikasi Android. Pengaturan kecepatan motor dilakukan dengan memanfaatkan variasi lebar pulsa PWM tanpa mengubah besar tegangan keluaran dari pin Arduino. Tegangan aktual untuk motor dihasilkan oleh *motor driver* IC L293D yang mengonversi sinyal PWM menjadi tegangan yang sesuai. Sistem ini dirancang untuk mengontrol dua motor DC dengan arah putaran maju dan mundur, serta sembilan level kecepatan, dan dikendalikan melalui *Bluetooth* dengan jangkauan hingga 45 meter di dalam ruangan dan 70 meter di luar ruangan. Untuk aplikasi yang digunakan sebagai pengontrol adalah *MIT App Inventor*. Hasil penelitian menunjukkan bahwa sistem ini efisien, mudah diimplementasikan, dan cocok untuk aplikasi otomatisasi seperti pintu atau pagar otomatis yang membutuhkan pengaturan kecepatan dan arah motor secara nirkabel [23].

Penelitian yang dilakukan oleh Rosalina dkk (2017), berjudul “Analisis Pengaturan Kecepatan Motor DC Menggunakan Kontrol PID (*Proportional Integral Derivative*)” ini membahas tentang pengaturan kecepatan motor DC menggunakan kontrol PID untuk mengatasi ketidakstabilan akibat perubahan beban. Hasil penelitian menunjukkan bahwa sistem *open loop* memiliki performa lambat dengan waktu *settling* yang tinggi, sedangkan sistem dengan kontrol PI dan PID memberikan hasil yang jauh lebih baik. Kontrol PI menghasilkan *rise time* sangat cepat (0,0069 detik) dan tanpa *overshoot*, namun memerlukan waktu lebih lama untuk mencapai kestabilan penuh. Sebaliknya, kontrol PID memberikan waktu *settling* yang lebih cepat (0,5963

detik) meskipun dengan *overshoot* sebesar 13,54%. Keduanya menunjukkan sistem yang stabil, namun PID dinilai lebih efektif untuk kebutuhan tanggapan cepat dan pengendalian presisi dalam sistem motor DC [25].

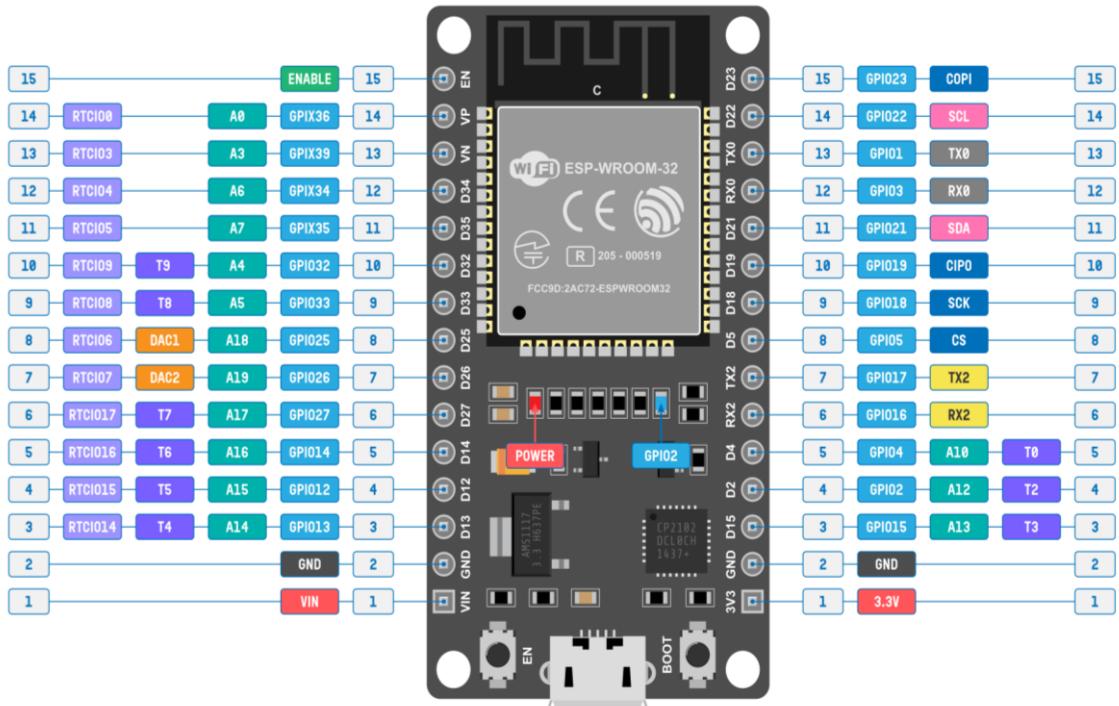
Penelitian yang dilakukan oleh Mila Diah Ika Putri dkk (2021), berjudul “Pengendali kecepatan sudut motor DC menggunakan *control PID* dan *tuning Ziegler-Nichols*” ini membahas tentang penalaan pengendali PID yang menggunakan metode *Ziegler-Nichols*, sehingga menghasilkan *output* dengan grafik kecepatan motor yang stabil. Metode *Ziegler-Nichols* 1 menghasilkan *output* dengan nilai *overshoot* terbaik sebesar 5%. Sedangkan metode *Ziegler-Nichols* 2 menghasilkan nilai *overshoot* terbaik sebesar 1%. Berdasarkan nilai *overshoot*, *settling time*, dan *rise time*, nilai pengendali PID terbaik yang didapatkan dari penelitian ini yaitu $K_p = 8,23712$, $K_i = 1,65072$, dan $K_d = 0,41268$. Berdasarkan nilai *overshoot* terkecil, nilai pengendali PID terbaik yang didapatkan dari penelitian ini yaitu $K_p = 0,05988$, $K_i = 0,038922$, dan $K_d = 0,0093413$ [10].

Penelitian yang dilakukan oleh Rizky Hansza dan Subuh Haryudo (2020), berjudul “Rancang Bangun Kontrol Motor DC dengan PID Menggunakan Perintah Suara dan *Monitoring* Berbasis *Internet of Things* (IOT)” ini membahas tentang sistem PID yang memanfaatkan sensor *voice recognition*, Arduino Uno, dan NodeMCU dengan aplikasi Blynk sebagai media *monitoring*. Parameter PID yang diperoleh melalui metode analitik adalah $K_p = 2,021751$, $K_i = 2,595857347$, dan $K_d = 0,7873077$, yang mampu memberikan respons kecepatan mendekati *setpoint* dengan *error* rata-rata di bawah 2%. Meski sistem menunjukkan performa yang cukup baik, ternyata ada beberapa keterbatasan yang ditemukan pada akurasi sensor *voice recognition* yang kondisinya itu kurang ideal serta adanya fluktuasi dalam pembacaan arus. Penelitian ini menunjukkan bahwa integrasi *voice recognition* dan IoT efektif dalam pengendalian motor DC dan berpotensi untuk dikembangkan lebih lanjut [22].

2.2. ESP32 DOIT DevKit-C V1

ESP32 merupakan mikrokontroler tipe *System on Chip* (SoC) yang telah dilengkapi dengan konektivitas *Wi-Fi* 802.11 b/g/n dan *Bluetooth* versi 4.2, serta berbagai fitur periferal lainnya. *Chip* ini dirancang secara komprehensif karena telah memiliki prosesor, memori, serta akses ke GPIO (*General Purpose Input Output*) dalam satu kesatuan. ESP32 dapat dijadikan sebagai alternatif dari mikrokontroler seperti Arduino karena memiliki kemampuan untuk terhubung langsung ke jaringan *Wi-Fi*, tanpa memerlukan modul tambahan. Hal ini menjadikannya

sangat cocok digunakan dalam berbagai proyek yang berkaitan dengan *Internet of Things* (IoT) dan perangkat yang memerlukan komunikasi nirkabel [26].



Sumber : electronicsforu.com

Gambar 2. 1. ESP32 DOIT DevKit-C V1

Berikut adalah beberapa fitur utama dari ESP32 DOIT DevKit-C V1 [27] :

a. Mikrokontroler ESP32

Chip ini memiliki dua inti prosesor 32-bit, *Wi-Fi*, *Bluetooth*, dan berbagai antarmuka seperti I2C, SPI, UART, dan ADC.

b. Papan Pengembangan

ESP32 DOIT DevKit-C V1 menyediakan 30 pin GPIO untuk menghubungkan ke berbagai komponen elektronik seperti sensor, aktuator, dan layar.

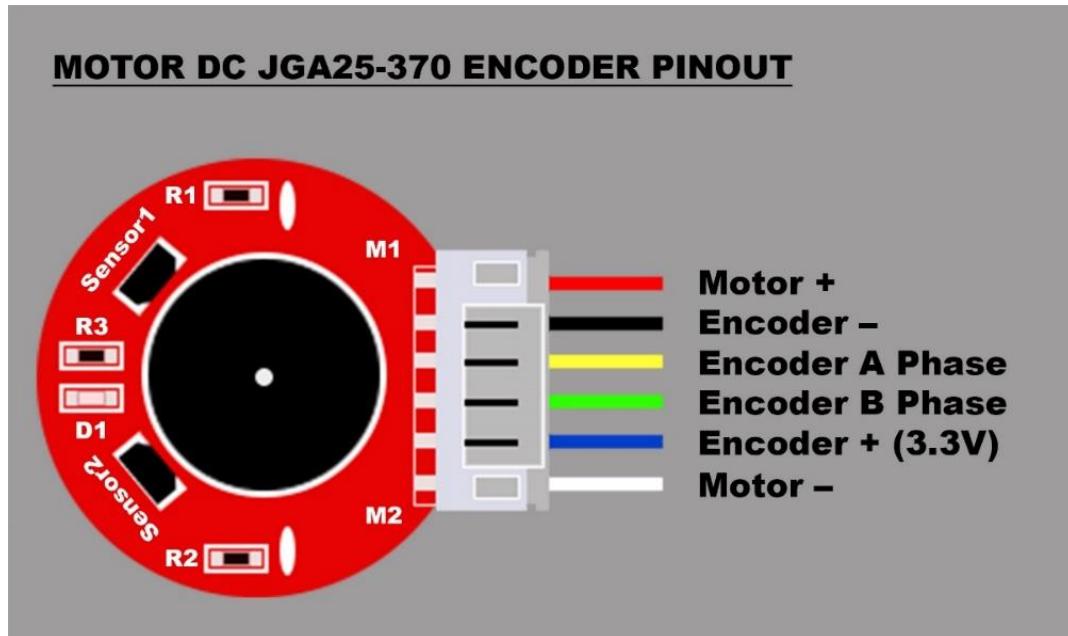
c. USB-to-Serial Converter

ESP32 DOIT DevKit-C V1 dilengkapi dengan konverter USB-serial bawaan yang memungkinkan pengunggahan program melalui *port USB micro*.

d. LED Indikator

ESP32 DOIT DevKit-C V1 memiliki LED indikator yang dapat menunjukkan status daya dan aktivitas ESP32.

2.3. Motor DC JGA25-370 12V *Encoder*



Gambar 2. 2. Motor DC JGA25-370 12V *Encoder* Pinout

Motor DC JGA25-370 12V *Encoder* termasuk salah satu jenis motor *gear* yang berukuran kecil, yang banyak digunakan karena mampu menghasilkan torsi cukup tinggi, efisiensi energi, serta kemudahan penggunaan. Motor ini tersedia dalam berbagai rasio *gear* dan sangat cocok untuk proyek DIY, robotika, serta sistem otomatisasi sederhana [28]. Dilengkapi dengan *encoder*, motor ini mampu memberikan kontrol yang lebih presisi terhadap kecepatan dan posisi, sehingga ideal untuk sistem kendali tertutup (*closed-loop*). Dengan fitur ini, motor dapat mengoreksi pergerakan secara otomatis dan memantau performa secara *real-time*, menjadikannya pilihan yang andal untuk aplikasi yang membutuhkan akurasi tinggi, seperti robotika dan navigasi.

Motor DC JGA25-370 12V yang dilengkapi dengan *encoder* umumnya memanfaatkan sensor *Hall Effect Magnetic* tipe *inkremental* dengan dua saluran. Sistem *encoder* ini menggunakan cakram atau cincin magnet kecil yang dipasang di ujung poros motor. Dua sensor *Hall Effect Magnetic* ditempatkan berdekatan (saluran A dan B) untuk mendeteksi perubahan medan magnet saat poros berputar. Sensor *Hall Effect Magnetic* ini bekerja dengan membaca perubahan medan magnet yang timbul dari pergerakan cakram magnet.

Ketika motor berputar, kutub-kutub magnet melewati sensor, menghasilkan sinyal listrik berupa pulsa digital. Karena terdapat dua sensor yang dipasang dengan perbedaan sudut 90 derajat, sinyal yang dihasilkan berupa sinyal kuadratur, di mana pulsa dari saluran A dan B bergeser seperempat siklus. Pola sinyal ini memungkinkan sistem mendekripsi arah putaran (misalnya, A mendahului B berarti maju, sebaliknya berarti mundur) serta menghitung jumlah putaran secara presisi. Keunggulan utama dari *Encoder Hall Effect Magnetic* ini adalah ketahanannya terhadap guncangan serta tidak terpengaruh terhadap debu, menjadikannya lebih andal dibanding *encoder* optik. Motor ini sendiri secara fundamental beroperasi dengan arus listrik searah.

2.3.1. Prinsip Kerja Motor DC

Motor DC bekerja berdasarkan prinsip bahwa arus listrik yang mengalir melalui medan magnet akan menghasilkan gaya elektromagnetik yang menggerakkan rotor. Prinsip dasar ini dikenal sebagai Hukum *Lorentz* atau hukum yang menjelaskan tentang gaya yang bekerja pada partikel bermuatan listrik yang bergerak dalam medan listrik atau medan magnet [29].

2.3.2. Elemen - Elemen Motor DC



Sumber : theengineeringknowledge.com

Gambar 2. 3. Elemen Motor DC

a. Stator (Bagian Statis)

Merupakan bagian tetap dari motor yang menyediakan medan magnet. Pada motor DC, medan magnet yang diperlukan untuk menghasilkan torsi dapat dibentuk menggunakan magnet permanen atau melalui lilitan elektromagnetik yang dialiri arus listrik [30].

b. Rotor (Bagian Berputar) atau *Armature*

Merupakan bagian yang berputar dari motor. Arus listrik mengalir melalui kumparan yang terpasang pada rotor, yang kemudian berinteraksi dengan medan magnet stator untuk menghasilkan torsi dan gerakan rotasi [30].

c. Cela udara (*Air Gap*)

Cela udara adalah bagian yang memisahkan bagian yang diam (stator) dan bagian yang berputar (rotor). Cela ini penting, agar motor bisa bergerak. Tanpa cela udara, rotor dan stator akan saling bergesekan, sehingga motor tidak akan bisa berputar [30].

d. Komutator dan Sikat (*Brushes*)

Komutator adalah bagian yang memutuskan dan menghubungkan arus listrik pada *armature* sehingga arus listrik selalu mengalir dalam arah yang sama. Sikat (*brushes*) adalah komponen yang menyalurkan arus dari sumber daya listrik ke komutator [30].

2.4. Motor Driver L298N

L298N adalah salah satu *Motor Driver* yang paling banyak digunakan untuk mengendalikan kecepatan serta arah putaran motor, terutama pada aplikasi seperti robot *line follower* atau *line tracer*. *Driver* ini dikenal memiliki tingkat akurasi yang baik dalam mengatur motor, serta mudah dalam penggunaannya. Untuk mengoperasikan L298N, diperlukan enam pin dari mikrokontroler, dua pin digunakan sebagai *Enable* (masing-masing untuk satu motor, karena L298N mampu mengendalikan dua motor DC), dan empat pin lainnya digunakan untuk mengatur arah dan kecepatan putaran kedua motor. Sementara itu, *output* dari *driver* ini menyediakan dua pin untuk masing-masing motor [31].

Fungsi Utama [32] :

a. Penguat Arus

L298N berfungsi sebagai penguat arus. Arus listrik yang lemah dari mikrokontroler akan diperkuat oleh modul ini sehingga cukup kuat untuk menggerakkan motor DC.

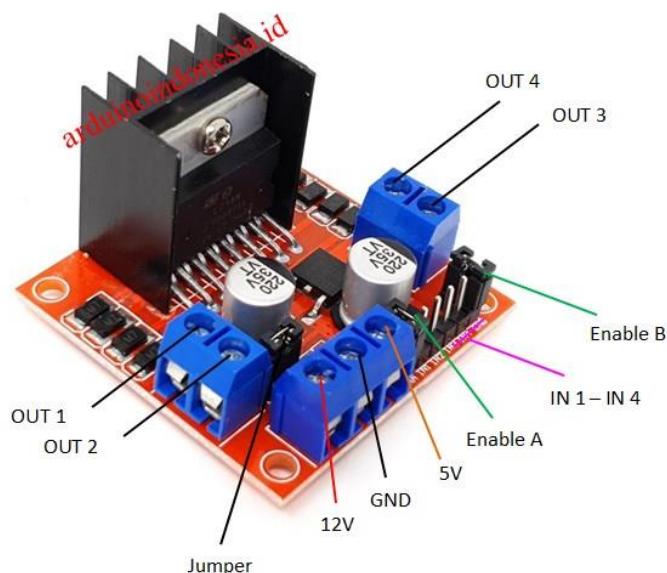
b. Pengontrol Kecepatan dan Arah

Modul ini memungkinkan kita untuk mengatur kecepatan putaran motor DC dan mengubah arah putarannya secara digital melalui mikrokontroler.

c. Perlindungan

Dilengkapi dengan fitur perlindungan seperti dioda *flyback*, *heatsink*, *fuse*, dan lain-lain yang berfungsi untuk melindungi komponen dari kerusakan akibat tegangan balik.

Modul L298N bekerja berdasarkan prinsip *H-bridge*. *H-bridge* adalah sebuah rangkaian elektronik yang memungkinkan arus listrik mengalir ke beban (motor DC) dalam dua arah (maju atau mundur). Dengan mengatur sinyal digital yang diberikan ke L298N, kita dapat mengontrol arah arus yang mengalir ke motor DC tersebut [32].



Sumber : arduinoindonesia.id

Gambar 2. 4. Modul Motor Driver L298N

2.5. LCD (*Liquid Crystal Display*) I2C

LCD I2C merupakan salah satu jenis *display* yang dilengkapi dengan fitur I2C sehingga lebih ringkas dalam hal pengkabelan. *Display* ini dapat menampilkan data, baik berupa karakter, huruf, maupun grafik. LCD tidak menghasilkan cahaya sendiri, melainkan bekerja dengan cara memantulkan cahaya dari lingkungan sekitarnya melalui sistem *front-lit*, atau mentransmisikan cahaya dari sumber *back-lit* yang berada di belakang layar [31].



Sumber : kursuselektronikaku.blogspot.com

Gambar 2. 5. LCD I2C

Cara Kerja LCD I2C [33] :

a. Mikrokontroler mengirimkan data

Mikrokontroler mengirimkan perintah dan data ke LCD melalui bus I2C. SDA digunakan sebagai media pertukaran data pada LCD I2C, sedangkan SCL digunakan sebagai media sinkronisasi waktu pada LCD I2C.

b. LCD menerima data

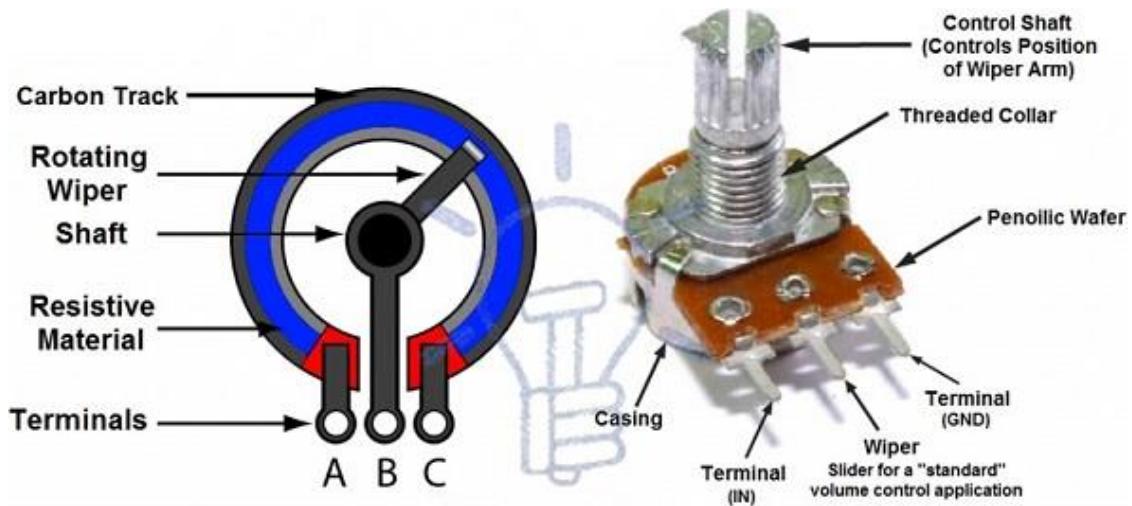
LCD menerima data dan memprosesnya untuk menampilkan informasi pada layar.

c. LCD menampilkan informasi

Piksel-piksel pada LCD akan menyala atau redup sesuai dengan data yang diterima, sehingga membentuk gambar atau teks yang diinginkan.

2.6. Potensiometer

Potensiometer adalah komponen elektronik yang berperan sebagai resistor variabel, yang memungkinkan pengguna untuk mengatur besar kecilnya resistansi dalam suatu rangkaian. Umumnya, potensiometer memiliki tiga terminal: satu terminal *input*, satu terminal *output*, dan satu terminal geser (*wiper*). Dengan memutar atau menggeser porosnya, posisi terminal geser berubah, sehingga nilai resistansi yang dihasilkan pun ikut berubah. Selain itu, potensiometer juga sering digunakan untuk mengatur kecepatan motor DC sesuai dengan kebutuhan sistem [34].



Sumber : wikielektronika.com

Gambar 2. 6. Potensiometer

Potensiometer umumnya terdiri dari tiga bagian utama [35] :

a. Elemen Resistif

Bagian ini terbuat dari bahan *resistif* seperti karbon atau logam, yang berfungsi sebagai jalur aliran arus listrik.

b. Wiper

Bagian yang dapat digerakkan dan berfungsi sebagai kontak geser pada elemen resistif. Dengan menggerakkan *wiper*, kita dapat mengubah nilai resistansi.

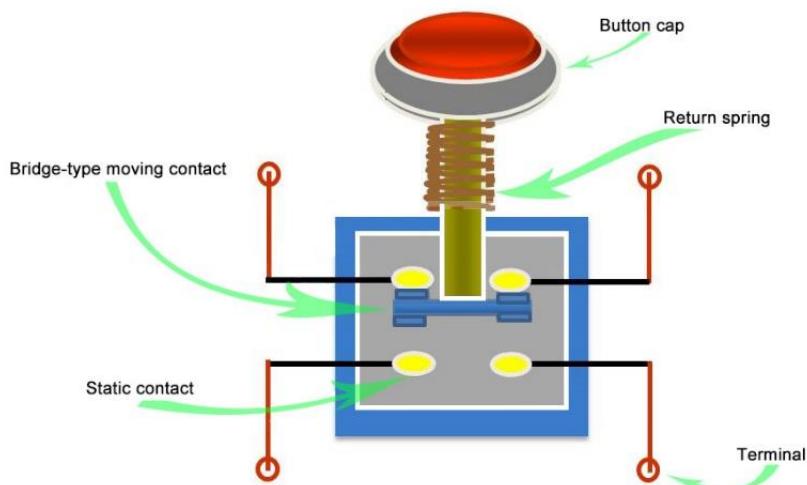
c. Terminal

Bagian tempat menghubungkan potensiometer dengan rangkaian elektronik lainnya.

Potensiometer adalah komponen elektronik vital yang memungkinkan kita untuk mengatur secara manual nilai resistansi dalam suatu rangkaian. Dengan memahami prinsip kerjanya sebagai pembagi tegangan variabel, Anda dapat mengoptimalkan kinerja berbagai sirkuit elektronik, mulai dari pengatur volume hingga pengatur kecepatan motor DC [35].

2.7. Push Button

Push button atau tombol tekan adalah perangkat sederhana yang berfungsi untuk menyambungkan atau memutuskan aliran listrik dengan mekanisme tekan tanpa kunci (*unlock*). Artinya, saklar hanya aktif saat ditekan dan akan kembali ke kondisi semula ketika dilepas. Saklar ini memiliki dua kondisi utama, yaitu *On* (1) dan *Off* (0), yang sangat penting dalam pengoperasian perangkat listrik. Karena sistem kerjanya langsung dikendalikan oleh operator, *push button* sering digunakan sebagai alat utama untuk memulai atau menghentikan operasi mesin di industri. Bahkan pada mesin otomatis sekalipun, keberadaan saklar seperti ini tetap menjadi bagian yang paling penting untuk mengatur kondisi *On* dan *Off* [36].

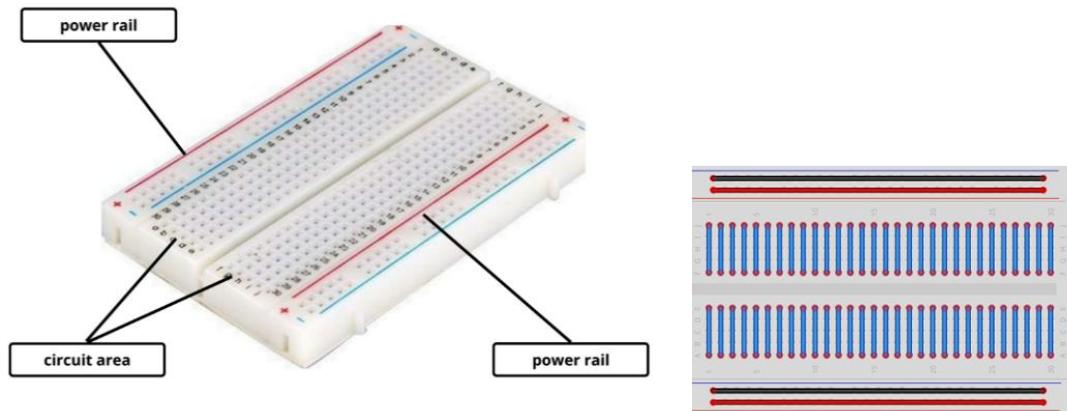


Sumber : [FutureBotz](#)

Gambar 2. 7. *Push Button*

2.8. Breadboard

Breadboard adalah papan yang digunakan untuk merancang dan menguji rangkaian elektronik sederhana tanpa perlu menyolder komponen. Biasanya terbuat dari bahan plastik dan dilengkapi dengan deretan lubang yang telah disusun sesuai dengan pola koneksi internal tertentu. *Breadboard* memudahkan proses pembuatan prototipe, karena memungkinkan penyusunan ulang komponen dengan cepat. Di pasaran, *Breadboard* tersedia dalam tiga ukuran utama: *Mini*, *Medium*, dan *Large* [37]. *Medium Breadboard* umumnya memiliki sekitar 400 titik koneksi (*Tie-Points*) dan dilengkapi dengan baris-baris lubang yang terhubung secara internal dan dapat memudahkan pemasangan komponen seperti resistor, LED, dan kabel *jumper*. Beberapa model juga memiliki bagian belakang yang dilapisi perekat, sehingga dapat ditempelkan ke permukaan atau mikrokontroler seperti Arduino.



Sumber : atl.aim.gov.in dan Kelas Robot

Gambar 2. 8. Medium Breadboard

2.9. Digital Tachometer

Tachometer digital adalah alat yang digunakan untuk mengukur kecepatan putaran suatu poros atau objek berputar dalam satuan RPM (*Revolutions Per Minute*). Berbeda dengan versi analog, *tachometer* digital menampilkan hasil pengukuran secara langsung melalui layar LCD. Alat ini dikenal karena akurasinya yang tinggi dan respons yang cepat terhadap perubahan kecepatan, sehingga sangat andal dalam berbagai aplikasi [38].

Tachometer digital banyak digunakan dalam industri, otomotif, dan proyek berbasis mikrokontroler seperti Arduino, terutama untuk memantau performa motor, kipas, atau mesin lainnya agar tetap beroperasi dalam batas kecepatan yang aman dan efisien.



Sumber : jakartanotebook.com

Gambar 2. 9. Digital Tachometer

2.10. Kendali PID (*Proportional-Integral-Derivative*)

Kendali PID adalah jenis kendali yang digunakan untuk mencapai tingkat presisi yang tinggi dalam sistem instrumentasi dengan memanfaatkan mekanisme umpan balik (*feedback*). Kendali PID terdiri dari tiga elemen utama, yaitu kendali Proporsional (P), Integral (I), dan Derivatif (D). Ketiga elemen ini dapat dioperasikan secara bersama-sama atau secara terpisah, tergantung pada karakteristik respons yang diharapkan dari sistem yang dikendalikan [39].

2.10.1. Sejarah Kendali PID

Pada awal abad ke-20, sistem kendali *feedback* mulai berkembang pesat, terutama dalam industri otomotif dan manufaktur. Metode PID sendiri pertama kali dijelaskan oleh *Elmer Sperry* pada tahun 1911 ketika ia merancang sistem kendali untuk *autopilot* pesawat. Namun, pengembangan formal dari kendali PID dikreditkan kepada *Nicolas Minorsky*, yang merupakan seorang insinyur Rusia-Amerika pada tahun 1922. Ia menggunakan prinsip-prinsip PID untuk mengendalikan arah kapal laut, yang membutuhkan kestabilan dan kecepatan dalam menghadapi perubahan kondisi lingkungan. Hal ini dapat ditentukan dari persamaan diferensial sistem [18].

Metode *tuning* PID mulai dikembangkan lebih lanjut pada tahun 1940-an. Pada tahun 1942, salah satu metode *tuning* PID yang paling terkenal yaitu metode *Ziegler-Nichols* yang diperkenalkan oleh *John G. Ziegler* dan *Nathaniel B. Nichols*. Keduanya bekerja di *Taylor Instrument Company*, sebuah perusahaan instrumen kontrol [18]. Tujuan mereka adalah untuk memberikan metode sederhana dan praktis, yang memungkinkan insinyur mengoptimalkan parameter PID secara efektif tanpa harus memahami kompleksitas matematika di balik model proses [40].

2.10.2. Komponen Utama dalam Kendali PID

Kendali PID terdiri dari tiga komponen utama [41], masing-masing dengan fungsi spesifik dalam mengendalikan sistem yaitu :

a. *Proportional* (P)

Komponen ini menghasilkan respons kendali yang sebanding dengan *error* (perbedaan antara nilai *setpoint* dan nilai aktual). Semakin besar *error*, semakin besar juga tindakan korektif yang diberikan. Komponen ini membantu mengurangi *error* dengan cepat. Namun, jika hanya menggunakan kendali P, sistem mungkin tidak dapat menghilangkan *error* secara sempurna (*error steady-state* tetap ada).

b. Integral (I)

Komponen integral menghitung jumlah kumulatif dari *error* dari waktu ke waktu dan menghasilkan tindakan korektif berdasarkan akumulasi tersebut. Komponen ini berfungsi menghilangkan *error steady-state* yang tidak bisa dihilangkan oleh komponen P saja dengan mengatur *output* agar *error total* menjadi nol.

c. Derivative (D)

Komponen ini memberikan respons kendali yang sebanding dengan laju perubahan *error*. Ini berarti, jika *error* berubah dengan cepat, komponen D akan menghasilkan tindakan korektif yang signifikan. Fungsi dari komponen ini untuk membantu meningkatkan stabilitas sistem dan mengurangi *overshoot* (melampaui *setpoint*). Ini juga dapat mempercepat respons sistem.

2.10.3. Prinsip Kerja Kendali PID

Kendali PID bekerja dengan cara menghitung nilai *output* berdasarkan kombinasi linier dari ketiga komponen tersebut. *Output* dari sistem kendali ini diberikan pada aktuator (misalnya: motor, *valve*, atau *heater*) yang kemudian mengubah proses untuk mengurangi *error* [15], [41].

2.10.4. Karakteristik Sistem Kendali PID

a. Stabilitas

Sistem PID yang dirancang dengan baik akan memberikan respons yang stabil tanpa osilasi berlebihan [42].

b. Presisi

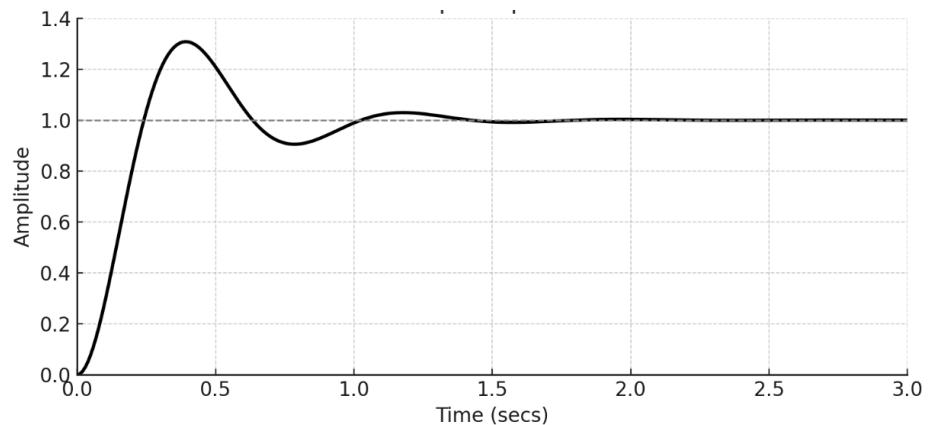
Dengan kombinasi P, I, dan D, sistem dapat mencapai presisi yang tinggi dalam menjaga variabel proses sesuai dengan *setpoint* [42].

c. Respons Dinamis

Sistem PID dapat disesuaikan untuk memberikan respons yang cepat terhadap perubahan *setpoint* / gangguan, sambil meminimalkan *overshoot* dan waktu *settling* [42].

2.10.5. Metode Tuning: Ziegler-Nichols

Metode ini adalah pendekatan eksperimental untuk mengatur kendali PID agar sistem dapat mencapai performa yang baik. Metode *Ziegler-Nichols* merupakan metode yang sangat berguna sebagai titik acuan awal dalam penyetelan parameter PID. Walaupun demikian, penyesuaian lebih lanjut sering kali diperlukan untuk mencapai kinerja optimal, terutama karena metode ini cenderung menghasilkan respons sistem yang melebihi nilai tunjuk (*overshoot*) dan kurang cocok untuk sistem dengan penundaan waktu yang signifikan atau karakteristik non-linear yang kuat. *Ziegler* dan *Nichols* mengusulkan aturan untuk menentukan nilai penguatan proporsional (K_p), waktu integral (T_i) dan waktu derivatif (T_d) berdasarkan pada karakteristik tanggapan peralihan dari kendalian yang diberikan. Ada dua metode aturan penalaan *Ziegler-Nichols*, di mana keduanya diarahkan untuk mendapatkan *overshoot* maksimum 25% dengan masukan undak [43]. Lihat Gambar 2. 10 berikut :



Sumber : [43]

Gambar 2. 10. Kurva Respons Dengan *Overshoot* Maksimum 25%

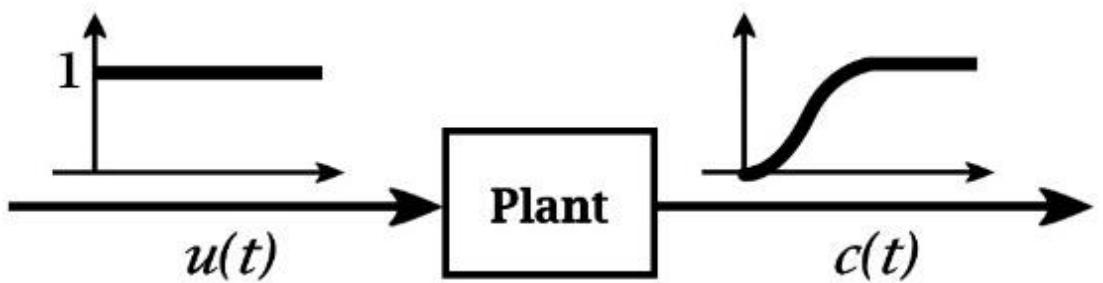
2.10.6. Pembagian Metode Tuning Ziegler-Nichols

Ada dua macam metode *Tuning Ziegler-Nichols*, yaitu :

a. ***Ziegler-Nichols* Metode Pertama**

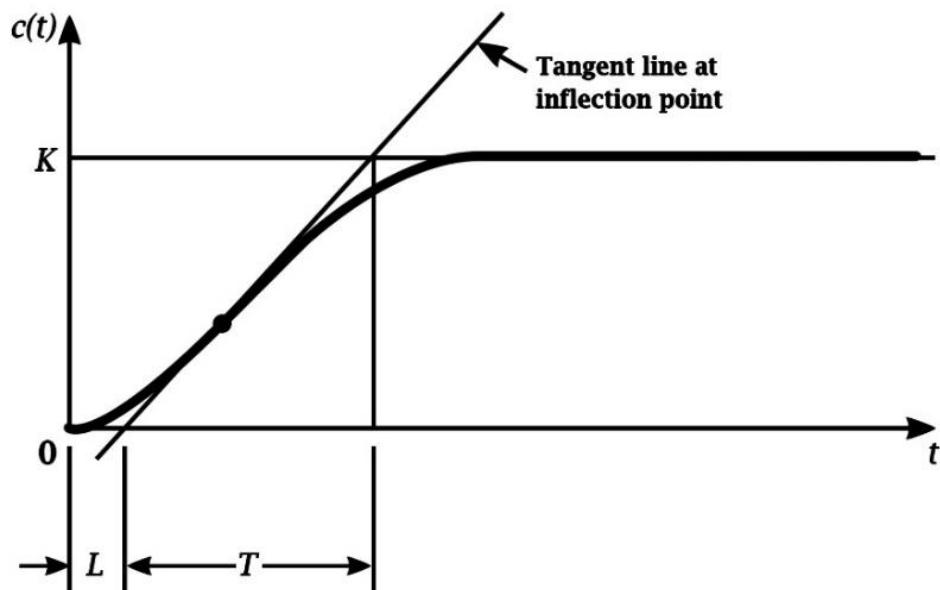
Metode *tuning* yang pertama menggunakan sinyal *step* sebagai masukan, kemudian dengan percobaan secara eksperimen mengamati respons sistem seperti yang ditunjukkan pada Gambar 2. 11. Jika sistem bukan merupakan sistem dengan *integrator* atau memiliki *pole* dominan, maka respons sistem akan berbentuk seperti kurva S seperti yang ditunjukkan pada Gambar 2. 12. Metode ini dapat diterapkan jika respons masukan sinyal

step itu membentuk tampilan kurva S. Kurva respons *step* ini dapat dihasilkan secara eksperimen atau simulasi *dynamic system* [18], [19].



Sumber : [18]

Gambar 2. 11. Respons Sistem Terhadap Masukan Fungsi Step



Sumber : [18]

Gambar 2. 12. Respons Sistem Berbentuk Kurva S

Berdasarkan kurva berbentuk S, dapat ditandai dengan dua buah konstanta parameter, yaitu parameter waktu tunda (L) dan konstanta waktu (T). Dua konstanta tersebut ditentukan dengan cara menggambar garis kemiringan yang bersinggungan dengan sumbu x dan garis K seperti yang ditunjukkan pada Gambar 2. 12. Fungsi alih $C_{(s)}/U_{(s)}$ dapat diperkirakan dengan sistem orde 1 sebagai berikut [18], [19].

$$\frac{du(t)}{du} = k_i e(t) \quad (2.1)$$

Tabel 2. 1. Formula Ziegler-Nichols Metode 1

Pengendali	K_p	T_i	T_d
P	$\frac{T}{L}$	∞	0
PI	$0.9 \frac{T}{L}$	$\frac{T}{0.3}$	0
PID	$1.2 \frac{T}{L}$	$2L$	$0.5L$

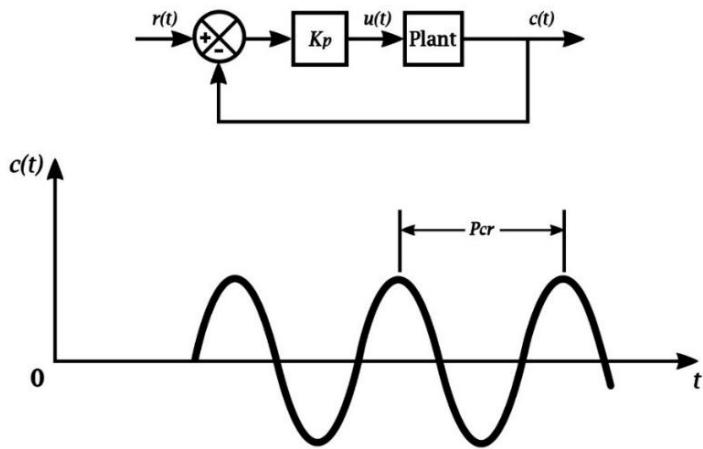
Mengasumsikan bahwa pengendali PID di *tuning* menggunakan metode pertama Ziegler-Nichols, memberikan [18], [19] :

$$\begin{aligned}
 G_c(s) &= K_p \left(1 + \frac{1}{T_i s} + T_d s \right) \\
 &= 1.2 \frac{T}{L} \left(1 + \frac{1}{2Ls} + 0.5Ls \right) \\
 &= 0.6T \frac{(s + \frac{1}{L})^2}{s}
 \end{aligned} \quad (2.2)$$

Jadi, kendali PID memiliki sebuah *pole* di *origin* dan *zero* pada $s = -1/L$.

b. Ziegler-Nichols Metode Kedua

Pada metode Ziegler-Nichols kedua, nilai parameter T_i dan parameter T_d perlu diatur ulang sesuai dengan ketentuan berikut: $T_i = 0$ (integral tidak aktif) dan $T_d = 0$ (derivatif tidak aktif), sehingga hanya aksi kendali proporsional (K_p) yang digunakan (lihat Gambar 2.13). Nilai K_p kemudian dinaikkan secara bertahap, mulai dari nol hingga mencapai nilai *Ultimate Gain* (K_u), hingga sistem menunjukkan osilasi kontinu pada keluarannya. Apabila sistem tidak menunjukkan osilasi kontinu untuk nilai K_p berapa pun, maka metode ini tidak dapat diterapkan [18], [19].



Sumber : [18]

Gambar 2. 13. Sistem Loop Tertutup dengan Pengendali Proporsional

Jadi, nilai *Ultimate Gain* (K_u) dan *Ultimate Period* (P_u) ditentukan secara eksperimen. *Ziegler* dan *Nichols* merekomendasikan bahwa nilai parameter K_p , T_i , dan T_d ditentukan berdasarkan formula yang ditunjukkan pada Tabel di bawah ini [18], [19].

Tabel 2. 2. Formula Ziegler-Nichols Metode 2

Pengendali	K_p	T_i	T_d
P	$0.5K_u$	∞	0
PI	$0.45K_u$	$\frac{1}{1.2}P_u$	0
PID	$0.6K_u$	$0.5P_u$	$0.125P_u$

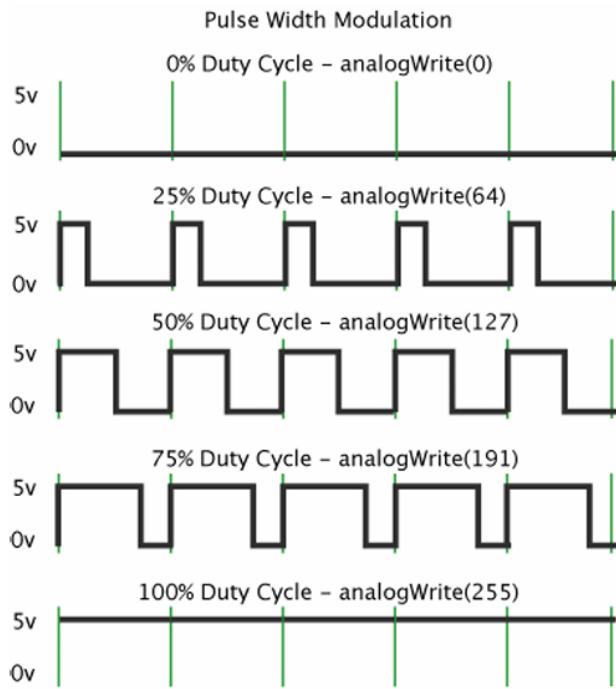
Mengasumsikan pengendali PID yang di *tuning* menggunakan metode kedua *Ziegler-Nichols*, memberikan [18], [19] :

$$\begin{aligned}
 G_c(s) &= K_p \left(1 + \frac{1}{T_i s} + T_d s \right) \\
 &= 0.6K_u \left(1 + \frac{1}{0.5P_u s} + 0.125P_u s \right) \\
 &= 0.075K_u P_u \frac{(s + \frac{4}{P_u})^2}{s}
 \end{aligned} \tag{2. 3}$$

Jadi, pengendali PID memiliki *pole* di *origin* dan dua buah *zero* di $s = -4/P_u$.

2.11. PWM (*Pulse Width Modulation*)

PWM adalah metode yang digunakan untuk membuat sinyal digital menyerupai sinyal analog, di mana sinyal berbentuk gelombang kotak dengan lebar pulsa yang dapat diatur sesuai kebutuhan melalui penyesuaian *duty cycle* [44]. Implementasi PWM biasanya digunakan pada pengendalian kecepatan motor DC, pengendalian motor *servo*, dan pengaturan nyala terang LED.



Sumber : Kelas IoT x NusaBot

Gambar 2. 14. Manipulasi PWM

2.12. IoT (*Internet of Things*)

IoT merupakan istilah yang menggambarkan konsep internet di masa depan, di mana berbagai objek fisik dapat saling berkomunikasi melalui jaringan internet. Namun, istilah “*things*” dalam konteks IoT belum memiliki definisi standar yang sepenuhnya jelas [45].

Dalam IoT, protokol adalah seperangkat aturan dan standar yang memungkinkan perangkat, sensor, *gateway*, dan platform untuk berkomunikasi dan bertukar data satu sama lain. Tanpa protokol, perangkat IoT tidak akan bisa memahami satu sama lain. Salah satu jenis protokol yang umum digunakan dalam IoT adalah MQTT (*Message Queue Telemetry Transport*).

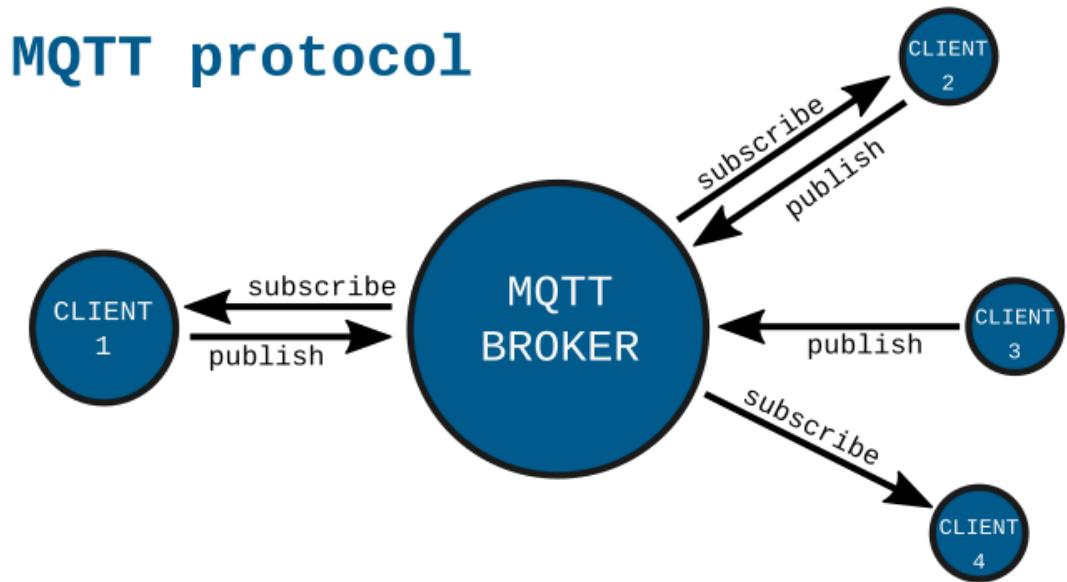
MQTT merupakan protokol komunikasi ringan berbasis *publish-subscribe* yang berjalan di atas protokol TCP / IP. Protokol ini dirancang dengan *overhead* data yang sangat kecil (minimal hanya 2 byte), serta hemat daya, menjadikannya efisien untuk perangkat dengan keterbatasan

sumber daya. MQTT bersifat *open source*, sederhana, dan mudah untuk diimplementasikan, serta mampu menangani ribuan klien jarak jauh hanya dengan satu server. Karakteristik ini menjadikannya sangat cocok untuk digunakan dalam berbagai kondisi, terutama pada sistem dengan keterbatasan jaringan seperti komunikasi antar mesin (*Machine to Machine / M2M*) dan aplikasi IoT yang membutuhkan jejak kode (*code footprint*) kecil. Dalam skema *publish-subscribe*, diperlukan *broker*, yang berfungsi mendistribusikan pesan ke klien (orang yang berlangganan) sesuai topik tertentu [46].

Karakteristik Utama MQTT [47] :

a. **Arsitektur: Publish / Subscribe**

- *Publisher*: Mengirimkan pesan ke topik tertentu tanpa mengetahui siapa yang menerima.
- *Subscriber*: Berlangganan topik tertentu untuk menerima pesan.
- *Broker*: Bertindak sebagai perantara yang mengelola semua pesan dan mendistribusikannya ke *subscriber* yang relevan.



Sumber : robotyka.net

Gambar 2. 15. Pengiriman Data MQTT

b. **Ringan dan Efisien**

MQTT menggunakan sedikit *bandwidth* untuk perangkat IoT yang sering memiliki konektivitas terbatas dan sumber daya komputasi rendah.

c. QoS (*Quality of Service*)

MQTT mendukung tiga level QoS untuk memastikan keandalan dalam pengiriman pesan yaitu diantaranya meliputi :

- QoS 0 (*At most once*): Pesan dikirim sekali tanpa jaminan pengiriman.
- QoS 1 (*At least once*): Pesan dijamin terkirim setidaknya sekali, tetapi bisa diterima lebih dari sekali.
- QoS 2 (*Exactly once*): Pesan dijamin terkirim tepat satu kali.

d. Penyimpanan Pesan (*Retained Messages*)

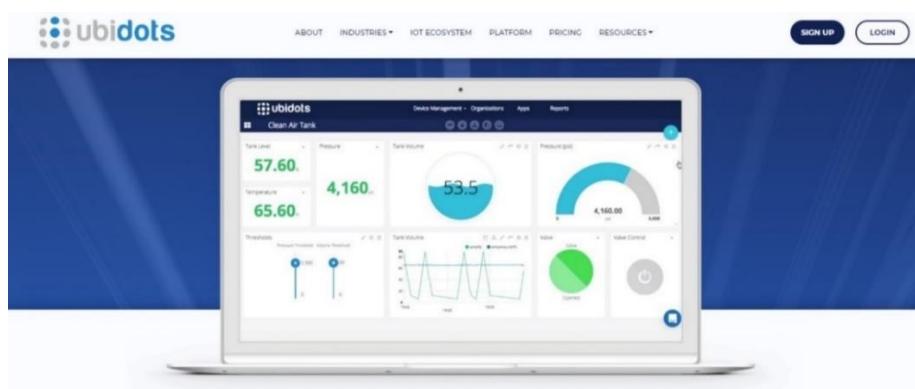
Fitur ini memungkinkan *broker* menyimpan pesan terakhir dari suatu topik dan mengirimkannya kembali kepada klien yang baru saja melakukan *subscribe*, meskipun klien tersebut belum terhubung saat pesan awal dikirimkan.

e. *Last Will and Testament* (LWT)

Fitur ini memungkinkan pemberitahuan otomatis ke *subscriber* jika suatu perangkat tiba-tiba terputus dari jaringan.

2.13. IoT Platform: Ubidots

Ubidots merupakan platform yang ditujukan bagi para pengembang *Internet of Things* (IoT), yang menyediakan layanan, baik gratis maupun berbayar untuk membantu pengguna merancang dan membangun aplikasi IoT dengan cepat, tanpa perlu menulis kode pemrograman, atau menyewa jasa pengembang perangkat lunak. Platform ini dirancang untuk memudahkan *developer* dalam menciptakan aplikasi IoT secara mandiri [48].



Sumber : kmtech.id

Gambar 2. 16. Tampilan IoT Platform: Ubidots

Fitur Utama Ubidots [49] :

a. Pengumpulan Data

Ubidots memungkinkan pengumpulan data dari berbagai sensor dan perangkat IoT melalui protokol komunikasi seperti HTTP, MQTT, TCP / IP, UDP, dan lainnya.

b. Visualisasi Data

Platform ini menyediakan *dashboard* yang dapat disesuaikan, di mana pengguna dapat membuat grafik, peta, dan tabel untuk memvisualisasikan data yang dikumpulkan.

c. Analisis Data

Ubidots memungkinkan pengguna untuk menerapkan logika bisnis dan analisis data, termasuk penerapan algoritma dan aturan untuk mengotomatisasi respons terhadap data tertentu.

d. Notifikasi dan Automasi

Pengguna dapat mengatur notifikasi berbasis data untuk mendapatkan peringatan atau mengambil tindakan otomatis ketika kondisi tertentu tercapai.

e. API terbuka

Ubidots menyediakan API yang memungkinkan pengembang untuk mengintegrasikan data dari platform ini ke dalam aplikasi atau sistem lain.

2.14. Diagram Blok

Diagram Blok digunakan untuk menampilkan suatu sistem secara lengkap, di mana fungsi dari setiap komponen masih tampak jelas. Setiap komponen digambarkan dengan sebuah blok (kotak) yang memiliki masukan dan keluaran, sesuai dengan masukan dan keluaran dari komponen yang diwakili [43].

Di dalam blok tersebut, dituliskan fungsi alih dari komponen tersebut. Hubungan antara beberapa blok dapat menggambarkan keseluruhan sistem. Dalam pembuatan diagram blok, beberapa blok dapat digabungkan untuk penyederhanaan, tetapi sifat asli dari sistem harus tetap dipertahankan.

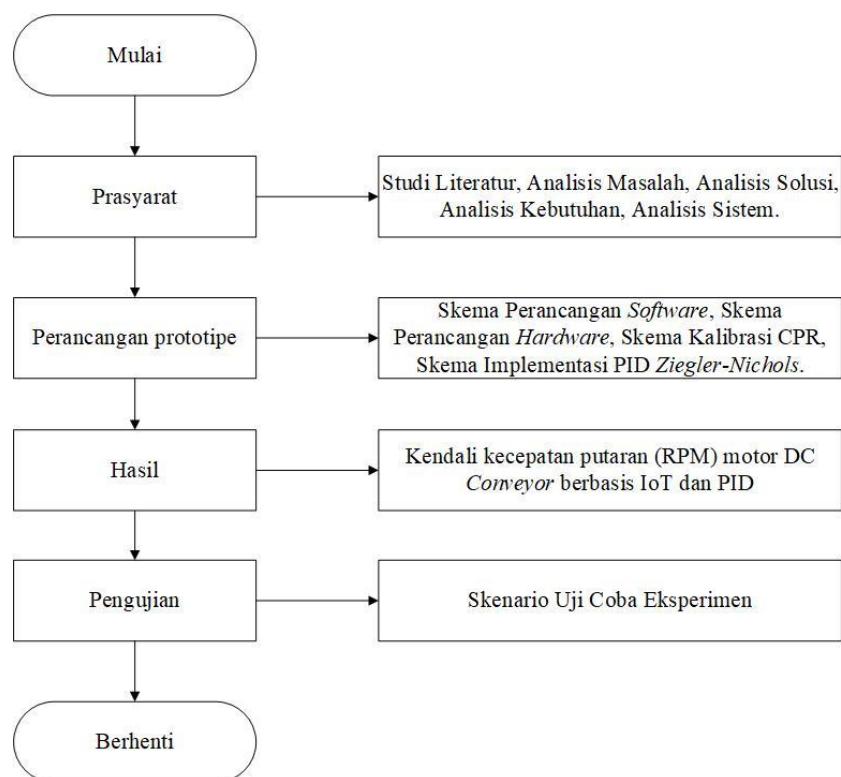
2.15. Diagram *Wiring*

Diagram *wiring* merupakan gambaran visual dari sambungan kabel antar komponen yang saling terhubung secara fisik untuk membentuk suatu rangkaian [50]. Dalam diagram ini, kabel dan koneksi antar komponen, seperti *push button*, motor DC, *motor driver* L298N, atau perangkat elektronik lainnya itu dapat ditunjukkan dengan menggunakan garis-garis berwarna. Tujuan utama dari diagram *wiring* adalah untuk memudahkan instalasi, perbaikan, dan pemeliharaan sistem listrik atau elektronik dengan memberikan panduan yang jelas mengenai bagaimana kabel harus disambungkan.

BAB III

METODOLOGI PENELITIAN

Penelitian ini menggunakan metodologi yang bernama *Rapid Application Development* (RAD). Metodologi ini terdiri dari empat tahap pelaksanaan, yaitu prasyarat (*requirement*), perancangan prototipe (*prototyping*), hasil (*output*), dan pengujian (*testing*). Proses pelaksanaannya dapat digambarkan melalui *flowchart*, seperti yang terlihat pada Gambar 3. 1.



Gambar 3. 1. Tahapan Penelitian

Gambar 3. 1 merupakan tahapan-tahapan yang dilakukan oleh penulis dalam penelitian ini yang secara eksplisit dibahas sebagai berikut.

3.1. Prasyarat (*Requirement*)

Tahapan prasyarat (*requirement*) penelitian merupakan fondasi utama sebelum memulai perancangan sistem. Kegiatan seperti studi literatur, analisis masalah, analisis solusi, analisis kebutuhan, dan analisis sistem merupakan langkah-langkah yang harus dipenuhi agar penelitian memiliki tujuan yang jelas dan terstruktur.

3.1.1. Studi Literatur

Studi literatur ini untuk mengumpulkan data dan informasi dari berbagai sumber pustaka yang kredibel. Sumber-sumber tersebut meliputi tesis, disertasi, artikel ilmiah, buku, serta publikasi relevan lainnya [24].

3.1.2. Analisis Masalah

Motor DC *Conveyor* memiliki peran penting dalam industri, yaitu untuk memindahkan material secara efisien dari satu titik ke titik lainnya. Kecepatan yang stabil dan terkontrol sangat dibutuhkan agar tidak merusak produk dan tetap menjaga efisiensi dalam produksi. Namun, pada penelitian sebelumnya masih ditemukan beberapa kendala teknis, seperti kurang tepatnya pemilihan mikrokontroler, implementasi kendali PID yang belum optimal, serta belum maksimalnya pemanfaatan kendali jarak jauh. Selain itu, integrasi antar komponen sistem seperti sensor, aktuator, dan platform kontrol masih belum sinkron dan responsif. Kondisi ini menghambat fleksibilitas dan keandalan sistem secara keseluruhan. Oleh karena itu, diperlukan penelitian lanjutan yang fokus pada peningkatan mikrokontroler, penerapan kendali PID yang lebih adaptif, serta integrasi IoT untuk *monitoring* dan kendali *real-time* agar sistem *Conveyor* lebih efisien, fleksibel, dan andal dalam mendukung otomasi industri.

3.1.3. Analisis Solusi

Pembuatan sistem kendali kecepatan pada motor DC *Conveyor* menggunakan metode PID *Ziegler-Nichols* berbasis IoT, ini dirancang sebagai solusi atas permasalahan yang ada. Inovasinya mencakup penggunaan ESP32 untuk mengatur tombol fisik *On / Off*, arah putaran, dan *setpoint RPM* melalui potensiometer, serta pemanfaatan Ubidots sebagai platform IoT untuk kendali virtual dan pemantauan kecepatan putaran (RPM) motor DC secara *real-time*. Metode PID *Ziegler-Nichols* digunakan untuk menstabilkan kecepatan putaran (RPM) motor DC pada *Conveyor* secara optimal.

3.1.4. Analisis Kebutuhan

Analisis kebutuhan ini diperoleh melalui pengamatan langsung terhadap permasalahan yang ingin dipecahkan dan solusi yang telah ada. Hasil analisis menunjukkan adanya tiga jenis kebutuhan utama, yaitu kebutuhan perangkat keras (*hardware*), kebutuhan perangkat lunak (*software*), serta kebutuhan bahan dan alat. Daftar perangkat keras (*hardware*) yang digunakan dalam penelitian ini dapat Anda ketahui pada Tabel 3. 1.

Tabel 3. 1. Kebutuhan Perangkat Keras

No	Nama Komponen Elektronik	Penggunaan Dalam Riset
1.	ESP32 DOIT DevKit-C V1	Papan pengembangan Mikrokontroler (<i>Development Board</i>).
2.	Motor DC JGA25-370 12V <i>Encoder</i>	Alat penggerak (aktuator) yang dilengkapi dengan sensor penghitung rotasi.
3.	<i>Motor Driver</i> L298N	Modul pengendali arus dan arah putaran Motor DC.
4.	LCD I2C	Layar penampil informasi.
5.	Potensiometer	Pengatur nilai resistansi (hambatan) secara manual yang digunakan sebagai masukan (<i>input</i>).
6.	<i>Adaptor</i> 12V 2A	Catu daya untuk Motor DC.
7.	<i>Breadboard</i>	Media untuk menyusun rangkaian elektronik sementara tanpa perlu menyolder.
8.	<i>Adaptor</i> 5V 2A	Catu daya untuk <i>board</i> ESP32.
9.	USB type C	Untuk mengunggah program (transfer data).
10.	Kabel <i>jumper</i> pita pelangi (<i>Female-Female</i> , <i>Male-Male</i> , dan <i>Female-Male</i>)	Penghubung listrik antar komponen.
11.	DC Jack <i>Female Power Adapter</i>	Konektor daya untuk adaptori eksternal.
12.	<i>Push Button</i>	Saklar mekanik yang digunakan sebagai masukan (<i>input</i>).

Kebutuhan perangkat lunak (*software*) yang ada pada penelitian ini dapat dilihat pada Tabel 3. 2 berikut.

Tabel 3. 2. Kebutuhan Perangkat Lunak

No	Nama <i>Software</i>	Keterangan
1.	Arduino IDE	Aplikasi pemrograman berbasis mikrokontroler.
2.	Ubidots	Platform IoT.
3.	CP210X <i>Driver</i>	Penghubung antara ESP32 dan komputer.
4.	Photoshop	Aplikasi pengolah gambar / desain.

Kebutuhan alat dan bahan pada penelitian ini dapat dilihat pada Tabel 3. 3 berikut.

Tabel 3. 3. Kebutuhan Alat dan Bahan

No	Nama alat dan bahan	Keterangan
1.	Triplek 50 x 50 cm	Kerangka <i>Conveyor</i> .
2.	Besi Beton <i>Stainless</i> 30 cm	AS (<i>Shaft</i>) pada Motor DC.
3.	Pipa $\frac{1}{2}$ Inch	<i>Roller</i> pada Motor DC.
4.	Lem Castol, Lem G, dan Lem Tembak	Perekat.
5.	Kain Oscar	Sabuk <i>Conveyor</i> .
6.	Ampelas <i>Grid</i> 180	Penghalus permukaan material, selain itu juga digunakan pada <i>roller</i> agar sabuk <i>Conveyor</i> tidak licin.
7.	Gergaji Besi	Alat pemotong besi, pipa, dan triplek.
8.	Penggaris	Alat ukur material.
9.	Bolpoin / Spidol	Penanda.
10.	Gunting	Alat pemotong ampelas.
11.	Mesin Bor	Alat pelubang triplek.
12.	Baut dan mur	Pengikat mekanis <i>bracket</i> atau komponen.
13.	Perekat L / <i>Bracket</i> L	Alat penyambung atau penyangga pada kerangka <i>Conveyor</i> .
14.	Obeng	Alat pemasang / pelepas baut.
15.	Tang	Perkakas multifungsi.
16.	<i>Tachometer</i> Digital	Pengukur kecepatan (RPM) Motor DC.
17.	<i>Velg</i> roda <i>Smart Car</i>	Penghubung Motor DC dan <i>roller</i> .
18.	Pisau <i>Cutter</i>	Alat pemotong kain (sabuk <i>Conveyor</i>).
19.	<i>Hairdryer</i> (pengganti <i>heatgun</i>)	Pengering lem.
20.	<i>Solder</i>	Alat pematri konduktor antar komponen elektronik.
21.	Isolasi listrik PVC	Pencegah kebocoran arus listrik.
22.	Timbangan Digital	Alat untuk mengukur massa barang yang digunakan selama pengujian.
23.	<i>Smartphone</i>	Perangkat multifungsi, yang digunakan sebagai alat dokumentasi, alat bantu dalam pengujian, dan sumber informasi.

3.1.5. Analisis Sistem

Analisis sistem ini untuk mengidentifikasi secara mendalam segala kebutuhan pengguna, baik dari segi tampilan antarmuka pengguna (GUI) maupun aspek fungsional lainnya. Proses analisis yang cermat ini bertujuan untuk menghasilkan produk yang tidak hanya memenuhi ekspektasi pengguna, tetapi juga sesuai dengan konteks penggunaannya. Berdasarkan hasil tinjauan penulis, sistem yang dibuat harus mampu melakukan kendali kecepatan, kendali *On / Off*, kendali arah putaran, dan mampu memantau kecepatan putaran (RPM) dari motor DC *Conveyor*.

3.2. Perancangan Prototipe (*Prototyping*)

Prototyping adalah tahapan yang fokusnya lebih mengarah ke cara mendesain suatu produk. Perancangan prototipe yang dikerjakan oleh penulis yaitu meliputi: skema perancangan *software*, skema perancangan *hardware*, skema kalibrasi CPR, dan skema implementasi PID *Ziegler-Nichols*.

3.2.1. Skema Perancangan *Software*

Perangkat lunak dalam penelitian ini dirancang menggunakan Arduino IDE (*Integrated Development Environment*) dengan bahasa Arduino (C/C++). Untuk mendukung proses pemrograman ke *hardware*, diperlukan juga USB *driver* dan *board* ESP32, agar program dapat diunggah dan dijalankan dengan baik.

3.2.1.1. Skenario Sistem

Skenario sistem yang ada pada penelitian ini dibagi menjadi 2 bagian yaitu skenario otomatisasi sistem dan skenario kendali sistem, yang dapat Anda lihat pada Tabel 3. 4 dan Tabel 3. 5 berikut.

Tabel 3. 4. Skenario Otomatisasi Sistem

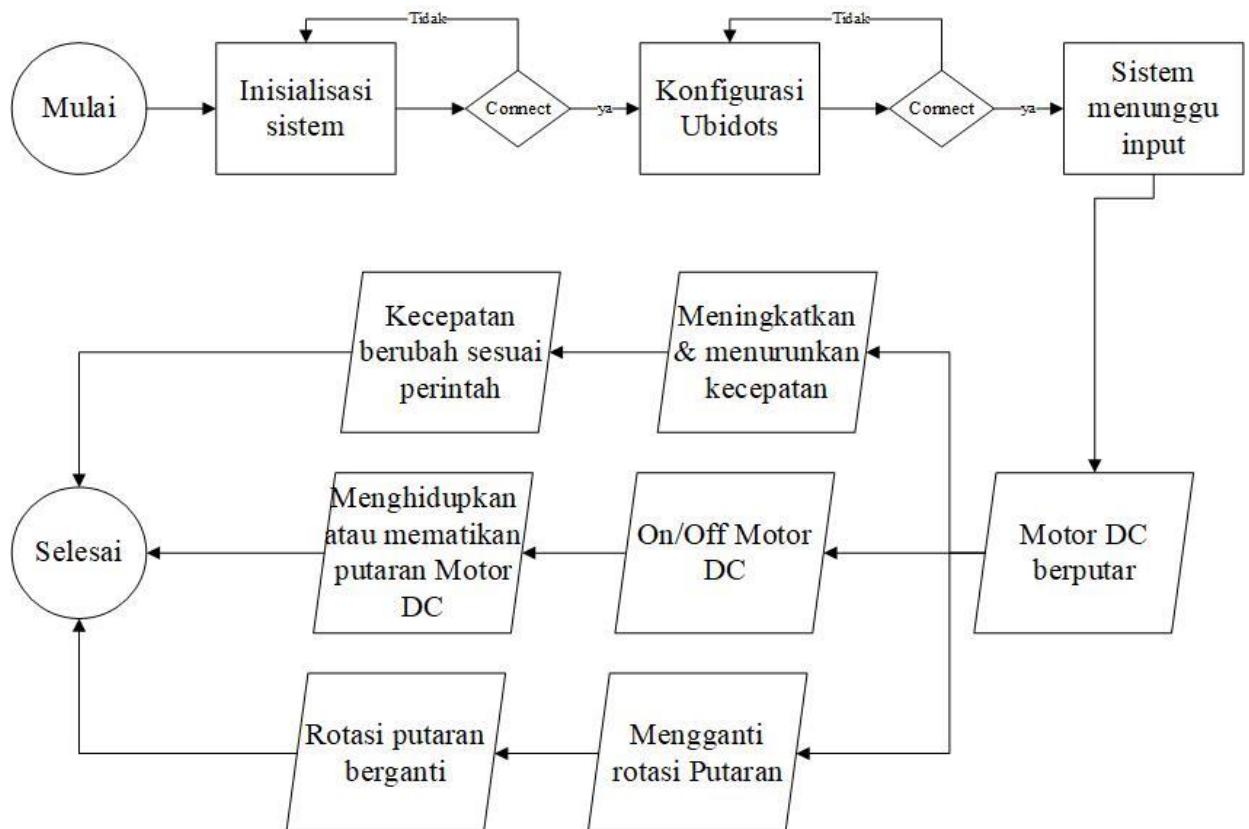
No.	Nama Proses	Respons	
		Sukses	Gagal
1.	Konfigurasi <i>Wi-Fi</i>	Sistem menyambung ke <i>Wi-Fi</i> .	<i>Reconnect</i> ke <i>Wi-Fi</i> .
2.	Konfigurasi IoT	Sistem menyambung ke server IoT (Ubidots).	<i>Reconnect</i> ke server IoT (Ubidots).

Tabel 3. 5. Skenario Kendali Sistem

No.	Nama Proses	Respons
1.	Meningkatkan kecepatan <i>Conveyor</i>	Kecepatan <i>Conveyor</i> naik, LCD menampilkan jumlah kecepatan yang meningkat.
2.	Menurunkan kecepatan <i>Conveyor</i>	Kecepatan <i>Conveyor</i> turun, LCD menampilkan jumlah kecepatan yang menurun.
3.	Mengganti arah rotasi putar <i>Conveyor</i>	<i>Conveyor</i> berputar ke arah yang berlawanan.
4.	Saklar <i>On / Off Conveyor</i>	Menghidupkan atau mematikan <i>Conveyor</i> .

3.2.1.2. Workflow Sistem

Workflow sistem ini dirancang khusus oleh penulis dengan menyesuaikan kondisi yang ada di lapangan. Workflow sistem dapat dilihat pada Gambar 3. 2.



Gambar 3. 2. Workflow Sistem

Gambar 3. 2 menggambarkan proses kerja dari sistem kendali kecepatan motor DC berbasis IoT, proses dimulai dari tahap inisialisasi sistem, di mana perangkat melakukan persiapan awal. Setelah inisialisasi, sistem memeriksa koneksi jaringan. Jika koneksi tidak berhasil, sistem akan kembali ke tahap inisialisasi untuk mencoba kembali. Namun, jika koneksi berhasil, sistem melanjutkan ke tahap konfigurasi Ubidots.

Pada tahap ini, sistem mengatur parameter dan konektivitas agar dapat berkomunikasi dengan platform Ubidots secara efektif. Setelah konfigurasi selesai, sistem kembali melakukan pengecekan koneksi. Jika koneksi ke Ubidots berhasil, maka sistem akan masuk ke kondisi “menunggu *input*”, yaitu menunggu perintah dari pengguna. Begitu sistem menerima *input*, motor DC akan mulai berputar dan pengguna dapat melakukan tiga jenis kendali: mengatur kecepatan, menyalakan atau mematikan motor, dan mengganti arah rotasi putaran motor.

Pada kendali kecepatan, pengguna dapat memberikan perintah untuk meningkatkan atau menurunkan kecepatan motor, dan sistem akan menyesuaikan kecepatan sesuai dengan perintah tersebut. Pada kendali *On / Off*, sistem dapat menghentikan putaran motor atau memutarnya kembali berdasarkan perintah pengguna. Sedangkan pada kendali arah putaran, sistem dapat membalik arah rotasi motor (misalnya dari searah jarum jam menjadi berlawanan arah jarum jam).

Semua fungsi ini bekerja secara *real-time* tergantung pada *input* yang dikirim baik secara manual atau melalui Ubidots. Setelah aksi dilakukan dan motor merespons sesuai perintah, proses dapat dianggap selesai atau sistem akan kembali menunggu *input* berikutnya untuk siklus kendali yang baru dan dilakukan secara terus-menerus selagi sistem masih menyala.

3.2.2. Skema Perancangan *Hardware*

Pada penelitian ini, *hardware* dirancang untuk menghubungkan mikrokontroler ESP32 dengan berbagai perangkat seperti Motor DC JGA25-370, Potensiometer, *Push Button*, *Motor Driver* L298N, serta LCD.

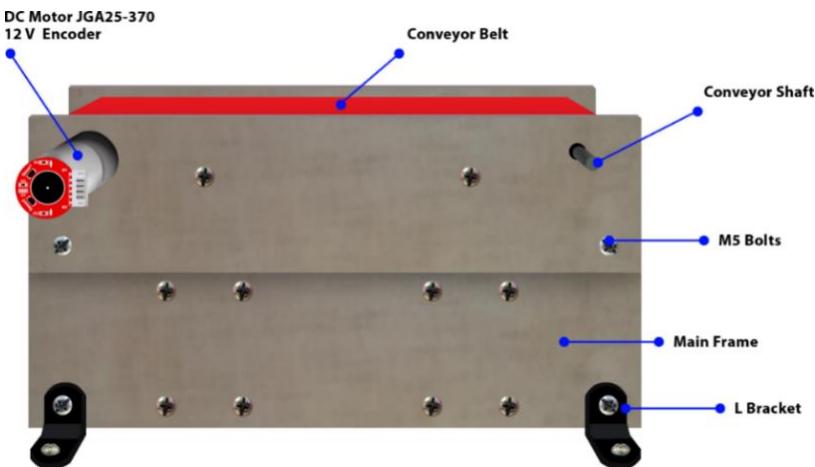
3.2.2.1. Pembuatan Kerangka *Conveyor*

Untuk membuat kerangka *Conveyor*, adapun langkah-langkah yang harus dilalui, yaitu dapat dilihat pada Tabel 3. 6.

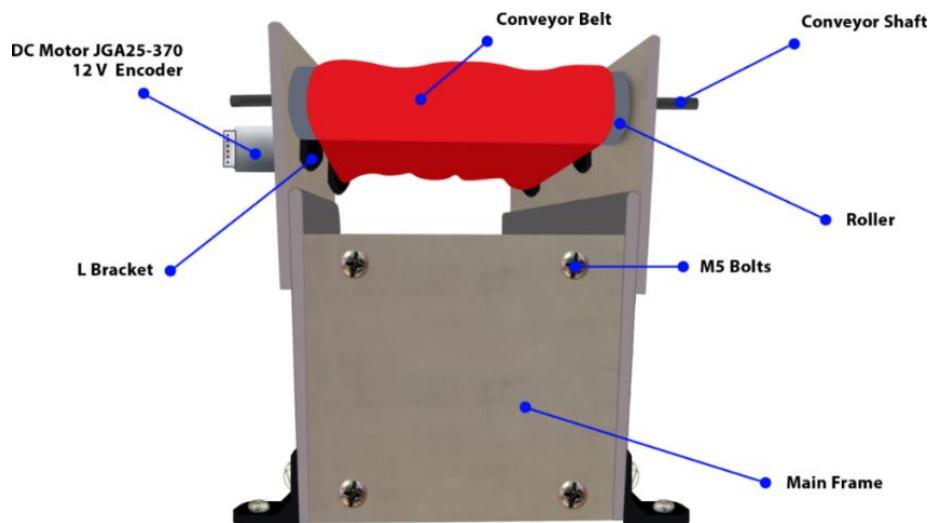
Tabel 3. 6. Perancangan Conveyor

No.	Langkah – Langkah
1.	Persiapkan alat dan bahan yang diperlukan.
2.	Ukur triplek pada bagian yang akan dipotong.
3.	Potong triplek menggunakan gergaji sesuai ukuran yang sudah ditentukan.
4.	Haluskan permukaan triplek menggunakan ampelas.
5.	Tandai semua komponen yang sudah diukur.
6.	Lubangi bagian-bagian tertentu yang ada di triplek berdasarkan tanda yang telah diberikan menggunakan mesin bor sebagai tempat baut.
7.	Satukan triplek dengan <i>bracket L</i> , baut, dan mur sehingga membentuk kerangka <i>Conveyor</i> .
8.	Siapkan pipa dan besi beton <i>stainless</i> , lalu ukur dan potong sesuai kebutuhan.
9.	Siapkan <i>Velg</i> roda <i>Smart Car</i> , lalu ambil bagian tengahnya saja dengan cara dipotong.
10.	Rekatkan bagian Tengah roda tersebut pada pipa menggunakan lem castol dan lem tembak. Hal ini hanya berlaku pada satu sisi <i>roller</i> penggerak.
11.	Pasangkan triplek yang sudah dipotong bulat pada masing-masing pipa.
12.	Pasang besi <i>stainless</i> di tengah-tengahnya sebagai AS (<i>Shaft</i>).
13.	Pasang ampelas yang sudah dipotong pada sisi luar masing-masing <i>roller</i> .
14.	Pasang <i>roller</i> yang sudah jadi pada kerangka <i>Conveyor</i> .
15.	Potong kain oscar untuk menjadi sabuk <i>Conveyor</i> .
16.	Pasang kain oscar yang sudah dipotong ke <i>roller</i> .
17.	Rekatkan seluruh komponen pada papan panel.

Gambaran detail dari sistem *Conveyor* dapat dilihat pada Gambar 3. 3 dan 3. 4 berikut.



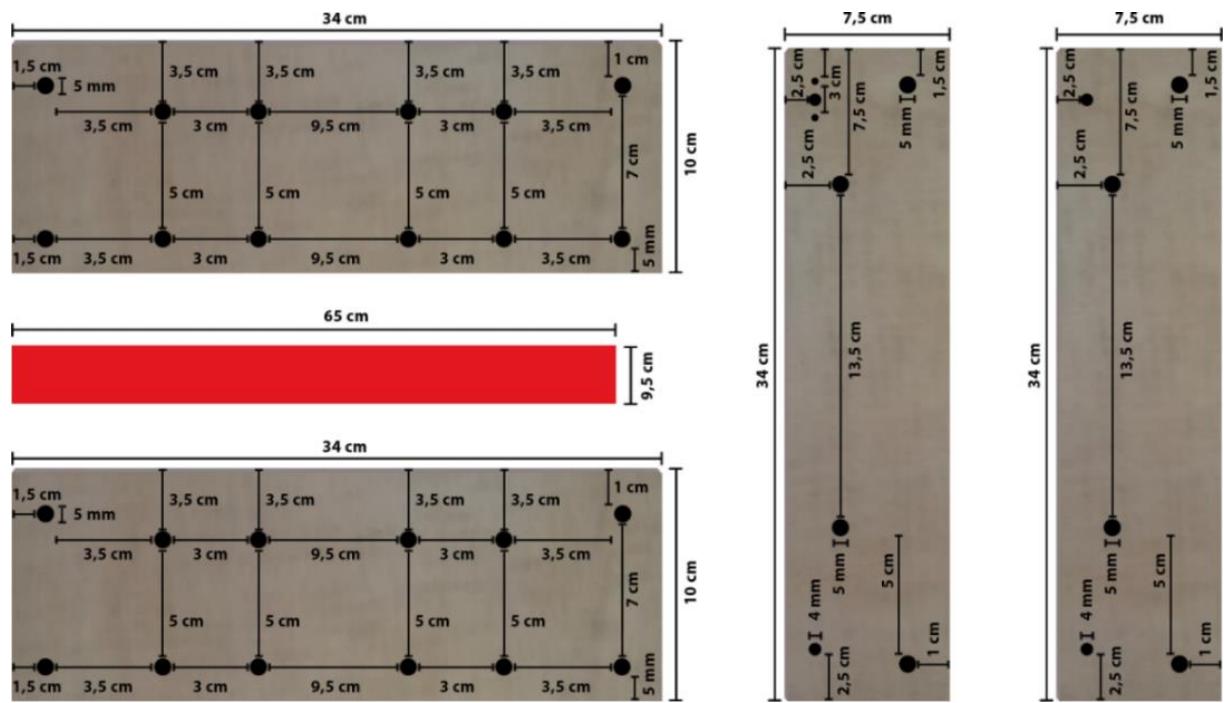
Gambar 3. 3. Perancangan Conveyor Tampak Samping



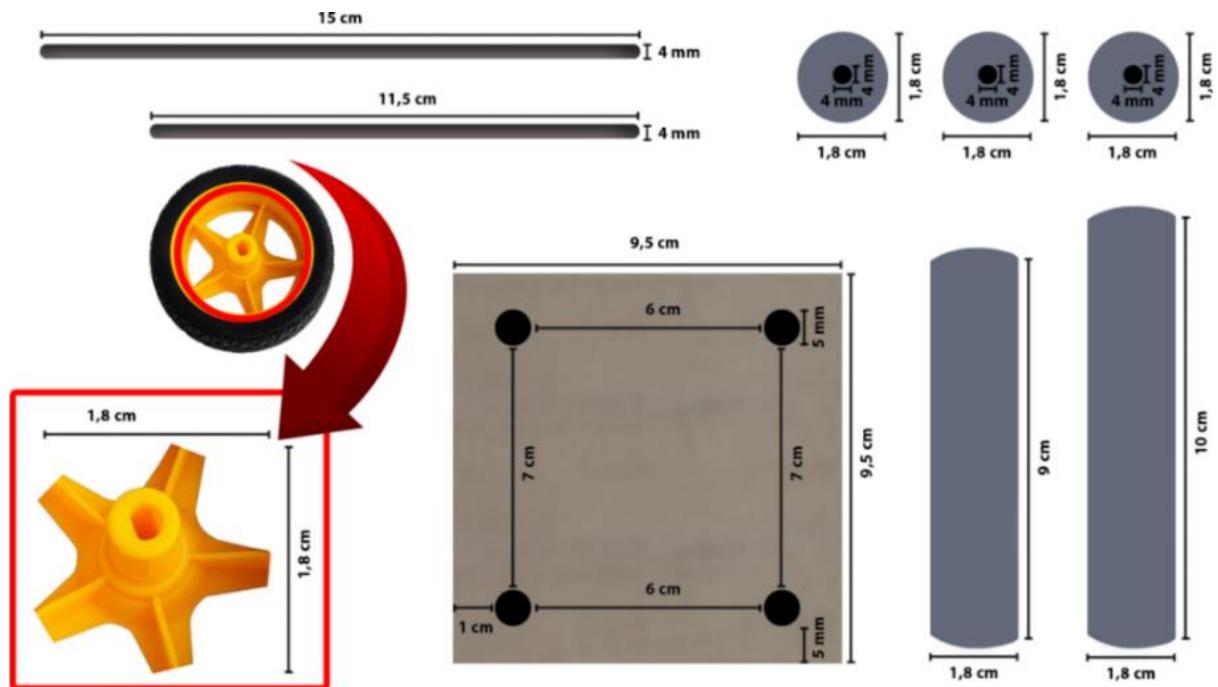
Gambar 3. 4. Perancangan *Conveyor* Tampak Depan

Gambar di atas menampilkan tampak samping dan tampak depan dari sebuah sistem *Conveyor mini* yang terdiri dari beberapa komponen mekanik dan elektrik utama. Pada bagian atas sistem terdapat sabuk *Conveyor* berwarna merah yang berfungsi untuk memindahkan benda dari satu titik ke titik lainnya secara kontinu. Sabuk ini digerakkan oleh motor DC JGA25-370 12V melalui *roller*, yang dilengkapi dengan *Magnetic Encoder* 3V di sisi belakang motor DC tersebut. *Encoder* ini merupakan perpaduan antara sensor *Hall Effect* dan magnet, yang berfungsi untuk mendeteksi jumlah rotasi motor secara akurat. Hampir seluruh sistem dirakit menggunakan baut M5 (ulir berdiameter 5mm), namun ada juga yang menggunakan baut M3 (ulir berdiameter 3mm) sebanyak 2 buah pada triplek yang berkenaan langsung dengan kepala motor DC. Hal ini dapat memastikan struktur tetap kokoh dan stabil.

Bagian utama dari struktur sistem *Conveyor* ini disebut *main frame*, sedangkan bagian dudukan komponen atau dudukan *main frame* disebut papan panel. Adapun di bagian bawah, terdapat *L bracket* yang berfungsi sebagai penopang atau kaki sistem, membantu menstabilkan posisi *Conveyor* di atas papan panel. Dalam Tugas Akhir ini, triplek digunakan sebagai bahan utama konstruksi, karena memberikan beberapa keuntungan, seperti kemudahan dalam perakitan, bobot yang ringan, serta biaya yang lebih rendah dibandingkan dengan material logam. Selain itu, triplek cukup kuat untuk mendukung komponen seperti motor, poros, dan sabuk *Conveyor* pada aplikasi ringan hingga menengah. Namun, karena sifat dasarnya yang kurang tahan terhadap kelembaban dan benturan keras, diperlukan pelapisan pelindung atau perkuatan tambahan jika sistem ini digunakan dalam lingkungan yang lebih menantang. Secara keseluruhan, sistem ini dirancang secara efisien dan cocok digunakan untuk keperluan pendidikan, penelitian, atau prototipe dalam skala kecil.

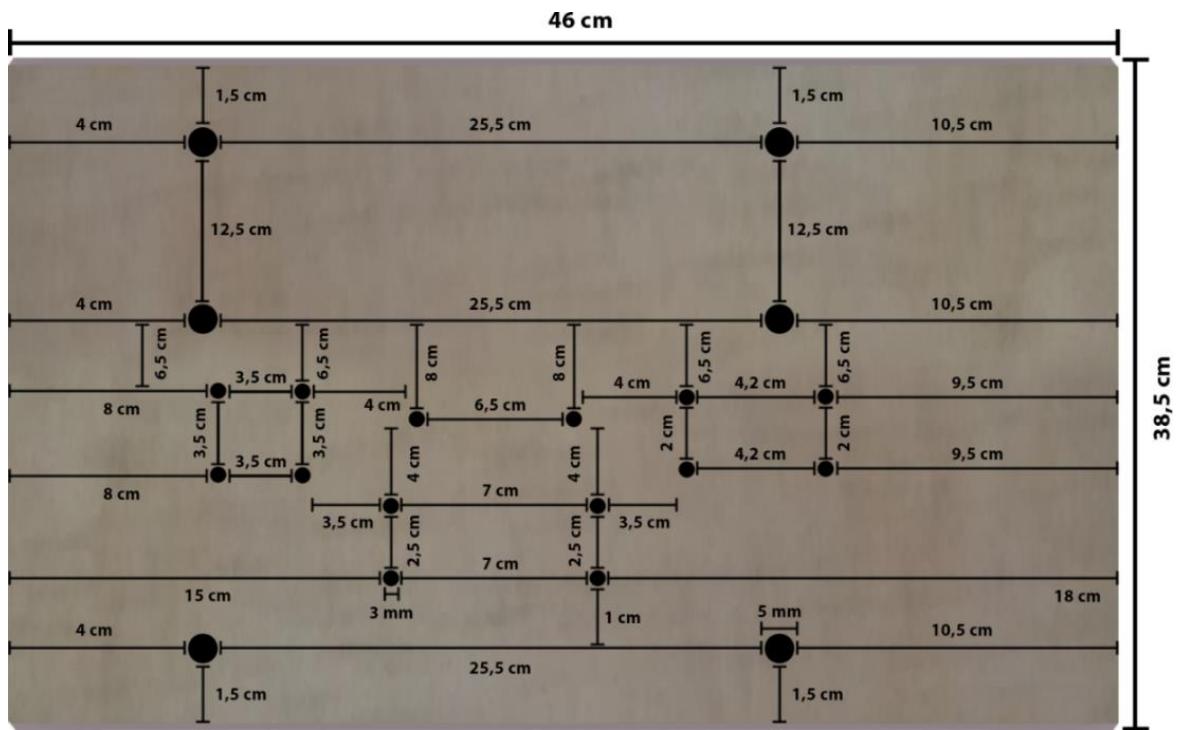


Gambar 3. 5. *Blueprint Kerangka Bagian Samping dan Belt Conveyor*



Gambar 3. 6. *Blueprint Kerangka Bagian Depan dan Roller Conveyor*

Pada Gambar 3. 5 dan Gambar 3. 6 menunjukkan dimensi kerangka *Conveyor mini*, termasuk tampak samping, tampak depan, *belt conveyor*, dan *roller conveyor*, yang digunakan sebagai panduan dalam proses perakitan sistem secara keseluruhan.

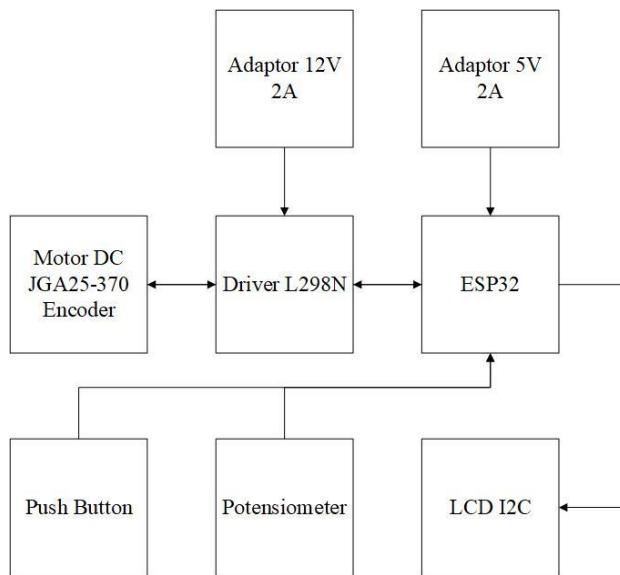


Gambar 3. 7. Blueprint Papan Panel

Pada Gambar 3. 7 dijelaskan juga *blueprint* dari dudukan kerangka dan komponen *Conveyor*, yang semakin mempermudah dalam proses perakitan kerangka ataupun komponen.

3.2.2.2. Block Diagram

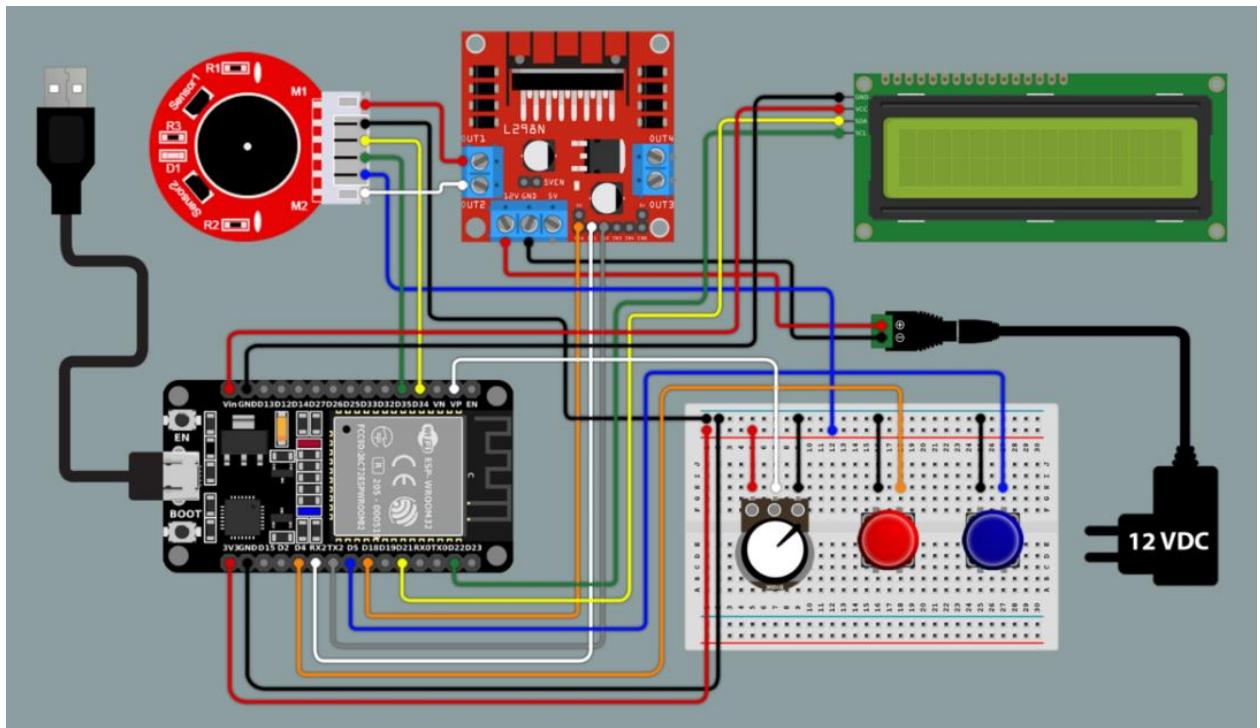
Diagram Blok dari keseluruhan sistem *Conveyor* telah diatur sedemikian rupa sesuai dengan kebutuhan, hal ini seperti yang ditunjukkan pada Gambar 3. 8 berikut.



Gambar 3. 8. Diagram Blok Perancangan Hardware

3.2.2.3. Wiring Diagram

Perancangan keseluruhan kabel *hardware* dapat dilihat pada Gambar 3. 9.



Gambar 3. 9. Diagram *Wiring* Perancangan *Hardware*

Pada Gambar 3. 9 di atas, dapat diketahui bahwa sistem menggunakan modul *Motor Driver* L298N untuk mengatur arah dan kecepatan motor DC. Sumber daya pertama berasal dari *adaptor* 12V DC yang mengalirkan arus dan tegangan langsung ke *Motor Driver*, sedangkan mikrokontroler ESP32 mendapatkan suplai daya melalui koneksi USB dari komputer atau *adaptor* 5V DC. *Motor DC* terhubung ke terminal OUT1 dan 2 pada *Driver* L298N, sementara pin kontrol berada di IN1, IN2, dan ENA L298N terhubung ke pin digital ESP32 untuk mengatur arah dan kecepatan motor. Sistem juga dilengkapi dengan LCD I2C sebagai antarmuka tampilan, yang menampilkan status sistem atau data kecepatan motor.

Untuk *input* pengguna, terdapat dua buah *push button* dan satu potensiometer yang dirangkai di atas *breadboard*. Potensiometer digunakan untuk mengatur kecepatan putaran (RPM) motor DC secara manual melalui pembacaan nilai analog oleh ESP32. Tombol Merah berfungsi untuk menghidupkan atau mematikan motor DC, sedangkan Tombol Biru berfungsi untuk mengubah arah rotasi motor DC tersebut. Semua komponen *input* dan *output* terhubung ke ESP32 menggunakan kabel *jumper*, kemudian komunikasi data ke platform IoT (Ubidots) menggunakan protokol MQTT melalui koneksi *Wi-Fi*. Rangkaian ini merupakan implementasi dari sistem kendali motor berbasis IoT yang memungkinkan pengendalian motor secara lokal maupun jarak

jauh dengan kontrol kecepatan putaran (RPM), arah putaran, dan penampilan status operasional motor secara *real-time*. Mengenai detail lengkap dari pengkabelan *hardware* di atas bisa dilihat pada Tabel 3. 7 berikut.

Tabel 3. 7. Detail Pengkabelan *Hardware*

Nama <i>Hardware</i>	Kaki perangkat		Warna kabel koneksi	Perangkat yang dihubungkan
	Titik awal	Titik tujuan		
<i>Breadboard</i>	GND	GND	Hitam	<i>Encoder</i> -
				ESP32 DOIT DevKit-C V1
				Potensiometer
				<i>Push button</i> 1
				<i>Push button</i> 2
	VCC	3V3	Merah	ESP32 DOIT DevKit-C V1
		VCC		Potensiometer
		<i>Encoder</i> +	Biru	Motor DC JGA25-370 12V <i>Encoder</i>
ESP32 DOIT DevKit-C V1	Vin (5V)	VCC	Merah	LCD I2C
	GND	GND	Hitam	
	3V3	VCC	Merah	<i>Breadboard</i>
	GPIO4		Oranye	
	GPIO5		Biru	
	GPIO16	IN1	Putih	<i>Motor Driver</i> L298N
	GPIO17	IN2	Abu-abu	
	GPIO18	ENA	Oranye	
	GPIO21	SDA	Kuning	LCD I2C
	GPIO22	SCL	Hijau	
	GPIO34	<i>Encoder</i> A	Kuning	Motor DC JGA25-370 12V <i>Encoder</i>
	GPIO35	<i>Encoder</i> B	Hijau	
	GPIO36	DATA	Putih	Potensiometer
<i>Motor Driver</i> L298N	OUT1	Motor +	Merah	Motor DC JGA25-370 12V <i>Encoder</i>
	OUT2	Motor -	Putih	

	Vin (Max: 12V)	<i>Adaptor</i> +	Merah	<i>Adaptor</i> 12V
	GND	<i>Adaptor</i> -	Hitam	
	IN1	GPIO16	Putih	ESP32 DOIT DevKit-C V1
	IN2	GPIO17	Abu-abu	
	ENA	GPIO18	Oranye	

3.2.3. Skema Kalibrasi CPR

CPR (*Counts Per Revolution*) adalah jumlah total hitungan dari *encoder* dalam satu kali putaran penuh poros keluaran (*output shaft*). Nilai CPR sangat penting dalam sistem kendali kecepatan berbasis PID karena menjadi dasar perhitungan kecepatan dan posisi aktual motor. Sedangkan *Pulses Per Revolution* (PPR) adalah jumlah pulsa yang dihasilkan oleh satu *channel encoder* dalam satu putaran penuh poros *encoder* (biasanya poros motor sebelum *gearbox*). Pada *encoder*, terdapat dua *channel* (A dan B), dan pembacaan secara *full quadrature* menghasilkan empat kali jumlah pulsa per putaran.

Pada penelitian ini digunakan motor DC JGA25-370 dengan *encoder* bawaan. *Encoder* tersebut terpasang pada poros motor sebelum *gearbox*, dengan spesifikasi internal 11 PPR (*Pulses Per Revolution*) per *channel*. Karena menggunakan pembacaan *full quadrature* (4x), maka total hitungan dari satu putaran poros motor menjadi :

$$CPR_{motor} = 11 \times 4 = 44 \text{ counts per putaran motor} \quad (3.1)$$

Untuk mendapatkan CPR pada *output shaft*, nilai tersebut dikalikan dengan rasio *gearbox*. Karena rasio *gearbox* tidak selalu tercantum secara presisi, maka dilakukan kalibrasi manual dengan memutar *output shaft* sebanyak 1 putaran penuh, dan total hitungan *encoder* dicatat menggunakan mikrokontroler (ESP32). Misalnya, jika hasil pembacaan menunjukkan 496 *count* untuk 1 putaran penuh *output shaft*, maka :

- CPR = **496**
- PPR = CPR / 4 = **124**
- Estimasi Rasio *Gearbox* = $496 / (11 \times 4) = 11.27 \rightarrow$ Jadi Rasio *Gearbox* sebesar **1:11**

Nilai CPR ini selanjutnya digunakan dalam perhitungan kecepatan putaran (RPM) dan sebagai umpan balik dalam kontrol PID.

Berikut adalah kode program yang digunakan untuk kalibrasi CPR :

Source Code 3. 1. Kalibrasi CPR

```
#include <ESP32Encoder.h>

#define ENCODER_PIN_A 34
#define ENCODER_PIN_B 35

ESP32Encoder encoder;

long startCount = 0;
long lastCount = 0;
long deltaCount = 0;

float CPR = 0;
float PPR = 0;
float gearRatio = 0;
const float internalEncoderPPR = 11.0;

float totalRotations = 0;
int targetRotations = 1;

bool calibrationDone = false;

void tampilanPanduan() {
    Serial.println("===== KALIBRASI CPR =====");
    Serial.println("Langkah-langkah:");
    Serial.println("1. Pastikan motor dan encoder terhubung.");
    Serial.println("2. Putar shaft OUTPUT searah jarum jam.");
    Serial.println("3. Putar 1x penuh (360 derajat) dengan stabil.");
    Serial.println("4. Tunggu hasil kalibrasi muncul.");
    Serial.println("-----");
}

void setup() {
    Serial.begin(115200);

    encoder.attachFullQuad(ENCODER_PIN_B, ENCODER_PIN_A);
    encoder.clearCount();

    lastCount = encoder.getCount();
    startCount = lastCount;

    while (!Serial);

    delay(5000);

    tampilanPanduan();
}

void loop() {
    if (calibrationDone) return;

    long currentCount = encoder.getCount();
    deltaCount = currentCount - lastCount;

    if (deltaCount > 0) {
        totalRotations += deltaCount / 500.0;

        if (totalRotations >= targetRotations) {
            calibrationDone = true;
        }
    }
}
```

```

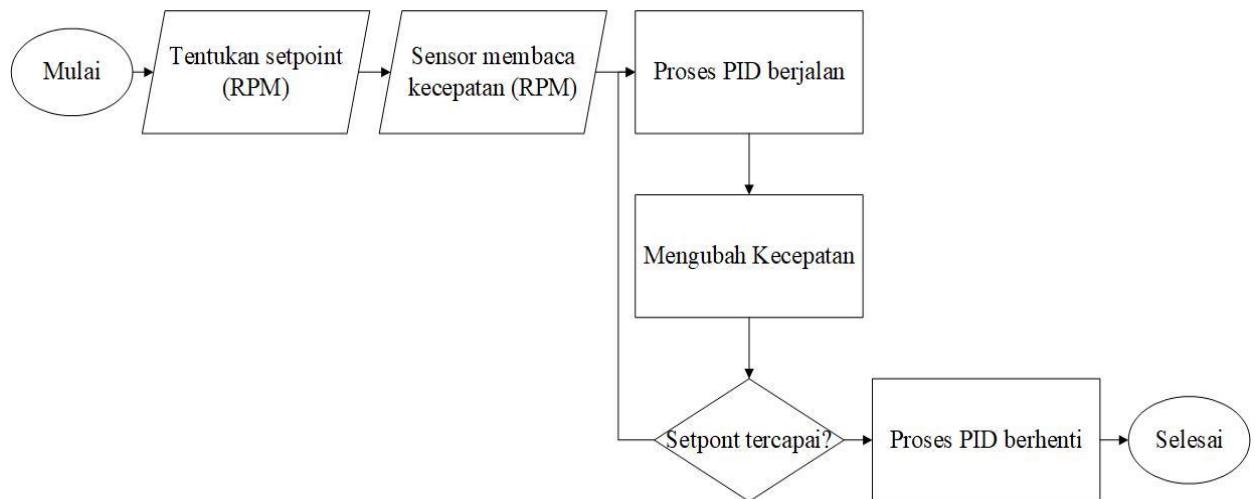
long totalCounts = currentCount - startCount;
CPR = (float)totalCounts;
PPR = CPR / 4.0;
gearRatio = CPR / (internalEncoderPPR * 4.0);

Serial.println();
Serial.println("===== HASIL KALIBRASI =====");
Serial.print("PPR (Pulse/Revolusi) = "); Serial.println(round(PPR));
Serial.print("CPR (Count/Revolusi) = "); Serial.println(round(CPR));
Serial.print("Gear Ratio (Motor:Out) = 1:"); Serial.println(round(gearRatio));
Serial.println("===== ");
Serial.println("✓ Kalibrasi selesai. Gunakan nilai di atas.");
}
}
lastCount = currentCount;
}

```

3.2.4. Skema Implementasi PID Ziegler-Nichols

Tujuan dari proses penalaan ini adalah untuk memperoleh parameter yang optimal sesuai dengan karakteristik *plant*, sehingga respons keluaran dapat merefleksikan respons masukan secara akurat dan responsif.



Gambar 3. 10. Flowchart Implementasi PID

Untuk menyetel parameter PID menggunakan *Ziegler-Nichols* Metode 2, pertama-tama sistem dikonfigurasikan ke dalam mode kontrol proporsional (*P – Only*), yaitu hanya mengaktifkan pengendali proporsional (K_p) saja, sedangkan pengendali integral (K_i) dan derivatif (K_d) diatur ke nol. Selanjutnya, nilai K_p dinaikkan secara bertahap hingga sistem menunjukkan osilasi kontinu yang stabil, yaitu kondisi di mana sinyal *output* sistem berosilasi secara konstan

tanpa meredam atau membesar. Nilai K_p pada titik ini disebut sebagai *Ultimate Gain* (K_u), dan waktu yang dibutuhkan untuk satu siklus osilasi penuh disebut *Ultimate Period* (P_u).

Tabel 3. 8. Langkah – Langkah Menentukan K_u dan P_u

No.	Langkah – Langkah
1.	Gunakan kontrol P (<i>Proportional</i>) saja di sistem, dengan mematikan komponen Integral (I) dan Derivatif (D) dari kontroler PID.
2.	Atur nilai K_p dengan yang kecil dulu, misalnya 0.5 atau 1.
3.	Naikkan nilai K_p secara perlahan, sambil mengamati respons sistem yang ada pada <i>Serial Plotter</i> . <ul style="list-style-type: none"> • Sistem awalnya akan stabil atau lambat berosilasi. • Terus naikkan sampai sistem mulai berosilasi terus-menerus dengan amplitudo yang tetap (konstan).
4.	Catat Nilai Saat Osilasi Stabil. <ul style="list-style-type: none"> • K_u adalah nilai K_p saat osilasi konstan dan stabil, tidak membesar dan tidak mengecil. • P_u adalah periode satu siklus osilasi (waktu dari satu puncak ke puncak berikutnya).

Contoh Studi Kasus :

Pak Burhan, seorang pengusaha batik, mengandalkan beberapa mesin *Conveyor* untuk mendukung proses produksinya. Namun, salah satu mesin *Conveyor* mengalami penurunan performa, terutama dalam menjaga kestabilan kecepatan putaran motor DC. Akibatnya, proses pemindahan kain batik di salah satu lini produksi dihentikan sementara demi menjaga keamanan proses, hal ini dapat berdampak pada keterlambatan pengemasan dan distribusi produk. Untuk mengatasi masalah tersebut, Pak Burhan berencana melakukan *tuning* ulang pada sistem kendali kecepatan motor DC, agar dapat mencapai *setpoint* 100 RPM dengan respons yang cepat dan stabil. Mengingat metode yang biasa digunakan adalah PID *Ziegler-Nichols* Tipe 2 (ZN-2), Pak Burhan pun mempercayakan penyelesaiannya kepada pegawai yang ahli di bidang kendali sistem.

Jika Anda adalah pegawai yang diminta oleh Pak Burhan, bagaimana langkah-langkah yang akan Anda lakukan untuk menyelesaikan permasalahan tersebut ?

Penyelesaian Studi Kasus :

Tabel 3. 9. Langkah-Langkah Penyelesaian Studi Kasus

No.	Langkah - Langkah
1.	$K_p = 0.5, K_i = 0, K_d = 0.$
2.	Naikkan K_p secara perlahan : $K_p = 0.5 \rightarrow 1 \rightarrow 2 \rightarrow \dots$ dst.
3.	<ul style="list-style-type: none"> Catat nilai : Setelah beberapa kali percobaan, diketahui bahwa saat $K_p = 5.6 \rightarrow$ Osilasi terlihat stabil → maka $K_u = 5.6$. Ukur periode osilasi dari grafik : Misalnya 1 siklus = 3.4 detik → maka $P_u = 3.4$ detik.

Tabel 3. 10. Formula Diterapkan Pada Studi Kasus

Parameter	Rumus	Hasil
K_p	$0.6 \times K_u$	$0.6 \times 5.6 = 3.36$
T_i	$P_u / 2$	$3.4 / 2 = 1.7$ detik
T_d	$P_u / 8$	$3.4 / 8 = 0.425$ detik

Maka K_p, K_i, K_d yang diperoleh dari kasus tersebut adalah :

$$K_p = 3.36$$

$$K_i = \frac{K_p}{T_i} = \frac{3.36}{1.7} = 1.976$$

$$K_d = K_p \times T_d = 3.36 \times 0.425 = 1.428$$

Pengendalian kecepatan motor DC dengan metode *Ziegler-Nichols* dianggap berhasil jika motor bisa mencapai kecepatan yang diinginkan dengan kesalahan akhir (*steady-state error*) kurang dari 5%, waktu untuk mencapai kecepatan tersebut kurang dari 1 detik, dan *overshoot* tidak

lebih dari 10%. Sistem juga harus stabil, merespons perubahan kecepatan secara konsisten, tidak mengalami osilasi terus-menerus, serta menghasilkan sinyal kendali yang efisien.

3.3. Hasil (*Output*)

Hasil (*output*) merupakan isi atau konten dari suatu penelitian yang biasanya direpresentasikan melalui narasi, gambar, dan tabel. Hasil yang diharapkan dari penelitian ini dapat dilihat pada Tabel 3. 11 berikut.

Tabel 3. 11. Hasil Yang Diharapkan

No.	Hasil yang diharapkan
1.	Motor DC pada <i>Conveyor</i> dapat dikendalikan dari jarak jauh maupun jarak dekat (<i>On / Off</i> dan arah rotasi).
2.	Memperoleh performa terbaik dalam pengendalian kecepatan putaran (RPM) motor DC pada <i>Conveyor</i> , sehingga memperoleh pengendalian yang lebih optimal.

3.4. Pengujian (*Testing*)

Pengujian merupakan proses yang sistematis untuk mengevaluasi kinerja dan keandalan suatu sistem. Dengan melakukan eksperimen dan menyusun skenario uji coba yang relevan, kita dapat mengukur sejauh mana sistem mencapai tujuan yang telah ditetapkan. Selain itu, pengujian juga bertujuan untuk mengidentifikasi dan memperbaiki kesalahan (*error*) yang mungkin terjadi. Dalam penelitian ini, skenario uji coba dilakukan secara *by manual* dan *by system*.

Skenario uji coba (*by manual*) dapat dilihat pada Tabel 3. 12.

Tabel 3. 12. Skenario Uji Coba (*by manual*)

Uji ke-	Keterangan
1.	Membandingkan benda berat dan ringan di atas <i>Conveyor</i> .
2.	Memantau kecepatan putaran (RPM) motor DC menggunakan <i>Digital Tachometer</i> .

3.	Menghidupkan dan mematikan <i>Conveyor</i> .
4.	Mengatur arah putar <i>Conveyor</i> .
5.	Menguji fungsional kendali <i>ON / OFF</i> motor DC dari jarak dekat melalui penggunaan Tombol Fisik (<i>Push Button</i>).
6.	Menguji fungsional kendali arah putaran motor DC dari jarak dekat melalui penggunaan Tombol Fisik (<i>Push Button</i>).

Adapun skenario uji coba (*by system*) dapat dilihat pada Tabel 3. 13.

Tabel 3. 13. Skenario Uji Coba (*by system*)

Uji ke-	Keterangan
1.	Menguji koneksi <i>Wi-Fi</i> .
2.	Menguji koneksi IoT.
3.	Menguji fungsionalitas <i>Publish</i> dan <i>Subscribe</i> .
4.	Menguji tampilan kecepatan putaran (RPM) motor DC secara <i>real-time</i> pada LCD.
5.	Mengukur dan memantau kecepatan putaran (RPM) motor DC secara <i>real-time</i> melalui <i>framework</i> Arduino IDE.
6.	Mengukur dan memantau kecepatan putaran (RPM) motor DC secara <i>real-time</i> melalui platform Ubidots.
7.	Menguji fungsional kendali <i>ON / OFF</i> motor DC dari jarak jauh melalui penggunaan Tombol Non-Fisik (<i>Virtual Button</i> Ubidots).
8.	Menguji fungsional kendali arah putaran motor DC dari jarak jauh melalui penggunaan Tombol Non-Fisik (<i>Virtual Button</i> Ubidots).

BAB IV

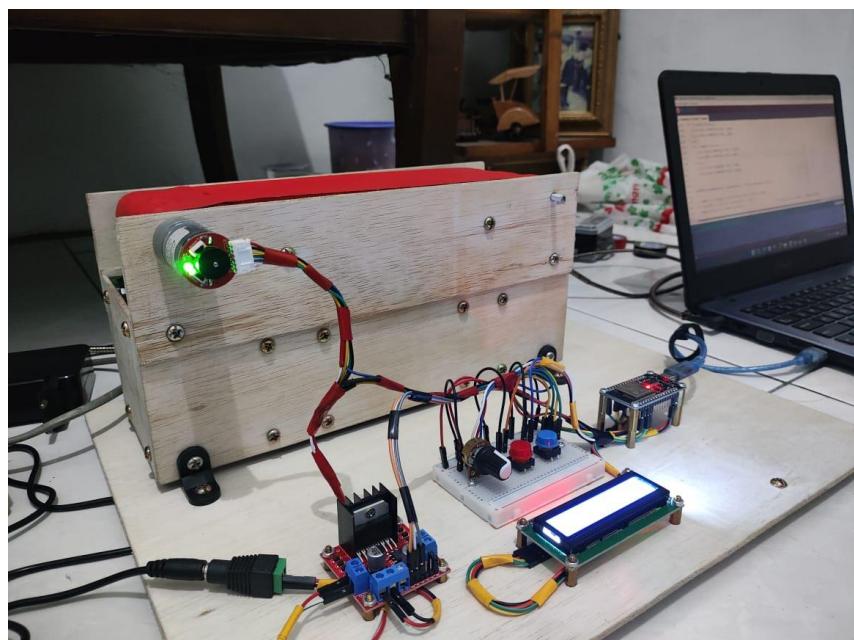
HASIL DAN PEMBAHASAN

Bab ini menjelaskan tentang implementasi rancangan *hardware*, implementasi rancangan *software*, implementasi IoT (*Internet Of Things*), implementasi kalibrasi CPR, dan implementasi PID *Ziegler-Nichols*, serta dilakukannya eksperimen (pengujian) baik *by manual* maupun *by system*. Hal tersebut juga disertai dengan pembahasan yang mendukung penelitian.

4.1. Hasil Penelitian

4.1.1. Implementasi Rancangan *Hardware*

Seluruh komponen *hardware* dirakit berdasarkan spesifikasi dan rancangan prototipe yang telah diuraikan pada bab sebelumnya. Gambar 4. 1 berikut menyajikan tampilan utuh dari perakitan *hardware*, yang terdiri dari komponen *Display*, *Actuator*, dan *Sensor*.



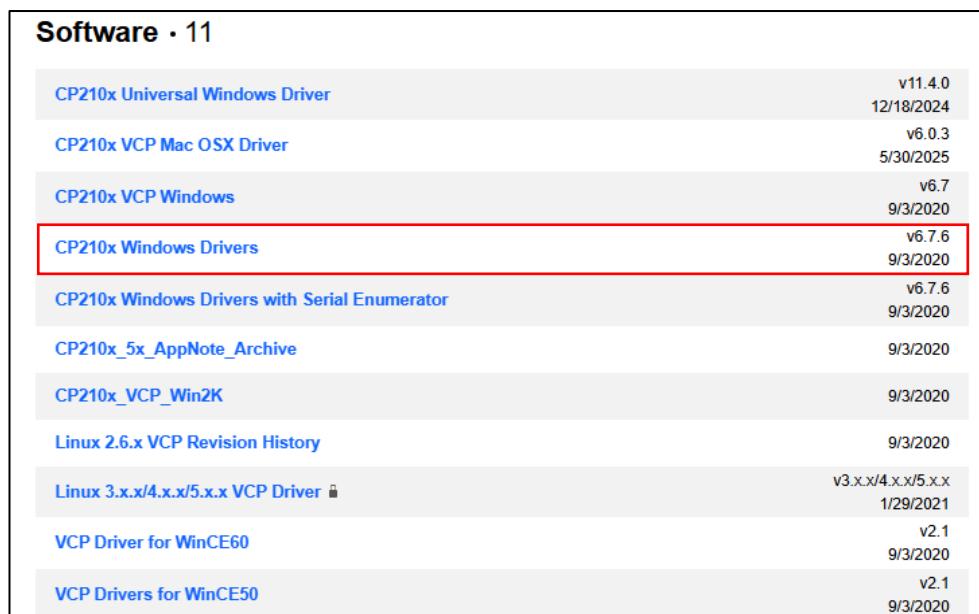
Gambar 4. 1. Rancangan *Hardware* keseluruhan

4.1.2. Implementasi Rancangan *Software*

Pada sub bab ini dijelaskan tentang bagaimana cara menerapkan suatu perancangan *software* yang telah dibuat sebelumnya. Berikut merupakan penjelasan dari setiap tahapan yang penulis lakukan dalam menciptakan Sistem Kendali Kecepatan Motor DC Conveyor menggunakan metode PID *Ziegler-Nichols* berbasis IoT, yaitu sebagai berikut :

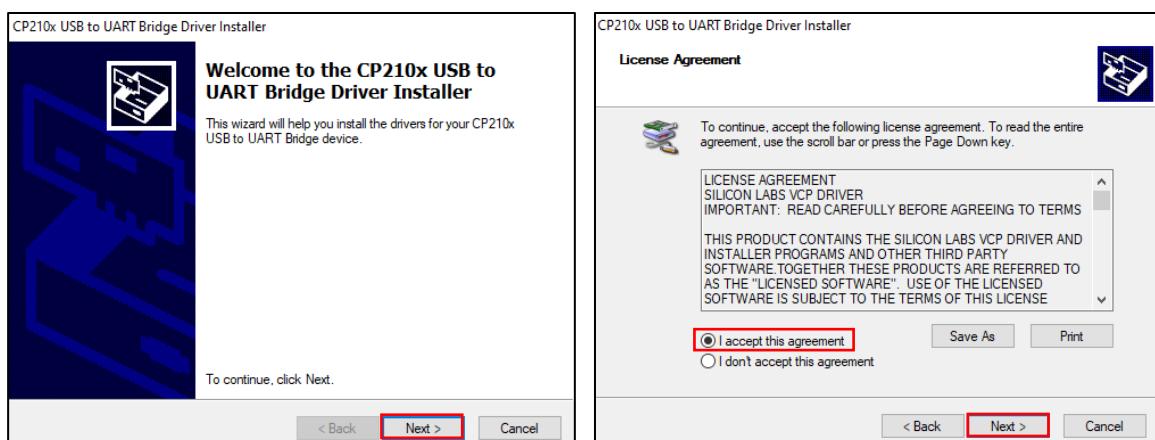
4.1.2.1. Instalasi CP210X Driver

Instalasi CP210X Driver diperlukan apabila perangkat USB tidak terdeteksi pada komputer. Langkah pertama yang perlu dilakukan adalah mengunduh *software* tersebut melalui situs berikut: <https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers>. Proses pengunduhan ini dapat dilakukan seperti yang ditunjukkan pada Gambar 4. 2 untuk OS Windows.



Gambar 4. 2. Download CP210X Driver

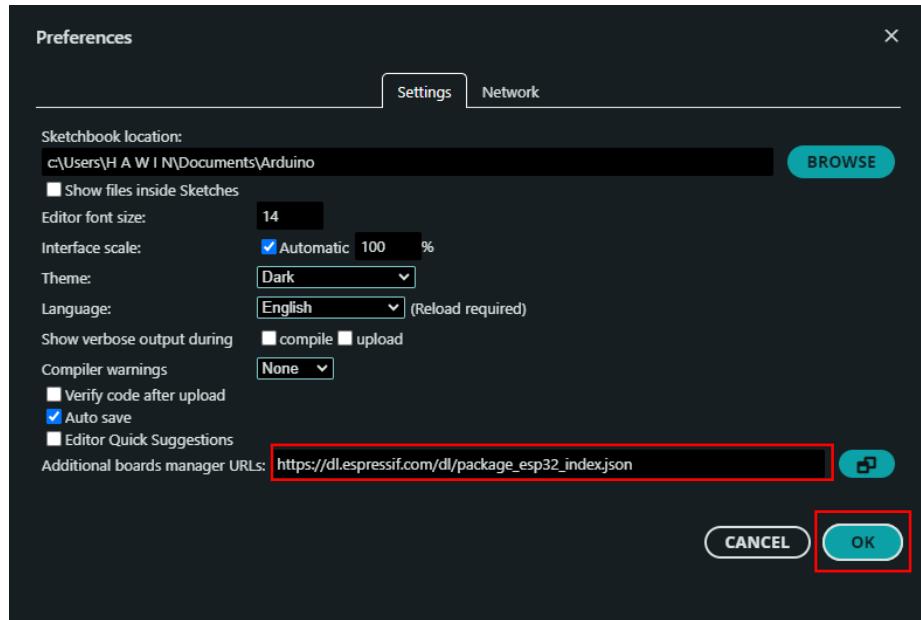
Setelah proses pengunduhan selesai, lakukan ekstraksi terhadap berkas yang telah diunduh, kemudian buka folder hasil ekstraksi dan jalankan berkas instalasi sesuai dengan arsitektur sistem operasi yang digunakan. Selanjutnya, lakukan instalasi dengan memilih opsi “I accept” dan klik tombol “Next”, kemudian tunggu hingga proses instalasi selesai dan akhiri dengan menekan tombol “Finish”. Prosedur ini ditampilkan pada Gambar 4. 3.



Gambar 4. 3. Instalasi CP210X Driver

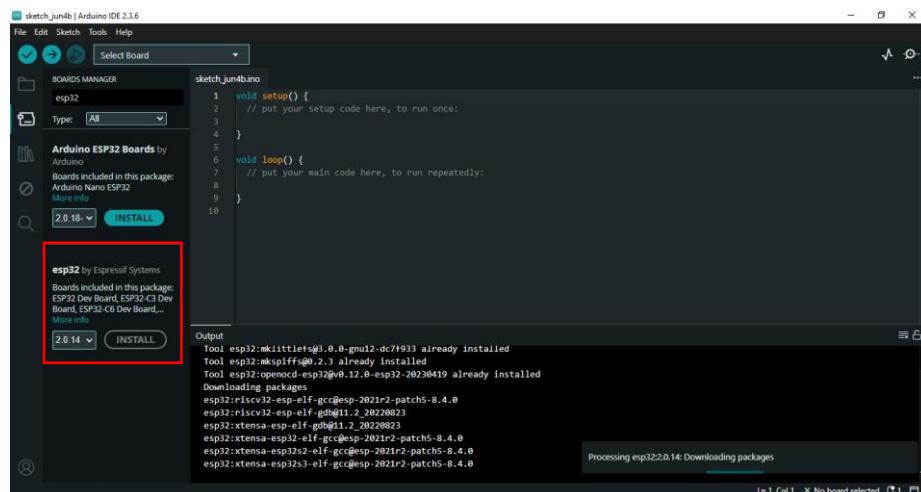
4.1.2.2. Instalasi dan Pengaturan *Board* ESP32 DOIT DevKit V1

Proses instalasi *board* dilakukan melalui perangkat lunak Arduino IDE, yang bertujuan untuk mengatur konfigurasi pada papan mikrokontroler. Dalam penelitian ini, penulis menggunakan *board* ESP32 DOIT DevKit V1. Langkah awal yang perlu dilakukan oleh pengguna adalah membuka menu *File*, kemudian memilih *Preferences*, dan menambahkan URL: https://dl.espressif.com/dl/package_esp32_index.json sebagaimana ditunjukkan pada Gambar 4. 4 berikut.



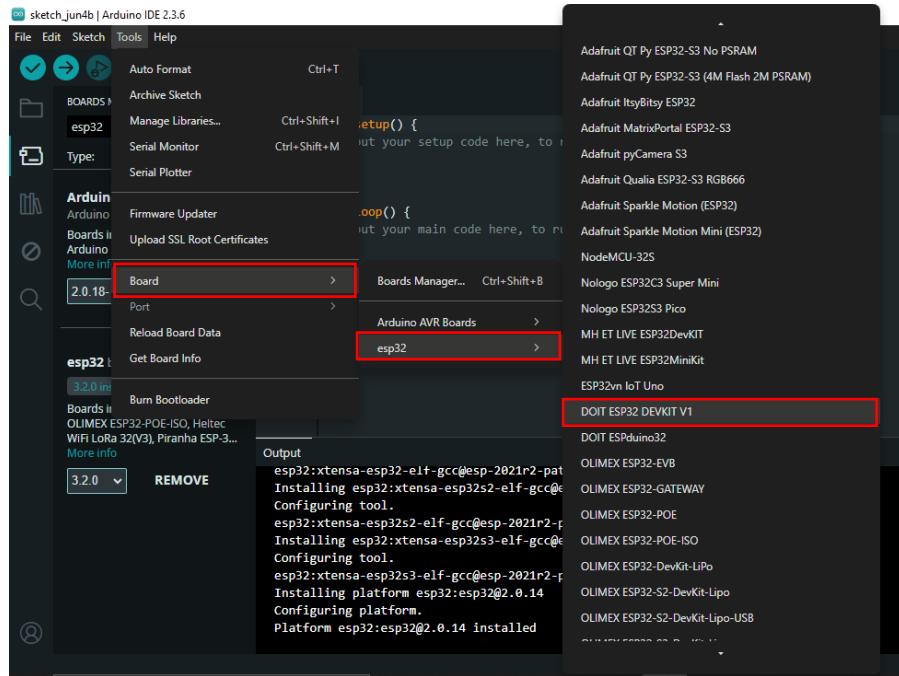
Gambar 4. 4. Additional Board Manager URL ESP32

Langkah terakhir yang perlu dilakukan oleh pengguna adalah membuka menu *Tools*, kemudian mengarahkan kursor ke bagian *Board* dan memilih opsi *Boards Manager* untuk mencari serta menginstal *board* ESP32. Proses ini dapat dilihat pada Gambar 4. 5.



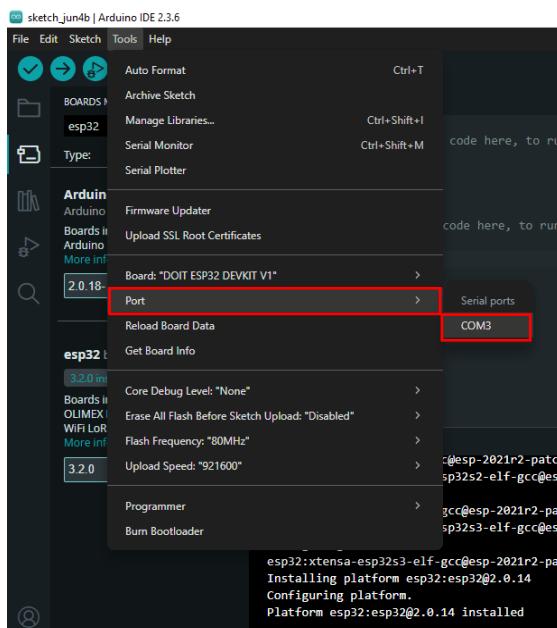
Gambar 4. 5. Instalasi ESP32 *Board*

Setelah proses instalasi *board* ESP32 selesai, langkah selanjutnya adalah mengonfigurasi opsi *board* agar sesuai dengan jenis mikrokontroler yang digunakan, yaitu dengan memilih tipe *board* “DOIT ESP32 DEVKIT V1”. Proses ini ditunjukkan pada Gambar 4. 6 berikut.



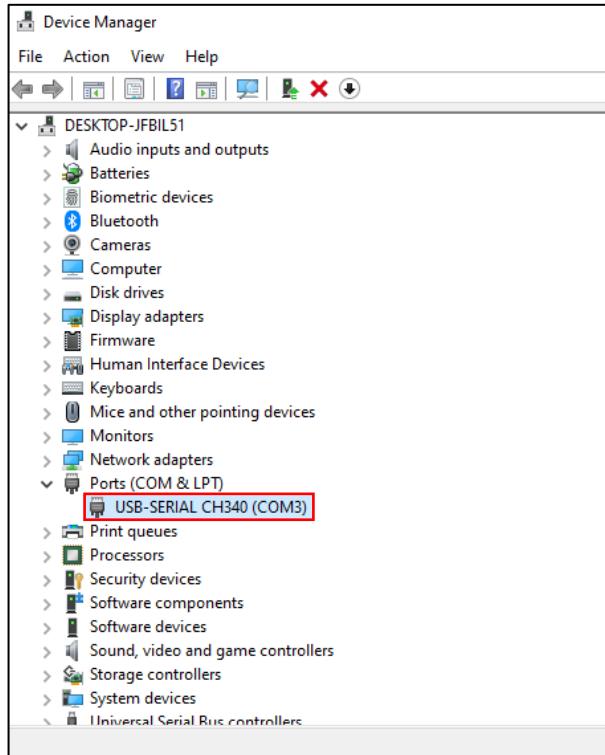
Gambar 4. 6. Pengaturan ESP32 Board

Selain memilih jenis *board*, pengguna juga perlu mengatur *port* (COM) yang sesuai dengan yang terdeteksi di Arduino IDE, dan menyesuaikannya kembali jika diperlukan di *Device Manager*. Tahapan konfigurasi ini dapat dilihat pada Gambar 4. 7 dan Gambar 4. 8.



Gambar 4. 7. Pengaturan Port (COM) ESP32 di Arduino IDE

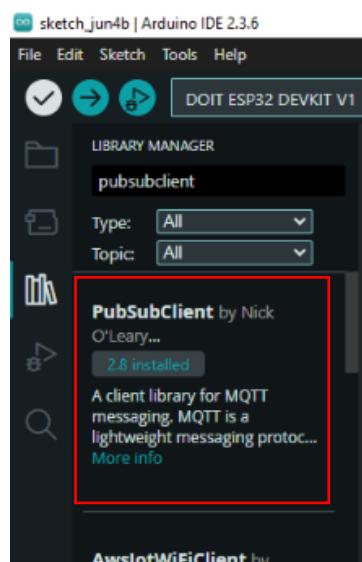
Buka *Device Manager* untuk melihat apakah *Driver* sudah terinstal atau belum.



Gambar 4. 8. Mengecek Port (COM) ESP32 di *Device Manager*

4.1.2.3. Instalasi dan Pengaturan *Library*

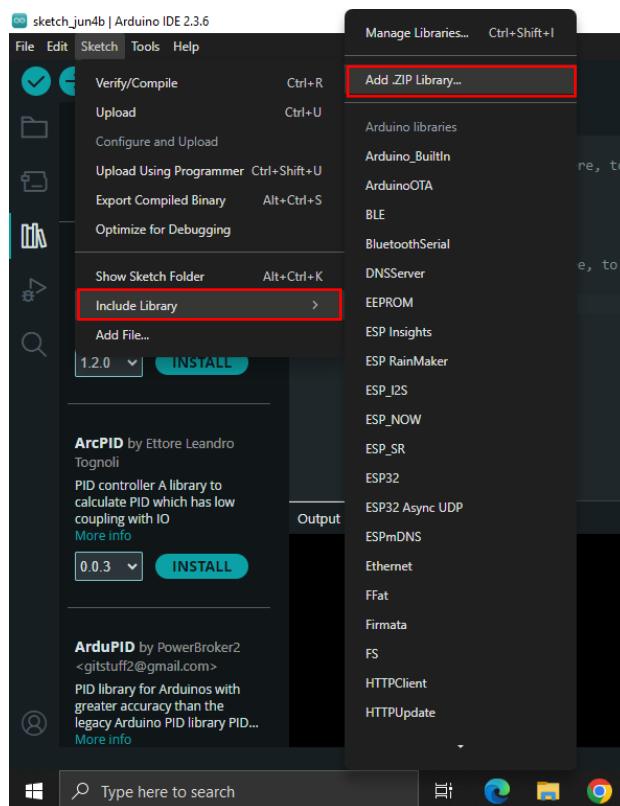
Library umumnya digunakan untuk mempermudah implementasi fungsi tertentu dalam pemrograman. Terdapat beberapa metode untuk menginstal *library* pada perangkat lunak Arduino IDE, salah satunya yang paling umum dan direkomendasikan bagi pemula adalah melalui fitur *Manage Libraries* karena prosesnya relatif mudah. Langkah ini dapat dilihat pada Gambar 4. 9.



Gambar 4. 9. Instalasi *Library* di *Library Manager*

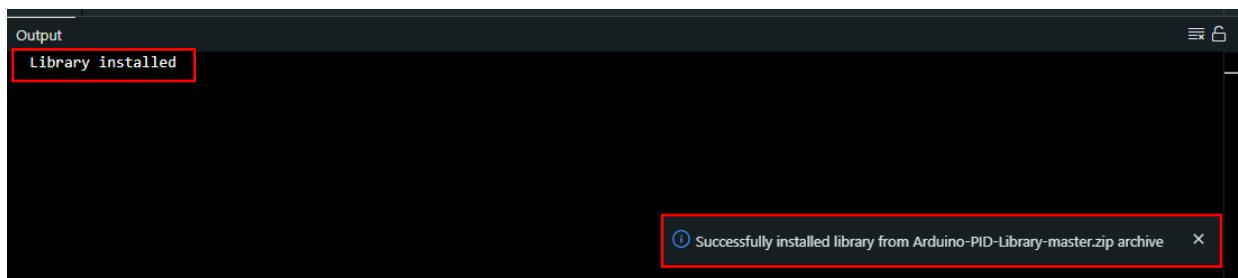
Metode alternatif lainnya adalah dengan melakukan instalasi *library* secara manual. Langkah pertama yang perlu dilakukan adalah mengunduh file *library* dalam format (.zip), kemudian mengekstraknya ke dalam *folder* atau direktori penyimpanan *library* Arduino IDE. Setelah itu, lakukan *refresh* pada Arduino IDE agar sistem dapat memuat pembaruan tersebut.

Adapun opsi lain, pengguna dapat langsung menggunakan file hasil unduhan (.zip) tanpa mengekstraknya terlebih dahulu. Caranya, buka Arduino IDE, pilih menu *Sketch*, arahkan kursor ke *Include Library*, kemudian pilih *Add .ZIP Library* dan tentukan file *library* yang ingin ditambahkan. Proses ini dapat dilihat pada Gambar 4. 10 berikut.



Gambar 4. 10. Instalasi *Library* Secara Manual

Jika penambahan *library* berhasil, maka hasilnya akan ditampilkan seperti yang terlihat pada Gambar 4. 11.



Gambar 4. 11. *Library* Berhasil Ditambahkan pada Arduino IDE

Beberapa *library* yang digunakan antara lain sebagai berikut :

- a. LiquidCrystal I2C by Frank de Brabander.
- b. PubSubClient by Nick O'Leary Knolleary.
- c. ArduinoJson by Benoit Blanchon.
- d. ESP32Encoder by Kevin Harrington.

.

4.1.2.4. Kalibrasi CPR (*Counts Per Revolution*)

Kalibrasi CPR (*Counts Per Revolution*) dapat dilakukan dengan cara memutar *encoder* sebanyak satu putaran penuh (360 derajat) secara manual, lalu membaca nilai *count* yang tercatat oleh sistem. Nilai akhir dari *count* tersebut kemudian ditetapkan sebagai nilai CPR yang sesungguhnya dari *encoder* yang digunakan. Proses ini bertujuan untuk memastikan bahwa nilai CPR sesuai dengan karakteristik fisik *encoder*, karena tiap jenis *encoder* bisa memiliki jumlah pulsa berbeda per putaran. Dengan kalibrasi ini, sistem dapat menghitung RPM motor secara lebih akurat, karena nilai CPR yang digunakan dalam rumus perhitungan benar-benar merefleksikan jumlah pulsa aktual per satu rotasi penuh poros *encoder*.



Gambar 4. 12. Kalibrasi CPR Secara Manual

4.1.2.5. Struktur Kode Program

Struktur program yang digunakan dalam penelitian ini dirancang secara modular untuk memudahkan pengembangan, pemeliharaan, dan pemahaman alur kerja sistem. Program utama berada pada file Main.ino, yang berfungsi sebagai titik awal eksekusi sistem dan mengintegrasikan seluruh komponen yang diperlukan. Struktur direktori program mencakup beberapa file *header* (.h) dan implementasi file Arduino (.ino) yang masing-masing memiliki peran tersendiri. Adapun detailnya dapat Anda lihat sebagai berikut :

```

Main
|__ Main.ino
|__ Connection.h
|__ WiFi_Connection.ino
|__ Ubidots_Connection.ino
|__ Ubidots_PubSub.ino
|__ IO_Device.h
|__ IO_Device.ino
|__ PID_Control.ino

```

4.1.2.6. Program Utama

Seperti namanya, ini berfungsi sebagai pusat eksekusi utama dalam program dan memiliki prioritas tertinggi saat sistem dijalankan. Di dalamnya terdapat dua fungsi utama, yaitu `setup()` dan `loop()`. Fungsi `setup()` digunakan untuk melakukan inisialisasi awal, seperti konfigurasi perangkat, konfigurasi jaringan, dan inisialisasi PID. Sementara itu, fungsi `loop()` bertugas menjalankan logika sistem secara berulang, mencakup pengecekan koneksi *Wi-Fi* dan IoT, pengiriman data ke platform Ubidots, pembacaan *input* dari tombol dan potensiometer, serta penghitungan dan penerapan nilai kendali PID untuk mengatur kecepatan putaran (RPM) motor DC pada *Conveyor*. File ini juga memuat *header* seperti `Connection.h` dan `IO_Device.h`, yang berperan penting dalam mengintegrasikan berbagai modul lain agar sistem dapat berjalan secara terstruktur dan efisien.

Source Code 4. 1. Main.ino

```

#include "Connection.h"
#include "IO_Device.h"
void setup() {
    initIO();
    debugSerialConn(true);
    debugSerialPubSub(true);
    debugSerialPidPlotter(false);
    connectWiFi();
    connectUbidots();
    initPID();
}
void loop() {
    checkWiFiConnection();
    checkUbidotsConnection();
    handlePotentiometer();
    handleButtons();
    publishUbidots();
    updatePid();
}

```

4.1.2.7. Konfigurasi *Wi-Fi* dan IoT

Bagian *header* pertama, yaitu Connection.h, berfungsi untuk mengatur koneksi antara ESP32 dan platform IoT Ubidots melalui jaringan *Wi-Fi* dengan protokol MQTT. File ini mengimpor beberapa *library* penting seperti WiFi.h, PubSubClient.h, dan ArduinoJson.h, serta mendefinisikan sejumlah parameter koneksi, seperti SSID *Wi-Fi*, token autentikasi Ubidots, alamat server MQTT, dan label perangkat maupun variabel (topik) yang digunakan. Selain itu, file ini juga membuat objek WiFiClient dan PubSubClient sebagai media komunikasi data. Untuk memastikan file hanya disertakan sekali saat proses kompilasi, digunakan *header guard* dengan direktif #ifndef CONNECTION_H.

Source Code 4. 2. Connection.h

```
#ifndef CONNECTION_H
#define CONNECTION_H
#include <WiFi.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
WiFiClient espClient;
PubSubClient client(espClient);
#define WIFI_NAME "YOUR_WIFI_NAME"
#define WIFI_PASSWORD "YOUR_WIFI_PASSWORD"
#define MQTT_SERVER "industrial.api.ubidots.com"
#define MQTT_PORT 1883
#define UBIDOTS_TOKEN "YOUR_UBIDOTS_TOKEN"
#define UBIDOTS_PASSWORD ""
#define DEVICE_LABEL "esp32_motor_dc"
#define VARIABLE_LABEL1 "rpm"
#define VARIABLE_LABEL2 "switch1"
#define VARIABLE_LABEL3 "switch2"
String CLIENT_ID = "esp32-" + String(random(1000));
#endif
```

WiFi_Connection.ino berfungsi untuk mengelola koneksi *Wi-Fi* pada mikrokontroler ESP32 secara otomatis dan berkala. File ini mencakup variabel yang merekam status koneksi serta waktu pemeriksaan terakhir, dan menyediakan sejumlah fungsi untuk menangani proses penyambungan ke jaringan *Wi-Fi*. Fungsi connectWiFi() bertugas menyambungkan ESP32 ke jaringan menggunakan SSID dan *password* yang telah ditentukan, dengan mekanisme percobaan terbatas serta menampilkan status koneksi melalui *Serial Monitor*. Sementara itu, fungsi checkWiFiConnection() digunakan untuk memantau koneksi secara periodik dan akan memanggil ulang proses penyambungan jika koneksi terputus. Selain itu, fitur *debugging* opsional tersedia untuk menampilkan *log* proses secara *real-time* melalui *Serial Monitor*. Seluruh mekanisme ini dirancang agar ESP32 tetap terhubung secara stabil ke jaringan *Wi-Fi*, serta mampu melakukan *reconnect* otomatis saat koneksi terputus.

Source Code 4.3. WiFi_Connection.ino

```
bool debugConnMode = false;
void debugSerialConn(bool enable) {
    debugConnMode = enable;
}
void debugConnPrint(const String& msg) {
    if (debugConnMode) Serial.print(msg);
}
void debugConnPrintln(const String& msg) {
    if (debugConnMode) Serial.println(msg);
}
int retry = 0;
const int maxRetry = 20;
bool wasConnected = false;
bool isWiFiConnecting = false;
bool hasTriedConnecting = false;
unsigned long lastWiFiCheckTime = 0;
const unsigned long wifiCheckInterval = 1000;
const unsigned long wifiRetryInterval = 500;
unsigned long lastWiFiRetryTime = 0;
void connectWiFi() {
    if (!isWiFiConnecting && WiFi.status() != WL_CONNECTED) {
        WiFi.mode(WIFI_STA);
        WiFi.begin(WIFI_NAME, WIFI_PASSWORD);
        debugConnPrintln("=====KONFIGURASI WI-FI=====");
        debugConnPrintln("=====Menghubungkan ke Wi-Fi");
        retry = 0;
        lastWiFiRetryTime = millis();
        isWiFiConnecting = true;
        hasTriedConnecting = true;
    }
}
void checkWiFiConnection() {
    unsigned long currentTime = millis();
    if (WiFi.status() == WL_CONNECTED) {
        if (!wasConnected) {
            debugConnPrintln("\n\n===== STATUS KONFIGURASI WI-FI =====");
            debugConnPrint("Berhasil tersambung ke Wi-Fi: ");
            debugConnPrintln(WIFI_NAME);
            debugConnPrint("IP Lokal: ");
            debugConnPrintln(WiFi.localIP().toString());

            if (WiFi.localIP().toString() == "0.0.0.0") {
                debugConnPrintln("!!! IP tidak valid. Coba ulang koneksi...");
            }
            WiFi.setAutoReconnect(true);
            WiFi.persistent(true);
            debugConnPrintln("\n");
        }
        wasConnected = true;
        isWiFiConnecting = false;
        retry = 0;
        return;
    }
    if (isWiFiConnecting && currentTime - lastWiFiRetryTime >= wifiRetryInterval) {
        lastWiFiRetryTime = currentTime;
        retry++;
        debugConnPrint(".");
        if (retry >= maxRetry) {
    }
```

```

        debugConnPrintln("\n\n===== STATUS KONFIGURASI WI-FI =====");
        debugConnPrintln("!!! GAGAL TERHUBUNG KE WI-FI !!!");
        debugConnPrintln("Silakan periksa SSID dan Password WiFi Anda.");
        debugConnPrint("ESP32 akan mencoba lagi nanti.\n\n\n");
        isWiFiConnecting = false;
        retry = 0;
    }
}
if (!isWiFiConnecting && WiFi.status() != WL_CONNECTED && currentTime - lastWiFiCheckTime >= 1000) {
    lastWiFiCheckTime = currentTime;
    if (wasConnected) {
        debugConnPrintln("===== STATUS KONFIGURASI WI-FI =====");
        debugConnPrintln("Wi-Fi terputus!\nMencoba menghubungkan kembali...\n\n");
        wasConnected = false;
    }
    connectWiFi();
}
}

```

`Ubidots_Connection.ino` bertanggung jawab terhadap proses inisialisasi dan pemeliharaan koneksi antara ESP32 dan platform IoT Ubidots melalui protokol MQTT. Fungsi `connectUbidots()` digunakan untuk menginisialisasi koneksi awal, mencakup pengaturan server MQTT, *port*, dan penetapan *callback handler*. Selanjutnya, fungsi `checkUbidotsConnection()` memastikan koneksi tetap aktif dengan secara berkala menjalankan `client.loop()`. Apabila terjadi pemutusan koneksi, fungsi `reconnect()` akan berusaha menyambungkan ulang secara periodik setiap 5 detik. Mekanisme *reconnect* ini dilengkapi dengan sistem *logging* dan penanganan kesalahan berdasarkan kode status koneksi MQTT, sehingga memudahkan proses *debugging* dan pemantauan stabilitas koneksi perangkat dengan server Ubidots.

Source Code 4. 4. Ubidots_Connection.ino

```

unsigned long lastReconnectAttempt = 0;
bool isConnecting = false;
const unsigned long reconnectInterval = 5000;
void connectUbidots() {
    client.setServer(MQTT_SERVER, MQTT_PORT);
    client.setCallback(callback);
    lastReconnectAttempt = millis();
    isConnecting = false;
}
void checkUbidotsConnection() {
    if (!client.connected()) {
        reconnect();
    }
    client.loop();
}
void reconnect() {
    unsigned long currentTime = millis();
    if (!client.connected() && (currentTime - lastReconnectAttempt >=
reconnectInterval)) {
        lastReconnectAttempt = currentTime;
        isConnecting = true;
        debugConnPrintln("=====");
    }
}

```

```

debugConnPrintln("KONFIGURASI IOT PLATFORM: UBIDOTS");
debugConnPrintln("===== ===== ===== ===== ===== =====");
debugConnPrintln("Menghubungkan ke Ubidots.....");
if (client.connect(CLIENT_ID.c_str(), UBIDOTS_TOKEN, UBIDOTS_PASSWORD)) {
    debugConnPrintln("\n===== STATUS KONFIGURASI UBIDOTS =====");
    debugConnPrintln("Berhasil tersambung ke IoT Platform: Ubidots");
    subscribeUbidots();
    debugConnPrintln("\n");
    isConnecting = false;
} else {
    debugConnPrintln("\n===== STATUS KONFIGURASI UBIDOTS =====");
    debugConnPrint("Gagal, Kode Error = ");
    debugConnPrintln(String(client.state()));
    switch (client.state()) {
        case -2:
            debugConnPrintln("• WiFi / Hotspot Anda tidak aktif !!!");
            debugConnPrintln("• Atau alamat Server MQTT yang Anda masukkan salah !!!");
        break;
        case -4:
            debugConnPrintln("• Tidak ada akses internet / paket data !!!");
            break;
        case 5:
            debugConnPrintln("• Token Ubidots yang Anda masukkan salah !!!");
            break;
        default:
            debugConnPrintln("• Gagal terkoneksi karena alasan lain.");
            break;
    }
    debugConnPrintln("• Coba lagi 5 detik.\n\n");
    isConnecting = false;
}
}
}
}

```

`Ubidots_PubSub.ino` menangani komunikasi antara mikrokontroler dan platform IoT Ubidots menggunakan protokol MQTT. Program ini memproses dan menyimpan data dari tiga variabel (topik), yaitu `rpm`, `switch1`, dan `switch2`. Data yang diterima dari Ubidots melalui fungsi `callback` akan diperiksa berdasarkan topik, kemudian disimpan dan, jika mode `debug` diaktifkan, ditampilkan melalui `output debug`. Selanjutnya, data yang diperbarui akan dikemas dalam format JSON dan dipublikasikan secara berkala ke server Ubidots. Fungsi `subscribeUbidots()` digunakan untuk melakukan langganan ke topik MQTT tertentu, sedangkan fungsi `publishUbidots()` bertugas mengatur pengiriman data ke Ubidots setiap 5 detik atau saat terjadi perubahan nilai.

Source Code 4. 5. Ubidots_PubSub.ino

```

static unsigned long lastSend = 0;
const unsigned long interval = 5000;
String variables[3] = {VARIABLE_LABEL1, VARIABLE_LABEL2, VARIABLE_LABEL3};
int values[3] = {0, 0, 0};
bool debugPubSubMode = false;
void debugSerialPubSub(bool enable) {
    debugPubSubMode = enable;
}

```

```

}

void debugPubSubPrint(const String& msg) {
    if (debugPubSubMode) Serial.print(msg);
}

void debugPubSubPrintln(const String& msg) {
    if (debugPubSubMode) Serial.println(msg);
}

bool rpmDataReceived = false;
bool needPublish = false;
void addUbidots(String var, int val) {
    for (int i = 0; i < 3; i++) {
        if (var == variables[i]) {
            values[i] = val;
            break;
        }
    }
}

void publishUbidots() {
    unsigned long currentTime = millis();
    if (currentTime - lastSend >= interval) {
        lastSend = currentTime;
        StaticJsonDocument<128> DOC;
        for (int i = 0; i < 3; i++) {
            DOC[variables[i]] = values[i];
        }
        char PAYLOAD[128];
        serializeJson(DOC, PAYLOAD);
        String TOPIC = "/v1.6/devices/" + String(DEVICE_LABEL);
        client.publish(TOPIC.c_str(), PAYLOAD);
        debugPubSubPrintln("\n\n=====");
    );
        debugPubSubPrintln("PUBLISH MQTT");
        debugPubSubPrintln("=====");
        debugPubSubPrintln("Topic: " + TOPIC);
        debugPubSubPrintln("Json Data: " + String(PAYLOAD));
        debugPubSubPrint("\n\n");
    }
}

void subscribeUbidots() {
    client.subscribe(("v1.6/devices/" + String(DEVICE_LABEL) + "/" +
String(VARIABLE_LABEL1) + "/lv").c_str());
    client.subscribe(("v1.6/devices/" + String(DEVICE_LABEL) + "/" +
String(VARIABLE_LABEL2) + "/lv").c_str());
    client.subscribe(("v1.6/devices/" + String(DEVICE_LABEL) + "/" +
String(VARIABLE_LABEL3) + "/lv").c_str());
}

void callback(char* TOPIC, byte* PAYLOAD, unsigned int length) {
    debugPubSubPrintln("=====");
    debugPubSubPrintln("SUBSCRIBE MQTT");
    debugPubSubPrintln("=====");
    debugPubSubPrintln("Topic: " + String(TOPIC));
    String message = "";
    for (unsigned int i = 0; i < length; i++) {
        message += (char)PAYLOAD[i];
    }
    float value = message.toFloat();
    String topicStr = String(TOPIC);
    if (topicStr.indexOf(VARIABLE_LABEL1) >= 0) {
        rpmDataReceived = true;
        debugPubSubPrint("RPM: ");
        debugPubSubPrintln(String((int)value));
    }
}

```

```

else if (topicStr.indexOf(VARIABLE_LABEL2) >= 0) {
    debugPubSubPrint("Status Aktivasi Motor: ");
    motorEnabled = ((int)value == 1);
    debugPubSubPrintln(motorEnabled ? "Menyala" : "Mati");
    addUbidots(VARIABLE_LABEL2, motorEnabled ? 1 : 0);
    needPublish = true;
}
else if (topicStr.indexOf(VARIABLE_LABEL3) >= 0) {
    debugPubSubPrint("Arah Rotasi Motor: ");
    directionForward = ((int)value == 0);
    debugPubSubPrintln(directionForward ? "Maju" : "Mundur");
    addUbidots(VARIABLE_LABEL3, directionForward ? 0 : 1);
    needPublish = true;
}
if (needPublish) {
    publishUbidots();
    needPublish = false;
}
}

```

4.1.2.8. Konfigurasi I/O Perangkat

Bagian *header* kedua, yaitu *IO_Device.h*, berfungsi untuk mendeklarasikan dan menginisialisasi perangkat *input* dan *output* yang digunakan dalam sistem. File ini juga menyertakan *library* pendukung yang diperlukan, serta menggunakan *include guard* untuk mencegah penyertaan ganda saat proses kompilasi.

Source Code 4. 6. IO_Device.h

```

#ifndef IO_DEVICE_H
#define IO_DEVICE_H
#include <ESP32Encoder.h>
#include <LiquidCrystal_I2C.h>
ESP32Encoder encoder;
LiquidCrystal_I2C lcd(0x27, 16, 2);
#endif

```

IO_Device.ino berfungsi untuk mengatur berbagai proses *input* dan *output* dari perangkat keras, seperti *encoder*, potensiometer, *push button*, LCD, serta komponen lainnya. Motor DC dikendalikan melalui sinyal PWM untuk mengatur kecepatan putaran, dan pin digital untuk menentukan arah rotasi. Potensiometer digunakan sebagai penentu nilai RPM target, sedangkan *encoder* berfungsi membaca nilai RPM aktual. Dua *push button* dimanfaatkan untuk mengatur status *On / Off* dan arah putaran motor. Pada tahap awal, LCD menampilkan animasi sederhana sebagai tampilan pembuka, kemudian beralih menampilkan informasi seperti nilai RPM dan status sistem. Data tersebut juga dikirimkan ke platform Ubidots untuk keperluan *monitoring* dan kendali jarak jauh. Proses pembacaan data dan penyesuaian kendali dilakukan secara berkala guna memastikan sistem bekerja secara responsif dan akurat.

Source Code 4.7. IO_Device.ino

```
const int pwmPin = 18, in1Pin = 16, in2Pin = 17;
const int pwmChannel = 0, pwmFreq = 5000, pwmResolution = 8;
const float pwmMinEffective = 60;
const byte encoderPinA = 34, encoderPinB = 35;
long ticksCount;
const int potPin = 36;
const int POT_DEADBAND = 20;
int lastPotValue = -1;
unsigned long lastPotReadTime = 0;
const int POT_READ_INTERVAL = 100;
int potValue;
const int onOffButtonPin = 4, directionButtonPin = 5;
bool lastOnOffState = HIGH, lastDirState = HIGH;
bool currentOnOff;
bool currentDir;
bool motorEnabled = false;
bool directionForward = true;
float setpointRPM = 0.0;
float input = 0.0;
unsigned long lastDisplayTime = 0;
const unsigned long displayInterval = 200;
void initIO() {
    Serial.begin(115200);
    pinMode(in1Pin, OUTPUT); pinMode(in2Pin, OUTPUT);
    ledcSetup(pwmChannel, pwmFreq, pwmResolution);
    ledcAttachPin(pwmPin, pwmChannel);
    encoder.attachFullQuad(encoderPinB, encoderPinA);
    pinMode(potPin, INPUT);
    pinMode(onOffButtonPin, INPUT_PULLUP);
    pinMode(directionButtonPin, INPUT_PULLUP);
    lcd.init();
    lcd.backlight();
    startingLCD();
    lastPotReadTime = millis();
    stopMotor();
}
void handleButtons() {
    currentOnOff = digitalRead(onOffButtonPin);
    if (currentOnOff == LOW && lastOnOffState == HIGH) {
        motorEnabled = !motorEnabled;
        if (!motorEnabled) {
            stopMotor(); updateLCD(setpointRPM, input, "OFF");
        }
        addUbidots(VARIABLE_LABEL2, motorEnabled ? 1 : 0);
        publishUbidots();
        delay(200);
    }
    lastOnOffState = currentOnOff;
    currentDir = digitalRead(directionButtonPin);
    if (currentDir == LOW && lastDirState == HIGH) {
        directionForward = !directionForward;
        addUbidots(VARIABLE_LABEL3, directionForward ? 0 : 1);
        publishUbidots();
        delay(200);
    }
    lastDirState = currentDir;
}
void handlePotentiometer() {
    unsigned long currentTime = millis();
    if (currentTime - lastPotReadTime >= POT_READ_INTERVAL) {
```

```

        lastPotReadTime = currentTime;
        potValue = analogRead(potPin);
        if (abs(potValue - lastPotValue) > POT_DEADBAND) {
            setpointRPM = map(potValue, 0, 4095, 0, 492);
            lastPotValue = potValue;
        }
    }
}

long readAndResetEncoderTicks() {
    ticksCount = encoder.getCount();
    encoder.setCount(0);
    return ticksCount;
}

void setMotorPWM(float pwmVal, bool forward) {
    digitalWrite(in1Pin, forward ? HIGH : LOW);
    digitalWrite(in2Pin, forward ? LOW : HIGH);
    ledcWrite(pwmChannel, (int)pwmVal);
}

void stopMotor() {
    ledcWrite(pwmChannel, 0);
    digitalWrite(in1Pin, LOW); digitalWrite(in2Pin, LOW);
}

void startingLCD() {
    lcd.clear();
    scrollTextFromRight("Project Skripsi", 0);
    scrollTextFromRight("PID-IoT", 1);
    lcd.clear();
    lcd.setCursor(1, 0);
    lcd.print("Project Skripsi");
    lcd.setCursor(5, 1);
    lcd.print("PID-IoT");
    delay(5000);
    loadingLCD();
}

void loadingLCD() {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Loading");
    for (int i = 0; i < 4; i++) {
        delay(500);
        lcd.print(".");
    }
    delay(1000);
}

void scrollTextFromRight(String text, int row) {
    for (int i = 0; i <= 16; i++) {
        lcd.clear();
        lcd.setCursor(16 - i, row);
        lcd.print(text);
        delay(200);
        if ((row == 0 && i == 16 - 0) && text == "Project Skripsi") break;
        if ((row == 1 && i == 16 - 5) && text == "PID-IoT") break;
    }
}

void updateLCD(float setpoint, float actual, String status) {
    if ((millis() - lastDisplayTime) >= displayInterval) {
        lastDisplayTime = millis();
        lcd.setCursor(0, 0); lcd.print("Set Point :      ");
        lcd.setCursor(12, 0); lcd.print((int)setpoint); lcd.print("      ");
        lcd.setCursor(0, 1); lcd.print("      ");
        lcd.setCursor(0, 1);
        if (status == "RUNNING") {

```

```

        lcd.print("C.PID RPM : ");
        lcd.setCursor(12, 1); lcd.print((int)actual);
    } else {
        lcd.print("Status : "); lcd.print(status);
    }
}
}

void printSerialPlot(float setpoint, float rpm, float pidOutput) {
    debugPidPlotterPrint("Setpoint:"); debugPidPlotterPrint(String(setpoint));
    debugPidPlotterPrint(",RPM:"); debugPidPlotterPrint(String(abs(rpm)));
    debugPidPlotterPrint(",Output:"); debugPidPlotterPrintln(String(pidOutput));
}

```

4.1.2.9. Konfigurasi PID dengan *Serial Plotter*

Pada tahap ini dilakukan *tuning* PID dengan menggunakan metode *Ziegler-Nichols* untuk mendapatkan nilai dari K_p , K_i , dan K_d (Proporsional, Integral, Derivatif). Fungsi `initPID()` bertugas untuk mengatur nilai parameter dari PID. Program memiliki dua mode kendali yang ditentukan oleh variabel `controlMode`. Jika diatur ke opsi 1, maka sistem hanya menggunakan kendali proporsional (K_p) saja. Artinya, kecepatan motor akan dikendalikan berdasarkan seberapa besar perbedaan antara kecepatan yang diinginkan dan kecepatan saat ini tanpa mempertimbangkan kesalahan dari waktu ke waktu atau perubahan kecepatan. Mode ini digunakan ketika dalam proses penentuan nilai PID.

Sedangkan jika `controlMode` bernilai 2, maka sistem akan menggunakan kendali PID penuh (Proporsional, Integral, Derivatif). Pada mode ini, nilai K_p , K_i , dan K_d dihitung secara otomatis menggunakan metode *Ziegler-Nichols* berdasarkan parameter osilasi sistem, yaitu K_u (*ultimate gain*) dan T_u (*ultimate period*). Dengan metode ini, sistem lebih mampu menyesuaikan diri terhadap perubahan dan gangguan, sehingga kecepatan motor dapat lebih stabil dan mendekati nilai yang ditargetkan.

Fungsi `updatePID()` bertugas membaca sinyal *encoder*, menghitung kecepatan motor dalam RPM, dan menjalankan algoritma PID untuk mengatur kecepatan sesuai target (`setpointRPM`). Pertama, fungsi memeriksa apakah waktu *sampling* telah tercapai. Jika belum, proses dihentikan lebih awal. Jika iya, maka jumlah pulsa dari *encoder* sejak *update* terakhir dibaca melalui `readAndResetEncoderTicks()`, lalu dikonversi menjadi RPM berdasarkan CPR.

Jika motor aktif, maka perhitungan *error* dihitung dari selisih `setpointRPM` dan *input* (RPM aktual). Dalam mode P, *output* hanya dihitung dari komponen proporsional (P). Sedangkan dalam mode PID lengkap, komponen integral dan derivatif juga digunakan. Komponen integral dihitung dari akumulasi *error* dikali waktu, dengan *anti-windup* sederhana yang mereset nilai

integral jika arah *error* berubah. Komponen derivatif dihitung dari perubahan nilai *input* (RPM aktual) dibandingkan dengan nilai sebelumnya.

Setelah menghitung *output* PID, nilai ini dibatasi antara 0 hingga 255 (*range* PWM standar) menggunakan fungsi *constrain*. Jika nilai *output* terlalu kecil namun bukan nol, maka akan dinaikkan ke nilai minimum efektif PWM (*pwmMinEffective*) agar motor tetap bisa berputar dengan torsi cukup.

Fungsi *addUbidots* dan *publishUbidots* digunakan untuk mengirim data RPM atau status ke platform IoT Ubidots, sedangkan *setMotorPWM* akan mengatur kecepatan dan arah motor berdasarkan *output* PID. Tampilan LCD diperbarui dengan status *RUNNING*, dan semua data ditampilkan melalui *Serial Plotter* agar bisa dianalisis secara visual.

Source Code 4. 8. PID_Control.ino

```
const float CPR = 496.0;
const unsigned long sampleTime = 50;
int controlMode = 2;
float Ku = 12.0, Tu = 0.3;
float Kp = 7.2, Ki = 48, Kd = 0.27;
unsigned long lastTime = 0;
long ticks;
float ticksPerSecond;
bool motorActive;
float output = 0, lastInput = 0, integral = 0, error = 0, derivative = 0;
bool debugPidPlotterMode = false;
void debugSerialPidPlotter(bool enable) {
    debugPidPlotterMode = enable;
}
void debugPidPlotterPrint(const String& msg) {
    if (debugPidPlotterMode) Serial.print(msg);
}
void debugPidPlotterPrintln(const String& msg) {
    if (debugPidPlotterMode) Serial.println(msg);
}
void initPID() {
    if (controlMode == 2 && Ku > 0 && Tu > 0) {
        Kp = 0.6 * Ku;
        Ki = 2.0 * Kp / Tu;
        Kd = Kp * Tu / 8.0;
    } else if (controlMode == 1) {
        Ki = 0.0; Kd = 0.0;
    }
    lastTime = millis();
}
void updatePid() {
    unsigned long currentTime = millis();
    if (currentTime - lastTime < sampleTime) return;
    lastTime = currentTime;
    ticks = readAndResetEncoderTicks();
    ticksPerSecond = abs(ticks * (1000.0 / sampleTime));
    input = abs((float(ticksPerSecond) / CPR) * 60.0);
    if (!directionForward) input *= abs(-1);
    motorActive = motorEnabled && setpointRPM > 5.0;
    if (!motorActive) {
```

```

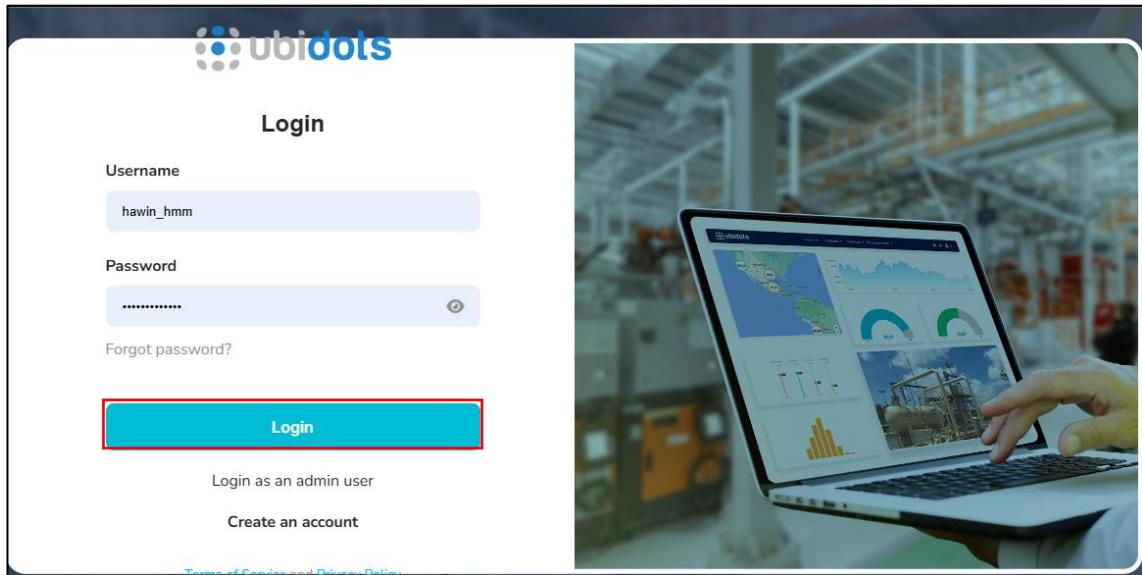
stopMotor();
updateLCD(setpointRPM, input, motorEnabled ? "STANDBY" : "OFF");
output = 0;
printSerialPlot(setpointRPM, input, output);
return;
}
error = setpointRPM - input;
if (controlMode == 1) {
    output = Kp * error;
} else {
    integral += error * (sampleTime / 1000.0);
    if ((error * (setpointRPM - lastInput)) < 0) integral = 0;
    derivative = (input - lastInput) / (sampleTime / 1000.0);
    output = Kp * error + Ki * integral - Kd * derivative;
    lastInput = input;
}
output = constrain(output, 0, 255);
if (output > 0 && output < pwmMinEffective) output = pwmMinEffective;
addUbidots(VARIABLE_LABEL1, input);
publishUbidots();
setMotorPWM(output, directionForward);
updateLCD(setpointRPM, input, "RUNNING");
printSerialPlot(setpointRPM, input, output);
}

```

4.1.2.10. Konfigurasi IoT Platform : Ubidots

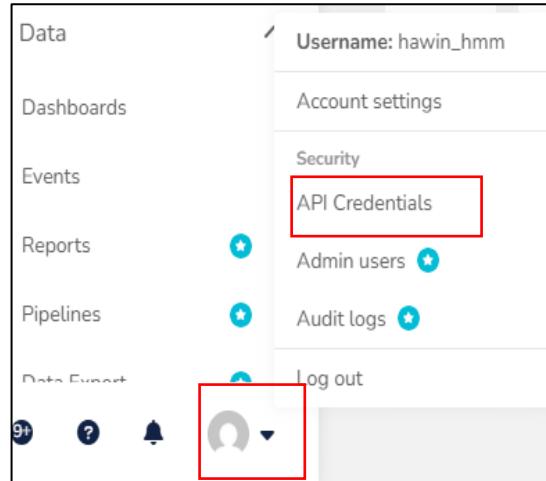
Langkah pertama dalam menggunakan Ubidots adalah mengakses laman berikut:

<https://industrial.Ubidots.com/accounts/signin/>. Jika Anda belum memiliki akun, silakan buat akun Ubidots terlebih dahulu.



Gambar 4. 13. *Login Ubidots*

Untuk mengecek token Ubidots yang dibutuhkan dalam kode program, silakan klik menu *User* di pojok kiri bawah, kemudian pilih opsi API *Credentials* untuk melihat informasi token.



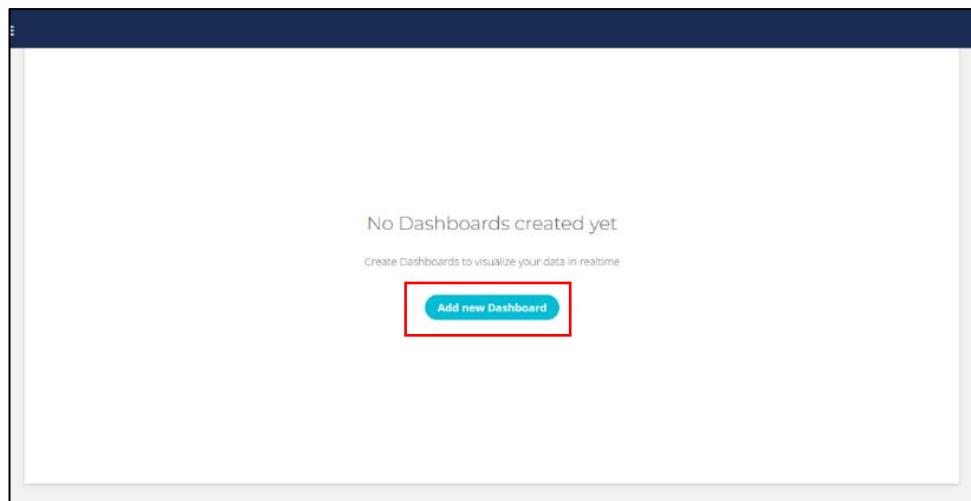
Gambar 4. 14. Menu User Ubidots

Setelah itu, salin *Default Token* yang tersedia, lalu tempelkan *token* tersebut ke dalam kode program Anda.

Name	Created at ↑	Token	Rate limit
Default token	2025-05-11 17:11:38	BBUS-zTeocjjiI099o9NCVBx8e8cGjhifQG5	1 req/1s

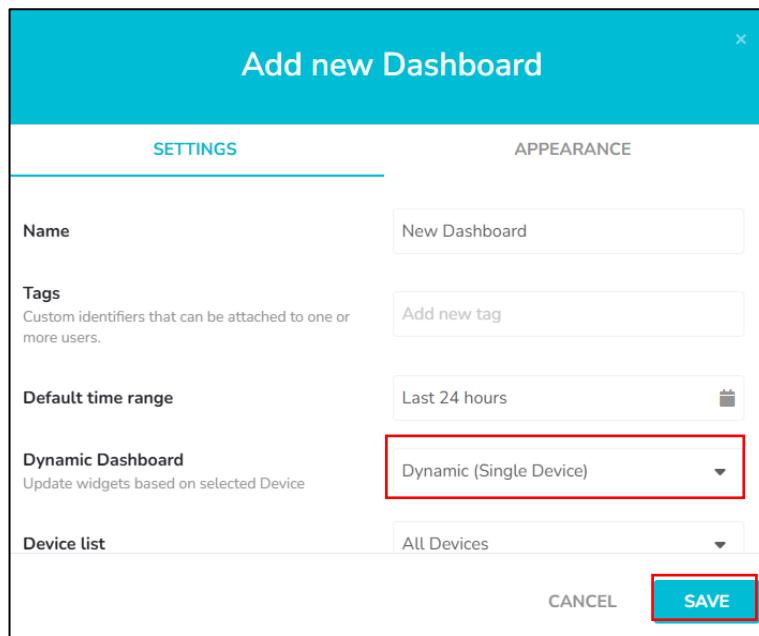
Gambar 4. 15. Token Ubidots

Setelah itu, pilih menu *Dashboards* untuk mulai membuat tampilan visualisasi data. Sebelum membuat dasbor baru, disarankan untuk menghapus terlebih dahulu dasbor demo bawaan dari Ubidots. Selanjutnya, klik tombol “Add new Dashboard” untuk menambahkan dasbor.



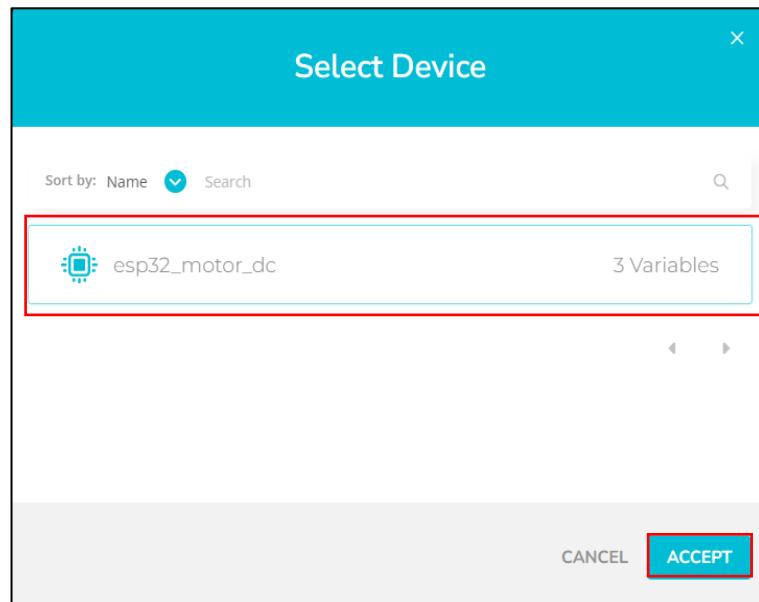
Gambar 4. 16. Menambah Dasbor Baru pada Ubidots

Isi bagian *Name*, *Tags*, dan *Default Time Range* sesuai kebutuhan. Untuk opsi *Dynamic Dashboard*, ubah pengaturannya menjadi “*Dynamic (Single Device)*”, agar dasbor hanya menampilkan data dari satu perangkat saja.



Gambar 4. 17. Mengganti Beberapa Opsi pada Menu *Dashboards*

Selanjutnya, pada bagian *Default Device*, pilih nama perangkat yang telah dibuat sebelumnya, yaitu “esp32_motor_dc”, lalu klik “*ACCEPT*”. Biarkan pengaturan lainnya tetap menggunakan nilai bawaan, kemudian klik “*SAVE*” untuk menyimpan konfigurasi dasbor.



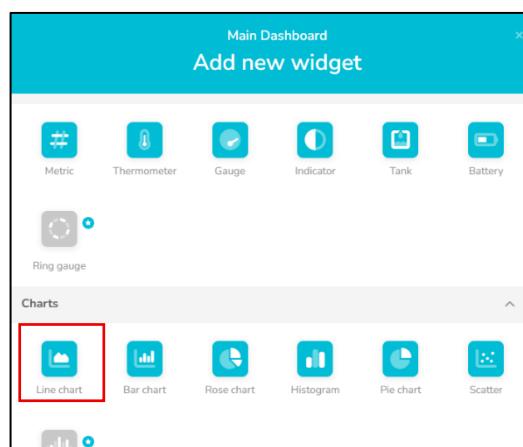
Gambar 4. 18. Pilih Perangkat

Tombol “+ Add new Widget” hanya akan muncul sekali di tengah layar saat awal pembuatan dasbor, kemudian nantinya akan berpindah ke pojok kanan atas dengan tampilan berwarna biru. Klik tombol ini untuk menambahkan *widget* baru.



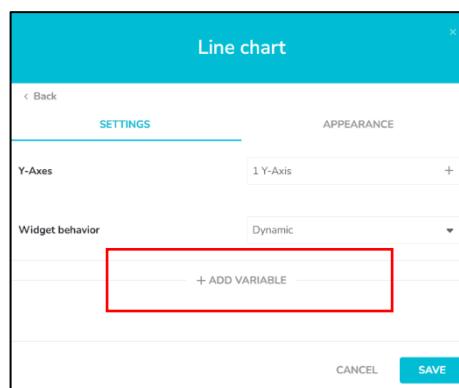
Gambar 4. 19. Menambahkan *Widget* Baru pada Ubidots

“*Widget: Line Chart*” ini dipilih karena bisa menampilkan data dalam bentuk garis, sehingga perubahan nilai dari waktu ke waktu bisa dipantau secara langsung dan terus-menerus.



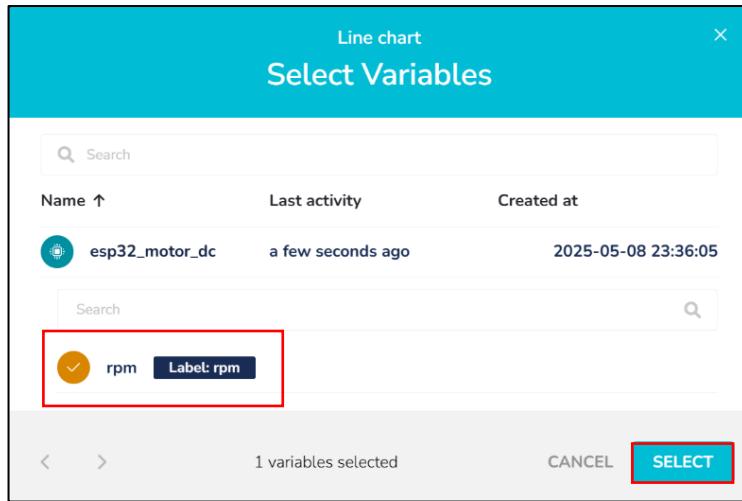
Gambar 4. 20. Pilihan *Widget* pada Ubidots

Setelah memilih *widget* tersebut, silakan klik tombol “+ ADD VARIABLE” untuk menambahkan variabel yang datanya ingin ditampilkan.



Gambar 4. 21. Opsi pada *Widget*

Pilih variabel “*rpm*”. Setelah itu, klik “*SELECT*”, dan kemudian tekan “*SAVE*” untuk menyimpan perubahan dari pengaturan tersebut.



Gambar 4. 22. Memilih Variabel

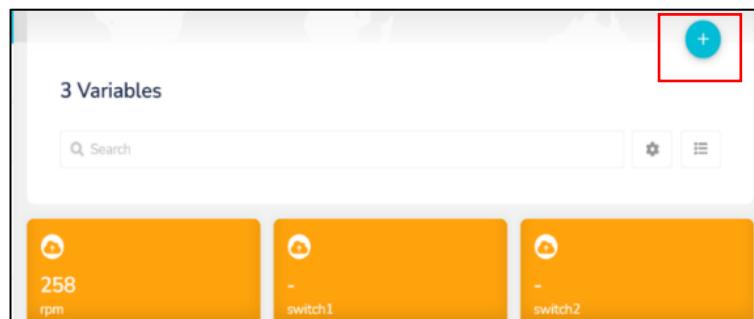
Hasil akhirnya dapat dilihat pada Gambar 4. 23. *Line Chart* ini akan memperbarui data secara otomatis sesuai dengan pengaturan waktu yang dipilih (*real-time* atau dalam rentang waktu tertentu), dan dapat digunakan untuk analisis visual sederhana maupun pemantauan performa perangkat IoT secara berkelanjutan.



Gambar 4. 23. Tampilan Widget: Line Chart

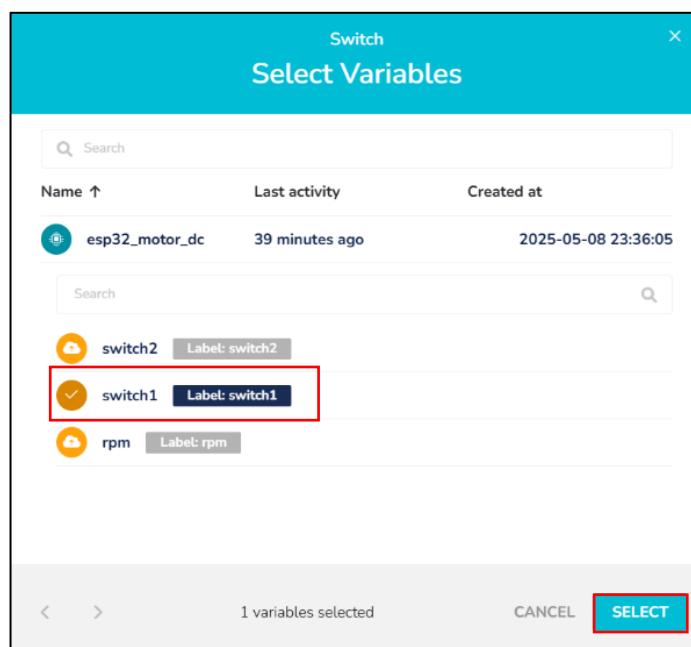
Proses selanjutnya adalah menambahkan variabel baru untuk menampung data dari *push button*. Klik tombol “(+)*Create new*” untuk membuat variabel baru. Beri nama “*switch1*” dan “*switch2*” pada masing-masing variabel tersebut. Variabel “*switch1*” difungsikan untuk kontrol *On* atau *Off*, sedangkan “*switch2*” digunakan untuk kontrol arah putaran motor (Maju atau Mundur).

Pastikan API *Label* disesuaikan dengan fungsi masing-masing variabel agar integrasi dengan kode program berjalan dengan baik.



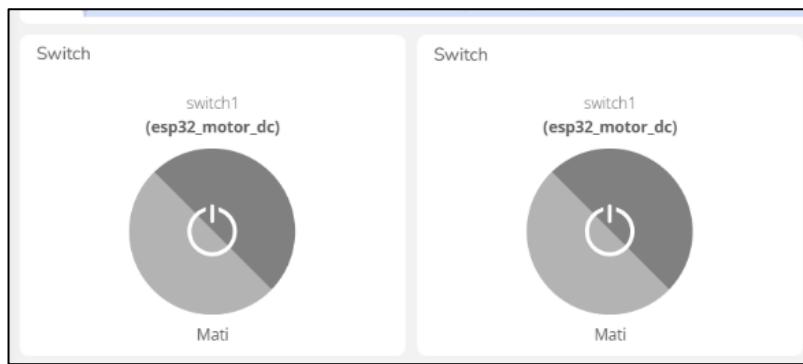
Gambar 4. 24. Hasil Pembuatan Variabel

Langkah selanjutnya adalah menambahkan “*Widget: Switch* (tombol)”. Setelah *widget* dipilih, tambahkan variabel dengan menekan “+ ADD VARIABLE”, lalu pilih variabel “switch1” yang berfungsi untuk mengaktifkan atau menonaktifkan perangkat (*On* atau *Off*). Tambahkan keterangan pada *widget* dengan label “Tombol Aktivasi Motor” agar memudahkan identifikasi fungsi pada dasbor.



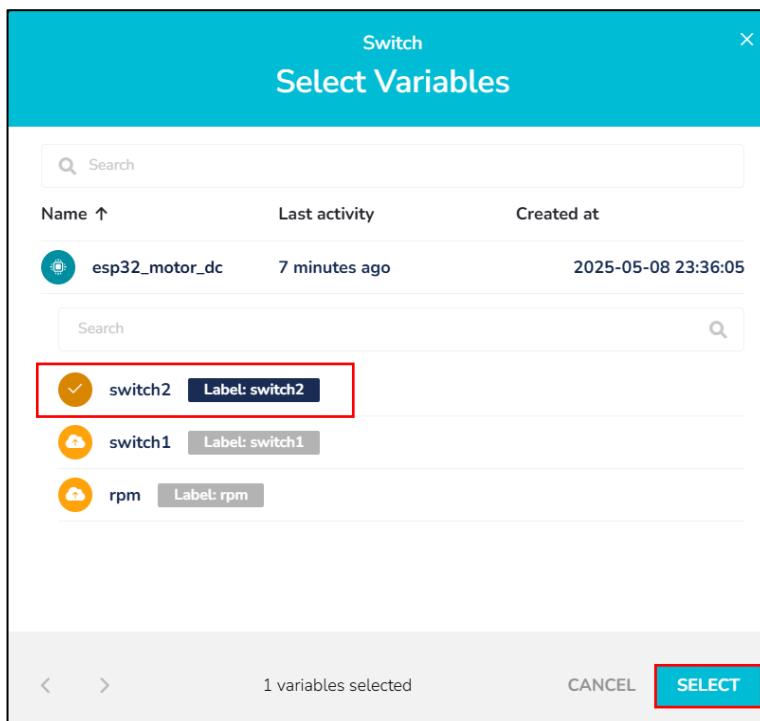
Gambar 4. 25. Memilih Variabel pada *Widget: Switch* 1

Langkah berikutnya, klik kanan pada “*Widget: Switch 1* (tombol)”, lalu pilih opsi “*Duplicate*” untuk menyalin *widget* sehingga dapat mempercepat proses pembuatan. Hasil duplikasi dapat dilihat pada Gambar 4. 26.



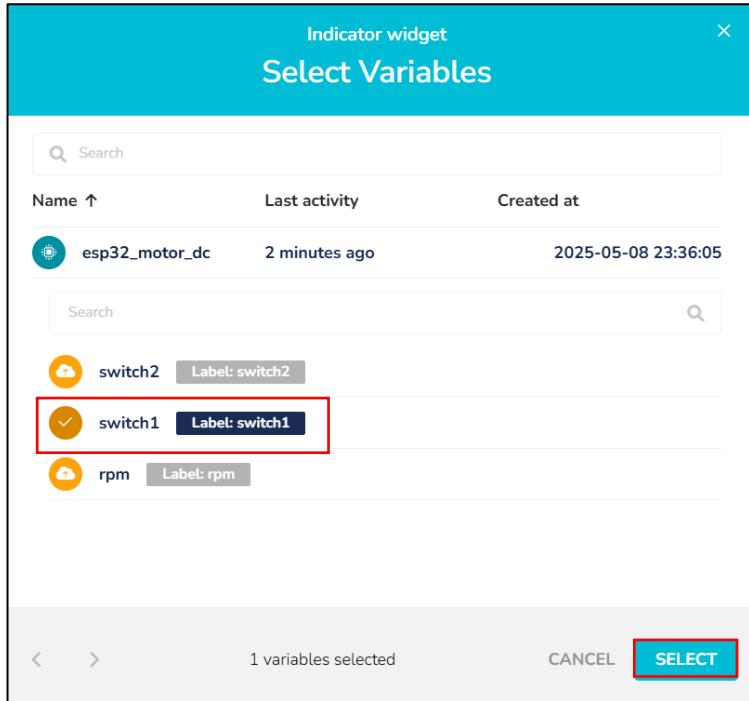
Gambar 4. 26. Hasil *Duplicate*

Ubah pengaturan variabel dari “switch1” menjadi “switch2” untuk mengatur fungsi arah putaran motor (maju atau mundur), lalu tambahkan keterangan “Tombol Rotasi Motor” agar memudahkan identifikasi fungsi pada dasbor.



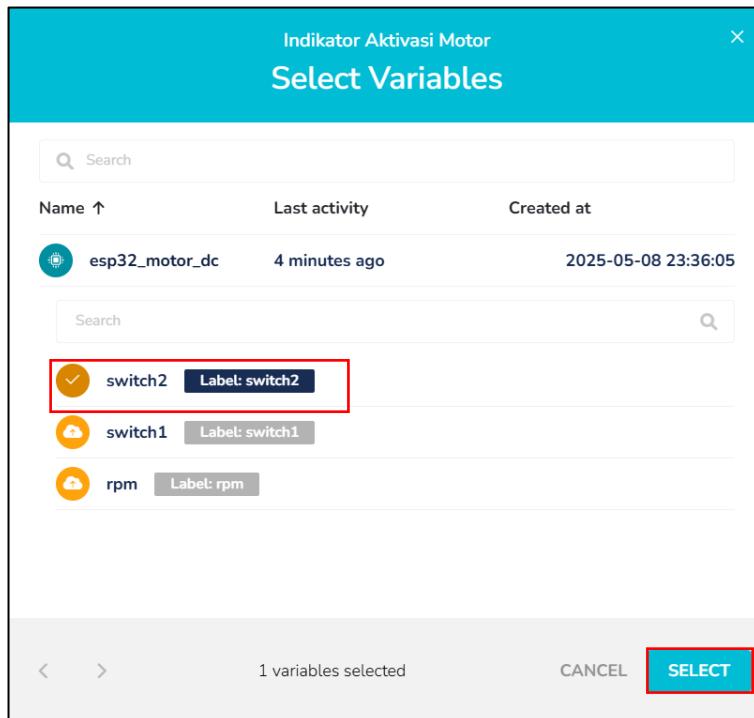
Gambar 4. 27. Memilih Variabel pada *Widget: Switch* 2

Langkah selanjutnya, tambahkan *widget* baru dan pilih jenis “Indicator” untuk menampilkan respons data secara visual. Setelah itu, tambahkan variabel dengan memilih “switch1” yang berfungsi sebagai kontrol *On* atau *Off*. Ubah nama *widget* menjadi “Indikator Aktivasi Motor”, lalu klik “SAVE” untuk menyimpan pengaturan.



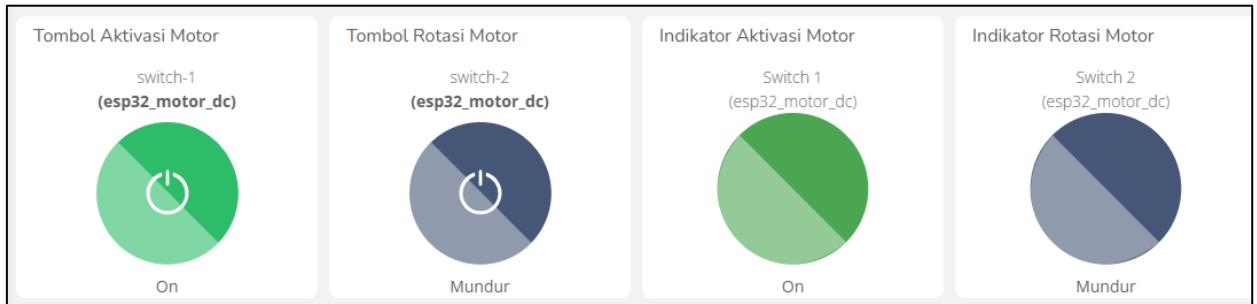
Gambar 4. 28. Memilih Variabel pada *Widget*: Indikator 1

Setelah itu, lakukan *duplicate* pada “*Widget*: Indikator 1” yang telah dibuat sebelumnya. Klik hasil duplikasi tersebut, kemudian tambahkan variabel baru. Pilih variabel “switch2” untuk fungsi Maju atau Mundur, lalu ubah nama *widget* menjadi “Indikator Rotasi Motor”. Setelah selesai, klik “*SAVE*” untuk menyimpan perubahan.



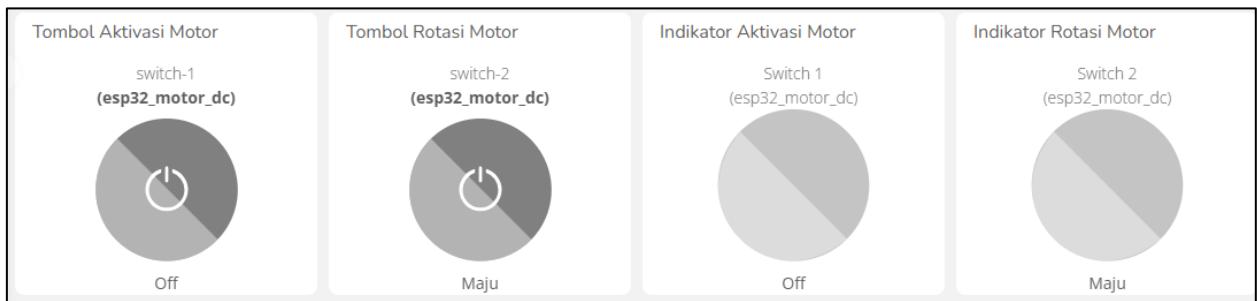
Gambar 4. 29. Memilih Variabel pada *Widget*: Indikator 2

Ketika “Widget: Switch” dinyalakan maka warna “Widget: Indikator” juga akan berubah seperti yang ditunjukkan pada Gambar 4. 30 berikut.



Gambar 4. 30. Tampilan Widget: Switch Ketika Dinyalakan

Apabila “Widget: Switch” dimatikan, maka warna “Widget: Indikator” juga akan mengikuti “Widget: Switch” seperti yang ditunjukkan pada Gambar 4. 31 berikut.



Gambar 4. 31. Tampilan Widget: Switch Ketika Dimatikan

Sebagai tambahan informasi mengenai pengaturan *Dashboards*, seperti pengubahan nama, pengubahan warna, pengubahan bentuk dan ukuran, itu bisa di *edit* sendiri sesuai kebutuhan. Jika ingin mendapatkan fitur yang lebih baik, Ubidots menyediakan paket premium untuk berlangganan.

4.2. Hasil Pengujian

4.2.1. Pengujian Konektivitas Wi-Fi

Ketika menguji konektivitas *Wi-Fi*, *Serial Monitor* Arduino IDE menampilkan proses konfigurasi *Wi-Fi* pada *board* ESP32. Sistem kemudian menampilkan pesan "Menghubungkan ke Wi-Fi", dan saat berhasil terhubung, muncul pesan "Berhasil tersambung ke Wi-Fi: HawinHmm". HawinHmm ini adalah nama *Wi-Fi / Hotspot* yang digunakan. Perangkat ESP32 juga telah memperoleh alamat IP lokal yang digunakan saat ini, yaitu 524724416.

```
=====
KONFIGURASI WI-FI
=====

Menghubungkan ke Wi-Fi.

=====
STATUS KONFIGURASI WI-FI =====
Berhasil tersambung ke Wi-Fi: HawinHm
IP Lokal: 524724416
```

Gambar 4. 32. Status *Wi-Fi* Berhasil Terhubung

Ketika menguji konektivitas *Wi-Fi*, perangkat yang awalnya terhubung kemudian tiba-tiba terputus, maka sistem secara otomatis melakukan penyambungan ulang, seperti yang terlihat pada Gambar 4. 33 berikut.

```
=====
STATUS KONFIGURASI WI-FI =====
Wi-Fi terputus!
Mencoba menghubungkan kembali...

=====
KONFIGURASI WI-FI
=====

Menghubungkan ke Wi-Fi.....
```

Gambar 4. 33. *Wi-Fi* Mencoba Menghubungkan Kembali

Ketika menguji konektivitas *Wi-Fi*, namun perangkat gagal terhubung, maka sistem secara otomatis menampilkan pesan "!!! GAGAL TERHUBUNG KE WI-FI !!!" yang disertai peringatan untuk memeriksa kembali SSID dan *password Wi-Fi* yang digunakan. Sistem kemudian memberikan informasi untuk mencoba lagi nanti.

```
=====
KONFIGURASI WI-FI
=====

Menghubungkan ke Wi-Fi.....
```

=====
STATUS KONFIGURASI WI-FI =====

!!! GAGAL TERHUBUNG KE WI-FI !!!

Silakan periksa SSID dan Password WiFi Anda.

ESP32 akan mencoba lagi nanti.

Gambar 4. 34. *Wi-Fi* Gagal Terhubung

4.2.2. Pengujian Konektivitas IoT

Tampilan *Serial Monitor* Arduino IDE menunjukkan bahwa ESP32 berhasil terhubung ke platform IoT Ubidots, menandakan bahwa sistem siap untuk digunakan dalam komunikasi dan pengiriman data IoT.

```
=====
KONFIGURASI IOT PLATFORM: UBIDOTS
=====

Menghubungkan ke Ubidots.....


===== STATUS KONFIGURASI UBIDOTS =====
Berhasil tersambung ke IoT Platform: Ubidots
```

Gambar 4. 35. IoT Berhasil Tersambung

Tampilan *Serial Monitor* Arduino IDE menunjukkan bahwa ESP32 gagal terhubung ke platform IoT Ubidots dengan kode *error -2*. Kegagalan ini kemungkinan disebabkan oleh *Wi-Fi* yang tidak aktif atau alamat *MQTT Server* yang salah. Sistem akan mencoba kembali dalam 5 detik.

```
=====
KONFIGURASI IOT PLATFORM: UBIDOTS
=====

Menghubungkan ke Ubidots.....


===== STATUS KONFIGURASI UBIDOTS =====
Gagal, Kode Error = -2
• WiFi / Hotspot Anda tidak aktif !!!
• Atau alamat Server MQTT yang Anda masukkan salah !!!
• Coba lagi 5 detik.
```

Gambar 4. 36. IoT Gagal Tersambung dengan Kode *Error -2*

Tampilan *Serial Monitor* Arduino IDE menunjukkan bahwa ESP32 gagal terhubung ke platform IoT Ubidots dengan kode *error -4*. Kegagalan ini disebabkan oleh tidak adanya akses internet atau paket data pada jaringan yang digunakan. Sistem akan mencoba kembali dalam 5 detik.

```
=====
KONFIGURASI IOT PLATFORM: UBIDOTS
=====

Menghubungkan ke Ubidots.....


===== STATUS KONFIGURASI UBIDOTS =====

Gagal, Kode Error = -4
• Tidak ada akses internet / paket data !!!
• Coba lagi 5 detik.
```

Gambar 4. 37. IoT Gagal Tersambung dengan Kode *Error* -4

Tampilan *Serial Monitor* Arduino IDE menunjukkan bahwa ESP32 gagal terhubung ke platform IoT Ubidots dengan kode *error* 5. Kegagalan ini disebabkan oleh adanya Token Ubidots yang salah. Sistem akan mencoba kembali dalam 5 detik.

```
=====
KONFIGURASI IOT PLATFORM: UBIDOTS
=====

Menghubungkan ke Ubidots.....


===== STATUS KONFIGURASI UBIDOTS =====

Gagal, Kode Error = 5
• Token Ubidots yang Anda masukkan salah !!!
• Coba lagi 5 detik.
```

Gambar 4. 38. IoT Gagal Tersambung dengan Kode *Error* 5

4.2.3. Pengujian *Publish–Subscribe* pada MQTT

Pengujian *publish* dan *subscribe* pada protokol MQTT dilakukan untuk memastikan komunikasi data antara perangkat ESP32 dan *broker* berjalan dengan baik. Dalam pengujian ini, ESP32 berhasil menerima (*subscribe*) data dari variabel (topik):

- /v1.6/devices/esp32_motor_dc/rpm/lv (RPM Motor).
- /v1.6/devices/esp32_motor_dc/switch1/lv (Aktivasi Motor).
- /v1.6/devices/esp32_motor_dc/switch2/lv (Arah Rotasi Motor).

Selain itu, ESP32 juga berhasil mengirim (*publish*) data ke variabel (topik): /v1.6/devices/esp32_motor_dc dalam format JSON, yang memuat informasi nilai rpm, status *switch1*, dan *switch2*.

Hasil ini menunjukkan bahwa komunikasi dua arah menggunakan protokol MQTT telah berjalan sesuai harapan.

```
=====
SUBSCRIBE MQTT
=====
Topic: /v1.6/devices/esp32_motor_dc/rpm/lv
RPM: 312
=====
SUBSCRIBE MQTT
=====
Topic: /v1.6/devices/esp32_motor_dc/switch1/lv
Status Aktivasi Motor: Menyala
=====
SUBSCRIBE MQTT
=====
Topic: /v1.6/devices/esp32_motor_dc/switch2/lv
Arah Rotasi Motor: Mundur
=====
PUBLISH MQTT
=====
Topic: /v1.6/devices/esp32_motor_dc
Json Data: {"rpm":341,"switch1":1,"switch2":1}
```

Gambar 4. 39. Publish dan Subscribe

4.2.4. Pengujian LCD I2C

Pengujian LCD I2C dilakukan untuk memastikan bahwa layar dapat menampilkan informasi secara benar setelah terhubung dengan ESP32. Jika semua informasi tampil sesuai harapan, maka LCD I2C dinyatakan berhasil teruji dan siap digunakan dalam aplikasi utama untuk menampilkan status motor dan nilai RPM.



Gambar 4. 40. Tampilan Animasi Awal

Tampilan LCD I2C ketika sistem baru dinyalakan menampilkan status *Loading*.



Gambar 4. 41. Tampilan *Loading*

Tampilan LCD I2C ketika menampilkan status *STANDBY*.



Gambar 4. 42. Tampilan *STANDBY*

Tampilan LCD I2C ketika menampilkan status *OFF*.



Gambar 4. 43. Tampilan *OFF*

Tampilan LCD I2C ketika menampilkan kecepatan putaran (RPM) motor DC.



Gambar 4. 44. Tampilan Kecepatan Putaran (RPM) Motor DC secara aktual pada LCD

4.2.5. Pengujian Kecepatan Putaran (RPM) Motor DC pada Platform Ubidots

Pengukuran kecepatan putaran (RPM) motor DC dalam rentang waktu tertentu dilakukan menggunakan platform Ubidots. Hasil pengukuran ditampilkan pada Tabel 4. 1 berikut.

Tabel 4. 1. Monitoring Kecepatan Putaran (RPM) Motor DC pada Platform Ubidots

Date	Created At	Timestamp	Value
2025-06-08 13:21:29.350000+07:00	1,749E+12	1,75E+12	210
2025-06-08 13:21:24.352000+07:00	1,749E+12	1,75E+12	210
2025-06-08 13:21:19.350000+07:00	1,749E+12	1,75E+12	261
2025-06-08 13:21:14.404000+07:00	1,749E+12	1,75E+12	382
2025-06-08 13:21:09.366000+07:00	1,749E+12	1,75E+12	312
2025-06-08 13:21:04.308000+07:00	1,749E+12	1,75E+12	329
2025-06-08 13:20:59.305000+07:00	1,749E+12	1,75E+12	300
2025-06-08 13:20:54.288000+07:00	1,749E+12	1,75E+12	319
2025-06-08 13:20:49.126000+07:00	1,749E+12	1,75E+12	326
2025-06-08 13:20:44.071000+07:00	1,749E+12	1,75E+12	287
2025-06-08 13:20:38.990000+07:00	1,749E+12	1,75E+12	316
2025-06-08 13:20:33.945000+07:00	1,749E+12	1,75E+12	333
2025-06-08 13:20:28.861000+07:00	1,749E+12	1,75E+12	324
2025-06-08 13:20:23.842000+07:00	1,749E+12	1,75E+12	382
2025-06-08 13:20:18.844000+07:00	1,749E+12	1,75E+12	316
2025-06-08 13:20:13.766000+07:00	1,749E+12	1,75E+12	220
2025-06-08 13:20:08.762000+07:00	1,749E+12	1,75E+12	377
2025-06-08 13:20:03.764000+07:00	1,749E+12	1,75E+12	326
2025-06-08 13:19:58.677000+07:00	1,749E+12	1,75E+12	329
2025-06-08 13:19:53.587000+07:00	1,749E+12	1,75E+12	321
2025-06-08 13:19:48.493000+07:00	1,749E+12	1,75E+12	319
2025-06-08 13:19:43.403000+07:00	1,749E+12	1,75E+12	331
2025-06-08 13:19:38.315000+07:00	1,749E+12	1,75E+12	312
2025-06-08 13:19:33.222000+07:00	1,749E+12	1,75E+12	273
2025-06-08 13:19:28.225000+07:00	1,749E+12	1,75E+12	316
2025-06-08 13:19:23.223000+07:00	1,749E+12	1,75E+12	304

2025-06-08 13:19:18.195000+07:00	1,749E+12	1,75E+12	212
2025-06-08 13:19:13.197000+07:00	1,749E+12	1,75E+12	290
2025-06-08 13:19:08.245000+07:00	1,749E+12	1,75E+12	307
2025-06-08 13:19:03.121000+07:00	1,749E+12	1,75E+12	314

Tampilan grafik kecepatan putaran (RPM) Motor DC pada platform Ubidots ditampilkan pada Gambar 4. 45 berikut.



Gambar 4. 45. Tampilan Kecepatan Putaran (RPM) Motor DC pada *Line Charts*

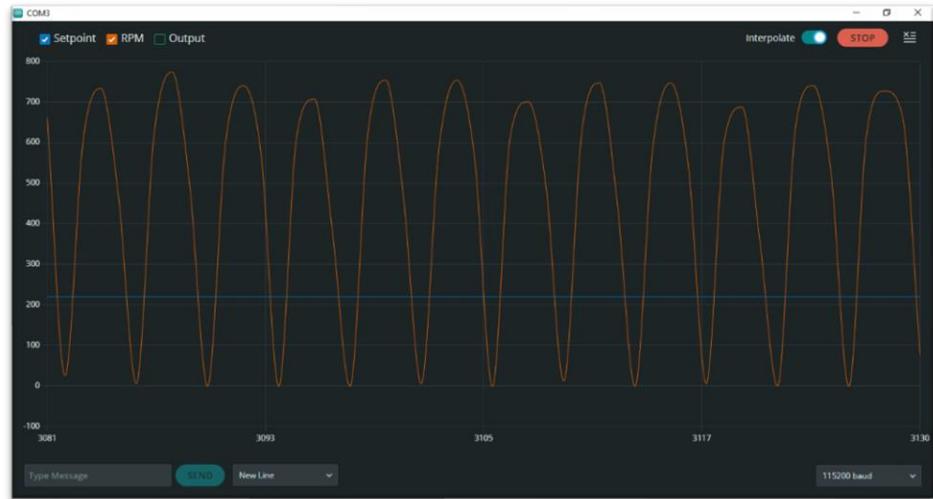
4.2.6. Pengujian Kecepatan Putaran (RPM) Motor DC pada Arduino IDE

Proses pengukuran kecepatan putaran (RPM) motor DC dalam interval waktu tertentu dilakukan menggunakan Arduino IDE, dengan hasil yang ditampilkan pada *Serial Monitor*, sebagaimana ditunjukkan pada Gambar 4. 46 di bawah ini.

```
462.00 533.33 71.33
462.00 280.00 182.00
462.00 680.00 218.00
462.00 726.67 264.67
462.00 446.67 15.33
462.00 813.33 351.33
462.00 813.33 351.33
462.00 360.00 102.00
462.00 726.67 264.67
462.00 840.00 378.00
462.00 466.67 4.67
462.00 726.67 264.67
462.00 853.33 391.33
462.00 566.67 104.67
462.00 280.00 182.00
462.00 673.33 211.33
462.00 786.67 324.67
462.00 506.67 44.67
462.00 813.33 351.33
462.00 800.00 338.00
```

Gambar 4. 46. Tampilan Kecepatan Putaran (RPM) Motor DC pada Arduino IDE

Grafik hasil kendali PID *Ziegler-Nichols* Metode 2 terhadap kecepatan putaran (RPM) motor DC divisualisasikan melalui *Serial Plotter*, sebagaimana ditunjukkan pada Gambar 4. 47.



Gambar 4. 47. Grafik Kendali PID ZN-2 terhadap Kecepatan Putaran (RPM) Motor DC

Hasil pengujian menunjukkan bahwa parameter PID paling sesuai untuk sistem ini berdasarkan teknik *Ziegler-Nichols* Metode 2 adalah $K_p = 7.2$, $K_i = 48$, dan $K_d = 0.27$. Nilai tersebut memberikan kinerja pengendalian yang stabil dan responsif, serta mampu menyesuaikan dengan karakteristik dinamis motor DC dan mekanisme *Conveyor*.

4.2.7. Pengujian Kecepatan Putaran (RPM) Motor DC dengan *Digital Tachometer*

Untuk melakukan pengujian, biasanya permukaan benda (seperti poros motor) ditempeli stiker reflektif. Kemudian, sinar laser atau inframerah dari *tachometer* diarahkan ke stiker tersebut. Setiap kali stiker melewati sensor, alat mencatat satu putaran. Dalam waktu tertentu, *tachometer* menghitung jumlah putaran dan mengubahnya menjadi nilai RPM yang ditampilkan secara digital.



Gambar 4. 48. Pengujian dengan *Digital Tachometer*

Hasil pengukuran kecepatan putaran (RPM) menggunakan alat *Tachometer* ditampilkan pada Tabel 4. 2. Data ini digunakan sebagai acuan untuk validasi performa sistem pengendalian Motor DC.

Tabel 4. 2. Hasil Pengukuran *Digital Tachometer*

No	Pengukuran ke-	Hasil pengukuran
1.	Pengukuran ke-1	370,5 RPM
2.	Pengukuran ke-2	393,9 RPM
3.	Pengukuran ke-3	218,1 RPM
4.	Pengukuran ke-4	146,1 RPM
5.	Pengukuran ke-5	288,7 RPM
6.	Pengukuran ke-6	463,4 RPM
7.	Pengukuran ke-7	350,0 RPM
8.	Pengukuran ke-8	335,4 RPM
9.	Pengukuran ke-9	210,5 RPM
10.	Pengukuran ke-10	114,4 RPM
11.	Pengukuran ke-11	47,7 RPM
12.	Pengukuran ke-12	144,5 RPM
13.	Pengukuran ke-13	88,0 RPM
14.	Pengukuran ke-14	99,1 RPM
15.	Pengukuran ke-15	149,9 RPM
16.	Pengukuran ke-16	238,1 RPM
17.	Pengukuran ke-17	244,8 RPM
18.	Pengukuran ke-18	188,0 RPM
19.	Pengukuran ke-19	113,4 RPM
20.	Pengukuran ke-20	44,4 RPM

4.2.8. Pengujian Tombol Fisik dan Non-Fisik

Berikut adalah hasil pengujian terhadap sistem kendali motor menggunakan tombol fisik dan non-fisik (Ubidots). Pengujian dilakukan untuk memastikan bahwa setiap jenis tombol dapat menjalankan fungsinya dengan baik, seperti menyalakan (*ON*), mematikan (*OFF*), serta mengatur arah putaran motor (Maju / Mundur). Berdasarkan Tabel 4. 3, hasil menunjukkan bahwa sistem bekerja sesuai dengan yang diharapkan.

Tabel 4. 3. Hasil Pengujian Tombol

No	Jenis Tombol	Fungsi	Aksi
1.	Non-Fisik (Ubidots)	<i>ON</i>	Sukses
2.	Non-Fisik (Ubidots)	Maju	Sukses
3.	Fisik	<i>OFF</i>	Sukses
4.	Non-Fisik (Ubidots)	<i>ON</i>	Sukses
5.	Non-Fisik (Ubidots)	<i>OFF</i>	Sukses
6.	Non-Fisik (Ubidots)	Mundur	Sukses
7.	Fisik	<i>ON</i>	Sukses
8.	Fisik	Maju	Sukses
9.	Non-Fisik (Ubidots)	<i>OFF</i>	Sukses
10.	Fisik	Mundur	Sukses
11.	Fisik	<i>ON</i>	Sukses
12.	Fisik	<i>OFF</i>	Sukses
13.	Fisik	Maju	Sukses
14.	Fisik	<i>ON</i>	Sukses
15.	Non-Fisik (Ubidots)	Mundur	Sukses
16.	Non-Fisik (Ubidots)	Maju	Sukses
17.	Non-Fisik (Ubidots)	<i>OFF</i>	Sukses
18.	Non-Fisik (Ubidots)	<i>ON</i>	Sukses
19.	Fisik	<i>OFF</i>	Sukses
20.	Fisik	Maju	Sukses
21.	Non-Fisik (Ubidots)	<i>ON</i>	Sukses
22.	Non-Fisik (Ubidots)	Mundur	Sukses
23.	Fisik	Maju	Sukses
24.	Fisik	Mundur	Sukses
25.	Non-Fisik (Ubidots)	<i>OFF</i>	Sukses

4.2.9. Pengujian Efek Bobot Barang terhadap Waktu Tempuh *Conveyor*

Pengujian ini bertujuan untuk mengetahui waktu tempuh masing-masing barang saat melewati jalur *Conveyor* dengan kecepatan motor tertentu. Jarak antara titik awal dan titik akhir *Conveyor* sekitar ± 30 cm. Setiap barang ditempatkan di titik awal, kemudian waktu tempuhnya

dicatat saat mencapai titik akhir. Variasi hasil pengujian dipengaruhi oleh berat dan bentuk fisik barang, yang turut memengaruhi kecepatan geraknya di atas permukaan *Conveyor*.

Tabel 4. 4. Hasil Pengujian Waktu Tempuh Barang

No	Nama Benda	Berat	Waktu tempuh pada kecepatan	
			212 (RPM)	492 (RPM)
1.	Air Le Mineral Botol 600 ml	645 gram	17,2 detik	9,3 detik
2.	Taffware 117 in 1 Toolset Obeng	480 gram	6 detik	1,5 detik
3.	Air Cleo Botol 220 ml	224 gram	1,2 detik	1 detik
4.	HDD Eksternal	166 gram	1,5 detik	0,8 detik
5.	Pembolong Kertas Kenko	129 gram	1,4 detik	0,7 detik

BAB V

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Berdasarkan penelitian yang telah dilakukan oleh penulis, maka dapat ditarik beberapa kesimpulan yaitu meliputi :

1. Metodologi pengembangan sistem menggunakan *Rapid Application Development* (RAD), yang terbukti efisien dalam membangun sistem prototipe dengan waktu terbatas, tanpa mengurangi kualitas hasil akhir.
2. Kendali *On / Off* dan kendali rotasi Motor DC berhasil diimplementasikan, baik dari jarak jauh maupun jarak dekat.
3. Kendali PID *Ziegler-Nichols* 2 berhasil diimplementasikan pada sistem kendali kecepatan putaran (RPM) Motor DC pada *Conveyor* dengan parameter *tuning* akhir $K_p = 7.2$, $K_i = 48$, dan $K_d = 0.27$, sehingga menunjukkan efektivitas dalam mencapai kestabilan rotasi motor secara berkelanjutan.
4. Komunikasi data pada sistem ini menggunakan protokol MQTT, yang dikenal ringan dan andal dalam aplikasi *Internet of Things* (IoT). Sebagai platform IoT, Ubidots digunakan untuk memantau serta mengendalikan sistem secara *real-time* melalui jaringan internet.
5. Selama nilai potensiometer berada pada posisi nol (0), sistem *Conveyor* tidak akan menyala, meskipun perintah pengaktifan telah dikirim melalui platform Ubidots. Kondisi ini menunjukkan bahwa pengaturan kecepatan masih memerlukan intervensi langsung dari operator di lokasi. Oleh karena itu, sistem perlu dikembangkan lebih lanjut agar pengaturan kecepatan putaran (RPM) motor DC dapat dilakukan melalui platform IoT, sehingga efisiensi dan fleksibilitas operasional dapat ditingkatkan.
6. Hasil pengujian menunjukkan bahwa sistem dapat mempertahankan kecepatan mendekati *Setpoint* meskipun terjadi perubahan beban. Ini membuktikan bahwa penerapan kendali PID pada sistem *Conveyor* telah memberikan performa yang andal.

5.2. Saran

Berdasarkan penelitian yang telah dilakukan oleh penulis, didapatkan beberapa kekurangan, sehingga diperlukan saran yang sesuai sebagai pelengkap pada penelitian ke depannya yaitu meliputi :

1. Penggunaan bahan akrilik, agar terlihat lebih elegan, lebih aman karena tidak mudah pecah akibat benturan, dan dapat melindungi perangkat listrik dari cipratan air.
2. Penggunaan mekanisme *Pulley* sebagai alternatif sistem penggerak pada *Conveyor* dapat dipertimbangkan, mengingat sistem ini dinilai mampu meningkatkan efisiensi transmisi daya, mengurangi beban langsung pada motor, serta meningkatkan kinerja mekanis dan efisiensi energi secara keseluruhan.
3. Penggunaan alternatif *framework* selain Arduino IDE, seperti ESP-IDF atau Platform IO, dapat dipertimbangkan untuk mendukung pengembangan yang lebih profesional.
4. Pemanfaatan *MicroPython* sebagai alternatif bahasa pemrograman selain Arduino (C/C++) dapat dipertimbangkan untuk menyederhanakan proses pengembangan, khususnya dalam pembuatan prototipe berbasis ESP32.
5. Penggabungan metode atau algoritma lain dapat dilakukan untuk meningkatkan akurasi dan adaptivitas kendali PID *Ziegler-Nichols* terhadap variasi beban dan gangguan sistem.
6. Penambahan komponen IoT seperti sensor jarak, sensor beban, sensor warna, RTC *module*, *motor servo*, dan komponen lain yang sekiranya dapat mengembangkan penelitian ini lebih jauh.
7. Penambahan potensiometer virtual pada Ubidots untuk meningkatkan fleksibilitas pengaturan kecepatan motor DC secara *remote*.
8. Peningkatan ke versi *Premium* Ubidots atau migrasi ke platform lain seperti ThingsBoard, Blynk, atau Node-RED untuk menunjang skalabilitas dan keandalan sistem IoT.
9. Penambahan integrasi *multi-platform* untuk meningkatkan fleksibilitas akses lintas *device*.
10. Penggunaan metode OTA (*Over-The-Air*) pada sistem IoT untuk pembaruan *Firmware* secara nirkabel, sehingga lebih praktis dan efisien.

LAMPIRAN

Berikut adalah beberapa lampiran dari pembuatan kerangka *Conveyor*.

1. Pemotongan triplek.



2. Pengampelasan triplek.



3. Pemotongan pipa.



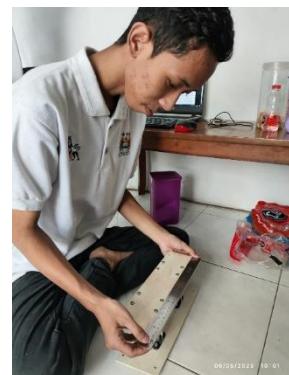
4. Pelubangan rangka dengan mesin bor.



5. Pemasangan baut.



6. Pengukuran rangka *Conveyor*.



7. Pembuatan *roller*.



8. Pemotongan kain oscar.



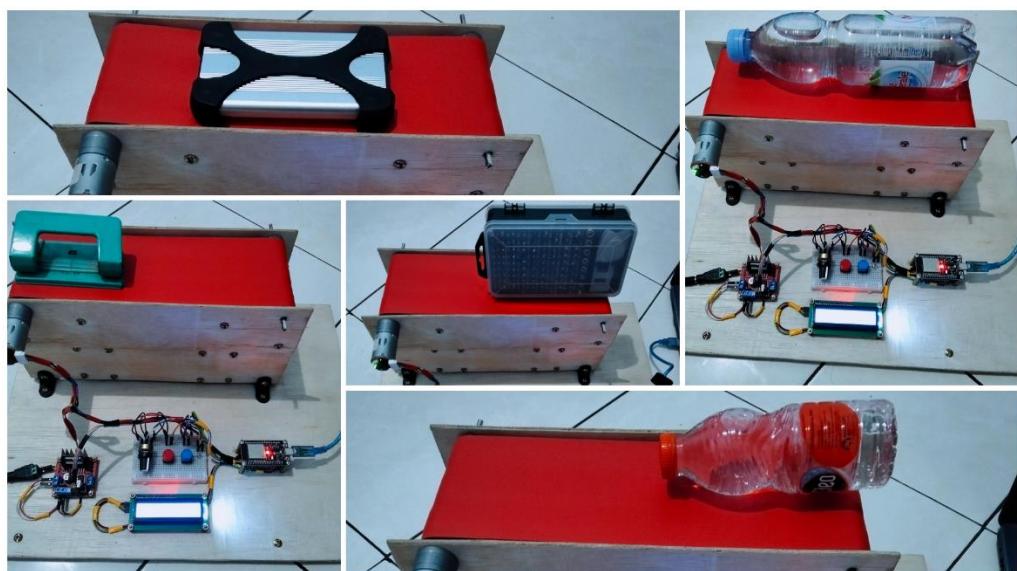
9. Kerangka *Conveyor* setengah jadi.



Proses penimbangan barang untuk uji pengaruh beban berat dan ringan pada Conveyor.



Uji pengaruh beban berat dan ringan pada Conveyor.



DAFTAR PUSTAKA

- [1] F. Tamimi and S. Munawaroh, “Teknologi Sebagai Kegiatan Manusia Dalam Era Modern Kehidupan Masyarakat,” *Saturnus J. Teknol. dan Sist. Inf.*, vol. 2, no. 3, pp. 66–74, 2024, doi: <https://doi.org/10.61132/saturnus.v2i3.157>.
- [2] B. M. Danta, S. R. Sentiuwo, and M. D. Putro, “Implementasi Teknologi Radio Frequency Identification untuk Identifikasi dan Autentikasi pada Gerbang Masuk diUniversitas Sam Ratulangi Manado,” *J. Tek. Inform.*, vol. 9, no. 1, pp. 1–9, 2016, doi: 10.35793/jti.9.1.2016.13458.
- [3] Nurhanudin and Kartimi, “Memahami Penciptaan, Perkembangan, dan Tantangan Manusia di Era Digital,” *J. Educ.*, vol. 7, no. 2, pp. 9283–9292, 2025, doi: <https://doi.org/10.31004/joe.v7i2.7868>.
- [4] R. Parlika, D. C. M. Wijaya, and A. Pratama, “BOT PENYIMPAN DATA PENGUMPULAN TUGAS PESERTA ELEARNING BERBASIS TELEGRAM [ER-BOT PDPT],” *SCAN - J. Teknol. Inf. dan Komun.*, vol. 16, no. 1, pp. 34–41, 2021, doi: <https://doi.org/10.33005/scan.v16i1.2352>.
- [5] D. Sinaga and Peniarsih, “MENGHADAPI PERUBAHAN DUNIA MELALUI TRANSFORMASI DIGITAL MENUJU KESUKSESAN PADA ERA DIGITALISASI,” *JSI (Jurnal Sist. Informasi) Univ. Suryadarma*, vol. 11, no. 2, pp. 51–58, 2024, doi: <https://doi.org/10.31219/osf.io/gdm68>.
- [6] M. R. Fahlevi, V. Naubnome, and F. C. Suci, “Rancang Bangun Mesin Belt Conveyor Dengan Speedcontrol Sebagai Pengatur Kecepatan,” *Patria Artha Technol. J.*, vol. 7, no. 1, pp. 20–26, 2023, doi: 10.33857/patj.v7i1.593.
- [7] B. D. Ushofa, L. Anifah, I. G. P. A. Budijahjanto, and Endryansyah, “Sistem Kendali Kecepatan Putaran Motor DC pada Conveyor dengan Metode Kontrol PID,” *J. Tek. Elektro*, vol. 11, no. 2, pp. 332–342, 2022, doi: <https://doi.org/10.26740/jte.v11n2.p332-342>.
- [8] M. R. A. N. Putera and R. Hidayat, “Kendali Kecepatan Motor DC Menggunakan Pengendali PID dengan Encoder sebagai Feedback,” *STRING (Satuan Tulisan Ris. dan Inov. Teknol.)*, vol. 7, no. 1, pp. 50–56, 2022, doi: 10.30998/string.v7i1.13026.
- [9] N. Soedjarwanto, “PENGENDALIAN KECEPATAN MOTOR DC MENGGUNAKAN BUCK-BOOST CONVERTER BERBASIS IoT,” *J. Inform. dan Tek. Elektro Terap.*, vol.

11, no. 3s1, pp. 943–950, 2023, doi: 10.23960/jitet.v11i3s1.3399.

- [10] M. Diah Ika Putri, A. Ma’arif, and R. Dwi Puriyanto, “Pengendali Kecepatan Sudut Motor DC Menggunakan Kontrol PID dan Tuning Ziegler Nichols,” *Techno (Jurnal Fak. Tek. Univ. Muhammadiyah Purwokerto)*, vol. 23, no. 1, pp. 9–18, 2022, doi: 10.30595/techno.v23i1.10773.
- [11] K. Gunawan, Sugiarto, and J. Prasojo, “SISTEM KENDALI KECEPATAN MOTOR DC MENGGUNAKAN HYBRID PID-PSO,” *JMTE (Jurnal Mhs. Tek. Elektro)*, vol. 5, no. 1, pp. 1–9, 2024, [Online]. Available: <https://journal.itny.ac.id/index.php/JMTE/article/view/4915>.
- [12] K. J. Åström and T. Hägglund, “The future of PID control,” *Control Eng. Pract.*, vol. 9, no. 11, pp. 1163–1175, Nov. 2001, doi: 10.1016/S0967-0661(01)00062-4.
- [13] K. J. Åström and T. Hägglund, “Revisiting the Ziegler–Nichols step response method for PID control,” *J. Process Control*, vol. 14, no. 6, pp. 635–650, 2004, doi: 10.1016/j.jprocont.2004.01.002.
- [14] A. R. Utami, R. J. Yuniar, A. Giyantara, and A. D. Saputra, “Cohen-Coon PID Tuning Method for Self-Balancing Robot,” in *2022 International Symposium on Electronics and Smart Devices (ISESD)*, IEEE, 2022, pp. 1–5. doi: 10.1109/ISESD56103.2022.9980830.
- [15] C. Kravaris and I. K. Kookos, *Understanding Process Dynamics and Control*, vol. 27, no. 2. Cambridge University Press, 2021. doi: 10.1017/9781139565080.
- [16] D. D. O. Francisco, J. O. Trierweiler, and M. Farenzena, “A novel PID autotuning approach: how to correct bad tuning by closed-loop performance assessment,” *IFAC-PapersOnLine*, vol. 52, no. 1, pp. 184–189, 2019, doi: 10.1016/J.IFACOL.2019.06.058.
- [17] D. C. Nunes, J. E. M. G. Pinto, D. G. V. Fonseca, A. L. Maitelli, and F. M. U. Araújo, “Relay Based PID Auto-tuning Applied to a Multivariable Level Control System,” in *Proceedings of the 11th International Conference on Informatics in Control, Automation and Robotics*, SCITEPRESS - Science and Technology Publications, 2014, pp. 741–748. doi: 10.5220/0005063907410748.
- [18] A. Ma’arif, *Dasar Kendali Sistem: Pemodelan, Pengendalian, Analisis, Simulasi, dan Implementasi*, 1st ed. Yogyakarta: UAD PRESS, 2021. [Online]. Available: <https://eprints.uad.ac.id/32516/>.
- [19] B. Rahmat and Muljono, *Pemrograman Internet of Things (IoT) DENGAN ARDUINO DAN*

PYTHON Jilid 1, 1st ed. Purbalingga: CV. EUREKA MEDIA AKSARA, 2024. [Online]. Available: <https://repository.penerbiteureka.com/publications/568337/pemrograman-internet-of-things-iot-dengan-arduino-dan-python-jilid-1>.

- [20] Z. Jamal, “IMPLEMENTASI KENDALI PID PENALAAAN ZIEGLER-NICHOLS MENGGUNAKAN MIKROKONTROLER,” *J. Inform.*, vol. 15, no. 1, pp. 81–88, 2015, doi: <https://doi.org/10.30873/ji.v15i1.410>.
- [21] A. R. J. Wiriawan and A. Irawan, “Motor DC Speed Adjustment By Propotional Integral Derivative (PID) Based on LabView,” *Telekontran J. Ilm. Telekomun. Kendali dan Elektron. Terap.*, vol. 4, no. 2, pp. 13–24, 2016, doi: 10.34010/telekontran.v4i2.1886.
- [22] R. Hansza and S. I. Haryudo, “Rancang Bangun Kontrol Motor DC dengan PID Menggunakan Perintah Suara dan Monitoring Berbasis Internet of Things (IOT),” *J. Tek. Elektro*, vol. 9, no. 2, pp. 477–485, 2020, doi: <https://doi.org/10.26740/jte.v9n2.p%25p>.
- [23] D. Setiawan, “Sistem Kontrol Motor Dc Menggunakan Pwm Arduino Berbasis Android System,” *J. Sains, Teknol. dan Ind.*, vol. 15, no. 1, pp. 7–14, 2017, doi: <http://dx.doi.org/10.24014/sitekin.v15i1.4131>.
- [24] D. C. M. Wijaya, “Kendali dan Monitoring pH Air Akuaponik Berbasis IoT Dengan Metode Fuzzy Type-2,” UPN “Veteran” Jawa Timur, 2022. [Online]. Available: <http://repository.upnjatim.ac.id/id/eprint/7014>.
- [25] Rosalina, I. Qosim, and M. Mujirudin, “Analisis Pengaturan Kecepatan Motor DC Menggunakan Kontrol PID (Proportional Integral Derivative),” *Proceeding TEKNOKA Natl. Semin.*, vol. 2, pp. 89–94, 2017, doi: <https://doi.org/10.30595/techno.v23i1.10773>.
- [26] M. Nizam, H. Yuana, and Z. Wulansari, “MIKROKONTROLER ESP 32 SEBAGAI ALAT MONITORING PINTU BERBASIS WEB,” *JATI (Jurnal Mhs. Tek. Inform.)*, vol. 6, no. 2, pp. 767–772, 2022, doi: 10.36040/jati.v6i2.5713.
- [27] Espressif Systems, “ESP32-DevKitC Getting Started Guide,” 2024. [Online]. Available: <https://docs.espressif.com/projects/esp-dev-kits/en/latest/esp32/esp32-devkitc/index.html>.
- [28] CNC Store, “MOTOR DC JGA25-370 DC 12V GEARBOX HIGH TORQUE DINAMO WITH DISC ENCODER,” CNC Store. Accessed: May 21, 2025. [Online]. Available: <https://shopee.co.id/MOTOR-DC-JGA25-370-DC-12V-GEARBOX-HIGH-TORQUE-DINAMO-WITH-DISC-ENCODER-i.62956347.26412883166>.
- [29] R. Sinaga, “Pengendali Kecepatan Motor Dc Menggunakan Sensor Hall Berbasis

Mikrokontroler Atmega 8535,” *Saintia Fis.*, vol. 1, no. 1, 2013.

- [30] I. Hudati, A. P. Aji, and S. Nurrahma, “Kendali Posisi Motor DC dengan Menggunakan Kendali PID,” *J. List. Instrumentasi dan Elektron. Terap.*, vol. 2, no. 2, pp. 25–30, 2021, doi: 10.22146/juliet.v2i2.71148.
- [31] F. Musthofa and H. Winarno, “SISTEM DESELERASI KECEPATAN OTOMATIS PADA MOBIL BERDASARKAN JARAK MENGGUNAKAN SENSOR ULTRASONIK HC-SR04 BERBASIS ARDUINO MEGA 2560,” *Gema Teknol.*, vol. 18, no. 3, pp. 110–116, 2015, doi: 10.14710/gt.v18i3.21933.
- [32] Y. A. R. A. P. M. V. K. P. Kishor Jadhav Nagnath V. Khandekar, “DC Motor Speed Control Using Arduino & L298N Motor Driver,” *Journal of Mechatronics and Automation*, 2024, pp. 1–7.
- [33] A. Y. Raisal *et al.*, “Perancangan Sistem Monitoring Suhu dan Kelembaban Berbasis Arduino UNO pada Kubah Obervatorium Ilmu Falak Universitas Muhammadiyah Sumatera Utara (OIF UMSU),” vol. 9, no. 1, pp. 1–11, 2025.
- [34] T. Rohman and W. Aribowo, “Perancangan Sistem Pengendali Kecepatan Motor DC Menggunakan Kontroler Proposional Integral Derivative pada Palang Pintu Parkir,” *J. Tek. ELEKTRO*, vol. 12, no. 2, pp. 48–54, Jul. 2023, doi: 10.26740/jte.v12n2.p48-54.
- [35] C. Anjali, G. Dropadi, P. Sakshi, K. Gayatri, and V. V Sarwadnya, “Measurement of Angular Displacement Using Potentiometer,” *Int. J. Res. Appl. Sci. Eng. Technol.*, vol. 13, no. IV, 2025.
- [36] M. D. Riski, “Rancang Alat Lampu Otomatis Di Cargo Compartment Pesawat Berbasis Arduino Menggunakan Push Botton Switch Sebagai Pembelajaran Di Politeknik Penerbangan Surabaya (udah),” *Pros. Semin. Nas. Inov. Teknol. Penerbangan*, vol. 3, no. 2, pp. 1–9, 2019, doi: <https://doi.org/10.46491/snntp.v3i2.414>.
- [37] T. Sulistyorini, N. Sofi, and E. Sova, “PEMANFAATAN NODEMCU ESP8266 BERBASIS ANDROID (BLYNK) SEBAGAI ALAT ALAT MEMATIKAN DAN MENGHIDUPKAN LAMPU,” *J. Ilm. Tek.*, vol. 1, no. 3, pp. 40–53, 2022, doi: 10.56127/juit.v1i3.334.
- [38] P. S. Maria and E. Susanti, “Implementasi Algoritma Kalkulasi Interupsi pada Rancang Bangun Tachometer Digital,” *J. Tek. Elektro*, vol. 10, no. 2, pp. 47–53, 2018, doi: 10.15294/jte.v10i2.16350.

- [39] S. A. Sukarno, N. Lilansa, N. W. Nugraha, and A. F. Rifai, “RANCANG BANGUN GRAPHICAL USER INTERFACE PADA SISTEM KENDALI SUHU NIRKABEL PEMANAS TANGKI BERPENGADUK KONTINYU,” *J. Inform. dan Tek. Elektro Terap.*, vol. 12, no. 1, pp. 155–166, 2024, doi: 10.23960/jitet.v12i1.3953.
- [40] J. G. Ziegler and N. B. Nichols, “Optimum Settings for Automatic Controllers,” *J. Dyn. Syst. Meas. Control. Trans. ASME*, vol. 115, no. 2B, pp. 759–765, Jun. 1993, doi: 10.1115/1.2899060.
- [41] M. Effendy, *Pengetahuan Dasar Sistem Kendali*. Surakarta: MUP UMS, 2019. [Online]. Available: https://www.researchgate.net/publication/335854419_Pengetahuan_Dasar_Sistem_Kendali.
- [42] K. H. Ang, G. Chong, and Y. Li, “PID control system analysis, design, and technology,” *IEEE Trans. Control Syst. Technol.*, vol. 13, no. 4, pp. 559–576, 2005, doi: 10.1109/TCST.2005.847331.
- [43] M. santo Gitakarma, *Sistem Kendali : Disertai Contoh Soal dan Penyelesaian*, 1st ed. Yogyakarta: GRAHA ILMU, 2014. [Online]. Available: <https://grahailmu.co.id/previewpdf/978-602-262-367-0-1364.pdf>.
- [44] R. Muhardian and Krismadinata, “Kendali Kecepatan Motor DC Dengan Kontroller PID dan Antarmuka Visual Basic,” *J. Tek. Elektro Dan Vokasional*, vol. 06, no. 01, pp. 328–339, 2020, doi: <https://doi.org/10.24036/jtev.v6i1.108034>.
- [45] O. M. Prabowo, “Pembatasan Definisi Things Dalam Konteks Internet of Things Berdasarkan Keterkaitan Embedded System dan Internet Protocol,” *J. Inf. Technol.*, vol. 1, no. 2, pp. 43–46, Aug. 2019, doi: 10.47292/joint.v1i2.8.
- [46] G. Y. Saputra, A. D. Afrizal, F. K. R. Mahfud, F. A. Pribadi, and F. J. Pamungkas, “Penerapan Protokol MQTT Pada Teknologi Wan (Studi Kasus Sistem Parkir Univeristas Brawijaya),” *Inform. Mulawarman J. Ilm. Ilmu Komput.*, vol. 12, no. 2, pp. 69–75, 2017, doi: 10.30872/jim.v12i2.653.
- [47] M. A. Alqarni, “Priority-enabled MQTT: A Robust Approach to Emergency Event Detection in IoT,” *J. Electr. Appl. Syst.*, vol. 3, 2024.
- [48] Tristuti Aulia, “Belajar Software Ubidots Untuk IoT Enthusiast!,” kmtech.id. Accessed: May 22, 2025. [Online]. Available: <https://www.kmtech.id/post/belajar-software-ubidots-untuk-iot-enthusiast>.

- [49] Ubidots, “Platform Overview.” [Online]. Available: <https://ubidots.com/platform>.
- [50] I. A. Rozaq and N. Y. D. Setyaningsih, “Pengembangan Tempat Sampah Otomatis Berbasis Internet of Things (IoT) untuk Pengelolaan Sampah Logam dan Non-Logam,” *JEECOM J. Electr. Eng. Comput.*, vol. 5, no. 2, pp. 250–257, Oct. 2023, doi: 10.33650/jecom.v5i2.6908.