



**TUGAS AKHIR - EE184801**

## **Implementasi Kontroler Aktuator Dynamixel MX-Series menggunakan Microcontroller ESP32-WROOM-32**

**DAVID BONAR**

**NRP 07111840000220**

Dosen Pembimbing

Ronny Mardiyanto, ST., MT., Ph.D.

NIP : 198101182003121003

Dr. Rudy Dikairono, S.T., M.Sc

NIP : 198103252005011002

**Program Studi Sarjana Teknik Elektro**

Departemen Teknik Elektro

Fakultas Teknologi Elektro dan Informatika Cerdas

Institut Teknologi Sepuluh Nopember

Surabaya

2022





## **TUGAS AKHIR - EE184801**

# **Implementasi Kontroler Aktuator Dynamixel MX-Series menggunakan Microcontroller ESP32-WROOM-32**

**DAVID BONAR**

**NRP 07111840000220**

Dosen Pembimbing

Ronny Mardiyanto, ST., MT., Ph.D.

NIP : 198101182003121003

Dr. Rudy Dikairono, S.T., M.Sc

NIP : 198103252005011002

**Program Studi Sarjana Teknik Elektro**

Departemen Teknik Elektro

Fakultas Teknologi Elektro dan Informatika Cerdas

Institut Teknologi Sepuluh Nopember

Surabaya

2022





**FINAL PROJECT - EE184801**

## **ESP32-WROOM-32 Microcontroller based Dynamixel Controller for MX-Series Actuators**

**David Bonar**

**07111840000220**

**Supervisor**

**Ronny Mardiyanto, ST., MT., Ph.D.**

**NIP : 198101182003121003**

**Dr. Rudy Dikairono, S.T., M.Sc**

**NIP : 198103252005011002**

**Electrical Engineering Undergraduate Study Program**

**Department of Electrical Engineering**

**Faculty of Intelligent Electrical and Informatics Technology**

**Institut Teknologi Sepuluh Nopember**

**Surabaya**

**2022**

(Halaman ini sengaja dikosongkan)

## **LEMBAR PENGESAHAN**

### **IMPLEMENTASI KONTROLER AKTUATOR DYNAMIXEL MX-SERIES MENGGUNAKAN MIKROKONTROLER ESP-WROOM-32**

#### **TUGAS AKHIR**

Diajukan untuk memenuhi salah satu syarat  
memperoleh gelar Sarjana Teknik pada  
Program Studi Sarjana Teknik Elektro  
Departemen Teknik Elektro  
Fakultas Teknologi Elektro dan Informatika Cerdas  
Institut Teknologi Sepuluh Nopember

Oleh : **DAVID BONAR**

NRP. 07111840000220

Disetujui oleh Tim Penguji Tugas Akhir :

1. Ronny Mardiyanto, ST., MT., Ph.D.  Pembimbing

2. Dr. Rudy Dikairono, S.T., M.Sc  Ko-pembimbing

3. Dr.Ir. Totok Mujiono, M.IKom.  Penguji

4. Suwito, ST., MT.  Penguji

5. Ir. Tasripan, MT.  Penguji

**SURABAYA**

**Juli 2022**

*(Halaman ini sengaja dikosongkan)*

## **APPROVAL SHEET**

### **ESP32-WROOM-32 Microcontroller based Dynamixel Controller for MX-Series Actuators**

#### **FINAL PROJECT**

Submitted to fulfill one of the requirements

for obtaining an undergraduate degree at

Undergraduate Study Program of Electronics

Department of Electrical Engineering

Faculty of Intelligent Electrical and Informatics Technology

Institut Teknologi Sepuluh Nopember

By: **DAVID BONAR**

NRP. 07111840000220

Approved by Final Project Examiner Team:

1. Ronny Mardiyanto, ST., MT., Ph.D.  Supervisor
2. Dr. Rudy Dikairono, S.T., M.Sc  Co- Supervisor
3. Dr.Ir. Totok Mujiono, M.IKom  Examiner
4. Suwito, ST., MT.  Examiner
5. Ir. Tasripan, MT.  Examiner

**SURABAYA**

**July 2022**

*(Halaman ini sengaja dikosongkan)*

## **PERNYATAAN ORISINALITAS**

Yang bertanda tangan di bawah ini:

Nama mahasiswa / NRP : David Bonar / 07111840000220  
Program studi : Teknik Elektronika  
Dosen Pembimbing / NIP : Ronny Mardiyanto, ST., MT., Ph.D./  
198101182003121003

dengan ini menyatakan bahwa Tugas Akhir dengan judul "**Implementasi Kontroler Aktuator Dynamixel MX-Series menggunakan Microcontroller ESP32-WROOM-32**" adalah hasil karya sendiri, bersifat orisinal, dan ditulis dengan mengikuti kaidah penulisan ilmiah.

Bilamana di kemudian hari ditemukan ketidaksesuaian dengan pernyataan ini, maka saya bersedia menerima sanksi sesuai dengan ketentuan yang berlaku di Institut Teknologi Sepuluh Nopember.

Surabaya, 20 Juli 2022

Mengetahui  
Dosen Pembimbing



Ronny Mardiyanto, ST., MT., Ph.D.  
NIP. 198101182003121003

Mahasiswa



David Bonar  
NRP. 07111840000220

*(Halaman ini sengaja dikosongkan)*

## **STATEMENT OF ORIGINALITY**

The undersigned below:

Name of student / NRP	:	David Bonar / 07111840000220
Department	:	Electrical Engineering
Advisor / NIP	:	Ronny Mardiyanto, ST., MT., Ph.D. / 198101182003121003

Hereby declare that the Final Project with the title of “ESP32-WROOM-32 Microcontroller based Dynamixel Controller for MX-Series Actuators” is the result of my own work, is original, and is written by following the rules of scientific writing.

If in the future there is a discrepancy with this statement, then I am willing to accept sanctions in accordance with the provisions that apply at Institut Teknologi Sepuluh Nopember.

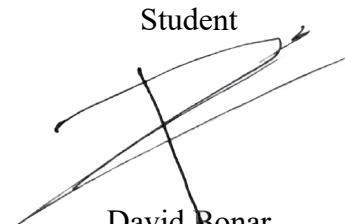
Surabaya, 20 July 2022

Acknowledged  
Supervisor



Ronny Mardiyanto, ST., MT., Ph.D.  
NIP. 198101182003121003

Student



David Bonar  
NRP. 07111840000220

*(Halaman ini sengaja dikosongkan)*

## **ABSTRAK**

### **IMPLEMENTASI KONTROLER AKTUATOR DYNAMIXEL MX-SERIES MENGGUNAKAN MICROCONTROLLER ESP32-WROOM-32**

**Nama Mahasiswa / NRP** : **David Bonar/ 07111840000220**  
**Departemen** : **Teknik Elektro FTEIC – ITS**  
**Dosen Pembimbing** : **Ronny Mardiyanto, ST., MT., Ph.D.**

#### **Abstrak**

Servo adalah salah satu jenis aktuator yang pada saat ini dipakai pada industri, riset, rumah tangga, hiburan, dan edukasi. Berbeda dari motor listrik pada umumnya, kelebihan servo sebagai aktuator ada pada kemampuannya untuk merubah kecepatan, posisi, dan akselerasi dengan presisi yang tinggi dan akurat. Pada tugas akhir ini, servo yang digunakan dalam penelitian adalah servo Dynamixel MX-Series yang diproduksi oleh perusahaan bernama Robotis yang berasal dari Korea Selatan. Untuk dapat mengontrol servo ini, dibutuhkan sebuah modul kontroler yang dipakai untuk berkomunikasi antara komputer pengkontrol dan servo. Umumnya kontroler yang dipakai adalah CM-740 *Sub-controller*. Namun, setelah CM-740 berhenti diproduksi, dibutuhkan kontroler yang dapat menggantikan CM-740 tersebut, maka dari itu Tugas Akhir ini dibuat. Penulis membuat rancangan kontroler baru berbasis mikrokontroler ESP32-WROOM-32 yang memproses data yang dikirimkan dan dibaca dari servo dan juga berkomunikasi dengan komputer, Dynamixel-SDK sebagai dasar *framework* untuk berkomunikasi dengan servo, dan IC 74LS241 sebagai jembatan komunikasi antar mikrokontroler dan servo. Dalam pelaksanaan Tugas Akhir ini, sebuah aplikasi dibuat dengan menggunakan .Net Core 3 sebagai dasar pembangunan sistem aplikasi yang dipakai untuk dapat membaca dan mengontrol data servo menggunakan dua mode yaitu *Single Servo Control* dan *Multiple Servos Control* yang dapat digunakan untuk mengontrol dan membaca data servo baik pada satu buah servo secara detil maupun setiap servo yang terkoneksi dengan kontroler secara bersamaan. Setelah dilakukan pengujian dengan menggunakan aplikasi yang mencakup akurasi data dan juga kecepatan komunikasi, rangkaian kontroler yang dibuat dapat memberikan hasil yang memuaskan dengan akurasi kontroler buatan dalam pembacaan dan pengkontrolan data mencapai nilai 99.96 % dan akurasi kecepatan komunikasi pada 96.48%. Dengan hasil penelitian tersebut, dapat disimpulkan bahwa sistem kontroler yang telah dibangun dapat dipakai untuk mengontrol servo Dynamixel MX-Series yang dapat diandalkan.

**Kata kunci:** *Servo, Kontroler, ESP32, DynamixelSDK, Dynamixel2Arduino, Aplikasi*

*(Halaman ini sengaja dikosongkan)*

## ABSTRACT

### **ESP32-WROOM-32 MICROCONTROLLER BASED DYNAMIXEL CONTROLLER FOR MX-SERIES ACTUATORS**

**Student Name / NRP** : **David Bonar / 07111840000220**  
**Department** : **Teknik Elektro FTEIC - ITS**  
**Advisor** : **Ronny Mardiyanto, ST., MT., Ph.D.**

#### **Abstract**

Servo is one type of actuator that is currently used in industry, research, household, entertainment, and education. In contrast to electric motors in general, the advantage of a servo as an actuator lies in its ability to change speed, position, and acceleration with high precision and accuracy. In this final project, the servo used in this research is the Dynamixel MX-Series servo produced by a company called Robotis from South Korea. To be able to control this servo, a controller module is needed which is used to communicate between the controller computer and the servo. Generally the controller used is the CM-740 Sub-controller. However, after the CM-740 stopped being produced, a controller was needed that could replace the CM-740, which is why this final project was created. The author designed a new controller based on the ESP32-WROOM-32 microcontroller which processes data sent and read from the servo and also communicates with the computer, the Dynamixel-SDK as the basic framework for communicating with the servo, and IC 74LS241 as a communication bridge between the microcontroller and the servo. In this final project, an application is made using .Net Core 3 as the basis for developing an application system that is used to be able to read and control servo data using two modes, namely Single Servo Control and Multiple Servos Control which can be used to control and read servo data both at one servo in detail as well as each servo connected to the controller simultaneously. After testing using an application that includes data accuracy and communication speed, the controller circuit made can give satisfactory results with the artificial controller accuracy in reading and controlling data reaching a value of 99.96% and communication speed accuracy of 96.48%. With these results, it can be concluded that the controller system that has been built can be used to control the Dynamixel MX-Series servo reliably.

**Keywords:** *Servo, Controller, ESP32, Dynamixel, SDK, Dynamixel2Arduino, Application*

*(Halaman ini sengaja dikosongkan)*

## KATA PENGANTAR

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa karena hanya dengan berkat dan penyertaan-Nya, penulis dapat menyelesaikan Tugas Akhir dengan judul “IMPLEMENTASI KONTROLER AKTUATOR DYNAMIXEL MX-SERIES MENGGUNAKAN MIKROKONTROLER ESP-WROOM-32” ini dapat diselesaikan dengan baik.

Adapun Tugas Akhir ini disusun sebagai salah satu persyaratan menyelesaikan studi S1 pada Bidang Studi Teknik Elektronika, Departemen Teknik Elektro, Fakultas Teknologi Elektro dan Informatika Cerdas, Institut Teknologi Sepuluh Nopember

Terima kasih banyak penulis ucapkan kepada pihak-pihak berikut yang telah membantu penulis dalam menyelesaikan Tugas Akhir ini:

1. Tuhan Yang Maha Esa. yang dengan karunia-Nya penulis dapat menyelesaikan Tugas Akhir ini dengan lancar.
2. Keluarga penulis yang selalu memberikan dukungan, semangat, nasihat dan doa kepada penulis.
3. Setiap Dosen dan Tendik Departemen Teknologi Elektro Institut Teknologi Sepuluh Nopember atas bimbingan dan pembelajaran yang diberikan selama masa perkuliahan.
4. Bapak Ronny Mardiyanto, ST., MT., Ph.D. dan Bapak Dr. Rudy Dikairono, S.T., M.Sc selaku dosen pembimbing I dan II yang telah memberikan bimbingan, dan motivasi dalam penggerjaan tugas akhir ini.
5. Mas Fahad, Riris, Riko, Bernard, Zidane dan teman-teman Ichiro yang membantu penulis dalam proses menyelesaikan Tugas Akhir
6. Semua pihak yang tidak dapat penulis sebutkan satu-persatu yang telah membantu dalam penyelesaian penulisan tugas akhir ini.

Penulis menyadari masih banyak kekurangan dalam penyusunan Tugas Akhir ini, maka dari ini penulis mengharapkan saran dan masukan untuk perbaikan yang bisa dilakukan dalam Tugas Akhir ini. Penulis juga berharap agar Tugas Akhir ini dapat memberikan manfaat bagi semua pihak yang membutuhkan dan memberi motivasi kepada pembaca untuk terus kreatif dan berkarya.

Surabaya, 10 Juni 2022



Penulis

*(Halaman ini sengaja dikosongkan)*

## DAFTAR ISI

LEMBAR PENGESAHAN .....	i
APPROVAL SHEET .....	iii
PERNYATAAN ORISINALITAS .....	v
STATEMENT OF ORIGINALITY .....	vii
ABSTRAK.....	ix
ABSTRACT .....	xi
KATA PENGANTAR .....	xiii
DAFTAR ISI.....	xv
DAFTAR GAMBAR.....	xviii
DAFTAR TABEL.....	xxi
BAB 1      PENDAHULUAN .....	1
1.1      Latar Belakang .....	1
1.2      Rumusan Masalah .....	1
1.3      Batasan Masalah.....	2
1.4      Tujuan .....	2
1.5      Manfaat .....	2
BAB 2      TINJAUAN PUSTAKA .....	3
2.1      Hasil Penelitian Terdahulu.....	3
2.1.1 <i>Robust Motion Control of High Precision Mechanical Servo Systems with Parameter Uncertainties</i> .....	3
2.1.2 <i>The Control of Dynamixel RX-28 Based on VC++ for the Locomotion of Cockroach Robot</i> .....	3
2.1.3 <i>High-Frequency Multi Bus Servo and Sensor Communication Using the Dynamixel Protocol</i> .....	4
2.1.4      Penerapan IC 74LS241 Untuk Multi Aktuator Dynamixel AX-12A Pada Biped Robot .....	4
2.1.5 <i>Final Degree Project Biomedical Engineering Degree Da Vinci robot at Hospital Clinic. Manoeuvrability devices and performance in robotic tech.</i> .5	5
2.1.6 <i>Hamburg Bit-Bots and WF Wolves Team Description for RoboCup 2019 – Humanoid TeenSize</i> .....	5
2.2      Dasar Teori.....	6
2.2.1      Servo Dynamixel.....	6
2.2.2      Komunikasi Serial .....	6
2.2.3      Sinyal TTL/RS-485 .....	8

2.2.4	Baterai <i>Lithium Polymer</i> .....	9
2.2.5	ESP32-WROOM-32 .....	10
2.2.6	Dynamixel-SDK.....	12
2.2.7	Dynamixel2Arduino.....	13
2.2.8	PlatformIO.....	14
2.2.9	<i>Windows Forms</i> .....	14
BAB 3	METODOLOGI DAN PEMODELAN SISTEM.....	15
3.1	Pemodelan Sistem Kontroler.....	15
3.1.1	Metode yang Digunakan .....	16
3.1.1.1	Perancangan Konsep Program Kontroler .....	16
3.2	Alat dan Bahan.....	19
3.2.1	ESP32 DevkitC V4 .....	19
3.2.2	Mini 360 DC-DC Buck Converter .....	19
3.2.3	<i>IC 74LS241</i> .....	20
3.2.4	<i>Logic Level Shifter</i> .....	20
3.2.5	USB Logic Analyzer .....	21
3.3	Implementasi Sistem Kontroler.....	22
3.3.1	Rangkaian skematik <i>PCB</i> kontroler .....	22
3.3.2	Perancangan Program Kontroler .....	24
3.3.2.1	Perancangan Program Aplikasi berbasis Windows Forms .....	28
BAB 4	HASIL DAN PEMBAHASAN.....	32
4.1	Hasil Penelitian .....	33
4.1.1	Implementasi dari Pengujian Akurasi Pembacaan dan Pengkontrolan Data Servo .....	33
4.1.1.1	Pembacaan dan Pengkontrolan mode Single Servo Control .....	34
4.1.1.2	Pembacaan dan Pengkontrolan mode Multiple Servos Control .....	35
4.1.2	Pengujian Pembacaan dan Pengkontrolan Data Servo menggunakan CM-740 .....	36
4.1.3	Implementasi Pengujian Durasi Paket Instruksi dengan Paket Status .....	37
4.1.3.1	Pembacaan Return Delay pada Single Servo Control .....	38
4.1.3.2	Pembacaan Return Delay pada Multiple Servos Control .....	38
4.2	Pembahasan.....	39
4.2.1	Perbandingan Akurasi data Kontroler buatan dengan Kontroler Acuan.....	39
BAB 5	Kesimpulan dan Saran .....	41
5.1	Kesimpulan .....	41

5.2 Saran.....	41
DAFTAR PUSTAKA .....	42
LAMPIRAN 1 Kode Program ESP32.....	44
LAMPIRAN 2 Hasil Pengujian Single Servo Control .....	49
LAMPIRAN 3 Hasil Pengujian <i>Multiple Servos Control</i> .....	54
LAMPIRAN 4 Hasil Pengujian Kontroler Acuan CM-740.....	56
LAMPIRAN 5 Hasil Pengujian durasi <i>Return Delay</i> .....	61
LAMPIRAN 6 Kode Program Kontrol Dynamixel.....	63
LAMPIRAN 7 Gambar Pengujian Penelitian.....	77
BIODATA PENULIS .....	78

*(Halaman ini sengaja dikosongkan)*

## DAFTAR GAMBAR

Gambar 2. 1 Servo Dynamixel MX-Series(Robotis, n.d.-a).....	6
Gambar 2. 2 Perbedaan sinyal komunikasi Sinkron dan Asinkron (El-Maleh, n.d.).....	7
Gambar 2. 3 Full-Duplex, Half-Duplex, dan Simplex.....	7
Gambar 2. 4 Perbedaan pin pada TTL dan RS-485(Robotis, n.d.-b).....	8
Gambar 2. 5 Logika sinyal TTL(Robotis, 2020) .....	8
Gambar 2. 6 Logika sinyal RS-485(Wikipedia, n.d.) .....	9
Gambar 2. 7 Baterai Lithium Polymer Tiger 5400 mAh 45C .....	10
Gambar 2. 8 ESP32-WROOM-32 pada <i>board</i> ESP32 DevkitC V4 (Espressif Systems, 2022b).....	11
Gambar 2. 9 Pinout development board ESP32 DevkitC V4(Espressif Systems, 2022b).11	11
Gambar 2. 10 Sistematika Operasi Dynamixel SDK (Robotis, 2022b).....	13
Gambar 3. 1 Diagram Pelaksanaan Tugas Akhir.....	15
Gambar 3. 2 Diagram Blok Sistem kontroler .....	16
Gambar 3. 3 Diagram Konsep Program.....	17
Gambar 3. 4 Implementasi paket instruksi dalam komunikasi .....	18
Gambar 3. 5 Paket status yang dikirimkan dari servo .....	18
Gambar 3. 6 Pinout 74LS241.....	20
Gambar 3. 7 Rangkaian logic level shifter per-Channel.....	21
Gambar 3. 8 Pinout dari <i>logic level shifter</i> .....	21
Gambar 3. 9 <i>USB Logic Analyzer</i> .....	22
Gambar 3. 10 Aplikasi <i>Logic</i> untuk pembacaan Sinyal.....	22
Gambar 3. 11 Hasil <i>PCB</i> yang dibuat.....	23
Gambar 3. 12 Rangkaian skematis <i>PCB</i> kontroler .....	24
Gambar 3. 13 Deklarasi variabel dan pengiriman data hasil pembacaan dari servo .....	25
Gambar 3. 14 Data yang dikirim oleh ESP32 saat <i>Single Servo Control</i> .....	26
Gambar 3. 15 Pengiriman data hasil pembacaan pada mode <i>Multiple Servos Control</i> ....	27
Gambar 3. 16 Pembacaan Posisi untuk setiap ID dan Model servo yang tersambung .....	27
Gambar 3. 17 Kode Pengkontrolan Servo .....	28
Gambar 3. 18 Diagram Pembangunan Aplikasi .....	28
Gambar 3. 19 Tampilan pemilihan <i>port</i> dan <i>baudrate</i> modul kontroler.....	29
Gambar 3. 20 Tampilan mode <i>Single Servo Control</i> .....	30
Gambar 3. 21 Tampilan mode Multiple Servo Control .....	30
Gambar 3. 22 Tampilan Multiple Servo Control setelah posisi digeser ke nilai 2049 .....	31
Gambar 4. 1 Susunan pengujian akurasi pembacaan dan pengkontrolan.....	33
Gambar 4. 2 Pengambilan data <i>Single Servo Control</i> .....	35
Gambar 4. 3 Pengujian <i>Multiple Servos Control</i> .....	36
Gambar 4. 4 Pengujian kontroler CM-740 .....	37
Gambar 4. 5 Rangkaian pengujian <i>return delay</i> dengan <i>logic analyzer</i> .....	37
Gambar 4. 6 Bentuk Sinyal <i>Single Servo Control</i> pada <i>logic analyzer</i> .....	38
Gambar 4. 7 Nilai <i>return delay</i> pada <i>Multiple Servos Control</i> .....	39
Gambar 4. 8 Perbandingan saat posisi 0° antara kontroler buatan dengan CM-740 .....	39
Gambar 4. 9 Perbandingan saat posisi 180° antara kontroler buatan dengan CM-740 ....	40
Gambar 4. 10 Perbandingan saat posisi 360° antara kontroler buatan dengan CM-740 ...	40

*(Halaman ini sengaja dikosongkan)*

## **DAFTAR TABEL**

Tabel 2. 1 Spesifikasi development board ESP32 DevkitC V4.....	11
Tabel 2. 2 Bentuk Paket Instruksi pada protokol 1.0.....	12
Tabel 2. 3 Jenis-jenis instruksi dalam paket protokol 1.0.....	12
Tabel 3. 1 Bentuk Paket Instruksi .....	17
Tabel 3. 2 Bentuk Paket Status .....	18
Tabel 3. 3 Desain Spesifikasi Konverter Buck boost .....	19
Tabel 3. 4 Variabel data servo yang dibaca .....	25
Tabel 4. 1 Variabel pembacaan data servo setiap siklus.....	34
Tabel 4. 2 Hasil Pengujian pembacaan variabel Single Servo Control .....	35
Tabel 4. 3 Hasil Pengujian <i>Multiple Servos Control</i> .....	36
Tabel 4. 4 Hasil Pengujian Akurasi CM-740.....	36
Tabel 4. 5 Hasil Pengukuran durasi return delay Single Servo Control .....	38
Tabel 4. 6 Hasil Pengukuran durasi <i>return delay Multiple Servos Control</i> .....	38
Tabel 4. 7 Perbandingan pembacaan antara kontroler buatan dengan kontroler CM-740..40	
Tabel 4. 8 Perbandingan error pembacaan kontroler buatan dengan kontroler CM-740...40	

(Halaman ini sengaja dikosongkan)

## BAB 1 PENDAHULUAN

### 1.1 Latar Belakang

Dalam masa modern ini, teknologi digunakan untuk menggantikan atau mempermudah pekerjaan manusia dapat ditemukan di berbagai macam bidang baik pada industri, rumah tangga, edukasi, dan hiburan. Saat ini mayoritas industri telah dipenuhi dengan sistem yang telah terotomasi sehingga kinerja manusia yang diperlukan semakin minim. Adanya otomasi sistem ini juga dikarenakan oleh berbagai macam keuntungan seperti akurasi, daya tahan dan juga konsistensi.

Penggunaan sistem yang memerlukan adanya pergerakan secara akurat pada umumnya menggunakan aktuator servo yang bergerak dengan penggunaan motor BLDC (Brushless DC). Servo memiliki fungsi yang luas dan dapat ditemukan dalam berbagai aplikasi seperti robot tangan yang umumnya dipakai dalam industri untuk memindahkan barang secara akurat dari satu tempat ke tempat lainnya, printer 3 Dimensi dimana printer tersebut membutuhkan pergerakan mata yang akurat sehingga dapat memberikan hasil sesuai dengan yang diinginkan, dan juga dipakai dalam robot yaitu sebagai alat yang digunakan untuk menggerakkan sebuah sistem. Pada Tugas Akhir ini, servo yang digunakan ialah servo yang dibuat oleh perusahaan Robotis yaitu servo Dynamixel MX-Series yaitu MX-28, MX-64, dan MX-106 dimana servo-servo ini merupakan jenis servo yang digunakan oleh mayoritas tim robotika pada Institut Teknologi Sepuluh Nopember.

Untuk dapat menggerakkan servo Dynamixel, dibutuhkan sebuah kontroler yang berguna untuk menyalurkan perintah dari komputer utama kepada setiap servo yang tersambung pada kontroler tersebut. Kontroler ini diperlukan karena jenis komunikasinya yang berbeda dengan komunikasi dari komputer sehingga dibutuhkan penerjemah antara servo dengan komputer sehingga pergerakan masing-masing servo yang tersambung pada kontroler dapat diatur sesuai dengan nilai yang diberikan untuk masing-masing servo tersebut. Untuk servo Dynamixel sendiri, perusahaan Robotis sendiri mengeluarkan berbagai jenis kontroler untuk berbagai macam aplikasi yang dibutuhkan oleh penggunanya. Pada saat ini, kontroler Dynamixel yang digunakan sebagai acuan dalam penelitian adalah CM-740 dikarenakan kontroler ini merupakan jenis kontroler yang digunakan oleh mayoritas tim robotika Institut Teknologi Sepuluh Nopember. CM-740 ini memiliki beberapa fitur yaitu menggunakan mikrokontroller STM32 sebagai otak yang mengatur jalannya proses kontroler, sensor akselerometer, giroskop dan tegangan, lalu memiliki kemampuan untuk komunikasi Serial dengan komputer dan juga komunikasi TTL untuk dapat berkomunikasi dengan servo. CM-740 ini merupakan salah satu komponen terpenting yang ada di dalam robot humanoid, namun sejak tahun 2019 kontroler ini sudah tidak diproduksi oleh perusahaan Robotis, maka dibutuhkan adanya kontroler baru yang dapat menggantikan fungsi CM-740 saat ini dikarenakan jumlahnya yang semakin terbatas. Oleh karena itu diajukan penelitian tugas akhir ini mengenai pembuatan kontroler baru yang dapat diimplementasikan pada servo Dynamixel MX-Series agar dapat mengontrol servo tersebut dengan sistematika pembangunan program dan rangkaian elektronika yang dibuat beserta dengan mempelajari lebih dalam bagaimana cara kerja servo tersebut beserta sistem komunikasinya.

### 1.2 Rumusan Masalah

Permasalahan yang dibahas dalam tugas akhir ini adalah :

1. Perancangan komponen yang dibutuhkan dalam papan sirkuit cetak agar mikrokontroler ESP32-Wroom-32 dapat berkomunikasi dengan servo dan aplikasi secara bersamaan.
2. Implementasi DynamixelSDK dan Dynamixel2Arduino pada mikrokontroler agar dapat mengirimkan data servo MX-Series
3. Implementasi software yang digunakan pada sistem kontroler Dynamixel MX-Series tidak mengganggu pembacaan servo jika menggunakan kontroler lainnya.

### **1.3 Batasan Masalah**

Untuk memfokuskan permasalahan yang akan dibahas maka dibutuhkan pembatasan masalah. Batasan-batasan tersebut diantaranya adalah :

1. Permasalahan servo yang digunakan ialah servo Dynamixel MX28T, Dynamixel MX64T, dan Dynamixel MX106T (MX Series).
2. Permasalahan dasar framework dari servo MX-Series yang akan diteliti dalam tugas akhir ini adalah penggunaan library Dynamixel2Arduino berbasis DynamixelSDK yang dikeluarkan oleh perusahaan Robotis untuk servo Dynamixel MX-Series dan protokol yang dipakai ialah protokol 1.0.
3. Kontroller dapat menggerakkan servo Dynamixel MX-Series sesuai dengan perintah yang diberikan pada aplikasi yang dibuat berbasis Windows Form dengan menggunakan bahasa C#.

### **1.4 Tujuan**

Adapun tujuan dari tugas akhir ini adalah sebagai berikut:

1. Membuat kontroler yang dapat digunakan untuk mengontrol servo Dynamixel MX-Series dengan sistem plug-and-play tanpa mempengaruhi memori EEPROM (electrically erasable program read-only memory) dan memori flash yang ada pada servo
2. Dapat mengontrol servo Dynamixel MX-Series menggunakan aplikasi tersendiri dengan interface yang mudah dipahami sehingga proses pengaturan servo dapat dipermudah.
3. Mengerti sistem kontrol servo Dynamixel MX-Series baik dari cara komunikasi, sistem kontrol, dan firmware yang dipakai sehingga dapat diterapkan pada riset yang menggunakan servo Dynamixel MX-Series kedepannya.

### **1.5 Manfaat**

Dari tugas akhir ini, hasil yang diharapkan dapat memberi manfaat bagi seseorang yang ingin menggunakan servo Dynamixel MX-Series agar dapat mengontrol servo tersebut dengan kontroler sendiri dan ter-inspirasi untuk mempelajari lebih dalam sistem komunikasi protokol pada servo dan pembuatan aplikasi yang dibutuhkan untuk mengontrolnya. Harapan dari penulis juga dengan adanya penelitian ini maka pembaca dapat mengerti cara kerja servo Dynamixel dan juga dapat melanjutkan penelitian ini untuk dapat mengembangkan servo buatan sendiri.

## BAB 2 TINJAUAN PUSTAKA

### 2.1 Hasil Penelitian Terdahulu

#### 2.1.1 *Robust Motion Control of High Precision Mechanical Servo Systems with Parameter Uncertainties*

Pada penelitian ini membahas mengenai implementasi sistem mekanikal servo secara umum harus dapat beradaptasi pada setiap tugas yang berbeda dengan inersia dan beban yang berbeda-beda sehingga menghasilkan jumlah parameter inersia beragam. Keberagaman parameter inersia yang dihasilkan namun tidak dapat diolah dengan sempurna dengan metode pengkontrolan sekarang yang menggunakan kontrol PD (*Proportional Derivative*) dan sistem desain kontrol berbasis pengamatan disturbansi dimana penggunaan metode tersebut menghasilkan pergerakan yang tidak stabil pada sistem mekanikal pada servo dan penurunan pada performa sistem dalam melakukan pembacaan posisi servo. Untuk mengatasi masalah tersebut, metode kontrol non-linear yang terbagi menjadi desain kontrol PD sebagai referensi model pada sistem dan kontrol mode geser digunakan untuk mendapatkan pergerakan yang stabil, dan respon transien yang terjamin.(Qiang Liu et al., 2006)

#### 2.1.2 *The Control of Dynamixel RX-28 Based on VC++ for the Locomotion of Cockroach Robot*

Dalam jurnal ini ditulis oleh Xin Kang,Wenda Shen, Weihai Chen, Jianhua Wang membahas mengenai cara mengontrol servo Dynamixel menggunakan komunikasi Serial asinkron RS-485 menggunakan mikrokontroller AVR 8-bit yaitu ATMega128. Dalam sistem kontrolnya, dengan menggunakan RS-485 UART maka setiap servo RX-28 dapat dikontrol dengan jumlah mencapai 254 buah menggunakan topologi bus dan setiap servo sendiri dapat dikontrol dengan perintah yang berbeda karena masing-masing memiliki ID yang dapat mengidentifikasi servo tertentu. Setiap servo RX-28 ini memiliki RAM dan EEPROM, karena EEPROM merupakan komponen memori yang non-volatile maka walaupun servo tidak terhubung dengan arus listrik, tetapi EEPROM tetap menyimpan data seperti Baud Rate, ID, dan limit torsi. Data seperti posisi dan kecepatan disimpan dalam RAM sehingga data-data tersebut hilang saat servo tidak tersambung dengan arus listrik. ATMega128 dipakai dalam pengkontrolannya karena Main Controller harus dapat mendukung komunikasi menggunakan UART RS-485. Pada percobaan yang penulis lakukan, satuan USART yang dipakai adalah Mega128 dan USART ini mengandung dua pemancar dan penerima untuk komunikasi full-duplex sinkron dan asinkron yaitu USART0 dan USART1. Fitur dari kontroler ini adalah operasi sinkron dan asinkron yang full-duplex, resolusi dari Baud Rate yang bernilai tinggi, frame paket Serial dengan 5,6,7,8 atau 9 data bit dan 1 atau 2 stop bit, deteksi *Error*, filter noise dan masih banyak lagi. Karena komunikasi pada servo RX-28 menggunakan RS-485, namun level komunikasi pada ATMega128 adalah TTL maka dari itu dibutuhkan modul MAX485 sebagai konverter sinyal TTL menjadi RS-485. Sedangkan dalam program mikrokontroler ini diisi oleh inisialisasi hardware, fungsi untuk kirim dan penerimaan paket, dan empat interrupt UART. Program ini di-loop sehingga saat komputer mengirimkan paket instruksi, SIGNAL (SIG\_UART1\_RECV) dan interrupt RX dari UART1 merespon dengan mengeluarkan fungsi kirim yang ada dari program mikrokontroller lalu menyalurkan paket instruksi tersebut kepada servo RX-28. Kemudian SIGNAL (SIG\_UART0\_TRANS) dan interrupt TX dari UART0 merespon sehingga arah dari RS-485 berubah menjadi RX ketika paket instruksi dari komputer telah selesai dikirim. Setelah itu kontroler menerima paket status yang dikirim dari servo RX-28,

lalu SIGNAL (SIG\_UART0\_RECV) dan interrupt RX dari UART0 merespon dengan memanggil fungsi terima (menerima paket status dari RX-28) lalu mengirimkan paket status tersebut kepada komputer sehingga arah dari RS-485 berubah menjadi TX dan proses pengiriman paket status ke komputer selesai.(Xin Kang et al., 2009)

### **2.1.3 *High-Frequency Multi Bus Servo and Sensor Communication Using the Dynamixel Protocol***

Jurnal yang ditulis oleh Marc Bestmann, Jasper Guldenstein, and Jianwei Zhang ini berisi tentang frekuensi kontrol servo Dynamixel menggunakan pendekatan multi-bus dengan tujuan mempercepat komunikasi dan reaksi dari robot. Komunikasi Dynamixel dengan topologi bus secara spesifik menggunakan protokol master/slave dengan 8 data bit, 1 stop bit dan tanpa menggunakan parity. Sama seperti penjelasan pada jurnal “The Control of Dynamixel RX-28 Based on VC++ for the Locomotion of Cockroach Robot”, dalam mengontrol servo digunakan dua buah paket yaitu paket instruksi yang dikirim dari master dan paket status yang dikirimkan oleh slave (servo). Paket instruksi ini sendiri dapat dibuat single, sync, atau bulk dimana instruksi single hanya diproses oleh satu slave, instruksi sync untuk satu set slave dan juga register berurut dalam range tertentu yang dapat di read atau write, dan paket bulk merupakan bentuk paket yang mirip dengan paket sync tetapi register dapat dispesifikasi secara individual untuk slave yang menerimanya. Semua paket yang ada terdiri dari header, ID servo, dan panjang paket tersebut dengan diakhiri checksum pada akhir paket untuk mengecek ulang apakah paket sudah benar. Performa dalam sebuah sistem dapat dipengaruhi oleh durasi siklus read dan write pada sistem tersebut. Durasi siklus ini dipengaruhi oleh beberapa faktor yaitu baud rate, jumlah servo dalam satu bus, dan jumlah byte yang di-write atau read, dari perhitungan yang dilakukan penulis jurnal ini menyimpulkan bahwa penggunaan pengiriman paket secara bulk atau sync menggunakan jumlah byte yang lebih sedikit sehingga performa sistem lebih baik.(Bestmann, Güldenstein, et al., n.d.)

### **2.1.4 *Penerapan IC 74LS241 Untuk Multi Aktuator Dynamixel AX-12A Pada Biped Robot***

Jurnal yang ditulis oleh Achmad Firman Choiri dan Beni Widiawan ini membahas mengenai penggunaan IC(Integrated Circuit) 74LS241 dalam pengkontrolan servo AX-12A. Tujuan dari percobaan ini adalah untuk menggerakkan servo AX-12A dengan menggunakan IC74LS241 sebagai driver untuk menjembatani koneksi TTL antara servo dengan mikrokontroler yang digunakan. Pada jurnal ini, percobaan dilakukan dengan menyambungkan 10 (sepuluh) buah servo AX-12A kepada driver IC 74LS241 ini lalu data TTL dikontrol oleh Arduino Uno. Dari percobaan yang dilakukan, servo dapat bergerak dan saat 10 servo yang digerakkan, arus yang dipakai adalah sebesar 1.01 mA dimana nilai tersebut masih ada dalam batasan aman sehingga dapat disimpulkan servo masih aman untuk dikontrol. Lalu percobaan selanjutnya dilakukan dengan mencoba pergerakan pada biped robot (robot berkaki dua) dimana dengan program yang dijalankan didapatkan hasil berupa kesimpulan yaitu penggunaan IC 74LS241 penting dalam pembuatan sistem robot menggunakan servo yang berkomunikasi dengan Serial tipe TTL, karena dengan adanya IC ini sebagai Line Driver, maka komunikasi TTL dapat dilakukan dengan membedakan sinyal transmitter dan sinyal receiver.(Firman Choiri et al., n.d.)

### **2.1.5 Final Degree Project Biomedical Engineering Degree Da Vinci robot at Hospital Clinic. Manoeuvrability devices and performance in robotic tech**

Jurnal ini ditulis oleh Júlia Meca Santamaría berisi mengenai sistem robotik yang bekerja sebagai alat bantu dalam melakukan operasi di ruang medis. Robot medis ini pada umumnya dipakai untuk membantu dokter agar dapat menggapai bagian tubuh yang sulit diraih jika menggunakan tangan manusia dan juga dikarenakan adanya motion scaling, pergerakan dari robot bantu medis dapat dilakukan dengan cepat, akurat dan stabil sehingga aplikasinya sangat membantu dalam proses operasi. Pada saat jurnal ini ditulis, sistem robot untuk operasi yang bernama "Da Vinci System" [6]. Sistem ini merupakan teknologi terbaru yang terdiri dari tiga elemen utama yaitu konsol dokter, meja dokter, dan sistem vision. Sistem ini fokus terhadap mentranslasi pergerakan dari konsol kepada efektor dengan presisi maksimum dan tanpa ada waktu jeda. Dalam jurnal ini, dilakukan percobaan untuk membuat sebuah divais yang dapat melakukan setiap kemampuan robot sistem Da Vinci, namun menggunakan alat yang lebih murah dan mudah ditemukan dan dapat diimplementasikan pada robot UR5. Pada percobaannya, dibuat alat menggunakan STM32 sebagai mikrokontroler yang digunakan lalu menggunakan servo AX-12A dan servo Longrunner ky66. Servo pada percobaan ini digunakan sebagai prototipe pergerakan jarum yang ada pada ujung robot UR5 nantinya. Kontrol yang ada dilakukan dengan menggunakan translasi nilai dari konsol yang dipegang ke nilai-nilai seperti roll, pitch, dan yaw. Dengan adanya perubahan dari nilai data dari konsol tersebut, maka Arduino menerima nilainya dan mentranslasikan nilai tersebut kepada pergerakan yang diinginkan pada servo robot UR5.(Meca Santamaría, 2021)

### **2.1.6 Hamburg Bit-Bots and WF Wolves Team Description for RoboCup 2019 – Humanoid TeenSize**

Artikel ini dibuat oleh Marc Bestmann, Hendrik Brandt, Timon Engelke, Niklas Fiedler , Alexander Gabel , Jasper G'uldenstein, Jonas Hagge , Judith Hartfill, Tom Lorenz, Tanja Heuer, Martin Poppinga, Ivan David Riano Salamanca, dan Daniel Speck. Artikel ini membahas deskripsi tim robot humanoid yaitu Hamburg Bit-Bots dan WF-Wolves. Kedua tim ini adalah tim yang ikut serta dalam lomba robot sepak bola humanoid internasional yang bernama RoboCup. Dalam artikel ini dibahas setiap komponen yang ada di dalam robot tim mereka dan bagaimana cara kerjanya. Pada bagian hardware, kedua tim menggunakan platform yang sama yang disebut dengan Wolfgang, lalu robot mereka memiliki total sebanyak 20 buah servo yang terdiri dari Dynamixel MX-64 dan MX-106. Ada tiga buah komputer yang terpasang dala satu badan robot yaitu Intel NUC, Nvidia Jetson TX2 dan Odroid XU-4. Adanya ketiga komputer ini memberikan tenaga yang cukup untuk dapat menggunakan FCNN (Fourier Convolutional Neural Network)[8] pada sistem vision dan melakukan perhitungan invers kinematik. Pada robot mereka dipakai protokol terbaru dari Dynamixel yaitu protocol 2.0 dimana dalam percobaan mereka membuat hasil komunikasi yang lebih cepat, stabil, dan memberikan estimasi seberapa besar torsi yang dikeluarkan dan juga dapat melakukan pergerakan servo mengontrol torsi yang dihasilkan. Mereka-pun menggunakan kontroler yang dibuat oleh tim Rhoban yang berasal dari Prancis bernama DXLBoard(Rhoban, 2018) dimana kontroler ini dapat memberikan kontrol yang lebih cepat dan error lebih sedikit dibandingkan menggunakan kontroler awal yaitu CM-730.(Bestmann, Brandt, et al., n.d.)

Hasil penelitian terdahulu merupakan upaya peneliti untuk menemukan perbandingan dan selanjutnya untuk menemukan inspirasi baru untuk peneltiain selanjutnya. Selain itu penelitian terdahulu membantu penelitian dapat memposisikan penelitian serta menujukkan orisinalitas dari

penelitian. Maka dalam tinjauan pustaka ini peneliti mencantumkan hasil-hasil penelitian terdahulu.

## 2.2 Dasar Teori

### 2.2.1 Servo Dynamixel

Dynamixel adalah jenis servo yang diproduksi oleh perusahaan Robotis, perusahaan yang berdiri di Korea Selatan yang berfokus pada pembuatan robot kit dan perangkat-perangkat pendukungnya. Servo ini dapat disebut sebagai smart actuator atau aktuator cerdas dimana dalamnya sudah ter-integrasi dengan mikrokontroler sendiri yang memberikan kemampuan untuk mengontrol posisi, kecepatan, dan torsi yang dikeluarkan oleh motor secara akurat, servo ini juga mampu untuk memberikan informasi seperti ID dan posisi secara realtime dengan kecepatan komunikasi yang tinggi. Pada dasarnya servo Dynamixel ini terbagi menjadi beberapa jenis yaitu servo MX, AX, X Series, dan PRO Series. Perbedaan dari tiap jenis servo tersebut ada pada kecepatan, besar torsi, bentuk badan, dan juga jenis komunikasinya. Setiap servo Dynamixel yang diproduksi oleh Robotis terbagi menjadi 2 jenis komunikasi yaitu TTL dan juga RS-485, kedua jenis komunikasi tersebut menggunakan komunikasi half-duplex yaitu dapat melakukan komunikasi secara bergantian dari satu titik ke titik lainnya. Perbedaan dari kedua jenis komunikasi ini ada pada jumlah pin yang dipakai dimana servo yang menggunakan komunikasi RS-485 memiliki 4 buah pin konektor sedangkan servo yang menggunakan komunikasi TTL memiliki 3 buah pin, perbedaan lainnya yaitu sinyal dalam komunikasi RS-485 lebih kuat dan akurat dibandingkan TTL. Walaupun jenis komunikasi servo yang berbeda, namun sebuah robot tetap dapat menggunakan kedua jenis servo yang menggunakan komunikasi TTL dan RS-485 secara bersamaan hanya dengan menambahkan modul konverter TTL to RS-485 maupun sebaliknya. (ROBOTIS, 2022a)

Jenis servo MX-Series menggunakan jenis komunikasi TTL dan mempunyai kemampuan untuk kontrol posisi akurat, pergerakan 360 derajat, koreksi PID, dan juga kecepatan komunikasi yang mencapai 3 Mbps. Gambar 2.1 menunjukkan Servo MX-28, MX-64, dan juga MX-106 dimana semakin tinggi angkanya maka kecepatan dan besar torsi yang dimiliki oleh motornya semakin tinggi. (Robotis, 2022a)

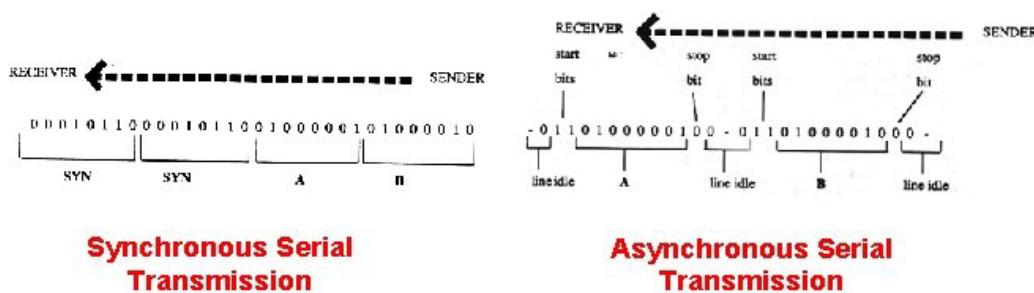


Gambar 2. 1 Servo Dynamixel MX-Series(Robotis, n.d.-a)

### 2.2.2 Komunikasi Serial

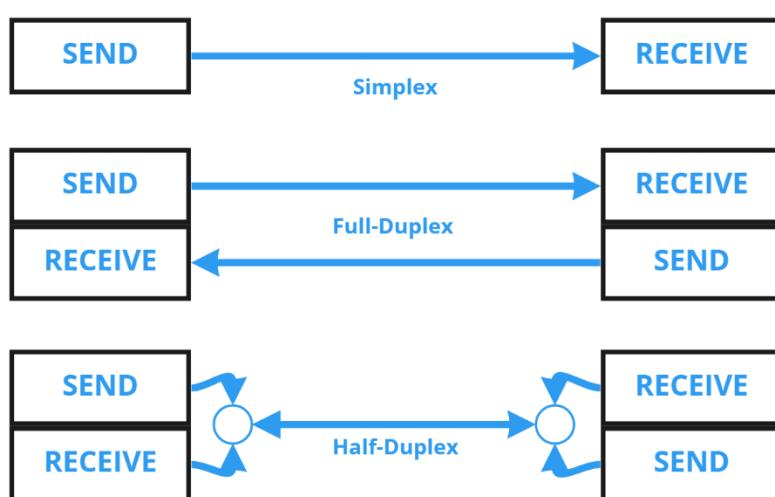
Komunikasi Serial merupakan proses pengiriman data dari satu titik ke titik lainnya dalam bentuk bit secara berurutan melalui sebuah kanal komunikasi atau bus. Komunikasi Serial ini pada umumnya dipakai untuk komunikasi jarak jauh dan juga pada jaringan komputer dikarenakan biaya penggunaan kabel dan sinkronisasi pada sistem pada Serial lebih sedikit jika dibandingkan dengan komunikasi paralel. Komunikasi Serial ini memiliki 2 macam yaitu sinkron dan sinkron. Komunikasi Serial sinkron atau biasa disebut Synchronous Serial adalah jenis

komunikasi dimana data dari titik A ke titik B dan sebaliknya dikirim menggunakan clock dari salah satu komponen pada salah satu titik atau clock buatan sehingga data dikirimkan beserta dengan sinyal clock. Sisi positif dari komunikasi synchronous ini ada pada efisiensi data yang ditransmisikan tetapi sisi negatifnya ada pada prosedur transmisi yang lebih sulit. Pada komunikasi Serial Asinkron atau disebut Asynchronous Serial, data yang dikirim dari titik A ke titik B atau sebaliknya dikirim menggunakan clock yang dimiliki oleh masing-masing titik, sehingga dibutuhkan frekuensi komunikasi yang sama antara keduanya dan pada awal komunikasi, kedua titik harus mengetahui berapa banyak bit yang dikirim dalam satu detik agar komunikasi dapat berjalan dengan lancar. Gambar 2.2 menunjukkan perbedaan data yang dikirim melalui komunikasi sinkron dan asinkron.



Gambar 2. 2 Perbedaan sinyal komunikasi Sinkron dan Asinkron (El-Maleh, n.d.)

Pada komunikasi Serial ini, ada juga beberapa jenis komunikasi di dalamnya yaitu komunikasi full-duplex, komunikasi half-duplex, dan komunikasi simplex. Pada komunikasi full-duplex, data dapat dikirimkan dari titik A ke titik B dan sebaliknya secara bersamaan dikarenakan kedua titik tersebut memiliki dua jalur transmisi dimana satu untuk mengirim data dan satu lagi untuk menerima data. Sedangkan pada komunikasi half-duplex, data hanya dapat dikirimkan dari titik A ke titik B dan sebaliknya secara bergantian karena hanya memiliki satu jalur transmisi. Pada komunikasi simplex, data dikirim dari titik A ke titik B namun tidak bisa sebaliknya sehingga hanya merupakan komunikasi satu arah, contohnya pada radio dan televisi. Gambar 2.3 adalah gambaran bagaimana komunikasi full-duplex, half-duplex, dan simplex dilakukan.



Gambar 2. 3 Full-Duplex, Half-Duplex, dan Simplex

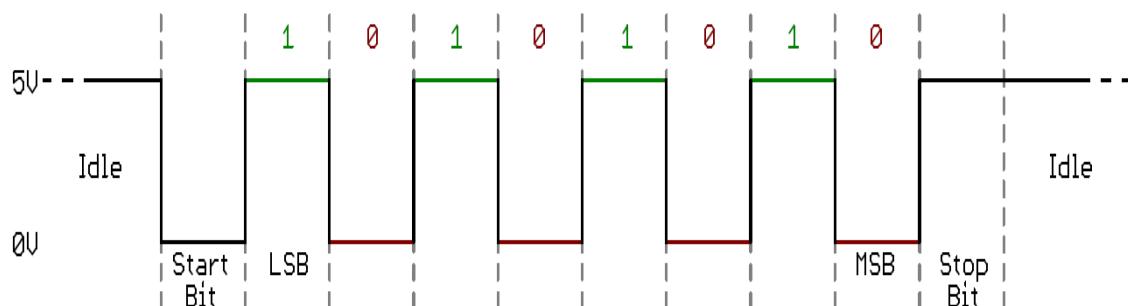
### 2.2.3 Sinyal TTL/RS-485

Servo Dynamixel merupakan servo yang beroperasi menggunakan komunikasi Serial yang bersifat asinkron (Half Duplex). Komunikasi asinkron ini adalah jenis komunikasi dimana dua komponen dapat berkomunikasi dengan satu sama lain tetapi data yang dikirim tidak dapat dikirim secara bersamaan, sehingga data yang dikirim dari satu titik ke titik lainnya dikirim secara bergantian sesuai dengan urutan yang telah ditentukan. Perbedaan antara tipe sinyal TTL (Transistor/Transistor Logic) dan RS-485 (Recommended Standard 485) pada servo Dynamixel yang paling signifikan dapat dilihat dari jumlah pin yang dipakai dimana TTL memiliki 3 buah pin sedangkan RS-485 memiliki 4. Dapat dilihat dari Gambar 2.4 dimana pada TTL ada 3 buah pin yaitu pin 1 sebagai GND (Ground), pin 2 sebagai VCC atau tegangan supply, pin 3 sebagai data TTL, sedangkan pada RS-485 pin 1 dan 2 memiliki fungsi yang sama seperti TTL tetapi pada pin 3 dan 4, pin data dibagi menjadi dua yaitu Data ‘+’ dan juga data ‘-’.

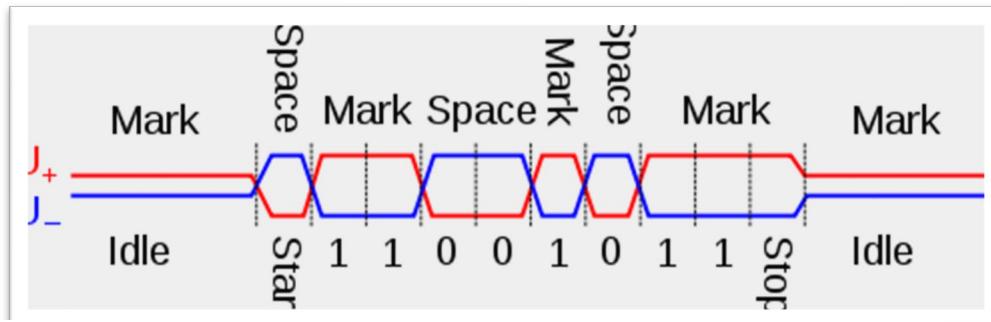
4 Pin Cable			3 Pin Cable		
Pin No.	Signal	Pin Figure	Pin No.	Signal	Pin Figure
1	GND		1	GND	
2	NOT Connected		2	NOT Connected	
3	DATA + [RS-485]		3	DATA [TTL]	
4	DATA - [RS-485]				

Gambar 2. 4 Perbedaan pin pada TTL dan RS-485(Robotis, n.d.-b)

Perbedaan lainnya pada kedua komunikasi ini ada pada logika ‘1’ (HIGH) dan logika ‘0’ (LOW). Pada TTL, logika ‘0’ didapat saat tegangan pada pin data ada pada tegangan 0 Volt sedangkan logika ‘1’ didapat saat tegangan pada pin data ada pada tegangan 5 Volt. Sedangkan pada RS-485, logika ‘1’ atau biasa disebut “Mark” ada pada saat sinyal dari pin data ‘+’ bernilai LOW dan pin data ‘-’ bernilai HIGH, dan logika ‘0’ atau yang biasa disebut “Space” didapat pada saat sinyal dari pin data ‘+’ bernilai HIGH dan pin data ‘-’ bernilai LOW. Sehingga dapat disimpulkan pada RS-485, logika ‘1’ direpresentasikan dengan perbedaan tegangan dari -(2-6) Volt sedangkan logika ‘0’ direpresentasikan dengan perbedaan tegangan dari +(2-6) Volt. Gambar 2.5 dan 2.6 menunjukkan bentuk logika sinyal High dan Low saat operasi pada komunikasi TTL dan komunikasi RS-485.



Gambar 2. 5 Logika sinyal TTL(Robotis, 2020)



Gambar 2. 6 Logika sinyal RS-485(Wikipedia, n.d.)

Oleh karena RS-485 menggunakan differensial sinyal sebagai tolak ukur logika HIGH atau LOW, maka jarak yang dapat ditempuh menggunakan protokol RS-485 jauh lebih tinggi mencapai ratusan atau ribuan meter dibandingkan dengan sinyal dari protokol TTL yang hanya memiliki jarak tempuh kurang dari 30 cm. Differensial sinyal pada RS-485 inipun juga membuat sinyal yang dikirim lebih stabil dan tahan gangguan dibandingkan dengan sinyal dari protokol TTL.

#### 2.2.4 Baterai *Lithium Polymer*

Baterai Lithium Polimer atau biasa disebut dengan LiPo merupakan salah satu jenis baterai yang sering digunakan dalam dunia RC. Utamanya untuk RC tipe pesawat dan helikopter. Baterai LiPo tidak menggunakan cairan sebagai elektrolit melainkan menggunakan elektrolit polimer kering yang berbentuk seperti lapisan plastik film tipis. Lapisan film ini disusun berlapis-lapis diantara anoda dan katoda yang mengakibatkan pertukaran ion. Dengan metode ini baterai LiPo dapat dibuat dalam berbagai bentuk dan ukuran. Gambar 2.7 menunjukkan baterai lipo yang dijual di pasaran dengan merk Tiger yang berkapasitas 5400 mAH 3S dengan discharge rate sebesar 45C. Discharge rate ini sendiri adalah besaran dimana sebuah baterai dapat discharge dalam waktu tertentu terhadap kapasitas maksimum yang dimiliki oleh baterai tersebut.

Adapun tiga kelebihan utama yang ditawarkan oleh baterai berjenis LiPo daripada baterai jenis lain seperti NiCad atau NiMH yaitu:

1. Baterai LiPo memiliki bobot yang ringan dan tersedia dalam berbagai macam bentuk dan ukuran.
2. Baterai LiPo memiliki kapasitas penyimpanan energi listrik yang besar.
3. Baterai LiPo memiliki tingkat discharge rate energi yang tinggi, dimana hal ini sangat berguna sekali dalam bidang RC atau motor DC lainnya.

Selain memiliki keuntungan, baterai ini juga memiliki beberapa kelemahan seperti:

1. Harga baterai LiPo masih tergolong mahal jika dibandingkan dengan baterai jenis NiCad dan NiMH
2. Performa yang tinggi dari baterai LiPo harus dibayar dengan umur yang lebih pendek. Usia baterai LiPo sekitar 300-400 kali siklus 31 pengisian ulang. Sesuai dengan perlakuan yang diberikan pada baterai.
3. Alasan keamanan. Baterai LiPo menggunakan bahan elektrolit yang mudah terbakar.
4. Baterai LiPo membutuhkan penanganan khusus agar dapat bertahan lama. Charging, Discharging, maupun penyimpanan dapat mempengaruhi usia dari baterai jenis ini.

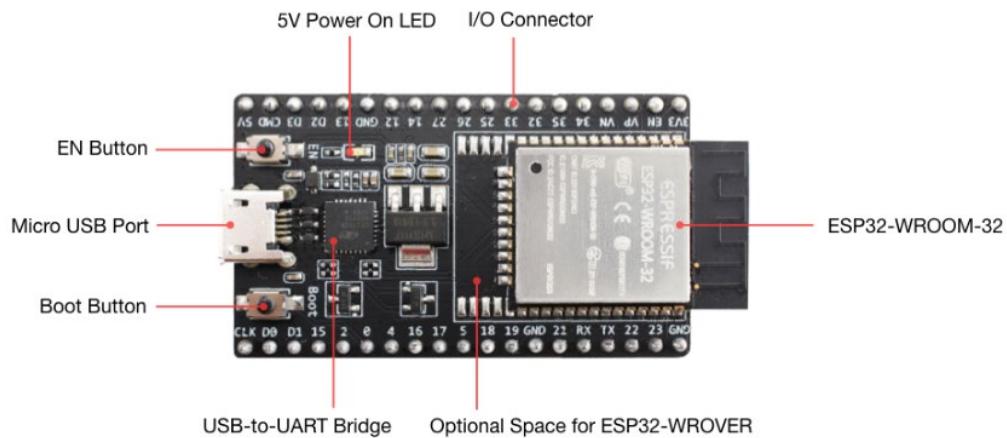


Gambar 2. 7 Baterai Lithium Polymer Tiger 5400 mAh 45C

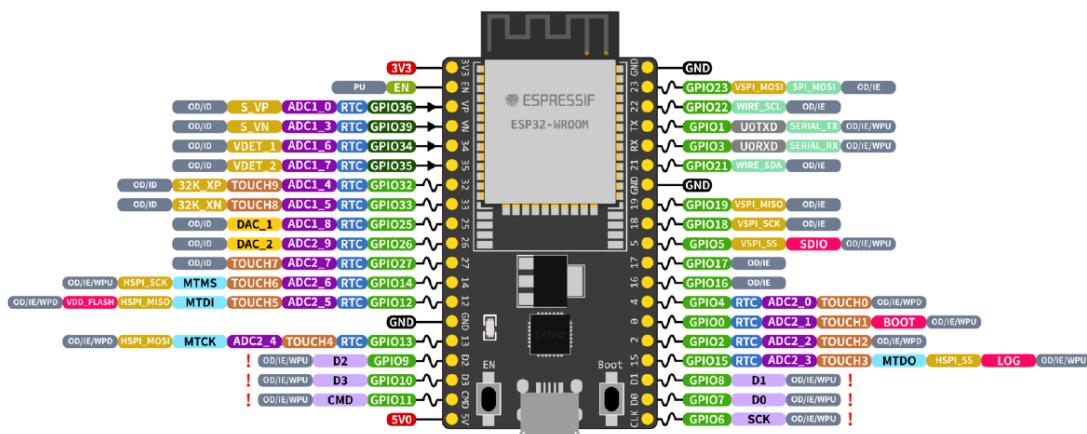
### 2.2.5 ESP32-WROOM-32

ESP32 adalah sebuah seri mikrokontroler yang terintegrasi dengan Wi-Fi dan Bluetooth yang murah dan menggunakan energi yang kecil. Seri ESP32 ini menggunakan Tensilica Xtensa LX6 sebagai mikroprosesor yang dipakai pada variasinya untuk dual-core dan single-core. Pada tugas akhir ini, seri ESP32 yang dipakai adalah ESP32-WROOM-32 yang telah terpasang pada *development board* ESP32-DevkitC. Seri ini memiliki modul Wi-Fi dan Bluetooth dengan kemampuan BLE (Bluetooth Low Energy). Mikrokontroler ini dapat beroperasi dengan tegangan yang berkisar dari 3.0-3.6 Volt, namun dengan voltage regulator yang ada pada *development board* ESP32-DevkitC V4 maka dapat menggunakan daya sebesar 5V yang nantinya dengan regulator akan dikonversikan menjadi 3.3V dan mikrokontroler dapat diprogram langsung melalui port micro-USB yang ada pada *development board* ini. Gambar 2.8 adalah *development board* ESP32 DevkitC V4 dan gambar 2.9 adalah pinout dari *board* ini beserta keterangan setiap pinnya. (Espresifff Systems, 2022b)

Spesifikasi dari *development board* ESP32 DevkitC V4 dapat dilihat dari tabel 2.1. Dapat dilihat dari tabel bahwa tegangan operasi yang dimiliki oleh ESP32-WROOM-32 ini ada pada kisaran 3.0~3.6 Volt, namun pada delevopment board telah diberikan regulator AMS1117 yang merupakan *low-dropout voltage regulator* (LDO) dimana nilai tegangan yang keluar dapat konstan dengan nilai 3.3 Volt dan development board dapat bekerja meskipun dengan tegangan masuk 5V, sehingga ada tiga cara untuk menyediakan daya pada board yaitu : Melalui port micro USB, pin 5V/GND, dan pin 3V3/GND. Ketiga opsi tersebut dapat dipakai untuk menyediakan daya namun hanya boleh menggunakan salah satu opsi dikarenakan jika menggunakan lebih dari satu opsi maka dapat merusak board. Meskipun daya yang diberikan dapat mencapai 5V, namun setiap pin GPIO (*General Pin Input/Output*) pada development board tetap bekerja pada 3.3V sehingga dibutuhkan modul tambahan bernama logic level shifter agar dapat mengubah daya sinyal dari 5V menjadi 3.3V agar dapat dibaca oleh mikrokontroler, dan mengubah daya sinyal dari 3.3V menjadi 5V agar sinyal dari ESP32 dapat dibaca dengan octal buffer yang mengontrol komunikasi dengan Servo secara TTL half-duplex.(Espresifff Systems, 2022a)



Gambar 2. 8 ESP32-WROOM-32 pada *board* ESP32 DevkitC V4 (Espressif Systems, 2022b)



Gambar 2. 9 Pinout development board ESP32 DevkitC V4(Espressif Systems, 2022b)

Tabel 2. 1 Spesifikasi development board ESP32 DevkitC V4

<b>Mikrokontroller</b>	ESP32-WROOM-32
<b>Core</b>	2 x ESP32-D0WQ6
<b>SPI flash</b>	32 Mbits, 3.3 V
<b>Kristal</b>	40 MHz
<b>Clock Speed</b>	onboard antenna
<b>Dimensi (mm)</b>	$54.40 \pm 0.10) \times (27.90 \pm 0.10) \times (3.10 \pm 0.10$
<b>Wi-Fi</b>	802.11 b/g/n 2.4 GHz ~ 2.5 GH, up to 150 Mbps
<b>Bluetooth</b>	Bluetooth v4.2 BR/EDR and BLE specification
<b>Tegangan Operasi</b>	3.0 V ~ 3.6 V
<b>Memori</b>	520 KB SRAM, 448 KB ROM

## 2.2.6 Dynamixel-SDK

Dynamixel SDK (Software Development Kit) adalah kumpulan dari program yang mengontrol fungsi Dynamixel menggunakan komunikasi paket. Pada dasarnya bahasa pemrograman yang dipakai pada Dynamixel SDK ini adalah C++ atau C dan SDK ini di desain untuk setiap aktuator dan platform yang berbasis Dynamixel. Dalam komunikasi yang digunakan pada Dynamixel, ada dua versi protokol yang dapat dipakai yaitu protokol versi 1.0 dan 2.0. Dalam pengajaran Tugas Akhir ini, jenis protokol yang digunakan adalah protokol 1.0 karena program dari firmware yang dimiliki oleh robot humanoid Ichiro menggunakan protokol tersebut. Pada protokol 1.0, dilakukan melalui komunikasi Serial asinkron dengan 8 bit untuk data, 1 stop bit dan tidak menggunakan parity. Komunikasi Serial half-duplex digunakan karena hanya menggunakan satu buah bus sehingga ketika semua perangkat terkoneksi pada satu bus, maka setiap perangkat lainnya harus berada pada mode input saat salah satu perangkat sedang mengirim data. Kontroler servo Dynamixel selalu berada pada mode input dan hanya berubah menjadi mode output ketika mengirimkan paket instruksi. Bentuk dari paket instruksi pada protokol 1.0 dapat dilihat dari tabel 2.2.

Tabel 2. 2 Bentuk Paket Instruksi pada protokol 1.0

Header 1	Header 2	Packet ID	Length	Instruction	Parameter 1	...	Parameter n	Checksum
0xFF	0xFF	Packet ID	Length	Instruction	Parameter 1	...	Parameter n	CHKSUM

Header pada paket ini mengindikasi awal dari sebuah paket, Packet ID adalah bagian yang menentukan perangkat mana yang ditujuan oleh paket ini. Nilai dari paket ID memiliki jarak dari 0 ~ 253 (0x00~0xFD) sehingga total perangkat yang dapat dipasang dalam satu bus adalah 254 dan jika nilai ID adalah 254 (ID Broadcast) maka setiap perangkat yang tersambung pada bus melakukan instruksi yang ada dalam paket yang dikirim. Length adalah nilai yang menentukan berapa besar Byte dari instruksi, parameter dan checksum. Nilai length dapat dihitung menggunakan persamaan

$$Length = \text{number of Parameters} + 2 \quad (2.1)$$

Pada bagian instruksi, ada 9 (sembilan) jenis instruksi yang dapat dipakai dalam sebuah paket. Jenis-jenis dari instruksi tersebut dapat dilihat dari tabel 2.3.

Tabel 2. 3 Jenis-jenis instruksi dalam paket protokol 1.0

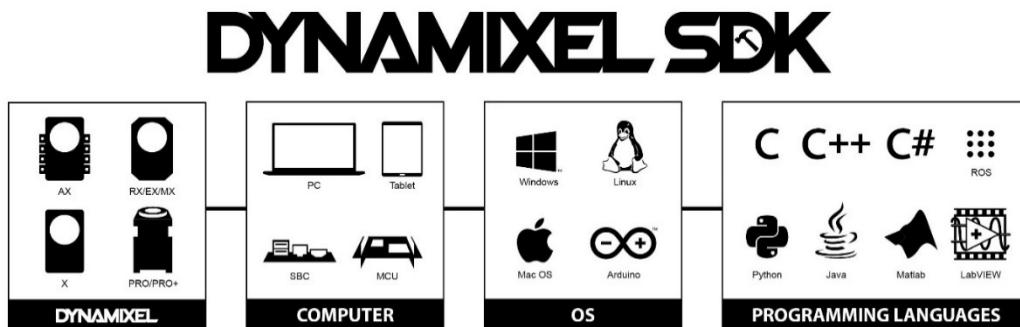
Value	Instruksi	Deskripsi
0x01	Ping	Instruksi untuk mengecek apakah paket telah sampai pada perangkat dengan ID yang telah ditentukan
0x02	Read	Membaca data dari sebuah perangkat
0x03	Write	Write data kepada sebuah perangkat
0x04	Reg Write	Instruksi yang meregister paket instruksi kepada status standby yang nantinya dieksekusi menggunakan instruksi Action
0x05	Action	Instruksi yang mengeksekusi paket yang ter-register sebelumnya menggunakan Reg Write

0x06	Factory Reset	Instruksi untuk mer-reset tabel kontrol ke <i>factory settings</i>
0x07	Reboot	Instruksi untuk reboot Dynamixel
0x08	Sync Write	Instruksi untuk menulis data dengan alamat dan panjang yang sama dalam waktu bersamaan
0x09	Bulk Read	Instruksi untuk membaca data dari alamat yang berbeda dengan panjang yang berbeda pada beberapa perangkat secara bersamaan

Parameter adalah bidang tambahan untuk instruksi. Jumlah parameter yang digunakan berbeda-beda tergantung dengan instruksi yang dipakai. Checksum merupakan bagian yang dipakai untuk mengecek apakah sebuah paket rusak pada saat komunikasi. Formula yang dipakai untuk menghitung *checksum* adalah

$$\text{Checksum} = \sim(\text{ID} + \text{Length} + \text{Instruction} + \text{Parameter1} + \dots + \text{Parameter N}) \quad (2.2)$$

Simbol “~” pada checksum adalah operator *ones complement*. Ketika hasil penjumlahan dari perhitungan checksum melebihi 255 (0xFF), maka yang dipakai hanya lower byte-nya.



Gambar 2. 10 Sistematika Operasi Dynamixel SDK (Robotis, 2022b)

Dalam aplikasinya menggunakan mikrokontroler, robotis sendiri mengeluarkan library yang dapat digunakan pada API (Application Programming Interface) Arduino yang bernama Dynamixel2Arduino. Adanya library ini memungkinkan setiap orang dapat menggunakan implementasi Dynamixel menggunakan mikrokontroler mereka sendiri dan perkembangan dalam SDK ini juga terus berkembang secara pesat karena setiap orang dapat memberikan ide mereka masing-masing dan ide tersebut dapat digabungkan dengan SDK yang ada.(Robotis, 2022b)

### 2.2.7 Dynamixel2Arduino

Library adalah kumpulan kode berupa fungsi-fungsi tambahan yang dapat dipanggil dalam pembuatan sebuah program. Penelitian ini menggunakan library Dynamixel2Arduino dimana setiap fungsi yang tersimpan dalam library ini dapat dipakai agar API (Application Programming Interface) dapat berkomunikasi dengan aktuator Dynamixel dengan menggunakan protokol yang dapat dibaca oleh aktuator tersebut.

Library ini juga berisi contoh-contoh pemakaian kode seperti parameter apa saja yang dipakai jika ingin memanggil sebuah fungsi dan bentuk-bentuk variabel yang harus digunakan

untuk setiap fungsi tertentu. Pada penelitian ini, program mikrokontroler menggunakan library Dynamixel2Arduino untuk memanggil fungsi-fungsi seperti : pembacaan dan penggantian ID, Baudrate, Model Number, kecepatan, posisi, dan fungsi lainnya yang nanti akan dibahas pada Bab 3. Library lainnya yang digunakan dalam program mikrokontroler nantinya bernama “List” yang berfungsi untuk menyimpan data setiap servo yang telah disambungkan dengan kontroler.(ROBOTIS, 2022b)

### **2.2.8 PlatformIO**

PlatformIO merupakan alat tambahan pada teks editor dimana untuk pengerjaan Tugas Akhir ini menggunakan teks editor Visual Studio Code. Alat ini memberikan penggunaanya keleluasan untuk membangun lingkungan *development* yang berbasis IoT (*Internet of Things*) dan mikrokontroler. PlatformIO mendukung lebih dari 20 *framework*, 40 *development platforms*, dan lebih dari 1000 *development board*, memiliki fitur-fitur seperti *Debugging* sehingga jika terdapat kesalahan pada kode maka lokasi kesalahan dapat dilihat dan juga diberikan saran secara langsung untuk pembenarannya, C/C++ *Code Completion*, *code analysis* dimana PlatformIO melakukan analisa pada keseluruhan kode untuk menemukan potensi-potensi kesalahan, jumlah memori yang dipakai dalam mikrokontroler baik dari memori *flash* maupun *ram* , Serial Port Monitor dan banyak kegunaan lainnya. (PlatformIO., 2022).

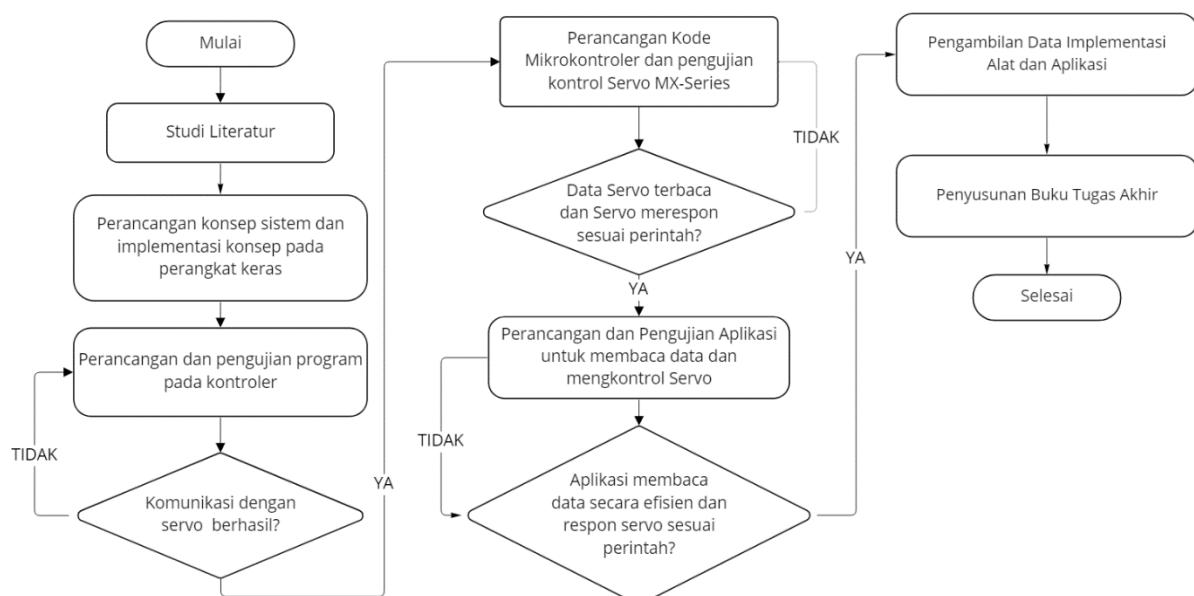
### **2.2.9 Windows Forms**

Windows Forms adalah *framework* yang dapat dipakai untuk membangun aplikasi berdasarkan *User Interface*. Framework ini beserta dengan *platform* pengembangan yang dipakai yaitu Visual Studio, mendukung pembuatan aplikasi secara mudah dan memiliki kapabilitas yang beragam, baik dari alat-alat pengontrolan aplikasi, grafik pendukung, pengolahan data, dan sistem penerimaan *input* dari pengguna. Dalam Windows Forms, sebuah *form* itu sendiri adalah representasi visual yang menunjukkan informasi kepada penggunanya, dan sebuah aplikasi berbasis Windows Forms dapat berisi lebih dari satu form sehingga tampilan berbeda dan kontrol yang berbeda dapat ditunjukkan dalam sebuah aplikasi yang sama berdasarkan fungsi dan parameter yang ditentukan oleh pembuatnya.

Pada penelitian kali ini, dalam pembuatan aplikasi akan menggunakan Windows Forms dengan dukungan .Net Core dikarenakan basis .NET Core merupakan dukungan terbaru yang diberikan oleh perusahaan Microsoft untuk membangun aplikasi yang aman dan perkembangan dari *platform* ini yang terus berjalan. Setelah aplikasi Windows Forms dibuat, maka dilakukan terlebih dahulu perencanaan desain aplikasi dan bagaimana cara aplikasi bekerja karena dengan adanya perencanaan tersebut, maka tujuan akhir pembuatan aplikasi ini yaitu memberikan kemudahan penggunanya untuk dapat membaca data servo dan mengaksesnya dapat tercapai. Pemakaian windows forms relatif mudah dikarenakan *platform* ini mendukung pembangunan program yang mudah dimengerti dimana setiap kontrol yang ada dalam aplikasi dapat langsung dimasukkan sesuai dengan kebutuhan dengan adanya bantuan *ToolStrip* yang dimiliki, sehingga berbagai macam jenis kontrol yang ditawarkan dapat dipakai dalam sebuah form untuk mendukung kegunaannya. (MacDonald, 2002; Microsoft, 2022)

### BAB 3 METODOLOGI DAN PEMODELAN SISTEM

Pada bab 3 ini akan membahas mengenai desain rangkaian elektronik pada perangkat keras, rangkaian kode yang dipakai dalam mikrokontroler dan *Windows Forms* dan implementasi dari kontroler Servo Dynamixel MX-Series menggunakan mikrokontroler ESP32-Wroom-32. Dalam proses mendesain rangkaian elektronik pada perangkat keras, dilakukan merencanakan dan penentuan modul-modul yang akan digunakan, komponen pendukung untuk *voltage regulator* dan IC 74LS241, dan rangkaian skematik untuk menggabungkan setiap komponen yang ada. Selanjutnya untuk rangkaian kode yang akan digunakan pada mikrokontroler, fungsi-fungsi dari library Dynamixel2Arduino beserta dengan data-data alamat dan protokol dari Dynamixel SDK dipakai dalam perangkaian kode tersebut dan juga penambahan kode yang mendukung agar komunikasi Serial dapat dilakukan antara servo dengan mikrokontroler dan komunikasi Serial dapat juga dilakukan antara mikrokontroler dengan aplikasi dengan menggunakan menambahkan *tool* Serial Port pada Windows Forms. Berikut merupakan diagram urutan pelaksanaan pada penelitian ini:

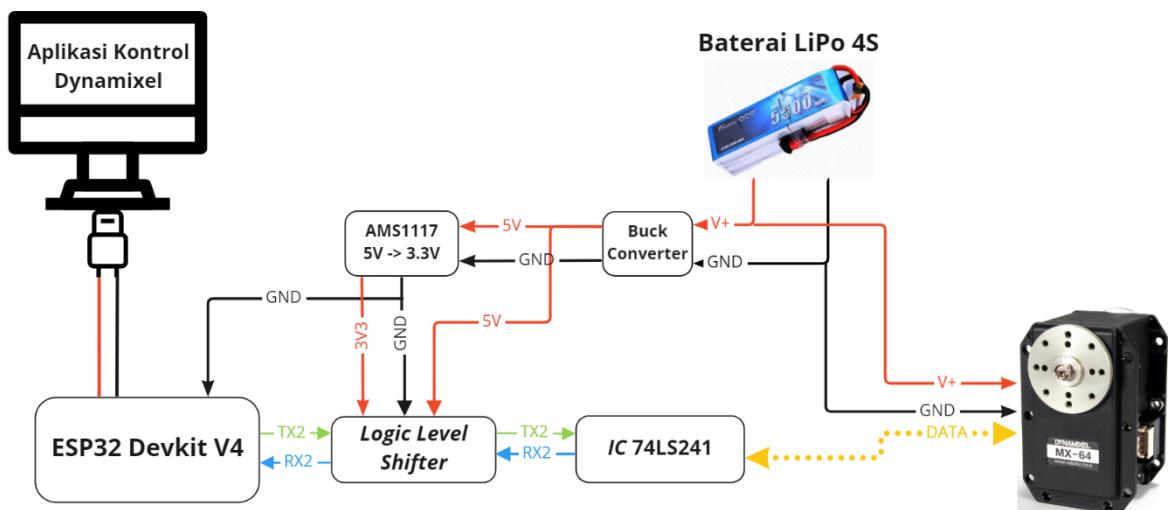


Gambar 3. 1 Diagram Pelaksanaan Tugas Akhir

#### 3.1 Pemodelan Sistem Kontroler

Pada penelitian yang dilakukan, ada dua sistem yang dipakai dalam rangkaian kontroler yaitu yang pertama adalah rangkaian pembagi daya untuk servo dan juga mikrokontroler yang berupa rangkaian pembagi daya yang menggunakan *fuse* sebesar 3A sebagai pengaman rangkaian jika arus melebihi nilai tersebut. Rangkaian ini menerima tegangan maksimal sebesar 16.8 V (tegangan maksimal baterai LiPo 4s) lalu V+ dan Gnd dari baterai diparalelkan dengan input *buck converter* dan pin V+ dan Gnd pada konektor servo. *Buck converter* yang dipakai adalah Mini-360 DC-DC *buck converter* yang dapat menerima tegangan *input* sebesar 4.75-23 Volt dan *output* sebesar 1-17 Volt (Monolithic Power Systems, 2008). *Output* dari *buck converter* ini diatur sehingga bernilai 5V lalu disambungan dengan pin Vin dari *development board* ESP32 dan pin HV (*High Voltage*) pada *logic level shifter* sehingga ESP32 dapat beroperasi dan tegangan acuan 5V dapat diberikan pada *logic level shifter* sebagai tegangan yang nantinya dipakai sebagai referensi tegangan pin RX2/TX2 dan pin data servo yang disambungkan pada IC 74LS241 sebagai tegangan *input*.

Rangkaian kedua yang dipakai adalah rangkaian komunikasi antar servo, ESP32, dan komputer. Rangkaian ini berisi *logic level shifter* dimana pin 3v3 dari *development board* ESP32 digunakan sebagai tegangan referensi LV (*Low Voltage*) pada *logic level shifter* dan pin RX2 pada *development board* ESP32 disambungkan dengan pin LV1 dan TX2 pada *logic level shifter* dan dengan pin LV2, lalu pin GPIO 4 disambungkan dengan pin LV3 sebagai pin yang digunakan untuk memberikan sinyal kontrol kepada servo. Lalu pada *logic level shifter*, pin HV1 yang tersambung dengan RX2 dari LV1 merupakan pin yang tersambung dengan IC 74LS241 agar data yang dikirim dari servo bernilai 5V dapat dikonversi menjadi 3,3 V menjadi sinyal yang dapat dibaca oleh ESP32, begitu pula dengan pin HV2 dan HV3 yang menerima data *transmit Serial* dari pin TX2 dan data output dari GPIO4 dari *development board* sehingga nilai yang ada pada pin HV2 dan HV3 yang disambungkan dengan *octal buffer* dapat dibaca dan dikirim ke pin data servo. Dari Gambar 3.2 dibawah, dapat dilihat bagaimana cara sistem kontroler bekerja dan terkoneksi antar satu sama lain. Interkoneksi antar tiap pin yang digunakan pada modul ESP32 Devkit dengan modul lainnya dijelaskan lebih dalam pada sub-bab 3.3.1.



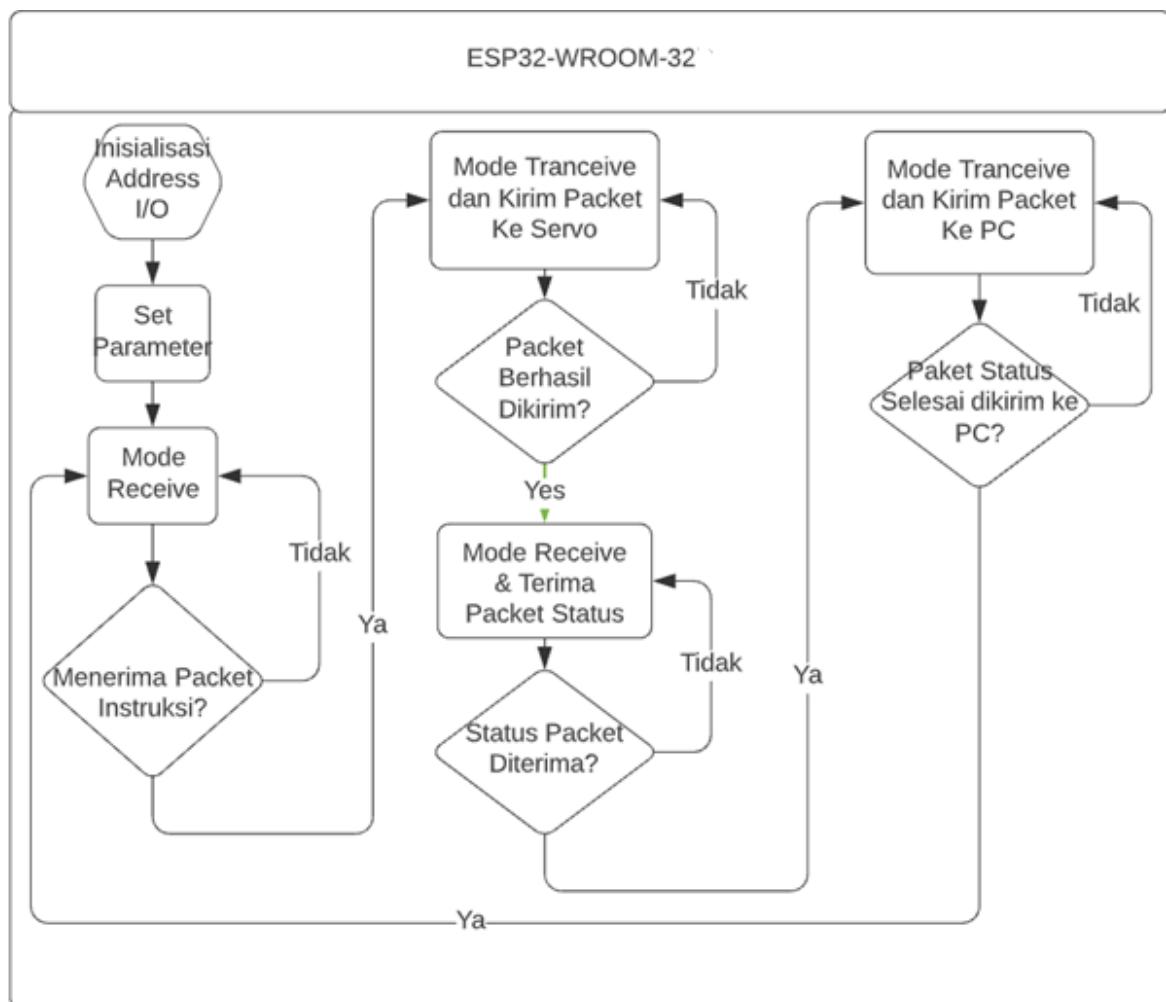
Gambar 3. 2 Diagram Blok Sistem kontroler

### 3.1.1 Metode yang Digunakan

#### 3.1.1.1 Perancangan Konsep Program Kontroler

Dalam penelitian ini, program dibentuk sedemikian rupa menggunakan *library* Dynamixel2Arduino agar API mikrokontroler dapat berkomunikasi dengan servo. Pertama, dilakukan inisialisasi terhadap setiap *library* yang digunakan yaitu Dynamixel2Arduino sebagai *library* yang menyimpan fungsi-fungsi yang dipakai untuk berkomunikasi dengan protokol Dynamixel berbasis Dynamixel SDK, lalu dilakukan inisialisasi untuk *porthandler* dan *packethandler* untuk dapat menginisialisasi protokol 1.0 Dynamixel, setelah itu mode pada kontroler diubah menjadi *Receive* dimana pin RX aktif dan siap untuk menerima paket instruksi yang dikirimkan oleh aplikasi. Jika paket tersebut telah diterima dengan baik oleh kontroler (tidak ada error setelah perhitungan Checksum), maka kontroler akan merubah modenya menjadi *Tranceive* dan mengirimkan paket tersebut kepada servo melalui pin TX dan data yang bernilai 3.3V akan di *pullup* menggunakan *Level Shifter* menjadi 5 Volt dan disalurkan ke komponen 74LS241 yaitu. Adanya komponen Line Driver ini penting dikarenakan komponen ini dibutuhkan dalam komunikasi TTL untuk dapat membedakan data transmitter dan data receiver antara servo dan juga kontroler. Setelah paket instruksi berhasil dikirimkan dari Line Driver menuju servo,

maka kontroler merubah modenya menjadi Receive dan menunggu paket status yang dikirimkan oleh servo. Paket status akan dikirim dari servo kepada kontroler melalui *IC 74LS241N* lalu *Level Shifter* dan masuk ke dalam kontroler sehingga data paket status diterima secara sepenuhnya oleh kontroler dan kontroler dapat mengirimkan data pada PC melalui komunikasi Serial biasa. Setelah siklus ini selesai, kontroler akan menunggu lagi paket dari PC untuk dapat mulai siklus selanjutnya.



Gambar 3. 3 Diagram Konsep Program

Sebelumnya dijelaskan ada dua buah paket yaitu paket instruksi dan paket status, kedua paket ini merupakan sistem yang digunakan pada Dynamixel-SDK agar kontroler dapat berkomunikasi dengan servo Dynamixel dengan menggunakan protokol yang ada. Dalam penggeraan penelitian ini, dikarenakan protokol yang digunakan adalah protokol 1.0, maka bentuk paket instruksi memiliki susunan seperti pada tabel 3.1 dan contohnya dapat dilihat pada gambar 3.4, sedangkan untuk paket status yang dikirimkan dari servo kepada kontroler yang nantinya digunakan untuk validasi pengiriman data menggunakan checksum memiliki susunan seperti pada tabel 3.2 dengan contohnya dapat dilihat pada gambar 3.5.

Tabel 3. 1 Bentuk Paket Instruksi

Header1	Header2	Packet ID	Length	Instruction	Param 1	...	Param N	Checksum
---------	---------	-----------	--------	-------------	---------	-----	---------	----------

0xFF	0xFF	Packet ID	Length	Instruction	Param 1	...	Param N	CHKSUM
------	------	-----------	--------	-------------	---------	-----	---------	--------



Gambar 3. 4 Implementasi paket instruksi dalam komunikasi

Tabel 3. 2 Bentuk Paket Status

Header1	Header2	Packet ID	Length	Error	Param 1	...	Param N	Checksum
0xFF	0xFF	ID	Length	Error	Param 1	...	Param N	CHKSUM



Gambar 3. 5 Paket status yang dikirimkan dari servo

Dapat dilihat dari gambar dan tabel diatas bahwa dalam implementasinya, komunikasi protokol 1.0 Dynamixel ini seperti pada gambar 3.4 dan 3.5 jika berpacu pada *OSI Model (Open Systems Interconnection model)* menggunakan implementasi *physical layer* dikarenakan transmisi data melewati komunikasi serial menggunakan nilai *RAW* yang melewati medium yaitu pin TX dan RX masing-masing. Dalam sistematika komunikasi protokol 1.0 Dynamixel, checksum digunakan untuk mengetahui apabila sebuah paket baik itu paket instruksi maupun paket status mengalami kerusakan apakah tidak. Nilai checksum dihitung dengan menggunakan rumus 2.2 dimana jika dihitung menggunakan contoh dari gambar 3.4 untuk perhitungan checksum paket instruksi dimana isi paketnya adalah ID=1(0x01), Length=4(0x04), Instruction=2(0x02), Parameter 1=36(0x24), Parameter2=2(0x02), maka nilainya adalah :

$$\begin{aligned} \text{Checksum} &= \sim(0x01+0x04+0x02+0x24+0x02) = \sim(0x2D) \\ \text{Checksum} &= 0xD2 \end{aligned} \quad (3.1)$$

Dari perhitungan diatas, didapatkan nilai checksum yaitu 0xD2 dan dari gambar 3.4 dapat dilihat bahwa nilai tersebut merupakan nilai checksum yang ada pada paket instruksi sehingga dapat disimpulkan tidak ada *error* pada paket instruksi dan servo dapat eksekusi paket instruksi yang dikirimkan. Setelah paket instruksi dikirim, maka servo mengirimkan paket status seperti pada gambar 3.5 dimana untuk dapat validasi apabila tidak terdapat *error* pada paket status yang dikirimkan maka dihitung kembali checksumnya yaitu dengan isi paket status yang terdiri dari ID=1(0x01), Length=4(0x04, Error=0(0x00), Parameter 1=0(0x00), Parameter2=0(0x00), maka nilai checksumnya dihitung dengan :

$$\begin{aligned} \text{Checksum} &= \sim(0x01+0x04+0x00+0x00+0x00) = \sim(0x2D) \\ \text{Checksum} &= 0xFA \end{aligned} \quad (3.2)$$

Dari perhitungan diatas, maka jika dibandingkan nilai perhitungan checksum yaitu 0xFA dengan nilai checksum yang dikirimkan oleh paket status, maka dapat disimpulkan bahwa tidak terdapat *error* pada paket status sehingga setiap nilai paket status diterima dan paket dapat dibaca dengan baik.

## 3.2 Alat dan Bahan

### 3.2.1 ESP32 DevkitC V4

Dalam penggunaan Mikrokontroler ESP32-Wroom-32 yang dipakai sebagai pusat pengkontrolan servo dalam kontroler ini digunakan *development board* yaitu ESP32 DevkitC V4 dimana *development board* ini sudah memiliki regulator tegangan sehingga tegangan kerja dari ESP32 yang pada awalnya bekerja pada 3.0 ~3.6 V dapat dinyalakan dengan tegangan 5V dikarenakan pada *board* ini telah tersambung regulator AMS1117 yang dapat menurunkan tegangan menjadi 3.3V secara konstan dan efisien. Pada *board* ini juga telah terpasang *USB-to-UART Bridge* sehingga mikrokontroler ESP32-Wroom-32 yang ada pada board dapat diprogram langsung menggunakan *port micro-USB* dan juga menerima tegangan 5V menggunakan regulator tegangan yang telah dibahas sebelumnya.

### 3.2.2 Mini 360 DC-DC Buck Converter

Untuk dapat menurunkan tegangan yang masuk dari baterai Lithium Polymer 4s yang dipakai, dibutuhkan modul penurun tegangan agar tegangan masuk yang dibutuhkan oleh *IC* 74LS241 dan *logic level shifter* agar tidak melebihi tegangan maksimum yang dapat diterima oleh modul-modul tersebut. Pada penelitian ini, modul Mini 360 Buck Converter dipakai karena dengan ukurannya yang kecil, namun tegangan *input* yang dapat diterima memiliki rentang yang dapat disebut tinggi dan arus yang dapat diterima besar dengan nilai 2A. Spesifikasi dari modul ini dapat dilihat pada Tabel 3.1 dibawah.

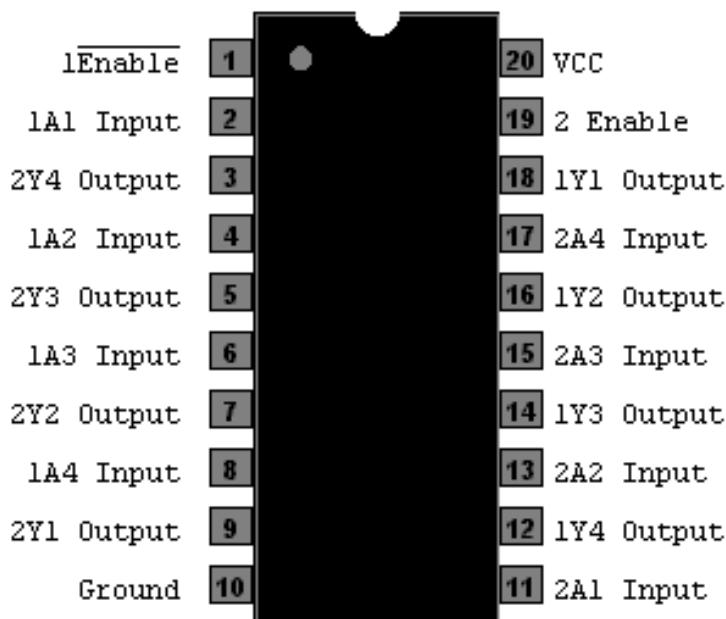
Tabel 3. 3 Desain Spesifikasi Konverter Buck boost

No.	Parameter	Nilai
1.	Tegangan <i>Input</i>	4.75 V – 23 V
2.	Tegangan <i>Output</i>	1.0 V – 17 V
3.	Arus <i>Output</i>	3A (Maksimum)
4.	Efisiensi konversi	96 %

5.	<i>Switching Frequency</i>	340 KHz
----	----------------------------	---------

### 3.2.3 IC 74LS241

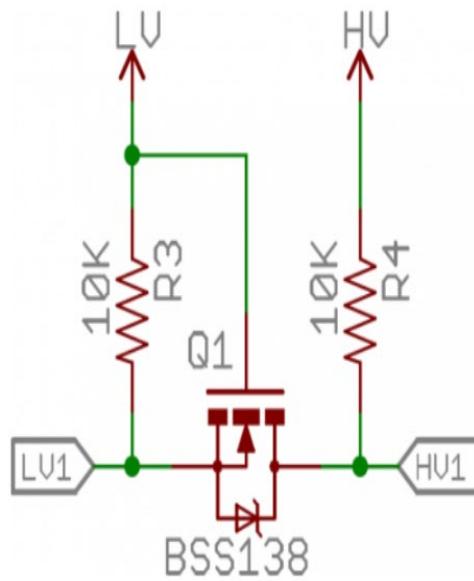
Dalam penelitian ini, *IC (Integrated Circuit)* 74LS241 adalah *octal buffer* digunakan dalam komunikasi TTL half-duplex yang juga dipakai oleh servo Dynamixel. *IC* ini beguna untuk meningkatkan kinerja komunikasi Serial berbasis TTL half-duplex dimana fitur yang ditawarkan yaitu peningkatan performa dan densitas dari *three memory address drivers*, *clock driver*, dan *bus-oriented receiver* dan *transmitter*.



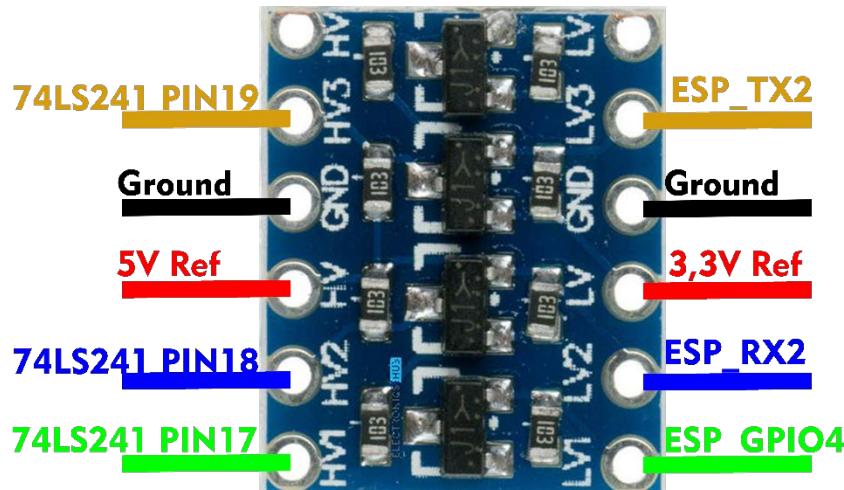
Gambar 3. 6 Pinout 74LS241

### 3.2.4 Logic Level Shifter

Modul ini digunakan untuk merubah nilai sinyal Serial yang bekerja pada ESP32 dimana setiap pin GPIO pada *development board* ini bekerja pada tegangan 3.3V sedangkan *IC* 74LS241 yang menjembatani komunikasi antar mikrokontroler dengan servo dikarenakan *IC* bekerja pada tegangan 5V. Dengan menggunakan modul ini, maka pin TX2 dan GPIO 4 dari ESP32 dinaikkan dari tegangan 3.3 V menjadi 5V dan sinyal servo yang diterima oleh IC dan dikirimkan kepada ESP32 dari bernilai 5V dapat diturunkan menjadi 3.3V. Modul ini dapat merubah tegangan sinyal dikarenakan komponen di dalamnya yang berisi *N-Channel Mosfet* yaitu BSS138 dan dua buah resistor 10 KOhm yang di pullup dengan tegangan 3.3V pada pin LV dan 5V pada pin HV sehingga rangkaian dapat bekerja secara bidireksi. Rangkaian modul ini dapat dilihat pada Gambar 3.7 dan Gambar 3.8.



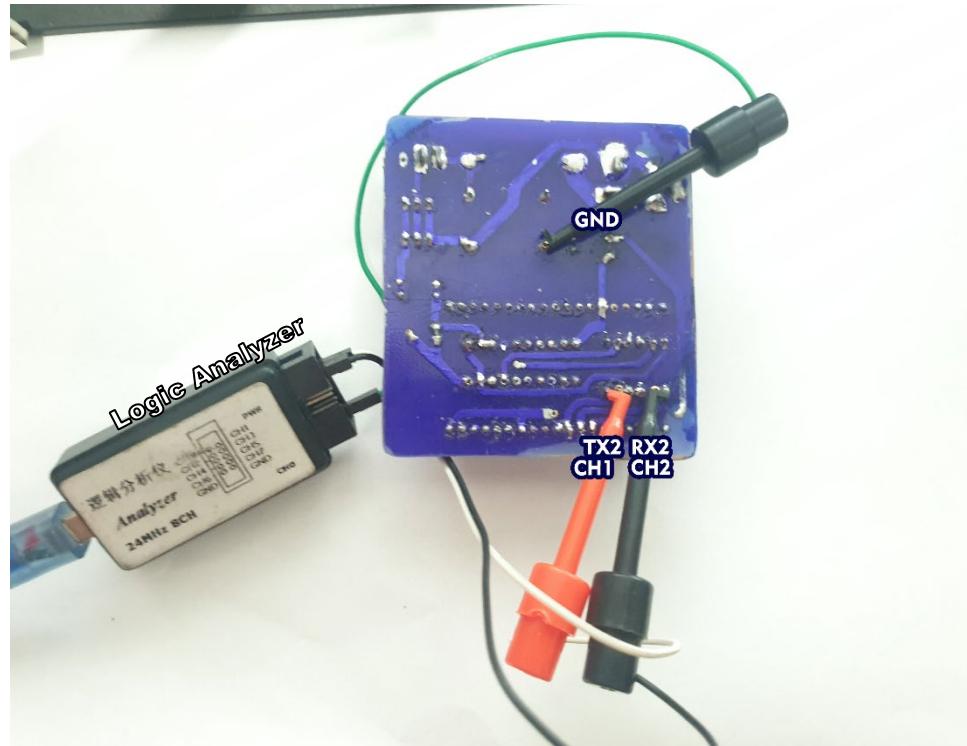
Gambar 3. 7 Rangkaian logic level shifter per-Channel



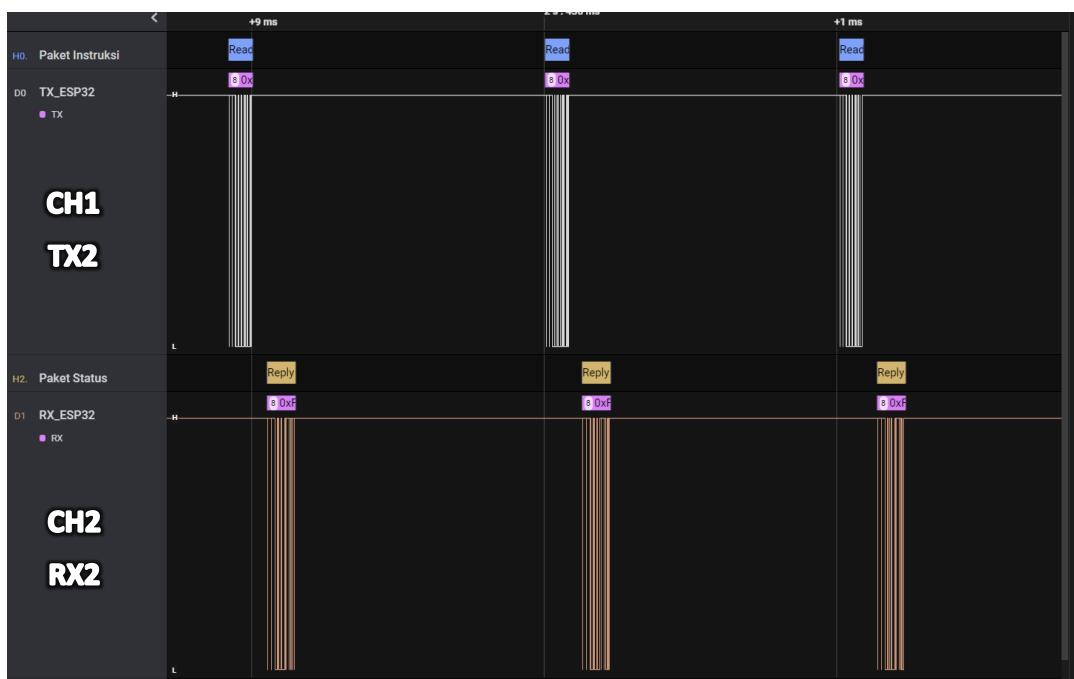
Gambar 3. 8 Pinout dari *logic level shifter*

### 3.2.5 USB Logic Analyzer

Modul ini merupakan alat yang digunakan dalam pengujian nantinya untuk membaca bentuk sinyal yang ada pada kontroler beserta. Logic Analyzer dipakai untuk dapat melihat bentuk sinyal yang keluar dan masuk dari pin RX2/TX2 pada development board ESP32 sehingga bentuk sinyal, frekuensi, dan durasi sinyal dapat dilihat dengan menggunakan aplikasi bernama Logic. Bentuk penggunaan *USB Logic Analyzer* pada rangkaian kontroler dapat dilihat pada Gambar 3.9 dan aplikasi Logic yang digunakan untuk melihat hasil pembacaan sinyal dapat dilihat pada Gambar 3.10.



Gambar 3. 9 *USB Logic Analyzer*



Gambar 3. 10 Aplikasi *Logic* untuk pembacaan Sinyal

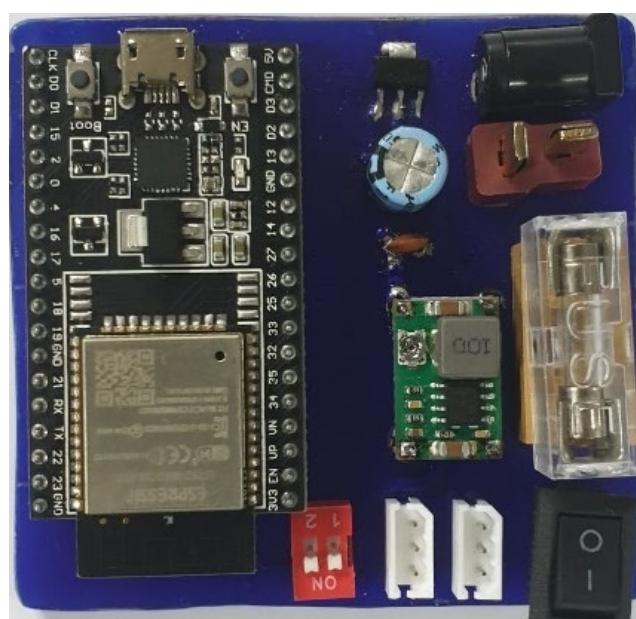
### 3.3 Implementasi Sistem Kontroler

Dengan menggunakan metodologi yang ada, maka penelitian dilakukan sesuai dengan urutan yang telah tercantum pada Gambar 3.1

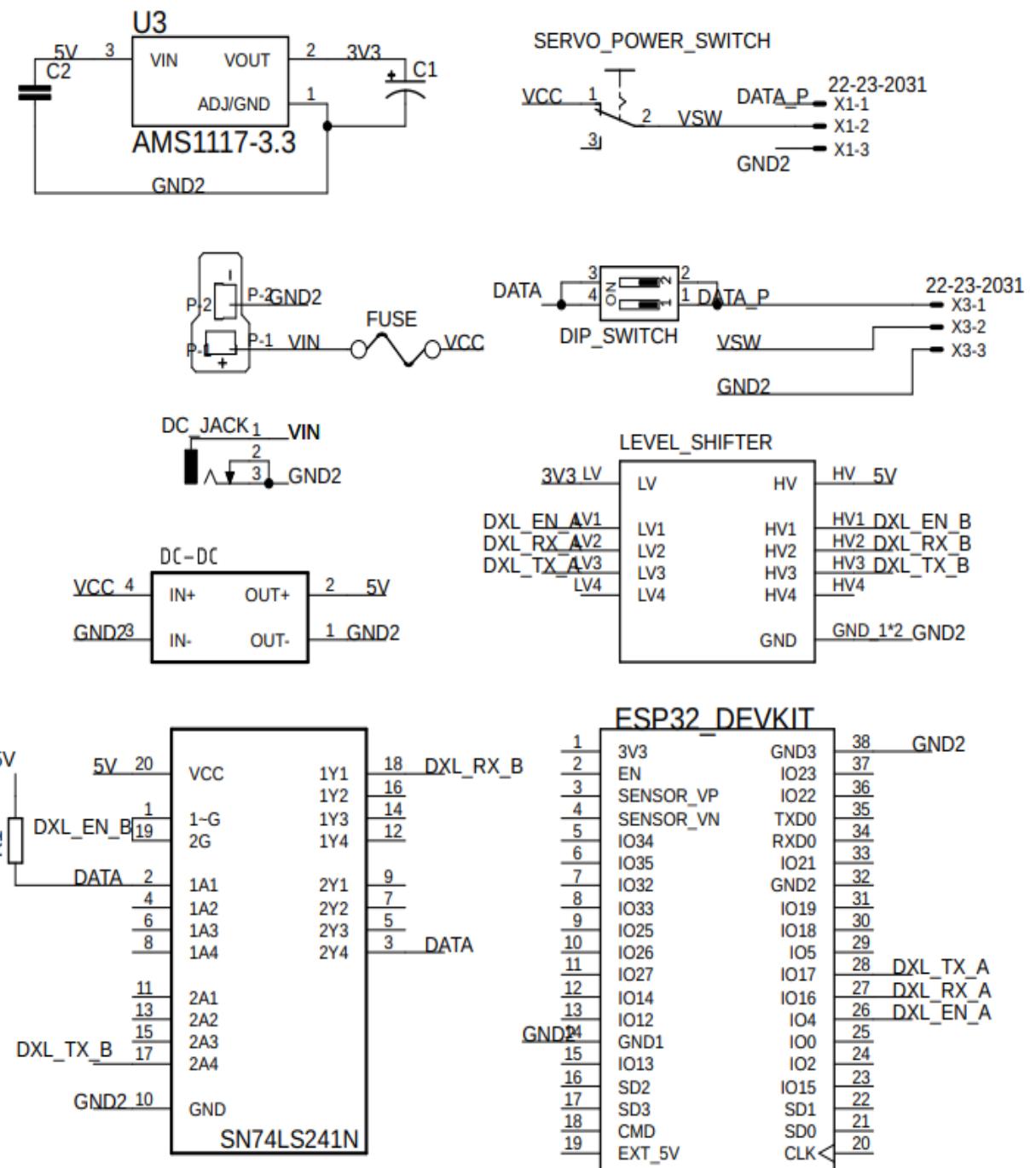
#### 3.3.1 Rangkaian skematik *PCB* kontroler

Rangkaian *PCB* dibuat sedemikian rupa dengan konsep yang telah ditampilkan pada diagram blok pada Gambar 3.2. Rangkaian skematik sistem kontroler dengan setiap modul yang dibutuhkan

ditunjukkan pada Gambar 3.10 yang dibuat menggunakan aplikasi *Eagle* versi 9.6.2 dan menunjukkan bagaimana setiap modul dirangkai dalam sistem dari pin konektor servo sampai sambungan jalur data yang masuk pada *development board* ESP32 DevkitC V4. Dalam perangkaian *PCB* ini, seperti yang terlihat pada Gambar 3.10, ada dua cara yang dapat digunakan untuk memberikan daya pada modul kontroler ini yaitu menggunakan konektor *T-Plug* dan juga menggunakan *DC Jack* dimana jalur untuk tegangan positif menggunakan nama VIN lalu tegangan negatif menggunakan GND2, lalu jalur VIN ini disambungkan kepada kaki soket sekring dimana sekring ini digunakan untuk menjaga jika arus yang mengalir pada modul melebihi nilai yang diinginkan. Setelah itu jalur yang keluar dari sekring bernama VCC disambungkan kepada saklar dan juga pin IN+ pada *buck converter* dimana saklar digunakan untuk memutus dan menyambungkan arus yang masuk untuk memberikan daya kepada setiap servo yang tersambung pada konektor Dynamixel 3 pin dan IN+ sebagai tegangan positif yang masuk ke dalam *buck converter* dimana tegangan hasil penurunan ini keluar pada pin OUT+ dengan nama jalur 5V. Jalur 5V kemudian disambungkan kepada pin HV sebagai tegangan referensi 5V untuk *Logic Level Shifter*, pin 20(VCC) pada *IC* 74LS241N, dan juga pin input pada regulator tegangan AMS1117 yang digunakan untuk menurunkan tegangan dari 5V menjadi 3.3V yang nantinya disambungkan dari jalur 3V3 untuk memberikan tegangan referensi pada *Logic Level Shifter*. Rangkaian penyambungan data antara servo dengan mikrokontroler dimulai dari pin data servo yang diterima dari konektor Dynamixel dengan nama jalur DATA\_P yang kemudian disambungkan kepada *DIP Switch* dimana jalur lainnya yang bernama DATA disambungkan kepada pin 2 dan 3 pada *IC* 74LS241N dimana kemudian dibagi menjadi jalur DXL\_TX\_B pada pin 17 dan juga DXL\_RX\_B pada pin 18 untuk kemudian disambungkan kepada *Logic Level Shifter* untuk berkomunikasi dengan mikrokontroler untuk data *input* dan *output* antara keduanya dengan pin 1 dan 19 pada *IC* ini disambungkan dengan jalur DXL\_EN\_B untuk menerima input perintah dari mikrokontroler yang nantinya disalurkan kepada servo, pin-pin data dari *IC* 74LS241N yang disambungkan dengan *Logic Level Shifter* menghasilkan tiga buah jalur dengan tegangan kerja 3.3V dimana ketiga output ini adalah DXL\_TX\_A yang disambungkan kepada pin GPIO17 pada ESP32\_DEVKIT, lalu jalur DXL\_RX\_A yang tersambung pada pin GPIO16, dan jalur DXL\_EN\_A disambungkan kepada pin GPIO4 pada mikrokontroler agar komunikasi serial berhasil dilakukan. Hasil *PCB* yang telah dibuat pada gambar 3.11 dan gambar rangkaian skematik ini dapat dilihat pada gambar 3.12.



Gambar 3. 11 Hasil *PCB* yang dibuat



Gambar 3. 12 Rangkaian skematik PCB kontroler

### 3.3.2 Perancangan Program Kontroler

Dalam proses pembangunan program yang akan dipakai pada mikrokontroler, digunakan ekstensi PlatformIO dengan teks editor Visual Studio Code. Dalam program yang dibuat, pertama dilakukan inisialisasi pemaggilan *library* Dynamixel2Arduino yang menyimpan fungsi-fungsi pengkontrolan servo Dynamixel, *library* List yang dipakai untuk menyimpan data pengidentifikasi masing-masing servo, lalu dilakukan deklarasi variabel yaitu float protocol dimana nilainya adalah 1.0 agar sesuai dengan protokol 1 Dynamixel-SDK yang digunakan, lalu pin 16 sebagai pin DXL\_RX yang digunakan untuk pin *receive* serial, pin 17 sebagai pin DXL\_TX yang digunakan untuk pin *transmit* serial, dan juga pin 4 sebagai DXL\_Enable yang dipakai untuk

mengirimkan paket instruksi kepada servo. Setelah itu dilakukan deklarasi inisialisasi *portHandler* untuk menentukan parameter mikrokontroler agar Dynamixel2Arduino dapat bekerja pada *development board* ESP32 DevkitC V4. Setelah itu dilakukan inisialisasi *task handler* agar proses jalannya program dapat dibagi pada *core 0* dan *core 1* ESP32 dimana nantinya *core 0* dipakai untuk menjalankan komunikasi dengan servo dan *core 1* dipakai untuk menjalankan komunikasi dengan aplikasi. Kedua komunikasi pada *core 0* dan *core 1* pada ESP32 menggunakan m etode Serial dimana utamanya *core 0* menggunakan Serial half-duplex dan *core 1* menggunakan Serial full-duplex, namun untuk beberapa fungsi tertentu, masing-masing core juga dapat menggunakan jenis Serial lainnya. Setelah melakukan inisialisasi *task handler*, dilakukan deklarasi akan variabel-variabel yang nantinya dipakai untuk menyimpan data servo yang telah dibaca oleh *core1*, dan variabel yang menerima perintah yang dikirim oleh aplikasi pada mikrokontroler yang berisi perintah pembacaan tersebut. Deklarasi variabel data yang dibaca dari servo dalam potongan program ini dapat dilihat pada Gambar 3.13 dan penjelasan setiap variabel pada Tabel 3.2.

```
//core 0 (instruksi pembacaan data dari masing-masing servo dan hasilnya disimpan dalam variabel)
PosN = dxl.getPresentPosition(id);
SpeedN = dxl.getPresentVelocity(id);
SpeedReal = dxl.getPresentVelocity(id, UNIT_RPM);
Torque = dxl.getTorqueEnableStat(id);
PosG = dxl.readControlTableItem(GOAL_POSITION, id);
SpeedG = dxl.readControlTableItem(GOAL_VELOCITY, id);
p_gain = dxl.readControlTableItem(P_GAIN, id);
i_gain = dxl.readControlTableItem(I_GAIN, id);
d_gain = dxl.readControlTableItem(D_GAIN, id);
ret_delay = dxl.readControlTableItem(RETURN_DELAY_TIME, id)* 2;
temp = dxl.readControlTableItem(PRESENT_TEMPERATURE, id);
Current = dxl.getPresentCurrent(id);
CurrentReal = dxl.getPresentCurrent(id, UNIT_MILLI_AMPERE);
//core 1
Serial.println((String)"ID" + id + "Baud" + Sbaud + "Model" + Model + "Prot" + protocol
+ "PosN" + PosN + "PosG" + PosG + "SpeedN" + SpeedN + "SpeedG" + SpeedG +
"Current" + Current + "Torque" + Torque +"pgain"+ p_gain + "igain" + i_gain + "dgain"
+ d_gain + "retdelay" + ret_delay + "temp" + temp + "speedReal" + SpeedReal+ "RealCurr"
+ CurrentReal);
```

Gambar 3. 13 Deklarasi variabel dan pengiriman data hasil pembacaan dari servo

Tabel 3. 4 Variabel data servo yang dibaca

Nama Variabel	Keterangan
ID	Nilai yang dipakai untuk mengidentifikasi sebuah servo (0-253), ID 254 dipakai sebagai ID broadcast dan seluruh servo akan melakukan perintah dari paket instruksi jika ID ini dipanggil.
Baud (Baudrate)	Kecepatan komunikasi serial antara kontroler dengan servo
Model (Model Number)	Nilai yang menyimpan nomor model servo (MX-28 = 29, MX-64 = 310, MX-128 = 320)
Prot (Protokol)	Protokol DynamixelSDK yang dipakai pada paket instruksi atau status

PosN (Posisi saat ini)	Nilai posisi dari sumbu putar aktuator terhadap titik 0
PosG (Posisi Goal)	Nilai posisi yang diinginkan pada sumbu putar aktuator
SpeedN (Kecepatan saat ini)	Kecepatan sumbu putar untuk mencapai posisi yang diperintahkan dalam satuan RAW (Nilai Analog)
SpeedG (Kecepatan Goal)	Kecepatan perputaran yang diinginkan
SpeedReal	Kecepatan sumbu putar untuk mencapai posisi yang diperintahkan dalam satuan RPM ( <i>Rotations per minute</i> )
Temp (Temperatur)	Temperatur yang dibaca oleh servo dalam satuan Celcius
Current	Arus yang dipakai oleh servo (Satuan RAW (nilai analog))
CurrentReal	Arus yang dipakai oleh servo (Satuan milliAmpere)
Status Torsi	Jika 1 maka servo akan mempertahankan posisinya sedangkan 0 maka sumbu servo dapat diputar secara bebas
P Gain	Nilai P Gain dari kontroler PID yang ada dalam servo
I Gain	Nilai I Gain dari kontroler PID yang ada dalam servo
D Gain	Nilai D Gain dari kontroler PID yang ada dalam servo
Return Delay	Durasi yang dibutuhkan servo untuk mengirimkan paket status setelah menerima paket instruksi

Setiap pembacaan variabel yang pada Tabel 3.2 dilakukan pada *core 0* ESP32 dan setelah pembacaan dilakukan *core 1* mengirimkan data-data yang telah dibaca melewati komunikasi serial kepada *port USB* yang tersambung dengan *development board* sehingga aplikasi dapat dipakai untuk menerima data Serial yang dikirim oleh ESP32. Pada Gambar 3.14, dapat dilihat bagaimana hasil pembacaan data servo dikirimkan kepada komputer melalui Serial.

```
ID1Baud100000Model1320Prot1.00PosN998PosG998SpeedN0SpeedG0Current2048Torquelpgain32igain0dgain0retdelay0temp35speedReal0RealCurro
```

Gambar 3. 14 Data yang dikirim oleh ESP32 saat *Single Servo Control*

Pada Gambar 3.14, dapat dilihat bahwa program pada *core 0* melakukan pembacaan setiap variabel yang ada pada dalam Tabel 3.2 dan hasil pembacaan dikirim melalui *Serial port* yang dikontrol oleh Serial ESP32 sehingga pembacaan dapat dikirim kepada aplikasi yang dilakukan pada setiap siklus yang memiliki jarak durasi yang ditentukan sebesar 25mS(*milli seconds*).

Dalam program kontroler ini, ada 2 (dua) mode yang dapat dipilih yaitu pembacaan data servo secara detail dengan cara pembacaan data pada satu buah servo, dan pembacaan dan pengkontrolan data pada banyak servo. Program ini memiliki dua metode dikarenakan pada aplikasi yang dibuat terdapat dua mode yaitu “*Single Servo Control*” dan “*Multiple Servos Control*”, namun pada pembacaan data banyak servo, data yang dikirim hanya berupa *ID* setiap servo yang tersambung, *Model Number* servo-servo tersebut, dan posisi setiap servo yang

dikontrol secara bersamaan menggunakan metode pemanggilan *ID* 254 (*Broadcast ID*) dikarenakan dengan pemanggilan *ID* tersebut, setiap servo yang tersambung akan mengikuti paket instruksi yang dikirim berdasarkan metode yang ada pada *DynamixelSDK*. Setelah memberikan paket instruksi, data setiap servo dibaca untuk mengetahui akurasi data pengiriman antara posisi yang diinginkan dengan posisi yang dibaca oleh servo itu sendiri. Kode pembacaan data dari mode *Multiple Servos Control* dapat dilihat pada Gambar 3.15. Hasil pembacaan yang dikirim kepada *Serial port* dapat dilihat pada Gambar 3.16.

```
for(int i =0;i<nilai.getSize();i+=2){  
    id = nilai[i];  
    Model = nilai[i+1];  
    PosN = dxl.getPresentPosition(id);  
    Serial.println((String)"ID" + id + "Model" + Model + "Pos" + PosN);  
}
```

Gambar 3. 15 Pengiriman data hasil pembacaan pada mode *Multiple Servos Control*

ID1Mode1320Pos998
ID2Mode129Pos998
ID3Mode1310Pos999
ID1Mode1320Pos998
ID2Mode129Pos998
ID3Mode1310Pos999
ID1Mode1320Pos998
ID2Mode129Pos998
ID3Mode1310Pos999
ID1Mode1320Pos998
ID2Mode129Pos998
ID3Mode1310Pos999
ID1Mode1320Pos998
ID2Mode129Pos998
ID3Mode1310Pos999
ID1Mode1320Pos998
ID2Mode129Pos998
ID3Mode1310Pos999

Gambar 3. 16 Pembacaan Posisi untuk setiap ID dan Model servo yang tersambung

Dalam upaya pengkontrolan servo, ada beberapa fungsi yang diberikan agar dapat dilakukan perubahan data yang diinginkan. Fungsi-fungsi yang dibuat terdiri dari :

1. Perubahan *ID*
2. Perubahan *baudrate*
3. Pengkontrolan Posisi Servo
4. Perubahan *return delay*

## 5. Perubahan *gain* untuk parameter P, I, dan D

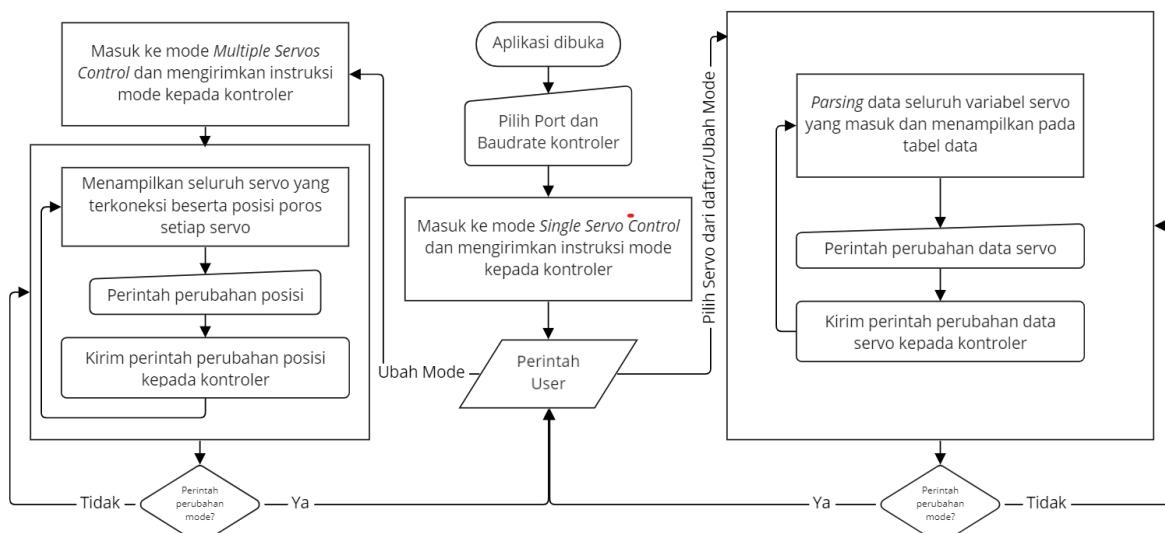
Fungsi-fungsi tersebut diakses dengan cara memberikan *input* berupa variabel dengan tipe String yang dibaca oleh program yang diberikan pada *core 1* sehingga jika *input* String yang diberikan sesuai dengan parameter yang telah ditentukan, maka ESP32 *core 1* akan membaca nilai index pada parameter tersebut dan mengambil angka yang ada pada String yang masuk dengan menggunakan kode seperti pada gambar 3.17. Program mikrokontroler dapat dilihat pada Lampiran 1.

```
while(Serial.available()>0){
    Input_str = Serial.readString();
    int8_t indexID = Input_str.indexOf("ID");
    int8_t indexBaud = Input_str.indexOf("Baud");
    int8_t indexPos = Input_str.indexOf("Pos");
    int8_t indexP = Input_str.indexOf("PG");
    int8_t indexI = Input_str.indexOf("IG");
    int8_t indexD = Input_str.indexOf("DG");
    int8_t indexRet = Input_str.indexOf("Ret");}
```

Gambar 3. 17 Kode Pengkontrolan Servo

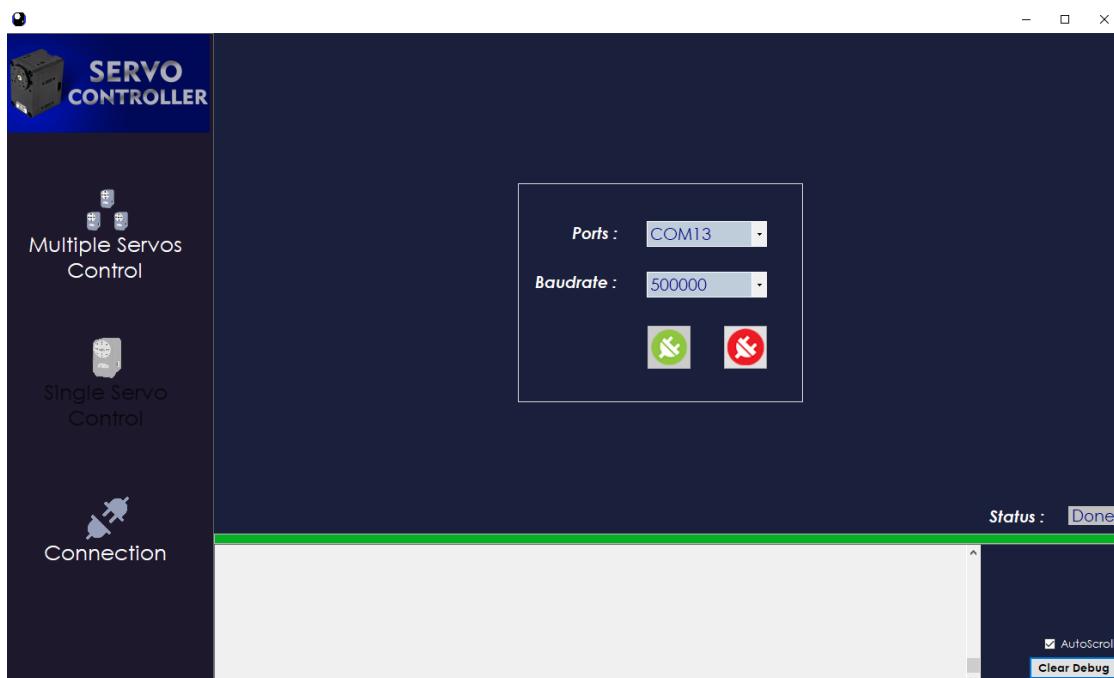
### 3.3.2.1 Perancangan Program Aplikasi berbasis *Windows Forms*

Dalam penelitian ini, aplikasi dibuat agar data servo hasil pembacaan ESP32 dapat dibaca dengan mudah oleh penggunanya dan perubahan data pada servo dapat dilakukan tanpa perlu menggunakan aplikasi lainnya yang membutuhkan *driver* tertentu, sehingga pengkontrolan dapat dilakukan pada semua komputer dengan basis OS Windows. Aplikasi ini dibuat menggunakan *toolkit* Visual Studio dan menggunakan bahasa pemrograman C#. Meskipun aplikasi menggunakan bahasa pemrograman yang berbeda dengan bahasa ESP32 yang menggunakan C++, namun dengan adanya sistematika komunikasi secara Serial, maka data dapat dikirim antar ESP32 dengan Aplikasi. Langkah yang dilakukan dalam pembangunan aplikasi ini mengikuti diagram yang dapat dilihat pada gambar 3.18.



Gambar 3. 18 Diagram Pembangunan Aplikasi

Langkah pertama yang dilakukan dalam pembangunan aplikasi ini adalah dengan membuat project baru pada Visual Studio dan memilih template Windows Forms dengan dukungan .NET Core. Lalu setelah project dibuat, maka dilakukan perencanaan pada bagaimana cara aplikasi berinteraksi dengan sistem kontroler servo dan tampilan apa saja yang akan disajikan di dalam aplikasi ini. Dalam aplikasi yang dibuat, ada tiga form yang digunakan, yaitu form pertama yang berjudul “*Connections*” dimana form ini yang dipakai agar pengguna dapat memilih *Port* yang ingin dipakai beserta dengan *Baudrate* yang akan digunakan untuk dapat berkomunikasi dengan *development board* ESP32. Tampilan form ini dapat dilihat dari Gambar 3.19.

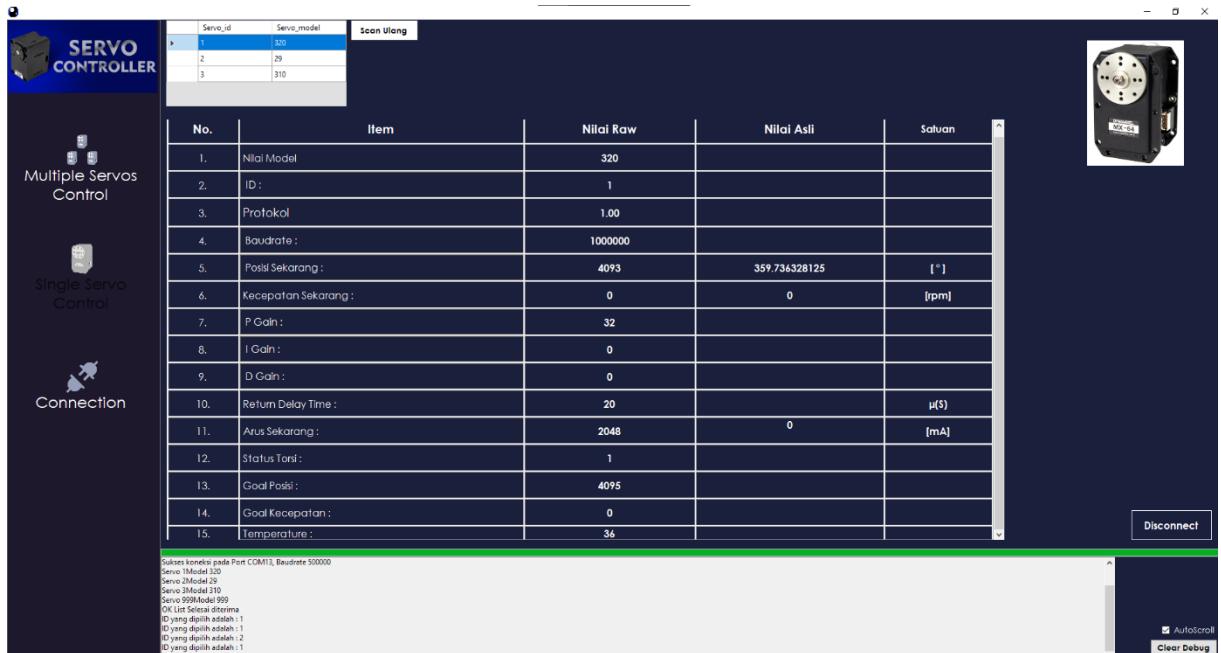


Gambar 3. 19 Tampilan pemilihan *port* dan *baudrate* modul kontroler

Pada form connection diatas, ada 3 buah panel utama yang menjadi fokus utama form ini. Pertama pada panel utama yang berguna untuk memberikan pengguna akses untuk memilih *Port* yang ingin digunakan dan besaran *Baudrate* yang akan digunakan dimana nilainya didapat dari besaran *baudrate* yang digunakan ESP32 untuk komunikasi Serial. Lalu pada panel kedua di sebelah kiri ada 3 buah pilihan yaitu Connection form itu sendiri, “*Single Servo Control*” dan “*Multiple Servos Control*”. Ketiga pilihan inilah yang nantinya akan merubah tampilan pada form utama pada saat salah satu menu tersebut di klik dan nantinya akan menggantikan tampilan dimana pilihan *Port* dan *Baudrate* ada sekarang. Panel ketiga adalah panel debug yang berguna untuk menampilkan teks jika sewaktu-waktu terjadi error pada komunikasi serial dan menampilkan *output* yang membantu pengguna untuk mengetahui permasalahan yang dihadapi pada komunikasi.

Form selanjutnya yang akan dibahas adalah form “*Single Servo Control*” dimana form ini berisi setiap data yang dikirim oleh kontroler seperti yang tertera pada Gambar 3.14 dan data tersebut dimasukkan ke dalam tabel yang membuat pembacaan data lebih mudah dan pengkontrolan servo dapat dilakukan dengan fungsi dalam kontroler servo yang dijelaskan pada sub-bab sebelumnya. Tampilan form “*Single Servo Control*” ini dapat dilihat pada Gambar 3.20.

Dalam tabel ini, setiap fungsi yang dijelaskan sebelumnya diberikan tombol sehingga jika tombol tersebut di klik maka akan keluar opsi pengkontrolan variabel yang ingin diubah nilainya.

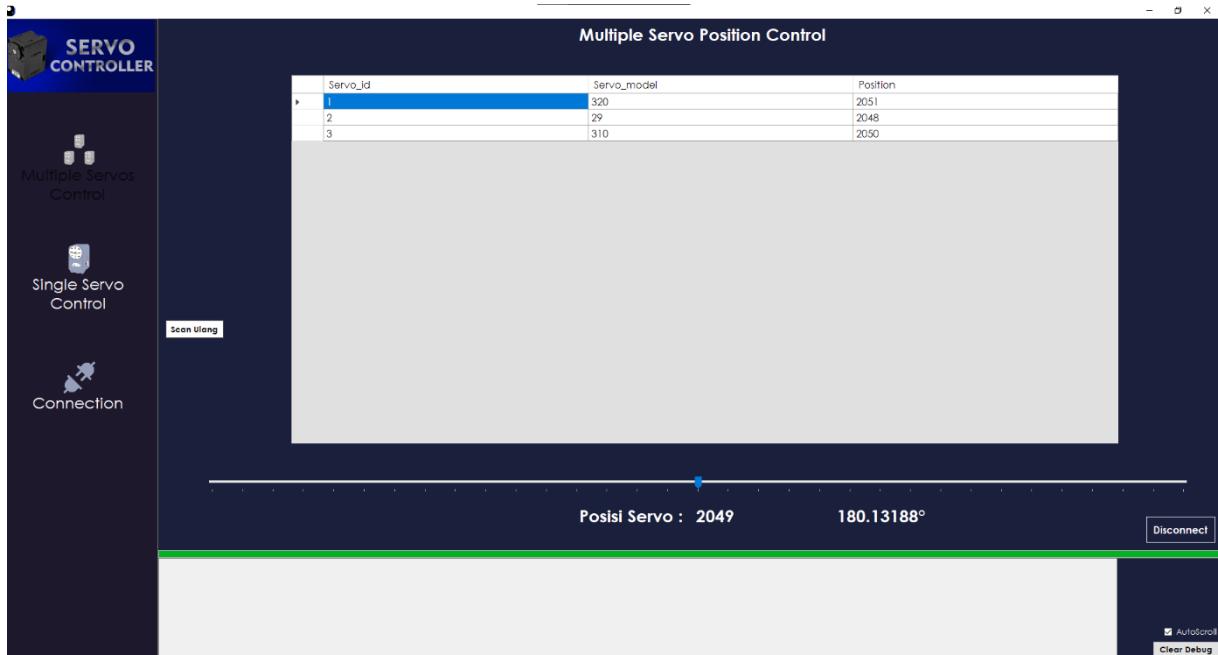


Gambar 3. 20 Tampilan mode *Single Servo Control*

Form terakhir yang ada pada aplikasi ini adalah form “*Multiple Servos Control*” dimana form ini berisi data dari setiap servo yang terkoneksi dengan kontroler dan dapat dilakukan pengkontrolan posisi pada setiap servo yang tersambung secara bersamaan. Bentuk data Serial yang masuk ke dalam aplikasi sama seperti yang tertera di Gambar 3.16. Tampilan form “*Multiple Servos Control*” dapat dilihat pada Gambar 3.21. Dapat terlihat pada gambar tersebut bahwa setiap data posisi yang awalnya dapat memiliki nilai yang berbeda, dengan menggeser *slider* yang ada pada aplikasi, maka nilai posisi setiap servo yang terkoneksi berubah sesuai dengan nilai yang telah dimasukkan pada *slider* sehingga tampilan posisi masing-masing servo dapat berubah seperti yang dapat dilihat pada Gambar 3.22.



Gambar 3. 21 Tampilan mode *Multiple Servo Control*



Gambar 3. 22 Tampilan Multiple Servo Control setelah posisi digeser ke nilai 2049

*(Halaman ini sengaja dikosongkan)*

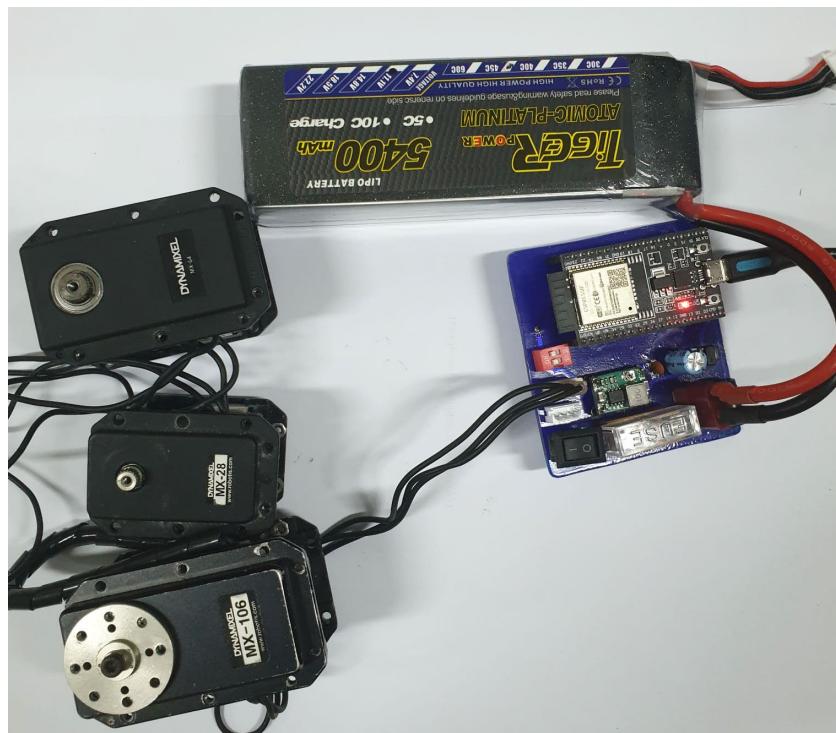
## BAB 4 HASIL DAN PEMBAHASAN

Pada bab 4 ini akan dibahas mengenai hasil pengujian dari kontroler Servo Dynamixel MX-Series. Tujuan dari pengujian ini untuk mengetahui performa dari kontroler apakah sudah sesuai dengan desain yang dibuat dan dapat menjalankan fungsi sebagai kontrol yang mengatur pembacaan data dari servo dan pengkontrolan data pada servo MX-Series dengan setiap variabel yang telah ditentukan dalam program ESP32-Wroom-32 dari sistem yang telah disesuaikan dengan aplikasi yang dibangun, yaitu *Single Servo Control* dan *Multiple Servos Control*. Dalam penelitian kali ini, ada dua buah pengujian yang dilakukan untuk mengetahui keberhasilan penelitian. Pengujian pertama yang dilakukan berupa uji keakuratan dari pembacaan dan pengkontrolan variabel data servo yang telah di tentukan pada mode *Single Servo Control* dan *Multiple Servo Control*, lalu hasil data pengujian dibandingkan dengan data yang didapatkan dengan percobaan menggunakan kontroler acuan yaitu CM-740. Lalu pengujian kedua yang dilakukan berupa uji durasi antara paket instruksi dan paket status.

### 4.1 Hasil Penelitian

#### 4.1.1 Implementasi dari Pengujian Akurasi Pembacaan dan Pengkontrolan Data Servo

Pada pengujian pembacaan dan pengkontrolan data servo, dilakukan pengujian dengan menggunakan aplikasi yang telah dibuat. Aplikasi ini bekerja pada dua mode yaitu mode *Single Servo Control* dan mode *Multiple Servo Control*, dimana variabel yang dipakai dalam pembacaan mode *Single Servo Control* berupa data dari satu buah servo yang telah dipilih dari sejumlah servo yang terkoneksi dengan *PCB* dan variabel yang dipakai dalam pembacaan mode *Multiple Servo Control* hanya berupa posisi. Penggeraan pengujian ini dilakukan dengan menyambungkan tiga buah servo yaitu servo Dynamixel MX-28, MX-64, dan MX-106 dengan menyambungkan kabel



Gambar 4. 1 Susunan pengujian akurasi pembacaan dan pengkontrolan konektor Dynamixel dari satu servo ke servo lainnya dan akhirnya dikoneksikan kepada konektor Dynamixel yang ada pada kontroler, lalu untuk setiap pengujian dilihat hasil dari variabel yang

diambil dibaca masing-masing dari mode aplikasi yang sedang ingin diujikan. Pengujian ini dilakukan seperti dengan gambar 4.1

Dalam pengujian pembacaan data servo, untuk mendapatkan nilai keakuratan data, maka digunakan rumus dibawah ini untuk mendapatkan *Standard Error* dari nilai yang didapat dan dihitung menggunakan rumus.

$$\text{Standard Error} = \frac{\text{Nilai Asli} - \text{Nilai Acuan}}{\text{Range Max Variabel}} * 100\% \quad (4.1)$$

#### 4.1.1.1 Pembacaan dan Pengontrolan mode *Single Servo Control*

Pengujian keakuratan data dilakukan dengan melakukan dua variasi pengujian dimana setiap variasi pengujian dilakukan pada masing-masing servo MX-28, MX-64, dan MX-106. Variasi pertama adalah dengan melakukan 20 (dua puluh) kali pembacaan pada saat poros servo berada di posisi 0 dan variasi ke-dua yaitu melakukan 20 (dua puluh) kali pembacaan pada saat poros di posisi 4095. Variabel pembacaan yang menjadi acuan dalam percobaan ini ada pada Tabel 4.1, bentuk pengujian pada Gambar 4.2, dan hasil dari pengujian data variabel yang terbaca oleh kontroler dapat dilihat pada Tabel 4.2.

Tabel 4. 1 Variabel pembacaan data servo setiap siklus

Nama Variabel	Keterangan
PosN (Posisi saat ini)	Nilai posisi dari sumbu putar aktuator terhadap titik 0
PosG (Posisi Goal)	Nilai posisi yang diinginkan pada sumbu putar aktuator
SpeedN (Kecepatan saat ini)	Kecepatan sumbu putar untuk mencapai posisi yang diperintahkan dalam satuan RAW (Nilai Analog)
SpeedG (Kecepatan Goal)	Kecepatan perputaran yang diinginkan
SpeedReal	Kecepatan sumbu putar untuk mencapai posisi yang diperintahkan dalam satuan RPM ( <i>Rotations per minute</i> )
Temp (Temperatur)	Temperatur yang dibaca oleh servo dalam satuan Celcius
Current	Arus yang dipakai oleh servo (Satuan RAW (nilai analog))
CurrentReal	Arus yang dipakai oleh servo (Satuan milliAmpere)
Status Torsi	Jika 1 maka servo akan mempertahankan posisinya sedangkan 0 maka sumbu servo dapat diputar secara bebas
P Gain	Nilai P Gain dari kontroler PID yang ada dalam servo
I Gain	Nilai I Gain dari kontroler PID yang ada dalam servo
D Gain	Nilai D Gain dari kontroler PID yang ada dalam servo
Return Delay	Durasi yang dibutuhkan servo untuk mengirimkan paket status setelah menerima paket instruksi



Gambar 4. 2 Pengambilan data *Single Servo Control*

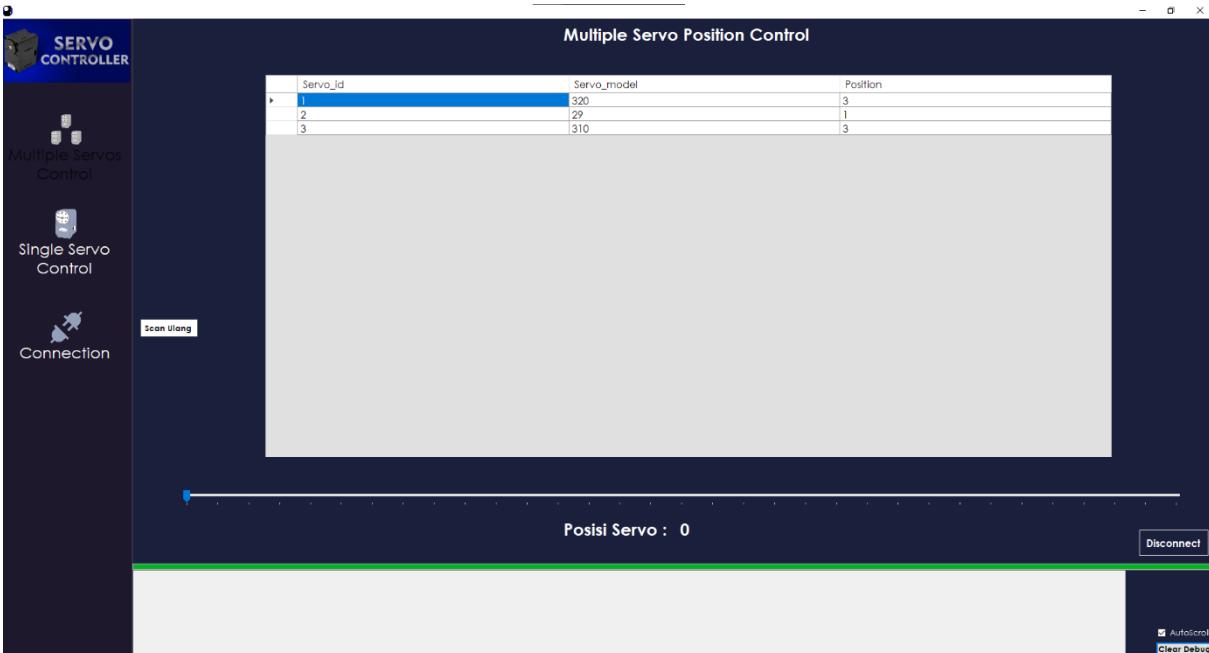
Tabel 4. 2 Hasil Pengujian pembacaan variabel Single Servo Control

	Posisi Acuan ( $^{\circ}$ )	MX-28	MX-64	MX-106
Nilai rata-rata ( $^{\circ}$ )	0 $^{\circ}$	0.16263	0.07912	0.08791
	180 $^{\circ}$	179.91208	180.13186	179.93406
	360 $^{\circ}$	359.86813	359.90769	359.86813

Dari data yang telah didapat pada pengujian *Single Servo Control*, ditemukan nilai *error* seperti yang dapat dilihat pada tabel 4.2. Dari pengujian ini dapat disimpulkan bahwa nilai *error* dari pengujian ini memiliki nilai rata-rata 0.0313% dimana nilai tersebut didapat dari variabel PosN yaitu posisi poros servo pada saat ini. Hasil pengujian kontrol *Single Servo Control* secara detil dapat dilihat pada Lampiran 2.

#### 4.1.1.2 Pembacaan dan Pengontrolan mode *Multiple Servos Control*

Pengujian keakuratan data pada mode *Multiple Servos Control* dilakukan dengan menggunakan variabel posisi yaitu pada saat poros berada di titik 0, titik 2048, dan titik 4095. Pada pengujian ini dilakukan dua variasi pengujian, variasi pertama dilakukan dengan cara menghitung perbandingan nilai posisi yang terbaca pada masing-masing servo dengan nilai posisi acuan, lalu variasi kedua membandingkan nilai rata-rata posisi terbaca setiap servo dengan posisi acuan. Dalam pengujian ini, servo yang digunakan adalah servo MX-28, MX-64, dan MX-106 yang masing-masing berjumlah satu buah dan pembacaan dilakukan sebanyak 20 (dua puluh) kali pada setiap pengambilan data. Bentuk pengujian ini dapat dilihat pada Gambar 4.2, dan hasil pengujian ini dapat dilihat pada tabel 4.3.



Gambar 4. 3 Pengujian *Multiple Servos Control*

Tabel 4. 3 Hasil Pengujian *Multiple Servos Control*

	Posisi Acuan (°)	MX-28	MX-64	MX-106
Nilai rata-rata (°)	0°	0.08791	0.12307	0.11428
	180°	179.87692	179.86813	179.86813
	360°	359.82417	359.91208	359.91208

Dari data yang telah diambil pada pengujian *Multiple Servos Control*, ditemukan error rate pada setiap pengujian kontrol dengan nilai rata-rata *error* 0.0369% dimana pergeseran nilai yang didapatkan berkisar 0-3 dari keseluruhan pengujian titik poros servo. Hasil pengujian kontrol *Multiple Servos Control* secara detil dapat dilihat pada Lampiran 3.

#### 4.1.2 Pengujian Pembacaan dan Pengontrolan Data Servo menggunakan CM-740

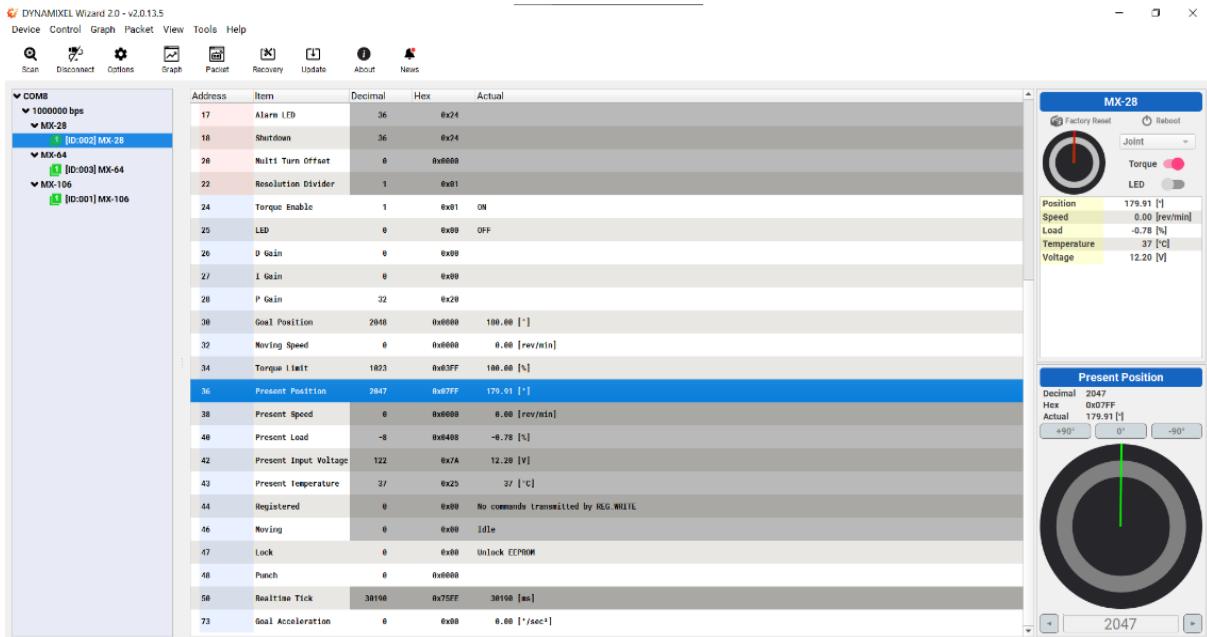
Pengujian ini dilakukan dengan melakukan pembacaan dan pengontrolan setiap variabel servo dari tabel 4.1 pada masing-masing servo MX-28, MX-64, dan MX-106 yang dibaca secara masing-masing dikarenakan untuk pengontrolan dalam aplikasi yang digunakan hanya dapat mengontrol satu buah servo dalam satu waktu. Pengujian ini menggunakan kontroler CM-740 dan aplikasi yang dikeluarkan oleh perusahaan Robotis yaitu *Dynamixel Wizard*. Untuk pengujian ini dilakukan seperti gambar 4.3 dan hasil pengujian ini dapat dilihat pada Tabel 4.4.

Tabel 4. 4 Hasil Pengujian Akurasi CM-740

	Posisi Acuan (°)	MX-28	MX-64	MX-106
Nilai rata-rata (°)	0°	0.11300	0.08789	0.17578
	180°	179.87444	179.87025	179.82003
	360°	359.78655	359.78236	359.72795

Dari pengujian akurasi yang dilakukan pada kontroler acuan yaitu CM-740, didapatkan hasil dengan total rata-rata *error* 0.0386%. Untuk pengujian ini, variabel yang memiliki error pada

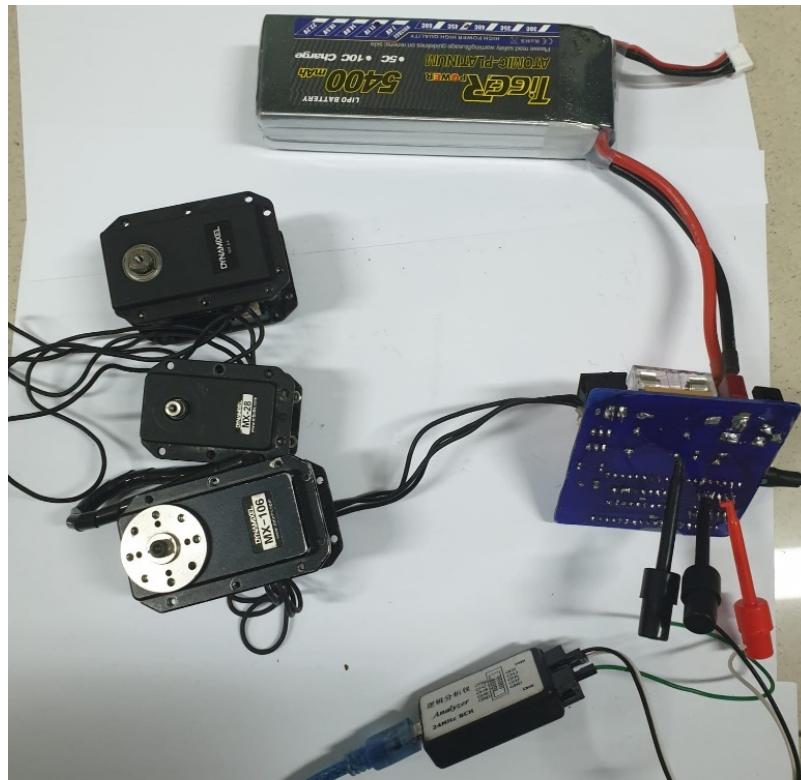
pembacaan juga terjadi pada *Present Position* yaitu sama seperti PosN yang menunjukkan posisi poros servo pada saat ini. Pengujian Akurasi CM-740 secara detil dapat dilihat pada Lampiran 4.



Gambar 4. 4 Pengujian kontroler CM-740

#### 4.1.3 Implementasi Pengujian Durasi Paket Instruksi dengan Paket Status

Pada pengujian durasi ini, dilakukan dengan membandingkan pembacaan nilai variabel *return delay* pada tabel 4.1 dengan nilai asli yang didapatkan dengan pengukuran durasi yang didapatkan pada gelombang yang terbaca oleh *Logic Analyzer* menggunakan aplikasi yang



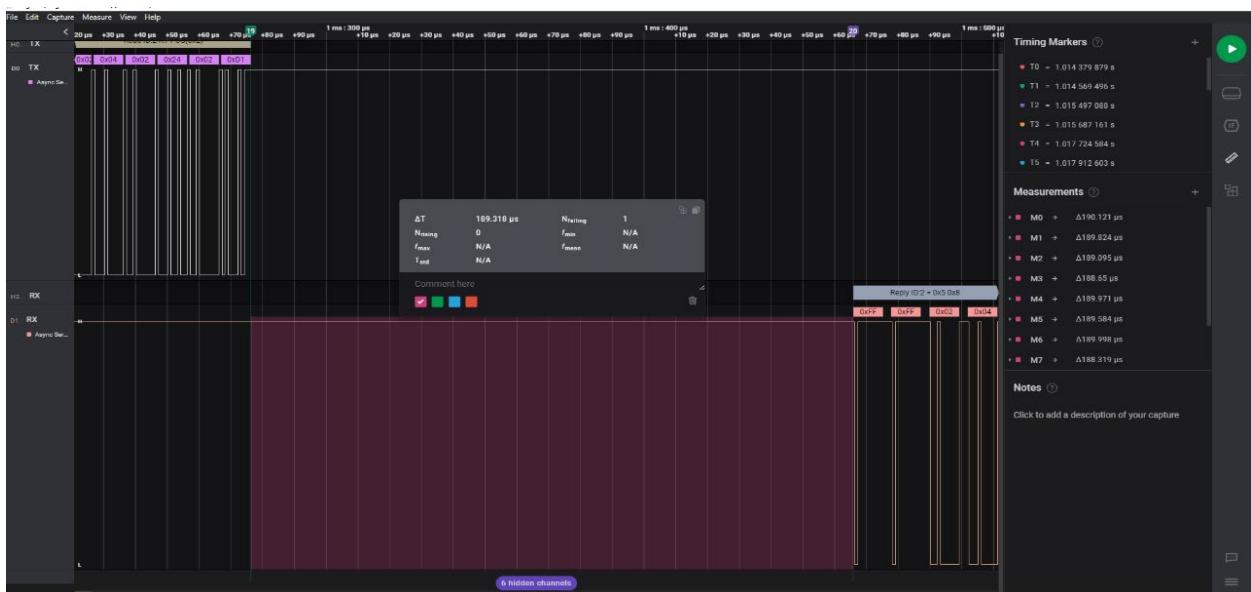
Gambar 4. 5 Rangkaian pengujian *return delay* dengan *logic analyzer*

bernama *Logic* yang dikeluarkan oleh perusahaan Saleae. Pengujian ini dilakukan pada mode *Single Servo Control* dan *Multiple Servos Control* dan variasi *return delay* yang dipakai adalah pada saat nilai durasi asli bernilai 0 $\mu$ S, 100 $\mu$ S, dan 200 $\mu$ S dimana durasi asli didapatkan dengan menggunakan rumus 4.2 dan susunan pengujian pada kontroler seperti gambar 4.5. Hasil pengujian *return delay* secara detil dapat dilihat pada Lampiran 5.

$$\text{Durasi Delay Real} = \text{return delay value} * 2\mu\text{s} \quad (4.2)$$

#### 4.1.3.1 Pembacaan *Return Delay* pada *Single Servo Control*

Pengujian ini dilakukan dengan melakukan pembacaan durasi *return delay* pada setiap variabel yang dibaca dari tabel 4.1 dan membandingkan nilai error tersebut dengan durasi asli yang terbaca oleh *Logic Analyzer*. Bentuk sinyal untuk mode *Single Servo Control* yang terbaca pada *logic analyzer* dapat dilihat pada gambar 4.6, sinyal return delay pada gambar 4.6, dan hasil pengujian ini dapat dilihat pada Tabel 4.5.



Gambar 4. 6 Bentuk Sinyal *Single Servo Control* pada *logic analyzer*

Tabel 4. 5 Hasil Pengukuran durasi return delay Single Servo Control

<i>Return delay acuan</i>	MX-28	MX-64	MX-106	Rata-rata <i>Return Delay</i>	Error rata-rata ( $\mu$ S)
0 $\mu$ S	11.14689	11.53695	12.10304	11.5956	11.5956
100 $\mu$ S	100.0862	105.9319	109.9169	105.3116	5.3116
200 $\mu$ S	189.6215	200.7331	219.3949	203.2498	3.2498

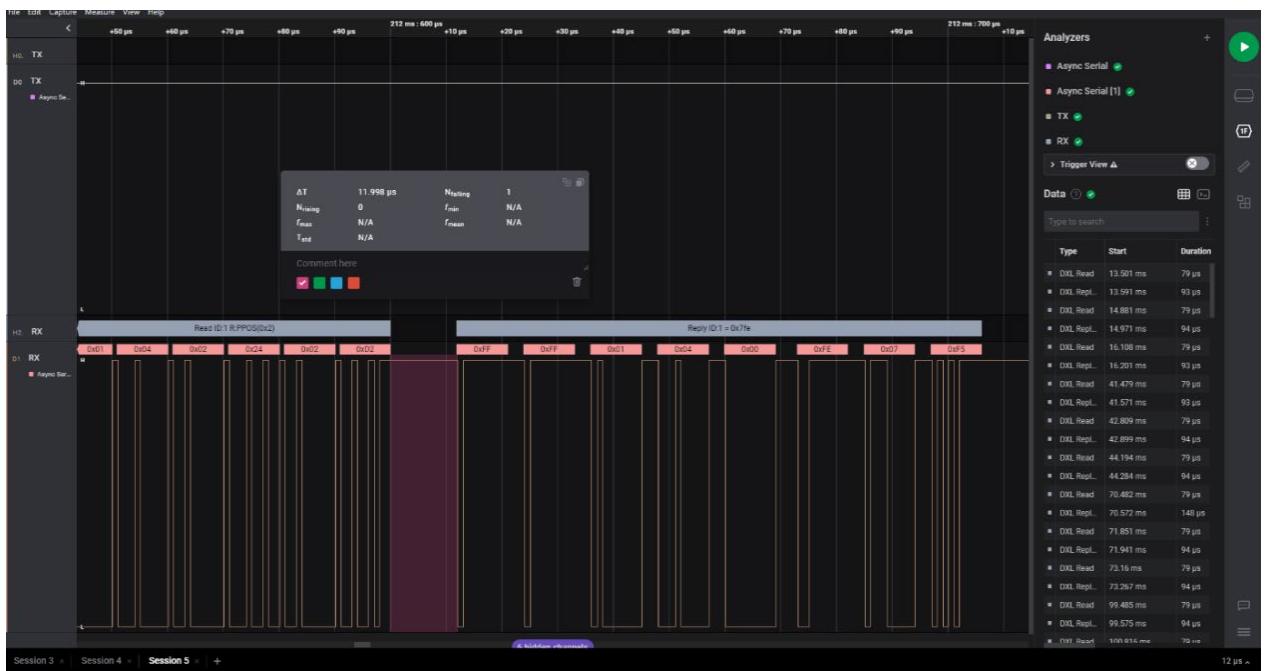
#### 4.1.3.2 Pembacaan *Return Delay* pada *Multiple Servos Control*

Pengujian ini dilakukan dengan pembacaan durasi return delay antar pembacaan posisi setiap servo yang tersambung seperti pada gambar 4.7 dan hasil dapat dilihat pada Tabel 4.6.

Tabel 4. 6 Hasil Pengukuran durasi *return delay* *Multiple Servos Control*

<i>Return delay acuan (<math>\mu</math>S)</i>	MX-28	MX-64	MX-106	Rata-rata <i>Return Delay</i>	Error rata-rata ( $\mu$ S)
0 $\mu$ S	11.751	10.5692	11.5873	11.30253	10.3025
100 $\mu$ S	100.3684	105.6842	110.6842	105.5789	5.5789

200μS	246.1053	199.8947	212	219.3333	19.3333
-------	----------	----------	-----	----------	---------

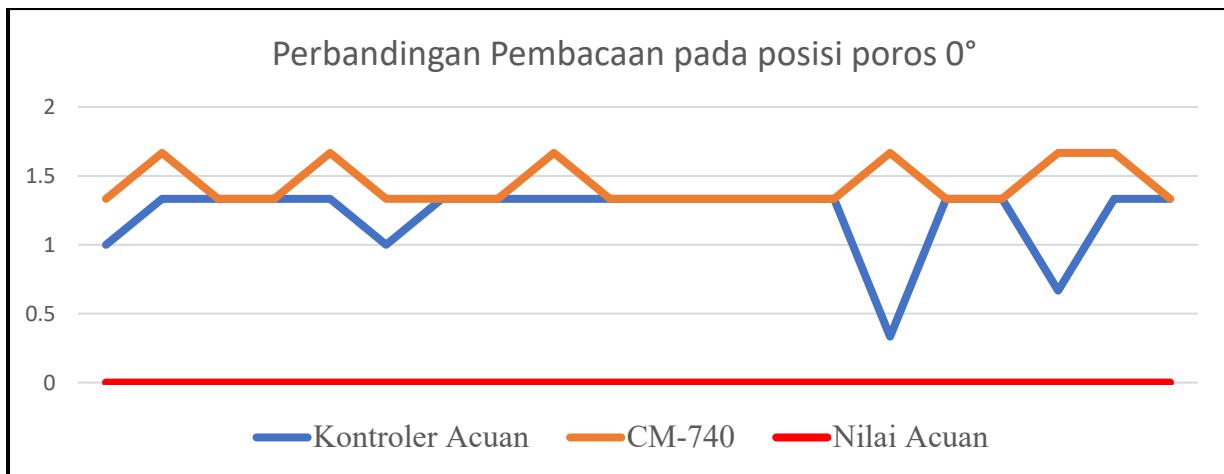


Gambar 4. 7 Nilai *return delay* pada *Multiple Servos Control*

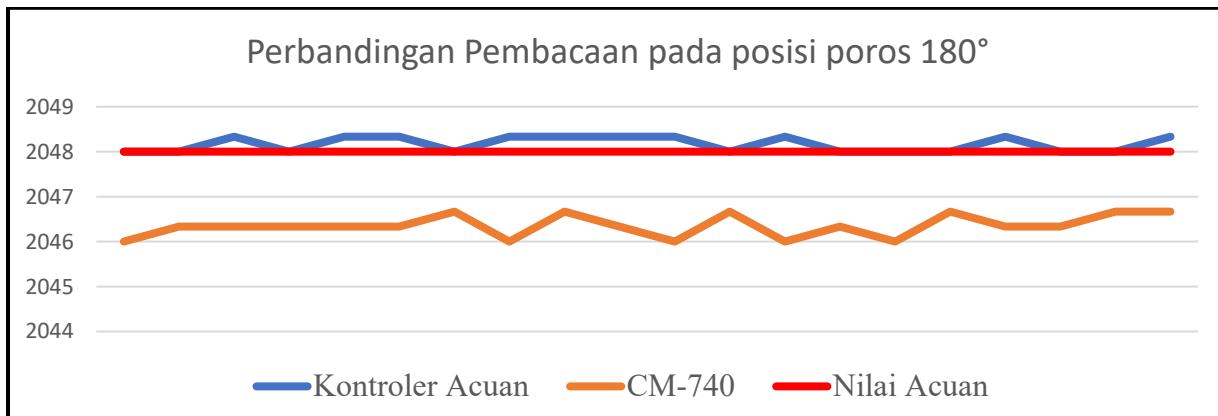
## 4.2 Pembahasan

### 4.2.1 Perbandingan Akurasi data Kontroler buatan dengan Kontroler Acuan

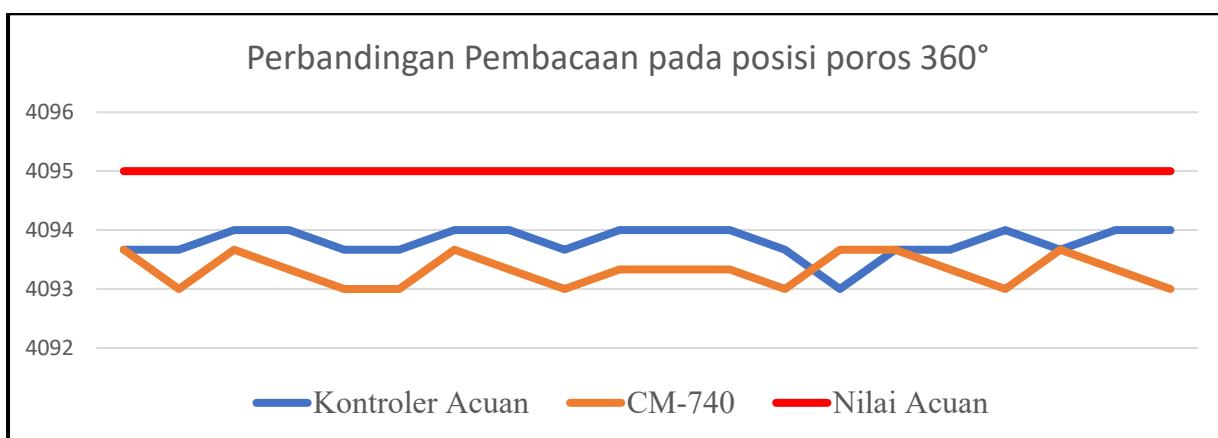
Setelah melakukan pengukuran akurasi pengkontrolan dan pembacaan dari sistem kontroler yang telah dibuat dan juga kontroler CM-740, data dari kedua kontroler dibandingkan seperti pada Tabel 4.5 untuk posisi rata-rata poros yang terbaca dan Tabel 4.6 untuk nilai error rata-rata setiap pembacaan, lalu untuk grafik perbandingan nilai posisi yang terbaca dari kontroler buatan dan CM-740 dapat dilihat dari Gambar 4.8, Gambar 4.9, dan Gambar 4.10.



Gambar 4. 8 Perbandingan saat posisi 0° antara kontroler buatan dengan CM-740



Gambar 4. 9 Perbandingan saat posisi  $180^\circ$  antara kontroler buatan dengan CM-740



Gambar 4. 10 Perbandingan saat posisi  $360^\circ$  antara kontroler buatan dengan CM-740

Tabel 4. 7 Perbandingan pembacaan antara kontroler buatan dengan kontroler CM-740

Posisi Acuan ( $^\circ$ )	MX-28		MX-64		MX-106	
	Kontroler Buatan	CM-740	Kontroler Buatan	CM-740	Kontroler Buatan	CM-740
0°	0.16263	0.113	0.07912	0.08789	0.08791	0.17578
180°	179.912	179.874	180.132	179.87	179.934	179.82
360°	359.868	359.787	359.908	359.782	359.868	359.728

Tabel 4. 8 Perbandingan error pembacaan kontroler buatan dengan kontroler CM-740

Posisi Acuan ( $^\circ$ )	MX-28		MX-64		MX-106	
	Kontroler Buatan	CM-740	Kontroler Buatan	CM-740	Kontroler Buatan	CM-740
0°	0.0451	0.03139	0.0219	0.024414	0.0244	0.048828
180°	0.0366	0.034877	0.0244	0.03604	0.0305	0.049991
360°	0.0366	0.034877	0.0256	0.03604	0.0366	0.051153

Dari kedua tabel diatas, dapat disimpulkan bahwa sistem kontroler yang telah dibuat menghasilkan nilai pembacaan yang akurat dengan nilai yang hampir sama dengan pembacaan dari CM-740 dimana nilai rata-rata *error* keseluruhan pembacaan pada kontroler buatan ada pada nilai 0.0313% sedangkan rata-rata *error* keseluruhan CM-740 pada 0.0386%

## BAB 5 Kesimpulan dan Saran

### 5.1 Kesimpulan

Berdasarkan hasil desain, implementasi dan pengujian dari Implementasi Kontroler Aktuator Dynamixel MX-Series menggunakan Microcontroller ESP32-WROOM-32 dapat ditarik kesimpulan sebagai berikut:

1. Implementasi kontroler servo Dynamixel MX-Series dapat digunakan untuk mengontrol dan membaca data pada servo dengan akurasi yang tinggi yaitu dengan nilai rata-rata pembacaan data yang serupa dengan pembacaan kontroler acuan CM-740 dan kecepatan pengiriman dan pembacaan data serupa dengan nilai aktual.
2. Aplikasi Windows Form yang dibuat dapat digunakan untuk membaca data olahan ESP32-Wroom-32 beserta memberikan input pada kontroler untuk mengontrol dan membaca data yang ada pada servo Dynamixel MX-Series.
3. Sistem kontroler berbasis mikrokontroler ESP32-WROOM-32 beserta dengan aplikasi yang telah dibuat dapat diubah sesuai dengan kebutuhan pengguna sehingga penambahan modul elektronik lainnya dapat dilakukan.

### 5.2 Saran

Dari penelitian ini ada beberapa saran untuk penelitian selanjutnya:

1. Metode *SyncWrite* dapat dilakukan untuk pengontrolan lebih dari satu servo dibandingkan dengan metode pemanggilan ID *Broadcast* jika ingin mengontrol beberapa servo dengan ID yang spesifik.
2. Untuk dapat meminimalisir ukuran rangkaian elektronika kontroler, gunakan *IC* mikrokontroler ESP32-Wroom-32 secara langsung, *IC* 74HC241 yang berbentuk SMD (*Surface Mounted Device*) dan komponen pasif lainnya yang dipakai dalam *schematic* rangkaian *PCB*.

## DAFTAR PUSTAKA

- Bestmann, M., Brandt, H., Engelke, T., Fiedler, N., Gabel, A., Güldenstein, J., Hagge, J., Hartfill, J., Lorenz, T., Heuer, T., Poppinga, M., David, I., Salamanca, R., & Speck, D. (n.d.). *Hamburg Bit-Bots and WF Wolves Team Description for RoboCup 2019-Humanoid KidSize*. <https://www.wf-wolves.de>
- Bestmann, M., Güldenstein, J., & Zhang, J. (n.d.). *High-Frequency Multi Bus Servo and Sensor Communication Using the Dynamixel Protocol*. <http://robocup.informatik.uni-hamburg.de>
- El-Maleh, A. (n.d.). *Asynchronous and synchronous serial communication*. Retrieved July 20, 2022, from <https://slidetodoc.com/asynchronous-and-synchronous-serial-communication-coe-306-introduction/>
- Espressif Systems. (2022a). *ESP32 ESP-IDF Programming Guide Release v5.0-dev-3290-g01d014c42d* Espressif Systems. <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html>
- Espressif Systems. (2022b). *ESP32WROOM32*. [https://espressif.com/sites/default/files/documentation/esp32-wroom-32\\_datasheet\\_en.pdf](https://espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf)
- Firman Choiri, A., Widiawan, B., Teknologi Informasi, J., & Negeri Jember Kampus Politeknik Negeri Jember -Jl Mastrap Box, P. P. (n.d.). *Robotika Penerapan IC 74LS241 Untuk Multi Aktuator Dynamixel AX-12A Pada Biped Robot*.
- MacDonald, M. (2002). *User Interfaces in C#: Windows Forms and Custom Controls*. Apress. <https://doi.org/10.1007/978-1-4302-0837-2>
- Meca Santamaria, J. (2021). *Final Degree Project Biomedical Engineering Degree Da Vinci robot at Hospital Clinic. Manoeuvrability devices and performance in robotic tech*.
- Microsoft. (2022). *Desktop Guide (Windows Forms .NET)*. <https://docs.microsoft.com/en-us/dotnet/desktop/winforms/overview/?view=netdesktop-6.0>
- Monolithic Power Systems. (2008). *3A, 23V, 340KHz Synchronous Rectified Step-Down Converter The Future of Analog IC Technology DESCRIPTION*. [www.MonolithicPower.com](http://www.MonolithicPower.com)
- PlatformIO. (2022). *What is PlatformIO?* <https://docs.platformio.org/en/latest/what-is-platformio.html#what-is-platformio>
- Qiang Liu, Peien Feng, & Shuangxia Pan. (2006). Robust Motion Control of High Precision Mechanical Servo Systems with Parameter Uncertainties. *2006 6th World Congress on Intelligent Control and Automation*, 8054–8058. <https://doi.org/10.1109/WCICA.2006.1713542>
- Rhoban. (2018, November 26). *Rhoban/DXLBoard: Rhoban communication board, featuring 9DOF IMU and 3 dynamixel buses*. GitHub. <https://github.com/Rhoban/DXLBoard>
- Robotis. (n.d.-a). *Dynamixel MX-Series Picture*. Retrieved July 20, 2022, from [https://emanual.robotis.com/assets/images/dxl/mx/mx-106t\\_product.jpg](https://emanual.robotis.com/assets/images/dxl/mx/mx-106t_product.jpg)

- Robotis. (n.d.-b). *USB2DYNAMIXEL*. Retrieved July 20, 2022, from [https://emanual.robotis.com/assets/images/parts/interface/usb2dynamixel\\_07.jpg](https://emanual.robotis.com/assets/images/parts/interface/usb2dynamixel_07.jpg)
- Robotis. (2020, April 30). *DYNAMIXEL -T & -R?* <https://www.robotis.us/robotis-blog/dynamixel-t-r-/>
- Robotis. (2022a). *Datasheet Dynamixel MX-106.* <https://emanual.robotis.com/docs/en/dxl/mx/mx-106/#specifications>
- Robotis. (2022b). *Dynamixel SDK.* [https://emanual.robotis.com/docs/en/software/dynamixel/dynamixel\\_sdk/overview/](https://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_sdk/overview/)
- ROBOTIS. (2022a). *Dynamixel Servos.* <https://emanual.robotis.com/docs/en/dxl/>
- ROBOTIS. (2022b, May 10). *DYNAMIXEL protocol library for Arduino.* <https://github.com/ROBOTIS-GIT/Dynamixel2Arduino>
- Wikipedia. (n.d.). *RS-485*. Retrieved July 20, 2022, from <https://en.wikipedia.org/wiki/RS-485>
- Xin Kang, Wenda Shen, Weihai Chen, & Jianhua Wang. (2009). The control of Dynamixel RX-28 based on VC++ for the locomotion of cockroach robot. *2009 4th IEEE Conference on Industrial Electronics and Applications*, 495–499. <https://doi.org/10.1109/ICIEA.2009.5138255>

## LAMPIRAN 1 Kode Program ESP32

```
#include <Dynamixel2Arduino.h> //Library untuk interface arduino dengan #include <List.hpp>
//Library untuk membuat list pada Arduino API  #include "ESP32SerialPortHandler.cpp"
#define DEBUG_SERIAL Serial                                #define DXL_SERIAL Serial1
#define MAX_BAUD 3
TaskHandle_t Task1;
TaskHandle_t Task2;
List<int> nilai;
const uint8_t DXL_DIR_PIN = 4; // Pin output dari ESP untuk kontrol DXL
const uint8_t DXL_RX_PIN = 16; // RX Pin Serial1
const uint8_t DXL_TX_PIN = 17; // TX Pin Serial1
String Input_str;
int id, Model, id_new;
uint16_t PosN, PosG, SpeedN, SpeedG, Current, nilai_Pos;
bool Torque, list_kirim, ModeKirim, ok;
const float protocol = 1.0;
uint32_t Sbaud, Sbaud_new, nilaiBaud, SpeedReal;
uint8_t nilaiP, nilaiI, nilaiD, p_gain, i_gain, d_gain, ret_delay, temp, nilaiRet, CurrentReal;
uint32_t baud[MAX_BAUD] = {57600, 115200, 1000000};
ESP32SerialPortHandler esp_dxl_port(Serial1, DXL_RX_PIN, DXL_TX_PIN, DXL_DIR_PIN);
Dynamixel2Arduino dxl;
using namespace ControlTableItem;
uint8_t op_mode = OP_POSITION;
void scan_dxl() {
    nilai.removeAll();
    int8_t index = 0;
    int8_t found_dynamixel = 0
    dxl.setPortProtocolVersion(protocol);
    for(index = 0; index < 3; index++) {
        dxl.begin(baud[index]);
        for(id = 0; id < DXL_BROADCAST_ID; id++) {
            if(dxl.ping(id)) {
                Model = dxl.getModelNumber(id);
                nilai.add(id);
                nilai.add(Model);}}}
void List_send(){
    int *test = nilai.toArray();
    delay(100);
    for(int i = 0; i < nilai.getSize(); i += 2){
        Serial.println((String)"Servo" + nilai[i] + "SModel" + nilai[i+1]);
        delay(100); }
        delay(1000);
        Serial.println((String)"Servo999SModel999");
        free(test);
        list_kirim = true;}
```

```

void Pchange(){
    if(op_mode != OP_POSITION){
        dxl.torqueOff(id);
        dxl.setOperatingMode((uint8_t)id, OP_POSITION);
        dxl.torqueOn(id); }
        dxl.writeControlTableItem(P_GAIN, id, nilaiP);
        op_mode = OP_POSITION;}
void Ichange(){
    if(op_mode != OP_POSITION){
        dxl.torqueOff(id);
        dxl.setOperatingMode((uint8_t)id, OP_POSITION);
        dxl.torqueOn(id); }
        dxl.writeControlTableItem(I_GAIN, id, nilaiI);
        op_mode = OP_POSITION;}
void Dchange(){
    if(op_mode != OP_POSITION){
        dxl.torqueOff(id);
        dxl.setOperatingMode((uint8_t)id, OP_POSITION);
        dxl.torqueOn(id); }
        dxl.writeControlTableItem(D_GAIN, id, nilaiD);
        op_mode = OP_POSITION;}
void ret_change(){
    dxl.torqueOff(id);
    dxl.writeControlTableItem(RETURN_DELAY_TIME, (uint8_t)id, ret_delay);
    dxl.torqueOn(id);}
void Serial_Flush(){
    while(!Serial.available()) {Serial.flush();}}
void set_pos(){
    if(op_mode != OP_POSITION){
        op_mode = OP_POSITION;
        dxl.torqueOff(id);
        dxl.setOperatingMode((uint8_t)id, OP_POSITION);
        dxl.torqueOn(id);}
    if(ModeKirim == false){
        dxl.writeControlTableItem(GOAL_VELOCITY, id, 50);
        dxl.setGoalPosition(id, nilai_Pos);
        PosN = dxl.getPresentPosition(id);}
    else{
        dxl.writeControlTableItem(GOAL_VELOCITY, 254, 50);
        dxl.setGoalPosition(254, nilai_Pos);}}
void Serial_flush(){
    if(Serial.read() >= 0){Serial.flush();}}
void set_Baud(){
    if(dxl.ping(id) == true){
        dxl.torqueOff(id);
        dxl.setBaudrate(Sbaud, Sbaud_new);
        dxl.setOperatingMode((uint8_t)id, OP_POSITION);
        dxl.torqueOn(id);
        delay(100);
        Sbaud = Sbaud_new;}}

```

```

void set_ID(){
    if(dxl.ping(id) == true){
        dxl.torqueOff(id);
        dxl.setID(id, id_new);
        dxl.torqueOn(id_new);
        delay(100);
        for(int i =0; i<nilai.getSize();i+=2){
            if(nilai[i] == id){
                nilai.remove(i);
                nilai.addAtIndex(i,id_new);}
            id = id_new;  }}
void Core0( void * pvParameters ){
for(;;){
    delay(25);
    if(ModeKirim == false){
        PosN = dxl.getPresentPosition(id);
        SpeedN = dxl.getPresentVelocity(id);
        SpeedReal = dxl.getPresentVelocity(id, UNIT_RPM);
        Torque = dxl.getTorqueEnableStat(id);
        PosG = dxl.readControlTableItem(GOAL_POSITION, id);
        SpeedG = dxl.readControlTableItem(GOAL_VELOCITY, id);
        p_gain = dxl.readControlTableItem(P_GAIN, id);
        i_gain = dxl.readControlTableItem(I_GAIN, id);
        d_gain = dxl.readControlTableItem(D_GAIN, id);
        ret_delay = dxl.readControlTableItem(RETURN_DELAY_TIME, id)* 2;
        temp = dxl.readControlTableItem(PRESENT_TEMPERATURE, id);
        Current = dxl.getPresentCurrent(id);
        CurrentReal = dxl.getPresentCurrent(id, UNIT_MILLI_AMPERE);    }
    else{
        for(int i =0;i<nilai.getSize();i+=2){
            id = nilai[i];
            Model = nilai[i+1];
            PosN = dxl.getPresentPosition(id);
            Serial.println((String)"ID" + id + "Model" + Model + "Pos" + PosN);
            }}}}
void Core1( void * pvParameters ){
for(;;{
    delay(100);
    if(list_kirim == false){
        List_send();
        list_kirim = true;
        if(ModeKirim == false){
            Serial_Flush();}
        else{if(ModeKirim == false){
            Serial.println((String)"ID" + id + "Baud" + Sbaud + "Model" + Model + "Prot" + protocol + "PosN"
+ PosN + "PosG" + PosG + "SpeedN" + SpeedN + "SpeedG" + SpeedG + "Current" + Current + "Torque"
+ Torque + "pgain"+ p_gain + "igain" + i_gain + "dgain" + d_gain + "retdelay" + ret_delay + "temp" +
temp + "speedReal" + SpeedReal+ "RealCurr" + CurrentReal); }}}

```

```

while(Serial.available()>0){
    Input_str = Serial.readString();
    int8_t indexganti = Input_str.indexOf("ganti");
    int8_t indexID = Input_str.indexOf("ID");
    int8_t indexBaud = Input_str.indexOf("Baud");
    int8_t indexPos = Input_str.indexOf("Pos");
    int8_t indexP = Input_str.indexOf("PG");
    int8_t indexI = Input_str.indexOf("IG");
    int8_t indexD = Input_str.indexOf("DG");
    int8_t indexRet = Input_str.indexOf("Ret");
    int8_t indexMode = Input_str.indexOf("mode");
    int8_t indexreset = Input_str.indexOf("reset");
    if(indexreset >= 0){scan_dxl();}
    if(indexganti >=0){
        uint8_t id_ganti = Input_str.substring(indexganti+5).toInt();
        if(ModeKirim == false){
            id = id_ganti;
            for(int i=0;i<nilai.getSize();i+=2){
                if(id_ganti == nilai[i]){
                    Model = nilai[i+1]; } }
            else{
                ModeKirim = false;delay(1000);}}
        if(indexID >= 0){
            id_new = Input_str.substring((indexID+2), indexBaud).toInt();
            if(id_new != id){set_ID();}
        if(indexBaud >= 0){
            Sbaud_new = Input_str.substring(indexBaud+4).toInt();
            if(Sbaud_new != Sbaud){set_Baud();}
        if(indexPos >=0){
            nilai_Pos = Input_str.substring(indexPos+3).toInt();
            set_pos();}
        if(indexP >=0){
            nilaiP = Input_str.substring(indexPos+2).toInt();
            if(nilaiP != p_gain){Pchange();}
        if(indexI >=0){
            nilaiI = Input_str.substring(indexPos+2).toInt();
            if(nilaiI != i_gain){ Ichange();}
        if(indexD >=0){
            nilaiD = Input_str.substring(indexPos+2).toInt();
            if(nilaiD != d_gain){Dchange();}
        if(indexRet >=0){
            nilaiRet = Input_str.substring(indexPos+3).toInt();
            if(nilaiRet != ret_delay){ret_change();}
        if(indexMode >=0){
            ModeKirim = Input_str.substring(indexMode+4).toInt();
            list_kirim = false;
            if(ModeKirim == false){}
            delay(2000);}
            Serial_flush();}}}

```

```

void setup() {
    int8_t index = 0;
    int8_t found_dynamixel = 0;
    dxl.setPort(esp_dxl_port);
    Serial.begin(500000);
    dxl.begin(1000000);
    dxl.setPortProtocolVersion(protocol);
    for(index = 0; index < MAX_BAUD; index++) {
        dxl.begin(baud[index]); //Mulai Scan Dynamixel dengan Baudrate sesuai dengan nilai index
        for(uint8_t id_scan = 0; id_scan < DXL_BROADCAST_ID; id_scan++) {
            if(dxl.ping(id_scan)) {
                Sbaud = baud[index];
                id = id_scan;
                nilai.add(id);
                nilai.add(Model);
                dxl.torqueOff(id_scan);
                dxl.writeControlTableItem(DRIVE_MODE,id_scan,4);
                dxl.writeControlTableItem(RETURN_DELAY_TIME,id_scan,10);
                dxl.writeControlTableItem(PROFILE_VELOCITY, id_scan, 0);
                dxl.setOperatingMode(id, OP_POSITION);
                dxl.torqueOn(id_scan);
                dxl.writeControlTableItem(P_GAIN,id_scan,32);
                dxl.writeControlTableItem(I_GAIN,id_scan,0);
                dxl.writeControlTableItem(D_GAIN,id_scan,0);
                dxl.writeControlTableItem(PROFILE_ACCELERATION, id_scan, 50);
                dxl.writeControlTableItem(PROFILE_VELOCITY, id_scan, 300); } } }
    ModeKirim = true;
    xTaskCreatePinnedToCore(
        Core0, /* Task function. */
        "Task1", /* name of task. */
        60000, /* Stack size of task */
        NULL, /* parameter of the task */
        2, /* priority of the task */
        &Task1, /* Task handle to keep track of created task */
        0); /* pin task to core 0 */
    delay(500);
    xTaskCreatePinnedToCore(
        Core1, /* Task function. */
        "Task2", /* name of task. */
        100000, /* Stack size of task */
        NULL, /* parameter of the task */
        1, /* priority of the task */
        &Task2, /* Task handle to keep track of created task */
        1); /* pin task to core 1 */
    delay(500);
}

```

## LAMPIRAN 2 Hasil Pengujian Single Servo Control

Servo MX-28, Posisi Poros berada pada Titik 0

No.	PosN	PosG	SpeedN	SpeedG	Current	Status Torsi	P Gain	I Gain	D Gain	Return Delay	Temp	Speed Real	Current Real
1	1	0	0	0	0	1	32	0	0	20	40	0	0
2	2	0	0	0	0	1	32	0	0	20	40	0	0
3	2	0	0	0	0	1	32	0	0	20	40	0	0
4	2	0	0	0	0	1	32	0	0	20	40	0	0
5	2	0	0	0	0	1	32	0	0	20	40	0	0
6	1	0	0	0	0	1	32	0	0	20	40	0	0
7	2	0	0	0	0	1	32	0	0	20	40	0	0
8	2	0	0	0	0	1	32	0	0	20	40	0	0
9	2	0	0	0	0	1	32	0	0	20	40	0	0
10	2	0	0	0	0	1	32	0	0	20	40	0	0
11	2	0	0	0	0	1	32	0	0	20	40	0	0
12	2	0	0	0	0	1	32	0	0	20	40	0	0
13	2	0	0	0	0	1	32	0	0	20	40	0	0
14	2	0	0	0	0	1	32	0	0	20	40	0	0
15	1	0	0	0	0	1	32	0	0	20	43	0	0
16	2	0	0	0	0	1	32	0	0	20	40	0	0
17	2	0	0	0	0	1	32	0	0	20	40	0	0
18	2	0	0	0	0	1	32	0	0	20	40	0	0
19	2	0	0	0	0	1	32	0	0	20	40	0	0
20	2	0	0	0	0	1	32	0	0	20	40	0	0
Error (%)	0.0452	0	0	0	0	0	0	0	0	0	0	0	0

Servo MX-28, Posisi Poros berada pada Titik 2048

No.	PosN	PosG	SpeedN	SpeedG	Current	Status Torsi	P Gain	I Gain	D Gain	Return Delay	Temp	Speed Real	Current Real
1	2046	2048	0	0	2048	1	32	0	0	20	39	0	0
2	2046	2048	0	0	2048	1	32	0	0	20	39	0	0
3	2047	2048	0	0	2048	1	32	0	0	20	39	0	0
4	2046	2048	0	0	2048	1	32	0	0	20	39	0	0
5	2047	2048	0	0	2048	1	32	0	0	20	39	0	0
6	2047	2048	0	0	2048	1	32	0	0	20	39	0	0
7	2046	2048	0	0	2048	1	32	0	0	20	39	0	0
8	2047	2048	0	0	2048	1	32	0	0	20	39	0	0
9	2047	2048	0	0	2048	1	32	0	0	20	39	0	0
10	2047	2048	0	0	2048	1	32	0	0	20	39	0	0
11	2047	2048	0	0	2048	1	32	0	0	20	39	0	0
12	2046	2048	0	0	2048	1	32	0	0	20	39	0	0
13	2047	2048	0	0	2048	1	32	0	0	20	39	0	0
14	2046	2048	0	0	2048	1	32	0	0	20	39	0	0
15	2046	2048	0	0	2048	1	32	0	0	20	39	0	0
16	2046	2048	0	0	2048	1	32	0	0	20	39	0	0
17	2047	2048	0	0	2048	1	32	0	0	20	39	0	0

18	2046	2048	0	0	2048	1	32	0	0	20	39	0	0
19	2046	2048	0	0	2048	1	32	0	0	20	39	0	0
20	2047	2048	0	0	2048	1	32	0	0	20	39	0	0
Error (%)	0.0366	0	0	0	0	0	0	0	0	0	0	0	0

#### Servo MX-28, Posisi Poros berada pada titik 4095

No.	PosN	PosG	SpeedN	SpeedG	Current	Status Torsi	P Gain	I Gain	D Gain	Return Delay	Temp	Speed Real	Current Real
1	4093	4095	0	0	2048	1	32	0	0	20	39	0	0
2	4093	4095	0	0	2048	1	32	0	0	20	39	0	0
3	4094	4095	0	0	2048	1	32	0	0	20	39	0	0
4	4094	4095	0	0	2048	1	32	0	0	20	39	0	0
5	4093	4095	0	0	2048	1	32	0	0	20	39	0	0
6	4093	4095	0	0	2048	1	32	0	0	20	39	0	0
7	4094	4095	0	0	2048	1	32	0	0	20	39	0	0
8	4094	4095	0	0	2048	1	32	0	0	20	39	0	0
9	4093	4095	0	0	2048	1	32	0	0	20	39	0	0
10	4094	4095	0	0	2048	1	32	0	0	20	39	0	0
11	4094	4095	0	0	2048	1	32	0	0	20	39	0	0
12	4094	4095	0	0	2048	1	32	0	0	20	39	0	0
13	4093	4095	0	0	2048	1	32	0	0	20	39	0	0
14	4093	4095	0	0	2048	1	32	0	0	20	39	0	0
15	4093	4095	0	0	2048	1	32	0	0	20	39	0	0
16	4093	4095	0	0	2048	1	32	0	0	20	39	0	0
17	4094	4095	0	0	2048	1	32	0	0	20	39	0	0
18	4093	4095	0	0	2048	1	32	0	0	20	39	0	0
19	4094	4095	0	0	2048	1	32	0	0	20	39	0	0
20	4094	4095	0	0	2048	1	32	0	0	20	39	0	0
Error (%)	0.0366	0	0	0	0	0	0	0	0	0	0	0	0

#### Servo MX-64, Posisi Poros pada titik 0

No.	PosN	PosG	SpeedN	SpeedG	Current	Status Torsi	P Gain	I Gain	D Gain	Return Delay	Temp	Speed Real	Current Real
1	1	0	0	0	2048	1	32	0	0	20	39	0	0
2	1	0	0	0	2048	1	32	0	0	20	39	0	0
3	1	0	0	0	2048	1	32	0	0	20	39	0	0
4	1	0	0	0	2048	1	32	0	0	20	39	0	0
5	1	0	0	0	2048	1	32	0	0	20	39	0	0
6	1	0	0	0	2048	1	32	0	0	20	39	0	0
7	1	0	0	0	2048	1	32	0	0	20	39	0	0
8	1	0	0	0	2048	1	32	0	0	20	39	0	0
9	1	0	0	0	2048	1	32	0	0	20	39	0	0
10	1	0	0	0	2048	1	32	0	0	20	39	0	0
11	1	0	0	0	2048	1	32	0	0	20	39	0	0
12	1	0	0	0	2048	1	32	0	0	20	39	0	0
13	1	0	0	0	2048	1	32	0	0	20	39	0	0
14	1	0	0	0	2048	1	32	0	0	20	39	0	0

15	0	0	0	0	2048	1	32	0	0	20	39	0	0
16	1	0	0	0	2048	1	32	0	0	20	39	0	0
17	1	0	0	0	2048	1	32	0	0	20	39	0	0
18	0	0	0	0	2048	1	32	0	0	20	39	0	0
19	1	0	0	0	2048	1	32	0	0	20	39	0	0
20	1	0	0	0	2048	1	32	0	0	20	39	0	0
Error (%)	0.0219	0	0	0	0	0	0	0	0	0	0	0	0

Servo MX-64, Posisi Poros pada titik 2048

No.	PosN	PosG	SpeedN	SpeedG	Current	Status Torsi	P Gain	I Gain	D Gain	Return Delay	Temp	Speed Real	Current Real
1	2049	2048	0	0	2048	1	32	0	0	20	39	0	0
2	2049	2048	0	0	2048	1	32	0	0	20	39	0	0
3	2049	2048	0	0	2048	1	32	0	0	20	39	0	0
4	2049	2048	0	0	2048	1	32	0	0	20	39	0	0
5	2049	2048	0	0	2048	1	32	0	0	20	39	0	0
6	2049	2048	0	0	2048	1	32	0	0	20	39	0	0
7	2049	2048	0	0	2048	1	32	0	0	20	39	0	0
8	2049	2048	0	0	2048	1	32	0	0	20	39	0	0
9	2049	2048	0	0	2048	1	32	0	0	20	39	0	0
10	2049	2048	0	0	2048	1	32	0	0	20	39	0	0
11	2049	2048	0	0	2048	1	32	0	0	20	39	0	0
12	2049	2048	0	0	2048	1	32	0	0	20	39	0	0
13	2049	2048	0	0	2048	1	32	0	0	20	39	0	0
14	2049	2048	0	0	2048	1	32	0	0	20	39	0	0
15	2049	2048	0	0	2048	1	32	0	0	20	39	0	0
16	2049	2048	0	0	2048	1	32	0	0	20	39	0	0
17	2049	2048	0	0	2048	1	32	0	0	20	39	0	0
18	2049	2048	0	0	2048	1	32	0	0	20	39	0	0
19	2049	2048	0	0	2048	1	32	0	0	20	39	0	0
20	2049	2048	0	0	2048	1	32	0	0	20	39	0	0
Error (%)	0.0244	0	0	0	0	0	0	0	0	0	0	0	0

Servo MX-64, Posisi Poros pada titik 4095

No.	PosN	PosG	SpeedN	SpeedG	Current	Status Torsi	P Gain	I Gain	D Gain	Return Delay	Temp	Speed Real	Current Real
1	4094	4095	0	0	2048	1	32	0	0	20	39	0	0
2	4094	4095	0	0	2048	1	32	0	0	20	39	0	0
3	4094	4095	0	0	2048	1	32	0	0	20	39	0	0
4	4094	4095	0	0	2048	1	32	0	0	20	39	0	0
5	4094	4095	0	0	2048	1	32	0	0	20	39	0	0
6	4094	4095	0	0	2048	1	32	0	0	20	39	0	0
7	4094	4095	0	0	2048	1	32	0	0	20	39	0	0
8	4094	4095	0	0	2048	1	32	0	0	20	39	0	0
9	4094	4095	0	0	2048	1	32	0	0	20	39	0	0
10	4094	4095	0	0	2048	1	32	0	0	20	39	0	0
11	4094	4095	0	0	2048	1	32	0	0	20	39	0	0

12	4094	4095	0	0	2048	1	32	0	0	20	39	0	0
13	4094	4095	0	0	2048	1	32	0	0	20	39	0	0
14	4093	4095	0	0	2048	1	32	0	0	20	39	0	0
15	4094	4095	0	0	2048	1	32	0	0	20	39	0	0
16	4094	4095	0	0	2048	1	32	0	0	20	39	0	0
17	4094	4095	0	0	2048	1	32	0	0	20	39	0	0
18	4094	4095	0	0	2048	1	32	0	0	20	39	0	0
19	4094	4095	0	0	2048	1	32	0	0	20	39	0	0
20	4094	4095	0	0	2048	1	32	0	0	20	39	0	0
Error (%)	0.0256	0	0	0	0	0	0	0	0	0	0	0	0

Servo MX-106, Posisi poros pada titik 0

No.	PosN	PosG	SpeedN	SpeedG	Current	Status Torsi	P Gain	I Gain	D Gain	Return Delay	Temp	Speed Real	Current Real
1	1	0	0	0	0	1	32	0	0	20	36	0	0
2	1	0	0	0	0	1	32	0	0	20	36	0	0
3	1	0	0	0	0	1	32	0	0	20	36	0	0
4	1	0	0	0	0	1	32	0	0	20	36	0	0
5	1	0	0	0	0	1	32	0	0	20	36	0	0
6	1	0	0	0	0	1	32	0	0	20	36	0	0
7	1	0	0	0	0	1	32	0	0	20	36	0	0
8	1	0	0	0	0	1	32	0	0	20	36	0	0
9	1	0	0	0	0	1	32	0	0	20	36	0	0
10	1	0	0	0	0	1	32	0	0	20	36	0	0
11	1	0	0	0	0	1	32	0	0	20	36	0	0
12	1	0	0	0	0	1	32	0	0	20	36	0	0
13	1	0	0	0	0	1	32	0	0	20	36	0	0
14	1	0	0	0	0	1	32	0	0	20	36	0	0
15	1	0	0	0	0	1	32	0	0	20	36	0	0
16	1	0	0	0	0	1	32	0	0	20	36	0	0
17	1	0	0	0	0	1	32	0	0	20	36	0	0
18	1	0	0	0	0	1	32	0	0	20	36	0	0
19	1	0	0	0	0	1	32	0	0	20	36	0	0
20	1	0	0	0	0	1	32	0	0	20	36	0	0
Error (%)	0.0244	0	0	0	0	0	0	0	0	0	0	0	0

Servo MX-106, Posisi poros pada titik 2048

No.	PosN	PosG	SpeedN	SpeedG	Current	Status Torsi	P Gain	I Gain	D Gain	Return Delay	Temp	Speed Real	Current Real
1	2047	2048	0	0	0	0	1	32	0	0	20	36	0
2	2046	2048	0	0	0	0	1	32	0	0	20	36	0
3	2047	2048	0	0	0	0	1	32	0	0	20	36	0
4	2046	2048	0	0	0	0	1	32	0	0	20	36	0
5	2047	2048	0	0	0	0	1	32	0	0	20	36	0
6	2047	2048	0	0	0	0	1	32	0	0	20	36	0
7	2047	2048	0	0	0	0	1	32	0	0	20	36	0
8	2047	2048	0	0	0	0	1	32	0	0	20	36	0

9	2047	2048	0	0	0	0	1	32	0	0	20	36	0
10	2047	2048	0	0	0	0	1	32	0	0	20	36	0
11	2046	2048	0	0	0	0	1	32	0	0	20	36	0
12	2047	2048	0	0	0	0	1	32	0	0	20	36	0
13	2047	2048	0	0	0	0	1	32	0	0	20	36	0
14	2047	2048	0	0	0	0	1	32	0	0	20	36	0
15	2047	2048	0	0	0	0	1	32	0	0	20	36	0
16	2047	2048	0	0	0	0	1	32	0	0	20	36	0
17	2047	2048	0	0	0	0	1	32	0	0	20	36	0
18	2046	2048	0	0	0	0	1	32	0	0	20	36	0
19	2046	2048	0	0	0	0	1	32	0	0	20	36	0
20	2047	2048	0	0	0	0	1	32	0	0	20	36	0
Error (%)	0.0305	0	0	0	0	0	0	0	0	0	0	0	0

Servo MX-106, Posisi poros pada titik 4095

No.	PosN	PosG	SpeedN	SpeedG	Current	Status Torsi	P Gain	I Gain	D Gain	Return Delay	Temp	Speed Real	Current Real
1	4093	4095	0	0	0	0	1	32	0	0	20	36	0
2	4093	4095	0	0	0	0	1	32	0	0	20	36	0
3	4094	4095	0	0	0	0	1	32	0	0	20	36	0
4	4094	4095	0	0	0	0	1	32	0	0	20	36	0
5	4093	4095	0	0	0	0	1	32	0	0	20	36	0
6	4093	4095	0	0	0	0	1	32	0	0	20	36	0
7	4094	4095	0	0	0	0	1	32	0	0	20	36	0
8	4094	4095	0	0	0	0	1	32	0	0	20	36	0
9	4093	4095	0	0	0	0	1	32	0	0	20	36	0
10	4094	4095	0	0	0	0	1	32	0	0	20	36	0
11	4094	4095	0	0	0	0	1	32	0	0	20	36	0
12	4094	4095	0	0	0	0	1	32	0	0	20	36	0
13	4093	4095	0	0	0	0	1	32	0	0	20	36	0
14	4093	4095	0	0	0	0	1	32	0	0	20	36	0
15	4093	4095	0	0	0	0	1	32	0	0	20	36	0
16	4093	4095	0	0	0	0	1	32	0	0	20	36	0
17	4094	4095	0	0	0	0	1	32	0	0	20	36	0
18	4093	4095	0	0	0	0	1	32	0	0	20	36	0
19	4094	4095	0	0	0	0	1	32	0	0	20	36	0
20	4094	4095	0	0	0	0	1	32	0	0	20	36	0
Error (%)	0.0366	0	0	0	0	0	0	0	0	0	0	0	0

### LAMPIRAN 3 Hasil Pengujian *Multiple Servos Control*

Tiga buah servo pada poros di titik 0

No.	MX-28	MX-64	MX-106
1	2	2	2
2	2	1	2
3	2	1	2
4	2	2	2
5	2	2	2
6	2	1	2
7	2	2	2
8	2	1	2
9	2	1	2
10	2	2	2
11	2	1	3
12	1	2	2
13	2	1	2
14	2	1	2
15	2	2	2
16	2	2	2
17	1	2	2
18	2	1	2
19	2	2	2
20	2	2	2
Error (%)	0.046387	0.037842	0.050049

Tiga buah servo pada poros di titik 2048

No.	MX-28	MX-64	MX-106
1	2046	2046	2046
2	2046	2046	2046
3	2046	2046	2046
4	2046	2046	2046
5	2047	2046	2046
6	2046	2046	2046
7	2046	2046	2046
8	2046	2046	2046
9	2046	2046	2046
10	2046	2046	2046
11	2046	2046	2046
12	2046	2046	2046
13	2046	2046	2046
14	2046	2046	2046
15	2047	2046	2046
16	2046	2046	2046
17	2046	2046	2046
18	2046	2046	2046
19	2046	2046	2046

20	2046	2046	2046
Error (%)	0.046387	0.048828	0.048828

Tiga buah servo pada poros di titik 4095

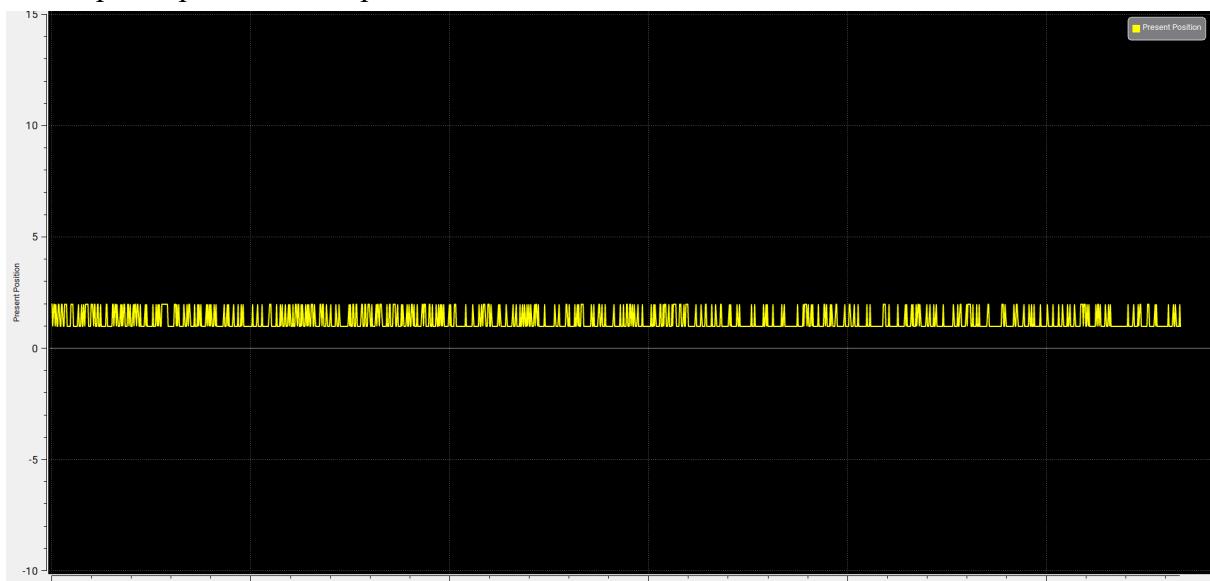
No.	MX-28	MX-64	MX-106
1	4093	4094	4094
2	4093	4094	4094
3	4093	4094	4094
4	4093	4094	4094
5	4093	4094	4094
6	4093	4094	4094
7	4093	4094	4094
8	4093	4094	4094
9	4093	4094	4094
10	4093	4094	4094
11	4093	4094	4094
12	4093	4094	4094
13	4093	4094	4094
14	4093	4094	4094
15	4093	4094	4094
16	4093	4094	4094
17	4093	4094	4094
18	4093	4094	4094
19	4093	4094	4094
20	4093	4094	4094
Error (%)	0.048828	0.024414063	0.024414

## LAMPIRAN 4 Hasil Pengujian Kontroler Acuan CM-740

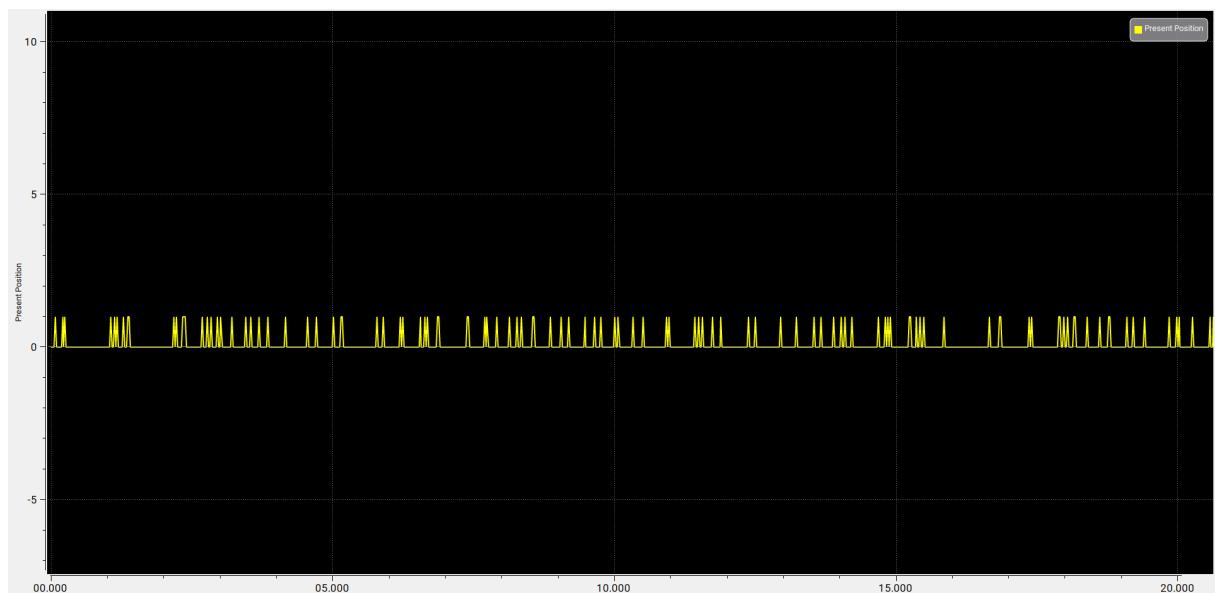
Poros di titik 0

No.	MX-28	MX-64	MX-106
1	1	1	2
2	2	1	2
3	1	1	2
4	1	1	2
5	2	1	2
6	1	1	2
7	1	1	2
8	1	1	2
9	2	1	2
10	1	1	2
11	1	1	2
12	1	1	2
13	1	1	2
14	1	1	2
15	2	1	2
16	1	1	2
17	1	1	2
18	2	1	2
19	2	1	2
20	1	1	2
Error (%)	0.03139	0.024414063	0.048828

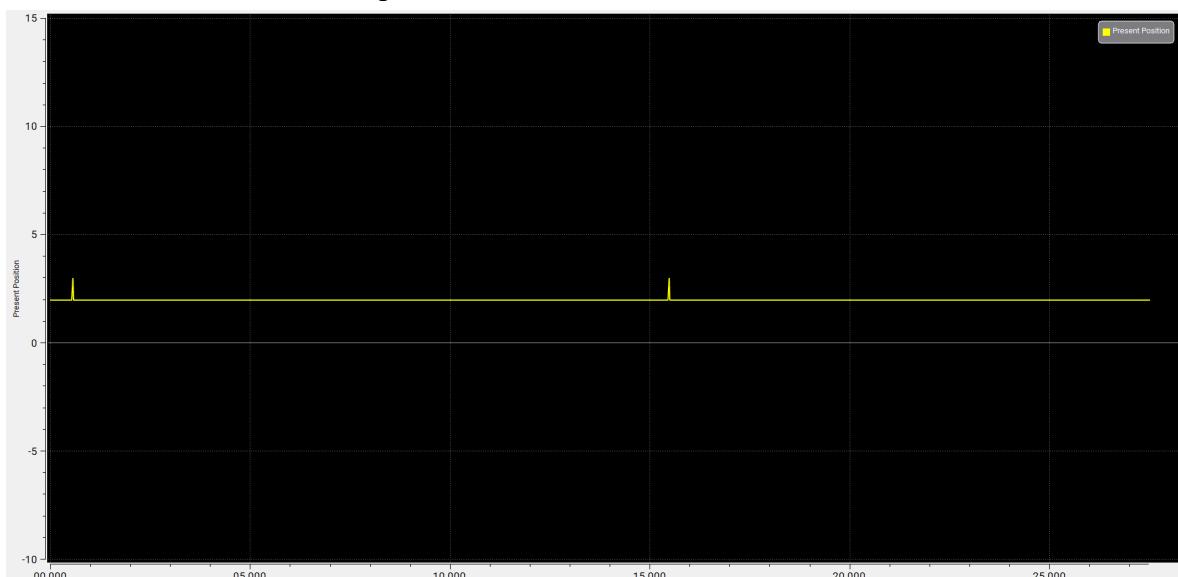
Grafik posisi poros MX-28 pada titik 0



Grafik Posisi Poros MX-64 pada titik 0



Grafik Posisi Poros MX-106 pada titik 0

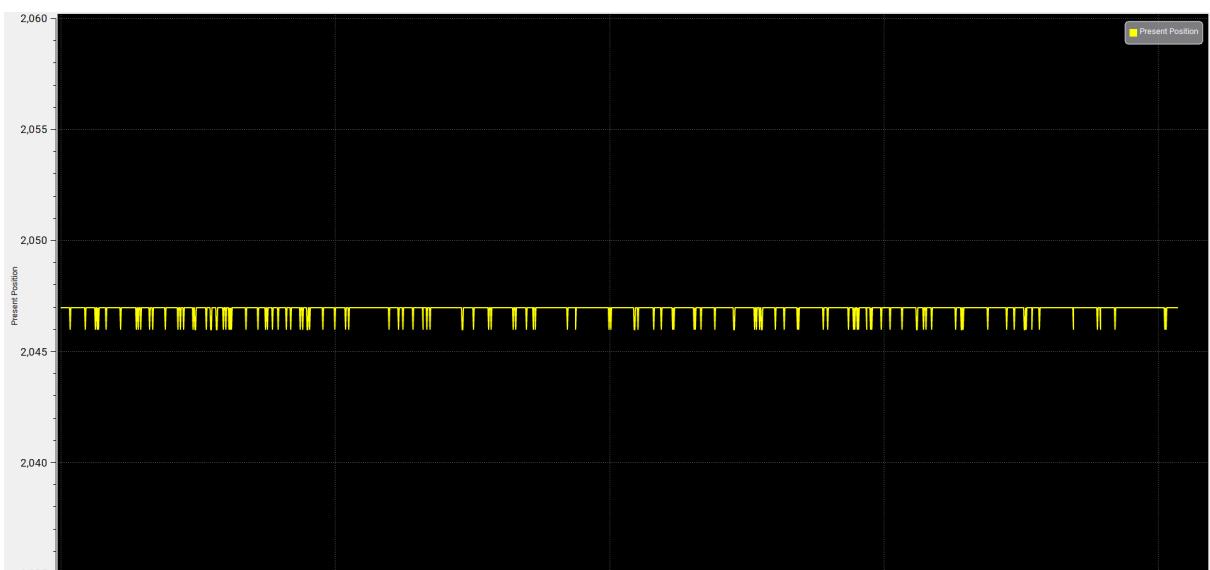


Poros di titik 2048

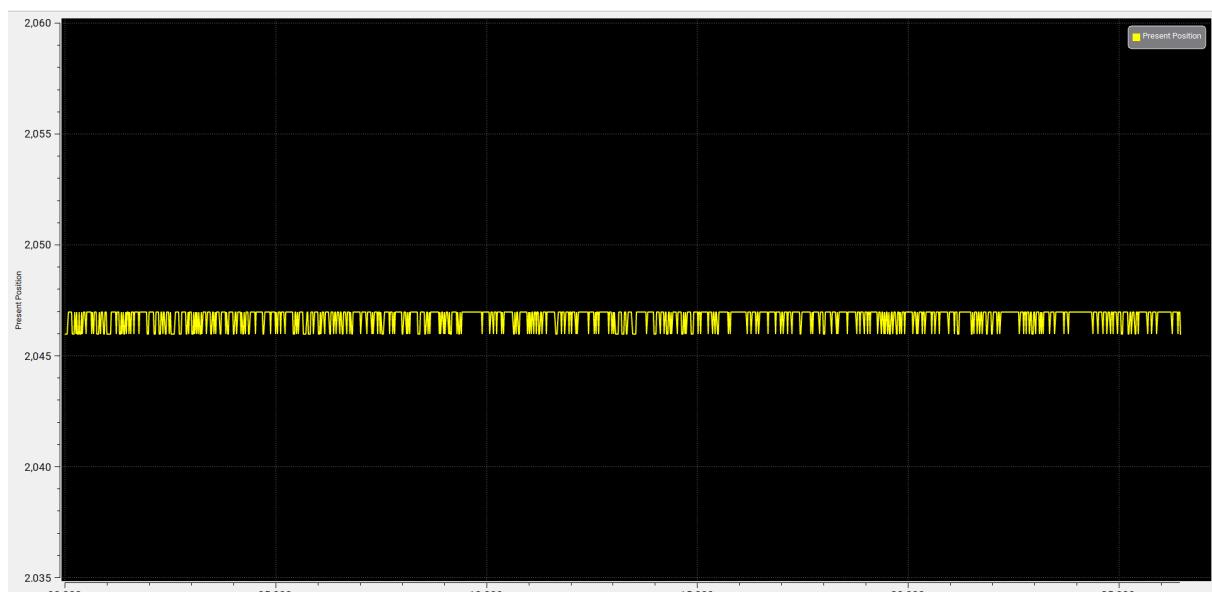
No.	MX-28	MX-64	MX-106
1	2046	2047	2045
2	2046	2047	2046
3	2047	2046	2046
4	2047	2046	2046
5	2046	2047	2046
6	2047	2046	2046
7	2047	2047	2046
8	2046	2046	2046
9	2047	2047	2046
10	2047	2046	2046
11	2046	2046	2046

12	2047	2047	2046
13	2046	2046	2046
14	2047	2046	2046
15	2046	2046	2046
16	2047	2047	2046
17	2046	2047	2046
18	2047	2046	2046
19	2047	2047	2046
20	2047	2047	2046
Error (%)	0.034877	0.03604	0.049991

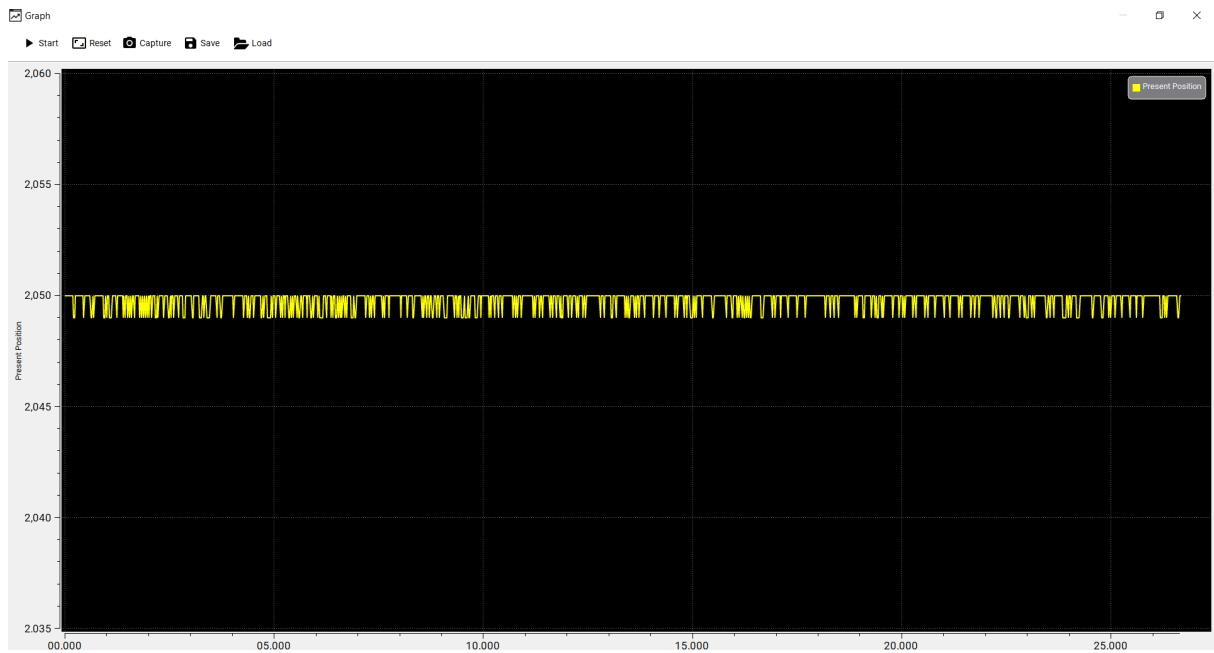
Grafik Posisi Poros MX-28 pada titik 2048



Grafik Posisi Poros MX-64 pada titik 2048



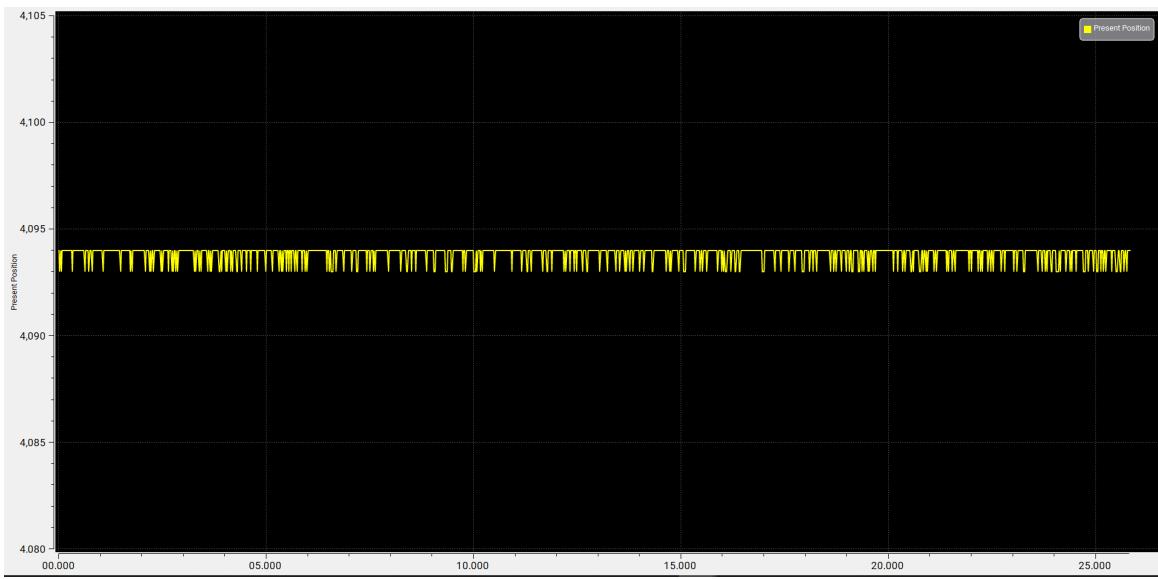
Grafik Posisi Poros MX-106 pada titik 2048



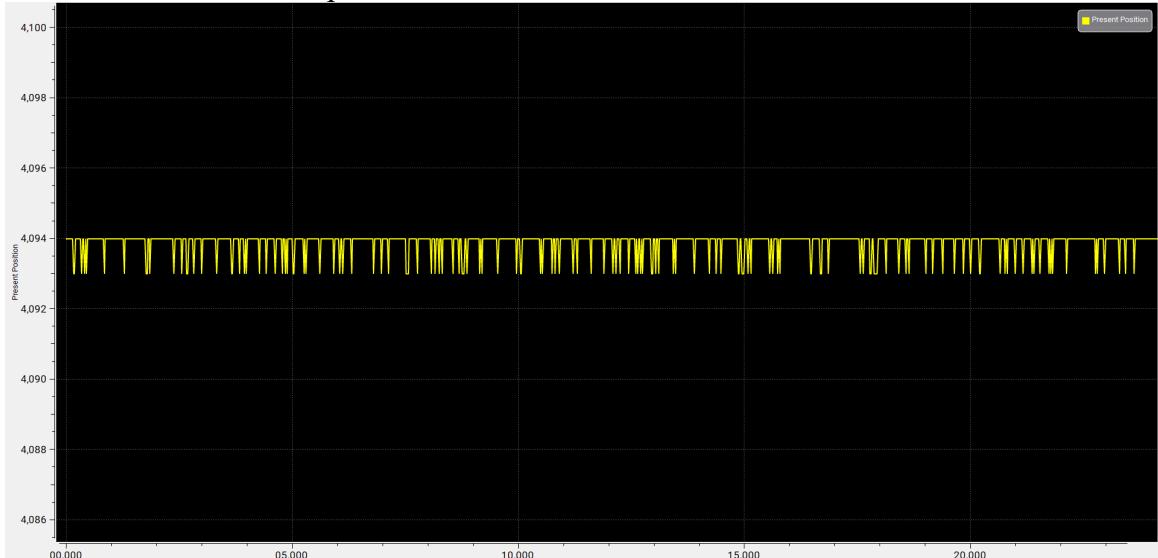
Poros di titik 4095

No.	MX-28	MX-64	MX-106
1	4094	4094	4093
2	4093	4093	4093
3	4094	4094	4093
4	4093	4094	4093
5	4093	4093	4093
6	4093	4094	4092
7	4094	4094	4093
8	4094	4093	4093
9	4093	4094	4092
10	4094	4093	4093
11	4093	4094	4093
12	4094	4093	4093
13	4093	4093	4093
14	4094	4094	4093
15	4094	4094	4093
16	4094	4093	4093
17	4093	4093	4093
18	4094	4094	4093
19	4094	4093	4093
20	4093	4093	4093
Error (%)	0.034877	0.03604	0.051153

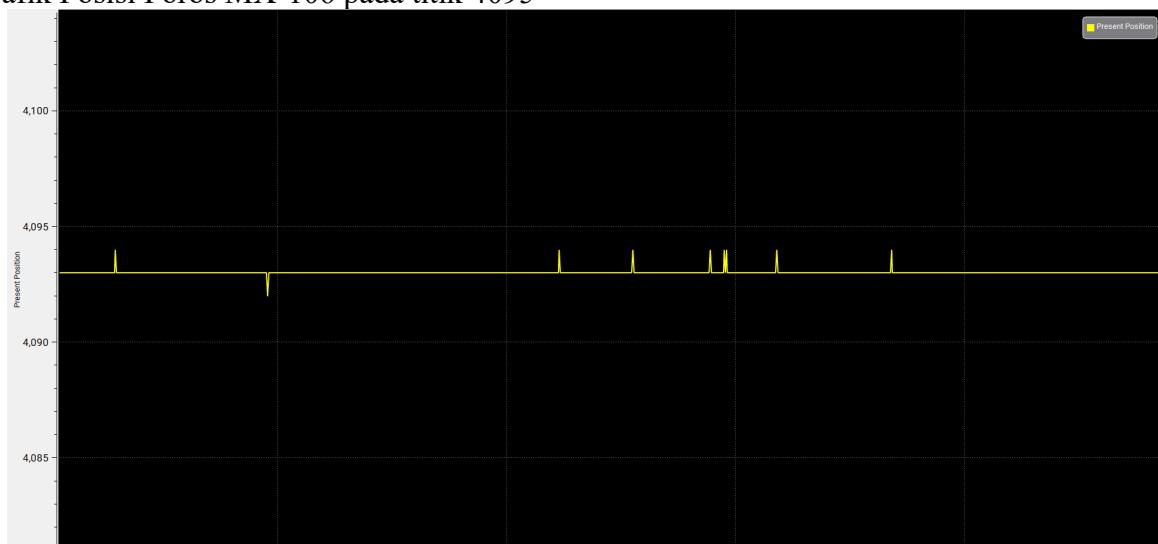
Grafik Posisi Poros MX-28 pada titik 4095



Grafik Posisi Poros MX-64 pada titik 4095



Grafik Posisi Poros MX-106 pada titik 4095



## LAMPIRAN 5 Hasil Pengujian durasi *Return Delay*

Durasi *return delay* 0  $\mu$ S

No.	MX28	MX64	MX-106
1	10.92	11.026	11.672
2	11.88	11.931	12.787
3	11.685	9.896	10.899
4	10.567	10.009	12.0158
5	11.837	11.818	13.903
6	10.923	12.977	12.873
7	10.821	10.942	13.817
8	9.805	11.055	10.728
9	11.024	13.939	13.817
10	10.364	10.998	10.728
11	11.837	10.942	11.672
12	11.837	11.959	12.53
13	10.923	10.998	11.672
14	10.923	10.942	11.672
15	10.821	10.942	11.757
16	10.923	14.956	11.757
17	11.837	9.924	11.757
18	12.04	10.998	11.843
19	10.821	11.055	11.744
20	10.923	12.921	11.986
Error (%)	0.034877	0.03604	0.051153

Durasi *return delay* 50  $\mu$ S

No.	MX28	MX64	MX-106
1	100.87	106.516	111.186
2	101.151	119.186	112.003
3	99.184	105.442	108.688
4	100.589	107.375	111.798
5	100.87	105.872	112.062
6	99.184	105.657	107.101
7	99.465	103.796	107.519
8	100.589	104.583	107.798
9	100.589	103.509	110.275
10	99.184	103.939	108.901
11	99.4655	103.509	109.798
12	100.308	106.516	108.424
13	100.308	104.154	111.068
14	100.027	105.872	111.862
15	100.027	106.945	110.327
16	100.589	105.227	107.895
17	100.87	103.939	113.533
18	100.308	102.865	107.63

19	100.027	106.731	111.333
20	98.903	107.59	110.407
Error (%)	0.034877	0.03604	0.051153

Durasi *return delay* 100  $\mu$ S

No.	MX28	MX64	MX-106
1	190	197.486	209.028
2	189.824	197.727	208.395
3	189.095	198.448	269.779
4	188.65	196.38	207.879
5	189.971	199.836	207.047
6	189.584	203.383	208.395
7	189.998	200.03	208.709
8	188.319	210.015	269.605
9	189.477	204.682	271.949
10	190.923	217.281	210.207
11	189.318	198.286	208.395
12	189.426	197.015	209.855
13	189.065	197.898	208.892
14	190.247	196.929	210.691
15	189.872	199.449	206.348
16	190.485	199.061	227.481
17	189.705	198.821	209.566
18	189.971	198.673	208.911
19	190.237	199.015	209.809
20	188.641	200.999	206.59
Error (%)	0.034877	0.03604	0.051153

## LAMPIRAN 6 Kode Program Kontrol Dynamixel

```
//Kode Untuk Halaman Pilih Port dan Baudrate
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO.Ports;
using System.Diagnostics;
using System.Reflection;
namespace Connect_Form
{
    public partial class Connection : Form
    {
        string portname;
        Int32 baudrate;
        public static Connection instance;
        public RichTextBox t;
        public Connection()
        {
            InitializeComponent();
            instance = this;
            t = debugBox;
        }
        private void Connection_Load(object sender, EventArgs e)
        {
            SetPortNameValues();
            box_Ports.Enabled = true;
            box_Bauds.Enabled = true;
            button_Connect.Enabled = true;
            button_Disconnect.Enabled = false;
            check_automscroll.Enabled = true;
        }
        public void AppendText(string value)
        {
            if (InvokeRequired)
            {
                this.BeginInvoke(new Action<string>(AppendText), new object[]
{ value });
                return;
            }
            debugBox.Text += value;
        }
        public void SetPortNameValues()
        {
            String[] ports = SerialPort.GetPortNames();
            box_Ports.Items.Clear();
            foreach (string port in ports)
            {
                box_Ports.Items.Add(port);
            }
            if (box_Ports.Items.Count > 0)
            {
                (box_Ports).SelectedIndex = 0;
            }
        }
    }
}
```

```

        else
        {
            box_Ports.Text = " ";
        }

    }
    public async void initiate_progress()
{
    List<string> list = new List<string>();
    for (int i = 0; i < list.Count; i++)
    {
        list.Add(i.ToString());
    }
    var progress = new Progress<ProgressReport>();
    progress.ProgressChanged += (o, report) =>
    {
        label_progress.Text = string.Format("{0}%", report.PercentComplete);
        progress_Connect.Value = report.PercentComplete;
        progress_Connect.Update();
    };
    await ProcessData(list, progress);
    label_progress.Text = "Done!";
}
public void button_Connect_Click(object sender, EventArgs e)
{
    try
    {
        if (!Serial_Port.IsOpen)
        {
            initiate_progress();
            portname = box_Ports.Text;
            baudrate = Convert.ToInt32(box_Bauds.Text);

            Serial_Port.PortName = portname;
            Serial_Port.BaudRate = baudrate;
            Serial_Port.Open();
            if (Serial_Port.IsOpen)
            {
                Thread.Sleep(200);
                Serial_Port.Close();
                openChildForm(new Single_Servo(portname, baudrate));
                Thread.Sleep(200);
                cekFrm("Single_Servo", true);

                button_Connect.Enabled = false;
                button_Disconnect.Enabled = true;
                //button_Analysis.Enabled = true;
                button_single_servo.Enabled = false;
                button_multiple_servo.Enabled = true;
                debugBox.Text += Environment.NewLine + "Sukses koneksi pada Port " + portname + ", Baudrate " + baudrate.ToString());
            }
        }
        else
        {
        }
    }
}

```

```

        catch (Exception err)
    {
        Thread.Sleep(1000);
        debugBox.Text = err.Message;
        MessageBox.Show(err.Message, "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        Serial_Port.Close();
        SetPortNameValues();
        button_Connect.Enabled = true;
        button_Disconnect.Enabled = false;
    }
}

private void button_Disconnect_Click(object sender, EventArgs e)
{
    try
    {
        if (Serial_Port.IsOpen)
        {
            Serial_Port.Close();
            progress_Connect.Value = 0;
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    button_Connect.Enabled = true;
    button_Disconnect.Enabled = false;
    progress_Connect.Value = 0;
}

private void Connection_FormClosing(object sender, FormClosingEventArgs e)
{
    try
    {
        if (Serial_Port.IsOpen)
        {
            Serial_Port.Close();
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
    }
}

private Task ProcessData(List<string> list, IProgress<ProgressReport>
progress)
{
    int index = 1;
    int totalProcess=list.Count;
    var progressReport = new ProgressReport();
    return Task.Run(() =>
    {
        for (int i = 0; i < 100; i++)
        {
            progressReport.PercentComplete = index++ * 100 / 100;
            progress.Report(progressReport);
        }
    });
}

```

```

        catch (Exception err)
        {
            Thread.Sleep(1000);
            debugBox.Text = err.Message;
            MessageBox.Show(err.Message, "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            Serial_Port.Close();
            SetPortNameValues();
            button_Connect.Enabled = true;
            button_Disconnect.Enabled = false;
        }
    }
private void button_Disconnect_Click(object sender, EventArgs e)
{
    try
    {
        if (Serial_Port.IsOpen)
        {
            Serial_Port.Close();
            progress_Connect.Value = 0;
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    button_Connect.Enabled = true;
    button_Disconnect.Enabled = false;
    progress_Connect.Value = 0;
}

private void Connection_FormClosing(object sender, FormClosingEventArgs e)
{
    try
    {
        if (Serial_Port.IsOpen)
        {
            Serial_Port.Close();

        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
    }
}
private Task ProcessData(List<string> list, IProgress<ProgressReport>
progress)
{
    int index = 1;
    int totalProcess=list.Count;
    var progressReport = new ProgressReport();
    return Task.Run(() =>
    {
        for (int i = 0; i < 100; i++)
        {
            progressReport.PercentComplete = index++ * 100 / 100;
            progress.Report(progressReport);
        }
    });
}

```

```

        Thread.Sleep(10);
    });
}

#region buka form kecil
private Form activeForm = null;
private void openChildForm(Form childForm)
{
    if(activeForm != null)
        activeForm.Close();
    activeForm = childForm;
    childForm.TopLevel = false;
    childForm.FormBorderStyle = FormBorderStyle.None;
    childForm.Dock = DockStyle.Fill;
    panelChildForm.Controls.Add(childForm);
    panelChildForm.Tag = childForm;
    childForm.BringToFront();
    childForm.Show();
}
private void closeChildForm(Form childform)
{
    if (activeForm != null)
        activeForm.Close();
    activeForm = childform;
    childform.Close();
}
public void cekFrm(string cekform, bool status){
    try
    {
        Form form_name = Application.OpenForms[cekform];
        if (activeForm == form_name && status)
        {
            debugBox.Text += ("Sub dibuka".Insert(4, cekform));
            if (cekform == "Single_Servo")
            {
                button_single_servo.Enabled = false;
                button_multiple_servo.Enabled = true;
                //button_Analysis.Enabled = true;

            }
            else if (cekform == "Multiple_Servo")
            {
                button_multiple_servo.Enabled = false;
                button_single_servo.Enabled = true;
                //button_Analysis.Enabled = true;

            }
            else if (cekform == "Analysis"){
                //Enabled = false;
                button_single_servo.Enabled = true;
                button_multiple_servo.Enabled = true;
            }
        }
        else if (activeForm == form_name && !status)
        {
            debugBox.Text += ("Sub ditutup".Insert(4, cekform));
            if (cekform == "Single_Servo")
                button_single_servo.Enabled = true;
            else if (cekform == "Multiple_Servo")
                button_multiple_servo.Enabled = true;
            //else if (cekform == "Analysis")
                //button_Analysis.Enabled = true;
        }
    }
}

```

```

        else{ }}
    catch (Exception ex) {
        debugBox.Text += ex.Message;}
private void button_Connection_Click(object sender, EventArgs e)
{
    if (Serial_Port.IsOpen)
    {
        button_single_servo.Enabled = true;
        button_multiple_servo.Enabled = true;
        //xbutton_Analysis.Enabled = true;
        foreach (Form frm in Application.OpenForms) {
            closeChildForm(frm); }}
        group_connection.Enabled = true;
        group_connection.Visible = true; }
private void button_single_servo_Click(object sender, EventArgs e) {
    if(Serial_Port.IsOpen)
        Serial_Port.Close();
    progress_Connect.Value = 0;
    if (activeForm != null)
        activeForm.Close();
    Thread.Sleep(200);
    openChildForm(new Single_Servo(portname, baudrate));
    Thread.Sleep(200);
    cekFrm("Single_Servo", true); }
private void button_multiple_servo_Click(object sender, EventArgs e)
{
    if (Serial_Port.IsOpen)
        Serial_Port.Close();
    progress_Connect.Value = 0;
    initiate_progress();
    if (activeForm != null)
        activeForm.Close();
    Thread.Sleep(1000);
    openChildForm(new Multiple_Servo(portname, baudrate));
    cekFrm("Multiple_Servo", true);

}
private void button_Analysis_Click(object sender, EventArgs e)
{
    progress_Connect.Value = 0;
    initiate_progress();
    if (activeForm != null)
        activeForm.Close();
    Serial_Port.Close();
    openChildForm(new Analysis());
    cekFrm("Analysis", true);
}
#endregion
private void debugBox_TextChanged(object sender, EventArgs e)
{
    debugBox.Text += "\n";
    if (check_automscroll.Checked)
    {
        debugBox.SelectionStart = debugBox.TextLength;
        debugBox.ScrollToCaret();
    }
}
private void button1_Click(object sender, EventArgs e)
{
    debugBox.Text = "";
}

```

```

//Kode Untuk Single Servo Control
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO.Ports;
using System.Threading;
namespace Connect_Form
{
    public partial class Single_Servo : Form
    {

        Int32 indexCurrReal,nilaiCurrReal,indexSpeedReal,idpilih,indexList,
        nilaiServo, nilaiSModel,indexServo, indexSModel,indexID, indexModel, indexBaud,
        indexProtocol, indexPos_now, indexPos_Goal, index_Speed_now, index_Speed_goal,
        index_torque, index_Current, index_p_gain, index_i_gain, index_d_gain,
        index_return_delay, index_temp;
        string dataCurrReal,dataSpeedReal,pos_val, dataServo, dataSModel,
        dataIN, dataID, dataModel, dataBaud,dataProtocol,dataPos_now,dataPos_Goal,
        dataSpeed_now,dataSpeed_goal,dataTorque,dataCurrent,data_p_gain,data_i_gain,data
        _d_gain,data_return_delay,data_temp;
        public DataTable List_servo = new DataTable("List_servo");
        public static Single_Servo instance;
        #region Form Load
        public Single_Servo(String port, Int32 baud)
        {
            try
            {
                InitializeComponent();
                Serial_Port2.PortName = port;
                Serial_Port2.BaudRate = baud;
                Serial_Port2.Open();
                instance = this;
                Serial_Port2.WriteLine("mode0");
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        private void Single_Servo_Load(object sender, EventArgs e)
        {
            try
            {
                if (Serial_Port2.IsOpen)
                {
                    List_servo.Columns.Add("Servo_id",
Type.GetType("System.Int32"));
                    List_servo.Columns.Add("Servo_model",
Type.GetType("System.Int32"));
                    List_Servo_Single.DataSource = List_servo;
                    Console.WriteLine("Open");
                }
                else
                {

```

```

                Console.WriteLine("Close");}}
        catch (Exception ex)
        { Console.WriteLine(ex.Message); }}
#endregion
#region Disconnect Form
private void Disconnect_Click(object sender, EventArgs e) {
    Serial_Port2.Close();
    Thread.Sleep(100);
    Connection.instance.t.Text += "Sub Single_Servo ditutup";
    this.Close();}
private void Single_Servo_FormClosing(object sender, FormClosingEventArgs
e) {
    try{
        if (Serial_Port2.IsOpen) {
            Serial_Port2.Close();
            Connection.instance.t.Text += "Sub Single_Servo ditutup";}
        catch (Exception ex) {
            Connection.instance.t.Text += ex.Message; } }
#endregion
#region .. Double Buffered function ..
public static void SetDoubleBuffered(System.Windows.Forms.Control c) {
    if (System.Windows.Forms.SystemInformation.TerminalServerSession)
        return;
    System.Reflection.PropertyInfo aProp =
typeof(System.Windows.Forms.Control).GetProperty("DoubleBuffered",
System.Reflection.BindingFlags.NonPublic |
System.Reflection.BindingFlags.Instance);
    aProp.SetValue(c, true, null); }
#endregion
#region .. code for Flucuring ..
protected override CreateParams CreateParams {
    get {
        CreateParams cp = base.CreateParams;
        cp.ExStyle |= 0x02000000;
        return cp; } }
#endregion
#region Terima data serial
public bool Udahpilih = false;
private void SP_DataReceived(object sender, SerialDataReceivedEventArgs e){
    try{
        dataIN = Serial_Port2.ReadLine();
        this.Invoke(new EventHandler(setText)); }
    catch (Exception x) {
        MessageBox.Show(x.Message); } }
public void setText(object sender, EventArgs e) {
    try{
        SetDoubleBuffered(table_single);
        if(Udahpilih == false){
            indexServo = Convert.ToInt32(dataIN.IndexOf("Servo"));
            indexSModel =
Convert.ToInt32(dataIN.IndexOf("SModel"));
            dataServo = dataIN.Substring(indexServo +
5,(indexSModel-indexServo)-5);
            dataSModel = dataIN.Substring(indexSModel + 6);
            nilaiServo = Convert.ToInt32(dataServo);
            nilaiSModel = Convert.ToInt32(dataSModel);
            Connection.instance.t.Text += "Servo " + dataServo +
"Model " + nilaiSModel;
        }
    }
}

```

```

        if (nilaiServo == 999) {
            Connection.instance.t.Text += "OK List Selesai
diterima"; }
        else{
            List_servo.Rows.Add(nilaiServo, nilaiSModel); } }
        else {
            terima_data(sender, e); }}
    catch (Exception er) {
        Connection.instance.t.Text += er.Message; }
public void terima_data(object sender, EventArgs e)
{
    indexID = Convert.ToInt32(dataIN.IndexOf("ID"));
    indexBaud = Convert.ToInt32(dataIN.IndexOf("Baud"));
    indexModel = Convert.ToInt32(dataIN.IndexOf("Model"));
    indexProtocol = Convert.ToInt32(dataIN.IndexOf("Prot"));
    indexPos_now = Convert.ToInt32(dataIN.IndexOf("PosN"));
    indexPos_Goal = Convert.ToInt32(dataIN.IndexOf("PosG"));
    index_Speed_now = Convert.ToInt32(dataIN.IndexOf("SpeedN"));
    index_Speed_goal = Convert.ToInt32(dataIN.IndexOf("SpeedG"));
    index_Current = Convert.ToInt32(dataIN.IndexOf("Current"));
    index_torque = Convert.ToInt32(dataIN.IndexOf("Torque"));
    index_p_gain = Convert.ToInt32(dataIN.IndexOf("pgain"));
    index_i_gain = Convert.ToInt32(dataIN.IndexOf("igain"));
    index_d_gain = Convert.ToInt32(dataIN.IndexOf("dgain"));
    index_return_delay = Convert.ToInt32(dataIN.IndexOf("retdelay"));
    index_temp = Convert.ToInt32(dataIN.IndexOf("temp"));
    indexSpeedReal = Convert.ToInt32(dataIN.IndexOf("speedReal"));
    indexCurrReal = Convert.ToInt32(dataIN.IndexOf("RealCurr"));

    dataID = dataIN.Substring(indexID + 2, (indexBaud - indexID) - 2);
    dataBaud = dataIN.Substring(indexBaud + 4, (indexModel -
indexBaud) - 4);
    dataModel = dataIN.Substring(indexModel + 5, (indexProtocol -
indexModel) - 5);
    dataProtocol = dataIN.Substring(indexProtocol + 4, (indexPos_now -
indexProtocol) - 4);
    dataPos_now = dataIN.Substring(indexPos_now + 4, (indexPos_Goal -
indexPos_now) - 4);
    dataPos_Goal = dataIN.Substring(indexPos_Goal + 4,
(index_Speed_now - indexPos_Goal) - 4);
    dataSpeed_now = dataIN.Substring(index_Speed_now + 6,
(index_Speed_goal - index_Speed_now) - 6);
    dataSpeed_goal = dataIN.Substring(index_Speed_goal + 6,
(index_Current - index_Speed_goal) - 6);
    dataCurrent = dataIN.Substring(index_Current + 7, (index_torque -
index_Current) - 7);
    dataTorque = dataIN.Substring(index_torque + 6, (index_p_gain -
index_torque) - 6);
    data_p_gain = dataIN.Substring(index_p_gain + 5, (index_i_gain -
index_p_gain) - 5);
    data_i_gain = dataIN.Substring(index_i_gain + 5, (index_d_gain -
index_i_gain) - 5);
    data_d_gain = dataIN.Substring(index_d_gain + 5,
(index_return_delay - index_d_gain) - 5);
    data_return_delay = dataIN.Substring(index_return_delay + 8,
(index_temp - index_return_delay) - 8);
    data_temp = dataIN.Substring(index_temp + 4, (indexSpeedReal -
index_temp)-4);
    dataSpeedReal = dataIN.Substring(indexSpeedReal + 9,
(indexCurrReal - indexSpeedReal) - 9);
}

```

```

        dataCurrReal = dataIN.Substring(indexCurrReal + 8);
        label_Value_ID.Text = dataID;
        label_Value_Model.Text = dataModel;
        label_Value_Baud.Text = dataBaud;
        label_Value_Protocol.Text = dataProtocol;
        label_Value_Torque.Text = dataTorque;
        label_Value_Goal_Pos.Text = dataPos_Goal;
        label_Value_Now_Pos.Text = dataPos_now;
        label_Act_Pos.Text =
Convert.ToString((Convert.ToDouble(dataPos_now)) / 4096 * 360);
        label_Value_Goal_Speed.Text = dataSpeed_goal;
        label_Value_Now_Speed.Text = dataSpeed_now;
        label_Value_Torque.Text = dataTorque;
        label_Value_Now_Current.Text = dataCurrent;
        label_id_box.Text = dataID;
        label_Now_baud.Text = dataBaud;
        label_p_gain.Text = data_p_gain;
        label_i_gain.Text = data_i_gain;
        label_d_gain.Text = data_d_gain;
        label_ret_val.Text = data_return_delay;
        label_temp.Text = data_temp;
        label_P_now.Text = data_p_gain;
        label_I_now.Text = data_i_gain;
        label_Act_Speed.Text = dataSpeedReal;
        label_Act_Current.Text = dataCurrReal; }

#endregion
#region Clickable dalam Table
private void List_Servo_Single_CellClick(object sender,
DataGridViewCellEventArgs e) {
    indexList = e.RowIndex;
    DataGridViewRow row = List_Servo_Single.Rows[indexList];
    idpilih = Convert.ToInt32(row.Cells[0].Value);
    Connection.instance.t.Text += "ID yang dipilih adalah : " +
idpilih;
    Serial_Port2.WriteLine("ganti" + idpilih.ToString());
    Udahpilih = true; }
private void button_rescan_Click(object sender, EventArgs e) {
    Serial_Port2.WriteLine("reset");}
private void checkBox_Pos_CheckedChanged(object sender, EventArgs e){
    if (checkBox_Pos.Checked) {
        Bar_Position.Enabled = true; }
    else {
        Bar_Position.Enabled = false; }}
private void label_ID_Click(object sender, EventArgs e) {
    group_ID.Visible = true;
    group_Baud.Visible = false;
    group_Pos.Visible = false;
    group_P_gain.Visible = false;
    group_i_gain.Visible = false;
    group_d_gain.Visible = false; }
private void button_p_gain_Click(object sender, EventArgs e) {
    group_P_gain.Visible = true;
    group_ID.Visible = false;
    group_Baud.Visible = false;
    group_Pos.Visible = false;
    group_i_gain.Visible = false;
    group_d_gain.Visible = false; }
private void Bar_Position_MouseUp(object sender, MouseEventArgs e) {
    Serial_Port2.WriteLine("Pos" + pos_val); }

```

```

private void Bar_Position_Scroll(object sender, EventArgs e) {
    label_Now_pos.Text = Bar_Position.Value.ToString();
    pos_val = Convert.ToString(Bar_Position.Value);
    if (button_center_clicked == true) {
        Bar_Position.Value = 2048;
        label_Now_pos.Text = 2048.ToString();
        Serial_Port2.WriteLine("Pos" + 2048);
        button_center_clicked = false; }
private void label_Pos_Now_Click(object sender, EventArgs e) {
    try{
        group_P_gain.Visible = false;
        group_ID.Visible = false;
        group_Baud.Visible = false;
        group_Pos.Visible = true;
        group_i_gain.Visible = false;
        group_d_gain.Visible = false; }
    catch (Exception ex) {
        MessageBox.Show(ex.Message); } }
private void button_send_baud_Click(object sender, EventArgs e) {
    string Baud_out = box_Send_Baud.Text;
    try{
        if (Baud_out != dataBaud) {
            Serial_Port2.WriteLine("Baud" + Baud_out); }
        catch (Exception ex) {
            Connection.instance.t.Text += "Baudrate sama, tidak perlu
dikirim ulang"; } }
private void button_d_gain_Click(object sender, EventArgs e) {
    group_P_gain.Visible = false;
    group_ID.Visible = false;
    group_Baud.Visible = false;
    group_Pos.Visible = false;
    group_i_gain.Visible = false;
    group_d_gain.Visible = true; }
private void button_i_gain_Click(object sender, EventArgs e) {
    group_P_gain.Visible = false;
    group_ID.Visible = false;
    group_Baud.Visible = false;
    group_Pos.Visible = false;
    group_i_gain.Visible = true;
    group_d_gain.Visible = false; }
private void label_Baudrate_Click(object sender, EventArgs e) {
    group_P_gain.Visible = false;
    group_ID.Visible = false;
    group_Baud.Visible = true;
    group_Pos.Visible = false;
    group_i_gain.Visible = false;
    group_d_gain.Visible = false; }
public bool button_center_clicked = false;
private void button_center_Click(object sender, EventArgs e) {
    try {
        button_center_clicked = true; }
    catch (Exception ex) {
        MessageBox.Show(ex.Message); } }
private void button_Send_Click_ID(object sender, EventArgs e) {
    try{
        string ID_out = box_Send_ID.Text;
        if ((ID_out != dataID)) {
            Serial_Port2.WriteLine("ID" + ID_out);
            List_Servo_Single.Rows[indexList].Cells[0].Value =
Convert.ToInt32(ID_out); } }

```

```

        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
    private void button_i_send_Click(object sender, EventArgs e)
    {
        string i_out = box_i_gain.Text;
        try
        {
            if (i_out != data_p_gain)
            {
                Serial_Port2.WriteLine("IG" + i_out);
            }
        }
        catch (Exception ex)
        {
            Connection.instance.t.Text += "I Gain sama, tidak perlu
dikirim ulang";
        }
    }

    private void button_d_send_Click(object sender, EventArgs e)
    {
        string d_out = box_d_gain.Text;
        try
        {
            if (d_out != data_d_gain)
            {
                Serial_Port2.WriteLine("DG" + d_out);
            }
        }
        catch (Exception ex)
        {
            Connection.instance.t.Text += "D Gain sama, tidak perlu
dikirim ulang";
        }
    }
    private void button_send_p_gain_Click(object sender, EventArgs e)
    {
        string p_out = box_p_gain.Text;
        try
        {
            if (p_out!= data_p_gain)
            {
                Serial_Port2.WriteLine("PG" + p_out);
            }
        }
        catch (Exception ex)
        {
            Connection.instance.t.Text += "P Gain sama, tidak perlu
dikirim ulang";
        }
    }
    #endregion
}
}

```

```

//Kode Untuk Multiple Servos Control
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO.Ports;
using System.Threading;
namespace Connect_Form
{
    public partial class Multiple_Servo : Form {
        Int32 indexID,
indexModel,indexPos,indexSModel,indexServo,nilaiPos,nilaiSModel,nilaiServo,nilai
ID,nilaiModel;
        string dataIN,dataID,dataModel,dataPos,dataServo,dataSModel;
        private void button_rescan_Click(object sender, EventArgs e) {
            Serial_Port3.WriteLine("mode1");
            List_servo2.Clear();
            Udahkirim = false; }
        public static Multiple_Servo instance;
        public DataTable List_servo2 = new DataTable("List_Servo2");
        public Multiple_Servo(String port2, Int32 baud2) {
            InitializeComponent();
            Serial_Port3.PortName = port2;
            Serial_Port3.BaudRate = baud2;
            Serial_Port3.Open();
            instance = this; }
        private void Multiple_Servo_Load(object sender, EventArgs e) {
            Thread.Sleep(1000);
            Serial_Port3.WriteLine("mode1");
            List_servo2.Columns.Add("Servo_id", Type.GetType("System.Int32"));
            List_servo2.Columns.Add("Servo_model",
Type.GetType("System.Int32"));
            List_servo2.Columns.Add("Position", Type.GetType("System.Int32"));
            List_Servo_Multi.DataSource = List_servo2;
            label_pos_now.Text = "0";}
        private void bar_posisi_multi_Scroll(object sender, EventArgs e) {
            label_pos_now.Text = bar_posisi_multi.Value.ToString();
            label_PosDegree.Text = String.Format("{0}°", (float)
bar_posisi_multi.Value / 4095*360);
        }
        private void bar_posisi_multi_MouseUp(object sender, MouseEventArgs e) {
            Serial_Port3.WriteLine("Pos" + bar_posisi_multi.Value.ToString());}
        private void Disconnect_Click(object sender, EventArgs e) {
            Serial_Port3.Close();}
        #region Terima data serial
        private void Serial_Port3_DataReceived(object sender,
SerialDataReceivedEventArgs e) {
            try{
                dataIN = Serial_Port3.ReadLine();
                this.Invoke(new EventHandler(setText)); }
            catch (Exception x) {
                MessageBox.Show(x.Message); } }
        public bool Udahkirim = false;
    }
}

```

```

public void setText(object sender, EventArgs e) {
    try{
        if(Udahkirim == false) {
            indexServo = Convert.ToInt32(dataIN.IndexOf("Servo"));
            indexSModel = Convert.ToInt32(dataIN.IndexOf("SModel"));
            dataServo = dataIN.Substring(indexServo + 5, (indexSModel
- indexServo) - 5);
            dataSModel = dataIN.Substring(indexSModel + 6);
            nilaiServo = Convert.ToInt32(dataServo);
            nilaiSModel = Convert.ToInt32(dataSModel);
            Connection.instance.t.Text += "Servo " + dataServo +
"Model " + nilaiSModel;
            if (nilaiServo == 999) {
                Connection.instance.t.Text += "OK List Selesai
diterima";
                Udahkirim = true;
            } else {
                List_servo2.Rows.Add(nilaiServo, nilaiSModel); }
            indexID = Convert.ToInt32(dataIN.IndexOf("ID"));
            indexModel = Convert.ToInt32(dataIN.IndexOf("Model"));
            indexPos = Convert.ToInt32(dataIN.IndexOf("Pos"));
            dataID = dataIN.Substring(indexID + 2, (indexModel - indexID)
- 2);
            dataModel = dataIN.Substring(indexModel + 5, (indexPos -
indexModel) - 5);
            dataPos = dataIN.Substring(indexPos + 3);
            nilaiID = Convert.ToInt32(dataID);
            nilaiModel = Convert.ToInt32(dataModel);
            nilaiPos = Convert.ToInt32(dataPos);
            baca_list();
        }
        catch (Exception er) {
            Connection.instance.t.Text += "Gabisa" + er.Message; }
    }
    private void baca_list()
    {
        try
        {
            var count =
List_Servo_Multi.Rows.Cast<DataGridViewRow>().Count();
            for (int i = 0; i < count; i++)
            {
                DataGridViewRow row = List_Servo_Multi.Rows[i];
                if(row.Cells[0].Value.ToString() == dataID &&
row.Cells[1].Value.ToString() == dataModel)
                {
                    row.Cells[2].Value = nilaiPos;
                }
            }
        }
        catch (Exception ex)
        {
        }
    }
}
#endregion
}
}

```

## LAMPIRAN 7 Gambar Pengujian Penelitian

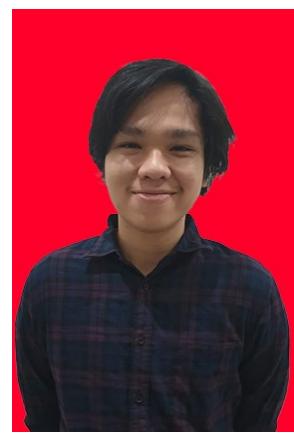
Pengujian kontrol pada 20 Servo



Tampilan Pengkontrolan 20 Buah Servo

Servo_id	Servo_model	Position
1	29	2044
2	29	2055
3	29	2045
4	29	2055
5	29	2056
6	29	2051
7	310	2046
8	310	2046
9	310	0
10	310	2044
11	310	2053
12	310	2043
13	310	2051
14	310	2038
15	310	2043
16	310	2055

## BIODATA PENULIS



Penulis dilahirkan di Jakarta, 7 Juli 2000, merupakan anak kedua dari 2 bersaudara. Penulis telah menempuh pendidikan formal yaitu di TK Budhaya II, SD Budhaya II, SMPK 5 Penabur dan SMAN 81 Jakarta . Setelah lulus dari SMAN tahun 2018, Penulis mengikuti seleksi Mandiri dan diterima di Departemen Teknik Elektro FTEIC - ITS pada tahun 2018 dan terdaftar dengan NRP 07111840000220.

Di lingkungan perkuliahan, penulis aktif di Tim Robotika dan mengikuti seminar yang diselenggarakan oleh Departemen Himpunan Mahasiswa Teknik Elektro (HIMATEKTRO) dan juga sempat membawakan seminar pelatihan dasar robotika dari robotika ITS.