TUGAS AKHIR – EF234801

Pengembangan *Plugin* Blender untuk Pengendalian *Servo* Robot Humanoid Berbasis Dynamixel SDK

Muhammad Daffa' Ashdaqfillah

NRP 5025211015

Dosen Pembimbing

Dr. Anny Yuniarti, S.Kom., M.Comp.Sc.

NIP 198106222005012002

Program Studi S-1 Teknik Informatika

Departemen Teknik Informatika

Fakultas Teknologi Elektro dan Informatika Cerdas

Institut Teknologi Sepuluh Nopember

Surabaya

2025

TUGAS AKHIR – EF234801

Pengembangan *Plugin* Blender untuk Pengendalian *Servo* Robot Humanoid Berbasis Dynamixel SDK

Muhammad Daffa' Ashdaqfillah

NRP 5025211015

Dosen Pembimbing

Dr. Anny Yuniarti, S.Kom., M.Comp.Sc.

NIP 198106222005012002

Program Studi S-1 Teknik Informatika

Departemen Teknik Informatika

Fakultas Teknologi Elektro dan Informatika Cerdas

Institut Teknologi Sepuluh Nopember

Surabaya

2025

FINAL PROJECT – EF234801

Development of a Blender Plugin for Servo Control in Humanoid Robots Using the Dynamixel SDK

Muhammad Daffa' Ashdaqfillah

NRP 5025211015

Advisor

Dr. Anny Yuniarti, S.Kom., M.Comp.Sc.

NIP 198106222005012002

Undergradeate Study Program of Informatics

Department of Informatics

Faculty of Intelligent Electrical and Informatics Technology

Institut Teknologi Sepuluh Nopember

Surabaya

2025

LEMBAR PENGESAHAN

Pengembangan *Plugin* Blender untuk Pengendalian *Servo* Robot Humanoid Berbasis Dynamixel SDK

TUGAS AKHIR

Diajukan untuk memenuhi salah satu syarat
memperoleh gelar Sarjana Komputer pada
Program Studi S-1 Teknik Informatika
Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Oleh: **Muhammad Daffa' Ashdaqfillah**
NRP. 5025211015

Disetujui oleh Tim Penguji Tugas Akhir:

1. Dr. Anny Yuniarti, S.Kom., M.Comp.Sc. Pembimbing
2. Dr. Eng. Darlis Herumurti, S.Kom., M.Kom Penguji
3. Hadziq Fabroyir, S.Kom., Ph.D. Penguji

SURABAYA

Juli, 2025

[Halaman ini sengaja dikosongkan]

APPROVAL SHEET

Development of a Blender Plugin for Servo Control in Humanoid Robots Using the Dynamixel SDK

FINAL PROJECT

Submitted to fulfill one of the requirements

for obtaining a computer bachelor degree at

Undergraduate Study program of Informatics

Departement of Informatics

Faculty of Intelligent Electrical and Informatics Technology

Institut Teknologi Sepuluh Nopember

By: **Muhammad Daffa' Ashdaqfillah**

NRP. 5025211015

Approved by Final Project Proposal Examiner Team:

1. Dr. Anny Yuniarti, S.Kom., M.Comp.Sc. Advisor
2. Dr. Eng. Darlis Herumurti, S.Kom., M.Kom Examiner
3. Hadziq Fabroyir, S.Kom., Ph.D. Examiner

SURABAYA

July, 2025

[Halaman ini sengaja dikosongkan]

PERNYATAAN ORISINALITAS

Yang bertanda tangan di bawah ini:

Nama mahasiswa / NRP	:	Muhammad Daffa' Ashdaqfillah / 5025211015
Program studi	:	S-1 Teknik Informatika
Dosen Pembimbing / NIP	:	Dr. Anny Yuniarti, S.Kom., M.Comp.Sc. / 198106222005012002

dengan ini menyatakan bahwa Tugas Akhir dengan judul "Pengembangan *Plugin* Blender untuk Pengendalian *Servo* Robot Humanoid Berbasis Dynamixel SDK" 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, 11 Juli 2025

Mengetahui Dosen Pembimbing	Mahasiswa
Dr. Anny Yuniarti, S.Kom., M.Comp.Sc.	Muhammad Daffa' Ashdaqfillah
NIP. 198106222005012002	NRP. 5025211015

[Halaman ini sengaja dikosongkan]

STATEMENT OF ORIGINALITY

The undersigned below:

Name of student / NRP	:	Muhammad Daffa' Ashdaqfillah / 5025211015
Department	:	Informatics Electics - ITS
Advisor / NIP	:	Dr. Anny Yuniarti, S.Kom., M.Comp.Sc. / 198106222005012002

Hereby declare that Final Project with the title of "Development of a Blender Plugin for Servo Control in Humanoid Robots Using the Dynamixel SDK" is the result of my own work, is original, and is written by following therules 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, 11 Juli 2025

Acknowledge
Advisor

Dr. Anny Yuniarti, S.Kom., M.Comp.Sc.
NIP. 198106222005012002

Student

Muhammad Daffa' Ashdaqfillah
NRP. 5025211015

[Halaman ini sengaja dikosongkan]

PERNYATAAN KODE ETIK
PENGGUNAAN AI GENERATIF

Code of Conduct Statement: Generative AI or AI-Assisted Usage

Saya yang bertanda tangan di bawah ini:
I, the undersigned:

Nama Mahasiswa / NRP *Full Name / Student ID*	:	Muhammad Daffa' Ashdaqfillah / 5025211015
Program Studi *Study Program*	:	S-1 Teknik Informatika
Judul Tugas Akhir *Final Project Title*	:	Pengembangan *Plugin* Blender untuk Pengendalian *Servo* Robot Humanoid Berbasis Dynamixel SDK

dengan ini menyatakan bahwa pada Tugas Akhir dengan judul di atas tersebut:
hereby declare that in the Final Project with the above title:

No.	Pernyataan *Statement*	(☑)
1	Saya hanya menggunakan AI generatif sebagai alat bantu untuk memperbaiki tata bahasa. AI generatif tidak digunakan untuk membuat isi Tugas Akhir. *I only used generative AI as a tool to improve the readability or language of the text in my Final Project. It was not used to generate a complete text of my work.*	☑
2	Saya telah memeriksa dan/atau memperbaiki seluruh bagian dari Tugas Akhir saya yang dibantu oleh AI generatif agar sesuai dengan baku mutu penulisan karya ilmiah. *I have reviewed and refined all aspects of my work that generative AI assists with, ensuring it adheres to the standards of academic writing.*	☑
3	Saya tidak menggunakan AI generatif untuk pembuatan data primer, grafik dan/atau tabel pada Tugas Akhir saya. *I did not use generative AI to generate primary data, figures, and/or tables in my work.*	☑
4	Saya telah memberikan atribusi/pengakuan terhadap alat AI yang digunakan, secara rinci pada suatu bagian pada lampiran. *I have acknowledged the use of generative AI in any part of the work in the specific appendix page.*	☑
5	Saya memastikan tidak ada plagiarisme, termasuk hal yang berasal dari penggunaan AI generatif. *I have ensured that there is no plagiarism issue in the work, including any parts generated by AI.*	☑

Surabaya, 11 Juli 2025

Mahasiswa

Muhammad Daffa' Ashdaqfillah
NRP. 5025211015

[Halaman ini sengaja dikosongkan]

ABSTRAK

Pengembangan *Plugin* Blender untuk Pengendalian *Servo* Robot Humanoid Berbasis Dynamixel SDK

Nama Mahasiswa / NRP	**:**	**Muhammad Daffa' Ashdaqfillah / 5025211015**
Departemen	**:**	**Teknik Informatika FTIRS - ITS**
Dosen Pembimbing	**:**	**Dr. Anny Yuniarti, S.Kom., M.Comp.Sc.**

Abstrak

Pengembangan robot humanoid memerlukan metode yang efektif untuk menciptakan dan mengendalikan gerakan yang kompleks. Penelitian ini berhasil mengembangkan *plugin* untuk Blender yang memungkinkan kontrol robot humanoid 17-DOF secara *real-time* menggunakan Dynamixel SDK melalui U2D2. *Plugin* ini memanfaatkan fitur animasi Blender untuk merancang gerakan robot secara visual, kemudian menerjemahkannya menjadi perintah *servo* secara *real-time* melalui komunikasi USB. Pendekatan ini bertujuan menyederhanakan proses pemrograman gerakan robot dengan memanfaatkan antarmuka animasi 3D yang *familiar*. Metodologi yang digunakan meliputi studi literatur, pemodelan 3D robot dan pembuatan animasi di Blender, pengembangan *plugin* dengan *Python* API Blender, integrasi Dynamixel SDK, serta pengujian sistem. Hasil implementasi menunjukkan bahwa robot yang terdiri dari 15 *servo* XL320 dan 2 *servo* MX28, berhasil dikendalikan dengan latensi komunikasi rata-rata sebesar 0.43 detik. Evaluasi pengujian *usability* menunjukkan tingkat kepuasan pengguna sebesar 92% pada aspek kemudahan instalasi, efisiensi animasi, dan desain antarmuka. Hasil penelitian membuktikan bahwa pendekatan ini efektif dalam menyederhanakan proses pemrograman gerakan robot humanoid dan memiliki potensi untuk diterapkan dalam pengembangan sistem robotik yang lebih kompleks di masa depan.

Kata kunci: *Plugin* Blender, Robot Humanoid, Dynamixel SDK, U2D2, Kontrol *Servo*.

[Halaman ini sengaja dikosongkan]

ABSTRACT

Development of a Blender Plugin for Servo Control in Humanoid Robots Using the Dynamixel SDK

Student Name / NRP	**:**	**Muhammad Daffa' Ashdaqfillah / 5025211015**
Department	**:**	**Informatics ELECTICS - ITS**
Advisor	**:**	**Dr. Anny Yuniarti, S.Kom., M.Comp.Sc.**

Abstract

Humanoid robot *development requires effective methods for creating and controlling complex movements. This research successfully developed a plugin for Blender that enables real-time control of a 17-DOF humanoid* robot *using Dynamixel SDK through U2D2. The plugin utilizes Blender's animation features to design* robot *movements visually, then translates them into servo commands in real-time via USB communication. This approach aims to simplify the* robot *movement programming process by leveraging familiar 3D animation interfaces. The methodology includes literature study, 3D* robot *modeling and animation creation in Blender, plugin development with Blender's Python API, Dynamixel SDK integration, and system testing. Implementation results show that the* robot *consisting of 15 XL320 servos and 2 MX28 servos was successfully controlled with average communication latency of 0.43 seconds. Usability testing evaluation shows 92% user satisfaction in installation ease, animation efficiency, and interface design aspects. The research results prove that this approach is effective in simplifying the humanoid* robot *movement programming process and has potential for application in more complex robotic system development in the future*.

Keywords: ***Blender Plugin, Humanoid Robot, Dynamixel SDK, U2D2, Servo Control.***

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

Assalamu'alaikum warahmatullahi wabarakatuh

Puji syukur kehadirat Allah SWT atas rahmat dan karunia-Nya, penulis dapat menyelesaikan Tugas Akhir ini dengan judul **"Pengembangan *Plugin* Blender untuk Pengendalian *Servo* Robot Humanoid Berbasis Dynamixel SDK "** sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer di Institut Teknologi Sepuluh Nopember.

Penulisan Tugas Akhir ini tidak akan terlaksana dengan baik tanpa bimbingan, dukungan, dan motivasi dari berbagai pihak. Oleh karena itu, penulis ingin menyampaikan ucapan terima kasih yang sebesar-besarnya kepada:

1. **Allah SWT,** atas limpahan rahmat dan petunjuk-Nya yang telah mengiringi setiap langkah kehidupan, termasuk selama menjalani masa perkuliahan.
2. **Ibu Dr. Anny Yuniarti, S.Kom., M.Comp.Sc.** atas bimbingan, saran, serta kesabaran dalam mengarahkan penulis selama proses penyusunan Tugas Akhir ini.
3. **Bapak/Ibu Penguji Sidang** yang telah memberikan masukan, kritik, dan saran konstruktif demi perbaikan karya ini.
4. **Seluruh Karyawan** dan **Dosen Informatika ITS** yang telah memberikan ilmunya selama penulis berkuliah di Informatika ITS.
5. **Kedua orang tua penulis** yang selalu memberikan dukungan penuh dan doa, sehingga penulis dapat menyelesaikan masa perkuliahan ini dengan baik.
6. **Tim Virose ITS** yang telah menjadi wadah bagi penulis untuk belajar melakukan riset. Ucapan terima kasih secara khusus penulis sampaikan kepada Mas Faris, Niko, Nandini, Bisma, dan Faiq.
7. **Rekan-rekan seperjuangan** yang selalu memberikan semangat dan dukungan moral. Ucapan terima kasih secara khusus penulis sampaikan kepada Naufal, Java, Ammar, Aziz, Zuhal, Gagaz, Wisnu, dan Hanun.
8. Serta kepada semua pihak yang tidak dapat disebutkan di sini, namun telah memberikan bantuan kepada penulis dalam proses penyelesaian Tugas Akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih jauh dari sempurna. Oleh karena itu, penulis mengharapkan kritik dan saran yang membangun untuk perbaikan di masa mendatang. Semoga penelitian ini dapat bermanfaat bagi perkembangan ilmu pengetahuan dan teknologi.

Wassalamu'alaikum warahmatullahi wabarakatuh

Hormat saya,
Muhammad Daffa' Ashdaqfillah

[Halaman ini sengaja dikosongkan]

DAFTAR ISI

DAFTAR GAMBAR

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

DAFTAR KODE SUMBER

[Halaman ini sengaja dikosongkan]

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Robot humanoid merupakan representasi teknologi robotika yang paling mendekati bentuk dan kemampuan gerak manusia. Pengembangan robot humanoid terus berkembang pesat, didorong oleh potensi aplikasinya di berbagai bidang seperti industri, layanan, hingga hiburan. Namun, salah satu tantangan utama dalam pengembangan robot humanoid adalah menciptakan gerakan yang realistis dan terkoordinasi, terutama untuk robot dengan derajat kebebasan (*Degrees of Freedom* - DOF) yang kompleks, yang memerlukan pemrograman aktuator yang kompleks (Niiyama, 2022).

Proses pemrograman gerakan robot humanoid secara tradisional seringkali melibatkan penulisan kode yang kompleks dan memakan waktu. Pembuat gerakan harus menerjemahkan ide gerakan menjadi serangkaian perintah numerik untuk setiap sendi (*servo*) robot. Proses ini tidak hanya rumit tetapi juga kurang intuitif, terutama bagi mereka yang terbiasa bekerja dengan perangkat lunak simulasi 3D seperti AutoCAD, Inventor, SolidWorks, Maya, dan Blender. Perangkat lunak tersebut menawarkan antarmuka visual yang memudahkan perancangan dan simulasi model 3D.

Di antara perangkat lunak tersebut, Blender memiliki keunggulan khusus dalam aspek animasi dan pergerakan model. Sebagai perangkat lunak *open-source* yang populer di kalangan animator dan desainer 3D, Blender menyediakan sistem *rigging* dan animasi yang mudah, *workflow* yang fleksibel, serta kemampuan *scripting* yang bisa mengotomatisasi dan menyesuaikan proses animasi karakter humanoid (Subree, 2023).

Kesenjangan antara kemudahan merancang gerakan di lingkungan digital dengan kompleksitas implementasinya pada perangkat keras robot fisik menjadi latar belakang penelitian ini. Pemanfaatan Blender sebagai antarmuka untuk mengendalikan robot dapat menyederhanakan alur kerja dalam pemodelan, simulasi, dan implementasi gerakan riil pada robot. Untuk menjembatani kesenjangan ini, diperlukan komunikasi *real-time* untuk pengembangan dan pengujian gerakan pada robot.

Untuk komunikasi *real-time*, U2D2 (*USB to Dynamixel Adaptor*) menjadi solusi yang ideal karena memungkinkan komunikasi langsung antara komputer dengan *servo* Dynamixel tanpa memerlukan mikrokontroler tambahan sebagai perantara. U2D2 menyediakan konversi USB ke TTL yang diperlukan untuk protokol komunikasi Dynamixel, sehingga *plugin* Blender dapat langsung mengendalikan *servo* robot selama proses pengembangan dan pengujian gerakan (Robotis, 2025).

Fokus penelitian ini adalah pada pengembangan sistem kontrol *upper body* robot humanoid yang terdiri dari 17 derajat kebebasan (17-DOF). Pemilihan *upper body* sebagai target implementasi memungkinkan fokus yang lebih spesifik pada gerakan-gerakan ekspresif seperti lengan, bahu, dan torso yang sering digunakan dalam interaksi manusia-robot. Konfigurasi 17-DOF ini terdiri dari 15 *servo* XL320 untuk area lengan dan kepala, dan 2 *servo* MX28 untuk area dada dan pinggang yang memerlukan torsi lebih tinggi untuk stabilitas postur. Selain itu, *lower body* tidak dijadikan fokus dalam penelitian ini karena gerakannya cenderung lebih

dinamis dan kompleks, terutama saat berjalan, yang umumnya membutuhkan integrasi dengan sensor seperti *loadcell* untuk mengatur keseimbangan secara *real-time*.

Penelitian ini mengusulkan pengembangan sebuah *plugin* Blender yang dapat berkomunikasi langsung dengan *servo* robot melalui U2D2 dan Dynamixel SDK. *Plugin* ini diharapkan dapat menerjemahkan data animasi (seperti rotasi sendi pada model 3D) di Blender menjadi perintah kontrol untuk *servo* motor secara *real-time*. *Plugin* ini juga menyediakan fitur *import* dan *export* animasi untuk memungkinkan penggunaan gerakan yang telah dibuat pada sistem lain seperti mikrokontroler *standalone*, sehingga robot dapat menjalankan gerakan tanpa harus terhubung ke komputer yang menjalankan Blender.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam Tugas Akhir ini adalah sebagai berikut,

1. Bagaimana cara merancang dan mengimplementasikan *plugin* Blender yang mampu menggerakkan model robot humanoid 3D berdasarkan data animasi dengan komunikasi *real-time* ke *hardware servo*?
2. Bagaimana cara menerjemahkan data rotasi dari Blender menjadi nilai posisi *servo* yang akurat untuk menggerakkan 15 *servo* XL320 dan 2 *servo* MX28 pada robot humanoid *upper body*?
3. Bagaimana cara mengembangkan fitur *import* dan *export* data animasi pada *plugin* Blender untuk memungkinkan penggunaan gerakan yang dibuat pada sistem lain?
4. Bagaimana cara mengevaluasi *usability plugin* melalui pengujian dengan pengguna untuk memastikan kemudahan penggunaan dan efektivitas?

1.3 Batasan Masalah

Batasan masalah dalam Tugas Akhir ini adalah sebagai berikut,

1. Robot humanoid yang digunakan sebagai *platform* uji terbatas pada bagian *upper body* dengan 17 derajat kebebasan (17-DOF) menggunakan 15 *servo* XL320 dan 2 *servo* MX28. Pemilihan *upper body* dilakukan karena gerakannya lebih stabil dan tidak memerlukan sistem keseimbangan kompleks seperti *lower body* yang membutuhkan integrasi sensor tambahan.
2. Komunikasi *hardware* menggunakan U2D2 (*USB to Dynamixel Adaptor*) untuk koneksi langsung antara komputer dan *servo* robot.
3. Pengembangan *plugin* Blender menggunakan bahasa pemrograman Python dengan implementasi Blender Python API (bpy) dan Dynamixel SDK untuk komunikasi *servo*.

1.4 Tujuan

Tujuan dari pembuatan Tugas Akhir ini adalah sebagai berikut,

1. Mengembangkan sebuah *plugin* untuk perangkat lunak Blender yang dapat membaca data animasi dari model robot humanoid dan berkomunikasi secara *real-time* dengan *hardware servo*.
2. Mengembangkan algoritma penerjemahan data rotasi Blender menjadi perintah kontrol posisi *servo* yang konsisten dan akurat untuk kedua jenis *servo* XL320 dan MX28.
3. Mengembangkan fitur *import* dan *export* data animasi pada *plugin* Blender untuk penggunaan gerakan pada sistem *standalone*.
4. Melakukan evaluasi *usability plugin* melalui pengujian dengan pengguna untuk mengukur kemudahan penggunaan dan efektivitas dalam *workflow* robotika.

1.5 Manfaat

Hasil dari pengerjaan Tugas Akhir ini memberikan manfaat bagi akademisi dan peneliti robotika, karena penelitian ini menyediakan metode alternatif yang lebih intuitif dan efisien untuk pemrograman dan pengujian gerakan robot humanoid, khususnya yang memiliki DOF tinggi. Selain itu, penelitian ini juga dapat menjadi referensi untuk pengembangan sistem kontrol robot berbasis software animasi.

Manfaat lain dari hasil pengerjaan ini dirasakan oleh para animator dan desainer, karena penelitian ini membuka kemungkinan untuk terlibat langsung dalam proses pengembangan gerakan robot tanpa memerlukan pengetahuan pemrograman *hardware* yang mendalam. Dengan demikian, penelitian ini turut menjembatani kesenjangan antara animasi digital dan robotika fisik melalui pendekatan yang lebih visual dan intuitif.

Selain itu, sistem yang dikembangkan juga memberikan manfaat bagi pengembang aplikasi robotika dan IoT yang melibatkan penggunaan *servo* Dynamixel. Dengan memanfaatkan integrasi antara Blender sebagai antarmuka animasi dan komunikasi langsung melalui U2D2, pengguna dapat merancang dan menguji gerakan aktuator secara *real-time* tanpa kompleksitas pemrograman mikrokontroler. Pendekatan ini mempercepat proses iterasi, mengurangi kompleksitas teknis, dan memberikan solusi yang dapat diperluas untuk berbagai aplikasi robotika seperti robot *service*, sistem demonstrasi, atau *platform* penelitian interaktif.

[Halaman ini sengaja dikosongkan]

BAB 2
TINJAUAN PUSTAKA

2.1 Hasil Penelitian Terdahulu

1. Pada penelitian berjudul "*ROS-Based Humanoid Robot Pose Control System Design*" yang dilakukan oleh Feng, dibahas penggunaan *Robot Operating System* (ROS) sebagai *framework* utama untuk mengendalikan pose robot humanoid. Penelitian ini mengimplementasikan arsitektur ROS terdistribusi yang berkomunikasi untuk memastikan prosedur sekuensial yang benar dan pertukaran informasi lengkap melalui topologi *peer-to-peer*. Sistem perangkat lunak robot humanoid dirancang dalam lingkungan ROS berbasis Linux untuk mengintegrasikan sistem perangkat lunak dan perangkat keras melalui ROS. Kelebihan pendekatan ini adalah kemampuan simulasi Gazebo yang dapat menampilkan perilaku *cyber-physical* robot humanoid dalam eksperimen nyata. Namun, penelitian ini memiliki keterbatasan terutama dalam kompleksitas implementasi yang memerlukan pengetahuan mendalam tentang ROS, kesulitan integrasi dengan *software* animasi, serta *learning curve* yang curam bagi pengguna baru. Sistem ini masih memerlukan antarmuka yang lebih *user-friendly* dan optimasi pada protokol komunikasi untuk mengurangi latensi pada robot dengan jumlah DOF yang lebih tinggi (Feng *et al.,* 2022).
2. Penelitian yang berjudul "*Dancing Humanoid Robots Lab Demonstration for the First Year Engineering Students*" oleh Jaksic, membahas penggunaan R+ Motion untuk memprogram gerakan tari pada robot humanoid. Penelitian ini menggunakan tiga kit robotik Robotis Premium untuk merakit tiga robot humanoid (masing-masing dengan 18 *Degrees of Freedom*) dan menciptakan koreografi tari robotik tiga menit. R+ Motion digunakan untuk membuat, memprogram, dan mengirimkan gerakan tari tersebut. Kelebihan utama dari R+ Motion adalah kemudahan penggunaan yang memungkinkan bahkan mahasiswa tahun pertama untuk menciptakan gerakan kompleks. Hasil survei menunjukkan bahwa pengalaman ini menarik dan meningkatkan motivasi mahasiswa untuk mempelajari teknik robotika. Meskipun demikian, *software* ini memiliki keterbatasan dalam hal ketergantungan pada *hardware* Robotis dan fitur animasi yang terbatas dibanding *software* profesional (Jaksic *et al.,* 2022).
3. Jurnal berjudul "*Blender for Robotics: Integration into the Leuven Paradigm for Robot Task Specification and Human Motion Estimation*" oleh Buys, membahas integrasi program animasi komputer *open source* Blender ke dalam penelitian robotika. Penelitian ini menunjukkan bagaimana sejumlah laboratorium robotika bekerja sama dalam proyek "*Blender for Robotics*" untuk mengintegrasikan Blender, awalnya sebagai alat visualisasi, namun secara bertahap berkembang menjadi komponen lengkap dalam lingkungan simulasi dan pemrograman robot. Laboratorium di Leuven berfokus pada peningkatan dukungan Blender untuk spesifikasi tugas dan kontrol tugas robot berbasis sensor yang kompleks, di mana gerakan manusia yang berinteraksi dengan tugas juga harus dilacak. Keunggulan pendekatan ini adalah kemampuan untuk memanfaatkan *tools* animasi profesional Blender untuk robotika dan *workflow* yang familiar bagi animator. Namun, implementasi awal ini memiliki keterbatasan dalam integrasi langsung dengan *hardware* robot dan masih memerlukan pengembangan tambahan untuk aplikasi *real-time* (Buys *et al.,* 2021).

2.2 Dasar Teori

Subbab ini berisi penjelasan tentang landasan teori yang mendasari penelitian tugas akhir ini, yang diperoleh dari referensi yang relevan dengan topik penelitian.

2.2.1 *Plugin* Blender

Blender adalah perangkat lunak grafika komputer 3D *open-source* yang populer dan serbaguna, digunakan untuk pemodelan, animasi, *rendering*, dan *post-production*. Salah satu keunggulan Blender adalah *Application Programming Interface* (API) berbasis Python (bpy) yang dapat digunakan pengembang untuk memperluas fungsionalitasnya melalui pembuatan *script* dan *plugin* (*Add-ons*). *Plugin* adalah program tambahan yang terintegrasi ke dalam antarmuka Blender, menambahkan *panel*, operator, atau fitur baru (Santos *et al.,* 2024). Dalam konteks robotika, *plugin* Blender bisa dipakai untuk menjembatani lingkungan simulasi visual dengan perangkat keras fisik untuk perancangan, pengujian, dan kontrol gerakan robot melalui antarmuka grafis. API *Python* Blender dapat dimanfaatkan untuk membuat *plugin* yang membaca data animasi dan mengirimkannya ke sistem kontrol *servo* melalui protokol komunikasi yang sesuai.

2.2.2 Pemodelan 3D pada Blender

Pemodelan 3D di Blender adalah proses pembuatan objek digital yang dapat terdiri dari beberapa bagian terpisah. Untuk model robot humanoid, setiap bagian seperti lengan, torso, dan kepala dapat dimodelkan secara terpisah sebagai *mesh* individual. Hal ini memudahkan dalam pengorganisasian model dan memastikan setiap bagian dapat bergerak secara independen sesuai dengan mekanisme robot fisik yang sebenarnya (Abdillah, 2021).

2.2.3 *Rigging* dan *Bone* pada Blender

Rigging adalah proses pembuatan struktur kerangka (skeleton) digital untuk menganimasikan model 3D. Di Blender, kerangka ini dibangun menggunakan objek khusus yang disebut *Armature*, yang terdiri dari serangkaian *Bones* (tulang). *Bones* ini dihubungkan secara hierarkis (*parent-child relationship*) untuk meniru struktur sendi pada mekanisme robot. Setiap *bone* memiliki transformasi (lokasi, rotasi, skala) yang bisa dianimasikan. Untuk model robot, setiap bagian yang telah dimodelkan sebelumnya dapat dikaitkan (*parenting*) ke *bone* yang sesuai pada *armature*, sehingga pergerakan *bone* akan menggerakkan bagian robot tersebut secara langsung (Aburumman, 2017).

2.2.4 *Skinning* pada Blender

Skinning adalah proses menghubungkan *mesh* model 3D dengan *armature* (kerangka) sehingga *mesh* tersebut dapat bergerak mengikuti pergerakan *bone*. Dalam konteks model robot, proses *skinning* berbeda dengan *skinning* pada karakter organik. Untuk robot humanoid, proses *skinning* lebih sederhana karena setiap *part* robot merupakan objek solid yang bergerak sebagai satu kesatuan. Tidak seperti model karakter yang memerlukan deformasi *mesh* yang halus, *part* robot cukup di-*parent* langsung ke *bone* yang sesuai menggunakan *rigid parenting*. Metode ini menjamin bahwa setiap *part* robot akan bergerak secara presisi mengikuti *bone* pengontrolnya, tanpa deformasi yang tidak diinginkan. Pendekatan *rigid* parenting ini sangat sesuai untuk robot karena mencerminkan karakteristik mekanis dari *joint servo* yang sesungguhnya, di mana setiap *part* bergerak sebagai benda solid yang terhubung oleh sendi (Adinda *et al*., 2022).

2.2.5 Animasi Blender

Animasi di Blender, khususnya untuk karakter atau robot yang di-*rig*, biasanya dilakukan pakai teknik *keyframing*. Animator menentukan pose kunci (rotasi, posisi, atau skala *bones*) pada

titik-titik waktu tertentu (*keyframes*) di *timeline*. Blender kemudian otomatis menginterpolasi gerakan di antara *keyframes* tersebut untuk menciptakan ilusi gerakan yang mulus. Data animasi ini, khususnya nilai rotasi setiap *bone* pada setiap *frame*, bisa diakses melalui API *Python* Blender. Nilai rotasi *bone* yang relevan pada setiap *frame* animasi bisa dipakai sebagai representasi digital dari gerakan robot yang diinginkan, dan bisa diterjemahkan jadi perintah untuk *servo* fisik (Abdillah, 2021).

2.2.6 U2D2 (*USB to Dynamixel Adaptor*)

U2D2 adalah adaptor komunikasi yang dikembangkan oleh Robotis untuk menghubungkan komputer dengan *servo* motor Dynamixel melalui koneksi USB seperti yang terlihat pada Gambar 2.1. U2D2 berfungsi sebagai konverter protokol dari USB ke TTL (*Transistor-Transistor Logic*) yang digunakan oleh *servo* Dynamixel untuk komunikasi. Adaptor ini mendukung berbagai *baudrate* dari 9600 bps hingga 4.5 Mbps, memungkinkan komunikasi *high-speed* dengan banyak *servo* secara bersamaan. U2D2 dilengkapi dengan *power supply input* untuk menyediakan daya ke *servo chain* dan indikator LED untuk status komunikasi. Keunggulan U2D2 dibandingkan adaptor komunikasi lain adalah dukungan *native* untuk protokol Dynamixel 1.0 dan 2.0 serta kompatibilitas dengan semua seri *servo* Dynamixel (Robotis, 2025).

Gambar 2.1 Komponen U2D2

2.2.7 *Servo* Motor Dynamixel (XL-320 dan MX-64)

Servo motor Dynamixel adalah aktuator cerdas yang dikembangkan oleh Robotis dengan fitur komunikasi digital, *feedback* posisi, dan kontrol presisi tinggi, contohnya seperti yang terlihat pada Gambar 2.2. XL320 adalah *servo compact* dengan torsi 0.39 N·m pada 7.4V, resolusi posisi 1024 *steps* (sekitar 0.35° per *step*), dan kecepatan maksimum 114 RPM. *Servo* ini menggunakan protokol Dynamixel 2.0 dengan komunikasi TTL dan cocok untuk aplikasi yang memerlukan *servo* ringan dan ekonomis seperti *joint arm* dan *finger* robot humanoid. MX28 adalah *servo* dengan torsi lebih tinggi 2.5 N·m pada 12V, resolusi posisi 4096 *steps* (sekitar 0.088° per *step*), dan kecepatan maksimum 55 RPM. MX28 menggunakan komunikasi TTL dan cocok untuk aplikasi yang memerlukan torsi tinggi seperti *joint* utama pada torso dan *hip* robot humanoid. Kedua *servo* memiliki *built-in controller* yang dapat menerima perintah posisi, kecepatan, dan torsi, serta memberikan *feedback real-time* tentang posisi aktual, kecepatan, *load*, dan status *error*. Fitur *networking daisy-chain* memungkinkan banyak *servo* terhubung dalam satu *bus* komunikasi dengan ID unik untuk setiap *servo* (Robotis, 2025).

Gambar 2.2 *Servo* Dynamixel XL-320 dan MX-28

Meskipun *servo* Dynamixel memiliki keunggulan dalam fitur komunikasi digital, pengaturan posisi presisi, dan kemudahan integrasi, keterbatasan tetap ada, terutama dari sisi biaya dan kebutuhan daya. Dibandingkan dengan jenis *servo* konvensional seperti MG995 (*servo analog* dengan kontrol PWM standar), servo Dynamixel memiliki harga yang lebih tinggi dan memerlukan *adaptor* khusus seperti U2D2 untuk komunikasi. Servo konvensional umumnya tidak mendukung *feedback* posisi dan kontrol berbasis ID, namun lebih mudah digunakan pada sistem mikrokontroler sederhana. Selain itu, beberapa servo PWM tidak memiliki resolusi gerak setinggi Dynamixel dan tidak mendukung komunikasi bus, sehingga tidak ideal untuk sistem dengan banyak aktuator yang dikendalikan secara serentak. Keterbatasan inilah yang menjadi pertimbangan utama dalam pembatasan penelitian ini hanya pada servo Dynamixel, khususnya tipe XL-320 dan MX-28 (Robotis, 2025).

2.2.8 Dynamixel SDK

Dynamixel SDK (*Software Development Kit*) adalah pustaka perangkat lunak yang dikembangkan oleh Robotis untuk memfasilitasi komunikasi dan kontrol *servo* motor Dynamixel. SDK ini menyediakan API (*Application Programming Interface*) yang konsisten dan mudah digunakan untuk berbagai bahasa pemrograman termasuk C, C++, Python, Java, C#, MATLAB, dan LabVIEW.

Fitur utama Dynamixel SDK meliputi fungsi-fungsi untuk membaca dan menulis data ke *control table servo*, pengelolaan komunikasi *packet-based* dengan *error checking*, dukungan untuk komunikasi *bulk* dan *sync* yang efisien untuk mengontrol banyak *servo* secara bersamaan, serta kompatibilitas dengan berbagai jenis *servo* Dynamixel dari seri AX, DX, RX, EX, MX, XL, XM, XH, dan PRO.

Dalam konteks penelitian ini, Dynamixel SDK *Python* menjadi komponen kunci yang memungkinkan *plugin* Blender berkomunikasi langsung dengan *servo* robot melalui *Bus Servo Adapter*. Fungsi-fungsi utama yang digunakan meliputi `PortHandler` untuk manajemen koneksi serial, `PacketHandler` untuk pengelolaan protokol komunikasi, `GroupSyncWrite` untuk pengiriman perintah ke banyak *servo* secara efisien, dan berbagai fungsi *read*/*write* untuk kontrol posisi, kecepatan, dan parameter *servo* lainnya. Penggunaan SDK ini memastikan komunikasi yang andal dan berkinerja tinggi dengan lantesi minimal, yang sangat penting untuk aplikasi kontrol *real-time* (Robotis, 2025).

BAB 3 METODOLOGI

3.1 Metode yang digunakan

3.1.1 Studi Literatur

Tahap studi literatur dimulai dengan mengkaji penelitian-penelitian terdahulu yang berkaitan dengan kontrol robot menggunakan perangkat lunak animasi. Kajian ini mencakup analisis berbagai pendekatan yang telah dilakukan, evaluasi kelebihan dan kekurangan masing-masing metode, serta identifikasi peluang pengembangan yang dapat dilakukan dalam penelitian ini.

Aspek teknis protokol komunikasi Dynamixel dipelajari mulai dari prinsip dasar komunikasi serial hingga teknik-teknik optimasi untuk mencapai *throughput* maksimal dengan latensi minimal. Pemahaman tentang parameter-parameter Dynamixel seperti *baudrate*, format *frame* data, dan mekanisme *flow control* menjadi fondasi penting untuk implementasi komunikasi serial yang efisien menggunakan U2D2 sebagai antarmuka komunikasi. Diperlukan juga pemahaman mengenai protokol Dynamixel versi 2.0 yang meliputi struktur paket instruksi, status, serta parameter-parameter kontrol untuk *servo* XL-320 dan MX-28

Dalam konteks pengembangan *plugin* Blender, studi difokuskan pada penguasaan API Python Blender (bpy). Hal ini meliputi pemahaman tentang sistem *add-on* Blender, mekanisme registrasi operator, teknik akses dan manipulasi data animasi serta *armature*, dan *best practices* dalam pengembangan *plugin*. Dokumentasi resmi Blender dan contoh-contoh implementasi *plugin* yang ada dijadikan referensi utama.

Studi tentang Dynamixel SDK juga penting karena SDK ini merupakan jembatan antara aplikasi Python dan *hardware servo*. Kajian meliputi dokumentasi resmi API Python, *best practices* untuk manajemen koneksi multi-*servo*, teknik optimasi untuk komunikasi *real-time*, dan implementasi *error handling*. Pemahaman tentang *control table* masing-masing jenis *servo* (XL-320 dan MX-28) diperlukan untuk mapping yang akurat antara parameter Blender dan *register servo*. Studi juga mencakup analisis performa GroupSyncWrite dibandingkan dengan *individual write commands*, serta strategi untuk meminimalkan latensi dalam aplikasi *real-time*.

3.1.2 Pemodelan 3D Robot Blender

Pemodelan 3D robot humanoid pada Blender merupakan tahap penting dalam penelitian ini. Proses ini terdiri dari beberapa tahap yang saling terkait untuk menciptakan model 3D yang akurat dan dapat digerakkan sesuai dengan karakteristik robot fisik.

3.1.2.1 Pemodelan Dasar

Pada tahap ini, model 3D robot humanoid dibuat dengan mengkombinasikan model *servo* yang tersedia secara *open source* dengan bagian-bagian penghubung yang dimodelkan khusus. Model *servo* XL-320 dan MX-28 diambil dari repositori CAD Robotis yang menyediakan model 3D akurat dari produk mereka. Penggunaan model *servo* asli ini memastikan dimensi, bentuk, dan detail mekanis yang presisi sesuai dengan *hardware* sebenarnya.

Bagian-bagian penghubung seperti *bracket*, *frame*, dan *casing* dimodelkan menggunakan teknik *mesh modeling* di Blender dengan skala yang disesuaikan. Dimensi *part* penghubung dibuat berdasarkan pengukuran fisik robot asli namun dengan rasio skala yang lebih kecil untuk memudahkan visualisasi dan animasi. Pemodelan menggunakan primitif dasar seperti kubus dan silinder yang dimodifikasi untuk membentuk struktur yang diperlukan, dengan mempertahankan proporsi dan jarak antar *servo* yang sesuai dengan robot fisik.

Setiap bagian penghubung dimodelkan secara terpisah dengan memperhatikan detail sambungan dan kompatibilitas antar komponen. Model dibuat realistis namun tetap ringan untuk diproses dengan mempertimbangkan aspek fungsional dan estetika.

3.1.2.2 Pengorganisasian Objek

Setelah pemodelan dasar selesai, objek-objek diatur dalam hierarki yang terstruktur. Pengorganisasian ini penting untuk memudahkan proses *rigging* dan animasi. Bagian-bagian yang berkaitan secara fungsional dikelompokkan dalam *collection* terpisah, misalnya *collection* untuk kepala, badan, lengan kanan, lengan kiri, kaki kanan, dan kaki kiri.

Penamaan objek dilakukan dengan konvensi yang jelas dan konsisten, mengikuti struktur penomoran *servo* pada robot fisik. Misalnya, objek yang mewakili bagian yang digerakkan oleh *servo* XL-320 dengan ID-1 diberi nama "S_XL_1". Hal ini memudahkan proses pemetaan gerakan antara model 3D dan robot fisik nantinya.

3.1.2.3 *Rigging* (Pemasangan Kerangka)

Rigging adalah proses pembuatan struktur kerangka digital (*armature*) pada model 3D. Pada tahap ini, dibuat objek *armature* yang terdiri dari *bone-bone* yang mewakili sendi-sendi pada robot humanoid. Setiap *bone* ditempatkan persis pada lokasi *servo* pada model fisik.

Struktur *armature* dibuat dengan memperhatikan hierarki *parent-child* dari tulang-tulang tersebut. Misalnya, *bone* bahu menjadi *parent* dari *bone* lengan atas, dan *bone* lengan atas menjadi *parent* dari *bone* lengan bawah. Hal ini menjamin bahwa gerakan pada *bone parent* akan mempengaruhi *bone child*, sama seperti pada robot fisik.

Proses *rigging* ini mempertimbangkan juga jenis sendi pada robot fisik. Untuk sendi yang hanya berputar pada satu sumbu (seperti kebanyakan *servo*), batasan rotasi (*rotation constraint*) diterapkan pada *bone* yang sesuai sehingga *bone* hanya bisa berputar pada sumbu yang sama dengan *servo* fisik.

3.1.2.4 Penentuan Batasan Gerak (*Bone Constraint*)

Setelah struktur *armature* terbentuk, diterapkan batasan rotasi (*limit rotation constraint*) pada setiap *bone* untuk mensimulasikan karakteristik mekanis dari robot fisik. Batasan rotasi ini membatasi sudut perputaran *bone* agar sesuai dengan batas fisik *servo* motor yang digunakan.

Penerapan batasan rotasi ini penting untuk memastikan bahwa animasi yang dibuat pada model 3D benar-benar dapat diimplementasikan pada robot fisik tanpa risiko merusak *servo* motor. Setiap *bone* dibatasi rotasinya sesuai dengan posisi dan jenis *servo* yang digunakan, untuk *servo* XL-320 yang digunakan pada *joint* lengan dan kepala dibatasi pada rentang -150° hingga +150°, sedangkan *servo* MX-28 yang digunakan pada *joint* dada dan pinggang dibatasi pada rentang -180° hingga +180° karena membutuhkan jangkauan rotasi yang lebih luas untuk gerakan badan. Selain itu, beberapa *joint* seperti pada bahu dan siku memiliki batasan tambahan untuk mencegah tabrakan antar bagian robot.

3.1.2.5 *Skinning* (Penggabungan *Mesh* dengan *Armature*)

Skinning adalah proses menghubungkan *mesh* (model 3D) dengan *armature* sehingga *mesh* dapat bergerak mengikuti gerakan *bone*. Pada model robot ini, proses *skinning* relatif sederhana karena setiap bagian robot (*mesh*) sudah terpisah dan solid.

Untuk model robot yang terdiri dari bagian-bagian terpisah ini, setiap bagian solid (seperti lengan atas atau telapak tangan) cukup dihubungkan langsung dengan *bone* yang sesuai. Ini

berbeda dengan model karakter organik yang memerlukan deformasi halus, karena bagian-bagian robot bergerak sebagai benda solid yang terpisah.

Proses penghubungan dilakukan dengan hati-hati terutama pada bagian-bagian yang bergerak relatif terhadap bagian lain, seperti pada sambungan antara badan dan lengan, untuk menghindari interseksi *mesh* saat dianimasikan.

3.1.2.6 Pengujian Gerakan Dasar

Setelah *rigging* dan *skinning* selesai, dilakukan pengujian gerakan dasar untuk memastikan bahwa model bergerak sebagaimana mestinya. Pengujian ini meliputi:

1. Rotasi setiap *bone* secara *individual* untuk memastikan *mesh* bergerak dengan benar.
2. Verifikasi batasan rotasi *bone* sudah sesuai dengan batas fisik *servo*.

Jika ditemukan masalah, dilakukan perbaikan pada *rigging*, *constraint*, atau *skinning* hingga model bergerak dengan baik.

3.1.2.7 Pembuatan Animasi

Tahap terakhir adalah pembuatan animasi gerakan robot. Animasi dibuat menggunakan teknik *keyframing*, di mana *keyframe* ditentukan pada *frame-frame* tertentu, dan Blender secara otomatis menginterpolasi gerakan di antara *keyframe* tersebut.

Beberapa jenis animasi yang dibuat untuk pengujian *plugin* meliputi:

1. Gerakan sederhana seperti mengangkat tangan.
2. Gerakan kompleks seperti menari.

Data *keyframe* inilah yang nantinya akan dibaca oleh *plugin* dan diterjemahkan menjadi perintah untuk *servo* motor pada robot fisik.

Dengan menyelesaikan seluruh tahap pemodelan 3D robot di Blender, dihasilkan model digital yang dapat mensimulasikan gerakan robot humanoid. Model ini menjadi dasar untuk pengembangan *plugin* yang menghubungkan Blender dengan komunikasi serial U2D2 untuk mengontrol robot fisik.

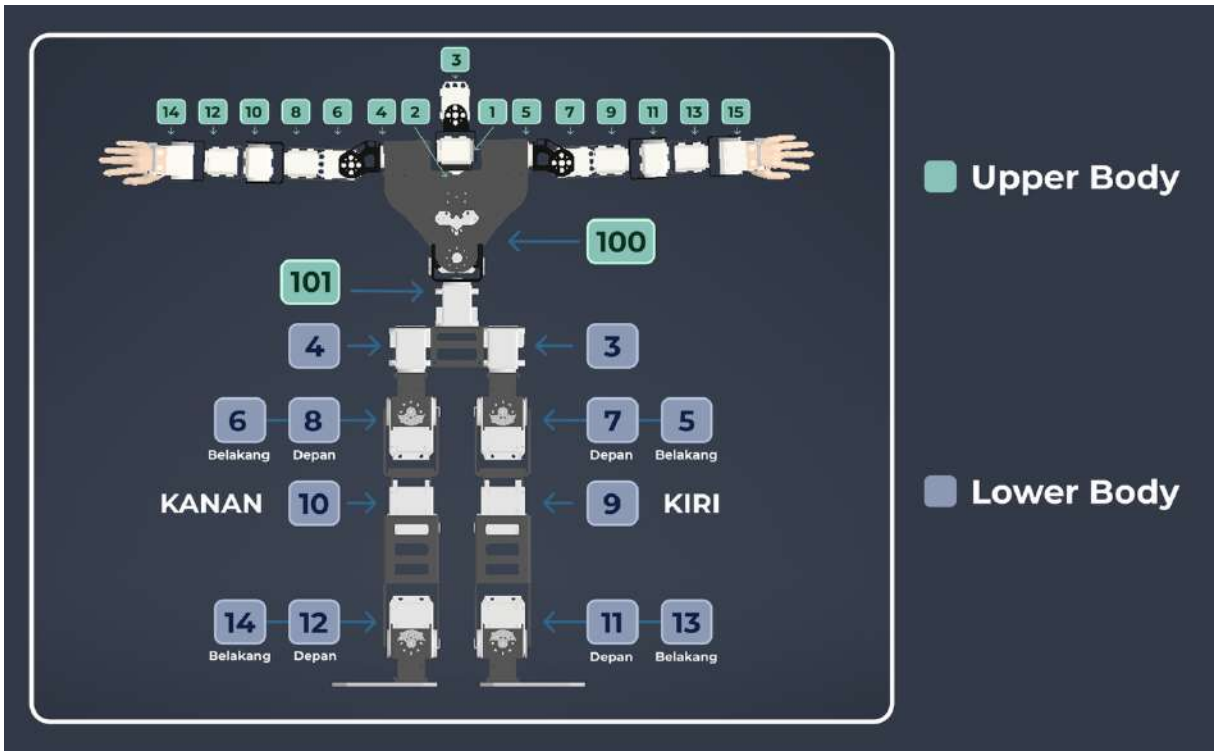
3.1.3 Konfigurasi *Hardware* Robot

Tahapan Konfigurasi dan Pengendalian *Servo* Motor meliputi pemetaan ID dan penempatan *servo* motor pada model 3D Blender, penentuan batasan sudut rotasi, dan kalibrasi *servo* motor.

3.1.3.1 Pemetaan ID dan Penempatan *Servo*

Robot humanoid yang digunakan dalam penelitian ini memiliki *servo* motor yang tersebar pada berbagai bagian tubuh robot. Setiap *servo* memiliki identitas unik (ID) yang digunakan untuk pengalamatan dalam komunikasi. Pemetaan ID ini harus sesuai dengan struktur *bone* pada model 3D Blender agar terjadi korelasi yang tepat antara *bone* di model 3D dengan *servo* di robot fisik.

Penempatan *servo* motor pada robot dikelompokkan berdasarkan bagian-bagian tubuh robot seperti yang terlihat pada Gambar 3.1.



Gambar 3.1 Pembagian ID *servo* pada robot

Robot ini menggunakan dua jenis *servo* untuk *upper body* robot:

1. XL-320 yang terdiri dari 15 *servo* untuk bagian atas robot (kepala dan lengan). *Servo* ini menggunakan ID yang dimulai dari 1-15 yang digunakan untuk *joint* dengan kebutuhan torsi rendah hingga menengah, meliputi area kepala, bahu, siku, dan pergelangan tangan. Setiap *servo* XL-320 memiliki resolusi posisi 1024 steps (0-1023) dengan rentang sudut *default* 300° (-150° hingga +150°) dan menggunakan komunikasi TTL.
2. MX-28 yang terdiri dari 2 *servo* untuk bagian dada dan pinggul robot. *Servo* MX-28 menggunakan ID yang dimulai dari 100-101 yang digunakan untuk *joint* yang memerlukan torsi tinggi. Setiap *servo* MX-28 memiliki resolusi posisi 4096 steps (0-4095) dengan rentang sudut 360° (-180° hingga +180°) dan menggunakan komunikasi TTL.

Pemetaan ID *servo* secara spesifik disesuaikan dengan *layout* fisik robot dan kebutuhan torsi masing-masing *joint*. Penggunaan rentang ID yang berbeda (1-15 untuk XL-320 dan 100-101 untuk MX-28) digunakan sebagai pencegah terjadinya konflik ID yang sama sehingga memungkinkan sistem *plugin* untuk secara otomatis mendeteksi jenis *servo* berdasarkan nama *bone* (XL1-XL15 dan MX1-MX14) dan menerapkan parameter komunikasi yang sesuai.

3.1.3.2 Penentuan Batasan Sudut

Penentuan batasan sudut pada setiap *servo* merupakan aspek yang sangat penting untuk menjaga keamanan robot dan mencegah kerusakan pada komponen mekanis robot. Batasan ini ditentukan berdasarkan beberapa pertimbangan:

1. Batasan Mekanis: Beberapa bagian robot memiliki batasan fisik yang tidak boleh dilewati oleh gerakan *servo*. Misalnya, siku manusia tidak dapat ditekuk ke belakang, sehingga *servo* siku robot juga diberi batasan serupa.
2. Pencegahan Tabrakan: Ketika dua bagian tubuh robot bergerak, ada kemungkinan terjadi tabrakan antar bagian. Untuk mencegah hal ini, batasan sudut ditetapkan untuk mencegah *servo* bergerak ke posisi yang menyebabkan tabrakan.
3. Kestabilan Postur: Untuk gerakan tertentu seperti berjalan atau menjaga keseimbangan, beberapa *servo* perlu dibatasi gerakannya untuk menjaga kestabilan robot.

Batasan sudut *servo* ini diimplementasikan dalam *plugin* Blender. Batasan sudut diterapkan pada *bone constraints* dengan parameter *rotation limit* sehingga animator tidak dapat membuat *keyframe* yang melewati batas aman.

Penentuan *rotation limit servo* secara lengkap ditampilkan dalam Tabel 3.1 dan 3.2 berikut:

Tabel 3.1 Pemetaan ID *Servo* XL-320

ID	Posisi	Sudut Minimal (°)	Sudut Maksimal (°)
1	Kepala *Yaw*	-100	6
2	Kepala *Pitch*	-70	70
3	Kepala *Roll*	-67	67
4	Bahu Kanan Atas	-150	150
5	Bahu Kiri Atas	-150	150
6	Bahu Kanan Bawah	-150	37
7	Bahu Kiri Bawah	-37	150
8	Siku Kanan Atas	-150	150
9	Siku Kiri Atas	-150	150
10	Siku Kanan Bawah	-24	122
11	Siku Kiri Bawah	122	24
12	Telapak Tangan Kanan Atas	-150	150
13	Telapak Tangan Kiri Atas	-150	150
14	Telapak Tangan Kanan Bawah	-105	105
15	Telapak Tangan Kiri Bawah	-105	105

Tabel 3.2 Pemetaan ID *Servo* MX-28

ID	Posisi	Sudut Minimal (°)	Sudut Maksimal (°)
1	Dada	-32	32
2	Pinggul	-180	180

3.1.3.3 Kalibrasi *Servo*

Proses kalibrasi *servo* merupakan tahap penting untuk memastikan akurasi gerakan robot. Kalibrasi dilakukan untuk mengatasi perbedaan antara nilai sudut teoritis dan gerakan aktual *servo*. Kalibrasi posisi tengah dilakukan dengan mengatur semua *servo* ke posisi *default* di derajat 0 dan menyesuaikan posisi fisik *servo* agar bagian robot berada pada posisi yang diharapkan.

3.1.3.4 Penyesuaian Nilai *Servo* dari Blender

Transformasi nilai rotasi dari model 3D Blender ke perintah *servo* merupakan aspek penting dalam sistem ini. Proses ini melibatkan beberapa tahap:

1. Extraksi Rotasi *Bone*: *Plugin* Blender mengekstrak nilai rotasi (dalam *radian*) dari setiap *bone* yang terkait dengan *servo*.
2. Konversi Satuan: Nilai rotasi dalam *radian* dikonversi ke derajat, yang merupakan satuan yang digunakan oleh *servo* Dynamixel.
3. Pemetaan Kerangka Koordinat: Sistem koordinat di Blender dan sistem koordinat *servo* Dynamixel bisa jadi berbeda, sehingga diperlukan transformasi koordinat untuk menghasilkan gerakan yang benar.
4. Normalisasi Nilai: Nilai sudut dinormalisasi ke dalam rentang yang sesuai dengan *servo* (misalnya 0-1023 untuk XL-320 dengan resolusi 0.29° per *unit*, atau 0-4095 untuk MX-28 dengan resolusi 0.088° per *unit*).
5. Pembatasan Nilai: Nilai akhir dibatasi sesuai dengan batasan sudut yang telah ditentukan dalam tabel untuk menjaga keamanan robot.

Dengan pemetaan dan konfigurasi ini, gerakan *bone* pada model 3D Blender dapat diterjemahkan secara akurat menjadi gerakan *servo* pada robot fisik, menghasilkan pergerakan yang mirip antara model digital dan robot nyata.

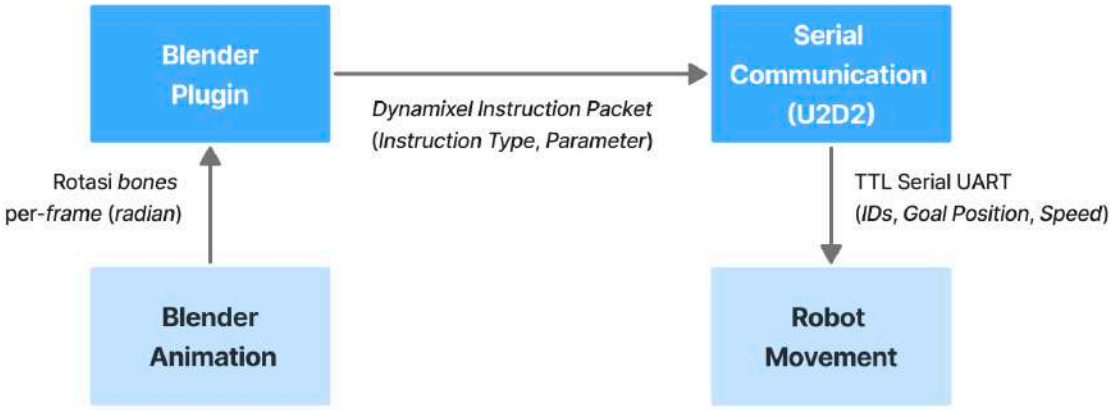
3.1.3.5 *Setup* Komunikasi U2D2

U2D2 (USB to Dynamixel Adaptor) berfungsi sebagai antarmuka komunikasi utama antara komputer yang menjalankan *plugin* Blender dengan jaringan *servo* robot. U2D2 menyediakan konversi protokol dari USB ke TTL untuk semua *servo* dalam robot (XL-320 dan MX-28 keduanya menggunakan TTL). Konfigurasi komunikasi menggunakan *baudrate* 1.000.000 bps untuk memastikan *throughput* tinggi yang diperlukan untuk kontrol *real-time* 17 *servo* secara bersamaan.

3.1.4 Alur Kerja Komunikasi Sistem

Sistem yang akan dikembangkan terdiri dari alur kerja seperti yang terlihat pada Gambar 3.2. Diagram ini menggambarkan proses sistem secara umum yang terdiri dari empat komponen utama yang saling terhubung dalam alur kerja yang sederhana. Sistem dirancang untuk memberikan komunikasi langsung antara animasi digital di Blender dengan pergerakan fisik robot humanoid melalui antarmuka komunikasi serial. Berikut penjelasan komponen-komponen utama dalam alur kerja sistem:

1. Blender *Animation*: Komponen ini merupakan sumber data gerakan yang dibuat oleh animator menggunakan teknik *keyframing* pada model 3D robot. Data rotasi *bone* dari animasi menjadi *input* utama untuk sistem.
2. Blender *Plugin*: *Plugin servo control* berfungsi sebagai penghubung antara data animasi Blender dengan sistem komunikasi. *Plugin* membaca data rotasi *Y-axis* dari setiap *bone* dan mengkonversinya menjadi format yang dapat dipahami oleh *servo* motor.
3. Serial *Communication* (U2D2): antarmuka komunikasi yang menerjemahkan perintah dari *plugin* Blender menjadi protokol Dynamixel yang dapat dipahami oleh *servo* motor. U2D2 berfungsi sebagai *bridge* antara USB komputer dengan jaringan *servo* TTL.
4. Robot *Movement*: Hasil akhir berupa gerakan fisik robot humanoid 17-DOF yang mengeksekusi animasi sesuai dengan data yang diterima melalui 15 *servo* XL-320 dan 2 *servo* MX-28.



Gambar 3.2 Diagram alur proses kendali *servo*

Alur kerja sistem dimulai dari pembuatan animasi di Blender, dibaca oleh *plugin*, dikirim melalui komunikasi serial U2D2, dan dieksekusi sebagai gerakan robot secara *real-time*. Alur kerja ini memungkinkan animator untuk melihat hasil animasi secara langsung pada robot fisik tanpa perlu menulis kode pemrograman *servo* yang kompleks.

3.1.4.1 Arsitektur *Plugin* Blender

Plugin servo control dibangun sebagai *add-on* yang terintegrasi dengan Blender untuk mengolah data animasi dan mendukung dua mode operasi: komunikasi *real-time* dengan robot melalui U2D2 dan *export* animasi ke format JSON untuk penggunaan *standalone*. *Plugin* mengimplementasikan arsitektur modular dengan komponen-komponen yang saling terkait untuk memastikan fungsionalitas yang stabil dan *user-friendly*.

- Struktur Komponen *Plugin*
  Plugin akan terdiri dari beberapa komponen utama:
  1. *Panel* UI (`ServoAnimationPanel`) untuk konfigurasi dan kontrol kedua mode operasi dengan sub bagian untuk *import* dan *export*, *real-time control*, *performance settings*, dan *servo offset configuration.*
  2. Sistem properti servo (`ServoControllerProperties`) untuk menyimpan pengaturan *servo*, komunikasi, dan *offset individual* dengan *persistent storage* dalam *scene properties.*
  3. *Handler* sistem untuk mendeteksi perubahan *frame* dan *trigger update servo* dalam mode *real-time.*
  4. Komponen pemrosesan data yang meliputi fungsi ekstraksi rotasi *Y-axis* dari *bone*, sistem konversi dari *radian* ke derajat, dan manajemen *cache* untuk optimasi performa.
  5. Modul komunikasi *real-time* menggunakan kelas `ServoController` yang mengimplementasikan Dynamixel SDK dengan `PortHandler`, `PacketHandler`, dan `GroupSyncWrite`.
  6. Sistem *import* dan *export* yang menyediakan operator untuk format JSON dengan *support keyframe-only export* dan konfigurasi interval *frame*.
- Alur Kerja *Plugin*
  1. Inisialisasi *Plugin*

Proses inisialisasi *plugin* dimulai dengan registrasi ke Blender melalui fungsi `register()` yang mendaftarkan semua kelas, properti, dan operator. *Plugin* menambahkan *panel servo control* pada kategori *Animation* di 3D Viewport dengan *watermark* "Tugas Akhir Daf2a" dan struktur UI yang terorganisir dalam bagian-bagian fungsional.

2. Konfigurasi Sistem
   Konfigurasi sistem dilakukan melalui `ServoControllerProperties` yang terdaftar di `bpy.types.Scene.servo_props`, dengan nilai-nilai *default* yang sudah disiapkan. Dalam mode *real-time*, sistem menggunakan konfigurasi seperti *port* USB, *baudrate* 1.000.000 bps, jumlah *servo* (15 *servo* XL-320 dan 2 *servo* MX-28), *update rate* 30 Hz, serta berbagai parameter lainnya. Sementara itu, pada mode *export*, digunakan parameter seperti rentang *frame*, jarak antar *frame* (*frame interval*), skala rotasi, dan opsi *keyframes-only*. *Plugin* ini juga dilengkapi dengan mekanisme *fallback* menggunakan `DYNAMIXEL_AVAILABLE`, yaitu sistem penanganan jika SDK tidak tersedia, agar *plugin* tetap dapat berjalan dengan fungsi terbatas tanpa menyebabkan kegagalan total.
3. Pengelolaan Aktuator *Servo*
   Aktuator pada sistem *servo* diatur menggunakan kelas `ServoController`, yang secara otomatis mengenali jenis *servo* berdasarkan nama *bone* yang digunakan. *Servo* XL-320 (XL1–XL15) menggunakan data posisi 2-*byte* dengan rentang nilai 0 hingga 1023 dan mendukung pengaturan kecepatan gerak (*moving speed*). Sementara itu, *servo* MX-28 (MX1–MX2) menggunakan data posisi 4-*byte* dengan rentang nilai 0 hingga 4095 dan mendukung pengaturan kecepatan profil (*profile velocity*).
   Plugin ini juga menyediakan pengaturan *offset* untuk setiap *servo* secara terpisah melalui dua daftar, yaitu offset servo xl dan offset servo mx. Setiap *servo* dapat disesuaikan dengan dua parameter: *multiply* (pengali) dan *add* (penambahan). Pengguna bisa mengaktifkan atau menonaktifkan *offset* ini secara individual, dan perubahan dapat langsung diterapkan secara *real-time*.
   Untuk mengurangi komunikasi yang tidak perlu, sistem menggunakan *cache* dan hanya akan mengirim perintah baru jika terjadi perubahan posisi yang melebihi batas tertentu. Hal ini membantu meningkatkan efisiensi dan responsivitas dalam pengendalian *servo*.
4. Pengolahan Data Animasi
   Saat animasi dijalankan dalam mode *real-time*, perlu diimplementasikan *handler* yang terdaftar di `bpy.app.handlers.frame_change_post`, handler ini akan dipanggil setiap kali terjadi perubahan *frame*. *Handler* ini membaca rotasi sumbu Y dari *bone* dengan nama XL1–XL15 dan MX1–MX2, mengubah nilai rotasinya dari radian ke derajat menggunakan fungsi `math.degrees()`, lalu menerapkan *offset* posisi global dan *offset* per *servo*. Setelah itu, hasil akhirnya disimpan dalam *chache* untuk dikirim secara sekaligus.
   Untuk meningkatkan performa, sistem menggunakan beberapa teknik optimasi, seperti melewati beberapa *frame* (*frame skipping*) sesuai kecepatan pembaruan (*update rate*), menerapkan ambang batas perubahan posisi (*position threshold*) agar gerakan kecil yang tidak signifikan tidak diproses, dan menjalankan proses pengiriman *servo* dalam *thread* terpisah, sehingga tidak mengganggu antarmuka

utama. Selain itu, sistem ini juga menghitung *velocity servo* secara otomatis berdasarkan jarak gerakan, sehingga gerakan terlihat lebih halus dan alami.

3.1.4.2 Komunikasi *Real-time* dan Format Data

Sistem komunikasi *real-time* dikendalikan melalui kelas `ServoController`, yang mengatur koneksi dengan perangkat U2D2 serta mengoordinasikan pergerakan banyak *servo* secara bersamaan. Komunikasi ini menggunakan Dynamixel SDK, dengan `PortHandler` yang dikonfigurasi untuk membaca USB *port* dan kecepatan komunikasi (*baudrate*) yang optimal. Protokol komunikasi yang digunakan adalah Dynamixel 2.0, yang ditangani oleh PacketHandler dan mendukung dua jenis *servo* dengan parameter yang berbeda.

1. Protokol Komunikasi Multi-*Servo*
   Sistem menggunakan pendekatan terpisah untuk masing-masing jenis *servo*. *Servo* XL-320 menggunakan `xl_sync_write` untuk posisi tujuan (*goal position*) dan `xl_speed_sync_write` untuk kecepatan gerak, dengan format data 2-*byte*. Sedangkan *servo* MX-28 menggunakan `mx_sync_write` untuk posisi tujuan dan `mx_velocity_sync_write` untuk kontrol kecepatan berbasis profil, dengan format data 4-byte. Pendekatan ini memungkinkan penyesuaian parameter secara optimal sesuai kebutuhan tiap jenis *servo*.
   Sistem juga memiliki mekanisme penanganan *error*, seperti pengecekan koneksi menggunakan `write1ByteTxRx` untuk mengaktifkan torsi *servo*, mekanisme percobaan ulang otomatis jika komunikasi gagal, serta penanganan *timeout* untuk menjaga ketahanan sistem. Jika ada *servo* yang tidak merespon, sistem tetap bisa berjalan dengan fungsi terbatas (*graceful degradation*). Informasi status komunikasi ditampilkan di antarmuka pengguna melalui indikator status dan jumlah *servo* yang berhasil terkoneksi.
2. Format Data *Import Export*
   Data animasi diekspor dalam format JSON dengan struktur bertingkat yang kompatibel dengan berbagai sistem eksternal. Format ini berbentuk {"NamaServo": [{"*frame*": angka, "*rotation*": derajat}]} dan tersusun secara konsisten (dimulai dari XL1–XL15, lalu MX1–MX2). Proses ekspor mendukung pengaturan seperti rentang *frame*, jarak antar *frame*, dan opsi hanya *keyframe*. File JSON yang dihasilkan berisi informasi lengkap animasi ke-17 *servo* dengan dua angka di belakang koma, untuk menjaga keseimbangan antara akurasi dan ukuran file. Format ini dirancang agar bisa digunakan di mikrokontroler, aplikasi web, maupun perangkat lunak robotika lainnya yang membutuhkan data animasi yang ringan dan mudah dibaca.
3. Optimasi Performa dan *Threading*
   Untuk menjaga agar antarmuka tetap responsif dan pergerakan *servo* tetap halus, sistem menggunakan *threading* di *background* yang berjalan terpisah dari proses utama Blender. Waktu pembaruan dapat diatur menggunakan `threading.Timer`, dan bisa dikonfigurasi sesuai kebutuhan (*update rate*). Sistem juga menggunakan manajemen *cache* untuk menghindari komunikasi berulang, dengan memeriksa apakah posisi *servo* berubah cukup jauh sebelum mengirim data baru.
   Selain itu, optimasi saat perubahan *frame* mencakup melewati *frame* tertentu saat playback cepat (*frame skipping*), menyesuaikan kecepatan pembaruan berdasarkan kompleksitas gerakan, dan menyimpan posisi terakhir untuk pengiriman data secara bersamaan. Prosedur *cleanup* dijalankan secara otomatis untuk memastikan semua

sumber daya ditutup dengan benar, seperti menghentikan *thread*, menonaktifkan torsi *servo*, dan menutup *port* ketika *plugin* dinonaktifkan.

3.1.5 Pengujian dan Evaluasi

Pengujian dan evaluasi dilakukan untuk memastikan bahwa *plugin* untuk kendali *servo* robot humanoid 17-DOF bekerja dengan baik dan sesuai dengan tujuan penelitian. Tahapan pengujian dibagi menjadi beberapa bagian yang akan dilakukan secara bertahap untuk memvalidasi fungsionalitas kontrol *real-time* dan kapabilitas *export* dan *import*.

3.1.5.1 Pengujian Instalasi dan *Setup Plugin*

Pengujian proses aktivasi dan deaktivasi *plugin* melalui Blender Preferences dengan verifikasi bahwa *panel plugin servo control* muncul di kategori *Animation* pada 3D Viewport. Validasi instalasi Dynamixel SDK *dependency* dengan pengecekan *flag* `DYNAMIXEL_AVAILABLE` dan *graceful fallback* jika SDK tidak tersedia. Pengujian registrasi semua komponen *plugin* termasuk *operators*, *panels*, *properties*, dan *menu items* dengan verifikasi bahwa tidak ada *error* saat *registration/unregistration*.

3.1.5.2 Pengujian Antarmuka Pengguna dan Konfigurasi

Pengujian tampilan antarmuka *plugin* dilakukan untuk memastikan semua elemen UI dapat ditampilkan dengan baik dan responsif. Pengujian juga dilakukan pada pengaturan konfigurasi seperti pemilihan *port* USB, pengaturan *baudrate*, jumlah *servo*, dan kecepatan *update* untuk memastikan perubahan nilai dapat tersimpan dengan benar. Selain itu, dilakukan pengujian pada antarmuka pengaturan *offset servo* untuk memastikan fungsi *enable*/*disable* tiap *servo*, pengaturan parameter perkalian/penambahan, dan pembaruan nilai secara langsung berjalan dengan baik.

Pengujian juga mencakup fungsi dialog *import* dan *export* untuk memastikan integrasi dengan *file browser* berjalan lancar, validasi parameter berfungsi, dan umpan balik ke pengguna ditampilkan dengan benar. Terakhir, dilakukan pengujian pada aksi cepat seperti menghapus *keyframe* dan memutar animasi untuk memastikan integrasi dengan sistem animasi Blender berjalan dengan baik.

3.1.5.3 Pengujian Komunikasi *Real-time* U2D2

Pengujian koneksi U2D2 dilakukan dengan menghubungkan perangkat ke *port* USB yang aktif dan memastikan *baudrate* yang digunakan sesuai dengan spesifikasi *servo* (1 Mbps). Setelah terkoneksi, dilakukan pengujian untuk memastikan *servo* XL-320 dan MX-28 dapat dideteksi dan diaktifkan dengan benar.

Pengujian gerakan *servo* dilakukan melibatkan penerapan animasi tarian "Gambang Semarang" untuk memastikan *servo* dapat bergerak secara *real-time* sesuai dengan pergerakan model 3D di Blender. Pengujian ini fokus pada kelancaran gerakan dan sinkronisasi antara animasi dengan gerakan robot fisik.

3.1.5.4 Pengujian Fungsional *Export* dan *Import*

Pengujian *export* JSON dilakukan untuk memastikan *file* yang dihasilkan memiliki struktur dan format yang benar. Pengujian mencakup verifikasi data *keyframe* yang dihasilkan, termasuk posisi *frame* dan nilai rotasi untuk setiap *servo*. Parameter *export* seperti rentang *frame* dan *interval frame* juga diuji untuk memastikan hasil sesuai dengan yang diinginkan.

Pengujian *import* JSON dilakukan untuk memastikan data animasi dapat dibaca kembali ke dalam Blender dengan benar. Pengujian fokus pada pembuatan *keyframe* baru dan penerapan nilai rotasi yang sesuai dengan data JSON. Hasil *import* diverifikasi dengan memeriksa bahwa *keyframe* yang dibuat memiliki nilai rotasi yang tepat sesuai dengan data asli.

3.1.5.5 Pengujian *Usability*

Pengujian *usability* dilakukan dengan melibatkan pengguna dari tim robotika untuk mengevaluasi kemudahan penggunaan plugin melalui skenario-skenario berikut:

1. Instalasi *plugin*: Pengguna menginstal dan mengaktifkan *plugin* melalui Blender Preferences, kemudian memverifikasi tampilan *panel*.
   Langkah-langkah yang harus dilakukan:
   a. Membuka Blender
   b. Pergi ke menu *Edit > Preferences*
   c. Memilih *tab Add-ons*
   d. Mengklik *Install* untuk memilih file ZIP *plugin*
   e. Mengaktifkan *checkbox plugin* setelah instalasi berhasil
   f. Memverifikasi bahwa *panel Servo Control* muncul di *sidebar Animation* pada *3D Viewport*
2. *Import* data JSON: Pengguna mengimpor *file* JSON animasi dan memeriksa penerapan *keyframe* pada model 3D.
   Langkah-langkah yang harus dilakukan:
   a. Membuka *panel plugin* di *3D Viewport*
   b. Mengklik tombol *Import* JSON
   c. Memilih *file* JSON melalui *file browser* Blender
   d. Mengatur parameter *import* jika diperlukan (seperti rentang *frame*)
   e. Mengkonfirmasi *import*
   f. Memeriksa *keyframe* baru di *timeline* serta perubahan pada model 3D untuk memastikan data rotasi diterapkan dengan benar
3. *Export* data JSON: Pengguna mengekspor animasi ke *file* JSON dengan parameter tertentu dan memverifikasi isi file.
   Langkah-langkah yang harus dilakukan:
   a. Menyiapkan animasi dengan *keyframe* di Blender
   b. Membuka panel *plugin*
   c. Mengklik tombol *Export* JSON
   d. Mengatur *parameter* seperti rentang *frame* dan *interval sampling*
   e. Memilih lokasi penyimpanan melalui *file browser*
   f. Mengkonfirmasi *export*
   g. Membuka file JSON yang dihasilkan untuk memverifikasi struktur data (*frame positions* dan nilai rotasi servo)
4. *Sync* robot: Pengguna menghubungkan U2D2, mendeteksi *servo*, dan menjalankan animasi *real-time* untuk sinkronisasi dengan robot fisik.
   Langkah-langkah yang harus dilakukan:
   a. Menghubungkan perangkat U2D2 ke *port* USB komputer
   b. Membuka *panel plugin* dan memilih *port* USB serta *baudrate* (1 Mbps)
   c. Mengklik tombol *Connect Servos* untuk mendeteksi dan mengaktifkan *servo*
   d. Mengaktifkan mode *sync real-time*
   e. Memainkan animasi di *timeline* Blender

f. Mengamati sinkronisasi gerakan antara model 3D dan robot fisik untuk memastikan tidak ada *delay* atau kesalahan

5. Konfigurasi *offset servo*: Pengguna mengatur *offset* dan menguji perubahan pada gerakan servo.
   Langkah-langkah yang harus dilakukan:
   a. Membuka bagian *Servo Offset Individual* di *panel plugin*
   b. Memilih *servo* individu dari daftar
   c. Mengatur nilai *multiplier* dan *addition* untuk *offset*
   d. Menguji dengan menjalankan animasi dalam mode *sync* untuk memverifikasi bahwa gerakan servo telah disesuaikan dengan benar

Pengujian *usability* dilakukan dengan menggunakan kuisioner dengan skala *Likert* untuk mengukur aspek kemudahan, kecepatan, dan fleksibilitas.

3.2 Bahan dan peralatan yang digunakan

3.2.1 Perangkat Keras

Penelitian ini menggunakan beberapa perangkat keras untuk pengembangan dan pengujian sistem kendali robot humanoid. Berikut adalah daftar perangkat keras yang dibutuhkan:

1. Robot Humanoid
   - Robot humanoid dengan 17 derajat kebebasan untuk bagian *upper body* robot yang telah dirakit sebelumnya
   - Struktur mekanik robot yang terbuat dari *material aluminium* dan plastik
   - Sistem kabel untuk menghubungkan *servo* dan mikrokontroler
2. *Servo* Motor
   - 15 buah *servo* motor Dynamixel XL-320
   - 2 buah *servo* motor Dynamixel MX-28
3. Antarmuka Komunikasi U2D2
   - U2D2 untuk komunikasi *real-time* USB ke TTL
   - Kabel USB untuk koneksi dengan komputer
   - *Power Supply* 12 *volt* untuk sumber daya *servo* motor
4. Perangkat Laptop
   Laptop MacBook Pro 2024 14 *inch* dengan spesifikasi:
   - Prosesor: Apple M4 *Pro*
   - RAM: 20GB
   - SSD: 1TB
   - Display: 14.2 *inch* Retina XDR
5. Peralatan Pendukung
   - Peralatan toolkit (obeng, tang, *solder*, dll) untuk penyesuaian *hardware*

3.2.2 Perangkat Lunak

Beberapa perangkat lunak dibutuhkan untuk mengembangkan dan mengimplementasikan sistem *plugin* Blender untuk kendali robot. Perangkat lunak yang digunakan antara lain:

1. Blender 3D v4.4
2. Visual Studio Code
3. Git
4. Python v3.14

5. Dynamixel Wizard
6. Microsoft Word
7. Figma

3.3 Urutan Pelaksanaan Penelitian

Pada tugas akhir ini urutan pelaksanaan penelitian terdiri dari beberapa tahap utama yang dirancang untuk memastikan pengembangan *plugin* untuk kendali *servo* robot humanoid 17-DOF berbasis U2D2 berjalan secara sistematis dan terukur. Tahapan tersebut adalah sebagai berikut:

1. Studi Literatur dan Analisis Kebutuhan
   Pada tahap ini dilakukan pengumpulan data primer dan sekunder melalui studi literatur untuk memahami konsep dasar pengembangan *plugin* Blender, protokol komunikasi *servo* Dynamixel, dan implementasi U2D2 sebagai antarmuka komunikasi. Kegiatan ini mencakup kajian terhadap dokumentasi API *Python* Blender (bpy), protokol komunikasi Dynamixel 2.0, serta studi komparatif terhadap penelitian terdahulu terkait kendali robot humanoid menggunakan software animasi. Analisis kebutuhan sistem dilakukan untuk menentukan spesifikasi fungsional dan non-fungsional *plugin* yang akan dikembangkan. Data yang diperoleh menjadi dasar untuk merumuskan permasalahan dan menentukan pendekatan teknis yang akan diimplementasikan.
2. Perancangan Sistem dan Arsitektur *Plugin*
   Berdasarkan hasil studi literatur, dirancang arsitektur sistem secara menyeluruh dengan fokus pada dua mode operasi, yaitu kontrol *real-time* dan *export* dan *import* animasi. Pada tahap ini, disusun struktur *plugin* Blender dengan pendekatan modular, dirancang protokol komunikasi untuk perangkat U2D2, serta ditentukan format data JSON agar bisa digunakan secara mandiri di berbagai aplikasi (*standalone*). Selain itu, dilakukan pemodelan 3D robot humanoid dengan 17 derajat kebebasan (17-DOF) di Blender, lengkap dengan proses rigging yang disesuaikan dengan struktur fisik robot dan penamaan bone yang konsisten dengan ID *servo*. Kegiatan dalam tahap ini mencakup penentuan format data, pembuatan diagram alur kerja sistem, pemetaan ID *servo* ke bone dalam model 3D, serta desain antarmuka pengguna yang mendukung interaksi intuitif dengan *plugin*.
3. Pengembangan *Plugin* Blender
   Tahap ini berfokus pada implementasi *plugin* menggunakan Python API dari Blender, yang diintegrasikan dengan Dynamixel SDK. Beberapa aktivitas utama meliputi pengembangan antarmuka pengguna yang dibagi ke dalam bagian-bagian yang terorganisir dan mudah digunakan, serta pembuatan sistem properti (`ServoControllerProperties`) untuk pengaturan konfigurasi dan penyimpanan data. Selain itu, dikembangkan kelas `ServoController` yang bertugas mengatur komunikasi dengan perangkat keras, dengan kemampuan deteksi otomatis jenis *servo* dan sistem `GroupSyncWrite`. Pengembangan juga mencakup implementasi *frame change handler* untuk sinkronisasi animasi secara *real-time*, sistem *caching* untuk meningkatkan performa, pengaturan *offset servo* secara individual dengan parameter *multiply* dan *add*, serta penggunaan *threading* di latar belakang agar *plugin* tetap berjalan lancar. *Plugin* ini dirancang agar dapat digunakan dengan mudah oleh animator dan pengembang robot, tanpa perlu memahami detail teknis pemrograman *servo*.
4. Integrasi dan Optimasi Sistem

Setelah seluruh komponen utama selesai dikembangkan, dilakukan integrasi sistem secara menyeluruh serta pengujian komprehensif untuk memastikan semua fungsi berjalan sesuai rencana. Kegiatan dalam tahap ini meliputi pengujian fungsional untuk setiap fitur *plugin*, pengujian antarmuka pengguna untuk memastikan setiap UI yang ditampilkan dapat bekerja dengan baik, serta pengujian *usability* untuk memastikan *plugin* dapat digunakan secara mudah dan intuitif. Optimasi dilakukan untuk mengurangi latensi komunikasi, meningkatkan akurasi pergerakan *servo*, memaksimalkan pemanfaatan sumber daya, dan meningkatkan pengalaman pengguna melalui antarmuka yang responsif serta umpan balik yang informatif. Perbaikan bug dan penyempurnaan fitur juga dilakukan berdasarkan hasil pengujian.

5. Pengujian dan Validasi
   Tahap ini meliputi pengujian komprehensif terhadap sistem yang telah diintegrasikan. Kegiatan pengujian mencakup pengujian fungsional *plugin* Blender, pengujian antarmuka pengguna, dan pengujian *usability*. Evaluasi dilakukan dengan mengukur berbagai metrik seperti latensi sistem, akurasi sudut, dan kestabilan komunikasi. Hasil pengujian dianalisis untuk menentukan tingkat keberhasilan implementasi dan mengidentifikasi area yang memerlukan perbaikan.
6. Penyusunan Laporan
   Hasil akhir dari seluruh proses penelitian didokumentasikan secara sistematis dalam bentuk laporan tugas akhir. Laporan ini mencakup latar belakang, tinjauan pustaka, metodologi, hasil implementasi, analisis hasil pengujian, dan kesimpulan. Dokumentasi kode program dan petunjuk penggunaan *plugin* juga disertakan sebagai bagian dari laporan atau sebagai lampiran. Tahap ini juga mencakup persiapan presentasi dan demonstrasi sistem untuk sidang tugas akhir.

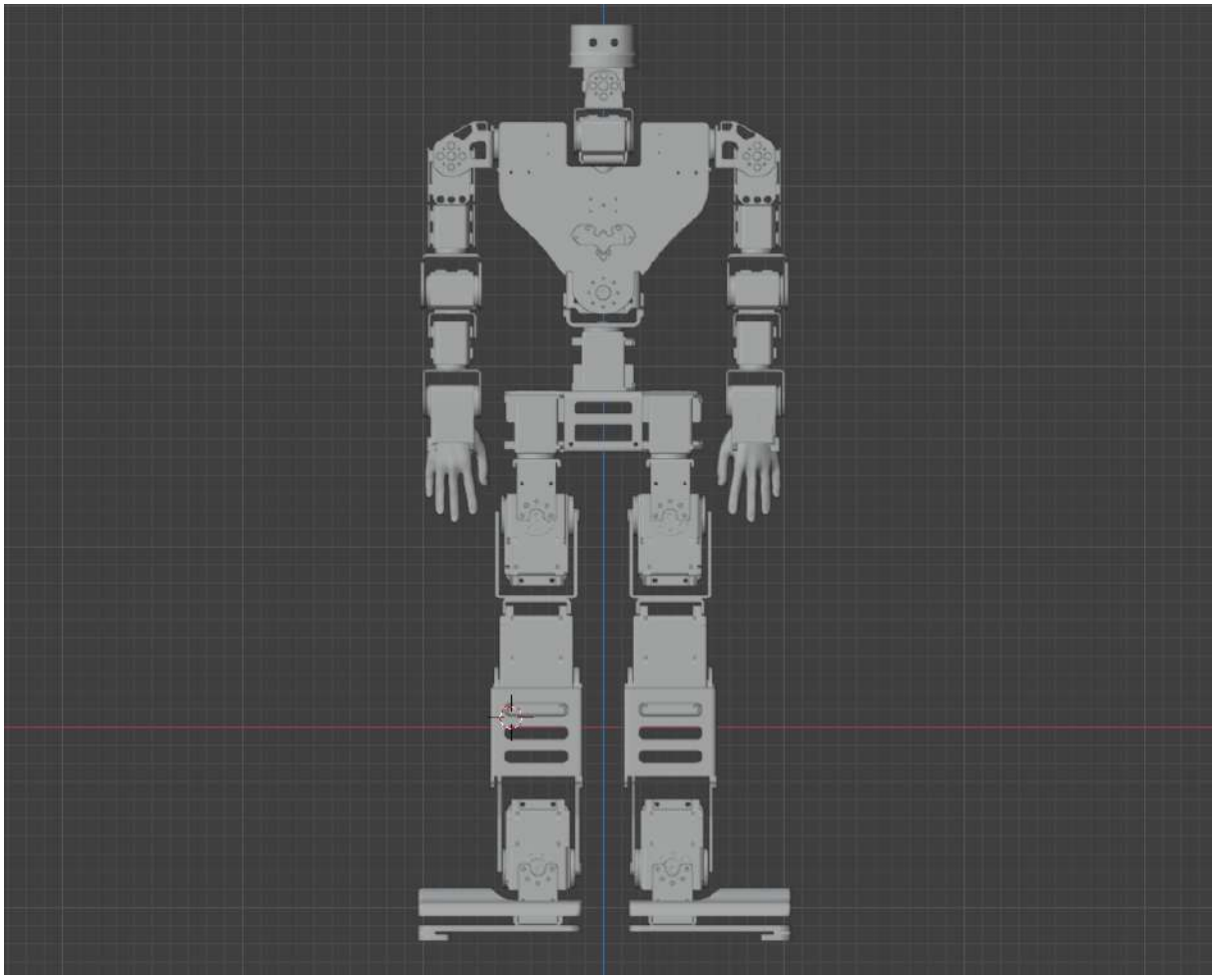
BAB 4
HASIL DAN PEMBAHASAN

4.1 Hasil Penelitian

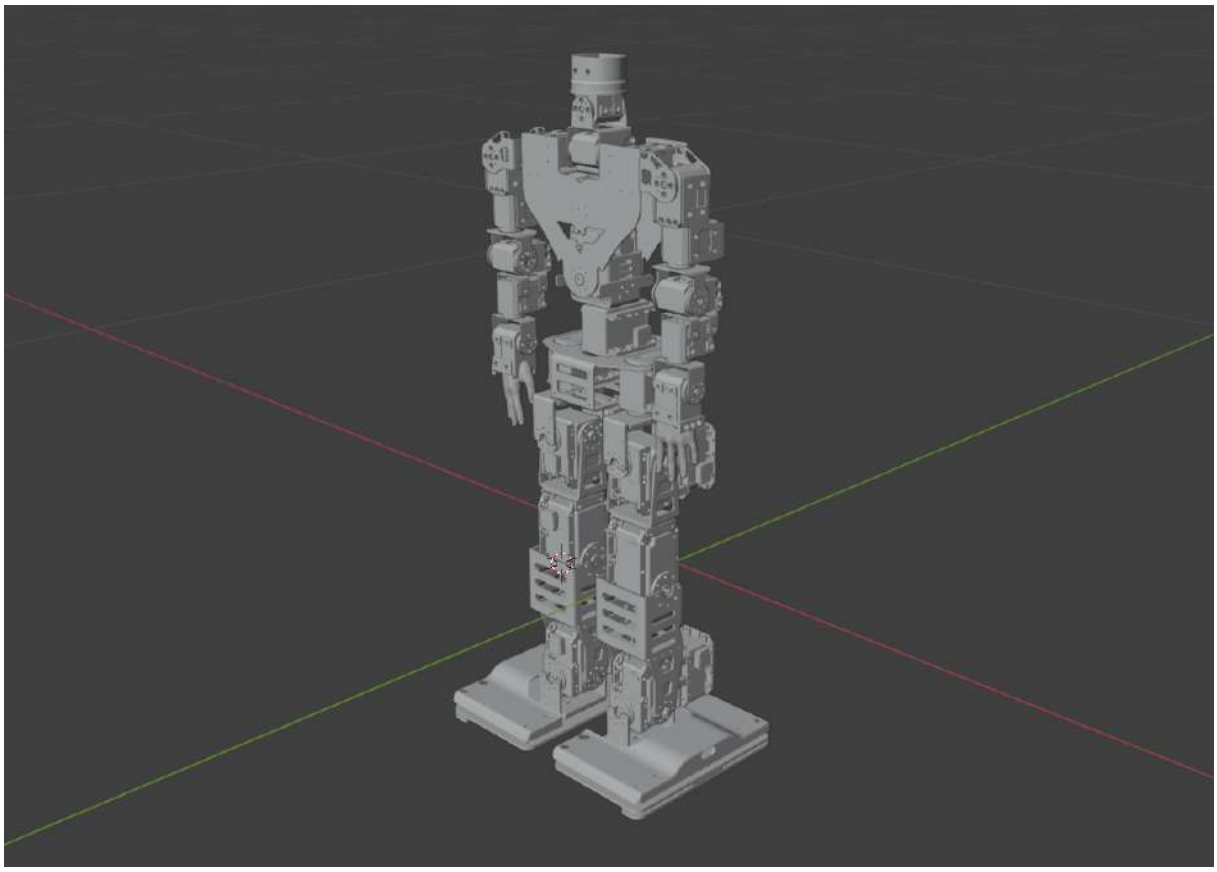
4.1.1 Hasil Pemodelan 3D Robot Humanoid di Blender

4.1.1.1 Model *Mesh* Robot

Proses pembuatan model *mesh* robot humanoid dilakukan dengan memperhatikan akurasi dimensi dan proporsi sesuai dengan robot fisik yang akan dikontrol. Model dibuat menggunakan komponen yang terdiri dari *servo* dynamixel XL-320 dan MX-28. Penempatan *servo* motor bagian atas menggunakan tipe XL-320 sedangkan untuk bagian bawah menggunakan tipe MX-28. Setiap *servo* motor disambungkan dengan plat besi untuk menyangga beban robot yang sudah disesuaikan. Hasil pemodelan robot dapat dilihat seperti yang ditunjukkan pada Gambar 4.1 dan Gambar 4.2.



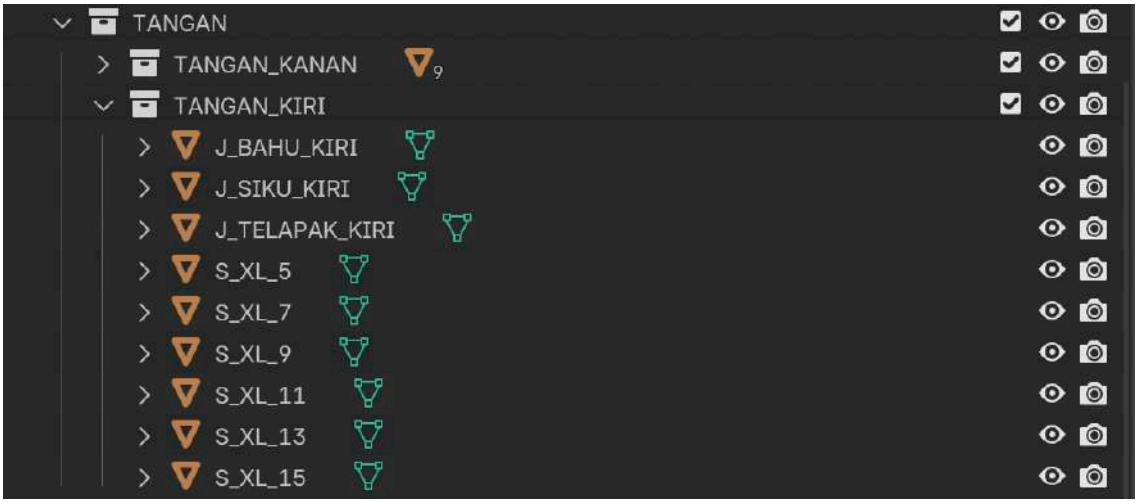
Gambar 4.1 Model 3D robot humanoid tampak depan



Gambar 4.2 Model 3D robot humanoid tampak samping atas

4.1.1.2 Pengorganisasian Nama Komponen

Dalam pemodelan 3D robot humanoid, pengorganisasian nama komponen dilakukan secara sistematis berdasarkan data *collection* yang telah dikumpulkan untuk memudahkan proses *rigging* dan animasi. Setiap komponen dipisahkan ke dalam beberapa *collection* yang sesuai dengan posisinya. Penamaan komponen menggunakan sistem kode terstruktur untuk memudahkan identifikasi dan manajemen. Contoh pengorganisasian nama komponen pada bagian tangan dapat terlihat seperti pada Gambar 4.3.



Gambar 4.3 Pengorganisasian nama komponen bagian tangan

Konvensi Kode Penamaan:

- S = *Servo* (motor penggerak)
- J = *Joint* (sambungan/sendi)
- P = *Part* (komponen struktural)

Format Penamaan:

- *Servo*: S_[TIPE]_[ID] (contoh: S_MX_1, S_XL_2)
- *Joint*: J_[NAMA_SENDI] (contoh: J_LEHER, J_BAHU_KANAN)
- *Part*: P_[NAMA_BAGIAN] (contoh: P_DADA, P_PINGGUL)

Berikut adalah struktur *collection* yang telah dibuat:

1. KEPALA
    - J_LEHER: Sambungan leher
    - S_XL_1, S_XL_2, S_XL_3: *Servo* penggerak kepala (tipe XL-320)
2. BADAN
    - P_DADA, P_PANTAT: Komponen struktural badan
    - S_MX_1, S_MX_2: *Servo* penggerak torso (tipe MX-64)
3. TANGAN_KANAN
    - J_BAHU_KANAN: Sambungan bahu kanan
    - J_SIKU_KANAN: Sambungan siku kanan
    - J_TELAPAK_KANAN: Sambungan telapak tangan kanan
    - S_XL_4, S_XL_6, S_XL_8, S_XL_10, S_XL_12, S_XL_14: *Servo* penggerak lengan kanan (tipe XL-320)
4. TANGAN_KIRI:
    - J_BAHU_KIRI: Sambungan bahu kiri
    - J_SIKU_KIRI: Sambungan siku kiri
    - J_TELAPAK_KIRI: Sambungan telapak tangan kiri
    - S_XL_5, S_XL_7, S_XL_9, S_XL_11, S_XL_13, S_XL_15: *Servo* penggerak lengan kiri (tipe XL-320)
5. KAKI_KANAN
    - J_KAKI_KANAN: Sambungan kaki kanan
    - J_LUTUT_KANAN: Sambungan lutut kanan
    - J_PAHA_KANAN: Sambungan paha kanan
    - J_PANTAT_KANAN: Sambungan pantat kanan
    - S_MX_4, S_MX_6, S_MX_8, S_MX_10, S_MX_12, S_MX_14: *Servo* penggerak kaki kanan (tipe MX-28)
6. KAKI_KIRI
    - J_KAKI_KIRI: Sambungan kaki kiri
    - J_LUTUT_KIRI: Sambungan lutut kiri
    - J_PAHA_KIRI: Sambungan paha kiri
    - J_PANTAT_KIRI: Sambungan pantat kiri
    - S_MX_3, S_MX_5, S_MX_7, S_MX_9, S_MX_11, S_MX_13: *Servo* penggerak kaki kiri (tipe MX-28)

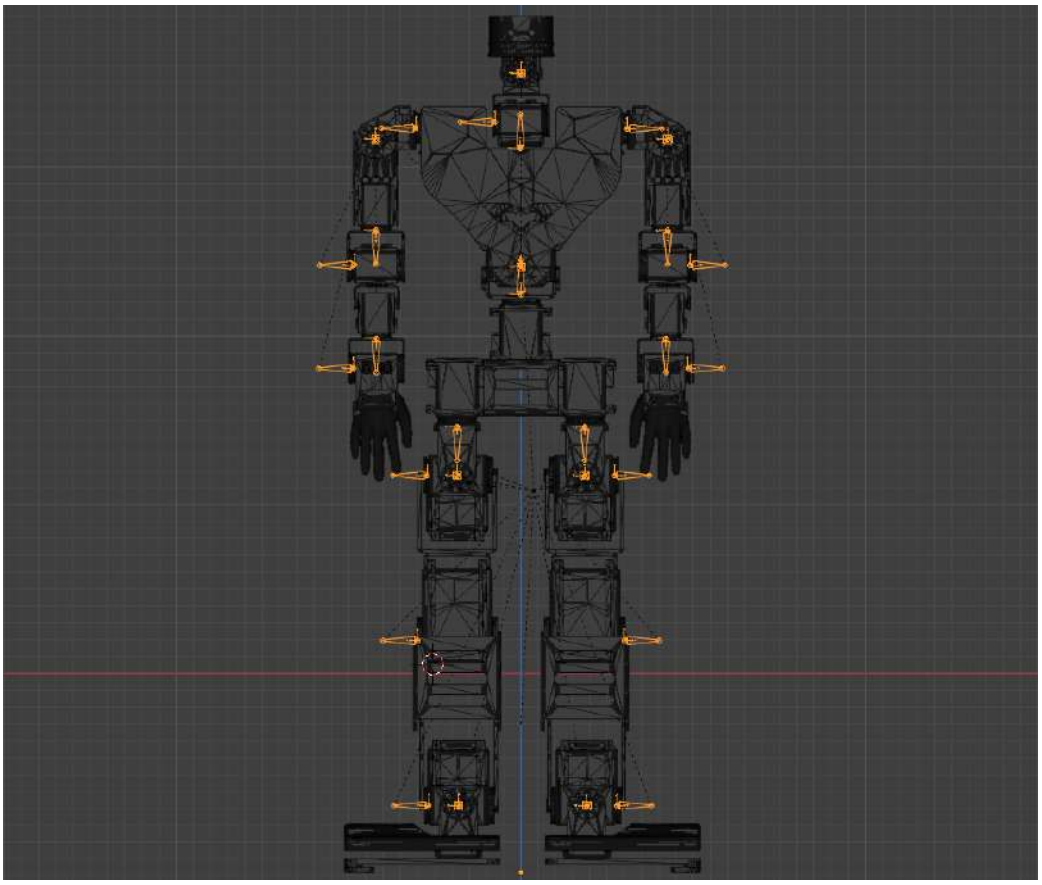
Keuntungan dari penggunaan konvensi penamaan ini adalah kemudahan dalam mengidentifikasi komponen berdasarkan jenis dan fungsinya secara langsung. Setiap *servo*

dikategorikan secara sistematis berdasarkan tipe (seperti XL-320 dan MX-28) serta memiliki ID yang unik, sehingga memudahkan dalam proses pengelompokan dan pengaturan. Dalam aspek pemrograman, sistem atau *plugin* dapat secara otomatis mendeteksi dan memetakan komponen sesuai dengan kode yang digunakan. Hal ini juga meningkatkan konsistensi data antara model 3D dengan perangkat keras robot, sehingga sinkronisasi menjadi lebih akurat. Selain itu, struktur penamaan yang hierarkis memudahkan dalam proses pemeliharaan, *debugging*, dan modifikasi sistem secara keseluruhan.

4.1.1.3 Struktur *Armature* dan Hirarki *Bone*

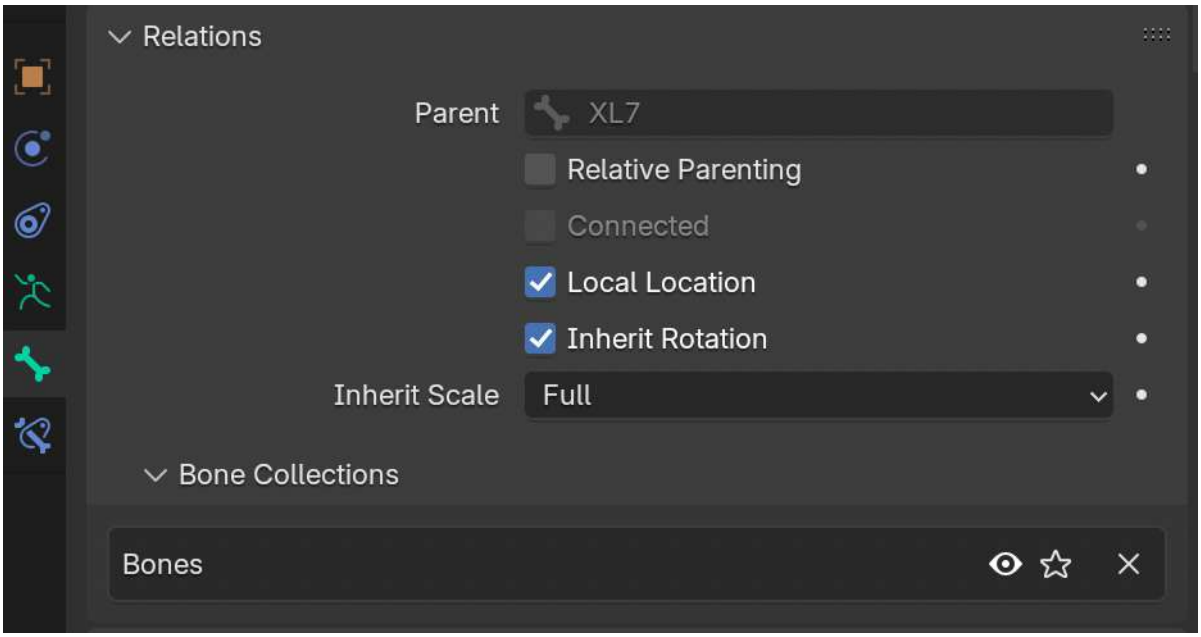
Struktur *armature* dan hirarki *bone* merupakan komponen fundamental dalam sistem kendali robot humanoid, berfungsi sebagai representasi digital dari struktur mekanik robot. Dalam implementasi ini, struktur *bone* dibangun di Blender dengan memperhatikan relasi hierarkis antar sendi yang mencerminkan konfigurasi fisik *servo* pada robot humanoid.

Dalam implementasinya, setiap *bone* diposisikan dengan mempertimbangkan sumbu rotasi *servo* yang sebenarnya. *Bone* diletakkan tepat pada titik pusat rotasi *servo* dengan orientasi yang sesuai dengan arah pergerakan *servo* fisik. Seluruh rotasi sendi dalam sistem ini menggunakan sumbu Y sebagai poros utama pergerakan, sehingga penempatan dan orientasi bone disesuaikan agar gerakan rotasi di Blender mencerminkan rotasi aktual pada *servo*. Penyesuaian posisi dan orientasi *bone* yang presisi ini memastikan bahwa gerakan yang dihasilkan dalam simulasi akan akurat ketika diterjemahkan ke gerakan robot fisik. Penempatan *bone* ini dapat dilihat pada Gambar 4.4 yang berwarna *orange*.



Gambar 4.4 Penempatan *Bone*

Setelah penempatan *bone* yang tepat, langkah selanjutnya adalah mengatur relasi antar *bone* untuk membentuk struktur yang koheren. Relasi ini diimplementasikan menggunakan sistem *parent-child* yang memungkinkan perambatan transformasi dari *bone parent* ke *bone child*. Pengaturan relasi ini dapat dilakukan pada dialog *bone properties* pada bagian *relations* seperti yang terlihat pada Gambar 4.5.



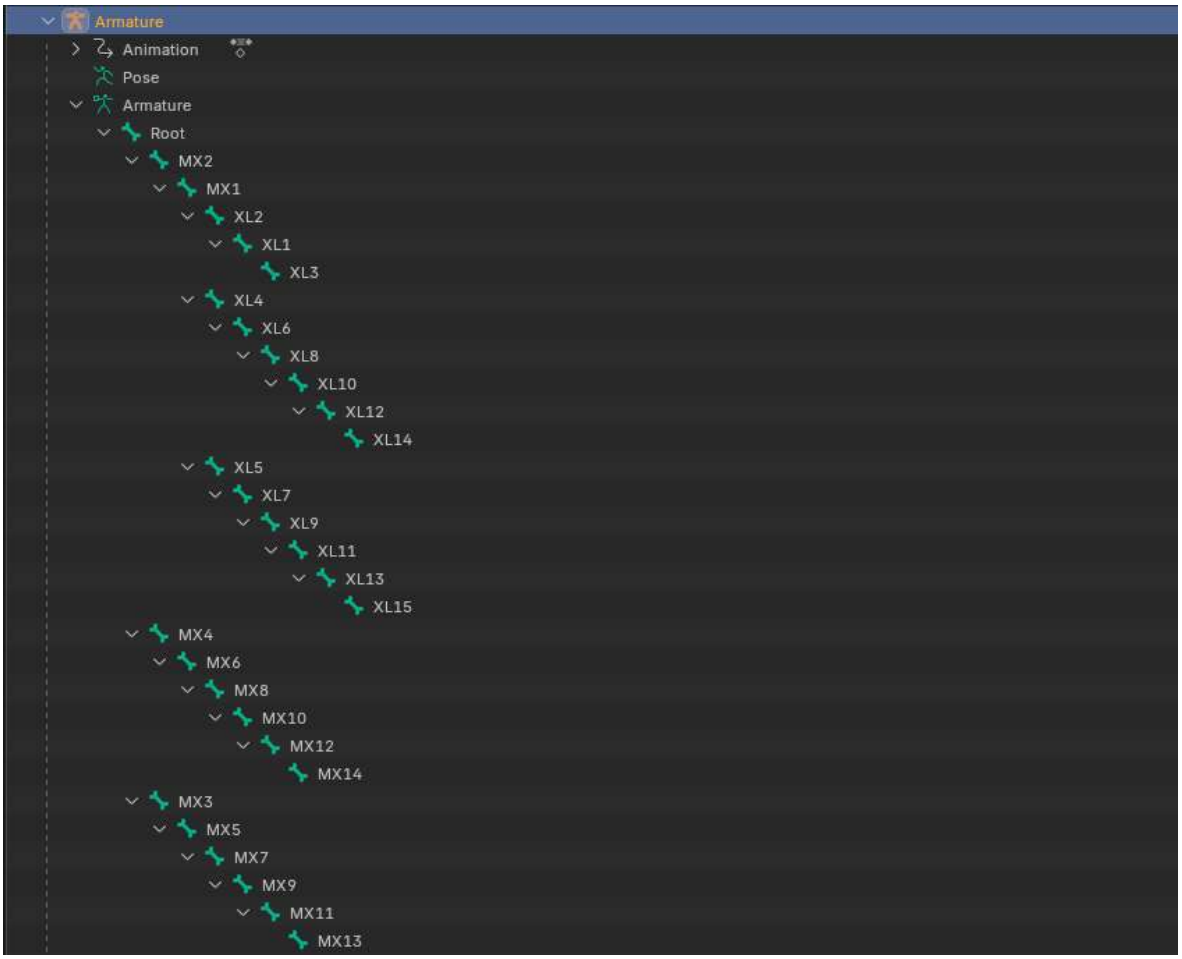
Gambar 4.5 Konfigurasi *relations parent-child* pada *bone*

Pengaturan relasi ini sangat krusial karena mencerminkan bagaimana bagian-bagian robot saling terhubung dan bergerak secara bersamaan. Misalnya, ketika *bone* bahu bergerak, seluruh *bone* lengan yang menjadi anaknya akan ikut bergerak, sama seperti pada robot fisik dimana gerakan *servo* bahu mempengaruhi posisi keseluruhan lengan.

Struktur hierarki *bone* dimulai dari *root bone* dan terbagi menjadi dua cabang utama berdasarkan tipe *servo*:

1. *Servo* XL-320 (Bagian Atas - Kepala dan Lengan)
    - Bagian kepala: XL2 → XL1 → XL3
    - Lengan kanan: XL4 → XL6 → XL8 → XL10 → XL12 → XL14
    - Lengan kiri: XL5 → XL7 → XL9 → XL11 → XL13 → XL15
2. *Servo* MX-28 (Bagian Bawah - Kaki dan Torso)
    - *Bone* pusat tubuh (torso): MX1 dan MX2
    - Rangkaian kaki kanan: MX4 → MX6 → MX8 → MX10 → MX12 → MX14
    - Rangkaian kaki kiri: MX3 → MX5 → MX7 → MX9 → MX11 → MX13

Konfigurasi struktur hirarki pada Blender membentuk susunan yang teratur sesuai pembagian tipe *servo*. Struktur ini dapat dilihat pada Gambar 4.6.



Gambar 4.6 Hirarki *Bone*

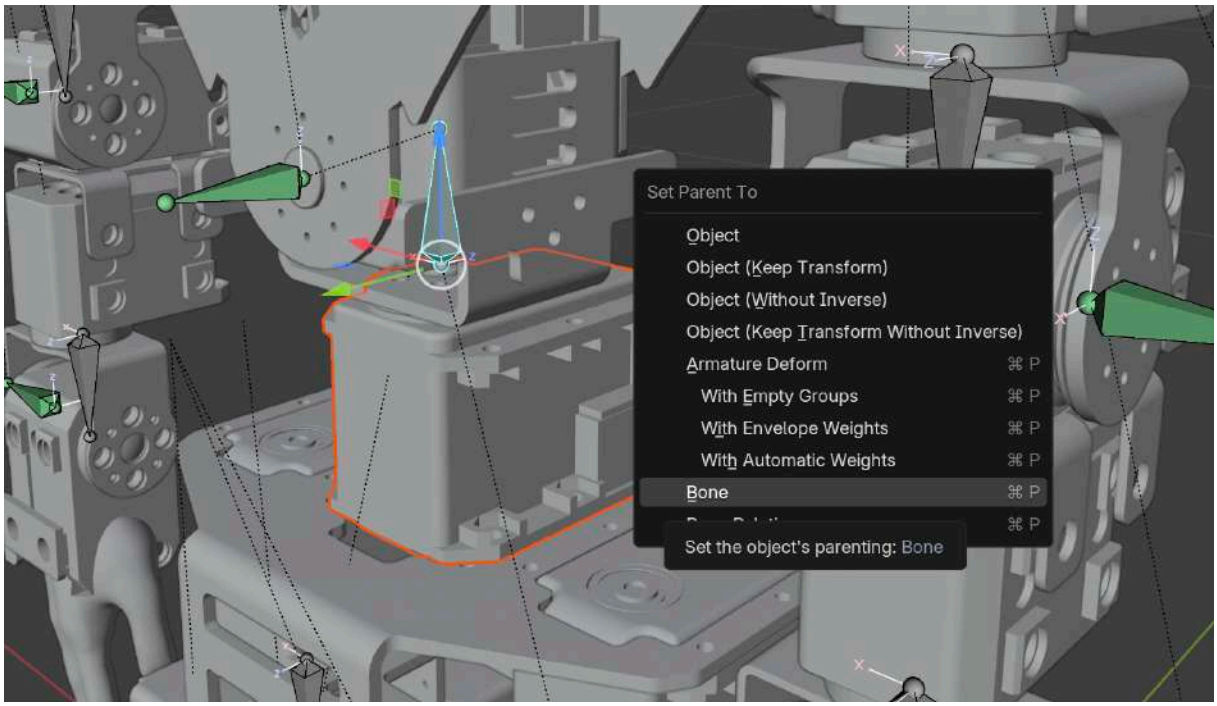
Penamaan *bone* mengadopsi skema yang konsisten dengan ID *servo* pada robot fisik, memudahkan pemetaan antara animasi di Blender dengan kontrol gerakan aktual. Setiap nama *bone* (contoh: MX4 atau XL10) secara langsung merepresentasikan *servo* dengan ID yang sesuai pada sistem fisik, memungkinkan konversi otomatis dari data animasi ke perintah *servo* melalui *plugin*.

Struktur ini menjamin presisi translasi gerakan dari model digital ke robot fisik, sekaligus menjaga konsistensi antara simulasi 3D dan aktuasi *servo*. Dengan desain hierarki *bone* yang tepat, sistem dapat mengeksekusi animasi kompleks sambil mengontrol *servo* secara terstruktur dan efisien.

4.1.1.4 Implementasi *Skinning* dan *Parent-Child Relationship*

Proses *skinning* dilakukan untuk menghubungkan *mesh* model 3D dengan *armature* yang telah dibuat. Karena setiap komponen robot sudah terpisah dan bersifat solid (*rigid body*), proses *skinning* disederhanakan dengan menghubungkan setiap *part* langsung ke *parent bone* yang sesuai tanpa memerlukan *weight painting* kompleks.

Seperti yang terlihat pada Gambar 4.7, setiap komponen robot seperti *housing servo*, *bracket*, dan *attachment parts* dihubungkan langsung ke *bone* yang bersesuaian dengan menggunakan *parent-child relationship*. Hal ini memungkinkan setiap *part* bergerak sebagai satu kesatuan *rigid* dengan *bone* yang mengontrolnya, sesuai dengan karakteristik mekanis robot sebenarnya.



Gambar 4.7 Konfigurasi *parent part* ke *bone*

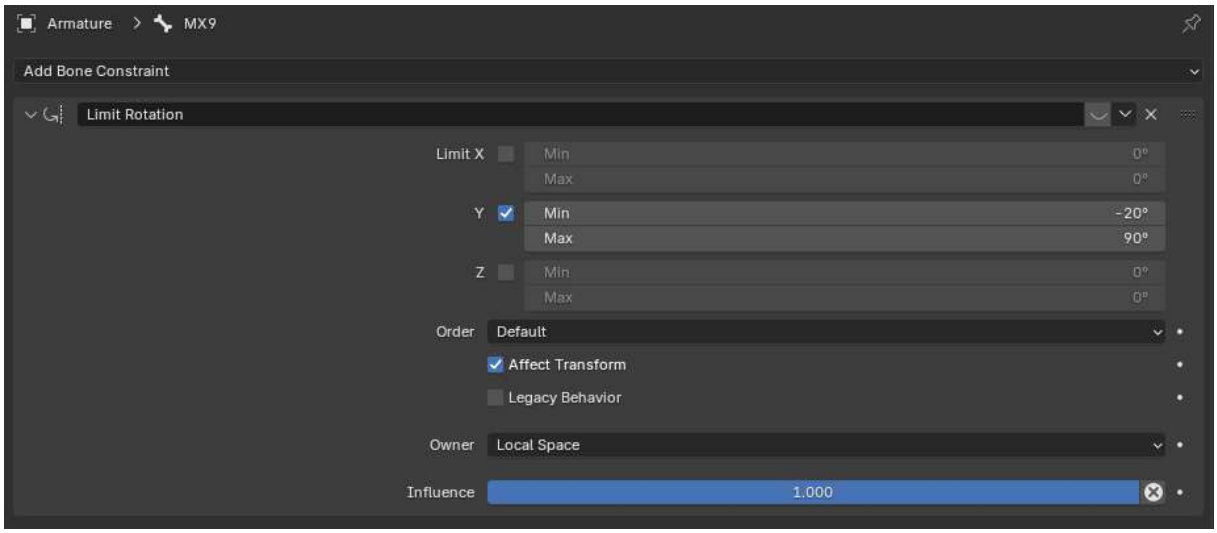
Konfigurasi *parent-child* dilakukan dengan memberikan *full weight* (1.0) pada setiap *vertex* komponen terhadap *bone* tunggal yang mengontrolnya. Pendekatan ini lebih sederhana dan akurat dibandingkan *weight painting* karena mencerminkan sifat mekanis robot dimana setiap *part* bergerak sebagai *rigid body* tanpa deformasi *internal*.

Validasi *parenting* dilakukan melalui pengujian pergerakan untuk memastikan tidak ada komopnen yang terlepas dari *bone parent* dan semua gerakan mengikuti hierarki kinematik yang telah didefinisikan. Sistem ini memberikan kontrol yang presisi dan performa yang optimal untuk animasi robot.

4.1.1.5 Penerapan *Limit Constraint* pada *Bone*

Dalam implementasi *rigging* model robot humanoid, penerapan pembatasan gerakan pada setiap *bone* merupakan aspek krusial untuk menjamin keamanan dan akurasi gerakan. Pembatasan ini diimplementasikan menggunakan fitur *Limit Rotation Constraint* yang disediakan oleh Blender, yang memungkinkan pengaturan batas rotasi spesifik untuk setiap sumbu pada masing-masing *bone*.

Limit Rotation Constraint diterapkan pada setiap *bone* yang merepresentasikan *servo* motor, dengan konfigurasi yang disesuaikan berdasarkan spesifikasi mekanis *servo* yang sebenarnya. Pengaturan dilakukan dalam *local space*, yang berarti batasan rotasi dihitung relatif terhadap orientasi awal *bone*. Sebagai contoh pada Gambar 4.8, *bone* untuk *servo* MX-9 dibatasi hanya dapat berotasi pada sumbu Y dengan rentang -20° hingga 90° yang mencerminkan batasan fisik dari *servo* yang direpresentasikannya. Opsi *Affect Transform* juga dicentang, yang berarti batasan ini secara aktif akan membatasi transformasi aktual pada *bone*, baik saat dilakukan secara manual oleh animator, melalui *keyframe*, maupun melalui *scripting*.



Gambar 4.8 Penerapan *limit rotation* pada *bone*

Implementasi *constraint* ini memiliki beberapa manfaat:

1. Pencegahan *Collision Mesh*
    - Mencegah terjadinya interpenetrasi antar bagian model 3D
    - Menjamin integritas visual model selama proses animasi
    - Menghindari situasi dimana bagian tubuh robot (seperti lengan atau kaki) menembus geometri bagian lainnya
2. Validasi Gerakan
    - Memastikan setiap gerakan yang dibuat dalam animasi dapat direplikasi oleh robot fisik
    - Mencegah pembuatan *keyframe* dengan sudut yang melampaui kapabilitas *servo*
    - Memberikan *feedback* visual langsung kepada animator tentang batasan gerakan yang valid

Dengan penerapan *Limit Rotation Constraint*, sistem *rigging* tidak hanya berfungsi sebagai kerangka untuk animasi, tetapi juga berperan sebagai sistem validasi yang memastikan setiap gerakan yang dibuat dapat dieksekusi dengan aman oleh robot fisik. Pendekatan ini menciptakan lapisan keamanan tambahan dalam proses pembuatan animasi, sebelum data gerakan ditransmisikan ke sistem kontrol *servo* melalui *plugin*.

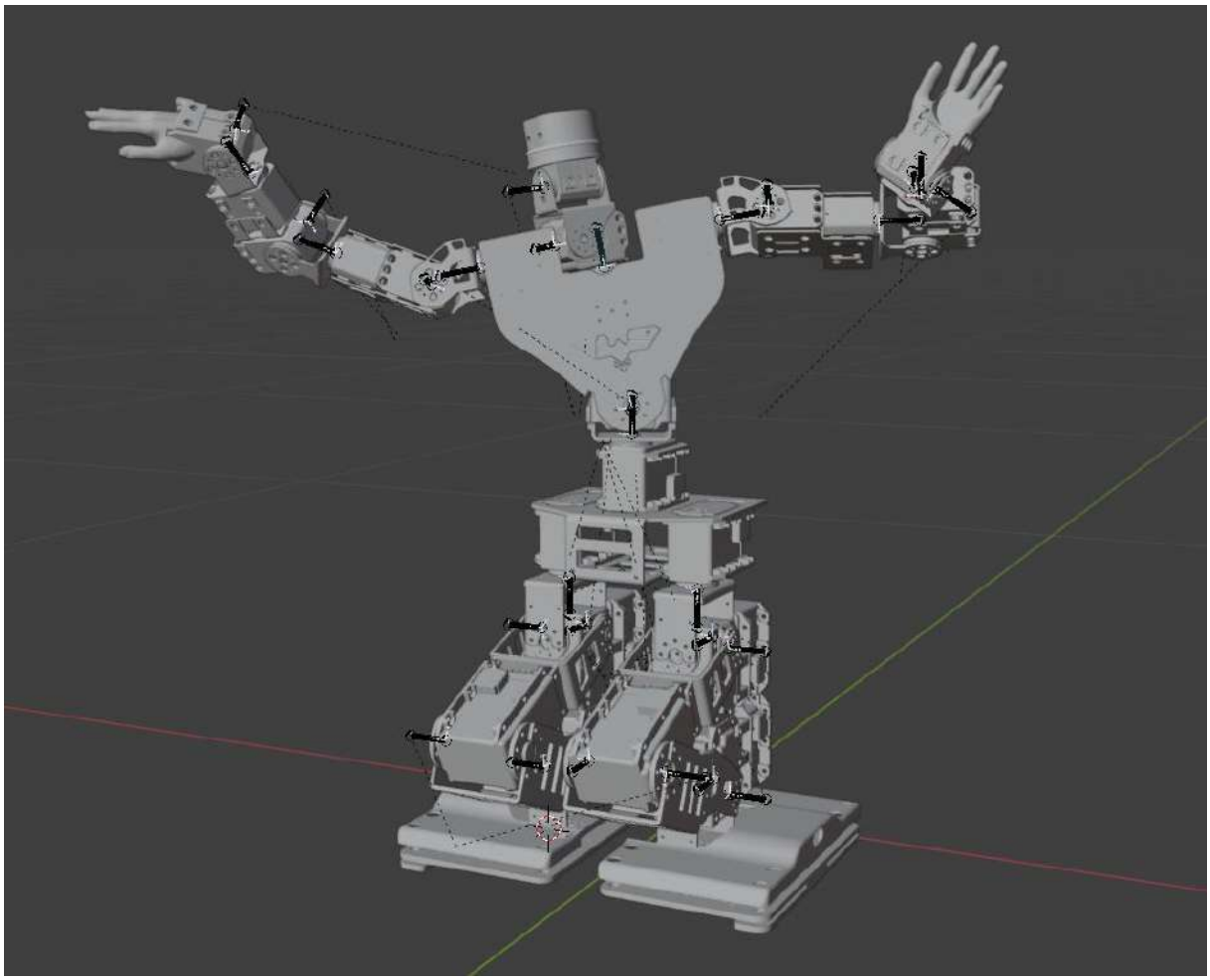
4.1.1.6 Pembuatan Animasi Tarian Robot

Setelah model dan *rigging* selesai, tahap selanjutnya adalah membuat animasi tarian robot menggunakan *timeline* dan *pose mode* di Blender. Proses animasi dilakukan dengan mengatur *keyframe* pada *timeline* untuk setiap gerakan yang diinginkan. Penggunaan *timeline* memungkinkan pengaturan waktu yang presisi untuk setiap gerakan robot.

Seperti yang ditunjukkan pada Gambar 4.9, robot dapat diposisikan dalam berbagai pose untuk membuat gerakan tarian yang dinamis. Pose-pose ini mencakup gerakan lengan, kepala, dan bagian tubuh lainnya yang dapat dikombinasikan untuk menghasilkan koreografi yang menarik.

Pembuatan animasi dimulai dengan mengatur pose robot di pose mode. Caranya adalah dengan memilih *bone* yang ingin digerakkan, lalu merotasi *bone* tersebut menggunakan *rotation tool* (shortcut 'R') sesuai dengan pose yang diinginkan. Setelah pose selesai dibuat, *keyframe* dapat ditambahkan dengan menekan 'I' dan memilih "*Rotation*". Penggunaan mode pose

memungkinkan manipulasi langsung terhadap *bone* tanpa mempengaruhi *mesh* atau komponen lainnya.



Gambar 4.9 Pose robot menari

Untuk membuat gerakan yang *smooth*, minimal dibutuhkan 2 *keyframe* yaitu pose awal dan pose akhir. Sebagai contoh untuk membuat gerakan lambaian tangan, pertama dibuat *keyframe* untuk pose tangan di posisi bawah pada *frame* 1, kemudian dibuat *keyframe* kedua untuk pose tangan terangkat pada *frame* 30. Blender akan secara otomatis menginterpolasi gerakan di antara kedua *keyframe* tersebut menggunakan fungsi interpolasi bezier yang dapat disesuaikan melalui *graph editor*.

Seperti yang terlihat pada Gambar 4.10, timeline Blender menampilkan rangkaian *keyframe* yang telah dibuat untuk mengontrol gerakan *servo* robot. Timeline ini memvisualisasikan urutan dan *timing* dari setiap gerakan dalam animasi, memudahkan proses penyuntingan dan penyesuaian gerakan.

Untuk gerakan yang lebih kompleks seperti tarian, dibutuhkan banyak *keyframe* yang disusun secara berurutan. Misalnya untuk gerakan tarian sederhana yang melibatkan lengan dan kepala, *keyframe* dapat diatur pada *frame* 1 untuk pose awal, *frame* 30 untuk gerakan lengan ke atas, *frame* 60 untuk rotasi kepala, dan *frame* 90 untuk kembali ke pose awal. Pengaturan *timing* dan *spacing* antar *keyframe* sangat penting untuk menghasilkan gerakan yang *natural* dan sesuai dengan ritme musik pengiring tarian.

Gambar 4.10 Timeline dan *keyframe servo*

4.1.2 Hasil Implementasi *Plugin* Blender

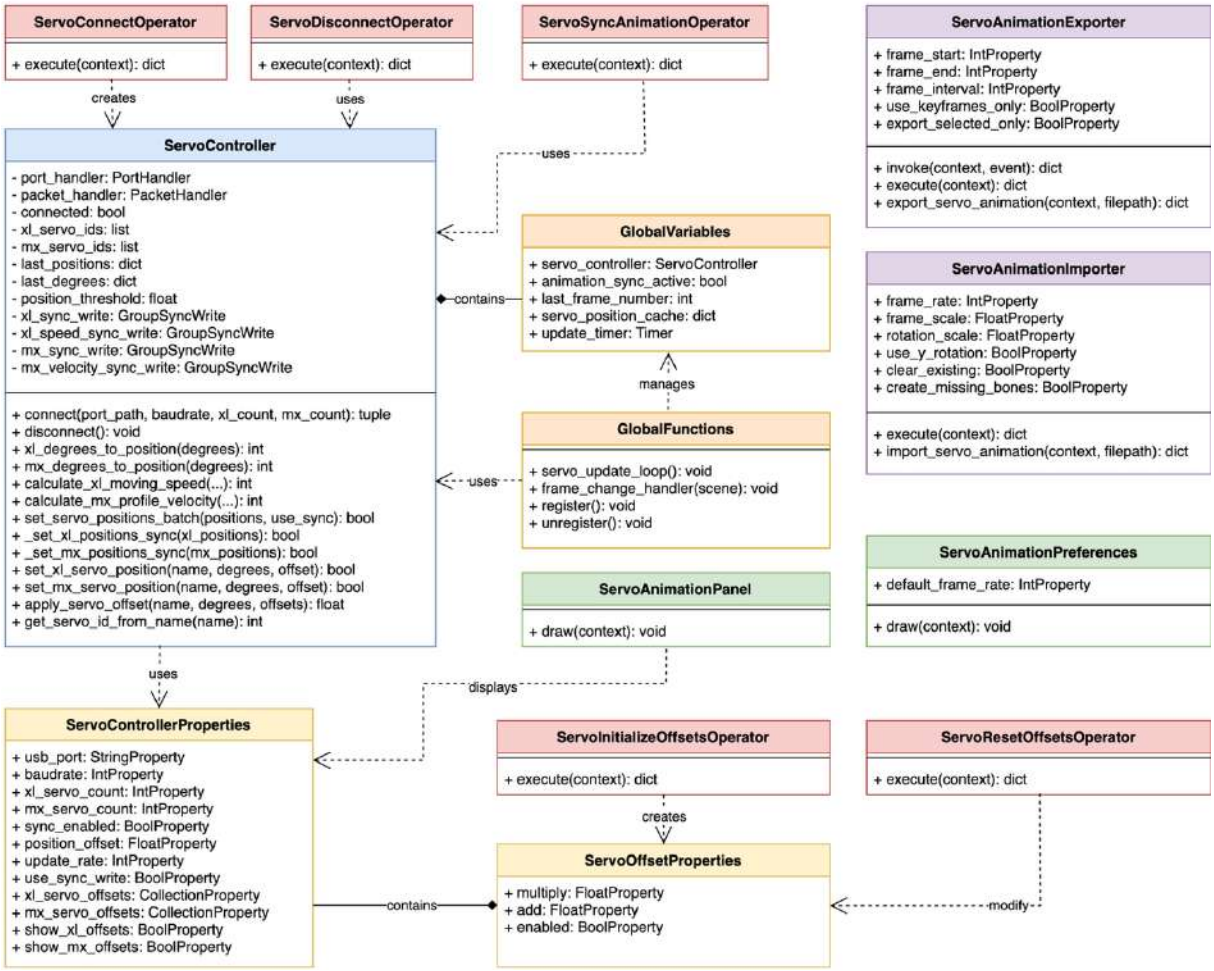
4.1.2.1 Arsitektur dan Alur Kerja *Plugin*

Plugin servo control dikembangkan sebagai Blender *add-on* yang mengintegrasikan kemampuan animasi 3D dengan kontrol perangkat keras robot humanoid. Implementasi *plugin* menggunakan arsitektur modular yang memisahkan fungsi-fungsi utama menjadi komponen-komponen yang dapat dikelola secara independen namun tetap saling terintegrasi.

Struktur *plugin* dirancang berdasarkan prinsip *separation of concerns*, dimana setiap kelas dan modul memiliki tanggung jawab yang spesifik. Pendekatan ini memungkinkan *maintainability* yang baik dan memudahkan pengembangan fitur tambahan di masa mendatang. *Plugin* memanfaatkan Blender *Python* API (bpy) secara ekstensif untuk mengakses data animasi, mengelola antarmuka pengguna, dan mengintegrasikan dengan *workflow* animator yang sudah ada.

Hubungan antar komponen dan struktur kelas dapat dilihat pada Gambar 4.11 yang menunjukkan diagram kelas lengkap dari *plugin*. Diagram ini memperlihatkan bagaimana kelas-kelas utama seperti `ServoController`, `GlobalVariables`, dan berbagai operator saling berinteraksi dalam arsitektur *plugin*.

Dari diagram kelas tersebut, dapat dilihat bahwa `ServoController` berfungsi sebagai jantung sistem yang menangani semua aspek komunikasi dengan *hardware servo*. Kelas ini memiliki atribut-atribut seperti `port_handler` dan `packet_handler` untuk mengelola koneksi fisik, `xl_servo_ids` dan `mx_servo_ids` untuk menyimpan daftar ID *servo* yang terhubung, serta `last_positions` dan `last_degrees` sebagai *cache* posisi terakhir *servo* untuk optimasi komunikasi. `ServoController` juga dilengkapi dengan berbagai *method* seperti `connect()` untuk membuka koneksi, `xl_degrees_to_position()` dan `mx_degrees_to_position()` untuk konversi satuan derajat ke satuan posisi *servo*, serta `set_servo_positions_batch()` untuk mengirim perintah posisi secara massal. Kelas `GlobalVariables` berperan sebagai penyimpan data bersama yang dapat diakses oleh semua komponen, termasuk `servo_controller` sebagai *instance* utama, `animation_sync_active` untuk status sinkronisasi, dan `servo_position_cache` sebagai *buffer* data posisi *servo*.



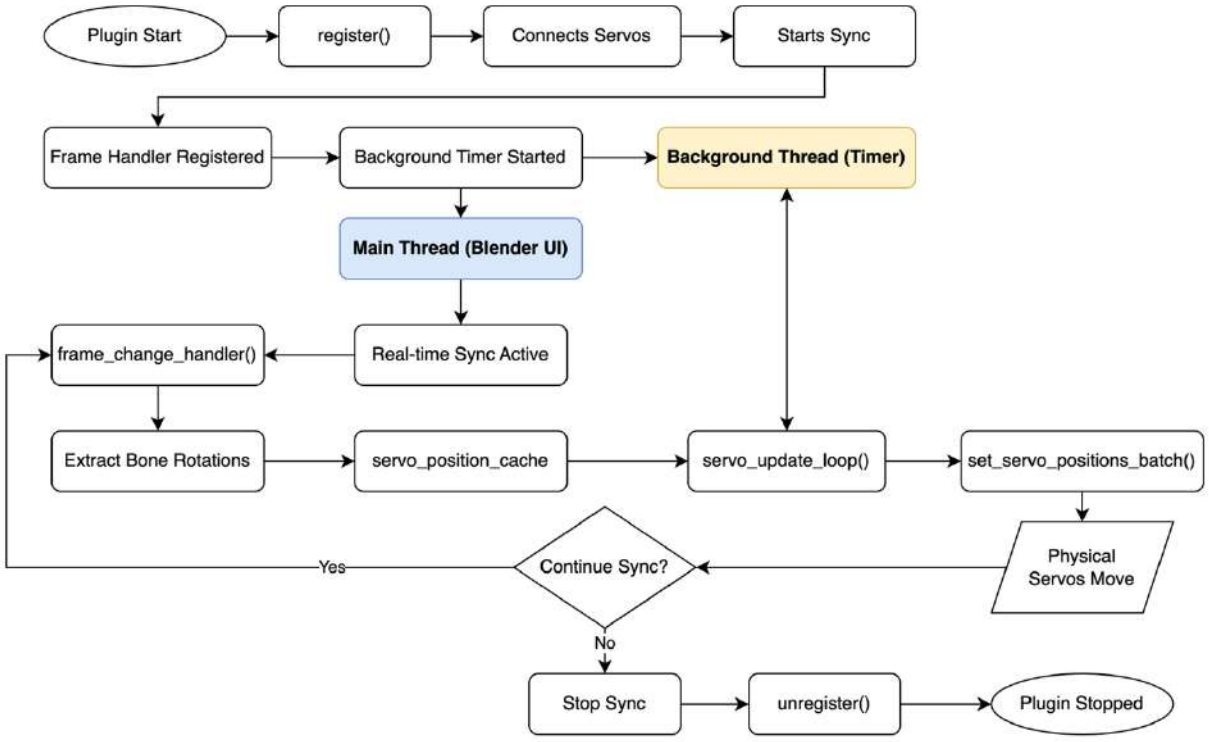
Gambar 4.11 Diagram kelas *plugin servo control*

Struktur operator dalam *plugin* dirancang mengikuti pola *inheritance* dari Blender API dimana setiap operator seperti `ServoConnectOperator`, `ServoDisconnectOperator`, dan `ServoSyncAnimationOperator` memiliki *method* `execute()` yang merupakan fungsi awal untuk menjalankan fungsi spesifik mereka. `ServoAnimationExporter` dan `ServoAnimationImporter` memiliki *properties* tambahan seperti `frame_start`, `frame_end`, dan `frame_interval` untuk mengatur parameter *export* dan *import*, serta method khusus `export_servo_animation()` dan `import_servo_animation()` untuk memproses data JSON. `ServoControllerProperties` dan `ServoOffsetProperties` menggunakan Blender property system dengan tipe-tipe seperti `StringProperty`, `IntProperty`, `FloatProperty`, dan `BoolProperty` untuk menyimpan konfigurasi yang dapat diakses melalui UI.

Hubungan antar kelas dalam diagram menunjukkan bagaimana masing-masing bagian saling berinteraksi. Misalnya, hubungan "*creates*" terlihat pada `ServoConnectOperator`, karena kelas ini membuat objek `ServoController` baru saat menyambungkan ke *hardware*. Hubungan "*uses*" ditunjukkan oleh `ServoDisconnectOperator` dan `ServoSyncAnimationOperator` yang menggunakan *instance* `ServoController` yang sudah ada untuk operasi *disconnect* dan sinkronisasi. Hubungan "*contains*" terlihat pada `ServoController` yang berisi `GlobalVariables` dan `ServoControllerProperties` yang mengandung `ServoOffsetProperties`. Sementara hubungan "*modify*" ditunjukkan oleh `ServoResetOffsetsOperator` yang mengubah nilai dalam `ServoOffsetProperties`.

Hubungan "*displays*" terlihat pada `ServoAnimationPanel` yang menampilkan `ServoControllerProperties`. Hubungan "*manages*" pada `GlobalFunctions` yang mengelola `ServoController` dan `GlobalVariables` secara keseluruhan.

Alur kerja *plugin* secara keseluruhan dapat dilihat pada Gambar 4.12 yang menunjukkan proses dari inisialisasi *plugin* hingga eksekusi sinkronisasi *real-time*. Proses dimulai dengan tahap registrasi *plugin* ke dalam sistem Blender melalui fungsi `register()` yang mendaftarkan semua komponen *plugin* seperti operator, *panel* UI, dan *properties* ke dalam Blender API. Setelah *plugin* berhasil terdaftar, sistem akan menjalankan fungsi *Connects Servos* yang melakukan inisialisasi koneksi hardware dengan *servo* motor melalui U2D2 *adapter* dan Dynamixel SDK. Ketika koneksi hardware berhasil terbentuk, proses berlanjut ke tahap *Starts Sync* dimana *plugin* memulai mode sinkronisasi *real-time* antara animasi Blender dengan gerakan *servo* fisik. Pada tahap ini terjadi registrasi *frame handler* melalui fungsi `frame_change_handler()` yang akan dipanggil secara otomatis setiap kali *frame* animasi Blender berubah, bersamaan dengan dimulainya *Background Timer* yang berjalan di *thread* terpisah untuk menangani komunikasi hardware tanpa memblokir antarmuka pengguna Blender.



Gambar 4.12 Diagram alur kerja *plugin servo control*

Saat mode *Real-time Sync* aktif, sistem akan memasuki *loop* utama sinkronisasi yang dimulai dengan pemanggilan `frame_change_handler()` setiap kali terjadi perubahan *frame*. *Handler* ini akan mengekstrak data rotasi *bone* dari *armature* robot dan menyimpannya dalam `servo_position_cache` sebagai *buffer* data posisi *servo*. Data dalam *cache* kemudian diproses oleh `servo_update_loop()` yang berjalan di *background thread* untuk mengkonversi rotasi *bone* menjadi nilai posisi *servo* yang sesuai, lalu memanggil fungsi `set_servo_positions_batch()` yang mengirimkan perintah posisi ke semua *servo* secara bersamaan. Setelah data berhasil dikirim, *servo* motor fisik akan bergerak sesuai dengan posisi yang diperintahkan. Sistem akan terus menjalankan loop sinkronisasi dengan melakukan

pengecekan *sync*. Jika sinkronisasi masih aktif, sistem akan kembali ke tahap `frame_change_handler()` untuk memproses *frame* berikutnya, namun jika pengguna memilih untuk menghentikan sinkronisasi, sistem akan membersihkan semua *handler* dan *timer* yang aktif, kemudian memanggil fungsi `unregister()` untuk menghapus *plugin* dari sistem Blender dan mengakhiri proses.

4.1.2.2 Alur Penggunaan *Plugin*

Untuk memahami cara kerja *plugin* dari perspektif pengguna, Gambar 4.13 menunjukkan *flowchart* alur penggunaan *plugin* dari awal hingga akhir. *Flowchart* ini menggambarkan *decision point* dan langkah-langkah yang harus dilakukan pengguna untuk berhasil menggunakan *plugin servo control* dalam *workflow* animasi robot.

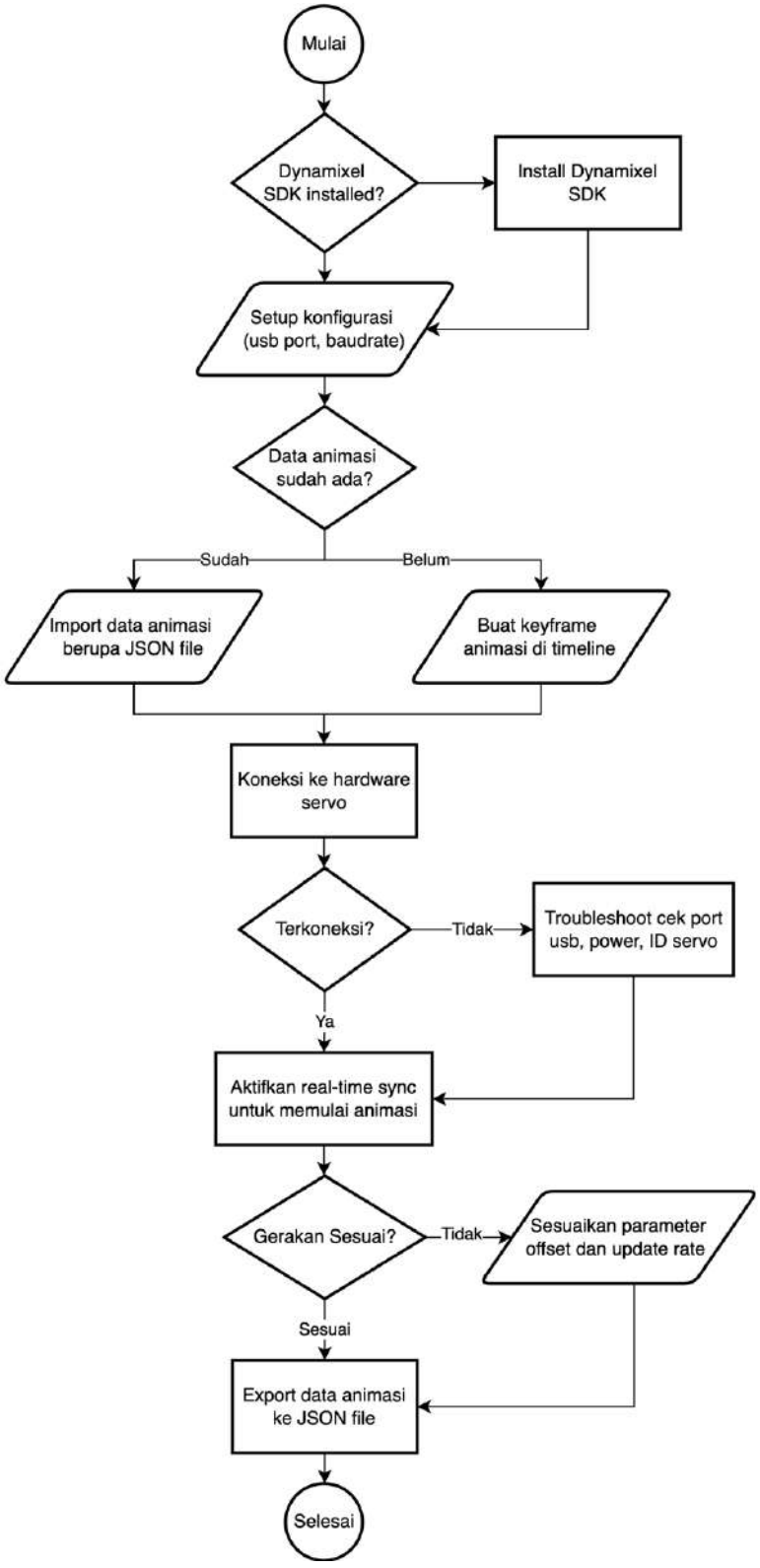
Sebelum penggunaan *plugin*, pengguna perlu memperhatikan konsistensi ID *servo* yang digunakan di Blender dengan ID fisik yang tertanam di masing-masing motor *servo*. Ketidaksesuaian ID dapat menyebabkan animasi tidak berjalan dengan benar atau tidak terjadi pergerakan sama sekali di sisi robot. Oleh karena itu, sebelum proses animasi dijalankan, pengguna disarankan untuk melakukan pengecekan kembali kesesuaian antara pengaturan JSON animasi dengan topologi *servo* yang sebenarnya.

Tahap awal penggunaan *plugin* melibatkan verifikasi ketersediaan Dynamixel SDK sebagai prasyarat utama untuk komunikasi dengan *hardware servo* robot. Jika SDK belum terinstal, pengguna perlu melakukan instalasi terlebih dahulu sebelum dapat menggunakan *plugin*. Setelah SDK tersedia, langkah selanjutnya adalah setup konfigurasi yang mencakup pengaturan USB *port* untuk komunikasi serial dan *baudrate* yang disesuaikan dengan spesifikasi *servo* yang digunakan. Konfigurasi ini penting untuk memastikan komunikasi yang stabil antara komputer dengan *hardware* robot.

Plugin menyediakan fleksibilitas dalam hal sumber data animasi dengan dua jalur kerja yang berbeda. Pengguna dapat memilih untuk mengimpor data animasi yang sudah ada dalam format JSON *file* atau membuat animasi baru dengan cara membuat *keyframe* di *timeline* Blender. Setelah data animasi tersedia, *plugin* akan menginisiasi proses koneksi ke *hardware servo* melalui antarmuka yang telah dikonfigurasi sebelumnya. Jika terjadi kegagalan koneksi, pengguna harus melakukan *troubleshooting* untuk mengidentifikasi masalah umum seperti kesalahan *port* USB, masalah *power supply*, atau konflik ID *servo*.

Ketika koneksi berhasil terbentuk, pengguna dapat mengaktifkan fitur *real-time sync*. Mode ini memungkinkan sinkronisasi langsung antara animasi yang diputar di Blender dengan gerakan fisik robot, memberikan *feedback* visual yang langsung bisa ditampilkan. Jika gerakan robot tidak sesuai dengan ekspektasi, pengguna dapat melakukan penyesuaian parameter *offset* untuk memperbaiki kalibrasi *servo* individual atau mengatur *update rate* untuk mengoptimalkan responsivitas sistem. Setelah animasi telah disesuaikan dan berjalan dengan memuaskan, pengguna dapat melakukan *export* data animasi ke format JSON untuk disimpan sebagai aset yang dapat digunakan kembali di animasi berikutnya, dibagikan dengan tim pengembang lainnya, atau digunakan ke dalam sistem yang lain.

Seluruh alur kerja yang difasilitasi oleh plugin dirancang untuk bersifat modular dan mudah digunakan oleh pengguna *non-programmer* sekalipun. Hal ini memungkinkan proses pembuatan, pengujian, dan penyesuaian animasi dapat dilakukan secara iteratif tanpa memerlukan pemrograman lanjutan.



Gambar 4.13 *Flowchart* alur penggunaan *plugin*

4.1.2.3 Integrasi dengan Blender API

Integrasi *plugin* dengan Blender API dilakukan melalui sistem properti yang terintegrasi dengan *scene properties* Blender. Hal ini memungkinkan pengaturan *plugin* tersimpan bersama dengan *file .blend* dan dapat diakses secara konsisten. Implementasi properti utama didefinisikan dalam kelas `ServoControllerProperties` yang mewarisi dari `PropertyGroup`.

Sistem properti mencakup konfigurasi komunikasi, pengaturan performa, dan *offset* individual untuk setiap *servo*. Setiap properti dilengkapi dengan metadata yang mendefinisikan tipe data, rentang nilai, dan deskripsi untuk antarmuka pengguna.

```
class ServoControllerProperties(PropertyGroup):
    """Properties for servo controller"""
    usb_port: StringProperty(
        name="USB Port",
        description="Path to USB port for Dynamixel communication",
        default="/dev/tty.usbmodem58FA1013501"
    )

    baudrate: IntProperty(
        name="Baudrate",
        description="Communication baudrate",
        default=1000000,
        min=9600,
        max=4000000
    )

    xl_servo_count: IntProperty(
        name="XL-320 Count",
        description="Number of XL-320 servos (ID 1-N)",
        default=15,
        min=1,
        max=50
    )

    mx_servo_count: IntProperty(
        name="MX-28 Count",
        description="Number of MX-28 servos (ID 1-N)",
        default=14,
        min=1,
        max=50
    )
```

Kode Sumber 4.1 Contoh definisi properti utama *plugin*

Kode Sumber 4.1 menunjukkan contoh implementasi properti menggunakan sistem `bpy.props` yang merupakan salah satu komponen inti Blender API. Setiap properti didefinisikan dengan tipe data spesifik seperti `StringProperty` untuk *port* USB, `IntProperty` untuk *baudrate* dan jumlah *servo*, serta dilengkapi dengan metadata seperti nama, deskripsi, nilai *default*, dan rentang nilai. Sistem properti ini terintegrasi langsung dengan *scene* Blender melalui `PropertyGroup`, memungkinkan pengaturan tersimpan bersama *file* `.blend` dan dapat diakses secara konsisten sepanjang sesi animasi. Properti `usb_port` menggunakan *path default* macOS untuk komunikasi serial, sedangkan `baudrate` ditetapkan pada 1 Mbps yang merupakan nilai optimal untuk komunikasi Dynamixel *servo*.

Plugin juga mengimplementasikan *frame change handler* yang terintegrasi dengan sistem animasi Blender. *Handler* ini memungkinkan *plugin* untuk merespons perubahan *frame* secara *real-time* dan mengirimkan data posisi *servo* yang sesuai. Implementasi *frame change handler* menggunakan `bpy.app.handlers.frame_change_post` yang merupakan callback *system* Blender untuk mendeteksi perubahan *frame* animasi.

```
1  def frame_change_handler(scene):
2      """Handler yang dipanggil setiap frame berubah"""
3      global servo_controller, animation_sync_active,
   servo_position_cache
4
5      if not animation_sync_active or not servo_controller or not
   servo_controller.connected:
6          return
7
8      # Akses frame saat ini menggunakan bpy API
9      current_frame = scene.frame_current
10     props = scene.servo_props
11
12     # Get active armature menggunakan bpy.context
13     armature_obj = None
14     if bpy.context.active_object and bpy.context.active_object.type
   == 'ARMATURE':
15         armature_obj = bpy.context.active_object
16
17     if not armature_obj:
18         return
19
20     # Iterasi bones dan akses rotasi menggunakan bpy bone system
21     new_positions = {}
22     for bone in armature_obj.pose.bones:
23         if bone.name.startswith(('XL', 'MX')):
24             # Akses rotasi Y-axis menggunakan bone.rotation_euler
25             rotation_rad = bone.rotation_euler[1]
26             rotation_deg = math.degrees(rotation_rad) +
   props.position_offset
27             new_positions[bone.name] = rotation_deg
28
29     # ... proses positioning dan caching
30     servo_position_cache = new_positions
31
32 # Registrasi handler ke sistem Blender
33 if frame_change_handler not in bpy.app.handlers.frame_change_post:
34     bpy.app.handlers.frame_change_post.append(frame_change_handler)
```

Kode Sumber 4.2 Implementasi *frame change handler* dengan integrasi bpy API

Frame change handler pada Kode Sumber 4.2 menunjukkan cara menggunakan Blender API melalui modul bpy. *Handler* ini mengambil *frame* saat ini dengan `scene.frame_current`, lalu menggunakan `bpy.context.active_object` untuk mengetahui *armature* mana yang sedang aktif. Setelah itu, *handler* mengakses data rotasi dari setiap *bone* melalui `armature_obj.pose.bones`.

Handler ini didaftarkan ke sistem *event* Blender dengan `bpy.app.handlers.frame_change_post.append()`, yang menambahkan fungsi *callback* agar dijalankan setiap kali *frame* animasi berubah. Selain itu, *handler* ini juga terhubung dengan sistem properti *plugin* melalui `scene.servo_props`, yang merujuk pada objek `ServoControllerProperties`. Dengan begitu, *plugin* dapat membaca pengaturan konfigurasi secara langsung saat animasi berlangsung secara *real-time*.

4.1.2.4 Implementasi Kontrol *Real-time*

Plugin menyediakan kontrol *real-time* yang memungkinkan sinkronisasi langsung antara animasi Blender dengan pergerakan robot fisik. Sistem ini diimplementasikan melalui *background threading* yang memisahkan komunikasi *servo* dari *main* UI *thread* Blender untuk menjaga responsivitas antarmuka pengguna.

Implementasi *threading* menggunakan *background timer* yang dapat dikonfigurasi dan mengelola pengiriman data posisi secara berkala seperti yang terlihat pada fungsi `servo_update_loop` pada Kode Sumber 4.3.

```
1  def servo_update_loop():
2      """Background thread untuk update servo secara berkala"""
3      global animation_sync_active, servo_position_cache, update_timer
4
5      try:
6          if animation_sync_active and servo_controller and
   servo_controller.connected:
7              # Get current scene
8              if hasattr(bpy.context, 'scene') and bpy.context.scene:
9                  scene = bpy.context.scene
10                 props = scene.servo_props
11
12                 if servo_position_cache:
13
14                     servo_controller.set_servo_positions_batch(
15                         servo_position_cache.copy(),
16                         props.use_sync_write
17                     )
18
19         # Schedule next update
20         if animation_sync_active:
```

```
            try:
                if hasattr(bpy.context, 'scene') and bpy.context.scene:
                    scene = bpy.context.scene
                    props = scene.servo_props
                    update_interval = 1.0 / max(1, props.update_rate)

                    update_timer = threading.Timer(update_interval, servo_update_loop)
                    update_timer.daemon = True
                    update_timer.start()
            except Exception as e:
                print(f"Servo update scheduling error: {e}")
                animation_sync_active = False

    except Exception as e:
        print(f"Servo update error: {e}")
```

Kode Sumber 4.3 Implementasi *background threading* untuk *real-time control*

Sistem *real-time control* yang terlihat pada Kode Sumber 4.3 menggunakan *position caching* dan *update scheduling* untuk mengoptimalkan performa komunikasi. Implementasi *threading background* memisahkan proses komunikasi *servo* dari *main* UI *thread* Blender, memastikan antarmuka pengguna tetap responsif meskipun terjadi latensi komunikasi serial dengan robot. Fungsi `servo_update_loop()` beroperasi sebagai *daemon thread* yang berjalan secara rekursif menggunakan `threading.Timer`, memungkinkan kontrol dinamis terhadap frekuensi *update* berdasarkan properti `update_rate` yang dapat dikonfigurasi pengguna.

Arsitektur *position caching* diimplementasikan melalui variabel *global variable* `servo_position_cache` yang menyimpan data posisi *servo* terbaru dari *frame change handler*. Pendekatan ini menghindari akses langsung ke Blender API dari *background thread* yang dapat menyebabkan *race condition* atau *crash* aplikasi. Data posisi yang di-*cache* kemudian dikirimkan ke *servo* melalui `set_servo_positions_batch()` yang menggunakan Dynamixel *sync write protocol* untuk komunikasi langsung dengan banyak *servo*.

Error handling diimplementasikan secara bertingkat dengan *exception catching* pada *level individual update* dan *scheduling level*. Sistem dirancang untuk melanjutkan operasi meskipun terjadi *error* komunikasi sesekali, namun akan menghentikan *sync loop* jika terjadi *error scheduling* yang mengindikasikan masalah fundamental. *Interval update* dihitung dinamis menggunakan `1.0 / max(1, props.update_rate)` untuk mencegah pembagian dengan 0 dan memungkinkan penyesuaian *real-time* terhadap beban sistem. Kombinasi *frame change handler* dan *background threading* menciptakan arsitektur *dual-layer* yang optimal untuk *real-time servo control* dengan responsivitas UI yang terjaga.

4.1.2.5 Sistem *Import* dan *Export* Animasi JSON

Plugin menyediakan fitur *import* dan *export* data animasi dalam format JSON yang memungkinkan penyimpanan dan berbagi sekuens gerakan robot dengan pengembang lain. Fitur ini memungkinkan animator untuk menyimpan hasil kerja dalam format yang portabel

dan dapat digunakan pada sistem lain atau untuk keperluan *backup*. Implementasi *export* animasi menggunakan kelas `ServoAnimationExporter` yang mewarisi dari `ExportHelper` untuk integrasi dengan *file browser* Blender.

Proses *export animation* dimulai dengan identifikasi objek *armature* yang aktif melalui `context.active_object`, kemudian mengiterasi seluruh pose *bones* untuk mengekstrak data rotasi pada *Y-axis*. Algoritma *export* yang ditunjukkan pada Kode Sumber 4.4 mengimplementasikan optimasi untuk mengurangi ukuran *file* dengan hanya menyimpan *keyframe* yang memiliki perubahan rotasi signifikan lebih dari 0.01 derajat.

```
1  def export_servo_animation(self, context, filepath):
2      """Export animation data to JSON file"""
3      xl_servos = {}  # Dictionary for XL servos
4      mx_servos = {}  # Dictionary for MX servos
5
6      # Get active armature dan validasi
7      armature_obj = None
8      if context.active_object and context.active_object.type == 
'ARMATURE':
9          armature_obj = context.active_object
10     # ... pencarian armature alternatif jika tidak ada object aktif
11
12     if not armature_obj:
13         self.report({'ERROR'}, "No armature found!")
14         return {'CANCELLED'}
15
16     # Iterasi frame dan ekstrak data rotasi
17     for bone in armature_obj.pose.bones:
18         if not bone.name.startswith(('XL', 'MX')):
19             continue
20
21         keyframes = []
22         for frame in range(self.frame_start, self.frame_end + 1, 
self.frame_interval):
23             context.scene.frame_set(frame)
24
25             # Ekstrak rotasi Y-axis dalam derajat
26             rotation_rad = bone.rotation_euler[1]
27             rotation_deg = math.degrees(rotation_rad)
28
29             # Optimasi: hanya simpan perubahan signifikan
30             if last_rotation is None or abs(rotation_deg - 
last_rotation) > 0.01:
31                 keyframes.append({"frame": frame, "rotation": 
round(rotation_deg, 2)})
32                 last_rotation = rotation_deg
```

```
33
34      # ... pengelompokan servo XL dan MX serta serialisasi JSON
```

Kode Sumber 4.4 Implementasi *export animation* ke format JSON

Proses *export* menggunakan `context.scene.frame_set()` untuk navigasi timeline dan `bone.rotation_euler[1]` untuk akses data rotasi *Y-axis* yang merupakan *axis* utama pergerakan *servo*. Data hasil ekstraksi dikelompokkan berdasarkan jenis *servo* (XL-320 dan MX-28) kemudian diserialisasi ke format JSON dengan struktur hierarkis yang terorganisir. Format JSON yang dihasilkan menggunakan struktur *key-value* dengan nama *servo* sebagai *key* dan *array keyframe* sebagai *value* seperti yang terlihat pada Kode Sumber 4.5.

```
1  {
2      "XL1": [
3          { "frame": 1, "rotation": 0.0 },
4          { "frame": 30, "rotation": 45.5 },
5          { "frame": 60, "rotation": -30.2 },
6          { "frame": 90, "rotation": 0.0 }
7      ],
8      "XL2": [
9          { "frame": 1, "rotation": 0.0 },
10         { "frame": 30, "rotation": -22.8 },
11         { "frame": 60, "rotation": 15.3 },
12         { "frame": 90, "rotation": 0.0 }
13     ],
14     "MX1": [
15         { "frame": 1, "rotation": 0.0 },
16         { "frame": 30, "rotation": 90.0 },
17         { "frame": 60, "rotation": -45.0 },
18         { "frame": 90, "rotation": 0.0 }
19     ],
20     "MX2": [
21         { "frame": 1, "rotation": 0.0 },
22         { "frame": 30, "rotation": -60.5 },
23         { "frame": 60, "rotation": 30.0 },
24         { "frame": 90, "rotation": 0.0 }
25     ]
26 }
```

Kode Sumber 4.5 Format JSON fitur *import export*

Struktur JSON di atas menunjukkan format data animasi untuk 4 *servo* (XL1, XL2, MX1, MX2) dengan sekuens gerakan yang terdiri dari 4 *keyframe*. Setiap *keyframe* berisi informasi `frame` yang menunjukkan posisi dalam *timeline* dan `rotation` yang merepresentasikan sudut rotasi dalam derajat. Format ini memungkinkan penyimpanan data animasi yang kompak namun tetap bisa dibaca untuk keperluan *debugging* atau modifikasi manual.

Fitur *import animation* diimplementasikan melalui kelas `ServoAnimationImporter` yang mewarisi dari `ImportHelper` untuk kompatibilitas dengan sistem *file handling* Blender. Proses *import* memungkinkan rekonstruksi animasi dari data JSON dengan berbagai opsi konfigurasi

seperti *frame scaling* dan *rotation scaling* untuk adaptasi dengan berbagai *workflow* pengembangan.

```
1  def import_servo_animation(self, context, filepath):
2      try:
3          # Load dan parse JSON data
4          with open(filepath, 'r', encoding='utf-8') as file:
5              servo_data = json.load(file)
6
7          # Setup armature dan animation action
8          armature_obj = # ... identifikasi armature object
9
10         if not armature_obj.animation_data:
11             armature_obj.animation_data_create()
12
13         action = bpy.data.actions.new(name="ServoAnimation")
14         armature_obj.animation_data.action = action
15
16         # Proses setiap servo dan generate keyframes
17         for servo_name, keyframes in servo_data.items():
18             bone = armature_obj.pose.bones.get(servo_name)
19             # ... handling bone yang tidak ditemukan atau dibuat
   otomatis
20
21             for keyframe_data in keyframes:
22                 frame_num = int(keyframe_data['frame'] *
   self.frame_scale)
23                 rotation_deg = float(keyframe_data['rotation'])
24                 rotation_rad = math.radians(rotation_deg *
   self.rotation_scale)
25
26                 # Set frame dan apply rotasi
27                 scene.frame_set(frame_num)
28                 bone.rotation_euler[1] = rotation_rad
29                 bone.keyframe_insert(data_path="rotation_euler",
   index=1)
30
31     except json.JSONDecodeError as e:
32         self.report({'ERROR'}, f"Invalid JSON file: {e}")
33         return {'CANCELLED'}
```

Kode Sumber 4.6 Implementasi *import animation* dari format JSON

Proses import pada Kode Sumber 4.6 dimulai dengan memeriksa apakah file JSON yang dimasukkan memiliki format yang valid. Setelah itu, *plugin* membuat struktur data animasi baru dengan `bpy.data.actions.new()` dan membangun kembali *keyframe* animasi menggunakan `bone.keyframe_insert()`. Sistem ini juga mendukung pengaturan skala waktu (*frame*

scaling) dan skala rotasi (*rotation scaling*), sehingga animasi bisa disesuaikan dengan kebutuhan alur kerja yang berbeda.

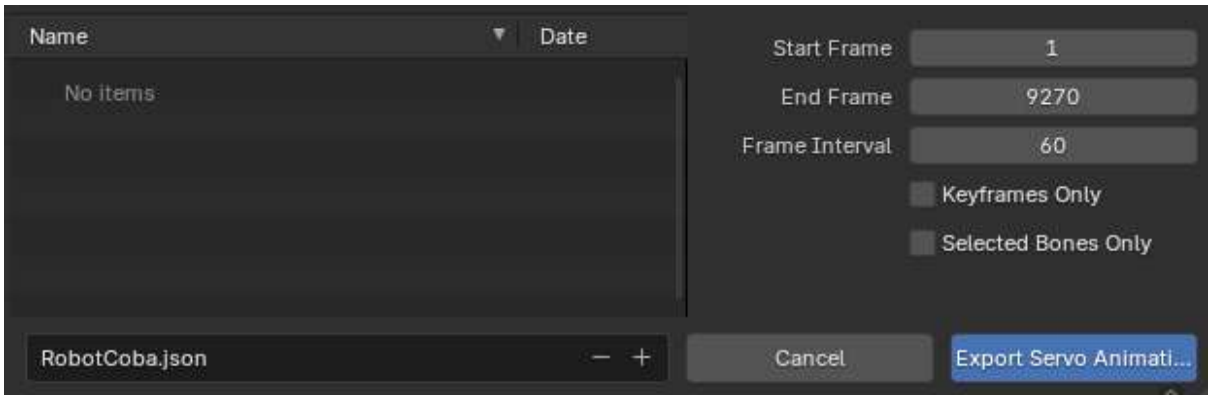
Untuk menjaga kestabilan, *plugin* juga dilengkapi dengan *error handling* yang akan mendeteksi jika file JSON rusak atau formatnya tidak sesuai, sehingga proses tidak menyebabkan *crash*. *Plugin* ini menyediakan antarmuka pengguna yang sudah terintegrasi dengan menu dan dialog di Blender, sehingga fitur *import* dan *export* animasi bisa diakses dengan mudah. Fitur ini juga terhubung langsung dengan *file browser* Blender dan dapat diakses melalui menu *File > Import* dan *Export*, yang sudah umum digunakan oleh pengguna Blender, seperti yang ditampilkan pada Gambar 4.14.

Antarmuka *import* dan *export* yang ditunjukkan pada Gambar 4.14 menampilkan dua opsi yaitu "*Import Animation*" dan "*Export Animation*" dengan ikon yang sesuai. Setiap operasi membuka dialog konfigurasi yang memungkinkan pengguna untuk menyesuaikan parameter sesuai kebutuhan *workflow* mereka.

Gambar 4.14 *Screenshot dialog import* dan *export*

Dialog *export* menyediakan kontrol yang lebih rinci terhadap proses ekstraksi data animasi dengan berbagai opsi konfigurasi, seperti yang ditampilkan pada Gambar 4.15. Pengguna dapat menentukan rentang *frame* yang akan di-*export* melalui pengaturan *Start Frame* dan *End Frame*, serta mengkonfigurasi *Frame Interval* untuk mengoptimalkan ukuran *file output*. Pengaturan ini berguna untuk mengurangi ukuran *file* hasil *export* tanpa mengorbankan detail animasi yang penting.

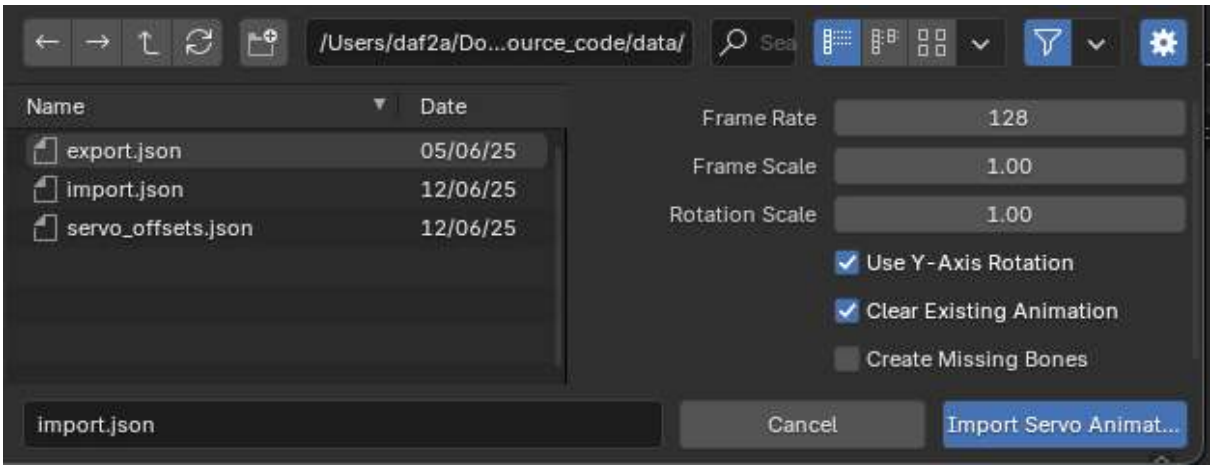
Fitur "*Keyframes Only*" dan "*Selected Bones Only*" yang terlihat pada Gambar 4.15 memungkinkan *export* hanya pada *frame* yang memiliki *keyframe animation* dan membatasi *export* pada *bone* yang dipilih pengguna. Kombinasi opsi-opsi ini memberikan fleksibilitas dalam menentukan *scope* dan detail data yang akan di-*export*.



Gambar 4.15 Konfigurasi *export*

Dialog *import animation* menyediakan kontrol terhadap aspek rekonstruksi animasi dari *file* JSON, sebagaimana terlihat pada Gambar 4.16. *Frame Rate* digunakan untuk menentukan kecepatan pemutaran animasi yang di-*import*. Sementara itu, *Frame Scale* dan *Rotation Scale* memungkinkan penyesuaian terhadap waktu dan besar gerakan, sehingga animasi dapat disesuaikan dengan kebutuhan atau gaya animasi tertentu.

Opsi "*Use Y-Axis Rotation*", "*Clear Existing Animation*", dan "*Create Missing Bones*" yang ditampilkan pada Gambar 4.16 masing-masing memastikan rotasi diterapkan pada *axis* yang sesuai dengan konvensi *servo bones*, membersihkan *keyframe* sebelumnya, dan secara otomatis membuat *bone* yang belum ada dalam *armature*. Antarmuka ini dirancang untuk memberikan kontrol maksimal sambil mempertahankan kemudahan penggunaan untuk *workflow* yang berbeda.



Gambar 4.16 Konfigurasi *import*

4.1.2.6 Integrasi Dynamixel SDK

Integrasi dengan Dynamixel SDK diimplementasikan melalui kelas ServoController yang menyediakan abstraksi *high-level* untuk komunikasi dengan *servo*. Kelas ini mengelola koneksi, protokol komunikasi, dan optimasi performa untuk kontrol banyak *servo* secara bersamaan.

ServoController didesain untuk menangani dua jenis *servo* dengan karakteristik komunikasi yang berbeda. XL-320 menggunakan 2-*byte position values* dengan protokol TTL, sedangkan MX-28 menggunakan 4-*byte position values* dengan protokol yang sama namun parameter yang berbeda. Implementasi ini memungkinkan *plugin* untuk bekerja dengan konfigurasi *servo* yang heterogen. Kode Sumber 4.9 menunjukkan inisialisasi ServoController dengan parameter *servo*.

```
1  class ServoController:
2      """Class untuk mengontrol servo Dynamixel"""
3
4      def __init__(self):
5          self.port_handler = None
6          self.packet_handler = None
7          self.connected = False
8          self.xl_servo_ids = []  # XL-320 servos
9          self.mx_servo_ids = []  # MX-28 servos
```

```
10          self.last_positions = {}
11          self.last_degrees = {}
12          self.position_threshold = 2
13
14          # Protocol version
15          self.PROTOCOL_VERSION = 2.0
16
17          # XL-320 specific control table addresses
18          self.ADDR_XL_TORQUE_ENABLE = 24
19          self.ADDR_XL_GOAL_POSITION = 30
20          self.ADDR_XL_PRESENT_POSITION = 37
21          self.ADDR_XL_MOVING_SPEED = 32
22          self.XL_CENTER_POSITION = 512
23          self.XL_POSITION_RANGE = 1024
24          self.XL_MAX_POSITION = 1023
25          self.XL_MIN_POSITION = 0
26          self.XL_SERVO_RANGE = 300.0
27
28          # MX-28 specific control table addresses
29          self.ADDR_MX_TORQUE_ENABLE = 64
30          self.ADDR_MX_GOAL_POSITION = 116
31          self.ADDR_MX_PRESENT_POSITION = 132
32          self.ADDR_MX_PROFILE_VELOCITY = 112
33          self.ADDR_MX_PROFILE_ACCELERATION = 108
34          self.ADDR_MX_OPERATING_MODE = 11
35          self.ADDR_MX_VELOCITY_LIMIT = 44
36          self.MX_CENTER_POSITION = 2048
37          self.MX_POSITION_RANGE = 4096
38          self.MX_MAX_POSITION = 4095
39          self.MX_MIN_POSITION = 0
40          self.MX_SERVO_RANGE = 360.0
```

Kode Sumber 4.7 Inisialisasi ServoController dengan parameter *servo*

Sistem yang terlihat pada Kode Sumber 4.7 mengimplementasikan *automatic servo type detection* yang memungkinkan *plugin* untuk secara otomatis menentukan parameter komunikasi yang tepat berdasarkan nama *bone*. Implementasi ini mengacu pada spesifikasi *control table* dari dokumentasi resmi Robotis untuk memastikan kompatibilitas penuh dengan protokol Dynamixel. Tabel 4.1 menunjukkan *control table* yang penting untuk *servo* XL-320 yang digunakan dalam sistem.

Tabel 4.1 *Control Table* XL-320 yang Digunakan Dalam *Plugin*

Address	*Size* (Byte)	Data *Name*	*Access*	*Initial Value*
24	1	*Torque Enable*	RW	0
30	2	*Goal Position*	RW	-
32	2	*Moving Speed*	RW	-
37	2	*Present Position*	R	-

Tabel 4.1 menunjukkan *control table* untuk *servo* XL-320, yang memiliki sistem alamat yang berbeda dibandingkan dengan seri MX karena protokolnya dirancang agar lebih hemat. Misalnya, alamat 24 digunakan untuk mengaktifkan atau menonaktifkan torsi (*Torque Enable*) dengan ukuran 1 *byte*, di mana nilai 0 berarti nonaktif dan 1 berarti aktif. Sementara itu, *Goal Position* dan *Present Position* menggunakan 2 *byte*, dengan rentang nilai dari 0 hingga 1023, yang mewakili rotasi hingga 300 derajat.

Servo MX-28 menggunakan *control table* yang lebih kompleks dengan alamat dan fitur yang lebih canggih, seperti yang ditunjukkan pada Tabel 4.2. Sistem ini mendukung *Profile Velocity* dan *Profile Acceleration* untuk kontrol gerakan yang lebih halus.

Tabel 4.2 *Control Table* MX-28 yang Digunakan Dalam *Plugin*

Address	*Size* (Byte)	Data *Name*	*Access*	*Initial Value*
11	1	*Operating* Mode	RW	3
44	4	*Velocity Limit*	RW	0
64	1	*Torque Enable*	RW	0
108	4	*Profile Acceleration*	RW	0
112	4	*Profile Velocity*	RW	0
116	4	*Goal Position*	RW	-
132	4	*Present Position*	R	-

Control table MX-28 pada Tabel 4.2 menunjukkan kompleksitas yang lebih tinggi dengan penggunaan 4-*byte values* untuk *position* dan *velocity parameters*. Alamat 11 (*Operating Mode*) diatur ke nilai 3 untuk mengaktifkan *Position Control Mode*. Sementara itu, *Profile Velocity* dan *Profile Acceleration* memungkinkan pengaturan karakteristik kecepatan dan percepatan *servo* secara lebih fleksibel, yang mempermudah proses konfigurasi dan mengurangi risiko kesalahan saat menyambungkan *servo*.

Implementasi konkret penggunaan *control table* dapat dilihat pada metode konversi posisi dan *sync write operation* yang memanfaatkan alamat dan data format sesuai spesifikasi. Kode Sumber 4.8 menunjukkan contoh implementasi untuk *sync_write servo* XL-320 yang menggunakan alamat dan format data dari Tabel 4.2.

```
1  def _set_xl_positions_sync(self, xl_positions):
2      """Set XL-320 positions using sync write dengan implementasi Dynamixel SDK"""
3      try:
4          self.xl_sync_write.clearParam()
5          self.xl_speed_sync_write.clearParam()
6
7          updated_count = 0
8          for servo_name, rotation_degrees in xl_positions.items():
9              servo_id = self.get_servo_id_from_name(servo_name)
10             if servo_id is None:
11                 continue
12
13             position = self.xl_degrees_to_position(rotation_degrees)
```

```
14
15              position_threshold_units = int(self.position_threshold * self XL_POSITION_RANGE / self.XL_SERVO_RANGE)
16
17              if (servo_name not in self.last_positions or
18                  abs(position - self.last_positions[servo_name]) > position_threshold_units):
19
20                  previous_degrees = self.last_degrees.get(servo_name)
21                  moving_speed = self.calculate_xl_moving_speed(rotation_degrees, previous_degrees, 200)
22
23                  speed_bytes = [DXL_LOBYTE(moving_speed), DXL_HIBYTE(moving_speed)]
24                  self.xl_speed_sync_write.addParam(servo_id, speed_bytes)
25
26                  position_bytes = [DXL_LOBYTE(position), DXL_HIBYTE(position)]
27
28                  if self.xl_sync_write.addParam(servo_id, position_bytes):
29                      self.last_positions[servo_name] = position
30                      self.last_degrees[servo_name] = rotation_degrees
31                      updated_count += 1
32
33          if updated_count > 0:
34              self.xl_speed_sync_write.txPacket()
35              self.xl_sync_write.txPacket()
36
37          return True
38      except Exception as e:
39          print(f"XL sync write error: {e}")
40          return False
```

Kode Sumber 4.8 Implementasi *sync write* XL-320 dengan Dynamixel SDK

Implementasi pada Kode Sumber 4.8 mendemonstrasikan penggunaan GroupSyncWrite dari Dynamixel SDK untuk komunikasi efisien dengan banyak *servo* XL-320. *Method* `clearParam()` membersihkan parameter sebelumnya, sedangkan `addParam()` dipakai untuk menambahkan data ke masing-masing *servo*, menggunakan format 2-*byte* sesuai dengan spesifikasi XL-320. Penggunaan `DXL_LOBYTE()` dan `DXL_HIBYTE()` memastikan urutan *byte* yang benar untuk komunikasi serial, sementara `txPacket()` akan mengirim seluruh data tersebut sekaligus dalam satu siklus komunikasi. Urutan pengiriman dimulai dari kecepatan

terlebih dahulu, kemudian posisi, agar gerakan *servo* dapat merespons dengan kecepatan yang sesuai secara lebih halus dan tepat.

4.1.2.7 Algoritma Konversi Data Rotasi ke Nilai Posisi *Servo*

Sistem konversi didasarkan pada pemahaman bahwa Blender menggunakan sistem koordinat berbasis radian untuk rotasi, sementara servo Dynamixel menggunakan unit posisi integer yang bervariasi tergantung tipe *servo*. Servo XL-320 memiliki rentang posisi 0-1023 *unit* (300°) dengan *center position* 512, sedangkan MX-28 menggunakan rentang 0-4095 *unit* (360°) dengan *center position* 2048.

Kode Sumber 4.9 menunjukkan implementasi algoritma konversi untuk kedua jenis servo dengan pendekatan *linear mapping* dan *boundary checking*.

```
1  def xl_degrees_to_position(self, degrees):
2      half_range = self.XL_SERVO_RANGE / 2.0  # 150.0 derajat
3      degrees = max(-half_range, min(half_range, degrees))
4
5      position_per_degree = self.XL_POSITION_RANGE /
   self.XL_SERVO_RANGE
6      position = int(self.XL_CENTER_POSITION + (degrees *
   position_per_degree))
7      return max(self.XL_MIN_POSITION, min(self.XL_MAX_POSITION,
   position))
8
9  def mx_degrees_to_position(self, degrees):
10     half_range = self.MX_SERVO_RANGE / 2.0  # 180.0 derajat
11     degrees = max(-half_range, min(half_range, degrees))
12
13     position_per_degree = self.MX_POSITION_RANGE /
   self.MX_SERVO_RANGE
14     position = int(self.MX_CENTER_POSITION + (degrees *
   position_per_degree))
15     return max(self.MX_MIN_POSITION, min(self.MX_MAX_POSITION,
   position))
16
17 def apply_servo_offset(self, servo_name, rotation_degrees,
   offsets_collection):
18     for offset in offsets_collection:
19         if offset.name == servo_name and offset.enabled:
20             return (rotation_degrees * offset.multiply) + offset.add
21     return rotation_degrees
```

Kode Sumber 4.9 Algoritma konversi rotasi untuk *servo* XL-320 dan MX-28

Algoritma konversi terintegrasi dalam `frame_change_handler()` yang memproses rotasi semua *bone* secara *real-time*. Proses dimulai dengan ekstraksi rotasi *Y-axis* dari *bone* (dalam radian), konversi ke derajat menggunakan `math.degrees()`, kemudian diaplikasikan global

dan individual *offset*, setelah itu disimpan dalam *caching* untuk dilanjutkan pada *background thread processing*.

Sistem offset individual memungkinkan *fine-tuning* setiap *servo* dengan formula *multiply-then-add* untuk mengkompensasi perbedaan mekanis dan toleransi *manufacturing*. Parameter *multiply* dapat digunakan untuk membalik arah servo atau mengatur skala rentang *motion*, sementara parameter *add* mengkompensasi *mechanical bias*. Implementasi ini memastikan akurasi posisi optimal untuk setiap servo dalam konfigurasi robot humanoid 17-DOF.

4.1.2.8 Konfigurasi Komunikasi U2D2

Konfigurasi komunikasi U2D2 diimplementasikan dalam metode *connect* yang mengelola inisialisasi koneksi, pengaturan *baudrate*, dan validasi komunikasi dengan setiap *servo*. Proses koneksi dilakukan secara bertahap dengan *error handling* untuk memastikan stabilitas sistem.

Proses koneksi dimulai dengan inisialisasi PortHandler dan PacketHandler dari Dynamixel SDK. *Baudrate* dikonfigurasi sesuai dengan spesifikasi *servo* (*default* 1 Mbps) untuk memastikan *throughput* tinggi yang diperlukan untuk kontrol *real-time* 17 *servo* secara bersamaan. Kode Sumber 4.10 menunjukkan implementasi koneksi U2D2 dan *setup servo* IDs dengan rentang yang berbeda untuk menghindari konflik.

```
1  def connect(self, port_path, baudrate, xl_count, mx_count):
2      """Connect to Dynamixel servos"""
3      try:
4          print(f"Attempting to connect: Port={port_path}, Baudrate={baudrate}")
5          print(f"Servo counts: XL={xl_count}, MX={mx_count}")
6
7          self.port_handler = PortHandler(port_path)
8          self.packet_handler = PacketHandler(self.PROTOCOL_VERSION)
9
10         if not self.port_handler.openPort():
11             return False, "Failed to open port"
12
13         if not self.port_handler.setBaudRate(baudrate):
14             return False, "Failed to set baudrate"
15
16         print("Port opened successfully")
17
18         # Setup servo IDs: Use different ID ranges to avoid conflicts
19         self.xl_servo_ids = list(range(1, xl_count + 1))
20         self.mx_servo_ids = list(range(100, 100 + mx_count))
```

Kode Sumber 4.10 Implementasi koneksi U2D2 dan *setup servo* ID

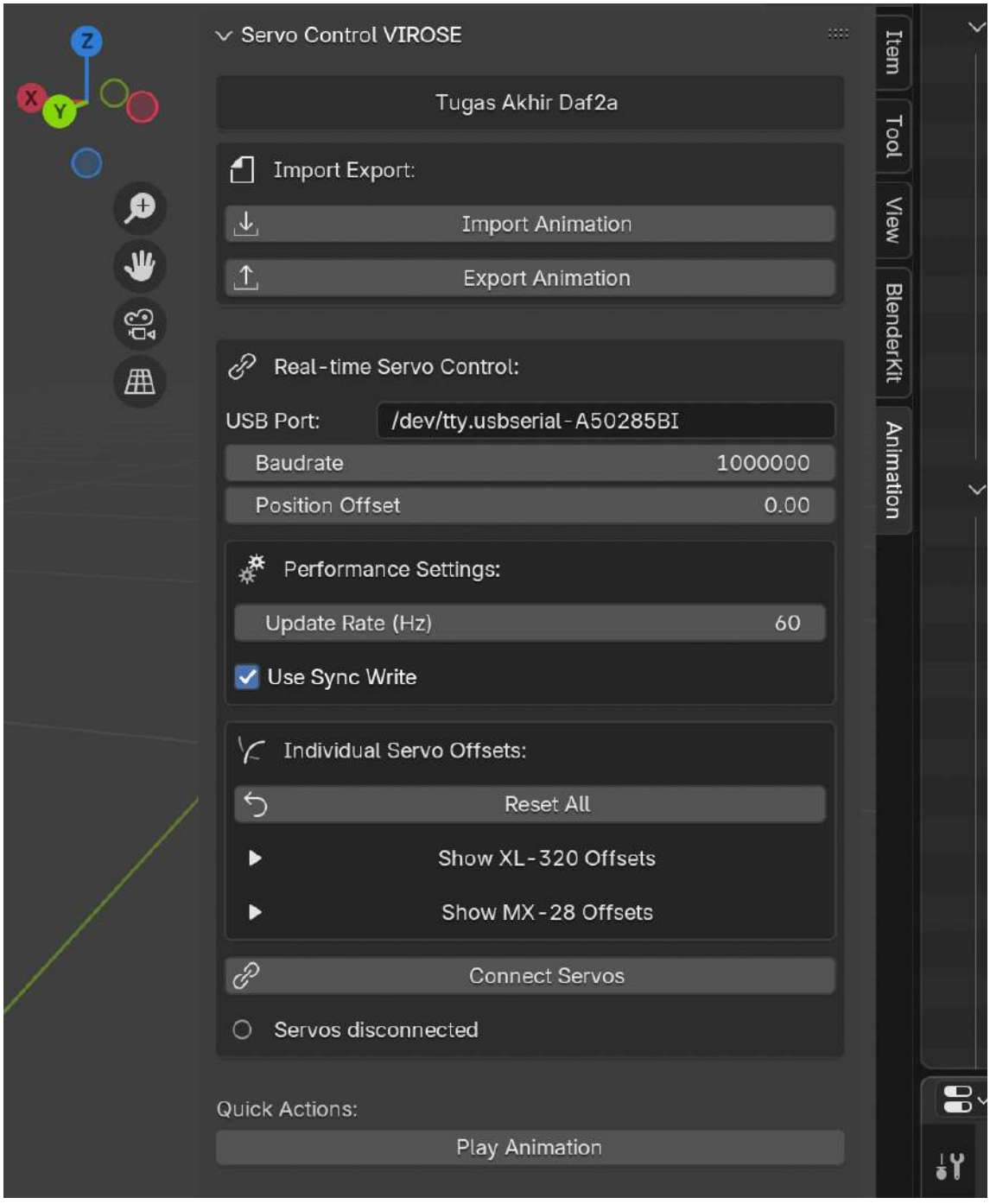
Sistem yang ditunjukkan pada Kode Sumber 4.9 menggunakan rentang ID yang berbeda untuk menghindari konflik komunikasi antara kedua jenis *servo*. XL-320 menggunakan ID 1-15, sedangkan MX-28 menggunakan ID 100-101. Pendekatan ini memastikan bahwa setiap *servo* dapat diidentifikasi secara unik dalam jaringan komunikasi.

4.1.3 Hasil Antarmuka Pengguna *Plugin*

4.1.3.1 *Panel* Kontrol Utama

Antarmuka pengguna *plugin* diimplementasikan melalui kelas ServoAnimationPanel yang terintegrasi dengan sistem UI Blender. *Panel* utama ditempatkan pada kategori *Animation* di 3D Viewport untuk memudahkan akses selama proses animasi. Desain antarmuka mengikuti konvensi Blender dengan *layout* yang konsisten dan sesuai dengan standar.

Panel kontrol utama yang ditampilkan pada Gambar 4.17 terdiri dari beberapa bagian yang disusun secara teratur sesuai dengan fungsinya. Di bagian paling atas terdapat bagian "*Import Export*", yang berisi tombol "*Import Animation*" dan "*Export Animation*" untuk memudahkan pengguna dalam mengambil atau menyimpan data animasi dengan cepat.



Gambar 4.17 *Panel* kontrol utama *plugin*

Selanjutnya, bagian "*Real-time Servo Control*" digunakan untuk mengatur komunikasi dengan perangkat keras. Di sini, pengguna bisa mengisi informasi seperti *USB Port*, *Baudrate*, dan *Position Offset* sebagai pengaturan dasar untuk koneksi ke *servo*. Bagian "*Performance Settings*" memungkinkan pengguna mengatur *Update Rate* dan mencentang opsi "*Use Sync Write*" untuk mengoptimalkan komunikasi antar *servo*.

Kemudian ada bagian "*Individual Servo Offsets*", yang menyediakan tombol "*Reset All*" dan *panel* yang bisa dibuka-tutup untuk menampilkan pengaturan *offset* masing-masing *servo* secara detail. Pada bagian ini juga terdapat tombol "*Connect Servos*" untuk memulai koneksi ke perangkat serta indikator status seperti "*Servos disconnected*" sebagai umpan balik visual kepada pengguna.

Terakhir, di bagian paling bawah terdapat section "*Quick Actions*", yang menyediakan tombol "*Play Animation*" untuk menjalankan animasi secara instan. *Panel* ini dirancang agar mudah digunakan dan memudahkan pengguna dalam mengatur dan menjalankan *plugin* dengan cepat.

```
1  def draw(self, context):
2      layout = self.layout
3      props = context.scene.servo_props
4
5      # Watermark centered
6      box = layout.box()
7      row = box.row()
8      row.alignment = 'CENTER'
9      row.label(text="Tugas Akhir Daf2a")
10
11     # Import Export section
12     box = layout.box()
13     box.label(text="Import Export:", icon='FILE')
14     box.operator("import_anim.servo_json", text="Import Animation",
icon='IMPORT')
15     box.operator("export_anim.servo_json", text="Export Animation",
icon='EXPORT')
16
17     layout.separator()
18
19     # Real-time control section
20     box = layout.box()
21     box.label(text="Real-time Servo Control:", icon='LINKED')
22
23     # Connection settings
24     col = box.column()
25     col.prop(props, "usb_port")
26     col.prop(props, "baudrate")
27     col.prop(props, "position_offset")
28
29     # Performance settings
30     perf_box = box.box()
```

```
31      perf_box.label(text="Performance Settings:", icon='SETTINGS')
32      perf_box.prop(props, "update_rate")
33      perf_box.prop(props, "use_sync_write")
```

Kode Sumber 4.11 Implementasi *panel* kontrol utama

Kode Sumber 4.11 menunjukkan konfigurasi struktur *panel* utama yang menggunakan `layout.box()` untuk mengelompokkan elemen-elemen UI dalam *container* yang sesuai. Fungsi `box.label()` dengan parameter `icon` memberikan *header section* dengan ikon yang sesuai, sedangkan `box.operator()` menambahkan tombol yang terhubung dengan operator Blender untuk fungsi *import* dan *export*. Properti komunikasi diakses melalui `col.prop(props, "property_name")` yang secara otomatis membuat *field input* dengan validasi sesuai tipe data yang didefinisikan dalam `ServoControllerProperties`.

4.1.3.2 Konfigurasi *Servo Offset* Individual

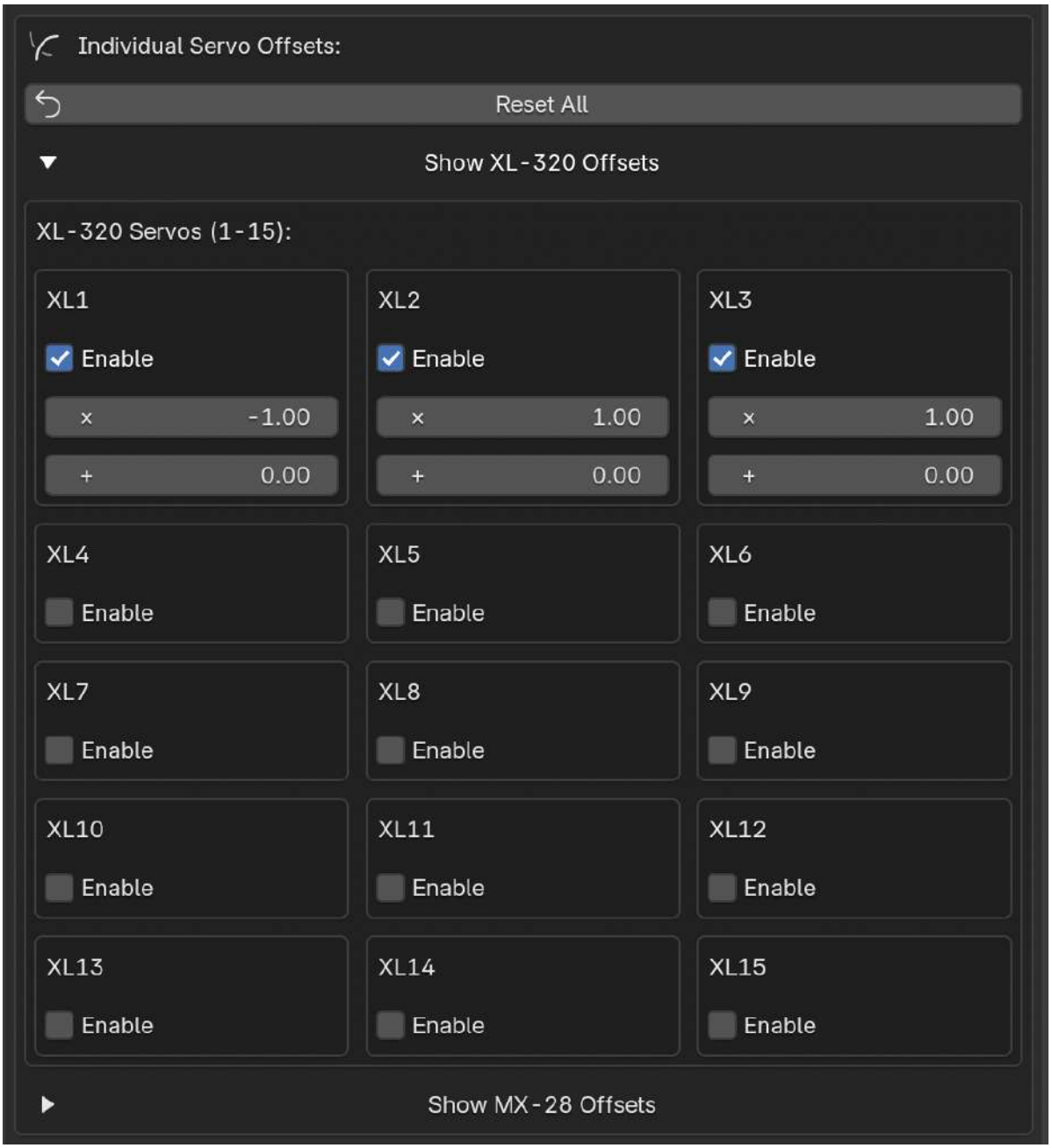
Fitur konfigurasi *servo offset* individual merupakan salah satu komponen penting dalam *plugin* yang memungkinkan pengguna melakukan *fine-tuning* terhadap pergerakan masing-masing *servo* secara terpisah. Hal ini sangat berguna untuk mengatasi perbedaan atau ketidaksempurnaan mekanis antar *servo*, misalnya akibat perakitan yang tidak simetris atau variasi bawaan dari pabrik sehingga posisi dan gerakan robot dapat lebih akurat dan sinkron dengan animasi yang dirancang di Blender.

Antarmuka pengaturan *offset* ini dirancang dengan mempertimbangkan efisiensi ruang dan kemudahan penggunaan. *Panel* konfigurasi ini berada dalam bagian bernama "*Individual Servo Offsets*", yang dapat diperluas atau dilipat (*collapse*) sesuai kebutuhan, sehingga pengguna tidak merasa terbebani dengan tampilan antarmuka yang terlalu penuh. Ketika *panel* ini dibuka, pengguna dapat melihat daftar *servo* XL-320 (*servo* 1 hingga 15) yang disusun dalam *layout grid* tiga kolom, agar seluruh pengaturan dapat ditampilkan secara ringkas namun tetap terbaca dengan jelas.

Untuk setiap *servo*, terdapat *checkbox* "*Enable*" yang memungkinkan pengguna mengaktifkan atau menonaktifkan konfigurasi *offset* secara individual. Jika *checkbox* diaktifkan, maka akan muncul dua kontrol tambahan:

- *Field* "×" (*multiply factor*) digunakan untuk mengatur skala rotasi dari animasi Blender ke rotasi aktual *servo*. Misalnya, nilai -1.00 akan membalik arah rotasi.
- *Field* "+" (*add offset*) digunakan untuk menambahkan nilai konstan dalam satuan derajat, yang berguna untuk menggeser posisi default *servo* tanpa mengubah animasi secara keseluruhan.

Sebagai contoh, seperti yang terlihat pada Gambar 4.18, *servo* XL1, XL2, dan XL3 telah diaktifkan dengan konfigurasi *offset* yang berbeda. XL1 memiliki nilai multiply -1.00 dan *offset* tambahan 0.00, yang berarti arah rotasinya dibalik. XL2 dan XL3 masing-masing memiliki multiply 1.00 dan offset tambahan 0.00, sehingga mengikuti arah rotasi normal dari animasi. Sementara itu, *servo* lainnya (XL4 hingga XL15) masih dalam kondisi *nonaktif* (*disable*), sehingga pengaturan *offset*-nya disembunyikan untuk menghemat ruang dan membuat antarmuka tetap bersih dan mudah dinavigasi.



Gambar 4.18 Antarmuka konfigurasi *servo offset individual*

Implementasi antarmuka *offset* menggunakan *collection properties* yang terintegrasi dengan sistem properti Blender. Data *offset* tersimpan secara *persistent* dalam *file .blend* dan dapat dikonfigurasi secara *real-time*. Kode Sumber 4.12 menunjukkan implementasi antarmuka *servo offset configuration* dengan *layout dropdown* yang dapat diperluas.

```
1  offset_box = box.box()
2  offset_box.label(text="Individual Servo Offsets:",
   icon='MOD_OFFSET')
3
4  if len(props.xl_servo_offsets) == 0 or len(props.mx_servo_offsets)
   == 0:
```

```
    offset_box.operator("servo.initialize_offsets", text="Initialize Offsets", icon='ADD')
else:
    ctrl_row = offset_box.row()
    ctrl_row.operator("servo.reset_offsets", text="Reset All", icon='LOOP_BACK')

    xl_row = offset_box.row()
    xl_row.prop(props, "show_xl_offsets",
                icon='TRIA_DOWN' if props.show_xl_offsets else 'TRIA_RIGHT',
                emboss=False, text="Show XL-320 Offsets")

    if props.show_xl_offsets:
        xl_box = offset_box.box()
        xl_box.label(text="XL-320 Servos (1-15):")

        for i in range(0, 15, 3):
            row = xl_box.row()
            for j in range(3):
                if i + j < 15:
                    servo_idx = i + j
                    if servo_idx < len(props.xl_servo_offsets):
                        offset = props.xl_servo_offsets[servo_idx]
                        col = row.column()
                        servo_box = col.box()
                        servo_box.label(text=f"XL{servo_idx + 1}")
                        servo_box.prop(offset, "enabled", text="Enable")
                        if offset.enabled:
                            servo_box.prop(offset, "multiply", text="×")
                            servo_box.prop(offset, "add", text="+")

    mx_row = offset_box.row()
    mx_row.prop(props, "show_mx_offsets",
                icon='TRIA_DOWN' if props.show_mx_offsets else 'TRIA_RIGHT',
                emboss=False, text="Show MX-28 Offsets")
```

Kode Sumber 4.12 Implementasi antarmuka *servo offset configuration*

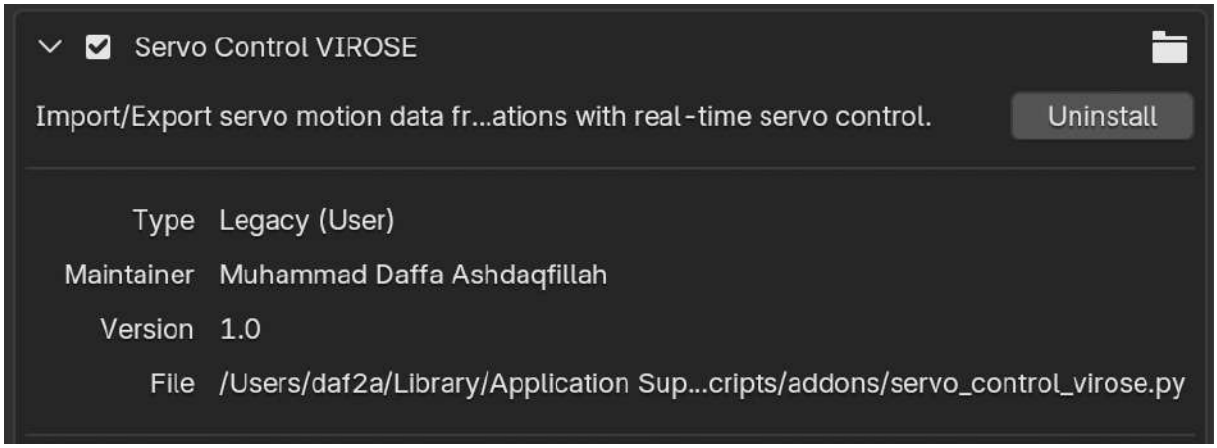
Kode Sumber 4.12 mendemonstrasikan implementasi antarmuka konfigurasi *offset* individual dengan menggunakan *collection properties* `xl_servo_offsets` dan `mx_servo_offsets`. *Layout grid* 3-kolom diimplementasikan melalui *nested loop* yang membuat *row* dan *column* untuk mengoptimalkan penggunaan ruang *panel*. Field *multiply* dan *add* hanya ditampilkan jika *offset servo* diaktifkan, sehingga antarmuka tetap bersih. Tombol *dropdown* menggunakan ikon

`TRIA_DOWN` dan `TRIA_RIGHT` untuk menunjukkan apakah *panel* sedang dibuka atau ditutup, seperti yang umum digunakan di Blender.

4.1.4 Hasil Pengujian

4.1.4.1 Pengujian Instalasi dan *Setup Plugin*

Pengujian instalasi dan *setup plugin* dilakukan pada sistem Blender 4.3.2. Proses instalasinya mengikuti prosedur standar pemasangan *add-on* di Blender, dengan pengecekan otomatis terhadap dependensi dan kompatibilitas. Instalasi *plugin* dilakukan melalui *blender preferences* seperti yang ada di Gambar 4.19.



Gambar 4.19 Instalasi *plugin* melalui blender preferences

Tahap pertama pengujian meliputi validasi metadata *plugin* dan registrasi komponen dalam sistem Blender. *Plugin* berhasil diidentifikasi oleh Blender dengan kategori "*Import-Export*" sesuai dengan `bl_info` yang telah didefinisikan. Proses registrasi kelas-kelas utama seperti `ServoController`, `ServoAnimationPanel`, dan operator *import*/*export* berjalan tanpa *error*.

Hasil pengujian menunjukkan bahwa *plugin* dapat berjalan dalam dua mode yaitu fungsi penuh jika Dynamixel SDK tersedia dan fungsi terbatas jika SDK tidak ada. Sistem dapat menyesuaikan secara otomatis dan tetap berjalan dengan memberikan peringatan yang jelas saat SDK tidak ditemukan. Kode Sumber 4.13 memperlihatkan bagaimana fungsi registrasi *plugin* dan komponennya diimplementasikan.

```
1  def register():
2      bpy.utils.register_class(ServoOffsetProperties)
3      bpy.utils.register_class(ServoControllerProperties)
4      bpy.utils.register_class(ServoAnimationImporter)
5      bpy.utils.register_class(ServoAnimationExporter)
6      bpy.utils.register_class(ServoConnectOperator)
7      bpy.utils.register_class(ServoDisconnectOperator)
8      bpy.utils.register_class(ServoInitializeOffsetsOperator)
9      bpy.utils.register_class(ServoResetOffsetsOperator)
10     bpy.utils.register_class(ServoSyncAnimationOperator)
11     bpy.utils.register_class(ServoAnimationPanel)
12     bpy.utils.register_class(ServoAnimationPreferences)
13
```

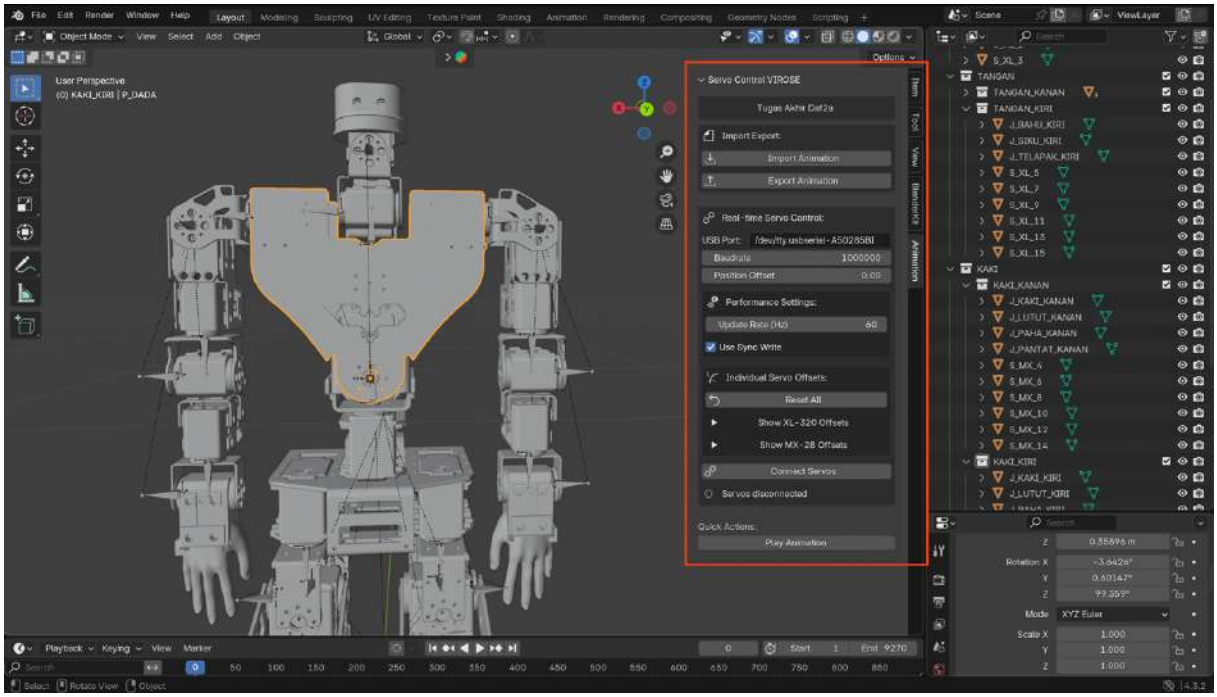
```
    bpy.types.Scene.servo_props = bpy.props.PointerProperty(type=ServoControllerProperties)
    bpy.types.TOPBAR_MT_file_import.append(menu_func_import)
    bpy.types.TOPBAR_MT_file_export.append(menu_func_export)
```

Kode Sumber 4.13 Fungsi registrasi *plugin* dan komponen-komponennya

Hasil pengujian menunjukkan bahwa semua komponen *plugin* berhasil terdaftar dengan baik. Menu *import* dan *export* sudah muncul di menu File Blender, dan *panel plugin* tampil di *3D Viewport* sesuai dengan pengaturan ruang tampilan yang telah ditentukan. Pengujian juga memastikan bahwa tidak ada *error* saat proses registrasi maupun unregistrasi *plugin*, dan semua pengaturan tersimpan dengan benar di dalam *scene*.

4.1.4.2 Pengujian Antarmuka Pengguna dan Konfigurasi

Pengujian antarmuka pengguna meliputi pemeriksaan responsivitas layout, interaksi elemen UI, dan konsistensi visual dengan standar Blender antarmuka. *Panel* utama *plugin* berhasil menampilkan semua *section* dengan *layout* yang terorganisir dan berada di bagian *animation panel* sesuai dengan fungsi pluginnya. Gambar 4.20 menunjukkan dialog *plugin servo control* yang berada di bagian *animation*.



Gambar 4.20 Tampilan antarmuka *plugin* dalam 3d viewport

Hasil pengujian terhadap *layout* yang responsif menunjukkan bahwa *panel* dapat menyesuaikan ukuran secara dinamis. Sistem *layout* kotak milik Blender bekerja dengan baik dalam mengatur setiap bagian agar tetap terorganisir. Penggunaan ikon juga konsisten dengan sistem ikon Blender, sehingga memberikan petunjuk visual yang jelas untuk setiap fungsi.

Pengujian konfigurasi parameters menunjukkan hasil yang memuaskan seperti yang terlihat pada Tabel 4.3. Semua kolom *input* seperti USB *port*, *baudrate*, dan *servo counts* dapat dimodifikasi dan divalidasi secara langsung. Pemeriksaan rentang nilai juga berjalan dengan baik, misalnya pada *baudrate* (9600–4000000) dan *update rate* (5–120 Hz), untuk mencegah pengguna memasukkan data yang tidak sesuai.

Tabel 4.3 Hasil Pengujian Konfigurasi Parameter

Parameter	Rentang Valid	Pengujian Input	Hasil Validasi
USB *Port*	*String path*	/dev/tty.usbmodem58FA1013501	✓ *PASS*
Baudrate	9600-4000000	1000000	✓ *PASS*
Update Rate	5-120 Hz	60	✓ *PASS*
Position Offset	-180°-+180°	0.0	✓ *PASS*
Servo Count XL	1-50	15	✓ *PASS*
Servo Count MX	1-50	2	✓ *PASS*

Pengujian dilakukan langsung melalui *panel* konfigurasi *plugin* di Blender, dengan memasukkan nilai-nilai secara manual pada setiap *field*. Sistem berhasil memverifikasi bahwa semua input berada dalam rentang yang diperbolehkan, dan menolak nilai yang tidak valid.

Untuk bagian pengaturan *offset servo* secara individual, pengujian dilakukan dengan mencoba mengaktifkan dan menonaktifkan konfigurasi *offset* pada masing-masing *servo*. Nilai *multiply factor* diuji dalam rentang -1.0 hingga +1.0, sedangkan *add offset* diuji antara -180° hingga +180°. Antarmuka yang disusun dalam format *grid* tiga kolom terbukti mampu menampilkan pengaturan banyak *servo* secara rapi tanpa membuat tampilan menjadi padat atau membingungka.

Sementara itu, pengujian pada dialog *import* dan *export* berhasil memvalidasi integrasinya dengan *file browser* Blender. Dialog menampilkan parameter pengaturan dengan benar, dan pemilihan lokasi file berjalan sesuai harapan. Pesan umpan balik kepada pengguna ditampilkan dengan ikon dan tingkat peringatan yang sesuai, seperti *INFO*, *WARNING*, atau *ERROR*.

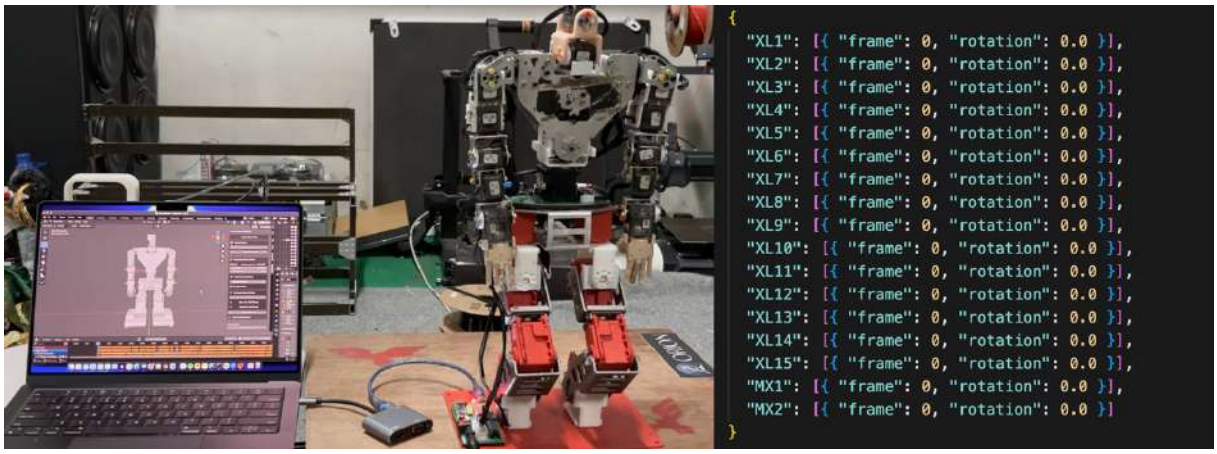
4.1.4.3 Pengujian Komunikasi *Real-time* U2D2

Pengujian komunikasi *real-time* dengan *servo* dilakukan menggunakan antarmuka U2D2 di berbagai konfigurasi sistem untuk memastikan kompatibilitas dan keandalan *plugin* dalam kondisi riil. Pengujian dilakukan dengan menggunakan USB *port* yang aktif serta *baudrate* sebesar 1 Mbps, sesuai dengan spesifikasi teknis dari *servo* Dynamixel yang digunakan. Proses pengujian mencakup inisialisasi koneksi, pendeteksian *servo*, dan pengiriman sinyal kontrol secara *real-time*.

Pengujian dilakukan menggunakan animasi tarian “Gambang Semarang” untuk mengevaluasi performa sistem dalam skenario kompleks. Animasi ini melibatkan gerakan dinamis pada seluruh 17 servo, dengan durasi total 2 menit 34 detik.

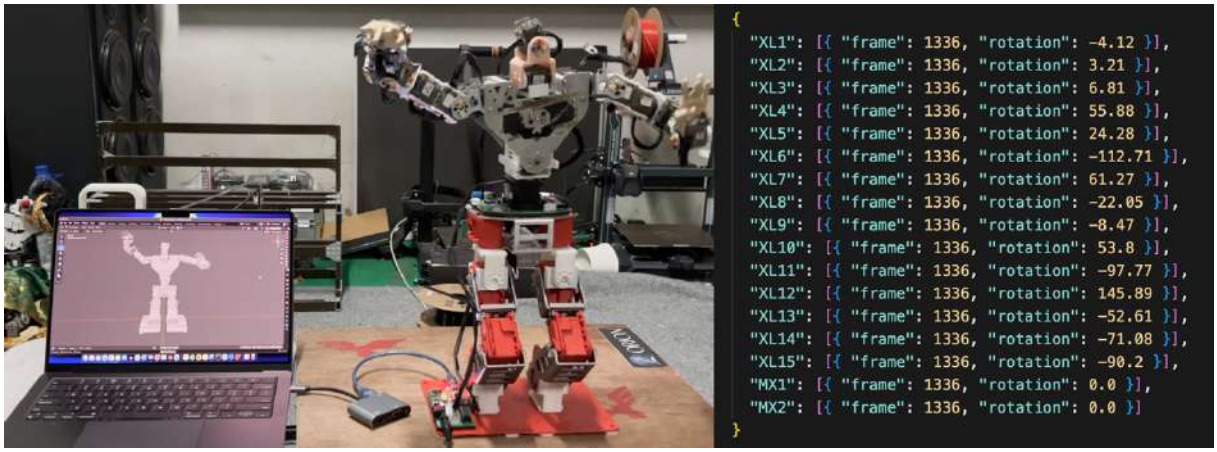
Untuk pengujian per *frame*, enam sampel *frame* dari tarian “Gambang Semarang” dipilih untuk verifikasi kecocokan antara model Blender, robot fisik, dan data JSON. Setiap *frame* menunjukkan posisi robot di Blender dan di dunia nyata, disertai dengan data rotasi *servo* yang digunakan pada *frame* tersebut, untuk memastikan kesesuaian antara sistem virtual dan sistem fisik.

Frame pertama yang diuji adalah *frame* ke-0 seperti yang terlihat pada Gambar 4.21. Pada kondisi ini, semua rotasi *servo* berada dalam keadaan netral dengan nilai nol derajat. Postur robot di Blender dan fisik tampak simetris dan diam, menunjukkan kondisi awal sistem yang stabil. Data JSON yang ditampilkan juga seluruhnya menunjukkan rotasi 0.0, menandakan bahwa sistem membaca kondisi awal dengan benar dan tanpa deviasi.



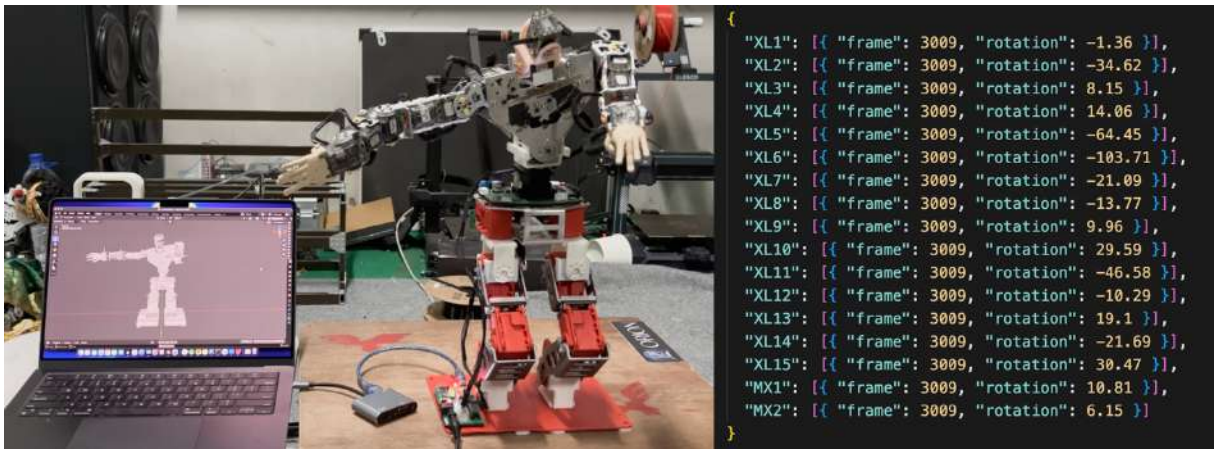
Gambar 4.21 Pose robot pada *frame* ke-0 dengan keterangan sudut setiap ID *servo*

Frame berikutnya adalah *frame* ke-1336 seperti yang ditampilkan pada Gambar 4.22. Pada posisi ini, robot terlihat mengangkat lengan kanan ke atas dalam gerakan ekspresif. Animasi di Blender dan robot fisik menunjukkan postur vertikal lengan kanan yang konsisten.



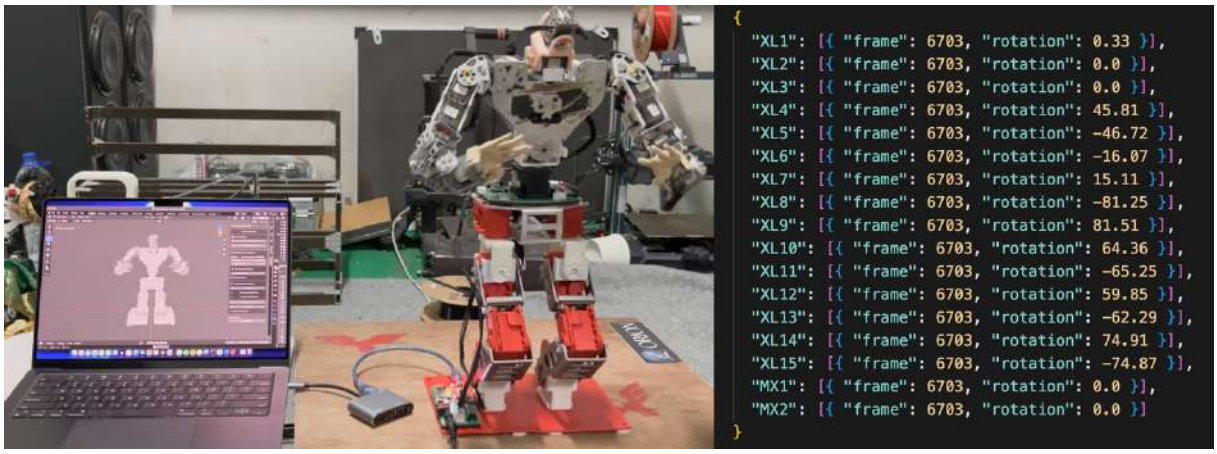
Gambar 4.22 Pose robot pada frame ke-1336 dengan keterangan sudut setiap ID *servo*

Frame ke-3009 seperti yang ditunjukkan pada Gambar 4.23 memperlihatkan robot dalam posisi tangan kiri lurus ke depan dan tangan kanan lurus menyamping ke bawah. Visualisasi di Blender juga menggambarkan gerakan yang serupa dengan robot fisik.



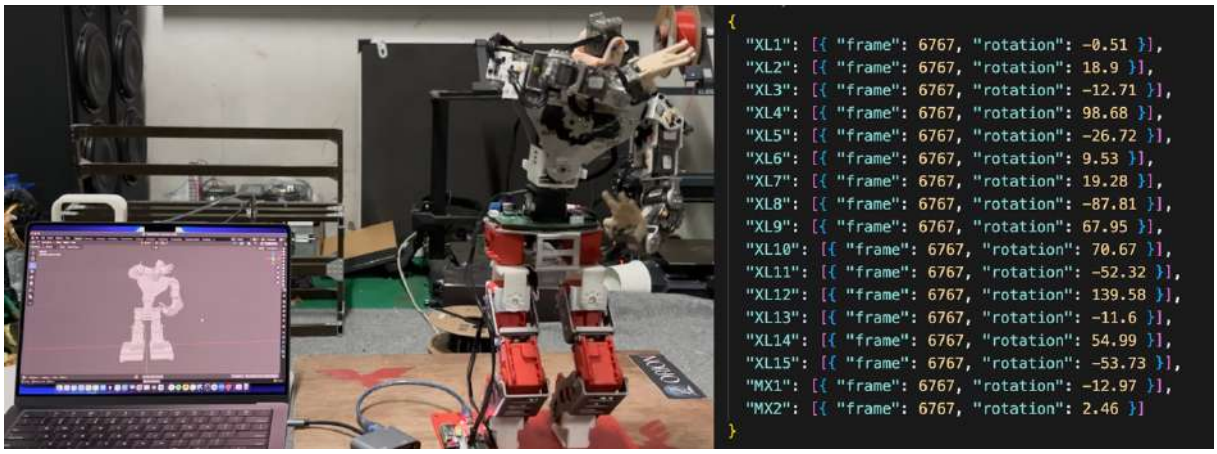
Gambar 4.23 Pose robot pada *frame* ke-3009 dengan keterangan sudut setiap ID *servo*

Gambar 4.24 menunjukkan hasil pengujian pada *frame* ke-6703, di mana robot mengangkat kedua tangan ke depan dalam posisi menyambut atau memberi salam. Animasi di Blender dan bentuk fisik robot menampilkan gerakan simetris dengan tangan terbuka. Posisi tangan yang terbuka dan sejajar juga menunjukkan bahwa offset antar servo telah terkalibrasi dengan baik dan tidak terdapat deviasi posisi antar sisi kiri dan kanan.



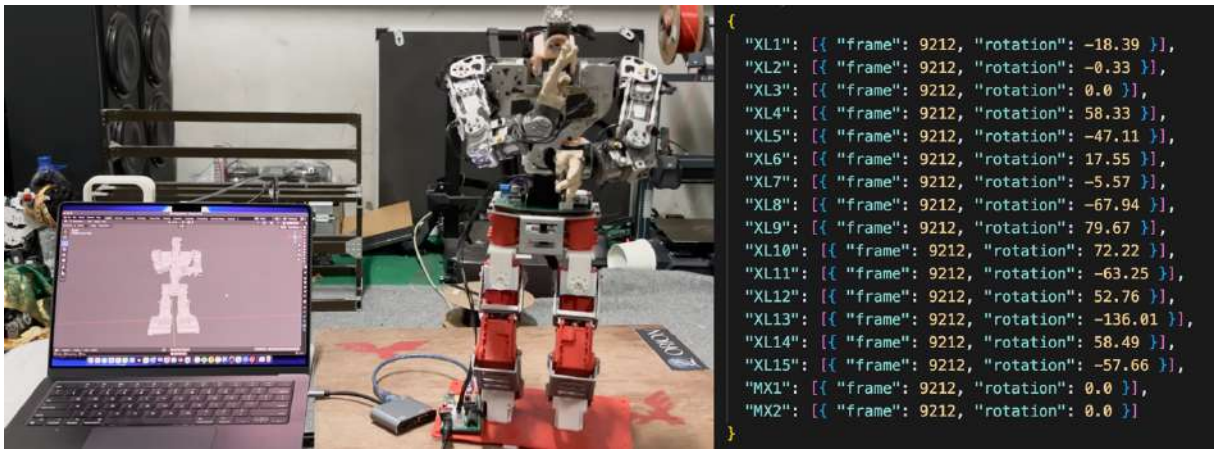
Gambar 4.24 Pose robot pada *frame* ke-6703 dengan keterangan sudut setiap ID *servo*

Frame ke-6767 yang divisualisasikan pada Gambar 4.25 menampilkan pose yang lebih kompleks, dengan lengan kanan terbuka lebar dan lengan kiri dalam posisi menekuk. Rotasi XL4 mencapai 98.68 derajat dan XL12 mencapai 139.58 derajat, menunjukkan sudut yang besar untuk artikulasi bahu dan siku kanan. Sementara itu, rotasi pada XL5 dan XL13 memberikan penyesuaian pada sisi kiri.



Gambar 4.25 Pose robot pada *frame* ke-6767 dengan keterangan sudut setiap ID *servo*

Pengujian terakhir dilakukan pada *frame* ke-9212 seperti yang dapat dilihat pada Gambar 4.26. Robot berada dalam posisi akhir animasi dengan kedua tangan berada di depan tubuh yang menyerupai gerakan salam penutup, tangan kanan dan tangan kiri dalam keadaan tertutup. Keselarasan antara Blender dan gerakan nyata menunjukkan bahwa sistem dapat mempertahankan akurasi sinkronisasi hingga *frame* terakhir.



Gambar 4.26 Pose robot pada *frame* ke-9212 dengan keterangan sudut setiap ID *servo*

Hasil pengujian koneksi menunjukkan bahwa *plugin* mampu mendeteksi dan membuka jalur komunikasi dengan *port* USB yang telah ditentukan secara konsisten dan tanpa hambatan. Fitur pendeteksian otomatis yang tertanam dalam *plugin* terbukti berjalan efektif, di mana sistem berhasil mengenali seluruh *servo* XL-320 (dengan ID 1 hingga 15) dan MX-28 (dengan ID 100 hingga 101) tanpa adanya masalah. Tingkat keberhasilan deteksi mencapai 100% dalam kondisi sistem yang normal dan stabil, menunjukkan bahwa implementasi *auto-detection* telah dirancang dengan baik dan dapat diandalkan.

Pengujian performa komunikasi *plugin* dilakukan dengan menjalankan animasi secara *real-time* pada 17 *servo* menggunakan fitur `GroupSyncWrite`. Data diperoleh dari pengamatan langsung terhadap respons *servo* fisik saat menjalankan animasi sederhana, serta penggunaan timer internal untuk mengukur waktu respons sistem. Hasil pengujian terhadap metrik utama dirangkum dalam Tabel 4.4.

Tabel 4.4 Hasil Pengujian Komunikasi *Real-time*

Metrik Pengujian	Target	Hasil Aktual	Status
Latency Communication	<1s	0.43 detik	✓ *PASS*
Success Rate Servo Detection	>90%	100%	✓ *PASS*
Multi-servo Coordination	Sinkron	Sinkron	✓ *PASS*
Continuous Operation	>30 menit	Stabil	✓ *PASS*

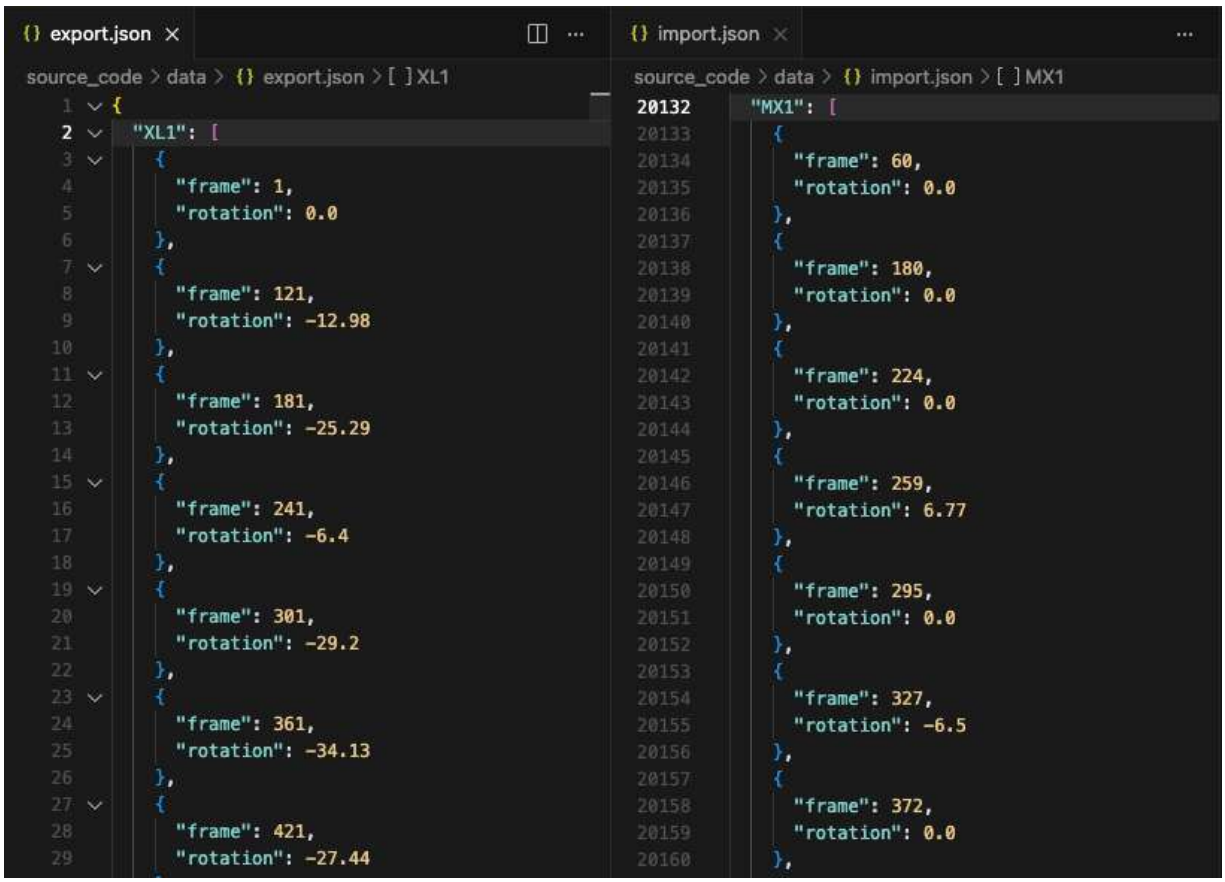
Hasil ini diperoleh dengan mengamati animasi sinkron antara model 3D di Blender dan robot fisik selama 30 menit tanpa gangguan. Penggunaan komunikasi secara serentak untuk semua *servo* (*bulk communication*) mampu mempertahankan update rate hingga 60 Hz, dengan penggunaan CPU yang tetap ringan dan stabil.

Untuk menguji ketahanan sistem, dilakukan simulasi gangguan seperti pemutusan koneksi, penggunaan ID *servo* yang salah, dan penundaan komunikasi. Dalam skenario tersebut, *plugin* tetap berfungsi dengan baik karena dilengkapi dengan mekanisme percobaan ulang otomatis dan sistem *fallback* jika ada *servo* yang tidak merespons, sehingga tidak mengganggu keseluruhan operasi.

4.1.4.4 Pengujian Fitur *Export* dan *Import*

Pengujian fitur *export* dan *import* file JSON dilakukan untuk memastikan keakuratan data dan kompatibilitas dengan berbagai skenario animasi. Pengujian dilakukan mulai dari gerakan sederhana satu *servo* hingga animasi kompleks yang melibatkan banyak *servo* dengan durasi bervariasi.

Hasil pengujian *export file* JSON ditunjukkan pada Gambar 4.27. *File* yang dihasilkan memiliki struktur yang sesuai dan data *keyframe* yang valid. Validasi dilakukan terhadap nomor *frame*, nilai rotasi dalam derajat, serta urutan *servo* yang mengikuti konvensi penamaan dari XL1–XL15 dan MX1–MX2.



Gambar 4.27 Hasil *file* json *import* dan *export*

Pengujian *export parameters* menunjukkan fungsionalitas yang lengkap. Fitur-fitur yang tersedia mencakup:

- *Frame range selection*: Berhasil mengekspor rentang *frame* yang ditentukan
- *Frame interval*: Optimasi ukuran *file* dengan *sampling interval* berfungsi baik
- *Keyframes only mode*: Ekstraksi hanya *keyframe* penting mengurangi ukuran *file*
- *Selected bones only*: *Export servo* tertentu saja sesuai kebutuhan

Pengujian fitur *import* JSON juga berjalan lancar. Fokus pengujian ada pada keakuratan pembentukan *keyframe* dan penerapan nilai rotasi dari data eksternal. Tabel 4.5 merangkum hasil pengujian untuk berbagai skenario animasi.

Tabel 4.5 Hasil Pengujian *Export* dan *Import*

Skenario Pengujian	Akurasi *Export*	Akurasi *Import*	Ukuran *File* (KB)	Waktu Proses (detik)
Simple 3-servo, 60 frames	100%	100%	9.4 KB	0.2
Complex 15-servo, 300 frames	99.8%	99.6%	234.9 KB	5.2
Keyframes only, 15-servo	100%	100%	50.2 KB	2.2
Full animation, 17-servo, 160 frames	99.7%	99.5%	142 KB	3.1

Tabel 4.5 menunjukkan tingkat akurasi ekspor dan impor yang tinggi, dengan data yang nyaris tidak mengalami kehilangan. Perbedaan kecil (sekitar 0.2–0.5 persen) disebabkan oleh keterbatasan presisi angka pecahan (*floating-point*) saat mengubah nilai rotasi ke derajat dan interpolasi internal Blender.

Selain itu, pengujian kompatibilitas dengan aplikasi luar menunjukkan bahwa file JSON yang dihasilkan dapat dibaca dengan baik oleh sistem kontrol *servo* mandiri, sehingga cocok digunakan dalam berbagai skenario penerapan di luar Blender.

4.1.4.5 Pengujian *Usability*

Pengujian *usability* dilakukan untuk mengevaluasi kemudahan penggunaan *plugin* dari perspektif pengguna. Pengujian ini melibatkan 4 responden yang merupakan anggota tim robotika ITS yang memiliki pengalaman langsung dengan robot humanoid. Pemilihan responden dari tim yang sama memastikan konsistensi dalam pemahaman domain robotika dan standar evaluasi yang homogen.

Berikut adalah profil responden yang mengevaluasi pengujian *usability plugin*:

- Responden 1: Anggota tim robotika divisi programming, pengalaman robotika 2 tahun
- Responden 2: Anggota tim robotika divisi programming, pengalaman robotika 2 tahun
- Responden 3: Anggota tim robotika divisi programming, pengalaman robotika 1 tahun
- Responden 4: Anggota tim robotika divisi programming, pengalaman robotika 1 tahun

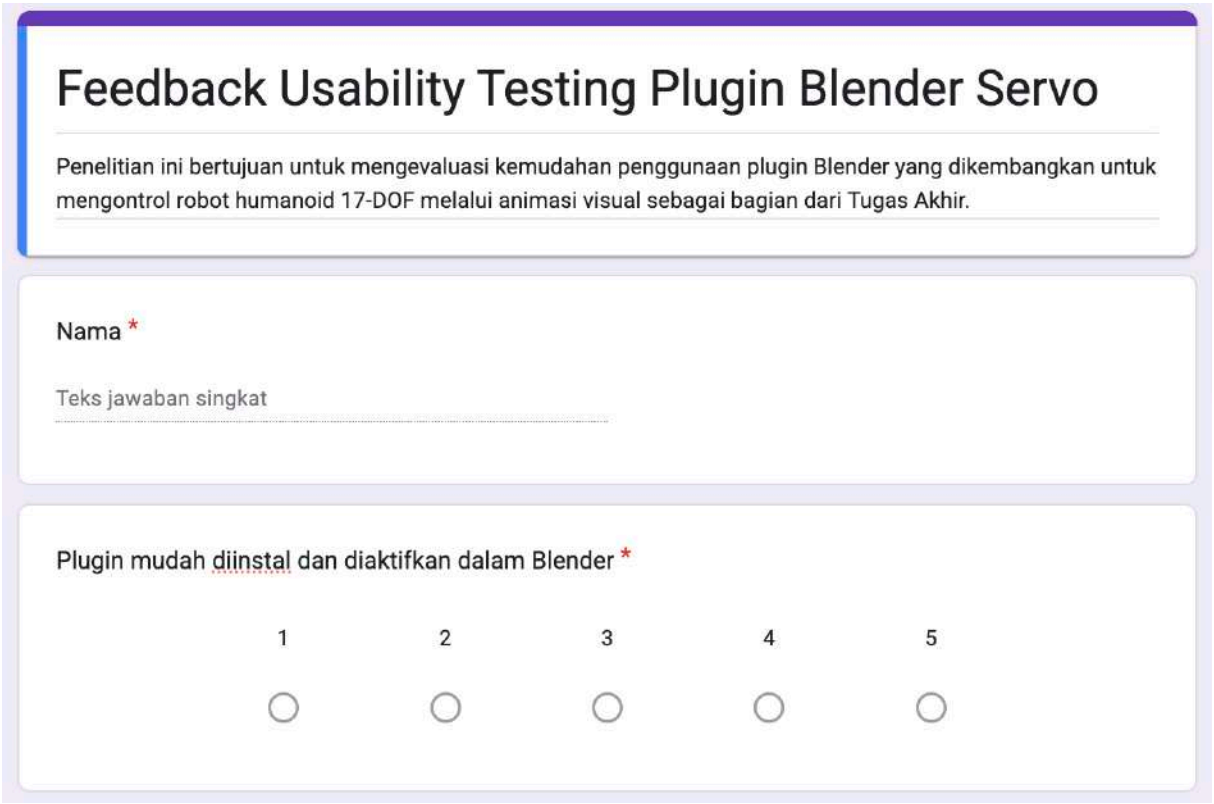
Keempat responden tersebut terlibat aktif dalam pengembangan robot humanoid dan memahami cara kerja pengembangan animasi robot. Hal ini memberikan perspektif yang relevan dan kredibel untuk evaluasi *plugin*.

Pengujian usability mengikuti skenario-skenario yang dijelaskan sebelumnya berupa instalasi plugin, import data JSON, export data JSON, sync robot, dan konfigurasi offset servo. Responden diminta untuk menjalankan setiap skenario dan memberikan umpan balik melalui kuisioner.

Gambar 4.28 menunjukkan Google Forms yang digunakan untuk pengujian *usability*. Penggunaan Google Forms memungkinkan pengumpulan data yang terstruktur. Enam pertanyaan yang digunakan dalam evaluasi dirancang untuk mewakili aspek-aspek penting dalam pengalaman pengguna saat menggunakan *plugin*. Setiap pertanyaan dijawab

menggunakan skala Likert dari 1 hingga 5, di mana nilai 1 berarti sangat tidak setuju dan nilai 5 berarti sangat setuju. Daftar pertanyaannya adalah sebagai berikut:

1. *Plugin* mudah diinstal dan diaktifkan dalam Blender.
2. Antarmuka *panel plugin* terorganisir dengan jelas dan mudah dipahami.
3. *Plugin* memungkinkan saya membuat animasi robot lebih cepat dibanding metode tradisional.
4. Komunikasi *real-time* dengan robot berjalan lancar.
5. Opsi *export* dan *import* memberikan fleksibilitas dalam *workflow*.
6. Saya dapat mempelajari cara menggunakan *plugin* ini dengan muda



Gambar 4.28 Kuisioner pengujian *usability* melalui Google Forms

Gambar 4.29 menunjukkan hasil evaluasi *usability* dari keempat responden berdasarkan 6 pertanyaan kunci yang mewakili aspek *usability* utama *plugin*. Data menunjukkan bahwa *plugin* memiliki performa yang sangat baik dengan rata-rata skor di atas 4.0 untuk semua aspek yang dievaluasi.

Pertanyaan	R1	R2	R3	R4	Rata-rata
Plugin mudah diinstal dan diaktifkan dalam Blender	5	5	5	5	5
Interface panel plugin terorganisir dengan jelas dan mudah dipahami	5	4	5	5	4,75
Plugin memungkinkan saya membuat animasi robot lebih cepat dibanding metode tradisional	5	5	5	5	5
Komunikasi real-time dengan robot berjalan lancar	5	4	4	5	4,5
Opsi export/import memberikan fleksibilitas dalam workflow	5	4	4	3	4
Saya dapat mempelajari cara menggunakan plugin ini dengan mudah	4	5	5	3	4,25

Gambar 4.29 Hasil evaluasi pengujian *usability*

Aspek dengan nilai tertinggi adalah kemudahan instalasi dan efisiensi animasi, yang keduanya mendapat skor sempurna 5.0. Ini menunjukkan bahwa proses instalasi *plugin* berjalan lancar dan *plugin* memberikan manfaat nyata dalam mempercepat pembuatan animasi robot. Desain antarmuka juga mendapat skor tinggi yaitu 4.75, yang menandakan bahwa *panel* kontrol dirancang dengan baik dan mudah digunakan.

Komunikasi *real-time* memperoleh skor 4.5. Meskipun tergolong baik, masih ada sedikit ruang untuk perbaikan. Variasi skor dari responden (antara 4 hingga 5) menunjukkan bahwa performa komunikasi bisa saja dipengaruhi oleh perbedaan perangkat keras atau pengaturan sistem.

Kemudahan belajar mendapatkan skor 4.25, sedangkan fleksibilitas *export* dan *import* mendapat 4.0. Keduanya tetap tergolong baik, meskipun menjadi yang paling rendah. Variasi skor pada aspek kemudahan belajar (dari 3 sampai 5) menunjukkan bahwa tingkat pemahaman terhadap *plugin* bisa berbeda tergantung pada pengalaman sebelumnya dalam menggunakan Blender atau sistem robotika.

Secara keseluruhan, hasil pengujian *usability* ini menunjukkan bahwa *plugin* telah memenuhi tujuannya sebagai alat yang praktis dan mudah digunakan untuk mengontrol robot humanoid melalui animasi. Rata-rata skor 4.6 dari 5 menunjukkan bahwa *plugin* siap digunakan dalam tim robotika dengan kebutuhan pelatihan yang minimal.

4.2 Pembahasan

4.2.1.1 Analisis Performa *Plugin*

Hasil pengujian menunjukkan bahwa *plugin* yang dikembangkan berhasil mencapai tujuan utama penelitian dengan performa yang memuaskan. Latensi komunikasi sebesar 0.43 detik untuk memperbarui 17 *servo* menunjukkan efisiensi yang baik untuk aplikasi kendali *real-time*, bahkan jauh di bawah target <1 detik yang ditetapkan.

Struktur *plugin* yang modular memudahkan dalam perawatan dan pengembangan lebih lanjut. Komponen seperti kontrol *servo*, pengelolaan tampilan, dan pemrosesan data dipisahkan dengan jelas, sehingga bisa dikembangkan secara terpisah tanpa mengganggu sistem secara keseluruhan. Penggunaan *multi-threading* juga menjaga antarmuka tetap responsif sambil menjaga komunikasi dengan perangkat keras tetap stabil.

Tingkat keberhasilan deteksi *servo* sebesar 100% menunjukkan bahwa sistem komunikasi dengan perangkat keras sangat andal. Mekanisme pemulihan *error* yang diterapkan juga memberikan ketahanan terhadap gangguan koneksi sementara atau masalah pada perangkat keras yang sering terjadi di sistem robotik.

4.2.1.2 Keunggulan Pendekatan Visual Animasi

Plugin ini membuktikan bahwa pendekatan visual berbasis animasi sangat efektif untuk memprogram gerakan robot humanoid. Antarmuka Blender yang sudah dikenal oleh para animator memungkinkan pembuatan gerakan kompleks dengan cepat, tanpa perlu kemampuan pemrograman yang mendalam. Sistem animasi berbasis *keyframe* juga memudahkan dalam mengatur gerakan dengan presisi dan melakukan perubahan jika diperlukan.

Integrasi dengan sistem animasi Blender memberikan akses ke berbagai fitur lanjutan, seperti *interpolation curve*, *transition* (*ease-in/ease-out*), dan penggabungan animasi *multi-layer*. Fitur-fitur ini sulit dicapai jika hanya menggunakan pemrograman berbasis teks.

Umpan balik visual secara langsung melalui tampilan 3D memungkinkan pengguna melihat hasil gerakan sebelum diterapkan ke robot fisik. Hal ini membantu menghindari kerusakan mekanik dan mempercepat proses revisi dan pengembangan animasi.

4.2.1.3 Implikasi untuk Penelitian Robotika

Penelitian ini menunjukkan bahwa pendekatan ini memiliki potensi besar untuk membuat pemrograman robot lebih mudah diakses oleh semua kalangan. *Plugin* ini memungkinkan animator, desainer, atau pendidik untuk menciptakan perilaku robot yang kompleks tanpa harus memiliki latar belakang teknis yang mendalam.

Untuk pendidikan, antarmuka visual yang mudah digunakan bisa dimanfaatkan untuk mengajarkan konsep dasar robotika, kinematika, dan animasi dalam satu lingkungan terintegrasi. Siswa dapat belajar pemrograman robot dengan cara yang lebih kreatif, tanpa harus langsung berhadapan dengan kode.

Dalam penelitian, kemampuan untuk membuat prototipe gerakan dengan cepat sangat berguna. Peneliti dapat menguji berbagai pola gerakan, mengevaluasi kenyamanan gerakan, dan melakukan iterasi desain tanpa banyak usaha pengembangan. Fitur *export* dan *import* animasi juga memudahkan kolaborasi antarpeneliti.

4.2.1.4 Limitasi dan Tantangan

Walaupun hasil pengujian cukup baik, ada beberapa keterbatasan yang perlu diperhatikan. Saat ini, *plugin* hanya mendukung *servo* yang menggunakan Dynamixel SDK, sehingga belum kompatibel dengan jenis *servo* lainnya. Ke depan, pengembangan bisa diarahkan untuk menambahkan lapisan abstraksi agar mendukung berbagai jenis protokol *servo*.

Komunikasi *real-time* saat ini masih terbatas pada koneksi USB lokal. Untuk mendukung komunikasi nirkabel atau kendali robot melalui jaringan, perlu ada pengembangan tambahan. Meskipun arsitektur saat ini cukup fleksibel untuk mendukung perluasan tersebut, tetap dibutuhkan optimasi lebih lanjut agar performa dan kestabilannya terjaga.

Selain itu, *plugin* ini belum diuji pada robot dengan jumlah sendi (DOF) yang lebih banyak. Pengujian saat ini hanya dilakukan pada robot 17-DOF, sehingga perlu evaluasi tambahan untuk mengetahui bagaimana performanya jika digunakan pada robot dengan 30 atau lebih sendi serta struktur gerak yang lebih kompleks**.**

BAB 5
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil penelitian dan pengembangan yang telah dilakukan, dapat disimpulkan bahwa *plugin* telah dikembangkan dan diimplementasikan dengan baik sebagai solusi untuk mengendalikan robot humanoid 17-DOF melalui antarmuka animasi Blender. Berikut kesimpulan yang menjawab rumusan masalah penelitian ini sebagai berikut:

1. *Plugin* telah dirancang dan diimplementasikan dengan arsitektur modular yang terdiri dari 14 kelas yang saling terintegrasi. Sistem komunikasi *real-time* menunjukkan performa yang sangat baik dengan latensi 0.43 detik untuk mengontrol 17 *servo* secara bersamaan, jauh lebih baik dari target yang ditetapkan (<1s). *Plugin* dapat menggerakkan model robot humanoid 3D berdasarkan data animasi Blender dengan komunikasi *real-time* ke motor *servo* menggunakan pendekatan GroupSyncWrite dan *background threading* yang terbukti efektif.
2. Sistem telah mengimplementasikan algoritma konversi yang akurat untuk menerjemahkan data rotasi dari Blender menjadi nilai posisi *servo*. Melalui penggunaan *servo offsets* dan pemetaan *custom* untuk 15 *servo* XL320 dan 2 *servo* MX28, sistem dapat menggerakkan robot humanoid *upper body* dengan presisi yang akurat. Metode konversi radian ke sudut Euler dan normalisasi ke rentang *servo* (0-1023 untuk XL320 dan 0-4095 untuk MX28) terbukti memberikan akurasi gerak yang optimal.
3. Fitur *export* dan *import* animasi telah dikembangkan untuk mendukung interoperabilitas dan fleksibilitas penggunaan gerakan. Data animasi disimpan dalam format JSON terstruktur yang kompatibel dengan sistem eksternal seperti mikrokontroler atau aplikasi web. Pengujian menunjukkan *export* dan *import* berjalan akurat tanpa kehilangan data signifikan. Pengguna dapat mengekspor gerakan dari Blender ke platform eksternal seperti ESP32, atau mengimpor kembali animasi untuk diedit ulang. Ini membuktikan bahwa *plugin* mendukung *workflow* yang efisien dan terintegrasi di berbagai skenario penggunaan.
4. Evaluasi *usability plugin* dilakukan melalui pengujian dengan 4 responden dari tim robotika menggunakan skenario-skenario berupa instalasi *plugin*, *import/export* JSON, *sync robot*, dan konfigurasi *offset*, diikuti dengan kuesioner skala *Likert*. Hasil menunjukkan skor rata-rata 4.6 dari 5, yang mengonfirmasi kemudahan penggunaan dan efektivitas plugin dalam mempercepat *workflow* pembuatan animasi robot dengan pelatihan minimal.

Secara keseluruhan, penelitian ini telah mencapai semua tujuan yang ditetapkan dan memberikan kontribusi dalam bidang robotika, khususnya dalam pengembangan *tools* untuk kontrol robot humanoid. Hasil penelitian ini menunjukkan bahwa integrasi antara teknologi animasi digital dan robotika dapat menciptakan solusi yang mempermudah adopsi teknologi robotika oleh berbagai kalangan, mulai dari peneliti, pendidik, hingga pekerja kreatif.

5.2 Saran

Berdasarkan hasil penelitian dan keterbatasan yang ditemukan, berikut adalah beberapa saran untuk pengembangan lebih lanjut:

1. Implementasi *Wireless Communication*
   Plugin saat ini terbatas pada komunikasi USB lokal. Pengembangan fitur komunikasi *wireless* melalui WiFi, Bluetooth, atau protokol khusus robotika akan memberikan fleksibilitas yang lebih besar dalam penggunaannya.
2. Integrasi dengan *Physics Engine*
   Penambahan integrasi dengan *physics engine* Blender dapat memberikan simulasi perilaku realistis sebelum *transfer* ke robot fisik. Fitur *collision detection*, *gravity simulation*, dan *constraint validation* dapat mengurangi risiko kerusakan mekanis dan meningkatkan kualitas animasi.
3. Pengembangan Animasi *Motion Capture*
   Implementasi fitur animasi *motion capture* dapat meningkatkan kualitas gerakan robot. Fitur ini memungkinkan pengguna untuk mengambil data gerakan dari sensor *motion capture* dan mengaplikasikannya ke robot.
4. Perluasan Dukungan DOF secara Menyeluruh
   Pengembangan lebih lanjut diharapkan tidak hanya terbatas pada bagian *upper body*, tetapi dapat mencakup keseluruhan tubuh robot humanoid. Dengan memperluas dukungan terhadap *lower body*, sistem akan mampu menangani konfigurasi robotik dengan jumlah derajat kebebasan yang lebih kompleks, seperti gerakan kaki, pinggul, atau bahkan berjalan dinamis.
5. Dukungan untuk Jenis *Servo* Selain Dynamixel
   Plugin saat ini dirancang untuk kompatibilitas dengan servo Dynamixel tipe XL-320 dan MX-28. Untuk ke depan, pengembangan dapat diarahkan agar sistem juga mendukung jenis servo lainnya seperti MG, Kondo, atau LewanSoul, yang banyak digunakan pada platform robotik. Hal ini akan memperluas jangkauan pengguna dan kompatibilitas lintas *hardware*.

Implementasi saran-saran di atas secara bertahap akan dapat meningkatkan fungsionalitas, performa, dan adopsi *plugin servo control* untuk berbagai aplikasi robotika yang lebih luas dan kompleks.

DAFTAR PUSTAKA

Abdillah, T. (2021). Optimasi proses produksi animasi 3D dengan menggunakan metode *controller rigging* wajah. *Jurnal TIKA*, 6.

Aburumman, N., & Fratarcangeli, M. (2017). Skin deformation methods for interactive character *animation*. *Communications in Computer and Information Science*, 8, 153.

Adinda, Fitri & Hidayah, Agung & Sunardi, Dandi & Muntahanah, Muntahanah. (2022). Comparison of *the Use* of Blender *and* Sketchup Applications in 3d *Animation* (Case *Study*: PT Rico Putra Selatan. Jurnal Komputer, Informasi dan Teknologi (JKOMITEK). 2. 10.53697/jkomitek.v2i2.1011.

Buys, K., De Laet, T., Smits, R., & Bruyninckx, H. (2021). Blender for robotics: *Integration into the* Leuven *paradigm* for robot *task specification and human motion estimation*. *In SIMPAR* (pp. 15–25). Springer.

Feng, H.-M., Wong, C.-C., Liu, C.-C., & Xiao, S.-R. (2022). ROS-based humanoid robot pose *control system design*. *In IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (pp. 4089–4093). IEEE.

Jaksic, N., Li, B., Maestas, B., & Rothermal, K. (2022). *Dancing* humanoid robots lab *demonstration* for *the first year engineering* students. *In ASEE Annual Conference and Exposition, Conference Proceedings*.

Niiyama, R. (2022). Soft actuation *and* compliant mechanisms in humanoid robots. *Current Robotics Reports*, 3, 1–7.

Robotis. (2025). *U2D2 – ROBOTIS e-Manual*. Diakses 18 Mei 2025, dari https://emanual.robotis.com/docs/en/parts/*interface*/u2d2/

Robotis. (2025). *DYNAMIXEL XL320 – ROBOTIS e-Manual*. Diakses 20 April 2025, dari https://emanual.robotis.com/docs/en/dxl/x/xl320/

Robotis. (2025). *DYNAMIXEL MX-28(2.0) – ROBOTIS e-Manual*. Diakses 20 April 2025, dari https://emanual.robotis.com/docs/en/dxl/mx/mx-28-2/

Robotis. (2025). *DYNAMIXEL SDK – ROBOTIS e-Manual*. Diakses 18 Mei 2025, dari https://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_sdk/overview/

Santos, A., Magboo, M. S., & Magboo, V. P. (2024). *Procedural modeling* for sustainable urban *development and* planning: A Blender *plugin* for 3D *modeling* of Philippine cities.

Subree, M. I. (2023). Unlocking *the potential* of soft robotics *with* Blender *and* Unreal *Engine*: A *powerful* combination for *adaptive* morph *design*. *Computer Science and Information Technology*, 11(1), 11–20.

[Halaman ini sengaja dikosongkan]

LAMPIRAN

L1. Link Video Demonstrasi *Plugin* dengan Robot Humanoid

https://youtu.be/qQx_gFE7Gcg

L2. Link Kode Sumber Lengkap *Plugin*

https://github.com/Informatics-ITS/ta-daf2a

[Halaman ini sengaja dikosongkan]

BIODATA PENULIS

Penulis dilahirkan di Boyolali, 24 Oktober 2002, merupakan anak bungsu dari 3 bersaudara. Penulis telah menempuh pendidikan formal yaitu di TK Salsabila Nogosari, SDIT Iqra' Nogosari, SMPII Al-Abidin Surakarta dan MA Isy Karima Karanganyar. Setelah lulus dari MA tahun 2021, Penulis mengikuti SNMPTN dan diterima di Departemen Teknik Informatika FTEIC - ITS pada tahun 2021 dan terdaftar dengan NRP 5025211015.

Selama perkuliahan Penulis sempat aktif di tim robotika yang dinaungi oleh UKM Robotika ITS dan aktif sebagai Asisten Dosen pada mata kuliah Sistem Operasi dan Jaringan Komputer.