KENDALI GERAKAN ROBOT OPENMANIPULATOR DALAM OPERASI PEMINDAHAN OBJEK BERBASIS ROBOT OPERATING SYSTEM 2

PRAKTIK KERJA LAPANGAN

IVAN LIE NAGASENA

NIM: 312210011

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI DAN DESAIN
UNIVERSITAS MA CHUNG
MALANG

2026

LEMBAR PENGESAHAN

PRAKTIK KERJA LAPANGAN

KENDALI GERAKAN ROBOT OPENMANIPULATOR DALAM OPERASI PEMINDAHAN OBJEK BERBASIS ROBOT OPERATING SYSTEM 2

Oleh:

IVAN LIE NAGASENA

NIM. 312210011

dari:

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS TEKNOLOGI DAN DESAIN

UNIVERSITAS MA CHUNG

Dosen Pembimbing,

Prof. Dr.Eng. Romy Budhi, ST., MT., M.Pd.

NIP. 20070035

Dekan Fakultas Teknologi dan Desain,

Prof. Dr.Eng. Romy Budhi, ST., MT., M.Pd.

NIP. 20070035

Kata Pengantar

Puji syukur penulis panjatkan ke hadirat Tuhan Yang Maha Esa. Atas rahmat dan karunia-Nya, penulis dapat menyelesaikan laporan Praktik Kerja Lapangan (PKL) ini tepat pada waktunya. Laporan ini disusun untuk memenuhi mata kuliah PKL bagi mahasiswa Program Studi Teknik Informatika Universitas Ma Chung.

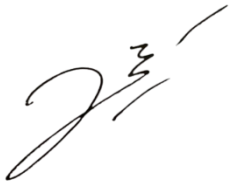
Penulis menyadari bahwa keberhasilan penelitian ini bergantung pada dukungan berbagai pihak. Oleh karena itu, penulis menyampaikan apresiasi dan terima kasih sebesar-besarnya kepada:

1. Prof. Dr.Eng. Romy Budhi, ST., MT., M.Pd, selaku Dekan Fakultas Teknologi dan Desain, Kepala Peneliti sekaligus Dosen Pembimbing. Penulis berterima kasih atas arahan, transfer ilmu, dan bimbingan teknis yang sangat berharga selama proses pembuatan program hingga penyusunan laporan ini.
2. Teman-teman yang telah ikut membantu penelitian ini.
3. Orang tua dan keluarga yang terus memberikan dukungan moral serta doa selama penulis menjalankan tugas ini.

Penulis meyakini bahwa pengalaman penelitian ini memberikan kontribusi nyata bagi pengembangan ilmu pengetahuan. Penelitian lapangan merupakan instrumen penting untuk memvalidasi teori, sebagaimana sering ditekankan dalam berbagai jurnal ilmiah mengenai metodologi penelitian terapan.

Penulis menyadari bahwa laporan ini masih memiliki kekurangan. Oleh karena itu, penulis mengharapkan kritik dan saran yang membangun dari pembaca untuk perbaikan di masa mendatang. Semoga laporan ini memberikan manfaat bagi pengembangan akademik di lingkungan Fakultas.

Malang, 11 Januari 2026



Ivan Lie Nagasena

UNIVERSITAS MA CHUNG

Daftar Isi

UNIVERSITAS MA CHUNG

UNIVERSITAS
MA CHUNG

Daftar Gambar

UNIVERSITAS MA CHUNG

UNIVERSITAS
MA CHUNG

Daftar Tabel

UNIVERSITAS
MA CHUNG

Bab I

Pendahuluan

1.1 Latar Belakang

Sektor manufaktur saat ini sangat mengandalkan teknologi otomasi untuk menjamin konsistensi kualitas dan produktivitas hasil produksi (Adzeman dkk., 2020). Robot lengan menjadi komponen vital dalam ekosistem industri modern karena mampu bekerja tanpa henti dengan tingkat presisi yang stabil. Penggunaan robot ini meliputi berbagai tugas berulang yang menuntut ketelitian tinggi, seperti perakitan komponen elektronik, pengelasan plat logam, hingga operasi pemindahan barang atau *pick and place* (Zhong Ting dkk., 2021). Efisiensi waktu menjadi alasan utama industri mengadopsi teknologi ini agar tetap kompetitif di era industri 4.0. Komunitas peneliti global saat ini memilih *Robot Operating System* (ROS) sebagai kerangka kerja utama untuk membangun logika kontrol yang kompleks dan modular (Deng dkk., 2022).

Interaksi waktu nyata dengan lingkungan yang dinamis sangat penting bagi sistem robotika untuk menjalankan fungsi persepsi visual dan perencanaan gerak secara akurat (Ye dkk., 2023). Meskipun ROS menyediakan ekosistem pengembangan yang kaya, implementasi pada robot fisik sering kali terbentur pada keterbatasan penjadwalan sistem operasi. Kernel standar pada Linux menggunakan prinsip *Completely Fair Scheduler* (CFS) yang fokus pada pembagian sumber daya CPU secara adil kepada seluruh proses aplikasi. Namun, prinsip keadilan ini justru menjadi kendala bagi aplikasi robotika yang membutuhkan prioritas waktu mutlak untuk pengiriman setiap perintah gerak (Ye dkk., 2023). Tanpa optimasi khusus, sistem operasi cenderung menunda proses kendali robot demi menjalankan proses latar belakang lainnya.

Penundaan proses ini menyebabkan lonjakan latensi yang membuat gerakan robot menjadi tersendat (jitter) dan tidak mulus. Ketidakpastian waktu eksekusi *end-to-end* meningkatkan risiko kegagalan manuver robot saat beroperasi di lingkungan dinamis yang kritis terhadap keselamatan (Teper dkk., 2022). Sistem robotika membutuhkan jaminan waktu eksekusi dari awal hingga akhir untuk memastikan perilaku robot yang aman dan dapat diprediksi (Teper dkk., 2022). Kendala latensi ini sering muncul pada pengendalian robot manipulator yang

UNIVERSITAS MA CHUNG

menuntut presisi tinggi saat memindahkan objek. Oleh karena itu, penggunaan sistem operasi yang mendukung komputasi waktu nyata menjadi kebutuhan mendesak dalam pengembangan robotika industri.

Di lingkungan Universitas Ma Chung, penelitian mengenai pengendalian robot OpenManipulator sebelumnya telah dilakukan menggunakan platform MATLAB (Alif, 2025; Kelvin, 2024). Penggunaan MATLAB mempermudah analisis kinematika melalui simulasi, namun memiliki keterbatasan dalam hal skalabilitas aplikasi dan keamanan distribusi data pada jaringan yang luas. Langkah pengembangan selanjutnya beralih menggunakan ROS 2 karena versi terbaru ini mengadopsi standar *Data Distribution Service* (DDS) untuk mendistribusikan data secara terdesentralisasi (Deng dkk., 2022). Arsitektur DDS ini menghilangkan titik kegagalan tunggal (*single point of failure*) dan meningkatkan keamanan data melalui enkripsi bawaan yang tidak tersedia secara maksimal pada platform sebelumnya.

Laporan ini membahas penelitian tentang penerapan kendali gerakan robot OpenManipulator-X berbasis ROS 2 pada versi Humble. Pengerjaan ini juga menerapkan optimasi kernel Linux menggunakan *patch Preempt_RT* untuk menjamin determinisme sistem agar robot mampu memindahkan objek dengan presisi tinggi secara waktu nyata (Ye dkk., 2023). Integrasi ini bertujuan untuk menciptakan sistem kendali yang tidak hanya stabil dalam simulasi Gazebo, tetapi juga responsif saat beroperasi pada perangkat keras fisik. Fokus utama pengerjaan ini adalah sinkronisasi antara perangkat lunak kendali dengan aktuator motor DYNAMIXEL melalui antarmuka komunikasi U2D2.

1.2 Batasan Masalah

Ruang lingkup praktik kerja lapangan ini mencakup poin-poin berikut:

1. Perangkat keras utama adalah robot lengan OpenManipulator-X dengan 4 *Degree of Freedom* (DOF).
2. Sistem operasi yang berjalan adalah Ubuntu 22.04 LTS pada lingkungan mesin virtual VMware.
3. Middleware yang digunakan adalah ROS 2 versi Humble sebagai kerangka kerja utama komunikasi antar node.

4. Fokus pengujian terbatas pada operasi gerak robot terhadap 5 titik tujuan yang ada.

1.3 Rumusan Masalah

Penelitian ini merumuskan konfigurasi lingkungan kerja ROS 2 untuk mengendalikan robot OpenManipulator-X secara stabil. Penelitian ini menganalisis pengaruh optimasi sistem operasi melalui patch *Preempt_RT* terhadap kepastian gerakan robot. Fokus kajian juga mencakup evaluasi efisiensi komunikasi antara perintah perangkat lunak dan respons fisik motor Dynamixel melalui antarmuka U2D2.

1.4 Tujuan

Kegiatan PKL ini memiliki beberapa tujuan utama:

1. Membangun sistem kendali robot OpenManipulator-X yang handal menggunakan arsitektur ROS 2 Humble.
2. Menguji efisiensi gerakan robot dalam lingkungan simulasi Gazebo dan memvalidasi hasilnya pada perangkat keras fisik.
3. Menganalisis stabilitas distribusi data dan latensi antara komputer pengendali dan unit aktuator robot.

UNIVERSITAS MA CHUNG

1.5 Manfaat

Hasil dari praktik kerja lapangan ini memberikan manfaat bagi beberapa pihak:

1. Bagi Mahasiswa: Memperdalam pemahaman teknis mengenai arsitektur ROS 2, manajemen kernel Linux, dan kendali aktuator robotika industri.
2. Bagi Pusat Studi HMI Ma Chung: Menyediakan referensi teknis mengenai prosedur migrasi kendali robot dari MATLAB ke ROS 2 yang mendukung riset lanjutan.
3. Bagi Universitas Ma Chung: Memperkaya portofolio penelitian terapan dalam bidang sistem cerdas dan interaksi manusia dengan mesin (Human-Machine Interaction).

Bab II

Gambaran Umum Perusahaan

2.1 Universitas Ma Chung

Universitas Ma Chung berlokasi di Villa Puncak Tidar N-01, Kecamatan Dau, Kabupaten Malang, Jawa Timur. Yayasan Harapan Bangsa Sejahtera menaungi universitas ini sejak peresmiannya pada tanggal 7 Juli 2007. Universitas Ma Chung menetapkan visi dan misi sebagai landasan operasional institusi.

Visi Memuliakan Tuhan Yang Maha Esa melalui pembentukan karakter, pengembangan ilmu pengetahuan, serta memberikan kontribusi nyata sebagai insan akademis yang kreatif dan inovatif.

Misi Universitas Ma Chung menjalankan misi sebagai berikut:

1. Menyelenggarakan Tri Dharma Perguruan Tinggi (pendidikan, penelitian, dan pengabdian masyarakat) dengan standar tinggi, fokus, dan relevan dengan perkembangan zaman.
2. Membentuk dan mengembangkan generasi pemimpin serta penggerak masyarakat yang berintegritas, berjiwa kepemimpinan, dan berkewirausahaan dengan penekanan pada karakter mulia, kerendahan hati, dan semangat pelayanan.
3. Mendorong sikap serta pemikiran kritis-prinsipil dan kreatif-realistis berdasarkan kepekaan hati nurani yang luhur.
4. Menghasilkan lulusan siap pakai yang berkualitas tinggi dan mampu bersaing di pasar global.
5. Mengambil peran aktif dalam peningkatan peradaban dunia dengan menghasilkan lulusan berwawasan global, toleran, dan cinta damai, serta produktif dalam menghasilkan karya cipta.
6. Melaksanakan pengelolaan perguruan tinggi berdasarkan prinsip ekonomis dan akuntabilitas.

Saat ini, Universitas Ma Chung mengelola 11 program studi yang mencakup Manajemen Bisnis, Akuntansi, Magister Manajemen Inovasi, Sastra Inggris, Pendidikan Bahasa Mandarin, Teknik Informatika, Sistem Informasi, Desain Komunikasi Visual, Teknik Industri, Optometri, serta Farmasi dan Profesi Apoteker.

UNIVERSITAS MA CHUNG

2.2 Program Studi Teknik Informatika

Program Studi Teknik Informatika (PSTI) merupakan bagian dari Fakultas Teknologi dan Desain Universitas Ma Chung. Program studi ini memiliki akreditasi B dari BAN-PT sejak tahun 2016 melalui Surat Keputusan Nomor 0356/SK/BANPT/Akred/S/IV/2016. Sertifikasi ini membuktikan bahwa PSTI memenuhi standar nasional dalam hal tata kelola dan kualitas pengajaran. Tim kurikulum menyusun materi pembelajaran agar selalu relevan dengan standar industri teknologi informasi global yang berkembang sangat cepat.

Mahasiswa dapat memilih satu dari dua jalur konsentrasi yang tersedia untuk mendalami keahlian khusus. Konsentrasi pertama adalah Sistem Cerdas yang menitikberatkan pada pengembangan algoritma kecerdasan buatan, pemrosesan bahasa alami, dan analisis data besar. Konsentrasi kedua adalah Sistem Komputer yang fokus pada integrasi perangkat keras dan lunak, keamanan jaringan, serta ekosistem *Internet of Things* (IoT). Pembagian jalur ini bertujuan untuk memberikan keunggulan kompetitif bagi mahasiswa saat memasuki dunia kerja profesional.

2.3 Pusat Studi Human-Machine Interaction Ma Chung

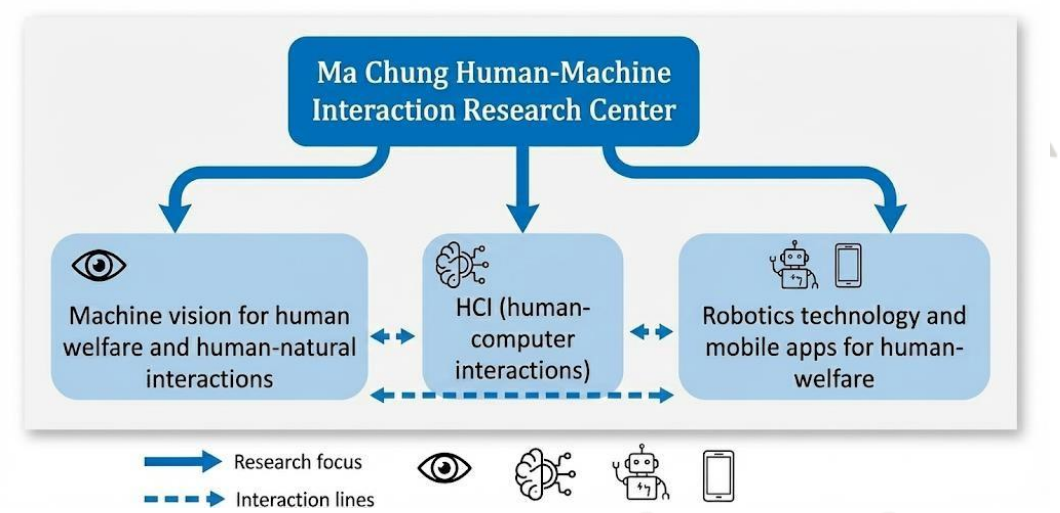
Pusat Studi *Human-Machine Interaction* (HMI) Ma Chung ditempatkan di bawah naungan Program Studi Teknik Informatika sejak didirikan secara resmi pada tanggal 11 September 2019. Lantai 6 Gedung *Research & Development* (R&D) Universitas Ma Chung digunakan sebagai lokasi operasional utama. Berbagai fasilitas komputasi dan perangkat keras modern disediakan di tempat ini guna mendukung kegiatan penelitian yang intensif. Fokus kegiatan diarahkan pada inovasi teknologi dan implementasinya dalam konteks interaksi antara manusia dan mesin.

Tiga bidang kajian riset unggulan dikembangkan oleh pusat studi ini, meliputi:

1. *Machine vision for human welfare and human-natural interactions*, teknologi pengolahan citra digital dikaji pada bidang ini untuk menciptakan

sistem yang mampu menganalisis informasi visual guna meningkatkan kesejahteraan dan kualitas hidup manusia.

2. Interaksi Manusia-Komputer (*Human-Computer Interaction*), penelitian pada bidang ini dipusatkan pada perancangan Desain Antarmuka (*User Interface*) dan Pengalaman Pengguna (*User Experience*) yang intuitif, ergonomis, dan mudah digunakan oleh berbagai kalangan pengguna.
3. Teknologi Robotika dan Aplikasi Seluler, sistem robotika cerdas dan aplikasi seluler dikembangkan pada bidang ini sebagai solusi praktis untuk membantu aktivitas manusia. Topik penelitian mengenai kendali robot OpenManipulator dalam laporan ini tercakup dalam lingkup kajian ini.



Gambar 2.1 Diagram bidang fokus riset pusat studi HMI

Dukungan infrastruktur dan bimbingan teknis diberikan oleh pusat studi untuk memastikan keberhasilan pengembangan sistem yang diteliti. Selain aspek teknis, eksplorasi potensi sumber daya alam dan inovasi pengelolaan bisnis turut didukung sebagai bentuk realisasi visi pusat studi dalam memberikan kontribusi nyata bagi masyarakat.

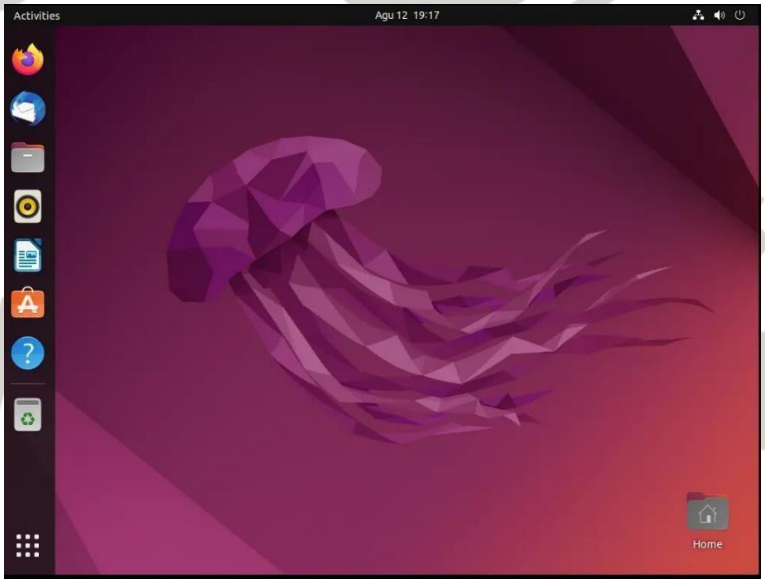
UNIVERSITAS MA CHUNG

Bab III

Tinjauan Pustaka

3.1 Ubuntu 22.04

Pemilihan Ubuntu 22.04 LTS sebagai basis sistem kendali robot didasarkan pada efisiensi arsitektur dan manajemen sumber daya yang superior dibandingkan sistem operasi tertutup seperti Windows 11, dapat dilihat pada gambar 3.1 yang menampilkan halaman utama dari Ubuntu. Berdasarkan studi komparatif terbaru oleh (Al Fajar dkk., 2025), Ubuntu menunjukkan keunggulan signifikan dalam lingkungan dengan sumber daya terbatas, di mana ia mampu beroperasi secara stabil hanya dengan RAM 2 GB, berbeda dengan Windows 11 yang mewajibkan minimal 4 GB dan modul TPM 2.0. Efisiensi ini krusial bagi komputer pendamping (*onboard computer*) robot, karena meminimalkan beban sistem operasi (*overhead*) dan mengalokasikan lebih banyak daya komputasi untuk pemrosesan algoritma robotika. Selain itu, sifat *open-source* dari Ubuntu menghilangkan biaya lisensi, menjadikannya solusi yang lebih ekonomis dan fleksibel untuk pengembangan skala luas.



Gambar 3.1 Halaman utama Ubuntu 22.04

Dalam konteks pengendalian robot manipulator yang menuntut presisi waktu, kinerja kernel menjadi faktor penentu utama. (Ye dkk., 2023)dalam evaluasi kinerja *real-time* ROS 2 mengungkapkan bahwa kernel Linux standar (*Native*-Linux) memiliki kelemahan dalam determinisme, dengan latensi maksimum yang

dapat melonjak hingga 6.243µs akibat mekanisme *Completely Fair Scheduler* (CFS). Untuk mengatasi hal ini, penggunaan *patch PREEMPT_RT* pada kernel Ubuntu 22.04 terbukti mampu menurunkan latensi maksimum secara drastis menjadi sekitar 82µs , dengan rata-rata latensi stabil di angka 2µs . Optimasi ini mengubah Ubuntu menjadi sistem yang mampu menangani tugas *hard real-time*, memastikan bahwa instruksi gerak dikirim ke aktuator robot dengan jeda waktu yang sangat minim dan konsisten.

Aspek keamanan dan stabilitas Ubuntu juga menjadi landasan kuat untuk penerapannya dalam sistem robotika yang terhubung. (Odun-Ayo dkk., 2021) menyoroti bahwa arsitektur keamanan Linux, yang menerapkan *Mandatory Access Control* (MAC) dan manajemen izin berkas yang ketat, memberikan perlindungan yang lebih baik terhadap *malware* dibandingkan sistem operasi lain. Keunggulan ini diperkuat oleh temuan (Al Fajar dkk., 2025)yang mencatat bahwa struktur keamanan *default* Ubuntu membuatnya lebih jarang menjadi target serangan siber. Kombinasi antara stabilitas jangka panjang (LTS), keamanan arsitektural yang ketat, dan fleksibilitas *portability* menjadikan Ubuntu 22.04 lingkungan yang paling andal untuk menjalankan *middleware* ROS 2 dan menjaga integritas operasional robot.

3.2 Robot Operating System 2

Perkembangan teknologi robotika modern telah mengalami pergeseran fundamental dari lingkungan laboratorium yang terkendali menuju aplikasi dunia nyata yang dinamis, tidak terstruktur, dan seringkali kritis terhadap keselamatan. Dalam dekade terakhir, Robot Operating System (ROS) generasi pertama telah berfungsi sebagai standar *de facto* untuk penelitian akademis, menyediakan kerangka kerja yang fleksibel untuk pengembangan perangkat lunak robotika. Namun, seiring dengan meningkatnya kebutuhan untuk menerjemahkan hasil penelitian menjadi produk komersial, keterbatasan arsitektural ROS1 menjadi semakin nyata dan menghambat skalabilitas.

ROS1 awalnya dirancang untuk penelitian akademis dan tidak dibangun dengan mempertimbangkan kendala *real-time* yang ketat, keamanan siber, atau keandalan jaringan yang buruk. Ketergantungan pada *master node* tunggal untuk

UNIVERSITAS MA CHUNG

penemuan (*discovery*) dan perutean komunikasi menciptakan titik kegagalan tunggal (*single point of failure*) yang kritis. Jika *master node* mengalami kegagalan, seluruh jaringan komunikasi robot akan runtuh. Selain itu, mekanisme transport kustom yang digunakan ROS1 (TCPROS/UDPROS) tidak memiliki fitur keamanan bawaan, membiarkan sistem terbuka terhadap penyadapan dan injeksi data, serta tidak menjamin determinisme waktu yang diperlukan untuk kontrol perangkat keras berkecepatan tinggi.

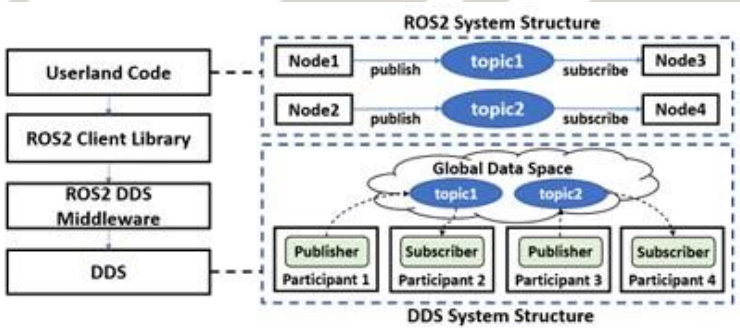
Menanggapi tantangan ini, komunitas robotika global memperkenalkan Robot Operating System 2 (ROS2). ROS2 bukan sekadar pembaruan inkremental, melainkan perombakan arsitektur total yang bertujuan untuk memenuhi standar industri. Perubahan paling radikal adalah penghapusan *master node* pusat dan adopsi *Data Distribution Service* (DDS) sebagai lapisan *middleware* komunikasi standar industri. Transisi ini menjanjikan desentralisasi penuh, dukungan keamanan asli melalui spesifikasi *DDS-Security*, dan potensi untuk operasi *real-time* yang deterministik. Laporan ini menyajikan analisis komprehensif dan mendalam mengenai arsitektur ROS2, mengevaluasi klaim kinerjanya melalui data empiris, membedah postur keamanannya, dan memberikan panduan teknis untuk implementasi sistem yang tangguh.

3.2.1 Fundamental Arsitektur ROS2 dan Integrasi DDS

a) *Data Distribution Service* (DDS) sebagai Tulang Punggung Komunikasi

Perbedaan paling mencolok antara ROS1 dan ROS2 terletak pada lapisan transportasinya. ROS2 mengadopsi *Data Distribution Service* (DDS), sebuah standar terbuka dari *Object Management Group* (OMG) yang dirancang khusus untuk sistem *real-time* terdistribusi yang memerlukan keandalan tinggi (Teper dkk., 2022; Ye dkk., 2023). DDS mengimplementasikan pola komunikasi *publish-subscribe* yang berpusat pada data (*Data-Centric Publish-Subscribe* - DCPS). Dalam paradigma ini, fokus utama bukan pada pengelolaan koneksi antar *node* atau proses, melainkan pada distribusi data itu sendiri dengan jaminan kualitas layanan (*Quality of Service* - QoS) yang spesifik(Deng dkk., 2022).

Struktur DCPS dalam DDS menciptakan "Ruang Data Global" (*Global Data Space*), sebuah konsep abstrak di mana semua data yang dipertukarkan dalam sistem seolah-olah tersedia secara lokal bagi setiap partisipan yang memiliki izin akses (Deng dkk., 2022). Pada gambar 3.2 dapat kita lihat gambaran ini menghilangkan kebutuhan akan *server* pusat atau *broker* pesan, memungkinkan setiap *node* (disebut sebagai *Domain Participant* dalam terminologi DDS) untuk menemukan dan berkomunikasi dengan *node* lain secara *peer-to-peer*. Mekanisme penemuan otomatis (*automatic discovery*) ini menggunakan *multicast* UDP untuk mendeteksi keberadaan partisipan baru dalam jaringan, menegosiasikan kompatibilitas QoS, dan membangun saluran komunikasi *unicast* untuk pertukaran data aktual (Teper dkk., 2022).



Gambar 3.2 Arsitektur DDS dengan protokol DCPS (Deng dkk., 2022)

UNIVERSITAS MA CHUNG

b) Lapisan Abstraksi ROS2 (RCL dan RMW)

Untuk mencegah pengembang ROS2 terkunci pada satu vendor DDS tertentu, arsitektur ROS2 memperkenalkan lapisan abstraksi yang canggih. Kode aplikasi pengguna (*Userland Code*) tidak berinteraksi langsung dengan API DDS, melainkan melalui ROS *Client Library* (RCL) (Deng dkk., 2022). RCL menyediakan antarmuka standar (dalam C++) yang konsisten, terlepas dari implementasi DDS yang digunakan di bawahnya.

Di bawah RCL, terdapat lapisan ROS *Middleware* (RMW). RMW berfungsi sebagai jembatan penerjemah yang memetakan konsep ROS (seperti *Node, Topic, Service, Action*) ke dalam primitif DDS (seperti *Participant, DataWriter, DataReader*) (Deng dkk., 2022). Desain ini memungkinkan integrasi berbagai implementasi DDS, seperti *eProsima*

Fast DDS, *Eclipse Cyclone* DDS, atau RTI *Connext*, yang dapat ditukar hanya dengan mengubah konfigurasi lingkungan (*environment variable*) tanpa perlu mengkompilasi ulang kode aplikasi (Ye dkk., 2023).

c) Kebijakan Quality of Service (QoS)

Salah satu fitur paling kuat yang dibawa DDS ke dalam ekosistem ROS2 adalah konfigurasi *Quality of Service* (QoS). QoS memungkinkan pengembang untuk mendefinisikan perilaku komunikasi secara granular untuk setiap topik, menyesuaikan dengan kebutuhan aplikasi yang spesifik (Deng dkk., 2022).

Beberapa kebijakan QoS yang paling relevan untuk sistem robotika meliputi:

***Reliability* (Keandalan):**

- *Reliable*: Menjamin pengiriman pesan, mirip dengan TCP. Jika paket hilang, *middleware* akan mencoba mengirim ulang. Ini penting untuk perintah kontrol kritis atau parameter konfigurasi.
- *Best Effort*: Mengirim pesan tanpa jaminan penerimaan, mirip dengan UDP. Ini ideal untuk aliran data sensor frekuensi tinggi (seperti video atau LiDAR) di mana data terbaru lebih penting daripada kelengkapan data historis.

***Durability* (Daya Tahan):**

- *Volatile*: Pesan hanya dikirim ke pelanggan yang saat ini terhubung. Pesan lama tidak disimpan.
- *Transient Local*: Penerbit menyimpan sejumlah pesan terakhir (sesuai *history depth*) dan mengirimkannya ke pelanggan baru yang bergabung belakangan ("*late-joiners*"). Ini sangat berguna untuk data statis seperti peta navigasi atau deskripsi robot.

***History & Depth*:** Menentukan berapa banyak pesan yang disimpan dalam antrian DDS sebelum pesan lama ditimpa.

UNIVERSITAS MA CHUNG

***Deadline*:** Menetapkan batas waktu maksimum yang diharapkan untuk kedatangan pesan baru. Jika batas ini dilanggar, sistem dapat memicu kejadian kesalahan (*error event*), yang krusial untuk pemantauan kesehatan sistem *real-time*.

d) Keunggulan Desain Terdistribusi

Adopsi arsitektur terdistribusi penuh melalui DDS memberikan keuntungan strategis bagi pengembangan sistem *Multi-Robot Systems* (MRS). Dalam arsitektur terpusat ROS1, penskalaan sistem ke banyak robot memerlukan konfigurasi jaringan yang rumit dan sangat rentan terhadap kegagalan jaringan yang memisahkan robot dari master. Dalam ROS2, setiap robot adalah entitas mandiri yang berpartisipasi dalam domain DDS yang sama. Hal ini memfasilitasi kolaborasi kawanan (*swarm robotics*), di mana robot dapat masuk dan keluar dari jaringan secara dinamis tanpa mengganggu operasi keseluruhan. Selain itu, DDS mendukung mekanisme pembagian data yang efisien, memungkinkan robot untuk berbagi persepsi lingkungan secara kolaboratif dengan latensi minimal.

3.2.2 Primitif Komunikasi: *Node, Topic, Service,* dan *Action*

Sistem ROS 2 membangun interaksi antar komponen melalui beberapa primitif komunikasi utama. Setiap primitif memiliki peran unik dan dipetakan secara spesifik ke entitas DDS di bawahnya (Deng dkk., 2022).

a) *Node* dan Klasifikasi Fungsional

Node adalah unit pemrosesan tunggal yang menjalankan algoritma tertentu. Di tingkat *middleware*, setiap *node* merepresentasikan satu atau lebih *DDS Domain Participant*. Teper dkk. (2022) mengklasifikasikan *node* berdasarkan mekanisme pemicu (*trigger*) untuk analisis waktu. *Sensor Node* bekerja secara *time-triggered* (berdasarkan interval waktu). *Filter* atau *Actuator Node* bekerja secara *event-triggered* (berdasarkan kedatangan pesan). Sementara *Fusion Node* bekerja secara hibrida untuk menggabungkan data dari berbagai sumber.

UNIVERSITAS MA CHUNG

b) Mekanisme *Discovery* (Penemuan)

ROS 2 tidak lagi menggunakan *master node* untuk menghubungkan antar *node*. Sebagai gantinya, sistem menggunakan *Simple Discovery Protocol* (SDP) dari DDS. Proses ini melibatkan dua tahap utama, tahap pertama adalah *Participant Discovery Protocol* (PDP) untuk mendeteksi *node* baru dalam jaringan melalui *multicast*, tahap kedua adalah *Endpoint Discovery Protocol* (EDP) untuk mencocokkan *publisher* dan *subscriber* berdasarkan topik dan kebijakan QoS yang sesuai (Deng dkk., 2022).

c) *Topic*

Topic memfasilitasi komunikasi asinkron melalui pola *publish-subscribe*. Pengembang menggunakan topik untuk aliran data kontinu. Dalam implementasi DDS, sebuah topik terdiri dari *DataWriter* pada sisi penerbit dan *DataReader* pada sisi pelanggan. ROS 2 mengelola distribusi data ini secara desentralisasi, sehingga setiap *node* mengirimkan data langsung ke pelanggan tanpa melalui perantara pusat (Deng dkk., 2022).

d) *Service*

Service bekerja dengan pola *request-response* yang bersifat sinkron. Sebuah *node* klien mengirimkan permintaan dan menunggu balasan dari *node* pelayan (*server*). Secara teknis, ROS 2 mengimplementasikan *service* menggunakan dua pasang topik internal. Sepasang topik digunakan untuk mengirim permintaan dan menerima balasan, sementara sepasang lainnya mengelola *metadata* transaksi. Primitif ini sangat efektif untuk tugas transaksional singkat seperti permintaan kalibrasi.

e) *Action*

Action merupakan primitif yang menggabungkan mekanisme *service* dan *topic* untuk menangani tugas berdurasi panjang. *Action* terdiri dari tiga bagian: *goal* (permintaan target), *feedback* (laporan progres asinkron), dan *result* (hasil akhir). ROS 2 menggunakan lima topik internal DDS untuk mendukung siklus hidup sebuah *action* (Deng dkk., 2022).

UNIVERSITAS MA CHUNG

Mekanisme ini memastikan robot manipulator dapat mengirimkan progres gerakan lengan secara berkala ke antarmuka pengguna sambil tetap menjalankan kalkulasi lintasan.

f) Interface dan Parameter

Komunikasi memerlukan definisi struktur data baku melalui file .msg, .srv, dan .action. Selain itu, ROS 2 mengelola *Parameter* sebagai nilai konfigurasi di dalam *node*. *Node* lain dapat membaca atau memperbarui nilai ini secara dinamis melalui *service* standar yang disediakan oleh RCL. Hal ini mempermudah penyesuaian batas kecepatan atau parameter kendali robot tanpa memerlukan kompilasi ulang kode.

3.2.3 Manajemen Eksekusi dan Analisis Waktu (*Timing Analysis*)

a) Model Eksekutor ROS2: Jantung Determinisme

Dalam sistem operasi robotika, penerimaan pesan hanyalah langkah awal. Langkah kritis berikutnya adalah penjadwalan komputasi: menentukan kapan dan dalam urutan apa kode pengguna (*callback*) dieksekusi sebagai respons terhadap pesan atau *timer*. Dalam ROS2, tanggung jawab ini dipegang oleh *Executor*.

Executor ROS2 berbeda secara fundamental dari model *threading* tradisional. Alih-alih membiarkan setiap *callback* berjalan pada *thread* sistem operasi tersendiri, *Executor* mengelola kumpulan *callback* di dalam satu atau beberapa *thread* pengguna. Memahami mekanisme *internal Executor* sangat penting untuk menjamin determinisme waktu (Teper dkk., 2022).

Mekanisme kerja *Executor* dapat dibagi menjadi dua fase utama:

- ***Polling Point* (Titik Poling):** Pada fase ini, *Executor* berinteraksi dengan lapisan RMW untuk memeriksa ketersediaan data baru. *Executor* mengumpulkan status dari semua *timer* dan *subscription* yang terdaftar. Jika sebuah *timer* telah habis waktunya atau pesan baru telah tiba di antrian *subscription*, *callback* terkait ditandai sebagai "Activated". *Executor*

kemudian mengambil sampel (*sampling*) dari *job* (instansiasi tugas) yang diaktifkan ini untuk dieksekusi.

- ***Processing Window*** **(Jendela Pemrosesan):** Setelah sampel diambil, *Executor* memasuki fase eksekusi. Dalam implementasi *Single-Threaded Executor* standar, *callback* dieksekusi secara serial (berurutan) tanpa *preemption*. Urutan eksekusi ditentukan oleh kebijakan prioritas internal *Executor*. Secara *default* (misalnya pada ROS2 Foxy), *timer* memiliki prioritas lebih tinggi daripada *subscription* untuk memastikan tugas periodik (seperti *loop* kontrol) didahulukan.

  Implikasi dari desain *non-preemptif* ini sangat signifikan: jika satu *callback* memakan waktu eksekusi terlalu lama (*long-running callback*), ia akan memblokir eksekusi *callback* lain yang mungkin lebih kritis, menyebabkan fenomena *blocking* dan meningkatkan latensi sistem secara keseluruhan.

b) Analisis Rantai Sebab-Akibat (*Cause-Effect Chains*)

Untuk mengevaluasi kinerja *end-to-end* sebuah sistem robot, kita tidak bisa hanya melihat satu *node* secara isolasi. Sebaliknya, kita harus menganalisis "Rantai Sebab-Akibat" (*Cause-Effect Chains*). Rantai ini merepresentasikan jalur propagasi informasi dari stimulus fisik (misalnya, deteksi rintangan oleh sensor) melalui serangkaian *node* pemrosesan (filter, fusi sensor, perencana jalur) hingga menghasilkan respons fisik (misalnya, perintah berhenti ke motor) (Teper dkk., 2022).

Dua metrik kunci didefinisikan untuk mengukur kinerja rantai ini:

- ***Maximum Reaction Time*** **(MRT):** Waktu maksimum yang diperlukan sistem untuk bereaksi terhadap peristiwa eksternal. Ini mengukur latensi terburuk dari saat peristiwa terjadi di lingkungan hingga aktuator mulai bergerak. Metrik ini krusial untuk keselamatan, misalnya dalam pengereman darurat.

- ***Maximum Data Age* (MDA):** Usia maksimum data yang digunakan untuk menghasilkan output aktuator. Ini mengukur kesegaran informasi. Dalam sistem kontrol umpan balik (*feedback control*), menggunakan data yang terlalu tua (MDA tinggi) dapat menyebabkan ketidakstabilan sistem atau osilasi.

c) Klasifikasi *Node* dan Propagasi Data

Dalam analisis *timing*, *node* ROS2 diklasifikasikan berdasarkan mekanisme pemicu (*trigger*) dan peran fungsionalnya:

- ***Sensor Node (Time-Triggered)*:** Dipicu oleh *timer* periodik untuk membaca perangkat keras dan menerbitkan data. *Node* ini adalah awal dari rantai sebab-akibat.

- ***Filter/Actuator Node (Event-Triggered)*:** Dipicu oleh kedatangan pesan pada topik *subscription*. *Node* ini memproses data dan meneruskannya (filter) atau mengonsumsinya (actuator).

- ***Fusion Node (Hybrid)*:** Menggabungkan data dari beberapa sumber. Bisa dipicu oleh *timer* (mengambil data terbaru dari *buffer*) atau oleh salah satu pesan masuk (mekanisme sinkronisasi pesan).

Propagasi data dalam rantai ini melibatkan dua jenis komunikasi:

- ***Inter-Node*:** Komunikasi antar *node* melalui topik DDS. Latensi dipengaruhi oleh *overhead* serialisasi, transmisi jaringan, dan antrian DDS.

- ***Intra-Node*:** Komunikasi antar *callback* di dalam *node* yang sama (misalnya, *callback* sensor menyimpan data ke variabel global yang kemudian dibaca oleh *callback timer*). Latensi di sini didominasi oleh jadwal eksekusi *Executor*.

UNIVERSITAS MA CHUNG

d) Analisis Utilisasi Sistem: *Under-Utilized vs Over-Utilized*

Temuan penelitian menyoroti dampak kritis dari tingkat utilisasi sistem terhadap kinerja *timing*.

- **Sistem *Under-Utilized*:** Jika total waktu komputasi semua *callback* dalam satu siklus kurang dari periode *timer* pemicu, sistem berada dalam kondisi stabil. Latensi *end-to-end* dapat diprediksi dan umumnya linear terhadap jumlah *node* dalam rantai.

- **Sistem *Over-Utilized*:** Jika total beban komputasi melebihi periode *timer*, antrian pesan akan mulai menumpuk. Karena sifat *Executor* yang serial, *callback* yang tertunda akan semakin mundur dalam antrian eksekusi. Penelitian menunjukkan bahwa degradasi kinerja dalam kondisi ini tidak linear, latensi dapat meledak secara eksponensial dan data lama mungkin diproses jauh setelah relevansinya hilang. Sistem robotika *real-time* harus dirancang untuk selalu beroperasi dalam kondisi *under-utilized* untuk menjamin determinisme.

3.2.4 Optimalisasi Sistem Operasi untuk *Real-Time* ROS2

a) Tantangan Kernel Linux Standar (*Native-Linux*)

Meskipun arsitektur ROS2 dirancang untuk efisiensi, kinerjanya pada akhirnya dibatasi oleh sistem operasi tempat ia berjalan. Sebagian besar implementasi ROS2 berjalan di atas distribusi Linux standar (seperti Ubuntu). Kernel Linux standar ("*Native-Linux*") dirancang sebagai sistem operasi *General Purpose* (GPOS). Tujuan utamanya adalah memaksimalkan *throughput* rata-rata dan keadilan (*fairness*) pembagian CPU antar proses.

Dalam desain GPOS, proses *real-time* tidak selalu mendapatkan prioritas mutlak. Ada bagian kode kernel (seperti *interrupt handlers* dan *critical sections* yang dilindungi oleh *spinlock*) yang bersifat *non-preemptif*. Artinya, jika kernel sedang mengeksekusi kode ini, ia tidak dapat dihentikan bahkan oleh tugas *real-time* dengan prioritas tertinggi sekalipun. Fenomena ini menciptakan *unbounded latency* (latensi tak terbatas).

Data empiris menunjukkan bahwa pada beban CPU tinggi (misalnya, saat menjalankan algoritma persepsi berat), *Native*-Linux dapat mengalami lonjakan latensi (*latency spikes*) hingga 6243 mikrotik (µs) (Ye dkk., 2023). Untuk robot industri yang memerlukan siklus kontrol 1 ms (1000 µs), lonjakan sebesar 6 ms berarti hilangnya 6 siklus kontrol berturut-turut, yang dapat menyebabkan gerakan robot tersendat (*jitter*) atau penyimpangan lintasan.

b) **Solusi *Real-Time*: *Patch Preempt_RT***

Untuk mengatasi masalah ini tanpa meninggalkan ekosistem Linux yang kaya, solusi standar industri adalah menerapkan *patch* Preempt_RT (*Real-Time Preemption*) pada kernel Linux (Ye dkk., 2023). Preempt_RT secara fundamental mengubah perilaku kernel dengan tujuan membuat hampir seluruh kode kernel dapat di-*preempt* (diinterupsi).

Mekanisme teknis Preempt_RT meliputi:

- **Mengubah *Spinlock* menjadi *Mutex:*** Dalam *Native-Linux*, *spinlock* menahan CPU secara aktif (*busy-wait*) dan mematikan *preemption*. Preempt_RT menggantinya dengan rt_mutex yang memungkinkan *thread* pemegang kunci untuk tidur dan di-*preempt* oleh tugas prioritas lebih tinggi.

- ***Threaded Interrupt Handlers*:** Menjalankan penangan interupsi (*interrupt handlers*) sebagai *thread* kernel biasa yang memiliki prioritas dan dapat dijadwalkan. Ini mencegah interupsi perangkat keras memblokir tugas *real-time* kritis.

- ***High-Resolution Timers*:** Mengaktifkan pewaktu presisi tinggi untuk penjadwalan yang akurat hingga tingkat mikro detik.

c) Evaluasi Komparatif: *Native vs Preempt_RT*

Pengujian ekstensif menggunakan alat *cyclictest* standar industri untuk mengukur latensi kernel menunjukkan perbedaan drastis antara

Native-Linux dan *Preempt_RT* Linux di bawah beban penuh (*stress test* CPU dan memori), data.

Tabel 3.1 Perbandingan *Native*-Linux *& Preempt_RT* Linux (Ye dkk., 2023)

Metrik Evaluasi	*Native-Linux* (Kernel Standar)	Preempt_RT Linux (Kernel Teroptimasi)	Peningkatan
Latensi Minimum	2 µs	1 µs	Marjinal
Latensi Rata-rata	3 µs	1-2 µs	Signifikan
Latensi Maksimum	6243 µs (6.2 ms)	82 µs (0.08 ms)	>98% Reduksi
Jitter Siklus	> 400 µs	< 60 µs	Sangat Stabil
Distribusi Jitter	Fluktuatif, Ekor Panjang	Distribusi Normal Sempit	Deterministik

Data ini menegaskan bahwa untuk aplikasi kontrol *real-time*, penggunaan *Preempt_RT* bukan sekadar opsi, melainkan kebutuhan. Kernel standar tidak dapat menjamin tenggat waktu (*deadline*) yang ketat, sementara kernel *Preempt_RT* menjaga latensi jauh di bawah ambang batas 100 µs yang umumnya ditoleransi dalam kontrol motor presisi tinggi.

d) Kinerja Komunikasi: ROS1 vs ROS2

Selain optimalisasi kernel, kinerja komunikasi antar-*node* juga dievaluasi. Perbandingan antara ROS1 (berbasis TCP/UDP kustom) dan ROS2 (berbasis DDS) menunjukkan karakteristik yang menarik:

- **Muatan Data Kecil (< 64 KB):** Kinerja ROS1 dan ROS2 relatif sebanding. *Overhead* serialisasi dan lapisan abstraksi DDS di ROS2 sedikit terlihat, namun tidak signifikan.

- **Muatan Data Besar (> 512 KB):** ROS2 mulai menunjukkan keunggulan kinerja yang jelas. Mekanisme fragmentasi dan penyusunan ulang data yang

efisien dalam DDS menangani *bandwidth* tinggi lebih baik daripada tumpukan jaringan ROS1.

- **Stabilitas Latensi:** Pada kernel *Native-Linux*, ROS2 menunjukkan fluktuasi latensi yang lebih besar seiring bertambahnya ukuran data dibandingkan ROS1. Namun, ketika dijalankan di atas Preempt_RT Linux, fluktuasi ini hilang, dan ROS2 memberikan latensi yang sangat rendah dan stabil, bahkan untuk data besar.

e) Optimalisasi *Intra-Process* (*Zero-Copy*)

Salah satu fitur kinerja terpenting di ROS2 adalah komunikasi *intra-process*. Ketika dua node berjalan dalam satu proses sistem operasi yang sama, ROS2 dapat melewati tumpukan jaringan DDS sepenuhnya. Alih-alih melakukan serialisasi data, pengiriman ke soket jaringan, dan deserialisasi, ROS2 hanya mengirimkan *pointer* ke lokasi memori data tersebut.

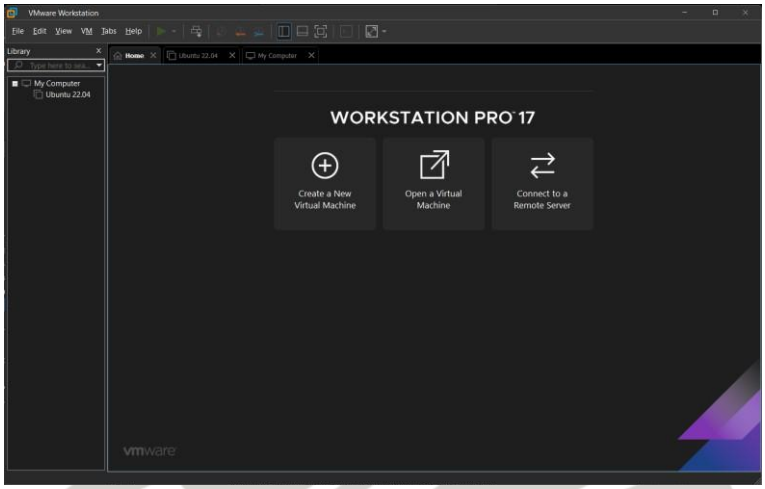
Metode "*Zero-Copy*" ini memberikan pengurangan latensi yang masif, terutama untuk data sensor besar seperti gambar kamera 4K atau awan poin LiDAR. Evaluasi menunjukkan bahwa latensi *intra-process* tetap datar dan hampir nol, tidak peduli seberapa besar ukuran datanya, berbeda jauh dengan komunikasi *inter-process* yang latensinya naik secara linear terhadap ukuran data.

3.3 VMware Workstation

VMware berfungsi sebagai platform virtualisasi yang memisahkan perangkat keras fisik dari sistem operasi sehingga satu *server* fisik dapat menjalankan banyak mesin virtual secara bersamaan. Tesis Savola menjelaskan bahwa *hypervisor* VMware, seperti ESXi, membuat lapisan virtual terpisah yang mengelola alokasi CPU, memori, penyimpanan, dan jaringan untuk setiap VM secara efisien (Savola, 2021). Dengan struktur ini, VMware meningkatkan utilisasi perangkat keras sehingga perusahaan tidak perlu menyediakan satu server fisik untuk satu aplikasi. Pendekatan ini menurunkan biaya *Capex* dan *Opex* serta

UNIVERSITAS MA CHUNG

mendukung tren *datacenter* modern dan *cloud computing.* Gambar 3.3 dibawah ini adalah tampilan halaman utama dari VMware.



Gambar 3.3 Halaman utama VMware

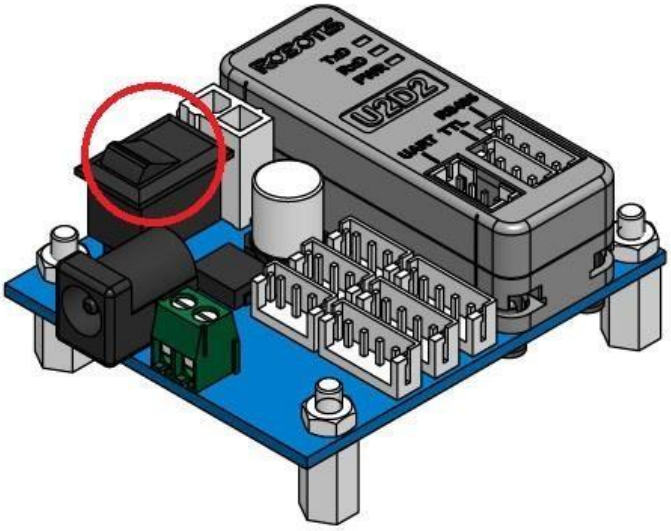
VMware memiliki ekosistem produk virtualisasi server yang luas. ESXi berfungsi sebagai *hypervisor* tipe-1 yang langsung berjalan di atas perangkat keras. vCenter menyediakan manajemen terpusat untuk banyak host ESXi, termasuk pengaturan kluster, distribusi beban, hingga automasi migrasi VM melalui vMotion atau Storage vMotion. Savola menekankan bahwa sejak rilis VMware Workstation pada 1999 dan peluncuran vSphere tahun 2009, VMware berkembang menjadi platform virtualisasi komersial yang dominan berkat inovasi berkelanjutan dan stabilitas produknya (Savola, 2021). Kombinasi ESXi dan vCenter menjadi dasar arsitektur virtualisasi di berbagai perusahaan, ISP, dan pusat data.

VMware juga menyediakan mekanisme migrasi dan pengelolaan infrastruktur berskala besar. Tesis tersebut menunjukkan proses migrasi VM dari satu *platform* ke *platform* VMware lain melalui perencanaan kapasitas, konfigurasi jaringan, pembuatan *datastore* migrasi, hingga pemindahan VM tanpa mengganggu layanan produksi pelanggan (Savola, 2021). Setelah migrasi selesai, *host* lama dapat dihapus, diamankan, dan digunakan kembali. Proses ini menggambarkan kekuatan VMware dalam menyediakan infrastruktur virtual yang fleksibel, mudah dikelola, dan mampu mendukung operasi tingkat perusahaan.

UNIVERSITAS MA CHUNG

3.4 U2D2 *Communication Interface*

Sistem *Robotic Arm System* menggunakan arsitektur perangkat keras terpusat untuk mengelola komunikasi antara unit komputasi dan aktuator. Pengembang menempatkan seluruh komponen kontrol utama dalam satu kotak kontrol tunggal. Kotak kontrol ini berisi catu daya, komputer pengendali, tombol darurat, dan adaptor komunikasi U2D2 (Kim dkk., 2023). Gambar 3.4 adalah bentuk dari *board* U2D2 yang digunakan untuk komunikasi antara komputer dengan robot OpenManipulator.



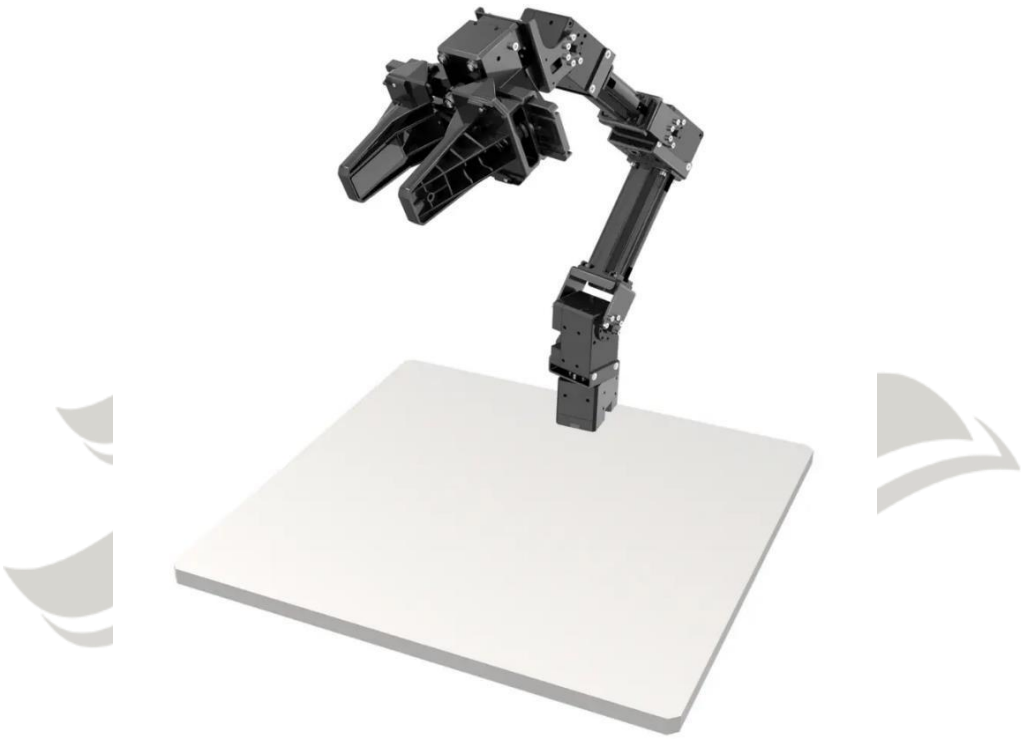
Gambar 3.4 *Board* U2D2

Komputer pengendali terhubung ke adaptor U2D2 melalui sambungan USB. Adaptor ini berfungsi mengubah sinyal USB dari komputer menjadi sinyal serial RS-485 yang diperlukan oleh motor (Kim dkk., 2023). Sifat serial dari protokol ini memungkinkan sistem menghubungkan beberapa dudukan (*mount*) secara paralel hanya dengan menggunakan satu unit U2D2 (Kim dkk., 2023). Desain ini menyederhanakan pengkabelan dan mendukung modularitas sistem tanpa menambah perangkat keras antarmuka yang berlebihan.

3.5 Robot OpenMANIPULATOR-X

OpenManipulator-X dirancang sebagai manipulator serial, sebuah konfigurasi di mana serangkaian *link* (ekstensi mekanis) dihubungkan secara berurutan oleh sambungan (*joint*) *revolute* yang digerakkan oleh motor. Struktur

topologi robot ini terdiri dari tujuh segmen utama: basis (*base*), pinggang (*waist*), bahu (*shoulder*), lengan atas (*upper arm*), siku (*elbow*), lengan bawah (*lower arm*), dan pergelangan (*wrist*) atau *end-effector* (Adzeman dkk., 2020). Gambar 3.5 dibawah ini adalah bentuk dari robot OpenManipulator-X.



Gambar 3.5 Robot OpenManipulator-X

Salah satu fitur paling distingtif dari OpenManipulator-X adalah penggunaan aktuator seri DYNAMIXEL-X seperti pada gambar 3.6. Motor pintar ini mengadopsi teknologi komunikasi *daisy chain*, yang memungkinkan kabel data dan daya dihubungkan secara seri dari satu motor ke motor berikutnya. Arsitektur ini secara drastis mengurangi kompleksitas pengkabelan yang biasanya menjadi masalah pada robot artikulasi, serta memungkinkan modularitas tinggi di mana pengguna dapat menambah atau mengurangi jumlah sambungan (DOF) sesuai kebutuhan aplikasi spesifik. Selain itu, sebagian besar komponen struktural robot ini dirancang agar dapat diproduksi menggunakan teknologi pencetakan 3D (*3D printing*), dengan fail desain yang disediakan secara terbuka oleh Robotis, menjadikannya platform yang sangat adaptif untuk modifikasi riset (Adzeman dkk., 2020).

Gambar 3.6 Motor servo Dynamixel

OpenManipulator-X memiliki spesifikasi operasional yang dioptimalkan untuk beban kerja ringan dan presisi moderat. Tabel 3.2 dibawah ini akan merangkum parameter teknis utama perangkat keras ini.

Tabel 3.2 Spesifikasi Robot OpenManipulator-X

Parameter	Satuan	Nilai spesiikasi
Derajat Kebebasan (DOF)	-	4
Beban Maksimum (Payload)	gram	500
Berat Total	kg	0,7
Rentang Gripper	mm	20-75
Kecepatan Sambungan	RPM	46
Repeatability	mm	<0,2
Kontroler Utama	-	PC, OpenCR

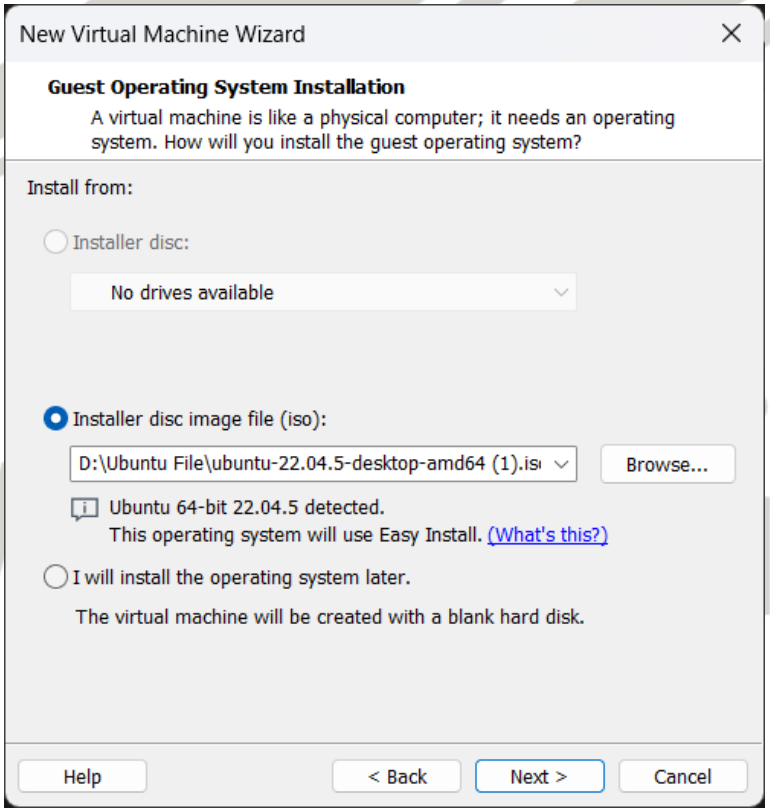
UNIVERSITAS
MA CHUNG

Bab IV

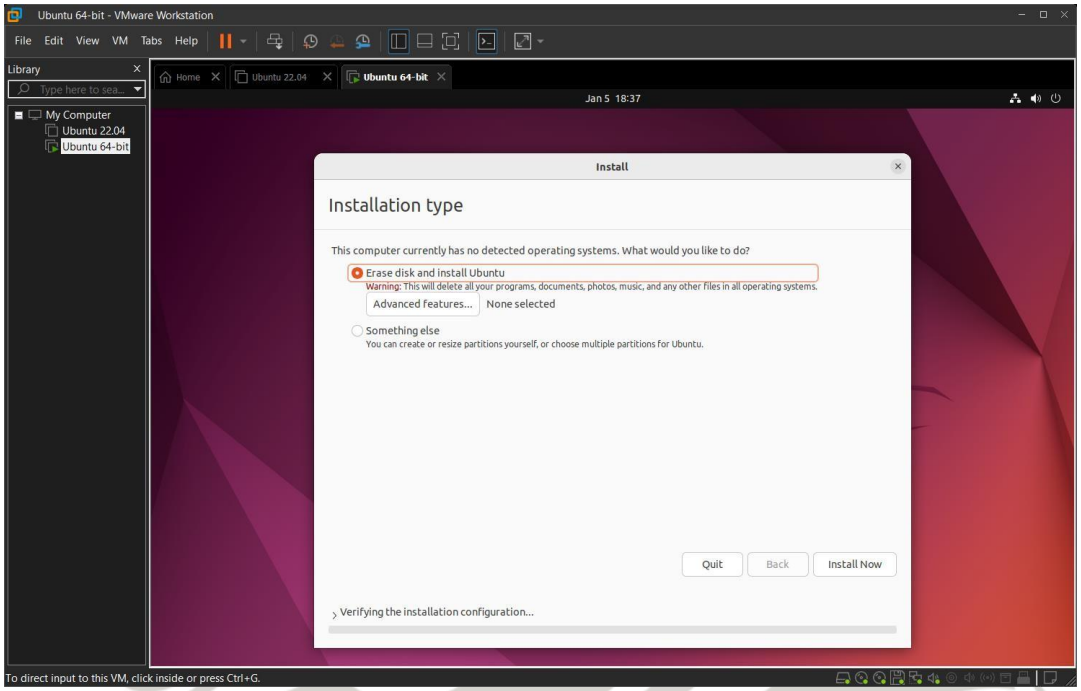
Deskripsi Data dan Hasil Praktik Kerja Lapangan

4.1 Persiapan *Environment* Kerja

Tahap awal pengembangan sistem kendali robot dimulai dengan mempersiapkan lingkungan kerja perangkat lunak (*software environment*). Instalasi sistem operasi Ubuntu 22.04 LTS dilakukan pada komputer pengendali (PC) di dalam *virtual machine* VMware seperti pada gambar 4.1. Setelah instalasi pada VMware selesai, maka akan dilanjutkan dengan instalasi OS Ubuntu yang bisa kita lihat pada gambar 4.2. Pemilihan sistem operasi ini didasarkan pada efisiensi arsitektur dan manajemen sumber daya yang lebih unggul dibandingkan sistem operasi lain, terutama pada perangkat dengan spesifikasi terbatas (Al Fajar dkk., 2025).



Gambar 4.1 Instalasi Ubuntu pada VMware



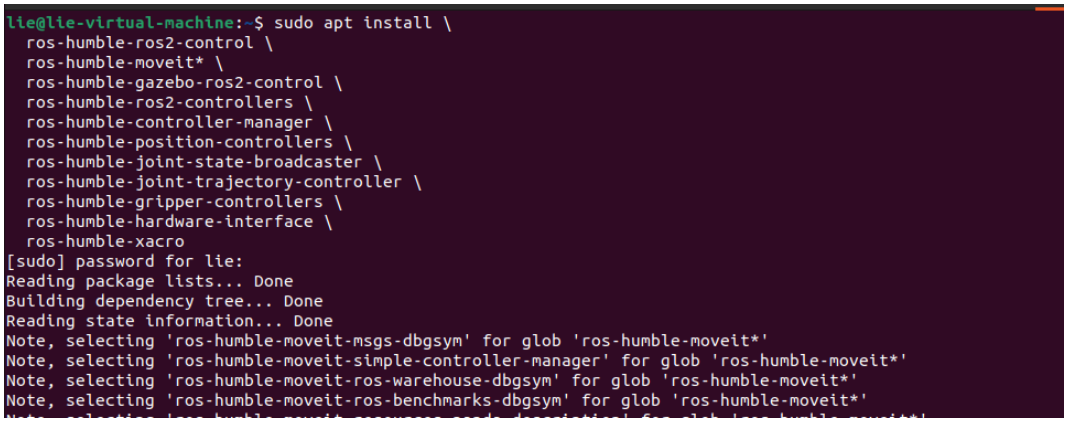
Gambar 4.2 Instalasi OS Ubuntu

Selanjutnya, pada gambar 4.3 dilakukan instalasi *Robot Operating System 2* (ROS 2) dikonfigurasi sebagai kerangka kerja utama untuk komunikasi antar *node* robot. Perintah yang digunakan adalah **sudo apt install ros-humble-desktop**. Password: <masukkan password OS>

```
lie@lie-virtual-machine:~$ sudo apt install ros-humble-desktop
[sudo] password for lie:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  autoconf automake autotools-dev binutils binutils-common
  binutils-x86-64-linux-gnu blt build-essential ca-certificates-java catch2
  cmake cmake-data cppcheck default-jdk default-jdk-headless default-jre
  default-jre-headless default-libmysqlclient-dev dh-elpa-helper
  docutils-common dpkg-dev fakeroot fonts-dejavu-extra fonts-lyx freeglut3 g++
  g++-11 gcc gcc-11 gdal-data gfortran gfortran-11 google-mock googletest
  graphviz hdf5-helpers i965-va-driver ibverbs-providers icu-devtools
  intel-media-va-driver java-common javascript-common libaacs0 libaec-dev
  libaec0 libalgorithm-diff-perl libalgorithm-diff-xs-perl
  libalgorithm-merge-perl libann0 libaom-dev libaom3 libarmadillo-dev
  libarmadillo10 libarpack2 libarpack2-dev libasan6 libasound2-dev
  libassimp-dev libassimp5 libatk-wrapper-java libatk-wrapper-java-jni
  libavcodec-dev libavcodec58 libavformat-dev libavformat58 libavutil-dev
  libavutil56 libbdplus0 libbinutils libblas-dev libblas3 libblkid-dev
```

Gambar 4.3 Instalasi ROS2

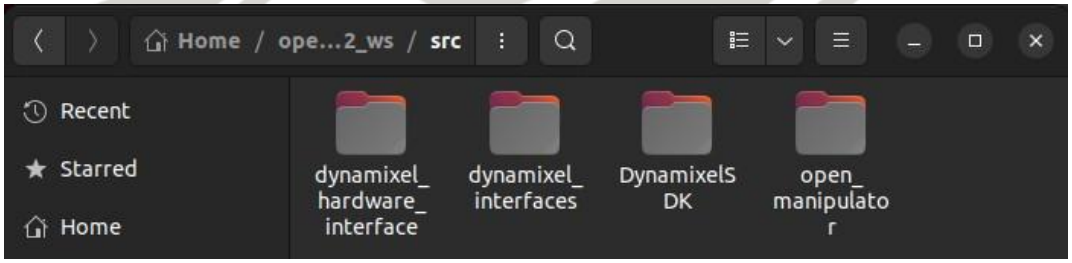
Workspace ROS 2 dibuat dan diatur, serta paket-paket *driver* yang diperlukan untuk OpenManipulator-X dikompilasi seperti pada gambar 4.4 dan 4.5. setelah berhasil maka akan *file* akan tertata seperti pada gambar 4.6.



Gambar 4.4 Instalasi paket OpenManipulator-X

```
$ mkdir -p colcon_ws/src
$ cd ~/colcon_ws/src/
$ git clone -b humble https://github.com/ROBOTIS-GIT/DynamixelSDK.git
$ git clone -b humble https://github.com/ROBOTIS-GIT/open_manipulator.git
$ git clone -b humble https://github.com/ROBOTIS-GIT/dynamixel_hardware_interface.git
$ git clone -b humble https://github.com/ROBOTIS-GIT/dynamixel_interfaces.git
$ cd ~/colcon_ws && colcon build --symlink-install
```

Gambar 4.5 Pembuatan *folder workspace* dan instalasi paket independen



Gambar 4.6 Tampilan paket independen yang telah di instalasi

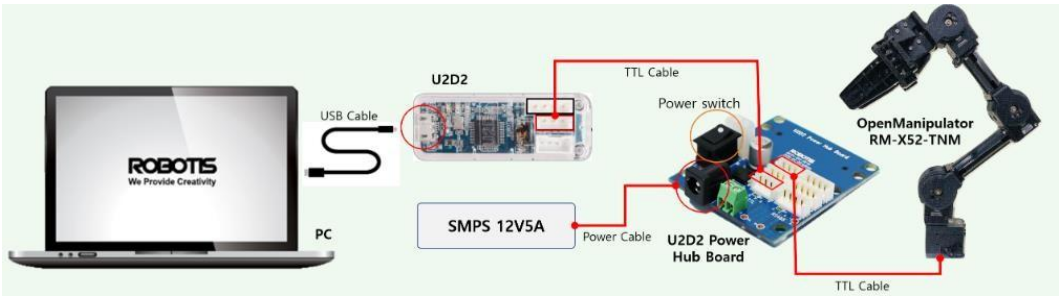
Setelah proses instalasi selesai, pada gambar 4.7 dapat kita lihat cara untuk memverifikasi keberhasilan konfigurasi dilakukan dengan memeriksa daftar topik yang aktif pada sistem ROS 2 melalui terminal dengan perintah **ros2 topic list**. Langkah ini krusial untuk memastikan bahwa lingkungan pengembangan telah siap digunakan untuk tahap simulasi maupun eksekusi *real-time*. *Topic* yang digunakan pada penelitian ini ada 3, yaitu:

a. /arm_controller/joint_trajectory
b. /rosout
c. /clock

```
lie@lie-virtual-machine:~$ ros2 topic list
/arm_controller/controller_state
/arm_controller/joint_trajectory
/arm_controller/state
/arm_controller/transition_event
/attached_collision_object
/clock
/collision_object
/display_contacts
/display_planned_path
/dynamic_joint_states
/dynamixel_hardware_interface/dxl_state
/gripper_controller/transition_event
/joint_state_broadcaster/transition_event
/joint_states
/monitored_planning_scene
/motion_plan_request
/parameter_events
/planning_scene
/planning_scene_world
/recognized_object_array
/robot_description
/robot_description_semantic
/rosout
/servo_node/collision_velocity_scale
/servo_node/delta_joint_cmds
/servo_node/delta_twist_cmds
/servo_node/publish_planning_scene
/servo_node/status
/tf
/tf_static
/trajectory_execution_event
lie@lie-virtual-machine:~$
```

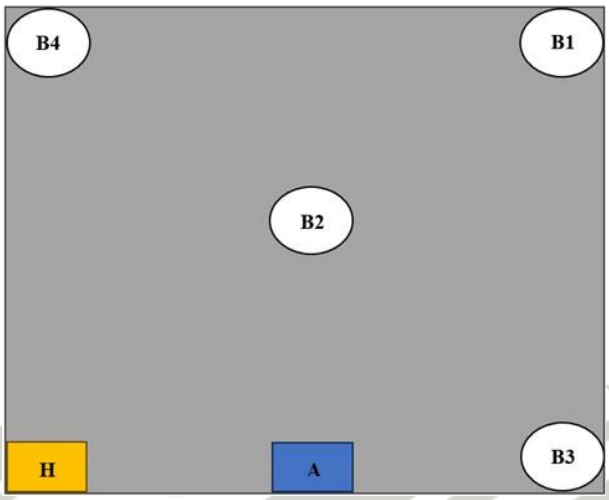
Gambar 4.7 Output dari ros2 *topic list*

Gambar 4.8 menunjukkan bagaimana cara distribusi data dari Komputer menuju robot OpenManipulator-X. Data pada komputer akan dikirimkan kepada U2D2, kemudian data akan di terjemahkan oleh U2D2 menjadi protokol komunikasi serial TTL(*Transistor-Transistor Logic*), sinyal data ini kemudian dikirim ke U2D2 *Power Hub Board* menggunakan kabel TTL, yang juga menerima daya listrik stabil 12V 5A. Kemudian Power Hub menyatukan jalur data dan sumber listrik ke dalam satu jalur distribusi. Akhirnya, kabel TTL mentransmisikan paket gabungan tersebut ke robot, dimana setiap aktuator Dynamixel membaca data, memvalidasi ID, dan menggerakkan sendi robot sesuai perintah.

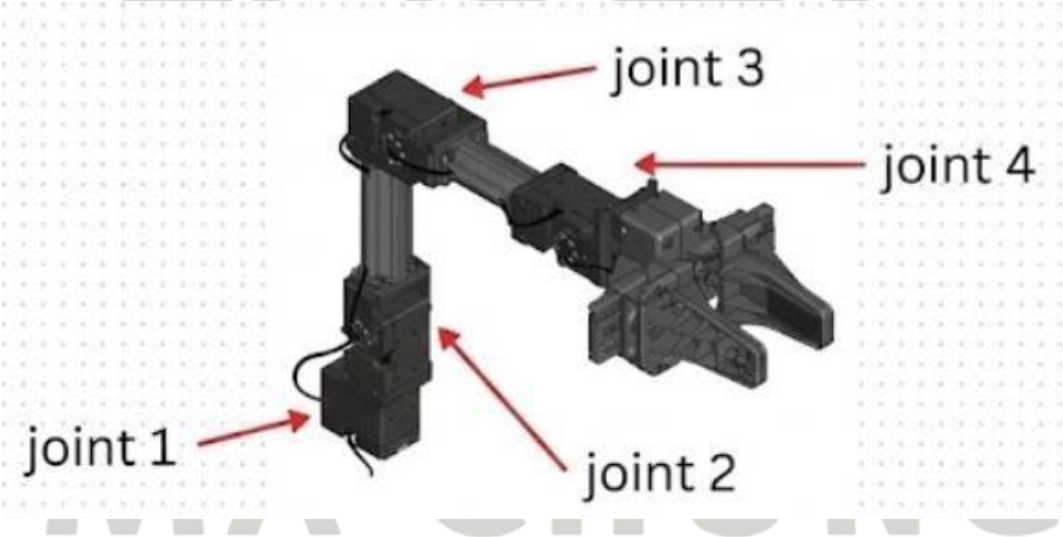


Gambar 4.8 Ilustrasi koneksi PC dengan robot OpenManipulator-X

Dapat dilihat pada gambar 4.9, ini adalah ilustrasi dari posisi robot arm dan titik tujuan yang digunakan untuk pengujian. Robot akan berada pada titik A, dan berdiri dengan posisi *init pose* seperti pada gambar 4.10.



Gambar 4.9 Ilustrasi posisi robot *arm* dan titik tujuan



Gambar 4.10 Keterangan *joint* dan *Init pose*

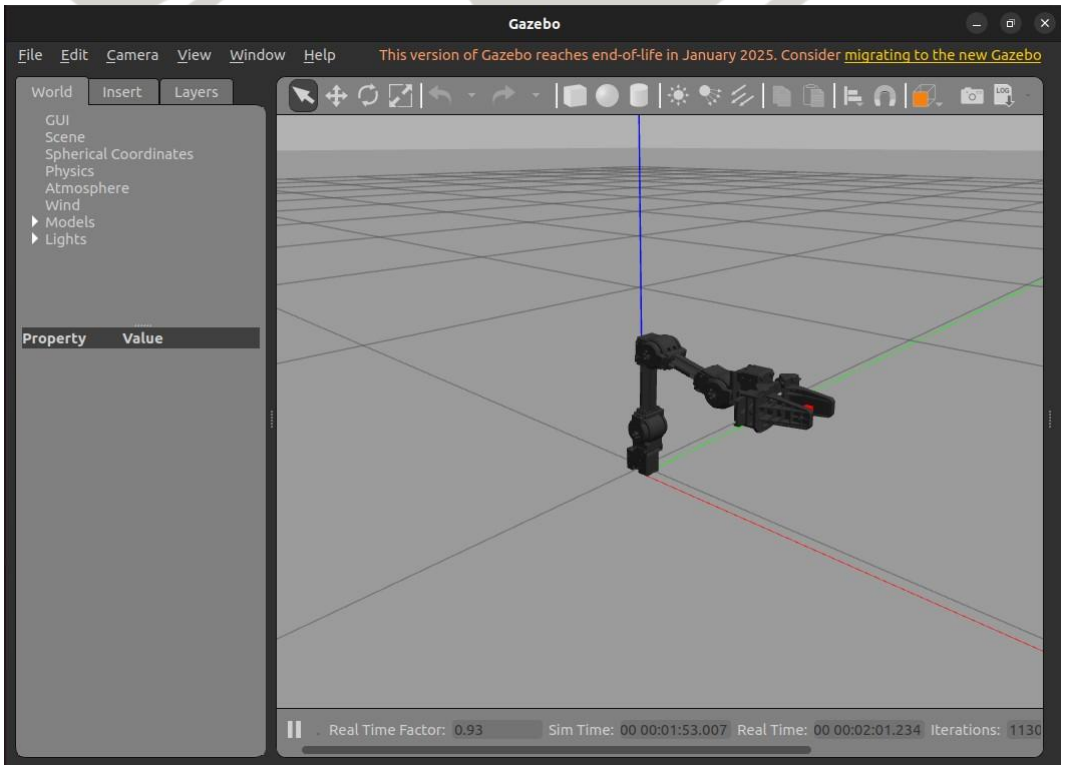
Pengambilan data dari setiap titik tujuan robot dilakukan dengan cara manual, yaitu dengan cara mengambil data sensor dari setiap *joint* yang pada robot *arm*. Untuk satuan koordinat sudut yang digunakan untuk setiap *joint* dari robot *arm* ini adalah radian, pada tabel 4.1 bisa kita lihat untuk koordinat setiap titik yang digunakan pada penelitian ini.

Tabel 4.1 Koordinat *joint* titik tujuan

Titik	Joint 1	Joint 2	Joint 3	Joint 4
Home	1.65	-0.89	0.82	1.03
B1	-0.5	0.23	0.15	0.22
B2	-0.02	-0.75	0.71	1.03
B3	-1.62	-0.93	1.04	0.73
B4	0.45	0.23	0.15	0.22

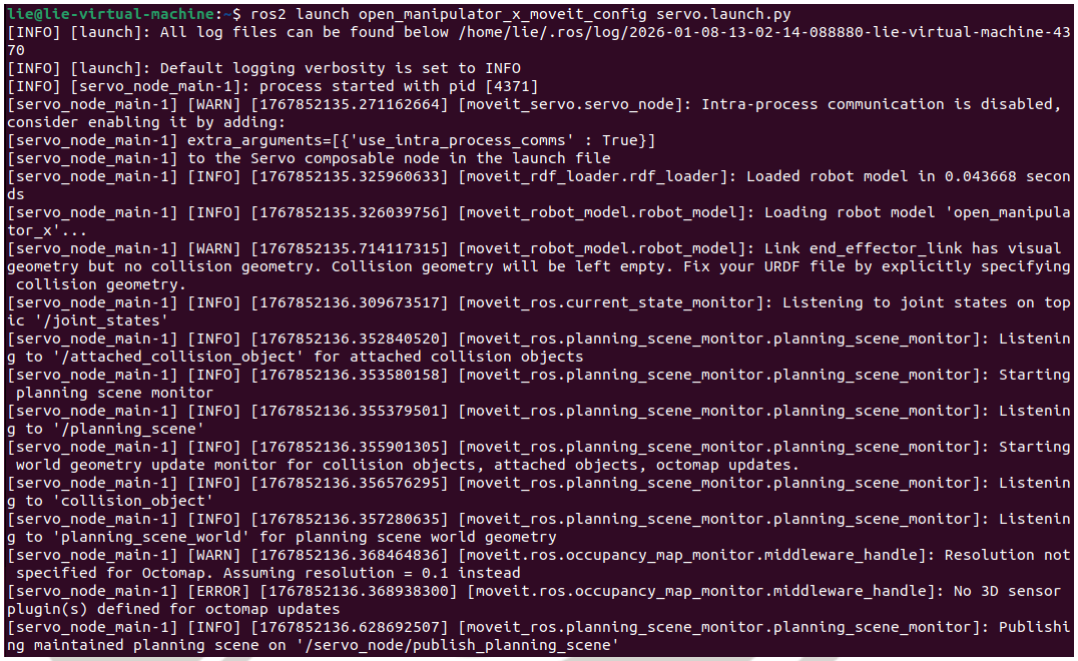
4.2 Pengoperasian Robot Menggunakan Simulator Gazebo

Sebelum pengujian dilakukan pada perangkat keras fisik, simulasi dijalankan menggunakan Gazebo. Hal ini bertujuan untuk memvalidasi logika pergerakan dan algoritma kendali tanpa risiko kerusakan pada perangkat keras. Model robot yang dideskripsikan dalam format URDF (*Unified Robot Description Format*) dimuat ke dalam lingkungan simulasi kosong di Gazebo seperti pada gambar 4.11.



Gambar 4.11 Tampilan model robot *arm* pada simulator Gazebo

Kemudian perintah **ros2 launch open_manipulator_x_moveit_config servo.launch.py** pada gambar 4.12 harus dijalankan terlebih dahulu agar robot arm pada simulator dapat berjalan.



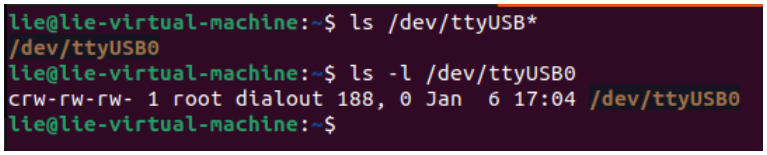
Gambar 4.12 Perintah *MoveIt servo* secara *real-time*

Perintah kecepatan dan posisi dikirimkan melalui terminal ROS 2 untuk menggerakkan sendi-sendi robot virtual. Respons visual robot dalam simulator diamati untuk memastikan kinematika gerak berjalan sesuai dengan logika yang diharapkan. Simulasi ini membuktikan bahwa *node* pengendali berhasil menerjemahkan perintah topik menjadi aksi gerak pada *joint* robot, sebagaimana ditunjukkan pada visualisasi simulasi. Keberhasilan pada tahap ini menjadi indikator bahwa algoritma siap diterapkan pada robot fisik.

4.3 Pengoperasian Robot Fisik Secara Real-Time

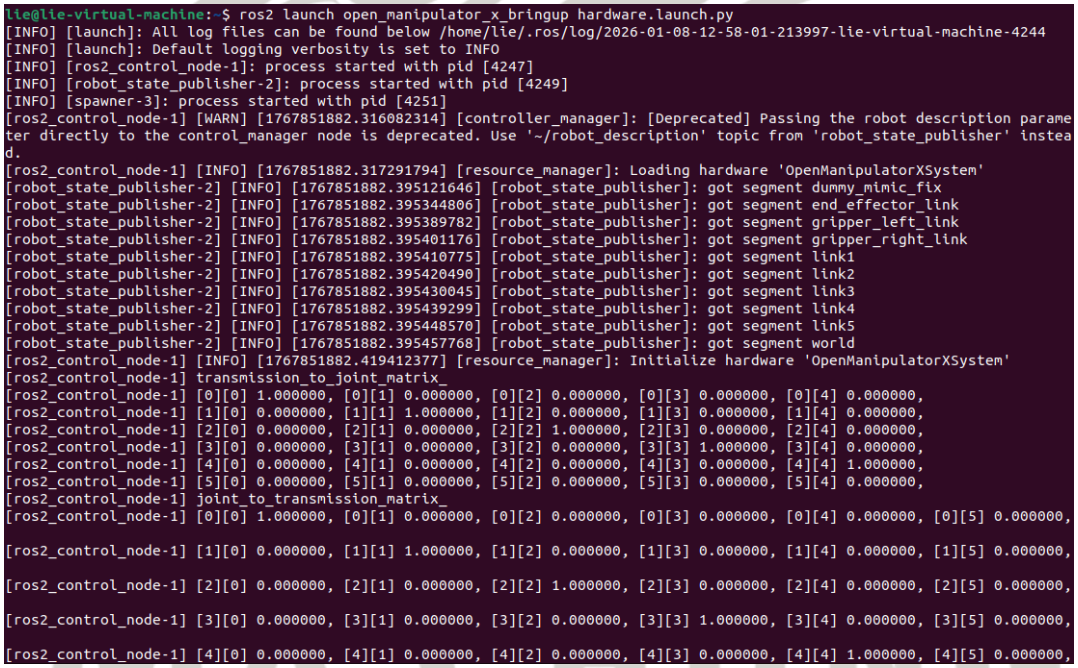
Pada tahap ini, robot fisik OpenManipulator-X dihubungkan ke komputer pengendali menggunakan adaptor U2D2. Adaptor ini berfungsi mengubah sinyal data dari USB komputer menjadi sinyal serial RS-485 yang digunakan oleh aktuator DYNAMIXEL (Kim dkk., 2023). Hak akses (*permission*) diberikan pada *port* USB ttyUSB0 agar ROS 2 dapat berkomunikasi secara langsung dengan motor servo. Seperti pada gambar 4.13 dapat kita lihat hasil dari perintah **ls -l /dev/ttyUSB0** yang menghasilkan **crw-rw-rw** berarti port telah memiliki izin untuk baca/tulis.

UNIVERSITAS MA CHUNG

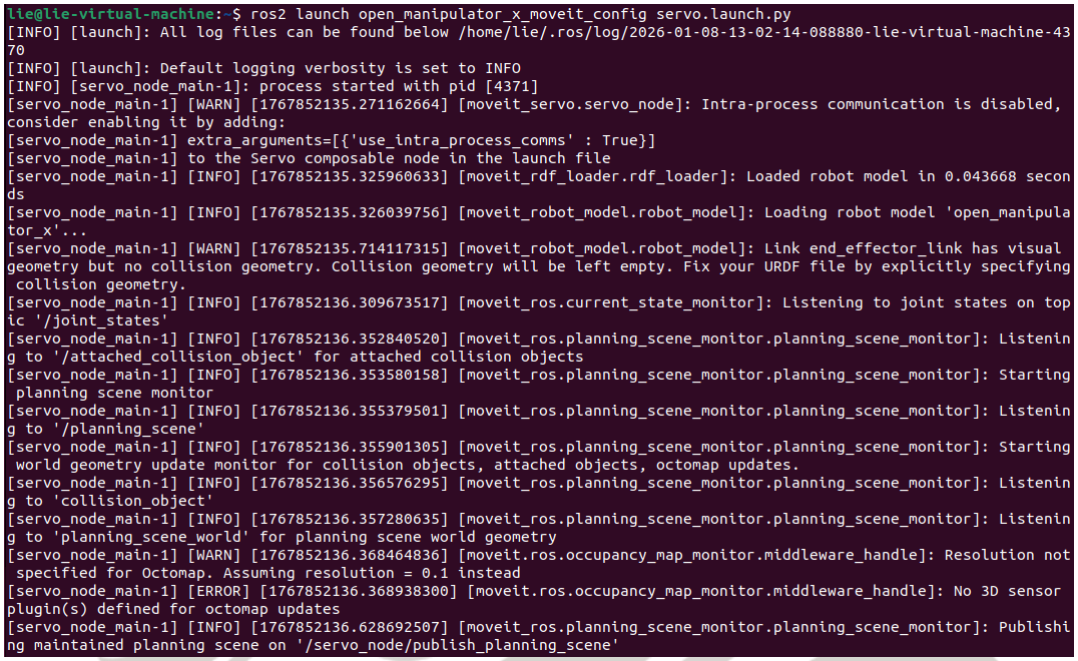


Gambar 4.13 Verifikasi *port* USB

Launch file utama dijalankan untuk mengaktifkan seluruh *node* pengendali perangkat keras. Dapat di lihat pada gambar 4.14 dan 4.15 terdapat dua perintah yang digunakan sebagai *launch file*, yaitu **ros2 launch open_manipulator_x_bringup hardware.launch.py** dan **ros2 launch open_manipulator_x_moveit_config servo.launch.py**, kedua perintah ini harus berjalan secara bersamaan di dua terminal yang berbeda saat akan melakukan pengoperasian robot fisik secara real-time.



Gambar 4.14 Perintah *launch* torsi ke hardware

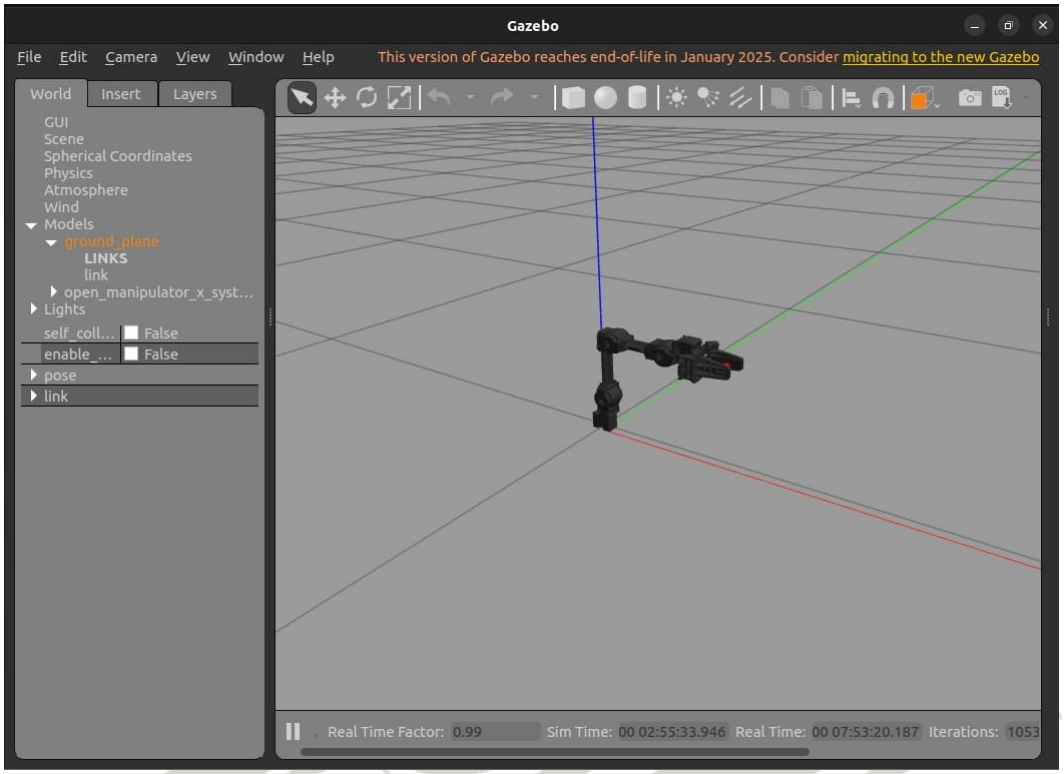


Gambar 4.15 Perintah *MoveIt servo* secara *real-time*

4.4 Pengujian Gerakan Robot Pada Simulator Gazebo

Pengujian pada simulator gazebo digunakan untuk memastikan koordinat sudut yang digunakan sudah benar dan tidak akan melewati batas kemampuan dari robot. Pengujian fungsionalitas gerak robot dibagi menjadi dua skenario utama untuk mengevaluasi presisi posisi dan fleksibilitas lintasan, yaitu pengujian *point-to-point* dan *multi-point*. Pengujian ini dimulai dengan robot arm berada pada init pose dimana semua koordinat sudut *joint* dimulai dari nilai 0.00 radian seperti yang bisa kita lihat pada gambar 4.16.

UNIVERSITAS MA CHUNG



Gambar 4.16 *Init pose* dari robot OpenManipulator-X

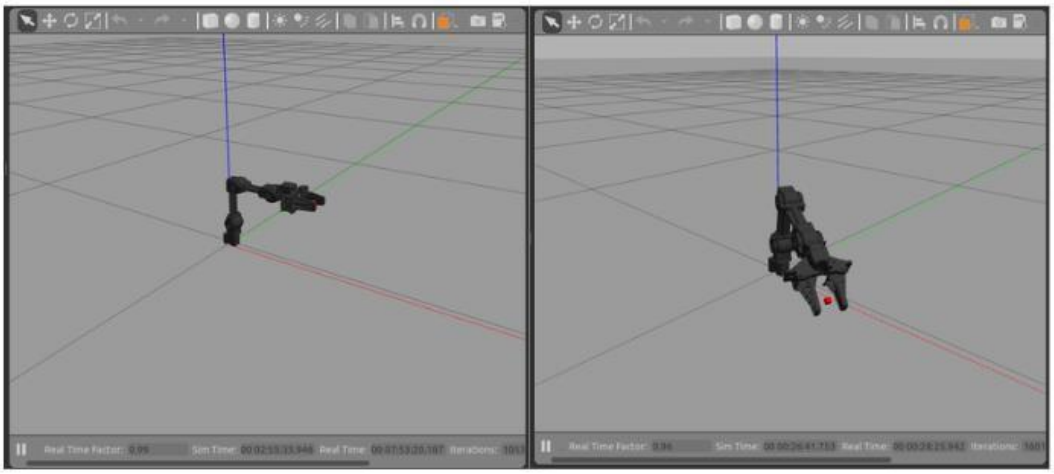
4.4.1 Pengujian Gerak Point to Point

Pengujian ini memfokuskan validasi pada akurasi pencapaian satu koordinat target dalam ruang simulasi. Sistem simulator menerima *input* koordinat tujuan dan menghitung perpindahan sudut sendi yang diperlukan secara otomatis. Lingkungan Gazebo kemudian memvisualisasikan respons pergerakan sendi robot dari posisi awal menuju posisi akhir tanpa hambatan fisik.

Sistem pemantau mencatat deviasi antara posisi yang diperintahkan dan posisi aktual yang dicapai oleh model robot dalam simulasi. Hasil pengamatan menunjukkan bahwa *controller* virtual mampu mengarahkan *end-effector* ke titik sasaran dengan presisi tinggi. Validasi ini memastikan bahwa parameter kinematika yang didefinisikan dalam file URDF telah sesuai dengan spesifikasi teknis robot, sehingga aman untuk diterapkan pada robot fisik.

Ilustrasi pergerakan robot pada simulator akan ditampilkan pada gambar 4.17, dimana bagian sebelah kiri adalah init pose dan sebelah kanan adalah ketika robot telah sampai pada koordinat titik B1. Hasil dari pengujian dapat dilihat pada tabel 4.2. Kode program dapat dilihat pada Lampiran 1.

UNIVERSITAS MA CHUNG



Gambar 4.17 Ilustrasi gerak *point-to-point* pada simulator

Tabel 4.2 Hasil pengujian *point-to-point* simulator

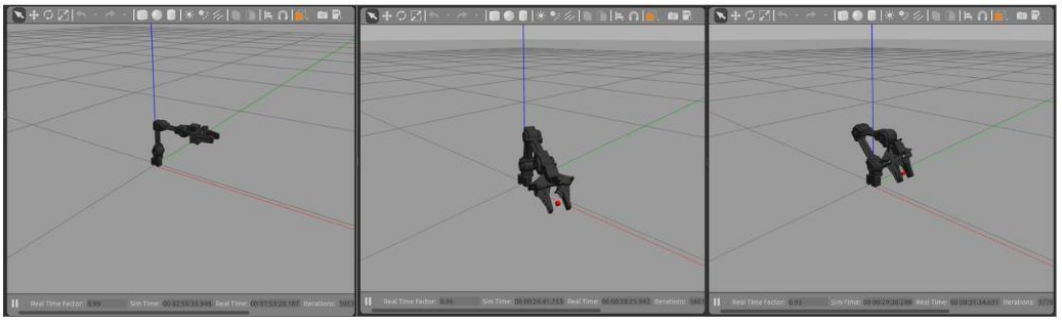
Gerakan	Keberhasilan
Init pose →Home	✓
Init pose → B1	✓
Init pose → B2	✓
Init pose → B3	✓
Init pose → B4	✓

4.4.2 Pengujian Gerak Multi Point

Pengujian gerak *multi-point* dalam simulasi bertujuan untuk mengevaluasi kelancaran interpolasi lintasan yang kompleks. Sistem mengirimkan serangkaian titik koordinat berurutan yang membentuk lintasan melengkung atau berpola tertentu ke dalam simulator. *Physics engine* Gazebo menyimulasikan dinamika gerakan, termasuk inersia dan gravitasi, saat robot berpindah antar titik.

Simulator memvisualisasikan bagaimana kontroler menangani transisi kecepatan dan akselerasi di setiap titik singgah. Pengamatan difokuskan pada deteksi gerakan yang tersendat atau *overshoot* yang mungkin terjadi akibat kesalahan parameter PID virtual. Hasil simulasi mengonfirmasi bahwa robot virtual dapat mengikuti lintasan yang telah ditentukan secara mulus, membuktikan bahwa algoritma pembentukan jalur berfungsi dengan baik sebelum diuji pada beban kerja nyata.

Ilutrasi pergerakan robot pada simulator akan ditampilkan pada gambar 4.18, dimana bagian sebelah kiri adalah init pose, tengah adalah ketika robot sampai pada koordinat titik B1 dan sebelah kanan adalah ketika robot telah sampai pada koordinat titik B2. Hasil dari pengujian dapat dilihat pada tabel 4.3. Kode program dapat dilihat pada Lampiran 2.



Gamar 4.18 Ilustrasi gerak *multi-point* pada simulator

Tabel 4.3 Hasil pengujian *multi-point* simulator

Gerakan	Keberhasilan
Init pose → B1 – B2	✓
Init pose → B3 – B2	✓
Init pose → B2 – B4	✓
Init pose → B1 – B3	✓
Init pose → B4 – B3	✓
Init pose → Home – B1	✓

4.5 Pengujian Gerakan Robot Fisik Secara Real-time

Pengujian gerakan robot fisik dilakukan dengan menggunakan robot OpenManipulator-X dimana posisi awal dan koordinat sudut yang digunakan sama dengan pengujian menggunakan simulator.

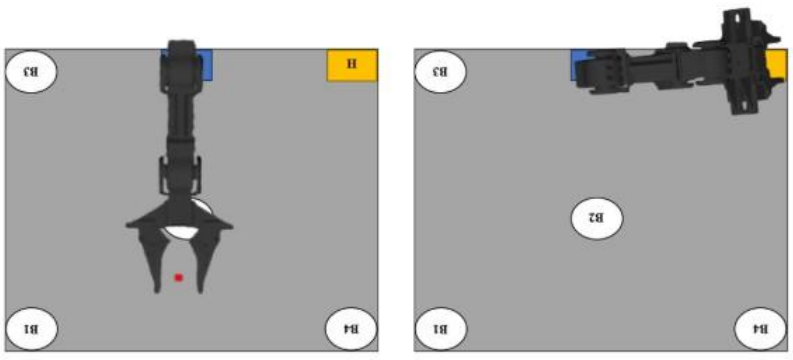
4.5.1 Pengujian Gerak Point to Point

Sistem mengawali operasi dengan membangkitkan *node* ROS 2 sebagai pusat kendali. *Node* ini segera membangun jalur komunikasi *publisher* ke topik perintah robot. Kode program kemudian mengaktifkan mekanisme penghitung waktu mundur (*timer*) sebelum transmisi data bermula. Langkah ini memberi jeda krusial bagi sistem untuk mematangkan stabilitas koneksi jaringan.

Setelah koneksi stabil, fungsi penyusun pesan mulai merakit paket instruksi gerak. Paket ini memuat identitas setiap sendi beserta koordinat sudut tujuan dalam satuan radian. Kode juga menetapkan batas durasi eksekusi agar kontroler robot dapat menghitung interpolasi kecepatan secara otomatis. Mekanisme pengirim lantas melontarkan paket data tersebut menuju aktuator robot dalam satu kali pengiriman.

Sistem tidak langsung mematikan proses setelah pengiriman data terjadi. Program sengaja menahan siklus eksekusi tetap hidup selama beberapa detik hingga robot tuntas mencapai target. Penahanan ini mencegah pemutusan koneksi prematur yang dapat menghentikan robot di tengah jalan. Akhirnya, fungsi utama menutup *node* secara bersih dan mengakhiri program tanpa memicu pesan kesalahan sistem.

Ilustrasi gerakan robot dapat dilihat pada gambar 4.19, dimana gambar sebelah kiri adalah *init pose* dan sebelah kanan adalah ketika robot telah sampai pada titik *home*. Hasil dari pengujian dapat dilihat pada tabel 4.4, waktu eksekusi diperoleh dari *stopwatch* yang dinyalakan pada saat program dijalankan pada komputer dan dihentikan ketika komputer telah menyatakan program selesai. Pada tabel 4.5 dapat dilihat berapa lama waktu respon robot saat perintah dijalankan, kolom waktu respon diperoleh dari pengurangan nilai kolom waktu eksekusi di tabel 4.4 dengan empat detik sebagai waktu operasi pada program. Empat detik tersebut diperoleh dari satu detik untuk delay pada program kemudian tiga detik untuk operasi robot (lampiran 3 line kode nomor 19 dan 47). Kode program dapat dilihat pada Lampiran 3.



Gambar 4.19 Ilustrasi Gerak *point-to-point*

Tabel 4.4 Hasil pengujian poin-to-point robot fisik

Gerakan	Keberhasilan	Waktu Eksekusi
Init pose →Home	✓	4,97 detik
Init pose → B1	✓	4,35 detik
Init pose → B2	✓	4,79 detik
Init pose → B3	✓	4,75 detik
Init pose → B4	✓	4,88 detik

Tabel 4.5 Waktu Respon robot

Gerakan	Waktu Respon
Init pose →Home	0,97 detik
Init pose → B1	0,35 detik
Init pose → B2	0,79 detik
Init pose → B3	0,75 detik
Init pose → B4	0,88 detik

4.5.2 Pengujian Gerak Multi Point

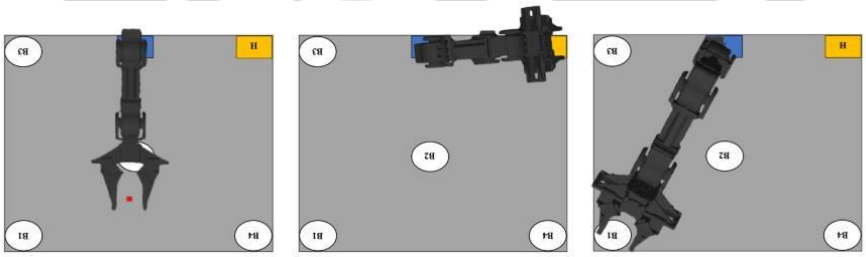
Sistem memulai rangkaian pengujian ini dengan menginisialisasi *node* pengendali yang terhubung ke topik lintasan standar antarmuka ROS 2. Kontroler pada topik ini bertugas menerima daftar koordinat dan melakukan perhitungan interpolasi menggunakan metode *Spline* agar transisi gerakan antar motor berjalan mulus.

Fungsi utama kemudian menyusun struktur pesan yang memuat tiga elemen vital yaitu penanda waktu aktual, daftar nama sendi yang terlibat, dan rangkaian titik tujuan. Kecerdasan sistem terlihat pada logika manipulasi waktu untuk menciptakan efek "jeda" tanpa mematikan motor. Sistem mengirimkan dua titik kembar dengan posisi yang sama persis namun memiliki cap waktu berbeda—misalnya titik A pada detik ke-3 dan titik A yang sama pada detik ke-6.

Kontroler merespons data "titik kembar" ini dengan mempertahankan torsi motor pada posisi tersebut selama interval waktu yang ditentukan, sehingga robot berhenti sejenak sebelum melanjutkan perjalanan ke titik B pada detik ke-9. Setelah

pesan terkirim, sistem membatalkan pemanggilan ulang fungsi dan menahan proses tetap aktif selama durasi penuh lintasan. Langkah ini menjamin program tidak berhenti di tengah jalan sebelum robot menyelesaikan seluruh manuvernya dengan sempurna.

Ilustrasi pergerakan robot akan ditampilkan pada gambar 4.20, dimana bagian sebelah kiri adalah init pose, tengah adalah ketika robot sampai pada koordinat titik Home dan sebelah kanan adalah ketika robot telah sampai pada koordinat titik B1. Hasil dari pengujian dapat dilihat pada tabel 4.6, pada tabel 4.7 dapat dilihat berapa lama waktu respon robot saat perintah dijalankan, kolom waktu respon diperoleh dari pengurangan nilai kolom waktu eksekusi di tabel 4.6 dengan sepuluh detik sebagai waktu operasi pada program. Sepuluh detik tersebut diperoleh dari satu detik untuk delay pada program kemudian sembilan detik untuk operasi robot (lampiran 4 line kode nomor 18 dan 56). Kode program dapat dilihat pada Lampiran 4.



Gambar 4.20 Ilustrasi gerak *multi-point*

Tabel 4.6 Hasil pengujian multi-point robot fisik

Gerakan	Keberhasilan	Waktu Eksekusi
Init pose → B1 – B2	✓	10,51 detik
Init pose → B3 – B2	✓	10,88 detik
Init pose → B2 – B4	✓	10,93 detik
Init pose → B1 – B3	✓	10,58 detik
Init pose → B4 – B3	✓	10,69 detik
Init pose → Home – B1	✓	10,93 detik

UNIVERSITAS MA CHUNG

Tabel 4.7 Waktu respon robot

Gerakan	Waktu Respon
Init pose → B1 – B2	0,51 detik
Init pose → B3 – B2	0,88 detik
Init pose → B2 – B4	0,93 detik
Init pose → B1 – B3	0,58 detik
Init pose → B4 – B3	0,69 detik
Init pose → Home – B1	0,93 detik

UNIVERSITAS
MA CHUNG

Bab V

Penutup

5.1 Kesimpulan

Implementasi sistem kendali robot OpenMANIPULATOR-X menggunakan arsitektur ROS 2 *Humble* telah berhasil dan berjalan secara stabil. Penggunaan *middleware* ROS 2 yang berbasis *Data Distribution Service* atau DDS meningkatkan fleksibilitas dan desentralisasi data sistem.

Pengujian memvalidasi akurasi gerakan robot dalam lingkungan simulasi Gazebo maupun pada perangkat keras fisik. Robot mampu mencapai target koordinat dengan presisi pada skenario gerak *point-to-point* dengan rata-rata waktu eksekusi 4,74 detik pada perangkat fisik. Validasi gerak *multi-point* juga menunjukkan hasil yang konsisten dengan rata-rata waktu penyelesaian lintasan sebesar 10,76 detik tanpa penyimpangan jalur.

Penerapan patch *Preempt_RT* pada *kernel* Linux menjadi faktor kunci dalam menjaga determinisme gerakan robot. Sesuai data penelitian Ye dkk. (2023), optimasi ini mereduksi latensi maksimum dari 6.243 µs menjadi 82 µs. Latensi tidak dapat di lihat karena penggunaan *PREEMPT_RT* sudah menekan latensi di bawah 10 µs.

5.2 Saran

Pusat Studi HMI sebaiknya mengintegrasikan sistem machine vision untuk mendukung deteksi objek secara otomatis dan dinamis. Pengembang selanjutnya perlu melakukan pengujian pada perangkat keras secara langsung tanpa melalui mesin virtual atau VMware. Langkah ini akan meminimalkan beban sistem atau overhead dan meningkatkan performa komputasi. Peneliti selanjutnya dapat mengembangkan variasi model gerak dari robot OpenManipulator supaya lebih bervariasi. Peneliti selanjutnya dapat juga membuat GUI sistem supaya lebih *user-friendly*.

UNIVERSITAS MA CHUNG

Daftar Pustaka

Adzeman, M. A. M., Zaman, M. H. M., Nasir, M. F., Ibrahim, M. F., & Mustaza, S. M. (2020). Kinematic Modeling of A Low Cost 4 DOF Robot Arm System. *International Journal of Emerging Trends in Engineering Research*, *8*(10), 6828–6834. https://doi.org/10.30534/ijeter/2020/328102020

Al Fajar, R., Lestari, A., & Teknologi Informasi, J. (2025). Analisis Perbandingan Sistem Operasi Windows 11 dan Linux Ubuntu Menggunakan Metode Studi Literatur (Studi Kasus: Kinerja Sistem, Keamanan dan Biaya). Dalam *Jurnal Bitwise ISSN xxxx-xxxx* (Vol. 1, Nomor 2). https://jurnal-bitwise.org/

Alif, M. (2025). *Pengendalian Gerakan Robot Openmanipulator Untuk Operasi Pemindahan Barang Berbasis MATLAB*.

Deng, G., Xu, G., Zhou, Y., Zhang, T., & Liu, Y. (2022). On the (In)Security of Secure ROS2. *Proceedings of the ACM Conference on Computer and Communications Security*, 739–753. https://doi.org/10.1145/3548606.3560681

Kelvin, D. (2024). *2024- Daniel Kelvin-Laporan Final(sudah kompre)*.

Kim, J., Mathur, D. C., Shin, K., & Taylor, S. (2023). *PAPRAS: Plug-And-Play Robotic Arm System*. http://arxiv.org/abs/2302.09655

Odun-Ayo, I., Okokpujie, K., Oputa, K., Ogbu, H., Emmanuel, F., Shofadekan, A., & Okuazun, G. (2021). Comparative Study of Operating System Quality Attributes. *IOP Conference Series: Materials Science and Engineering*, *1107*(1), 012061. https://doi.org/10.1088/1757-899x/1107/1/012061

Savola, A. (2021). *Antti Savola Server Virtualization with VMware*.

Teper, H., Unzel, M. G. ¨, Ueter, N., Von Der Brüggen, G., Brüggen, B., Chen, J.-J., & Günzel, M. (2022). *End-To-End Timing Analysis in ROS2 computer science 12 End-To-End Timing Analysis in ROS2*.

Ye, Y., Nie, Z., Liu, X., Xie, F., Li, Z., & Li, P. (2023). ROS2 Real-time Performance Optimization and Evaluation. *Chinese Journal of Mechanical Engineering (English Edition)*, *36*(1). https://doi.org/10.1186/s10033-023-00976-5

Zhong Ting, H., Hairi Mohd Zaman, M., Faisal Ibrahim, M., & Mohamed Moubark, A. (2021). Kinematic Analysis for Trajectory Planning of Open-Source 4-DoF

UNIVERSITAS MA CHUNG

Robot Arm. Dalam *IJACSA) International Journal of Advanced Computer Science and Applications* (Vol. 12, Nomor 6). www.ijacsa.thesai.org

UNIVERSITAS
MA CHUNG

Lampiran

Lampiran 1. Kode pengoperasian simulator Point-to-Point

move_b1.py

```
1. #!/usr/bin/env python3
2. import rclpy
3. from rclpy.node import Node
4. from rclpy.parameter import Parameter
5. from trajectory_msgs.msg import JointTrajectory, JointTrajectoryPoint
6. import time
7. import sys
8.
9. class MoveOpenManipulatorSinglePointGazebo(Node):
10.   def __init__(self):
11.       super().__init__('move_openmanipulator_single_point_gazebo')
12.
13.       # Mengaktifkan waktu simulasi (Sim Time) agar sinkron dengan
      Gazebo
14.       self.set_parameters([
15.           Parameter('use_sim_time', Parameter.Type.BOOL, True)
16.       ])
17.
18.       # Topik disesuaikan ke controller Gazebo
19.       self.publisher_ = self.create_publisher(
20.           JointTrajectory,
21.           '/arm_controller/joint_trajectory',
22.           10
23.       )
24.
25.       self.timer = self.create_timer(1.0, self.timer_callback)
26.       self.get_logger().info('Mode Gazebo Aktif. Siap mengirim 1 titik
      gerakan.')
27.
```

UNIVERSITAS MA CHUNG

```
    def timer_callback(self):
        self.timer.cancel()
        self.send_trajectory()

        self.get_logger().info("Perintah ke Gazebo dikirim.")

        # Waktu tunggu 4 detik (3 detik gerak + 1 detik buffer)
        time.sleep(4.0)

        self.destroy_node()
        sys.exit(0)

    def send_trajectory(self):
        traj = JointTrajectory()
        # Mengambil waktu dari Clock Simulasi
        traj.header.stamp = self.get_clock().now().to_msg()
        traj.joint_names = ['joint1', 'joint2', 'joint3', 'joint4']

        # --- TITIK TUJUAN TUNGGAL ---
        point = JointTrajectoryPoint()

        # Koordinat tujuan (Posisi B1)
        point.positions = [-0.5, 0.23, 0.15, 0.22]

        # Durasi gerakan ditetapkan 3 detik
        point.time_from_start.sec = 3

        # Memasukkan satu titik saja ke dalam daftar
        traj.points = [point]

        self.publisher_.publish(traj)

```

UNIVERSITAS MA CHUNG

```
60. def main(args=None):
61.     rclpy.init(args=args)
62.     node = MoveOpenManipulatorSinglePointGazebo()
63.     try:
64.         rclpy.spin(node)
65.     except SystemExit:
66.         rclpy.logging.get_logger("root").info("Program  selesai.")
67.     rclpy.shutdown()
68.
69. if __name__== '__main__':
70.     main()
```

Lampiran 2. Kode pengoperasian simulator Multi Points

move_b1b2.py

```
1.  #!/usr/bin/env  python3
2.  import rclpy
3.  from rclpy.node import Node
4.  from rclpy.parameter import Parameter
5.  from trajectory_msgs.msg import JointTrajectory, JointTrajectoryPoint
6.  import time
7.  import sys
8.
9.  class MoveOpenManipulatorPause(Node):
10.     def __init__(self):
11.         super().__init__('move_openmanipulator_pause')
12.
13.         # Mengaktifkan waktu simulasi (Sim Time) agar sinkron dengan
    Gazebo
14.         self.set_parameters([
15.             Parameter('use_sim_time', Parameter.Type.BOOL, True)
16.         ])
17.
```

UNIVERSITAS MA CHUNG

```
        self.publisher_ = self.create_publisher(
            JointTrajectory,
            '/arm_controller/joint_trajectory',
            10
        )

        self.timer = self.create_timer(1.0, self.timer_callback)
        self.get_logger().info('Mode Gazebo Aktif. Menunggu clock simulasi...')

    def timer_callback(self):
        self.timer.cancel()
        self.send_trajectory()

        self.get_logger().info("Perintah ke Gazebo dikirim.")

        # waktu tunggu 10 detik
        time.sleep(10.0)

        self.destroy_node()
        sys.exit(0)

    def send_trajectory(self):
        traj = JointTrajectory()
        # Mengambil waktu dari Clock Simulasi (bukan waktu komputer)
        traj.header.stamp = self.get_clock().now().to_msg()
        traj.joint_names = ['joint1', 'joint2', 'joint3', 'joint4']

        # --- DEFINISI WAKTU ---
        move_time = 3
        pause_time = 3
```

UNIVERSITAS MA CHUNG

```
        # --- TITIK 1: Bergerak ke Posisi B1 ---
        point1 = JointTrajectoryPoint()
        point1.positions = [-0.5, 0.23, 0.15, 0.22]
        point1.time_from_start.sec = move_time

        # --- TITIK 1 (HOLD): Diam di Posisi B1 ---
        point1_hold = JointTrajectoryPoint()
        point1_hold.positions = [-0.5, 0.23, 0.15, 0.22]
        point1_hold.time_from_start.sec = move_time + pause_time

        # --- TITIK 2: Bergerak ke Posisi B2 ---
        point2 = JointTrajectoryPoint()
        point2.positions = [-0.02, -0.75, 0.71, 1.03]
        point2.time_from_start.sec = move_time + pause_time + move_time

        traj.points = [point1, point1_hold, point2]

        self.publisher_.publish(traj)

def main(args=None):
    rclpy.init(args=args)
    node = MoveOpenManipulatorPause()
    try:
        rclpy.spin(node)
    except SystemExit:
        rclpy.logging.get_logger("root").info("Program  selesai.")
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

UNIVERSITAS MA CHUNG

Lampiran 3. Kode pengoperasian robot fisik Point-to-Point

Move_to_H.py

```
1.  #!/usr/bin/env python3
2.  import rclpy
3.  from rclpy.node import Node
4.  from trajectory_msgs.msg import JointTrajectory, JointTrajectoryPoint
5.  import time
6.  import sys
7.
8.  class MoveOpenManipulatorSinglePoint(Node):
9.      def __init__(self):
10.         super().__init__('move_openmanipulator_single_point')
11.
12.         # Publisher ke topik controller robot asli
13.         self.publisher_ = self.create_publisher(
14.             JointTrajectory,
15.             '/arm_controller/joint_trajectory',
16.             10
17.         )
18.
19.         self.timer = self.create_timer(1.0, self.timer_callback)
20.         self.get_logger().info('Robot Terhubung. Siap mengirim 1 titik
    gerakan.')
21.
22.     def timer_callback(self):
23.         self.timer.cancel()
24.         self.send_trajectory()
25.
26.         # Waktu tunggu 4 detik agar skrip tidak mati sebelum robot selesai
    bergerak
```

UNIVERSITAS MA CHUNG

```
        self.get_logger().info("Perintah dikirim. Robot bergerak ke Titik Tujuan.")
        time.sleep(4.0)

        self.destroy_node()
        sys.exit(0)

    def send_trajectory(self):
        traj = JointTrajectory()
        traj.header.stamp = self.get_clock().now().to_msg()

        # Pastikan nama joint sesuai dengan konfigurasi robot
        traj.joint_names = ['joint1', 'joint2', 'joint3', 'joint4']

        # --- TITIK TUJUAN TUNGGAL ---
        point = JointTrajectoryPoint()

        # Masukkan koordinat tujuan
        point.positions = [1.65, -0.89, 0.82, 1.03]

        # Robot akan sampai di titik ini dalam waktu 3 detik
        point.time_from_start.sec = 3

        traj.points = [point]

        self.publisher_.publish(traj)

def main(args=None):
    rclpy.init(args=args)
    node = MoveOpenManipulatorSinglePoint()
    try:
        rclpy.spin(node)
```

UNIVERSITAS MA CHUNG

```
    except SystemExit:
        # Menangkap sinyal keluar dari sys.exit(0) agar tidak error
        rclpy.logging.get_logger("root").info("Program  selesai.")
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Lampiran 4. Kode pengoperasian robot fisik Multi Points

multi_point_hb1.py

```
#!/usr/bin/env  python3
import rclpy
from rclpy.node import Node
from trajectory_msgs.msg import JointTrajectory, JointTrajectoryPoint
import time
import sys

class MoveOpenManipulatorPause(Node):
    def __init__(self):
        super().__init__('move_openmanipulator_pause')

        self.publisher_ = self.create_publisher(
            JointTrajectory,
            '/arm_controller/joint_trajectory',
            10
        )

        self.timer = self.create_timer(1.0, self.timer_callback)
        self.get_logger().info('Robot Terhubung. Siap mengirim lintasan dengan jeda...')

    def timer_callback(self):
```

UNIVERSITAS MA CHUNG

```
22.         self.timer.cancel()
23.         self.send_trajectory()
24.
25.         # Waktu tunggu 10 detik agar skrip tidak mati sebelum robot selesai
    bergerak
26.         self.get_logger().info("Perintah dikirim. Robot bergerak: Titik 1 ->
    Jeda -> Titik 2.")
27.         time.sleep(10.0)
28.
29.         self.destroy_node()
30.         sys.exit(0)
31.
32.     def send_trajectory(self):
33.         traj = JointTrajectory()
34.         traj.header.stamp = self.get_clock().now().to_msg()
35.         traj.joint_names = ['joint1', 'joint2', 'joint3', 'joint4']
36.
37.         # --- DEFINISI WAKTU ---
38.         move_time = 3  # Waktu untuk bergerak
39.         pause_time = 3 # Durasi jeda (diam)
40.
41.         # --- TITIK 1: Bergerak ke Posisi Home ---
42.         point1 = JointTrajectoryPoint()
43.         point1.positions = [1.65, -0.89, 0.82, 1.03]
44.         point1.time_from_start.sec = move_time # T=3
45.
46.         # --- TITIK 1 (HOLD): Diam di Posisi Home ---
47.         # Kita kirim posisi yang SAMA persis, tapi waktunya ditambah durasi
    jeda
48.         point1_hold = JointTrajectoryPoint()
49.         point1_hold.positions = [1.65, -0.89, 0.82, 1.03] # Posisi sama dengan
    point1
```

UNIVERSITAS MA CHUNG

```
        point1_hold.time_from_start.sec = move_time + pause_time # T=6 (3+3)

        # --- TITIK 2: Bergerak ke Posisi B1 ---
        point2 = JointTrajectoryPoint()
        point2.positions = [-0.5, 0.23, 0.15, 0.22]
        # Waktu tempuh dihitung dari akhir jeda
        point2.time_from_start.sec = move_time + pause_time + move_time # T=9 (6+3)

        # Masukkan urutan titik ke dalam list
        # Urutan: Gerak ke Home -> Diam di Home -> Gerak ke B1
        traj.points = [point1, point1_hold, point2]

        self.publisher_.publish(traj)

def main(args=None):
    rclpy.init(args=args)
    node = MoveOpenManipulatorPause()
    try:
        rclpy.spin(node)
    except SystemExit:
        # Menangkap sinyal keluar dari sys.exit(0) agar tidak error
        rclpy.logging.get_logger("root").info("Program  selesai.")
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

UNIVERSITAS MA CHUNG