PROPOSAL TUGAS AKHIR

Simulator Pembelajaran Pergerakan Robot: Titik ke Titik dan Pelacakan Trayek pada Differential Drive dan Omni-Directional Robot

Penyusun:

Dimas Fajar Pranaya (20524056)

Rifqi Zul Fahmi (20524158)

Program Studi Teknik Elektro

Fakultas Teknologi Industri

Universitas Islam Indonesia

Yogyakarta

2024

HALAMAN PENGESAHAN

Simulator Pembelajaran Pergerakan Robot: Titik ke Titik dan Pelacakan Trayek pada Differential Drive dan Omni-Directional Robot

Penyusun:
Dimas Fajar Pranaya (20524056)
Rifqi Zul Fahmi (20524158)

Yogyakarta, 08 Juli 2024

Ir. R.M. Sisdarmanto Adinandra S.T, M.Sc, Ph.D
NIK. 025240101

Program Studi Teknik Elektro
Fakultas Teknologi Industri
Universitas Islam Indonesia
Yogyakarta
2024

HALAMAN VERIFIKASI TA

Simulator Pembelajaran Pergerakan Robot: Titik ke Titik dan Pelacakan Trayek pada Differential Drive dan Omni-Directional Robot

VERIFIKASI TA201
BAB 1 : PENDAHULUAN **BAB 2 : IDENTIFIKASI KEBUTUHAN SISTEM** **BAB 3 : USULAN SOLUSI**
Dosen Pembimbing 1  **Ir. R.M. Sisdarmanto Adinandra S.T, M.Sc, Ph.D** **NIK. 025240101**

VERIFIKASI TA201
BAB 4 : HASIL RANCANGAN DAN METODE PENGUKURAN **BAB 5 : HASIL PENGUKURAN DAN ANALISIS** **BAB 6 : KESIMPULAN DAN SARAN**
Dosen Pembimbing 1  **Ir. R.M. Sisdarmanto Adinandra S.T, M.Sc, Ph.D** **NIK. 025240101**

LEMBAR PENGESAHAN TUGAS AKHIR

Simulator Pembelajaran Pergerakan Robot: Titik ke Titik dan Pelacakan Trayek pada Differential Drive dan Omni-Directional Robot

Disusun oleh:

Dimas Fajar Pranaya 20524056

Rifqi Zul Fahmi 20524158

Telah dipertahankan di depan dewan penguji
pada tanggal: 31 Juli 2024

Susunan Dewan Penguji

Ketua Penguji : **Ir. R.M. Sisdarmanto Adinandra S.T, M.Sc. Ph.D** _____

Anggota Penguji 1 : **Dwi Ana Ratna Wati, S.T., M.Eng.** _____

Anggota Penguji 2 : **Alim Safari, S.T.** _____

Tugas akhir ini telah disahkan sebagai salah satu persyaratan
untuk memperoleh gelar Sarjana Teknik

Tanggal: 4 Agustus 2024
Ketua Program Studi Teknik Elektro

Dwi Ana Ratna Wati, S.T., M.Eng.
035240102

PERNYATAAN

Dengan ini kami menyatakan bahwa:

1. Tugas Akhir ini tidak mengandung karya yang diajukan untuk memperoleh gelar kesarjanaan di suatu perguruan tinggi lainnya, dan sepanjang pengetahuan kami juga tidak mengandung karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis diacu dalam naskah ini dan disebutkan dalam daftar pustaka.
2. Informasi dan materi Tugas Akhir yang terkait hak milik, hak intelektual, dan paten merupakan milik bersama antara tiga pihak, yaitu penulis, dosen pembimbing, dan Universitas Islam Indonesia. Dalam hal ini, penggunaan informasi dan materi Tugas Akhir terkait paten maka akan didiskusikan lebih lanjut untuk mendapatkan persetujuan dari ketiga pihak tersebut di atas.

Yogyakarta, 4 Agustus 2024

Dimas Fajar Pranava (20524056)

Rifqi Zul Fahmi (20524158)

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

RINGKASAN

Dalam laporan tugas akhir ini, fokus utama adalah pada pengembangan dan implementasi robot beroda yang mampu memberikan pembelajaran mengenai pergerakan robot beroda. Robot yang digunakan menggunakan sistem differential drive dan omni-directional 3 roda dengan fokus pada kemampuan mobilitas yang memungkinkan mereka untuk melakukan berbagai pergerakan.

Metode pengukuran kinerja robot dilakukan melalui serangkaian uji coba dalam area lapangan berukuran 3x2 meter, dengan parameter seperti jarak, rotasi, dan orientasi robot diukur. Hasil uji coba menunjukkan bahwa robot yang dikembangkan mampu menavigasi cukup akurat dan efisien. Hal ini berdasarkan pengukuran manual menggunakan meteran yang disesuaikan dengan pembacaan sistem.

Hasil pengukuran robot, juga diverifikasi dengan menggunakan teknologi marker ArUco dan kamera eksternal, menunjukkan akurasi dan presisi yang sesuai dengan nilai aktual. Namun, tujuan yang belum tercapai adalah pengembangan antarmuka pengguna untuk memudahkan kontrol dan pemantauan robot serta visualisasi data hasil pengukuran. Langkah selanjutnya termasuk pengujian lebih lanjut terhadap berbagai jenis pergerakan dan fungsi robot untuk memastikan kesesuaian dengan spesifikasi yang diusulkan, sehingga sistem dapat memberikan solusi yang handal sesuai dengan kebutuhan aplikasi yang dituju.

BAB 1. PENDAHULUAN

1.1 Latar belakang dan Identifikasi Masalah

Perkembangan industri dalam beberapa tahun terakhir telah menghadirkan revolusi besar dalam dunia teknologi dan interaksi manusia dengan lingkungannya. Salah satu aspek penting dalam revolusi ini adalah penggunaan robot, yang memiliki peran kunci dalam mengubah cara kita berinteraksi dengan teknologi. Perkembangan tersebut terlihat di negara-negara di mana robot sudah menjadi bagian dari setiap pekerjaan yang mereka lakukan. Berdasarkan data yang dikumpulkan oleh International Federation of Robotics, negara yang paling maju teknologi otomatisasinya adalah Jepang, Republik Korea, dan Jerman. [1]

Robot, khususnya robot beroda, telah memainkan peran penting dalam memperluas batas-batas kemungkinan teknologi ini. Salah satu kegunaan utama robot beroda di dunia industri adalah meningkatkan efisiensi produksi. Dengan kemampuan mobilitasnya, robot beroda dapat digunakan untuk melakukan berbagai tugas seperti pemindahan barang berat, pemantauan lingkungan, dan manajemen barang. Mereka juga dapat bekerja dalam lingkungan yang berbahaya atau tidak ramah bagi manusia, seperti lingkungan beracun atau berpotensi meledak. Dengan demikian, robot beroda telah membantu mengubah lanskap industri modern dengan membuka peluang baru dalam otomatisasi dan menghasilkan manfaat besar dalam meningkatkan efisiensi, keamanan, dan kualitas dalam berbagai sektor industri. [2]

Jenis robot beroda sangat beragam, tergantung pada desain roda yang mereka gunakan dan cara mereka bergerak. Robot beroda dapat dibagi menjadi beberapa kategori berdasarkan jenis roda dan pergerakannya. Dua kategori yang umum dan banyak digunakan dalam berbagai aplikasi adalah robot beroda differential drive dan omni-directional. Robot differential drive memiliki dua roda yang dapat berputar secara independen di sisi kiri dan kanan. Jenis pergerakan ini memungkinkan robot untuk melakukan manuver dengan mengatur kecepatan roda kiri dan kanan secara terpisah, seperti berbelok dengan memperlambat satu roda atau bergerak mundur dengan menggerakkan roda berlawanan arah. Sedangkan robot omni-directional menggunakan roda khusus yang memungkinkan mereka bergerak dalam berbagai arah tanpa perlu berputar. Mereka

dapat berjalan lurus, berbelok, atau bahkan bergerak diagonal dengan mudah tanpa mengubah orientasi fisik robot.[3]

Selain dua jenis robot beroda yang telah disebutkan sebelumnya, yaitu robot beroda *differential drive* dan omni-directional, memiliki dua tipe pergerakan yang umum digunakan dalam mengendalikan robot beroda, yaitu pergerakan titik ke titik dan pelacakan trayektori titik ke titik dan *pelacakan trayek*. Pergerakan titik ke titik digunakan ketika robot perlu berpindah dari satu titik ke titik lain dengan presisi. Robot dengan tipe gerak ini menerima koordinat atau titik tujuan dan menghitung jalur terpendek untuk mencapai tujuan tersebut. Di sisi lain, pergerakan pelacakan trayek digunakan ketika robot harus mengikuti jalur dengan waktu tertentu yang sebelumnya telah ditentukan.[4] pelacakan trayek memerlukan kontrol yang sangat akurat terhadap posisi dan orientasi robot, sering kali menggunakan pengendalian PID atau pengendalian adaptif untuk mencapai presisi yang dibutuhkan. Dua tipe pergerakan ini memiliki peran krusial dalam berbagai aplikasi robotika, disesuaikan dengan kebutuhan spesifik dan lingkungan kerja robot tersebut.[5]

Dalam pengembangan robot beroda, terdapat permasalahan umum terkait pemahaman mendalam tentang bagaimana robot dapat bergerak secara efisien dan tepat sasaran. Jenis roda dan tipe pergerakan yang berbeda, seperti robot differential drive dan omni-directional, memiliki karakteristik yang berbeda dan memerlukan pendekatan pengendalian yang berbeda pula, terutama dalam konteks pergerakan titik ke titik dan pelacakan trayek. Namun, pemahaman yang baik tentang cara mengimplementasikan kontrol untuk jenis roda dan tipe pergerakan ini seringkali sulit diperoleh tanpa adanya simulasi.[6] Hal ini menjadi salah satu kendala dalam pemahaman dan pengembangan yang melibatkan pergerakan robot beroda. Kendala tersebut dapat dirasakan oleh peneliti dan pengembang di bidang robotika, salah satunya adalah mahasiswa FTI UII yang memiliki minat atau mata kuliah yang berhubungan dengan robotika.

Masalah yang dihadapi oleh mahasiswa tersebut adalah ketidaktersediaan perangkat keras yang memadai untuk pengujian nyata di bidang robotika. Selain itu, pengujian pada robot fisik memerlukan biaya yang cukup tinggi. Oleh karena itu, diperlukan pengembangan suatu sistem yang terdiri dari software simulasi, serta simulasi yang dilakukan juga oleh robot beroda (perangkat keras) secara langsung.[7] Sistem ini diharapkan dapat digunakan sebagai sarana

pembelajaran, khususnya dalam menguji dan memvalidasi konsep pergerakan robot tipe differential drive dan omni-directional.

Penyelesaian masalah ini memiliki signifikansi penting dalam konteks pengembangan dan pemahaman mengenai robot beroda. Melalui pengembangan perangkat lunak simulasi yang memungkinkan pengujian dua jenis robot beroda dengan masing-masing jenis robot dapat melakukan dua tipe gerakan, yaitu titik ke titik dan pelacakan trayek secara virtual dan langsung, mahasiswa yang dimaksud akan memiliki akses yang lebih mudah untuk menguji ide-ide mereka mengenai pergerakan robot tipe differential drive dan omni-directional tanpa harus menghadapi biaya tinggi terkait perangkat keras robot fisik yang mahal. Selain itu, simulasi ini juga memungkinkan pengguna untuk mendalami pemahaman tentang perbedaan karakteristik robot tersebut. Sehingga, pembuatan sistem ini diharapkan akan menjadi pondasi yang esensial bagi mahasiswa FTI UII yang memiliki minat atau mata kuliah terkait robotika untuk menjelajahi dunia robotika secara lebih mendalam dan turut berkontribusi dalam perkembangan teknologi masa depan.

1.2 Rumusan Masalah

Masalah yang dihadapi adalah bagaimana menciptakan sistem simulasi virtual dan langsung (robot asli) yang memfasilitasi mahasiswa FTI UII dalam memahami pergerakan robot beroda differential drive dan omni-directional, serta platform web yang memungkinkan pengaturan titik tujuan robot secara visual.

1.3 Tujuan

Tujuan dari proyek ini adalah:

1. Mengembangkan sebuah simulator yang memungkinkan pengguna untuk menguji dan memahami pergerakan titik ke titik dan pelacakan trayek pada differential drive dan omni directional robot dengan mudah dan efektif.
2. Mengembangkan platform berupa web yang dapat mengintegrasikan teknologi pattern agar pengguna dapat mengatur titik tujuan robot secara visual dan dapat menampilkan dokumentasi hasil simulasi pergerakan robot.

3. Membantu pengguna memperoleh pemahaman yang lebih baik tentang konsep pergerakan robot beroda melalui simulasi, serta kemampuan untuk menganalisis melalui dokumentasi yang diberikan.

1.4 Batasan Masalah

Dalam Tugas Akhir ini, terdapat beberapa batasan yang perlu diperhatikan diantaranya:

1. Sistem simulasi yang dikembangkan akan terbatas pada dua jenis robot beroda, yaitu differential drive dan omni-directional, serta dua tipe pergerakan robot, yaitu titik ke titik dan pelacakan trayek. Jenis robot beroda dan tipe pergerakan lainnya tidak dimasukkan.
2. Jumlah robot yang diakomodasi dalam simulasi ini akan mencakup 5 robot differential drive dan 1 robot omni-directional 3 roda.
3. Simulasi ini akan membatasi jumlah maksimal robot yang dapat dioperasikan secara bersamaan dalam satu sesi simulasi. Jumlah maksimal robot yang diizinkan adalah sebanyak 3 robot differential drive dan 1 robot omni-directional 3 roda dalam satu sesi simulasi.
4. Simulasi ini akan dibatasi pada lingkungan dengan ukuran lapangan tetap sebesar 4 x 3 meter, dan ukuran lapangan selain itu tidak akan dipertimbangkan.

1.5 Batasan Realistis Aspek Keteknikan

Dalam Tugas Akhir ini, terdapat pula beberapa batasan dari aspek keteknikan yang perlu diperhatikan diantaranya:

1. Durasi simulasi dan kecepatan waktu dapat dibatasi. Misalnya, simulasi akan berjalan dalam waktu percepatan tertentu atau memiliki batasan waktu maksimum untuk setiap simulasi.
2. Konektivitas yang digunakan antara robot dan simulator adalah konektivitas Wifi. Koneksi Wifi akan digunakan sebagai metode komunikasi utama dalam simulasi secara langsung.
3. Standar ketersediaan Wifi (802.11e) mendukung QoS (Quality of Service) dalam komunikasi nirkabel yang memungkinkan prioritas terhadap jenis data tertentu, seperti perintah pengguna atau informasi sensor, untuk memastikan transmisi yang konsisten dan dapat diandalkan.

4. Standar kinerja Wifi (802.11ac) untuk mencapai kinerja yang optimal dalam mentransfer data dan dapat menjamin kecepatan transmisi yang memadai untuk mendukung komunikasi real-time antara robot dan simulator.
5. Standar kompatibilitas peralatan lama wifi (802.11n) tetap kompatibel dengan standar untuk mendukung peralatan lama.

BAB 2. IDENTIFIKASI KEBUTUHAN SISTEM

2.1 Studi Literatur dan Observasi

Perlu dilakukan studi literatur untuk memahami proyek tugas akhir ini dengan lebih mendalam. Terdapat beberapa proyek/penelitian sejenis yang telah dilaksanakan sebelumnya, yang dapat menjadi landasan bagi proyek ini. Berikut adalah ringkasan temuan-temuan dari literatur yang menjadi dasar pemahaman dan konsep untuk proyek tugas akhir ini, diuraikan dalam tabel 2.1 berikut.

Tabel 2. 1 Hasil studi literatur solusi sejenis

Judul	Usulan solusi	Hasil/Evaluasi (Kelebihan/Kekurangan)
Design and Control for Differential Drive Mobile Robot	Penelitian berfokus pada desain mekanik dan sistem pengendalian robot differential drive untuk pengangkutan material. Tujuannya adalah mengembangkan platform robot yang dapat dengan efisien menavigasi dan mengangkut material dalam berbagai arah.	**Hasil:** Penelitian ini mencakup pemodelan matematis mengenai kinematika robot dan dinamika motor DC, hasil simulasi yang menggambarkan hubungan antara kecepatan motor dan kecepatan linear robot, serta perilaku robot dalam berbagai skenario pergerakan. Selain itu, penelitian juga mengidentifikasi karakteristik pergerakan seperti posisi, kecepatan, dan radius putar robot. Pengujian dan simulasi model Simulink yang dikembangkan menggunakan motor DC 12V dengan parameter tertentu juga dilakukan, termasuk penentuan kecepatan sudut robot berdasarkan kecepatan putaran setiap roda. **Kelebihan:** Penelitian ini mencakup pemodelan matematis mengenai kinematika robot dan dinamika motor DC, yang memberikan dasar untuk desain dan optimalisasi sistem pengendalian. Hasil simulasi menunjukkan hubungan antara kecepatan motor dan kecepatan linear robot, memungkinkan pengendalian dan koordinasi yang lebih baik terhadap pergerakan robot. Selain itu, penelitian mengidentifikasi

		karakteristik pergerakan seperti posisi, kecepatan, dan radius putar robot, memudahkan pemahaman dan analisis perilaku robot. **Kekurangan:** Kelemahan penelitian ini mencakup keterbatasan dalam lingkupnya, yang hanya berfokus pada desain mekanik dan sistem pengendalian robot differential drive untuk pengangkutan material, sehingga tidak mengatasi aspek lain dari robotika mobile atau aplikasi khusus. Selain itu, penelitian ini mengandalkan asumsi yang disederhanakan dalam pemodelan matematis dan simulasi, yang mungkin tidak sepenuhnya mencakup kompleksitas skenario dunia nyata.
Kinematics Control of an Omnidirectional Mobile Robot	Solusi yang diusulkan dalam penelitian ini adalah pengendali kinematika berdasarkan linearisasi umpan balik untuk mencapai stabilisasi titik dan pelacakan lintasan pada robot mobile beroda omni-direksional. Metode pengendalian ini menggunakan model kinematika robot dan mengimplementasikan pengendali PI dengan matriks gain	**Hasil:** Hasil dari penelitian ini menunjukkan bahwa pengendali kinematika yang diusulkan berdasarkan linearisasi umpan balik efektif dalam mencapai stabilisasi titik dan pelacakan lintasan pada robot mobile beroda omni-direksional. Simulasi dan hasil eksperimen menunjukkan kelayakan dan efektivitas metode pengendalian. Pengendali kinematika PI mampu memenuhi persyaratan regulasi yang diinginkan dan pelacakan lintasan. Metode pengendalian ini mampu mengarahkan robot mobile beroda omni-direksional untuk bergerak sepanjang lintasan yang diinginkan, seperti lintasan lingkaran dan elips. Metode pengendalian yang diusulkan menunjukkan potensi untuk aplikasi di masa depan dalam pengendali pelacakan dinamis. **Kelebihan:** Keunggulan hasil penelitian ini meliputi peningkatan kinerja pengendalian, terutama dalam stabilisasi titik dan pelacakan lintasan robot beroda omni-direksional, menunjukkan kelayakan serta efektivitas metode pengendalian yang praktis untuk implementasi di

	diagonal. Simulasi dan hasil eksperimen dilakukan untuk menunjukkan kelayakan dan efektivitas metode pengendalian yang diusulkan.	dunia nyata, serta fleksibilitas dalam menggerakkan robot sesuai berbagai lintasan termasuk lingkaran dan elips. Lebih lanjut, penelitian membuka peluang aplikasi di masa depan dalam pengendali pelacakan dinamis, menandakan potensi pengembangan metode pengendalian yang lebih kompleks untuk memperluas kapabilitas robot beroda omni-direksional. **Kekurangan:** Kekurangan penelitian tidak disebutkan secara langsung maupun tidak langsung dari jurnal tersebut.
Omni-Directional Mobile Robot Controller Design by Trajectory Linearization	Solusi yang diberikan melalui penelitian ini adalah trajectory linearization control (TLC) untuk robot mobile omni-direksional. Pengendali TLC mampu dengan akurat melacak berbagai lintasan dengan menyetel gain pengendali. Ia dapat secara efektif mengurai dan menstabilkan pergerakan robot, memungkinkannya untuk mengikuti lintasan translasi dan rotasi secara	**Hasil:** Pengendali TLC mampu dengan akurat melacak berbagai jenis lintasan dengan penyetelan gain pengendali, menghasilkan pelacakan lintasan yang presisi. Pengendali ini efektif dalam mengurai dan menstabilkan pergerakan robot, memungkinkan robot untuk mengikuti lintasan translasi dan rotasi secara bersamaan. Pengendali TLC juga mampu mengikuti lintasan-lintasan yang beragam, termasuk titik set, lintasan percepatan, dan lintasan lingkaran, dengan tingkat kesalahan pelacakan yang sangat rendah. Selain itu, pengendali ini menunjukkan stabilitas yang kuat, memudahkan penyetelan gain pengendali. Dengan demikian, pengendali TLC menawarkan solusi yang efektif dan presisi untuk mengendalikan robot mobile dalam berbagai situasi dan lintasan yang berbeda. **Kelebihan:** Keunggulan dari hasil penelitian mencakup pelacakan lintasan yang akurat, dengan trajectory linearization control (TLC) yang dapat mengikuti berbagai lintasan, yang bermanfaat untuk aplikasi yang membutuhkan presisi, serta penguraian dan penstabilan pergerakan yang efektif, memungkinkan robot untuk mengikuti lintasan translasi dan rotasi secara bersamaan,

	bersamaan. Stabilitas yang kuat dari pengendali TLC membuatnya mudah untuk menyetel gain-gain dan mencapai pelacakan lintasan yang akurat.	menghasilkan gerakan yang fleksibel dan gesit. Selain itu, pengendali ini menunjukkan stabilitas yang kuat, memudahkan penyetelan, dan dapat diterapkan dalam berbagai jenis lintasan, seperti titik set, percepatan, dan lintasan lingkaran, menawarkan fleksibilitas untuk berbagai aplikasi dan skenario. **Kekurangan:** Hasil penelitian didasarkan pada pengujian simulasi. Meskipun pengujian simulasi memberikan wawasan berharga dan validasi awal, tetapi tidak sepenuhnya menggambarkan kompleksitas dan ketidakpastian lingkungan dunia nyata. Oleh karena itu, pengujian dan implementasi lebih lanjut pada robot mobile omni-direksional nyata diperlukan untuk memvalidasi efektivitas dan ketangguhan trajectory linearization control (TLC) dalam aplikasi praktis.
Modeling, simulation and control of a differential steering type mobile robot	Solusi yang diusulkan dalam penelitian ini adalah menciptakan model simulasi dan sistem pengendalian untuk robot mobil dengan penggerak differential steering. Peneliti menggunakan perangkat lunak seperti CATIA, MSC ADAMS, dan MATLAB. Algoritma pengendalian yang	**Hasil:** Simulasi dilakukan dengan menggunakan perencana lintasan untuk menentukan jalur dan waktu bagi robot untuk mencapai posisi akhir yang diinginkan. Hasil simulasi meliputi lintasan pusat massa, posisi akhir, kesalahan posisi, aksi pengendalian yang dilakukan, dan torsi yang diperlukan untuk pergerakan. Makalah ini juga membahas ketahanan sistem ketika menghadapi sinyal noise, yang menunjukkan bahwa kesalahan posisi meningkat secara signifikan dengan adanya noise. Secara keseluruhan, makalah ini menyimpulkan bahwa model simulasi yang dibuat di MSC ADAMS dapat diterapkan pada konfigurasi robot bergerak apa pun, memungkinkan analisis dan simulasi menyeluruh sebelum proses manufaktur. Namun, disarankan untuk menerapkan jenis pengontrol lain untuk meningkatkan

	diimplementasikan adalah pengendalian PD dengan umpan balik kecepatan, yang didasarkan pada model dinamis robot dengan banyak derajat kebebasan.	ketahanan sistem dalam menghadapi kebisingan dan ketidakpastian. **Kelebihan:** Kelebihan hasil simulasi ini mencakup analisis mendalam yang memungkinkan penilaian menyeluruh terhadap kinerja robot, kemampuan skalabilitas yang memungkinkan aplikasi model simulasi pada berbagai konfigurasi robot, penggunaan perangkat lunak CATIA yang meningkatkan visualisasi dan pemahaman komponen robot, serta keberagaman aplikasi ko-simulasi dengan MSC ADAMS/MATLAB yang memungkinkan pengujian berbagai strategi pengendalian. **Kekurangan:** Kelemahan yang perlu diperhatikan, seperti ketidakrobustan terhadap gangguan suara yang dapat memengaruhi akurasi hasil, kurangnya perbandingan langsung dengan model atau strategi pengendalian yang sudah ada sehingga sulit menilai kebaruan metode yang diusulkan, dan penggunaan asumsi yang sederhana yang mungkin tidak mencakup seluruh dinamika situasi dunia nyata
Real Robot Remote Control Versus Simulations	Peneliti berfokus pada penggunaan simulasi dan eksperimen jarak jauh dengan robot beroda untuk melatih siswa di bidang robot bergerak. Dan juga berfokus pada penggunaan model robot linier yang	**Hasil:** Hasil dari simulasi lebih menyoroti manfaat penggunaan simulasi dan eksperimen jarak jauh dengan robot beroda untuk melatih siswa di bidang robot bergerak dan juga menekankan bahwa penggunaan teknologi pendidikan jarak jauh, seperti laboratorium jarak jauh dan eksperimen berbasis internet, dapat mengatasi keterbatasan biaya dan aksesibilitas robot sungguhan. Hal ini menunjukkan bahwa menggabungkan simulasi dengan eksperimen robot nyata memungkinkan siswa membandingkan hasil simulasi dengan perilaku robot nyata, sehingga meningkatkan pemahaman dan pembelajaran

	disederhanakan untuk desain kontrol, termasuk desain pengontrol PID. Tujuannya adalah untuk memberikan siswa pengalaman praktis dan kemampuan membandingkan hasil simulasi dengan eksperimen robot nyata. Penggunaan teknologi pendidikan jarak jauh, seperti laboratorium jarak jauh dan eksperimen berbasis internet, disorot sebagai cara untuk mengatasi keterbatasan biaya dan aksesibilitas robot sungguhan.	mereka. Artikel tersebut juga menyebutkan bahwa eksperimen jarak jauh dengan robot beroda TOM telah diadaptasi untuk memberikan akses belajar mandiri bagi siswa melalui internet. **Kelebihan:** Kelebihan penggunaan eksperimen jarak jauh dengan robot beroda untuk melatih mahasiswa dalam bidang robot mobile mencakup aksesibilitas yang memungkinkan mahasiswa untuk mengakses dan mengendalikan robot sungguhan melalui internet, mengatasi keterbatasan akses fisik ke peralatan mahal. Interaksi waktu nyata memungkinkan mahasiswa berinteraksi dengan robot dalam waktu nyata, menerima umpan balik segera, dan mengamati respons yang kohesif, yang meningkatkan minat dan motivasi mereka. Pengalaman praktis yang diberikan oleh eksperimen jarak jauh memberikan mahasiswa pengalaman praktis dalam menghadapi situasi nyata dan perilaku non-linear robot, yang tidak dapat sepenuhnya direplikasi dalam simulasi. Selain itu, mahasiswa dapat membandingkan hasil simulasi dengan perilaku robot sungguhan, membantu mereka memahami keterbatasan dan perbedaan antara model matematika dan sistem sesungguhnya. **Kekurangan:** Kekurangan penggunaan eksperimen jarak jauh dengan robot beroda untuk melatih mahasiswa dalam bidang robot mobile mencakup kurangnya interaksi fisik yang biasanya ditemui mahasiswa dalam pengaturan laboratorium tradisional dengan robot sungguhan. Eksperimen jarak jauh sangat tergantung pada konektivitas internet yang stabil, dan gangguan atau masalah keterlambatan dapat memengaruhi kualitas pengalaman. Selain itu, eksperimen jarak jauh mungkin tidak

		memberikan tingkat pengalaman langsung yang sama seperti bekerja langsung dengan robot fisik di laboratorium. Selain itu, ada keterbatasan dalam hal kustomisasi dan modifikasi setup robot, karena kontrolnya dilakukan dari jarak jauh.

Berdasarkan dari studi literatur yang sudah dilakukan, dapat disimpulkan bahwa penggunaan simulator pembelajaran pergerakan robot memiliki peran yang cukup krusial dalam pemahaman dan pelatihan pergerakan robot, terutama pada robot differential drive dan omni-directional. Literatur menegaskan bahwa simulasi memungkinkan eksperimen tanpa risiko fisik, meningkatkan pemahaman konsep kinematika robot, dan memfasilitasi eksperimen dengan aman. Selain itu, penelitian literatur juga menunjukan pentingnya pemahaman model kinematika yang digunakan untuk menggambarkan pergerakan robot. Pemahaman algoritma pengendalian, termasuk titik ke titik dan pelacakan trayek, juga dianggap sebagai aspek penting dalam pembelajaran pergerakan robot. Selain itu, dalam literatur juga mencatat bahwa simulasi memiliki keterbatasan dalam mensimulasikan situasi dunia nyata, sehingga perbandingan dengan perilaku robot fisik menjadi penting. Kesimpulannya adalah bahwa pengembangan simulator pembelajaran pergerakan robot merupakan kontribusi yang berharga pada pendidikan dan pelatihan dalam bidang robotika.

2.2 Dasar Teori

2.2.1 Robot Beroda

Robot beroda adalah mesin yang dapat bergerak sebagai satu kesatuan dengan kendali tertentu dan tingkat otonomi yang bervariasi. Seperti halnya manipulator, gerakan robot beroda dapat dianalisis berdasarkan derajat kebebasan. Robot beroda bisa bergerak dalam tiga dimensi, seperti terbang atau mengapung, yang memungkinkan tiga derajat kebebasan translasi dan tiga derajat kebebasan rotasi yang perlu dikendalikan (tanpa mempertimbangkan sendi internal). Atau robot beroda dapat terbatas untuk bergerak di sepanjang permukaan tertentu, yang memiliki dua derajat kebebasan translasi dan satu derajat kebebasan rotasi (yaw atau azimut). Selain itu, robot beroda juga bisa dibatasi untuk bergerak di jalur atau pipa, yang biasanya memiliki satu derajat

kebebasan translasi dan mungkin satu derajat kebebasan rotasi (roll). Keberadaan sendi internal dalam robot permukaan dapat memungkinkan pengendalian orientasi tubuhnya dalam tiga dimensi [8].

Model kinematika pada robot beroda mengatur bagaimana kecepatan roda mempengaruhi kecepatan robot. Fokus pada aspek kinematika dengan tidak mempertimbangkan dinamika dan juga mengasumsikan bahwa robot bergerak di atas permukaan keras, datar, dan horizontal, tanpa tergelincir. Robot beroda diasumsikan memiliki sasis berbentuk benda padat tunggal (tidak bersendi) dengan konfigurasi $T_{sb} \in SE(2)$ yang mewakili bingkai yang melekat pada sasis {b} dalam hubungannya dengan bingkai ruang tetap {s} pada bidang horizontal. Asumsikan T_{sb} dengan tiga koordinat $q = (\phi, x, y)$. Kecepatan sasis sebagai turunan waktu dari koordinat tersebut, $q^{\cdot} = (\phi^{\cdot}, x^{\cdot}, y^{\cdot})$. Gerakan planar sasis $V_b = (w_{bz}, v_{bx}, v_{by})$ yang dinyatakan dalam {b} [9]. Sehingga memiliki persamaan sebagai berikut:

$$V_b = \left[w_{bz}\ v_{bx}\ v_{by}\right] = [1\ 0\ 0\ 0\ cos\ \phi\ sin\ \phi\ 0\ -sin\ \phi\ cos\ \phi\ ]\ [\phi^{\cdot}\ x^{\cdot}\ y^{\cdot}\ ] \quad (1)$$

$$q^{\cdot} = [\phi^{\cdot}\ x^{\cdot}\ y^{\cdot}\ ] = [1\ 0\ 0\ 0\ cos\ \phi\ -sin\ \phi\ 0\ sin\ \phi\ cos\ \phi\ ]\ \left[w_{bz}\ v_{bx}\ v_{by}\right] \quad (2)$$

Keterangan:

ϕ: Sudut orientasi sasis dalam bidang horizontal.

x: Posisi sasis dalam arah x pada bidang horizontal.

y: Posisi sasis dalam arah y pada bidang horizontal.

$\dot{\phi}$: Kecepatan angular (laju perubahan orientasi) sasis.

$\dot{x}$: Kecepatan linear dalam arah x.

$\dot{y}$: Kecepatan linear dalam arah y.

w_{bz}: Kecepatan angular sasis sekitar sumbu z dalam bingkai sasis.

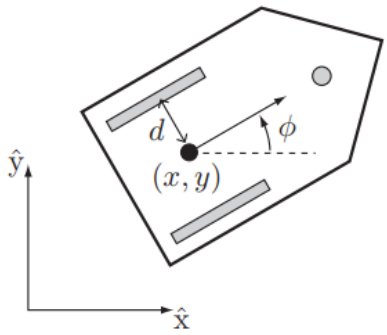
v_{bx}: Kecepatan linear sasis dalam arah x dalam bingkai sasis.

v_{by}: Kecepatan linear sasis dalam arah y dalam bingkai sasis.

Berdasarkan derajat kebebasan bergerak, robot beroda dapat dikelompokan menjadi dua kategori utama, yaitu *differential drive* dan *omni directional*.

2.2.1.1 Differential Drive Robot

Robot Differential-Drive merupakan salah satu struktur dasar dari robot beroda. Differential drive terdiri dari dua roda yang dapat diputar secara mandiri dengan radius r yang berputar sepanjang sumbu yang sama, dan biasanya dilengkapi dengan satu atau lebih roda penyangga, roda bola, atau peluncur gesekan rendah untuk menjaga robot tetap dalam posisi horizontal. Robot differential-drive memiliki dua keunggulan utama, yaitu sederhana dan kemampuan manuver yang tinggi (robot dapat berbelok/berputar dengan memutar roda dalam arah yang berlawanan) Jarak antara kedua roda yang dapat diputar adalah 2d, dan titik referensi *(x, y)* diposisikan di tengah-tengah antara kedua roda tersebut (seperti yang terlihat pada Gambar 2.1).



Gambar 2. 1 Sebuah robot differential drive dengan dua roda biasa dan satu roda penyangga.

Dalam notasi konfigurasi $q = (\phi, x, y, \theta_L, \theta_R)$, dengan θ_L dan θ_R mewakili sudut putaran roda kiri dan roda kanan secara berturut-turut, hubungan kinematika robot differential drive dapat dijelaskan melalui persamaan berikut:

$$q^{\cdot} = [\phi^{\cdot}\ x^{\cdot}\ x^{\cdot}\ \theta_L\ \theta_R\ ]$$

$$q^{\cdot} = \left[-r/2d\ \frac{r}{2}cos\ \phi\ \frac{r}{2}sin\ \phi\ \ r/2\ \frac{r}{2}cos\ \phi\ \frac{r}{2}sin\ \phi\ \ 1\ 0\ \ 0\ 1\ \right]\ [u_L\ u_R\ ] \qquad (3)$$

Kecepatan sudut roda kiri, disebut u_L , dan kecepatan sudut roda kanan, disebut u_R. Kecepatan sudut positif pada masing-masing roda menghasilkan gerakan maju pada roda tersebut. Kendali roda diambil dari rentang nilai antara $[-u_{max}, u_{max}]$. Karena biasanya tidak perlu memperhatikan sudut gulir kedua roda, baris terakhir dapat dihilangkan untuk menyederhanakan sistem kendali, sehingga persamaannya menjadi seperti berikut.

$$q^{\cdot} = [\phi^{\cdot}\ x^{\cdot}\ y^{\cdot}\ ] = \left[-r/2d\ r/2\ \frac{r}{2}cos\ \phi\ \frac{r}{2}cos\ \phi\ \frac{r}{2}sin\ \phi\ \frac{r}{2}sin\ \phi\right]\ [u_L\ u_R\ ] \quad (4)$$

Keterangan:

ϕ: Sudut orientasi sasis dalam bidang horizontal.

x: Posisi sasis dalam arah x pada bidang horizontal.

y: Posisi sasis dalam arah y pada bidang horizontal.

θ_L: Sudut putaran roda kiri.

θ_R: Sudut putaran roda kanan.

$\dot{\phi}$: Kecepatan angular (laju perubahan orientasi) sasis.

$\dot{x}$: Kecepatan linear dalam arah x.

$\dot{y}$: Kecepatan linear dalam arah y.

u_L: Kecepatan sudut roda kiri.

u_R: Kecepatan sudut roda kanan.

r: Jari-jari roda.

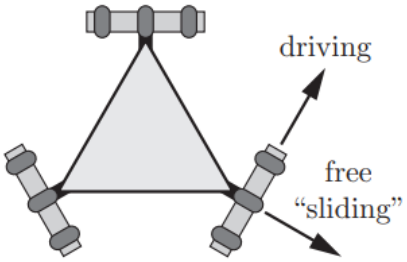
d: Jarak antara roda kiri dan kanan.

2.2.1.2 Omni Directional Robot

Robot beroda omnidireksional memiliki kemampuan unik untuk bergerak tanpa adanya kendala pada kecepatan sasisnya. Ini berarti bahwa robot ini memiliki kebebasan penuh dalam menggerakkan sasisnya tanpa adanya kendala khusus pada arah atau kecepatan tertentu. Dalam hal ini, robot omnidireksional dapat bergerak ke segala arah dengan mudah dan tidak terbatas dalam manuvernya. Robot jenis ini biasanya menggunakan roda omni yang memiliki rol di sekeliling cincin luarnya. Rol-rol ini dapat berputar dengan bebas sepanjang sumbu yang sejajar dengan roda dan sejajar dengan lingkaran luar roda. Hal ini memungkinkan pergerakan ke samping ketika roda maju atau mundur. Kemampuan pergerakan menyamping yang dimungkinkan oleh roda omni menjadikan sasis robot bebas dari pembatasan kecepatan. Roda omni tidak melibatkan pengemudian, melainkan hanya digerakkan maju atau mundur. Karena memiliki rol-rol kecil, roda omni dapat optimal di permukaan yang keras dan rata.

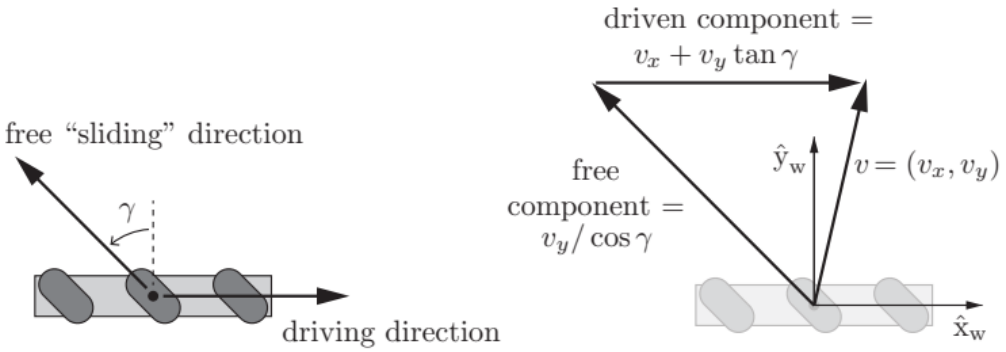
2.2.1.2.1 *Modeling*

Untuk mencapai kecepatan sasis tiga dimensi yang bebas, sebuah robot beroda omnidireksional harus memiliki minimal tiga roda, karena masing-masing roda hanya memiliki satu motor yang mengontrol kecepatan maju dan mundurnya. Berdasarkan gambar 2.2 dapat dilihat pergerakan yang diperoleh dengan menggerakkan motor dan pergerakan meluncur bebas yang diperoleh dari rol roda.



Gambar 2. 2 Robot omnidireksional dengan 3 roda omni

Untuk kinematiknya dapat dilihat pada Gambar 2.3 berikut.



Gambar 2. 3 Kinematik roda omni

Kinematika roda seperti yang diilustrasikan dalam Gambar 2.3 diketahui bahwa $\hat{}x_w - \hat{}y_w$ yang berpusat pada roda, serta kecepatan linier pusat roda ditulis sebagai $v = (v_x, v_y)$ yang memenuhi:

$$[v_x\ v_y\ ] = v_{drive}[1\ 0\ ] + v_{slide}[-sin\,\gamma\ \ cos\,\gamma\ ] \quad (5)$$

γ adalah sudut di mana pergerakan geser bebas terjadi, yang dimungkinkan oleh rol-rol pasif di sekitar roda. Kecepatan penggerak (v_{drive}) adalah kecepatan yang digunakan untuk menggerakkan roda, sementara kecepatan geser (v_{drive}) adalah kecepatan yang menghasilkan pergerakan geser. Untuk roda omni, γ = 0, dan untuk roda mecanum, γ biasanya adalah ±45. Dengan memecah Persamaan (sebelumnya), kita dapat menghitung v_{drive} dan v_{slide} ini.

$$v_{drive} = v_x + v_y \tan\gamma \tag{6}$$

$$v_{slide} = v_y / \cos\gamma \tag{7}$$

Jika jari-jari adalah *r*, serta kecepatan sudut penggerak adalah *u,* dapat ditulis persamaan berikut:

$$u = \frac{v_{drive}}{r} = \frac{1}{r}(v_x + v_y \tan\tan\gamma) \tag{8}$$

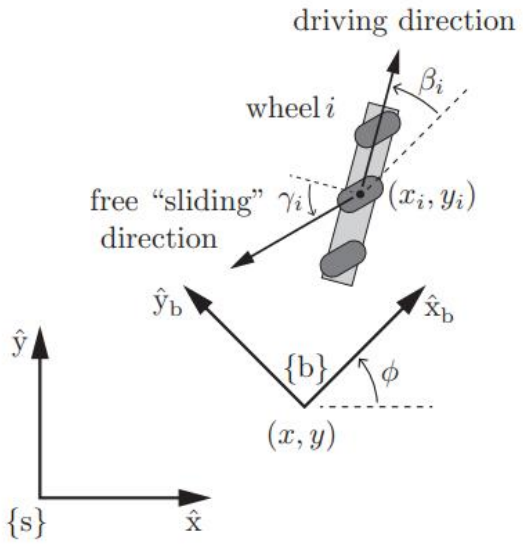
Rumus lengkap untuk mengubah kecepatan rangkaian ˙q = (ϕ˙, x˙, y˙) menjadi kecepatan sudut penggerak ui untuk roda i, perhatikan notasi yang ditunjukkan dalam persamaan berikut:

$$u_i = h_i(\phi)\dot{q}$$

$$u_i = \left[\frac{1}{r_i} \frac{\tan\tan\gamma_i}{r_i}\right][\cos\cos\beta_i \ \sin\sin\beta_i - \sin\sin\beta_i \ \cos\cos\beta_i][-y_i \ 1 \ 0 \ x_i \ 0 \ 1][1 \ 0 \ 0 \ 0 \ \cos\cos\phi \ \sin\sin\phi \ 0 \ -\sin\sin\phi \ \cos\cos\phi][\dot{\phi} \ \dot{x} \ \dot{y}] \tag{9}$$

Langkah pertama dalam transformasi ini mengubah $\dot{q}$ menjadi V_b; langkah kedua menghasilkan kecepatan linear di roda dalam kerangka {b}; langkah ketiga mengungkapkan kecepatan linear ini dalam kerangka roda $\hat{x}_w - \hat{y}_w$; dan langkah terakhir menghitung kecepatan sudut penggerak dengan menggunakan Persamaan (13.4). Ketika mengevaluasi persamaan (13.5) untuk $h_i(\phi)$, kita akan mendapatkan persamaan:

$$h_i(\phi) = \frac{1}{r_i \cos\cos\gamma_i}[x_i \sin(\beta_i + \gamma_i) - y_i \cos\cos(\beta_i + \gamma_i) \ \cos\cos(\beta_i + \gamma_i + \phi) \ \sin\sin(\beta_i + \gamma_i + \phi)]^T \tag{10}$$



Gambar 2. 4 Arah gerak dan geser roda

Berdasarkan gambar dapat dilihat bahwa kerangka rangkaian {b} berada pada koordinat $q = (\phi, x, y)$ dalam kerangka ruang tetap {s}. Pusat roda dan arah penggerakannya dinyatakan oleh vektor (β_i, x_i, y_i) dalam kerangka {b}, dengan jari-jari roda ri, dan arah geser roda dinyatakan oleh vektor γ_i. Selain itu, kecepatan sudut penggerak roda, ui, memiliki hubungan dengan vektor kecepatan rangkaian ˙q. Kerangka ruang tetap {s} dan kerangka rangkaian {b} menunjukkan posisi relatif rangkaian dan roda dalam {s}, sedangkan vektor (x_i, y_i) menggambarkan posisi roda dalam kerangka {b} dengan arah penggerak β_i, yang diukur dalam {b}. Arah geser roda ditentukan oleh vektor γ_i.

Untuk robot omnidireksional dengan m ≥ 3 roda, matriks $H(\phi) \in R^{m\times 3}$ memetakan kecepatan rangkaian yang diinginkan $\dot{q} \in R^3$ ke vektor kecepatan penggerak roda $u \in R^m$ dibangun dengan cara menggabungkan *m* baris $h_i(\phi)$ seperti persamaan berikut:

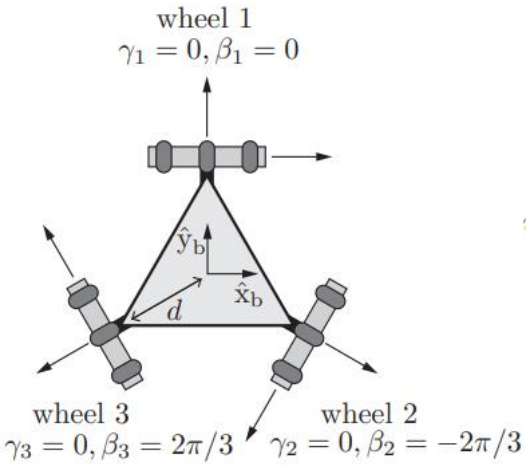
$$u = H(\phi)\dot{q} = [h_1(\phi)\ h_2(\phi)\ \vdots\ h_m(\phi)\ ][\dot{\phi}\ \dot{x}\ \dot{y}\ ] \quad (11)$$

Dapat juga menyatakan hubungan antara u dan perputaran tubuh rangkaian V_b. Pemetaan ini tidak bergantung pada posisi orientasi rangkaian ϕ.

$$u = H(0)V_b = [h_1(0)\ h_2(0)\ \vdots\ h_m(0)\ ]\left[w_{bz}\ v_{bx}\ v_{by}\ \right] \quad (12)$$

Dalam memilih posisi dan arah roda (β_i, x_i, y_i) dalam kerangka {b}, serta arah geser bebas mereka yi, harus dipastikan bahwa matriks H(0) memiliki peringkat 3. Sebagai contoh, jika

membuat sebuah robot beroda omnidireksional di mana arah penggerakannya dan arah gesernya semua sejajar, maka peringkat matriks H(0) akan menjadi 2, dan ini akan mengakibatkan tidak adanya cara untuk menghasilkan gerakan translasional yang dapat dikendalikan dalam arah geser tersebut.



Gambar 2. 5 Kinematika robot omnidireksional tiga roda

Menggunakan notasi dalam Gambar 2.5, robot beroda omnidireksional dengan tiga roda omniwheel memiliki model kinematika dengan persamaan berikut.

$$u = [u_1\ u_2\ u_3\ ]$$

$$u = H(0)V_b = \frac{1}{r}\left[-d\ 1\ 0\ \ -d\ \ -1/2\ \ -sin\left(\frac{\pi}{3}\right)\ -d\ \ -1/2\ sin\left(\frac{\pi}{3}\right)\right]\left[w_{bz}\ v_{bx}\ v_{by}\ \right] \qquad (13)$$

Keterangan:

ϕ: Sudut orientasi sasis dalam bidang horizontal.

x: Posisi sasis dalam arah xxx pada bidang horizontal.

y: Posisi sasis dalam arah yyy pada bidang horizontal.

θ_L: Sudut putaran roda kiri.

θ_R: Sudut putaran roda kanan.

$\dot{\phi}$: Kecepatan angular sasis.

$\dot{x}$: Kecepatan linear dalam arah xxx.

$\dot{y}$: Kecepatan linear dalam arah yyy.

v_x: Komponen kecepatan linier dalam arah xxx.

v_y: Komponen kecepatan linier dalam arah yyy.

v_{drive} : Kecepatan penggerak roda.

v_{slide}: Kecepatan geser roda.

γ: Sudut pergerakan geser roda.

r: Jari-jari roda.

u: Kecepatan sudut penggerak roda.

β_i: Sudut arah roda iii dalam kerangka {b}.

x_i: Posisi roda iii dalam arah xxx dalam kerangka {b}.

y_i: Posisi roda iii dalam arah yyy dalam kerangka {b}.

2.3 Analisis *Stakeholder*

Adapun, mereka yang dapat menggunakan simulator ini termasuk mahasiswa jurusan teknik di Universitas Islam Indonesia (FTI UII) yang memiliki minat atau mata kuliah terkait robotika, para dosen pengajar di FTI UII yang ingin memanfaatkannya dalam pendidikan sebagai sarana belajar, serta para peneliti dan praktisi di industri robotika yang memerlukan alat pengujian dan pengembangan yang efisien.

2.4 Analisis Aspek yang Mempengaruhi Sistem

2.4.1 Aspek Ekonomi

Kemudahan pengguna dalam aspek ekonomi sangat penting. Simulator harus mempertimbangkan ketersediaan sumber daya keuangan untuk pengembangan dan pemeliharaan. Ini termasuk memastikan bahwa simulator tetap terjangkau bagi mahasiswa, lembaga pendidikan, dan peneliti dengan anggaran terbatas. Ketersediaan alat dan sumber daya yang murah serta perencanaan biaya yang bijaksana adalah faktor penting dalam memastikan bahwa pengguna dapat dengan mudah mengakses dan menggunakan simulator ini tanpa hambatan finansial yang signifikan.

2.4.2 Aspek Teknologi

Kemudahan pengguna dalam aspek teknologi berkaitan dengan keterbaruan dan integrasi teknologi. Simulator harus dirancang sedemikian rupa sehingga dapat diperbarui seiring perkembangan teknologi robotika dan komputasi. Ini berarti pengguna tidak harus bersusah payah untuk menyusun alat dan perangkat lunak baru secara berkala. Selain itu, integrasi yang mudah

dengan teknologi dan perangkat terkini, seperti sistem pengenalan suara atau pengenalan citra untuk pengujian robot, akan meningkatkan kemudahan penggunaan dan memastikan bahwa simulator tetap relevan dalam lingkungan teknologi yang terus berkembang.

2.4.3 Aspek Hukum dan Regulasi

Kemudahan pengguna dalam aspek hukum dan regulasi berkaitan dengan memastikan bahwa pengguna dapat dengan mudah memahami dan mematuhi peraturan dan ketentuan hukum yang berlaku. Ini termasuk hak cipta dan lisensi perangkat lunak yang harus ditaati. Selain itu, jika simulator mengumpulkan atau memproses data pengguna, penting untuk memastikan bahwa simulator mematuhi persyaratan privasi data yang berlaku. Dengan menyediakan panduan dan informasi yang jelas mengenai aspek hukum dan regulasi, pengguna akan merasa lebih nyaman dan dapat dengan mudah mematuhi aturan yang berlaku.

2.4.4 Aspek Lingkungan

Kemudahan pengguna dalam aspek lingkungan melibatkan pertimbangan efisiensi energi dan dampak lingkungan. Simulator harus dirancang untuk mengkonsumsi daya dengan efisien, sehingga pengguna tidak perlu menghadapi beban energi yang berlebihan. Dalam hal dampak lingkungan, penggunaan perangkat keras yang ramah lingkungan atau kebijakan pengurangan limbah dapat menjadi bagian dari pendekatan yang berkelanjutan. Ini memastikan bahwa pengguna dapat dengan mudah menggunakan simulator tanpa terlalu banyak mempengaruhi lingkungan sekitar.

2.5 Spesifikasi Sistem

Berdasarkan kajian literatur, dasar teori dan informasi yang didapat, spesifikasi dan kriteria sistem simulator pembelajaran pergerakan robot yang ingin kami buat adalah sebagai berikut:

1. Sistem memiliki perangkat keras yang mencakup dua jenis robot, yaitu differential drive dan omni-directional.
2. Differential drive robot menggunakan dua motor DC, dua roda biasa, dan satu roda bola sebagai penyeimbang, sedangkan omni-directional robot menggunakan tiga motor DC dan

tiga roda omni. Selain itu, kedua jenis robot ini menggunakan jenis mikrokontroler yang sama.

3. Jumlah robot yang dapat disimulasikan 3 untuk robot differential drive dan 1 untuk omni-directional drive.
4. Sistem memiliki perangkat lunak yang terbagi berdasarkan fungsinya, yaitu mengolah pembacaan pattern, mensimulasikan pergerakan robot, mengendalikan pergerakan robot fisik, serta berfungsi sebagai antarmuka pengguna.
5. Antarmuka pengguna memfasilitasi pengguna untuk mengatur titik tujuan pergerakan robot, menampilkan simulasi robot virtual dan dokumentasi pergerakan robot, termasuk algoritma yang digunakan.
6. Simulasi pergerakan robot secara langsung dilakukan pada lapangan seluas 3×4 m2.

BAB 3. USULAN SOLUSI

Dalam pengembangan simulasi pembelajaran pergerakan robot beroda ini, terdapat usulan solusi yang harus dicetuskan dengan mengikuti proses yang terarah untuk menyelesaikan masalah yang sedang dihadapi. Adapun solusi hadir untuk memenuhi kebutuhan untuk bekerjanya sistem ini dengan lancar. Nantinya akan dipilih sebuah usulan solusi yang terbaik dari setiap usulan yang akan diberikan.

3.1 Usulan Solusi 1

Usulan solusi 1 adalah sistem simulasi virtual dan langsung menggunakan protokol ROSbridge Suite. Sistem ini mencakup pengembangan perangkat lunak simulasi virtual dan simulasi langsung menggunakan perangkat keras (robot fisik) dan lapangan nyata. Perangkat keras terdiri dari differential drive dan omni-directional yang menggunakan mikrokontroler ESP32, motor driver L298N, motor DC, dan baterai 12V. Dengan solusi ini mahasiswa dapat interaktif memahami pergerakan robot beroda differential drive dan omni-directional melalui platform web yang dapat mengatur titik tujuan robot, menyajikan peta lingkungan simulasi 2D, dan memberikan dokumentasi pergerakan robot.

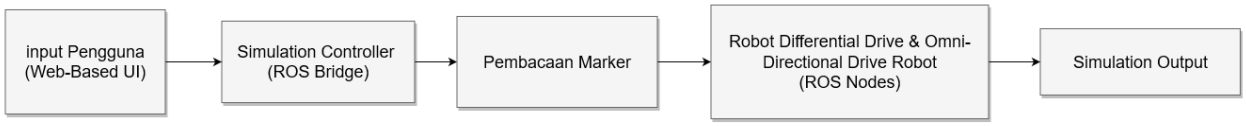
Mikrokontroler ESP32 dipilih sebagai otak utama perangkat keras karena menawarkan dua inti pemrosesan (dual-core) yang dapat dioptimalkan untuk menjalankan tugas komputasi yang kompleks dan mengelola komunikasi dengan sistem ROS melalui koneksi WiFi. Keunggulan ESP32 terletak pada konektivitas nirkabelnya yang dapat diandalkan, mendukung interaksi real-time antara simulasi dan robot fisik. Motor driver L298N dan motor DC dipilih untuk menggerakkan differential drive dan omni-directional robot. Motor driver L298N menonjol dalam memberikan kontrol yang stabil terhadap motor DC, memastikan pergerakan robot yang akurat sesuai dengan instruksi yang diterima dari simulasi.

Usulan solusi ini dipilih karena ROSbridge Suite menyediakan jembatan yang kuat antara ROS dan aplikasi eksternal melalui protokol JSON dan WebSocket. Integrasi dengan ROSbridge memungkinkan komunikasi yang efisien dan real-time antara platform web dan sistem ROS. Penggunaan WebSocket memberikan kemampuan untuk mengirim dan menerima pesan dengan cepat, yang penting untuk pengendalian robot secara langsung dan responsif. Dengan demikian, solusi ini memberikan pengalaman belajar yang terintegrasi dan komprehensif, memungkinkan

mahasiswa untuk merasakan pergerakan robot secara virtual dan langsung dengan baik.

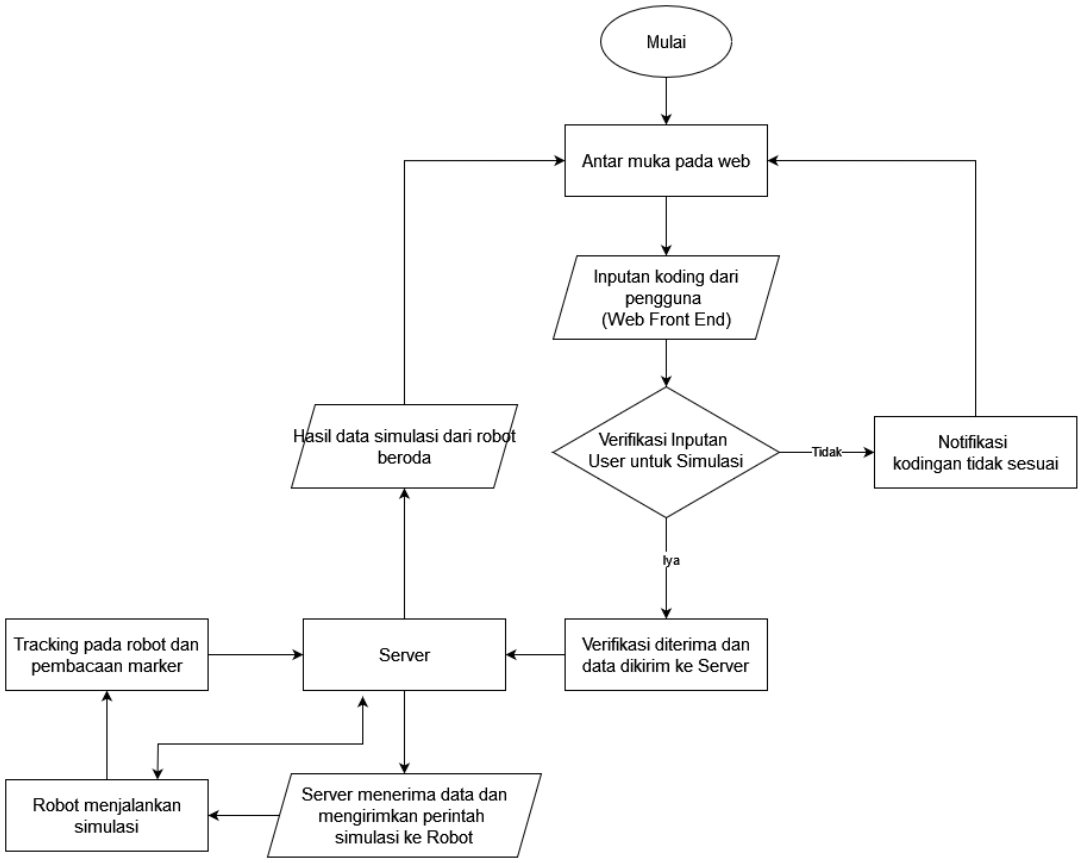
Keunggulan ROSbridge juga memberikan fleksibilitas dalam mengelola aliran informasi, menciptakan pengalaman belajar yang mendalam dalam bidang pengendalian robot. Dengan integritas dan responsivitas yang tinggi, solusi ini memberikan mahasiswa kesempatan untuk mendalami konsep-konsep kontrol robot dengan lebih baik, mengoptimalkan pemahaman mereka melalui pengalaman simulasi yang terhubung secara langsung dengan sistem fisik robot.

3.1.1 Desain Sistem 1

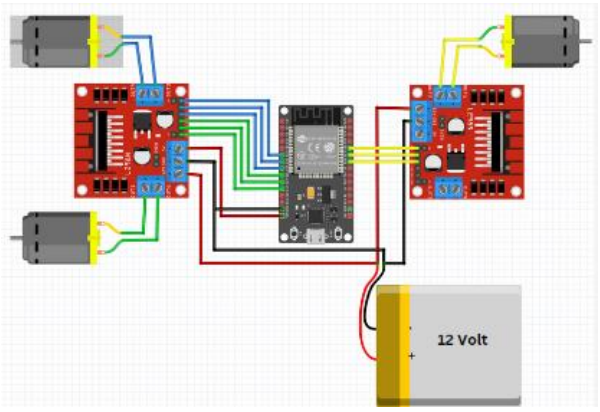


Gambar 3. 1 Diagram blok usulan solusi 1

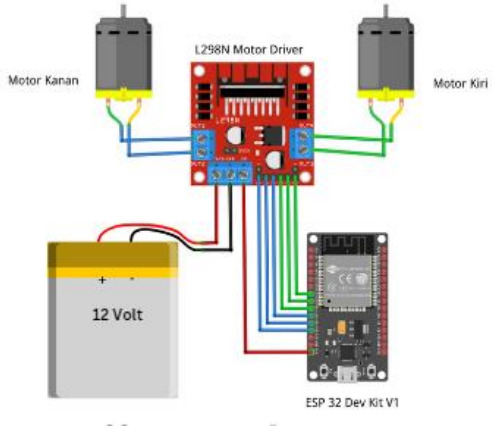
Simulasi pergerakan robot beroda dengan menggunakan ROS bridge Suite dimulai dengan mengakses robot beroda melalui sistem ROS. Robot differential drive dan omni-directional akan terhubung ke simulator melalui jembatan ROS Bridge. Proses ini memungkinkan pengguna untuk mengatur titik tujuan robot dan mengobservasi pergerakan robot secara virtual melalui platform web. Selanjutnya, sistem akan menyajikan peta lingkungan simulasi 2D yang mencerminkan kondisi lapangan nyata. Selama simulasi, komunikasi real-time antara platform web dan ROS memastikan responsifitas tinggi terhadap setiap perintah pengguna. Pengguna dapat melihat dokumentasi pergerakan robot, memahami karakteristik pergerakan dari jenis robot tertentu, dan menganalisis data yang dihasilkan oleh simulasi.


Mulai
Antar muka pada web
Inputan koding dari pengguna (Web Front End)
Verifikasi Inputan User untuk Simulasi
Tidak
Notifikasi kodingan tidak sesuai
Iya
Verifikasi diterima dan data dikirim ke Server
Server
Tracking pada robot dan pembacaan marker
Hasil data simulasi dari robot beroda
Server menerima data dan mengirimkan perintah simulasi ke Robot
Robot menjalankan simulasi

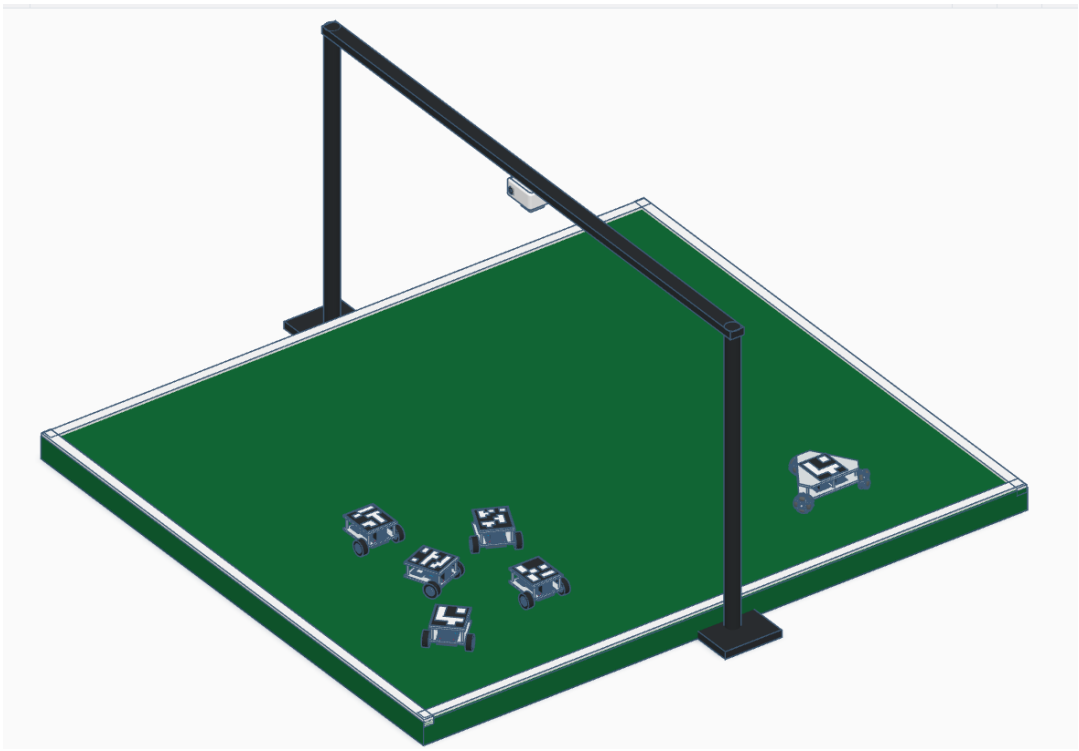
(a)


12 Volt

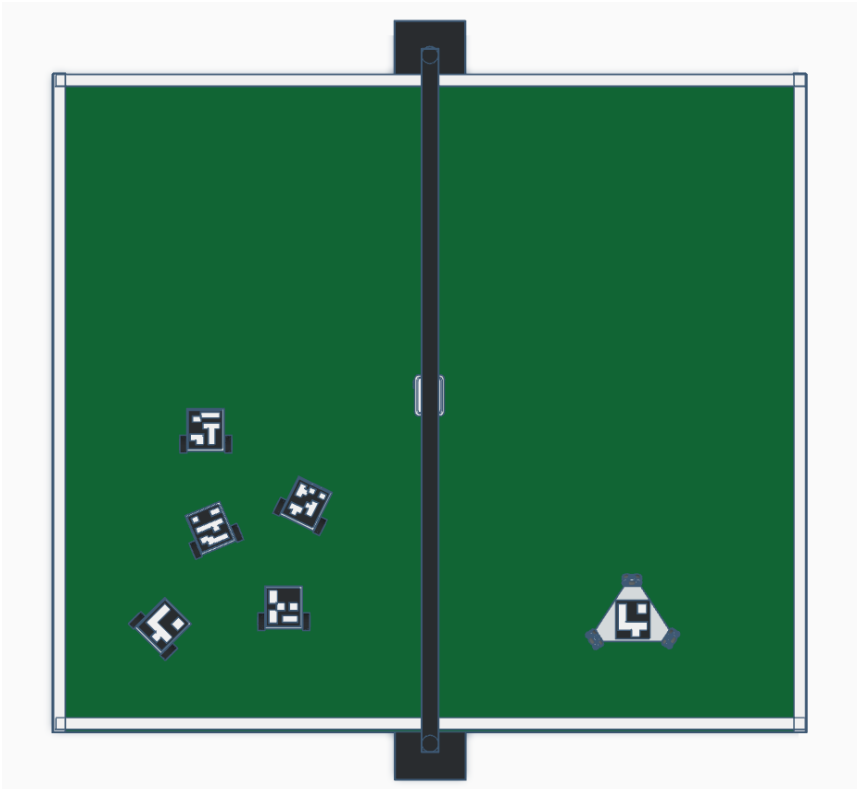
(b)


L298N Motor Driver
Motor Kanan
Motor Kiri
12 Volt
ESP 32 Dev Kit V1

(c)



(d)



(e)

Gambar 3. 2 Ilustrasi usulan rancangan sistem secara umum. (a) Proses cara kerja sistem, (b) Rangkaian robot beroda Omni-Directional 3 roda, (c) Rangkaian robot beroda Differential Drive 2 roda, (d) Gambaran sistem dalam tampilan 3D tampak samping, (e) Gambaran sistem dalam tampilan 3D tampak dari atas

Untuk dapat memenuhi usulan sistem tersebut, maka diperlukan inventarisasi kebutuhan sistem perangkat keras. Tabel 3.1 memperlihatkan kebutuhan sistem sesuai usulan dan spesifikasi yang dibutuhkan.

Tabel 3. 1 Inventarisasi kebutuhan usulan sistem perangkat keras

No	Nama Alat	Keterangan
1	Robot beroda Differential	Robot dengan dua roda yang dapat berputar secara independen, mendukung pergerakan differential drive. Berperan sebagai perangkat aktif dalam simulasi.
2	Robot beroda Omni-Directional	Robot dengan roda khusus yang memungkinkan pergerakan dalam berbagai arah tanpa perlu berputar secara fisik. Berperan sebagai perangkat aktif dalam simulasi.
3	Mikroprosessor	Komponen utama yang menjalankan logika dan pengolahan data pada simulator. Disini mikroprosesor yang digunakan ialah ESP32 untuk pengiriman data.
4	Marker	Penanda visual yang digunakan dalam identifikasi robot pada simulasi. Disini menggunakan ArUco marker sebagai penanda yang akan digunakan pada robot untuk simulasi
5	Kamera atau sensor penginderaan	Kamera yang digunakan untuk pembacaan dari marker yang terpasang pada robot sehingga dapat mendapatkan informasi visual akan lapangan robot beroda.
6	Sistem Koneksi Wi-fi atau Ethernet	Infrastruktur untuk menghubungkan simulator dengan web dan serta komunikasi real-time antara web dan robot.
7.	Baterai	Baterai merupakan sumber daya energi yang digunakan untuk dapat menggerakan robot ataupun perangkat lain. Baterai disini digunakan sebagai sumber daya mikroprosesor ESP32 dan motor

Dikarenakan sistem simulasi robot beroda ini tidak hanya mengandalkan perangkat keras, namun juga memanfaatkan perangkat lunak untuk memberikan pengalaman belajar yang lebih

baik, maka dalam perancangan ini, kami menyertakan aplikasi berbasis web yang dapat diakses oleh pengguna melalui browser. Aplikasi web ini dirancang untuk memberikan antarmuka yang intuitif dan mudah digunakan, sehingga mahasiswa dapat dengan nyaman menjalankan simulasi robot beroda, mengatur pergerakan, dan memahami konsep-konsep pengendalian robot secara langsung melalui website.

3.1.2. Rencana Anggaran Desain Sistem 1

Tabel 3. 2 Rencana anggaran pengembangan sistem simulasi robot beroda

No	Item/Pengeluaran	Satuan	Harga satuan	Jumlah	Harga Total	Link Pembelian
1	ESP32 ESP-32 WIFI Bluetooth IoT ESP-32S Development Board	pcs	Rp 62,000.00	5	Rp 310,000.00	https://t.ly/bDZ3I
2	L298N Dual Motor Driver Module L298 H-Bridge	pcs	Rp 15,350.00	7	Rp107,450.00	https://t.ly/mv8R0
3	[CNC] 40PCS JUMPER CABLE KABEL 20CM FEMALE TO FEMALE DUPONT	set	Rp 8,500.00	1	Rp 8,500.00	https://shorturl.at/bBHQU
4	[CNC] 40PCS JUMPER CABLE KABEL 10CM	set	Rp 9,000.00	1	Rp 9,000.00	https://shorturl.at/bBHQU

No	Item/Pengeluaran	Satuan	Harga satuan	Jumlah	Harga Total	Link Pembelian
	MALE TO MALE DUPONT					
5	[CNC] 40PCS JUMPER CABLE KABEL 10CM MALE TO FEMALE DUPONT	set	Rp 8,500.00	1	Rp 8,500.00	https://shorturl.at/gGZ19
6	WHEEL RODA RUBBER FOR SMART ROBOT CAR 4WD 2WD	pcs	Rp 5,000.00	12	Rp 60,000.00	https://shorturl.at/lmxGJ
7	Holder Bracket Braket Breket 3 Sambungan Battery Baterai 18650 - 1solt	set	Rp 30,000.00	2	Rp 60,000.00	https://shorturl.at/lmHMZ
8	Nickel Strip Battery 18650 1 Meter/Strip Nikel Baterai Lithium	pcs	Rp 28,000.00	1	Rp 28,000.00	https://shorturl.at/ehruX
9	DINAMO MOTOR DC 12-18V 3.5A MOTOR DINAMO	pcs	Rp 33,500.00	12	Rp 402,000.00	https://t.ly/BiMdV

No	Item/Pengeluaran	Satuan	Harga satuan	Jumlah	Harga Total	Link Pembelian
	12V 18V DC LS775 WITH CABLE					
10	Baterai Charger/Cas Ulang Li-ion FS 18650 13000mAh 4,2V Hijau Isi 2 pc	set	Rp 27,000.00	9	Rp 243,000.00	https://t.ly/qT1KD
11	PCB DOT MATRIX THRU HOLE SINGLE LAYER 5X7CM 5*7CM LUBANG BOLONG MATRIK	pcs	Rp 2,700.00	6	Rp 16,200.00	https://rb.gy/rhc8bc
Total					Rp 1,262,650.00	

3.1.3 Analisis Risiko Desain 1

Implementasi ROS Bridge Suite sebagai protokol komunikasi utama dalam simulasi memiliki beberapa risiko potensial. Ketergantungan yang signifikan pada ekosistem ROS dapat menyebabkan kerentanannya terhadap perubahan atau masalah yang mungkin terjadi pada ROS. Selain itu, kebutuhan akan koneksi internet yang stabil dapat menjadi hambatan, mengingat simulasi memerlukan akses online untuk berfungsi.

Kesulitan integrasi dengan hardware fisik seperti robot beroda dan lapangan nyata juga merupakan risiko yang perlu diperhitungkan, karena konfigurasi yang diperlukan mungkin memerlukan keahlian teknis tertentu. Selain itu, ada risiko bahwa simulasi tidak sepenuhnya memenuhi kebutuhan pembelajaran mahasiswa, dan aspek pergerakan robot mungkin tidak dapat diterapkan secara realistis.

3.1.4 Pengukuran Performa Usulan 1

Usulan solusi 1, yang melibatkan sistem simulasi virtual dan langsung dengan ROSbridge Suite, dapat dinilai melalui beberapa parameter performa. Responsivitas sistem menjadi fokus utama evaluasi, dengan pengukuran tingkat keterlambatan dalam pengiriman dan penerimaan pesan antara platform web dan robot fisik melalui ROSbridge Suite. Responsivitas yang tinggi diharapkan dapat menghasilkan pengendalian robot yang lebih akurat dan real-time.

Akurasi kontrol robot menjadi parameter berikutnya yang harus dievaluasi, menilai sejauh mana platform web mampu mengatur titik tujuan robot dan sejauh mana robot fisik dapat mereproduksi pergerakan yang diinginkan. Kemampuan simulasi 2D dari platform web juga menjadi fokus, dengan penilaian terhadap keakuratan peta lingkungan simulasi yang disajikan. Keberhasilan solusi ini juga dapat diukur dari interaktivitas mahasiswa dalam mengatur titik tujuan dan mengamati pergerakan robot secara virtual.

Melalui pengukuran performa pada parameter-parameter tersebut, kita dapat mengevaluasi efektivitas solusi 1 dalam mencapai tujuan pengembangan sistem simulasi yang terintegrasi, mendukung pembelajaran pengendalian robot, dan memenuhi standar keteknikan yang diperlukan.

3.2 Usulan Solusi 2

Usulan solusi 2 adalah sistem simulasi virtual dan langsung dengan Message Queueing Telemetry Transport (MQTT). Sistem ini menggabungkan keunggulan protokol MQTT untuk menciptakan solusi terintegrasi dalam pengembangan perangkat lunak simulasi virtual dan simulasi langsung menggunakan perangkat keras (robot fisik) dan lapangan nyata. Perangkat keras terdiri dari differential drive dan omni-directional yang menggunakan mikrokontroler ESP32, motor driver L298N, motor DC, dan baterai 12V. Dengan solusi ini mahasiswa dapat interaktif memahami pergerakan robot beroda differential drive dan omni-directional melalui platform web yang dapat mengatur titik tujuan robot, menyajikan peta lingkungan simulasi 2D, dan memberikan dokumentasi pergerakan robot.

Mikrokontroler ESP32 dipilih sebagai otak utama perangkat keras karena menawarkan dua inti pemrosesan (dual-core) yang dapat dioptimalkan untuk menjalankan tugas komputasi yang kompleks dan mengelola komunikasi dengan sistem ROS melalui koneksi WiFi. Keunggulan ESP32 terletak pada konektivitas nirkabelnya yang dapat diandalkan, mendukung interaksi real-time antara simulasi dan robot fisik. Motor driver L298N dan motor DC dipilih untuk

menggerakkan differential drive dan omni-directional robot. Motor driver L298N menonjol dalam memberikan kontrol yang stabil terhadap motor DC, memastikan pergerakan robot yang akurat sesuai dengan instruksi yang diterima dari simulasi.

Pilihan ini didasarkan pada keunggulan MQTT dalam komunikasi yang ringan dan efisien. MQTT cocok untuk skenario Internet of Things (IoT) dan sistem terdistribusi, memungkinkan pengiriman pesan dengan latensi rendah. Solusi ini menggunakan MQTT sebagai protokol komunikasi antara platform web dan ROS, memastikan respons yang cepat terhadap perubahan dalam pengaturan tujuan robot pada platform web.

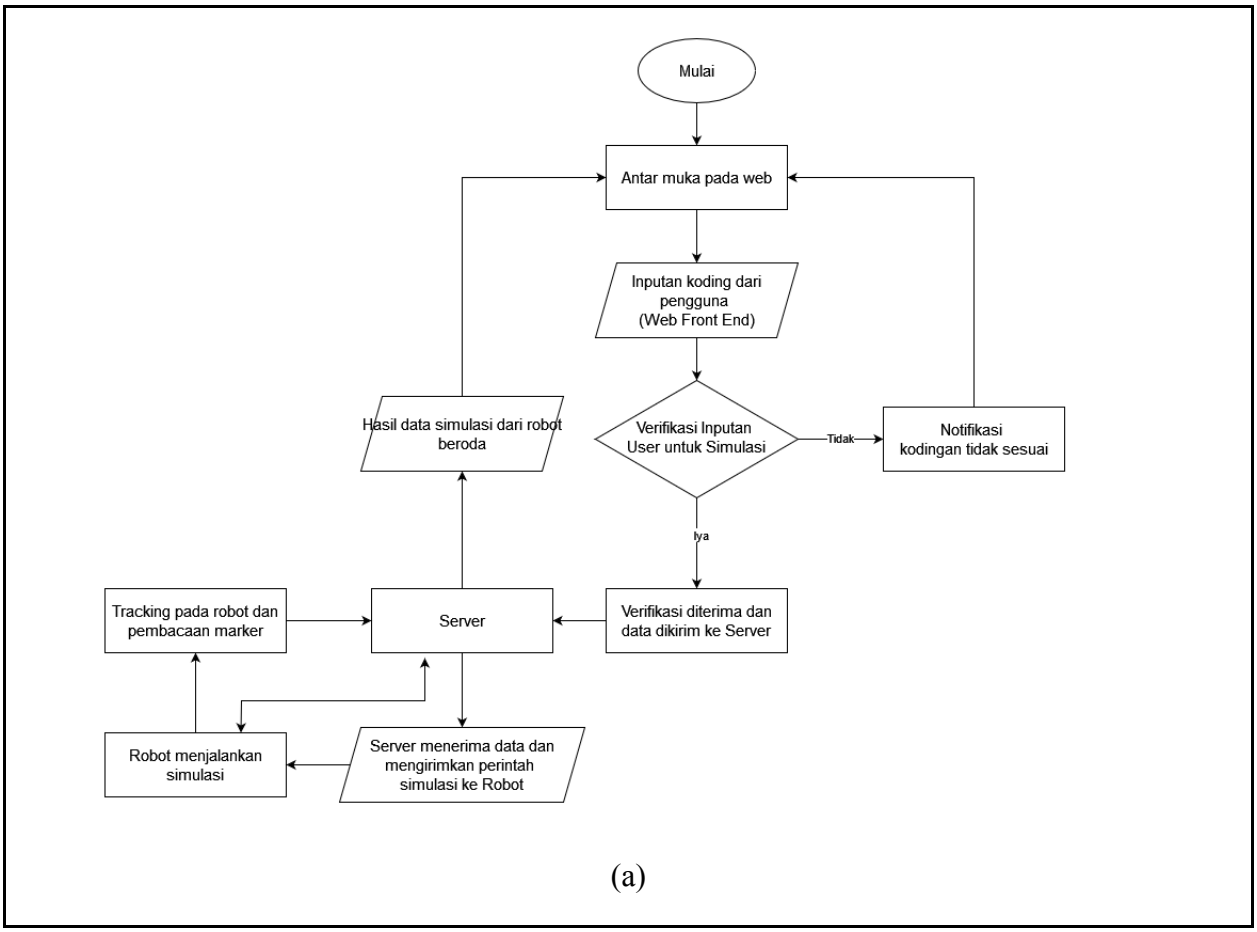
Keunggulan MQTT dalam menangani banyak koneksi secara bersamaan memberikan fleksibilitas dan kinerja yang baik dalam mengelola aliran informasi antara komponen sistem secara real-time. Dengan integritas, efisiensi, dan responsivitas yang tinggi, solusi ini memberikan mahasiswa kesempatan untuk mendalami konsep-konsep kontrol robot melalui pengalaman simulasi yang terhubung secara langsung dengan sistem fisik robot. Melalui protokol MQTT, sistem ini mampu menyediakan pengalaman belajar yang mendalam dan terhubung dengan baik antara simulasi dan robot fisik.

Secara umum, sistem yang menggunakan MQTT dirancang untuk komunikasi yang efisien dan ringan, terutama digunakan dalam konteks Internet of Things (IoT). hal ini menawarkan kemampuan untuk menangani banyak koneksi secara bersamaan dan memberikan pengiriman pesan dengan latensi rendah. Di sisi lain, usulan solusi sebelumnya yaitu menggunakan ROSbridge Suite menghubungkan aplikasi eksternal dengan Robot Operating System (ROS) melalui protokol JSON dan WebSocket. ROSbridge memberikan jembatan yang kuat antara ROS dan aplikasi eksternal, menonjolkan responsivitas tinggi dan kemampuan untuk mentransfer data kompleks dan pesan ROS secara real-time.

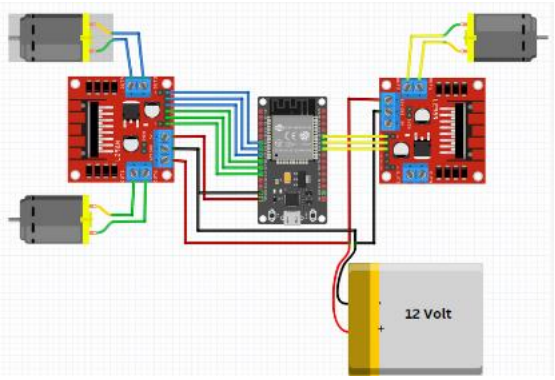
3.2.1 Desain Sistem 2

Gambar 3. 3 Diagram blok usulan solusi 2

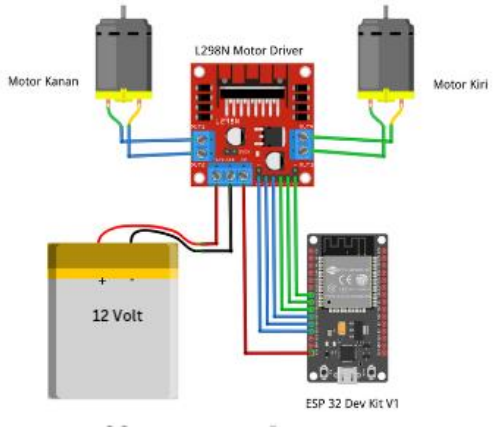
Simulasi pergerakan robot beroda dengan menggunakan MQTT dimulai dengan menjembatani komunikasi antara platform web dan robot beroda menggunakan protokol MQTT. Pengguna dapat mengatur titik tujuan robot dan mengamati pergerakan virtual melalui platform web. Informasi pergerakan robot akan dikirim melalui MQTT ke simulator, menciptakan tampilan 2D lingkungan simulasi. Responsivitas tinggi terhadap setiap perintah pengguna dijamin melalui keunggulan komunikasi ringan dan efisien yang ditawarkan oleh MQTT. Pengguna dapat mengakses dokumentasi pergerakan robot, menganalisis data simulasi, dan memahami karakteristik pergerakan robot secara mendalam. Proses ini menjadi siklus iteratif yang terus berlangsung, memberikan pengalaman belajar yang komprehensif dan terhubung secara langsung dengan sistem fisik robot.



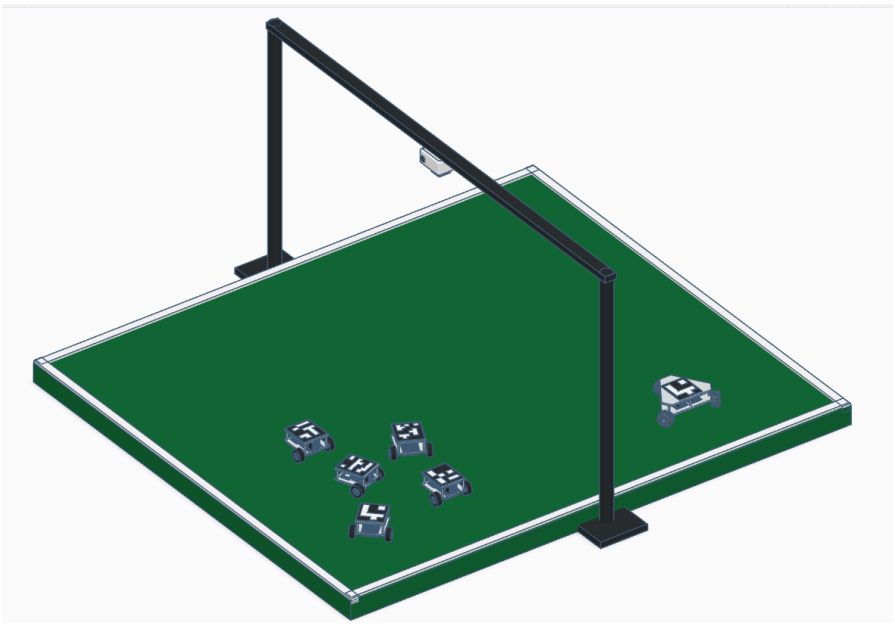
(a)


12 Volt

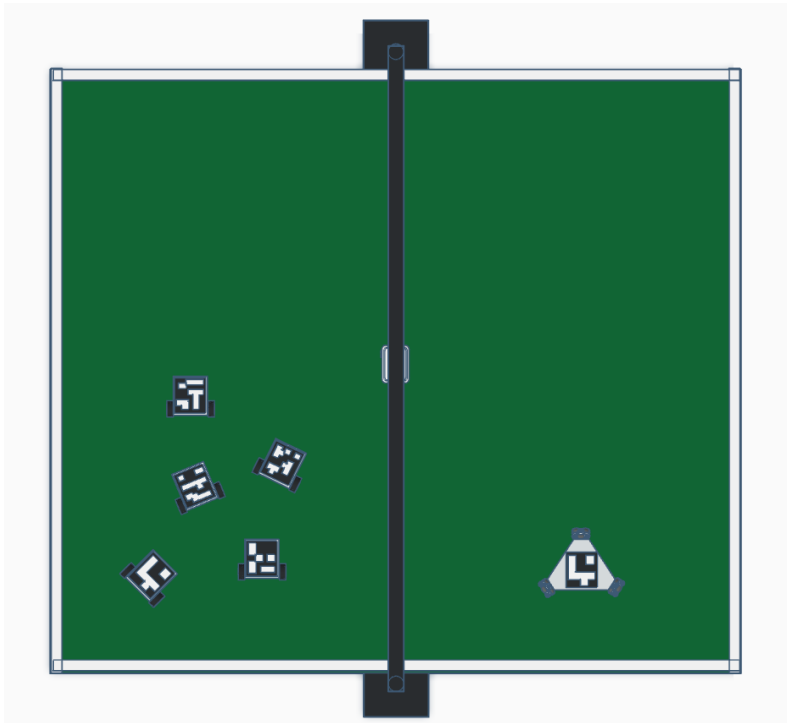
(b)


L298N Motor Driver
Motor Kanan
Motor Kiri
12 Volt
ESP 32 Dev Kit V1

(c)



(d)



(e)

Gambar 3. 4 Ilustrasi usulan rancangan sistem secara umum. (a) Proses cara kerja sistem, (b) Rangkaian robot beroda Omni-Directional 3 roda, (c) Rangkaian robot beroda Differential Drive 2 roda, (d) Gambaran sistem dalam tampilan 3D tampak samping, (e) Gambaran sistem dalam tampilan 3D tampak dari atas

3.2.2 Rencana Anggaran Desain 2

Tabel 3. 3 Rencana anggaran pengembangan sistem simulasi robot beroda

No	Item/Pengeluaran	Satuan	Harga satuan	Jumlah	Harga Total	Link Pembelian
1	ESP32 ESP-32 WIFI Bluetooth IoT ESP-32S Development Board	pcs	Rp 62,000.00	5	Rp 310,000.00	https://t.ly/bDZ3I

No	Item/Pengeluaran	Satuan	Harga satuan	Jumlah	Harga Total	Link Pembelian
2	L298N Dual Motor Driver Module L298 H-Bridge	pcs	Rp 15,350.00	7	Rp107,450.00	https://t.ly/mv8R0
3	[CNC] 40PCS JUMPER CABLE KABEL 20CM FEMALE TO FEMALE DUPONT	set	Rp 8,500.00	1	Rp 8,500.00	https://shorturl.at/bBHQU
4	[CNC] 40PCS JUMPER CABLE KABEL 10CM MALE TO MALE DUPONT	set	Rp 9,000.00	1	Rp 9,000.00	https://shorturl.at/bBHQU
5	[CNC] 40PCS JUMPER CABLE KABEL 10CM MALE TO FEMALE DUPONT	set	Rp 8,500.00	1	Rp 8,500.00	https://shorturl.at/gGZ19
6	WHEEL RODA RUBBER FOR SMART ROBOT CAR 4WD 2WD	pcs	Rp 5,000.00	12	Rp 60,000.00	https://shorturl.at/lmxGJ

No	Item/Pengeluaran	Satuan	Harga satuan	Jumlah	Harga Total	Link Pembelian
7	Holder Bracket Braket Breket 3 Sambungan Battery Baterai 18650 - 1solt	set	Rp 30,000.00	2	Rp 60,000.00	https://shorturl.at/lmHMZ
8	Nickel Strip Battery 18650 1 Meter/Strip Nikel Baterai Lithium	pcs	Rp 28,000.00	1	Rp 28,000.00	https://shorturl.at/ehruX
9	DINAMO MOTOR DC 12-18V 3.5A MOTOR DINAMO 12V 18V DC LS775 WITH CABLE	pcs	Rp 33,500.00	12	Rp 402,000.00	https://t.ly/BiMdV
10	Baterai Charger/Cas Ulang Li-ion FS 18650 13000mAh 4,2V Hijau Isi 2 pc	set	Rp 27,000.00	9	Rp 243,000.00	https://t.ly/qT1KD
11	PCB DOT MATRIX THRU HOLE SINGLE LAYER 5X7CM 5*7CM LUBANG BOLONG MATRIK	pcs	Rp 2,700.00	6	Rp 16,200.00	https://rb.gy/rhc8bc

No	Item/Pengeluaran	Satuan	Harga satuan	Jumlah	Harga Total	Link Pembelian
Total					Rp 1,262,650.00	

3.2.3 Analisis Risiko Desain 2

Penggunaan protokol MQTT sebagai alternatif untuk simulasi membawa risiko terkait integrasi dengan platform web. Kompatibilitas yang rumit dapat menyebabkan kesalahan komunikasi antara robot dan simulasi. Selain itu, risiko keamanan muncul dalam konteks MQTT jika tidak ada langkah-langkah keamanan yang memadai, seperti enkripsi data dan autentikasi. Kinerja sistem juga bisa dipengaruhi oleh kondisi jaringan tertentu, seperti latensi atau kehilangan paket. Terakhir, kompleksitas penggunaan menjadi risiko potensial, terutama bagi pengguna yang kurang berpengalaman dengan MQTT. Oleh karena itu, perlu disiapkan panduan pengguna yang jelas dan dukungan teknis untuk mengatasi potensi kesulitan pengguna.

3.2.4 Pengukuran Performa Usulan 2

Usulan solusi 2, yang melibatkan sistem simulasi virtual dan langsung dengan Message Queueing Telemetry Transport (MQTT), dapat dinilai melalui berbagai parameter performa. Responsivitas sistem menjadi fokus utama evaluasi, dengan penilaian terhadap keterlambatan pengiriman dan penerimaan pesan antara platform web dan robot fisik melalui protokol MQTT. Keunggulan utama MQTT dalam komunikasi ringan dan efisien diharapkan dapat menciptakan pengendalian robot yang responsif dan real-time.

Akurasi kontrol robot menjadi parameter penting, menilai kemampuan platform web untuk mengatur titik tujuan robot dan sejauh mana robot fisik dapat mereproduksi pergerakan yang diinginkan. Kemampuan simulasi 2D dari platform web juga menjadi fokus, dengan penilaian terhadap keakuratan peta lingkungan simulasi yang disajikan. Interaktivitas mahasiswa dalam mengatur titik tujuan dan memantau pergerakan robot secara virtual juga menjadi indikator keberhasilan solusi ini.

Melalui pengukuran performa pada parameter-parameter ini, dapat dievaluasi sejauh mana solusi 2 berhasil mencapai tujuan pengembangan sistem simulasi yang terintegrasi, mendukung pembelajaran pengendalian robot, dan memenuhi standar keteknikan yang diperlukan.

3.3 Usulan Solusi 3

Usulan solusi ini menyajikan sistem simulasi virtual dan langsung yang memanfaatkan protokol ROSbridge Suite, dengan perangkat keras yang diperbarui menggunakan mikrokontroler Arduino R4. Sistem ini mencakup pengembangan perangkat lunak simulasi virtual dan simulasi langsung pada robot fisik yang dilengkapi dengan differential drive dan omni-directional. Komponen perangkat keras mencakup mikrokontroler Arduino R4, motor driver L298N, motor DC, dan baterai 12V.

Mikrokontroler Arduino R4 dipilih sebagai otak utama perangkat keras karena menawarkan kemampuan pemrosesan 32-bit dengan inti Arm Cortex-M4. Mikrokontroler ini beroperasi pada tegangan 5V, memiliki 14 I/O digital, 6 input analog (dengan resolusi hingga 14-bit), kecepatan clock 48 MHz, SRAM 32 kB, memori flash 256 kB, dan 8 kB EEPROM. Keunggulan Arduino R4 terletak pada keandalan, kemudahan penggunaan, serta dukungan yang luas dari komunitas pengembang. Motor driver L298N dan motor DC tetap dipertahankan untuk menggerakkan differential drive dan omni-directional robot. Motor driver L298N memastikan kontrol yang stabil terhadap motor DC, sehingga pergerakan robot tetap akurat sesuai instruksi dari simulasi.

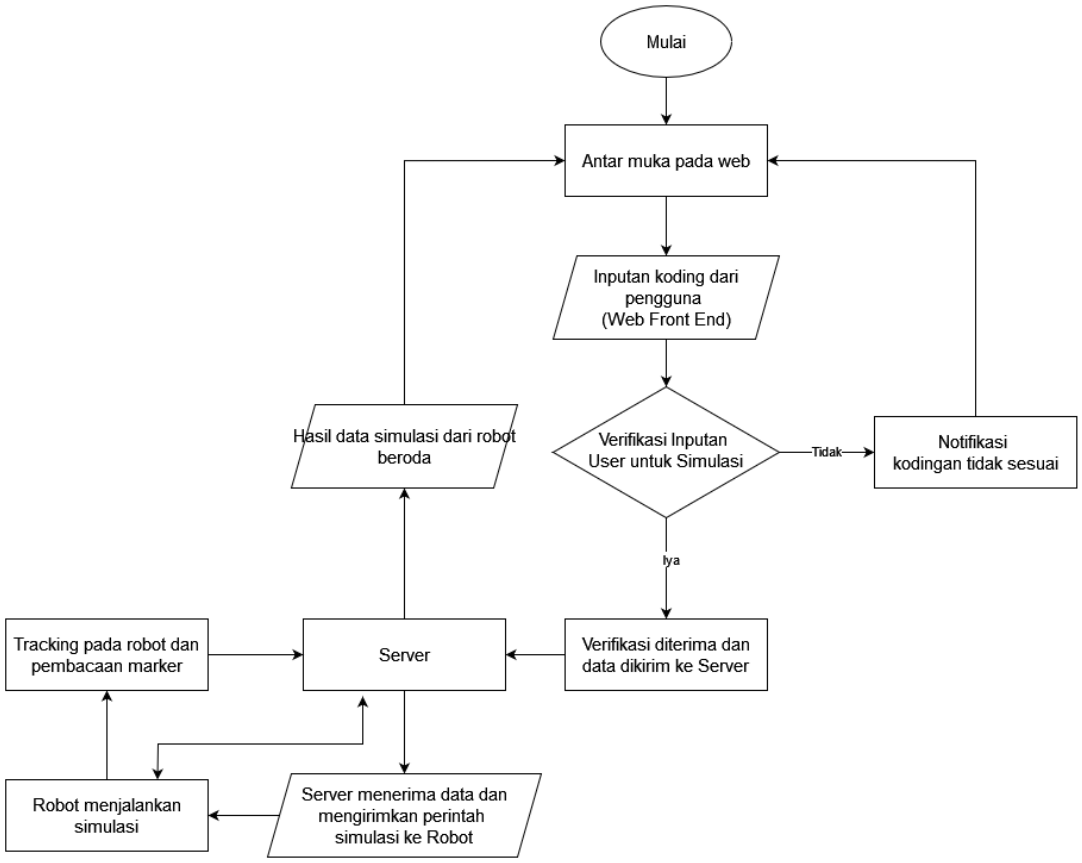
Solusi ini dipilih karena keunggulan ROSbridge Suite dalam menyediakan jembatan yang kuat antara ROS dan aplikasi eksternal melalui protokol JSON dan WebSocket. Integrasi dengan ROSbridge memungkinkan komunikasi efisien dan real-time antara platform web dan sistem ROS, sementara penggunaan WebSocket tetap memberikan responsivitas yang diperlukan untuk pengendalian robot secara langsung. Dengan demikian, solusi ini memberikan pengalaman belajar yang terintegrasi dan komprehensif, memungkinkan mahasiswa untuk merasakan pergerakan robot secara virtual dan langsung dengan baik.

Keunggulan ROSbridge juga memberikan fleksibilitas dalam mengelola aliran informasi, menciptakan pengalaman belajar yang mendalam dalam bidang pengendalian robot. Dengan integritas dan responsivitas yang tinggi, solusi ini memberikan mahasiswa kesempatan untuk mendalami konsep-konsep kontrol robot dengan lebih baik, mengoptimalkan pemahaman mereka melalui pengalaman simulasi yang terhubung secara langsung dengan sistem fisik robot.

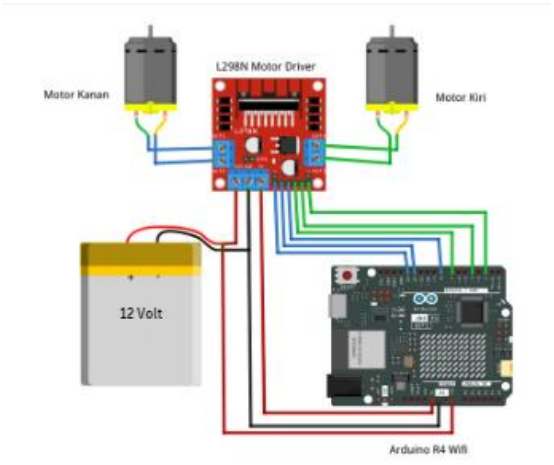
3.3.1 Desain Sistem 3

Gambar 3. 5 Diagram blok usulan solusi 3

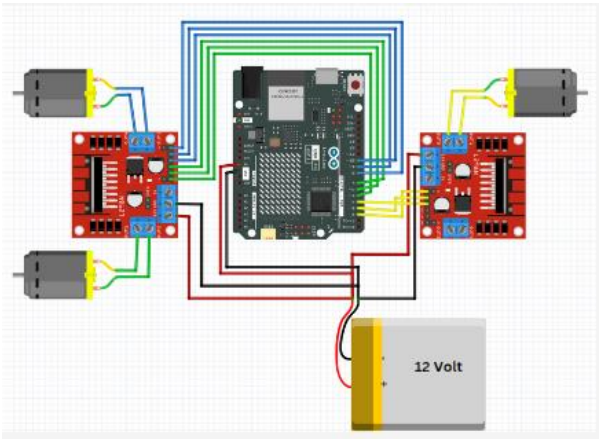
Simulasi pergerakan robot beroda dengan menggunakan ROS bridge Suite dimulai dengan mengakses robot beroda melalui sistem ROS. Robot differential drive dan omni-directional akan terhubung ke simulator melalui jembatan ROS Bridge. Proses ini memungkinkan pengguna untuk mengatur titik tujuan robot dan mengobservasi pergerakan robot secara virtual melalui platform web. Selanjutnya, sistem akan menyajikan peta lingkungan simulasi 2D yang mencerminkan kondisi lapangan nyata. Selama simulasi, komunikasi real-time antara platform web dan ROS memastikan responsifitas tinggi terhadap setiap perintah pengguna. Pengguna dapat melihat dokumentasi pergerakan robot, memahami karakteristik pergerakan dari jenis robot tertentu, dan menganalisis data yang dihasilkan oleh simulasi. Proses ini terus berulang dan bersifat iteratif, memungkinkan pengguna untuk terlibat dalam percobaan dan pembelajaran yang mendalam.


Mulai
Antar muka pada web
Inputan koding dari pengguna (Web Front End)
Verifikasi Inputan User untuk Simulasi
Tidak
Notifikasi kodingan tidak sesuai
Iya
Verifikasi diterima dan data dikirim ke Server
Server
Tracking pada robot dan pembacaan marker
Hasil data simulasi dari robot beroda
Server menerima data dan mengirimkan perintah simulasi ke Robot
Robot menjalankan simulasi

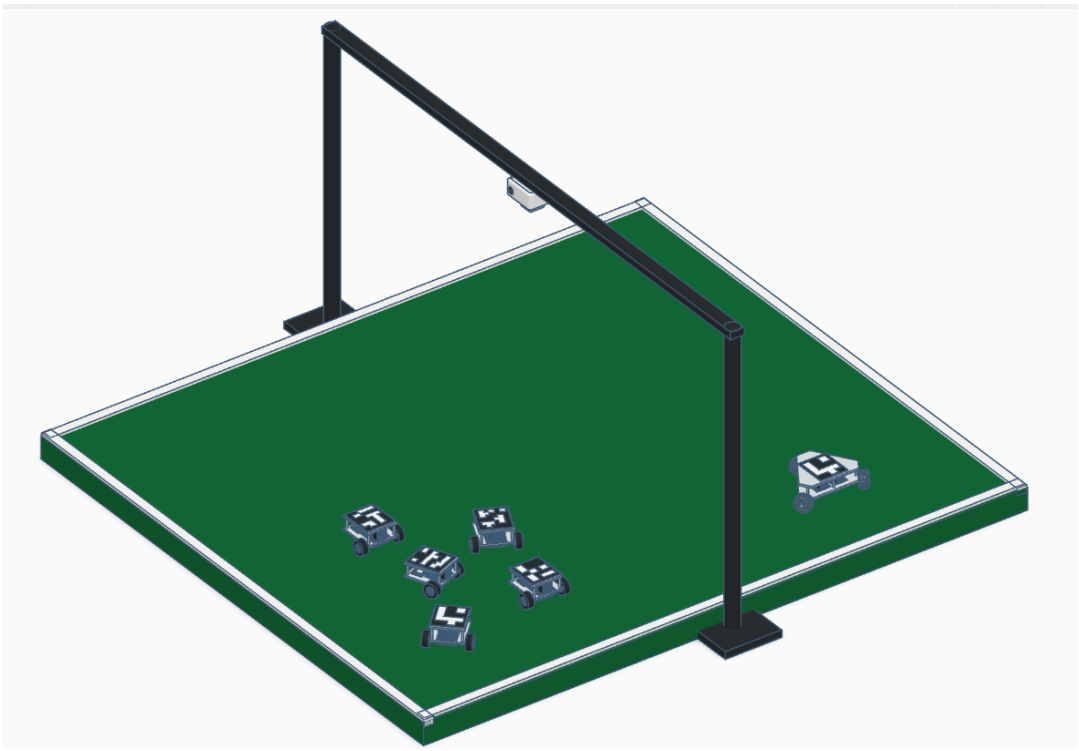
(a)


Motor Kanan
L298N Motor Driver
Motor Kiri
12 Volt
Arduino R4 Wifi

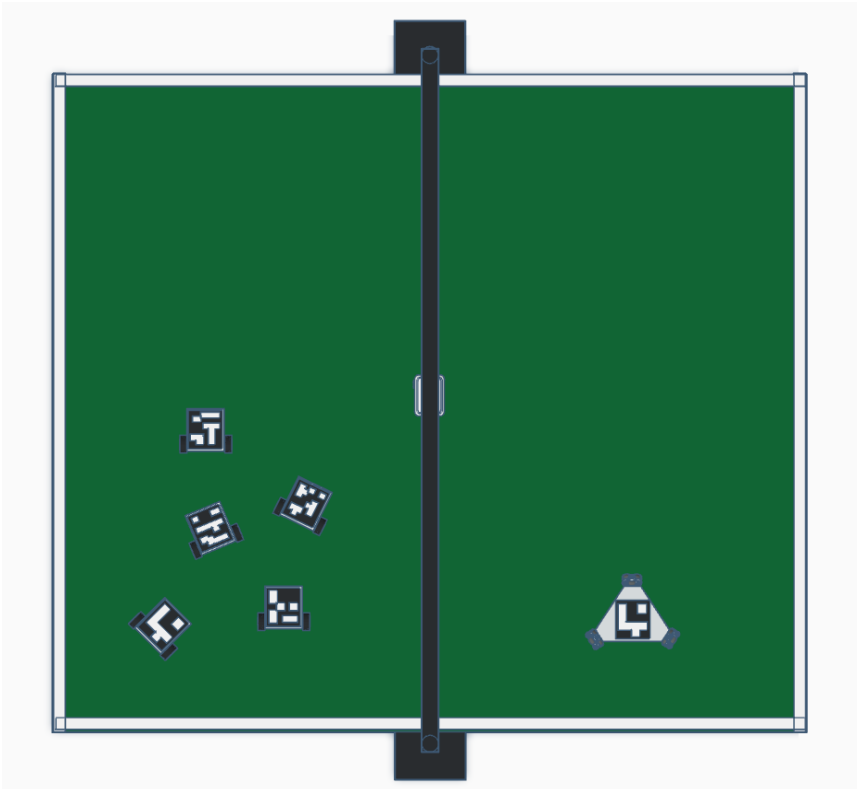
(b)


12 Volt

(c)



(d)



(e)

Gambar 3. 6 Ilustrasi usulan rancangan sistem secara umum. (a) Proses cara kerja sistem, (b) Rangkaian robot beroda Differential Drive 2 roda, (c) Rangkaian robot beroda Omni-Directional 3 roda, (d) Gambaran sistem dalam tampilan 3D tampak samping, (e) Gambaran sistem dalam tampilan 3D tampak dari atas

Untuk dapat memenuhi usulan sistem tersebut, maka diperlukan inventarisasi kebutuhan sistem perangkat keras. Tabel 3.1 memperlihatkan kebutuhan sistem sesuai usulan dan spesifikasi yang dibutuhkan.

Tabel 3. 4 Inventarisasi kebutuhan usulan sistem perangkat keras

No	Nama Alat	Keterangan
1	Robot beroda Differential	Robot dengan dua roda yang dapat berputar secara independen, mendukung pergerakan differential drive. Berperan sebagai perangkat aktif dalam simulasi.
2	Robot beroda Omni-Directional	Robot dengan roda khusus yang memungkinkan pergerakan dalam berbagai arah tanpa perlu berputar secara fisik. Berperan sebagai perangkat aktif dalam simulasi.
3	Mikroprosessor	Komponen utama yang menjalankan logika dan pengolahan data pada simulator. Disini mikroprosesor yang digunakan ialah Arduino R4 untuk pengiriman data.
4	Marker	Penanda visual yang digunakan dalam identifikasi robot pada simulasi. Disini menggunakan ArUco marker sebagai penanda yang akan digunakan pada robot untuk simulasi
5	Kamera atau sensor penginderaan	Kamera yang digunakan untuk pembacaan dari marker yang terpasang pada robot sehingga dapat mendapatkan informasi visual akan lapangan robot beroda.
6	Sistem Koneksi Wi-fi atau Ethernet	Infrastruktur untuk menghubungkan simulator dengan web dan serta komunikasi real-time antara web dan robot.
7.	Baterai	Baterai merupakan sumber daya energi yang digunakan untuk dapat menggerakan robot ataupun perangkat lain. Baterai disini digunakan sebagai sumber daya mikroprosesor ESP32 dan motor

Dikarenakan sistem simulasi robot beroda ini tidak hanya mengandalkan perangkat keras, namun juga memanfaatkan perangkat lunak untuk memberikan pengalaman belajar yang lebih baik, maka dalam perancangan ini, kami menyertakan aplikasi berbasis web yang dapat diakses

oleh pengguna melalui browser. Aplikasi web ini dirancang untuk memberikan antarmuka yang intuitif dan mudah digunakan, sehingga mahasiswa dapat dengan nyaman menjalankan simulasi robot beroda, mengatur pergerakan, dan memahami konsep-konsep pengendalian robot secara langsung melalui website. Desain antarmuka web ini disusun untuk mendukung keberlanjutan penggunaan, memastikan kesesuaian dengan berbagai perangkat yang terhubung dengan internet, dan menyediakan pengalaman belajar yang responsif dan terintegrasi.

3.3.2 Rencana Anggaran Desain 3

Tabel 3. 5 Rencana anggaran pengembangan sistem simulasi robot beroda

No	Item/Pengeluaran	Satuan	Harga satuan	Jumlah	Harga Total	Link Pembelian
1	Arduino UNO R4 Wifi Original Made In Italy	pcs	Rp 493,750.00	6	Rp 2,962,500.00	https://t.ly/AhWaF
2	L298N Dual Motor Driver Module L298 H-Bridge	pcs	Rp 15,350.00	7	Rp107,450.00	https://t.ly/mv8R0
3	[CNC] 40PCS JUMPER CABLE KABEL 20CM FEMALE TO FEMALE DUPONT	set	Rp 8,500.00	1	Rp 8,500.00	https://shorturl.at/bBHQU
4	[CNC] 40PCS JUMPER CABLE KABEL 10CM	set	Rp 9,000.00	1	Rp 9,000.00	https://shorturl.at/bBHQU

No	Item/Pengeluaran	Satuan	Harga satuan	Jumlah	Harga Total	Link Pembelian
	MALE TO MALE DUPONT					
5	[CNC] 40PCS JUMPER CABLE KABEL 10CM MALE TO FEMALE DUPONT	set	Rp 8,500.00	1	Rp 8,500.00	https://shorturl.at/gGZ19
6	WHEEL RODA RUBBER FOR SMART ROBOT CAR 4WD 2WD	pcs	Rp 5,000.00	12	Rp 60,000.00	https://shorturl.at/lmxGJ
7	Holder Bracket Braket Breket 3 Sambungan Battery Baterai 18650 - 1solt	set	Rp 30,000.00	2	Rp 60,000.00	https://shorturl.at/lmHMZ
8	Nickel Strip Battery 18650 1 Meter/Strip Nikel Baterai Lithium	pcs	Rp 28,000.00	1	Rp 28,000.00	https://shorturl.at/ehruX
9	DINAMO MOTOR DC 12-18V 3.5A MOTOR DINAMO	pcs	Rp 33,500.00	12	Rp 402,000.00	https://t.ly/BiMdV

No	Item/Pengeluaran	Satuan	Harga satuan	Jumlah	Harga Total	Link Pembelian
	12V 18V DC LS775 WITH CABLE					
10	Baterai Charger/Cas Ulang Li-ion FS 18650 13000mAh 4,2V Hijau Isi 2 pc	set	Rp 27,000.00	9	Rp 243,000.00	https://t.ly/qT1KD
11	PCB DOT MATRIX THRU HOLE SINGLE LAYER 5X7CM 5*7CM LUBANG BOLONG MATRIK	pcs	Rp 2,700.00	6	Rp 16,200.00	https://rb.gy/rhc8bc
Total					Rp 3,907,150.00	

3.3.3 Analisis Risiko Desain

Implementasi ROS Bridge Suite sebagai protokol komunikasi utama dalam simulasi memiliki risiko yang perlu diperhatikan. Ketergantungan yang signifikan pada ekosistem ROS dapat menyebabkan kerentanannya terhadap perubahan atau masalah yang mungkin terjadi pada ROS. Diperlukan pemantauan kontinu terhadap pembaruan dan perubahan dalam ekosistem ROS untuk memastikan kompatibilitas yang berkelanjutan. Selain itu, risiko koneksi internet yang tidak stabil menjadi pertimbangan penting, mengingat simulasi memerlukan akses online untuk berfungsi. Solusi mitigasi dapat mencakup penyediaan opsi simulasi offline atau implementasi mekanisme cadangan untuk mengatasi potensi gangguan koneksi.

Kesulitan integrasi dengan hardware fisik, seperti robot beroda dan lapangan nyata, juga merupakan risiko yang harus diperhitungkan. Konfigurasi yang diperlukan mungkin

membutuhkan keahlian teknis tertentu, dan perlu mempertimbangkan upaya tambahan yang mungkin diperlukan untuk mengatasi kendala teknis tersebut. Selain itu, ada risiko bahwa simulasi mungkin tidak sepenuhnya memenuhi kebutuhan pembelajaran mahasiswa, dan aspek pergerakan robot mungkin tidak dapat diterapkan secara realistis. Perlu dilakukan evaluasi reguler terhadap respons dan kebutuhan pengguna untuk memastikan bahwa simulasi tetap relevan dan efektif dalam mendukung pembelajaran.

Penggunaan mikrokontroler Arduino R4 sebagai pengganti mikrokontroler ESP32 juga membawa risiko tersendiri. Arduino R4 merupakan mikrokontroler yang relatif baru, dan dokumentasinya masih terbatas, yang dapat menyulitkan pengembangan sistem. Pemilihan alternatif mikrokontroler yang lebih mapan dan terdokumentasi dengan baik juga dapat menjadi opsi untuk mengurangi risiko ini.

3.3.4 Pengukuran Performa

Pengukuran performa pada usulan solusi ini dilakukan melalui beberapa parameter kinerja yang krusial. Responsivitas sistem menjadi fokus pertama, dengan penekanan pada waktu respons sistem terhadap perintah pengguna di simulasi virtual maupun robot fisik. Evaluasi terhadap akurasi gerakan robot juga menjadi aspek penting, mencakup sejauh mana robot fisik dapat mereproduksi gerakan dari simulasi virtual, dan memperhitungkan deviasi antara pergerakan yang diinginkan dan yang terealisasi.

Stabilitas kontrol motor, khususnya dalam konteks penggunaan motor driver L298N pada differential drive dan omni-directional robot, menjadi parameter lain yang diukur. Evaluasi mencakup keakuratan dalam mengubah kecepatan dan arah gerakan robot. Efisiensi penggunaan sumber daya, termasuk kemampuan mikrokontroler Arduino R4 dalam memproses instruksi dan menjalankan tugas kontrol robot, menjadi pertimbangan penting dengan memantau konsumsi daya sistem.

3.4 Analisis dan Penentuan Usulan Solusi/Desain Terbaik

Tabel 3. 6 Decision Matriks usulan solusi

Kriteria	Bobot	Usulan Solusi 1	Usulan Solusi 2	Usulan Solusi 3

Kecepatan Respons	4	4	3	4
Kompatibilitas	3	4	3	3
Keamanan	4	3	2	3
Kemudahan Pengguna	3	2	4	1
Efisiensi	3	3	3	3
Skor Total		55	50	49

Berdasarkan analisis dan pertimbangan dari ketiga usulan solusi yang kami berikan. Kami memilih usulan solusi pertama, yaitu sistem yang menggunakan protokol ROS Bridge, untuk simulasi robot beroda berdasarkan evaluasi yang kami lakukan. Keputusan ini didasarkan pula pada nilai tertinggi dari Decision Matrix yang kami gunakan. ROS Bridge memiliki kecepatan respons yang baik dan tingkat keamanan yang tinggi, membuatnya menjadi pilihan terbaik. Keunggulan ROS Bridge terletak pada cara efisiennya dalam menghubungkan robot dengan aplikasi eksternal, memungkinkan kontrol robot secara langsung dan responsif. Selain itu, Mikrokontroler ESP32 dipilih sebagai otak utama perangkat keras karena menawarkan dua inti pemrosesan (dual-core) yang dapat dioptimalkan untuk menjalankan tugas komputasi dan mengelola komunikasi dengan sistem ROS melalui koneksi WiFi. Keunggulan ESP32 terletak pada konektivitas nirkabelnya yang dapat diandalkan, mendukung interaksi real-time antara simulasi dan robot fisik, serta referensinya yang banyak di internet.

3.5 Gantt *Chart*

Perencanaan disini meliputi ketiga tahapan dalam perancangan sistem keteknikan dan dilaksanakan selama 2 semester (Tugas Akhir 1 dan Tugas Akhir 2) menggunakan *Gantt chart* seperti pada Tabel 3.7 berikut.

Tabel 3. 7 *Gantt chart* pelaksanaan *Capstone Project* simulator robot beroda

No.	Kegiatan/Capaian	Tahun										
		2023				2024						
		sep	okt	nov	des	jan	feb	mar	apr	mei	jun	jul
1	Survei dan identifikasi permasalahan	D,R	D,R									
2	Mencari literatur dan informasi untuk kebutuhan dan spesifikasi sistem		D,R									
3	Penentuan usulan solusi dari hasil studi literatur		D,R									
	Pembuatan rancangan 3d dan rangkaian elektronis dari ide yang ada			D,R								
4	Mengumpulkan seluruh ide solusi dan finalisasi usulan perancangan sistem beserta manajemen dan rancangan belanja			D,R	D,R							
5	Pengumpulan proposal Tugas Akhir 1/*Capstone Project* dan seminar				D,R							
6	Pembelian kebutuhan alat dan bahan					D	D					
7	Perakitan komponen perangkat keras						D,R					
8	Pembuatan koding program untuk menghubungkan esp32 dan robot simulasi						R	R				
9	Pembuatan robot simulasi differential drive dan omni-						D	D				

No.	Kegiatan/Capaian	Tahun										
		2023				2024						
		sep	okt	nov	des	jan	feb	mar	apr	mei	jun	jul
	directional sesuai dengan rancangan											
10	Pengembangan web server dengan protokol websocket								R	R		
11	Integrasi ROS bridge dengan simulasi robot								D	D	D	
12	Pengembangan antarmuka pengguna serta integrasi sistem dengan simulasi								D,R	D,R	D,R	
13	Uji Coba dan Debugging									D,R	D,R	
14	Expo dan pengumpulan laporan akhir											D,R

Ket. : PIC – *Person in Charge*, D : Dimas, R : Rifqi

3.6 Realisasi Pelaksanaan Tugas Akhir 1

Tabel 3. 8 Realisasi aktivitas pelaksanaan tugas akhir 1

No	Hari, Tanggal, Durasi (jam atau hari)	Aktivitas	Pelaksana
1	Senin, 18 September 2023, 16.30	Bimbingan pertama (Pembahasan template & timeline)	Dosen Pembimbing 1
2	Selasa, 19 September 2023, 3 jam	Studi literatur jurnal *project* terkait	Dimas Rifqi

3	Rabu, 20 September 2023, 2 jam	Studi literatur jurnal *project* terkait	Dimas Rifqi
4	Jumat, 22 September 2023, 3 Jam	Penyusunan BAB 1 Latar Belakang & Pendahuluan	Dimas Rifqi
5	Sabtu, 23 September 2023, 5 jam	Penyusunan BAB 1 Latar Belakang & Pendahuluan	Dimas Rifqi
6	Selasa, 26 September 2023,	Bimbingan kedua (Pembahasan *project* secara garis besar)	Dosen Pembimbing 1
7	Rabu, 27 September 2023, 2 jam	Penyusunan BAB 1 Latar Belakang & Pendahuluan	Dimas Rifqi
8	Selasa, 3 Oktober 2023, 14.30	Bimbingan ke-3 (Pembahasan latar belakang dan pendahuluan)	Dosen Pembimbing 1
9	Rabu, 11 Oktober 2023, 2 jam	Penyusunan BAB 1 (Tujuan, rumusan masalah, batasan masalah)	Dimas Rifqi
10	Jumat, 13 Oktober 2023, 2 jam	Bimbingan ke - 4	Dosen Pembimbing 1
11	Sabtu, 14 Oktober 2023, 4 jam	Penyusunan BAB 1 (Tujuan, rumusan masalah, batasan masalah dan Batasan	Dimas Rifqi

		masalah aspek keteknikan)	
12	Selasa, 18 Oktober 2023, 3 jam	Revisi BAB 1 dan Penyusunan BAB 2 (Studi literatur)	Dimas Rifqi
13	Jumat, 20 Oktober 2023, 2 jam	Penyusunan BAB 2 (Studi literatur & Observasi)	Dimas Rifqi
14	Sabtu, 21 oktober 2023. 4 jam	Penyusunan BAB 2 (Dasar Teori)	Dimas Rifqi
15	Selasa, 24 Oktober 2023, 16.30	Bimbingan ke - 5 (zoom)	Dosen Pembimbing 1
16	Rabu, 25 oktober 2023, 3 jam	Penyusunan BAB 2 (Dasar Teori)	Dimas Rifqi
17	Sabtu, 27 Oktober 2023, 1 jam	Penyusunan BAB 2 (Analisis Stakeholder, dan Ananlisis Aspek yang Mempengaruhi Sistem)	Dimas Rifqi
18	Senin, 30 Oktober 2023, 1 jam	Penyusunan BAB 2 (Analisis aspek yang mempengaruhi sistem)	Dimas Rifqi
19	Jumat, 3 November 2023, 2 jam	Penyusunan BAB 2 (Spesifikasi Sistem)	Dimas Rifqi
20	Sabtu, 4 November 2023, 2 jam	Penyusunan BAB 2 (Spesifikasi Sistem)	Dimas Rifqi

21	Jumat, 10 November 2023, 2 jam	Revisi Bab 2 dan Penyusunan BAB 3 (Usulan Solusi 1)	Dimas Rifqi
22	Sabtu, 11 November 2023, 2 jam	Penyusunan BAB 3 (Usulan Solusi 1)	Dimas Rifqi
23	Jumat, 17 November 2023, 2 jam	Penyusunan BAB 3 (Usulan Solusi 2)	Dimas Rifqi
24	Sabtu, 18 November 2023, 2 jam	Penyusunan BAB 3 (Usulan Solusi 2)	Dimas Rifqi
25	Sabtu, 25 November 2023, 2 jam	Revisi BAB 2 (Spesifikasi Sistem)	Dimas Rifqi
26	Minggu, 3 Desember 2023, 4 jam	Penyusunan BAB 3 (Desain sistem 1 dan 2, Analisis Risiko Desain 1 dan 2, serta Pengukuran Performa 1 dan 2)	Dimas Rifqi
27	Rabu, 6 Desember 2023, 16.00	Bimbingan ke 9	Dosen pembimbing 1
28	Jumat, 8 Desember 2023, 5 jam	Revisi bab 3	Dimas Rifqi
29	Sabtu, 9 Desember 2023, 4 jam	Penyusunan BAB 3 (Usulan Solusi 3, Desain Sistem, Analisis Risiko Desain, Pengukuran Performa)	Dimas Rifqi
30	Minggu, 10 Desember 2023, 7 jam	Penyusunan BAB 3 (Diagram Blok,	Dimas Rifqi

		Rangkaian Elektronis, Desain 3D, Proses Kerja Sistem untuk usulan 1, 2, dan 3)	
31	Selasa, 12 Desember 2023, 4 jam	Penyusunan BAB 3 (Diagram Blok, Rangkaian Elektronis, Desain 3D, Proses Kerja Sistem untuk usulan 1, 2, dan 3)	Dimas Rifqi
32	Rabu, 13 Desember 2023, 14.00	Bimbingan ke 10	Dimas Rifqi
33	Jumat, 15 Desember 2023, 2 jam	Penyusunan BAB 3 (Gantt Chart)	Dimas Rifqi

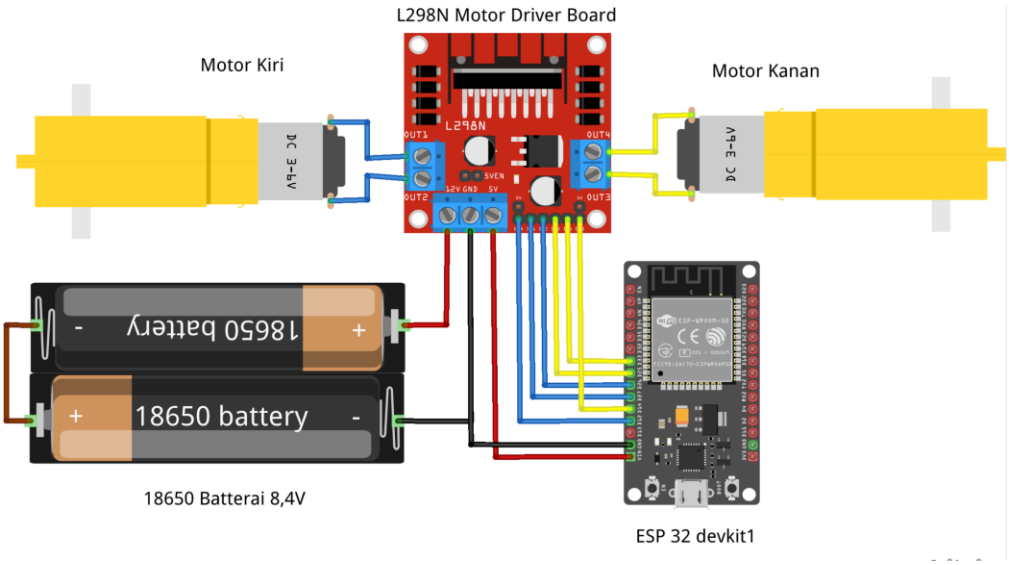
BAB 4. HASIL RANCANGAN DAN METODE PENGUKURAN

4.1 Hasil Perancangan Sistem

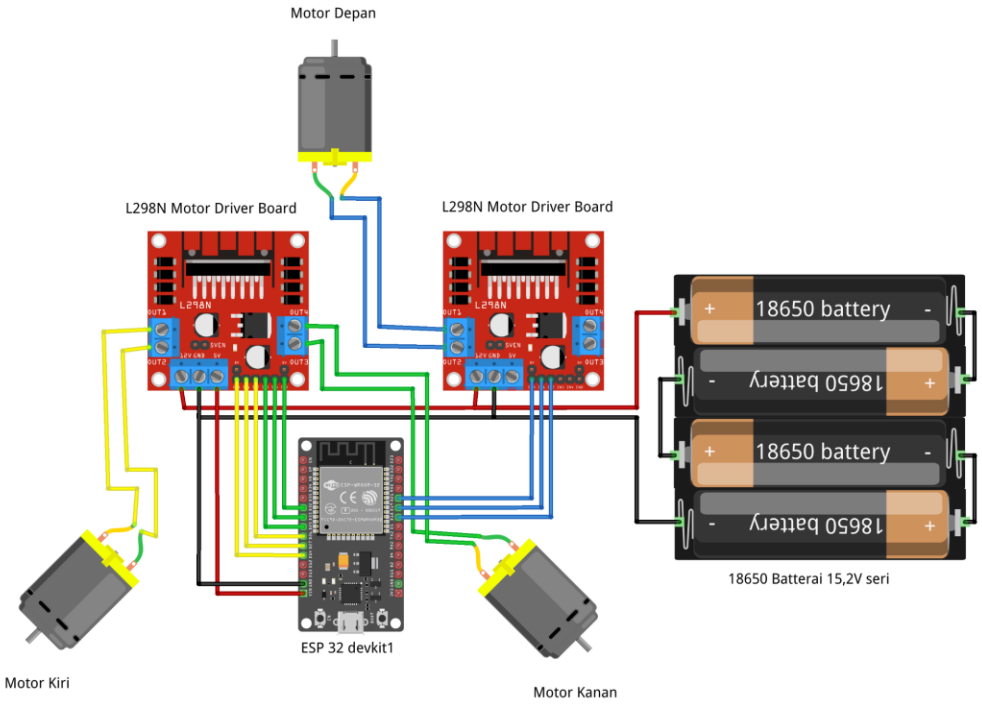
4.1.1 Rangkaian elektronik

Desain rangkaian elektronik yang telah kami buat mengalami sedikit modifikasi pada bagian baterai. Pada desain awal, kami menggunakan baterai standar 12 V untuk semua robot simulasi, namun dalam implementasinya, kami menggantinya dengan baterai Li-ion 18650. Baterai ini terbukti optimal dan memberikan daya yang cukup untuk operasional robot.

Robot dua roda dengan sistem differential drive menggunakan baterai Li-ion 18650 dengan besar daya 8,4V, sementara robot tiga roda dengan sistem omnidirectional menggunakan baterai Li-ion 18650 dengan besar daya 15,2V yang sama-sama dirangkai seri. Selain perubahan pada baterai, desain rangkaian elektronis yang kami gunakan tetap sama seperti sebelumnya. Desain tersebut dapat dilihat pada Gambar 4.1 dan Gambar 4.2



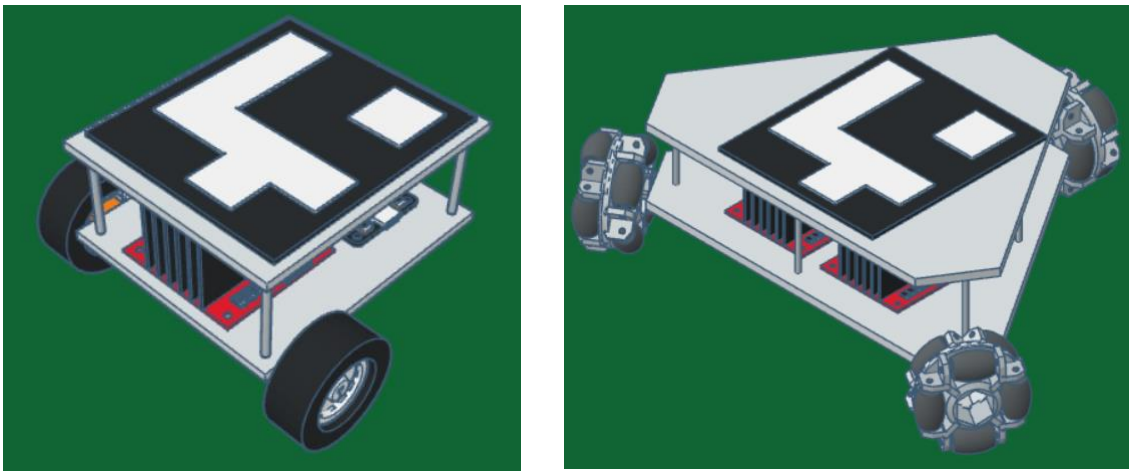
Gambar 4. 1 Rangkaian elektronik untuk robot 2 roda differential drive



Gambar 4. 2 Rangkaian elektronik untuk robot 3 roda Omni directional

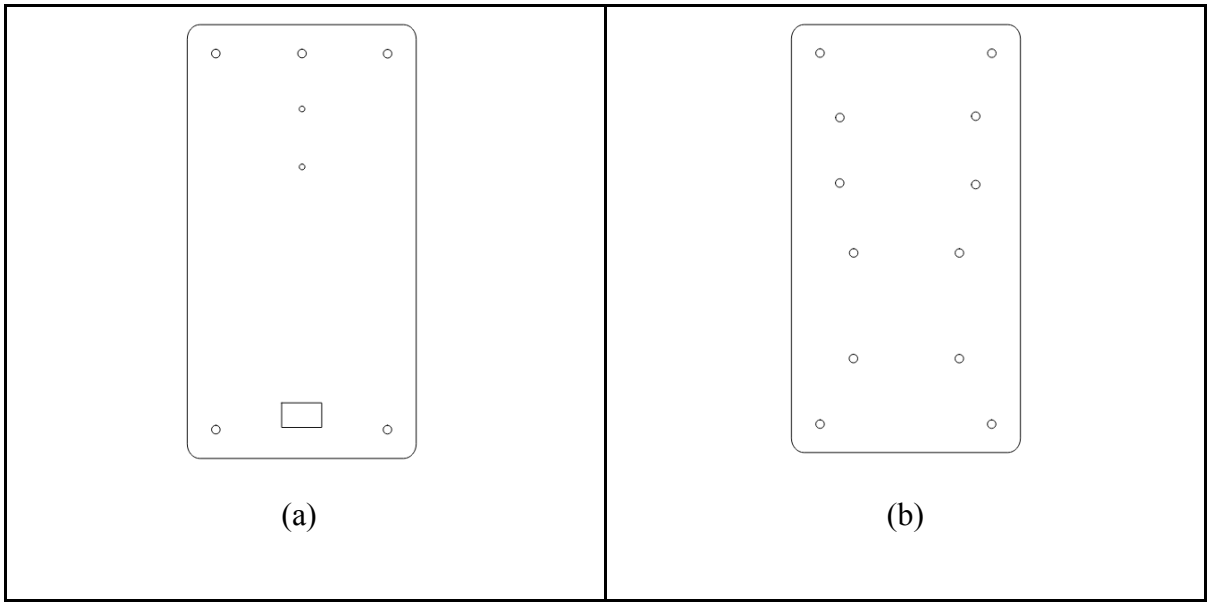
4.1.2 Gambar desain tiga dimensi (3D)

Dalam proses pembuatan robot dua roda dan tiga roda, kami mengalami beberapa penyesuaian dan sedikit kendala. Desain dasar dari robot yang kami buat tetap menggunakan akrilik, sesuai dengan perancangan awal. Meskipun demikian, terdapat sedikit penyesuaian pada letak komponen-komponen yang digunakan di kedua robot yang dibuat. Desain Akhir dari robot yang dibuat masih sama dengan yang sebelumnya seperti pada Gambar 4.3 dan Gambar 4.4

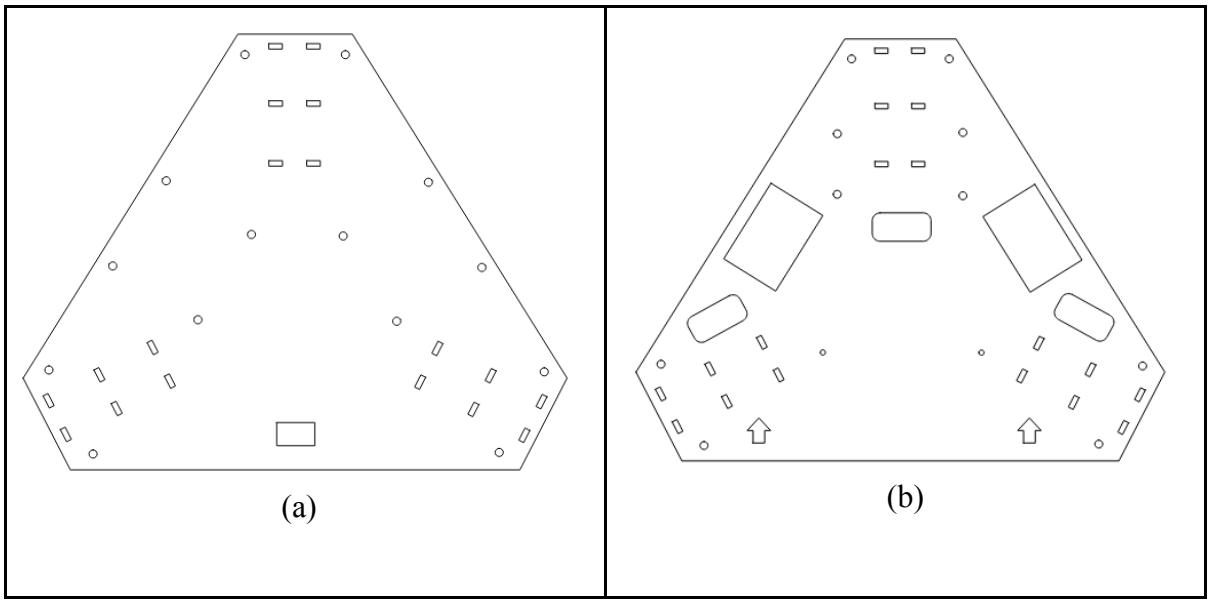


Gambar 4. 3 Desain 3d robot differential drive dan omni-directional

Penyesuaian yang kami lakukan pada robot yang dibuat adalah menyesuaikan ukuran dan posisi lubang pada akrilik untuk memastikan bahwa motor dan ban dapat terpasang dengan presisi yang tepat beserta dengan komponen-komponen yang lain, sehingga robot dapat beroperasi dengan stabil dan efisien.



Gambar 4. 4 Desain Akrilik pada robot Differential Drive 2 roda. (a) Layer pertama atau dasar, (b) Layer kedua

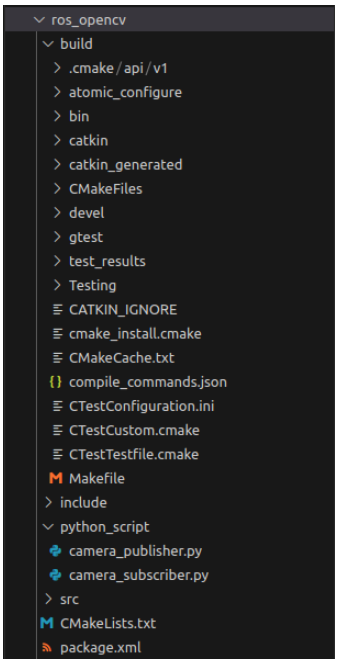


Gambar 4. 5 Desain Akrilik pada robot Omni directional 3 roda. (a) Layer pertama atau dasar, (b) Layer kedua

4.1.3 Software atau interface

4.1.3.1 Struktur Paket “ros_opencv”

Struktur paket proyek ROS (Robot Operating System) yang menggunakan OpenCV untuk pemrosesan gambar dapat dilihat pada Visual Studio Code seperti pada gambar yang diberikan.



Gambar 4. 6 Struktur paket "ros_opencv"

Paket ini terdiri dari beberapa direktori dan file yang diatur secara sistematis untuk mendukung pengembangan dan implementasi proyek. Direktori build berisi file dan folder yang dihasilkan selama proses build proyek, termasuk konfigurasi dan file yang diperlukan untuk membangun proyek. Direktori dan file terkait seperti `cmake/api/v1`, `atomic_configure`, `bin`, `catkin`, `catkin_generated`, `CMakeFiles`, `devel`, `gtest`, `test_results`, dan Testing adalah bagian dari proses build dan konfigurasi yang dikelola oleh CMake, sebuah alat yang digunakan untuk mengelola proses build perangkat lunak.

File-file seperti `CMakeCache.txt`, `compile_commands.json`, `CTestConfiguration.ini`, `CTestCustom.cmake`, dan `CTestTestfile.cmake` digunakan oleh CMake untuk konfigurasi dan pengujian unit. Makefile adalah bagian dari sistem build Make yang dihasilkan oleh CMake untuk memudahkan kompilasi proyek. Direktori include biasanya berisi file header (.h) yang mendeklarasikan fungsi, kelas, dan variabel yang dapat digunakan di berbagai bagian proyek.

Dalam direktori python_script, terdapat skrip Python yang digunakan untuk komunikasi ROS. Skrip `camera_publisher.py` bertanggung jawab untuk mengambil gambar dari kamera dan menerbitkannya sebagai topik ROS yang akan digunakan dalam

sistem yang kami buat. Skrip kode lengkap “camera_publisher” terdapat pada lampiran nomor 1. Beberapa bagian kode yang penting adalah sebagai berikut:

Kode ini menginisialisasi node ROS dengan nama '`camera_sensor_publisher`' dan membuat publisher untuk mengirimkan gambar dari kamera melalui topik 'video_topic'.

```
rospy.init_node('camera_sensor_publisher', anonymous=True)
```

Kode ini menggunakan OpenCV untuk menangkap video dari kamera dengan ID yang sesuai (dalam contoh ini, kamera dengan ID 2). Objek CvBridge digunakan untuk mengkonversi frame dari OpenCV menjadi pesan ROS Image.

```
cap = cv2.VideoCapture(2)  # Ganti dengan ID kamera yang sesuai
    bridge = CvBridge()
```

Kode ini berjalan dalam loop utama. Setiap iterasi, frame dari kamera dibaca menggunakan `cap.read()`, kemudian dikonversi menjadi pesan ROS Image dengan `bridge .cv2_imgsmg()`. Pesan ini kemudian dipublikasikan melalui publisher yang telah dibuat `(pub.publish(msg))`. Frekuensi loop diatur menggunakan `rate.sleep()` untuk memastikan loop berjalan pada 30 Hz.

```
while not rospy.is_shutdown():
        ret, frame = cap.read()
        if not ret:
            continue

        # Konversi frame ke ROS Image message
        msg = bridge.cv2_to_imgmsg(frame, encoding="bgr8")
        pub.publish(msg)
        rospy.loginfo("Video frame captured and published")
        rate.sleep()
```

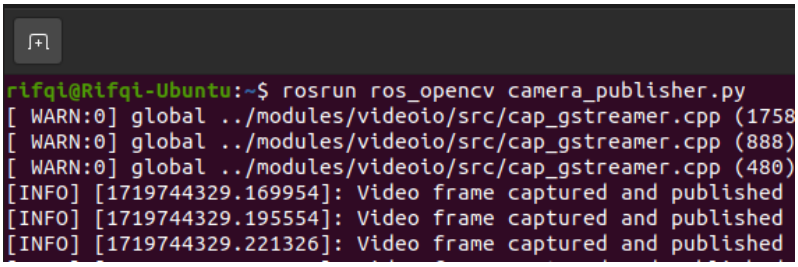
File `CMakeLists.txt` adalah file konfigurasi utama untuk CMake, mendefinisikan bagaimana proyek harus dibangun, termasuk dependensi dan pengaturan build lainnya. `package.xml` berisi metadata tentang paket ROS, seperti nama paket, versi, deskripsi,

dependensi, dan informasi penulis, yang penting untuk manajemen paket dalam ekosistem ROS.

Secara keseluruhan, struktur paket ini mencerminkan praktik terbaik dalam pengembangan perangkat lunak dengan ROS dan OpenCV, memisahkan berbagai aspek proyek seperti konfigurasi build, kode sumber, dan skrip Python, serta memastikan bahwa semua bagian proyek terorganisir dengan baik untuk memudahkan pengelolaan dan pemeliharaan.

4.1.3.2 Paket "ros_opencv"

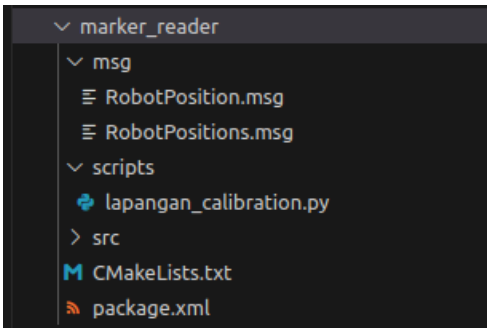
Paket ini digunakan untuk mengintegrasikan ROS dengan OpenCV, sebuah pustaka pengolahan citra yang populer. Dalam sistem simulasi robot ini, `ros_opencv` berfungsi untuk membaca dan memproses gambar dari kamera eksternal (Bpro5) yang digunakan untuk mendeteksi marker ArUco. Dengan file utama `camera_publisher.py`, paket ini mengambil gambar dari kamera dan mempublikasikannya sebagai topic di ROS, memungkinkan data gambar tersebut digunakan oleh komponen lain dalam sistem. Untuk menjalankan skrip `camera_publisher.py`, gunakan perintah "`rosrun ros_opencv camera_publisher.py`". Ini akan memulai proses pengambilan gambar dari kamera eksternal dan mempublikasikannya sebagai topic di ROS. Saat menjalankan perintah tersebut akan tampak seperti pada gambar berikut:



Gambar 4. 7 Tampilan terminal ubuntu ketika menjalankan paket ros_opencv

4.1.3.3 Struktur Paket "marker_reader"

Direktori marker_reader dalam proyek ROS yang terlihat pada Visual Studio Code mencakup beberapa subdirektori dan file yang diatur untuk mendukung pembacaan marker ArUco dan pengiriman informasi posisi robot.



Gambar 4. 8 Struktur paket "marker_reader"

Di dalam direktori ini, terdapat subdirektori msg yang berisi file `RobotPosition.msg` dan `RobotPositions.msg`. Kedua file ini mendefinisikan pesan khusus yang digunakan untuk komunikasi antar node dalam ROS, di mana RobotPosition.msg mendefinisikan posisi individu robot, sementara `RobotPositions.msg` mendefinisikan daftar posisi beberapa robot. Selain itu, terdapat subdirektori skrip yang berisi skrip `lapangan_calibration.py`, yang digunakan untuk kalibrasi lapangan dan pengaturan parameter kamera serta marker ArUco untuk memastikan akurasi pembacaan posisi robot. Skrip kode lengkap "camera_publisher" terdapat pada lampiran nomor 2. Beberapa bagian kode yang penting adalah sebagai berikut:

Bagian kode yang menunjukkan inisialisasi node ROS dan subscriber untuk menerima gambar dari topik video yang dipublikasikan.

```
rospy.init_node('lapangan_calibration', anonymous=True)  # Inisialisasi node ROS

self.image_sub = rospy.Subscriber("video_topic", Image, self.image_callback)  # Subscriber untuk topik video
```

Bagian kode yang berfungsi untuk mendeteksi marker ArUco pada gambar grayscale dan menghitung posisi 3D relatif (rotasi dan translasi) dari setiap marker.

```
corners, ids, rejected = aruco.detectMarkers(gray, self.aruco_dict, parameters=self.parameters)  # Deteksi marker ArUco

rvec, tvec, _ = aruco.estimatePoseSingleMarkers(corners[i], 0.05, self.camera_matrix, self.dist_coeffs)  # Estimasi pose marker
```

Bagian kode yang berfungsi menghitung posisi dan orientasi robot berdasarkan posisi sudut marker dan vektor rotasi dari estimasi pose.

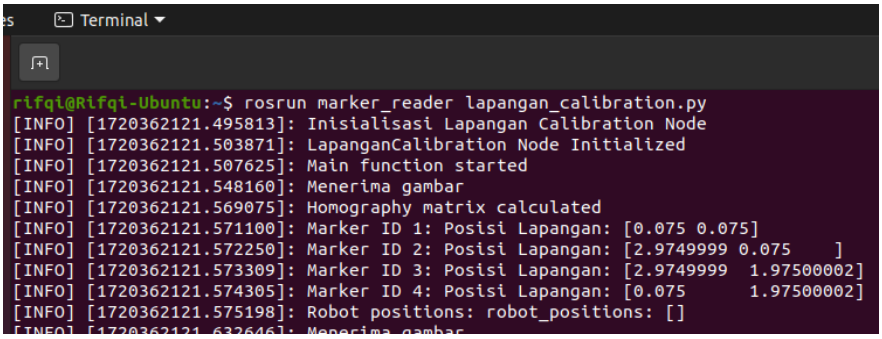
```
robot_position, robot_orientation = self.get_robot_position_and_orientation(corner[0][0],
rvec)  # Hitung posisi dan orientasi robot
```

Bagian kode ini menampilkan gambar asli dengan sumbu frame yang digambar di sekitar setiap marker dan memperbarui tampilan setiap 3 ms.

```
cv2.imshow("Image window", cv_image)  # Tampilkan gambar
```

4.1.3.4 Paket "marker_reader"

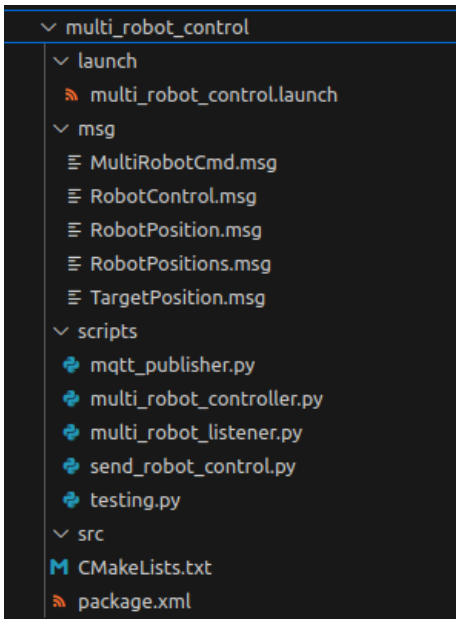
Paket marker_reader bertanggung jawab untuk membaca dan memproses marker ArUco yang terdeteksi oleh kamera. Menggunakan data gambar yang dipublikasikan oleh rso_opencv, paket ini mengidentifikasi marker dan menentukan posisi (x dan y) serta orientasinya. Informasi ini kemudian dikirim ke ROS agar dapat digunakan oleh robot untuk navigasi dan berbagai tugas lainnya, membantu robot dalam memahami lingkungan sekitarnya berdasarkan deteksi marker. Untuk menjalankan paket ini perlu memastikan terlebih dahulu bahwa topic gambar dari ros_opencv sudah tersedia. Jalankan node marker_reader dengan perintah "`rosrun marker_reader lapangan_calibration.py`". Node ini akan mendeteksi marker ArUco dari gambar yang dipublikasikan oleh ros_opencv dan mengirimkan informasi posisi dan orientasi marker ke ROS. Saat menjalankan perintah tersebut akan tampak seperti pada gambar berikut:



Gambar 4. 9 Tampilan terminal ubuntu ketika menjalankan paket marker_reader

4.1.3.5 Struktur Paket “multi_robot_control”

Struktur paket multi_robot_control dalam proyek ROS yang menggunakan Visual Studio Code mencakup beberapa file dan skrip penting yang mendukung pengendalian beberapa robot secara simultan. File multi_robot_control.launch digunakan untuk memulai seluruh sistem multi-robot dengan meluncurkan berbagai node ROS yang diperlukan. Struktur paketnya adalah sebagai berikut:



Gambar 4. 10 Struktur paket “multi_robot_control”

Skrip `mqtt_publisher.py` bertanggung jawab untuk menerjemahkan pesan ROS menjadi pesan MQTT dan mengirimkannya ke broker MQTT, memungkinkan kontrol jarak jauh robot melalui protokol MQTT. Skrip `multi_robot_listener.py` mendengarkan pesan kontrol yang dikirim melalui topik ROS dan meneruskannya ke masing-masing robot melalui topik individu seperti `` `/robot1/cmd_vel` ``, `` `/robot2/cmd_vel` ``, `` `/robot3/cmd_vel` ``, dan `` `/omni_robot/cmd_vel` ``. `send_robot_control.py` mengirim perintah kecepatan dan posisi target ke robot dengan mempublikasikan pesan ke topik yang sesuai dan menggunakan MQTT untuk mengirim perintah berdasarkan ID robot. Skrip `multi_robot_controller.py` mengelola logika kontrol untuk beberapa robot, termasuk menghitung kecepatan berdasarkan posisi target, mengirim perintah ke robot, dan memperbarui posisi robot secara real-time. Dengan

struktur ini, paket multi_robot_control memastikan integrasi yang baik dari semua aspek pengendalian multi-robot, menggunakan kombinasi ROS dan MQTT untuk memungkinkan kontrol yang fleksibel. Skrip kode lengkap terdapat pada lampiran nomor 3. Beberapa bagian kode yang penting adalah sebagai berikut:

Kode ini menginisialisasi node ROS dengan nama 'robot_controller' dan setup untuk MQTT clients untuk setiap robot yang dihubungkan ke broker MQTT pada alamat dan port tertentu. Publisher juga dibuat untuk mengirim perintah kontrol (RobotControl) ke robot.

```
    rospy.init_node('robot_controller', anonymous=True)

    # Menginisialisasi client MQTT untuk masing-masing robot
    setup_mqtt_clients()

    pub = rospy.Publisher('/robot_control', RobotControl, queue_size=10)
```

Kode ini membuat subscriber untuk menerima posisi robot (RobotPositions) dan posisi target (TargetPosition) dari topik yang sesuai di ROS.

```
    # Menyiapkan subscriber untuk menerima posisi robot dan posisi target
    rospy.Subscriber("/robot_positions", RobotPositions, robot_positions_callback)
    rospy.Subscriber("/target_position", TargetPosition, target_position_callback)
```

Fungsi ini membuat dan menghubungkan client MQTT untuk setiap robot dalam daftar robot_ids. Setiap client dihubungkan ke broker MQTT pada alamat IP dan port yang disediakan.

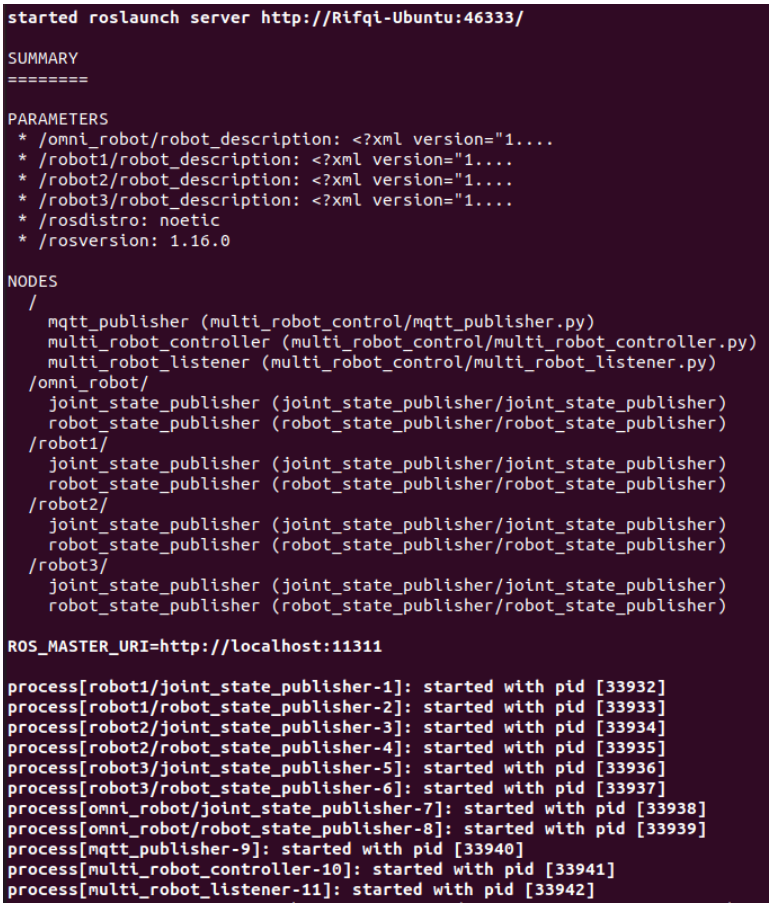
```
# Fungsi untuk mengatur dan memulai client MQTT
def setup_mqtt_clients():
    global mqtt_clients

    # Daftar robot
    robot_ids = ["robot1", "robot2", "robot3", "omni_robot"]

    # Membuat client MQTT untuk setiap robot
    for robot_id in robot_ids:
        client = mqtt.Client(client_id=robot_id, protocol=mqtt.MQTTv311)
        client.on_connect = on_connect
        client.connect("192.168.33.223", 1883, 60)
        client.loop_start()  # Memulai loop untuk client MQTT
        mqtt_clients[robot_id] = client
```

Selain itu, ada beberapa fungsi seperti berikut:

- `on_connect`: Fungsi callback saat terhubung ke broker MQTT.
- `send_command`: Fungsi untuk mengirim perintah ke robot melalui MQTT dengan payload berupa kecepatan linear dan angular.
- `calculate_velocity_unicycle`: Fungsi untuk menghitung kecepatan robot differential drive berdasarkan posisi target dan posisi saat ini.
- `update_command_for_robo`t: Fungsi untuk memperbarui perintah untuk robot berdasarkan posisi mereka dan posisi target.
- `robot_positions_callback`: Callback untuk menerima posisi robot dan memperbarui perintah kontrol berdasarkan posisi tersebut.
- `target_position_callback`: Callback untuk menerima posisi target dan menyimpannya untuk digunakan dalam memperbarui perintah kontrol.

4.1.3.6 Paket "multi_robot_control"

Paket mullti_robot_control mengatur kontrol dan pergerakan robot berdasarkan data yang diterima dari ROS, termasuk informasi dari marker_reader dan perintah pergerakan lainnya. Paket ini mengirimkan perintah ke ESP32 yang mengontrol motor driver dan motor DC pada robot. Dengan demikian, multi_robot_control memastikan bahwa robot bergerak sesuai dengan perintah yang diterima, memungkinkan koordinasi yang tepat antara sensor dan aktuator untuk navigasi dan tugas-tugas robotik lainnya. Jalankan node multi_robot_control dengan perintah "`roslaunch multi_robot_control multi_robot_control.launch`". Node ini akan menerima data dari ROS, termasuk informasi dari marker_reader dan mengirimkan perintah pergerakan ke ESP32 untuk mengontrol pergerakan roda dan motor robot. Saat menjalankan perintah tersebut akan tampak seperti pada gambar berikut:



Gambar 4. 11 Tampilan terminal ubuntu ketika menjalankan paket “multi_robot_control”

4.1.3.7 Kode Robot pada Arduino IDE

ESP32 yang diprogram menggunakan Arduino IDE agar dapat berinteraksi dengan node ROS, memungkinkan integrasi yang mulus antara perangkat keras robot dan sistem ROS yang baik. Skrip kode untuk setiap robot terdapat pada lampiran nomor 4. Beberapa bagian penting pada kode yang diupload ke ESP32 tiap robot adalah sebagai berikut:

Fungsi callback yang dipanggil ketika pesan MQTT diterima dari topik tertentu. Pesan kemudian diteruskan ke fungsi controlMotors() untuk mengontrol motor sesuai perintah, serta fungsi untuk mencoba kembali menghubungkan klien MQTT ke broker jika koneksi terputus.

```
void callback(char* topic, byte* message, unsigned int length) {
  String command;
```

```
  for (int i = 0; i < length; i++) {
    command += (char)message[i];
  }
  Serial.print("Message arrived on topic: ");
  Serial.print(topic);
  Serial.print(". Message: ");
  Serial.println(command);

  controlMotors(command);
}

void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect("ESP32Client")) {
      Serial.println("connected");
      client.subscribe("robot1/control");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}
```

Fungsi setup Arduino yang digunakan untuk menginisialisasi pin-pindan motor, mengatur koneksi WiFi, menghubungkan ke broker MQTT, dan menetapkan fungsi callback, serta fungsi loop Arduino yang digunakan untuk mempertahankan koneksi MQTT dan menangani pesan yang masuk dari broker.

```
void setup() {
  pinMode(enableLeftMotor, OUTPUT);
  pinMode(leftMotorPin1, OUTPUT);
  pinMode(leftMotorPin2, OUTPUT);

  pinMode(enableRightMotor, OUTPUT);
  pinMode(rightMotorPin1, OUTPUT);
  pinMode(rightMotorPin2, OUTPUT);

  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
}

void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
```

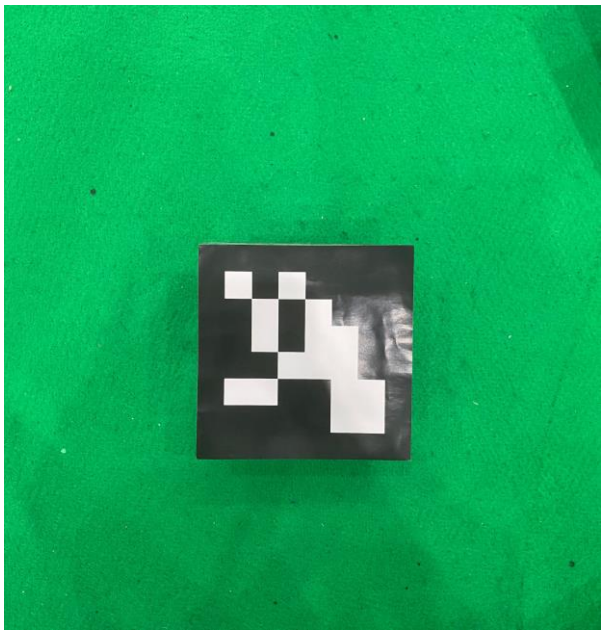
Berikut tampilan serial monitor ketika robot menerima perintah pergerakan dari ROS:

```
19:51:44.469 -> WiFi connected
19:51:44.469 -> IP address:
19:51:44.469 -> 192.168.33.8
19:51:44.469 -> Attempting MQTT connection...connected
19:51:44.501 -> Subscribed to robot/control/robot1 and robot/position/robot1
19:52:21.018 -> Callback invoked on topic: robot/control/robot1
19:52:21.018 -> Message arrived: 1.0,0.0
19:52:21.018 -> Command Message: 1.0,0.0
19:52:21.018 -> Parsed Target X: 1.00
19:52:21.018 -> Parsed Target Y: 0.00
19:52:21.018 -> Setting Left Motor Speed to: 100
19:52:21.053 -> Setting Right Motor Speed to: 100
19:52:21.053 -> Linear velocity: 100.00 Angular velocity: 0.00
19:52:21.053 -> Command executed successfully
```

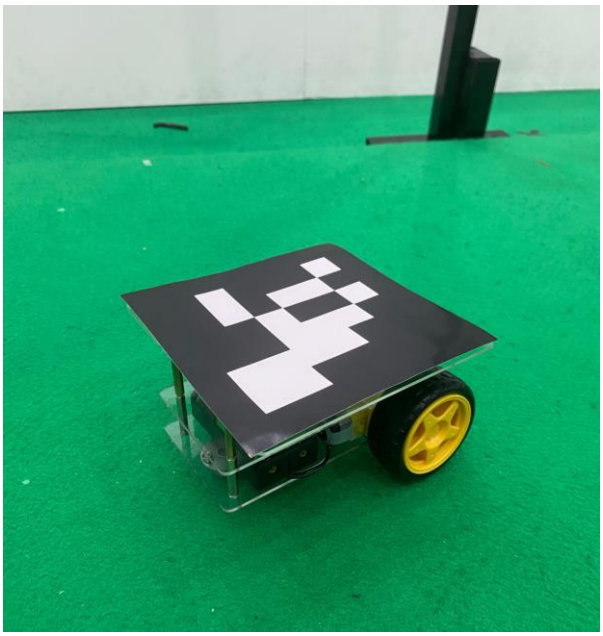
Gambar 4. 12 Tampilan serial monitor ketika robot menerima *command* dari ROS

4.1.4 Foto hasil akhir perancangan

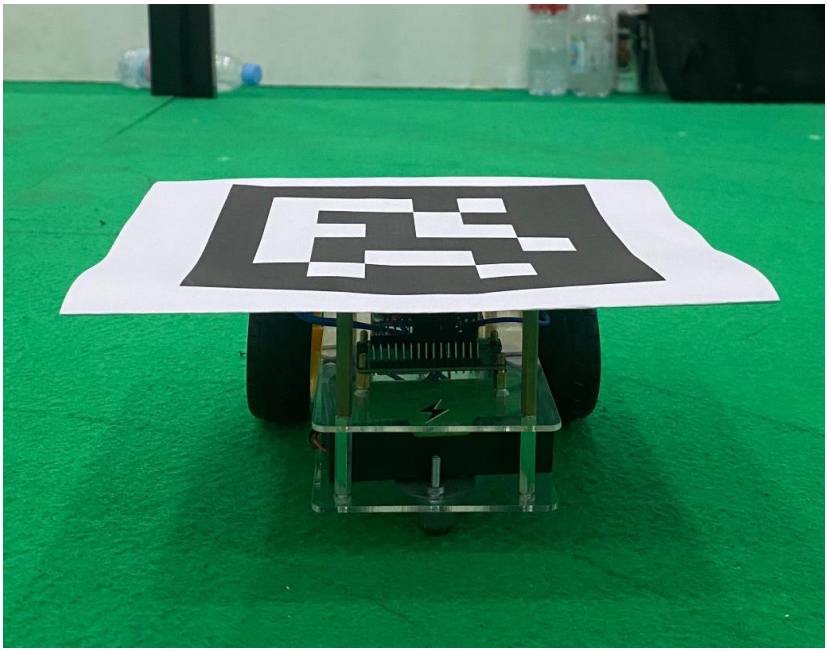
Berikut ini hasil akhir perancangan dari simulator robot beroda diantaranya robot beroda differential drive, omni-directional dan lapangan simulasi.



(a)



(b)



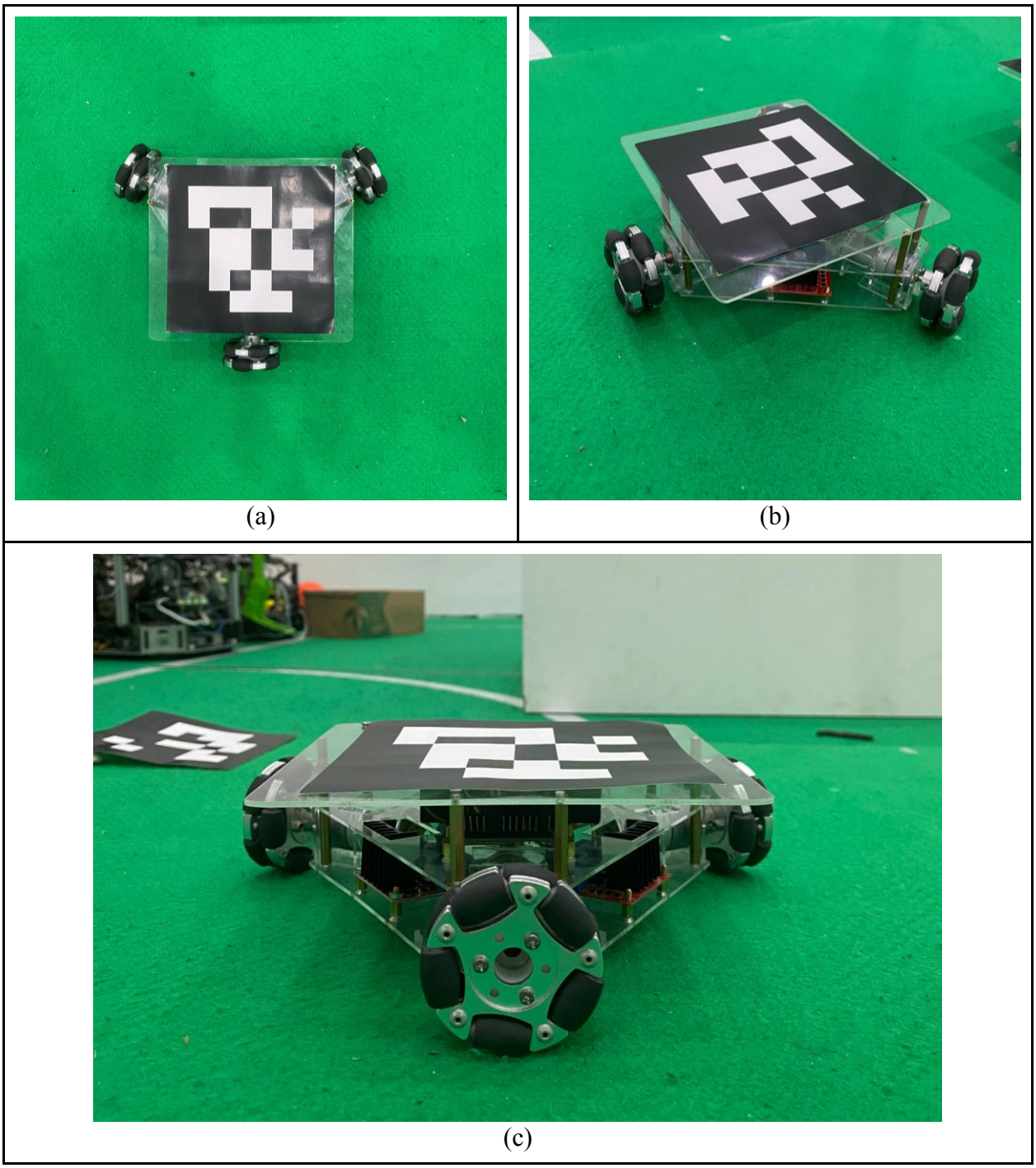
(c)

Gambar 4. 13 Hasil akhir perancangan robot beroda differential drive 2 roda, (a) Tampak atas ,(b) Tampak serong, (c) Tampak depan

Desain untuk robot beroda differential drive 2 roda terbagi dalam tiga lapisan yang memiliki fungsi tersendiri. Pada lapisan pertama, kami menempatkan motor DC gearbox kuning dengan tegangan 6V - 8V. Motor ini didukung oleh dua baterai Li-ion 18650 yang dihubungkan secara seri, memberikan daya yang stabil untuk operasi robot.

Di lapisan kedua, kami menempatkan motor driver L298N dan modul ESP32 Dev Kit1. Motor driver L298N bertugas mengatur motor dan arah geraknya, sedangkan modul ESP32 Dev Kit1 melakukan komunikasi nirkabel serta pengolahan data.

Lapisan ketiga didesain untuk dapat diletakan sebuah penanda atau marker. Marker yang dipakai ialah marker ArUco. Marker ArUco ini digunakan untuk pelacakan gerak robot dengan kamera yang diletakkan di bagian atas robot sebagai pembacaan gerak robot beroda dari sistem yang kami buat.



(a) (b)

(c)

Gambar 4. 14 Hasil akhir perancangan robot beroda omnidirectional 3 roda, (a) Tampak atas ,(b) Tampak serong, (c) Tampak depan

Pada desain robot beroda omni directional roda 3 perbedaan tidak terlalu signifikan dari yang 2 roda differential drive. Lapis dasar atau pertama pada robot beroda 3 diletakkan 3 buah

motor DC 12V dengan kecepatan 620 rpm. Beserta dengan 2 motor driver l298n sebagai kontrol motor.

Lapisan kedua baterai ditempatkan 4 buah baterai Li-ion 18650 yang dihubungkan secara seri untuk memberikan daya yang cukup untuk operasi robot. Modul ESP32 Dev Kit1 juga dipasang di lapisan ini untuk menerima perintah dan mengolah data yang dibutuhkan.

Lapisan ketiga juga sama didesain untuk dapat diletakan sebuah penanda atau marker. Marker yang dipakai ialah marker ArUco. Marker ArUco ini digunakan untuk pelacakan gerak robot dengan kamera yang diletakkan di bagian atas robot sebagai pembacaan gerak robot beroda dari sistem yang kami buat.

(a)

(b)

Gambar 4. 15 Hasil akhir lapangan simulasi robot, (a) lapangan ukuran 4x3m, (b) lapangan ukuran 2x3m

Gambar 4.5 memperlihatkan lapangan yang digunakan untuk simulasi robot beroda. Lapangan ini memiliki ukuran 4 x 3 meter, namun dapat diubah menjadi 3 x 2 meter sesuai kebutuhan pembacaan marker pada robot beroda. Tiang-tiang di lapangan dapat diatur dengan ketinggian minimal 2,3 meter hingga maksimal 3,3 meter, dan dilengkapi dengan dudukan untuk meletakkan kamera pada tiang di bagian atas.

4.2 Metode Pengukuran Kinerja Hasil Perancangan

Metode pengukuran kinerja hasil perancangan dalam sistem simulasi robot beroda dilakukan pada lapangan 3x2 meter mencakup beberapa parameter penting yang diukur serta langkah-langkah pengukurannya. Parameter yang diukur meliputi jarak antara sudut-sudut lapangan yang telah ditentukan, yaitu titik 0.0 meter, 3.0 meter, 3.2 meter, dan 0.2 meter. Selain itu, rotasi dan orientasi setiap robot diukur untuk mengevaluasi arah yang dituju oleh robot.

Langkah-langkah pengukuran dimulai dengan persiapan lapangan yang melibatkan penentuan area dan penandaan sudut-sudut lapangan. Kemudian, dilakukan kalibrasi kamera eksternal BPro5 menggunakan metode papan catur untuk memastikan hasil pembacaan yang akurat. Setelah itu, marker ArUco ditempatkan pada setiap robot dan di beberapa titik referensi di lapangan untuk memungkinkan pembacaan yang jelas oleh kamera. Setelah itu, empat robot ditempatkan pada posisi yang telah ditentukan di lapangan, memastikan setiap robot memiliki marker ArUco yang terbaca dengan baik oleh kamera.

Selanjutnya, simulasi dijalankan di mana kamera BPro5 akan menangkap marker di setiap sudut lapangan, serta posisi dan orientasi setiap robot berdasarkan pembacaan marker ArUco. Sebagai langkah verifikasi, dilakukan pula pengukuran menggunakan meteran sebagai perbandingan pembacaan kamera. Data yang dikumpulkan kemudian dianalisis untuk membandingkan hasil pembacaan menggunakan kamera dengan posisi asli dari tiap sudut lapangan dan keempat robot tersebut. Hasil analisis ini digunakan untuk mengevaluasi apakah hasil pembacaan kamera dapat merepresentasikan posisi asli dengan akurasi dan presisi yang tinggi. Metode ini memastikan bahwa kinerja sistem dapat dievaluasi secara komprehensif berdasarkan parameter-parameter yang relevan.

BAB 5. HASIL PENGUKURAN DAN ANALISIS.

5.1. Analisis Hasil

5.1.1 Hasil dan Analisis Pengujian Indikator

5.1.1.1 Kalibrasi Kamera BPro5

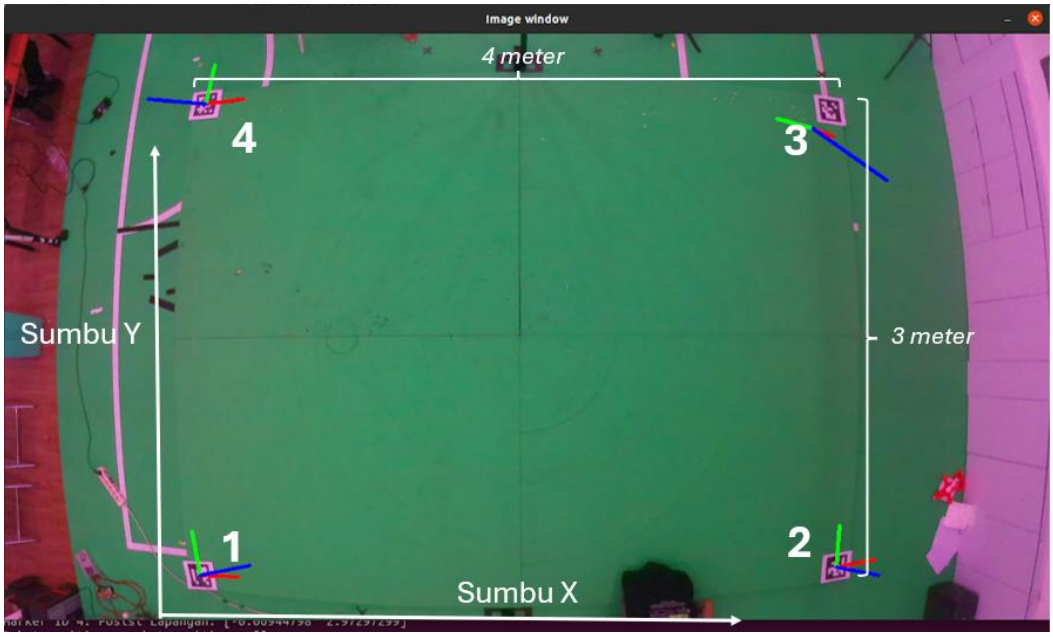
Kalibrasi kamera menggunakan skrip "calibration_script", dengan skrip lengkapnya terdapat di lampiran nomor 5. Hasil kalibrasi kamera menggunakan metode papan catur adalah sebagai berikut:

```
            # Matriks dan distorsi kamera (hasil kalibrasi)
self.camera_matrix = np.array([[9009.10085, 0, 411.009725],     # fx, 0, cx
                               [0, 6233.08793, 508.473989],  # 0, fy, cy
                                [0, 0, 1]])                   # 0, 0, 1
    self.dist_coeffs = np.array([11.7752601, -2109.97792, 0.0963147492,
         0.0510228698, -24.1195436])  # Koefisien distorsi kamera
```

Hasil kalibrasi kamera tersebut akan diinputkan skrip python dengan nama "lapangan_calibration" pada paket "marker_reader". Skrip lengkapnya terdapat pada lampiran nomor 2.

5.1.1.2 Hasil Pembacaan Marker pada Lapangan 4 x 3 meter

Marker dengan ID 1, 2, 3, dan 4 menandakan tiap sudut dari lapangan dengan ukuran 4 x 3 meter, yaitu marker dengan ID 1 menandakan titik (0.075,0.075), ID 2 (3.975,0.075), ID 3 (3.975,2.975), serta ID 4 (0.075,2.975). Selain itu, tiap robot memiliki marker dengan ID 11 menandakan robot1, ID 22 menandakan robot2, ID 33 menandakan robot3, dan ID 44 menandakan omni_robot. Pembacaan marker juga menghasilkan 3 sumbu dengan warna yang berbeda, yaitu sumbu x berwarna merah, sumbu y berwarna hijau, dan sumbu z berwarna biru.



Gambar 5. 1 Pembacaan marker di sudut lapangan pada lapangan berukuran 4 x 3 m

Berdasarkan gambar 5.1 pembacaan marker sudut lapangan, terlihat kesalahan pada marker dengan ID 3. Kesalahan tersebut karena pembacaannya tidak mengenai marker yang dipasang pada lapangan. Selain itu, kesalahan ini terus terjadi walaupun menggunakan marker dengan ID yang berbeda untuk titik (4,3).

```
[INFO] [1720432442.567928]: Menerima gambar
[INFO] [1720432442.576267]: Marker ID 1: Posisi Lapangan: [1.72880875e-15 2.16101094e-15]
[INFO] [1720432442.577571]: Marker ID 2: Posisi Lapangan: [4.00009186 0.00515237]
[INFO] [1720432442.578727]: Marker ID 3: Posisi Lapangan: [4. 3.]
[INFO] [1720432442.580317]: Marker ID 4: Posisi Lapangan: [1.33439513e-15 3.00000000e+00]
[INFO] [1720432442.581647]: Robot positions: robot_positions: []
[INFO] [1720432442.591838]: Menerima gambar
[INFO] [1720432442.601799]: Marker ID 1: Posisi Lapangan: [ 0.00010457 -0.0050614 ]
[INFO] [1720432442.602877]: Marker ID 2: Posisi Lapangan: [4.00000000e+00 4.39927288e-16]
[INFO] [1720432442.603664]: Marker ID 3: Posisi Lapangan: [4. 3.]
[INFO] [1720432442.604347]: Marker ID 4: Posisi Lapangan: [1.33439513e-15 3.00000000e+00]
[INFO] [1720432442.604989]: Robot positions: robot_positions: []
[INFO] [1720432442.622421]: Menerima gambar
[INFO] [1720432442.629874]: Marker ID 1: Posisi Lapangan: [1.72880875e-15 2.16101094e-15]
[INFO] [1720432442.630932]: Marker ID 2: Posisi Lapangan: [4.00000000e+00 4.39927288e-16]
[INFO] [1720432442.631740]: Marker ID 3: Posisi Lapangan: [4. 3.]
[INFO] [1720432442.632544]: Marker ID 4: Posisi Lapangan: [1.33439513e-15 3.00000000e+00]
[INFO] [1720432442.633207]: Robot positions: robot_positions: []
```

Gambar 5. 2 Sampel pembacaan marker pada sistem

Perhitungan rata-rata pembacaan marker sudut lapangan:

Marker ID 1:

x: $(1.729 + 0.000 + 1.729) / 3 = 1.152$
y: $(2.161 + (-0.005) + 2.161) / 3 = 1.439$

Marker ID 2:
x: $(4.000 + 4.000 + 4.000) / 3 = 4.00$
y: $(0.005 + 4.000 + 0.005) / 3 = 1.336$

Marker ID 3:
x: $(4.0 + 4.0 + 4.0) / 3 = 4.0$
y: $(3.0 + 3.0 + 3.0) / 3 = 3.0$

Marker ID 4:
x: $(1.335 + 1.335 + 1.335) / 3 = 1.335$
y: $(3.0 + 3.0 + 3.0) / 3 = 3.0$

Tabel 5. 1 Selisih pembacaan marker lapangan

ID	**Rata-rata Pembacaan (x)**	**Rata-Rata Pembacaan (y)**	**Nilai Sebenarnya (x)**	**Nilai Sebenarnya (y)**	**Selisih Pembacaan (x)**	**Selisih Pembacaan (y)**
1	1.152	1.439	0.000	0.000	1.152 m	1.439 m
2	4.000	1.336	4.000	0.000	0 m	1.336 m
3	4.000	3.000	4.000	3.000	0 m	0 m
4	1.335	3.000	0.000	3.000	1.335 m	0 m

Berdasarkan sampel pada gambar 5.2 yang diambil dan perhitungan rata-rata kesalahan, serta nilai error untuk tiap ID, terlihat bahwa hasil pembacaan marker pada sistem mengalami kesalahan yang cukup besar dengan pergeseran nilai sampai 4.39 meter dari titik seharusnya. Namun untuk marker dengan ID 3 yang diketahui salah pembacaannya, menunjukkan nilai pembacaan yang benar yaitu titik (4,3). Selain itu, berdasarkan perhitungan nilai error, diketahui bahwa presentasi kesalahan pembacaan sangat besar untuk marker ID 1, marker ID 2 dan marker ID 4. Kesalahan tersebut sangat jelas untuk marker ID 1 yang bergeser sekitar 1.152 meter di sumbu x dan bergeser sekitar 1.439 meter, marker ID 2 yang bergeser sekitar 1.336 meter di sumbu x, serta marker ID 4 yang bergeser 1.335 meter di sumbu x.

Gambar 5. 3 Pembacaan marker sudut lapangan dan marker robot pada lapangan berukuran 4 x 3 m

Berdasarkan gambar 5.1 pembacaan marker sudut lapangan, terlihat kesalahan pada marker dengan ID 3 masih terjadi.

```
Menerima gambar
Marker ID 1: Posisi Lapangan: [-4.77328208e-03  6.01725087e-05]
Marker ID 2: Posisi Lapangan: [4.00000000e+00 2.43757065e-15]
Marker ID 3: Posisi Lapangan: [4. 3.]
Marker ID 4: Posisi Lapangan: [0.00475031 3.0001911 ]
robot 1: Posisi: [0.90354144 0.46151928], Orientasi: -3.1253297943233638
robot 2: Posisi: [3.20744087 2.37256111], Orientasi: -2.954776844093427
robot 3: Posisi: [3.09614421 0.67773767], Orientasi: -0.11830340404220188
omni_robot: Posisi: [0.89712742 2.49866078], Orientasi: 1.609436242084127
Robot positions: robot_positions:
```

```
Menerima gambar
Marker ID 1: Posisi Lapangan: [-4.77328208e-03  6.01725087e-05]
Marker ID 2: Posisi Lapangan: [4.00000000e+00 2.43757065e-15]
Marker ID 3: Posisi Lapangan: [4. 3.]
Marker ID 4: Posisi Lapangan: [0.00475031 3.0001911 ]
robot 1: Posisi: [0.90354144 0.46151928], Orientasi: -3.1253297943233638
robot 2: Posisi: [3.20744087 2.37256111], Orientasi: -2.954776844093427
robot 3: Posisi: [3.09614421 0.67773767], Orientasi: -0.11830340404220188
omni_robot: Posisi: [0.89712742 2.49866078], Orientasi: 1.609436242084127
Robot positions: robot_positions:
```

Gambar 5. 4 Sampel pembacaan marker ketika ada robot di lapangan

Perhitungan rata-rata pembacaan marker sudut lapangan:

Marker ID 1:

x: $(-4.773 + 9.109) / 2 = 2.168$

y: $(6.017 + 3.188) / 2 = 4.602$

Marker ID 2:
x: $(4.000 + 4.000) / 2 = 4.000$
y: $(2.437 + 2.437) / 2 = 2.437$

Marker ID 3:
x: $(4.000 + 4.000) / 2 = 4.0$
y: $(3.000 + 3.000) / 2 = 3.0$

Marker ID 4:
x: $(0.004 + 2.945) / 2 = 1.474$
y: $(3.000 + 3.000) / 2 = 3.000$

Marker ID 11:
x: $(0.9035 + 0.9035) / 2 = 0.9035$
y: $(0.4615 + 0.4615) / 2 = 0.4615$

Marker ID 22:
x: $(3.2074 + 3.2074) / 2 = 3.2074$
y: $(2.3726 + 2.3726) / 2 = 2.3726$

Marker ID 33:
x: $(3.0964 + 3.0964) / 2 = 3.0964$
y: $(0.6777 + 0.6777) / 2 = 0.6777$

Marker ID 44:
x: $(0.8971 + 0.8971) / 2 = 0.8971$
y: $(2.4986 + 2.4986) / 2 = 2.4986$

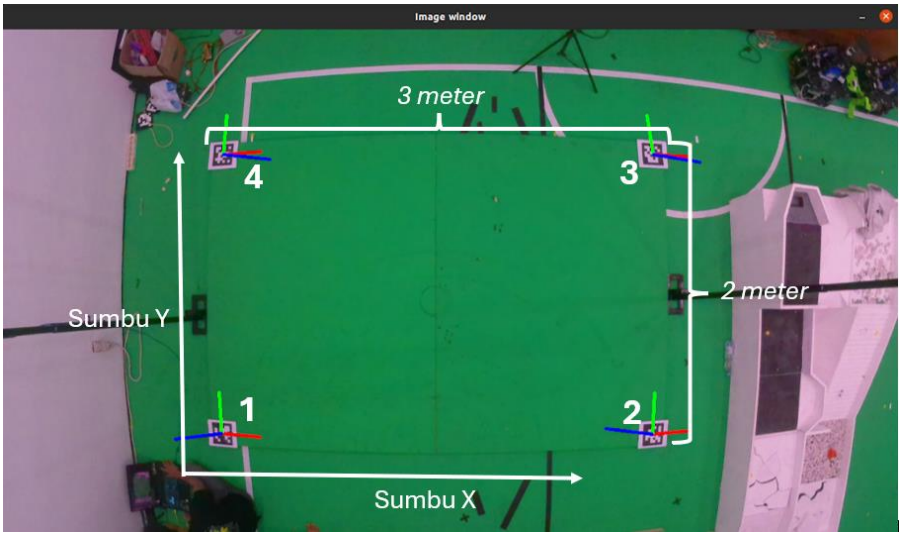
Tabel 5. 2 Selisih pembacaan marker lapangan dan robot (sekitar titik tengah lapangan)

ID	**Rata-Rata Pembacaan (x)**	**Rata-Rata Pembacaan (y)**	**Nilai Sebenarnya (x)**	**Nilai Sebenarnya (y)**	**Selisih Pembacaan (x)**	**Selisih Pembacaan (y)**
1	2.168	4.602	0.000	0.000	2.168 m	4.602 m
2	4.000	2.437	4.000	0.000	0 m	2.437 m
3	4.000	3.000	4.000	3.000	0 m	0 m
4	1.474	3.000	0.000	3.000	1.474 m	0 m
11	0.9035	0.4615	1.000	0.750	0.0965 m	0.2885 m
22	3.2074	2.3726	3.000	0.750	0.2074 m	1.6226 m
33	3.0964	0.6777	3.000	2.250	0.0964 m	1.5723 m
44	0.8971	2.4986	1.000	2.250	0.1029 m	0.2486 m

Berdasarkan gambar 5.4 yang diambil dan perhitungan rata-rata kesalahan, serta nilai error untuk tiap ID, terlihat kesalahan yang makin besar untuk marker sudut lapangan dan robot. Sama seperti sebelum meletakkan robot di lapangan, ID 3 yang diketahui salah pembacaannya, tetap menunjukkan nilai pembacaan yang benar yaitu titik (4,3). Selain itu, sudut lapangan lainnya dengan ID 1, 2, dan 4 mengalami perubahan pembacaan. Untuk pembacaan marker tiap robot juga mengalami kesalahan yang cukup besar. Terlihat bahwa marker ID 1 menunjukkan kesalahan terbesar pada sumbu x dan y, dengan pergeseran nilai mencapai 4.602 meter dari titik seharusnya. Marker ID 22 dan ID 33 juga menunjukkan kesalahan signifikan pada sumbu y, dengan pergeseran sekitar 1.6226 meter dan 1.5723 meter masing-masing marker ID.

5.1.1.3 Hasil Pembacaan Marker pada Lapangan 3x2 meter

Pembacaan marker yang salah saat menggunakan lapangan berukuran 4 x 3 m, membuat kami melakukan pengukuran dan pengambilan data untuk ukuran lapangan yang lebih kecil, yaitu 3x2 meter (setengah dari lapangan sebelumnya). Marker dengan ID 1, 2, 3, dan 4 menandakan tiap sudut dari lapangan, yaitu marker dengan ID 1 menandakan titik (0.075,0.075), ID 2 (2.975,0.075), ID 3 (2.975,1.975), serta ID 4 (0.075,1.975). Selain itu, tiap robot memiliki marker dengan ID 11 menandakan robot1, ID 22 menandakan robot2, ID 33 menandakan robot3, dan ID 44 menandakan omni_robot. Pembacaan marker juga menghasilkan 3 sumbu dengan warna yang berbeda, yaitu sumbu x berwarna merah, sumbu y berwarna hijau, dan sumbu z berwarna biru.



Gambar 5. 5 Hasil pembacaan marker pada lapangan 3x2

Berdasarkan gambar 5.5 hasil pembacaan marker pada lapangan berukuran 3x2 meter, terlihat bahwa hasil pembacaan marker tidak mengalami kesalahan seperti sebelumnya.

```
[INFO] [1722231591.881300]: Menerima gambar
[INFO] [1722231591.893762]: Marker ID 1: Posisi Lapangan: [0.075 0.075]
[INFO] [1722231591.895232]: Marker ID 2: Posisi Lapangan: [2.92499995 0.075     ]
[INFO] [1722231591.896410]: Marker ID 3: Posisi Lapangan: [2.92499995 1.92499995]
[INFO] [1722231591.897650]: Marker ID 4: Posisi Lapangan: [0.075      1.92499995]
[INFO] [1722231591.898745]: Robot positions: robot_positions: []
[INFO] [1722231591.914024]: Menerima gambar
[INFO] [1722231591.923399]: Marker ID 1: Posisi Lapangan: [0.075 0.075]
[INFO] [1722231591.924751]: Marker ID 2: Posisi Lapangan: [2.92499995 0.075     ]
[INFO] [1722231591.925849]: Marker ID 3: Posisi Lapangan: [2.92499995 1.92499995]
[INFO] [1722231591.926987]: Marker ID 4: Posisi Lapangan: [0.075      1.92499995]
[INFO] [1722231591.927913]: Robot positions: robot_positions: []
[INFO] [1722231591.946571]: Menerima gambar
[INFO] [1722231591.956022]: Marker ID 1: Posisi Lapangan: [0.075 0.075]
[INFO] [1722231591.957425]: Marker ID 2: Posisi Lapangan: [2.92499995 0.075     ]
[INFO] [1722231591.958541]: Marker ID 3: Posisi Lapangan: [2.92499995 1.92499995]
[INFO] [1722231591.959664]: Marker ID 4: Posisi Lapangan: [0.075      1.92499995]
[INFO] [1722231591.960611]: Robot positions: robot_positions: []
```

Gambar 5. 6 Sampel pembacaan marker ketika ada robot di lapangan

Perhitungan rata-rata pembacaan marker sudut lapangan:

Marker ID 1:

x: $(0.075 + 0.075 + 0.075) / 3 = 0.075$

y: $(0.075 + 0.075 + 0.075) / 3 = 0.075$

Marker ID 2:

x: $(2.925 + 2.925 + 2.925) / 3 = 2.925$

y: $(0.075 + 0.075 + 0.075) / 3 = 0.075$

Marker ID 3:

x: $(2.925 + 2.925 + 2.925) / 3 = 2.925$

y: $(1.925 + 1.925 + 1.925) / 3 = 1.925$

Marker ID 4:

x: $(0.075 + 0.075 + 0.075) / 3 = 0.075$

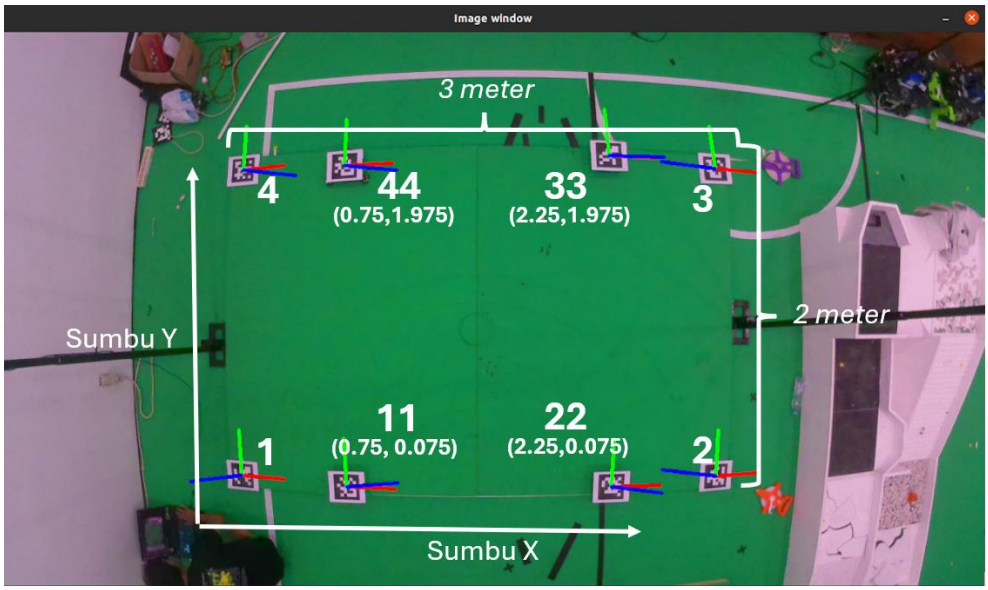
y: $(1.925 + 1.925 + 1.925) / 3 = 1.925$

Tabel 5. 3 Selisih pembacaan marker lapangan

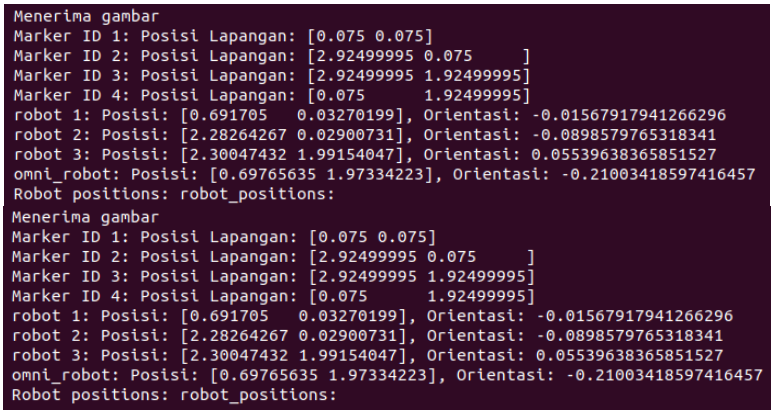
Marker ID	Rata-Rata Pembacaan (x)	Rata-Rata Pembacaan (y)	Nilai Sebenarnya (x)	Nilai Sebenarnya (y)	Selisih Pembacaan (x)	Selisih Pembacaan (y)
1	0.075	0.075	0.075	0.075	0 m	0 m
2	2.925	0.075	2.925	0.075	0 m	0 m
3	2.925	1.925	2.925	1.925	0 m	0 m
4	0.075	1.925	0.075	1.925	0 m	0 m

Berdasarkan sampel pada gambar 5.6 yang diambil dan perhitungan rata-rata kesalahan, serta nilai error untuk tiap ID, terlihat bahwa hasil pembacaan marker pada sistem tidak mengalami kesalahan. Hasil ini menunjukkan peningkatan akurasi yang signifikan dibandingkan dengan pengambilan data pada lapangan yang lebih besar.

5.1.1.4 Marker Lapangan 3x2 dengan Robot di Sekitar Sumbu x



Gambar 5. 7 Hasil pembacaan marker robot (sekitar sumbu x) dan sudut lapangan



Gambar 5. 8 Sampel pembacaan marker ketika ada robot (sekitar sumbu x) di lapangan

Hasil akhir rata-rata pembacaan marker saat ada robot di lapangan (sekitar sumbu x):

Marker ID 1:
x: $(0.075 + 0.075) / 2 = 0.075$
y: $(0.075 + 0.075) / 2 = 0.075$

Marker ID 2:
x: $(2.925 + 2.925) / 2 = 2.925$
y: $(0.075 + 0.075) / 2 = 0.075$

Marker ID 3:
x: $(2.925 + 2.925) / 2 = 2.925$
y: $(1.925 + 1.925) / 2 = 1.925$

Marker ID 4:
x: $(0.075 + 0.075) / 2 = 0.075$
y: $(1.925 + 1.925) / 2 = 1.925$

Marker ID 11:
x: $(0.691 + 0.691) / 2 = 0.691$
y: $(0.032 + 0.032) / 2 = 0.032$

Marker ID 22:
x: $(2.282 + 2.282) / 2 = 2.282$
y: $(0.029 + 0.029) / 2 = 0.029$

Marker ID 33:
x: $(2.300 + 2.300) / 2 = 2.300$
y: $(1.991 + 1.991) / 2 = 1.991$

Marker ID 44:
x: $(0.697 + 0.697) / 2 = 0.697$

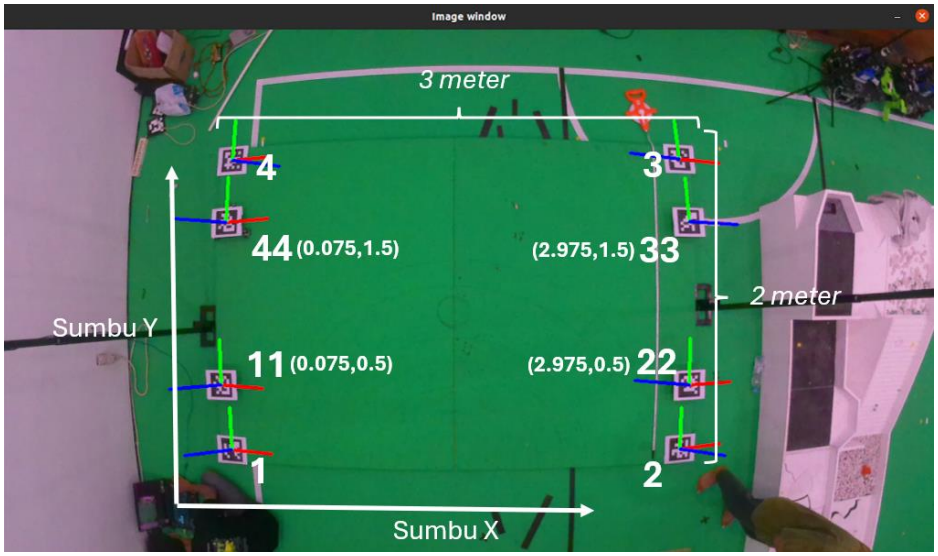
y: (1.973 + 1.973) / 2 = 1.973

Tabel 5. 4 Selisih pembacaan marker lapangan dan robot (sekitar sumbu x lapangan)

ID	**Rata-Rata Pembacaan (x)**	**Rata-Rata Pembacaan (y)**	**Nilai Sebenarnya (x)**	**Nilai Sebenarnya (y)**	**Selisih Pembacaan (x)**	**Selisih Pembacaan (y)**
1	0.075	0.075	0.075	0.075	0 m	0 m
2	2.925	0.075	2.925	0.075	0 m	0 m
3	2.925	1.925	2.925	1.925	0 m	0 m
4	0.075	1.925	0.075	1.925	0 m	0 m
11	0.691	0.032	0.750	0.075	0.059 m	0.043 m
22	2.282	0.029	2.250	0.075	0.032 m	0.046 m
33	2.300	1.991	2.250	1.975	0.050 m	0.016 m
44	0.697	1.973	0.750	1.975	0.053 m	0.002 m

Berdasarkan gambar 5.8 yang diambil dan perhitungan rata-rata kesalahan serta nilai error untuk tiap ID, terlihat bahwa hasil pembacaan marker pada sistem menunjukkan akurasi yang cukup baik dibandingkan dengan pengambilan data di lapangan sebelumnya. Marker ID 1, 2, 3, dan 4 tidak mengalami kesalahan pada sumbu x dan y, menunjukkan akurasi yang sangat baik. Secara keseluruhan, hasil pembacaan marker pada sistem menunjukkan bahwa meskipun terdapat beberapa kesalahan kecil, pembacaan marker masih dalam batas toleransi yang wajar untuk sebagian besar marker.

5.1.1.5 Marker Lapangan 3x2 dengan Robot di Sekitar Sumbu y



Gambar 5. 9 Hasil pembacaan marker robot (sekitar sumbu y) dan sudut lapangan

```
Menerima gambar
Marker ID 1: Posisi Lapangan: [0.075 0.075]
Marker ID 2: Posisi Lapangan: [2.92499995 0.075     ]
Marker ID 3: Posisi Lapangan: [2.92493181 1.92044919]
Marker ID 4: Posisi Lapangan: [0.075      1.92499995]
robot 1: Posisi: [0.00483364 0.49182974], Orientasi: -0.019582624882220668
robot 2: Posisi: [2.97719774 0.49338313], Orientasi: -0.13371610050414268
robot 3: Posisi: [2.99667598 1.52594094], Orientasi: 0.049499545588501934
omni_robot: Posisi: [0.03237958 1.51916511], Orientasi: -0.12597646310957253
Robot positions: robot_positions:
Menerima gambar
Marker ID 1: Posisi Lapangan: [0.075 0.075]
Marker ID 2: Posisi Lapangan: [2.92499995 0.075     ]
Marker ID 3: Posisi Lapangan: [2.92499995 1.92499995]
Marker ID 4: Posisi Lapangan: [0.075      1.92499995]
robot 1: Posisi: [0.00483364 0.49182974], Orientasi: -0.019582624882220668
robot 2: Posisi: [2.97719774 0.49338313], Orientasi: -0.13371610050414268
robot 3: Posisi: [2.99667598 1.52594094], Orientasi: 0.049499545588501934
omni_robot: Posisi: [0.03237958 1.51916511], Orientasi: -0.25191848719541887
Robot positions: robot_positions:
```

Gambar 5. 10 Sampel pembacaan marker ketika ada robot (sekitar sumbu y) di lapangan

Hasil akhir rata-rata pembacaan marker saat ada robot di lapangan (sekitar sumbu y):

Marker ID 1:

x: $(0.075 + 0.075) / 2 = 0.075$

y: $(0.075 + 0.075) / 2 = 0.075$

Marker ID 2:

x: $(2.925 + 2.925) / 2 = 2.925$

y: $(0.075 + 0.075) / 2 = 0.075$

Marker ID 3:

x: $(2.925 + 2.925) / 2 = 2.925$
y: $(1.925 + 1.925) / 2 = 1.925$

Marker ID 4:
x: $(0.075 + 0.075) / 2 = 0.075$
y: $(1.925 + 1.925) / 2 = 1.925$

Marker ID 11:
x: $(0.004 + 0.004) / 2 = 0.004$
y: $(0.491 + 0.491) / 2 = 0.491$

Marker ID 22:
x: $(2.977 + 2.977) / 2 = 2.977$
y: $(0.493 + 0.493) / 2 = 0.493$

Marker ID 33:
x: $(2.996 + 2.996) / 2 = 2.996$
y: $(1.525 + 1.525) / 2 = 1.525$

Marker ID 44:
x: $(0.032 + 0.032) / 2 = 0.032$
y: $(1.519 + 1.519) / 2 = 1.519$

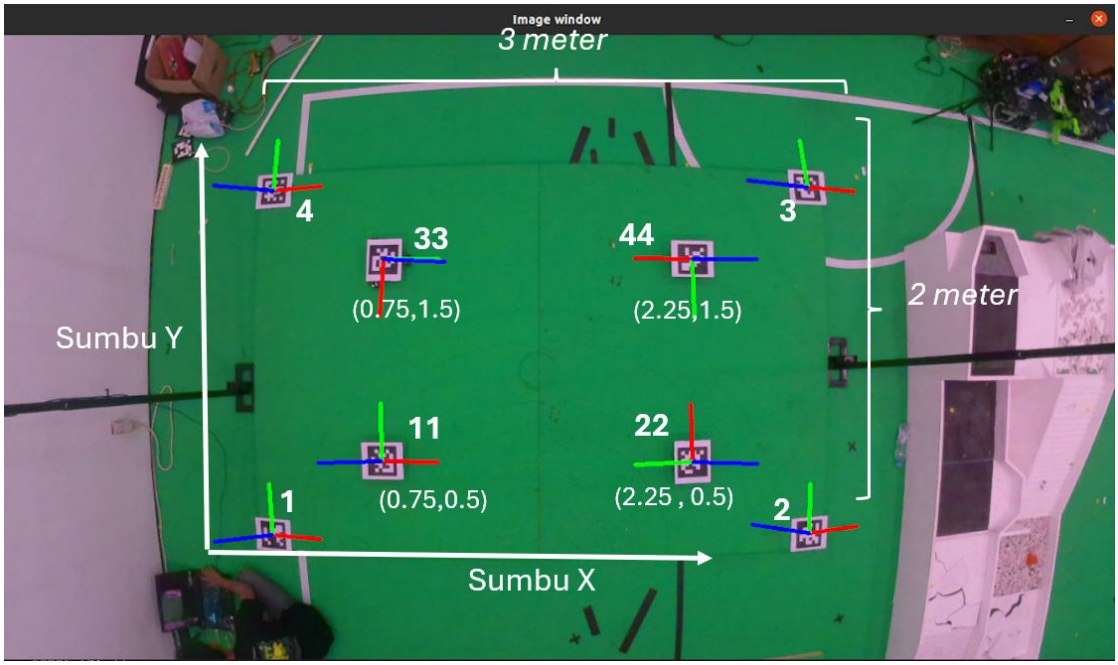
Tabel 5. 5 Selisih pembacaan marker lapangan dan robot (sekitar sumbu y lapangan)

ID	**Rata-Rata Pembacaan (x)**	**Rata-Rata Pembacaan (y)**	**Nilai Sebenarnya (x)**	**Nilai Sebenarnya (y)**	**Selisih Pembacaan (x)**	**Selisih Pembacaan (y)**
1	0.075	0.075	0.075	0.075	0 m	0 m
2	2.975	0.075	2.975	0.075	0 m	0 m
3	2.975	1.925	2.975	1.925	0 m	0 m
4	0.075	1.925	0.075	1.925	0 m	0 m
11	0.004	0.491	0.075	0.075	0.071 m	0.416 m
22	2.977	0.493	2.925	0.075	0.052 m	0.418 m
33	2.996	1.525	2.925	1.925	0.071 m	0.400 m
44	0.032	1.519	0.075	1.925	0.043 m	0.406 m

Berdasarkan perhitungan rata-rata kesalahan serta nilai error untuk tiap ID, terlihat bahwa hasil pembacaan marker pada sistem menunjukkan beberapa kesalahan yang perlu diperbaiki. Marker ID 1, 2, 3, dan 4 tidak mengalami kesalahan pada sumbu x dan y, menunjukkan akurasi yang sangat baik. Namun, marker ID 11 sampai ID 44 menunjukkan

kesalahan yang bervariasi pada sumbu x dan y, dengan kesalahan terbesar adalah 0.071 m pada sumbu x untuk marker ID 11 dan 0.416 m pada sumbu y untuk marker ID 22. Secara keseluruhan, hasil pembacaan marker pada sistem masih dalam batas toleransi yang wajar untuk sebagian besar marker.

5.1.1.6 Marker Lapangan 3x2 dengan Robot di Sekitar Titik Tengah



Gambar 5. 11 Hasil pembacaan marker robot (sekitar tengah lapangan) dan sudut lapangan

```
Menerima gambar
Marker ID 1: Posisi Lapangan: [0.07006342 0.07032799]
Marker ID 2: Posisi Lapangan: [2.92506895 0.07968954]
Marker ID 3: Posisi Lapangan: [2.92499995 1.92499995]
Marker ID 4: Posisi Lapangan: [0.07481096 1.92039764]
robot 1: Posisi: [0.67636049 0.48504025], Orientasi: 0.02650664238049234
robot 2: Posisi: [2.31284782 0.4660693 ], Orientasi: -0.11881109810423283
robot 3: Posisi: [2.31202219 1.55364884], Orientasi: 0.057391651918627974
omni_robot: Posisi: [0.67245447 1.5678305 ], Orientasi: -0.08316416012748495
Robot positions: robot_positions:
Menerima gambar
Marker ID 1: Posisi Lapangan: [0.07006342 0.07032799]
Marker ID 2: Posisi Lapangan: [2.92506895 0.07968954]
Marker ID 3: Posisi Lapangan: [2.92499995 1.92499995]
Marker ID 4: Posisi Lapangan: [0.075      1.92499995]
robot 1: Posisi: [0.67636049 0.48504025], Orientasi: 0.027036559350785035
robot 2: Posisi: [2.30812881 0.46149112], Orientasi: -0.13839609540191472
robot 3: Posisi: [2.31202219 1.55364884], Orientasi: 0.057391651918627974
omni_robot: Posisi: [0.67229059 1.56321241], Orientasi: 0.004756424759077677
Robot positions: robot_positions:
```

Gambar 5. 12 Sampel pembacaan marker ketika ada robot (sekitar titik tengah lapangan) dan marker sudut lapangan

Hasil akhir rata-rata pembacaan marker saat ada robot di lapangan (titik tengah):

Marker ID 1:
x: $(0.070 + 0.070) / 2 = 0.070$
y: $(0.070 + 0.070) / 2 = 0.070$

Marker ID 2:
x: $(2.925 + 2.925) / 2 = 2.925$
y: $(0.079 + 0.079) / 2 = 0.079$

Marker ID 3:
x: $(2.925 + 2.925) / 2 = 2.925$
y: $(1.925 + 1.925) / 2 = 1.925$

Marker ID 4:
x: $(0.075 + 0.075) / 2 = 0.075$
y: $(1.925 + 1.925) / 2 = 1.925$

Marker ID 11:
x: $(0.676 + 0.676) / 2 = 0.676$
y: $(0.485 + 0.485) / 2 = 0.485$

Marker ID 22:
x: $(2.312 + 2.308) / 2 = 2.2865$
y: $(0.466 + 0.461) / 2 = 0.0104$

Marker ID 33:
x: $(2.312 + 2.312) / 2 = 2.312$
y: $(1.553 + 1.553) / 2 = 1.553$

Marker ID 44:
x: $(0.672 + 0.672) / 2 = 0.672$
y: $(1.567 + 1.567) / 2 = 1.567$

Tabel 5. 6 Selisih pembacaan marker lapangan dan robot (sekitar titik tengah lapangan)

ID	**Rata-Rata Pembacaan (x)**	**Rata-Rata Pembacaan (y)**	**Nilai Sebenarnya (x)**	**Nilai Sebenarnya (y)**	**Selisih Pembacaan (x)**	**Selisih Pembacaan (y)**
1	0.070	0.070	0.075	0.075	0.005 m	0.005 m
2	2.925	1.925	2.925	1.925	0 m	0 m
3	2.925	1.925	2.925	1.925	0 m	0 m
4	0.075	1.925	0.075	1.925	0 m	0 m
11	0.676	0.485	0.075	0.075	0.601 m	0.410 m

22	2.310	0.464	2.925	0.075	0.615 m	0.389 m
33	2.312	1.553	2.925	1.925	0.613 m	0.372 m
44	0.672	1.567	0.075	1.925	0.597 m	0.358 m

Berdasarkan perhitungan rata-rata kesalahan serta nilai error untuk tiap ID, terlihat bahwa hasil pembacaan marker pada sistem menunjukkan beberapa kesalahan yang cukup signifikan pada beberapa marker, terutama marker ID 11 sampai ID 44. Marker ID 1, 2, 3, dan 4 menunjukkan kesalahan yang sangat kecil pada sumbu x dan y, dengan kesalahan maksimum sebesar 0.005 m pada marker ID 1, menunjukkan akurasi yang sangat baik. Sementara itu, marker ID 11 hingga ID 44 menunjukkan kesalahan yang lebih besar pada kedua sumbu, namun secara keseluruhan hasil pembacaan marker masih dalam batas toleransi yang dapat diterima untuk sebagian besar marker.

5.1.2 Pemenuhan Spesifikasi Sistem

Secara umum, sistem yang direalisasikan telah memenuhi sebagian besar spesifikasi yang diusulkan dengan beberapa penyesuaian pada beberapa aspek. Spesifikasi awal mencakup perangkat keras untuk dua jenis robot (differential drive dan omni-directional), perangkat lunak untuk mengolah pembacaan marker, mensimulasikan pergerakan robot, mengendalikan pergerakan robot fisik, serta menyediakan antarmuka pengguna. Namun, terdapat beberapa perubahan, seperti ukuran lapangan yang digunakan menjadi 3x2 meter dan keterbatasan robot yang belum dapat berjalan otomatis dari titik ke titik. Pada pembacaan marker pula masih terdapat kesalahan dikarenakan marker yang digunakan mengkilap sehingga tidak dapat terbaca oleh kamera. Solusi yang dilakukan ialah dengan menggunakan marker yang tidak mengkilap sehingga dapat dibaca dengan baik

Meskipun belum sepenuhnya dapat mensimulasikan 3 robot differential drive dan 1 robot omni-directional, pencapaian yang telah dicapai termasuk kemampuan robot untuk bergerak lurus sesuai perintah dan pembacaan marker sudah berhasil. Oleh karena itu, dapat disimpulkan bahwa spesifikasi sistem secara keseluruhan telah terpenuhi dengan beberapa penyesuaian dan untuk memperbaiki kinerja sistem.

Tabel 5. 7 Perbandingkan usulan dan hasil perancangan sistem

No	Spesifikasi	Usulan	Realisasi

1	Dimensi lapangan (panjang x lebar)	4 x 3 m	3 x 2 m
2	Robot differential drive	5 buah	3 buah
3	Robot omni-directional	1 buah	1 buah
4	Pergerakan robot	Otomatis penuh (setelah diberi perintah)	Bergerak lurus (perintah maju saja)
5	Mikrokontroller	ESP 32 dev kit 1	ESP 32 dev kit 1
6	Jumlah robot yang disimulasikan	3 buah robot differential drive dan 1 robot omni-directional	2 buah robot differential drive
7	Fungsi perangkat lunak	Mengolah marker, mensimulasikan pergerakan, mengendalikan robot, antarmuka pengguna	Sebagian besar terpenuhi
8	Antarmuka pengguna	Menampilkan simulasi robot virtual dan dokumentasi pergerakan, mengatur titik tujuan	Belum terpenuhi

Berdasarkan tabel 5.3 terkait perbandingan usulan dan realisasi perancangan sistem, terdapat beberapa perubahan dan penyesuaian yang diterapkan pada spesifikasi sistem. Perubahan yang dilakukan pada ukuran lapangan dari 4 x 3 meter menjadi 3x2 meter disebabkan oleh keterbatasan ruang tangkapan kamera dan kebutuhan pembacaan marker. Jumlah robot differential drive yang direalisasikan hanya 3 buah dari yang diusulkan 5 buah, karena keterbatasan penyesuaian waktu antara pengembangan program yang digunakan dengan pembuatan robot yang akan dijalankan.

Kendala pada pembacaan marker ArUco yang awalnya tidak terbaca karena permukaan mengkilap berhasil diatasi dengan mencetak marker yang tidak silau. Hal ini memastikan marker dapat terbaca dengan baik oleh kamera, sehingga sistem dapat bekerja lebih optimal. Namun, antarmuka pengguna yang diusulkan belum sepenuhnya terpenuhi, menyebabkan sistem pergerakan otomatis yang belum sepenuhnya tercapai. Pengembangan antarmuka pengguna yang

efektif dan responsif masih menjadi tantangan yang perlu diselesaikan agar sistem dapat berfungsi sesuai rencana.

Pencapaian yang telah dicapai termasuk kemampuan robot untuk bergerak lurus sesuai perintah dan pembacaan marker sudah berhasil. Walaupun belum sepenuhnya dapat mensimulasikan 3 robot differential drive dan 1 robot omni-directional, beberapa penyesuaian telah dilakukan untuk meningkatkan kinerja sistem secara keseluruhan. Penyesuaian pada dimensi lapangan dan pembacaan marker yang lebih efektif adalah langkah yang diambil dan telah meningkatkan kualitas simulasi dan kontrol robot.

Meskipun demikian, masih terdapat beberapa aspek yang perlu ditingkatkan, seperti kemampuan robot untuk berjalan otomatis dari titik ke titik dan pengembangan antarmuka pengguna yang lebih intuitif. Kendala-kendala yang dihadapi, seperti pembacaan marker yang tidak terbaca dan keterbatasan pengetahuan, menunjukkan bahwa proyek ini memiliki potensi besar untuk dikembangkan lebih lanjut dengan peningkatan pada aspek-aspek yang belum terpenuhi.

5.1.3 Pengalaman Pengguna

Sistem yang kami buat dicoba dan digunakan oleh kami sendiri yang membuat sistem ini. Selama mencoba dan melakukan pengambilan data berikut beberapa pengalaman yang kami dapatkan:

Tabel 5. 8 Pengalaman Pengguna

No	Fitur/Komponen	Capaian	Aksi/Perbaikan
1	Fungsi	Fungsi sebagai media pembelajaran pergerakan robot beroda dengan tipe differential drive dan omni-directional	Dipertahankan, dikembangkan, dan diperbaiki
2	Kemudahan	Sistem yang dibuat masih tergolong sulit untuk	Membuat antarmuka dan dengan tampilan yang mudah dipahami

		digunakan karena antarmuka pengguna yang belum ada	
3	Kompatibilitas	Robot yang dibuat sesuai dan cocok untuk pembelajaran antara robot beroda differential drive dan omni-directional namun perlu diberikan penjelas struktur komponen dari robot	Diberikan penjelasan struktur disetiap bagian pada robot sehingga dapat dipahami oleh pengguna
4	Keberlanjutan	Sistem memiliki potensi untuk berkembang dengan mengatasi kendala teknis dan menyesuaikan spesifikasi kebutuhan nyata di lapangan simulasinya	Diperbaiki dan ditingkatkan

5.1.4 Kesesuaian Perencanaan dalam Manajemen Tim dan Realisasinya

Selama pengerjaan Tugas Akhir 2, kami berupaya untuk menyesuaikan usulan dan realisasi timeline pengerjaan. Namun, dalam praktiknya terdapat beberapa ketidaktepatan antara waktu yang direncanakan dengan waktu realisasi. Ketidaktepatan ini, meskipun menyebabkan mundurnya jadwal, masih dapat kami atasi. Beberapa faktor penyebab ketidaktepatan waktu tersebut meliputi kendala teknis, perubahan spesifikasi atau kebutuhan lain, serta penyesuaian metode pengerjaan yang diperlukan selama proses pengerjaan Tugas Akhir 2.

Tabel 5. 9 Kesesuaian antara usulan solusi dan realisasi *timeline* pengerjaan Tugas Akhir 2

No	Kegiatan	Usulan waktu	Realisasi Pelaksanaan
1	Pembelian alat dan bahan	Jan – Feb	Feb – April
2	Perancangan sistem sesuai dengan usulan solusi terpilih	Feb – Mar	April - Juni

3	Testing dan Validasi	Mei - Juni	Juli
4.	Expo dan pengumpulan laporan akhir	Juli	Juli

Tabel 5. 10 Kesesuaian RAB Tugas Akhir antara usulan solusi dan realisasi

No	Jenis Pengeluaran	Usulan Biaya		Realisasi Biaya	
		Kuantitas	Total Harga	Kuantitas	Total Harga
1	ESP-32 DEVKIT V1 WIFI Bluetooth ESP-WROOM-32	5 pcs	Rp 310.000,00	4 pcs	Milik Pribadi
2	L298N Dual Motor Driver Module L298 H-Bridge	7 pcs	Rp 107.450,00	5 pcs	Rp 85.000,00
3	JUMPER CABLE KABEL 20CM FEMALE TO FEMALE	40 pcs	Rp 8.500,00	20 pcs	Rp 10.000,00
4	JUMPER CABLE KABEL 10CM MALE TO MALE	40 pcs	Rp 9.000,00	-	-
5	JUMPER CABLE KABEL 10CM MALE TO FEMALE	40 pcs	Rp 8.500,00	20 pcs	Rp 6.000,00
6	WHEEL RODA RUBBER FOR SMART ROBOT CAR 4WD 2WD	12 pcs	Rp 60.000,00	-	-

7	Holder Bracket Braket Breket 3 Sambungan Battery Baterai 18650 - 1solt	2 set	Rp 60.000,00	-	-
8	Nickel Strip Battery 18650 1 Meter/Strip Nikel Baterai Lithium	1 pcs	Rp 28.000,00	-	-
9	DINAMO MOTOR DC 12-18V 3.5A MOTOR DINAMO 12V 18V DC LS775 WITH CABLE	12 pcs	Rp 402.000,00	3 pcs	Rp 112.000,00
10	Baterai Charger/Cas Ulang Li-ion FS 18650 13000mAh 4,2V Hijau Isi 2 pc	9 set	Rp 243.000,00	5 set	Rp 280.000,00
11	PCB DOT MATRIX THRU HOLE SINGLE LAYER 5X7CM 5*7CM LUBANG BOLONG MATRIK	6 pcs	Rp 16.200,00	-	-
12	Box baterai 2x size 18650	-	-	3 pcs	Rp 15.000,00
13	Motor DC gearbox kuning 1:48 3-7V	-	-	6 pcs	Rp 54.000,00
14	18650 Battery holder kotak batre seri baterai 4 slot kabel box	-	-	1 pcs	Rp 7.500,00

15	Besi Kotak (Lapangan)	-	-	6 m	Rp 110.000,00
16	Jasa Las (lapangan)	-	-	1 set	Rp 400.000,00
17	Custom laser cutting akrilik bening 2mm bahan + jasa (robot differential drive dan omni-directional	-	-	4 set	Rp 263.000,00
18	Marker - stiker vinnyl A3 beserta potong	-	-	8 pcs	Rp 35.000,00
19	JUMPER CABLE KABEL 20CM MALE TO FEMALE	-	-	20 pcs	Rp 10.000,00
20	Kabel AWG 18 0,75mm [merah]	-	-	2 m	Rp 4.000,00
21	Motor DC 370 630 RPM	-	-	3 pcs	Rp 195.000,00
22	Pylox sapporo ultimate b13 400cc	-	-	1 pcs	Rp 43.000,00
23	Marker papan catur (Kalibrasi lapangan)	-	-	1 pcs	Rp 32.000,00
24	Saklar ON-OFF AC KCD11-101 rocker	-	-	5 pcs	Rp 10.000,00
25	Spacer kuningan M3 6mm + 3mm	-	-	41 pcs	Rp 41.000,00
26	Spacer kuningan M3 30mm + 3mm	-	-	20 pcs	Rp 20.000,00
27	Spacer kuningan M3 40mm + 3mm	-	-	20 pcs	Rp 30.000,00
28	Spacer nickel M3 25mm + 3mm	-	-	20 pcs	Rp 20.000,00

Total	Rp 1,262,650,00	Rp 2,178,500,00

Dapat dilihat bahwa pada pengerjaan Tugas Akhir 2, antara usulan dan realisasi biaya kami mengalami kenaikan hampir 2 kali lipat. Dari yang semula Rp 1,262,650,00 menjadi Rp 2,178,500,00. Hal ini dikarenakan adanya sedikit perubahan dimensi pada ukuran robot differential drive dan omni-directional. Pada usulan juga kami belum menambahkan biaya untuk *base*/akrilik dari robot yang dibuat dan baru kami tambahkan pada saat perancangan alat.

Tabel 5. 11 Realisasi aktivitas pelaksanaan tugas akhir 2

No	Hari, Tanggal, Durasi (jam atau hari)	Aktivitas	Pelaksana
1	Senin, 13 Februari 2023	Pembelian komponen elektronik	Dimas
2	Jumat, 3 Maret 2023	Perancangan dan pembelian rangka robot differential drive dan omni-directional	Dimas
3	Rabu, 15 Maret 2023	Perancangan sistem mekanik robot differential drive dan omni-directional	Dimas
4	Senin, 10 April 2023	Perancangan sistem elektronik	Dimas
5	Kamis, 20 April 2023	Pembuatan prototipe robot	Dimas
6	Selasa, 9 Mei 2023	Perakitan komponen dan pengujian awal	Dimas
7	Senin, 15 Mei 2023	Perakitan komponen dan pengujian awal	Dimas
8	Senin, 22 Mei 2023	Perancangan perangkat lunak untuk pengendalian robot	Rifqi

9	Jumat, 2 Juni 2023	Penyempurnaan sistem mekanik	Dimas
10	Senin, 5 Juni 2023	Pengembangan algoritma pengendalian dan navigasi robot	Rifqi
11	Senin, 12 Juni 2023	Integrasi perangkat lunak dengan sistem hardware	Rifqi
12	Senin, 6 Juli 2023	Pengujian dan validasi sistem keseluruhan	Dimas, Rifqi
13	Jumat, 28 Juli 2023	Expo dan presentasi hasil pengujian	Dimas

5.2. Dampak Implementasi Sistem

Implementasi sistem simulasi robot beroda yang kami buat memberikan dampak yang signifikan dalam beberapa bidang krusial: teknologi, pendidikan, dan kenyamanan. Dalam konteks teknologi, sistem ini tidak hanya meningkatkan penggunaan teknologi dalam pembelajaran, tetapi juga memperluas pemahaman mahasiswa terhadap konsep-konsep robotika secara praktis. Melalui simulasi ini, mahasiswa dapat mengembangkan keterampilan teknis mereka dengan cara yang lebih interaktif dan realistis.

Di sisi pendidikan, penggunaan sistem simulasi ini mendorong inovasi dalam metode pengajaran. Mata kuliah atau minat yang berhubungan dengan robotika dapat diajarkan dengan pendekatan yang lebih mendalam dan terstruktur, memungkinkan mahasiswa untuk menghadapi tantangan yang lebih kompleks dalam lingkungan yang aman dan terkendali.

Tidak hanya itu, dalam aspek kenyamanan, sistem ini menghilangkan hambatan fisik yang mungkin terjadi dalam praktik langsung dengan robot. Dengan demikian, mahasiswa dapat belajar tanpa adanya kekhawatiran akan keselamatan atau kerusakan perangkat, meningkatkan kenyamanan dan fleksibilitas dalam pengalaman pembelajaran mereka.

Secara keseluruhan, implementasi sistem simulasi robot beroda tidak hanya memberikan manfaat dalam pengembangan teknologi dan pendidikan, tetapi juga memperkuat kenyamanan dan

keamanan dalam pembelajaran robotika, menciptakan lingkungan yang mendukung untuk eksplorasi dan inovasi.

BAB 6. HASIL PENGUKURAN DAN ANALISIS.

6.1 Kesimpulan

Sistem dari tugas akhir yang telah dibuat cukup memenuhi sebagian spesifikasi yang telah diusulkan dengan beberapa penyesuaian atas sistem yang dibuat. Robot yang dirancang menggunakan sistem differential drive menunjukkan performa yang baik dalam pengujian di lapangan berukuran 3x2 meter. Namun pengujian yang dilakukan masih hanya pergerakan lurus setelah diberi perintah. Hasil yang didapatkan dari pembacaan yang diberikan robot sudah benar sesuai dengan hasil aktual. Validasi atas nilai ini kami sesuaikan menggunakan meteran dengan nilai asli yang terbaca. Hal ini juga didukung dengan penggunaan teknologi seperti marker ArUco dan kamera eksternal dalam proses pengukuran dan evaluasi yang terbukti efektif dalam memastikan akurasi dan presisi sistem yang dikembangkan.

Tujuan yang belum tercapai pada tugas akhir ini ialah menciptakan antarmuka pengguna. Hal ini masih perlu dibuat dan dikembangkan untuk kemudahan penggunaan sistem simulator robot beroda yang dibuat. Antarmuka ini penting untuk mempermudah penggunaan sistem simulator robot beroda yang telah dirancang. Dengan antarmuka yang tepat, pengguna dapat dengan mudah mengontrol dan memantau robot serta melihat data hasil pengukuran secara intuitif. Selain pengembangan antarmuka, langkah selanjutnya yang perlu diperhatikan adalah pengujian lebih lanjut terhadap berbagai jenis pergerakan dan fungsi robot, untuk memastikan kesesuaian dengan semua spesifikasi yang telah diusulkan. Dengan melakukan langkah ini, diharapkan sistem yang dikembangkan dapat memberikan solusi yang handal dan efektif sesuai dengan kebutuhan aplikasi yang dituju.

6.2 Saran

Untuk meningkatkan kualitas dan fungsionalitas sistem yang telah dikembangkan, berikut beberapa rekomendasi yang dapat dipertimbangkan:

1. Pengembangan Antarmuka Pengguna (User Interface)

   Menyelesaikan dan pengembangan antarmuka pengguna untuk sistem simulator robot. Memastikan antarmuka ini dirancang secara intuitif agar pengguna dapat dengan mudah

mengontrol robot, memantau status operasional, serta melihat data hasil pengukuran dengan jelas dan akurat.

2. Pengujian Lanjutan

   Melakuan pengujian lebih lanjut terhadap berbagai jenis pergerakan dan fungsi robot, termasuk manuver seperti putar balik (turning) dan navigasi rute yang kompleks. Hal ini akan memastikan bahwa sistem dapat menangani berbagai skenario operasional dengan baik sesuai dengan spesifikasi yang telah ditetapkan.

3. Optimisasi Performa

   Pengoptimalan performa robot dalam hal kecepatan, akurasi navigasi, dan respons terhadap perintah pengguna. Tinjau kembali penggunaan sensor dan aktuator untuk memastikan keandalan dan efisiensi dalam setiap operasi.

4. Pengembangan Modul Tambahan

   Mengembangkan modul tambahan seperti sistem pengendalian otomatis (autonomous control) atau integrasi dengan platform sensor yang lebih canggih untuk meningkatkan kemampuan adaptasi dan respons sistem terhadap lingkungan.

DAFTAR PUSTAKA

[1] W. Jatmiko *et al.*, *Robotika Teori Dan Aplikasi*, 1st ed. Sukabumi: Fakultas Ilmu Komputer, 2012. pp. 31–98. [Online]. Available: https://www.researchgate.net/profile/Wisnu-Jatmiko/publication/305769100_Robotika_Teori_dan_Aplikasi/links/57a0664608aec29aed23f914/Robotika-Teori-dan-Aplikasi.pdf. [1 Accessed 2023].

[2] M. A. Kamel and Y. Zhang, "Developments and Challenges in Wheeled Mobile Robot Control," 10th Int. Conf. Intell. Unmanned Syst., no. October 2015, pp. 1–7, 2014, doi: 10.13140/RG.2.1.3595.3363.[Online]. Available: https://www.researchgate.net/publication/280578831_Developments_and_Challenges_in_Wheeled_Mobile_Robot_Control (Accessed 01 October 2023).

[3] R. Vestman, "A Comparative Study of Omnidirectional and Differential Drive Systems for Mobile Manipulator Robots," vol. 3. KTH Royal Institute of Technology, pp. 12–30, 2023, [Online]. Available: http://kth.diva-portal.org/smash/get/diva2:1783655/FULLTEXT01.pdf. (Accessed 05 October 2023).

[4] A. De Luca, G. Oriolo, and M. Vendittelli, "Control of Wheeled Mobile Robots: An Experimental Overview," in *Ramsete: Articulated and Mobile Robotics for Services and Technologies*, 1st ed., P. V. Salvatore Nicosia, Bruno Siciliano, Antonio Bicchi, Ed. Springer Berlin, Heidelberg, 2001, pp. 181–226.

[5] X. Yin and L. Pan, "Enhancing trajectory tracking accuracy for industrial robot with robust adaptive control," *Robot. Comput. Integr. Manuf.*, vol. 51, pp. 97–102, 2018, [Online]. Available : https://doi.org/10.1016/j.rcim.2017.11.007 (Accessed 06 October 2023).

[6] F. Rubio, F. Valero, and C. Llopis-Albert, "A review of mobile robots: Concepts, methods, theoretical framework, and applications," *Int. J. Adv. Robot. Syst.*, vol. 16, no. 2, pp. 1–22, 2019, doi: 10.1177/1729881419839596. [Online]. Available : https://journals.sagepub.com/doi/epub/10.1177/1729881419839596 (Accessed 06 October 2023).

[7] M. Arief "Riset Robotika di Teknik Elektro Unpad : Kontribusi untuk Kemajuan Teknologi Indonesia." unpad.ac.id. https://www.unpad.ac.id/2023/06/riset-robotika-di-teknik-elektro-unpad-kontribusi-untuk-kemajuan-teknologi-indonesia/ (Accessed 07 October 2023)

LAMPIRAN – LAMPIRAN

1. Lampiran 1 (“camera_publisher.py”)

```python
#!/usr/bin/env python3

import rospy
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import cv2

def main():
    # Inisialisasi node
    rospy.init_node('camera_sensor_publisher', anonymous=True)

    # Membuat publisher
    pub = rospy.Publisher('video_topic', Image, queue_size=10)

    # Mengatur video capture
    cap = cv2.VideoCapture(4)  # Ganti dengan ID kamera yang sesuai
    bridge = CvBridge()
    rate = rospy.Rate(30)  # 30 Hz

    while not rospy.is_shutdown():
        ret, frame = cap.read()
        if not ret:
            continue

        # Konversi frame ke ROS Image message
        msg = bridge.cv2_to_imgmsg(frame, encoding="bgr8")
        pub.publish(msg)
        rospy.loginfo("Video frame captured and published")
        rate.sleep()

    cap.release()

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

2. Lampiran 2 (“lapangan_calibration.py”)

```python
#!/usr/bin/env python

import rospy
import cv2
from cv2 import aruco
import numpy as np
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError
from marker_reader.msg import RobotPosition, RobotPositions
```

```
class LapanganCalibration:
    def __init__(self):
        rospy.loginfo("Inisialisasi Lapangan Calibration Node")
        self.bridge = CvBridge()  # Membuat objek CvBridge untuk konversi antara ROS Image dan
OpenCV Image
        self.image_sub = rospy.Subscriber("video_topic", Image, self.image_callback)  #
Subscriber untuk topik video
        self.aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)  # Menggunakan dictionary
ArUco 6x6
        self.parameters = aruco.DetectorParameters_create()  # Parameter default untuk deteksi
ArUco

        # Matriks dan distorsi kamera (hasil kalibrasi)
        self.camera_matrix = np.array([[9009.10085, 0, 411.009725],  # fx, 0, cx
                                       [0, 6233.08793, 508.473989],  # 0, fy, cy
                                       [0, 0, 1]])                   # 0, 0, 1
        self.dist_coeffs = np.array([11.7752601, -2109.97792, 0.0963147492, 0.0510228698, -
24.1195436])  # Koefisien distorsi kamera

        # Posisi marker di lapangan (x, y)
        self.marker_positions = {
            1: np.array([0.075, 0.075]),  # Marker 1 di (0.075,0.075)
            2: np.array([2.975, 0.075]),  # Marker 2 di (2.975,0.075)
            3: np.array([2.975, 1.975]),  # Marker 3 di (2.975,1.975)
            4: np.array([0.075, 1.975])   # Marker 4 di (0.075,1.975)
        }

        # Mapping antara marker ID dan nama robot
        self.robot_markers = {
            11: "robot 1",
            22: "robot 2",
            33: "robot 3",
            44: "omni_robot"
        }

        # Publisher untuk posisi robot
        self.robot_positions_pub = rospy.Publisher("/robot_positions", RobotPositions,
queue_size=10)
        rospy.loginfo("LapanganCalibration Node Initialized")

        # Inisialisasi matriks homografi
        self.homography_matrix = None

    def image_callback(self, data):
        rospy.loginfo("Menerima gambar")
        try:
            cv_image = self.bridge.imgmsg_to_cv2(data, "bgr8")  # Konversi dari ROS Image ke
OpenCV Image
        except CvBridgeError as e:
            rospy.logerr(e)
            return

        gray = cv2.cvtColor(cv_image, cv2.COLOR_BGR2GRAY)  # Konversi gambar ke grayscale
        corners, ids, rejected = aruco.detectMarkers(gray, self.aruco_dict,
parameters=self.parameters)  # Deteksi marker ArUco
```

```
        if ids is not None:
            ids_list = ids.flatten().tolist()  # Flatten dan konversi ids ke list
            markers = []
            for i, marker_id in enumerate(ids_list):
                rvec, tvec, _ = aruco.estimatePoseSingleMarkers(corners[i], 0.05,
self.camera_matrix, self.dist_coeffs)  # Estimasi pose marker
                markers.append((marker_id, corners[i], rvec, tvec))  # Tambahkan marker ke
list
            markers.sort(key=lambda x: x[0])  # Urutkan markers berdasarkan id

            if self.homography_matrix is None:
                self.calculate_homography(markers)  # Hitung matriks homografi jika belum ada

            robot_positions_msg = RobotPositions()
            robot_positions_msg.robot_positions = []
            for marker in markers:
                marker_id, corner, rvec, tvec = marker
                cv2.drawFrameAxes(cv_image, self.camera_matrix, self.dist_coeffs, rvec, tvec,
0.1)  # Gambar sumbu frame

                if marker_id in self.marker_positions:
                    posisi_lapangan = self.get_position_on_field(marker_id, corner[0][0])  #
Hitung posisi marker di lapangan
                    rospy.loginfo(f"Marker ID {marker_id}: Posisi Lapangan:
{posisi_lapangan}")

                if marker_id in self.robot_markers:
                    robot_position, robot_orientation =
self.get_robot_position_and_orientation(corner[0][0], rvec)  # Hitung posisi dan orientasi
robot
                    rospy.loginfo(f"{self.robot_markers[marker_id]}: Posisi: {robot_position},
Orientasi: {robot_orientation}")
                    robot_position_msg = RobotPosition(
                        robot_id=int(marker_id),
                        x=float(robot_position[0]),
                        y=float(robot_position[1]),
                        yaw=float(robot_orientation)
                    )
                    robot_positions_msg.robot_positions.append(robot_position_msg)  #
Tambahkan posisi robot ke pesan

            rospy.loginfo(f"Robot positions: {robot_positions_msg}")
            self.robot_positions_pub.publish(robot_positions_msg)  # Publish pesan posisi
robot

        cv2.imshow("Image window", cv_image)  # Tampilkan gambar
        cv2.waitKey(3)  # Tunggu 3 ms untuk input keyboard

    def calculate_homography(self, markers):
        src_points = []
        dst_points = []
        for marker in markers:
            marker_id, corner, rvec, tvec = marker
            if marker_id in self.marker_positions:
```

```
                src_points.append(corner[0][0])  # Tambahkan titik sumber dari gambar
                dst_points.append(self.marker_positions[marker_id])  # Tambahkan titik tujuan
di lapangan

        if len(src_points) >= 4:
            src_points = np.array(src_points)
            dst_points = np.array(dst_points)
            self.homography_matrix, _ = cv2.findHomography(src_points, dst_points)  # Hitung
matriks homografi
            rospy.loginfo("Homography matrix calculated")

    def get_position_on_field(self, marker_id, point):
        if self.homography_matrix is not None:
            point_homogeneous = np.append(point, 1).reshape((3, 1))  # Tambahkan 1 untuk
koordinat homogen
            field_position_homogeneous = np.dot(self.homography_matrix, point_homogeneous)  #
Hitung posisi di lapangan
            field_position = field_position_homogeneous[:2] / field_position_homogeneous[2]  #
Ubah kembali ke koordinat Cartesian
            return field_position.flatten()
        return None

    def get_robot_position_and_orientation(self, point, rvec):
        if self.homography_matrix is not None:
            point_homogeneous = np.append(point, 1).reshape((3, 1))  # Tambahkan 1 untuk
koordinat homogen
            robot_position_homogeneous = np.dot(self.homography_matrix, point_homogeneous)  #
Hitung posisi robot
            robot_position = robot_position_homogeneous[:2] / robot_position_homogeneous[2]  #
Ubah kembali ke koordinat Cartesian

            # Hitung rotasi (yaw) dari rvec
            rotation_matrix, _ = cv2.Rodrigues(rvec)
            yaw = np.arctan2(rotation_matrix[1, 0], rotation_matrix[0, 0])  # Hitung yaw dari
matriks rotasi

            return robot_position.flatten(), yaw
        return None, None

def main():
    rospy.init_node('lapangan_calibration', anonymous=True)  # Inisialisasi node ROS
    lc = LapanganCalibration()  # Buat objek LapanganCalibration
    rospy.loginfo("Main function started")
    try:
        rospy.spin()  # Jalankan loop ROS
    except KeyboardInterrupt:
        cv2.destroyAllWindows()  # Tutup semua jendela OpenCV jika ada interupsi keyboard

if __name__ == '__main__':
    main()  # Panggil fungsi main
```

3. Lampiran 3
    - “multi_robot_control_launch”

```
<launch>
    <!-- Start robot state publishers -->
    <group ns="robot1">
        <node name="joint_state_publisher" pkg="joint_state_publisher"
type="joint_state_publisher" />
        <node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher" />
    </group>

    <group ns="robot2">
        <node name="joint_state_publisher" pkg="joint_state_publisher"
type="joint_state_publisher" />
        <node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher" />
    </group>

    <group ns="robot3">
        <node name="joint_state_publisher" pkg="joint_state_publisher"
type="joint_state_publisher" />
        <node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher" />
    </group>

    <group ns="omni_robot">
        <node name="joint_state_publisher" pkg="joint_state_publisher"
type="joint_state_publisher" />
        <node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher" />
    </group>

    <!-- Start MQTT publisher -->
    <node name="mqtt_publisher" pkg="multi_robot_control" type="mqtt_publisher.py"
output="screen" />

    <!-- Start multi robot controller -->
    <node name="multi_robot_controller" pkg="multi_robot_control"
type="multi_robot_controller.py" output="screen" />

    <!-- Start multi_robot_listener -->
    <node name="multi_robot_listener" pkg="multi_robot_control" type="multi_robot_listener.py"
output="screen"/>

</launch>
```

- MultiRobotCmd.msg

```
geometry_msgs/Twist robot1_cmd
                    geometry_msgs/Twist robot2_cmd
```

- RobotControl.msg

```
string robot_id
float64 target_x
float64 target_y
float64 target_theta  # Only used for omni-directional robot
```

- RobotPosition.msg

```
int32 robot_id
float32 x
float32 y
float32 yaw
```

- RobotPositions.msg

```
RobotPosition[] robot_positions
```

- TargetPosition.msg

```
string robot_id
float32 target_x
float32 target_y
float32 target_theta
```

- “mqtt_publisher.py”

```
#!/usr/bin/env python

import rospy
import paho.mqtt.client as mqtt
from multi_robot_control.msg import RobotControl

# Detail broker MQTT
broker_address = "192.168.33.223"  # Alamat broker MQTT
broker_port = 1883  # Port broker MQTT

# Setup client MQTT
mqtt_client = mqtt.Client(client_id="ROS_MQTT_Client", protocol=mqtt.MQTTv311)  #
Inisialisasi client MQTT dengan ID dan protokol tertentu
mqtt_client._callback_api_version = 5  # Set versi API callback

def on_connect(client, userdata, flags, rc):
```

```
    # Callback saat terhubung ke broker MQTT
    if rc == 0:
        rospy.loginfo("Connected to MQTT Broker")  # Jika koneksi berhasil
    else:
        rospy.logwarn("Failed to connect, return code %d\n", rc)  # Jika koneksi gagal

# Set fungsi callback untuk koneksi MQTT
mqtt_client.on_connect = on_connect
mqtt_client.connect(broker_address, broker_port, 60)  # Koneksikan ke broker MQTT
mqtt_client.loop_start()  # Mulai loop MQTT untuk menjaga koneksi tetap hidup

def robot_control_callback(msg):
    # Callback untuk subscriber ROS yang menerima pesan RobotControl
    # Konversi pesan RobotControl menjadi format string
    payload = "{},{},{}".format(msg.target_x, msg.target_y, msg.target_theta)
    topic = "robot/control/{}".format(msg.robot_id)  # Tentukan topik MQTT berdasarkan ID
robot
    # Publikasikan pesan ke topik MQTT
    mqtt_client.publish(topic, payload)
    rospy.loginfo("Published to MQTT topic {}: {}".format(topic, payload))  # Log pesan yang
dipublikasikan

def main():
    rospy.init_node('mqtt_publisher_node', anonymous=True)  # Inisialisasi node ROS
    rospy.Subscriber("/robot_control", RobotControl, robot_control_callback)  # Buat
subscriber untuk topik /robot_control
    rospy.spin()  # Jalankan loop ROS

if __name__ == '__main__':
    main()  # Panggil fungsi main
```

- “multi_robot_controller.py”

```
#!/usr/bin/env python

import rospy
from geometry_msgs.msg import Twist
from marker_reader.msg import RobotPositions, RobotPosition
from multi_robot_control.msg import TargetPosition, RobotControl
import paho.mqtt.client as mqtt
import numpy as np

# Deklarasi global untuk MQTT client dan publisher
mqtt_clients = {}
pub = None

robot_positions = {}  # Menyimpan posisi robot
target_positions = {}  # Menyimpan posisi target
TOLERANCE = 0.1  # Toleransi jarak untuk menghentikan robot

# Fungsi callback saat terhubung ke broker MQTT
def on_connect(client, userdata, flags, rc):
    if rc == 0:
```

```
        rospy.loginfo(f"Connected to MQTT Broker successfully: {client._client_id.decode()}")
    else:
        rospy.logwarn(f"Failed to connect, return code {rc}")

# Fungsi untuk mengirim perintah ke robot
def send_command(twist_msg, robot_id):
    payload = "{},{}".format(twist_msg.linear.x, twist_msg.angular.z)
    topic = f"robot/control/{robot_id}"

    rospy.loginfo(f"Publishing to topic: {topic} with payload: {payload}")

    if robot_id in mqtt_clients:
        result = mqtt_clients[robot_id].publish(topic, payload)
        rospy.loginfo(f"Sent command to {robot_id}: {payload}, result: {result.rc}")
    else:
        rospy.logwarn(f"MQTT client for {robot_id} not found")

# Fungsi untuk menghitung kecepatan robot differential drive
def calculate_velocity_unicycle(target_x, target_y, current_x, current_y, theta):
    k_linear = 0.5  # Koefisien kecepatan linear
    k_angular = 1.0  # Koefisien kecepatan angular

    error_x = target_x - current_x  # Error pada sumbu x
    error_y = target_y - current_y  # Error pada sumbu y

    distance = np.sqrt(error_x**2 + error_y**2)  # Jarak Euclidean ke target
    angle_to_target = np.arctan2(error_y, error_x)  # Sudut ke target relatif terhadap sumbu x

    linear_velocity = k_linear * distance  # Kecepatan linear dihitung berdasarkan jarak dan
koefisien
    angular_velocity = k_angular * (angle_to_target - theta)  # Kecepatan angular dihitung
berdasarkan sudut dan koefisien

    twist_msg = Twist()
    twist_msg.linear.x = linear_velocity  # Menetapkan kecepatan linear pada pesan Twist
    twist_msg.angular.z = angular_velocity  # Menetapkan kecepatan angular pada pesan Twist

    return twist_msg

# Fungsi untuk memperbarui perintah untuk robot
def update_command_for_robot(robot_id, current_x, current_y, theta):
    if robot_id in target_positions:
        target_x, target_y = target_positions[robot_id]
        distance = np.sqrt((target_x - current_x)**2 + (target_y - current_y)**2)
        if distance > TOLERANCE:
            twist_msg = calculate_velocity_unicycle(target_x, target_y, current_x, current_y,
theta)
            send_command(twist_msg, robot_id)
        else:
            twist_msg = Twist()
            twist_msg.linear.x = 0
            twist_msg.angular.z = 0
            send_command(twist_msg, robot_id)
            rospy.loginfo(f"Robot {robot_id} has reached the target: ({target_x},
{target_y})")
```

```
# Callback untuk menerima posisi robot
def robot_positions_callback(msg):
    global robot_positions
    robot_positions.clear()  # Membersihkan dictionary posisi robot sebelumnya
    for robot_position in msg.robot_positions:
        robot_positions[robot_position.robot_id] = (robot_position.x, robot_position.y,
robot_position.yaw)
    rospy.loginfo(f"Updated robot positions: {robot_positions}")

    # Memperbarui perintah untuk setiap robot berdasarkan posisinya
    for robot_id, (current_x, current_y, yaw) in robot_positions.items():
        update_command_for_robot(robot_id, current_x, current_y, yaw)

# Callback untuk menerima posisi target
def target_position_callback(msg):
    global target_positions
    target_positions[msg.robot_id] = (msg.target_x, msg.target_y)
    rospy.loginfo(f"Updated target position for {msg.robot_id}: ({msg.target_x},
{msg.target_y})")

# Fungsi untuk mengatur dan memulai client MQTT
def setup_mqtt_clients():
    global mqtt_clients

    # Daftar robot
    robot_ids = ["robot1", "robot2", "robot3", "omni_robot"]

    # Membuat client MQTT untuk setiap robot
    for robot_id in robot_ids:
        client = mqtt.Client(client_id=robot_id, protocol=mqtt.MQTTv311)
        client.on_connect = on_connect
        client.connect("192.168.33.223", 1883, 60)
        client.loop_start()  # Memulai loop untuk client MQTT
        mqtt_clients[robot_id] = client

def main():
    global pub
    rospy.init_node('robot_controller', anonymous=True)

    # Menginisialisasi client MQTT untuk masing-masing robot
    setup_mqtt_clients()

    pub = rospy.Publisher('/robot_control', RobotControl, queue_size=10)

    # Menyiapkan subscriber untuk menerima posisi robot dan posisi target
    rospy.Subscriber("/robot_positions", RobotPositions, robot_positions_callback)
    rospy.Subscriber("/target_position", TargetPosition, target_position_callback)

    rospy.spin()  # Menjalankan loop ROS

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

4. Lampiran 4
    - robot1.ino

```
#include <WiFi.h>
#include <PubSubClient.h>

// Update these with your network credentials
const char* ssid = "rifqizul";
const char* password = "rifqizul_";

// Update the MQTT Broker address
const char* mqtt_server = "192.168.33.223";

// Define motor control pins
int enableRightMotor = 12;
int rightMotorPin1 = 25;
int rightMotorPin2 = 33;

int enableLeftMotor = 14;
int leftMotorPin1 = 27;
int leftMotorPin2 = 26;

WiFiClient espClient;
PubSubClient client(espClient);

float current_x = 0.0;
float current_y = 0.0;

void setup_wifi() {
  delay(10);
  Serial.begin(115200);
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void controlMotors(float linear_velocity, float angular_velocity) {
  int leftMotorSpeed = linear_velocity - angular_velocity;
  int rightMotorSpeed = linear_velocity + angular_velocity;

  leftMotorSpeed = constrain(leftMotorSpeed, -255, 255);
  rightMotorSpeed = constrain(rightMotorSpeed, -255, 255);
```

```
  Serial.print("Setting Left Motor Speed to: ");
  Serial.println(leftMotorSpeed);
  Serial.print("Setting Right Motor Speed to: ");
  Serial.println(rightMotorSpeed);

  if (leftMotorSpeed > 0) {
    digitalWrite(leftMotorPin1, HIGH);
    digitalWrite(leftMotorPin2, LOW);
    analogWrite(enableLeftMotor, leftMotorSpeed);
  } else {
    digitalWrite(leftMotorPin1, LOW);
    digitalWrite(leftMotorPin2, HIGH);
    analogWrite(enableLeftMotor, -leftMotorSpeed);
  }

  if (rightMotorSpeed > 0) {
    digitalWrite(rightMotorPin1, HIGH);
    digitalWrite(rightMotorPin2, LOW);
    analogWrite(enableRightMotor, rightMotorSpeed);
  } else {
    digitalWrite(rightMotorPin1, LOW);
    digitalWrite(rightMotorPin2, HIGH);
    analogWrite(enableRightMotor, -rightMotorSpeed);
  }

  Serial.print("Linear velocity: ");
  Serial.print(linear_velocity);
  Serial.print(" Angular velocity: ");
  Serial.println(angular_velocity);
}

void sendPosition() {
  String payload = String(current_x) + "," + String(current_y);
  client.publish("robot/position/robot1", payload.c_str());
}

void calculateAndSendVelocity(float target_x, float target_y) {
  float distance = sqrt(pow(target_x - current_x, 2) + pow(target_y - current_y, 2));
  float angle_to_target = atan2(target_y - current_y, target_x - current_x);

  // Control gains
  float k_linear = 100.0; // Adjusted to provide higher speeds
  float k_angular = 50.0; // Adjusted to provide higher speeds

  // Calculate linear and angular velocities
  float linear_velocity = k_linear * distance;
  float angular_velocity = k_angular * angle_to_target;

  // Cap the velocities
  linear_velocity = min(linear_velocity, 255.0f);
  angular_velocity = min(angular_velocity, 255.0f);

  // Send the velocities to the motors
  controlMotors(linear_velocity, angular_velocity);
```

```
  // Update current position (simulation for now, replace with actual sensor data)
  current_x += linear_velocity * 0.1;
  current_y += linear_velocity * 0.1;

  // Send position
  sendPosition();
}

void handlePositionMessage(String message) {
  Serial.print("Position Message: ");
  Serial.println(message);
  // Assuming the message format is "current_x,current_y"
  int comma_index = message.indexOf(',');
  if (comma_index == -1) {
    Serial.println("Invalid position format");
    return;
  }

  current_x = message.substring(0, comma_index).toFloat();
  current_y = message.substring(comma_index + 1).toFloat();

  Serial.print("Updated Current X: ");
  Serial.println(current_x);
  Serial.print("Updated Current Y: ");
  Serial.println(current_y);
}

void handleCommandMessage(String message) {
  Serial.print("Command Message: ");
  Serial.println(message);
  // Assuming the message format is "target_x,target_y"
  int comma_index = message.indexOf(',');
  if (comma_index == -1) {
    Serial.println("Invalid command format");
    return;
  }

  float target_x = message.substring(0, comma_index).toFloat();
  float target_y = message.substring(comma_index + 1).toFloat();

  Serial.print("Parsed Target X: ");
  Serial.println(target_x);
  Serial.print("Parsed Target Y: ");
  Serial.println(target_y);

  // Calculate velocities based on target position
  calculateAndSendVelocity(target_x, target_y);

  Serial.println("Command executed successfully");
}

void callback(char* topic, byte* message, unsigned int length) {
  String topicStr = String(topic);
  String payloadStr = "";
```

```
  for (int i = 0; i < length; i++) {
    payloadStr += (char)message[i];
  }

  Serial.print("Callback invoked on topic: ");
  Serial.println(topicStr);
  Serial.print("Message arrived: ");
  Serial.println(payloadStr);

  if (topicStr == "robot/position/robot1") {
    handlePositionMessage(payloadStr);
  } else if (topicStr == "robot/control/robot1") {
    handleCommandMessage(payloadStr);
  }
}

void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect("ESP32Client2")) {  // Mengubah nama client menjadi ESP32Client2 untuk
robot1
      Serial.println("connected");
      client.subscribe("robot/control/robot1");  // Mengubah topik menjadi
robot/control/robot1
      client.subscribe("robot/position/robot1");  // Tambahkan topik untuk posisi
      Serial.println("Subscribed to robot/control/robot1 and robot/position/robot1");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}

void setup() {
  pinMode(leftMotorPin1, OUTPUT);
  pinMode(leftMotorPin2, OUTPUT);
  pinMode(rightMotorPin1, OUTPUT);
  pinMode(rightMotorPin2, OUTPUT);

  pinMode(enableLeftMotor, OUTPUT);
  pinMode(enableRightMotor, OUTPUT);

  setup_wifi();
  client.setServer(mqtt_server, 1883);

  // Set the callback to the main callback function
  client.setCallback(callback);

  // Subscribe to the topics
  client.subscribe("robot/control/robot1");
  client.subscribe("robot/position/robot1");
}
```

```
void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
}
```

- robot2.ino

```
#include <WiFi.h>
#include <PubSubClient.h>

// Update these with your network credentials
const char* ssid = "rifqizul";
const char* password = "rifqizul_";

// Update the MQTT Broker address
const char* mqtt_server = "192.168.33.223";

// Define motor control pins
int enableRightMotor = 12;
int rightMotorPin1 = 25;
int rightMotorPin2 = 33;

int enableLeftMotor = 14;
int leftMotorPin1 = 27;
int leftMotorPin2 = 26;

WiFiClient espClient;
PubSubClient client(espClient);

float current_x = 0.0;
float current_y = 0.0;

void setup_wifi() {
  delay(10);
  Serial.begin(115200);
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
```

```
}

void controlMotors(float linear_velocity, float angular_velocity) {
  int leftMotorSpeed = linear_velocity - angular_velocity;
  int rightMotorSpeed = linear_velocity + angular_velocity;

  leftMotorSpeed = constrain(leftMotorSpeed, -255, 255);
  rightMotorSpeed = constrain(rightMotorSpeed, -255, 255);

  Serial.print("Setting Left Motor Speed to: ");
  Serial.println(leftMotorSpeed);
  Serial.print("Setting Right Motor Speed to: ");
  Serial.println(rightMotorSpeed);

  if (leftMotorSpeed > 0) {
    digitalWrite(leftMotorPin1, HIGH);
    digitalWrite(leftMotorPin2, LOW);
    analogWrite(enableLeftMotor, leftMotorSpeed);
  } else {
    digitalWrite(leftMotorPin1, LOW);
    digitalWrite(leftMotorPin2, HIGH);
    analogWrite(enableLeftMotor, -leftMotorSpeed);
  }

  if (rightMotorSpeed > 0) {
    digitalWrite(rightMotorPin1, HIGH);
    digitalWrite(rightMotorPin2, LOW);
    analogWrite(enableRightMotor, rightMotorSpeed);
  } else {
    digitalWrite(rightMotorPin1, LOW);
    digitalWrite(rightMotorPin2, HIGH);
    analogWrite(enableRightMotor, -rightMotorSpeed);
  }

  Serial.print("Linear velocity: ");
  Serial.print(linear_velocity);
  Serial.print(" Angular velocity: ");
  Serial.println(angular_velocity);
}

void sendPosition() {
  String payload = String(current_x) + "," + String(current_y);
  client.publish("robot/position/robot2", payload.c_str());
}

void calculateAndSendVelocity(float target_x, float target_y) {
  float distance = sqrt(pow(target_x - current_x, 2) + pow(target_y - current_y, 2));
  float angle_to_target = atan2(target_y - current_y, target_x - current_x);

  // Control gains
  float k_linear = 100.0; // Adjusted to provide higher speeds
  float k_angular = 50.0; // Adjusted to provide higher speeds

  // Calculate linear and angular velocities
  float linear_velocity = k_linear * distance;
```

```
  float angular_velocity = k_angular * angle_to_target;

  // Cap the velocities
  linear_velocity = min(linear_velocity, 255.0f);
  angular_velocity = min(angular_velocity, 255.0f);

  // Send the velocities to the motors
  controlMotors(linear_velocity, angular_velocity);

  // Update current position (simulation for now, replace with actual sensor data)
  current_x += linear_velocity * 0.1;
  current_y += linear_velocity * 0.1;

  // Send position
  sendPosition();
}

void handlePositionMessage(String message) {
  Serial.print("Position Message: ");
  Serial.println(message);
  // Assuming the message format is "current_x,current_y"
  int comma_index = message.indexOf(',');
  if (comma_index == -1) {
    Serial.println("Invalid position format");
    return;
  }

  current_x = message.substring(0, comma_index).toFloat();
  current_y = message.substring(comma_index + 1).toFloat();

  Serial.print("Updated Current X: ");
  Serial.println(current_x);
  Serial.print("Updated Current Y: ");
  Serial.println(current_y);
}

void handleCommandMessage(String message) {
  Serial.print("Command Message: ");
  Serial.println(message);
  // Assuming the message format is "target_x,target_y"
  int comma_index = message.indexOf(',');
  if (comma_index == -1) {
    Serial.println("Invalid command format");
    return;
  }

  float target_x = message.substring(0, comma_index).toFloat();
  float target_y = message.substring(comma_index + 1).toFloat();

  Serial.print("Parsed Target X: ");
  Serial.println(target_x);
  Serial.print("Parsed Target Y: ");
  Serial.println(target_y);

  // Calculate velocities based on target position
```

```
  calculateAndSendVelocity(target_x, target_y);

  Serial.println("Command executed successfully");
}

void callback(char* topic, byte* message, unsigned int length) {
  String topicStr = String(topic);
  String payloadStr = "";

  for (int i = 0; i < length; i++) {
    payloadStr += (char)message[i];
  }

  Serial.print("Callback invoked on topic: ");
  Serial.println(topicStr);
  Serial.print("Message arrived: ");
  Serial.println(payloadStr);

  if (topicStr == "robot/position/robot2") {
    handlePositionMessage(payloadStr);
  } else if (topicStr == "robot/control/robot2") {
    handleCommandMessage(payloadStr);
  }
}

void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect("ESP32Client2")) {  // Mengubah nama client menjadi ESP32Client2 untuk
robot2
      Serial.println("connected");
      client.subscribe("robot/control/robot2");  // Mengubah topik menjadi
robot/control/robot2
      client.subscribe("robot/position/robot2");  // Tambahkan topik untuk posisi
      Serial.println("Subscribed to robot/control/robot2 and robot/position/robot2");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}

void setup() {
  pinMode(leftMotorPin1, OUTPUT);
  pinMode(leftMotorPin2, OUTPUT);
  pinMode(rightMotorPin1, OUTPUT);
  pinMode(rightMotorPin2, OUTPUT);

  pinMode(enableLeftMotor, OUTPUT);
  pinMode(enableRightMotor, OUTPUT);

  setup_wifi();
  client.setServer(mqtt_server, 1883);
```

```
  // Set the callback to the main callback function
  client.setCallback(callback);

  // Subscribe to the topics
  client.subscribe("robot/control/robot2");
  client.subscribe("robot/position/robot2");
}

void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
}
```

- robot3.ino

```
#include <WiFi.h>
#include <PubSubClient.h>

// Update these with your network credentials
const char* ssid = "rifqizul";
const char* password = "rifqizul_";

// Update the MQTT Broker address
const char* mqtt_server = "192.168.33.223";

// Define motor control pins
int enableRightMotor = 12;
int rightMotorPin1 = 25;
int rightMotorPin2 = 33;

int enableLeftMotor = 14;
int leftMotorPin1 = 27;
int leftMotorPin2 = 26;

WiFiClient espClient;
PubSubClient client(espClient);

float current_x = 0.0;
float current_y = 0.0;
float target_x = 0.0;
float target_y = 0.0;

const float TOLERANCE = 0.1;  // Tolerance distance to stop the robot

void setup_wifi() {
  delay(10);
  Serial.begin(115200);
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
```

```
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void controlMotors(float linear_velocity, float angular_velocity) {
  int leftMotorSpeed = linear_velocity - angular_velocity;
  int rightMotorSpeed = linear_velocity + angular_velocity;

  leftMotorSpeed = constrain(leftMotorSpeed, -255, 255);
  rightMotorSpeed = constrain(rightMotorSpeed, -255, 255);

  Serial.print("Setting Left Motor Speed to: ");
  Serial.println(leftMotorSpeed);
  Serial.print("Setting Right Motor Speed to: ");
  Serial.println(rightMotorSpeed);

  if (leftMotorSpeed > 0) {
    digitalWrite(leftMotorPin1, HIGH);
    digitalWrite(leftMotorPin2, LOW);
    analogWrite(enableLeftMotor, leftMotorSpeed);
  } else {
    digitalWrite(leftMotorPin1, LOW);
    digitalWrite(leftMotorPin2, HIGH);
    analogWrite(enableLeftMotor, -leftMotorSpeed);
  }

  if (rightMotorSpeed > 0) {
    digitalWrite(rightMotorPin1, HIGH);
    digitalWrite(rightMotorPin2, LOW);
    analogWrite(enableRightMotor, rightMotorSpeed);
  } else {
    digitalWrite(rightMotorPin1, LOW);
    digitalWrite(rightMotorPin2, HIGH);
    analogWrite(enableRightMotor, -rightMotorSpeed);
  }

  Serial.print("Linear velocity: ");
  Serial.print(linear_velocity);
  Serial.print(" Angular velocity: ");
  Serial.println(angular_velocity);
}

void stopMotors() {
  analogWrite(enableLeftMotor, 0);
  analogWrite(enableRightMotor, 0);
```

```
  Serial.println("Motors stopped.");
}

void sendPosition() {
  String payload = String(current_x) + "," + String(current_y);
  client.publish("robot/position/robot3", payload.c_str());
}

void calculateAndSendVelocity() {
  float distance = sqrt(pow(target_x - current_x, 2) + pow(target_y - current_y, 2));

  if (distance < TOLERANCE) {
    stopMotors();
    return;
  }

  float angle_to_target = atan2(target_y - current_y, target_x - current_x);

  // Control gains
  float k_linear = 100.0; // Adjusted to provide higher speeds
  float k_angular = 50.0; // Adjusted to provide higher speeds

  // Calculate linear and angular velocities
  float linear_velocity = k_linear * distance;
  float angular_velocity = k_angular * angle_to_target;

  // Cap the velocities
  linear_velocity = min(linear_velocity, 255.0f);
  angular_velocity = min(angular_velocity, 255.0f);

  // Send the velocities to the motors
  controlMotors(linear_velocity, angular_velocity);

  // Update current position (simulation for now, replace with actual sensor data)
  current_x += linear_velocity * 0.1;
  current_y += linear_velocity * 0.1;

  // Send position
  sendPosition();
}

void handlePositionMessage(String message) {
  Serial.print("Position Message: ");
  Serial.println(message);
  int comma_index = message.indexOf(',');
  if (comma_index == -1) {
    Serial.println("Invalid position format");
    return;
  }

  current_x = message.substring(0, comma_index).toFloat();
  current_y = message.substring(comma_index + 1).toFloat();

  Serial.print("Updated Current X: ");
  Serial.println(current_x);
```

```
  Serial.print("Updated Current Y: ");
  Serial.println(current_y);
}

void handleCommandMessage(String message) {
  Serial.print("Command Message: ");
  Serial.println(message);
  int comma_index = message.indexOf(',');
  if (comma_index == -1) {
    Serial.println("Invalid command format");
    return;
  }

  target_x = message.substring(0, comma_index).toFloat();
  target_y = message.substring(comma_index + 1).toFloat();

  Serial.print("Parsed Target X: ");
  Serial.println(target_x);
  Serial.print("Parsed Target Y: ");
  Serial.println(target_y);

  calculateAndSendVelocity();
  Serial.println("Command executed successfully");
}

void callback(char* topic, byte* message, unsigned int length) {
  String topicStr = String(topic);
  String payloadStr = "";

  for (int i = 0; i < length; i++) {
    payloadStr += (char)message[i];
  }

  Serial.print("Callback invoked on topic: ");
  Serial.println(topicStr);
  Serial.print("Message arrived: ");
  Serial.println(payloadStr);

  if (topicStr == "robot/position/robot3") {
    handlePositionMessage(payloadStr);
  } else if (topicStr == "robot/control/robot3") {
    handleCommandMessage(payloadStr);
  }
}

void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect("ESP32Client1")) {
      Serial.println("connected");
      client.subscribe("robot/control/robot3");
      client.subscribe("robot/position/robot3");
      Serial.println("Subscribed to robot/control/robot3 and robot/position/robot3");
    } else {
      Serial.print("failed, rc=");
```

```
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}

void setup() {
  pinMode(leftMotorPin1, OUTPUT);
  pinMode(leftMotorPin2, OUTPUT);
  pinMode(rightMotorPin1, OUTPUT);
  pinMode(rightMotorPin2, OUTPUT);

  pinMode(enableLeftMotor, OUTPUT);
  pinMode(enableRightMotor, OUTPUT);

  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
  client.subscribe("robot/control/robot3");
  client.subscribe("robot/position/robot3");
}

void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
}
```

- omni_robot.ino

```
#include <WiFi.h>
#include <PubSubClient.h>

// Update these with your network credentials
const char* ssid = "rifqizul";
const char* password = "rifqizul_";

// Update the MQTT Broker address
const char* mqtt_server = "192.168.33.223";

// Define motor control pins for omni-wheel robot
int enableMotor1 = 12;
int motor1Pin1 = 25;
int motor1Pin2 = 33;

int enableMotor2 = 14;
int motor2Pin1 = 27;
int motor2Pin2 = 26;

int enableMotor3 = 13;
int motor3Pin1 = 32;
```

```
int motor3Pin2 = 15;

int enableMotor4 = 16;
int motor4Pin1 = 2;
int motor4Pin2 = 4;

WiFiClient espClient;
PubSubClient client(espClient);

float current_x = 0.0;
float current_y = 0.0;
float target_x = 0.0;
float target_y = 0.0;

const float TOLERANCE = 0.1;  // Tolerance distance to stop the robot

void setup_wifi() {
 delay(10);
 Serial.begin(115200);
 Serial.println();
 Serial.print("Connecting to ");
 Serial.println(ssid);

 WiFi.begin(ssid, password);

 while (WiFi.status() != WL_CONNECTED) {
   delay(500);
   Serial.print(".");
 }

 Serial.println("");
 Serial.println("WiFi connected");
 Serial.println("IP address: ");
 Serial.println(WiFi.localIP());
}

void controlMotors(float linear_velocity_x, float linear_velocity_y, float angular_velocity)
{
 int motor1Speed = linear_velocity_y + linear_velocity_x - angular_velocity;
 int motor2Speed = linear_velocity_y - linear_velocity_x + angular_velocity;
 int motor3Speed = linear_velocity_y + linear_velocity_x + angular_velocity;
 int motor4Speed = linear_velocity_y - linear_velocity_x - angular_velocity;

 motor1Speed = constrain(motor1Speed, -255, 255);
 motor2Speed = constrain(motor2Speed, -255, 255);
 motor3Speed = constrain(motor3Speed, -255, 255);
 motor4Speed = constrain(motor4Speed, -255, 255);

 Serial.print("Setting Motor 1 Speed to: ");
 Serial.println(motor1Speed);
 Serial.print("Setting Motor 2 Speed to: ");
 Serial.println(motor2Speed);
 Serial.print("Setting Motor 3 Speed to: ");
 Serial.println(motor3Speed);
 Serial.print("Setting Motor 4 Speed to: ");
```

```
  Serial.println(motor4Speed);

  // Motor 1 control
  if (motor1Speed > 0) {
    digitalWrite(motor1Pin1, HIGH);
    digitalWrite(motor1Pin2, LOW);
    analogWrite(enableMotor1, motor1Speed);
  } else {
    digitalWrite(motor1Pin1, LOW);
    digitalWrite(motor1Pin2, HIGH);
    analogWrite(enableMotor1, -motor1Speed);
  }

  // Motor 2 control
  if (motor2Speed > 0) {
    digitalWrite(motor2Pin1, HIGH);
    digitalWrite(motor2Pin2, LOW);
    analogWrite(enableMotor2, motor2Speed);
  } else {
    digitalWrite(motor2Pin1, LOW);
    digitalWrite(motor2Pin2, HIGH);
    analogWrite(enableMotor2, -motor2Speed);
  }

  // Motor 3 control
  if (motor3Speed > 0) {
    digitalWrite(motor3Pin1, HIGH);
    digitalWrite(motor3Pin2, LOW);
    analogWrite(enableMotor3, motor3Speed);
  } else {
    digitalWrite(motor3Pin1, LOW);
    digitalWrite(motor3Pin2, HIGH);
    analogWrite(enableMotor3, -motor3Speed);
  }

  // Motor 4 control
  if (motor4Speed > 0) {
    digitalWrite(motor4Pin1, HIGH);
    digitalWrite(motor4Pin2, LOW);
    analogWrite(enableMotor4, motor4Speed);
  } else {
    digitalWrite(motor4Pin1, LOW);
    digitalWrite(motor4Pin2, HIGH);
    analogWrite(enableMotor4, -motor4Speed);
  }

  Serial.print("Linear velocity X: ");
  Serial.print(linear_velocity_x);
  Serial.print(" Linear velocity Y: ");
  Serial.print(linear_velocity_y);
  Serial.print(" Angular velocity: ");
  Serial.println(angular_velocity);
}

void stopMotors() {
```

```
  analogWrite(enableMotor1, 0);
  analogWrite(enableMotor2, 0);
  analogWrite(enableMotor3, 0);
  analogWrite(enableMotor4, 0);
  Serial.println("Motors stopped.");
}

void sendPosition() {
  String payload = String(current_x) + "," + String(current_y);
  client.publish("robot/position/omni_robot", payload.c_str());
}

void calculateAndSendVelocity() {
  float distance = sqrt(pow(target_x - current_x, 2) + pow(target_y - current_y, 2));

  if (distance < TOLERANCE) {
    stopMotors();
    return;
  }

  float angle_to_target = atan2(target_y - current_y, target_x - current_x);

  // Control gains
  float k_linear = 100.0; // Adjusted to provide higher speeds
  float k_angular = 50.0; // Adjusted to provide higher speeds

  // Calculate linear and angular velocities
  float linear_velocity_x = k_linear * cos(angle_to_target);
  float linear_velocity_y = k_linear * sin(angle_to_target);
  float angular_velocity = k_angular * angle_to_target;

  // Cap the velocities
  linear_velocity_x = min(linear_velocity_x, 255.0f);
  linear_velocity_y = min(linear_velocity_y, 255.0f);
  angular_velocity = min(angular_velocity, 255.0f);

  // Send the velocities to the motors
  controlMotors(linear_velocity_x, linear_velocity_y, angular_velocity);

  // Update current position (simulation for now, replace with actual sensor data)
  current_x += linear_velocity_x * 0.1;
  current_y += linear_velocity_y * 0.1;

  // Send position
  sendPosition();
}

void handlePositionMessage(String message) {
  Serial.print("Position Message: ");
  Serial.println(message);
  int comma_index = message.indexOf(',');
  if (comma_index == -1) {
    Serial.println("Invalid position format");
    return;
  }
```

```
  current_x = message.substring(0, comma_index).toFloat();
  current_y = message.substring(comma_index + 1).toFloat();

  Serial.print("Updated Current X: ");
  Serial.println(current_x);
  Serial.print("Updated Current Y: ");
  Serial.println(current_y);
}

void handleCommandMessage(String message) {
  Serial.print("Command Message: ");
  Serial.println(message);
  int comma_index = message.indexOf(',');
  if (comma_index == -1) {
    Serial.println("Invalid command format");
    return;
  }

  target_x = message.substring(0, comma_index).toFloat();
  target_y = message.substring(comma_index + 1).toFloat();

  Serial.print("Parsed Target X: ");
  Serial.println(target_x);
  Serial.print("Parsed Target Y: ");
  Serial.println(target_y);

  calculateAndSendVelocity();
  Serial.println("Command executed successfully");
}

void callback(char* topic, byte* message, unsigned int length) {
  String topicStr = String(topic);
  String payloadStr = "";

  for (int i = 0; i < length; i++) {
    payloadStr += (char)message[i];
  }

  Serial.print("Callback invoked on topic: ");
  Serial.println(topicStr);
  Serial.print("Message arrived: ");
  Serial.println(payloadStr);

  if (topicStr == "robot/position/omni_robot") {
    handlePositionMessage(payloadStr);
  } else if (topicStr == "robot/control/omni_robot") {
    handleCommandMessage(payloadStr);
  }
}

void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect("ESP32Client1")) {
```

```
      Serial.println("connected");
      client.subscribe("robot/control/omni_robot");
      client.subscribe("robot/position/omni_robot");
      Serial.println("Subscribed to robot/control/omni_robot and robot/position/omni_robot");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}

void setup() {
  pinMode(motor1Pin1, OUTPUT);
  pinMode(motor1Pin2, OUTPUT);
  pinMode(motor2Pin1, OUTPUT);
  pinMode(motor2Pin2, OUTPUT);
  pinMode(motor3Pin1, OUTPUT);
  pinMode(motor3Pin2, OUTPUT);
  pinMode(motor4Pin1, OUTPUT);
  pinMode(motor4Pin2, OUTPUT);

  pinMode(enableMotor1, OUTPUT);
  pinMode(enableMotor2, OUTPUT);
  pinMode(enableMotor3, OUTPUT);
  pinMode(enableMotor4, OUTPUT);

  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
  client.subscribe("robot/control/omni_robot");
  client.subscribe("robot/position/omni_robot");
}

void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
}
```

5. Lampiran 5

```
import numpy as np
import cv2 as cv
import glob
import os
```

```
# FIND CHESSBOARD CORNERS - objPoints and imgPoints
chessboardSize = (23, 15)  # Ukuran pola papan catur yang baru
frameSize = (720, 1080)

# termination criteria
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)

# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
objp = np.zeros((chessboardSize[0] * chessboardSize[1], 3), np.float32)
objp[:, :2] = np.mgrid[0:chessboardSize[0], 0:chessboardSize[1]].T.reshape(-1, 2)

# Arrays to store object points and image points from all the images.
objPoints = []  # 3d point in real world space
imgPoints = []  # 2d points in image plane.

# Path to calibration images
calibration_images_path = os.path.join(os.getcwd(), 'calibration_images', '*.jpg')
images = glob.glob(calibration_images_path)

print(f"Jumlah gambar yang ditemukan: {len(images)}")

for image in images:
    print(f"Memproses gambar: {image}")
    img = cv.imread(image)
    if img is None:
        print(f"Error: Gambar {image} tidak dapat dimuat.")
        continue
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

    # Tampilkan gambar grayscale
    cv.imshow('Gambar Grayscale', gray)
    cv.waitKey(500)
    cv.imwrite(f'grayscale_{os.path.basename(image)}', gray)  # Simpan gambar grayscale untuk
debugging

    # Find the chess board corners
    ret, corners = cv.findChessboardCorners(gray, chessboardSize, None)

    if ret:
        print(f"Sudut ditemukan di {image}")
        objPoints.append(objp)
        corners2 = cv.cornerSubPix(gray, corners, (11, 11), (-1, -1), criteria)
        imgPoints.append(corners2)

        # Draw and display the corners
        cv.drawChessboardCorners(img, chessboardSize, corners2, ret)
        cv.imshow('Chessboard', img)
        cv.waitKey(500)
    else:
        print(f"Sudut tidak ditemukan di {image}")

cv.destroyAllWindows()

if len(objPoints) > 0 and len(imgPoints) > 0:
    # CALIBRATION
```

```
    ret, cameraMatrix, dist, rvecs, tvecs = cv.calibrateCamera(objPoints, imgPoints,
frameSize, None, None)

    print("Camera calibrated:", ret)
    print("\nCamera matrix:\n", cameraMatrix)
    print("\nDistortion parameters:\n", dist)
    print("\nRotation vectors:\n", rvecs)
    print("\nTranslation vectors:\n", tvecs)

    # UNDISTORTION
    img = cv.imread(images[0])
    h, w = img.shape[:2]
    newCameraMatrix, roi = cv.getOptimalNewCameraMatrix(cameraMatrix, dist, (w, h), 1, (w, h))

    # Undistort
    dst = cv.undistort(img, cameraMatrix, dist, None, newCameraMatrix)

    # Crop the image
    x, y, w, h = roi
    dst = dst[y:y + h, x:x + w]
    cv.imwrite('calibresult.png', dst)

    # Undistort with Remapping
    mapx, mapy = cv.initUndistortRectifyMap(cameraMatrix, dist, None, newCameraMatrix, (w, h),
5)
    dst = cv.remap(img, mapx, mapy, cv.INTER_LINEAR)

    # Crop the image
    x, y, w, h = roi
    dst = dst[y:y + h, x:x + w]
    cv.imwrite('calibresult2.png', dst)

    # Reprojection Error
    mean_error = 0
    for i in range(len(objPoints)):
        imgPoints2, _ = cv.projectPoints(objPoints[i], rvecs[i], tvecs[i], cameraMatrix, dist)
        error = cv.norm(imgPoints[i], imgPoints2, cv.NORM_L2) / len(imgPoints2)
        mean_error += error

    print("\nTotal error: {}".format(mean_error / len(objPoints)))
else:
    print("Tidak ada gambar kalibrasi yang berhasil diproses atau tidak ada sudut yang
terdeteksi.")
```

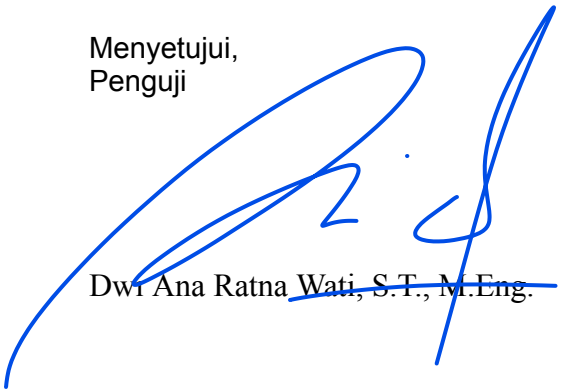
TABEL PERBAIKAN
LAPORAN AKHIR CAPSTONE

MAHASISWA #1 : 20524056 Dimas Fajar Pranaya :
MAHASISWA #2 : 20524158 Rifqi Zul Fahmi
JUDUL/TOPIK : Simulator Pembelajaran Pergerakan Robot: Titik ke Titik dan Pelacakan Trayek pada Differential Drive dan Omni-Directional Robot

No	Saran penguji	Perbaikan oleh mahasiswa	Halaman	Status
1	Daftar Isi: format font, ada daftar pustaka di bagian bawah??	Telah dilakukan perbaikan pada font yang digunakan menjadi sama, menggunakan font *times new roman* dengan ukuran 12 dan menghapus daftar pustaka di bagian bawah setelah lampiran pada daftar isi	6-8	In progress
2	Daftar gambar, tabel??	Telah ditambahkan daftar gambar dan daftar tabel pada laporan	9-11	In progress
3	Persamaan belum ada nomornya, deksripsi dari setiap kuantitas	Telah diberikan nomor pada setiap persamaan yang ada pada laporan berserta pula dengan deskripsinya	25,26,31	In progress
4	Belum nampak keterkaitan antara poin-poin spesifikasi, desain, indikator kinerja dan pengujiannya	Telah dibuat tabel selisih pembacaan marker pada setiap percobaan dengan satuan meter untuk menunjukan indikator kinerja dan pengujiannya serta memberikan penjelasan tambahan keterkaitan antara spesifikasi yang diberikan pada sub bab 5.1.2 dan realisasi desain yang digunakan	86-100	In progress
5				Not started

Yogyakarta, 22 Juli 2024

Menyetujui,
Penguji



Dwi Ana Ratna Wati, S.T., M.Eng.