LAPORAN TUGAS AKHIR

RANCANG BANGUN ALGORITMA KENDALI PERGERAKAN ROBOT OTONOM BERODA BERKONFIGURASI *X-DRIVE* BERBASIS PID DAN KINEMATIKA

Disusun untuk memenuhi sebagian persyaratan memperoleh gelar Sarjana Teknik di Jurusan Teknik Elektro Universitas Jenderal Soedirman

Disusun oleh:

Rafif Susena
H1A021025

KEMENTERIAN PENDIDIKAN TINGGI, SAINS DAN TEKNOLOGI
UNIVERSITAS JENDERAL SOEDIRMAN
FAKULTAS TEKNIK
JURUSAN/PROGRAM STUDI TEKNIK ELEKTRO
PURBALINGGA

2025


UNIVERSITAS JENDERAL SOEDIRMAN
1963

LAPORAN TUGAS AKHIR

RANCANG BANGUN ALGORITMA KENDALI PERGERAKAN ROBOT OTONOM BERODA BERKONFIGURASI X-DRIVE BERBASIS PID DAN KINEMATIKA

Disusun untuk memenuhi sebagian persyaratan memperoleh gelar Sarjana Teknik di Jurusan Teknik Elektro Universitas Jenderal Soedirman

Disusun oleh:

Rafif Susena
H1A021025

KEMENTERIAN PENDIDIKAN TINGGI, SAINS DAN TEKNOLOGI
UNIVERSITAS JENDERAL SOEDIRMAN
FAKULTAS TEKNIK
JURUSAN/PROGRAM STUDI TEKNIK ELEKTRO
PURBALINGGA

2025

HALAMAN PENGESAHAN

Laporan Tugas Akhir dengan Judul:

RANCANG BANGUN ALGORITMA KENDALI PERGERAKAN ROBOT OTONOM BERODA BERKONFIGURASI *X-DRIVE* BERBASIS PID DAN KINEMATIKA

Disusun oleh:

Rafif Susena
H1A021025

Diajukan untuk memenuhi salah satu persyaratan
memperoleh gelar Sarjana Teknik pada
Jurusan/Program Studi Teknik Elektro
Fakultas Teknik
Universitas Jenderal Soedirman

Diterima dan disetujui
Pada Tanggal : _____

Pembimbing I	Pembimbing II
Agung Mubyarto, S.T., M.T.	Ir. Priswanto, S.T., M.Eng.
NIP. 197410062002121001	NIP. 197802192001121001

Mengetahui:
Dekan Fakultas Teknik

Prof. Dr. Eng. Ir. Agus Maryoto, S.T., M.T., IPU.
NIP. 197109202006041001

HALAMAN PERNYATAAN

Dengan ini saya menyatakan bahwa dalam Laporan Tugas Akhir/Laporan Tugas Akhir dengan judul "**RANCANG BANGUN ALGORITMA KENDALI PERGERAKAN ROBOT OTONOM BERODA BERKONFIGURASI *X-DRIVE* BERBASIS PID DAN KINEMATIKA**" ini tidak terdapat karya yang pernah diajukan untuk memperoleh gelar kesarjanaaan di suatu Perguruan Tinggi, dan sepanjang pengetahuan saya juga tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis diacu dalam naskah ini dan disebutkan dalam daftar pustaka.

Purbalingga, 14 Februari 2025

Rafif Susena
NIM H1A021025

HALAMAN MOTTO DAN PERSEMBAHAN

MOTTO

"Tidak ada kesuksesan yang didapat dengan instan, tetapi harus melalui perjuangan dan kerja keras."

PERSEMBAHAN

Laporan Tugas Akhir ini disusun dengan penuh dedikasi dan kerja keras. Penulis ingin mengucapkan terima kasih yang sebesar-besarnya kepada:

1. Allah SWT yang telah memberikan rahmat dan karunia-Nya.
2. Dosen pembimbing penulis yang memberikan arahan, bimbingan, serta masukan yang sangat berharga dalam menyelesaikan tugas akhir ini.
3. Orang tua penulis yang selalu memberikan dukungan dan motivasi dalam menyelesaikan studi.
4. Teman-teman yang selalu memberikan semangat dan dukungan moril.
5. Semua pihak yang tidak dapat disebutkan satu per satu yang telah membantu dan mendukung penulis dalam menyelesaikan tugas akhir ini.

RINGKASAN

RANCANG BANGUN ALGORITMA KENDALI PERGERAKAN ROBOT OTONOM BERODA BERKONFIGURASI *X-DRIVE* BERBASIS PID DAN KINEMATIKA

Rafif Susena

Perkembangan teknologi robotika telah meningkat pesat, terutama dalam aplikasi robot otonom beroda yang digunakan dalam berbagai bidang, termasuk kompetisi robotika seperti Kontes Robot ABU Indonesia. Dalam konteks ini, kemampuan untuk bergerak secara presisi dan akurat dari satu titik ke titik lainnya sangat penting, terutama bagi robot beroda omni dengan konfigurasi *X-Drive* yang memungkinkan pergerakan *omnidirectional*. Penelitian terdahulu menunjukkan keberhasilan implementasi algoritma kontrol PID dalam pengendalian kecepatan dan posisi, namun seringkali tidak mempertimbangkan aspek non-linear seperti efek koriolis dan slip akibat percepatan berlebihan. Hal ini menimbulkan rumusan masalah, yaitu bagaimana merancang model sistem kendali pergerakan yang mampu mengontrol gerakan linear dan rotasi robot dengan akurasi tinggi serta menganalisis pengaruh variasi konstanta PID dan kecepatan maksimum terhadap pergerakan dan akurasi robot.

Metode yang digunakan dalam penelitian ini meliputi rancang bangun sistem kendali untuk mendapatkan algoritma kendali kecepatan dan posisi robot. Algoritma juga mencakup perhitungan kinematika untuk menghasilkan nilai kecepatan linear dan rotasi dengan mempertimbangkan efek dinamis tanpa melibatkan gaya secara langsung. Pengujian dilakukan dengan mengukur parameter performa sistem, yaitu *settling time*, *steady-state error*, dan *overshoot*, serta mengevaluasi akurasi pergerakan robot berdasarkan *RMSE* dan *lateral deviation* dalam berbagai skenario lintasan.

Hasil penelitian menunjukkan bahwa sistem kendali berbasis PID mampu mengontrol pergerakan robot dengan akurasi tinggi. Dengan konstanta optimal Kp = 0.6, Ki = 12, dan Kd = 0.008, robot mencapai setpoint kecepatan dengan *settling time* sebesar 63 ms, tanpa *overshoot*, dan dengan *steady-state error* 0.0069 m/s. Pada lintasan lurus, nilai RMSE terkecil diperoleh sebesar 0.2156 m pada kecepatan renda, sedangkan pada kecepatan tinggi, RMSE meningkat menjadi 0.3220 m. Nilai *Lateral deviation* juga meningkat dari 0.1497 m pada kecepatan rendah menjadi 0.1825 m pada kecepatan tinggi. Namun, untuk lintasan kompleks seperti pergerakan serong atau kombinasi translasi dan rotasi, kecepatan tinggi justru menghasilkan kestabilan heading yang lebih baik.

Kata kunci : Kendali Kecepatan, Kendali Posisi, PID, Kinematika Robot Beroda, Robot Otonom

SUMMARY

DESIGN OF A PID AND KINEMATICS BASED X-DRIVE CONFIGURED WHEELED AUTONOMOUS ROBOT MOVEMENT CONTROL ALGORITHM

Rafif Susena

The development of robotics technology has increased rapidly, especially in the application of wheeled autonomous robots used in various fields, including robotics competitions such as the ABU Indonesia Robot Contest. In this context, the ability to move precisely and accurately from one point to another is very important, especially for omni-wheeled robots with an X-Drive configuration that allows omnidirectional movement. Previous studies have shown the successful implementation of PID control algorithms in speed and position control, but often do not consider non-linear aspects such as the coriolis effect and slip due to excessive acceleration. This led to the formulation of the problem, namely how to design a movement control system model that is able to control the linear and rotational movements of the robot with high accuracy and analyze the effect of variations in PID constants and maximum speed on the movement and accuracy of the robot.

The method used in this research includes the design of a control system to obtain a robot speed and position control algorithm. The algorithm also includes kinematics calculations to generate linear and rotational velocity values by considering dynamic effects without directly involving forces. Tests were conducted by measuring system performance parameters, namely settling time, steady-state error, and overshoot, and evaluating the accuracy of robot movement based on RMSE and lateral deviation in various trajectory scenarios.

The results show that the PID-based control system is able to control the movement of the robot with high accuracy. With optimal constants Kp = 0.6, Ki = 12, and Kd = 0.008, the robot reaches the speed setpoint with a settling time of 63 ms, without overshoot, and with a steady-state error of 0.0069 m/s. On a straight trajectory, the smallest RMSE value was obtained as 0.2156 m at lace speed, while at high speed, the RMSE increased to 0.3220 m. Lateral deviation also increased from 0.1497 m at low speed to 0.1825 m at high speed. However, for complex trajectories such as oblique movement or a combination of translation and rotation, high speed results in better heading stability.

Keywords : Velocity Control, Position Control, PID , Wheeled Robot Kinematic, Autonomous Robot

PRAKATA

Puji syukur penulis panjatkan ke hadirat Allah SWT atas segala rahmat dan karunia-Nya, sehingga penulis dapat menyelesaikan laporan tugas akhir ini dengan baik. Laporan tugas akhir ini disusun sebagai salah satu syarat untuk memperoleh gelar Sarjana Teknik di Program Studi Teknik Elektro Universitas Jenderal Soedirman. Judul yang penulis pilih adalah "**RANCANG BANGUN ALGORITMA KENDALI PERGERAKAN ROBOT OTONOM BERODA BERKONFIGURASI *X-DRIVE* BERBASIS PID DAN KINEMATIKA**."

Tak lupa penulis juga mengucapkan terimakasih banyak kepada setiap pihak yang telah membantu dan memberikan dukungan kepada penulis selama proses penyelesaian tugas akhir ini. Ucapan terima kasih penulis sampaikan pada :

1. Bapak Daru Tri Nugroho, S.T., M.T., selaku Ketua Jurusan Teknik Elektro Universitas Jenderal Soedirman.
2. Bapak Agung Mubyarto, S.T., M.T serta Bapak Ir. Priswanto, S.T., M.Eng. selaku Dosen Pembimbing yang memberikan pengarahan dan dukungan dalam pelaksanaan serta penyusunan laporan tugas akhir.
3. Kedua orang tua dan saudara-saudara penulis atas limpahan doa, dukungan dan semangat yang selalu mengalir.
4. Teman-teman seperjuangan Teknik Elektro 2021 yang selalu ada untuk mendukung penulis.
5. Serta pihak-pihak lain yang telah membantu dalam menyelesaikan laporan ini yang tidak dapat disebutkan satu per satu.

Penulis menyadari bahwa dalam penyusunan laporan tugas akhir ini masih banyak kekurangan dan kelemahan yang perlu diperbaiki. Oleh karena itu, saran dan kritik yang membangun sangat penulis harapkan untuk dapat meningkatkan kualitas laporan ini.

Akhir kata, penulis berharap laporan tugas akhir ini dapat memberikan manfaat dan kontribusi bagi pengembangan teknologi robotika di masa depan.

Purbalingga, 14 Februari 2025

Rafif Susena

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR ISTILAH DAN SINGKATAN

PID : Proporsional-Integral-Derivatif, metode kontrol untuk mencapai kecepatan atau posisi target.

IMU : *Inertial Measurement Unit*, sensor yang mengukur orientasi dan percepatan.

X-Drive : Konfigurasi roda omni yang memungkinkan gerakan *omnidirectional*.

Odometry : Teknik untuk menghitung posisi robot berdasarkan data dari encoder roda.

Raspberry Pi : *Single board computer* yang digunakan untuk mengelola algoritma dan kontrol.

STM32 : Mikrokontroler yang digunakan untuk kontrol motor dan pembacaan sensor.

RMSE : *Root Mean Square Error*, metrik untuk mengukur akurasi posisi.

Lateral Deviation : Ukuran seberapa jauh nilai riil menyimpang dari nilai yang diinginkan.

Settling Time : Waktu yang diperlukan sistem untuk mencapai kondisi stabil.

Overshoot : Selisih antara respons maksimum sistem dan setpoint.

SSE : Steady State Error, kesalahan sistem setelah mencapai kestabilan.

Holonomik : Sistem yang dapat bergerak bebas dalam semua arah.

Kinematika : Studi tentang gerakan benda tanpa mempertimbangkan penyebab gerakannya.

Dinamika : Studi tentang gerakan benda yang melibatkan gaya dan massa.

Encoder : Sensor yang digunakan untuk menghitung rotasi roda dan mengukur jarak tempuh.

Fusi Sensor : Penggabungan data dari beberapa sensor untuk meningkatkan akurasi.

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Dalam beberapa tahun belakangan ini, teknologi robotika telah mengalami perkembangan yang sangat pesat, terutama dalam hal robot otonom bergerak yang digunakan untuk berbagai macam keperluan, seperti industri, transportasi hingga untuk keperluan kompetisi robotika. Mengutip dari laporan terbaru oleh *Internasional Federation of Robotic* menunjukkan pada tahun 2022, volume penjualan tahunan unit robot tumbuh sebesar 5% (+ 553.052 robot industri terjual) lebih banyak daripada tahun 2021. Menurut laporan yang sama, stok robot operasional meningkat menjadi 3.903.633 unit, 12% lebih banyak dari tahun 2021, dan nilai instalasi robot tumbuh menjadi $ 15,8 miliar, 4% lebih banyak dari tahun 2021 [1].

Pada kalangan pelajar, berbagai kompetisi diadakan sebagai bentuk pengenalan dan pengembangan teknologi robotika yang setiap tahun, tantangan yang diujikan semakin sulit dan semakin mengarah ke otomatisasi robot. Sebagai contoh, dalam Kontes Robot ABU Indonesia ataupun *ABU Robocon* 2024 yang mengambil tema “Panen Raya”, tantangan yang harus dihadapi oleh peserta adalah bagaimana robot beroda dapat melaksanakan tugas secara otonom baik bergerak dari satu zona ke zona lain, maupun melakukan pengambilan keputusan dalam hal penyortiran dan penempatan objek [2].

Dari sekian banyak tugas yang perlu dilakukan oleh sebuah robot otonom, hal yang paling mendasar adalah terkait pergerakan atau bagaimana robot dapat bergerak dari satu titik ke titik lain secara mandiri. Untuk itu, dalam konteks robot

beroda, terutama yang menggunakan roda omni, pengendalian gerakan untuk mencapai nilai posisi dan orientasi tertentu secara presisi menjadi semakin penting karena karakteristik roda omni memungkinkan pergerakan *omnidirectional* (ke segala arah) yang kompleks serta memastikan adanya sistem kendali yang optimal untuk menjaga stabilitas kecepatan dengan tetap mempertahankan posisi robot secara bersamaan.

Algoritma kendali yang sering kali digunakan adalah PID (Proporsional-Integral-Derivatif). Implementasi algoritma PID dalam sistem kendali robot telah terbukti andal dalam mengatur kecepatan dan posisi robot di samping kesederhanaannya dalam implementasi serta kemampuannya untuk menghasilkan sistem yang stabil dan responsif [3].

Beberapa penelitian terdahulu telah menunjukkan keberhasilan implementasi PID dalam sistem kendali robot beroda. Sebagai contoh, menurut Iswanto et al. (2021) tentang penelitiannya dalam penggunaan kontrol PID berbasis odometri ditemukan bahwa galat kecepatan linear yang dihasilkan sebesar 2% dengan nilai rata-rata 1,3%, rata-rata kesalahan pergerakan robot pada koordinat X dan Y adalah 0,0059 m serta kesalahan arah hadap robot pada semua lintasan sebesar 0,6% [4].

Penelitian lain yang dilakukan oleh Kurniawan et al. (2019) tentang *trajectory tracking* berbasis kontrol PID (Proporsional, Integral, Derivatif) menunjukkan hasil bahwa pola gerak robot dapat mengikuti pola perencanaan yang telah ditentukan dengan baik dengan nilai rata-rata kesalahan sebesar 6,48% dan 6,83% [5]. Serta penelitian oleh Sofwan et al. (2020) terkait kontrol PID pada robot

menunjukkan bahwa robot roda omni yang dikembangkan mampu melakukan gerak ke posisi yang diinginkan dan mengikuti lintasan yang dikontrol dengan mempertahankan kesalahan minimum relatif ke lintasan yang direferensikan [6].

Algoritma PID dalam sistem kendali robot beroda pada intinya memiliki peranan dalam pengendalian posisi atau kecepatan. Dalam keadaan yang nyata, implementasi kendali saja tidaklah cukup, aspek lain seperti efek coriolis dan gravitasi yang muncul ketika robot bergerak juga mempengaruhi pergerakan. Selain itu, akselerasi dan deselerasi pergerakan yang bisa saja nilainya berlebihan juga dapat mengakibatkan adanya slip pada roda yang sering kali tidak terlalu dibahas dalam penelitian-penelitian sebelumnya.

Berdasarkan permasalahan tersebut, maka peneliti memutuskan untuk melakukan penelitian dengan judul **"RANCANG BANGUN ALGORITMA KENDALI PERGERAKAN ROBOT OTONOM BERODA BERKONFIGURASI *X-DRIVE* BERBASIS PID DAN KINEMATIKA"**. Fokus pembahasan dalam penelitian ini adalah terkait perancangan sistem kendali kecepatan berdasarkan posisi pada robot otonom holonomik (bergerak ke segala arah tanpa mengubah orientasi) beroda bertipe kemudi *X-Drive* menggunakan kontrol PID untuk menghasilkan nilai dan arah kecepatan linear serta kecepatan rotasi robot. Selain itu dalam rancangan algoritma, dipertimbangkan aspek kinematika linear ataupun non-linear yang terjadi akibat dinamika robot selama bergerak tanpa melibatkan secara langsung parameter dinamika seperti gaya, torsi dan massa dalam perhitungannya dalam menentukan nilai kecepatan akhir.

1.2 Rumusan Masalah

Berdasarkan latar belakang tersebut, didapat beberapa rumusan masalah sebagai berikut.

1. Bagaimana merancang sistem kendali pergerakan robot yang mampu mengontrol kecepatan gerak linear maupun rotasi agar robot dapat mencapai target koordinat yang ditentukan ?
2. Bagaimana pengaruh variasi penggunaan konstanta sistem kendali PID terhadap kecepatan pergerakan robot ?
3. Bagaimana pengaruh nilai variasi kecepatan maksimal gerak robot terhadap akurasi gerak robot ?

1.3 Batasan Masalah

Agar penelitian ini lebih terarah dan tanpa mengurangi maksud juga tujuannya, maka ditetapkan batasan penelitian sebagai berikut.

1. Penelitian ini hanya akan difokuskan pada pengembangan sistem kendali kecepatan untuk robot holonomik dengan konfigurasi roda omni tipe *X-Drive* yang dibuat untuk perlombaan Kontes Robot ABU Indonesia 2024.
2. Sistem kendali yang dibahas hanya mengontrol nilai kecepatan yang harus ditempuh oleh robot berdasarkan titik poin koordinat yang diberikan, tanpa membahas bagaimana cara menentukan titik-titik *trajectory* atau jalur optimal menuju target koordinat.
3. Penggunaan algoritma PID terbatas hanya untuk mengontrol kecepatan linear dan rotasi robot, tanpa membahas metode kendali lanjutan lainnya seperti kendali adaptif atau kendali optimal.

4. Data umpan balik kendali berasal dari fusi sensor yang menggunakan data dari *rotary encoder* dan IMU. Penelitian ini tidak membahas algoritma fusi sensor, melainkan hanya mengaplikasikan algoritma yang sudah ada.
5. Pengujian sistem dilakukan dalam lingkungan dengan asumsi kondisi permukaan lantai yang datar dan tidak melibatkan skenario lingkungan yang lebih kompleks seperti medan berbatu atau medan yang dinamis.

1.4 Tujuan

Tujuan dari penulisan laporan tugas akhir ini adalah sebagai berikut.

1. Merancang sistem kendali pergerakan yang mampu mengontrol kecepatan agar robot dapat mencapai target posisi dan orientasi yang ditentukan secara akurat dan stabil.
2. Menganalisis pengaruh variasi konstanta PID terhadap pergerakan robot dalam hal kontrol kecepatan untuk menemukan konfigurasi kendali yang optimal.
3. Mengevaluasi pengaruh variasi kecepatan maksimal terhadap akurasi gerak robot, dengan tujuan menemukan keseimbangan antara kecepatan, stabilitas, dan akurasi pada pergerakan robot.

1.5 Manfaat

Adapun manfaat yang didapat diambil dari tugas akhir ini yaitu:

1. Manfaat praktis bagi penulis yakni dapat memberikan kontribusi terhadap pengembangan ilmu pengetahuan, khususnya di bidang robotika dan kontrol sistem lewat aplikasi dari ilmu yang telah diperoleh selama masa perkuliahan.

2. Manfaat praktis bagi bidang robotika, di mana hasil penelitian ini dapat diterapkan pada pengembangan robot otonom beroda yang membutuhkan kendali gerak presisi, seperti dalam aplikasi industri, otomasi, atau robotika layanan.
3. Manfaat akademis dalam memperkaya referensi dan studi kasus terkait pengendalian robot berbasis PID dengan fusi sensor, yang dapat menjadi acuan untuk penelitian-penelitian lanjutan di bidang kendali robotika.

BAB 2
TINJAUAN PUSTAKA

2.1 Penelitian Terdahulu

Topik seputar implementasi algoritma PID sebagai pilihan algoritma kontrol pada robot beroda sudah menjadi bahasan pada beberapa penelitian sebelumnya. Beberapa diantaranya adalah seperti yang telah ditulis oleh Iswanto et al. (2021) dalam penelitiannya membahas tentang perancangan pelacakan jalur gerak dari robot. Isinya adalah terkait analisa rangka robot untuk memodelkan pergerakan, perancangan sistem kendali dan pengujian komponen kendali PID, pengujian motor dan pengujian jalur gerak atau *trajectory* [4].

Selain itu penelitian oleh Kurniawan et al. (2019) membahas tentang perancangan pelacakan jalur gerak dari robot menggunakan PID. Spesifik pembahasan adalah terkait pemodelan pergerakan, perancangan sistem kendali dan pengujian motor dan pengujian jalur gerak atau *trajectory* [5]. Serta penelitian yang dilakukan oleh Sofwan et al. (2019) membahas tentang pengembangan robot bergerak beroda banyak berdasarkan kinematika terbalik dan odometri berbasis PID untuk memperkirakan posisi robot relatif terhadap posisi awal posisi dengan (fusi sensor) penggunaan encoder dan IMU sebagai sinyal umpan baliknya. Untuk mempertahankan posisi robot relatif terhadap posisi yang diinginkan, kontrol PID diterapkan pada algoritma [6].

Dari beberapa referensi penelitian lain yang telah disebutkan, tiga penelitian yang telah disebutkan diambil untuk dijadikan bahan referensi dari penelitian ini. Berikut dapat dilihat pada Tabel 2.1 terkait perbedaan dengan penelitian yang dilakukan.

Tabel 2.1 Penelitian Terdahulu

No	Judul	Penulis	Perbedaan
1.	*PID-based with Odometry for Trajectory Tracking Control on Four-wheel Omnidirectional Covid-19 Aromatherapy Robot*	• Iswanto • Alfian Ma'arif • Nia Maharani Raharja • Gatot Supangkat • Fitri Arofiati • Ravi Sekhar • Dhiya Uddin • Rijalusalam	• Penggunaan pemodelan gerak robot terbatas hanya dalam konversi kecepatan robot terhadap kecepatan roda dan sebaliknya. • Sinyal umpan balik untuk algoritma kontrol hanya berasal dari encoder pada motor dc. • Nilai referensi kecepatan akhir robot diambil langsung dari hasil PID dengan metode pembatasan akselerasi dan deselerasi, tidak mempertimbangkan aspek dinamis lain ketika bergerak. • Pengujian pergerakan terbatas hanya pada gerakan linear dengan *heading* yang statis, tidak membahas terkait kombinasi gerakan linear dan rotasi secara bersamaan.
2.	*PID Trajectory Tracking Control 4 Omni-Wheel Robot*	• Ghufron Wahyu Kurniawan • Novendra Setyawan • Ermanu Azizul Hakim	• Perhitungan nilai target kecepatan didapat dari hasil PID secara langsung, tidak membahas aspek percepatan dan dinamis robot. • Pengujian dilakukan terbatas pada gerakan linear ke satu titik tujuan saja dengan beberapa repetisi untuk menemukan rerata galat dengan variasi lintasan yang tidak beragam (hanya gerak lurus). • Sinyal umpan balik algoritma berasal dari sensor encoder dengan 2 cara pemasangan yakni pada roda dan *freewheel* pada rangka robot.

No	Judul	Penulis	Perbedaan
3.	*Development of Omni-Wheeled Mobile Robot Based-on Inverse Kinematics and Odometry*	• Aghus Sofwan • Hafidz Rizqi Mulyana • Hadha Afrisal • Abdul Goni	• Pembahasan perancangan algoritma lebih banyak membahas tentang kinematika robot dibandingkan pembahasan terkait sistem kontrol yang digunakan (PID). • Pembahasan Kinematika terbatas untuk memperoleh hubungan antara kecepatan robot dengan kecepatan setiap roda. • Hasil pengujian hanya membahas terkait pergerakan robot dalam lintasan tertentu untuk hanya kombinasi gerakan linear tanpa melibatkan gerakan rotasi secara bersamaan.

2.2 Robot Otonom Beroda

2.2.1 Pengertian Robot Otonom

Waypoint Robotics mendefinisikan robot otonom sebagai mesin cerdas yang mampu melakukan tugas dan beroperasi di lingkungan secara mandiri, tanpa kontrol atau intervensi manusia. Robot otonom, seperti halnya manusia, juga dapat membuat keputusan sendiri dan kemudian melakukan tindakan yang sesuai. Robot yang benar-benar otomatis bisa:

1. Memahami lingkungannya
2. Membuat keputusan berdasarkan hal di atas
3. Menggerakkan suatu gerakan atau berinteraksi di dalam lingkungan tersebut.

Tindakan ini melibatkan pengambilan keputusan seperti memulai, berhenti, dan menavigasi di sekitar rintangan. Pada intinya, robot otonom adalah

mesin pintar yang dapat melakukan tugas dan bekerja sendiri tanpa perlu dikendalikan oleh manusia [7].

2.2.2 Roda Universal / *Omniwheel* dalam Robot Beroda

Berbagai desain roda *omnidirectional* dapat dibagi menjadi dua tipe dasar yakni yang menggunakan roda khusus dengan atau tanpa mekanisme khusus, dan tipe lainnya adalah roda konvensional. Roda khusus contohnya roda universal, dan roda mecanum, sedangkan roda konvensional yang dibagi menjadi roda kastor bertenaga (PCW) dan roda standar yang dapat dikemudikan.

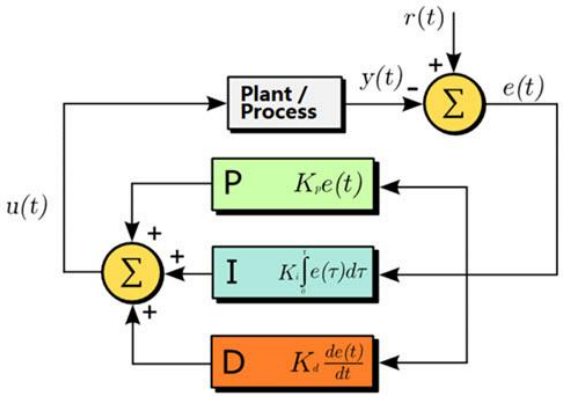
Roda universal atau roda omni dirancang sedemikian rupa sehingga ada sejumlah rol pasif kecil yang dipasang di pinggiran roda normal seperti yang terlihat pada Gambar 2.1. Sumbu rol ini tegak lurus dengan roda. Roda digerakkan dengan cara normal, sementara rol dapat berputar secara ortogonal ke arah penggeraknya dengan bebas. Dengan menggunakan beberapa roda universal yang didistribusikan di sekeliling platform robot, gaya yang diterapkan pada setiap roda dapat digabungkan untuk memberikan mobilitas *omnidirectional* [8].

Gambar 2.1 Roda Omni

2.3 Kontrol PID (Proporsional-Integral-Diferensial)

2.3.1 Pengertian Kontrol PID

Kontrol PID adalah metode kendali yang digunakan seperti untuk mencapai posisi atau kecepatan target dengan mempertimbangkan tiga komponen: Proporsional (P), Integral (I), dan Derivatif (D). Kontrol PID memanfaatkan sinyal kesalahan yang berasal dari penjumlahan sinyal referensi dan sinyal masukannya, kemudian digunakan oleh 3 komponen pengendali untuk memperoleh sinyal keluaran yang akan masuk kembali ke *plant* seperti diilustrasikan pada Gambar 2.2 terkait diagram blok PID di bawah ini.



Gambar 2.2 Diagram Blok Kontrol PID

Komponen proporsional bertindak berdasarkan besar galat/error saat ini, semakin besar error antara nilai target dan nilai aktual, semakin besar respon yang diberikan untuk mengurangi error tersebut. Namun, penggunaan P saja sering kali menyebabkan sistem tidak mampu mencapai setpoin dengan sempurna karena adanya *steady-state error* atau selisih nilai stabil dengan nilai setpoin. Serta jika hanya mengandalkan komponen P, respon sistem bisa cepat tetapi mungkin tidak mencapai posisi akhir dengan akurat karena adanya faktor-faktor atau gangguan lain.

Komponen integral ditambahkan untuk mengatasi kelemahan ini dengan mempertimbangkan akumulasi error dari waktu ke waktu. Artinya, kontrol PI (P + I) dapat memberikan dorongan tambahan jika error tetap ada dalam jangka waktu lama, seperti saat ada gesekan yang menghambat pergerakan. Komponen ini efektif untuk mencapai posisi akhir secara lebih akurat, namun dapat menyebabkan sistem berosilasi saat mendekati target jika akumulasi error menjadi terlalu besar dan juga memperpanjang *settling time* atau waktu menuju keadaan stabilnya.

Komponen derivatif digunakan untuk mengantisipasi perubahan error di masa depan berdasarkan kecepatan perubahan error saat ini, meredam osilasi dan mempercepat waktu stabilisasi dengan memperhitungkan laju perubahan error terhadap waktu. Dengan demikian, kontrol PID (gabungan dari P, I, dan D), komponen derivatif membantu memperlambat gerakan ketika posisi target hampir tercapai, mengurangi *overshoot* dan meningkatkan stabilitas.

Sistem kendali PID ini dapat disesuaikan dengan "gain" atau faktor pengali yang memungkinkan fleksibilitas dalam pengaturan, fungsi matematis PID dapat dilihat seperti pada persamaan 2.1. Pendekatan PID memungkinkan robot atau sistem lainnya untuk bergerak menuju posisi atau kecepatan target dengan memperhitungkan perubahan kondisi secara *realtime*, sehingga cocok untuk situasi di mana pergerakan atau posisi objek bergerak perlu diantisipasi [9].

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{d}{dt} e(t) \qquad (2.1)$$

2.3.2 Tuning Kontrol PID

Tuning tiga parameter PID yakni K_p, K_i, dan K_p adalah masalah yang penting. Panduan berikut ini dapat digunakan untuk menemukan nilai yang sesuai secara eksperimental atau *trial-and-error* :

1. Pilih pengaturan operasi tipikal untuk kecepatan yang diinginkan, matikan bagian integral dan derivatif, kemudian naikkan K_p hingga maksimum atau hingga terjadi osilasi.
2. Jika sistem berosilasi, bagi K_p dengan 2.
3. Naikkan K_p dan amati perilaku ketika menambah/mengurangi kecepatan setpoin sekitar 5%. Pilih nilai K_p yang memberikan respon teredam.
4. Tingkatkan K_i secara perlahan hingga osilasi dimulai. Kemudian bagi K_i dengan 2 atau 3.
5. Periksa apakah kinerja pengontrol secara keseluruhan memuaskan dalam kondisi sistem yang umum [10].

2.4 Pemodelan Gerak Robot

2.4.1 Kinematika Robot Beroda

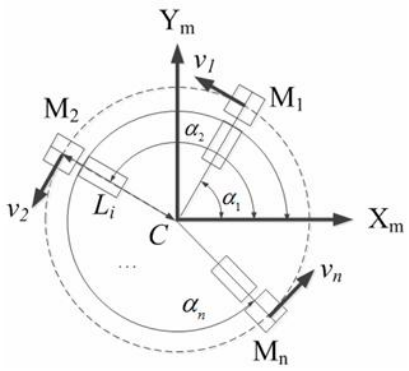
Kinematika merupakan cabang dari ilmu fisika tepatnya mekanika klasik yang berfokus mengenai gerak benda serta sistem benda. Kinematika berkaitan dengan materi kecepatan, perpindahan serta percepatan gerak benda yang terkait dengan waktu, tanpa membahas mengenai penyebab dari gerak benda tersebut. Dalam kinematika, benda dikatakan bergerak jika benda tersebut mengalami perpindahan dari posisi awal ke posisi akhir [11].

Untuk memahami bagaimana sebuah robot beroda dengan roda omni berperilaku dan bergerak pada ruang, kinematikanya harus dijelaskan. Untuk n

roda, berikut bisa didapatkan ekspresi kinematika berikut untuk hubungan antara kecepatan linear roda $(v_1, ..., v_n)^T$ dan kecepatan robot $(u, v, \omega)^T$ (lihat Gambar 2.3):

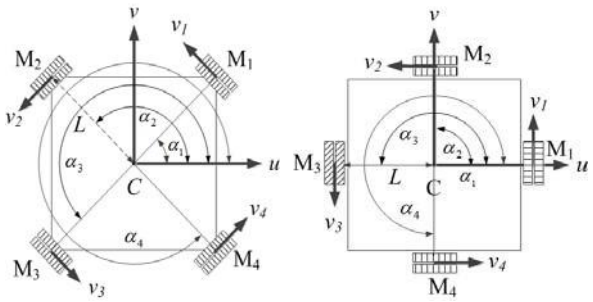
$$\begin{bmatrix} v_1 \\ v_2 \\ ... \\ v_n \end{bmatrix} = \begin{bmatrix} -\sin(\alpha_1) & \cos(\alpha_1) & L_1 \\ -\sin(\alpha_2) & cos(\alpha_2) & L_2 \\ ... & ... & ... \\ -\sin(\alpha_n) & cos(\alpha_n) & L_n \end{bmatrix} \begin{bmatrix} u \\ v \\ \omega \end{bmatrix} \quad (2.2)$$

Dengan nilai α_i, i = 1, ..., n adalah sudut sumbu motor yang diukur relatif terhadap arah sumbu pusat rangka dari robot. L_i, i = 1, ..., n menunjukkan jarak dari pusat massa ke pusat roda. Variabel u, v masing-masing adalah komponen kecepatan linier lokal dalam arah sumbu Xm dan sumbu Ym dari kerangka yang bergerak, sedangkan ω mewakili kecepatan sudut.



Gambar 2.3 Vektor Gerak Robot dengan Roda Universal

Gambar 2.4 menunjukkan *Omniwheel Mobile Robot* yang dilengkapi dengan empat roda penggerak independen dengan jarak yang sama pada 90 derajat dari satu sama lain. Perhatikan bahwa perbedaan utama antara kedua diagram tersebut adalah bahwa sumbu dari kerangka yang bergerak didefinisikan secara berbeda. Pada Gambar 2.4(a), αi, dengan i = 1, 2, 3, 4 masing-masing sama dengan 45°, 135°, -135°, dan -45°, sedangkan pada Gambar 2.4(b), αi, dengan i = 1, 2, 3, 4 sama dengan 0°, 90°, 180°, dan -90°, masing-masing [8].



Gambar 2.4 Parameter geometris dari Robot Beroda dengan 4 roda universal

2.4.2 Dinamika Robot Beroda

Dalam konteks robot beroda, pemodelan dinamika mempertimbangkan gaya yang diperlukan untuk mempercepat, memperlambat, atau mengubah arah gerak robot. Persamaan dinamika menggabungkan variabel seperti massa robot, kecepatan sudut roda, dan gesekan dengan permukaan untuk menggambarkan perilaku gerakan robot yang realistis. Pendekatan ini penting dalam desain sistem kendali agar robot dapat bergerak dengan stabil dan efisien, terutama dalam menghadapi perubahan kecepatan atau medan yang berbeda.

Pemodelan dinamika robot berkaitan dengan penurunan persamaan dinamis dari gerakan robot. Hal ini dapat dilakukan dengan menggunakan dua metodologi yakni Metode Newton-Euler serta Metode Lagrange. Metode Newton-Euler menghitung gaya dan momen secara langsung, sedangkan Metode Lagrange, menggunakan energi kinetik dan potensial untuk menurunkan persamaan gerak. Persamaan dinamika robot yang meliputi gangguan terhadap sistem secara umum adalah sebagai berikut [12]:

$$M(q) \cdot \ddot{q} + V(q, \dot{q}) + F(\dot{q}) + G(q) + \tau_d = E(q)u - A^T(q)\lambda \quad (2.3)$$

Dengan :

q : Vektor koordinat

$M(q)$: Matriks inersia robot yang berhubungan dengan massa dan inersia setiap komponen.

$V(q, \dot{q})$: Matriks gaya Coriolis dan sentripetal yang bergantung pada kecepatan, mencakup efek gaya yang timbul saat robot bergerak dengan cepat.

$F(\dot{q})$: Vektor untuk gaya gesek dan gaya reduksi.

$G(q)$: Vektor gaya gravitasi, untuk robot bergerak pada bidang horizontal, ini sering diabaikan.

τ_d : Vektor dari gangguan lain yang tidak dimodelkan dalam sistem.

$E(q)$: Matriks transformasi dari koordinat lokal ke global.

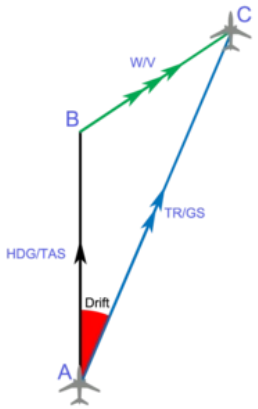
u : Vektor masukan.

$A^T(q)$: Matriks kinematika.

λ : Konstanta pengali Lagrange.

2.5 Dead Reckoning

Dead Reckoning adalah metode navigasi yang melibatkan estimasi posisi saat ini berdasarkan posisi yang telah ditentukan sebelumnya (perhatikan Gambar 2.5), memajukan posisi tersebut berdasarkan kecepatan yang diketahui atau diperkirakan selama waktu yang telah berlalu, dan arah.



Gambar 2.5 Ilustrasi Dead Reckoning Calculation Model

Odometri (perhitungan jarak) digunakan untuk memperkirakan pose robot dengan mengintegrasikan peningkatan gerakan yang dapat diukur atau

dikumpulkan dari gerakan yang diterapkan perintah. Peningkatan gerakan relatif dalam robotika bergerak biasanya diperoleh dari sensor (misalnya, encoder tambahan) yang terpasang ke roda robot. Keampuhan *Dead Reckoning* bergantung pada tiga elemen penting: titik awal, kecepatan, dan arah.

Rumus dasar yang digunakan dalam *dead reckoning* untuk menghitung posisi saat ini relatif mudah. Rumus-rumus ini melibatkan aritmatika sederhana berdasarkan kecepatan, waktu, dan arah. Berikut ini adalah rumus-rumus kuncinya:

- Jarak Tempuh (D): dihitung dengan mengalikan kecepatan (S) dengan waktu (T).

  $$D = S \times t \tag{2.4}$$

  Di sini, 'D' adalah jarak yang ditempuh, 'S' adalah kecepatan objek bergerak, dan 't' adalah waktu objek bergerak.
- Perhitungan Posisi Baru: Posisi baru dihitung dengan memperbarui posisi sebelumnya berdasarkan jarak yang ditempuh dalam arah tertentu (*heading*).
- Jika bergerak dalam garis lurus ke sumbu y, cukup menambahkan atau mengurangi jarak yang ditempuh dari garis di sumbu y awal.
- Jika bergerak dalam garis lurus ke arah sumbu x, cukup menambah atau mengurangi jarak yang ditempuh dari garis sumbu x awal.
- Namun, jika arahnya tidak tepat di sepanjang arah komponen sumbu koordinat, digunakan trigonometri untuk menghitung posisi baru dengan θ adalah sudut yang terbentuk terhadap sumbu x positif [13].

$$\Delta x = D \times \cos(\theta) \quad (2.5)$$

$$\Delta y = D \times \sin(\theta) \quad (2.6)$$

2.6 Sensor IMU (*Inertial Measurement Unit*)

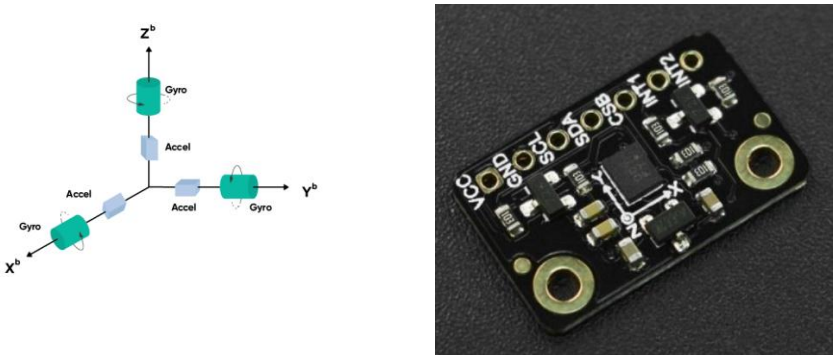
Pengukuran gerakan, khususnya, akselerasi, rotasi, dan kecepatan, sangat penting untuk memahami orientasi suatu objek. Hal ini juga dapat diterapkan secara luas pada banyak aplikasi; misalnya, mesin lini produksi, perangkat robotika, kendaraan, sistem otonom, gimbal, peralatan mesin, dan bahkan prostetik robotika. Kemajuan dalam teknologi, teknik manufaktur, miniaturisasi, dan pemrosesan terkomputerisasi telah sangat menyederhanakan produksi perangkat pengindraan gerak yang digunakan dalam IMU.

IMU (kadang-kadang disebut sebagai unit referensi inersia [IRU] atau unit referensi gerak [MRU]) biasanya merupakan perangkat elektromekanis atau *solid-state* yang berisi serangkaian sensor yang mampu mengukur gerakan. Yaitu, untuk mendeteksi akselerasi linier (laju perubahan kecepatan) dan laju sudut (perubahan kecepatan sudut) pada sekitar sumbu X, Y dan Z. Ada banyak jenis sensor IMU yang ada di pasaran, sebagai contoh seperti seri MPU6050, BMI160, BNO055 serta BMX160 seperti pada Gambar 2.6.

IMU dapat mengukur gerakan dengan mengubah inersia yang terdeteksi, yang merupakan gaya yang tercipta karena resistensi objek terhadap perubahan arah, menjadi data keluaran yang menggambarkan gerakan objek. Data ini akan digunakan oleh beberapa sistem lain, misalnya, untuk mengontrol kendaraan. Keluaran dari IMU biasanya berupa data sensor yakni :

- Akselerometer (pengukuran akselerasi linier di setiap sumbu)

- Giroskop (laju rotasi/pengukuran kecepatan sudut di sekitar setiap sumbu) [14]

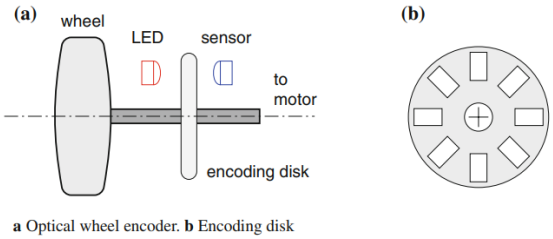


Gambar 2.6 Akselerometer dan giroskop dalam tiga sumbu gerakan (kiri) dan Sensor IMU seri BMX160 (kanan)

2.7 Rotary Encoder

Rotary encoder adalah sensor yang dipasang pada sumbu roda untuk mengukur rotasi roda. Encoder ini bekerja dengan menghitung jumlah putaran (atau sebagian putaran) roda dan mengubahnya menjadi sinyal digital yang dapat diproses oleh mikrokontroler. Keliling dari sebuah roda adalah 2πr, dengan r adalah jari-jari roda dalam cm, jika n rotasi dihitung, roda telah bergerak sejauh 2πnr cm. Encoder dapat dibuat yang mengukur pecahan revolusi. Jika sebuah sinyal dihasilkan 8 kali per revolusi, jarak yang ditempuh adalah 2πnr/8 cm, dengan n adalah jumlah sinyal yang dihitung oleh komputer.

Ada banyak cara yang berbeda untuk mengimplementasikan encoder. Desain yang popular adalah dengan menggunakan sumber cahaya seperti dioda pemancar cahaya (LED) seperti diilustrasikan pada Gambar 2.7, sensor cahaya dan sebuah piringan penyandi yang dipasang pada sumbu roda (a). Piringan tersebut adalah dilubangi dengan lubang (b) sehingga setiap kali lubang berlawanan dengan sumber Cahaya sumber cahaya, sensor menghasilkan sinyal [9].



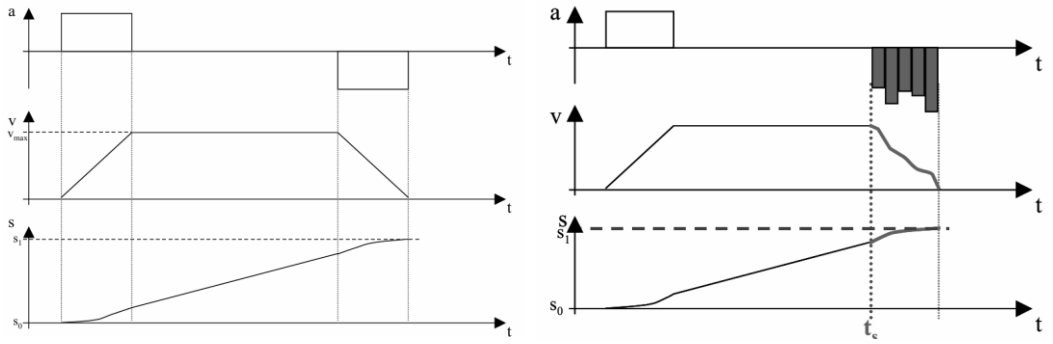
a Optical wheel encoder. **b** Encoding disk

Gambar 2.7 Struktur Rotary Encoder

2.8 Kontrol Kecepatan dan Posisi

Kontrol kecepatan dan posisi berfungsi untuk mempertahankan parameter kecepatan dan posisi di nilai tertentu. Kontrol posisi memerlukan pengontrol tambahan di atas pengontrol kecepatan. Pengontrol posisi mengatur kecepatan yang diinginkan di semua fase pergerakan, terutama selama fase akselerasi dan deselerasi (mulai dan berhenti). Asumsikan sebuah motor tunggal menggerakkan kendaraan robot yang pada awalnya dalam keadaan diam dan yang ingin kita hentikan pada posisi tertentu. Gambar 2.8 (kiri) menunjukkan grafik parameter kontrol yang menunjukkan “ramp kecepatan” untuk fase start, fase kecepatan konstan, dan fase berhenti.

Ketika mengabaikan gesekan, yang diperhatikan hanya perlu memberikan gaya tertentu selama fase start, yang akan diterjemahkan ke dalam akselerasi kendaraan. Akselerasi konstan akan meningkatkan kecepatan kendaraan secara linier dari 0 ke nilai yang diinginkan, sedangkan posisi kendaraan akan meningkat secara kuadratik. Ketika gaya (akselerasi) berhenti, kecepatan kendaraan akan tetap konstan, dengan asumsi tidak ada gesekan, dan posisinya akan meningkat secara linier.



Gambar 2.8 Grafik Parameter Kontrol Gerak Posisi (Kiri) dan Grafik Adaptasi Pengereman (kanan)

Selama fase berhenti (deselerasi, pengereman), gaya negatif (akselerasi negatif) diterapkan pada kendaraan. Kecepatannya akan berkurang secara linier menjadi nol (atau bahkan mungkin menjadi negatif - kendaraan sekarang melaju mundur - jika akselerasi negatif diterapkan untuk waktu yang terlalu lama). Posisi kendaraan akan meningkat secara perlahan, mengikuti fungsi akar kuadrat.

Hal yang menyulitkan sekarang adalah mengendalikan jumlah akselerasi sedemikian rupa sehingga kendaraan :

a. Berhenti (tidak bergerak perlahan dari depan ke belakang).
b. Berhenti tepat pada posisi yang ditentukan (misalnya kita ingin kendaraan melaju tepat 1 meter dan berhenti dalam jarak 1mm).

Gambar 2.8 (kanan) menunjukkan bagaimana adaptasi pengereman dapat mempengaruhi perilaku pergerakan agar dapat sesuai dengan poin a dan b diatas dengan mengontrol (terus memperbarui) percepatan pengereman yang diterapkan. Prosedur kontrol ini harus memperhitungkan kecepatan dan posisi saat ini sebagai nilai umpan balik, karena perubahan atau ketidakakuratan kecepatan sebelumnya mungkin telah berpengaruh pada posisi kendaraan [10].

BAB 3
METODE PENELITIAN

3.1 Tempat Penelitian

Penelitian tugas akhir dilaksanakan dalam jangka waktu 4 bulan. Penelitian bertempat di Laboratorium Mekatronika, Jurusan Teknik Elektro, Universitas Jenderal Soedirman yang bertempat di Jl. Raya Mayjen Sungkono No.KM 5, Dusun 2, Blater, Kec. Kalimanah, Kabupaten Purbalingga.

3.2 Alat dan Bahan

Penelitian ini membutuhkan alat dan bahan sebagai berikut.

3.2.1 Alat:

- Robot Prototipe *X-Drive*: Robot dengan konfigurasi *X-Drive* dilengkapi dengan 4 roda omni untuk pergerakan holonomik.
- Mikrokontroler STM32F411CEU6: Digunakan untuk kontrol motor dan pembacaan *rotary encoder*.
- Raspberry Pi Pico: Digunakan untuk membaca sensor IMU.
- Raspberry Pi 5: Berfungsi sebagai *single board computer* untuk menjalankan algoritma *dead reckoning*, kendali PID, dan mengelola komunikasi antar mikrokontroler.
- *Rotary Encoder*: Digunakan untuk mengukur kecepatan dan jarak tempuh roda.
- IMU (*Inertial Measurement Unit*): Mengukur orientasi robot (roll, pitch, yaw).
- DC Motor: Motor penggerak roda dengan pengendali kecepatan yang diatur melalui PID.

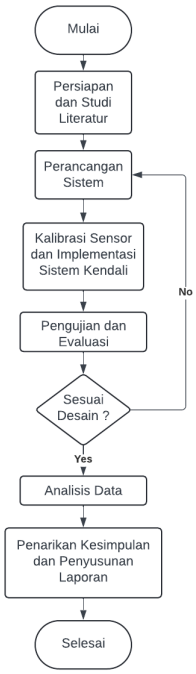
- Power Supply: Digunakan untuk memberikan daya pada seluruh komponen elektronik.
- Komputer: Digunakan untuk memprogram STM32 dan Raspberry Pi.

3.2.2 Bahan:

- Kabel dan konektor: Untuk menghubungkan komponen elektronik.
- Sistem rangka robot: Struktur fisik robot yang mendukung komponen-komponen seperti motor dan roda omni.

3.3 Alur dan Tahapan Penelitian

Metode yang digunakan dalam penelitian ini berupa rancang bangun yang terbagi menjadi beberapa tahap utama, mulai dari perancangan sistem hingga pengujian robot di lapangan. Gambar 3.1 berikut adalah alur dan tahapan penelitian yang akan dilaksanakan :



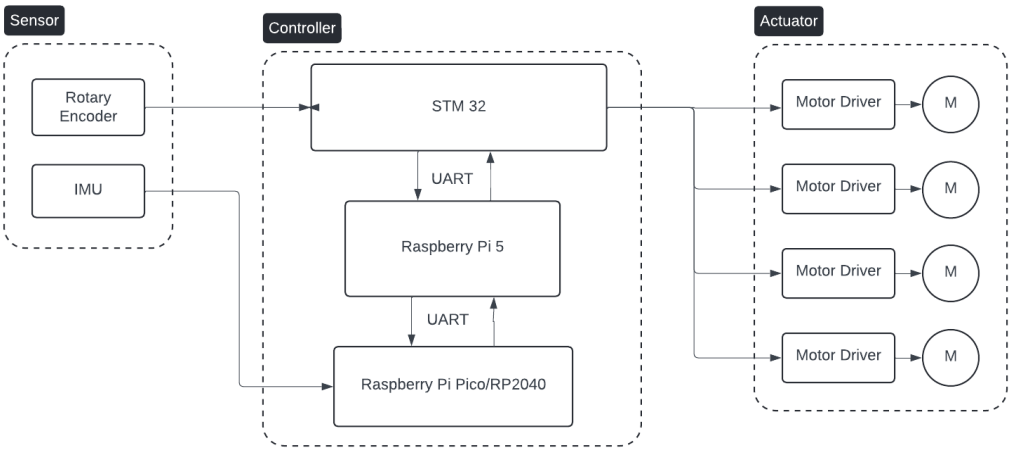
Gambar 3.1 Diagram Alir Tahapan Penelitian

3.3.1 Persiapan dan Studi Literatur

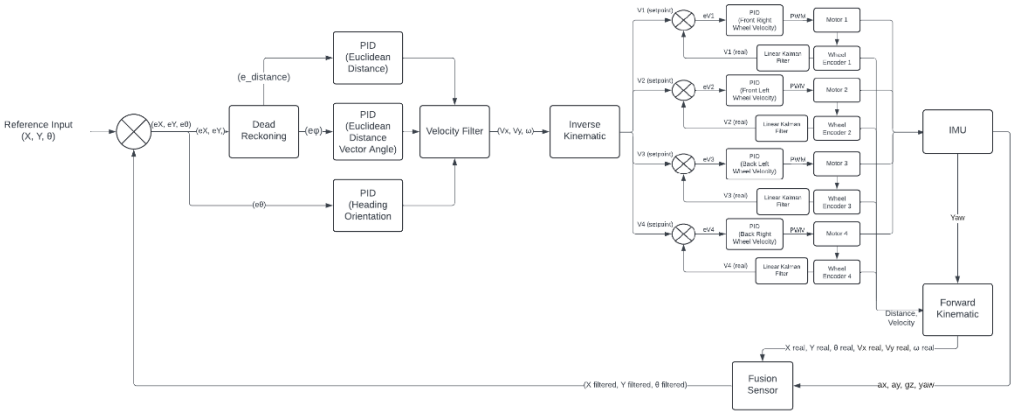
Tahap awal dalam penelitian yaitu persiapan dan studi literatur. Persiapan ini meliputi observasi objek penelitian, identifikasi masalah, menentukan rumusan masalah, tujuan penelitian dan batasan-batasan masalah serta studi pustaka yang bersumber dari buku, jurnal ilmiah dan sumber referensi lain yang kredibel. Langkah studi literatur diperlukan untuk memperkuat dasar penelitian dan menjadi acuan dalam menyelesaikan masalah yang dibahas serta melakukan perbandingan dengan penelitian-penelitian lain yang sejenis.

3.3.2 Perancangan Sistem

Pada tahap ini, peneliti melakukan analisa terhadap bentuk robot untuk membuat model pergerakan robot. Lalu dirancang skema algoritma dan membuat *source code* kontrol PID kecepatan dan posisi robot pada Raspberry Pi 5, program untuk membaca sensor IMU serta pembacaan *rotary encoder* dan kontrol kecepatan motor akan dibuat pada Raspberry Pi Pico dan STM32F411CEU6 dengan spesifikasi sistem seperti pada Gambar 3.2 dan 3.3.



Gambar 3.2 Diagram Blok Perangkat Keras



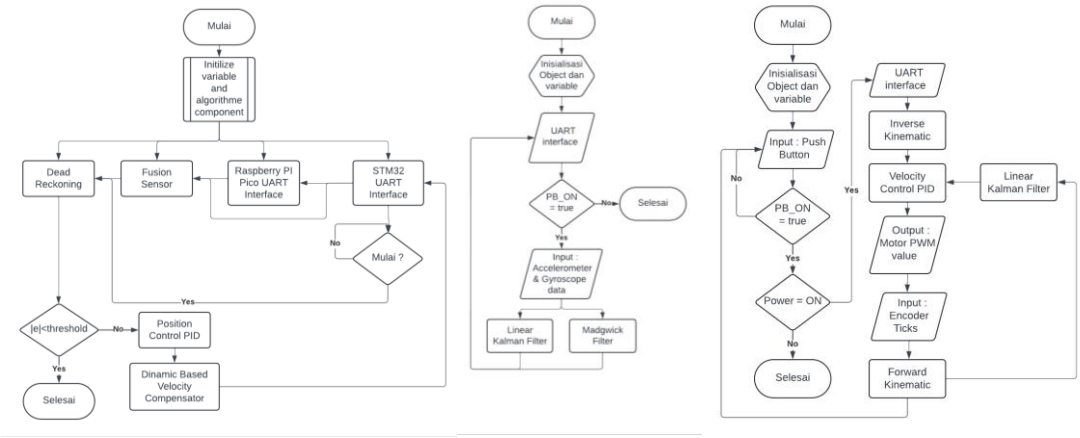
Gambar 3.3 Diagram Blok Sistem Kendali

Diagram blok pada gambar 3.3 diatas menunjukkan proses dari sinyal error masukan yang berasal dari nilai koordinat (X, Y, θ) dan sinyal feedback dari fusi sensor yang diolah oleh *Dead Reckoning* untuk menghasilkan nilai dan arah posisi serta rotasi sebagai masukan dari 3 buah algoritma PID posisi untuk menghasilkan nilai kecepatan. Sebelum dapat digunakan sebagai perintah kecepatan bagi robot, nilai kecepatan yang didapatkan terlebih dahulu dilakukan *filtering* untuk menghasilkan nilai kecepatan akhir untuk mengkompensasi efek-efek dinamis dari robot ketika bergerak serta melakukan pembatasan percepatan untuk menghindari slip pada roda.

Pada prinsipnya, filter kecepatan yang digunakan berfungsi untuk menghitung nilai kecepatan akhir dengan mengadopsi persamaan dinamika robot beroda secara umum yang dimodifikasi agar tidak melibatkan masa, inersia, torsi atau gaya secara langsung. Selain itu terdapat pembatas percepatan untuk menghindari adanya lonjakan atau penurunan nilai kecepatan yang terlalu besar yang dihasilkan oleh PID. Cara kerjanya adalah dengan menghitung selisih kecepatan hasil PID dengan kecepatan robot saat ini untuk memperoleh percepatan,

lalu nilainya dibandingkan dengan nilai percepatan maksimum sistem untuk periode kontrol tertentu untuk mendapat nilai minimum dari keduanya sebagai nilai akhir. Nilai kecepatan akhir adalah nilai dari kecepatan robot saat ini ditambah dengan percepatan akhir yang diperoleh.

Nilai kecepatan akhir yang didapatkan kemudian digunakan bagi algoritma kinematika invers untuk menghitung nilai kecepatan setiap roda sebagai masukan referensi pagi PID kecepatan roda. Untuk detail rancangan algoritma dari sistem adalah seperti yang ditunjukkan pada Gambar 3.4 di bawah ini.



(a) (b) (c)

Gambar 3.4 Diagram Alir Program (a) Raspberry Pi, (b) Raspberry Pi Pico, (c) STM32F411CEu6

3.3.3 Kalibrasi Sensor dan Implementasi Sistem Kendali

Algoritma kendali yang telah dirancang diimplementasikan pada robot. Tahap ini meliputi pengujian akhir sensor dan penentuan parameter kendali yang berupa pengujian *Rotary Encoder* dan IMU untuk menguji akurasi pembacaan encoder dan sensor IMU dalam mengukur kecepatan, jarak tempuh roda dan orientasi.

Selain itu, juga dilakukan kalibrasi parameter algoritma PID menggunakan metode *trial-and-error*. Nilai konstanta Kp disesuaikan untuk mempercepat respons sistem tanpa menyebabkan osilasi berlebihan. Ki ditambahkan untuk mengurangi *steady-state error*, namun diatur agar tidak menyebabkan osilasi. Serta Kd digunakan untuk mengantisipasi perubahan error yang tiba-tiba dan menekan *overshoot*.

Parameter PID dianggap optimal jika nilai *overshoot* kurang dari 10%, *settling time* minimal tanpa timbul osilasi tambahan serta *steady state error* mendekati 0 dengan tuning dilakukan pada kondisi motor tanpa beban atau dalam artian robot tidak digerakan langsung pada lantai. Nilai respon sistem dengan sedikit *overshoot* (*overdamped*) ataupun *critically damped* adalah yang dicari untuk itu agar saat dijalankan pada permukaan redaman alami sistem tidak membuat responnya menjadi *underdamped* yang terlalu parah.

3.3.4 Pengujian dan Evaluasi

Pada tahap ini, dilakukan serangkaian pengujian untuk mengevaluasi performa algoritma kendali yang telah diimplementasikan. Pengujian dilakukan untuk :

- Mengevaluasi pengaruh variasi konstanta PID terhadap akurasi pergerakan robot dengan 3 variasi kecepatan.
- Menguji pengaruh kecepatan robot terhadap akurasi pergerakan.
- Menilai akurasi kendali posisi dalam menentukan pergerakan robot terhadap *trajectory* ideal dan diuji dengan cara memberikan sejumlah koordinat posisi

yang mengharuskan robot untuk bergerak linear, berotasi serta bergerak linear sembari berotasi.

Pengujian akhir dilakukan di lapangan datar untuk melihat kinerja robot dalam skenario nyata. Robot akan bergerak secara otonom dari titik awal menuju beberapa titik poin koordinat yang telah dipersiapkan sebelumnya dengan pola tertentu dalam 3 variasi kecepatan yang berbeda untuk menghitung nilai kesalahan dari posisi robot.

3.3.5 Analisis Data

Data hasil pengujian dianalisis untuk mengetahui sejauh mana sistem kendali yang dirancang mampu mencapai target yang diinginkan. Analisis ini mencakup evaluasi kesalahan posisi robot, serta efektivitas kendali PID dalam mencapai target kecepatan. Data hasil pengujian untuk (posisi serta kecepatan) akan dikumpulkan dalam berkas berektensi mcap. Analisis dilakukan menggunakan Bahasa Python dalam menghitung untuk data posisi dengan menghitung nilai :

1. RMSE

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Target_i - Aktual_i)^2} \quad (3.1)$$

2. *Deviasi Lateral*

$$Deviation = \sqrt{(X\ target_i - X\ aktual_i)^2 + (Y\ target_i - Y\ aktual_i)^2} \quad (3.2)$$

Serta untuk data kecepatan yakni :

1. *Settling time* / waktu tempuh mencapai kestabilan
2. *Overshoot*

$$Overshoot\ (\%) = \frac{Max\ Response - Setpoint}{Setpoint} x\ 100\% \quad (3.3)$$

3. *Steady State Error*

$$SSE = |Setpoint - Nilai\ Akhir| \quad (3.4)$$

Hasil analisis ini akan digunakan untuk mengevaluasi apakah sistem kendali yang dikembangkan telah memenuhi kriteria desain dan dapat menghasilkan pergerakan yang optimal.

3.3.6 Penarikan Kesimpulan dan Penyusunan Laporan

Setelah analisis data selesai dilakukan, peneliti akan menarik kesimpulan berdasarkan hasil penelitian. Kesimpulan ini mencakup sejauh mana sistem kendali yang dirancang mampu mengontrol pergerakan robot *X-Drive* dengan akurasi yang baik. Tahap ini juga melibatkan penyusunan laporan akhir yang mendokumentasikan seluruh proses dan hasil penelitian.

3.4 Waktu dan Jadwal Penelitian

Penelitian ini direncanakan berlangsung selama 4 bulan yang dimulai sejak bulan Agustus hingga November 2024, dengan rincian waktu untuk setiap tahapan seperti pada Tabel 3.1 sebagai berikut :

Tabel 3.1 Jadwal Penelitian

No	Tahapan Penelitian	Bulan 1				Bulan 2				Bulan 3				Bulan 4			
		1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
1	Persiapan dan Studi Literatur																
2	Perancangan Sistem																
3	Kalibrasi Sensor dan Implementasi Sistem Kendali																
4	Pengujian dan Evaluasi																
5	Analisis Data																
6	Penarikan Kesimpulan dan Penyusunan Laporan																

BAB 4
HASIL DAN PEMBAHASAN

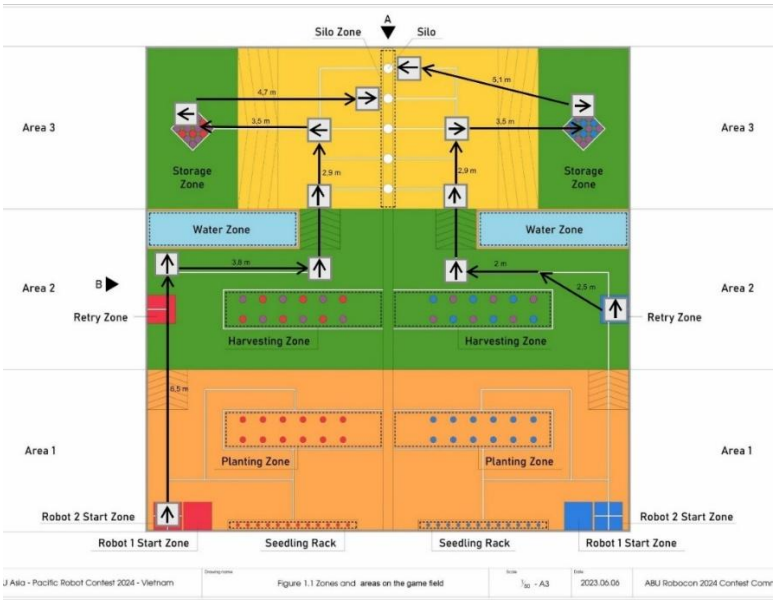
4.1 Robot Otonom Beroda *X-Drive* dengan Kendali PID dan Kinematika

Konfigurasi *X-Drive* pada robot beroda adalah ketika roda omni yang dipasang, diletakan dalam pola menyerupai huruf "X" jika dilihat dari atas. Setiap roda dipasang dengan sudut ideal sekitar 45° terhadap sumbu utama robot, memungkinkan robot untuk bergerak dalam segala arah tanpa perlu perubahan orientasi atau holonomik. Dalam sistem kendali robot beroda *X-Drive*, digunakan metode kontrol berbasis Proporsional-Integral-Derivatif (PID) untuk mengatur nilai kecepatan, arah gerak serta arah hadap robot berdasarkan posisi saat ini.

Pada penelitian ini, kendali PID juga diterapkan untuk memastikan bahwa setiap roda omni dapat mencapai kecepatan yang sesuai dengan target yang telah dihitung dari model kinematika robot. Dengan menggunakan sistem kendali ini, diharapkan robot dapat bergerak dengan akurasi tinggi dalam skenario kompetisi, termasuk gerakan translasi ke segala arah, rotasi pada tempat, serta pergerakan kombinasi translasi dan rotasi secara simultan.

4.2 Analisis Spesifikasi Gerak Robot

Pada Kontes Robot ABU Indonesia 2024, robot dituntut untuk bergerak secara otonom dari satu titik ke titik lain dalam arena. Untuk itu skema pergerakan robot dicanangkan seperti yang ditunjukkan pada Gambar 4.1 di bawah ini dengan arah panah dan kotak menyatakan arah orientasi dan posisi robot. Secara umum, dapat diuraikan sebagai berikut.



Gambar 4.1 Rencana Jalur Gerak Robot

Untuk skema pergerakan terpanjang dilakukan saat awal permainan dengan asumsi posisi relatifnya dituliskan dalam nilai koordinat yang terdiri dari posisi dalam *centimeter* ke arah X dan Y serta orientasi dalam derajat (0, 0, 0^o). Robot harus bergerak lurus dari *start zone* menuju ke titik (0, 650, 0^o), lalu diikuti dengan pergerakan lateral untuk menuju ke titik (380, 650, 0^o). Tidak berhenti disitu, robot harus bergerak lurus melewati jembatan ke titik (380, 750, 0^o) untuk kemudian melakukan perubahan orientasi ketika bergerak ke titik (380, 840, 90^o) dan menuju ke *storage zone* pada zona 3 yang terletak pada titik (30, 840, 90^o). Selain itu juga terdapat pergerakan dari *retry zone* yang perlu dilakukan untuk menuju ke *storage zone* serta pergerakan secara bolak-balik dari *silo zone* ke *storage zone*. Untuk itu robot harus memiliki :

1. **Kemampuan Gerak Linear (X, Y):**
    - Robot harus mampu bergerak bebas dalam ruang dua dimensi tanpa terganggu oleh orientasi awal.

- Kecepatan linear harus diatur agar sesuai dengan kondisi lingkungan, seperti zona sempit atau lintasan lurus panjang.

2. **Kemampuan Gerak Rotasi (θ):**
   - Robot perlu melakukan rotasi di tempat untuk menyesuaikan orientasi saat berinteraksi dengan objek atau navigasi lintasan.
   - Keakuratan rotasi sangat penting untuk memastikan bahwa robot menghadap ke arah yang benar.
3. **Kombinasi Linear dan Rotasi:**
   - Dalam beberapa skenario, robot harus bergerak sambil berputar untuk mencapai posisi yang diinginkan dengan orientasi tertentu untuk mempercepat pergerakan.
   - Hal ini membutuhkan algoritma kendali yang mampu menangani penggabungan kecepatan linear dan rotasi secara simultan.

Luas arena total adalah 12x12 meter yang dibagi menjadi 2 buah area tim. Berdasarkan Gambar 4.1 diatas, diperoleh bahwa jarak terpanjang yang harus ditempuh oleh robot untuk satu siklus pergerakan adalah 16,7 m yakni jalur gerak robot dari *start zone* hingga *storage zone* saat game dimulai. Dengan target waktu tempuh untuk memasuki *storage zone* pertama kali adalah sekitar 15 detik, maka robot setidaknya harus menempuh kecepatan sebanyak 16,7/15 yakni 1,113 m/s.

Jika robot bergerak berkecepatan 1.113 m/s dalam jarak 16,7 m, dengan periode kontrol 10 ms, robot akan bergerak sekitar 0,001113 m (1,113 cm) selama setiap siklus 10 ms dari algoritme kontrol. Jika ditetapkan akurasi gerak robot yang digunakan adalah 0,5 cm sebagai jarak yang berarti dalam hal mendekati suatu

objek atau titik. Maka dengan kecepatan yang sama diperoleh kebutuhan periode kontrol sebesar :

$$T = \frac{target\ accuracy}{velocity} \tag{4.1}$$

$$= \frac{0{,}005}{1{,}113} = 0{,}00449 \text{ s} = 4{,}49 \text{ ms} \approx 5 \text{ ms}$$

Angka akurasi 0.5 cm ini ditentukan berdasarkan kebutuhan aplikasi robot dalam kompetisi, yakni Ketika robot harus mampu mendekati objek atau lokasi dengan presisi tinggi, seperti mengambil atau menempatkan objek. Dalam skala arena kompetisi, akurasi 0,5 cm dianggap cukup kecil untuk memastikan robot tidak meleset dari target, terutama dalam tugas seperti navigasi lintasan sempit. Selain itu, akurasi ini sejalan dengan standar umum dalam robotika kompetisi, di mana kesalahan posisi di bawah 1 cm sudah dianggap optimal.

Pada *framework* Arduino IDE, terdapat fungsi millis yang akan mengembalikan nilai berupa waktu sejak piranti dihidupkan dalam skala 1 milidetik. Nilai 4,49 milidetik tidaklah bulat sehingga untuk mempermudah visualisasi dan implementasi pada program, nilai 4,49 dibulatkan keatas menjadi 5 ms, sehingga nilai periode sistem yang digunakan adalah 5 ms. Namun dengan target akurasi yang sama yakni 0,5 cm, kecepatan robot yang digunakan berkurang yakni sebanyak 1 m/s saja sehingga nilai ideal robot dapat mencapai *storage zone* adalah 16,7 detik yang seharusnya tidak terlalu bermasalah.

4.3 Pemodelan Pergerakan Robot

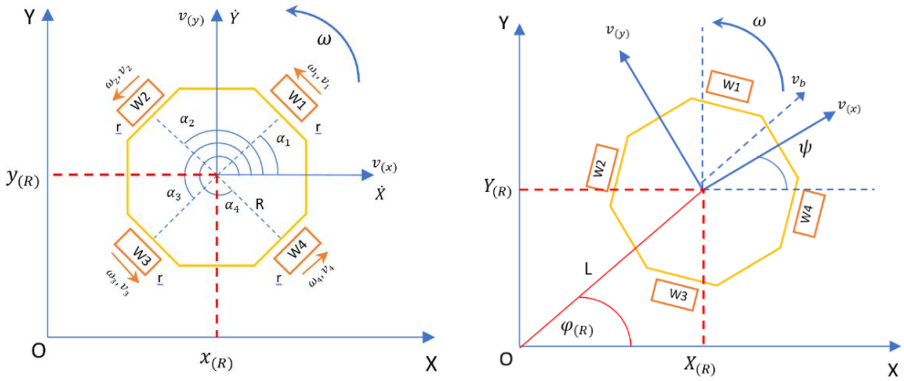
Pemodelan kinematika bertujuan untuk menggambarkan hubungan antara kecepatan gerak motor yang diberikan sebagai masukan dan pergerakan keseluruhan robot sebagai keluaran. Model ini sangat penting dalam sistem kendali

karena menjadi dasar dalam perhitungan nilai kecepatan roda agar robot dapat bergerak sesuai dengan perintah yang diberikan.

Robot yang digunakan dalam penelitian ini mengadopsi konfigurasi *X-Drive* dan diperuntukan untuk menjalani seleksi perlombaan dalam Kontes Robot ABU Indonesia 2024. Setelah memahami dasar kinematika dari sistem *X-Drive*, selanjutnya akan dibahas lebih lanjut mengenai model pergerakan robot berdasarkan analisis kinematika maju (*forward kinematics*) dan kinematika balik (*inverse kinematics*), yang menjadi dasar perhitungan kecepatan roda dalam setiap skenario pergerakan.

4.3.1 Kinematika Rangka Robot Roda Omni bertipe *X-Drive*

Untuk robot dengan 4 buah roda omni seperti pada Gambar 4.2 dibawah ini, berdasarkan persamaan 2.2, maka rumus invers kinematika yang berlaku dengan nilai v_g merupakan nilai yang telah ditentukan baik oleh pengguna ataupun oleh algoritma kontrol, persamaannya yakni :



Gambar 4.2 Kinematika Robot Beroda X-Drive

$$v_w = \begin{bmatrix} v_{w1} \\ v_{w2} \\ v_{w3} \\ v_{w4} \end{bmatrix} = T(\alpha) v_g \tag{4.2}$$

Nilai v_g terdiri dari $[v_x, v_y, \text{dan } \omega]^T$ adalah kecepatan robot, kecepatan untuk setiap rodanya adalah $v_w = [v_1\ v_2\ v_3\ v_4]^T$. Dengan matriks $T(\alpha)$ adalah persamaan matriks rotasi untuk setiap roda pada robot yang bisa ditulis dengan persamaan berikut :

$$T(\alpha) = \begin{bmatrix} -sin(\alpha_1) & \cos(\alpha_1) & R\cos(\alpha_{ideal1} - \alpha_1) \\ -sin(\alpha_2) & \cos(\alpha_1) & R\cos(\alpha_{ideal2} - \alpha_2) \\ -sin(\alpha_3) & \cos(\alpha_1) & R\cos(\alpha_{ideal3} - \alpha_3) \\ -sin(\alpha_4) & \cos(\alpha_1) & R\cos(\alpha_{ideal4} - \alpha_4) \end{bmatrix} \tag{4.3}$$

Nilai α_{ideal1} pada matriks $T(\alpha)$ menunjukkan nilai sudut ideal dari posisi roda, dalam manufaktur sering kali terdapat error yang membuat nilai sudut yang sebenarnya tidak sama dengan sudut ideal pada rancangan awal. Ini mungkin tidak akan berpengaruh terhadap komponen gerak linear, namun tidak dengan komponen gerak angular atau rotasinya. Oleh karena itu komponen angular dari matriks $T(\alpha)$ perlu dikalikan dengan nilai cos dari selisih sudut ideal dan riil. Lalu untuk solusi persamaan *forward* kinematika untuk menemukan nilai kecepatan robot v_g didapat melalui persamaan :

$$v_g = \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} = D(\alpha) \begin{bmatrix} v_{w1} \\ v_{w2} \\ v_{w3} \\ v_{w4} \end{bmatrix} \tag{4.4}$$

Matriks $D(\alpha)$ adalah invers dari matriks rotasi $T(\alpha)$. Karena matriks $T(\alpha)$ bukanlah matriks persegi, maka invers yang didapat adalah berupa invers semu. Sehingga untuk mempermudah, dibuat persamaan baru untuk nilai v_g sebagai fungsi dari v_w berdasarkan Gambar 4.2 diatas. Karena nilai sudut α tidak selalu sama dengan nilai rancangannya, matriks w_n ditambahkan untuk menyeimbangkan kontribusi dari setiap roda (v_{wn}). Berikut persamaanya :

$$D(\alpha) = \frac{1}{2}\begin{bmatrix} -sin(\alpha_1) & -sin(\alpha_2) & -sin(\alpha_2) & -sin(\alpha_2) \\ \cos(\alpha_1) & \cos(\alpha_2) & \cos(\alpha_2) & \cos(\alpha_2) \\ 2R^{-1}\cos(\alpha_1) & 2R^{-1}\cos(\alpha_2) & 2R^{-1}\cos(\alpha_3) & 2R^{-1}\cos(\alpha_4) \end{bmatrix}\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} \quad (4.5)$$

Nilai kecepatan linear roda (v_{wn}) perlu diperoleh melalui pembacaan data odometri yang berasal dari *rotary encoder* yang berupa perubahan posisi robot menggunakan persamaan :

$$v_{wn} = \frac{\dot{\text{p}}_n}{PPR}\, x\, K\,, \qquad n = 1, 2, 3, 4 \quad (4.6)$$

Nilai v_{wn} adalah kecepatan aktual linear roda, dan $\dot{p}_{\text{n}}$ adalah jumlah perubahan pulsa encoder dengan waktu yang dihitung menggunakan persamaan 4.6. PPR adalah jumlah pulsa per revolusi yang ditentukan oleh spesifikasi dari sensor encoder yang dipakai, K adalah keliling roda, serta n notasi nomor roda.

$$\dot{\text{p}}_n = \frac{dp_n}{dt}\,, \qquad n = 1, 2, 3, 4 \quad (4.7)$$

Hasil dari persamaan *direct* kinematika, seperti pada persamaan 4.4, digunakan untuk menghitung koordinat posisi dan orientasi robot ($X(R)$, $Y(R)$, θ) dengan mengakumulasi nilai perubahan pulsa per selang periode sistem yang diperoleh dari perhitungan kinematika maju yakni sebagai berikut :

$$x_{(R)} = \sum_{i=0}^{n} \Delta\dot{x}_i\,, \qquad i = 1, 2, 3, \ldots \quad (4.8)$$

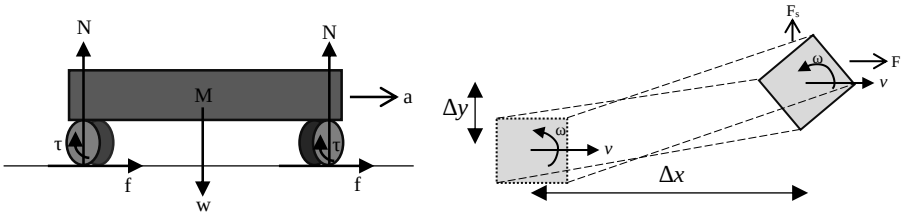
$$y_{(R)} = \sum_{i=0}^{n} \Delta\dot{y}_i\,, \qquad i = 1, 2, 3, \ldots \quad (4.9)$$

$$\theta = \sum_{i=0}^{n} \Delta\dot{\theta}_i\,, \qquad i = 1, 2, 3, \ldots \quad (4.10)$$

4.3.2 Kinematika Pergerakan Robot

Idealnya, ketika sebuah robot diberikan masukan berupa kecepatan tertentu ke arah tertentu, nilai dan arah yang sama dihasilkan oleh robot tersebut. Namun pada kenyataannya, banyak aspek yang dapat mempengaruhi bagaimana

sebuah robot itu bergerak terutama ketika kecepatan tinggi diaplikasikan. Kinematika hanya fokus pada hubungan antara masukan gerakan (kecepatan roda atau robot) dengan keluaran gerakan (posisi, kecepatan, atau orientasi) tanpa mempertimbangkan gaya atau momen yang menyebabkannya. Untuk mengatasi efek gangguan yang terjadi diperlukan dinamika. Dinamika mempelajari gaya dan momen yang menyebabkan gerakan robot, termasuk bagaimana gaya-gaya tersebut memengaruhi perubahan kecepatan dan posisi robot. Gambar 4.3 dibawah ini menunjukkan arah gaya-gaya bekerja pada sebuah robot beroda.



Gambar 4.3 Diagram Benda Bebas Pergerakan Robot Beroda

Persamaan 2.3 menunjukkan model dinamika bagaimana pergerakan sebuah robot dapat dipengaruhi oleh beberapa efek dan gangguan yang terjadi. Robot beroda seperti namanya menggunakan roda untuk menghasilkan pergerakan linear (ke depan, belakang, serong atau ke samping) dan rotasi (mengubah orientasi). Gerakan ini ditentukan oleh masukan kecepatan atau torsi pada roda, yang menghasilkan posisi, kecepatan, dan orientasi tertentu. Namun untuk melakukan implementasi dinamika pada robot, diperlukan parameter gaya/torsi dan massa/inersia selain posisi, kecepatan dan percepatan. Nilai massa/inersia dapat diperoleh melalui pengukuran ataupun perhitungan secara matematis, namun untuk memperoleh nilai gaya/torsi diperlukan parameter tegangan dan arus yang digunakan oleh setiap komponen aktuator, parameter tersebut digunakan untuk

memperoleh nilai daya yang digunakan untuk menghitung gaya/torsi, hal tersebut belum bisa disediakan oleh robot untuk saat ini.

Agar bisa setidaknya mengurangi efek-efek dinamis yang bisa terjadi dengan parameter tersedia yang ada, maka rumus pada persamaan 2.3 dapat diadaptasi ke dalam model kinematika baru yang tidak melibatkan aspek massa/inersia dan gaya/torsi secara langsung. Untuk itu, didapatkan persamaan model dinamika yang disederhanakan dengan menghilangkan aspek vektor gangguan, matriks kinematika dan konstanta lagrange yang sebelumnya digunakan, maka persamaannya menjadi :

$$E(q)u = M(q) \cdot \ddot{q} + V(q, \dot{q}) + F(\dot{q}) + G(q) \quad (4.11)$$

Dari persamaan 4.11 diatas, dapat disimpulkan bahwa aspek-aspek yang dapat mempengaruhi nilai keluaran u pergerakan robot adalah akumulasi dari gaya akibat dari efek akselerasi pergerakan robot, gaya sentrifugal dan koriolis akibat dari gerakan dalam kerangka acuan yang berputar, lalu gaya friksi antara roda dengan permukaan medan serta gaya gravitasi ketika robot bergerak dalam bidang miring (dalam kasus medan datar, gaya gravitasi dapat diabaikan).

A. Identifikasi Pengaruh Gaya Dinamis

1. **Massa/Inersia (*M(q)*) dan akselerasi/percepatan:**
   - Inersia menggambarkan resistansi robot terhadap perubahan kecepatan. Semakin besar massa robot, semakin besar gaya yang dibutuhkan untuk mempercepat atau memperlambatnya.
   - Efek ini memengaruhi respons waktu sistem, terutama ketika robot bergerak dengan percepatan tinggi.

2. **Gaya Koriolis dan Sentrifugal ($\boldsymbol{V(q, \dot{q})}$):**
   - Gaya koriolis timbul ketika robot bergerak dengan kecepatan sudut tinggi, menghasilkan gaya lateral yang dapat menyebabkan slip atau ketidakstabilan arah.
   - Dalam robot *X-Drive*, pengaruh ini harus diatasi untuk menjaga lintasan yang diinginkan.
3. **Gesekan ($\boldsymbol{F(\dot{q})}$):**
   - Gesekan statis memengaruhi inisiasi gerak, sedangkan gesekan dinamis menahan pergerakan selama robot bergerak.
   - Dalam roda omni, gesekan cenderung lebih rendah, sehingga meningkatkan risiko slip, terutama saat percepatan tinggi atau ketika beban tambahan diberikan.
4. **Gravitasi ($\boldsymbol{G(q)}$):**
   - Jika robot berjalan pada permukaan miring dengan sudut kemiringan α, efek gravitasi akan mempengaruhi kecepatan pada sumbu x dan y.

B. Adaptasi Dinamika ke Model Kinematika Baru

1. **Mengabaikan Faktor Massa/Inersia Langsung:**
   - Massa, inersia, dan parameter gaya lainnya tidak dimasukkan secara eksplisit. Sebagai gantinya, efeknya dimodelkan sebagai batasan percepatan ($a_{\max}$) pada robot.
   - Persamaan percepatan disederhanakan sebagai:

$$a = \frac{v_{target} - v_{current}}{\Delta t} \tag{4.12}$$

   dengan a dibatasi oleh $a_{\max}$.

2. **Kompensasi Efek Coriolis:**
   - Efek coriolis disimulasikan secara kinematik dengan memperhatikan kecepatan gerak linear robot (*v*) dan laju kecepatan angular (ω). Kompensasi ini dilakukan dengan menambahkan faktor koreksi (k_c) dalam perhitungan kinematika:

$$v_{coriolis} = -k_c v\omega = -k_c \omega^2 R \quad (4.13)$$

3. **Kompensasi Efek Gesekan dan Grafitasi :**
   - Efek gesekan dimasukkan melalui limiter percepatan, yakni untuk memperoleh percepatan maksimum $a_{\max}$ yang diperoleh melalui perhitungan dibawah ini, nilai α adalah sudut kemiringan bidang lintasan robot dan nilai μ merupakan nilai pendekatan untuk koefisien gaya gesek.

$$a_{max} = \mu g\, cos(\alpha) - g \sin(\alpha) \quad (4.14)$$

Sehingga, untuk persamaan kinematika akhir hasil modifikasi tersebut didapat :

$$v = v_{current} + \min(a, a_{max}) \quad (4.15)$$

$$v_{final} = v - v_{coriolis} \quad (4.16)$$

Dengan pendekatan ini, efek dinamis diakomodasi dalam model kinematika tanpa perlu melibatkan parameter seperti massa, torsi, atau gaya secara langsung. Pendekatan ini lebih sederhana untuk implementasi program dan tetap mampu memberikan respons yang lebih stabil pada sistem kendali robot. Slip diperhitungkan dengan menambahkan filter percepatan pada algoritma kendali. Filter ini memastikan bahwa perubahan percepatan tidak melebihi batas tertentu yang dapat menyebabkan roda kehilangan traksi. Meskipun efek coriolis biasanya kecil dalam skala robot, model ini melakukan kompensasi dengan memperhitungkan gaya lateral yang muncul saat robot berakselerasi di lintasan melengkung dan/atau dengan berotasi.

4.4 Implementasi Sistem Kendali

4.4.1 Kendali Kecepatan Pergerakan Robot Melalui Aktuator Motor DC

Implementasi kendali untuk kecepatan pergerakan robot dilakukan dengan mengatur kecepatan setiap roda secara presisi berdasarkan masukan kecepatan berupa nilai yang sudah ditentukan sebelumnya ataupun nilai yang berasal dari sistem kendali utama pada Raspberry Pi 5. Pengendalian kecepatan pergerakan robot diwujudkan melalui aktuator motor DC menggunakan algoritme PID yang diimplementasikan pada mikrokontroler STM32.

Algoritma diterapkan pada mikrokontroler berjenis STM32F4Ceu6 yang diprogram menggunakan *software* dan *framework* Arduino IDE. Hasil dari algoritma ini adalah sinyal PWM (*Pulse Width Modulation*) yang digunakan oleh motor DC untuk mencapai kecepatan yang diinginkan. Kecepatan motor diawasi menggunakan *rotary encoder* sebagai umpan balik, yang memberikan data aktual mengenai kecepatan rotasi roda. Umpan balik ini digunakan untuk menghitung error kecepatan dan memperbarui sinyal PWM dengan algoritma PID.

A. Pemrosesan Nilai Referensi/*Setpoint* dan Umpan Balik ke Sistem Kendali Utama

Sistem kendali posisi dan kecepatan robot pada Raspberry Pi 5 mengirimkan beberapa nilai kecepatan yang harus ditempuh oleh robot. Ada 3 buah nilai yang diberikan yakni kecepatan linear terhadap sumbu Y serta sumbu X dan juga kecepatan angular (V_x, V_y, ω) (Program 4.2). Pengiriman nilai kecepatan tersebut dilakukan melalui komunikasi serial melalui protokol UART yang menghubungkan antara sistem kendali utama di Raspberry Pi 5 dan sistem kendali pergerakan robot pada STM32F4Ceu6. Implementasi

program komunikasi serial yang digunakan dilakukan menggunakan *library* `<SerialTransfer.h>` sehingga peneliti hanya perlu melakukan pembuatan objek serta pemanggilan method-method yang ada pada *library* tersebut. Berikut adalah potongan program implementasi protokol UART dalam komunikasi antar sistem kendali.

Program 4.1 Komunikasi Serial Raspberry Pi 5 ke STM32

```
#include "SerialTransfer.h"

/* --- Object --- */
SerialTransfer myTransfer;
struct __attribute__((packed)) STRUCTRX {
  float Vx;
  float Vy;
  float W;
  float psi;
} rxStruct;

struct __attribute__((packed)) STRUCTTX {
  float timestamp;
  float X; float Y; float tetha;
  float Vx; float Vy; float Wr;
  char cmd;
  char team;
} txStruct;
...
void setup(){
  Serial.begin(115200);
  ...
  myTransfer.begin(Serial);
}

void loop(){
  if (millis() - input_prevmillis >= inputrate){
    ...
    txStruct.timestamp = (float) (Timing-prev_Timing);
    txStruct.X = (float) calc.dist_travel[0];
    txStruct.Y = (float) calc.dist_travel[1];
    txStruct.tetha = (float) calc.dist_travel[2];
    txStruct.Vx = (float) calc.Vreal[0];
    txStruct.Vy = (float) calc.Vreal[1];
    txStruct.Wr = (float) calc.Vreal[2];
```

```
    txStruct.team = (char) team;
    if(myTransfer.available()){
      uint16_t sendSize = 0;
      sendSize = myTransfer.txObj(txStruct, sendSize);
      myTransfer.sendData(sendSize);

      uint16_t recSize = 0;
      recSize = myTransfer.rxObj(rxStruct, recSize);
    }
    input_prevmillis = millis();
  }
}
```

Untuk membuat program komunikasi serial, peneliti memasukan *library* ke dalam program diikuti dengan pembuatan objek serta variabel tempat menyimpan pesan yang akan dikirim atau diterima dalam format *typedef struct.* Hal ini dilakukan karena nilai dan jenis pesan yang dikomunikasikan cukup beragam. Untuk memulai komunikasi, *method* `begin()` dipanggil pada prosedur setup. Pada prosedur loop, peneliti hanya perlu memodifikasi nilai variable pesan yang telah dibuat dan secara berkala mengirimkan variable tersebut sebagai umpan balik dengan memanggil method `txObj()`, `sendData()`, serta `rxObj()` untuk membaca data yang diterima.

Selanjutnya, ketiga nilai kecepatan yang diterima tersebut diolah melalui algoritma kinematika invers (Program 4.3) berdasarkan persamaan 4.2 untuk memperoleh kecepatan setiap rodanya, hasil kecepatan yang positif mengindikasikan kondisi motor dc yang akan berputar berlawanan arah jarum jam sedangkan nilai yang negatif menunjukkan arah putaran motor dc yang harus searah jarum jam. Selain itu sebagai pesan umpan balik kepada sistem kendali utama, dikirimkan data nilai kecepatan maupun posisi linear X dan Y

serta kecepatan angular dan orientasi dari robot secara *realtime.* Data tersebut nantinya akan digunakan kembali oleh sistem kendali utama untuk melakukan pengambilan keputusan bagi sistem kendali posisi.

Program 4.2 Konversi Kecepatan Robot ke Kecepatan Roda

```
void MoveRobot(){
  int Vx = rxStruct.Vx*max_linear_speed;
  int Vy = rxStruct.Vy*max_linear_speed;
  int W = rxStruct.W*max_angular_speed;
  float psi = rxStruct.psi;
  calc.update_angle(psi);
  calc.inverse_kinematics(Vx, Vy, W);
  ...
  calc.forward_kinematics(Vfilt1,Vfilt2,Vfilt3,Vfilt4,
  false);
  calc.forward_kinematics(ENCFR.read(),ENCFL.read(),
  ENCBL.read(),ENCBR.read(),true);
 }
```

Nilai hasil dari *method* inverse kinematika dari kelas *Kinematics* adalah berupa nilai kecepatan yang harus ditempuh oleh tiap rodanya. Nilai tersebut nantinya akan digunakan sebagai input dari algoritma PID pengendali kecepatan motor DC untuk menentukan nilai PWM yang diperlukan. Sedangkan pesan umpan balik kepada sistem kendali utama diperoleh salah satunya melalui *method direct* kinematika pada Program 4.3 dibawah ini. Nilai kecepatan serta perubahan jarak yang diperoleh dari sensor encoder berupa nilai *Rotation per Minute* (RPM) dan jumlah pulsa digunakan sebagai parameter masukannya. Sesuai dengan persamaan 4.6 diatas, maka nilai tersebut bisa dikonversi menjadi nilai nyata dari kecepatan robot serta posisinya saat ini.

Program 4.3 Fungsi Kinematika

```
void Kinematics::inverse_kinematics(float Vx,float Vy,float Wr){
  float V1, V2, V3, V4 = 0;
  V1 = round(-Vx*sin_value[0]+Vy*cos_value[0]+ Wr*abs(err_cos_value[0]));
  V2 = round(-Vx*sin_value[1] + Vy*cos_value[1] + Wr*abs(err_cos_value[1]));
  V3 = round(-Vxsin_value[2] + Vy*cos_value[2] + Wr*abs(err_cos_value[2]));
  V4 = round(-Vx*sin_value[3] + Vy*cos_value[3] + Wr*abs(err_cos_value[3]));
  Vwheel[0]=V1; Vwheel[1]=V2; Vwheel[2]=V3; Vwheel[3]=V4;
}

void Kinematics::forward_kinematics(float v1, float v2, float v3, float v4,
bool odom){
  float x_, y_, w_ = 0;
  if(odom){
    float err_v1=v1-this->prev_v1;
    float err_v2=v2-this->prev_v2;
    float err_v3=v3-this->prev_v3;
    float err_v4 = v4-this->prev_v4;
    x_=(err_v1*w1*(-sin_value[0])+err_v2*w2*(-sin_value[1])+
err_v3*w3*(-sin_value[2])+err_v4*(w4*-sin_value[3]))/2;
    y_=((err_v1*w1*(cos_value[0]))+(err_v2*w2*(cos_value[1]))+
(err_v3*w3*(cos_value[2]))+(err_v4*w4*(cos_value[3])))/2;
    w_=(err_v1*abs(err_cos_value[0])+
        err_v2*abs(err_cos_value[1])+
        err_v3*abs(err_cos_value[2])+
        err_v4*abs(this->err_cos_value[3]))/4;
    dist_travel[0] += x_*10*PI/PPR;
    dist_travel[1] += y_*10*PI/PPR;                dist_travel[2]       +=
(w_*10/PPR)/47.1699*360;
    prev_v1 = v1; prev_v2 = v2; prev_v3 = v3; prev_v4 = v4;
  } else{
    x_=(v1*w1*(-sin_value[0])+v2*w2*(-sin_value[1])+
        v3*w3*(-sin_value[2])+v4*w4*(-sin_value[3]))/2;
    y_=((v1*w1*(cos_value[0]))+(v2*w2*(cos_value[1]))+
(v3*w3*(cos_value[2]))+(v4*w4*(cos_value[3])))/2;
    w_=(v1*w1*abs(err_cos_value[0])+
        v2*w2*abs(err_cos_value[1])+
        v3*w3*abs(err_cos_value[2])+
        v4*w4*abs(this->err_cos_value[3]))/4;
    Vreal[0]=x_*10*PI/60; Vreal[1]=y_*10*PI/60;
    Vreal[2] = (w_*10/60)/47.1699*360;
  }
}
```

B. Realisasi Algoritma Kontrol

Kendali PID direalisasikan dengan memanfaatkan *library* `<QuickPID.h>` yang dikembangkan berdasarkan diagram blok dan *flowchart* program pada Gambar 3.2, 3.3 dan 3.4 (c). Hasil dari Program 4.3 merupakan nilai kecepatan roda 1 hingga 4, yang berarti setiap rodanya harus menempuh kecepatan sesuai dengan nilai yang dihasilkan. Untuk mencapai hal tersebut, fungsi nilai error yang otomatis dikalkulasi dalam *library* program adalah nilai dari :

$$e_n(t) = v_{inverse\_kinematic} - v_{actual} \tag{4.17}$$

Dalam aplikasinya, *library* `<QuickPID.h>` digunakan untuk mempermudah pengelolaan konstanta PID (K_p, K_i, K_d), setpoin serta error dengan membuat objek PID sesuai dengan argumen yang dibutuhkan. Berikut (Program 4.4) adalah potongan program untuk pembuatan objek menggunakan *library* pada salah satu motor.

Program 4.4 Objek Algoritma PID

```
#include <QuickPID.h>
float Output1 = 0, Kp1 = 0.475, Ki1 = 11, Kd1 = 0.01;
QuickPID motor1(&Vfilt1,&Output1,&calc.Vwheel[0],Kp1,Ki1,Kd1,
                      motor1.pMode::pOnError,
                      motor1.dMode::dOnMeas,
                      motor1.iAwMode::iAwCondition,
               motor1.Action::direct);
```

Program 4.5 menunjukkan potongan salah satu prosedur yang dipanggil untuk menjalankan algoritma PID pada motor 1. Prosedur `PID_init()` dipanggil sekali ketika program pertama kali dijalankan, yang berfungsi untuk melakukan pengaturan batasan keluar nilai hasil PID, waktu sampling, pengaturan konstanta serta mode kontrol yang digunakan. Lalu

sesuai periode sistem yang ditentukan, salah satu dari 2 prosedur lain dapat dipanggil secara rutin sesuai dengan kebutuhan, `PID_compute()` dipanggil setiap loop nya untuk menjalankan algoritma serta `PID_reset()` untuk melakukan reset terhadap algoritma PID, yang artinya nilai akumulasi error dan perubahan error yang tersimpan dari hasil komputasi sebelumnya dihapus.

Program 4.5 Pemanggilan Method pada Library PID Kendali Motor DC

```
void PID_init(){
  motor1.SetOutputLimits(-250, 250);
  motor1.SetSampleTimeUs(inputrate*1000);
  motor1.SetTunings(Kp1, Ki1, Kd1);
  motor1.SetMode(motor1.Control::automatic);
}
void PID_compute(){
  motor1.Compute();
}
void PID_reset(){
  motor1.Reset();
}
```

Keluaran dari PID adalah berupa nilai PWM, yang diteruskan ke *motor driver* untuk mengatur kecepatan motor sesuai kebutuhan. Nilai Kp, Ki, Kd ditentukan melalui eksperimen untuk mendapatkan respons sistem yang stabil dan cepat. Variasi nilai ini diuji pada kecepatan rendah, sedang, dan tinggi. Program 4.6 dan 4.7 menunjukkan potongan program dalam penggunaan nilai PWM hasil PID untuk diteruskan ke setiap motornya.

Program 4.6 Penggunaan nilai PWM hasil PID

```
void MoveRobot(){
  ...
  PID_compute();
  float pwm_1, pwm_2, pwm_3, pwm_4 = 0;
  if(calc.Vwheel[0] > 0){ pwm_1 = Output1 + min_motor_pwm;}
  else if(calc.Vwheel[0]<0){pwm_1=Output1-min_motor_pwm;}
  else{pwm_1 = Output1;}
```

```
  rangkabawah.Movement(pwm_1, pwm_2, pwm_3, pwm_4);
  ...
}
```

Nilai sinyal PWM yang dihasilkan dapat berupa angka positif ataupun negatif, hal ini sebagai penanda ke arah mana motor dc harus berputar. Dalam algoritma pengontrol gerak motor seperti pada Program 4. dibawah ini, logikanya cukup sederhana, yakni argumen berupa nilai PWM setiap motornya akan melalui sebuah kondisi percabangan yakni ketika nilainya diatas 0 maka roda akan diperintahkan untuk berputar berlawanan arah jarum jam , jika nilainya dibawah 0 akan berputar searah jarum jam serta apabila nilainya 0 maka roda tidak akan berputar.

Program 4.7 Pengatur Arah Putar dan Suplai PWM Setiap Motor

```
LowerPart::LowerPart(byte sel_1, byte pwm_1, byte sel_2, byte pwm_2, byte
sel_3, byte pwm_3, byte sel_4, byte pwm_4){
  roda_1 = Motor(sel_1, pwm_1);
  ...
}

void LowerPart::Movement(float V1, float V2, float V3, float V4){
  if(V1 > 0){
    roda_1.ccw(abs(V1));
  } else if(V1 < 0){
    roda_1.cw(abs(V1));
  } else{
    roda_1.stop();
  }

  ...
}
```

Motor DC dikendalikan arah putaran dan kecepatannya melalui pengaturan pemberian nilai pada *port* M+ dan M-. Dalam aplikasinya, rangkaian yang digunakan pada prototipe robot uji coba menggunakan demux untuk mengalirkan sinyal listrik ke *motor driver*. Cara kerjanya, yakni dengan mengatur logika pin selektor untuk memilih apakah sinyal PWM yang

diperoleh diteruskan menuju ke port M+ atau M- pada motor tergantung dari pengaturan sinyal pada pin selector pada demux, sementara port lainnya otomatis akan menerima nilai 0. Lebih jelasnya, implementasi pengendalian motor dc dapat dilihat pada Program 4.8 di bawah ini.

Program 4.8 Pengatur Arah Putar dan Suplai Motor

```
/*MOTOR MOVEMENT*/
void Motor::cw(byte speed){
  digitalWrite(pin_selector, LOW);
  analogWrite(pin_pwm, speed);
}

void Motor::ccw(byte speed){
  digitalWrite(pin_selector, HIGH);
  analogWrite(pin_pwm, speed);
}

void Motor::stop(){
  analogWrite(pin_pwm, 0);
}
```

Umpan balik dari kendali PID diperoleh dengan membaca data putaran roda melalui *rotary encoder* untuk merubahnya menjadi data kecepatan setiap rodanya. Untuk membaca sensor encoder, digunakan *library custom* `Encoder.h` sehingga hanya perlu dilakukan pembuatan objek pada program dengan argumen berupa pin-pin yang digunakan sebagai masukan sensor serta nilai *pulse per rotation* nya. Berikut adalah contoh implementasinya untuk salah satu encoder yang digunakan.

Program 4.9 Pembuatan Objek Pembacaan Encoder

```
#include "Encoder.h"

#define enc_fr_b PA8
#define enc_fr_a PB14

/* --- QUADRATURE ENCODER MOTOR --- */
const float PPR = 537.6; //PPR Encoder
Encoder_internal_state_t * Encoder::interruptArgs[];
Encoder ENCFR(enc_fr_a, enc_fr_b, PPR);
```

Karena nilai setpoin atau referensi yang digunakan adalah berupa nilai kecepatan, maka hal yang sama diterapkan pada masukan umpan baliknya, dari kelas objek yang telah dibuat, *method* untuk memanggil nilai pembacaan diperlukan. Kemudian, nilai yang ada dilakukan perhitungan sesuai dengan persamaan 4.6 dan 4.7 di atas. Secara garis besar, nilai kecepatan diperoleh dari nilai keliling roda dikalikan dengan pulsa pembacaan encoder saat ini yang dikurangkan dengan nilai dari pembacaan pada iterasi program sebelumnya dan dibagi dengan periode sistem yang digunakan serta nilai PPR yang digunakan.

Program 4.10 Perhitungan kecepatan motor

```
void loop(){
  if (millis() - input_prevmillis >= inputrate){
   ...
    MoveRobot();
    prev_fr_tics = fr_tics;
    fr_tics = ENCFR.read();
    prev_Timing = Timing;
    Timing = micros();
     Vreal1=((fr_tics-prev_fr_tics)/(Timing-prev_Timing))
     /1.0e6))/PPR*60.0;
   ...
  }
}
```

4.4.2 Kendali Posisi dan Kecepatan Tempuh Robot

Berdasarkan skema jalur gerak robot sebagaimana ditunjukkan oleh Gambar 4.1 menunjukkan bagaimana robot bergerak dalam skema permainan. Bila titik-titik pemberhentian dirunut dari awal mula lokasi pergerakan, setidaknya robot harus bisa bergerak lurus ke depan (dari *startzone*) atau serong (dari *retry zone*) untuk mencapai titik poin 1, lalu bergerak ke arah samping menuju titik poin 2, bergerak lurus kembali ke titik poin 3 dan bermanuver rotasi sembari tetap bergerak lurus menuju titik poin 4 dan berakhir pada titik poin 5 agar bisa mencapai zona 3.

Untuk itu sebuah algoritma diperlukan untuk mengontrol kecepatan robot berdasarkan letak posisi serta orientasinya saat itu, jika ia berada jauh dari tujuan maka pergerakan dibuat secepat mungkin serta melambat ketika mendekati tujuan. Lalu kemampuan robot melakukan koreksi ketika terjadi penyimpangan arah juga sangat diperlukan. Oleh karena itu, berdasarkan Gambar 4.1 dan 4.2 beberapa persamaan diformulasikan untuk mengakomodir kendali PID yang akan digunakan, yakni :

$$\Delta X = X_{t+1} - X_t \tag{4.18}$$

$$\Delta Y = Y_{t+1} - Y_t \tag{4.19}$$

$$S = \sqrt{\Delta X^2 + \Delta Y^2} \tag{4.20}$$

$$\varphi = tan^{-1}\left(\frac{\Delta Y}{\Delta X}\right) \tag{4.21}$$

Nilai X_{t+1} dan Y_{t+1} merupakan nilai koordinat tujuan sedangkan nilai X_t dan Y_t merupakan nilai koordinat awal atau saat ini. Sistem kendali posisi dan kecepatan tempuh robot dibuat pada raspberry pi 5 dan menggunakan *framework* ROS2 dengan bahasa pemrograman yang digunakan adalah python, serta menggunakan *library* `simple_pid` untuk mengimplementasikan algoritma PID yang digunakan. Jumlah PID yang digunakan dalam kendali posisi robot ini adalah berjumlah 3 yakni masing-masing untuk menghitung kecepatan linear robot berdasarkan nilai jarak euclidean, arah gerak linear robot berdasarkan nilai sudut yang terbentuk dari jarak euclidean terhadap sumbu X positif serta pengendali kecepatan angular berdasarkan nilai sudut orientasi robot. Berikut adalah potongan program implementasi sistem kendali yang telah dirancang.

Program 4.11 Blok Program Inisiasi Kendali Posisi dan Kecepatan Robot

```
from simple_pid.pid import PID
...
class TrajectoryControl(Node):
    def __init__(self):
        super().__init__("Trajectory_Controller")
        #Publisher and Subscriber
        self.command = self.create_subscription(String,
               "command_msg", self.sub_callback,10)
        self.move_pub = self.create_publisher(Twist, "cmd_vel",
               10)
        self.odom_filtered = self.create_subscription(Odometry,
               'sensor/odom_filtered',
               self.odom_callback, 10)
        self.odom = self.create_subscription(Float32MultiArray,
               'sensor/odom', self.odom_, 10)

        #Create timer loop
        self.dt = 0.005
        self.create_timer(self.dt, self.timer_callback)

        self.state_machine = False
        #Enable add the setpoint in the beginning
        self.enable_setpoint = True
        self.enable_move = True

        #Point planning
        self.path_from_start = [[[0, 630, 0], [405, 610, 0], [410,         900,
0], [410, 900, 0]],[[0, 250, 90], [0, 0, 180],              [400,   230,   180],
[700, 230, 270], [700, 80, 270],             [150, 80, 270], [0, 0, 0]],[[0, 630,
0], [-405, 610,               0], [-410, 900, 0], [-410, 900, 0]], [[-150, 85,
0],            [-405, 85, 0], [-400, 430, 0], [-39, 430, 0]]]
        self.idx = 0
        self.start_index = 0
        #Initial variable
        self.init_pose = np.array([0,0,0], np.float32)
        self.distance_to_goal = 0
        self.angle_to_goal = 0
        #Robot Dinamic Control
        self.pose_data = np.array([0,0,0], np.float32)
        self.twist_data = np.array([0,0,0], np.float32)
        self.max_acc = 0.495962/2
        self.rasio = 0.9
        self.max_a_lin = (self.max_acc*self.rasio)
        self.max_a_ang = (self.max_acc*np.sqrt(1-np.square(self.rasio)))
        self.V_real = 0.0
        self.W_real = 0.0
        #PID
        self.Heading_PID = PID(0.8, 0.0, 0.005, 0.0,
         auto_mode=True, output_limits=(-99, 99))
        self.Distance_PID = PID(2, 0, 0, 0.0, auto_mode=True,
         output_limits=(-45, 45))
        self.Angle_PID = PID(2, 0.0, 0.0, 0.0, auto_mode=True,
         output_limits=(-99, 99)) #1.05 0.4
        #PID
        self.Heading_PID = PID(0.8, 0.0, 0.005, 0.0,
         auto_mode=True,      output_limits=(-99, 99))
        self.Distance_PID = PID(2, 0, 0, 0.0, auto_mode=True,
         output_limits=(-45, 45))
        self.Angle_PID = PID(2, 0.0, 0.0, 0.0, auto_mode=True,
         output_limits=(-99, 99))
           ...
```

1. **Kendali Posisi Melalui Pengaturan Kecepatan dan Arah Gerak Linear Robot**

Algoritma kendali untuk mengatur kecepatan gerak linear robot ditentukan berdasarkan jarak tempuh yang harus dilalui oleh robot menuju ke titik tujuan. Jarak tempuh dihitung berdasarkan nilai selisih koordinat awal dan akhir robot seperti pada persamaan 4.18 hingga 4.21 di atas. Untuk nilai setpoin ditentukan dengan menghitung jarak euclidean antara titik tujuan dan titik awal mula sebelum memulai pergerakan sedangkan nilai *realtime* masukannya didapat dengan menghitung nilai selisih titik akhir dan titik saat ini.

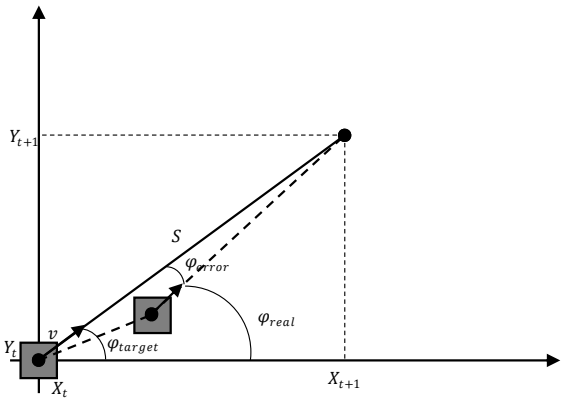
$$e_s(t) = S_{total} - S_{realtime} \tag{4.22}$$

Dengan menggunakan nilai jarak euclidean tersebut pada PID, didapat nilai keluaran berupa kecepatan linear yang harus ditempuh oleh robot. Namun, nilai kecepatan yang diperoleh masihlah kecepatan total sehingga meskipun sudah berbentuk besaran vektor, nilai positif dan negatifnya masih belum bisa ditentukan apakah melambangkan gerak ke kanan atau kiri, depan atau belakang dan yang lainnya. Maka dari itu perlu dilakukan pencarian nilai arah dari gerak robot dalam 360 derajat. Arah gerak ditentukan oleh nilai φ yang merupakan nilai invers tan dari selisih titik koordinat tujuan dan saat ini dari robot seperti pada persamaan 4.20.

Lalu mengapa tidak langsung menggunakan nilai selisih koordinat X dan Y masing-masing untuk mencari nilai kecepatan linear X dan Y pada robot ? Menurut peneliti, penggunaan metode pencarian kecepatan dengan

memisahkan parameter nilai kecepatan dan arah gerak mempunyai beberapa keuntungan. Dengan hanya menghitung satu nilai kecepatan linear total dan arah gerak, algoritma menjadi lebih intuitif. Misalnya, setpoin hanya memerlukan magnitudo kecepatan dan sudut arah, yang lebih mudah dipahami dalam kerangka koordinat global. Pendekatan ini lebih alami untuk gerakan omnidirectional, memungkinkan linear dan rotasi berjalan simultan dengan kontrol yang lebih halus. Lalu mengurangi redundansi kontrol, karena aspek linear langsung diperlakukan sebagai vektor total. Mempermudah pengendalian gerakan kombinasi linear dan rotasi yakni sistem tidak perlu memisahkan kontrol kecepatan X dan Y, cukup bekerja dengan arah dan kecepatan linear total. Misalnya, saat robot bergerak melingkar atau bergerak menuju target dengan sudut tertentu.

Pemisahan ini memungkinkan sistem untuk fokus pada arah gerak yang diinginkan, terlepas dari orientasi robot. Ini sangat berguna untuk robot dengan mekanisme holonomik seperti *X-Drive*, yang mampu bergerak ke segala arah tanpa harus mengubah orientasi. Serta, karena kecepatan ditentukan berdasarkan arah relatif terhadap tujuan, pendekatan ini lebih efektif untuk gerakan diagonal atau non-ortogonal, sehingga menghasilkan lintasan yang lebih optimal dan alami.



Gambar 4.4 Kesalahan dalam Deteksi Arah Gerakan Robot

Kontrol PID diimplementasikan untuk mengontrol pergerakan robot agar tetap berada pada perencanaan lintasan yang telah ditentukan. Gambar 4.4 menunjukkan ilustrasi kesalahan pergerakan robot *omnidirectional* terhadap lintasan. Untuk melakukan koreksi terhadap arah gerak robot menggunakan PID, maka sinyal error pada algoritma didapat melalui persamaan berikut :

$$e_{\varphi}(t) = \varphi_{target} - \varphi_{real} \tag{4.23}$$

Dari sinyal φ_{error} tersebut, kemudian menghasilkan keluaran kontroler PID untuk koreksi. Nilai tersebut digunakan untuk memperbaiki arah pergerakan robot $\varphi(n)$ agar robot bergerak mendekati garis ideal lintasan yang diinginkan, seperti yang ditunjukkan pada persamaan 4.24 dan implementasinya pada Program 4.12 di bawah ini.

$$\varphi_{new} = \varphi_{target} + \varphi_{PID} \tag{4.24}$$

Program 4.12 Blok Program Kontrol Parameter Linear

```
...
def timer_callback(self):
  new_vel = Twist()
  if self.state_machine:
    goal_x=self.path_from_start[self.start_index]    [self.idx][0]
    goal_y=self.path_from_start[self.start_index]    [self.idx][1]
    ...
    if self.enable_move:
       if self.enable_setpoint:
```

```
          #Get initial variable value
          self.init_pose[0] = self.pose_data[0]
          self.init_pose[1] = self.pose_data[1]
          ...
          self.distance_to_goal=np.sqrt(np.square(goal_x-
         self.init_pose[0])+np.square(goal_y-
         self.init_pose[1]))
          self.angle_to_goal=np.arctan2((goal_y-
         self.init_pose[1]), (goal_x-
         self.init_pose[0]))

          self.enable_setpoint = False

         distance = np.sqrt(np.square(self.pose_data[0]-
self.init_pose[0])+np.square(self.pose_data[1]-
self.init_pose[1]))
             angle=np.arctan2(round((goal_y-
         self.pose_data[1])), round((goal_x-
         self.pose_data[0])))
             Vr=abs(self.Distance_PID(distance-
         self.distance_to_goal))/100
         correct_angle=self.Angle_PID(self.angle_to_goal-
         angle)
          ...
```

2. Kendali Orientasi Melalui Pengaturan Kecepatan dan Arah Gerak Angular

Selain kecepatan dan arah gerakan robot, kontrol *heading* atau arah hadap merupakan aspek penting yang perlu dipertimbangkan dalam kontrol gerak robot *omnidirectional*. Ini karena mengubah arah robot akan bisa menyebabkannya melakukan perjalanan ke arah yang salah.

Nilai error dari heading robot dihitung menggunakan data odometrik yang berasal dari algoritma sensor fusi yang menggabungkan data pembacaan sudut dari encoder dan sensor IMU BMX160, dengan membandingkan nilai heading robot ψ yang diperoleh dengan nilai setpoin heading ψ_{target}, seperti yang ditunjukkan pada persamaan 4.25.

$$e_{\psi} = \psi_{target} + \psi_{real} \tag{4.25}$$

Nilai ini kemudian dimasukkan ke dalam Persamaan 2.1 sehingga keluaran PID dapat digunakan untuk mengoreksi arah hadapan melalui pengaturan nilai kecepatan sudut rotasi robot seperti pada Program 4.13.

Program 4.13 Blok Program Kontrol Parameter Angular

```
...
def timer_callback(self):
  new_vel = Twist()
  if self.state_machine:
    ...
    goal_z=self.path_from_start[self.start_index]    [self.idx][2]
    if self.enable_move:
       if self.enable_setpoint:
          #Get initial variable value
          ...
          self.init_pose[2] = self.pose_data[2]
          ...
          self.enable_setpoint = False
          ...
          err = (540 + goal_z - self.pose_data[2])%360-180
          w = -self.Heading_PID(err)/100
          ...
```

3. Pengaturan Percepatan Gerak dan Toleransi Posisi

Dalam pergerakan robot beroda, aspek yang dapat memperbesar nilai kesalahan lokalisasi (pengukuran posisi robot terhadap lingkungan) adalah terkait dengan slip roda akibat percepatan berlebih. Meskipun penggunaan data fusi sensor digunakan untuk meminimasi kesalahan posisi dan orientasi robot, ini belum cukup. Dari segi orientasi, tidak ada masalah karena sensor encoder dan IMU dapat saling mengoreksi. Namun untuk data bagian posisi dalam arah sumbu X dan Y, data dari kedua sensor masih belum cukup.

Sensor akselerometer pada sensor BMX160 menghasilkan nilai akselerasi yang bisa diintegralkan menjadi nilai kecepatan dan posisi, namun penggunaan integral tersebut menghasilkan nilai yang besar error nya karena pada dasarnya data akselerasi memiliki derau yang tinggi.

Dalam mengaplikasikan persamaan 4.15 dan 4.16 untuk mengurangi kemungkinan slip diperlukan nilai percepatan maksimum yang belum diketahui nilainya. Untuk memperoleh nilai percepatan maksimum

diperlukan informasi mengenai koefisien gaya gesek yang dalam hal ini berarti gesekan antara karet (roda) dan kayu (lantai) sesuai dengan spesifikasi arena perlombaan. Menurut Kurniawan (2018), menjelaskan bahwa “koefisien gesek kayu dengan karet sebesar 0,48±0,004” [15]. Nilai tersebut bisa berbeda tergantung dari kondisi lingkungan sehingga perlu dikalibrasi ulang. Namun sebagai pendekatan jika nilai 0.48 dapat digunakan dan dengan asumsi lantai berkontur datar sehingga α = 0, maka :

$$a_{max} = 0{,}48 * 9.8\, cos(0) - \ 9{,}8\, sin(0) = 0{,}4704\ m/s^2$$

Nilai percepatan di atas merupakan nilai percepatan total robot. Dalam pergerakan robot, terdapat komponen linear serta angular sehingga nilai percepatan maksimal tersebut adalah nilai resultan dari percepatan maksimum angular dan linear robot. Rasio antara percepatan linear dan angular dapat diperoleh dari perbandingan kecepatan linear dan angular maksimumnya dengan hubungan sebagai berikut.

$$a_{max}^2 = a_{linear}^2 + (r * a_{angular})^2 \tag{4.26}$$

$$rasio = \frac{v_{maks}}{r * \omega_{maks}} \tag{4.27}$$

$$a_{linear\_max} = \frac{a_{max}}{\sqrt{1 + rasio^{-2}}} \tag{4.28}$$

$$a_{angular\_max} = \frac{a_{max}}{\sqrt{1 + rasio^2}} \tag{4.29}$$

Sebagai contoh, jika robot harus bergerak dari keadaan diam ke kecepatan maksimal 1 m/s dan 0.5 rad/s dengan jari-jari rotasi 0.25 m. Maka apabila nilai percepatan maksimumnya adalah 0,4704 m/s^2,nilai percepatan linear dan angularnya dapat dihitung sebagai berikut :

$$rasio = \frac{1}{0.25 * 0.5} = 8$$

$$a_{linear} = \frac{4{,}704}{\sqrt{1+8^2}} = 0{,}46676\ m/s^2$$

$$a_{angular} = \frac{4{,}704}{\sqrt{1+8^{-2}}} = 0{,}05834\ m/s^2 = 0{,}01458\ rad/s^2$$

Satuan yang digunakan dalam hasil perhitungan di atas berada dalam rentang detik, sedangkan program sistem kendali yang dibuat mempunyai periode sistem tertentu yang digunakan, sehingga nilai tersebut perlu dikonversi menjadi nilai maksimum per periode sistem dengan mengalikan nilai percepatan yang diperoleh dengan periode sistem yang digunakan.

Robot bergerak dari satu titik ke titik lain, oleh karena itu perlu parameter yang dapat menentukan kapan robot dinyatakan berhasil mencapai titik tujuan. Dalam konteks ini, karena posisi robot bergerak ada dalam arah sumbu X dan Y, maka robot dapat menentukan bahwa ia telah mencapai target lokasi dengan membandingkan posisi saat ini terhadap target, menggunakan parameter toleransi error berikut :

$$e_{pos} = \sqrt{(X_{t+1} - X_t)^2 + (Y_{t+1} - Y_t)^2}\ , \quad e_{pos} \leq\ tolerance \tag{4.30}$$

Robot dianggap telah mencapai target posisi jika nilai errornya kurang dari sama dengan nilai toleransinya. Untuk menentukan nilai toleransi, diputuskan sebelumnya bahwa periode sistem yang digunakan adalah 5 ms dengan akurasi 0,5 cm, ini berarti bahwa dalam satu siklus program robot dapat bergerak sejauh 0,5 cm untuk kecepatan 1 m/s sehingga batas toleransi posisi yang bisa ditetapkan kepada sistem adalah minimal diangka 0,5 cm. Nilai toleransi yang terlalu kecil dapat mengakibatkan osilasi sedangkan jika terlalu besar maka robot dapat dianggap telah

mencapai tujuan meskipun posisinya jauh dari target. Untuk itu digunakan nilai faktor keamanan sebesar 2, sehingga nilai toleransi yang digunakan adalah 2 kali dari nilai akurasinya yakni 1 cm.

Untuk kasus pengaturan orientasi, tidak perlu dilakukan pencarian nilai toleransi karena koreksi sudut hadapan robot dilakukan secara *realtime* dan robot hanya perlu merubah setpoin arah tujuan hadapannya saja jika ingin berganti arah. Berikut adalah implementasi program dalam mengaplikasikan filter kecepatan yang telah dirancang serta pengaturan posisi robot.

Program 4.14 Filter Kecepatan dan Pengatur Posisi

```
def final_vel(self, V_desire, W_desire):
     a = (V_desire - self.V_real)
     alpha = (W_desire - self.W_real)
     a_lin_max = 4,704/np.sqrt(1+np.square(alpha/a)))
     a_ang_max = 4,704/np.sqrt(1+np.square(a/alpha)))
     a_final = min(abs(a), a_lin_max)
     alpha_final = min(abs(alpha), a_ang_max)
     if a>0:
         a_final = -a_final
     if alpha>0:
         alpha_final = -alpha_final
     return a_final, alpha_final
def timer_callback(self):
     ...
     acc_lin, acc_ang = self.final_vel(Vr, w)
     acc_lin, acc_ang = self.final_vel(Vr, w)
     Vel = self.V_real + acc_lin
     W_ = self.W_real + acc_ang
     new_vel.angular.z = W_
     if abs(self.distance_to_goal - distance) >= 1:
        phi = self.angle_to_goal+correct_angle
        Vx = Vel * np.cos(phi)
        Vy = Vel * np.sin(phi)
        Vcorx = -2*(np.deg2rad(W_))*Vy/78.54
        Vcory = -2*(np.deg2rad(W_))*Vx/78.54
        new_vel.linear.x = (Vx + Vcorx)
        new_vel.linear.y = (Vy + Vcory)
      else :
```

```
        if self.idx < 1:
           self.idx+=1
           self.enable_setpoint = True
        else:
           self.Distance_PID.reset()
           self.Angle_PID.reset()
           self.Heading_PID.reset()
           self.V_final = 0.0
           #new_vel.angular.z = 0.0
           new_vel.linear.x = 0.0
           new_vel.linear.y = 0.0
           self.enable_move = False
           self.get_logger().info("Goal Reached")
         …
```

4.5 Pengujian dan Analisis

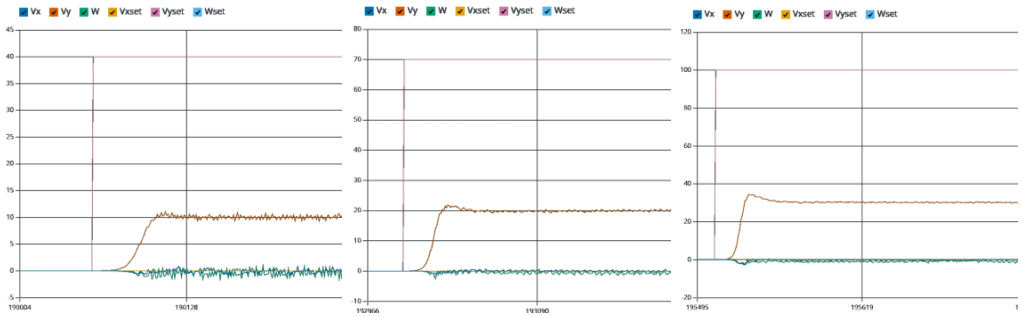
4.5.1 Uji Pengaruh Variasi Konstanta PID terhadap Kecepatan Pergerakan Robot

Pengujian ini dilakukan untuk memahami bagaimana variasi konstanta PID mempengaruhi respons sistem kendali dalam mengontrol kecepatan robot. Fokus utama dalam pengujian ini adalah menganalisis tiga parameter utama, yaitu *settling time*, *steady-state error*, dan *overshoot*, yang masing-masing mencerminkan bagaimana sistem merespons perubahan setpoin dan sejauh mana kestabilan sistem dapat dicapai dalam berbagai kondisi kecepatan.

Dalam penelitian ini, tuning PID dilakukan secara bertahap menggunakan *trial-and-error* dengan tujuan memperoleh satu set nilai konstanta PID yang paling optimal untuk digunakan dalam tiga variasi kecepatan maksimum yang telah ditentukan, yaitu 0.4 m/s, 0.7 m/s, dan 1.0 m/s. Proses tuning dimulai dengan menyesuaikan nilai konstanta P terlebih dahulu, sementara konstanta I dan D dinonaktifkan (Ki = 0), (Kd = 0). Pengujian awal ini bertujuan untuk mengetahui sejauh mana sistem dapat mencapai setpoin hanya dengan koreksi proporsional.

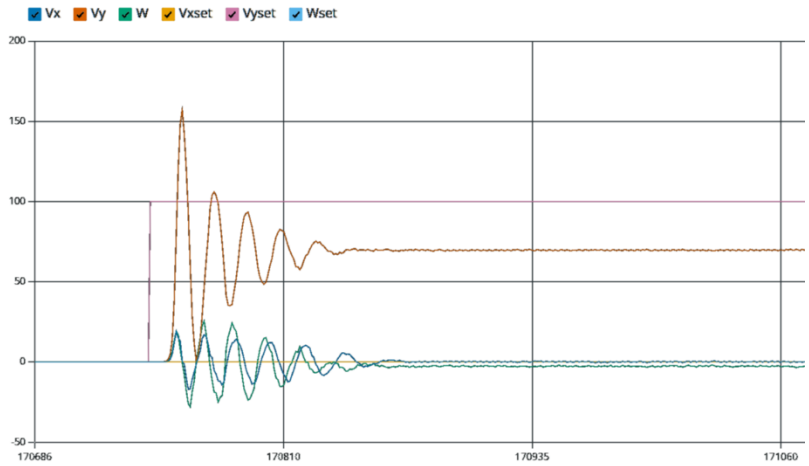
Tabel 4.1 Uji Penentuan Konstanta Proporsional P Kecepatan

Kecepatan (m/s)	**Kp**	**Ki**	**Kd**	***Settling Time* (ms)**	***Steady State Error* (m/s)**	***Overshoot* (%)**
0.4	0.4	0.0	0.0	∞	0.297	0
0.7	0.4	0.0	0.0	∞	0.50	0
1.0	0.4	0.0	0.0	∞	0.6988	0
0.4	0.7	0.0	0.0	∞	0.236	0
0.7	0.7	0.0	0.0	∞	0.4036	0
1.0	0.7	0.0	0.0	∞	0.5617	0
0.4	1.0	0.0	0.0	∞	0.1999	0
0.7	1.0	0.0	0.0	∞	0.3348	0
1.0	1.0	0.0	0.0	∞	0.4711	0



Gambar 4.5 Respon Sistem Terhadap komponen Proporsional (Kiri: 0.4 m/s; Tengah: 0.7 m/s; Kanan: 1 m/s)

Hasil uji pada Tabel 4.1 serta Gambar 4.5 menunjukkan bahwa meskipun peningkatan (Kp) mampu mempercepat respons sistem, tetapi hingga nilai tertentu sistem tetap tidak dapat mencapai setpoin karena error yang tersisa tidak dapat dikoreksi hanya dengan komponen proporsional. Hasil menunjukkan bahwa dengan P saja, sistem tidak mampu mencapai setpoin kecepatan yang ditentukan. Bahkan, sistem hanya mampu mencapai sekitar setengah dari setpoin yang diinginkan, sebelum akhirnya mencapai keadaan stabil. Saat nilai Kp terus ditingkatkan lebih dari 1.0, sistem mulai menunjukkan gejala osilasi seperti yang ditunjukkan pada Gambar 4.6 di bawah ini.



Gambar 4.6 Osilasi Sistem Pada Nilai Kp Terlalu Besar

Fenomena ini dapat dijelaskan berdasarkan teori sistem kendali. Dalam sistem PID, konstanta P bekerja dengan memberikan koreksi proporsional terhadap error yang terjadi. Artinya, semakin besar error, semakin besar sinyal koreksi yang diberikan. Namun, dalam sistem ini terdapat gaya gesek (*friction*) dan faktor inersia dari motor dan beban robot yang menyebabkan sistem mengalami damping alami. Akibatnya, meskipun terus ditingkatkan, sistem tetap tidak bisa mencapai setpoin karena tidak ada mekanisme untuk mengoreksi *steady-state error*.

Ketika nilai Kp terlalu kecil, gaya dorong yang dihasilkan tidak cukup kuat untuk melawan resistansi sistem sehingga tidak mampu mencapai setpoin. Namun, saat Kp diperbesar, koreksi menjadi lebih agresif, menyebabkan sistem berosilasi karena motor merespons terlalu cepat terhadap perubahan error. Hal ini sejalan dengan teori sistem kendali, di mana sistem kendali P saja akan selalu *memiliki steady-state erro*r dan dapat mengalami instabilitas pada nilai Kp yang terlalu tinggi.

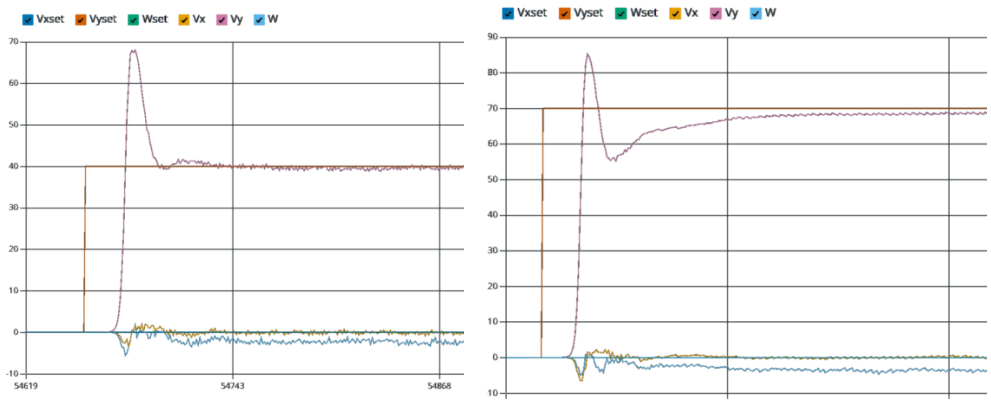
Setelah menentukan nilai (Kp) yang cukup besar untuk mendorong sistem menuju setpoin, komponen I mulai diaktifkan untuk mengurangi *steady-state error*.

Hasil pengujian pada Gambar 4.6 menunjukkan bahwa dengan mengaktifkan I, sistem akhirnya mampu mencapai setpoin, meskipun mengalami *overshoot* dan atau *undershoot* sebelum mencapai keadaan stabil. Nilai (Ki) ditingkatkan secara bertahap, dan hasil pengujian pada Tabel 4.2 menunjukkan bahwa semakin besar (Ki), semakin kecil error akhir yang terjadi. Namun, pada nilai tertentu, peningkatan (Ki) menyebabkan sistem mengalami osilasi berlebihan sebelum mencapai kestabilan. Hal ini menunjukkan bahwa penggunaannya komponen integral harus dikontrol agar tidak menyebabkan respons sistem terlalu agresif.

Tabel 4.2 Uji Penentuan Konstanta PI Kecepatan

Kecepatan (m/s)	**Kp**	**Ki**	**Kd**	***Settling Time* (ms)**	***Steady State Error* (m/s)**	***Overshoot* (%)**
0.4	0.7	5.0	0.0	275	0,005	0
0.7	0.7	5.0	0.0	358	0.0128	0
1.0	0.7	5.0	0.0	341	0.0141	0
0.4	0.7	10.0	0.0	101	0.0019	10,85
0.7	0.7	10.0	0.0	131	0.0125	21,81428571
1.0	0.7	10.0	0.0	145	0.0129	18,68
0.4	0.7	15.0	0.0	45	0.0004	70,075
0.7	0.7	15.0	0.0	66	0.0112	29,01428571
1.0	0.7	15.0	0.0	82	0.0116	28,78

Fenomena ini sesuai dengan teori integral dalam PID. Konstanta I bertugas untuk mengakumulasi error seiring waktu, sehingga semakin lama error terjadi, semakin besar koreksi yang diberikan. Hal ini menyebabkan sistem memiliki dorongan tambahan yang membuatnya mampu mencapai setpoin, tidak seperti saat hanya menggunakan P.



Gambar 4.7 Overshoot dan Undershoot pada Respon Sistem

Namun, efek samping dari penambahan I adalah munculnya *overshoot* dan atau *undershoot*, yang terjadi karena sistem menerima tambahan koreksi dari akumulasi error yang ada seperti Gambar 4.7 di atas. Semakin besar Ki, semakin cepat sistem dapat mencapai setpoin, tetapi juga semakin besar *overshoot* yang terjadi. Jika nilai Ki terlalu tinggi, sistem akan mengalami osilasi berulang sebelum mencapai kestabilan akibat efek integral yang berlebihan. Dari hasil pengujian, dapat disimpulkan bahwa meskipun I dapat menghilangkan *steady-state error*, penggunaannya harus dibatasi agar tidak menyebabkan osilasi berlebih.

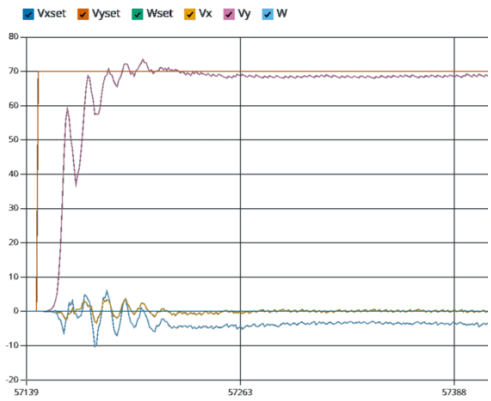
Tahap berikutnya dalam tuning PID adalah pengujian dengan mengaktifkan konstanta D, yang berfungsi untuk meredam osilasi yang masih terjadi akibat pengaruh (Ki). Berdasarkan Gambar 4.7 serta Tabel 4.3, hasil menunjukkan bahwa saat Kd diaktifkan dengan nilai yang kecil, sistem mengalami sedikit *jitter* atau derau osilatif tetapi tetap mampu mencapai kestabilan dibandingkan saat tidak diaktifkan. Pada beberapa kasus, masih terdapat *overshoot* dan atau *undershoot*, tetapi dengan amplitudo yang lebih kecil dibandingkan ketika hanya menggunakan komponen PI saja.

Tabel 4.3 Uji Penentuan Konstanta PID Kecepatan

Kecepatan (m/s)	**Kp**	**Ki**	**Kd**	***Settling Time* (ms)**	***Steady State Error* (m)**	***Overshoot* (%)**
0.4	0.7	15.0	0.005	51	0.049	14,05
0.7	0.7	15.0	0.005	36	0.0114	22,64285714
1.0	0.7	15.0	0.005	80	0.0139	6,71
0.4	0.7	15.0	0.0125	54	0.0007	4,2
0.7	0.7	15.0	0.0125	33	0.012	3,942857143
1.0	0.7	15.0	0.0125	30	0.0009	9,21
0.4	0.7	15.0	0.02	72	0.0004	5,9
0.7	0.7	15.0	0.02	65	0.0107	4,814285714
1.0	0.7	15.0	0.02	70	0.0102	2,86

Hal ini sesuai dengan fungsi D dalam PID, yaitu untuk memprediksi arah perubahan error dan memberikan koreksi dini untuk meredam osilasi. Karena komponen derivatif bekerja dengan menghitung laju perubahan error, maka respons sistem menjadi lebih halus dan stabil dibandingkan dengan hanya menggunakan PI.

Namun, ketika D ditingkatkan melebihi 0.02, terjadi osilasi pada sistem, yang menyebabkan sistem memerlukan beberapa siklus sebelum akhirnya mencapai kestabilan (lihat Gambar 4.8 di bawah ini). Jika D terus ditingkatkan lebih tinggi, sistem mulai berosilasi tanpa kendali. Hal ini disebabkan oleh fakta bahwa D sangat sensitif terhadap derau pada sensor, sehingga jika nilai Kd terlalu tinggi, sistem menjadi terlalu responsif terhadap fluktuasi kecil pada sinyal error, yang akhirnya menyebabkan osilasi terus-menerus.



Gambar 4.8 Osilasi Akibat Komponen Derivatif

Setelah ketiga konstanta PID diuji secara terpisah, dilakukan uji akhir dengan satu set konstanta PID yang telah disesuaikan untuk tiga variasi kecepatan maksimum robot. Pengujian ini bertujuan untuk memastikan bahwa sistem dapat bekerja secara optimal dalam semua kecepatan tanpa perlu melakukan tuning ulang setiap kali terjadi perubahan kecepatan. Dari hasil yang diperoleh pada Tabel 4.4, sistem dengan konstanta Kp = 0.6, Ki = 12, dan Kd = 0.008 menunjukkan respons yang paling stabil dengan *settling time* yang meskipun sedikit lebih lambat, namun mempunyai rata-rata *steady-state error* yang paling minimal, dan tidak ada *overshoot* di setiap skenario kecepatan.

Tabel 4.4 Uji Penentuan Konstanta Akhir Sistem PID Kecepatan

Kecepatan (m/s)	**Kp**	**Ki**	**Kd**	***Settling Time* (ms)**	***Steady State Error* (m)**	***Overshoot* (%)**
0.4	0.475	9.0	0.007	75	0.0034	0
0.7	0.475	9.0	0.007	115	0.0138	0
1.0	0.475	9.0	0.007	131	0.0106	0
0.4	0.6	12.0	0.008	50	0.0006	0
0.7	0.6	12.0	0.008	63	0.0107	0
1.0	0.6	12.0	0.008	71	0.0095	0
0.4	0.75	15.0	0.01	37	0.0013	2,525
0.7	0.75	15.0	0.01	34	0.0125	7,457142857
1.0	0.75	15.0	0.01	65	0.0148	1,48

Dengan skema pengujian ini, penelitian dapat membuktikan bagaimana teori PID bekerja dalam sistem kendali kecepatan motor DC pada robot *X-Drive* serta menunjukkan bagaimana nilai konstanta optimal dapat diperoleh melalui analisis berbasis parameter performa sistem. Hasil yang diperoleh juga membuktikan bahwa tuning PID yang tepat memungkinkan satu set nilai konstanta dapat digunakan secara konsisten pada berbagai kondisi kecepatan tanpa perlu dilakukan penyesuaian ulang.

4.5.2 Uji Pengaruh Variasi Kecepatan terhadap Akurasi Gerak Robot

Pada pembahasan ini, dilakukan pengujian untuk menganalisis pengaruh variasi kecepatan terhadap akurasi gerak robot *X-Drive* yang dirancang. Tujuan dari pengujian ini adalah untuk mengevaluasi kemampuan robot dalam mengikuti lintasan ideal (garis lurus) berdasarkan data posisi aktual yang direkam *secara realtime*. Akurasi gerak robot diukur dengan dua parameter utama, yaitu nilai *Root Mean Square Error* (RMSE) dan *lateral deviation* dari lintasan ideal.

RMSE digunakan untuk mengukur rata-rata kesalahan posisi antara data ideal dan data real, sementara *lateral deviation* menunjukkan sejauh mana robot menyimpang secara lateral dari lintasan ideal pada setiap segmen pergerakan. Pengujian ini relevan untuk menentukan keandalan algoritme kendali berbasis PID dalam mempertahankan akurasi pada lintasan yang beragam, termasuk lintasan lurus, serong, dan lintasan dengan rotasi.

Sebelum melakukan pengujian, terlebih dahulu peneliti menentukan nilai atau tuning konstanta untuk sistem kendali kecepatan dan arah kecepatan linear serta kecepatan angular berdasarkan posisi dan orientasi. Dari hasil tuning,

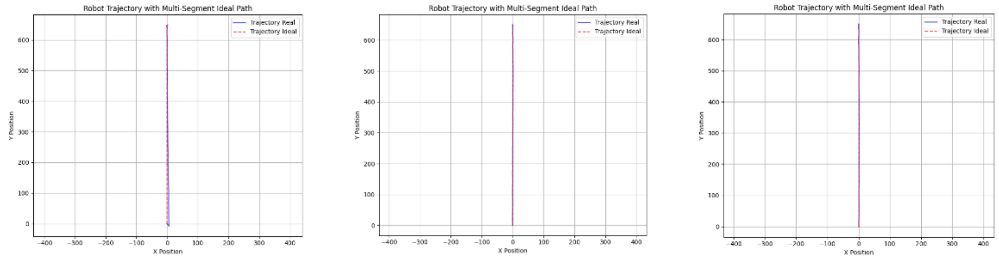
didapatkan nilai konstanta *Heading* PID (Kp=0,8;Ki=0,0;Kd=0,005), *Distance* PID (Kp=2,0;Ki=0,0;Kd=0,0) serta *Angle* PID (Kp=2,0;Ki=0,0;Kd=0,0).

Selanjutnya, pengujian dilakukan dengan menjalankan robot dari titik awal (*starting point* dalam cm dan derajat) atau (0, 0, 0°) menuju titik tujuan (*target point*) berdasarkan lima skenario lintasan berikut:

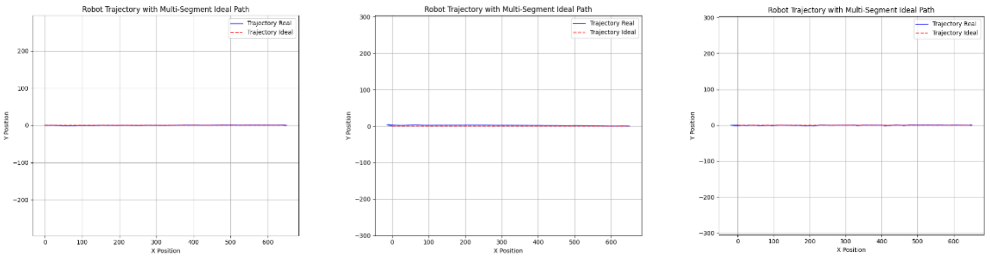
1. Lintasan Lurus ke Depan: Robot bergerak ke (0, 650, 0°) tanpa perubahan orientasi.
2. Lintasan Lurus ke Samping Kanan: Robot bergerak ke (650, 0, 0°) dengan arah lateral.
3. Lintasan Serong: Robot bergerak ke (460, 460, 0°) dengan komponen translasi pada sumbu X dan Y.
4. Lintasan Lurus dengan Rotasi: Robot bergerak ke (0, 650, 180°), melibatkan pergerakan translasi dan rotasi.
5. Lintasan Kompleks Berurutan: Robot bergerak melalui beberapa titik secara berurutan, yaitu:
    a. (0, 0, 0°)
    b. (150, 75, 0°),
    c. (390, 85, 0°),
    d. (390, 200, 0°),
    e. (390, 410, 90°),
    f. (40, 410, 90°).

Pada setiap skenario, pengujian dilakukan dengan menggunakan 3 buah variasi kecepatan yakni kecepatan tinggi (1 m/s), kecepatan sedang (0.7 m/s) serta kecepatan rendah (0.4 m/s), lalu posisi robot direkam menggunakan sensor odometri (*rotary encoder* dan IMU) dengan waktu sampling tertentu. Data posisi aktual dibandingkan dengan lintasan ideal untuk menghitung nilai RMSE dan *lateral deviation* melalui sebuah program dalam bahasa python yang selain itu juga

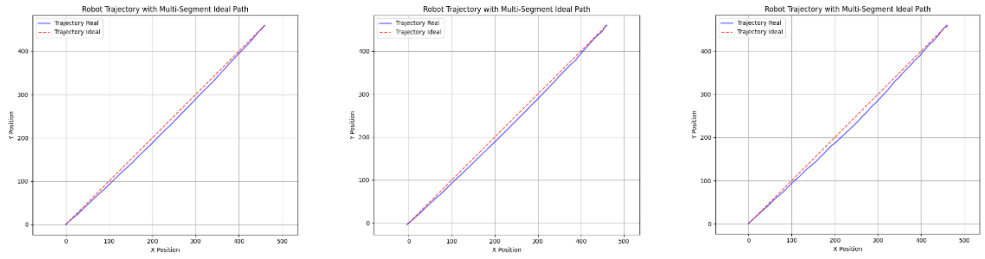
berfungsi untuk melakukan plot *trajectory* riil robot selama bergerak serta *trajectory* idealnya. Berikut adalah hasil pengujian yang telah dilakukan.



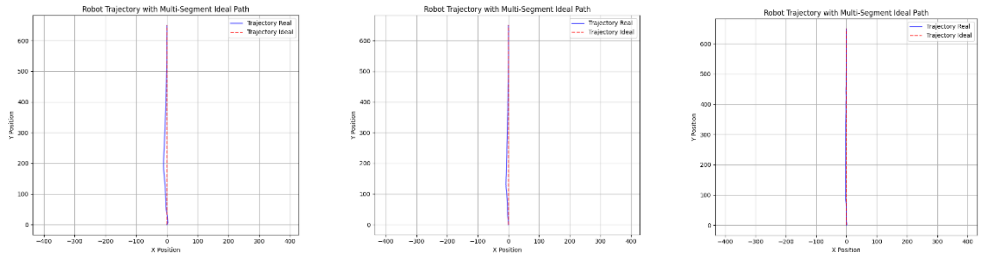
Gambar 4.9 Pengujian Gerak Lurus (kiri : 1 m/s, tengah : 0.7 m/s, kanan : 0.4 m/s)



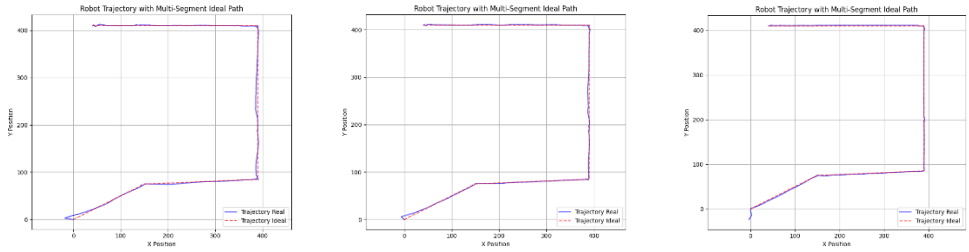
Gambar 4.10 Pengujian Gerak Menyamping (kiri : 1 m/s, tengah : 0.7 m/s, kanan : 0.4 m/s)



Gambar 4.11 Pengujian Gerak Serong (kiri : 1 m/s, tengah : 0.7 m/s, kanan : 0.4 m/s)



Gambar 4.12 Pengujian Gerak Linear dan Rotasi Simultan (kiri : 1 m/s, tengah : 0.7 m/s, kanan : 0.4 m/s)



Gambar 4.13 Pengujian Gerak Kombinasi (kiri : 1 m/s, tengah : 0.7 m/s, kanan : 0.4 m/s)

Berdasarkan Gambar 4.9 hingga 4.13 di atas. Grafik posisi hasil pengujian menunjukkan bahwa akurasi robot dipengaruhi oleh kompleksitas lintasan dan jenis gerakan yang dilakukan. Pada lintasan lurus murni ke depan, robot menunjukkan performa yang cukup baik. Hal ini mengindikasikan bahwa pergerakan translasi murni tanpa perubahan heading lebih stabil, meskipun terdapat sedikit penyimpangan pada awal lintasan akibat inisialisasi *heading*.

Sebaliknya, pada lintasan lurus ke samping kanan, alur lintasan robot tampak mengalami penyimpangan dan sedikit osilasi dibandingkan lintasan ke depan. Penyimpangan ini diduga disebabkan oleh distribusi daya motor serta beban robot antara bagian depan dan belakang yang tidak seimbang saat bergerak lateral. Akibatnya, kesalahan pada lintasan ini lebih tinggi, dengan pola penyimpangan yang menunjukkan adanya osilasi kecil pada jalur. Hal ini menunjukkan perlunya perbaikan lebih lanjut untuk memperbaiki distribusi daya motor di arah lateral dari segi sistem kendalinya ataupun perbaikan manufaktur untuk pendistribusian beban secara optimal.

Pada lintasan serong, koordinasi roda yang lebih kompleks mengakibatkan simpangan lintasan lebih tinggi dibandingkan lintasan lurus. Kesalahan terbesar tercatat pada tengah lintasan, saat robot menyesuaikan arah gerak untuk mengikuti arah diagonal. Namun, penyimpangan ini berkurang mendekati titik akhir,

menunjukkan bahwa kendali heading berangsur-angsur stabil selama pergerakan berlangsung.

Untuk lintasan lurus dengan rotasi, hasil menunjukkan bahwa kombinasi translasi dan rotasi memberikan tantangan tambahan bagi sistem kendali robot. Simpangan jalur cenderung meningkat selama fase rotasi menuju heading tertentu berjalan, dan tetap kecil selama translasi murni pasca heading berhasil dirubah. Hal ini menunjukkan bahwa algoritme kendali arah gerak maupun kecepatan memerlukan pengembangan lebih lanjut agar dapat mengurangi penyimpangan selama rotasi sehingga simpangannya seminimal mungkin tanpa mengorbankan akurasi translasi.

Lintasan kompleks dengan beberapa titik tujuan menunjukkan hasil yang paling menantang. Pada lintasan ini, *trajectory* cenderung fluktuatif tergantung pada jenis gerakan yang dilakukan saat itu serta pada setiap transisi antar titik, terutama saat robot harus mengubah arah dengan cepat. Penyimpangan lateral juga lebih besar pada lintasan panjang di antara dua titik dibandingkan lintasan pendek. Transisi antar titik menghasilkan osilasi kecil pada heading robot, yang memengaruhi akurasi gerak secara keseluruhan.

Tabel 4.5 Hasil Uji Pengaruh Kecepatan terhadap Akurasi Gerak untuk Gerakan Dasar

Jenis Gerakan	**Kecepatan**	**RMSE**	***Lateral Deviation (cm)***
Lurus ke depan (90^o)	1.0 m/s	0.3220	0.1825
Lurus ke depan (90^o)	0.7 m/s	0.2629	0.2118
Lurus ke depan (90^o)	0.4 m/s	0.2156	0.1497
Lurus ke samping (0^o)	1.0 m/s	0.3494	0.2627
Lurus ke samping (0^o)	0.7 m/s	0.3136	0.2293
Lurus ke samping (0^o)	0.4 m/s	0.2479	0.1971
Serong (45^o)	1.0 m/s	3.8741	2.3917
Serong (45^o)	0.7 m/s	4.2813	2.8836

Jenis Gerakan	Kecepatan	RMSE	*Lateral Deviation (cm)*
Serong (45^o)	0.4 m/s	5.7680	4.2010
Lurus ke depan dan berotasi	1.0 m/s	0.1801	0.1069
Lurus ke depan dan berotasi	0.7 m/s	0.2271	0.1400
Lurus ke depan dan berotasi	0.4 m/s	0.2586	0.2259

Tabel 4.6 Hasil Uji Pengaruh Kecepatan Terhadap Akurasi Gerak untuk Kombinasi Beberapa Titik Tujuan

Titik Gerakan	**Kecepatan**	**RMSE**	***Lateral Deviation (cm)***
a-b	1.0 m/s	1.2085	0.4582
b-c	1.0 m/s	1.0706	0.7671
c-d	1.0 m/s	0.4450	0.3620
d-e	1.0 m/s	0.5172	0.4527
e-f	1.0 m/s	0.6350	0.6060
a-b	0.7 m/s	1.2269	0.5271
b-c	0.7 m/s	0.5503	0.4510
c-d	0.7 m/s	0.3908	0.3487
d-e	0.7 m/s	0.3935	0.3368
e-f	0.7 m/s	0.3276	0.3007
a-b	0.4 m/s	1.1953	0.8286
b-c	0.4 m/s	0.6522	0.5848
c-d	0.4 m/s	0.2574	0.2207
d-e	0.4 m/s	0.3026	0.2698
e-f	0.4 m/s	0.2277	0.2011

Berdasarkan data matematis pada Tabel 4.5 dan 4.6 di atas, pada lintasan lurus ke depan (0, 650, 0°), hasil pengujian menunjukkan bahwa kecepatan rendah memberikan akurasi terbaik, dengan nilai RMSE sebesar 0,2156 dan *lateral deviation* sebesar 0,1497 cm. Akurasi menurun seiring bertambahnya kecepatan, yakni nilai RMSE meningkat menjadi 0,3220 pada kecepatan tinggi. Hal ini dapat dijelaskan dengan teori sistem kendali, saat pada kecepatan tinggi, pengaruh inersia robot meningkat sehingga lebih sulit bagi sistem kendali untuk mempertahankan

lintasan ideal. Selain itu, efek slip roda cenderung lebih besar pada kecepatan tinggi, yang memengaruhi akurasi pergerakan robot.

Pada lintasan lurus ke samping (650, 0, 0°), hasil pengujian menunjukkan pola yang serupa. Kecepatan rendah menghasilkan RMSE yang lebih kecil (0,2479) dibandingkan kecepatan tinggi (0,3494). Penyimpangan lateral juga lebih besar pada kecepatan tinggi, yang menunjukkan adanya tantangan dalam distribusi daya motor untuk gerakan lateral. Hal ini relevan dengan analisis kinematika robot *omniwheel*, gerakan lateral memerlukan koordinasi yang presisi antara roda, sehingga peningkatan kecepatan dapat memperbesar deviasi akibat ketidakseimbangan daya atau respon yang tidak sinkron antar aktuator. Dalam bidang Teknik Elektro dan Mekatronika, hasil ini menekankan perlunya tuning tambahan pada algoritme kendali berbasis PID untuk meningkatkan stabilitas lateral, khususnya pada kecepatan tinggi.

Lintasan serong (460, 460, 0°) menunjukkan hasil yang lebih kompleks. Nilai RMSE pada lintasan ini lebih tinggi dibandingkan lintasan lurus, dengan nilai RMSE mencapai 5,7680 pada kecepatan rendah dan menurun menjadi 3,8741 pada kecepatan tinggi. Penurunan RMSE pada kecepatan tinggi ini tampaknya kontradiktif dengan lintasan lurus, tetapi dapat dijelaskan dengan dinamika pergerakan robot. Pada lintasan serong, koordinasi roda lebih kompleks karena melibatkan komponen translasi pada dua sumbu (X dan Y). Pada kecepatan rendah, sistem kendali lebih sensitif terhadap ketidakseimbangan daya antar roda, yang menyebabkan peningkatan penyimpangan. Sebaliknya, pada kecepatan tinggi, dominasi kontrol proporsional (Kp) dalam PID membantu mengurangi

penyimpangan, meskipun tetap terdapat *lateral deviation* yang signifikan akibat efek slip. Hasil ini menunjukkan bahwa tuning PID yang digunakan cukup efektif untuk mengatasi dinamika lintasan serong, tetapi masih membutuhkan optimasi lebih lanjut untuk mengurangi deviasi lateral.

Lintasan dengan kombinasi translasi dan rotasi (0, 650, 180°) memberikan tantangan tambahan bagi sistem kendali. Data menunjukkan bahwa akurasi gerak lebih baik pada kecepatan tinggi, dengan RMSE sebesar 0,1801 dan *lateral deviation* sebesar 0,1069 cm. Sebaliknya, pada kecepatan rendah, RMSE meningkat menjadi 0,2586, sementara *lateral deviation* mencapai 0,2259 cm. Hasil ini menunjukkan bahwa rotasi robot lebih stabil pada kecepatan tinggi, yang menunjukkan tuning PID heading efektif untuk menjaga orientasi selama rotasi. Namun, peningkatan RMSE pada kecepatan rendah mengindikasikan adanya osilasi kecil pada heading robot akibat akumulasi error integral (Ki) dalam algoritme PID.

Lintasan kompleks dengan beberapa titik tujuan menunjukkan nilai RMSE dan *lateral deviation* yang jauh lebih besar dibandingkan lintasan lainnya, terutama pada titik belokan. Hasil ini sesuai dengan teori bahwa setiap perubahan arah meningkatkan kompleksitas dinamika robot, sehingga penyimpangan lebih besar cenderung terjadi. Meskipun tuning PID yang dirancang cukup efektif untuk mengontrol posisi pada lintasan sederhana, peningkatan deviasi pada lintasan kompleks menunjukkan perlunya optimasi lebih lanjut untuk mengantisipasi perubahan arah yang cepat.

Secara keseluruhan, hasil pengujian menunjukkan bahwa akurasi gerak robot sangat dipengaruhi oleh kecepatan, dengan kecepatan rendah memberikan hasil terbaik pada lintasan lurus. Namun, pada lintasan serong dengan kombinasi translasi dan rotasi, kecepatan tinggi cenderung lebih stabil.

BAB 5
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan penelitian yang telah dilakukan terkait rancang bangun dan implementasi algoritme kendali berbasis PID dan kinematika pada robot otonom *X-Drive*, beberapa kesimpulan yang dapat diambil adalah sebagai berikut:

1. Perancangan sistem kendali pergerakan robot mampu mengontrol kecepatan gerak linear maupun angular agar robot dapat mencapai target koordinat yang ditentukan dengan menerapkan algoritma PID pada sistem kendali kecepatan maupun posisi robot. Algoritma ini memastikan bahwa kecepatan setiap roda dikontrol dengan baik agar menghasilkan pergerakan yang sesuai dengan model kinematika robot *X-Drive*. Implementasi PID memungkinkan robot untuk mencapai target posisi dan orientasi yang telah ditentukan dengan respons sistem yang stabil, tanpa osilasi berlebih, dan dengan error minimum.
2. Variasi konstanta PID memiliki pengaruh signifikan terhadap respons sistem kendali kecepatan. Dari hasil pengujian tuning PID, ditemukan bahwa dengan Kp = 0.6, Ki = 12, dan Kd = 0.008, sistem mampu mencapai kecepatan yang diinginkan dengan settling time rata-rata sebanyak 63 ms, tanpa *overshoot*, dan *steady-state error* yang terendah dari 3 konstanta lain yakni (0.0069 m/s). Tanpa penggunaan konstanta integral, sistem tidak mampu mencapai setpoint kecepatan yang ditentukan, sementara penggunaan konstanta derivatif yang terlalu besar menyebabkan osilasi yang tidak terkendali. Oleh karena itu, tuning yang optimal diperlukan untuk memastikan sistem kendali bekerja dengan stabil dalam berbagai kondisi operasional.

3. Variasi kecepatan maksimal robot berpengaruh terhadap akurasi pergerakan. Pengujian menunjukkan bahwa pada kecepatan rendah (0.4 m/s), RMSE terkecil diperoleh sebesar 0.2156 sedangkan pada kecepatan tinggi (1.0 m/s), RMSE meningkat menjadi 0.3220 akibat efek inersia dan slip roda. *Lateral deviation* meningkat dari 0.1497 m pada kecepatan rendah menjadi 0.1825 m pada kecepatan tinggi. Namun, untuk lintasan kompleks seperti pergerakan serong atau kombinasi translasi dan rotasi, kecepatan tinggi justru menghasilkan kestabilan heading yang lebih baik. Dengan demikian, terdapat *trade-off* antara kecepatan, stabilitas, dan akurasi gerak, yang harus diperhitungkan dalam perencanaan navigasi robot.

Dari hasil ini, dapat disimpulkan bahwa sistem kendali berbasis PID telah berfungsi secara efektif, dengan tuning parameter yang tepat mampu membuat pergerakan robot yang stabil. Namun, kecepatan gerak memiliki dampak yang bervariasi tergantung pada jenis lintasan, yaitu kecepatan tinggi lebih cocok untuk lintasan kompleks, sedangkan kecepatan rendah lebih akurat pada lintasan lurus.

5.2 Saran

Berdasarkan hasil dan analisis penelitian, beberapa saran yang dapat diberikan untuk pengembangan lebih lanjut adalah sebagai berikut:

1. Penyempurnaan Algoritma Kendali: Untuk meningkatkan performa sistem, disarankan untuk mengembangkan metode tuning PID yang lebih optimal, seperti metode Ziegler-Nichols.
2. Melakukan analisis lebih mendalam terhadap dinamika robot dengan menganalisis keterlibatan gaya terhadap pergerakan, termasuk pengaruh

distribusi berat dan respons motor terhadap beban dinamis, guna memastikan distribusi daya yang lebih seimbang pada lintasan lateral maupun diagonal.

3. Pengujian tambahan pada berbagai permukaan dan kondisi lingkungan, seperti lantai dengan tingkat friksi yang berbeda, dapat memberikan data yang lebih lengkap terkait kekokohan sistem kendali terhadap slip roda dan gangguan eksternal.

Dengan implementasi saran-saran di atas, diharapkan pengembangan lebih lanjut dapat meningkatkan performa robot otonom *X-Drive* baik dari segi akurasi, stabilitas, maupun keandalannya dalam berbagai skenario operasi.

DAFTAR PUSTAKA

[1] Robotnik, “Latest mobile industrial robots: Trends and Uses,” Robotnik. Diakses: 10 Oktober 2024. [Daring]. Tersedia pada: https://robotnik.eu/latest-mobile-industrial-robots-trends-and-uses/

[2] VIETNAM TELEVISION (VTV), “ABU Robocon 2024 Theme & Rules (Revision on 11.09.2024),” *ABU Robocon 2024*, 2024. Diakses: 10 Oktober 2024. [Daring]. Tersedia pada: https://aburobocon2024.vtv.gov.vn/gamerules

[3] Timur Yuldashev dan Andrey Solovev, “Basics of PID Controllers: Design, Applications, Advantages & Disadvantages,” Integra Sources. Diakses: 10 Oktober 2024. [Daring]. Tersedia pada: https://www.integrasources.com/blog/basics-of-pid-controllers-design-applications/

[4] Iswanto *dkk.*, “PID-based with Odometry for Trajectory Tracking Control on Four-wheel Omnidirectional Covid-19 Aromatherapy Robot,” *Emerging Science Journal*, vol. 5, hlm. 157–181, Nov 2021, doi: 10.28991/esj-2021-SPER-13.

[5] G. Wahyu Kurniawan, N. Setyawan, E. Azizul Hakim, T. Elektro, dan U. Muhammadiyah Malang, “PID Trajectory Tracking Control 4 Omni-Wheel Robot,” *Seminar Nasional Fortei7-2 Forum Pendidikan Tinggi Teknik Elektro Indonesia Regional VII*, vol. 2, Agu 2019.

[6] A. Sofwan, H. R. Mulyana, H. Afrisal, dan A. Goni, “Development of Omni-Wheeled Mobile Robot Based-on Inverse Kinematics and Odometry,” dalam *2019 6th International Conference on Information Technology, Computer and Electrical Engineering (ICITACEE)*, IEEE, Sep 2019, hlm. 1–6. doi: 10.1109/ICITACEE.2019.8904418.

[7] Jason Walker, “What are Autonomous Robots? 8 Applications for Today’s AMRs,” Locus Robotics. Diakses: 10 Oktober 2024. [Daring]. Tersedia pada: https://locusrobotics.com/blog/what-are-autonomous-robots#:~:text=What%20does%20autonomous%20mean%20in%20the%20context%20of%20robotics%3F,without%20human%20control%20or%20intervention

[8] K. Kanjanawanishkul, “Omnidirectional wheeled mobile robots: Wheel types and practical applications,” *International Journal of Advanced Mechatronic Systems*, vol. 6, no. 6, hlm. 289–302, Feb 2015, doi: 10.1504/IJAMECHS.2015.074788.

[9] M. Ben-Ari dan F. Mondada, *Elements of Robotics*. Cham: Springer International Publishing, 2018. doi: 10.1007/978-3-319-62533-1.

[10] Thomas. Bräunl, *Embedded robotics : mobile robot design and applications with embedded systems*. Springer, 2006.
[11] Mila, “Kinematika (Pengertian, Rumus, Contoh Soal & Pembahasan),” https://mejakelas.com/kinematika/ .
[12] S. G. . Tzafestas, *Introduction to mobile robot control*. Elsevier, 2014.
[13] Drishya Manohar, “What is Dead Reckoning?,” Cavli Wireless. Diakses: 10 Oktober 2024. [Daring]. Tersedia pada: https://www.cavliwireless.com/blog/wireless-by-design/what-is-dead-reckoning.html
[14] Admin, “Inertial Measurement Unit (IMU) – An Introduction,” Advanced Navigation. Diakses: 10 Oktober 2024. [Daring]. Tersedia pada: https://www.advancednavigation.com/tech-articles/inertial-measurement-unit-imu-an-introduction/
[15] E. Supriana, S. Zulfa, M. Widyaswari, dan U. Fitriya, “Development of Friction Force Quiz (F2Q) to Support Learning Evaluation on Friction Force Material in Class X SHS,” dalam *Proceedings of The 6th Asia-Pacific Education And Science Conference, AECon 2020, 19-20 December 2020, Purwokerto, Indonesia*, EAI, 2021. doi: 10.4108/eai.19-12-2020.2309117.

BIODATA PENULIS

A. Identitas

Nama	:	Rafif Susena
NIM	:	H1A021025
Tempat, tanggal lahir	:	Banyumas, 21 Februari 2003
Alamat	:	Jipang,RT 005/ RW 004, Karang Lewas, Banyumas
No. Telp.	:	085163021636
Alamat e-mail	:	rafif.susena@mhs.unsoed.ac.id

B. Riwayat Pendidikan Akademik

Periode	Jenjang	Institusi
2021 – sekarang	S1	Teknik Elektro Universitas Jenderal Soedirman
2018 – 2021	SMA	SMAN 2 Purwokerto
2015 – 2018	SMP	SMPN 4 Purwokerto

C. Riwayat Pendidikan Non Formal (jika ada)

Tahun	Keahlian	Penyelenggara	Kota

D. Prestasi

Tahun	Tingkat	Prestasi