

**NAVIGASI DAN LOKALISASI AUTONOMOUS SWERVE-DRIVE ROBOT
BERDASARKAN ODOMETRY, KOMBINASI GYROSCOPE-ACCELEROMETER,
LIDAR, DAN KAMERA**

Laporan Ekuivalensi Medali Emas Kejuaraan PUSPRESNAS Pada Tugas Akhir



Ditulis untuk Memenuhi Sebagian Persyaratan Guna Memperoleh Gelar

Sarjana Sains

Program Studi Fisika

Oleh:

ERLANGGA SATRYA AS SYAHRASTANI CAHYANA

NIM 21306141022

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS NEGERI YOGYAKARTA

2024

**NAVIGASI DAN LOKALISASI AUTONOMOUS SWERVE-DRIVE ROBOT
BERDASARKAN ODOMETRY, KOMBINASI GYROSCOPE-ACCELEROMETER,
LIDAR, DAN KAMERA**

Laporan Ekuivalensi Medali Emas Kejuaraan PUSPRESNAS Pada Tugas Akhir



Ditulis untuk Memenuhi Sebagian Persyaratan Guna Memperoleh Gelar

Sarjana Sains

Program Studi Fisika

Oleh:

ERLANGGA SATRYA AS SYAHRASTANI CAHYANA

NIM 21306141022

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS NEGERI YOGYAKARTA

2024

ABSTRAK

Penelitian ini bertujuan untuk mengembangkan sistem lokalisasi dan navigasi otonom pada robot swerve-drive yang digunakan dalam kompetisi ABU Robocon 2024. Tujuan utama adalah merancang sistem yang andal dan mampu menyesuaikan pergerakan robot secara dinamis berdasarkan perubahan lingkungan secara real-time. Penelitian ini berfokus pada integrasi berbagai sensor—odometri, gyroscope-accelerometer, LiDAR, dan kamera—with algoritma kontrol adaptif untuk memastikan robot dapat mendekripsi objek dengan akurat dan menyelesaikan misi secara efisien.

Penelitian ini menggunakan pendekatan eksperimental dan pengembangan, dengan pengujian algoritma dan sensor di lingkungan yang mensimulasikan lapangan kompetisi. Subjek penelitian adalah robot swerve-drive beserta komponen sensoriknya. Data dikumpulkan melalui sistem sensor robot, dan fusi multi-sensor diterapkan untuk mencapai lokalisasi yang akurat. Validitas dan reliabilitas instrumen diuji secara empiris melalui kalibrasi. Kontrol PID dan algoritma Pure Pursuit digunakan untuk mengatur kecepatan dan lintasan robot, sementara deteksi objek secara real-time menggunakan bounding box dari input kamera. Sistem yang dirancang dianalisis dengan fokus pada akurasi, kecepatan, dan stabilitas.

Hasil penelitian menunjukkan bahwa integrasi sensor dan algoritma kontrol berhasil meningkatkan kemampuan robot dalam beradaptasi dengan lingkungan dinamis. Sistem ini mampu mendekripsi dan melacak objek seperti bola dan silo dengan akurasi tinggi dan minim drift. Keberhasilan sistem yang dirancang pada robot ini dibuktikan dengan kemenangan juara pertama di kompetisi ABU Robocon 2024 Indonesia. Penelitian selanjutnya disarankan untuk mengembangkan algoritma berbasis AI dan memperluas pengujian di lingkungan yang lebih kompleks.

Kata Kunci: fusi multi-sensor, lokalisasi, navigasi, robot, swerve-drive.

ABSTRACT

This study aims to develop an autonomous localization and navigation system for a swerve-drive robot used in the ABU Robocon 2024 competition. The objective is to design a reliable system capable of dynamically adjusting the robot's movement based on real-time environmental changes. The research focuses on integrating multiple sensors—odometry, gyroscope-accelerometer, LiDAR, and camera—with adaptive control algorithms, ensuring the robot can accurately detect objects and execute missions efficiently.

The research was conducted using an experimental and developmental approach, where various algorithms and sensors were tested in environments simulating the competition field. The subjects of the research were the swerve-drive robot and the sensory components installed on it. Data were collected using the robot's sensory system, and multi-sensor fusion was employed to achieve robust localization. Validity and reliability of the instruments were ensured through empirical testing and calibration. PID controllers and Pure Pursuit algorithms were applied to manage speed and trajectory, while real-time object detection was handled by bounding box methods using camera. Designed system were analyzed by focusing on accuracy, speed, and stability.

The results show that the integration of sensors and control algorithms successfully improved the robot's adaptability to dynamic environments. The system efficiently detected and tracked objects like balls and silos, demonstrating precise navigation with minimal drift. The robot's performance, validated by its first-place win at ABU Robocon 2024 Indonesia, proves that the designed system is both effective and reliable. Future research is suggested to explore more advanced algorithms, such as AI-based control, and expand testing to more complex environments.

Keywords: Localization, multi-sensor fusion, navigation, robot, swerve-drive.

LEMBAR PENGESAHAN

NAVIGASI DAN LOKALISASI AUTONOMOUS SWERVE-DRIVE ROBOT BERDASARKAN ODOMETRY, KOMBINASI GYROSCOPE-ACCELEROMETER, LIDAR, DAN KAMERA

TUGAS AKHIR SKRIPSI



Telah disetujui untuk Ekuivalensi Medali Emas Kejuaraan PUSPRESNAS Pada Tugas Akhir
Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Negeri Yogyakarta

Tanggal: 21 Oktober 2024

Dosen Pembimbing

Dr. Supardi, M.Si

NIP 197110151998021001

Dosen Penilai

Dr. Pujiyanto, M.Pd

NIP 197703232002121002

DAFTAR ISI

ABSTRAK	ii
ABSTRACT	iii
LEMBAR PENGESAHAN	iv
DAFTAR ISI.....	v
DAFTAR GAMBAR	vii
DAFTAR TABEL	ix
BAB I PENDAHULUAN.....	1
A. Latar Belakang	1
B. Identifikasi Masalah.....	2
C. Batasan Masalah	2
D. Rumusan Masalah.....	2
E. Tujuan Penelitian	3
F. Manfaat Penelitian.....	3
BAB II KAJIAN PUSTAKA.....	4
A. Kajian Teori.....	4
1. Lokalisasi	4
2. Navigasi	4
3. Three-Wheel Odometry	5
4. Kinematika Swerve Drive.....	8
5. Inertial Measurement Unit (IMU).....	13
6. Giroskop.....	15
7. Akselerometer	16
8. Filter Mahony	18
9. LiDAR	18
10. PID Controllers	20
11. Algoritma A-Star	22
12. Pure Pursuit Controller	23
13. YOLOv8 dan Evaluasi Model	25
B. Hasil Penelitian yang Relevan	27

C. Kerangka Pikir	28
D. Pertanyaan Penelitian dan Hipotesis	29
BAB III METODE PENELITIAN	31
A. Jenis atau Desain Penelitian	31
B. Tempat dan Waktu Penelitian.....	31
C. Sumber Data atau Subjek Penelitian	31
D. Definisi Operasional Variabel	31
E. Teknik dan Instrumenasi Pengumpulan Data	32
1. Desain Robot	33
2. Sistem Sensor dan Aktuator pada Robot	36
3. Integrasi Gyroscope dan Accelerometer Menggunakan Mahony Filter	37
4. Kalibrasi Gyroscope	40
5. Kalibrasi Accelerometer.....	41
6. Odometry	43
7. Integrasi Sudut Yaw dari IMU (Gyroscope-Accelerometer) dan Odometry.....	49
8. Kecepatan Robot Berdasarkan Odometri	49
9. Kinematika Swerve Drive.....	51
10. Path Planning Menggunakan Algoritma A*	55
11. Path Tracking Menggunakan Pure Pursuit Controllers	60
13. Deteksi Objek dengan YOLOv8.....	68
14. Graphical User Interface (GUI)	81
F. Validitas dan Reliabilitas Instrumen	83
G. Teknik Analisis Data	84
BAB IV HASIL PENELITIAN DAN PEMBAHASAN	85
BAB V SIMPULAN DAN SARAN	110
A. Simpulan	110
B. Implikasi	111
C. Saran	111
DAFTAR PUSTAKA.....	112
LAMPIRAN	113

DAFTAR GAMBAR

Gambar 1. Lokalisasi Robot Mobile	4
Gambar 2. Three-Wheel Odometry (17508 Rising Tau's 2019/20 Skystone Bot)	5
Gambar 3. (a) Representasi Umum dari Tiga Roda Omnidirectional dan (b) Parameter dari Satu Roda Omnidirectional, (XR, YR) mewakili kerangka robot mobile dengan XR adalah bagian depan robot.	7
Gambar 4. Gaya yang Dihasilkan Swerve Drive	9
Gambar 5. Gerakan Dasar Pada Swerve Drive	9
Gambar 6. Ilustrasi Tiga Komponen Utama IMU: (a) Akselerometer, (b) Magnetometer, (c) Giroskop	13
Gambar 7. Diagram Blok dari Kombinasi Ketiga Sensor IMU	14
Gambar 8. Ilustrasi Sumbu Utama Pada Sistem Koordinat Kartesius.....	15
Gambar 9. Ilustrasi Efek Coriolis Pada Giroskop	16
Gambar 10. Skema Akselerometer Kapasitif	17
Gambar 11. Skema dari Prinsip Kerja TOF	19
Gambar 12. Diagram Skema dari Ukuran Titik Cahaya.....	20
Gambar 13. Diagram Blok P Controller	21
Gambar 14. Diagram Blok PD Controller	21
Gambar 15. Diagram Blok PI Controller.....	22
Gambar 16. Diagram Blok PID Controller	22
Gambar 17. Kerangka Koordinat Referensi	24
Gambar 18. Look Ahead Distance	24
Gambar 19. Pengaruh Look Ahead Distance (a) Kecil dan (b) Besar	25
Gambar 20. Arsiteksur YOLOv8	26
Gambar 21. Desain Robot.....	34
Gambar 22. Desain Mekanisme (a) Intake Bola, (b) Gripper Bola, (c) Swerve Drive	36
Gambar 23. Diagram Blok Sistem Sensor dan Aktuator Robot	36
Gambar 24. Desain Odometri Tiga Roda Pada Base Robot	44
Gambar 25. Desain Odometri Tiga Roda Pada Base Robot (Robot Tampak Bawah).....	45
Gambar 26. Desain Dead Wheel Untuk Odometri.....	45
Gambar 27. Lapangan Permainan Abu Robocon 2024	55
Gambar 28. Desain Lapangan Menggunakan CorelDRAW untuk Pencarian Jalur A*....	56
Gambar 29. Diagram Alir Algoritma Pencarian Jalur A*	58
Gambar 30. Hasil Pencarian Jalur A*.....	60
Gambar 31. Diagram Alir Algoritma Pure Pursuit Controller	65
Gambar 32. Desain Peletakan Webcam Pada Robot	68
Gambar 33. Pembagian Dataset Bola	69
Gambar 34. Pembagian Dataset Silo	70
Gambar 35. Hasil Evaluasi Model Bola: (a) Confusion Matrix, (b) Kurva Precision-Recall, (c) Loss Function.....	73

Gambar 36. Hasil Evaluasi Model Silo: (a) Confusion Matrix, (b) Kurva Precision-Recall, (c) Loss Function.....	75
Gambar 37. (a), (b) Hasil Deteksi Bola	79
Gambar 38. (a), (b) Hasil Deteksi Silo.....	80
Gambar 39. Grapichal User Interface Robot Dengan Sensor Window Terbuka	81
Gambar 40. Graphical User Interface Robot Dengan Stick Window Terbuka	81
Gambar 41. Graphical User Interface Pengendali Mekanisme Robot.....	82
Gambar 42. Posisi Start Robot Pada Final Abu Robocon 2024 Indonesia	91
Gambar 43. Robot Melaksanakan Misi Menuju Lantai 3	95
Gambar 44. Aksi-Aksi Robot Dalam Pencarian Bola dan Pemilihan Silo Secara Otomatis	106

DAFTAR TABEL

Tabel 1. Parameter Karakteristik Utama dari TFmini..... 19

BAB I

PENDAHULUAN

A. Latar Belakang

Dalam kompetisi robotika internasional ABU Robocon 2024, tantangan utama yang dihadapi peserta adalah bagaimana menciptakan robot yang mampu beroperasi secara mandiri di lingkungan yang dinamis dan tidak pasti. Robot tidak hanya harus bergerak dengan akurat, tetapi juga harus mampu menavigasi lintasan yang kompleks dan berinteraksi dengan objek di lapangan, seperti bola dan silo, tanpa intervensi manusia. Di sinilah pentingnya penelitian terkait navigasi dan lokalisasi robot swerve-drive berbasis otonom.

Salah satu permasalahan yang sering dihadapi adalah keterbatasan kemampuan navigasi robot yang hanya mengandalkan satu jenis sensor. Misalnya, penggunaan odometri secara eksklusif dapat menyebabkan kesalahan akumulatif (drift) dalam estimasi posisi robot, terutama jika ada gesekan roda yang tidak merata atau medan yang tidak rata. Selain itu, sensor-sensor lain seperti gyroscope dan accelerometer dapat memberikan informasi yang lebih akurat tentang orientasi dan percepatan robot, tetapi tidak cukup untuk memberikan gambaran yang komprehensif tentang posisi robot di ruang tiga dimensi.

Teknologi seperti LiDAR dan kamera telah terbukti mampu memberikan solusi yang lebih baik untuk deteksi lingkungan dan objek di sekitarnya, namun, kedua teknologi ini juga memiliki keterbatasan masing-masing. Misalnya, LiDAR memiliki jangkauan terbatas dan dapat dipengaruhi oleh kondisi pencahayaan dan penghalang fisik, sedangkan kamera memerlukan pemrosesan gambar yang cepat dan akurat untuk memastikan pengenalan objek yang tepat. Oleh karena itu, kombinasi dari beberapa sensor seperti odometry, gyroscope-accelerometer, LiDAR, dan kamera menjadi sangat penting untuk mencapai navigasi dan lokalisasi yang presisi.

Realitas di lapangan menunjukkan bahwa tim-tim yang hanya mengandalkan satu teknologi sensor sering kali mengalami kegagalan dalam mencapai performa optimal, terutama dalam hal kecepatan, akurasi, dan stabilitas navigasi. Hal ini menimbulkan kesenjangan antara harapan untuk menghasilkan robot otonom yang efisien dan kenyataan bahwa banyak tim masih bergantung pada teknologi tunggal. Dengan demikian, penelitian ini penting untuk mengeksplorasi bagaimana kombinasi dari berbagai sensor dapat digunakan secara efektif dalam menciptakan robot swerve-drive otonom yang mampu menavigasi dan melokalisasi dirinya di lapangan kompetisi dengan lebih baik.

Melalui penelitian ini, diharapkan dapat ditemukan metode yang lebih optimal untuk memanfaatkan sinergi antara odometry, gyroscope-accelerometer, LiDAR, dan kamera dalam navigasi dan lokalisasi robot. Hal ini akan membuka peluang untuk meningkatkan performa robot dalam kompetisi robotika, khususnya dalam aspek keakuratan navigasi, ketepatan deteksi objek, dan efisiensi waktu.

B. Identifikasi Masalah

Berdasarkan latar belakang yang telah dijelaskan, terdapat beberapa masalah yang dapat diidentifikasi terkait dengan tantangan navigasi dan lokalisasi pada robot swerve-drive otonom untuk kompetisi ABU Robocon 2024.

1. Ketidakakuratan odometri akibat kesalahan akumulatif (*drift*).
2. Kesalahan bias, skala, dan offset dalam pembacaan sensor giroskop dan akselerometer.
3. Tantangan dalam penyesuaian dinamis lingkungan robot.
4. Kendala pemrosesan data multi-sensor secara tepat dan efisien.

C. Batasan Masalah

Berdasarkan identifikasi masalah di atas, peneliti membuat batasan masalah agar ruang lingkup dari penelitian menjadi jelas, maka untuk penelitian ini hanya akan berfokus pada pengembangan lokalisasi dan navigasi pada robot otomatis berbasis swerve-drive.

D. Rumusan Masalah

Berdasarkan identifikasi dan batasan masalah di atas, maka dapat dirumuskan masalah sebagai berikut:

1. Bagaimana cara meminimalkan kesalahan akumulatif (*drift*) pada sistem odometri robot swerve-drive agar dapat meningkatkan akurasi lokalisasi?
2. Bagaimana proses kalibrasi sensor giroskop dan akselerometer untuk meminimalkan kesalahan pembacaan?
3. Bagaimana merancang sistem lokalisasi dan navigasi robot yang mampu menyesuaikan pergerakan secara dinamis terhadap perubahan lingkungan di lapangan?
4. Bagaimana mengintegrasikan data multi-sensor secara tepat dan efisien untuk navigasi otonom robot?

E. Tujuan Penelitian

Berdasarkan permasalahan yang telah dirumuskan, penelitian ini bertujuan untuk:

1. Merancang sistem odometri yang akurat untuk proses lokalisasi robot.
2. Merancang sistem kalibrasi sensor giroskop dan akselerometer untuk meminimalkan kesalahan bias, skala, dan offset dalam pembacaan data.
3. Merancang sistem lokalisasi dan navigasi robot yang mampu menyesuaikan pergerakan secara dinamis terhadap perubahan lingkungan di lapangan?
4. Merancang sistem integrasi multi-sensor yang tepat dan efisien untuk navigasi otonom robot.

F. Manfaat Penelitian

Penelitian ini diharapkan dapat memberikan manfaat bagi beberapa pihak diantaranya:

1. Bagi Mahasiswa

Hasil penelitian ini dapat menghasilkan pemahaman lebih dalam tentang bagaimana sistem navigasi dan lokalisasi robot otomatis berbasis swerve-drive.

2. Bagi Universitas

Hasil penelitian ini dapat dijadikan sebagai sumber daya pendidikan bagi mahasiswa dan pendidik yang ingin memperluas keahlian mereka dalam dunia robotika.

3. Bagi Masyarakat

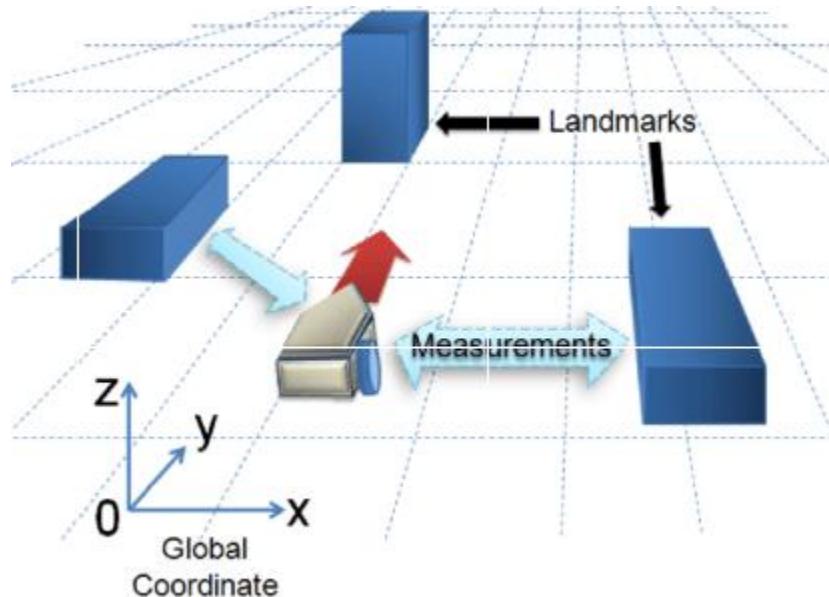
Masyarakat dapat memanfaatkan sistem navigasi dan lokalisasi yang dihasilkan untuk bidang pekerjaan yang berkaitan dengan otomasi dan robotika.

BAB II

KAJIAN PUSTAKA

A. Kajian Teori

1. Lokalisasi



Gambar 1. Lokalisasi Robot Mobile

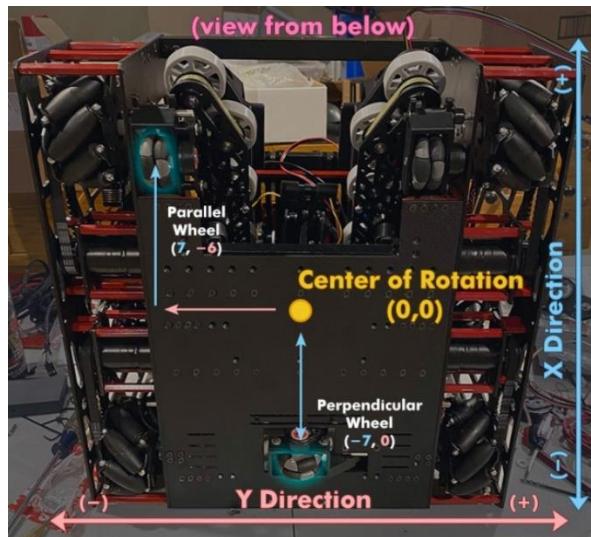
Lokalisasi adalah sarana untuk dapat menemukan posisi robot di beberapa titik waktu. Lokalisasi sangat penting dalam motion planning dan mode otonom lanjutan karena robot perlu mengetahui di mana mereka berada untuk menghasilkan gerakan yang diperlukan untuk mencapai tujuan yang diinginkan. Lokalisasi memungkinkan robot menentukan posisi dan orientasinya dalam suatu lingkungan. Proses ini sangat penting untuk navigasi otonom, memungkinkan robot untuk berinteraksi secara efektif dengan lingkungan mereka. Berbagai teknik dan teknologi digunakan untuk mencapai lokalisasi yang akurat, seperti odometri, Simultaneous Localization and Mapping (SLAM), Global Positioning System (GPS), lokalisasi visual menggunakan kamera, dan *sensor fusion*.

2. Navigasi

Navigasi robot adalah aspek mendasar dari robotika yang memungkinkan robot bergerak secara efisien dan efektif di lingkungan mereka. Navigasi dapat dikategorikan ke dalam navigasi global dan lokal. Navigasi global bergantung pada peta yang sudah ada sebelumnya untuk menentukan posisi robot dan merencanakan jalur ke tujuan,

seringkali menggunakan teknik seperti Simultaneous Localization and Mapping (SLAM), yang memungkinkan robot membangun peta sambil melacak lokasinya. Sebaliknya, navigasi lokal berfokus pada lingkungan sekitar robot, menggunakan sensor untuk mendeteksi dan menghindari rintangan secara real-time. Sensor memainkan peran penting dalam navigasi, dengan teknologi seperti Lidar untuk pemetaan terperinci, LiDAR untuk pengukuran jarak, kamera untuk pengenalan visual, dan Inertial Measurement Unit (IMU) untuk memantau orientasi dan gerakan. Algoritma perencanaan jalur, seperti algoritma A* dan Dijkstra, digunakan untuk menghitung rute terpendek dari titik awal ke tujuan. Strategi kontrol, termasuk umpan balik dan kontrol feedforward, membantu menyesuaikan tindakan robot berdasarkan data sensor atau model prediktif.

3. Three-Wheel Odometry



Gambar 2. Three-Wheel Odometry (17508 Rising Tau's 2019/20 Skystone Bot)

Odometri adalah metode sederhana dan praktis yang memberikan estimasi perpindahan relatif robot bergerak secara berkala dan real-time berdasarkan pengukuran kecepatan putar sudut rodanya. Estimasi odometri berlaku dalam kasus kondisi roda tanpa selip di mana putaran roda dapat diubah menjadi perpindahan linier relatif terhadap lantai.

Odometri mengacu pada penggunaan sensor gerak untuk lokalisasi. Bentuk odometri yang paling sederhana adalah lokalisasi encoder drive, yaitu penggunaan encoder yang mengukur rotasi motor yang menggerakkan drive train. Data pembacaan

encoder dimasukkan melalui persamaan kinematik drive train tertentu untuk mendapatkan kecepatan robot.

Tiga roda yang digunakan adalah omni wheel yang dipasang pada sensor encoder atau biasa disebut dead wheels. Desain roda tiga memanfaatkan dua pod paralel dan satu pod tegak lurus, yang masing-masing mengukur gerakan x dan y robot. Desain ini mengukur arah atau orientasi robot melalui perbedaan kedua roda paralel.

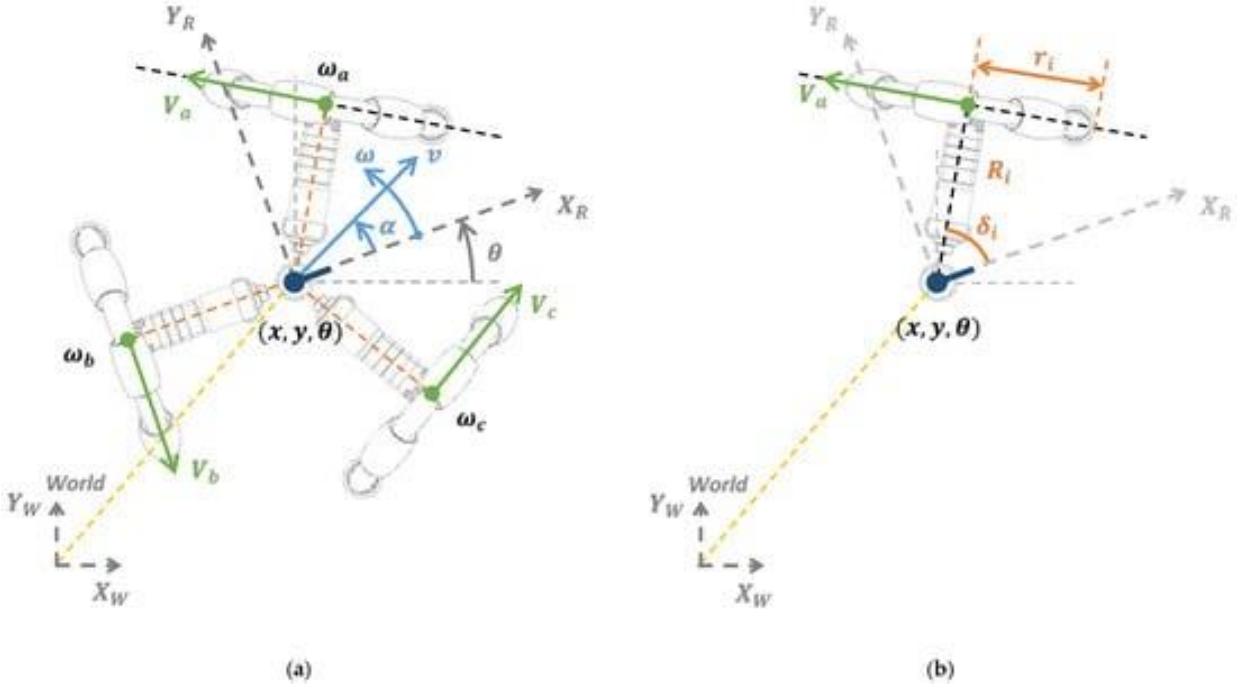
Encoder yang digunakan merupakan incremental encoders yang memberikan perubahan posisi relatif dengan menghitung jumlah rotasi atau penambahan. Saat robot bergerak, enkoder menghasilkan pulsa yang sesuai dengan putaran roda. Setiap pulsa mewakili jarak tempuh tertentu, yang ditentukan oleh keliling roda dan jumlah pulsa per putaran. Jarak yang ditempuh oleh setiap roda dapat dihitung menggunakan rumus:

$$\text{Jarak} = \text{Jumlah Pulsa} \times \text{Jarak Pulsa}$$

Jarak Pulsa diperoleh dari keliling roda dibagi dengan jumlah pulsa per putaran. Perubahan posisi yang bertahap diintegrasikan dari waktu ke waktu untuk memperbarui perkiraan posisi dan orientasi robot dalam sistem koordinat.

Digunakan perhitungan untuk merubah jumlah pulsa yang dihasilkan oleh sensor rotary encoder menjadi satuan milimeter.

Parameter utamanya adalah: posisi robot (x, y, θ) mengacu pada *fixed world frame* (kerangka lingkungan tetap) (X_W, Y_W) , perintah gerak robot (v, α, ω) , kecepatan sudut roda $(\omega_a, \omega_b, \omega_c)$, kecepatan linier roda (V_a, V_b, V_c) , jari-jari roda (r_a, r_b, r_c) , jarak antara pusat robot dan setiap roda (R_a, R_b, R_c) dan orientasi sudut masing-masing roda $(\delta_a, \delta_b, \delta_c)$ mengacu pada kerangka robot (X_R, Y_R) .



Gambar 3. (a) Representasi Umum dari Tiga Roda Omnidirectional dan (b) Parameter dari Satu Roda Omnidirectional, (X_R, Y_R) mewakili kerangka robot mobile dengan X_R adalah bagian depan robot.

Prosedur odometri menggunakan perkiraan sesaat dari kecepatan sudut saat ini dari tiga roda, a, b, c tersedia sebagai deret vektor $(\omega_a(k), \omega_b(k), \omega_c(k))$ untuk memperkirakan posisi sesaat robot bergerak $(x(k), y(k), \theta(k))$ dalam kerangka dunia (X_W, Y_W) . Hubungan antara estimasi sesaat dari kecepatan sudut roda $(\omega_a(k), \omega_b(k), \omega_c(k))$ dan kecepatan robot sesaat $(v_x(k), v_y(k), \omega(k))$ dalam kerangka dunia (X_W, Y_W) dapat diringkas sebagai:

$$\begin{bmatrix} \omega_a(k) \\ \omega_b(k) \\ \omega_c(k) \end{bmatrix} = \begin{bmatrix} 1/r_a & 0 & 0 \\ 0 & 1/r_b & 0 \\ 0 & 0 & 1/r_c \end{bmatrix} \cdot \begin{bmatrix} -\sin(\delta_a) & \cos(\delta_a) & R_a \\ -\sin(\delta_b) & \cos(\delta_b) & R_b \\ -\sin(\delta_c) & \cos(\delta_c) & R_c \end{bmatrix} \cdot \begin{bmatrix} \cos(\theta(k-1)) & \sin(\theta(k-1)) & 0 \\ -\sin(\theta(k-1)) & \cos(\theta(k-1)) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v_X(k) \\ v_Y(k) \\ \omega(k) \end{bmatrix}_{World} \quad (1)$$

Yang mencakup matriks rotasi $R(\theta)$ dan matriks kinematik kompak M yang mendefinisikan keseluruhan kinematika robot:

$$R(\theta) = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2)$$

$$M = \begin{bmatrix} -\sin(\delta_a)/r_a & \cos(\delta_a)/r_a & R_a/r_a \\ -\sin(\delta_b)/r_b & \cos(\delta_b)/r_b & R_b/r_b \\ -\sin(\delta_c)/r_c & \cos(\delta_c)/r_c & R_c/r_c \end{bmatrix} \quad (3)$$

Persamaan (1) dapat diatur untuk memperbarui posisi robot saat ini $(x(k), y(k), \theta(k))$ dalam kerangka dunia (X_W, Y_W) berdasarkan perkiraan baru kecepatan sudut saat ini dari tiga roda robot $(\omega_a(k), \omega_b(k), \omega_c(k))$ dan selang waktu Δt antara sampel $k - 1$ dan k , yang dalam robot bertepatan dengan waktunya pengambilan sampel T dari pengontrol motor proporsional-integral-turunan (PID), $\Delta t = T = 10 \text{ ms}$:

$$\begin{bmatrix} x(k) \\ y(k) \\ \theta(k) \end{bmatrix}_{World} = \begin{bmatrix} x(k-1) \\ y(k-1) \\ \theta(k-1) \end{bmatrix}_{World} + T \cdot R(\theta(k-1))^{-1} \cdot M^{-1} \cdot \begin{bmatrix} \omega_a(k) \\ \omega_b(k) \\ \omega_c(k) \end{bmatrix} \quad (4)$$

di mana $R^{-1}(\theta)$ adalah invers dari matriks rotasi yang didefinisikan oleh orientasi sudut sesaat sebelumnya oleh robot $\theta(k-1)$:

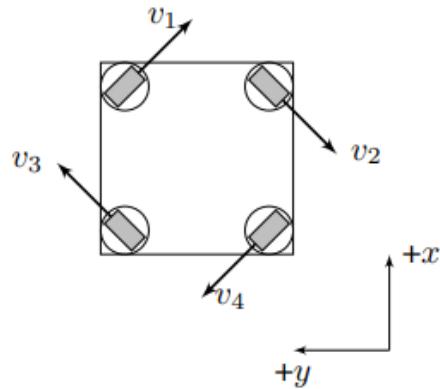
$$R(\theta)^{-1} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (5)$$

dan M^{-1} adalah invers dari matriks kinematic kompak M :

$$M^{-1} = \frac{1}{R_a \sin(\delta_b - \delta_c) - R_b \sin(\delta_a - \delta_c) + R_c \sin(\delta_a - \delta_b)} \cdot \begin{bmatrix} r_a (R_b \cos(\delta_c) - R_c \cos(\delta_b)) & -r_b (R_a \cos(\delta_c) - R_c \cos(\delta_a)) & r_c (R_a \cos(\delta_b) - R_b \cos(\delta_a)) \\ r_a (R_b \sin(\delta_c) - R_c \sin(\delta_b)) & -r_b (R_a \sin(\delta_c) - R_c \sin(\delta_a)) & r_c (R_a \sin(\delta_b) - R_b \sin(\delta_a)) \\ r_a \sin(\delta_b - \delta_c) & -r_b \sin(\delta_a - \delta_c) & r_c \sin(\delta_a - \delta_b) \end{bmatrix} \quad (6)$$

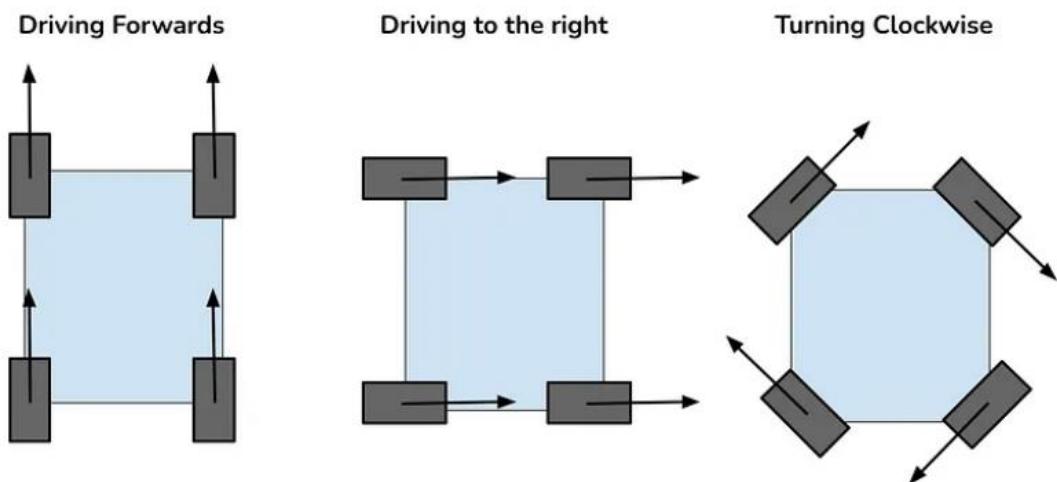
4. Kinematika Swerve Drive

Swerve drive memiliki sejumlah roda yang dapat berputar di tempatnya terlepas dari kerangka. Gaya yang dihasilkan ditunjukkan pada gambar 4.



Gambar 4. Gaya yang Dihasilkan Swerve Drive

Swerve drive adalah sistem penggerak yang memungkinkan setiap roda pada robot dapat bergerak secara independen, baik dalam hal rotasi (arah) maupun kecepatan. Ini memberikan fleksibilitas yang sangat tinggi untuk menggerakkan robot ke segala arah tanpa harus mengubah orientasi keseluruhan robot. Hal ini karena setiap roda digerakkan oleh dua motor, satu mengontrol arah kemudi roda, dan yang lainnya mengontrol kecepatan roda. Sehingga memungkinkan robot memiliki tingkat kebebasan yang lebih besar yang tidak dimiliki basis drive normal. Misalnya, drive normal tidak dapat bergerak ke samping atau menerjemahkan pada sudut seperti swerve drive. Berikut adalah beberapa konfigurasi dasar yang memungkinkan gerakan dasar pada swerve drive.



Gambar 5. Gerakan Dasar Pada Swerve Drive

Kinematika swerve drive melibatkan gerakan dan orientasi sistem robot yang memanfaatkan roda yang dapat berputar, memungkinkan gerakan multi-arah. Dua konsep kunci yang digunakan dalam pengendalian swerve drive adalah *forward kinematics* dan *inverse kinematics*.

a. Inverse Kinematics (Kinematika invers)

Kinematika invers secara efisien mengubah kecepatan casis atau kerangka ke masing masing roda secara individu. Kinematika invers pada swerve drive membantu dalam mencapai kontrol robot yang tepat dengan menghitung sudut roda dan kecepatan roda yang dibutuhkan untuk mencapai posisi dan orientasi yang diinginkan.

$$\vec{v}_{wheel1} = \vec{v}_{robot} + \vec{\omega}_{robot} \times \vec{r}_{robot2wheel1}$$

$$\vec{v}_{wheel2} = \vec{v}_{robot} + \vec{\omega}_{robot} \times \vec{r}_{robot2wheel2}$$

$$\vec{v}_{wheel3} = \vec{v}_{robot} + \vec{\omega}_{robot} \times \vec{r}_{robot2wheel3}$$

$$\vec{v}_{wheel4} = \vec{v}_{robot} + \vec{\omega}_{robot} \times \vec{r}_{robot2wheel4}$$

di mana \vec{v} adalah vector kecepatan linier, $\vec{\omega}$ adalah vector kecepatan sudut, \vec{r} adalah vector jarak dari pusat rotasi ke roda, $\vec{v}_{robot} = v_x\hat{i} + v_y\hat{j}$, dan $\vec{r}_{robot2wheel} = r_x\hat{i} + r_y\hat{j}$.

$$\vec{v}_1 = v_x\hat{i} + v_y\hat{j} + \omega\hat{k} \times (r_{1x}\hat{i} + r_{1y}\hat{j})$$

$$\vec{v}_2 = v_x\hat{i} + v_y\hat{j} + \omega\hat{k} \times (r_{2x}\hat{i} + r_{2y}\hat{j})$$

$$\vec{v}_3 = v_x\hat{i} + v_y\hat{j} + \omega\hat{k} \times (r_{3x}\hat{i} + r_{3y}\hat{j})$$

$$\vec{v}_4 = v_x\hat{i} + v_y\hat{j} + \omega\hat{k} \times (r_{4x}\hat{i} + r_{4y}\hat{j})$$

$$\vec{v}_1 = v_x\hat{i} + v_y\hat{j} + (\omega r_{1x}\hat{j} - \omega r_{1y}\hat{i})$$

$$\vec{v}_2 = v_x\hat{i} + v_y\hat{j} + (\omega r_{2x}\hat{j} - \omega r_{2y}\hat{i})$$

$$\vec{v}_3 = v_x\hat{i} + v_y\hat{j} + (\omega r_{3x}\hat{j} - \omega r_{3y}\hat{i})$$

$$\vec{v}_4 = v_x\hat{i} + v_y\hat{j} + (\omega r_{4x}\hat{j} - \omega r_{4y}\hat{i})$$

$$\begin{aligned}\vec{v}_1 &= v_x \hat{i} + v_y \hat{j} + \omega r_{1x} \hat{j} - \omega r_{1y} \hat{i} \\ \vec{v}_2 &= v_x \hat{i} + v_y \hat{j} + \omega r_{2x} \hat{j} - \omega r_{2y} \hat{i} \\ \vec{v}_3 &= v_x \hat{i} + v_y \hat{j} + \omega r_{3x} \hat{j} - \omega r_{3y} \hat{i} \\ \vec{v}_4 &= v_x \hat{i} + v_y \hat{j} + \omega r_{4x} \hat{j} - \omega r_{4y} \hat{i}\end{aligned}$$

$$\begin{aligned}\vec{v}_1 &= v_x \hat{i} - \omega r_{1y} \hat{i} + v_y \hat{j} + \omega r_{1x} \hat{j} \\ \vec{v}_2 &= v_x \hat{i} - \omega r_{2y} \hat{i} + v_y \hat{j} + \omega r_{2x} \hat{j} \\ \vec{v}_3 &= v_x \hat{i} - \omega r_{3y} \hat{i} + v_y \hat{j} + \omega r_{3x} \hat{j} \\ \vec{v}_4 &= v_x \hat{i} - \omega r_{4y} \hat{i} + v_y \hat{j} + \omega r_{4x} \hat{j}\end{aligned}$$

$$\begin{aligned}\vec{v}_1 &= (v_x - \omega r_{1y}) \hat{i} + (v_y + \omega r_{1x}) \hat{j} \\ \vec{v}_2 &= (v_x - \omega r_{2y}) \hat{i} + (v_y + \omega r_{2x}) \hat{j} \\ \vec{v}_3 &= (v_x - \omega r_{3y}) \hat{i} + (v_y + \omega r_{3x}) \hat{j} \\ \vec{v}_4 &= (v_x - \omega r_{4y}) \hat{i} + (v_y + \omega r_{4x}) \hat{j}\end{aligned}$$

Memisahkan komponen $-i$ ke persamaan independen:

$$\begin{aligned}v_{1x} &= v_x - \omega r_{1y} \\ v_{2x} &= v_x - \omega r_{2y} \\ v_{3x} &= v_x - \omega r_{3y} \\ v_{4x} &= v_x - \omega r_{4y}\end{aligned}$$

Memisahkan komponen $-j$ ke persamaan independent:

$$\begin{aligned}v_{1y} &= v_y + \omega r_{1x} \\ v_{2y} &= v_y + \omega r_{2x} \\ v_{3y} &= v_y + \omega r_{3x} \\ v_{4y} &= v_y + \omega r_{4x}\end{aligned}$$

Memfaktorkannya ke dalam matriks:

$$\begin{bmatrix} v_{1x} \\ v_{2x} \\ v_{3x} \\ v_{4x} \\ v_{1y} \\ v_{2y} \\ v_{3y} \\ v_{4y} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -r_{1y} \\ 1 & 0 & -r_{2y} \\ 1 & 0 & -r_{3y} \\ 1 & 0 & -r_{4y} \\ 0 & 1 & r_{1x} \\ 0 & 1 & r_{2x} \\ 0 & 1 & r_{3x} \\ 0 & 1 & r_{4x} \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}$$

Menyusun kembali baris-baris sehingga komponen x dan y berada pada baris yang berdekatan:

$$\begin{bmatrix} v_{1x} \\ v_{1y} \\ v_{2x} \\ v_{2y} \\ v_{3x} \\ v_{3y} \\ v_{4x} \\ v_{4y} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -r_{1y} \\ 0 & 1 & r_{1x} \\ 1 & 0 & -r_{2y} \\ 0 & 1 & r_{2x} \\ 1 & 0 & -r_{3y} \\ 0 & 1 & r_{3x} \\ 1 & 0 & -r_{4y} \\ 0 & 1 & r_{4x} \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}$$

Untuk mengubah dari modul swerve komponen kecepatan x dan y ke kecepatan dan arah, digunakan teorema Pythagoras dan arctangent secara berturut-turut:

$$v = \sqrt{v_x^2 + v_y^2}$$

$$\theta = \tan^{-1}\left(\frac{v_y}{v_x}\right)$$

b. Forward Kinematics (Kinematika Maju)

Kinematika maju pada swerve drive digunakan untuk menentukan pergerakan keseluruhan robot (kecepatan linier dan rotasi) berdasarkan kecepatan dan arah setiap roda. Jika kita mengetahui kecepatan dan sudut (arah) dari setiap roda, kita bisa menghitung bagaimana robot akan bergerak secara keseluruhan.

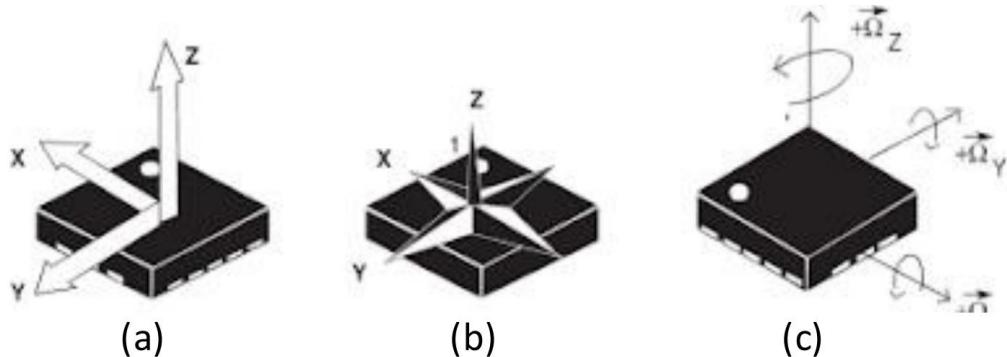
Misal \mathbf{M} menjadi matrik kinematika invers 8×3 di atas. Kinematika majunya adalah:

$$\begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} = \mathbf{M}^\dagger \begin{bmatrix} v_{1x} \\ v_{1y} \\ v_{2x} \\ v_{2y} \\ v_{3x} \\ v_{3y} \\ v_{4x} \\ v_{4y} \end{bmatrix}$$

di mana \mathbf{M}^\dagger adalah pseudoinverse dari \mathbf{M} .

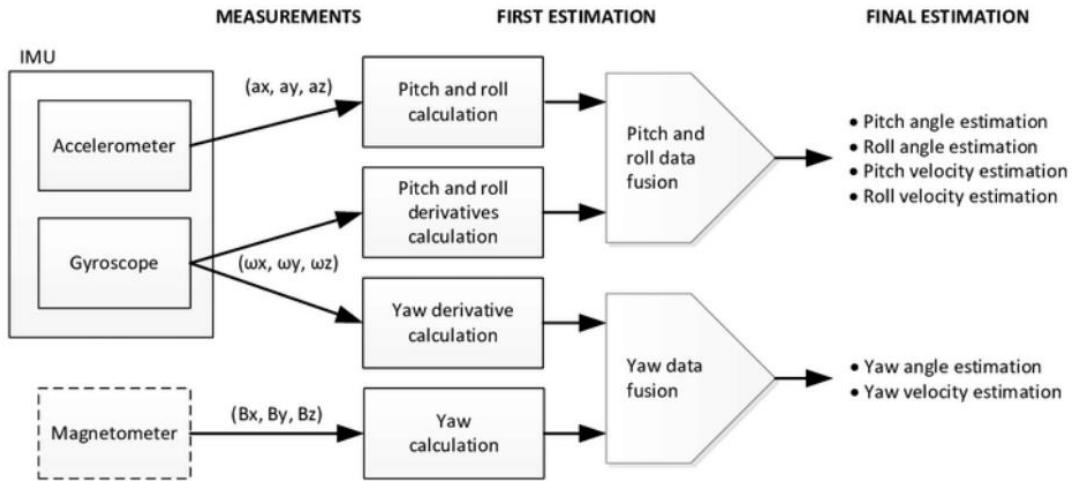
5. Inertial Measurement Unit (IMU)

Unit Pengukuran Inersia (IMU) adalah perangkat atau sensor MEMS yang menggabungkan beberapa sistem pengukuran, yaitu akselerometer, giroskop, dan magnetometer, di mana masing-masing sistem memiliki tiga sumbu utama. IMU berfungsi untuk memperkirakan posisi relatif, kecepatan, percepatan, dan kemiringan suatu objek terhadap orientasi tertentu.



Gambar 6. Ilustrasi Tiga Komponen Utama IMU: (a) Akselerometer, (b) Magnetometer, (c) Giroskop

Prinsip kerja IMU didasarkan pada sifat inersia atau kelembaman (yang secara numerik direpresentasikan oleh massa) dari sistem terhadap orientasi tertentu, serta menghitung perubahan nilai inersia secara terus-menerus. Gambar 6. menunjukkan penggabungan data dari akselerometer, giroskop, dan magnetometer untuk memperkirakan sudut roll, pitch, dan yaw.



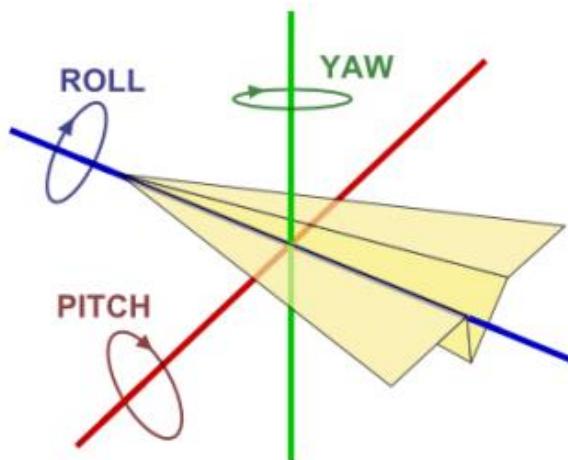
Gambar 7. Diagram Blok dari Kombinasi Ketiga Sensor IMU

Pada kolom blok pertama (IMU), akselerometer berfungsi untuk mengukur percepatan (a dalam m/s^2), giroskop untuk mengukur kecepatan sudut (ω dalam rad/s), dan magnetometer untuk membaca arah mata angin. Semua pengukuran tersebut berupa data mentah. Kolom blok kedua merupakan proses pengolahan data mentah menjadi nilai sudut pada sumbu yaw, pitch, dan roll melalui turunan pertama (kecepatan a) dan kedua (percepatan sudut ω). Akselerometer hanya bisa memberikan nilai pitch dan roll, sedangkan giroskop bisa memberikan nilai yaw, pitch, dan roll. Magnetometer hanya menghasilkan nilai yaw, yang digunakan untuk mengoreksi nilai yaw dari giroskop jika terjadi offset, karena magnetometer menggunakan medan magnet bumi sebagai referensi.

Kolom ketiga adalah tahap estimasi, di mana nilai yaw, pitch, dan roll dari berbagai sensor digabungkan. Sudut pitch dan roll dihasilkan dari kombinasi akselerometer dan giroskop, sedangkan sudut yaw merupakan kombinasi dari akselerometer dan magnetometer. Pada kolom blok terakhir, hasil penggabungan ini berupa posisi dan kecepatan sudut. Perhitungan yang digunakan untuk mengolah data mentah dilakukan dengan metode quarternion, menggunakan sudut Euler dan matriks rotasi.

Secara umum, sensor IMU dapat digunakan untuk menghitung pergerakan suatu objek terhadap lingkungannya, termasuk posisi, kecepatan, dan percepatan linier maupun angular. Dalam tugas akhir ini, sensor IMU digunakan untuk menghitung posisi sudut (angular) robot pada sumbu yaw, pitch, dan roll. Berikut adalah penjelasan mengenai ketiga sumbu tersebut:

- a) Sumbu Normal (Yaw): Sumbu ini memiliki titik asal di pusat gravitasi dan ditarik dari atas ke bawah, tegak lurus dengan dua sumbu lainnya. Gerakan rotasi melalui sumbu ini disebut yaw.
- b) Sumbu Lateral (Pitch): Sumbu ini memiliki titik asal di pusat gravitasi, ditarik dari samping kanan ke kiri sejajar dengan bidang horizontal. Gerakan rotasi pada sumbu ini disebut pitch.
- c) Sumbu Longitudinal (Roll): Sumbu ini memiliki titik asal di pusat gravitasi, ditarik dari depan ke belakang sejajar dengan bidang horizontal. Gerakan rotasi melalui sumbu ini disebut roll.



Gambar 8. Ilustrasi Sumbu Utama Pada Sistem Koordinat Kartesius

6. Giroskop

Giroskop berfungsi untuk mengukur kecepatan sudut pada sumbu acuan tetap (sumbu x, y, dan z) terhadap ruang inersia dalam skala mikro. Prinsip kerja giroskop ini didasarkan pada elemen yang bergetar dengan frekuensi tinggi untuk mendeteksi kecepatan sudut pada masing-masing sumbu. Teori yang digunakan untuk mengukur kecepatan sudut ini adalah Efek Coriolis. Efek Coriolis adalah pembelokan arah suatu benda yang bergerak ketika dilihat dari kerangka acuan yang berputar. Secara matematis, fenomena ini dapat dituliskan dalam persamaan:

$$\mathbf{F}_{\text{Coriolis}} = -2m\boldsymbol{\Omega} \times \mathbf{v}$$

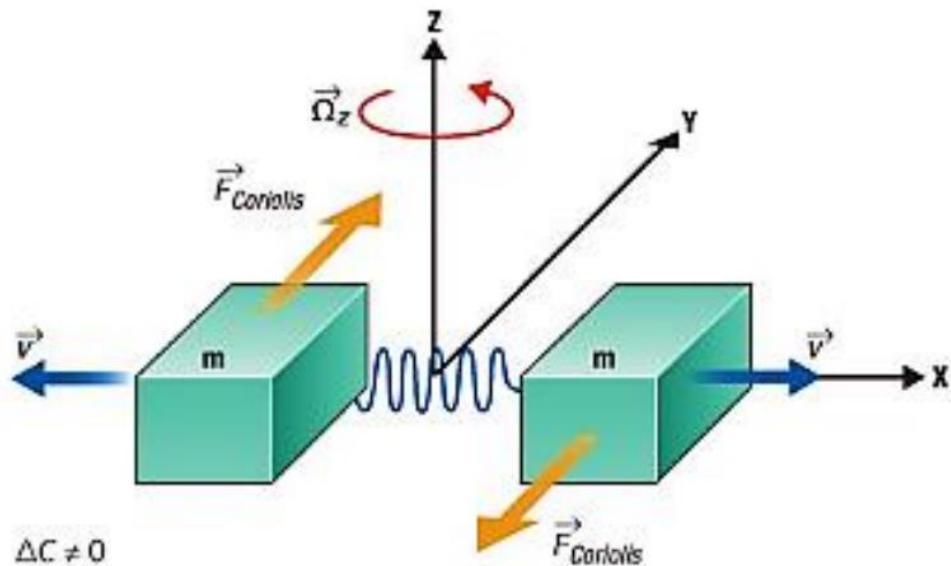
Dimana:

$\mathbf{F}_{\text{Coriolis}}$: Gaya Coriolis (N)

m : Massa obyek yang diterapkan (kg)

Ω : Kecepatan angular yang terjadi (rad/s)

v : Kecepatan (m/s)



Gambar 9. Ilustrasi Efek Coriolis Pada Giroskop

Dari ilustrasi di atas, dapat dijelaskan bahwa ketika sebuah massa uji m bergetar dan bergerak secara konstan ke arah yang berlawanan dengan kecepatan v , kemudian dikenakan rotasi sebesar Ω , massa tersebut akan mengalami suatu gaya (ditunjukkan oleh panah oranye). Gaya tersebut dikenal sebagai gaya Coriolis.

7. Akselerometer

Akselerometer adalah perangkat yang digunakan untuk mengukur percepatan suatu sistem terhadap lingkungannya, sehingga mampu mendeteksi perubahan posisi dan kecepatan serta seberapa besar perubahan yang terjadi pada sistem tersebut. Prinsip kerja akselerometer didasarkan pada Hukum Kedua Newton tentang gerak ($F = ma$), di mana alat ini memiliki massa yang bergerak dan instrumen khusus untuk mengukur perpindahan massa tersebut.

Struktur akselerometer terdiri dari jangkar (anchor) sebagai komponen tetap dan elemen sensor sebagai komponen yang bergerak, termasuk massa uji (proof mass), kapasitor komb (comb capacitor), dan pegas (springs) yang terbuat dari kristal silikon tunggal.

Prinsip kerja kapasitansi akselerometer didasarkan pada prinsip *self-balancing bridge* seperti Gambar 10. Ketika terjadi akselerasi, gaya inersia mempengaruhi proof mass, menyebabkan pergerakan yang kemudian diimbangi oleh gaya pegas saat akselerasi berkurang. Perpindahan proof mass ini mengubah kapasitansi pada C_1 dan C_2 . Pengendali (driver) mengirimkan sinyal AC ke elektroda tetap (pada anchor), dan elektroda tidak tetap (pada proof mass) yang terhubung ke terminal output akan membaca sinyal berdasarkan perubahan kapasitansi antara C_1 dan C_2 .

Maka persamaan dalam menghitung perubahan posisi tetap x_0 terhadap x pada masing masing kapasitor adalah:

$$C_1 = \varepsilon \frac{A}{x_0 + x} = \varepsilon \frac{A}{x_0 \cdot \left(1 + \frac{x}{x_0}\right)} = \frac{C_0}{1} + \delta$$

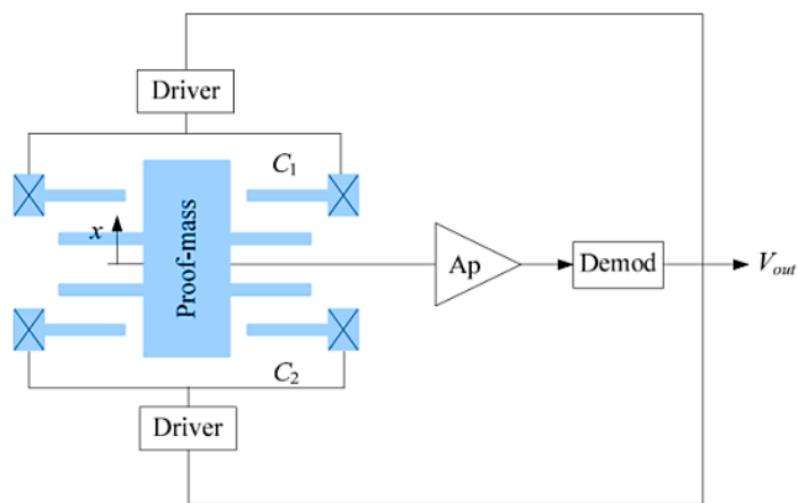
$$C_2 = \varepsilon \frac{A}{x_0 - x} = \varepsilon \frac{A}{x_0 \cdot \left(1 - \frac{x}{x_0}\right)} = \frac{C_0}{1} - \delta$$

Jika $\delta \ll 1$ dalam jembatan Wheatstone, maka:

$$V_{out} = \frac{V_{in}}{2} \times \delta$$

Dari persamaan kapasitor, diperoleh percepatan a :

$$a = \frac{K}{M} \cdot x = \frac{K}{M} \cdot x_0 \delta = \frac{K}{M} \cdot x_0 \cdot 2 \cdot \frac{V_{out}}{V_{in}}$$



Gambar 10. Skema Akselerometer Kapasitif

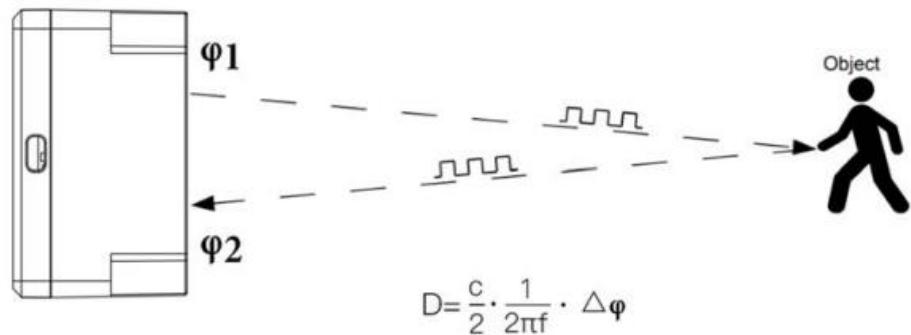
8. Filter Mahony

Filter Mahony yang diusulkan oleh Robert Mahony adalah metode yang dapat menghitung orientasi suatu objek dengan akurasi tinggi dalam waktu singkat menggunakan sensor akselerometer 3-sumbu, giroskop 3-sumbu, dan magnetometer 3-sumbu. Filter ini menggunakan quaternion sebagai representasi orientasi untuk menggambarkan orientasi objek dalam dunia 3D. Penggunaan quaternion ini penting karena quaternion dapat menghindari masalah singularitas pada sudut Euler (dikenal sebagai gimbal lock), yang dapat menyebabkan kesalahan perhitungan orientasi.

Prinsip dasar dari filter Mahony adalah meminimalkan error (bias pada giroskop) antara vektor pengukuran sensor di kerangka tubuh (body frame) dan vektor referensi yang dikonversi dari kerangka navigasi ke kerangka tubuh. Dengan cara ini, filter Mahony dapat mengoreksi bias pada giroskop secara efektif dan menghasilkan estimasi orientasi yang lebih akurat, terutama dalam skenario gerakan dinamis. Kombinasi data dari akselerometer, giroskop, dan magnetometer juga membantu dalam memberikan estimasi orientasi yang stabil dan responsif terhadap perubahan cepat dalam gerakan objek.

9. LiDAR

LiDAR adalah singkatan dari *Light Detection and Ranging*, sebuah teknologi yang menggunakan pulsa cahaya untuk mengukur jarak. Dalam hal ini, LiDAR yang digunakan adalah TFmini Plus. TFmini Plus bekerja berdasarkan prinsip *Time of Flight* (TOF). Secara khusus, perangkat ini memancarkan gelombang modulasikan sinar inframerah secara berkala, yang kemudian dipantulkan setelah mengenai objek. Perangkat ini menghitung waktu penerbangan (time of flight) dengan mengukur perbedaan fase dari perjalanan pulang-pergi sinar tersebut, lalu menghitung jarak relatif antara perangkat dan objek yang dideteksi, seperti ditunjukkan dalam gambar.

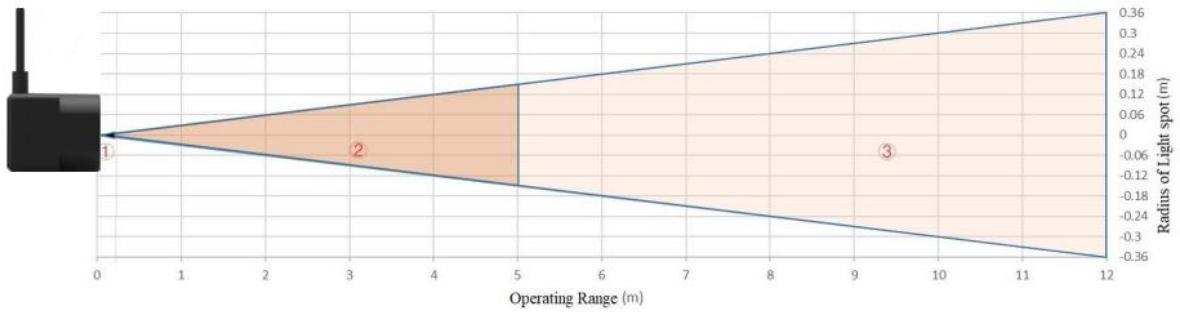


Gambar 11. Skema dari Prinsip Kerja TOF

Tabel 1. Parameter Karakteristik Utama dari TFmini

Description	Parameter value
Operating range	0.1m~12m ^①
Accuracy	±5cm@ (0.1-6m) ^②
	±1%@ (6m-12m)
Measurement unit	cm
Range resolution	5mm
FOV	3.6° ^③
Frame rate	1~1000Hz (adjustable) ^④

Tingkat keakuratan pengukuran jarak yang berulang pada TFmini Plus sangat bergantung pada nilai intensitas yang diukur dan kecepatan frame output (frekuensi). Dengan adanya optimasi pada jalur cahaya dan algoritma, TFmini Plus mampu meminimalkan pengaruh dari lingkungan eksternal terhadap kinerja pengukuran jarak. Meskipun demikian, jarak pengukuran masih dapat dipengaruhi oleh intensitas pencahayaan lingkungan dan tingkat reflektivitas objek yang dideteksi. Seperti yang terlihat pada gambar.



Gambar 12. Diagram Skema dari Ukuran Titik Cahaya

10. PID Controllers

PID controller (Proportional-Integral-Derivative controller) digunakan dalam sistem kendali untuk menghitung kesalahan (error) antara nilai yang diinginkan (setpoint) dan variabel proses yang terukur (process variable), lalu menyesuaikan keluaran untuk meminimalkan kesalahan tersebut. Pengontrol ini menggunakan tiga strategi utama: kontrol proporsional, kontrol integral, dan kontrol derivatif, sehingga mampu meningkatkan stabilitas dan kinerja dalam proses otomatisasi.

Referensi disebut setpoint (posisi yang diinginkan) dan outputnya disebut variabel proses (posisi terukur). Di bawah ini adalah beberapa konvensi penamaan variabel umum untuk jumlah yang relevan.

$$\begin{array}{llll} r(t) & \text{setpoint} & u(t) & \text{control input} \\ e(t) & \text{error} & y(t) & \text{output} \end{array}$$

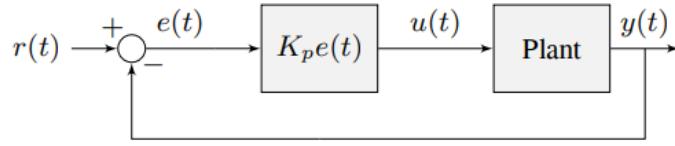
Eror $e(t)$ adalah $r(t) - y(t)$.

a. Proportional

Kontrol proporsional bertujuan untuk mengurangi kesalahan posisi menjadi nol. Kesalahan posisi dihitung sebagai selisih antara nilai setpoint dan posisi yang terukur saat ini.

$$u(t) = K_p e(t)$$

dimana K_p adalah penguatan proporsional and $e(t)$ adalah error pada waktu t .



Gambar 13. Diagram Blok P Controller

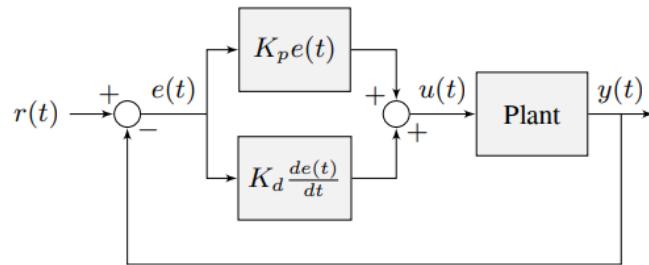
b. Derivative

Kontrol derivatif bertujuan untuk mengurangi kesalahan kecepatan menjadi nol.

Ini bekerja dengan melihat kecepatan perubahan kesalahan posisi, mengurangi efek perubahan tiba-tiba pada error.

$$u(t) = K_p e(t) + K_d \frac{de}{dt}$$

dimana K_p adalah penguatan proporsional, K_d adalah penguatan derivatif, $e(t)$ adalah error pada waktu t .



Gambar 14. Diagram Blok PD Controller

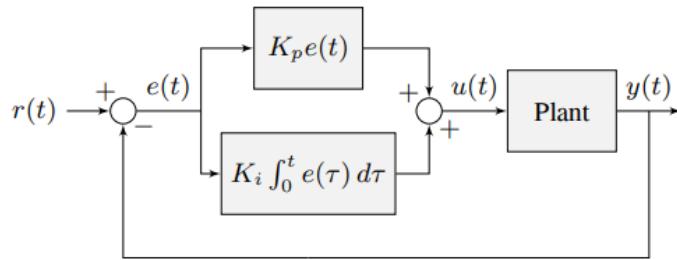
c. Integral

Kontrol integral mengakumulasikan kesalahan posisi dari waktu ke waktu.

Dengan menambahkan akumulasi kesalahan ke input kendali, kontrol integral membantu mengatasi kesalahan *steady-state* (kesalahan yang tidak hilang dalam jangka panjang).

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau$$

dimana K_p adalah penguatan proporsional, K_i adalah penguatan integral, $e(t)$ adalah error pada waktu t , and τ adalah variabel integrasi. $\int_0^t e(\tau) d\tau$ adalah integral dari kesalahan seiring waktu, mengakumulasi area di antara kurva setpoint dan keluaran.

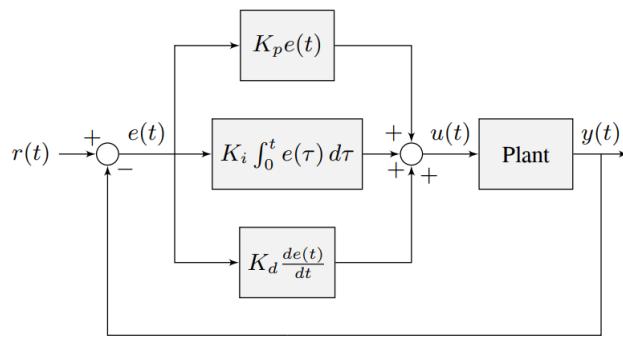


Gambar 15. Diagram Blok PI Controller

Ketika ketiga kontrol (proporsional, integral, dan derivatif) digabungkan, rumus PID menjadi:

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

dimana K_p adalah penguatan proporsional, K_d adalah penguatan derivative, K_i adalah penguatan integral, $e(t)$ adalah eror pada waktu t , and τ adalah variabel integrasi



Gambar 16. Diagram Blok PID Controller

11. Algoritma A-Star

Perencanaan jalur atau *path planning* mempunyai peranan yang penting untuk proses navigasi robot. Navigasi diartikan sebagai proses atau aktivitas untuk merencanakan atau menuju jalur secara langsung dalam sebuah misi yang diberikan pada sebuah autonomous mobile robot dari satu tempat ke tempat yang lain tanpa kehilangan arah atau mengalami tabrakan dengan object yang lain. Standar pencarian jalur terdekat menggunakan algoritma A* adalah menggunakan persamaan:

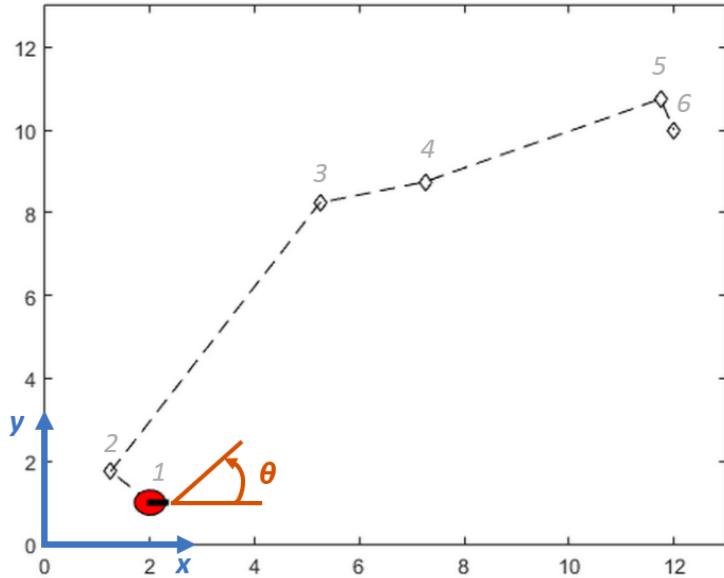
$$f(n) = g(n) + h(n)$$

Pada persamaan yang dipakai untuk algoritma A* tersebut, digunakan $g(n)$ untuk mewakili harga (cost) rute dari titik awal ke node n , lalu $h(n)$ mewakili perkiraan harga dari node awal ke node goal, yang dihitung dengan fungsi heuristic. A* menjumlahkan kedua nilai ini dalam mencari jalur dari node awal ke node goal. Algoritma ini akan memilih node dengan nilai $f(n)$ yang paling kecil.

12. Pure Pursuit Controller

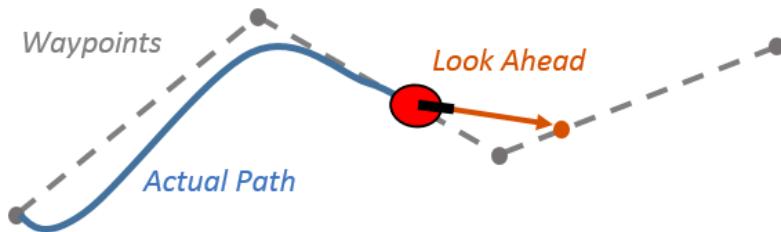
Pure Pursuit adalah algoritma pelacakan jalur yang digunakan untuk menghitung perintah kecepatan sudut yang akan menggerakkan robot dari posisinya saat ini untuk mencapai titik *look-ahead* di depan robot. Pengendali ini unik karena ditentukan untuk daftar *waypoint* yang telah ditetapkan. Kecepatan linear dianggap konstan, sehingga kecepatan linear robot bisa diubah kapan saja. Kecepatan linear yang diinginkan dan kecepatan sudut maksimum dapat ditentukan berdasarkan spesifikasi robot. Dengan memberikan pose (posisi dan orientasi) kendaraan sebagai input, objek ini dapat digunakan untuk menghitung perintah kecepatan linear dan sudut untuk robot. Algoritma ini kemudian menggerakkan titik *look-ahead* di sepanjang jalur berdasarkan posisi saat ini hingga mencapai titik akhir jalur. Dengan kata lain, robot terus-menerus mengejar titik di depannya. *Look Ahead Distance* menentukan seberapa jauh titik look-ahead ditempatkan.

Gambar 17. menunjukkan menunjukkan sistem koordinat referensi. Waypoint input adalah koordinat $[x \ y]$, yang digunakan untuk menghitung perintah kecepatan robot. Pose robot terdiri dari posisi xy dan sudut dalam bentuk $[x \ y \ theta]$. Arah x dan y positif masing-masing berada di arah kanan dan atas (warna biru pada gambar). Nilai θ adalah orientasi sudut robot yang diukur berlawanan arah jarum jam dalam radian dari sumbu x (robot saat ini berada pada 0 radian).



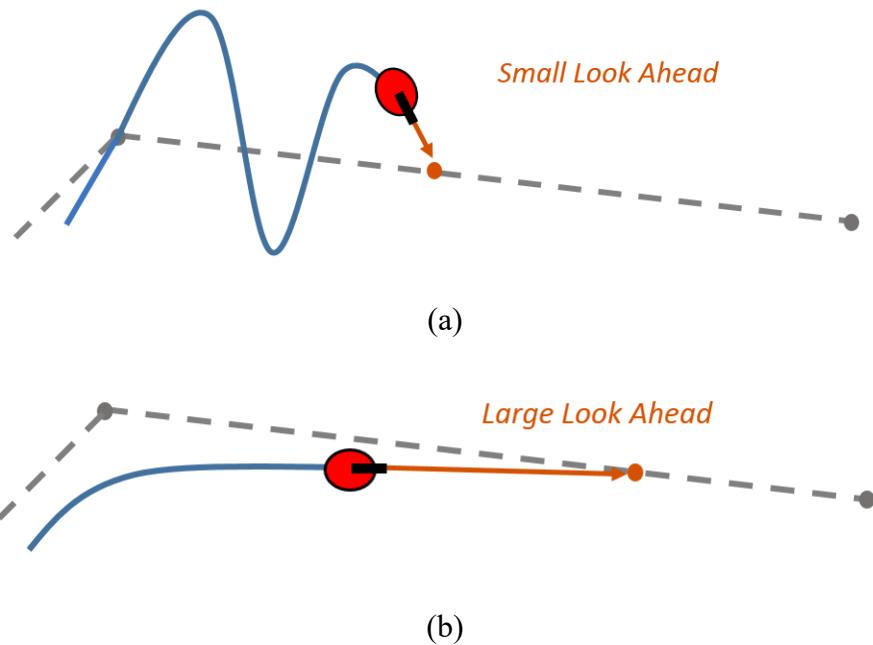
Gambar 17. Kerangka Koordinat Referensi

Look Ahead Distance adalah seberapa jauh di sepanjang jalur robot harus melihat dari posisi saat ini untuk menghitung perintah kecepatan sudut. Gambar 18. menunjukkan robot dan titik *look-ahead*. Seperti yang ditunjukkan dalam gambar, dapat dilihat bahwa jalur yang diinginkan tidak sesuai dengan garis lurus antara *waypoint*.



Gambar 18. Look Ahead Distance

Pengaruh perubahan parameter ini dapat mengubah cara robot mengikuti jalur, dan ada dua tujuan utama yaitu mengembalikan jalur dan mempertahankan jalur. Untuk cepat kembali ke jalur antara waypoint, jarak *Look Ahead Distance* yang kecil akan menyebabkan robot bergerak cepat menuju jalur. Namun, seperti yang terlihat pada gambar 19(a),, robot akan melewati jalur dan berosilasi di sepanjang jalur yang diinginkan. Untuk mengurangi osilasi di sepanjang jalur, jarak look-ahead yang lebih besar dapat dipilih, meskipun hal ini mungkin menyebabkan kurva yang lebih besar di dekat sudut-sudut seperti terlihat pada gambar 19(b).



Gambar 19. Pengaruh Look Ahead Distance (a) Kecil dan (b) Besar

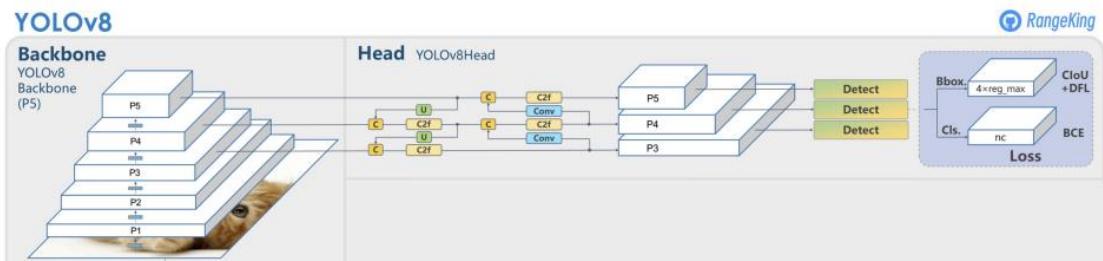
13. YOLOv8 dan Evaluasi Model

YOLO (You Only Look Once) adalah model deteksi objek dan segmentasi gambar yang dikembangkan oleh Joseph Redmon dan Ali Farhadi di Universitas Washington, dirilis pertama kali pada tahun 2015. YOLO cepat menjadi populer karena kecepatannya yang tinggi dan akurasinya yang baik. YOLOv8, versi terbaru dari YOLO yang dikembangkan oleh Ultralytics, menawarkan berbagai peningkatan performa, fleksibilitas, dan efisiensi dibandingkan versi sebelumnya. YOLOv8 mendukung berbagai tugas dalam visi komputer, seperti deteksi objek, segmentasi, estimasi pose, pelacakan, dan klasifikasi, menjadikannya model yang sangat serbaguna untuk berbagai aplikasi.

Versi terbaru ini memiliki arsitektur dasar yang mirip dengan pendahulunya, namun membawa banyak peningkatan. YOLOv8 menggunakan arsitektur jaringan saraf baru yang menggabungkan Feature Pyramid Network (FPN) dan Path Aggregation Network (PAN). Selain itu, YOLOv8 memperkenalkan alat pelabelan baru yang memudahkan proses anotasi data. Alat ini mencakup fitur-fitur seperti auto labeling, pintasan pelabelan, dan hotkey yang dapat dikustomisasi, yang semuanya bertujuan untuk menyederhanakan anotasi gambar untuk pelatihan model.

FPN bekerja dengan mengurangi resolusi spasial gambar input secara bertahap sambil meningkatkan jumlah saluran fitur. Ini menghasilkan peta fitur yang dapat mendeteksi objek pada skala dan resolusi yang berbeda. Sementara itu, PAN menggabungkan fitur dari berbagai level jaringan melalui skip connections (koneksi lompatan). Dengan pendekatan ini, jaringan dapat menangkap fitur dari berbagai skala dan resolusi dengan lebih baik, yang sangat penting untuk mendeteksi objek dengan berbagai ukuran dan bentuk secara akurat.

Model dasar YOLO bekerja dengan membagi gambar input menjadi grid berukuran $S \times S$. Setiap sel grid (i, j) memprediksi B kotak pembatas (bounding boxes), skor kepercayaan untuk setiap kotak, dan probabilitas kelas C . Output akhirnya akan berupa tensor dengan ukuran $S \times S \times (B \times 5 + C)$, yang mencakup informasi tentang posisi, ukuran, dan kelas objek yang terdeteksi di dalam gambar.



Gambar 20. Arsiteksur YOLOv8

Evaluasi model merupakan tahap penting dalam pengembangan model pembelajaran mesin, termasuk model deteksi objek seperti YOLOv8, yang bertujuan untuk mengukur performa model setelah proses pelatihan. Evaluasi ini memastikan bahwa model tidak hanya belajar mengikuti pola dari data pelatihan (train set) tetapi juga mampu menggeneralisasi dengan baik terhadap data yang belum pernah dilihat (validation/test set).

Berikut adalah beberapa aspek penting yang digunakan dalam evaluasi model YOLOv8:

1. Mean Average Precision (mAP)

mAP adalah ukuran yang mengevaluasi seberapa baik model mendeteksi objek secara keseluruhan dengan mempertimbangkan presisi dan recall. Presisi mengukur seberapa akurat model dalam mendeteksi objek yang benar. Semakin tinggi presisi, semakin sedikit deteksi palsu (false positives).

$$Precision = \frac{True\ Positives}{False\ Positives + True\ Positives}$$

Recall mengukur seberapa baik model dalam menemukan semua objek yang ada di gambar. Semakin tinggi recall, semakin sedikit objek yang terlewat (false negatives).

$$Recall = \frac{True\ Positives}{False\ Negatives + True\ Positives}$$

mAP dihitung dengan menghitung rata-rata presisi pada berbagai level IoU (Intersection over Union). IoU adalah ukuran yang menunjukkan seberapa besar tumpang tindih antara kotak prediksi model dan kotak ground truth (label asli). Pada umumnya, IoU minimum yang digunakan adalah 0.5.

2. Confusion Matrix

Confusion matrix dalam konteks deteksi objek juga dapat digunakan untuk mengevaluasi performa model. Dalam evaluasi ini, kita dapat melihat bagaimana model membedakan antara true positives (TP), false positives (FP), false negatives (FN), dan true negatives (TN). TP ketika model mendekksi objek dengan benar, FP ketika model mendekksi sesuatu yang salah sebagai objek, FN ketika model gagal mendekksi objek yang sebenarnya ada, dan TN ketika model dengan benar tidak mendekksi objek di lokasi di mana objek tidak ada.

3. Loss Function

Fungsi loss mengukur perbedaan antara prediksi model dan label ground truth. Pada YOLOv8, loss function terdiri dari beberapa komponen seperti classification loss (untuk memastikan kelas objek terprediksi dengan benar), bounding box loss (untuk akurasi lokasi bounding box), dan distribution focal loss (untuk mengidentifikasi apakah kotak prediksi benar-benar berisi objek).

B. Hasil Penelitian yang Relevan

Beberapa penelitian sebelumnya telah mengkaji sistem lokalisasi dan navigasi robot dengan menggunakan kombinasi sensor dan algoritma kontrol untuk meningkatkan akurasi dan efisiensi dalam lingkungan dinamis. Temuan-temuan dari penelitian tersebut menjadi landasan bagi penelitian ini dalam menyusun kerangka pemikiran dan merancang sistem multi-sensor pada robot swerve-drive.

Penelitian "Super Odometry: IMU-centric LiDAR-Visual-Inertial Estimator for Challenging Environments" oleh Shibo Zhao, Hengrui Zhang, Peng Wang, Lucas Nogueira, dan Sebastian Scherer (2021) memberikan kontribusi signifikan terhadap pengembangan sistem multi-sensor untuk robot. Penelitian ini memperkenalkan Super Odometry, sebuah kerangka kerja fusi sensor multi-modal yang memanfaatkan IMU, LiDAR, dan kamera untuk menghasilkan estimasi posisi dan orientasi yang akurat, terutama di lingkungan dengan kondisi persepsi yang menurun (misalnya, gelap atau berasap). Salah satu aspek penting dari penelitian ini adalah proses pengolahan data IMU-centric, yang menggabungkan keunggulan pendekatan loosely coupled dan tightly coupled untuk mengurangi bias drift pada IMU. Kerangka kerja ini terdiri dari tiga bagian: IMU odometry, visual-inertial odometry, dan LiDAR-inertial odometry. Visual-inertial dan laser-inertial odometry berfungsi untuk memberikan informasi awal posisi dan membantu mengoreksi bias pada IMU. Selain itu, prediksi gerakan yang berasal dari IMU digunakan untuk memperbaiki estimasi posisi secara lebih tepat. Penelitian ini juga menekankan pentingnya pemrosesan data secara real-time dengan efisiensi tinggi. Mereka menggunakan dynamic octree sebagai pengganti static KD-tree, yang mampu mengurangi waktu komputasi hingga 90%.

Penemuan dalam penelitian ini sangat relevan, terutama dalam hal integrasi multi-sensor untuk meningkatkan akurasi dan ketangguhan sistem. Pendekatan fusi sensor IMU, LiDAR, dan kamera yang diusulkan mendukung upaya dalam meminimalkan drift pada odometri dan memastikan robot tetap memiliki estimasi posisi yang akurat dalam lingkungan kompetisi yang dinamis.

C. Kerangka Pikir

Kerangka pemikiran dalam penelitian ini menjelaskan hubungan logis dan rasional antara variabel-variabel yang terlibat dalam sistem lokalisasi dan navigasi robot otonom berbasis swerve-drive. Penelitian ini berfokus pada bagaimana integrasi multi-sensor (odometri, gyroscope-accelerometer, LiDAR, dan kamera) serta algoritma kontrol (PID dan Pure Pursuit) dapat bekerja secara bersamaan untuk meningkatkan akurasi lokalisasi dan navigasi robot.

Odometri digunakan untuk memperkirakan perubahan posisi robot berdasarkan rotasi roda, namun sering mengalami kesalahan akumulatif (drift). Di sinilah gyroscope dan accelerometer berperan penting untuk mengoreksi kesalahan orientasi dan membantu menjaga kestabilan pergerakan robot. Kombinasi data dari odometri dan gyroscope-

accelerometer memungkinkan estimasi posisi dan orientasi yang lebih akurat. LiDAR dan kamera berfungsi sebagai sumber data eksternal untuk mendeteksi objek seperti bola dan silo, sekaligus membantu koreksi posisi robot di lingkungan yang dinamis. Data LiDAR memberikan jarak presisi tinggi ke objek di sekitar robot, sementara kamera digunakan untuk identifikasi objek dan pengenalan visual. Hubungan antara kedua sensor ini bersifat saling melengkapi, di mana LiDAR mengukur jarak dan kamera mengklasifikasi objek. Integrasi keduanya membantu robot menghindari rintangan dan mencapai target dengan akurasi tinggi.

Algoritma PID controller digunakan untuk mengatur kecepatan dan stabilitas robot, sedangkan Pure Pursuit berfungsi untuk menghitung jalur optimal yang harus dilalui robot agar mencapai target. Hubungan antara kedua algoritma ini sangat penting, karena PID controller menjaga pergerakan robot tetap stabil di sepanjang jalur yang telah dihitung oleh Pure Pursuit. Jika terjadi penyimpangan dari lintasan, Pure Pursuit akan segera memperbarui jalur, dan PID controller akan menyesuaikan kecepatan robot untuk kembali ke lintasan.

Semua sensor bekerja melalui multi-sensor fusion, di mana data dari odometri, gyroscope-accelerometer, LiDAR, dan kamera digabungkan untuk memperkirakan posisi dan orientasi robot secara akurat. Multi-sensor fusion memungkinkan sistem untuk memanfaatkan keunggulan masing-masing sensor dan mengkompensasi kelemahan sensor secara individu. Misalnya, drift pada odometri dikoreksi oleh data dari gyroscope dan LiDAR, sementara ketidakakuratan kamera dalam kondisi pencahayaan buruk dapat diatasi oleh LiDAR. Hubungan korelatif antara semua sensor ini memastikan bahwa robot selalu memiliki data posisi yang akurat dan dapat beradaptasi dengan lingkungan dinamis. Lingkungan lapangan yang dinamis, seperti keberadaan robot lain, rintangan, dan variasi pencahayaan, memengaruhi kinerja sistem navigasi. Oleh karena itu, integrasi antara sensor dan algoritma kontrol dirancang untuk mengatasi perubahan lingkungan secara real-time. Hubungan antara variabel navigasi dan kondisi lingkungan bersifat interdependen, karena perubahan pada lingkungan akan langsung memicu penyesuaian pada kontrol kecepatan, arah, dan jalur robot.

D. Pertanyaan Penelitian dan Hipotesis

1. Pertanyaan Penelitian

- a) Bagaimana data multi-sensor (odometri, gyroscope-accelerometer, LiDAR, dan kamera) dapat diintegrasikan secara akurat dan efisien untuk mendukung navigasi otonom?
- b) Bagaimana fusi data sensor dapat meminimalkan drift pada odometri dan meningkatkan akurasi posisi global robot?

2. Hipotesis

- a) Integrasi data multi-sensor (odometri, gyroscope-accelerometer, LiDAR, dan kamera) meningkatkan kemampuan robot untuk beradaptasi secara dinamis terhadap perubahan lingkungan secara real-time.
- b) Fusi data sensor secara signifikan mengurangi drift pada odometri, sehingga menghasilkan estimasi posisi global yang lebih akurat.

BAB III

METODE PENELITIAN

A. Jenis atau Desain Penelitian

Jenis penelitian ini adalah penelitian terapan. Penelitian terapan bertujuan untuk menyelesaikan masalah praktis. Dalam hal ini, mengembangkan sistem lokalisasi dan navigasi yang andal dan efisien untuk robot swerve-drive otonom dalam konteks kompetisi ABU Robocon 2024. Fokus penelitian ini adalah menerapkan pengetahuan teoretis tentang robotika, sensor, dan sistem kontrol untuk mengatasi tantangan nyata dalam navigasi robot dan performa dalam kompetisi.

Desain penelitian yang digunakan adalah penelitian eksperimental. Penelitian ini melibatkan perancangan, implementasi, dan pengujian algoritma serta sistem untuk lokalisasi dan navigasi robot. Dilakukan eksperimen untuk memvalidasi efektivitas metode integrasi sensor yang berbeda (odometri, gyroscope-accelerometer, LiDAR, dan kamera), mengatur algoritma kontrol (seperti PID dan Pure Pursuit), serta menilai seberapa baik robot beradaptasi di lingkungan yang dinamis dalam kondisi terkontrol. Desain ini memungkinkan Anda untuk memanipulasi variabel (input sensor, konfigurasi algoritma) dan mengamati hasilnya pada performa robot.

B. Tempat dan Waktu Penelitian

Penelitian dan pengembangan robot dilakukan di Aula Robotika dan Gedung KPLT lt. 3 Universitas Negeri Yogyakarta. Penelitian dan pengembangan ini dimulai dari 9 September 2023 hingga 2 Agustus 2024.

C. Sumber Data atau Subjek Penelitian

Subjek penelitian adalah robot yang sedang dikembangkan dan diuji dalam berbagai skenario untuk mengukur kinerja sistem navigasi dan lokalisasi yang dibuat. Sumber data berasal dari berbagai pengujian terhadap robot menggunakan sistem dan sensor-sensor yang telah diintegrasikan (odometri, gyroscope-accelerometer, LiDAR, kamera) di dalam lingkungan berupa lapangan kompetisi ABU Robocon.

D. Definisi Operasional Variabel

Dalam penelitian ini, variabel yang digunakan berkaitan dengan pengukuran dan evaluasi sistem lokalisasi dan navigasi robot otonom berbasis swerve-drive menggunakan integrasi beberapa sensor. Berikut adalah definisi operasional dari masing-masing variabel yang digunakan dalam penelitian ini:

1. Lokalisasi Robot

Lokalisasi adalah proses untuk menentukan posisi dan orientasi robot di lingkungan lapangan secara akurat. Dalam penelitian ini, lokalisasi mengacu pada kemampuan sistem untuk memperkirakan posisi robot secara global berdasarkan data dari sensor odometri, gyroscope-accelerometer, LiDAR, dan kamera. Indikator penelitian:

- a. Akurasi posisi robot.
- b. Kesalahan akumulatif (drift) pada odometri dan seberapa besar koreksi yang dilakukan menggunakan sensor lain (gyroscope-accelerometer, LiDAR).
- c. Tingkat penyimpangan posisi (error) dibandingkan dengan posisi sebenarnya di lapangan.

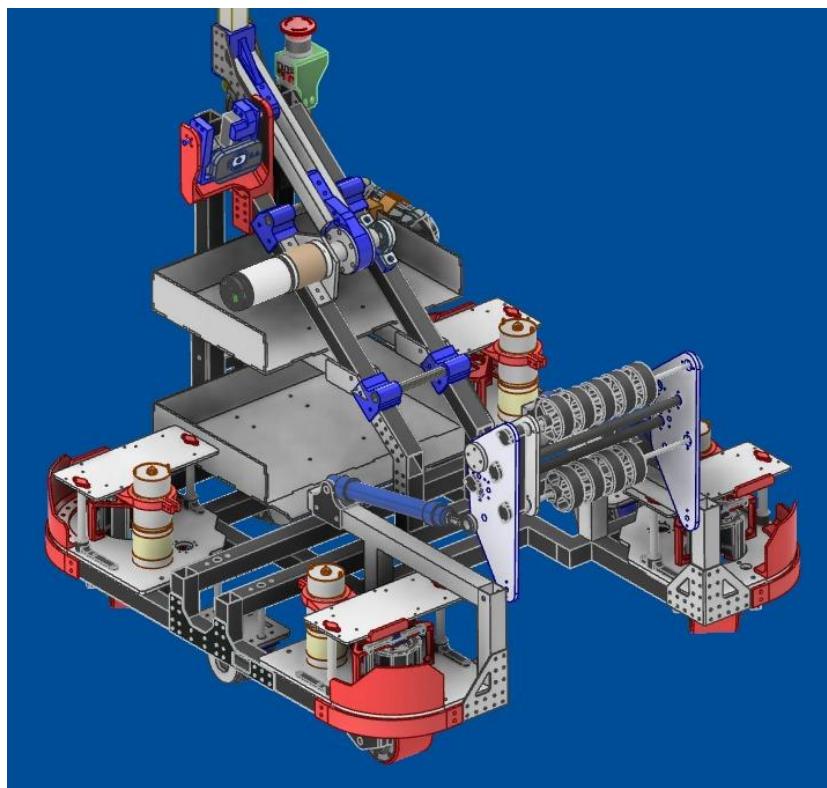
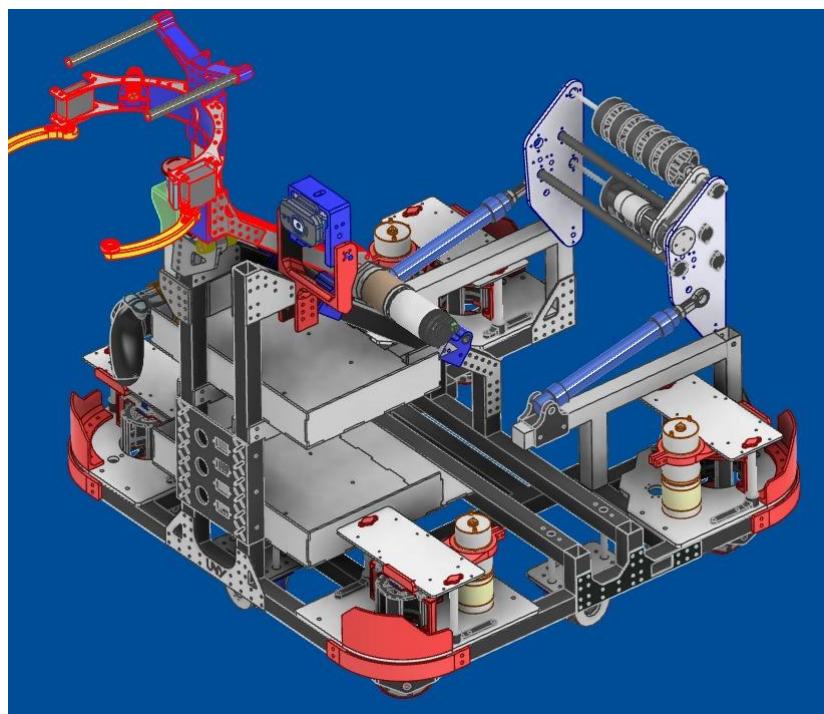
2. Navigasi Robot

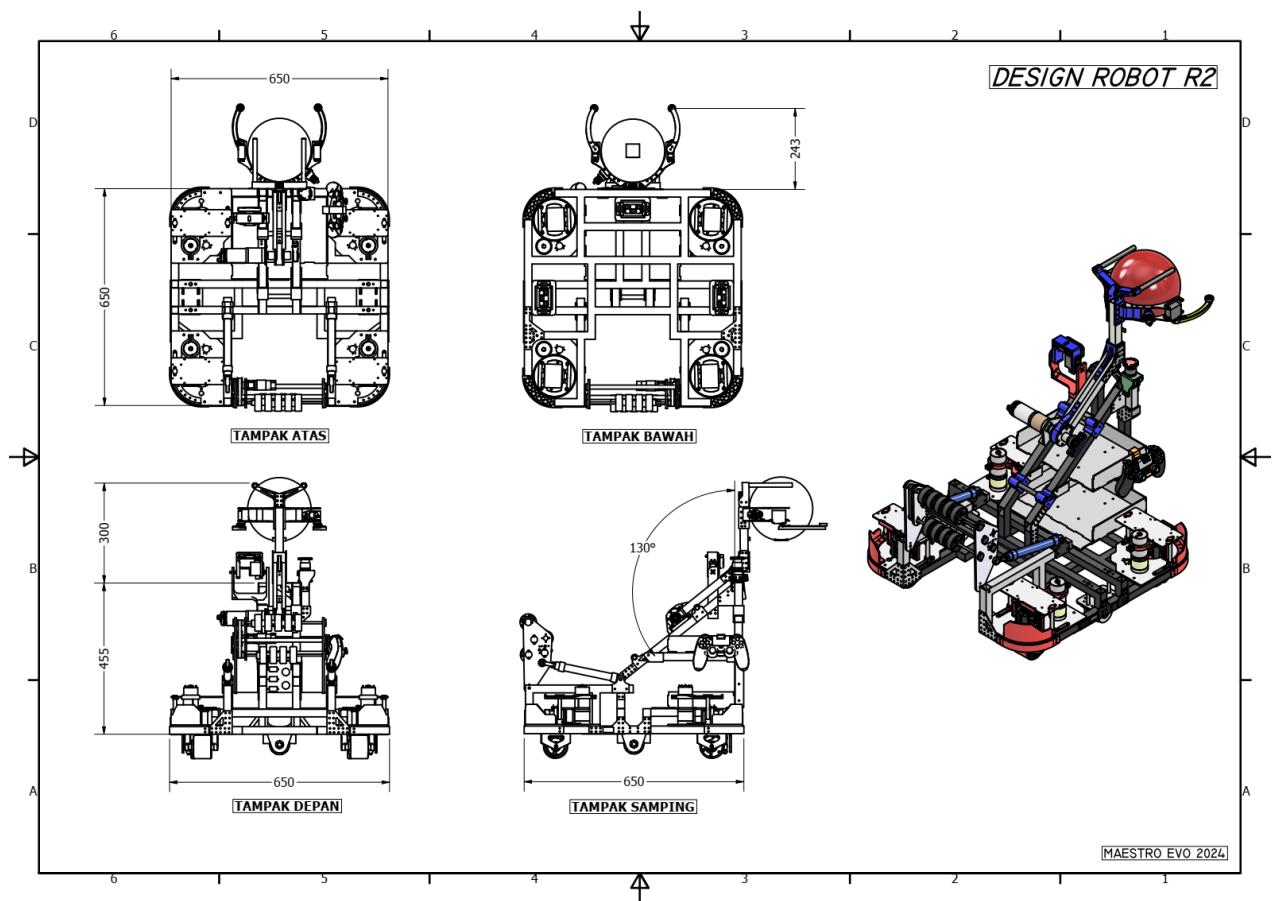
Navigasi adalah kemampuan robot untuk bergerak dari satu titik ke titik lainnya di lapangan kompetisi dengan rute yang optimal. Dalam penelitian ini, navigasi melibatkan pemrosesan data dari sensor untuk mengontrol kecepatan, arah, dan jalur robot menggunakan algoritma seperti PID dan Pure Pursuit. Indikator penelitian:

- a. Akurasi lintasan (seberapa dekat robot mengikuti jalur yang telah ditentukan).
- b. Akurasi robot dalam mencapai target (bola dan silo) yang dipilih.

E. Teknik dan Instrumentasi Pengumpulan Data

1. Desain Robot

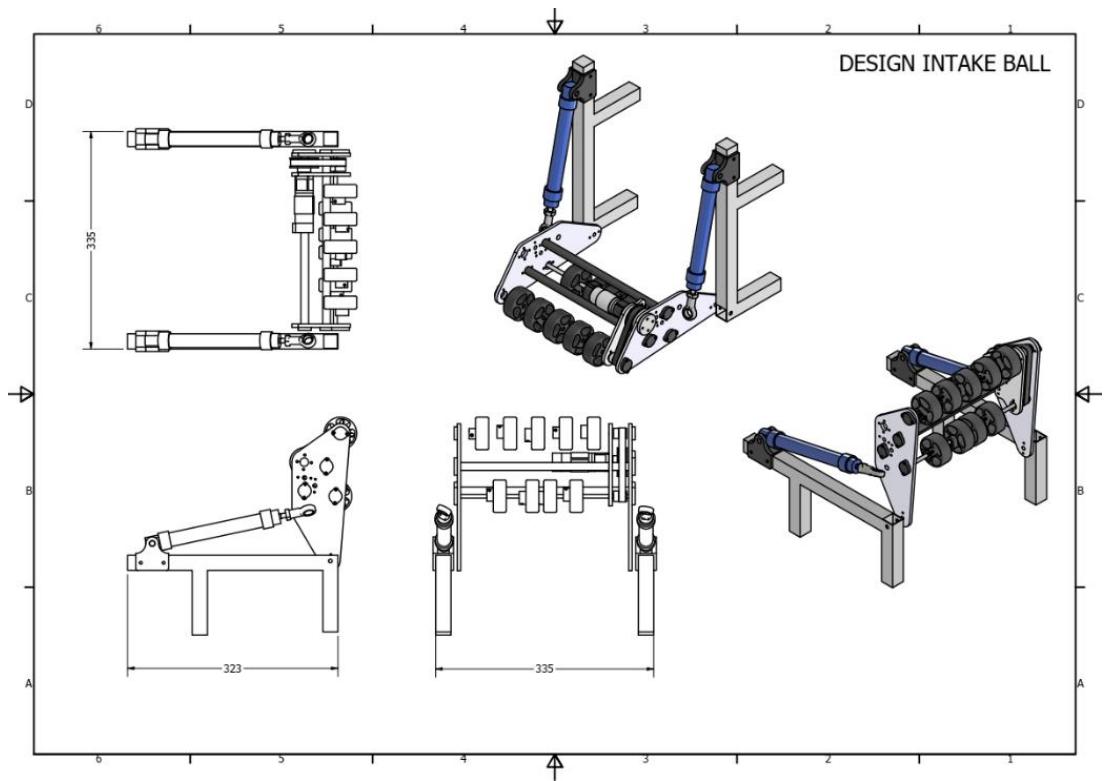




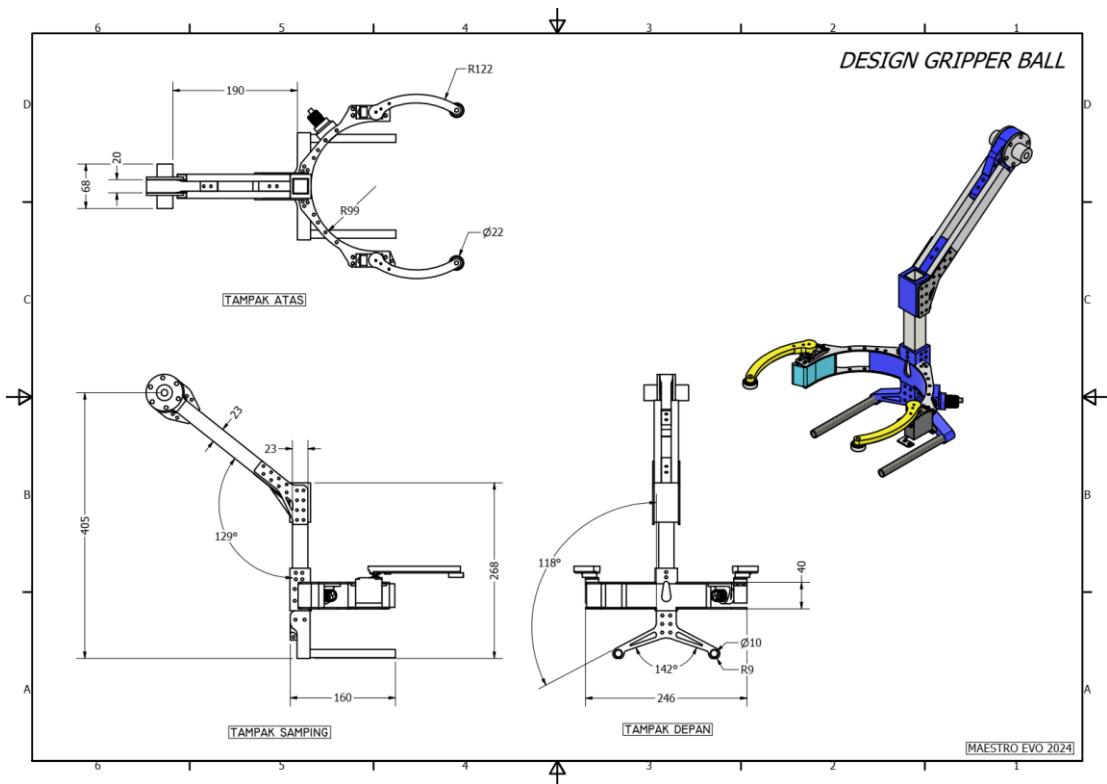
Gambar 21. Desain Robot

Keterangan:

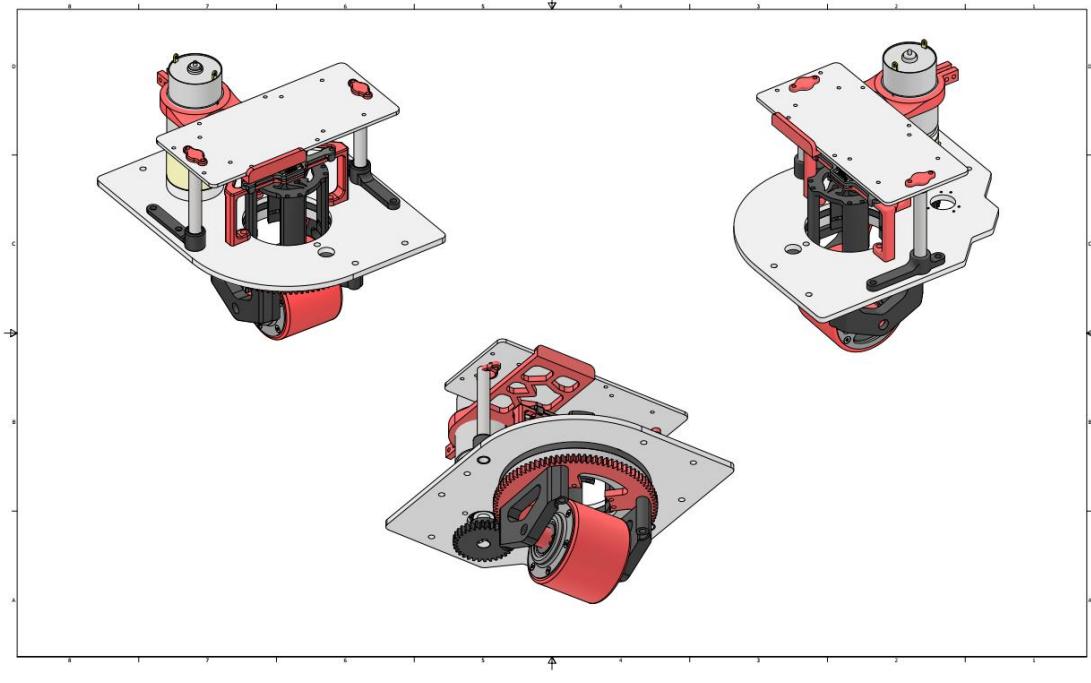
- Berat : 24.75 kg
- Panjang : 650 mm
- Lebar : 650 mm
- Tinggi : 455 mm
- Bahan : Aluminium, Stainless, PLA+



(a)



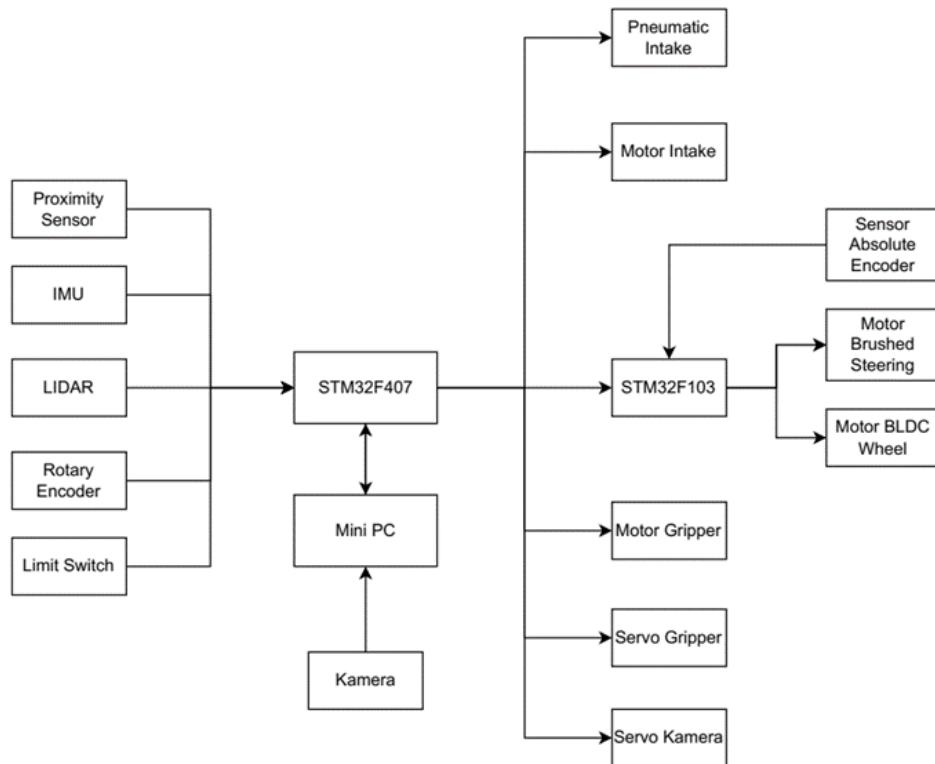
(b)



(c)

Gambar 22. Desain Mekanisme (a) Intake Bola, (b) Gripper Bola, (c) Swerve Drive

2. Sistem Sensor dan Aktuator pada Robot



Gambar 23. Diagram Blok Sistem Sensor dan Aktuator Robot

3. Integrasi Gyroscope dan Accelerometer Menggunakan Mahony Filter

Digunakan perangkat smartphone Xiaomi Redmi Note 10 sebagai sensor IMU dengan bantuan aplikasi Serial Sensor. Perangkat ini menggunakan sensor icm4x6xx untuk gyroscope dan accelerometer. Data sensor gyroscope dan accelerometer akan dikirim ke Mini PC melalui UDP packets. Smartphone disambungkan ke Mini PC menggunakan USB, lalu mengaktifkan USB-Tethering pada smartphone. IP Address pada aplikasi Serial Sensor disamakan dengan IP Address pada network yang digunakan supaya UDP packets bisa terkirim ke Mini PC.

Sinyal keluaran dari sistem IMU umumnya ditandai dengan resolusi sinyal rendah yang rentan terhadap level noise yang tinggi serta bias waktu yang bervariasi. Oleh karena itu, sinyal raw perlu diproses untuk merekonstruksi estimasi orientasi yang lebih akurat dan koreksi kecepatan sudut yang bias melalui algoritma sensor fusion yang sesuai. Penggunaan data akselerasi yang tepat dapat menghindari drift yang disebabkan oleh integrasi numerik dari pembacaan gyroscope. Oleh karena itu, filter Mahony diimplementasikan untuk mengatasi masalah noise pada sinyal gyroscope dan accelerometer.

Pertama, algoritma menormalisasi data accelerometer untuk memastikan bahwa magnitudo vektor percepatan tetap bernilai 1 (menghilangkan pengaruh skala sensor yang bervariasi atau gravitasi). Setelah itu, algoritma menghitung arah referensi dari vektor gravitasi berdasarkan estimasi quaternion orientasi saat ini. Dengan menggunakan arah referensi ini, algoritma menghitung error antara nilai accelerometer yang diukur dan arah gravitasi yang diharapkan berdasarkan orientasi saat ini. Error ini menunjukkan perbedaan antara arah gravitasi yang diukur oleh accelerometer dan yang diestimasi oleh sistem orientasi.

Selanjutnya, algoritma menggunakan kontrol proporsional dan integral untuk menyesuaikan pembacaan gyroscope berdasarkan error ini. Gain proporsional (K_p) merespons error secara langsung, sedangkan gain integral (K_i) mengakumulasi error seiring waktu untuk mengkompensasi drift jangka panjang. Nilai gyroscope yang telah disesuaikan ini kemudian digunakan untuk memperbarui orientasi dengan memodifikasi quaternion yang merepresentasikan orientasi saat ini.

Terakhir, quaternion yang diperbarui dinormalisasi untuk menjaga panjangnya tetap 1, sehingga quaternion tetap valid untuk merepresentasikan rotasi. Dari quaternion

ini, algoritma menghitung sudut roll, pitch, dan yaw yang merepresentasikan orientasi sistem dalam ruang 3D. Sudut-sudut ini didapatkan dari komponen quaternion menggunakan fungsi trigonometri. Berikut algoritma filter mahony:

a. Normalisasi Data Sensor Accelerometer

Diberikan pembacaan akselerometer $\mathbf{a} = (a_x, a_y, a_z)$, menormalisasi vektor untuk menghitung vektor satuan sepanjang arah percepatan yang terukur (biasanya gravitasi):

$$|\mathbf{a}| = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

$$\mathbf{a}_{norm} = \left(\frac{a_x}{|\mathbf{a}|}, \frac{a_y}{|\mathbf{a}|}, \frac{a_z}{|\mathbf{a}|} \right)$$

Di mana \mathbf{a}_{norm} adalah vektor akselerometer yang dinormalisasi.

b. Perkiraan Arah Gravitasi

Quaternion $\mathbf{q} = (q_0, q_1, q_2, q_3)$ merepresentasikan estimasi orientasi saat ini. Menggunakan quaternion ini, arah gravitasi yang diestimasi dalam *body frame* dapat dihitung dengan:

$$v_x = 2(q_1 q_3 - q_0 q_2)$$

$$v_y = 2(q_0 q_1 + q_2 q_3)$$

$$v_z = q_0^2 - 0.5 + q_3^2$$

Di sini, v_x, v_y, v_z merepresentasikan arah gravitasi yang diestimasi berdasarkan quaternion saat ini.

c. Komputasi Error (Umpang Balik Proporsional)

Error $\mathbf{e} = (e_x, e_y, e_z)$ antara arah gravitasi yang diukur dan diestimasi dihitung menggunakan perkalian silang antara vektor akselerometer yang dinormalisasi dan arah gravitasi yang diestimasi:

$$e_x = a_y v_z - a_z v_y$$

$$e_y = a_z v_x - a_x v_z$$

$$e_z = a_x v_y - a_y v_x$$

Error ini merepresentasikan perbedaan antara vektor gravitasi yang diukur dan diestimasi.

d. Umpang Balik Integral

Gain integral K_i mengakumulasikan error integral seiring waktu. Misalkan $i = (i_x, i_y, i_z)$ adalah error integral, maka:

$$i_x = i_x + K_i \cdot e_x \cdot \Delta t$$

$$i_y = i_y + K_i \cdot e_y \cdot \Delta t$$

$$i_z = i_z + K_i \cdot e_z \cdot \Delta t$$

e. Koreksi Data Gyroscope (Kompensasi Gyroscope)

Pembacaan gyroscope terkoreksi $\omega_c = (\omega_{cx}, \omega_{cy}, \omega_{cz})$ dihitung dengan menambahkan umpan balik proporsional K_p ke data raw gyroscope $\omega = (\omega_x, \omega_y, \omega_z)$:

$$\omega_{cx} = \omega_x + K_p \cdot e_x + i_x$$

$$\omega_{cy} = \omega_y + K_p \cdot e_y + i_y$$

$$\omega_{cz} = \omega_z + K_p \cdot e_z + i_z$$

f. Pembaruan Quaternion

Quaternion diperbarui menggunakan data gyroscope yang telah dikoreksi. Perubahan pada quaternion $\Delta q = (\Delta q_0, \Delta q_1, \Delta q_2, \Delta q_3)$ akibat kecepatan sudut adalah sebagai berikut:

$$\Delta q_0 = -0.5 \cdot (q_1 \omega_{cx} + q_2 \omega_{cy} + q_3 \omega_{cz}) \cdot \Delta t$$

$$\Delta q_1 = 0.5 \cdot (q_0 \omega_{cx} + q_2 \omega_{cz} - q_3 \omega_{cy}) \cdot \Delta t$$

$$\Delta q_2 = 0.5 \cdot (q_0 \omega_{cy} - q_1 \omega_{cz} + q_3 \omega_{cx}) \cdot \Delta t$$

$$\Delta q_3 = 0.5 \cdot (q_0 \omega_{cz} + q_1 \omega_{cy} - q_2 \omega_{cx}) \cdot \Delta t$$

Quaternion kemudian diperbarui dengan menambahkan perubahan ke quaternion saat ini:

$$q_0 = q_0 + \Delta q_0$$

$$q_1 = q_1 + \Delta q_1$$

$$q_2 = q_2 + \Delta q_2$$

$$q_3 = q_3 + \Delta q_3$$

g. Normalisasi Quaternion

Untuk memastikan quaternion tetap ternormalisasi, maka dihitung magnitudo quaternion:

$$|q| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}$$

Lalu, normalisasi quaternion:

$$q_0 = \frac{q_0}{|q|}, q_1 = \frac{q_1}{|q|}, q_2 = \frac{q_2}{|q|}, q_3 = \frac{q_3}{|q|}$$

h. Konversi Sudut Euler

Untuk menghitung roll, pitch, and yaw (sudut Euler) dari quaternion:

Roll (ϕ)

$$\phi = \text{atan} 2(2(q_0q_1 + q_2q_3), 1 - 2(q_1^2 + q_2^2))$$

Pitch (θ):

$$\theta = \text{asin}(2(q_0q_2 - q_1q_3))$$

Yaw (ψ):

$$\psi = \text{atan} 2((2(q_1q_2 + q_0q_3), 1 - 2(q_2^2 + q_3^2))$$

Alasan tidak memakai magnetometer pada pengimplementasian mahony filter karena magnetometer memberikan pembacaan yang sangat acak dan tidak dapat diandalkan. Kami berasumsi bahwa magnetometer memperoleh beberapa interferensi. Pembacaan berubah ketika motor berjalan, dan servo bergerak. Kami berasumsi bahwa fluks magnet dari motor dan servo mengganggu magnetometer. Selain itu, saat dilakukan eksperimen yang sama di tempat yang berbeda di ruangan, atau pada waktu yang berbeda, diperoleh pembacaan magnetometer yang berbeda.

4. Kalibrasi Gyroscope

Kalibrasi gyroscope dengan meletakkan ponsel di permukaan datar, seperti meja, merupakan metode untuk mengoreksi bias sensor dan meningkatkan akurasi pembacaan gyroscope. Ketika ponsel diletakkan di permukaan datar dan stabil, maka seharusnya tidak ada rotasi atau gerakan nyata. Proses kalibrasi ini bertujuan untuk mengidentifikasi dan menghilangkan kesalahan sistematis (bias) pada keluaran sensor, sehingga saat ponsel diam, gyroscope tidak mendekripsi adanya gerakan. Gyroscope mengukur kecepatan sudut di sekitar sumbu x, y, dan z. Jika ponsel dalam keadaan diam, kecepatan sudut seharusnya nol pada semua sumbu. Kalibrasi ini bertujuan untuk menghilangkan bias atau drift yang terjadi bahkan ketika ponsel tidak bergerak. Langkah-langkah kalibrasi gyroscope:

- a. Pertama dengan menginisialisasi nilai bias gyroscope untuk sumbu x, y, dan z menjadi nol. Pastikan ponsel tetap diam selama proses kalibrasi.
- b. Secara terus-menerus dilakukan pembacaan data raw gyroscope selama 15 hingga 20 detik. Data gyroscope terdiri dari tiga komponen: kecepatan sudut gx , gy dan gz untuk sumbu x, y, dan z masing-masing.

- c. Karena ponsel dalam keadaan diam, seharusnya tidak ada gerakan. Oleh karena itu, setiap pembacaan yang bukan nol dari gyroscope dianggap sebagai bias atau drift dari sensor. Pembacaan gyroscope dikumpulkan untuk menghitung rata-rata nilai untuk setiap sumbu. Rata-rata ini akan merepresentasikan bias gyroscope. Rumus untuk menghitung bias pada masing-masing sumbu adalah:

$$bias_x = \frac{1}{N} \sum_{i=1}^N gx_i$$

$$bias_y = \frac{1}{N} \sum_{i=1}^N gy_i$$

$$bias_z = \frac{1}{N} \sum_{i=1}^N gz_i$$

Di mana N adalah jumlah total sampel gyroscope yang dikumpulkan, dan gx_i, gy_i, gz_i adalah pembacaan gyroscope pada sampel ke-i.

- d. Setelah menghitung rata-rata pembacaan gyroscope, nilai rata-rata yang dihasilkan untuk setiap sumbu merepresentasikan bias atau drift gyroscope. Nilai-nilai ini merupakan kesalahan sistematis yang perlu dikoreksi. Bias ini akan digunakan untuk mengimbangi pembacaan data raw gyroscope seterusnya.
- e. Setelah bias terdeteksi, bias tersebut dikurangi dari pembacaan data raw gyroscope untuk mendapatkan kecepatan sudut yang sebenarnya. Rumus untuk pembacaan gyroscope yang telah dikoreksi adalah:

$$corrected_{gx} = gx_{raw} - bias_x$$

$$corrected_{gy} = gy_{raw} - bias_y$$

$$corrected_{gz} = gz_{raw} - bias_z$$

Ini memastikan bahwa ketika ponsel dalam keadaan diam, pembacaan gyroscope yang telah dikoreksi akan mendekati nol, sehingga menghilangkan drift.

5. Kalibrasi Accelerometer

Kalibrasi akselerometer dengan memutar ponsel secara perlahan di sekitar ketiga sumbunya (x, y, z) dirancang untuk mengoreksi kesalahan seperti bias,

ketidakakuratan faktor skala, dan ketidaksesuaian sumbu. Metode ini membantu memastikan bahwa akselerometer dapat mengukur gaya yang bekerja pada perangkat, terutama gaya gravitasi, di seluruh sumbu (x, y, z) secara akurat. Akselerometer seharusnya mengukur gaya gravitasi sebesar 1g ketika ponsel dalam keadaan diam dan sejajar sempurna dengan salah satu sumbu (baik arah positif atau negatif).

Namun, karena ketidak sempurnaan sensor, pengukuran mungkin menyimpang akibat bias (offset konstan pada setiap sumbu meskipun seharusnya tidak ada percepatan), kesalahan faktor skala (pengukuran mungkin lebih besar atau lebih kecil dari yang diharapkan secara proporsional), dan *cross-axis sensitivity* atau ketidaksesuaian (sumbu mungkin tidak benar-benar ortogonal, menyebabkan pengukuran yang salah saat diputar). Berikut Langkah-langkah kalibrasi akselerometer:

- a. Memutar ponsel secara perlahan satu putaran penuh di sekitar setiap sumbu (x, y, z). Ini memungkinkan accelerometer untuk mengukur seluruh rentang percepatan yang disebabkan oleh orientasi ponsel relatif terhadap gravitasi. Kumpulkan pembacaan accelerometer selama proses rotasi dengan merekam nilai accelerometer mentah ax_{raw} , ay_{raw} , az_{raw} .
- b. Setelah mengumpulkan data untuk rotasi penuh pada setiap sumbu, menentukan bias (offset konstan) untuk setiap sumbu dengan mengambil rata-rata pembacaan saat ponsel diam (atau hampir diam). Bias untuk setiap sumbu dapat dihitung sebagai:

$$bias_x = \frac{1}{N} \sum_{i=1}^N ax_{raw,i}$$

$$bias_y = \frac{1}{N} \sum_{i=1}^N ay_{raw,i}$$

$$bias_z = \frac{1}{N} \sum_{i=1}^N az_{raw,i}$$

Di mana N adalah jumlah sampel yang dikumpulkan saat ponsel diam, dan ax_{raw} , ay_{raw} , az_{raw} adalah pembacaan accelerometer mentah. Bias yang dihitung digunakan untuk menggeser nilai accelerometer mentah sehingga saat tidak ada gerakan, keluaran accelerometer menjadi nol.

- c. Langkah selanjutnya adalah menghitung faktor skala untuk setiap sumbu. Saat ponsel sejajar sempurna dengan salah satu sumbu (misalnya sumbu x), accelerometer seharusnya mengukur $\pm 1g$ (gravitasi). Jika pembacaan mentah

menyimpang dari nilai yang diharapkan, maka faktor skala diterapkan untuk mengoreksi pembacaan tersebut. Untuk setiap sumbu (x, y, dan z), ponsel diputar sehingga sumbu yang dimaksud mengarah ke atas dan ke bawah. Pembacaan maksimum dan minimum pada setiap sumbu seharusnya sesuai dengan +1g dan -1g masing-masing. Data ini digunakan untuk menghitung faktor skala untuk setiap sumbu:

$$scale_x = \frac{2g}{|ax_{max} - ax_{min}|}$$

$$scale_y = \frac{2g}{|ay_{max} - ay_{min}|}$$

$$scale_z = \frac{2g}{|az_{max} - az_{min}|}$$

Faktor skala ini memastikan bahwa keluaran dari setiap sumbu benar-benar sesuai dengan $\pm 1g$ saat sejajar dengan gravitasi.

- d. Setelah bias dan faktor skala telah ditentukan, pembacaan accelerometer seterusnya akan dikoreksi dengan menerapkan nilai-nilai ini. Nilai accelerometer yang telah dikoreksi adalah:

$$ax_{corrected} = (ax_{raw} - bias_x) \times scale_x$$

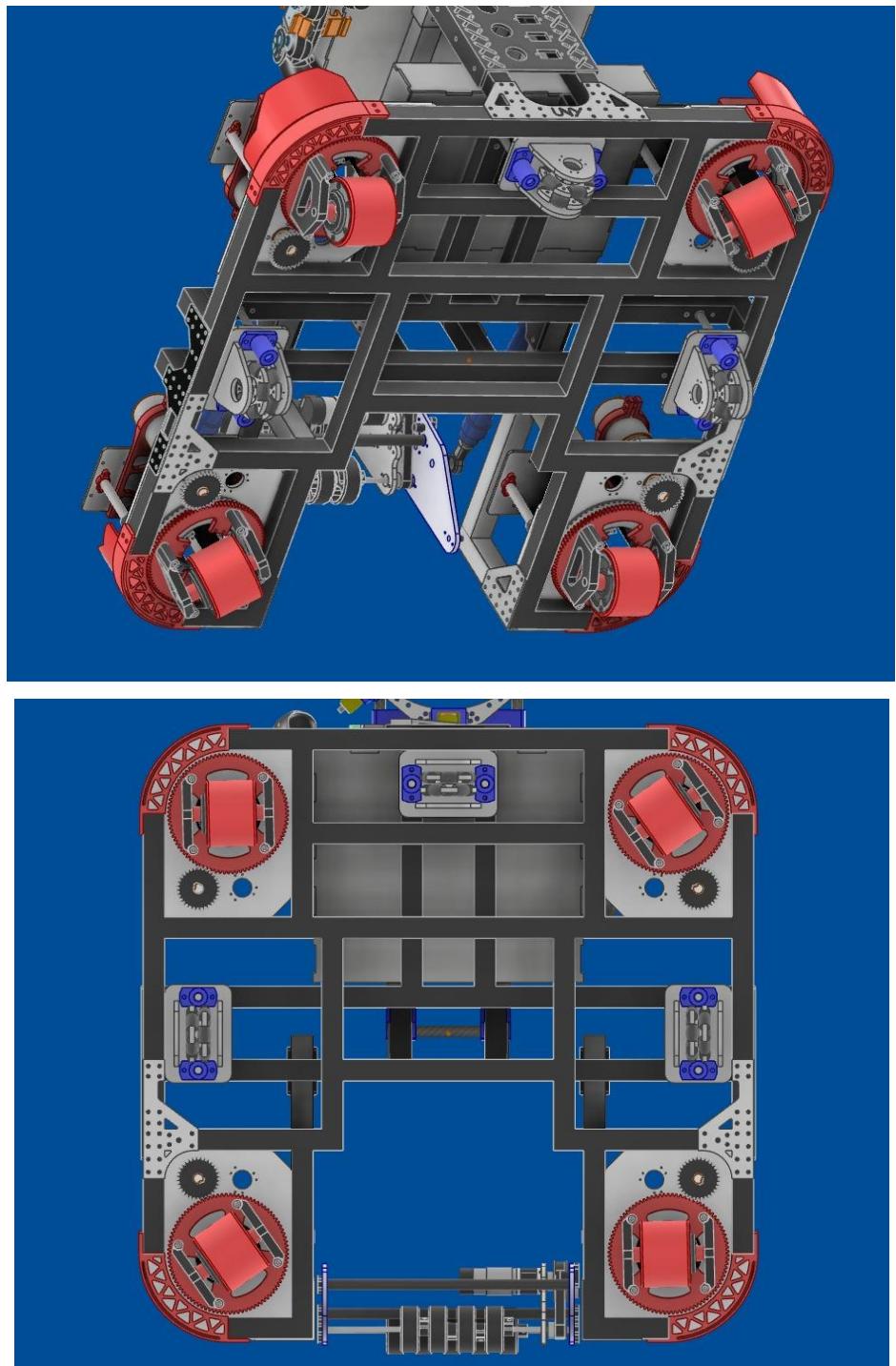
$$ay_{corrected} = (ay_{raw} - bias_y) \times scale_y$$

$$az_{corrected} = (az_{raw} - bias_z) \times scale_z$$

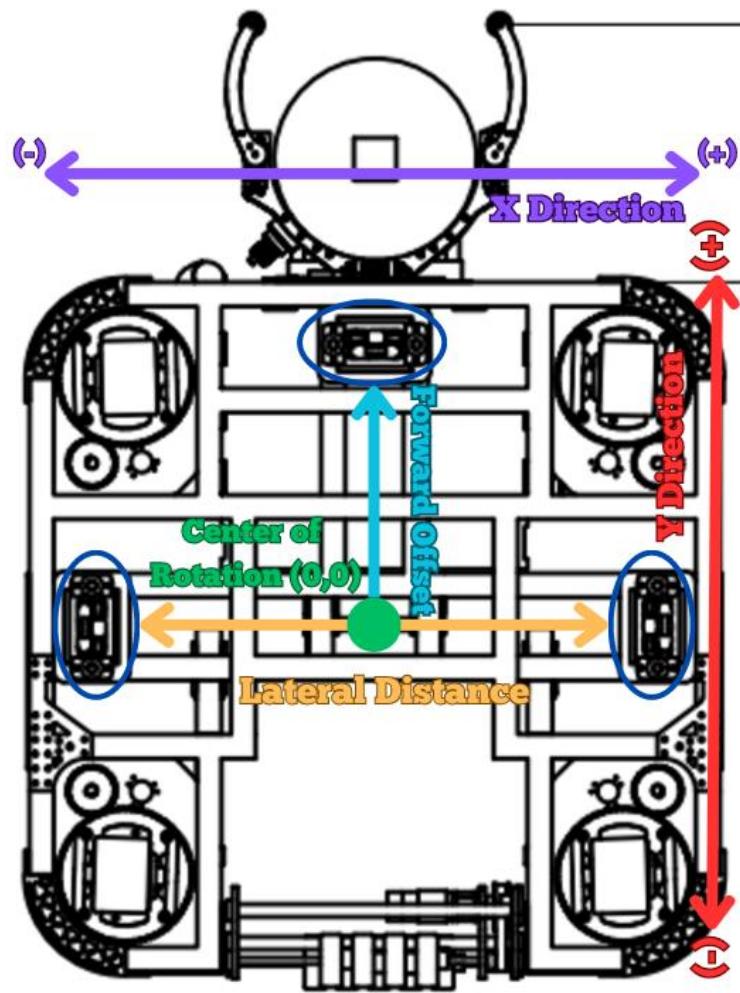
Dengan memutar ponsel dan mengumpulkan data, kita mengekspos accelerometer ke seluruh rentang percepatan yang mungkin (terutama karena gravitasi). Ini memungkinkan kita untuk mengoreksi offset konstan (bias) dan kesalahan proporsional (faktor skala) dalam pembacaan sensor. Metode ini, yang dilakukan dengan memutar ponsel secara perlahan, memastikan bahwa kalibrasi memperhitungkan penggunaan di dunia nyata, di mana ponsel akan mengalami gaya dari berbagai arah.

6. Odometry

a. Desain Three-Wheel Odometry pada Base Robot



Gambar 24. Desain Odometri Tiga Roda Pada Base Robot



Gambar 25. Desain Odometri Tiga Roda Pada Base Robot (Robot Tampak Bawah)



Gambar 26. Desain Dead Wheel Untuk Odometri

b. Sistem Three-Wheel Odometry

Implementasi odometri ini menggunakan data dari incremental encoder pada roda dan IMU (Inertial Measurement Unit) untuk melacak posisi dan orientasi robot dari waktu ke waktu. Odometri membantu menentukan pergerakan robot dengan menghitung perubahan posisi (x, y) dan arah (θ) berdasarkan data rotasi roda. Diketahui:

- Jari-jari roda atau $radius_{wheel} = 29.6028 \text{ mm}$
- Pulse Per Revolution encoder atau $PPR_{encoder} = 800$
- Jarak dari tengah robot ke tengah roda odometry atau $forward_offset = 274.5 \text{ mm}$
- Jarak antara roda odometry kiri dan roda odometry kanan atau $trackwidth = 537.7 \text{ mm}$

Pseudocode:

```
FUNCTION Odometry(pulse_kiri, pulse_kanan, pulse_belakang, sudut):
    CONSTANT constant = 2 * PI * radius_roda / encoder_ppr

    # Calculate tick differences from previous readings
    delta_left_ticks = pulse_kiri - last_left_ticks
    delta_right_ticks = pulse_kanan - last_right_ticks
    delta_center_ticks = pulse_belakang - last_center_ticks

    # Convert the IMU angle (in degrees) to radians, adjusting for 90-degree offset
    sudut_imu = DegreesToRadians(sudut - 90)

    # Update the current heading (theta) based on IMU reading
    theta_now = sudut

    # Calculate changes in x, y, and heading based on encoder readings
    delta_x = constant * (delta_left_ticks + delta_right_ticks) / 2
    delta_theta = constant * (delta_right_ticks - delta_left_ticks) / trackwidth
    delta_y = constant * (delta_center_ticks - (delta_right_ticks - delta_left_ticks)
    * forward_offset / trackwidth)
```

```

# Store the changes for debugging or further use
x_delta = delta_x
y_delta = delta_y
heading_delta = sudut - last_heading

# Calculate the updated global orientation
theta = sudut_global + (delta_theta / 2)

# Update global x and y positions using rotation and translation
x_global += delta_x * cos(sudut_imu) - delta_y * sin(sudut_imu)
y_global += delta_x * sin(sudut_imu) + delta_y * cos(sudut_imu)

# Update global orientation based on delta_theta
sudut_global += delta_theta

# Save current readings for use in the next cycle
last_left_ticks = pulse_kiri
last_right_ticks = pulse_kanan
last_center_ticks = pulse_belakang
last_heading = sudut

```

END FUNCTION

Pertama, kita menghitung keliling roda dan resolusi encoder yang digunakan. Keliling roda C_{wheel} dihitung menggunakan rumus keliling lingkaran:

$$C_{wheel} = 2\pi \cdot radius_{wheel}$$

Ini mewakili jarak yang ditempuh oleh roda dalam satu putaran penuh. Lalu, resolusi encoder per pulsa $C_{perPulse}$ atau jarak yang ditempuh oleh roda untuk setiap pulsa encoder:

$$C_{perPulse} = \frac{C_{wheel}}{PPR_{encoder}}$$

Perubahan posisi robot (x, y) dan perubahan orientasi (θ) dapat dihitung berdasarkan data yang diperoleh dari encoder. Misalkan:

- ΔL : Jarak yang ditempuh oleh roda kiri.
- ΔR : Jarak yang ditempuh oleh roda kanan.
- ΔC : Jarak yang ditempuh oleh roda tengah (belakang).

Perubahan dalam pulsa (*ticks*) yang diukur oleh encoder sejak pembacaan terakhir dihitung dengan mengurangi nilai pulsa sebelumnya:

$$\begin{aligned}\Delta L_{\text{ticks}} &= \text{pulse}_{\text{left}} - \text{last_left_pulse} \\ \Delta R_{\text{ticks}} &= \text{pulse}_{\text{right}} - \text{last_right_pulse} \\ \Delta C_{\text{ticks}} &= \text{pulse}_{\text{center}} - \text{last_center_pulse}\end{aligned}$$

Tanda delta mewakili perubahan pada rotasi roda sejak pembacaan terakhir. Pulsa-pulsa ini kemudian dikonversi menjadi jarak dengan menggunakan resolusi encoder:

$$\begin{aligned}\Delta L &= \Delta L_{\text{ticks}} \cdot C_{\text{perPulse}} \\ \Delta R &= \Delta R_{\text{ticks}} \cdot C_{\text{perPulse}} \\ \Delta C &= \Delta C_{\text{ticks}} \cdot C_{\text{perPulse}}\end{aligned}$$

Perubahan orientasi robot (yaw) dihitung dari perbedaan gerakan roda kiri dan kanan:

$$\Delta\theta = \frac{\Delta R - \Delta L}{\text{trackwidth}}$$

Jarak maju (*forward*) yang ditempuh oleh robot adalah rata-rata jarak yang ditempuh oleh roda kiri dan kanan:

$$\Delta x = \frac{\Delta L + \Delta R}{2}$$

Gerakan lateral (samping), berdasarkan data dari encoder roda tengah, dihitung sebagai:

$$\Delta y = \Delta C - \frac{(\Delta R - \Delta L) \cdot \text{forward_offset}}{\text{tarckwidth}}$$

Term kedua memperhitungkan gerakan lateral akibat robot berbelok.

Orientasi baru robot adalah:

$$\theta = \theta_{\text{global}} + \frac{\Delta\theta}{2}$$

Posisi global baru (x_{global} , y_{global}) dihitung dengan mentransformasikan gerakan lokal (Δx , Δy) ke dalam kerangka referensi global menggunakan sudut orientasi robot:

$$x_{\text{global}} = x_{\text{global}} + \Delta x \cdot \cos \theta - \Delta y \cdot \sin \theta$$

$$y_{global} = y_{global} + \Delta x \cdot \sin \theta + \Delta y \cdot \cos \theta$$

Orientasi atau arah global robot diperbarui sebagai:

$$\theta_{global} = \theta_{global} + \Delta \theta$$

Dengan metode odometry ini, robot dapat memperbarui posisi dan orientasinya secara terus-menerus berdasarkan data encoder. Pergerakan pada arah x dan y disesuaikan dengan orientasi robot untuk mentransformasi pergerakan lokal ke dalam koordinat global, sehingga menghasilkan estimasi posisi yang akurat.

7. Integrasi Sudut Yaw dari IMU (Gyroscope-Accelerometer) dan Odometry

Integrasi odometri dan IMU memungkinkan sistem lokalisasi robot untuk menghitung posisi dan orientasi dengan lebih akurat. Odometri digunakan untuk menghitung perubahan posisi robot berdasarkan pergerakan roda, sementara IMU memberikan orientasi sudut (yaw) yang lebih akurat untuk memperbaiki error akibat drift. Kombinasi ini memungkinkan robot untuk bergerak dan bernavigasi secara presisi di lingkungan dinamis. Seperti yang telah diketahui yaw dari robot diperoleh dari quaternion kombinasi gyroscope dan accelerometer:

Yaw (ψ):

$$\psi = \theta_{IMU} = \text{atan} 2((2(q_1 q_2 + q_0 q_3), 1 - 2(q_2^2 + q_3^2)))$$

Sudut yang diperoleh tersebut dikonversi ke radian sebelum diintegrasikan ke sistem odometri. Sudut ini digunakan dalam persamaan trigonometri untuk menghitung posisi global robot dalam koordinat x dan y:

$$x_{global} = x_{global} + \Delta x \cdot \cos \theta_{IMU} - \Delta y \cdot \sin \theta_{IMU}$$

$$y_{global} = y_{global} + \Delta x \cdot \sin \theta_{IMU} + \Delta y \cdot \cos \theta_{IMU}$$

Dengan cara ini, perubahan posisi robot dihitung tidak hanya berdasarkan data encoder, tetapi juga berdasarkan orientasi yang diberikan oleh IMU. Jika terjadi drift pada odometri, data dari IMU akan membantu menjaga orientasi robot tetap akurat.

8. Kecepatan Robot Berdasarkan Odometri

Pseudocode:

```
FUNCTION UpdateSpeed()
```

```

// Get the current time in milliseconds
currentTime = GlobalTimer.Timer.GetTimerMillis()

// Check if 50 milliseconds have passed since the last update
IF (currentTime - timestamp < 50) THEN
    RETURN

// Calculate the differences in position and orientation
delta_x = ABS(x_global - x_global_last)
delta_y = ABS(y_global - y_global_last)
delta_heading = ABS(theta_now - theta_last)

// Calculate elapsed time in milliseconds
time = currentTime - timestamp
// Calculate speeds in mm/s
x_speed = (delta_x * 1000) / time
y_speed = (delta_y * 1000) / time
theta_speed = (delta_heading * 1000) / time

// Calculate the overall speed in mm/s
speed = SQRT(POWER(x_speed, 2) + POWER(y_speed, 2))

// Update last speed for future reference
last_speed = speed

// Update timestamp and last positions
timestamp = currentTime
x_global_last = x_global
y_global_last = y_global
theta_last = theta_now

END FUNCTION

```

Fungsi “*UpdateSpeed*” ini bertujuan untuk memperbarui kecepatan robot berdasarkan perubahan posisi dan orientasinya dalam selang waktu tertentu. Pertama, fungsi ini mengambil waktu saat ini dalam satuan milidetik. Kemudian, dilakukan pengecekan apakah sudah 50 milidetik berlalu sejak pembaruan terakhir. Jika belum, fungsi akan dihentikan dan tidak ada perhitungan lebih lanjut.

Selanjutnya, fungsi menghitung perubahan posisi pada sumbu x (*delta_x*), sumbu y (*delta_y*), serta perubahan orientasi atau sudut (*delta_heading*) dengan mengambil selisih antara posisi saat ini dan posisi sebelumnya. Setelah itu, waktu yang telah berlalu sejak pembaruan terakhir dihitung. Dengan menggunakan perbedaan posisi dan waktu yang dihitung, fungsi ini menentukan kecepatan robot pada sumbu x, sumbu y, dan kecepatan rotasinya dalam satuan milimeter per detik (mm/s). Kecepatan keseluruhan dihitung menggunakan rumus Pythagoras dengan menggabungkan kecepatan pada sumbu x dan y. Setelah menghitung kecepatan, fungsi memperbarui nilai *last_speed* dengan kecepatan terbaru, lalu memperbarui *timestamp* dan menyimpan posisi serta orientasi saat ini untuk digunakan pada pembaruan berikutnya. Dengan demikian, fungsi ini memberikan informasi akurat tentang kecepatan robot dalam setiap interval waktu yang cukup.

9. Kinematika Swerve Drive

Pseudocode:

```
FUNCTION SwerveDriveMove(speed, heading_now, angle_offset, forward, strafe, rotation)
    IF forced_stop THEN
        StopAllSwerve()
        RETURN

        // Assign input values to local variables
        vx = forward // Forward: positive for forward, negative for backward
        vy = -strafe // Strafe: positive for left, negative for right
        vw = -rotation // Rotation: negative for clockwise

        // Robot dimensions (for simplicity, assume rx = ry = 1.0)
        rx = 1.0
```

```

ry = 1.0

// Convert heading angle from degrees to radians
current_angle_rad = DegreesToRadians(heading_now)

// Adjust vx and vy based on the robot's current heading
adjust_vx = vx * cos(current_angle_rad) + vy * sin(current_angle_rad)
adjust_vy = -vx * sin(current_angle_rad) + vy * cos(current_angle_rad)

// Save adjusted velocities for future use
fwd = adjust_vx
str = adjust_vy
rcw = vw

// Calculate velocities for each wheel based on swerve drive kinematics
v1x = adjust_vx - vw * ry
v1y = adjust_vy + vw * rx

v2x = adjust_vx - vw * (-ry)
v2y = adjust_vy + vw * rx

v3x = adjust_vx - vw * ry
v3y = adjust_vy + vw * -rx

v4x = adjust_vx - vw * (-ry)
v4y = adjust_vy + vw * -rx

// Calculate speed (magnitude) for each wheel using Pythagorean theorem
v1 = sqrt(v1x^2 + v1y^2)
v2 = sqrt(v2x^2 + v2y^2)
v3 = sqrt(v3x^2 + v3y^2)
v4 = sqrt(v4x^2 + v4y^2)

```

```

// Normalize wheel speeds if any exceeds 1.0
max_speed = max(v1, v2, v3, v4)

IF max_speed > 1 THEN
    v1 = v1 / max_speed
    v2 = v2 / max_speed
    v3 = v3 / max_speed
    v4 = v4 / max_speed

// Calculate wheel angles (theta) convert radians to degrees
t1 = AngleModulo(RadiansToDegrees(atan2(v1y, v1x)) + angle_offset)
t2 = AngleModulo(RadiansToDegrees(atan2(v2y, v2x)) + angle_offset)
t3 = AngleModulo(RadiansToDegrees(atan2(v3y, v3x)) + angle_offset)
t4 = AngleModulo(RadiansToDegrees(atan2(v4y, v4x)) + angle_offset)

// If there is no movement, maintain the current wheel angles
IF forward == 0.0 AND strafe == 0.0 AND rotation == 0.0 THEN
    SwerveKiriDepan.SetAngle(SwerveKiriDepan.GetAngle())
    SwerveKiriBelakang.SetAngle(SwerveKananDepan.GetAngle())
    SwerveKananBelakang.SetAngle(SwerveKananBelakang.GetAngle())
    SwerveKananDepan.SetAngle(SwerveKiriBelakang.GetAngle())

ELSE
    // Set wheel speeds and angles
    SwerveKiriDepan.SetSpeed(v1 * speed)
    SwerveKananDepan.SetSpeed(v2 * speed)
    SwerveKiriBelakang.SetSpeed(v3 * speed)
    SwerveKananBelakang.SetSpeed(v4 * speed)

    // Set the angle of each wheel to the calculated angles
    SwerveKiriDepan.SetAngle(t1)
    SwerveKananDepan.SetAngle(t2)
    SwerveKiriBelakang.SetAngle(t3)
    SwerveKananBelakang.SetAngle(t4)

END IF

```

```
END FUNCTION
```

Fungsi “*SwerveDriveMove*” menjelaskan kinematika swerve drive menggunakan beberapa konsep matematika dasar, seperti transformasi vektor dan trigonometri untuk menghitung sudut roda. Proses pertama adalah transformasi vektor kecepatan berdasarkan heading robot saat ini. Input kecepatan maju (v_x), strafe (v_y), dan rotasi (v_ω) diterima dan diubah sesuai dengan sudut heading robot. Transformasi ini dilakukan menggunakan rotasi matriks dengan fungsi kosinus dan sinus untuk menghasilkan kecepatan yang telah disesuaikan.

Selanjutnya, kecepatan translasi dan rotasi ini digunakan untuk menghitung kecepatan pada masing-masing dari empat roda (roda kiri depan, kanan depan, kiri belakang, kanan belakang). Kecepatan pada setiap roda dihitung berdasarkan kecepatan translasi yang sudah disesuaikan ($adjust_v_x$, $adjust_v_y$) dan kecepatan rotasi (v_ω). Untuk setiap roda, v_{1x} , v_{1y} adalah komponen kecepatan pada sumbu x dan y, dengan r_x dan r_y adalah panjang robot pada sumbu x dan y (disini disederhanakan sebagai 1). Perhitungan ini diterapkan pada empat roda dengan posisi yang berbeda-beda.

Kecepatan total (magnitudo) pada setiap roda dihitung dengan teorema Pythagoras untuk menghitung kecepatan aktual tiap roda berdasarkan perpaduan antara translasi dan rotasi. Jika salah satu dari kecepatan roda melebihi batas maksimum (1.0), semua kecepatan dinormalisasi dengan membagi semua nilai kecepatan dengan nilai maksimum untuk menjaga proporsi antar roda tetap sama.

Kemudian, sudut roda (theta) dihitung dengan menggunakan fungsi arc-tangent dua variabel (atan2) yang mengembalikan sudut dalam radian dari komponen x dan y roda, lalu dikonversi ke derajat dan ditambahkan dengan offset sudut. Ini diterapkan pada setiap roda untuk menghitung sudut yang sesuai berdasarkan kecepatan yang dituju oleh roda tersebut.

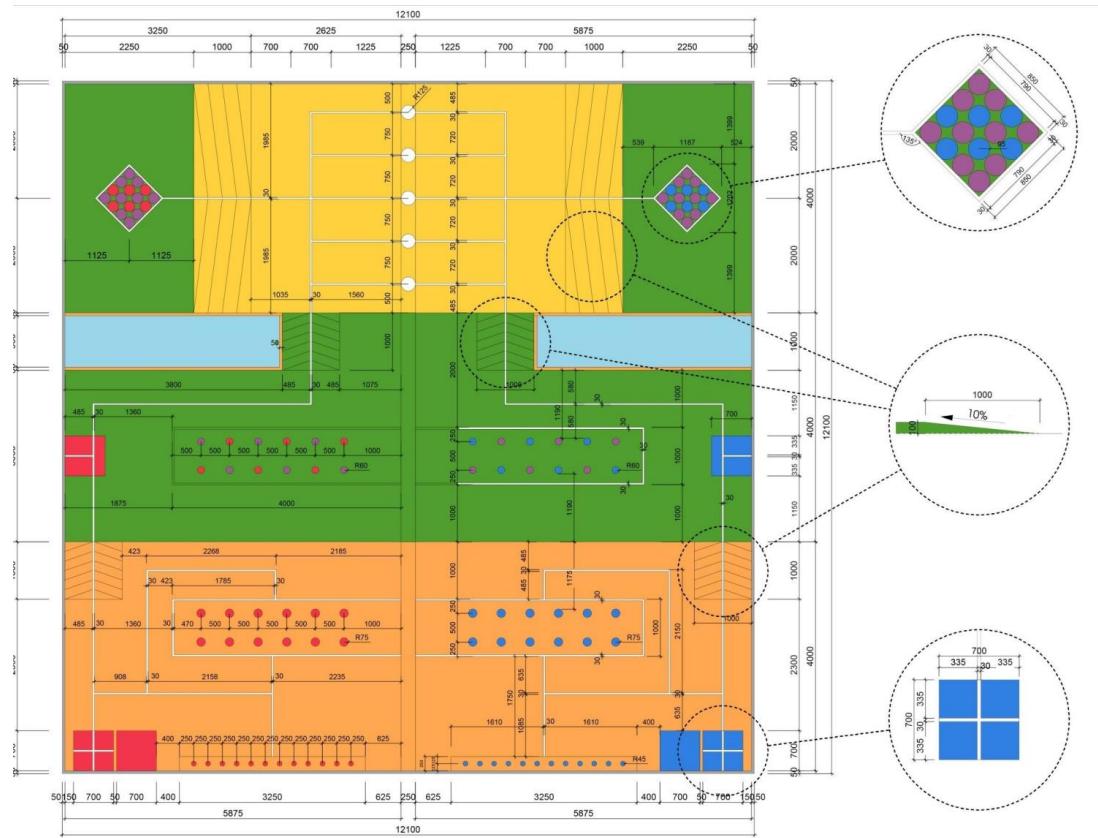
Terakhir, kecepatan yang telah dihitung untuk setiap roda dikalikan dengan faktor *speed* dan diatur untuk roda masing-masing. Jika robot tidak bergerak (kecepatan maju, strafe, dan rotasi semuanya nol), sudut roda tidak berubah, tetapi jika ada gerakan, sudut roda diperbarui sesuai dengan perhitungan tersebut. Dengan demikian, kinematika swerve drive ini mampu mengubah kecepatan translasi dan rotasi robot menjadi kecepatan dan sudut individu pada setiap roda, memastikan kontrol yang presisi dalam navigasi.

10. Path Planning Menggunakan Algoritma A*

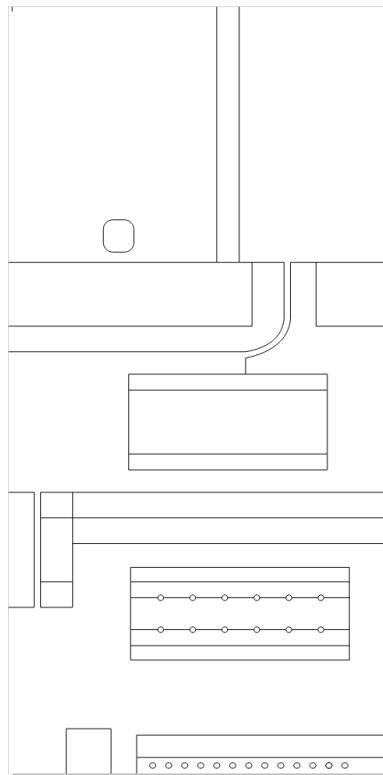
a. Desain ABU ROBOCON 2024 Game Field

Desain track yang akan dilalui robot dilakukan menggunakan CorelDRAW.

Dimensi lapangan permainan yang didesain akan berdasarkan pada rulebook Abu Robocon 2024. Desain track yang akan dilalui robot dilakukan menggunakan CorelDRAW. Obstacle atau penghalang direpresentasikan sebagai garis hitam, sedangkan jalur yang dapat dilalui robot direpresentasikan sebagai area putih.



Gambar 27. Lapangan Permainan Abu Robocon 2024



Gambar 28. Desain Lapangan Menggunakan CorelDRAW untuk Pencarian Jalur A*

b. Pencarian Jalur

Tahap awal pengujian sistem dimulai dengan membangun simulasi algoritma A*. Hasil simulasi A* dinamis menunjukkan proses pencarian jalur terdekat, dimulai dengan informasi global berupa koordinat awal robot di titik S (550, 350) dan koordinat akhir yang harus dicapai di titik G (4300, 10000). Pada tahap awal ini, robot belum mengetahui apakah ada halangan di jalur menuju titik tujuan.

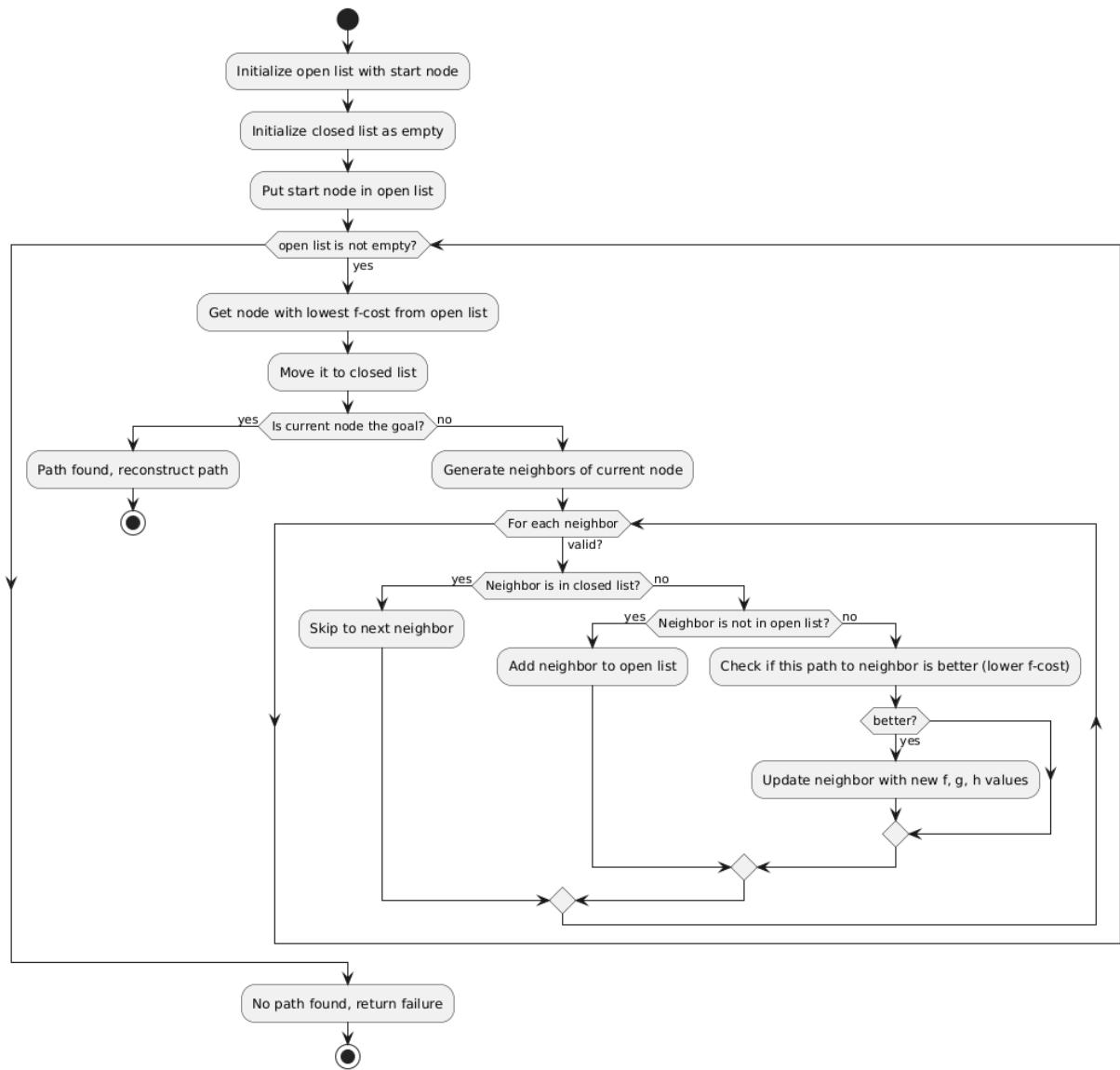
Pergerakan robot terhadap penghalang ditandai dengan warna merah, sementara penghalang yang belum diketahui robot berwarna hitam. Robot kemudian bergerak mengikuti jalur yang ada. Pada langkah berikutnya, informasi lokal robot mengidentifikasi adanya penghalang yang menghalangi robot mencapai titik goal. Robot pun melakukan perhitungan ulang untuk menentukan jalur terdekat yang memungkinkan.

Setelah jalur baru ditemukan, robot melanjutkan perjalanan menuju titik goal. Proses ini terus berulang karena robot membawa misi untuk menghindari halangan, dan penghalang baru hanya terdeteksi saat robot menelusuri jalurnya.

Setiap kali penghalang ditemukan, robot selalu melakukan perhitungan ulang untuk mencari jarak terdekat menuju finish.

Pseudocode:

```
Initialize:  
    - Set OPEN to an empty priority queue (for nodes to explore).  
    - Set CLOSED to an empty list (for nodes already explored).  
    - Set  $g(s_{\text{start}}) = 0$  (cost from start to start is zero).  
    - Set  $g(s_{\text{goal}}) = \text{infinity}$  (goal node initially has infinite cost).  
    - Set PARENT( $s_{\text{start}}$ ) =  $s_{\text{start}}$  (start node is its own parent).  
    - Push  $s_{\text{start}}$  into OPEN with priority  $f_{\text{value}}(s_{\text{start}}) = g(s_{\text{start}}) + \text{heuristic}(s_{\text{start}})$ .  
  
While OPEN is not empty:  
    Pop the node  $s$  from OPEN that has the lowest  $f_{\text{value}}$ .  
    Add  $s$  to CLOSED (mark it as explored).  
    If  $s == s_{\text{goal}}$ :  
        The goal has been reached, exit the loop (path found).  
    For each neighbor  $s_n$  of the current node  $s$ :  
        Compute new_cost =  $g(s) + \text{cost}(s, s_n)$  (calculate the cost to reach  $s_n$  from  $s$ ).  
        If  $s_n$  is not in  $g$  (not yet explored):  
            Set  $g(s_n) = \text{infinity}$  (initialize its cost as infinity).  
        If new_cost <  $g(s_n)$  AND there's no collision between  $s$  and  $s_n$ :  
            Update  $g(s_n) = \text{new\_cost}$  (record the new lower cost).  
            Set PARENT( $s_n$ ) =  $s$  (record that  $s$  is the parent of  $s_n$ ).  
            Compute  $f_{\text{value}}(s_n) = g(s_n) + \text{heuristic}(s_n)$ .  
            Push  $s_n$  into OPEN with  $f_{\text{value}}(s_n)$ .  
    return failure // If no path is found  
  
After exiting the loop:  
    If the goal was reached:  
        Extract the path from PARENT starting at  $s_{\text{goal}}$  and tracing back to  $s_{\text{start}}$ .  
        Reduce resolution of path based on specified resolution  
        Return the reduced path and CLOSED (visited nodes).
```



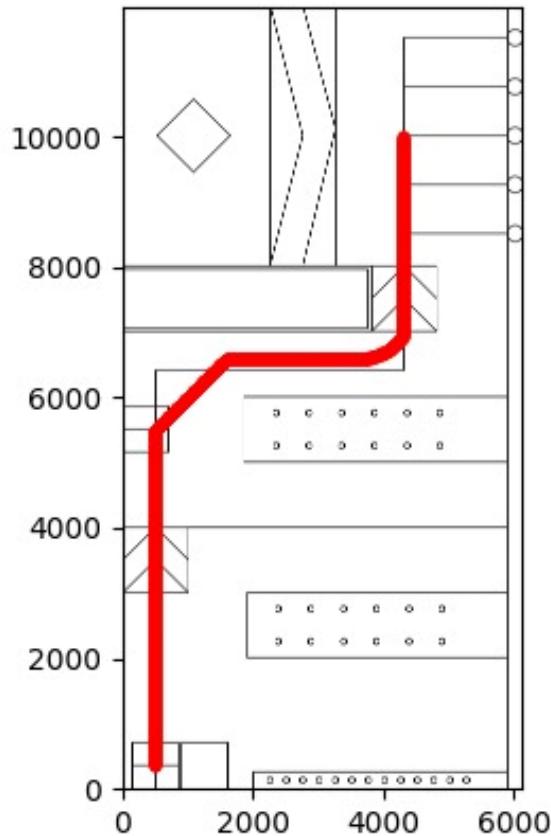
Gambar 29. Diagram Alir Algoritma Pencarian Jalur A*

Algoritma A* yang diimplementasikan di kode ini dirancang untuk mencari jalur terpendek dari titik awal (`s_start`) ke titik tujuan (`s_goal`) dalam sebuah lingkungan berbasis grid yang berisi rintangan. Kelas `AStar` menerima tiga parameter utama, yaitu titik awal, titik tujuan, dan jenis heuristik yang digunakan. Heuristik ini dapat berupa jarak Manhattan (menghitung jarak secara horizontal dan vertikal) atau Euclidean (menghitung jarak garis lurus). Lingkungan (Env) tempat algoritma bekerja ditentukan oleh sebuah peta gambar lapangan game ABU ROBOCON 2024 yang berisi rintangan, dan setiap pergerakan ditentukan oleh set pergerakan yang valid (`u_set`), serta rintangan disimpan dalam variabel `obs`. Algoritma menggunakan dua struktur data utama, yaitu

OPEN dan CLOSED. OPEN adalah priority queue yang menyimpan node yang akan dievaluasi, sedangkan CLOSED adalah daftar node yang sudah dikunjungi.

Dalam fungsi searching(), algoritma dimulai dengan menambahkan node awal ke dalam OPEN dengan nilai prioritas yang dihitung berdasarkan biaya kumulatif g dan estimasi jarak menggunakan heuristik. Setiap iterasi, node dengan nilai prioritas terendah diproses terlebih dahulu, dan jika node tersebut adalah tujuan, proses pencarian berhenti. Algoritma kemudian mengevaluasi tetangga dari node yang diproses, menghitung biaya untuk setiap tetangga, dan memperbarui informasi jalur serta biaya jika ditemukan jalur yang lebih pendek. Fungsi ini juga memeriksa apakah ada rintangan yang menghalangi jalur antara dua node dengan menggunakan fungsi `is_collision()`. Setelah jalur ditemukan, fungsi `reduce_resolution()` digunakan untuk mengurangi jumlah titik dalam jalur, sehingga hanya titik-titik penting yang diambil sesuai resolusi yang diinginkan.

Selain itu, algoritma mendukung pencarian jalur secara berulang dengan parameter e yang bervariasi melalui fungsi `repeated_searching()`, di mana pencarian dilakukan beberapa kali dengan mengurangi nilai e secara bertahap. Nilai e mengatur seberapa agresif algoritma mengejar jalur berdasarkan heuristik. Algoritma juga menyediakan fungsi pendukung lainnya, seperti `get_neighbor()` untuk mendapatkan tetangga dari suatu node, `cost()` untuk menghitung biaya antara dua node, dan `extract_path()` untuk menelusuri kembali jalur dari tujuan ke titik awal berdasarkan node induk yang disimpan dalam dictionary PARENT. Di akhir pencarian, jalur yang ditemukan ditulis ke dalam file teks.



Gambar 30. Hasil Pencarian Jalur A*

11. Path Tracking Menggunakan Pure Pursuit Controllers

Setelah diperoleh koordinat jalur robot dari pencarian menggunakan algoritma A*, maka dilakukan simulasi untuk mengikuti setiap titik koordinat jalur. Digunakan salah satu jenis controller pelacak jalur yaitu Pure Pursuit. Algoritma ini menghitung perintah kecepatan sudut yang menggerakkan robot dari posisinya saat ini untuk mencapai beberapa titik pandang ke depan di depan robot. Perhitungan kecepatan sudut diperlukan agar robot berada tetap berada di jalur yang telah dihitung sebelumnya. Kecepatan linier diasumsikan konstan. Oleh karena itu, pengontrol kecepatan tambahan diperlukan untuk memperlambat robot saat mendekati target. Kami memilih pengontrol PID di sini untuk menangani kecepatan linier robot.

Pseudocode:

Function SearchMinValue(data)

```
Set index to 0
Set minimum_value to data[0]
For i from 0 to length(data):
    If data[i] < minimum_value:
        Set minimum_value to data[i]
        Set index to i
Return index
```

Function SearchTargetIndex(x_now, y_now, velocity, path):

```
Initialize distance as an empty list
```

```
If index equals path.size():
    Set old_nearest_point_index to index
    Return
```

```
If old_nearest_point_index is -1:
```

```
    For each point in path:
        Calculate Euclidean distance to (x_now, y_now)
        Append the distance to the distance list
```

```
    Set index to SearchMinValue(distance)
    Set old_nearest_point_index to index
```

```
Else:
```

```
    Set index to old_nearest_point_index
    Calculate distance_this_index for index
```

```
While True:
```

```
    Calculate distance_next_index for index + 1
    If distance_this_index < distance_next_index, break
    If index + 1 is less than path.size() - 1, increment index
    If index >= path.size() - 1:
        Set index to path.size() - 1
        break
```

```
    Set distance_this_index to distance_next_index
```

```

Set old_nearest_point_index to index

Set Lf = k * velocity + Lfc

While Lf > EuclideanDistance(path[index].x, path[index].y, x_now, y_now):
    If index + 1 >= path.size() - 1, break
    Increment index
    Set old_nearest_point_index to index

Set index2 to index

Function PurePursuitControl(x_now, y_now, velocity, path):
    Call SearchTargetIndex(x_now, y_now, velocity, path)

    If index2 >= index:
        Set index to index2

    If index < path.size():
        Set targetx to path[index].x
        Set targety to path[index].y
    Else:
        Set index to path.size()
        Set targetx to path[path.size() - 1].x
        Set targety to path[path.size() - 1].y

    Create target_point with targetx and target
    Return target_point

```

Keterangan variabel yang digunakan:

- old_nearest_point_index: menyimpan indeks titik terdekat sebelumnya pada jalur.

- index: indeks titik terdekat saat ini.
- index2: indeks sekunder untuk mengontrol indeks target.
- k: gain parameter untuk *look-ahead distance* (jarak pandang ke depan).
- Lfc: *constant look-ahead distance* (jarak pandang ke depan konstan).
- Lf: *dynamic look-ahead distance* (jarak pandang ke depan dinamis).
- velocity: kecepatan robot saat ini.
- path: daftar titik yang mewakili jalur yang diinginkan.

Algoritma ini bekerja dengan mencari titik terdekat pada jalur dan menentukan titik target yang berada pada jarak tertentu di depan robot, yang dikenal sebagai jarak pandang ke depan atau *look-ahead distance*, lalu mengarahkan robot ke titik tersebut. Tujuan dari algoritma ini adalah mengendalikan pergerakan robot agar mengikuti jalur yang telah ditentukan secara akurat dan mulus. Pada awalnya, variabel-variabel seperti *index*, *old_nearest_point_index*, *k*, *Lfc*, *Lf*, dan *velocity* diinisialisasi untuk melacak titik terdekat, menentukan jarak pandang, dan memperbarui indeks secara efisien.

Tugas pertama dari algoritma ini adalah mengidentifikasi titik terdekat di jalur terhadap posisi robot, yang dilakukan dalam fungsi “*SearchTargetIndex*”. Pada iterasi pertama, algoritma menghitung jarak Euclidean antara posisi robot saat ini (*x_now*, *y_now*) dan setiap titik di jalur. Jarak-jarak ini disimpan dalam array, kemudian indeks dari nilai minimum dipilih menggunakan fungsi “*SearchMinValue*”. Indeks ini mewakili titik terdekat di jalur dengan posisi robot saat ini. Setelah diidentifikasi, titik ini disimpan dalam “*old_nearest_point_index*” untuk referensi selanjutnya agar tidak menghitung ulang semua jarak di iterasi berikutnya. Pada iterasi berikutnya, algoritma memulai dari “*old_nearest_point_index*” untuk menghindari perhitungan yang tidak perlu. Algoritma membandingkan jarak antara robot dan titik saat ini (*index*) dengan titik berikutnya di jalur (*index + 1*). Jika titik berikutnya lebih dekat, algoritma terus menginkrementasi indeks hingga menemukan titik yang meminimalkan jarak ke robot, memastikan bahwa selalu melacak titik terdekat. Titik terdekat ini disimpan dalam “*old_nearest_point_index*” untuk iterasi berikutnya.

Jarak pandang ke depan (*look-ahead distance*, *Lf*) sangat penting dalam algoritma Pure Pursuit karena menentukan seberapa jauh robot harus mengarah di jalur. Jarak ini disesuaikan secara dinamis berdasarkan kecepatan kendaraan menggunakan rumus:

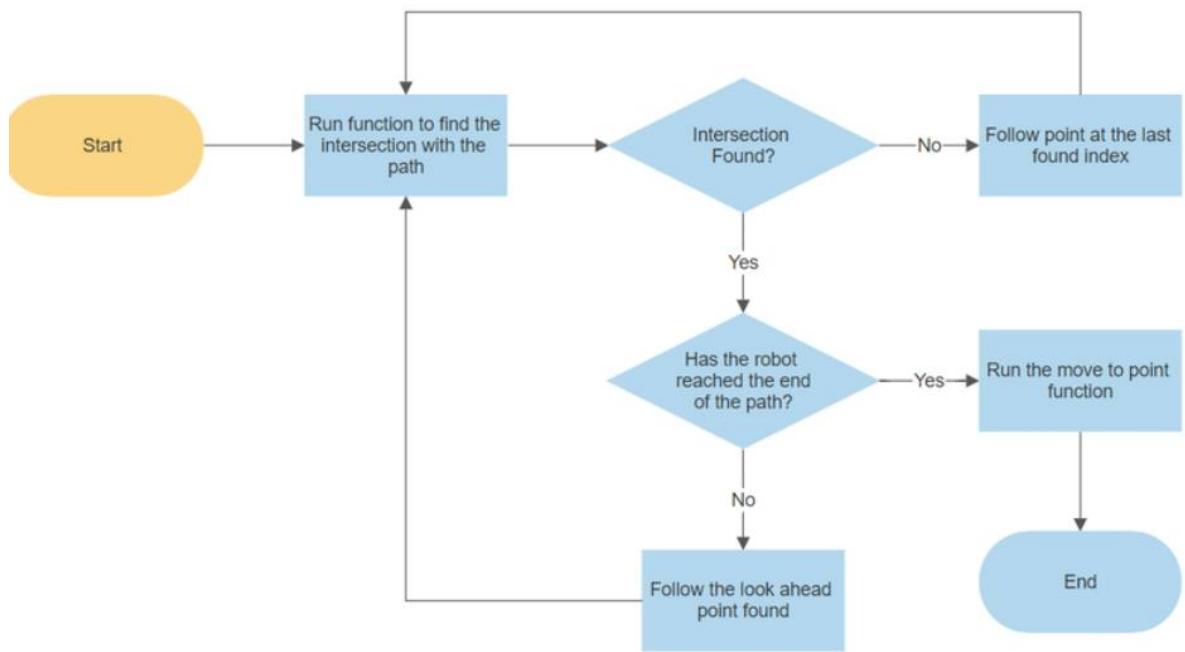
$$Lf = k \cdot velocity + Lfc$$

Di sini, k adalah penguat proporsional yang mengatur jarak pandang ke depan berdasarkan kecepatan robot. Semakin cepat robot bergerak, semakin jauh robot harus melihat atau semakin jauh titik target yang akan dipilih. Konstanta Lfc memastikan ada jarak pandang ke depan minimum, bahkan pada kecepatan rendah atau saat robot diam.

Setelah titik terdekat diidentifikasi dan jarak pandang ke depan dihitung, algoritma mencari titik target di jalur yang memiliki jarak sama atau lebih besar dari jarak pandang ke depan. Algoritma memeriksa apakah jarak dari titik saat ini di jalur ke robot lebih kecil dari jarak pandang ke depan Lf , dan jika iya, algoritma menginkrementasi indeks ke titik berikutnya. Proses ini diulang hingga ditemukan titik target yang tepat, yang kemudian digunakan untuk mengatur kemudi.

Langkah akhir dari algoritma diimplementasikan dalam fungsi “*PurePursuitControl*”. Setelah mengidentifikasi titik terdekat dan menentukan titik pandang ke depan, fungsi ini mengembalikan koordinat (x, y) dari titik target yang akan diikuti robot. Jika indeks saat ini melebihi panjang jalur, titik terakhir di jalur dikembalikan sebagai target, memastikan bahwa robot terus mengikuti jalur hingga akhir. Titik target ini kemudian digunakan oleh sistem kontrol robot untuk menyesuaikan kemudi dan menjaga pelacakan jalur yang mulus.

Secara keseluruhan, algoritma Pure Pursuit bekerja dengan terus-menerus mengidentifikasi titik terdekat pada jalur yang telah ditentukan dan memilih titik target yang berjarak tetap di depan. Orientasi robot kemudian disesuaikan untuk mengarah ke titik target ini. Jarak pandang ke depan disesuaikan secara dinamis berdasarkan kecepatan robot, memungkinkan kemudi yang lebih agresif pada kecepatan tinggi dan kontrol yang lebih halus pada kecepatan rendah. Algoritma ini sangat efisien untuk navigasi waktu nyata karena hanya mempertimbangkan sebagian jalur di depan robot, menghindari perhitungan yang tidak perlu untuk titik-titik yang lebih jauh di jalur.



Gambar 31. Diagram Alir Algoritma Pure Pursuit Controller

12. PID Controllers untuk Mengatur Linear Velocity Robot

Pseudocode:

```

FUNCTION Compute(input):
    currentTime = Get current time in milliseconds
    timeChange = currentTime - lastTime

    IF timeChange >= sampleTime THEN:
        error = setpoint - input
        errorSign = (error > 0) - (error < 0)

    IF errorSign ≠ prevErrorSign THEN:
        integral = 0

        prevErrorSign = errorSign

    # Batasi kesalahan dalam rentang yang diperbolehkan
    error = CONSTRAIN(error, -minMaxOutput, minMaxOutput)
    proportional = error
  
```

```

integral = integral + (error * timeChange)
# Batasi nilai integral
integral = CONSTRAIN(integral, -minMaxOutput / ki, minMaxOutput / ki)

derivative = (input - lastInput) / timeChange
# Batasi nilai derivatif
derivative = CONSTRAIN(derivative, -minMaxOutput / kd, minMaxOutput / kd)

output = (kp * error) + (ki * integral) + (kd * derivative) # Hitung output PID
# Batasi output dalam rentang yang diperbolehkan
output = CONSTRAIN(output, -minMaxOutput, minMaxOutput)

lastOutput = output # Simpan output untuk iterasi berikutnya
lastInput = input # Simpan input untuk perhitungan derivatif berikutnya
lastTime = currentTime # Simpan waktu saat ini untuk iterasi berikutnya

RETURN output
END IF
END FUNCTION

FUNCTION Control(speed_now):
    IF Get current time in milliseconds > timer_last + time_sampling_duration THEN:
        # Perbarui kecepatan kontrol menggunakan PID
        control_speed = control_speed + Compute(speed_now)

        # Batasi kecepatan kontrol dalam rentang [100, 1000]
        control_speed = CONSTRAIN(control_speed, 100.0, 1000.0)

        # Perbarui timer untuk iterasi berikutnya
        timer_last = Get current time in milliseconds
    END IF

```

```
    RETURN control_speed  
END FUNCTION
```

Kode tersebut mengimplementasikan algoritma PID Controller (Proportional-Integral-Derivative) untuk mengendalikan kecepatan linear robot. Fungsi utama yang digunakan adalah “Compute”, yang menghitung keluaran PID berdasarkan perbedaan antara kecepatan yang diinginkan (setpoint) dan kecepatan aktual robot (input). Fungsi “Control” kemudian menggunakan hasil dari PID ini untuk menyesuaikan kecepatan robot secara bertahap sesuai dengan waktu yang telah ditentukan.

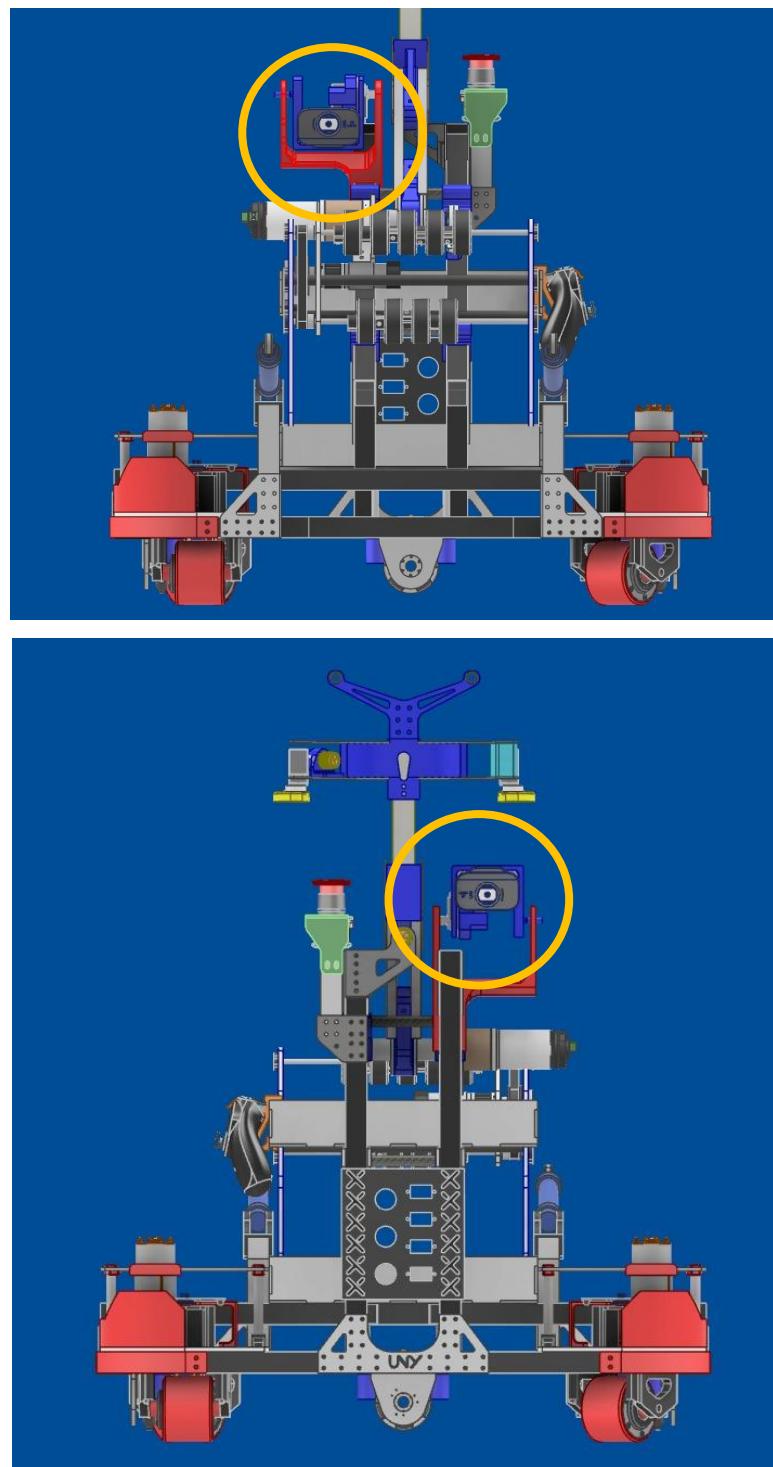
Pada fungsi “Compute”, pertama-tama, waktu saat ini dihitung, dan perubahan waktu sejak iterasi terakhir (timeChange) digunakan untuk menentukan seberapa banyak kontrol yang perlu dilakukan pada langkah ini. Selisih antara setpoint dan kecepatan aktual dihitung sebagai error. Jika tanda error berubah (dari positif ke negatif atau sebaliknya), integral di-reset untuk mencegah penumpukan kesalahan yang berlebihan. Kemudian, error ini dibatasi agar tidak melebihi nilai maksimum yang ditentukan.

Bagian integral menambahkan error ke total kesalahan yang terakumulasi dari waktu ke waktu, tetapi tetap dibatasi agar tidak terjadi "integral windup" (akumulasi berlebihan). Bagian derivatif menghitung perubahan input (kecepatan) dari waktu ke waktu, membantu mengurangi osilasi dengan mempertimbangkan seberapa cepat input berubah. Setelah itu, keluaran PID dihitung sebagai jumlah berbobot dari error (proportional), kesalahan terakumulasi (integral), dan perubahan input (derivative). Nilai ini kemudian dibatasi dalam kisaran tertentu untuk memastikan kontrol tetap stabil.

Pada fungsi “Control”, PID dihitung setiap kali durasi sampling waktu terpenuhi. Hasil dari fungsi PID digunakan untuk menambah atau mengurangi kecepatan kendali (control_speed) robot. Setelah disesuaikan, kecepatan dikonfirmasi agar tetap dalam batas yang diinginkan (antara 100.0 dan 1000.0), lalu dikembalikan sebagai output. Dengan demikian, PID controller ini berfungsi untuk memastikan kecepatan robot disesuaikan secara halus agar mendekati nilai yang diinginkan, mengurangi kesalahan dan osilasi dengan cepat, dan menjaga kecepatan tetap dalam batas yang telah ditentukan.

13. Deteksi Objek dengan YOLOv8

a. Desain Peletakan Webcam Sebagai Sensor Kamera Pada Robot



Gambar 32. Desain Peletakan Webcam Pada Robot

b. Pengambilan Data

Data yang diambil adalah berupa foto bola dengan tiga warna berbeda dan silo (keranjang). Foto ini diambil menggunakan webcam yang diletakkan pada sudut pandang robot. Bola yang dideteksi memiliki 3 warna yang berbeda, yaitu bola merah, bola biru, dan bola ungu. Sedangkan, silo yang dideteksi akan memiliki 15 kemungkinan berbeda berdasarkan bola yang berada dalam silo tersebut. Kemungkinan tersebut mulai dari silo yang kosong, silo berisi satu bola, silo berisi dua bola, dan silo penuh atau berisi tiga bola dengan semua kemungkinan susunan warna bola yang ada.

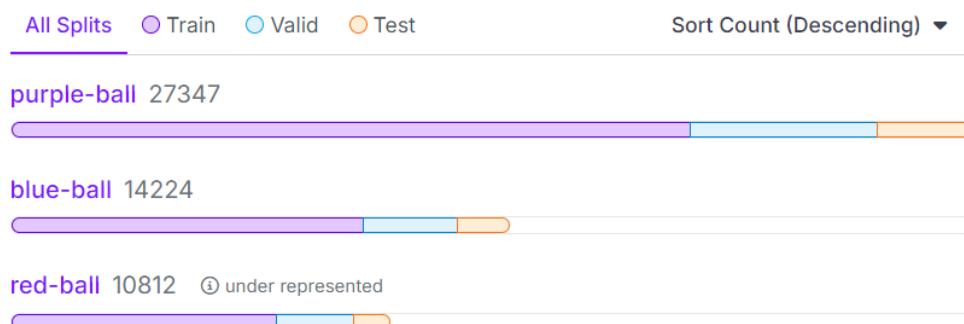
Foto-foto diambil pada sudut yang berbeda dan kemungkinan kondisi susunan bola yang berbeda. Hal ini dilakukan agar model yang dihasilkan nantinya akan semakin akurat. Selain pengambilan data berupa foto, dilakukan juga pengambilan dari dataset serupa yang telah tersedia di Roboflow. Dataset yang digunakan terdiri atas 12104 gambar silo dan 16571 gambar bola.

c. Anotasi Dataset

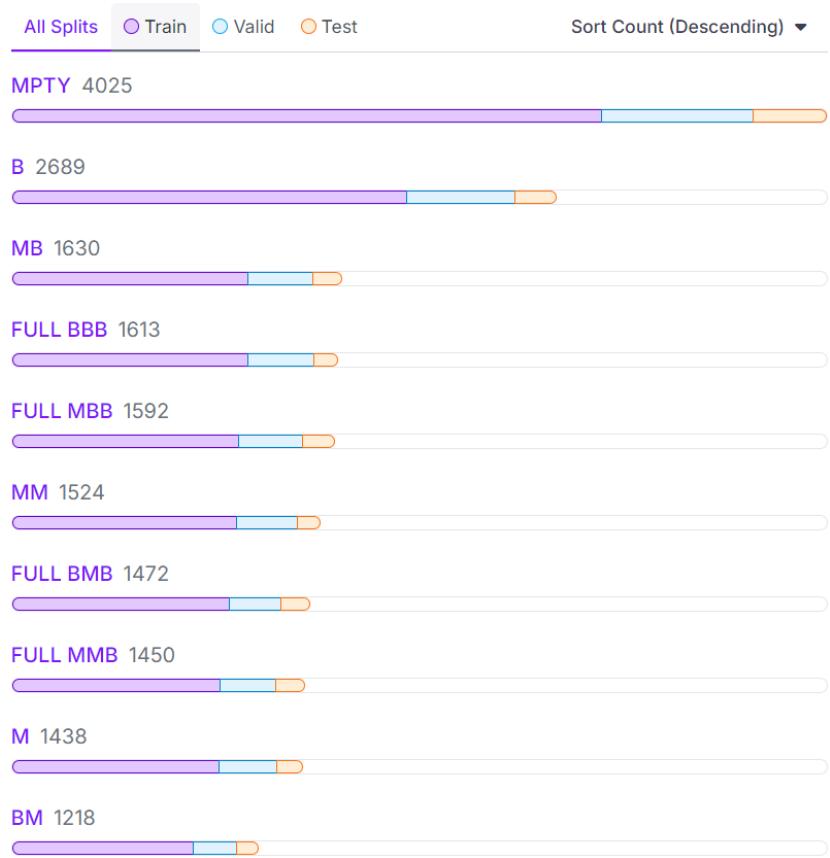
Anotasi pada dataset merupakan pemberian bounding box pada objek yang ingin dideteksi dan memberi label kelas pada setiap objek yang telah diberi bounding box. Pelabelan dataset dilakukan menggunakan Roboflow.

d. Pembagian Dataset

Pada tahap ini, akan dilakukan proses pembagian dataset yang telah dibuat. Dataset akan dibagi menjadi 3 bagian, yaitu Training Set sejumlah 70%, Validation Set sejumlah 20%, dan Test Set sejumlah 10% dari jumlah dataset keseluruhan. Train set digunakan untuk melatih model, validation set digunakan untuk memantau kinerja model selama pelatihan untuk menghindari *overfitting*, dan test set untuk mengevaluasi kinerja akhir model setelah pelatihan selesai.



Gambar 33. Pembagian Dataset Bola



Gambar 34. Pembagian Dataset Silo

e. Augmentasi Dataset

Augmentasi ini diimplementasikan untuk menambah ukuran dataset dan keberagaman dataset. Penambahan ini akan membantu meningkatkan performa model nantinya. Roboflow menyediakan augmentasi *built-in* yang akan diaplikasikan selama pelatihan dataset untuk membuat model lebih bervariasi untuk berbagai keadaan yang terjadi nantinya saat robot melakukan deteksi objek. Augmentasi yang digunakan adalah kecerahan ($\pm 15\%$), eksposur ($\pm 10\%$), dan saturasi ($\pm 15\%$), blur (hingga 1px), dan noise (hingga 0,1% pixels).

Augmentasi kecerahan dan eksposur akan menambahkan variabilitas pada kecerahan dan paparan gambar untuk membantu model lebih tahan terhadap perubahan pencahayaan dan pengaturan kamera. Augmentasi blur akan menambahkan Gaussian blur acak untuk membantu model lebih tahan terhadap fokus kamera. Noise akan menambahkan kebisingan atau gangguan pada gambar

untuk membantu model lebih tangguh tahan terhadap artefak kamera. Saturasi akan menyesuaikan kejemuhan pada gambar secara acak.

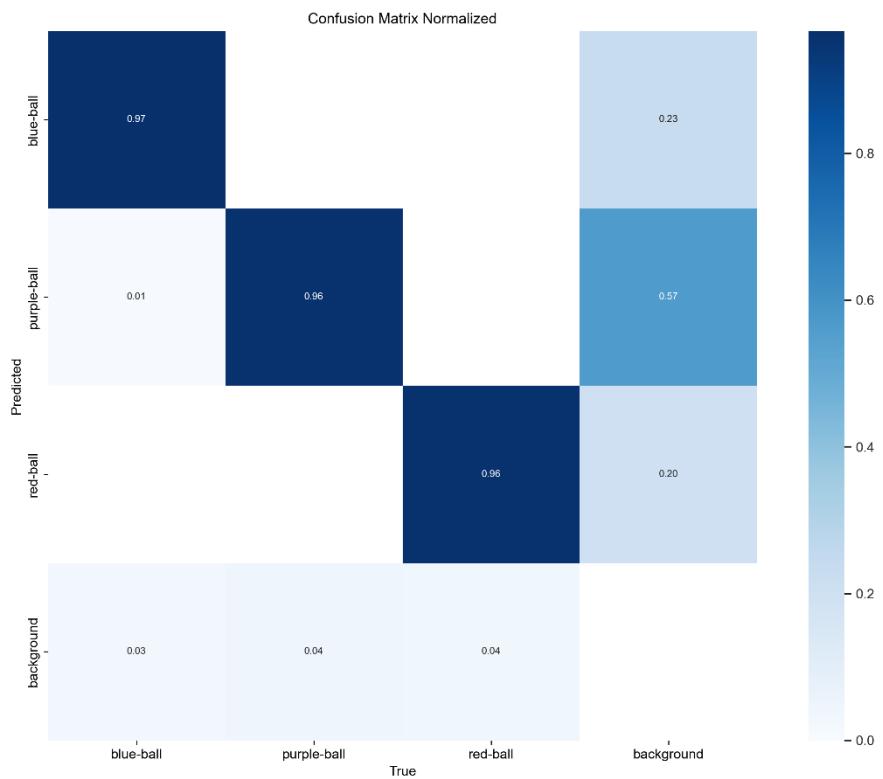
f. Pelatihan Dataset

Setelah melakukan proses labelling pada setiap data train, validation, dan test kemudian dilakukan proses training pada dataset tersebut. Proses training pada model YOLOv8 ini menggunakan parameter ukuran gambar sebesar 320x320 piksel dan jumlah Iterasi atau epoch sebesar 200 untuk dataset bola dan 100 untuk dataset silo. parameter ukuran gambar (image size) dan jumlah iterasi (epoch) sangat penting untuk mencapai hasil yang optimal.

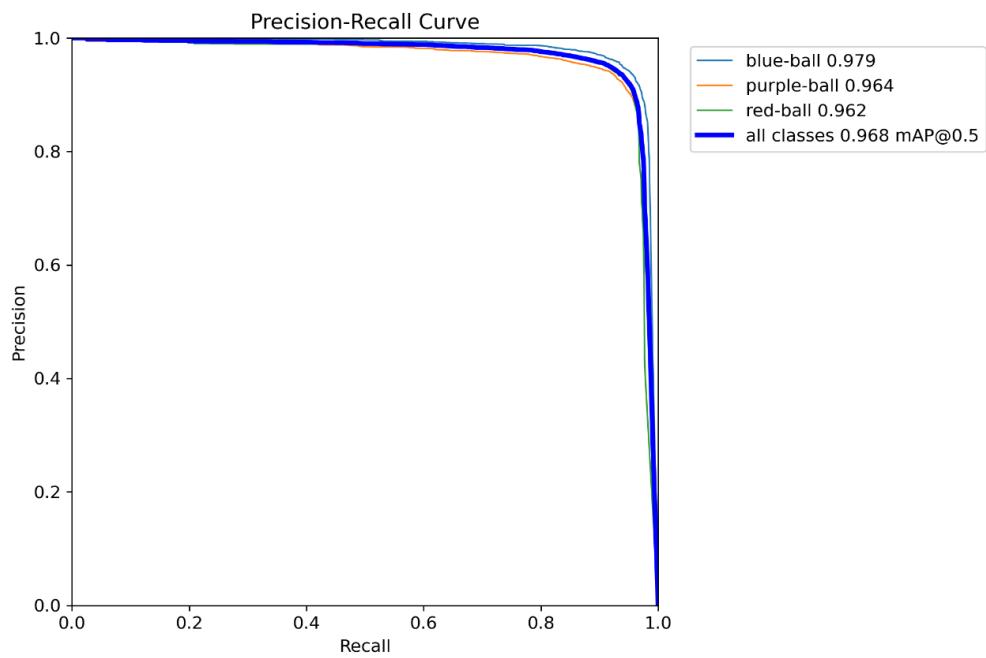
Ukuran gambar yang digunakan selama proses pelatihan berpengaruh terhadap akurasi model dan waktu komputasi. Ukuran ini mengacu pada resolusi gambar yang akan diproses oleh model. Alassan menggunakan ukuran gambar yang lebih kecil (320x320 piksel) agar proses pelatihan dapat berjalan lebih cepat. Ukuran yang lebih kecil memungkinkan model untuk memproses gambar lebih efisien dan mengurangi waktu pelatihan, terutama pada dataset yang besar. Meskipun ukuran gambar lebih kecil dapat mempercepat proses pelatihan, hal ini juga bisa mengurangi detail dalam gambar. Untuk dataset seperti bola dan silo, ukuran 320 cukup karena objek yang dideteksi biasanya cukup jelas dan tidak memerlukan resolusi yang sangat tinggi.

Sedangkan epoch adalah istilah yang digunakan untuk menyatakan satu kali siklus penuh pelatihan model pada seluruh dataset. Saat melakukan pelatihan model YOLOv8, jumlah epoch menentukan berapa kali model melihat seluruh data pelatihan dan mencoba memperbaiki prediksi. Jumlah epoch yang lebih tinggi memberikan kesempatan bagi model untuk belajar lebih lama dari dataset, sehingga menghasilkan prediksi yang lebih akurat. Dengan epoch yang lebih banyak, model lebih mungkin untuk menangkap pola-pola yang lebih kompleks pada data. Epoch yang terlalu tinggi juga bisa membuat model berisiko mengalami *overfitting*, di mana model terlalu cocok dengan data pelatihan dan tidak dapat melakukan generalisasi dengan baik pada data baru.

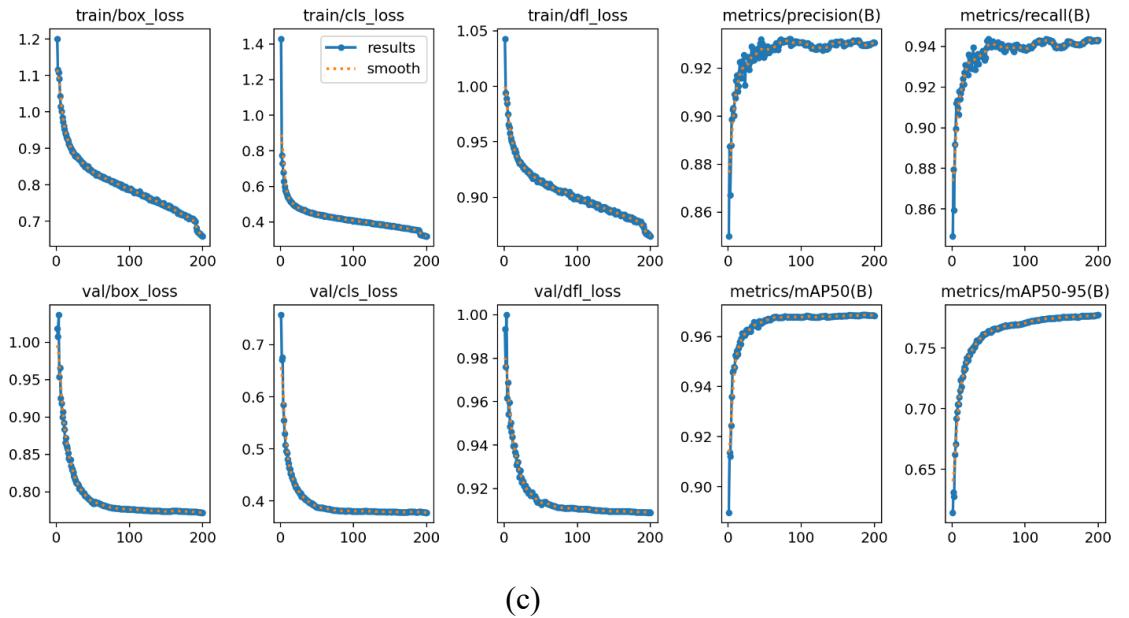
g. Evaluasi Model



(a)



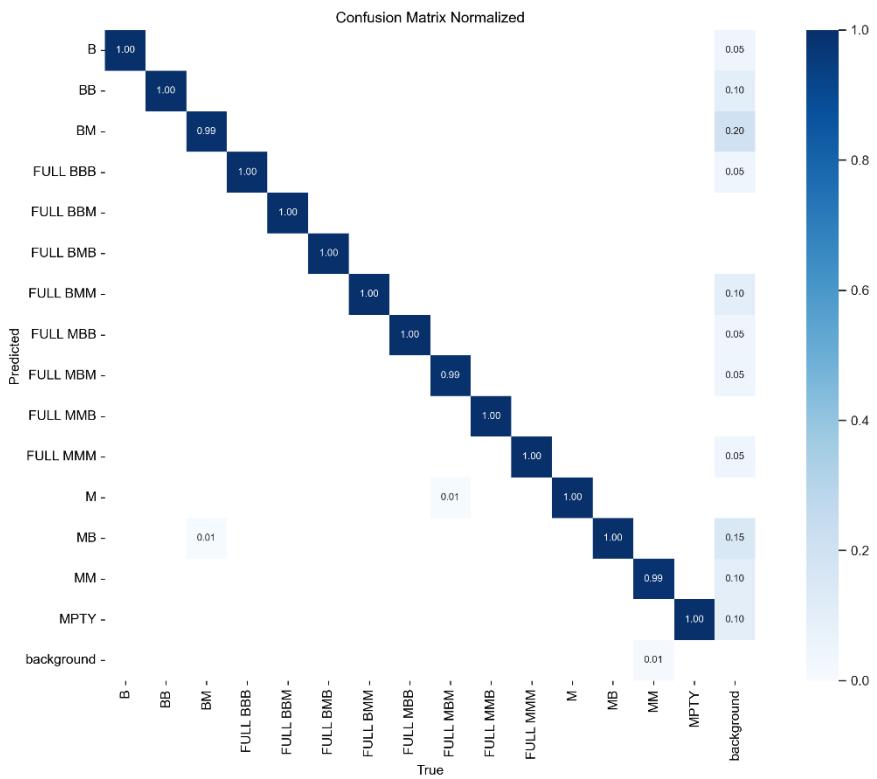
(b)



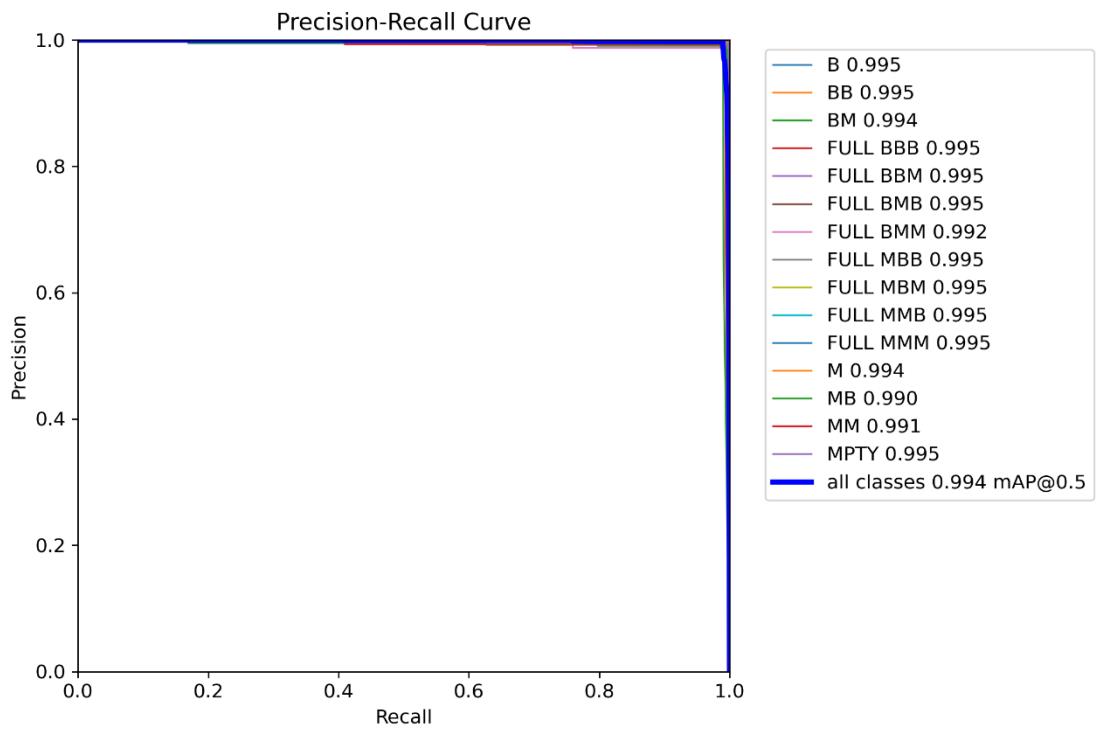
(c)

Gambar 35. Hasil Evaluasi Model Bola: (a) Confusion Matrix, (b) Kurva Precision-Recall, (c) Loss Function.

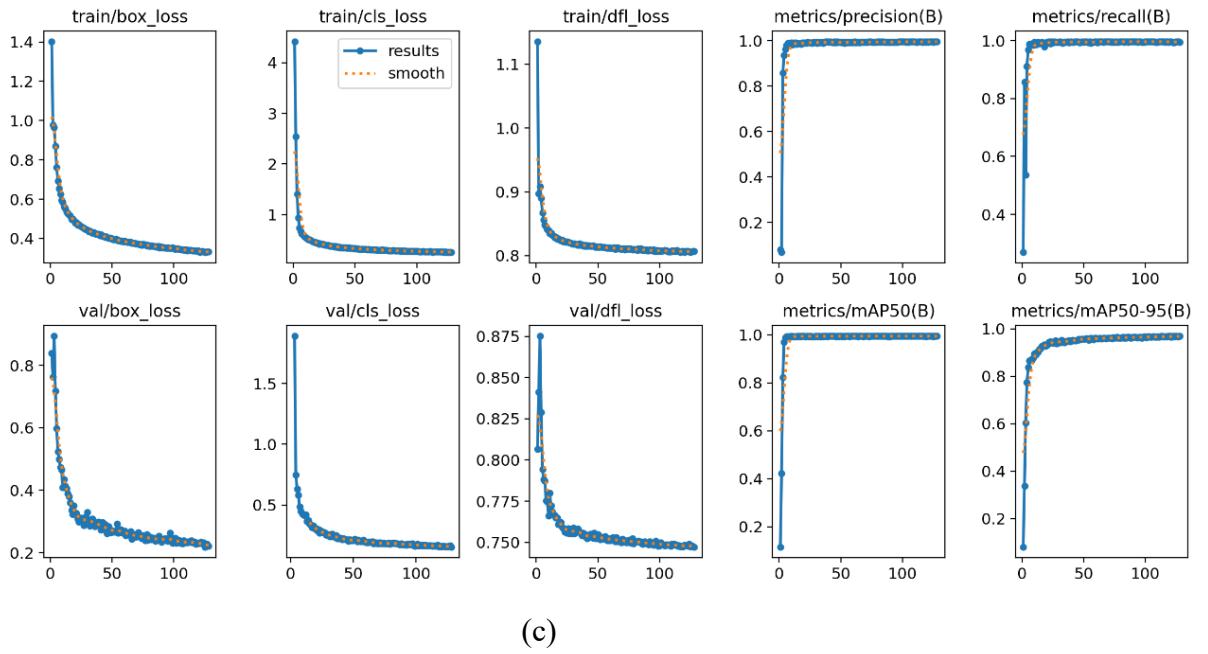
Gambar-gambar yang ditampilkan merupakan evaluasi model YOLOv8. Pada gambar 33(a), *Confusion Matrix* yang ditampilkan menunjukkan performa model YOLOv8 dalam mengklasifikasikan empat kelas berbeda, yaitu blue-ball, purple-ball, red-ball, dan background. Dari hasil tersebut, model menunjukkan kinerja yang sangat baik dalam mengklasifikasikan blue-ball dengan tingkat akurasi 97%, purple-ball dengan akurasi 96%, red-ball dengan akurasi 96%. Secara keseluruhan, model bekerja dengan baik dalam mendekripsi dan mengklasifikasikan setiap warna bola. Lalu, pada gambar 33(b). Kurva Precision-Recall yang menunjukkan hubungan antara precision dan recall untuk berbagai kelas objek, yaitu blue-ball, purple-ball, dan red-ball. Precision untuk masing-masing kelas adalah 0.979, 0.964, dan 0.962, sementara nilai mean Average Precision (mAP) untuk semua kelas adalah 0.968 pada ambang batas 0.5. Ini menunjukkan bahwa model memiliki kinerja yang sangat baik dalam mengidentifikasi objek dengan akurasi tinggi. Gambar 33(c) menunjukkan berbagai metrik kehilangan (loss) selama pelatihan dan validasi, termasuk box loss, class loss, dan detection loss. Grafik-grafik ini menunjukkan penurunan yang konsisten dalam nilai loss, menandakan bahwa model belajar dengan baik selama proses pelatihan.



(a)



(b)



(c)

Gambar 36. Hasil Evaluasi Model Silo: (a) Confusion Matrix, (b) Kurva Precision-Recall, (c) Loss Function.

Begitu juga pada evaluasi model silo yang ditunjukkan pada gambar 34. Confusion Matrix yang dihasilkan menunjukkan performa model sangat baik dalam mengklasifikasi berbagai kondisi silo dengan akurasi hingga 100%. Pada kurva Precision-Recall menunjukkan bahwa model memiliki kinerja yang sangat baik dalam mengidentifikasi objek dengan akurasi tinggi. Precision untuk masing-masing kelas mencapai lebih dari 0.99 dengan nilai mean Average Precision (mAP) untuk semua kelas adalah 0.994 pada ambang batas 0.5. Grafik pada metrik kehilangan (loss) menunjukkan bahwa model belajar dengan baik selama proses pelatihan dengan ditandai penurunan yang konsisten dalam nilai loss,

h. Deteksi dan Pengambilan Posisi Bola dan Silo

Deteksi objek berbasis kamera untuk mengidentifikasi bola dan silo, serta menghitung koordinat tengah dari objek yang terdeteksi. Proses deteksi ini menghasilkan bounding box yang memuat koordinat setiap objek, yaitu (x_1, y_1) sebagai sudut kiri atas dan (x_2, y_2) sebagai sudut kanan bawah. Dengan bounding box tersebut, koordinat tengah objek dihitung, dan informasi ini digunakan untuk menentukan posisi bola dan silo dalam navigasi robot.

Pseudocode:

```

function DetectObjects():

    ball_pos = []
    purple_ball_pos = []
    silo_counter = 0

    # Iterate over all detected balls
    for result in results_bola:
        boxes = result.boxes

        # Process each bounding box for balls
        for box in boxes:
            class_id = int(box.cls[0]) # class ID of detected object
            confidence = box.conf[0] # confidence score
            # red/blue ball
            if (class_id == 0 if field_side else class_id == 2) and confidence > 0.4:
                # Get bounding box coordinates
                (x1, y1, x2, y2) = map(int, box.xyxy[0])
                # Draw bounding box on the frame
                cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), 3)
                # Calculate the center of the bounding box
                cx = (x1 + x2) // 2
                cy = (y1 + y2) // 2

                # Store the center coordinates of the red/blue ball
                ball_pos.append((cx, cy))

            # purple ball
            if (class_id == 1) and confidence > 0.4:
                (x1, y1, x2, y2) = map(int, box.xyxy[0])
                cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 0, 255), 3)

                cx = (x1 + x2) // 2
                cy = (y1 + y2) // 2

```

```

purple_ball_pos.append((cx, cy))

# Iterate over all detected silos
for result in results_silo:
    boxes = result.bboxes

    # Process each bounding box for silos
    for box in boxes:
        class_id = int(box.cls[0])
        confidence = box.conf[0]
        if confidence > 0.6:
            # Increment the silo counter
            silo_counter += 1

            (x1, y1, x2, y2) = map(int, box.xyxy[0])
            cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 0, 255), 3)

            # Calculate the center of the bounding box
            cx = (x1 + x2) // 2
            cy = (y1 + y2) // 2

    # Return detected positions and silo count
    return red_ball_pos, purple_ball_pos, silo_counter

```

1. Deteksi Bola

Dilakukan iterasi pada setiap hasil deteksi bola menggunakan:

for r in results_bola:

Setiap objek yang terdeteksi memiliki kelas *cls* dan tingkat keyakinan (*confidence*) *conf*. Logika berikut memastikan bola yang relevan dideteksi berdasarkan kelas dan nilai *confidence*:

$$cls = 0 \text{ atau } 2 \wedge conf > 0.4$$

Kelas 0 atau 2 menandakan bahwa objek yang dideteksi harus terklasifikasi sebagai bola merah atau bola biru tergantung sisi lapangan yang diperoleh. Jika

syarat ini terpenuhi, bounding box bola akan diambil, dan koordinat tengahnya dihitung dengan:

$$cx = \frac{x_1 + x_2}{2}$$

$$cy = \frac{y_1 + y_2}{2}$$

Posisi tengah bola (cx, cy) disimpan dalam “ball_pos” untuk bola merah atau biru, tergantung pada sisi lapangan. Kemudian, bounding box digambar pada frame dengan warna merah menggunakan fungsi `cv2.rectangle()`.

Pada deteksi bola ungu, logika serupa digunakan tetapi memeriksa apakah kelas objek adalah 1. Koordinat tengah bola ungu dihitung dengan cara yang sama dan disimpan dalam “purple_ball_pos” untuk digunakan dalam navigasi.

2. Deteksi Silo

Deteksi silo dilakukan dengan iterasi pada setiap hasil deteksi silo:

for r in results_silo:

Setiap objek silo diperiksa berdasarkan nilai confidence:

$$conf > 0.6$$

Jika syarat ini terpenuhi, “silo_counter” ditingkatkan untuk mencatat jumlah silo yang ditemukan. Kemudian, bounding box digambar di frame dengan warna magenta menggunakan `cv2.rectangle()`, dan koordinat tengah silo dihitung dengan:

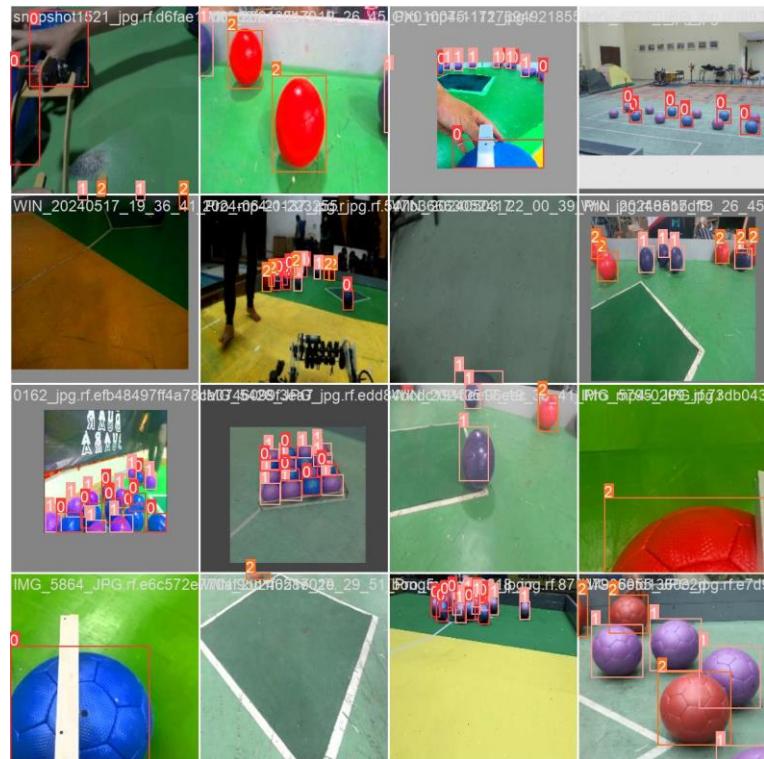
$$cx = \frac{x_1 + x_2}{2}$$

$$cy = \frac{y_1 + y_2}{2}$$

Secara keseluruhan, metode ini menggunakan bounding box untuk menentukan posisi bola dan silo secara real-time. Dengan menghitung koordinat tengah (cx, cy), robot dapat mengetahui lokasi setiap objek dan menyesuaikan navigasinya. Proses ini memanfaatkan nilai confidence untuk memfilter objek yang tidak relevan, sehingga memastikan hanya bola dan silo dengan tingkat keyakinan tinggi yang diproses. Data posisi ini sangat penting untuk memungkinkan robot mengambil keputusan yang tepat selama kompetisi, seperti mendekati bola atau memasukkan bola ke dalam silo.



(a)

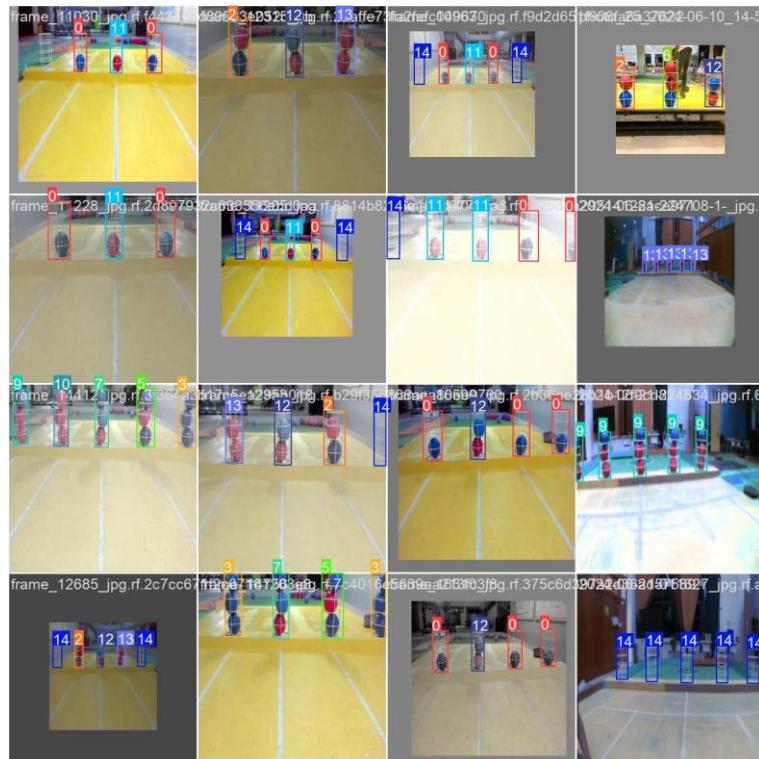


(b)

Gambar 37. (a), (b) Hasil Deteksi Bola



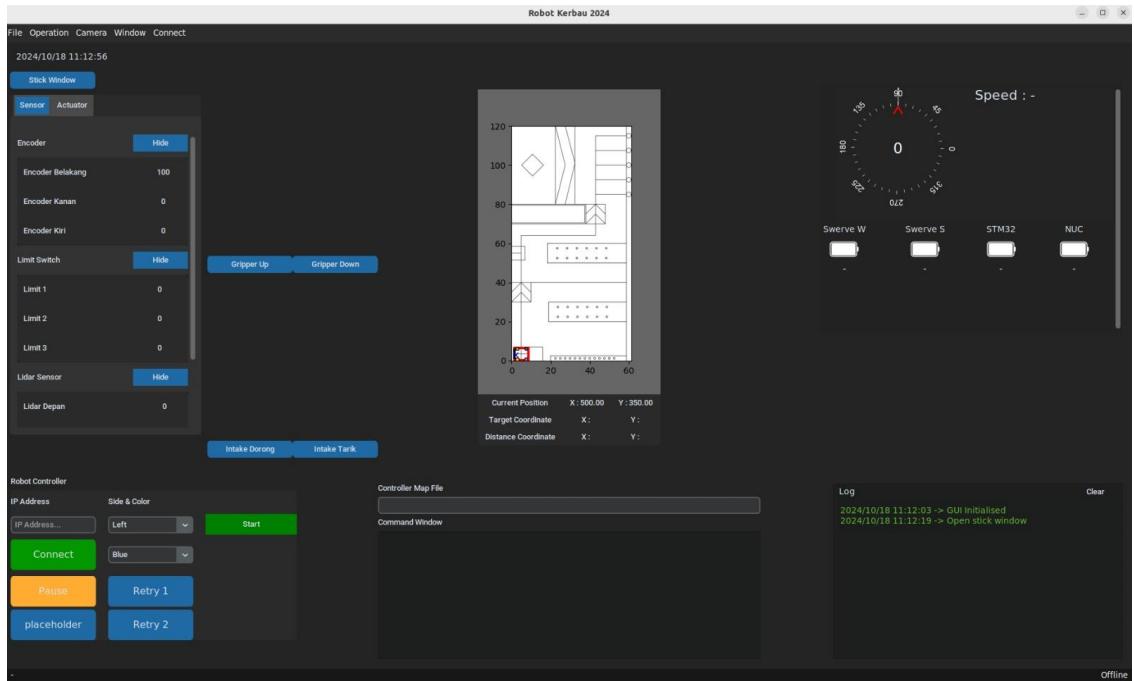
(a)



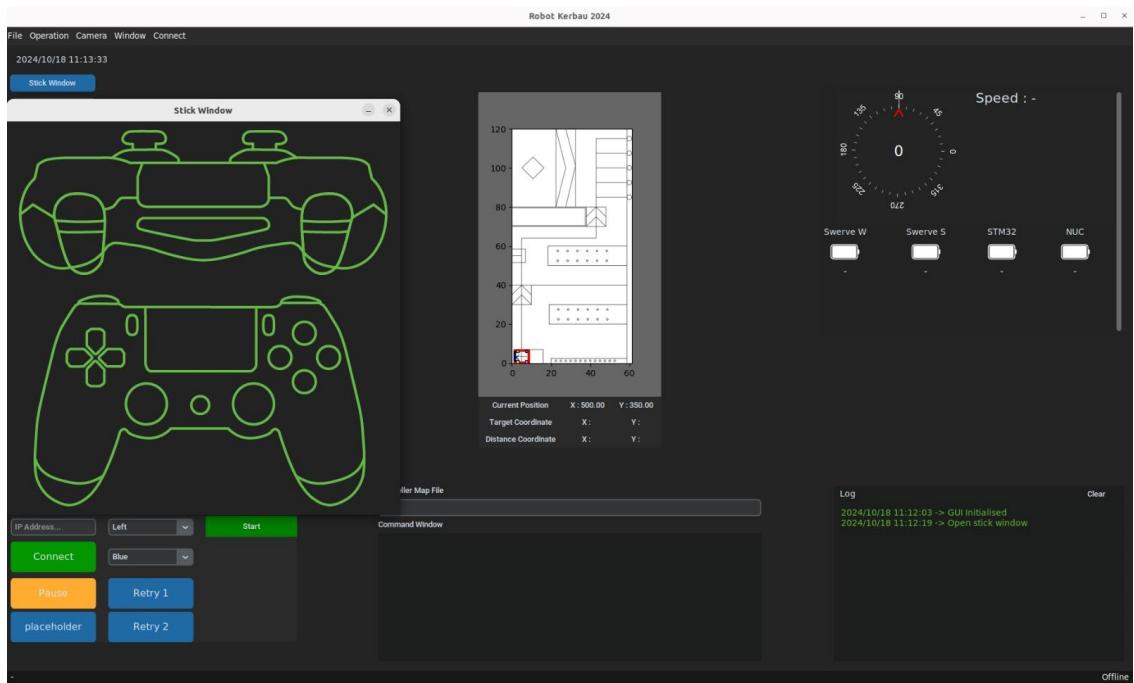
(b)

Gambar 38. (a), (b) Hasil Deteksi Silo

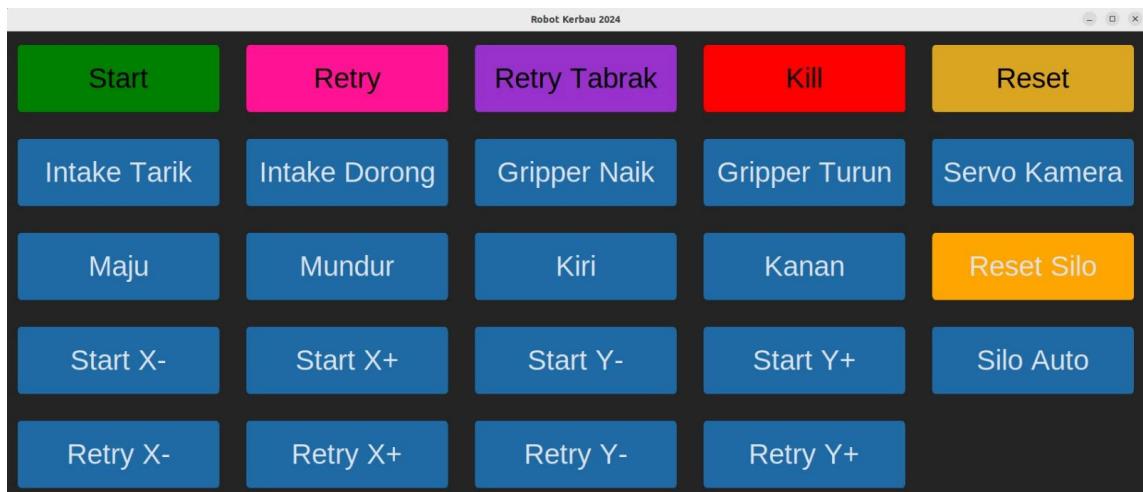
14. Graphical User Interface (GUI)



Gambar 39. Grapichal User Interface Robot Dengan Sensor Window Terbuka



Gambar 40. Graphical User Interface Robot Dengan Stick Window Terbuka



Gambar 41. Graphical User Interface Pengendali Mekanisme Robot

GUI dirancang untuk mengelola operasi robot secara real-time, menggunakan ROS2 (Robot Operating System 2) sebagai sistem komunikasi antara antarmuka dan robot. GUI ini dibangun menggunakan customtkinter untuk tampilan grafis dan memanfaatkan arsitektur publisher-subscriber dari ROS2. Antarmuka ini memberikan kemampuan kontrol dan pemantauan yang komprehensif terhadap berbagai subsistem robot, termasuk sensor, aktuator, dan mekanisme pergerakan.

Publisher mengirimkan perintah dari GUI ke controller robot untuk mengontrol gripper robot, mekanisme intake, serta perilaku penting robot lainnya seperti start/stop, permintaan retry, atau pemilihan warna bola. Pada Subscriber, GUI mendengarkan beberapa topik ROS2 untuk menerima data dari robot, termasuk input stick dan data IMU, posisi global dan kecepatan, dan input terkait kamera seperti deteksi bola dan posisinya. Komunikasi ini terstruktur sehingga robot terus mengirimkan umpan balik (data sensor, posisi, dll.) ke GUI, sementara operator dapat mengirimkan perintah (pergerakan, kontrol aktuator) dari GUI ke robot. Komunikasi dua arah ini memastikan robot dan GUI selalu sinkron.

Komponen inti lainnya adalah panel manajemen sensor dan aktuator, yang menampilkan data sensor secara real-time, termasuk nilai encoder untuk odometry (untuk encoder roda kiri, kanan, dan belakang) and pembacaan LiDAR dari berbagai arah. Nilai-nilai ini diperbarui secara terus-menerus berdasarkan pembacaan sensor robot, memastikan umpan balik real-time. GUI juga menangani pengoperasian aktuator seperti gripper dan mekanisme intake.

GUI juga mencakup tampilan peta (Map) yang melacak posisi global robot (x_{global} , y_{global}) dan heading robot, diperbarui melalui umpan balik sensor IMU

dan encoder. Posisi robot diperbarui berdasarkan data odometri yang diterima dari ROS2, yang mencakup koordinat global x, y, dan heading saat ini.

Log Frame menyimpan catatan peristiwa dan aksi yang diambil dalam GUI, menyediakan riwayat tekstual tentang perintah yang dikirim dan pembaruan yang diterima. Fitur logging ini berguna untuk debugging dan pemantauan aktivitas robot dari waktu ke waktu. Selain itu, Status Bar memberikan gambaran umum tentang status operasional robot saat ini, seperti status koneksi, kesalahan, atau peringatan.

Pembacaan sensor robot ditampilkan secara real-time dalam tab SensorActuator. Misalnya, nilai encoder ditampilkan terpisah untuk setiap encoder (kiri, kanan, dan belakang), sementara pembacaan LiDAR menunjukkan pengukuran jarak dari berbagai arah di sekitar robot. GUI juga memproses input stick untuk kontrol manual pergerakan atau orientasi robot. Fitur yang terakhir adalah tampilan tanggal dan waktu secara live di bagian atas antarmuka. Tanda waktu ini digunakan dalam log dan membantu operator melacak kapan tindakan tertentu diambil atau kapan umpan balik diterima dari robot.

GUI berjalan secara berkala menggunakan timer ROS2, memperbarui GUI dengan data sensor baru, memproses input dari operator, dan mengirimkan perintah kontrol ke robot. Timer ini memastikan bahwa GUI dan robot berkomunikasi secara real-time, memungkinkan kontrol yang responsif.

F. Validitas dan Reliabilitas Instrumen

Validitas konstruksi digunakan untuk memastikan bahwa sensor-sensor dan algoritma yang digunakan benar-benar mengukur apa yang seharusnya diukur dalam konteks navigasi dan lokalisasi robot. Setiap sensor (seperti odometri, gyroscope, accelerometer, LiDAR, dan kamera) serta algoritma (PID, Pure Pursuit) telah diuji berdasarkan standar yang ada dalam teori robotika dan sistem kontrol. Misalnya, sensor LiDAR divalidasi dengan menguji apakah jarak yang diukur sesuai dengan jarak aktual objek di lapangan. Kamera divalidasi dengan memverifikasi deteksi objek (bola dan silo) di berbagai kondisi pencahayaan dan lingkungan lapangan.

Reliabilitas internal digunakan untuk memastikan bahwa setiap sensor dan instrumen pengendali memberikan hasil yang konsisten dalam situasi yang serupa. Uji konsistensi dilakukan dengan menjalankan robot pada lintasan yang sama berulang kali dan mengukur apakah hasil dari odometri, LiDAR, gyroscope-accelerometer, dan kamera tetap

stabil. Jika hasil pengukuran tetap konsisten, maka instrumen dianggap memiliki reliabilitas yang baik. Uji stabilitas dilakukan dengan mengulang pengukuran pada interval waktu yang berbeda untuk memastikan bahwa sensor tetap memberikan data yang sama. Ini penting terutama untuk menguji drift pada sensor seperti gyroscope dan odometri. Uji stabilitas menguji apakah sensor tersebut tetap memberikan data yang akurat setelah periode waktu tertentu tanpa perlu kalibrasi ulang.

G. Teknik Analisis Data

Dalam penelitian yang berfokus pada pengembangan sistem lokalisasi dan navigasi robot otonom berbasis swerve-drive, teknik analisis data yang digunakan bertujuan untuk mengevaluasi kinerja sensor, algoritma kontrol, serta kemampuan robot dalam menavigasi dan melokalisasi diri secara akurat. Data yang dianalisis berasal dari hasil pengujian robot dalam berbagai skenario uji yang mensimulasikan kondisi lapangan kompetisi ABU Robocon 2024. Berikut adalah teknik analisis data yang digunakan dalam penelitian ini:

1. Analisis Akurasi Lokalisasi

Pengujian ini dilakukan untuk mengevaluasi seberapa akurat sistem lokalisasi robot dalam menentukan posisi globalnya berdasarkan data dari multi-sensor (odometri, gyroscope, accelerometer, LiDAR, dan kamera). Pengujian dilakukan dengan menganalisis rata-rata error posisi yang terjadi selama robot bergerak di lapangan uji serta menganalisis kesalahan akumulatif (drift) pada odometri dan seberapa efektif sensor lain (gyroscope-accelerometer, LiDAR) dalam mengoreksi kesalahan tersebut.

2. Analisis Performa Algoritma Navigasi

Pengujian ini dilakukan untuk menganalisis kinerja algoritma kontrol, seperti PID dan Pure Pursuit, dalam mengatur kecepatan dan orientasi robot untuk mencapai target dengan akurat. Dilakukan analisis error posisi robot yang dihitung sebagai selisih antara posisi yang diharapkan (target) dengan posisi aktual yang diukur oleh sensor serta mengukur bagaimana algoritma PID dan Pure Pursuit merespon perubahan lingkungan atau perubahan target navigasi.

BAB IV

HASIL PENELITIAN DAN PEMBAHASAN

Algoritma pengendalian robot otomatis dirancang untuk menjalankan berbagai misi secara otomatis berdasarkan masukan berbagai sensor. Algoritma ini menyediakan pemrograman logika untuk memastikan robot dapat beradaptasi secara dinamis selama kompetisi. Misi-misi yang ditugaskan pada robot diantaranya, pergerakan ke *storage zone* (area atau lantai 3), pendekslan dan pengejaran bola dan silo, pengimplementasian odometry untuk lokalisasi robot secara global, penggunaan PID dan Pure Pursuit controller untuk kontrol kecepatan robot, pengendalian mekanisme seperti servo, gripper, intake dan komponen lainnya untuk proses lokalisasi dan navigasi robot.

Pada awal program, robot diinisialisasi dengan parameter dan sensor utama seperti LIDAR, IMU, kamera, dan odometri. Parameter dan sensor ini memberikan robot posisi awal dan orientasinya. Perilaku robot terstruktur di sekitar serangkaian misi, yang dipicu berdasarkan masukan eksternal (misalnya, deteksi silo atau bola) dan status internal (misalnya, posisi robot). Setiap misi sesuai dengan tugas dalam kompetisi.

Pseudocode Pelaksanaan Misi Robot Secara Otomatis:

```
function AutonomousMode() {
    // Initial setup and variables
    coordinate_x_lidar = calculateLidarX();
    coordinate_y_lidar = calculateLidarY();
    LidarPosition = (coordinate_x_lidar, coordinate_y_lidar);

    while (game_not_finished) {
        // Get current mission
        mission = AutoMissionNasional.GetMissionIndex();

        // Switch case for handling different missions
        switch (mission) {

            case GAME_START_RESET:
                // Reset everything for the game start
        }
    }
}
```

```

print("Mission 0: Reset Robot");

resetAllSystems();
setStartPosition();
AutoMissionNasional.NextMission();
break;

case AUTO_KE_LANTAI_3:
    // Move to storage zone
    print("Mission 1: Move to Storage Zone");
    pursuit_target = PurePursuitControl();
    target_distance = calculateDistance(pursuit_target);
    moveRobotToTarget(pursuit_target);

    if (target_distance <= threshold_distance) {
        AutoMissionNasional.JumpTo(AREA_3_LIDAR_FIRST_RESET);
    }
    break;

case AREA_3_LIDAR_FIRST_RESET:
    // Reset LIDAR coordinates after reaching Area 3
    print("Mission 2: Reset LIDAR Coordinates");
    LidarNow = (coordinate_x_lidar, coordinate_y_lidar);
    updateOdometry(LidarNow);

    if (LidarPosition.y > some_threshold) {
        AutoMissionNasional.JumpTo(TABRAK_BOLA_AWAL);
    }
    break;

case TABRAK_BOLA_AWAL:
    // Push the ball
    print("Mission 3: Push Ball");
    target_ball = setBallTargetCoordinates();

```

```

moveRobotToTarget(target_ball);

if (ball_target_reached()) {
    AutoMissionNasional.JumpTo(MUNDUR_RESET);
}

break;

case MUNDUR_RESET:
    // Reverse after hitting the ball
    print("Mission 4: Reverse Reset");
    reverseRobot();
    AutoMissionNasional.JumpTo(MUNDUR_SETELAH_NABRAK);
    break;

case DETEKSI_BOLA:
    // Detect the ball using camera
    print("Mission: Detect Ball");
    if (Camera.ball_detected) {
        AutoMissionNasional.JumpTo(CARI_BOLA_RESET);
    }
    break;

case CARI_BOLA_RESET:
    // Search and approach the ball
    print("Mission: Search for Ball");
    ConstantSpeedController.SetSpeed(250, 1300);
    if (GripperFaiq2.RotasiTurun()) {
        AutoMissionNasional.JumpTo(CARI_BOLA);
    }
    break;

case CARI_BOLA:
    // Handle ball pursuit and interaction

```

```

print("Mission: Pursue Ball");

detectBallPosition();

if (ball_in_grip_position()) {
    IntakeSheva2.TarikBola();
    AutoMissionNasional.JumpTo(CARI_SILO_RESET);
}

break;

case CARI_SILO_RESET:
    // Reset and search for silos
    print("Mission: Search for Silo Reset");
    ServoKamera.SetAngle(SERVO_KAMERA_SILO);
    ConstantSpeedController.SetSpeed(800, 1600);
    if (silos_detected()) {
        AutoMissionNasional.JumpTo(CARI_SILO);
    }
    break;

case CARI_SILO:
    // Approach silo and prepare for ball deposit
    print("Mission: Search for Silo");
    TargetSilo = findSiloPosition();
    moveRobotToSilo(TargetSilo);

    if (silo_target_reached()) {
        AutoMissionNasional.JumpTo(PURSUIT_SILO_RESET);
    }
    break;

case PURSUIT_SILO_RESET:
    // Reset and approach silo for deposit
    print("Mission: Pursuit Silo Reset");
    ConstantSpeedController.SetSpeed(450, 600);

```

```

moveToSilo(LidarPosition);

if (silo_target == 4) {
    AutoMissionNasional.JumpTo(PURSUIT_SILO_3);
}
break;

case PURSUIT_SILO_3:
    // Approach the silo (variant path)
    print("Mission: Pursuit Silo Lidar 3");
    TargetSilo = calculateSiloTarget();
    moveToSilo(TargetSilo);

    if (close_to_silo()) {
        AutoMissionNasional.JumpTo(SILONYA_PENUH);
    }
    break;

case SILONYA_PENUH:
    // Handle the case when the silo is full
    print("Mission: Silo Full");
    adjustPositionForSilo();
    if (!Camera.silo_penuh) {
        AutoMissionNasional.JumpTo(PURSUIT_SILO);
    }
    break;

case RESET_ODOMETRY_LIDAR_HABIS_SILO:
    // Reset odometry after silo interaction
    print("Mission: Reset Odometry After Silo");
    GlobalOdometry.SetPosition(LidarPosition);
    resetAllBallHandling();
    AutoMissionNasional.JumpTo(KE_STORAGE_ZONE_HABIS_SILO);

```

```

        break;

    case KE_STORAGE_ZONE_HABIS_SILO:
        // Move to storage zone after silo interaction
        print("Mission: Move to Storage Zone After Silo");
        moveRobotToStorageZone();
        if (close_to_storage_zone()) {
            AutoMissionNasional.JumpTo(CARI_BOLA_RESET);
        }
        break;

    case ROBOT_STUCK:
        // Handle robot getting stuck
        print("Mission: Robot Stuck");
        reverseRobot();
        resetAfterStuck();
        AutoMissionNasional.JumpTo(CARI_BOLA_RESET);
        break;

    default:
        print("Unknown mission");
        break;
    }
}
}
}

```

3. Misi Reset Awal Robot

Dilakukan reset berbagai parameter robot seperti odometri, posisi robot, pengontrol kecepatan PID dan Pure Pursuit, kamera, dan silo. Sudut servo kamera diarahkan ke nilai untuk mencari bola dan beberapa variabel status di-reset. Setelah reset selesai, robot melanjutkan ke misi berikutnya.

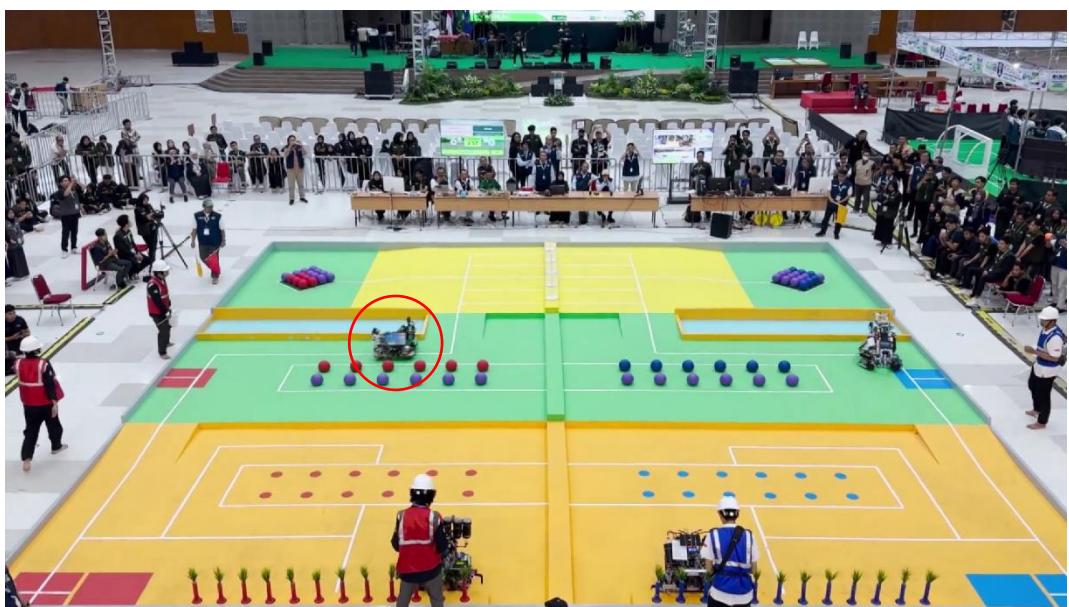
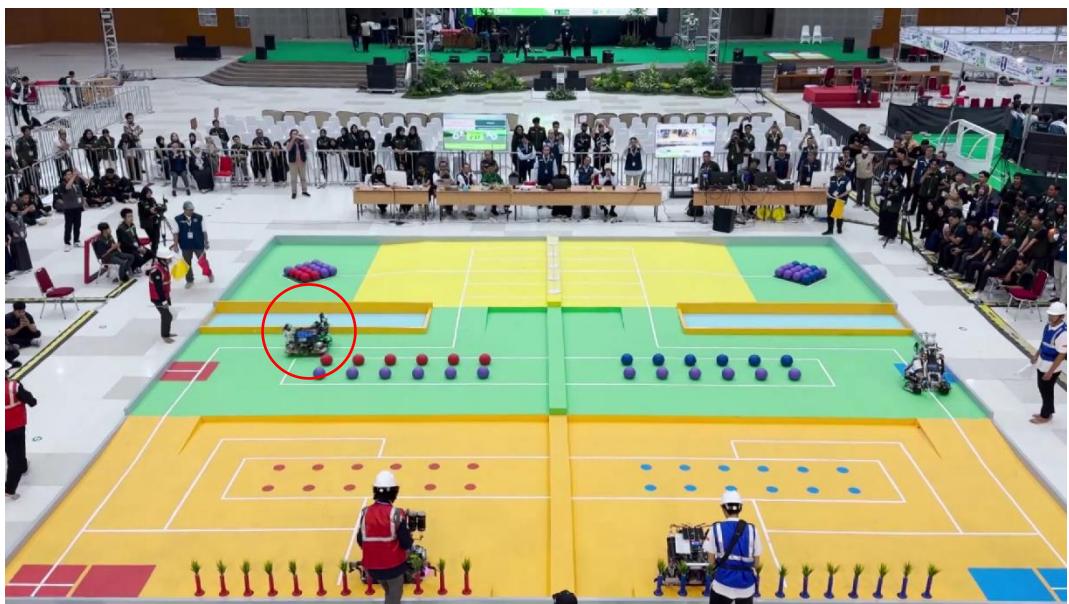


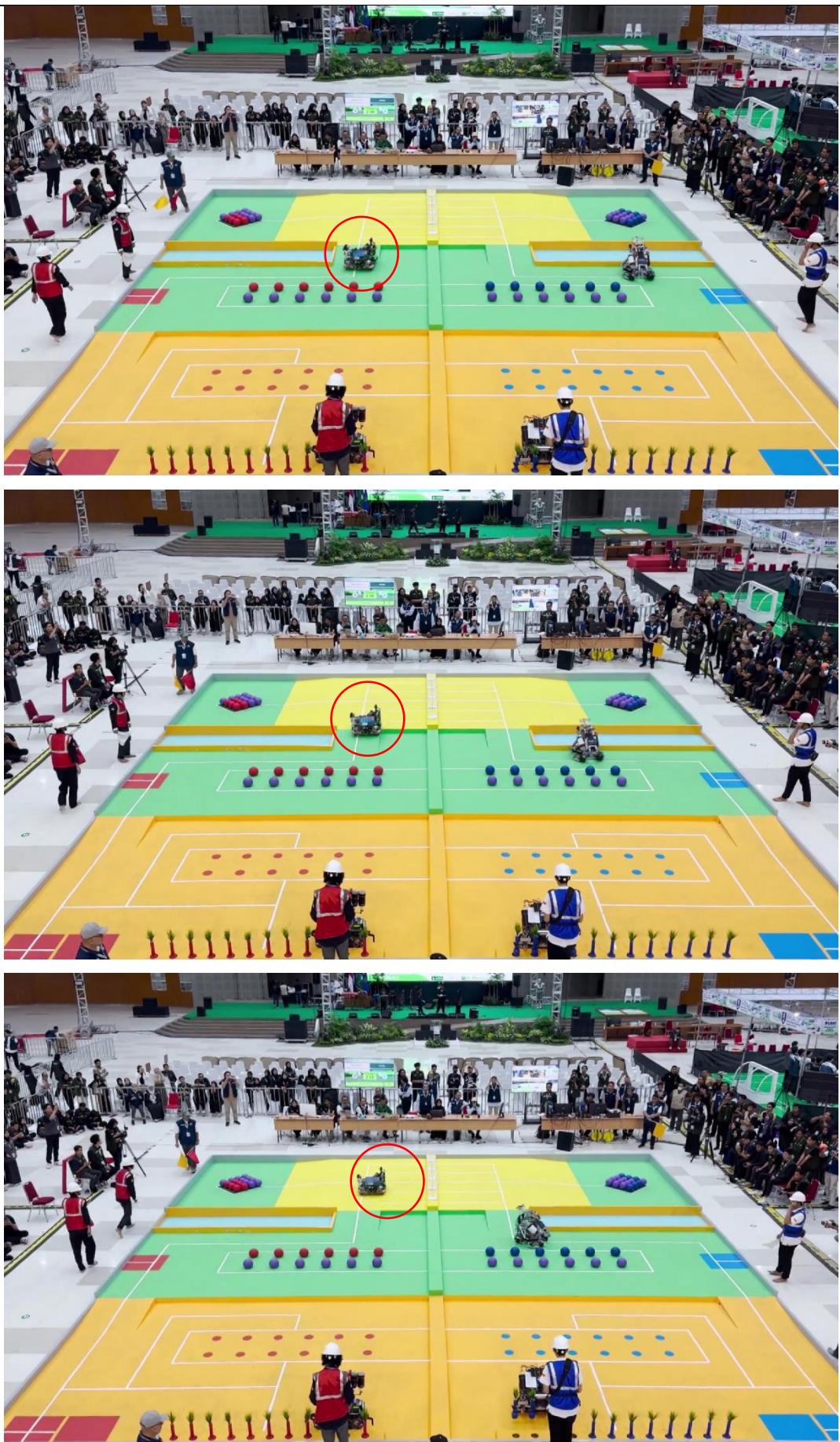
Gambar 42. Posisi Start Robot Pada Final Abu Robocon 2024 Indonesia

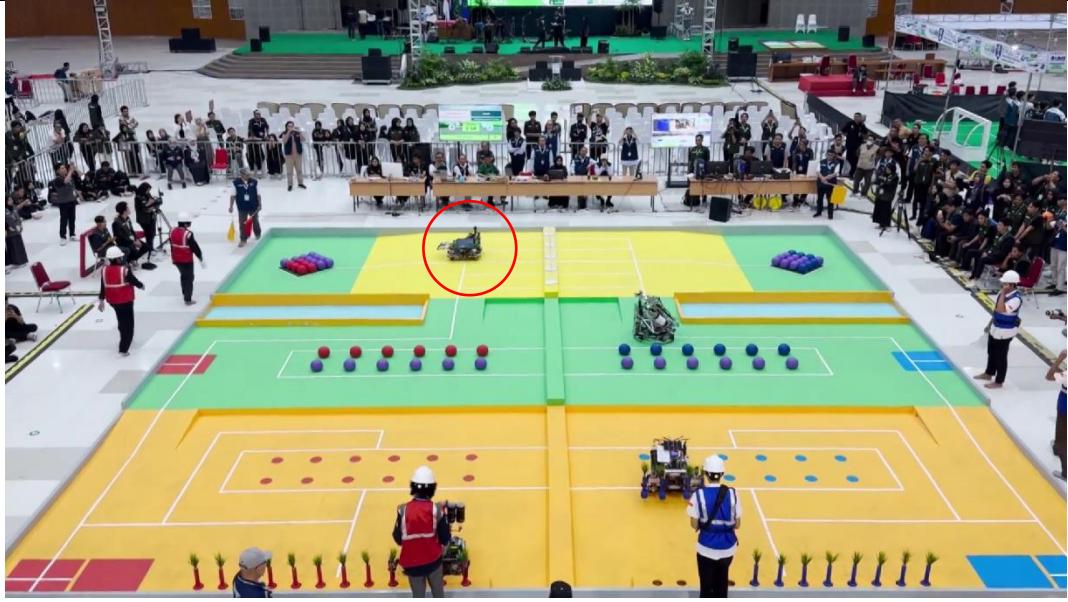
4. Misi Menuju Lantai 3

Robot menggunakan algoritma Pure Pursuit untuk menentukan koordinat tujuan berdasarkan posisi dan kecepatan saat ini. Koordinat jalur robot yang ditempuh diperoleh dari pencarian menggunakan algoritma A*. Jarak ke target akhir dihitung menggunakan jarak Euclidean antara posisi saat ini dan target. Target akhir lantai 3 yang dituju robot yaitu pada koordinat (4300, 10000) dalam mm. Robot dikendalikan menggunakan mekanisme swerve untuk menavigasi menuju target berdasarkan orientasi robot, posisi saat ini, dan kecepatan yang dikendalikan oleh pengontrol kecepatan PID. Jika jarak ke target akhir kurang dari 200 mm dan posisi LiDAR Y melebihi 8450 mm, robot akan melompat ke misi berikutnya. Ini berarti robot sudah tiba di area 3. Posisi robot di-update menggunakan data dari sensor LiDAR terkini (coordinate_x_lidar dan coordinate_y_lidar).









Gambar 43. Robot Melaksanakan Misi Menuju Lantai 3

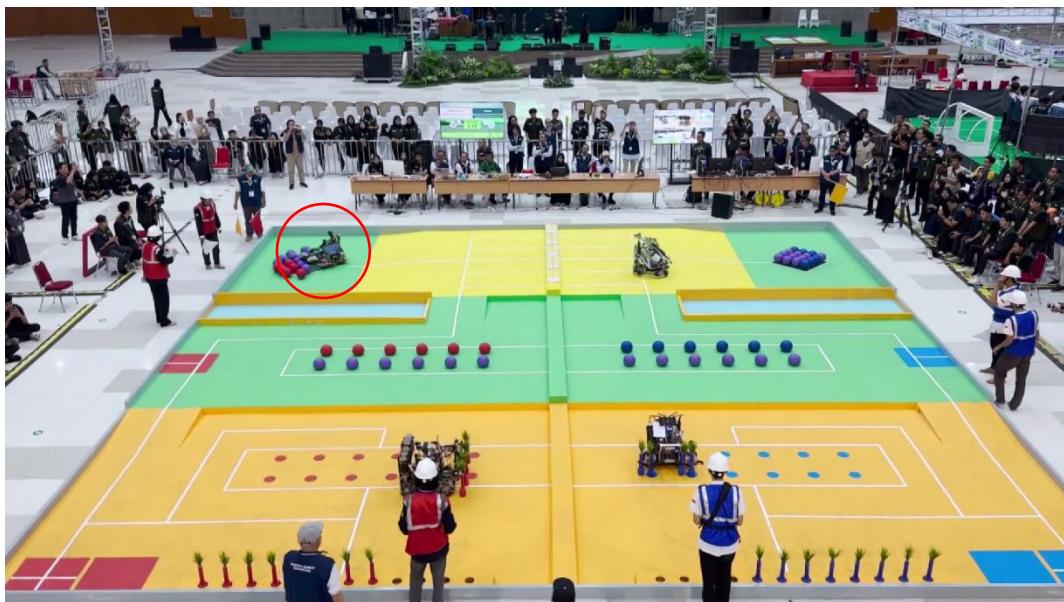
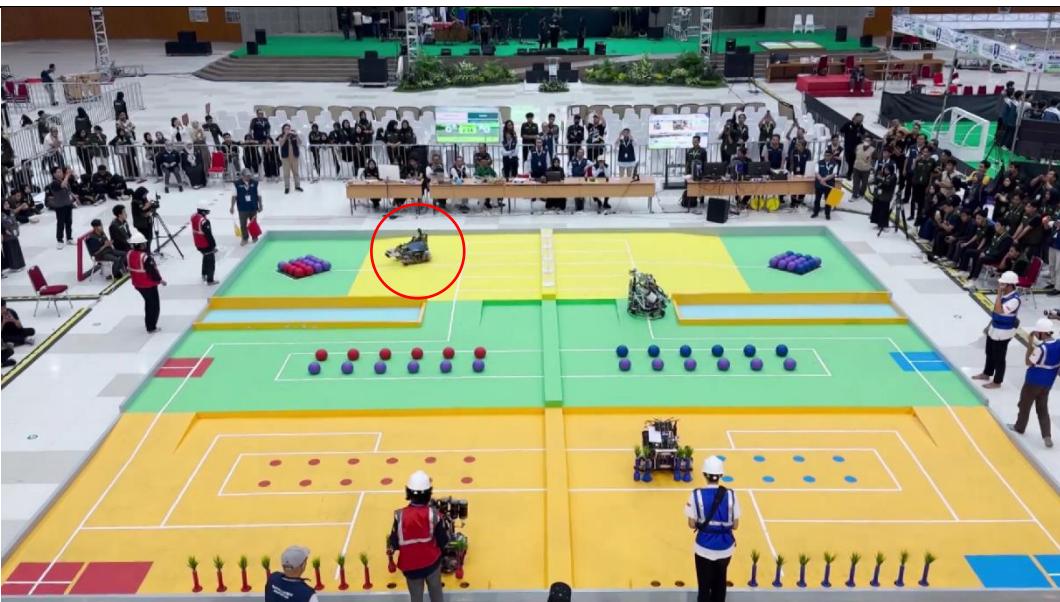
5. Misi Mencari Bola

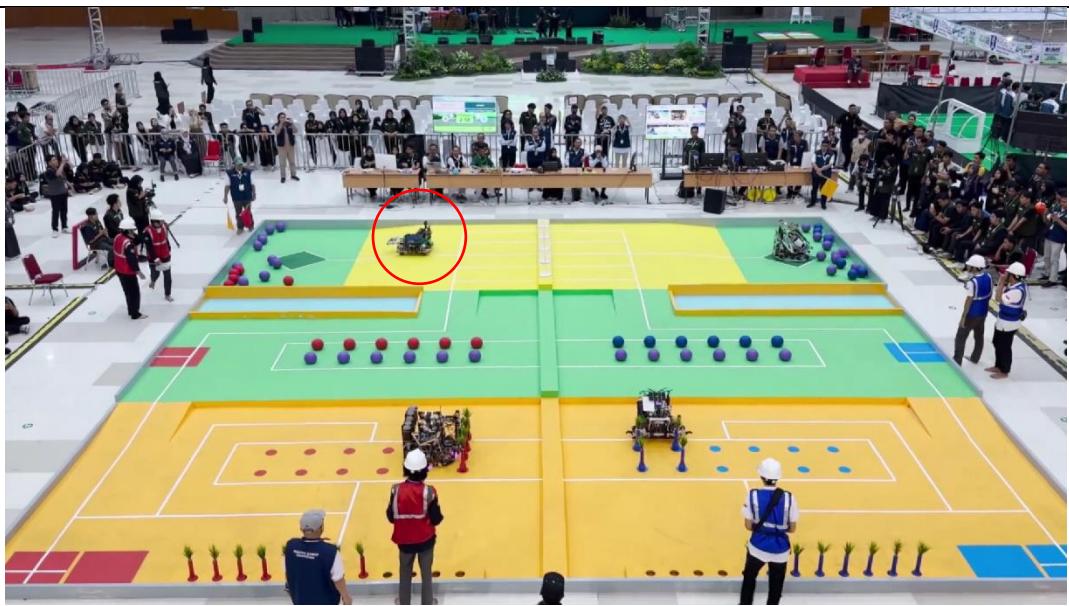
Robot menentukan titik target di lapangan (1000, 9980) untuk menabrak susunan bola dan menghitung jarak ke target menggunakan rumus jarak Euclidean. Kecepatan diatur dengan ConstantSpeedController.SetSpeed(900, 2000) dan pengendali kecepatan PID di-reset. Robot mendeteksi bola menggunakan koordinat kamera dalam x dan y dan robot menghitung jarak dan error berdasarkan posisi bola dan gripper robot. Ada beberapa skenario tergantung pada warna bola (merah atau ungu) dan kondisi tertentu. Jika bola tidak terdeteksi atau bola ungu yang ter-grip oleh robot, maka robot menangani pembuangan bola ungu. Jika robot mendeteksi bola merah, robot mencoba menggenggam bola menggunakan gripper dan menyesuaikan gerakannya. PID dan Pure Pursuit controller digunakan untuk mengontrol gerakan robot berdasarkan posisi bola.

6. Misi Mencari Silo

Setelah mendapatkan bola, robot menuju koordinat (4200, 9980) untuk mencari silo dengan prioritas tertinggi dengan mengatur sudut kamera menuju target. Jika sudah ditemukan, robot akan menggunakan data dari sensor LiDAR dan kamera untuk menghitung jarak dan kesalahan posisi relatif terhadap silo target. Jika posisi telah diketahui, maka robot akan bergerak menuju silo target menggunakan PID dan Pure Pursuit controller. Jika jarak sudah dekat, robot memperlambat kecepatan untuk

memasukkan bola ke dalam silo. Setelah berhasil memasukkan bola ke dalam silo, maka semua variabel terkait dengan misi silo diatur ulang. Robot siap untuk melanjutkan ke langkah berikutnya, yaitu kembali ke zona penyimpanan dan sudut kamera diatur untuk mencari bola kembali.

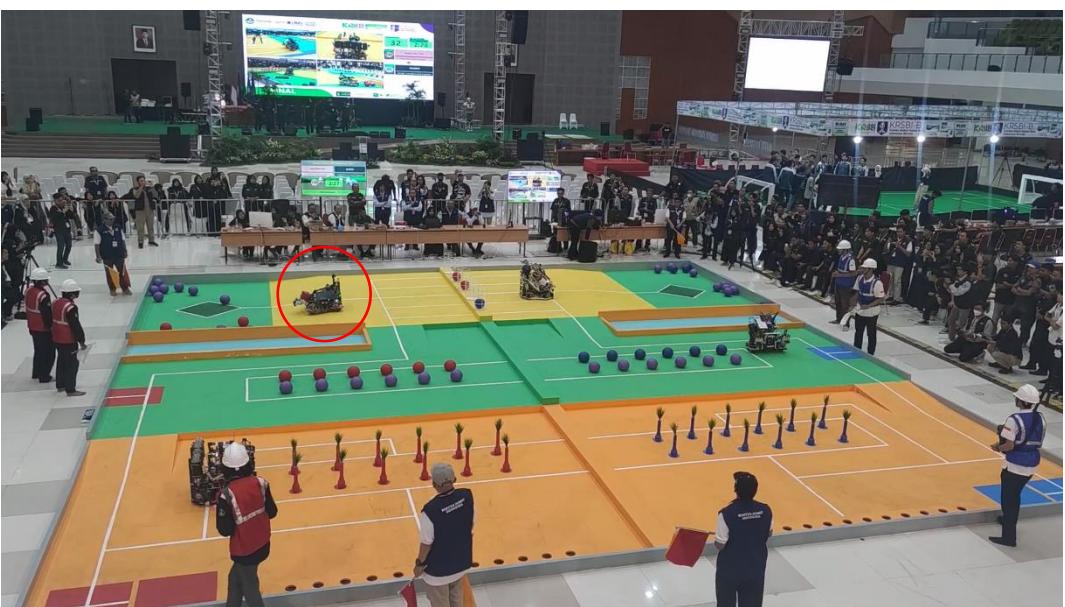






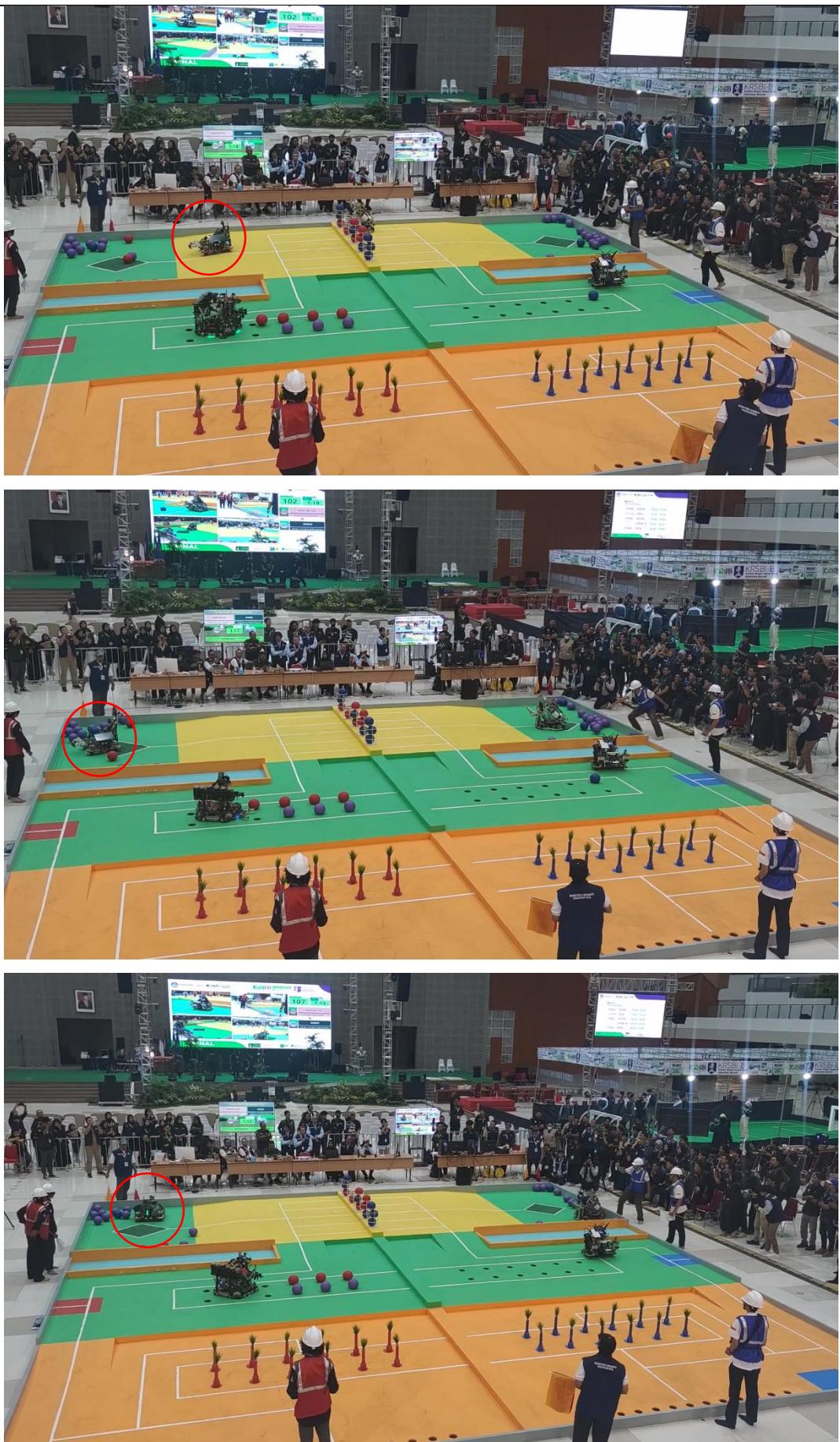
















Gambar 44. Aksi-Aksi Robot Dalam Pencarian Bola dan Pemilihan Silo Secara Otomatis

Misi mencari bola dan silo dilakukan berulang-ulang hingga game dinyatakan selesai. Setiap selesai melakukan setiap misi, dilakukan pengaturan ulang atau *reset* beberapa variabel dan parameter fungsi, seperti pengendali kecepatan PID dan Pure Pursuit, odometri atau posisi robot, dan variabel-variabel terkait pencarian bola dan silo. Hal ini penting dilakukan untuk menjaga akurasi, efisiensi, dan kemampuan robot selama menjalankan berbagai misi. Berikut adalah alasan mengapa proses reset ini diperlukan:

1. Menghindari Akumulasi Kesalahan (Odometri & Sensor)

Odometri bergantung pada sensor seperti encoder, giroskop, akselerometer, dan LIDAR untuk memperkirakan posisi robot. Seiring waktu, kesalahan kecil atau "*drift*" dalam pembacaan sensor dapat terakumulasi dan menyebabkan kesalahan yang lebih besar dalam estimasi posisi. Dengan mereset odometri, robot dapat memulai misi berikutnya dengan data posisi yang akurat.

Sensor seperti LIDAR dan kamera mungkin mengalami noise atau pembacaan yang kurang akurat karena perubahan lingkungan (misalnya, pencahayaan atau adanya rintangan). Mereset membantu mengkalibrasi ulang sensor-sensor tersebut agar data yang diperoleh akurat saat memulai misi baru.

2. Konsistensi Pengendalian PID dan Pengontrol Kecepatan untuk Navigasi Robot

PID controller mengatur gerakan robot berdasarkan perbedaan antara kondisi yang diinginkan dan kondisi yang aktual (misalnya, kecepatan atau posisi). Seiring

waktu, jika ada perubahan beban, gesekan, atau daya baterai, nilai penguat (P, I, D) mungkin perlu di-reset atau disetel ulang agar robot tetap berjalan dengan lancar. Mereset PID controller menghindari adanya "memori" dari misi sebelumnya (seperti integral windup), sehingga kontroler berfungsi optimal pada misi berikutnya.

Kecepatan yang diperlukan dan perilaku pengendalian untuk setiap misi bisa berbeda (misalnya, mendekati silo vs. bergerak menuju zona penyimpanan). Mereset pengendali kecepatan memastikan bahwa parameter yang sesuai (seperti kecepatan maksimum, akselerasi, dan faktor penyetelan) digunakan untuk setiap misi, sehingga menghindari masalah performa seperti overshoot atau respons yang lambat.

3. Penyesuaian Pure Pursuit untuk Pengikutan Jalur

Algoritma pengikut jalur ini menghitung titik target berdasarkan posisi robot saat ini. Setelah setiap misi, jalur mungkin berubah, dan robot harus menyesuaikan diri dengan koordinat atau target baru. Mereset memastikan algoritma memulai dengan data yang baru dan tidak menggunakan data lama dari misi sebelumnya, sehingga akurasi navigasi tetap terjaga. Jika robot mengikuti jalur atau lintasan yang berbeda untuk setiap misi, data lintasan sebelumnya harus dihapus. Jika tidak, robot bisa mengikuti jalur yang salah atau membuat asumsi yang keliru tentang pergerakannya.

4. Mencegah Pengambilan Keputusan yang Salah

Variabel yang terkait pencarian bola dan silo melacak status spesifik misi (misalnya, apakah bola sudah ter-grip oleh gripper, apakah bola sudah masuk ke dalam silo, apakah semua silo terdeteksi, dll.). Mereset variabel-variabel ini memastikan robot tidak membawa informasi dari satu misi ke misi lainnya, yang bisa menyebabkan pengambilan keputusan yang salah (misalnya, menganggap bola sudah dimasukkan padahal belum).

Berdasarkan analisis latihan dan simulasi pertandingan selama pengembangan sistem lokalisasi dan navigasi robot secara otonom, diperoleh hasil pengujian sistem yang dirancang:

1. Akurasi Lokalisasi dengan Multi-Sensor Fusion

Pengujian sistem lokalisasi menunjukkan bahwa integrasi multi-sensor memberikan hasil yang akurat dalam menentukan posisi dan orientasi robot di lapangan. Data dari odometri, gyroscope-accelerometer, dan LiDAR disinkronkan dengan baik untuk meminimalkan kesalahan posisi. Hasil pengujian menunjukkan bahwa kesalahan akumulatif pada odometri berhasil dikoreksi dengan data dari sensor gyroscope-accelerometer dan LiDAR, sehingga drift pada odometri dapat

diminimalkan hingga tingkat yang tidak signifikan. Rata-rata error posisi selama pengujian berada dalam rentang ± 5 mm, yang lebih dari cukup untuk memastikan robot dapat mencapai target dengan akurasi tinggi.

2. Kinerja Navigasi dengan Algoritma PID dan Pure Pursuit

Algoritma Pure Pursuit yang digunakan untuk menentukan jalur robot berdasarkan posisi dan kecepatan saat ini terbukti sangat efektif. Robot dapat mengikuti lintasan yang telah ditentukan dengan baik dan mampu melakukan koreksi jalur secara dinamis saat terjadi perubahan lingkungan, seperti pergerakan objek bola yang dituju atau adanya rintangan di lapangan. Overshoot dan error pada jalur diminimalkan oleh penggunaan PID controller, yang memastikan robot bergerak dengan kecepatan yang optimal dan stabil. Waktu respon sistem untuk menyesuaikan lintasan robot sangat cepat saat robot perlu mengubah jalurnya.

3. Efisiensi Pemrosesan Data Real-Time

Sistem pemrosesan data dari berbagai sensor, termasuk IMU, LiDAR dan kamera, berhasil dioptimalkan sehingga memungkinkan robot untuk mengambil keputusan secara real-time tanpa mengalami keterlambatan. Data dari sensor diproses dalam waktu yang cukup singkat, memungkinkan robot untuk menavigasi lapangan dengan cepat dan tepat.

Robot mampu menunjukkan ketangguhan yang tinggi selama kompetisi. Meskipun terdapat tantangan dalam lingkungan yang dinamis, seperti pergerakan robot lain, kondisi bola dan silo yang sedikit berbeda, dan kondisi pencahayaan yang bervariasi, robot tetap dapat melokalisasi dirinya dengan baik dan menyelesaikan semua misi yang diberikan. Sistem navigasi berhasil menjaga akurasi lintasan dan deteksi objek meskipun kondisi lingkungan yang berubah-ubah.

Pada kompetisi ABU ROBOCON 2024 Indonesia, tim MAESTRO EVO Universitas Negeri Yogyakarta berhasil menjadi juara 1 dari 32 tim yang mengikuti dari berbagai perguruan tinggi di Indonesia. Tim kami berhak mewakili Indonesia ke Vietnam, setelah mengalahkan tim RIVONE dari Institut Teknologi Sepuluh Nopember di final dengan kemenangan “*mua vàng*” (kemenangan mutlak). Kemenangan ini membuktikan keberhasilan desain sistem lokalisasi dan navigasi robot otomatis berbasis swerve drive yang dirancang secara otonom. Dengan mengintegrasikan berbagai sensor seperti encoder untuk odometri, gyroscope-accelerometer, LiDAR, dan kamera, robot mampu menyelesaikan semua tugas dengan tingkat akurasi yang tinggi. Penggunaan multi-sensor fusion, dengan kombinasi odometri, gyroscope-

accelerometer, LiDAR, dan kamera, terbukti mampu mengatasi kelemahan dari masing-masing sensor individu dan memberikan hasil yang akurat dan reliabel. Sistem ini mampu beradaptasi dengan baik terhadap perubahan lingkungan di lapangan dan memberikan kontrol yang presisi dalam setiap misi.

Keunggulan lain dari sistem ini adalah efisiensi pemrosesan data, yang memungkinkan robot untuk bekerja dalam kondisi real-time tanpa penundaan yang signifikan. Ini menjadi salah satu faktor kunci dalam keberhasilan robot menyelesaikan misi-misinya dengan cepat dan tepat. Secara keseluruhan, sistem yang dirancang berhasil memenuhi tujuan penelitian, yaitu menciptakan robot yang mampu melokalisasi dan menavigasi secara mandiri di lingkungan yang dinamis. Kemenangan mutlak yang diraih tim MAESTRO EVO membuktikan bahwa pendekatan ini sangat kompetitif dan dapat diandalkan dalam lingkungan kompetisi berteknologi tinggi seperti ABU Robocon 2024.

BAB V

SIMPULAN DAN SARAN

A. Simpulan

Kemenangan pada kompetisi ABU ROBOCON 2024 Indonesia membuktikan sistem lokalisasi dan navigasi robot otomatis berbasis swerve-drive yang kami rancang dapat diunggulkan. Berdasarkan penelitian yang dilakukan, dapat diambil kesimpulan untuk menjawab permasalahan yang telah dirumuskan. Odometri bergantung pada sensor seperti encoder, giroskop, akselerometer, dan LIDAR untuk memperkirakan posisi robot. Dengan mereset odometri global robot setiap selesai melakukan misi, robot dapat memulai misi berikutnya dengan data posisi yang akurat dan menghindari akumulasi kesalahan atau drift pada sistem odometri.

Kalibrasi giroskop dan akselerometer melibatkan perhitungan offset untuk menghilangkan kesalahan pembacaan data. Kalibrasi gyroscope dilakukan dengan mendiamkan ponsel di permukaan datar guna menghilangkan bias atau drift data saat sensor diam. Akselerometer dikalibrasi dengan memutar ponsel secara di sekitar ketiga sumbunya (x, y, z). Ini memastikan akselerometer dapat mengukur gaya yang bekerja pada perangkat, terutama gaya gravitasi, di seluruh sumbu secara akurat.

Untuk merancang sistem lokalisasi dan navigasi robot yang mampu menyesuaikan pergerakan secara dinamis dengan perubahan lingkungan di lapangan, diperlukan kombinasi algoritma pengendalian yang adaptif serta sensor yang dapat memberikan data secara real-time. Dalam penelitian ini, penggunaan algoritma Pure Pursuit dan PID controller pada robot swerve-drive terbukti mampu memberikan kendali yang akurat dan fleksibel, memungkinkan robot untuk menavigasi lingkungan yang dinamis. Robot juga menggunakan data dari sensor seperti LiDAR dan kamera untuk memperbarui posisinya secara kontinu, sehingga dapat menyesuaikan lintasan saat mendekati target.

Penggunaan beberapa sensor seperti odometri, giroskop-akselerometer, LiDAR, dan kamera memainkan peran penting dalam meningkatkan akurasi dan efisiensi sistem navigasi robot otonom. Integrasi data dari berbagai sensor dilakukan dengan cara yang efisien melalui pemrosesan multi-sensor fusion, di mana data dari masing-masing sensor digabungkan untuk memberikan estimasi posisi robot yang lebih tepat. Integrasi filter Mahony untuk giroskop-akselerometer guna mengatasi masalah noise pada sinyal sensor.

B. Implikasi

Hasil penelitian ini memiliki beberapa implikasi penting bagi pengembangan robot otonom di masa depan. Integrasi multi-sensor (odometri, gyroscope-accelerometer, LiDAR, dan kamera) terbukti meningkatkan akurasi dan efisiensi sistem lokalisasi dan navigasi, yang dapat menjadi pedoman untuk penerapan dalam pengembangan robot atau kendaraan otonom. Keberhasilan sistem navigasi dan lokalisasi ini di kompetisi ABU Robocon 2024 juga mengindikasikan standar baru bagi tim lain dalam pengembangan robot otonom, yang semakin fokus pada integrasi sensor.

C. Saran

Berdasarkan temuan penelitian, terdapat beberapa saran untuk pengembangan lebih lanjut. Pertama, perlu dikembangkan algoritma kontrol yang lebih canggih, seperti model prediktif (MPC) atau teknologi berbasis kecerdasan buatan (AI), agar robot dapat lebih adaptif terhadap kondisi lingkungan yang semakin kompleks. Kedua, disarankan untuk mengembangkan metode kalibrasi sensor otomatis guna menjaga akurasi data tanpa memerlukan kalibrasi manual yang rutin. Ketiga, pengujian sistem perlu diperluas ke lingkungan yang lebih beragam, seperti medan outdoor dengan permukaan tidak rata, untuk memastikan ketahanan sistem di berbagai kondisi.

DAFTAR PUSTAKA

- Ahmad, H., & Namerikawa, T. (2010). H Infinity Filter-based Robotic Localization and Mapping with Intermittent Measurements. *The 8th France-Japan and 6th Europe-Asia Congress on Mechatronics*.
- Coulter, R. (1992). *Implementation of the Pure Pursuit Path Tracking Algorithm*. Pennsylvania: Carnegie Mellon University.
- Natakusuma, B. P. (2018). *APLIKASI SENSOR INERTIA MEASUREMENT UNIT (IMU) UNTUK MEMPERBAIKI GERAK BERJALAN*. Surabaya: Institut Teknologi Sepuluh Nopember.
- Palacin, J., Rubies, E., & Clotet, E. (2022). Systematic Odometry Error Evaluation and Correction in a Human-Sized Three-Wheeled Omnidirectional Mobile Robot Using Flower-Shaped Calibration Trajectories. *Applied Sciences*, 12(5), 2606.
- Reis, D., Kupec, J., Hong, J., & Ahmad, D. (2024). Real-Time Flying Object Detection with YOLOv8. *arXiv*.
- Susanto, A. P. (2018). *AUTONOMOUS DOCKING SYSTEM UNTUK MOBILE*. Surabaya: Institut Teknologi Sepuluh Nopember.
- Susanto, J. (2016). *NAVIGASI MOBILE ROBOT MENGGUNAKAN DYNAMIC PATH PLANNING ALGORITHM BERBASIS GENETIC ALGORITHM*. Surabaya: Institut Teknologi Sepuluh Nopember.
- Utomo, E. B. (2015). *AUTONOMOUS MOBILE ROBOT BERBASIS LANDMARK MENGGUNAKAN PARTICLE FILTER DAN OCCUPANCY GRID MAPS UNTUK NAVIGASI, PENENTUAN POSISI, DAN PEMETAAN*. Surabaya: INSTITUT TEKNOLOGI SEPULUH NOPEMBER .
- Veness, T. (2020). *Controls Engineering in the FIRST Robotics Competition*. California: University of California.
- Ward, B. (2019). *Game Manual 0*. Retrieved from <https://gm0.org/en/latest/docs/software/concepts/odometry.html>
- Zhao, S. e. (2021). Super Odometry: IMU-centric LiDAR-Visual-Inertial Estimator for Challenging Environments. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 8729-8736.

LAMPIRAN

Gambar-Gambar Robot Dalam Jarak Dekat dan Menjalankan Misi

