BAB II
DASAR TEORI

2.1 Tinjauan Pustaka

Perkembangan robot mobile seperti *Automated Guided Vehicle* (AGV) dan sistem navigasi robot terus meningkat seiring kebutuhan otomasi industri. AGV digunakan untuk memindahkan barang secara otomatis dan meningkatkan efisiensi operasional [9]. Pada penelitian awal, navigasi robot umumnya menggunakan metode konvensional seperti PID dan sistem berbasis aturan (*rule-based*), misalnya pada robot *line follower*. Selain itu, fitur seperti *anti-collision* juga dikembangkan untuk meningkatkan keamanan sistem. Namun, metode tersebut masih memiliki keterbatasan karena tidak adaptif terhadap perubahan lingkungan [10].

Seiring perkembangan kecerdasan buatan, *reinforcement learning* mulai diterapkan pada navigasi robot [3]. Metode ini memungkinkan robot belajar dari interaksi dengan lingkungan melalui mekanisme *reward* dan *punishment*. Salah satu algoritma yang banyak digunakan adalah Q-*Learning* yang mampu menentukan aksi optimal secara mandiri [11]. Beberapa penelitian juga mengembangkan metode seperti *Fuzzy Q-Learning* untuk meningkatkan performa navigasi, namun umumnya masih terbatas pada simulasi atau jenis robot tertentu [8], [12].

Untuk mengetahui posisi penelitian ini, berikut disajikan perbandingan penelitian terdahulu:

Tabel 2.1 *State of the art*

No	Judul Penelitian	Peneliti & Tahun	Fokus Penelitian	Keterbaruan Penelitian Ini
1	Implementasi Sistem Kendali Keseimbangan Statis pada Robot Quadruped Menggunakan	Saputro et al. (2023)	Mengembangkan sistem kendali untuk menjaga keseimbangan robot quadruped dengan memanfaatkan proses pembelajaran berbasis *trial* and error sehingga robot mampu	Menerapkan RL pada navigasi AGV berbasis grid, bukan hanya kestabilan

	Reinforcement learning[13]		mempertahankan posisi stabil terhadap gangguan	
2	Robot Obstacle Avoidance dengan Algoritma Q-*Learning*[7]	Agustian et al. (2020)	Merancang sistem navigasi robot yang mampu memilih aksi gerakan secara optimal berdasarkan nilai *reward* untuk menghindari rintangan dalam lingkungan sederhana	Mengembangkan multi-*goal* navigation dengan lingkungan lebih kompleks
3	Rancang Bangun Sistem Robot AGV untuk Penyortiran Paket Ekspedisi dengan Fitur Anti Collision[10]	Tamara et al. (2020)	Membangun sistem AGV yang dapat bergerak dan mendeteksi rintangan menggunakan sensor serta menentukan aksi berdasarkan logika yang telah ditentukan (rule-based) untuk proses penyortiran paket	Menggunakan Q-*learning* untuk navigasi adaptif, bukan rule-based
4	A Method of Obstacle Avoidance for AMR Robot in Warehouse Automation[14]	Ngọc et al. (2022)	Mengembangkan sistem navigasi robot AMR di lingkungan warehouse dengan memanfaatkan data sensor untuk mendeteksi rintangan dan menentukan arah gerak berdasarkan	Menggunakan *reinforcement learning* untuk pengambilan keputusan adaptif

			aturan yang telah ditetapkan	
5	Implementasi Metode Waypoint pada Sistem Navigasi Automated Guided Vehicle (AGV)[9]	Ananta et al. (2025)	Mengimplementasikan metode waypoint untuk mengarahkan pergerakan AGV mengikuti jalur yang telah ditentukan sebelumnya sehingga robot bergerak secara terstruktur menuju titik tujuan	Mengembangkan navigasi tanpa jalur tetap menggunakan Q-*learning* berbasis grid

Berdasarkan Tabel 2.1, dapat diketahui bahwa berbagai penelitian telah menerapkan metode navigasi robot, mulai dari sistem berbasis aturan (*rule-based*), *path planning*, hingga *reinforcement learning*. Meskipun demikian, sebagian besar penelitian masih berfokus pada pengendalian gerak, penghindaran rintangan, atau navigasi dengan jalur yang telah ditentukan (*fixed path*). Selain itu, implementasi *Reinforcement Learning* umumnya masih diterapkan pada simulasi atau jenis robot tertentu sehingga implementasi *policy* hasil pembelajaran pada robot mobile di lingkungan multi-jalur masih terbatas.

Oleh karena itu, penelitian ini mengusulkan penerapan algoritma Q-Learning pada sistem navigasi AGV berbasis *grid* dengan skenario *multi-goal*. Berbeda dengan penelitian sebelumnya, sistem yang dikembangkan tidak bergantung pada jalur tetap, melainkan menentukan keputusan arah berdasarkan *policy* hasil pembelajaran yang diperoleh melalui mekanisme *reward* dan *punishment*. Dengan demikian, penelitian ini diharapkan dapat menghasilkan sistem navigasi AGV yang mampu menerapkan *policy* navigasi pada lingkungan terstruktur dengan beberapa tujuan.

2.2 Mobile Robot dan *Automated Guided Vehicle* (AGV)

Robot mobile merupakan jenis robot yang memiliki kemampuan untuk berpindah dari satu lokasi ke lokasi lain secara mandiri tanpa memerlukan

intervensi manusia secara langsung. Robot ini umumnya dilengkapi dengan sistem penggerak, sensor, serta sistem kontrol yang memungkinkan robot melakukan navigasi dalam suatu lingkungan [7]. Mobile robot banyak digunakan dalam berbagai bidang seperti industri manufaktur, logistik, hingga sistem transportasi otomatis.

Dalam lingkungan industri modern, penggunaan robot mobile semakin berkembang seiring meningkatnya kebutuhan akan sistem otomasi yang mampu meningkatkan efisiensi proses produksi dan distribusi material . Salah satu penerapan robot mobile dalam industri adalah Automated Guided Vehicle (AGV) dan Autonomous Mobile Robot (AMR), yang keduanya berfungsi sebagai sistem transportasi material secara otomatis.

AGV merupakan kendaraan robotik yang beroperasi dengan mengikuti jalur yang telah ditentukan sebelumnya, seperti garis (line follower), magnetic tape, atau marker lantai sebagai panduan navigasi [15]. Metode ini relatif sederhana dan mudah diimplementasikan, sehingga banyak digunakan pada sistem industri yang memiliki lingkungan terstruktur. Namun, AGV memiliki keterbatasan dalam hal fleksibilitas karena jalur navigasi harus dimodifikasi secara manual ketika terjadi perubahan tata letak lingkungan [9].

Berbeda dengan AGV, AMR memiliki kemampuan navigasi yang lebih cerdas karena dapat menentukan jalur pergerakan secara mandiri tanpa bergantung pada jalur fisik yang tetap. AMR memanfaatkan sensor dan algoritma navigasi untuk memahami lingkungan serta melakukan perencanaan jalur secara dinamis, sehingga lebih fleksibel dan adaptif terhadap perubahan kondisi lingkungan [10].

Gambar 2.1 AMR dan AGV [16]

Berdasarkan Gambar 2.1, AGV dan AMR memiliki perbedaan pada metode navigasi yang digunakan. AGV bergerak mengikuti jalur fisik yang telah ditentukan, sedangkan AMR memanfaatkan sensor dan algoritma navigasi untuk mengenali lingkungan serta menentukan jalur pergerakan secara mandiri. Perbedaan tersebut menunjukkan bahwa AMR memiliki fleksibilitas yang lebih tinggi dalam beroperasi pada lingkungan yang dinamis [16].

Seiring berkembangnya teknologi kecerdasan buatan dan robotika, berbagai pendekatan baru mulai dikembangkan untuk meningkatkan kemampuan navigasi *auto mobile*. Salah satu pendekatan yang banyak diteliti adalah sistem navigasi otonom yang memungkinkan robot menentukan jalur pergerakan secara mandiri tanpa bergantung pada jalur fisik yang telah ditentukan sebelumnya. Pendekatan ini memungkinkan sistem robot mobile melakukan navigasi tanpa bergantung pada jalur fisik melalui penerapan *policy* hasil pembelajaran.

Pada penelitian ini, sistem auto mobile dirancang menggunakan pendekatan navigasi berbasis *Reinforcement learning* yang memungkinkan robot mempelajari kebijakan navigasi melalui interaksi dengan lingkungan.

2.3 Sistem Navigasi Robot Mobile

Navigasi merupakan salah satu kemampuan utama yang harus dimiliki oleh robot mobile agar dapat bergerak dari suatu lokasi menuju lokasi tujuan secara mandiri. Sistem navigasi robot bertujuan untuk menentukan posisi robot dalam suatu lingkungan serta merencanakan jalur pergerakan yang harus ditempuh untuk mencapai tujuan yang telah ditentukan [7].

Secara umum, sistem navigasi robot terdiri dari beberapa komponen utama yaitu persepsi lingkungan, perencanaan jalur, dan kontrol pergerakan. Persepsi lingkungan diperoleh melalui berbagai sensor yang digunakan oleh robot untuk mendeteksi kondisi sekitar. Informasi yang diperoleh dari sensor kemudian diproses oleh sistem kontrol untuk menentukan keputusan pergerakan robot menuju tujuan tertentu.

Dalam pengembangan sistem navigasi robot mobile, terdapat beberapa pendekatan navigasi yang umum digunakan, antara lain:

1) Navigasi Berbasis Jalur (*Guided Navigation*)

Navigasi berbasis jalur merupakan metode navigasi yang menggunakan jalur fisik sebagai panduan pergerakan robot. Jalur tersebut dapat berupa garis pada lantai, pita magnetik, atau marker tertentu yang dideteksi oleh sensor robot. Metode ini banyak digunakan pada sistem AGV konvensional karena memiliki implementasi yang relatif sederhana [15]. Namun metode ini memiliki keterbatasan dalam hal fleksibilitas karena jalur navigasi harus dipasang secara fisik dan sulit diubah ketika terjadi perubahan tata letak lingkungan [9].

2) Navigasi Otonom (Autonomous Navigation)

Navigasi otonom memungkinkan robot untuk menentukan jalur pergerakan secara mandiri tanpa bergantung pada jalur fisik yang telah ditentukan sebelumnya. Dalam pendekatan ini robot memanfaatkan informasi dari sensor untuk memahami kondisi lingkungan serta menggunakan algoritma tertentu untuk menentukan keputusan pergerakan [10]. Metode navigasi otonom umumnya memanfaatkan teknik seperti *path planning*, *localization*, dan *obstacle avoidance* untuk menghasilkan pergerakan robot yang optimal.

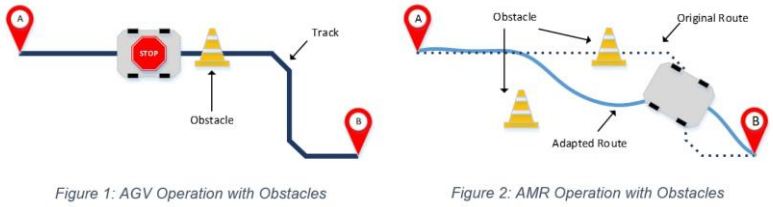


Figure 1: AGV Operation with Obstacles Figure 2: AMR Operation with Obstacles

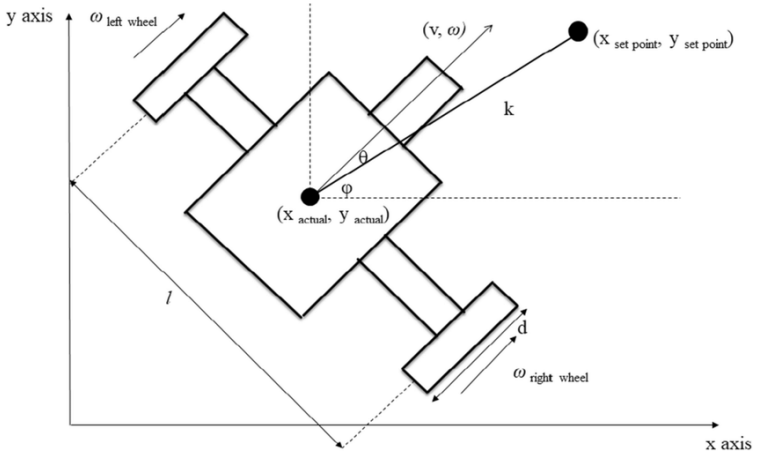
Gambar 2.2 Perbedaan Navigasi jalur dan otonom [17]

Berdasarkan Gambar 2.2, navigasi berbasis jalur (*guided navigation*) mengharuskan robot mengikuti lintasan yang telah ditentukan sebelumnya, seperti garis atau pita magnetik, sehingga pergerakannya kurang fleksibel terhadap perubahan lingkungan. Sebaliknya, navigasi otonom memungkinkan robot menentukan jalur secara mandiri berdasarkan informasi dari sensor dan algoritma navigasi pada lingkungan yang telah dipetakan. [17].

2.4 Sistem Gerak Robot Differential Drive

Differential drive merupakan konfigurasi penggerak robot yang menggunakan dua roda penggerak utama yang ditempatkan pada sisi kiri dan kanan robot. Kedua roda tersebut digerakkan secara independen sehingga robot dapat bergerak maju, berbelok, maupun berputar pada tempat dengan cara mengatur perbedaan kecepatan putaran roda kiri dan roda kanan. Pada sistem ini, variasi

kecepatan masing-masing roda akan menghasilkan perubahan arah dan lintasan pergerakan robot secara langsung. [18]



Gambar 2.3 Sistem penggerak differential drive [19]

Berdasarkan Gambar 2.3, sistem *differential drive* menggunakan dua roda penggerak utama yang bekerja secara independen pada sisi kiri dan kanan robot. Perbedaan kecepatan putaran kedua roda menghasilkan berbagai manuver, seperti bergerak lurus, berbelok, maupun berputar di tempat, sehingga konfigurasi ini banyak digunakan pada robot mobile karena sederhana dan mudah dikendalikan [19].

2.4.1 Konfigurasi Mekanik Differential Drive

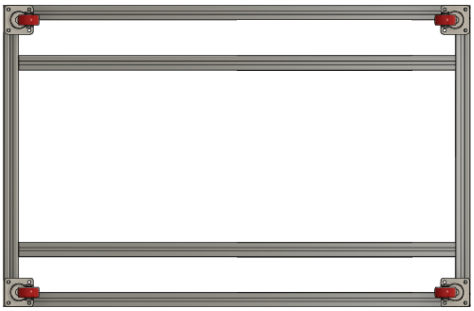
Robot mobile dapat menggunakan berbagai jenis konfigurasi sistem penggerak seperti *Ackermann steering*, *omnidirectional drive*, *skid steering*, dan *differential drive*. Pemilihan konfigurasi sistem penggerak sangat mempengaruhi kemampuan manuver, kompleksitas mekanik, serta metode kontrol robot. Pada penelitian ini digunakan konfigurasi *differential drive* karena memiliki struktur mekanik yang sederhana serta mudah dikontrol untuk aplikasi navigasi robot mobile [18].

Differential drive merupakan sistem penggerak robot yang menggunakan dua roda penggerak utama yang ditempatkan pada sisi kiri dan kanan robot. Kedua roda tersebut digerakkan secara independen oleh motor sehingga memungkinkan robot untuk melakukan berbagai jenis pergerakan seperti bergerak lurus, berbelok, maupun berputar pada tempat [18].

Arah pergerakan robot ditentukan oleh perbedaan kecepatan antara roda kiri dan roda kanan. Apabila kedua roda berputar dengan kecepatan yang sama maka

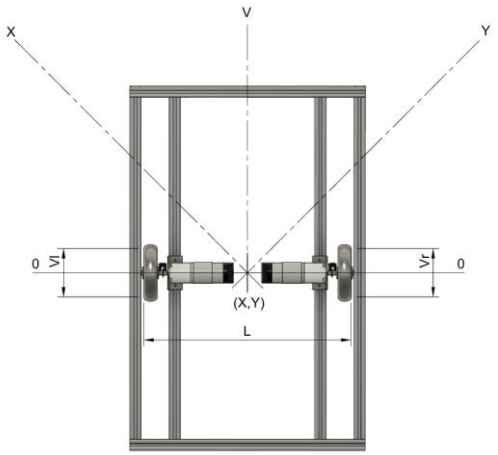
robot akan bergerak lurus. Sebaliknya, apabila terdapat perbedaan kecepatan antara kedua roda maka robot akan bergerak mengikuti lintasan melingkar.

Berdasarkan Gambar 2.4, roda penyeimbang dipasang pada bagian depan atau belakang robot sebagai penopang tambahan. Penggunaan roda ini membantu menjaga keseimbangan robot dan mengurangi gesekan selama proses manuver tanpa memberikan gaya penggerak.



Gambar 2.4 Roda Penyeimbang

Berdasarkan Gambar 2.5, roda penggerak terhubung langsung dengan poros motor sehingga menghasilkan gaya dorong yang menggerakkan robot. Kecepatan putar roda kiri dan roda kanan diatur secara independen untuk menghasilkan berbagai manuver pada sistem *differential drive*. Dalam penelitian ini, robot *mobile* menggunakan konfigurasi *differential drive* dengan dua motor penggerak utama, yaitu satu motor pada sisi kiri dan satu motor pada sisi kanan robot. Robot dilengkapi dengan empat roda pada *chassis*, terdiri atas dua roda penggerak utama dan dua roda penyeimbang yang tidak digerakkan oleh motor. Konfigurasi ini membentuk sistem *two-wheel drive* yang sederhana, stabil, dan mendukung pergerakan robot selama proses navigasi.



Gambar 2.5 Roda Penggerak

Dalam sistem *differential drive* terdapat beberapa parameter geometri yang penting untuk menentukan karakteristik pergerakan robot. Parameter-parameter tersebut ditunjukkan pada Tabel 2.2

Tabel 2.2 Parameter geometri sistem differential drive

Parameter	Keterangan
r	jari-jari roda
L	jarak antara roda kiri dan kanan
v_L	kecepatan roda kiri
v_R	kecepatan roda kanan
v	kecepatan linear robot
ω	kecepatan sudut robot

2.4.2 Derajat Kebebasan Robot (*Degree of Freedom*)

Derajat kebebasan atau *Degree of Freedom* (DOF) merupakan jumlah gerakan independen yang dapat dilakukan oleh suatu sistem mekanik. Dalam konteks robot mobile, DOF menggambarkan jumlah variabel independen yang diperlukan untuk menentukan posisi dan orientasi robot dalam suatu ruang.

Robot mobile yang bergerak pada bidang dua dimensi memiliki tiga derajat kebebasan utama, yaitu dua gerakan translasi pada sumbu horizontal dan vertikal serta satu gerakan rotasi terhadap sumbu vertikal. Dengan demikian posisi robot pada bidang dua dimensi dapat dinyatakan dalam bentuk koordinat sebagai berikut:

$$P = (x, y, \theta) \tag{2.1}$$

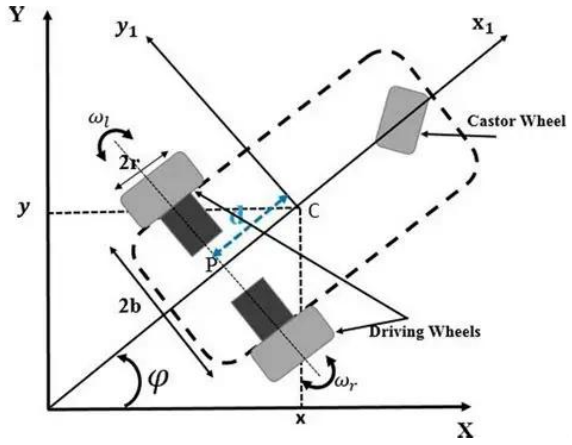
dengan:

x= posisi robot pada sumbu horizontal

y= posisi robot pada sumbu vertikal

θ= orientasi robot terhadap sumbu referensi

Persamaan 2.1 merupakan representasi *pose* robot pada bidang dua dimensi yang digunakan untuk menyatakan posisi dan arah robot dalam sistem koordinat global. Gambar 2.6 menunjukkan sistem koordinat pada robot mobile, yang menggambarkan hubungan antara posisi dan orientasi robot terhadap lingkungan.



Gambar 2.6 Sistem koordinat roda robot mobile[20]

Dalam sistem navigasi robot mobile, tiga parameter utama yaitu posisi x, posisi y, dan orientasi θdigunakan untuk menentukan lokasi robot di dalam lingkungan. Perubahan posisi robot terhadap waktu dapat dinyatakan sebagai berikut:

$$\dot{x} = v\cos(\theta)$$
$$\dot{y} = v\sin(\theta) \quad (2.2)$$
$$\dot{\theta} = \omega$$

dengan:

v= kecepatan linear robot

ω= kecepatan sudut robot

Persamaan 2.2 menunjukkan hubungan antara kecepatan robot dengan perubahan posisi dan orientasi robot pada bidang dua dimensi [21].

2.4.3 Kinematika Robot *Differential Drive*

Kinematika robot merupakan cabang ilmu yang mempelajari hubungan antara gerakan mekanik robot tanpa mempertimbangkan gaya yang menyebabkan gerakan tersebut. Dalam robot mobile, analisis kinematika digunakan untuk menentukan hubungan antara kecepatan putaran roda dengan pergerakan robot secara keseluruhan.

Pada robot dengan sistem differential drive, pergerakan robot ditentukan oleh kecepatan putaran roda kiri dan roda kanan. Dengan mengatur kecepatan masing-masing roda, robot dapat melakukan berbagai jenis pergerakan seperti bergerak lurus, berbelok, maupun berputar pada tempat[18].

Kecepatan linear robot merupakan rata-rata dari kecepatan roda kiri dan roda kanan. Persamaan kecepatan linear robot dapat dinyatakan sebagai berikut:

$$v = \frac{v_R + v_L}{2} \quad (2.3)$$

dimana:

v= kecepatan linear robot

v_R= kecepatan roda kanan

v_L= kecepatan roda kiri

Persamaan (2.3) menunjukkan bahwa robot akan bergerak lurus apabila kedua roda memiliki kecepatan yang sama.

Selain kecepatan linear, robot juga memiliki kecepatan sudut yang menunjukkan perubahan orientasi robot terhadap waktu. Kecepatan sudut robot dapat dihitung menggunakan persamaan:

$$\omega = \frac{v_R - v_L}{L} \quad (2.4)$$

dimana:

ω= kecepatan sudut robot

L= jarak antara roda kiri dan roda kanan

Persamaan (2.4) menunjukkan bahwa perbedaan kecepatan antara roda kiri dan roda kanan akan menyebabkan robot mengalami rotasi [21].

Kecepatan linear roda berhubungan dengan kecepatan sudut roda melalui persamaan:

$$v = \omega r \quad (2.5)$$

dimana:

v= kecepatan linear roda

ω= kecepatan sudut roda

r= jari-jari roda

Persamaan (2.5) menunjukkan bahwa kecepatan linear roda dipengaruhi oleh kecepatan sudut dan ukuran jari-jari roda.

Jika kecepatan putaran roda dinyatakan dalam satuan RPM, maka kecepatan sudut roda dapat dihitung menggunakan persamaan:

$$\omega = \frac{2\pi \times RPM}{60} \quad (2.6)$$

Persamaan 2.6 digunakan untuk menghubungkan spesifikasi motor dengan kecepatan gerak robot.

Berdasarkan hubungan kecepatan roda kiri dan kanan, robot *differential drive* dapat melakukan beberapa jenis pergerakan seperti yang ditunjukkan pada Tabel 2.3.

Tabel 2.3 Jenis Pergerakan Robot Differential Drive

No	Jenis Pergerakan	Hubungan Kecepatan	Deskripsi
1	Gerak Lurus	$v_R = v_L$	Robot bergerak lurus tanpa perubahan arah.
2	Belok ke Kanan	$v_L > v_R$	Robot berbelok ke arah kanan karena roda kiri berputar lebih cepat.
3	Belok ke Kiri	$v_R > v_L$	Robot berbelok ke arah kiri karena roda kanan berputar lebih cepat.
4	Rotasi pada Tempat	$v_R = -v_L$	Robot berputar pada tempat tanpa mengalami perpindahan posisi.

Analisis kinematika robot sangat penting dalam sistem navigasi karena menentukan bagaimana perintah kecepatan roda diterjemahkan menjadi pergerakan robot di lingkungan.

Dalam sistem auto mobile yang dikembangkan pada penelitian ini, algoritma navigasi berbasis *Reinforcement learning* menentukan aksi pergerakan robot seperti bergerak maju, berbelok ke kiri, atau berbelok ke kanan. Perintah tersebut kemudian diterjemahkan oleh sistem kontrol motor menjadi kecepatan roda kiri dan roda kanan yang sesuai. Dengan demikian model kinematika robot menjadi dasar dalam menghubungkan keputusan navigasi dengan pergerakan robot secara fisik.

2.4.4 Gaya dan Torsi pada Robot Mobile

Gaya merupakan besaran yang mempengaruhi pergerakan suatu benda. Pada robot mobile, gaya yang bekerja meliputi gaya berat, gaya normal, dan gaya gesek antara roda dan permukaan lantai. Pemahaman terhadap gaya-gaya ini diperlukan untuk menentukan kemampuan robot dalam bergerak serta kebutuhan torsi pada sistem penggerak.

Gaya berat merupakan gaya yang bekerja pada robot akibat pengaruh gravitasi bumi dan dapat dinyatakan sebagai berikut:

$$W = m \cdot g \tag{2.7}$$

di mana Wadalah gaya berat (N), $m$adalah massa benda (kg), dan $g$adalah percepatan gravitasi (9,81 m/s²).

Pada kondisi permukaan datar tanpa percepatan vertikal, gaya normal yang diberikan oleh permukaan terhadap robot memiliki nilai yang sama dengan gaya berat, sehingga dapat dinyatakan sebagai:

$$N = W \tag{2.8}$$

Gaya gesek merupakan gaya yang timbul akibat kontak antara roda dan permukaan lantai. Besarnya gaya gesek dipengaruhi oleh koefisien gesek dan gaya normal, dan dapat dinyatakan sebagai:

$$F = \mu \cdot N \tag{2.9}$$

di mana Fadalah gaya gesek (N), μadalah koefisien gesek, dan Nadalah gaya normal (N).

Torsi merupakan besaran yang menunjukkan kemampuan gaya dalam memutar suatu benda terhadap titik pusat. Pada robot mobile, torsi digunakan untuk menggerakkan roda sehingga menghasilkan pergerakan robot. Torsi dapat dinyatakan sebagai:

$$T = F \cdot r \tag{2.10}$$

di mana Tadalah torsi (Nm), Fadalah gaya (N), dan $r$adalah jari-jari roda (m).

Dalam beberapa spesifikasi motor, torsi dinyatakan dalam satuan kilogram-centimeter (kg·cm). Hubungan antara satuan Newton-meter dan kilogram-centimeter adalah sebagai berikut:

$$1 \text{ Nm} = 10.197 \text{ kgf.cm} \tag{2.11}$$

Konsep gaya dan torsi ini menjadi dasar dalam menentukan kebutuhan tenaga penggerak robot serta pemilihan motor yang sesuai agar robot dapat bergerak secara stabil.

2.4.5 Pengendalian Kecepatan Motor Menggunakan PWM

Motor DC merupakan salah satu aktuator yang paling banyak digunakan dalam sistem robot mobile karena memiliki struktur yang sederhana, mudah dikendalikan, serta mampu menghasilkan torsi yang cukup besar pada kecepatan

rendah. Kecepatan putaran motor DC berbanding lurus dengan tegangan yang diberikan pada terminal motor, sehingga pengaturan tegangan menjadi metode utama dalam mengontrol kecepatan motor[22].

Pada sistem kontrol berbasis mikrokontroler, pengaturan tegangan motor biasanya dilakukan menggunakan metode *Pulse Width Modulation* (PWM). PWM merupakan teknik pengaturan tegangan rata-rata dengan cara mengubah lebar pulsa sinyal digital yang diberikan pada motor driver[15].

PWM bekerja dengan menghasilkan sinyal digital yang memiliki dua kondisi yaitu *HIGH* dan *LOW*. Besarnya tegangan rata-rata yang diterima oleh motor ditentukan oleh *duty cycle* dari sinyal PWM.

Duty cycle merupakan perbandingan antara waktu sinyal berada pada kondisi HIGH terhadap periode total sinyal PWM. Duty cycle dapat dinyatakan sebagai berikut:

$$D = \frac{T_{on}}{T} \tag{2.12}$$

dimana:

D= duty cycle

T_{on}= waktu sinyal berada pada kondisi HIGH

T= periode sinyal PWM

Tegangan rata-rata yang diberikan pada motor dapat dihitung menggunakan persamaan berikut:

$$V_{out} = D \times V_{in} \tag{2.13}$$

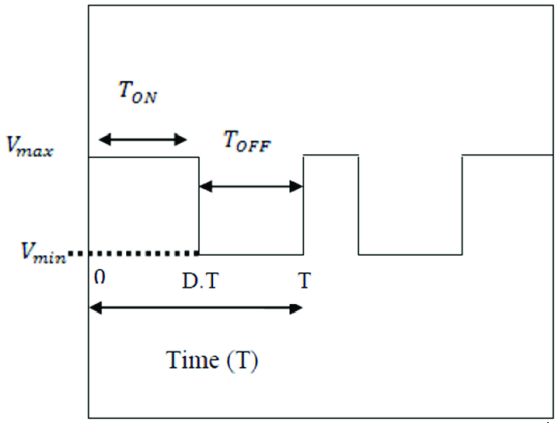
dimana:

V_{out}= tegangan rata-rata motor

D= duty cycle PWM

V_{in}= tegangan sumber

Persamaan tersebut menunjukkan bahwa semakin besar nilai duty cycle maka tegangan rata-rata yang diberikan kepada motor akan semakin besar sehingga kecepatan motor juga meningkat [21].



Gambar 2.7 Sinyal PWM [23]

Berdasarkan Gambar 2.7, sinyal PWM terdiri atas waktu aktif TON dan waktu tidak aktif TOFF dalam satu periode (*T*). Perbandingan antara waktu aktif terhadap satu periode disebut *duty cycle*, yang menentukan besar tegangan rata-rata yang diterima motor. Semakin besar nilai *duty cycle*, semakin besar tegangan rata-rata yang diberikan sehingga kecepatan putaran motor akan meningkat [21].

Dalam penelitian ini, sinyal PWM digunakan untuk mengatur kecepatan motor DC pada robot *mobile*. Nilai PWM dihasilkan oleh Arduino Mega 2560 berdasarkan perintah navigasi yang dikirimkan oleh Raspberry Pi, sehingga robot dapat melakukan pergerakan maju, belok kiri, dan belok kanan sesuai keputusan yang diperoleh dari algoritma *Q-Learning*.

2.5 Sensor pada Sistem Robot Mobile

Sensor merupakan komponen penting dalam sistem robot mobile karena berfungsi untuk memperoleh informasi mengenai kondisi lingkungan maupun kondisi internal robot. Informasi yang diperoleh dari sensor digunakan oleh sistem kontrol untuk menentukan keputusan pergerakan robot. Dalam sistem navigasi robot, sensor berperan dalam proses persepsi lingkungan (perception) yang menjadi dasar dalam proses pengambilan keputusan [7].

Pada robot mobile, sensor dapat dibedakan menjadi dua kategori utama yaitu sensor internal (*proprioceptive* sensor) dan sensor eksternal (*exteroceptive* sensor). Sensor internal digunakan untuk mengukur kondisi internal robot seperti kecepatan roda, orientasi, dan posisi relatif robot. Sedangkan sensor eksternal digunakan untuk memperoleh informasi mengenai lingkungan di sekitar robot seperti keberadaan objek, rintangan, maupun penanda lokasi [7].

2.5.1 Sensor Jarak E18-D80NK

Sensor E18-D80NK merupakan sensor proximity berbasis inframerah yang digunakan untuk mendeteksi keberadaan objek di depan sensor melalui prinsip refleksi cahaya inframerah. Sensor ini memancarkan sinar inframerah menggunakan LED IR dan menerima pantulan cahaya menggunakan fototransistor[24]. Ketika objek berada dalam jangkauan sensor, cahaya yang dipantulkan akan diterima oleh fotodetektor dan diproses oleh komparator internal sehingga menghasilkan sinyal keluaran digital.

Secara umum prinsip kerja sensor dapat dijelaskan sebagai berikut:

a. LED inframerah memancarkan cahaya ke arah depan sensor.
b. Jika terdapat objek, sebagian cahaya akan dipantulkan kembali.
c. Fotodetektor menerima pantulan cahaya tersebut.
d. Komparator internal mengubah sinyal menjadi output digital.

Tabel spesifikasi utama sensor E18-D80NK ditunjukkan pada Tabel 2.4 **.**

Tabel 2.4 Spesifikasi Sensor E18-D80NK [24]

Parameter	Spesifikasi
Tegangan kerja	5 V DC
Jarak deteksi	3 – 80 cm
Jenis output	Digital (HIGH/LOW)
Arus kerja	±25 mA
Jenis sensor	Infrared reflective

Jarak deteksi sensor dapat diatur menggunakan potensiometer yang terdapat pada modul. Dalam sistem AGV, sensor ini digunakan untuk mendeteksi obstacle statis yang berada di jalur robot. Output sensor berupa sinyal digital sehingga dapat langsung dibaca oleh GPIO Raspberry Pi tanpa memerlukan konversi analog-ke-digital. Ketika objek terdeteksi, sensor akan menghasilkan perubahan logika pada pin output yang digunakan sebagai indikator keberadaan hambatan di depan robot.

2.5.2 MPU6050 (Inertial Measurement Unit)

Robot mobile dapat menggunakan berbagai jenis konfigurasi sistem penggerak seperti Ackermann steering, omnidirectional drive, skid steering, dan differential drive. Pemilihan konfigurasi sistem penggerak sangat mempengaruhi kemampuan manuver, kompleksitas mekanik, serta metode kontrol robot.

MPU6050 merupakan sensor Inertial Measurement Unit (IMU) yang menggabungkan accelerometer tiga sumbu dan gyroscope tiga sumbu dalam satu chip. Sensor ini digunakan untuk mengukur percepatan linier serta kecepatan sudut pada sumbu x, y, dan z [24]. Tabel 2.5 menunjukkan spesifikasi utama dari MPU6050.

Tabel 2. 5 Spesifikasi Sensor MPU6050 [25]

Parameter	Spesifikasi
Tegangan kerja	3.3 V
Interface komunikasi	I2C / SPI
Accelerometer	±2g, ±4g, ±8g, ±16g
Gyroscope	±250, ±500, ±1000, ±2000 °/s
Resolusi	16-bit

Dalam sistem navigasi AGV, informasi yang paling penting adalah kecepatan sudut pada sumbu yaw yang diperoleh dari *gyroscope*, karena digunakan untuk menentukan arah hadap robot. Nilai kecepatan sudut tersebut kemudian diolah menjadi sudut orientasi dengan memperhitungkan perubahan terhadap waktu.
Pada sistem digital, perhitungan sudut dilakukan secara bertahap berdasarkan data pembacaan sensor, yang dapat dinyatakan sebagai:

$$\theta_k = \theta_{k-1} + \omega \cdot \Delta t \tag{2.14}$$

Persamaan 2.14 menunjukkan bahwa sudut orientasi robot diperbarui secara terus-menerus berdasarkan kecepatan sudut dan selang waktu pengambilan data.

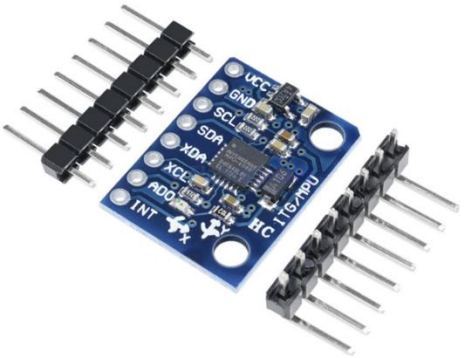
Nilai sudut yang diperoleh kemudian didiskritisasi menjadi empat arah utama untuk menyederhanakan representasi *state* pada algoritma *Q-Learning*, yaitu seperti yang ditunjukkan pada Tabel 2.6.

Tabel 2.6 Diskritisasi Arah Berdasarkan Sudut

Rentang Sudut	Arah
-45° hingga +45°	North
-135° hingga -45°	East
-180° hingga -135° dan +135° hingga +180°	South
+45° hingga +135°	West

Berdasarkan Tabel 2.6, nilai *heading* dari sensor IMU didiskritisasi menjadi empat arah utama, yaitu North pada rentang −45° hingga +45° (berpusat pada 0°), East pada rentang −135° hingga −45° (berpusat pada −90°), South pada rentang

−180° hingga −135° serta +135° hingga +180° (berpusat pada ±180°), dan West pada rentang +45° hingga +135° (berpusat pada +90°). Pembagian rentang sudut tersebut digunakan untuk menentukan orientasi robot sebagai salah satu *state* pada algoritma Q-Learning.



Gambar 2.8 Sensor MPU6050 [26]

Diskritisasi orientasi tersebut digunakan untuk menentukan kondisi state robot sehingga proses pengambilan keputusan pada algoritma Q-*Learning* menjadi lebih sederhana dan konsisten terhadap arah hadap robot.

Untuk mengetahui tingkat akurasi estimasi orientasi, dilakukan perhitungan error heading dengan membandingkan sudut target terhadap sudut orientasi hasil pembacaan sensor. Error heading dihitung menggunakan Persamaan 2.15.

$$e_h = | \theta_{target} - \theta_{aktual} | \tag{2.15}$$

dengan:

- e_h= error heading (°),
- θ_{target}= sudut orientasi yang diharapkan (°),
- θ_{aktual}= sudut orientasi hasil pembacaan MPU6050 (°).

Selain menghitung error pada setiap pengujian, tingkat akurasi keseluruhan sensor juga dievaluasi menggunakan Mean Absolute Error (MAE) yang dihitung berdasarkan seluruh data heading hasil pengujian. Persamaan MAE ditunjukkan pada Persamaan 2.16.

$$MAE = \frac{1}{n}\sum_{i=1}^{n} | \theta_{target} - \theta_i | \tag{2.16}$$

dengan:

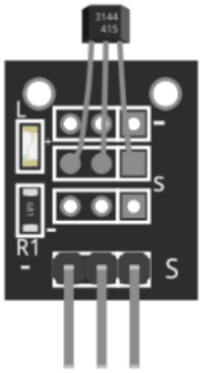
- MAE= Mean Absolute Error (°),
- n= jumlah data pengujian,

- θ_{target}= sudut orientasi yang diharapkan (°),
- θ_i= sudut orientasi hasil pembacaan sensor pada pengujian ke-i.

Semakin kecil nilai error heading maupun MAE, maka semakin tinggi tingkat akurasi sensor dalam mengestimasi orientasi robot sehingga performa navigasi yang dihasilkan menjadi lebih baik.

2.5.3 Sensor KY-003 Hall Effect Magnetic

Sensor KY-003 Hall Effect Magnetic Sensor merupakan modul sensor berbasis efek Hall yang digunakan untuk mendeteksi keberadaan medan magnet. Sensor ini sering digunakan sebagai rotary encoder sederhana dengan memanfaatkan magnet yang dipasang pada roda atau poros motor. Ketika magnet melewati sensor, medan magnet akan terdeteksi dan menghasilkan sinyal digital berupa pulsa [27].



Gambar 2.9 Sensor KY-003 Hall Effect Magnetic [27]

Prinsip kerja sensor ini didasarkan pada Hall Effect, yaitu fenomena munculnya tegangan listrik ketika konduktor yang dialiri arus berada dalam medan magnet. Tegangan Hall dapat dinyatakan sebagai:

$$V_H = \frac{I \cdot B}{n \cdot q \cdot t} \tag{2.17}$$

Keterangan:

V_H= tegangan Hall (V)

I= arus listrik pada sensor (A)

B= kerapatan fluks magnet (T)

n= jumlah pembawa muatan

q= muatan elektron

t= ketebalan material sensor

Pada modul KY-003, perubahan tegangan akibat medan magnet akan diproses oleh rangkaian komparator internal sehingga menghasilkan output digital HIGH atau LOW.

Tabel 2. 7 Spesifikasi Sensor KY-003 Hall Effect Magnetic [27]

Parameter	Spesifikasi
Tegangan kerja	3.3 – 5 V
Jenis sensor	Hall Effect
Output	Digital
IC utama	A3144 Hall Sensor
Interface	VCC, GND, Signal
Sensitivitas	Deteksi magnet permanen

Dalam sistem robot mobile, magnet kecil dipasang pada roda atau poros motor. Setiap kali magnet melewati sensor Hall, modul akan menghasilkan satu pulsa. Jumlah pulsa yang dihasilkan sebanding dengan jumlah putaran roda.

Jika dalam satu putaran roda dihasilkan N pulsa, maka sudut rotasi roda dapat dihitung dengan:

$$\theta = \frac{2\pi}{N} \times P \tag{2.18}$$

Keterangan:

P= jumlah pulsa yang terbaca

N= jumlah pulsa per putaran

Jarak tempuh roda kemudian dihitung menggunakan hubungan antara sudut rotasi dan jari-jari roda:

$$s = r\theta \tag{2.19}$$

Keterangan:

s= jarak tempuh roda

r= jari-jari roda

θ= sudut rotasi roda

Perhitungan jarak tempuh berdasarkan jumlah putaran roda tersebut merupakan prinsip dasar odometry, yaitu metode estimasi perpindahan posisi robot menggunakan data encoder pada roda. Informasi odometri dimanfaatkan untuk menentukan jarak yang telah ditempuh robot pada setiap perpindahan grid selama proses navigasi.

Untuk mengetahui tingkat akurasi hasil pengukuran odometri, dilakukan perhitungan persentase error jarak dengan membandingkan jarak hasil pengukuran terhadap jarak target yang telah ditentukan. Persentase error dihitung menggunakan Persamaan 2.20.

$$\text{Error Jarak} = \frac{|\ S_{aktual} - S_{teoritis}\ |}{S_{teoritis}} \times 100\% \quad (2.20)$$

Keterangan:

S_{aktual}= jarak hasil pengukuran menggunakan meteran (mm)

$S_{teoritis}$= jarak target atau jarak yang direncanakan (mm)

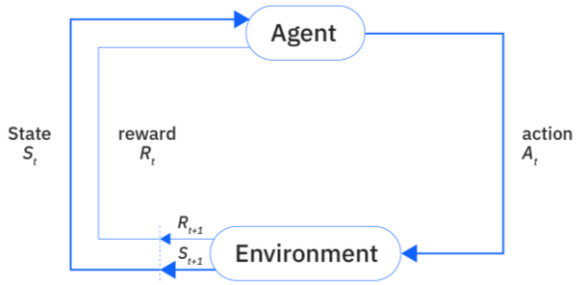
Pada penelitian ini, robot dirancang bergerak sejauh 650 mm untuk setiap satu perpindahan grid sehingga nilai $S_{teoritis}$yang digunakan dalam perhitungan error adalah 650 mm. Semakin kecil nilai persentase error yang diperoleh, maka semakin tinggi tingkat akurasi sensor Hall Effect dalam mengestimasi jarak tempuh robot.

2.6 *Reinforcement learning*

Reinforcement learning (RL) merupakan salah satu metode pembelajaran dalam bidang kecerdasan buatan yang memungkinkan suatu agen untuk belajar melalui interaksi langsung dengan lingkungan. Dalam metode ini, agen melakukan serangkaian aksi dan menerima umpan balik berupa *reward* atau punishment dari lingkungan sebagai evaluasi terhadap aksi yang dilakukan [3].

Berbeda dengan metode pembelajaran lainnya seperti supervised *learning* yang membutuhkan data berlabel, *Reinforcement learning* tidak memerlukan dataset eksplisit. Agen belajar secara bertahap melalui proses eksplorasi lingkungan untuk menemukan kebijakan (*policy*) yang mampu memaksimalkan akumulasi *reward* dalam jangka Panjang[1].

Konsep dasar *Reinforcement learning* terdiri dari beberapa komponen utama yaitu *agent*, *environment*, *state*, *action*, dan *reward*. *Agent* merupakan sistem yang melakukan aksi dalam lingkungan, sedangkan environment merupakan lingkungan tempat agen berinteraksi. State merepresentasikan kondisi lingkungan pada suatu waktu tertentu, *action* merupakan aksi yang dapat dilakukan oleh agen, dan *reward* merupakan nilai evaluasi yang diberikan oleh lingkungan terhadap aksi yang dilakukan agen.



Gambar 2.10 Konsep dasar *Reinforcement learning* [28]

Dalam sistem *Reinforcement learning*, agen akan melakukan proses pengambilan keputusan secara berulang. Pada setiap langkah, agen mengamati state lingkungan, memilih aksi tertentu, kemudian menerima *reward* dari lingkungan serta berpindah ke *state* berikutnya. Proses ini dilakukan secara terus menerus hingga agen mampu menemukan kebijakan optimal yang menghasilkan *reward* maksimum.

2.6.1 Komponen Dasar *Reinforcement learning*

Beberapa komponen utama dalam sistem *Reinforcement learning* dapat dijelaskan sebagai berikut:

a) *Agent*

Agent merupakan entitas yang melakukan aksi dalam lingkungan dan belajar melalui interaksi untuk mencapai tujuan tertentu. Dalam *Reinforcement learning*, *agent* memperoleh pengetahuan dengan menerima umpan balik berupa *reward* dari lingkungan atas setiap aksi yang dilakukan [3].

b) Environment

Environment merupakan lingkungan tempat agen berinteraksi. Dalam penelitian ini, environment direpresentasikan sebagai grid environment yang menggambarkan posisi robot, tujuan navigasi, serta obstacle yang terdapat dalam lingkungan[29].

c) State

State merupakan representasi kondisi lingkungan pada suatu waktu tertentu. Dalam sistem navigasi robot, state biasanya merepresentasikan posisi dan orientasi robot dalam lingkungan [29].

Pada penelitian ini state didefinisikan sebagai:

$$S = (x, y, \theta, g) \tag{2.21}$$

dengan:

x= posisi robot pada sumbu x

y= posisi robot pada sumbu y

θ= orientasi robot

g= tujuan navigasi yang aktif

d) *Action*

Action merupakan aksi yang dapat dilakukan oleh agen pada suatu state tertentu. Dalam sistem navigasi robot, aksi yang dapat dilakukan robot meliputi:

- bergerak maju (*forward*)
- belok kiri (*turn left*)
- belok kanan (*turn right*)

Aksi tersebut akan menghasilkan perubahan posisi robot dalam lingkungan navigasi.

e) *Reward*

Reward merupakan nilai evaluasi yang diberikan oleh lingkungan terhadap aksi yang dilakukan oleh agen. *Reward* digunakan untuk mengarahkan agen agar memilih aksi yang menghasilkan hasil terbaik [3].

Contoh *reward* yang digunakan dalam sistem navigasi robot antara lain:

- *reward* positif ketika mencapai tujuan
- *reward* negatif ketika menabrak obstacle
- penalti kecil untuk setiap langkah pergerakan

2.6.2 Algoritma Q-*Learning*

Q-*Learning* merupakan salah satu algoritma *Reinforcement learning* yang paling banyak digunakan dalam berbagai permasalahan navigasi robot. Algoritma ini diperkenalkan oleh Watkins dan Dayan [2] dan termasuk dalam kategori *model-free reinforcement learning*, yaitu algoritma yang tidak memerlukan model matematika dari lingkungan.

Q-*Learning* memungkinkan agen untuk belajar melalui interaksi langsung dengan lingkungan menggunakan pendekatan *trial and error*, di mana agen akan menerima *reward* sebagai evaluasi dari aksi yang dilakukan [8]. Pendekatan ini banyak diterapkan pada sistem robot otonom karena mampu beradaptasi terhadap lingkungan yang tidak diketahui sebelumnya [11].

Dalam Q-*Learning* digunakan sebuah fungsi nilai yang disebut *Q-function*, yang berfungsi untuk menyimpan nilai kualitas suatu aksi pada state tertentu. Nilai ini menggambarkan seberapa baik suatu aksi dilakukan dalam jangka panjang untuk mencapai tujuan tertentu [3]

Fungsi Q dapat dinyatakan sebagai:

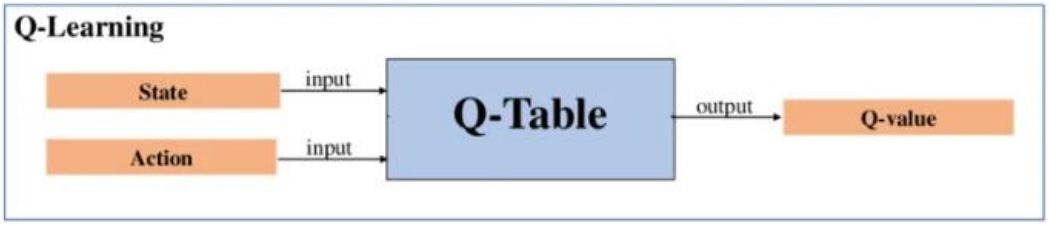
$$Q(s, a) \tag{2.22}$$

dimana:

s= state

a= *action*

Nilai Q akan diperbarui setiap kali agen melakukan aksi dan menerima *reward* dari lingkungan.



Gambar 2.11 Konsep dasar *Q-Learning* [30]

2.6.3 Persamaan Update Q-*Learning*

Proses pembaruan nilai Q dalam algoritma Q-*Learning* dapat dinyatakan dengan persamaan berikut:

$$Q(s, a) = Q(s, a) + \alpha[r + \gamma \max Q(s', a') - Q(s, a)] \tag{2.23}$$

dimana:

$Q(s, a)$= nilai Q pada state s dan *action* a

α= *learning* rate

γ= discount factorz

r= *reward* yang diperoleh

s'= state berikutnya

a'= *action* berikutnya

Persamaan tersebut menunjukkan bahwa nilai Q diperbarui berdasarkan kombinasi antara *reward* yang diperoleh pada kondisi saat ini serta estimasi *reward* maksimum pada kondisi berikutnya. Proses ini dilakukan secara iteratif hingga nilai Q konvergen dan menghasilkan kebijakan (*policy*) yang optimal.

Learning rate (α) merupakan parameter yang mengatur seberapa besar pengaruh informasi baru terhadap nilai Q sebelumnya. Nilai α berada pada rentang $0 < \alpha \leq 1$ [3] dimana nilai yang lebih kecil akan menghasilkan proses pembelajaran yang lebih stabil namun cenderung lebih lambat.

Discount factor (γ) digunakan untuk menentukan tingkat kepentingan *reward* di masa depan. Nilai γberada pada rentang $0 \leq \gamma < 1$ [3], dimana nilai yang mendekati 1 menunjukkan bahwa agen mempertimbangkan *reward* jangka panjang dalam proses pengambilan keputusan.

Reward (r) merupakan umpan balik yang diberikan oleh lingkungan terhadap aksi yang dilakukan oleh agen. Dalam konteks navigasi robot, *reward* umumnya dirancang berdasarkan kondisi lingkungan, seperti pemberian nilai positif ketika robot mencapai tujuan serta nilai negatif ketika robot mengalami kondisi yang tidak diinginkan, misalnya mendekati atau mengenai obstacle.

Persamaan ini merupakan dasar dari algoritma Q-*Learning* dalam memperbarui nilai aksi secara iteratif berdasarkan pengalaman interaksi agen dengan lingkungan [2]. Pendekatan ini termasuk dalam kategori *model-free reinforcement learning*, karena agen dapat mempelajari kebijakan optimal tanpa memerlukan model lingkungan secara eksplisit. Dengan mekanisme tersebut, agen akan belajar untuk memilih aksi yang memaksimalkan *cumulative reward* selama proses pembelajaran berlangsung, sehingga banyak diterapkan dalam sistem navigasi robot untuk menghasilkan jalur yang optimal [31].

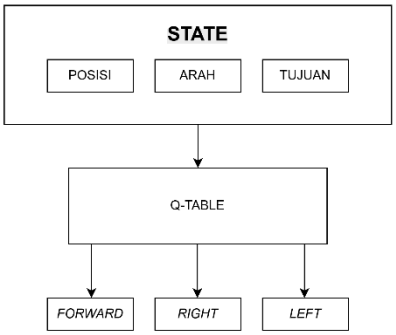
2.6.4 Q-Table

Dalam implementasi *Q-Learning* pada lingkungan diskrit, nilai Q biasanya disimpan dalam sebuah tabel yang disebut *Q-table*. *Q-table* menyimpan nilai kualitas setiap pasangan *state* dan *action* [13]. Struktur penyimpanan nilai pada Q-table ditunjukkan pada Tabel 2.8, sedangkan hubungan antara state, action, dan nilai Q dalam proses pengambilan keputusan ditunjukkan pada Tabel 2.12**.**

Tabel 2.8 Struktur Q-Table

row	col	direction	goal	forward	left	right
0	0	0	0	0	0	0
0	0	0	1	113,4384	235,1367	361,9332
0	0	0	2	20,85833	109,1024	270,3603
0	0	0	3	-18,0884	204,4391	208,7907

Berdasarkan Tabel 2.8, setiap baris merepresentasikan *state* tertentu, sedangkan setiap kolom merepresentasikan aksi yang dapat dilakukan. Nilai pada setiap sel menunjukkan kualitas (*Q-value*) dari suatu aksi pada *state* tertentu. Selama proses *training*, nilai-nilai pada *Q-table* akan diperbarui secara bertahap hingga diperoleh kebijakan yang optimal [1].



Gambar 2.12 Diagram Alir Q-Table

Berdasarkan Gambar 2.12, nilai Q yang tersimpan pada *Q-table* digunakan sebagai dasar pengambilan keputusan oleh agen. Pada setiap *state*, agen membandingkan nilai Q dari seluruh aksi yang tersedia, kemudian memilih aksi dengan nilai terbesar untuk memperoleh *reward* maksimum.

$$a = \arg\max Q(s, a) \tag{2.24}$$

Persamaan 2.24 menunjukkan bahwa agen akan memilih aksi dengan nilai Q tertinggi pada *state* tertentu, yang merepresentasikan aksi terbaik berdasarkan pengalaman pembelajaran. Pendekatan ini dikenal sebagai *greedy policy*, di mana agen memanfaatkan informasi yang telah dipelajari untuk menentukan tindakan yang optimal.

2.7 Grid Enviroment

Dalam pengembangan sistem navigasi robot berbasis *Reinforcement learning*, lingkungan tempat robot berinteraksi sering dimodelkan dalam bentuk grid environment. Grid environment merupakan representasi lingkungan dua

dimensi yang dibagi menjadi beberapa sel atau grid, di mana setiap sel merepresentasikan suatu posisi yang dapat ditempati oleh agen atau robot [1].

Pendekatan grid environment banyak digunakan dalam penelitian navigasi robot karena mempermudah proses pemodelan state, *action*, dan *reward* dalam lingkungan diskrit. Dengan menggunakan grid environment, posisi robot dapat direpresentasikan sebagai koordinat tertentu pada grid sehingga memudahkan proses pembelajaran algoritma *Reinforcement learning* dalam menentukan jalur navigasi yang optimal [31].

Dalam sistem navigasi berbasis *grid*, posisi robot dinyatakan dalam bentuk koordinat dua dimensi (x,y), di mana x menunjukkan posisi robot pada sumbu horizontal dan y menunjukkan posisi robot pada sumbu vertikal. Koordinat tersebut diperbarui berdasarkan hasil odometri yang diperoleh dari sensor Hall Effect. Jumlah pulsa yang terbaca digunakan untuk menghitung sudut putaran roda menggunakan Persamaan 2.18, kemudian dikonversi menjadi jarak tempuh melalui Persamaan 2.19. Setelah robot menempuh satu jarak yang setara dengan satu ukuran grid, koordinat (x,y) diperbarui sesuai arah perpindahan robot. Selain posisi, kondisi robot juga dipengaruhi oleh orientasi arah terhadap lingkungan serta tujuan navigasi yang aktif. Oleh karena itu, *state* pada penelitian ini terdiri atas posisi (x,y), orientasi robot, dan tujuan (*goal*) yang digunakan sebagai masukan algoritma *Q-Learning*.

Selain itu, agen memiliki aksi (*action*) yang digunakan untuk berpindah antar *state*. Aksi yang digunakan mengacu pada definisi sebelumnya, yaitu bergerak maju (*forward*), belok kiri (*turn left*), dan belok kanan (*turn right*).

Aksi-aksi tersebut akan menghasilkan perubahan *state* pada lingkungan grid, sehingga memungkinkan agen untuk mengeksplorasi berbagai kemungkinan jalur navigasi.

Lingkungan navigasi dalam penelitian ini dimodelkan dalam bentuk grid dua dimensi yang terdiri dari beberapa posisi penting seperti posisi awal robot, tujuan navigasi, serta obstacle. Contoh representasi lingkungan navigasi dapat ditunjukkan pada ilustrasi Gambar 2.13.

A	.	.	B	.	.	C
.	X	.	X	.	X	.
.	X	.	.	.	.	.
.	X	.	.	X	X	.
.	.	.	S	.	.	X

Gambar 2.13 Representasi Grid Environment Navigasi Robot

Keterangan:

S = posisi awal robot (*Home*)

A, B, C = tujuan navigasi

X = obstacle

. = area yang dapat dilalui robot

Pada lingkungan tersebut, robot akan dilatih untuk bergerak dari posisi Home (S) menuju tujuan tertentu seperti A, B, atau C, kemudian kembali ke posisi awal setelah tugas navigasi selesai.

Dalam algoritma *Reinforcement learning*, proses pembelajaran agen juga dipengaruhi oleh *reward* yang diberikan berdasarkan kondisi tertentu pada lingkungan [5]. *Reward* digunakan untuk mengevaluasi aksi yang dilakukan oleh agen sehingga sistem dapat mempelajari strategi navigasi yang optimal.

Contoh struktur *reward* yang digunakan pada sistem navigasi berbasis grid antara lain:

- *reward* positif ketika robot mencapai tujuan
- *reward* negatif ketika robot menabrak obstacle
- penalti kecil untuk setiap langkah pergerakan

Secara matematis, fungsi *reward* dapat dinyatakan sebagai berikut:

$$R = \begin{cases} +500 & \text{jika mencapai } \textit{goal} \\ -500 & \text{jika menabrak obstacle} \\ -1 & \text{untuk setiap langkah} \end{cases}$$

Dengan struktur *reward* tersebut, agen akan terdorong untuk menemukan jalur navigasi yang paling efisien menuju tujuan dengan meminimalkan jumlah langkah dan menghindari obstacle [32].

Meskipun lingkungan navigasi dimodelkan dalam bentuk grid pada tahap simulasi, implementasi pada robot fisik tidak secara langsung menggunakan grid yang digambar pada lantai. Grid hanya digunakan sebagai representasi logika navigasi dalam proses pembelajaran algoritma *Reinforcement learning* [4].

Dengan pendekatan ini, kebijakan navigasi (*navigation policy*) yang diperoleh melalui proses *training* pada lingkungan simulasi dapat diterapkan pada robot mobile nyata, sehingga robot mampu bergerak menuju tujuan yang diinginkan berdasarkan hasil pembelajaran yang telah dilakukan sebelumnya.

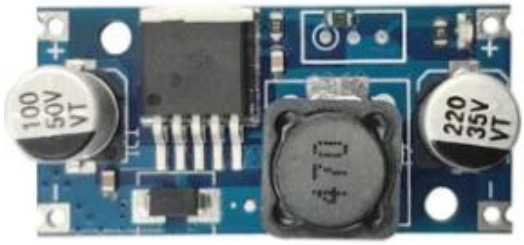
2.8 Buck Converter

Buck converter (konverter penurun tegangan DC-DC) merupakan jenis regulator daya beralih (*switching power supply*) yang berfungsi untuk menurunkan tegangan masukan (*input*) DC menjadi tegangan keluaran (*output*) DC yang lebih rendah dengan efisiensi daya yang tinggi. Modul yang digunakan pada penelitian ini adalah modul *step-down* berbasis LM2596 dengan keluaran tetap (*fixed output*) sebesar 5V. Spesifikasi teknis dari modul penurun tegangan *Buck Converter* LM2596 (Fixed 5V) ditunjukkan pada Tabel 2.9.

Tabel 2.9 Spesifikasi Buck Converter [33]

Parameter	Spesifikasi / Nilai
Tegangan Masukan (V_{in})	3.0 V – 40.0 V DC
Tegangan Keluaran (V_{out})	5.0 V DC (*Fixed*)
Arus Keluaran (I_{out})	Rated 2 A (Maksimum 3 A)
Frekuensi Beralih (*Switching*)	150 kHz
Efisiensi Konversi	Hingga 92%
Regulasi Beban (*Load Regulation*)	$\pm 0.5\%$
Regulasi Tegangan (*Voltage Regulation*)	$\pm 0.5\%$
Kecepatan Respon Dinamis	5% @ 200 μs
Suhu Operasi	-40°C hingga +85°C
Fitur Proteksi	Proteksi hubung singkat (*current limiting*, *auto recovery*)
Dimensi Fisik	$43 \times 21 \times 14$ mm
Parameter	Spesifikasi / Nilai

Berdasarkan spesifikasi pada Tabel 2.9, modul *buck converter* yang digunakan memiliki kemampuan untuk menurunkan tegangan input hingga 40 V menjadi 5 V dengan arus maksimum sebesar 3 A. Spesifikasi ini menunjukkan bahwa modul mampu menyuplai daya ke beberapa komponen sekaligus dalam sistem robot secara stabil.



Gambar 2.14 *Buck Converter* [34]

Berdasarkan Gambar 2.15, terlihat modul *buck converter* yang terdiri atas beberapa komponen utama, yaitu terminal masukan (IN+ dan IN−), terminal keluaran (OUT+ dan OUT−), induktor toroidal bertanda 470, kapasitor elektrolit, serta komponen regulator yang terpasang pada papan rangkaian. Terminal masukan (IN+ dan IN−) berfungsi sebagai jalur penerimaan tegangan DC dari sumber daya dengan rentang 3 V hingga 40 V, sedangkan terminal keluaran (OUT+ dan OUT−) digunakan untuk menyalurkan tegangan DC yang telah diregulasi sebesar 5 V menuju beban. Proses penurunan tegangan dilakukan oleh rangkaian *switching* yang didukung oleh induktor toroidal sebagai penyimpan energi selama proses konversi daya, sedangkan kapasitor elektrolit pada sisi masukan dan keluaran berfungsi mereduksi *ripple* tegangan sehingga menghasilkan keluaran yang lebih stabil. Susunan komponen tersebut memungkinkan modul *buck converter* menghasilkan catu daya yang efisien dan stabil untuk mendukung pengoperasian perangkat elektronik.

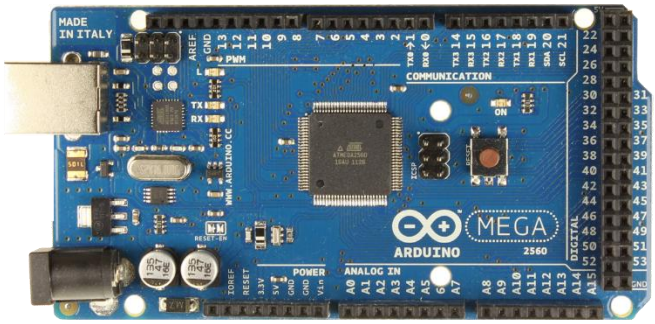
2.9 Mikrokontroler ATmega2560

ATmega2560 merupakan mikrokontroler berbasis arsitektur AVR 8-bit yang banyak digunakan pada sistem *embedded* karena memiliki jumlah pin input/output yang besar serta mendukung berbagai protokol komunikasi. Mikrokontroler ini digunakan pada papan Arduino Mega 2560 sebagai unit pemroses utama untuk pengendalian perangkat keras.

Tabel 2. 10 Spesifikasi Arduino Atmega2560 [35]

Parameter	Spesifikasi
Mikrokontroler	ATmega2560
Tegangan operasi	5 V
Digital I/O	54 pin

Analog input	16 pin
Clock speed	16 MHz
Flash memory	256 KB
SRAM	8 KB
EEPROM	4 KB
Komunikasi	UART, SPI, I2C



Gambar 2.15 Arduino Mega 2560 *[36]*

Berdasarkan Tabel 2.10, Arduino Mega 2560 memiliki jumlah pin *input/output*, kapasitas memori, serta dukungan antarmuka komunikasi seperti UART, SPI, dan I²C yang memungkinkan mikrokontroler mengendalikan berbagai sensor dan aktuator secara bersamaan sehingga sesuai digunakan pada sistem robot *mobile* [36].

Berdasarkan Gambar 2.16, Arduino Mega 2560 merupakan papan pengembangan berbasis mikrokontroler ATmega2560 yang menyediakan konektor untuk pin digital, pin analog, catu daya, serta port komunikasi. Susunan komponen tersebut memudahkan proses integrasi antara mikrokontroler dengan sensor, aktuator, dan modul komunikasi pada sistem robot *mobile* [37].

2.10 Raspberry Pi 3 Model B

Raspberry Pi merupakan *single board computer* (SBC) yang memiliki kemampuan komputasi lebih tinggi dibandingkan mikrokontroler. Perangkat ini mampu menjalankan sistem operasi berbasis Linux sehingga memungkinkan implementasi algoritma yang membutuhkan komputasi lebih kompleks, seperti *Reinforcement learning* pada sistem robotika [37].

Pada penelitian ini digunakan Raspberry Pi 3 Model B sebagai unit pemroses utama tingkat atas (*High-Level Controller*) yang bertanggung jawab dalam pengambilan keputusan navigasi robot.

Tabel 2.11 Spesifikasi Rasberry PI 3 Model B [35]

Parameter	Spesifikasi
Prosesor	ARM Cortex-A53 Quad Core
Kecepatan prosesor	1.2 GHz
RAM	1 GB
Konektivitas	WiFi, Bluetooth
Port	USB, HDMI, GPIO
Sistem operasi	Linux (Raspberry Pi OS)

Gambar 2.16 Rasberry PI 3 Model B [37]

Dalam penelitian ini, Raspberry Pi berfungsi sebagai *High-Level Controller* yang menjalankan algoritma *Reinforcement learning* (Q-*Learning*) untuk navigasi robot mobile. Raspberry Pi menerima data kondisi robot dari Arduino, kemudian memproses data tersebut untuk menentukan aksi navigasi seperti bergerak maju, belok kiri, atau belok kanan berdasarkan nilai pada Q-table [35].

Selanjutnya, hasil keputusan tersebut dikirimkan kembali ke Arduino melalui komunikasi serial untuk dieksekusi oleh sistem aktuator. Pendekatan ini memungkinkan pemisahan antara proses pengambilan keputusan dan kontrol perangkat keras.

Arsitektur sistem ini membagi fungsi kontrol menjadi dua bagian, yaitu:

- *Low-level control*, Arduino (sensor & aktuator).
- *High-level control*, Raspberry Pi (algoritma navigasi).

Pembagian ini membuat sistem menjadi lebih modular, stabil, serta memudahkan proses pengembangan dan debugging. Selain itu, penggunaan komputer mini seperti Raspberry Pi memungkinkan implementasi algoritma berbasis kecerdasan buatan yang memerlukan sumber daya komputasi lebih tinggi dibandingkan mikrokontroler [37].

2.11 Motor DC Permanent Magnet Worm Gear

Motor DC *Permanent Magnet Worm Gear* merupakan jenis motor DC yang dilengkapi dengan gearbox tipe *worm gear* untuk menghasilkan torsi besar pada kecepatan rendah. Penggunaan gearbox ini memberikan rasio reduksi yang tinggi serta kemampuan *self-locking*, sehingga motor mampu mempertahankan posisi ketika berhenti tanpa memerlukan daya tambahan.

Tabel 2.12 Spesifikasi Motor DC 5840-31ZY [38]

Parameter	Nilai
Kecepatan nominal	12 RPM
Rasio gear	1 : 670
Kecepatan tanpa beban	10 RPM
Arus tanpa beban	≤ 0,3 A
Arus beban standar	≤ 1,088 A
Arus beban maksimum	≤ 6,5 A
Torsi kontinu	30,57 kg·cm
Torsi stall	65,74 kg·cm
Berat	361 gram

Berdasarkan Tabel 2.11, motor DC 5840-31ZY memiliki kecepatan nominal 12 RPM, rasio gear 1:670, torsi kontinu sebesar 30,57 kg·cm, dan torsi *stall* mencapai 65,74 kg·cm, sehingga sesuai digunakan sebagai penggerak robot mobile.

Gambar 2.17 Motor DC Permanent Magnet Worm Gear 5840-31Zy [38]

Berdasarkan Gambar 2.18, motor ini terdiri atas motor DC yang terintegrasi dengan gearbox tipe *worm gear*, yang memungkinkan menghasilkan torsi besar dengan putaran rendah sehingga pergerakan robot menjadi lebih stabil. Dalam penelitian ini, motor DC *worm gear* digunakan sebagai aktuator utama untuk menggerakkan robot mobile dan dikendalikan melalui driver motor BTS7960 untuk mengatur kecepatan serta arah putaran.

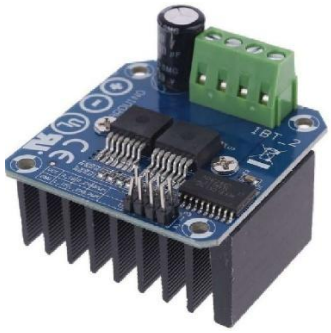
2.12 Driver Motor BTS7960

Driver motor BTS7960 merupakan modul driver berbasis H-Bridge yang digunakan untuk mengendalikan motor DC dengan arus besar. Modul ini mampu mengontrol arah putaran serta kecepatan motor melalui sinyal PWM dari mikrokontroler. Driver ini sangat cocok digunakan pada sistem robot karena memiliki kemampuan arus tinggi dan dilengkapi proteksi internal sehingga lebih aman digunakan dalam aplikasi robotika.

Tabel 2.13 Spesifikasi Driver Motor BTS7960 [39]

Parameter	Spesifikasi
Tipe Driver	H-Bridge
Tegangan Operasi	6 V – 27 V
Arus Maksimum	Hingga 43 A
Kontrol	PWM
Interface	Logic Level (Arduino)
Proteksi	Overcurrent, Overheat
Channel	Dual Half Bridge
Aplikasi	Motor DC, robot, otomasi

Berdasarkan Tabel 2.12, driver motor BTS7960 merupakan driver bertipe *H-Bridge* yang bekerja pada tegangan 6–27 V dengan arus maksimum hingga 43 A. Driver ini dikendalikan menggunakan sinyal PWM dan dilengkapi proteksi *overcurrent* serta *overheat*, sehingga sesuai digunakan untuk pengendalian motor DC pada sistem robot.



Gambar 2.18 Driver Motor BTS7960 [39]

Berdasarkan Gambar 2.19, modul BTS7960 terdiri atas terminal catu daya, terminal keluaran ke motor, serta pin kontrol yang terhubung ke mikrokontroler. Susunan komponen tersebut memungkinkan modul mengatur kecepatan dan arah putaran motor DC melalui sinyal PWM dari Arduino. Dalam penelitian ini, driver BTS7960 berfungsi sebagai penghubung antara Arduino dan motor DC. Driver menerima sinyal PWM dari Arduino untuk mengatur kecepatan serta arah putaran motor.

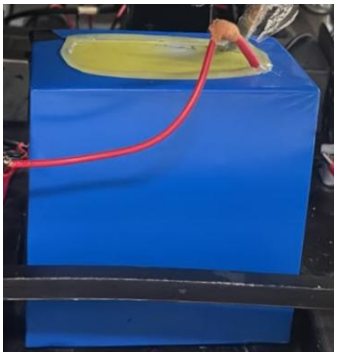
2.13 Baterai *Lithium Iron Phosphate* ($LiFePO_4$)

Baterai *Lithium Iron Phosphate* ($LiFePO_4$) merupakan salah satu jenis baterai lithium-ion yang banyak digunakan sebagai sumber energi pada sistem robot *mobile* karena memiliki umur pakai yang panjang, tingkat keamanan yang tinggi, serta mampu menghasilkan tegangan keluaran yang stabil selama proses *discharge*. Dibandingkan dengan aki timbal-asam, baterai $LiFePO_4$ memiliki bobot yang lebih ringan, efisiensi pengisian dan pengosongan yang lebih tinggi, serta tidak memerlukan perawatan khusus sehingga sesuai untuk aplikasi robot bergerak yang membutuhkan sumber daya yang andal [33].

Tabel 2.14 Spesifikasi Baterai *Lithium Iron Phosphate* [40]

Parameter	Nilai
Jenis baterai	Lithium Iron Phosphate ($LiFePO_4$)
Konfigurasi sel	4S
Tegangan nominal	12,8 V
Tegangan penuh	±14,6 V
Tegangan cut-off	10–11 V
Kapasitas	12 Ah
Sistem proteksi	Battery Management System (BMS)

Fitur BMS	Overcharge, overdischarge, overcurrent, short circuit, cell balancing



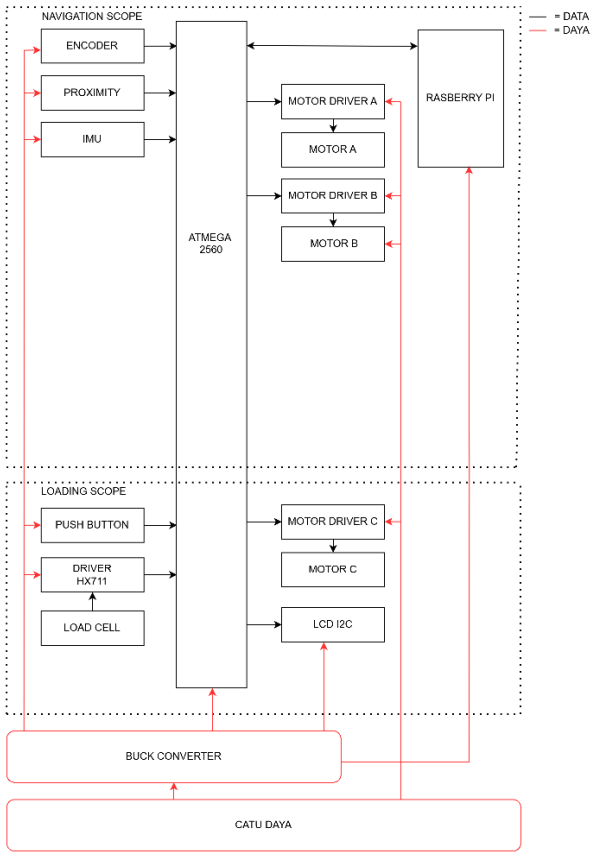
Gambar 2.19 Baterai *Lithium Iron Phosphate (LiFePO₄)*

Berdasarkan Tabel 2.14, baterai $LiFePO_4$ yang digunakan memiliki tegangan nominal 12,8 V dengan kapasitas 12 Ah sehingga mampu memenuhi kebutuhan daya sistem robot. Baterai ini juga telah dilengkapi *Battery Management System* (BMS) yang berfungsi melindungi baterai dari kondisi *overcharge*, *overdischarge*, *overcurrent*, *short circuit*, serta menjaga keseimbangan tegangan setiap sel (*cell balancing*) sehingga keamanan dan umur pakai baterai dapat ditingkatkan [33].

BAB III
METODE PENELITIAN

3.1 Blok Diagram Sistem

Blok diagram sistem digunakan untuk menggambarkan hubungan antar komponen utama pada robot *Automated Guided Vehicle* (AGV) yang dikembangkan. Diagram ini menunjukkan aliran data dari sensor menuju sistem kontrol serta aliran daya menuju aktuator yang menggerakkan robot. Pada sistem ini arsitektur dibagi menjadi dua bagian utama, yaitu *navigation scope* yang menangani proses navigasi robot dan *loading scope* yang menangani mekanisme pengangkatan beban.



Gambar 3.1 Diagram Blok Sistem

Penjelasan blok diagram sistem adalah sebagai berikut:

a. Catu daya sistem

Sistem memperoleh sumber daya dari baterai utama yang kemudian

didistribusikan ke seluruh komponen elektronik. Tegangan yang diperlukan oleh masing-masing perangkat disesuaikan menggunakan buck converter.

b. Sensor navigasi

Beberapa sensor digunakan untuk memperoleh informasi kondisi robot dan lingkungan, yaitu:

- Encoder untuk mengukur rotasi roda dan menghitung jarak tempuh robot.
- Proximity sensor untuk mendeteksi keberadaan obstacle di depan robot.
- IMU untuk menentukan orientasi atau arah hadap robot.

c. Mikrokontroler ATmega2560

Mikrokontroler berfungsi sebagai low-level controller yang membaca seluruh data sensor dan mengendalikan perangkat keras robot. Data sensor kemudian dikirimkan ke Raspberry Pi melalui komunikasi serial.

d. Raspberry Pi 3 (*High-Level Controller*)

Raspberry Pi menerima data kondisi robot dari mikrokontroler dan menjalankan algoritma *Reinforcement learning* dengan metode Q-*Learning* untuk menentukan keputusan navigasi robot.

e. Pengiriman perintah gerakan

Setelah aksi navigasi ditentukan, Raspberry Pi mengirimkan perintah gerakan ke ATmega2560 untuk dieksekusi.

f. Motor driver dan motor DC

Mikrokontroler mengendalikan motor driver untuk mengatur kecepatan dan arah putaran motor DC. Sistem penggerak menggunakan konfigurasi differential drive dengan dua motor penggerak pada sisi kiri dan kanan robot.

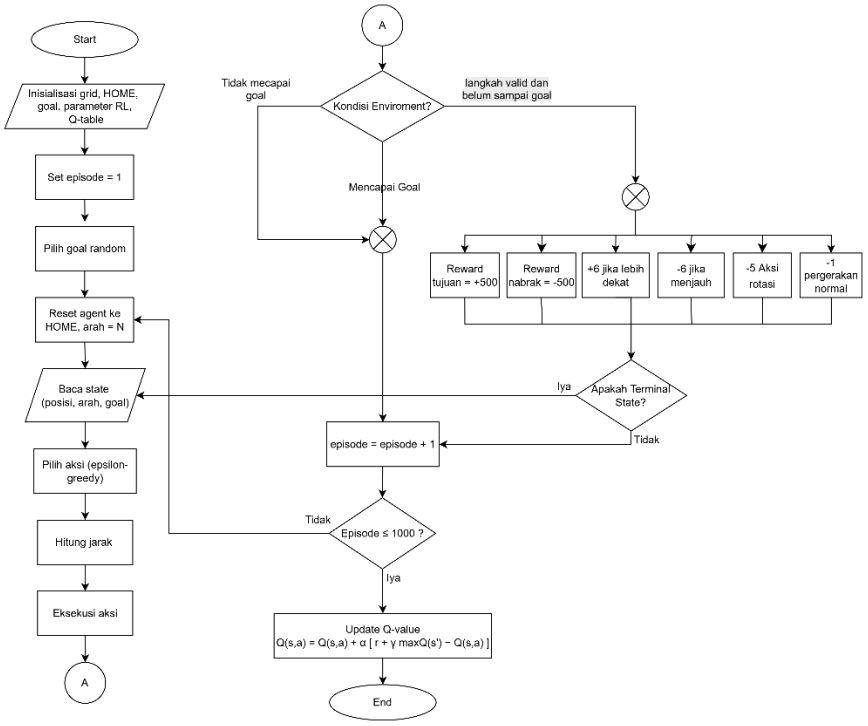
g. Penentuan tujuan navigasi

Terdapat 3 tujuan dalam menentuka navigasi berikut, yaitu *goal* A, *goal* B, dan *Goal* C

Dengan arsitektur ini, sistem kontrol robot dibagi menjadi dua tingkat yaitu kontrol perangkat keras oleh ATmega2560 dan pengambilan keputusan navigasi oleh Raspberry Pi, sehingga sistem menjadi lebih modular dan mudah dikembangkan.

3.2 Diagram Alir Sistem

Diagram alir sistem digunakan untuk menggambarkan urutan proses kerja robot mobile dalam melakukan navigasi dari posisi awal menuju target dan kembali ke posisi awal. Diagram ini menunjukkan alur proses mulai dari inisialisasi perangkat, proses pengambilan keputusan navigasi menggunakan algoritma *Reinforcement learning*, hingga robot menyelesaikan tugas pengantaran barang.



Gambar 3.2 Diagram Alir Sistem

Penjelasan diagram alir sistem adalah sebagai berikut:

a. Inisialisasi sistem

   Sistem mengaktifkan seluruh perangkat yang digunakan, yaitu ATmega2560, Raspberry Pi, serta sensor seperti IMU, proximity sensor, dan encoder.

b. Reset encoder

   Encoder direset untuk memastikan pembacaan jumlah pulsa dimulai dari kondisi awal sehingga perhitungan jarak tempuh roda menjadi akurat.

c. Memuat Q-table

   Raspberry Pi memuat Q-table hasil proses *training* algoritma *Reinforcement learning* yang akan digunakan sebagai dasar pengambilan keputusan navigasi.

d. Pemeriksaan kondisi lifting

   Jika state yang terbaca menunjukkan posisi Home, sistem akan memeriksa

status mekanisme lifting untuk mengetahui apakah robot membawa beban atau tidak.

e. Penentuan tujuan navigasi

   Jika robot membawa beban, sistem menentukan target pengantaran yaitu A, B, atau C sesuai kondisi yang telah ditentukan.

f. Proses navigasi robot

   Robot mulai bergerak menuju target dengan memanfaatkan algoritma Q-*Learning* sebagai pengambil keputusan navigasi.

g. Pembacaan sensor selama navigasi

   Selama robot bergerak, sistem secara terus menerus membaca data dari encoder, IMU, dan proximity sensor untuk mengetahui kondisi robot dan lingkungan.

h. Deteksi obstacle

   Jika proximity sensor mendeteksi adanya hambatan di depan robot, sistem akan menghentikan motor sementara hingga jalur kembali aman.

i. Penentuan aksi navigasi

   Jika jalur bebas dari obstacle, Raspberry Pi menentukan aksi navigasi berdasarkan state robot saat ini menggunakan Q-table. Aksi yang dihasilkan dapat berupa maju, belok kiri, atau belok kanan.

j. Eksekusi pergerakan robot

   Perintah gerakan dikirimkan ke ATmega2560 untuk mengendalikan motor melalui driver motor.

k. Deteksi kedatangan di target

   Robot akan terus bergerak hingga state mendeteksi bahwa robot telah mencapai lokasi tujuan.

3.3 Gambar Konsep 3D Alat

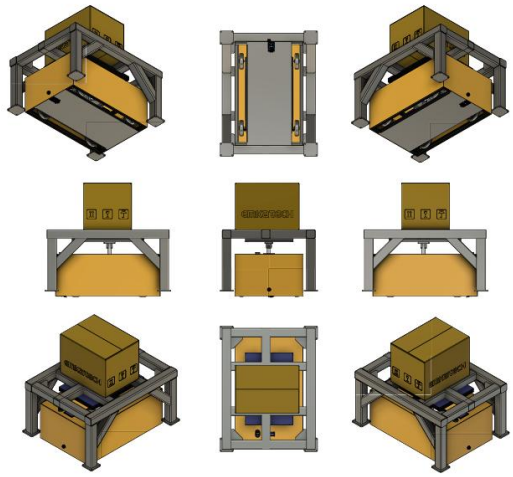
Desain mekanik robot mobile dirancang untuk menghasilkan struktur yang kuat, stabil, dan mampu menopang beban selama proses navigasi dan pengangkutan barang. Rangka robot dibuat menggunakan profil logam yang membentuk kerangka persegi panjang sebagai struktur utama. Desain ini bertujuan untuk memberikan kekuatan mekanik yang cukup sekaligus menjaga distribusi beban agar tetap seimbang saat robot bergerak.

Pada bagian bawah rangka dipasang sistem penggerak berupa dua roda utama yang terhubung dengan motor DC sebagai penggerak kiri dan kanan, sehingga membentuk konfigurasi differential drive. Konfigurasi ini memungkinkan robot untuk bergerak maju, berbelok, maupun berputar pada tempat dengan mengatur perbedaan kecepatan pada kedua roda.

Selain roda penggerak utama, robot juga dilengkapi dengan dua roda tambahan yang berfungsi sebagai penopang untuk menjaga kestabilan robot selama bergerak. Roda tambahan ini tidak digerakkan oleh motor dan berfungsi untuk mengurangi kemungkinan robot mengalami ketidakseimbangan saat membawa beban.

Pada bagian atas rangka disediakan ruang untuk penempatan komponen utama seperti mikrokontroler, driver motor, serta modul sensor. Sensor encoder dipasang pada motor untuk membaca perpindahan posisi roda, sedangkan sensor IMU digunakan untuk mengetahui orientasi atau arah pergerakan robot. Selain itu, sensor proximity dipasang pada bagian depan robot untuk mendeteksi adanya obstacle statis di jalur pergerakan.

Desain keseluruhan robot dibuat dengan mempertimbangkan kemudahan dalam perakitan, perawatan, serta integrasi antara sistem mekanik dan sistem kontrol. Representasi konsep desain mekanik robot dalam bentuk 3D ditunjukkan pada Gambar 3.3.

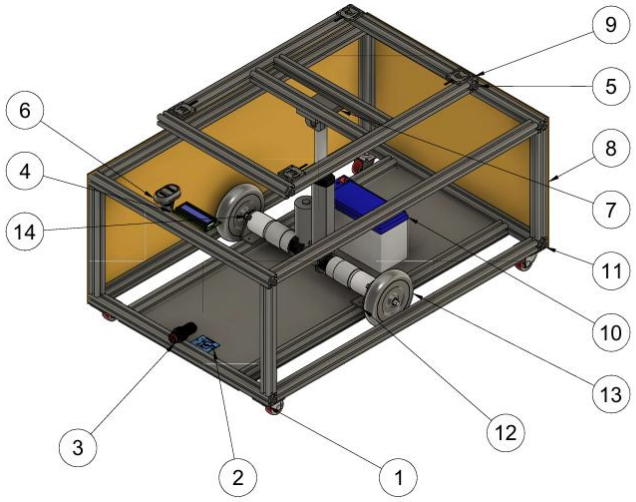


Gambar 3.3 Konsep 3D Alat

Robot menggunakan konfigurasi differential drive, yaitu sistem penggerak dengan dua roda penggerak utama yang terletak pada sisi kiri dan kanan robot.

Pergerakan robot dikendalikan dengan mengatur kecepatan putaran masing-masing roda sehingga robot dapat bergerak maju, berbelok, atau berputar di tempat. Konfigurasi ini banyak digunakan pada robot mobile karena memiliki konstruksi mekanik yang sederhana serta mudah dikendalikan.

Pada bagian rangka, sistem mekanik robot dibagi menjadi dua bagian utama yaitu rangka statis dan rangka dinamis. Rangka statis berfungsi sebagai struktur utama yang menopang seluruh komponen mekanik robot, sedangkan rangka dinamis merupakan bagian yang dapat bergerak pada mekanisme pengangkatan beban.



Parts List			
Item	Qty	Part Name	Material
1	1	RANGKA STATIS	Steel
2	1	RFID SENSOR v1	Steel
3	1	PROXIMITY SENSOR	Steel
4	1	LCD I2C	Steel
5	1	RANGKA DINAMIS	Steel
6	1	PUSH BUTTON	Steel
7	1	LINEAR	----
8	1	COVER	Steel
9	1	LOAD CELL	Steel
10	1	BATTERY	Steel
11	1	RODA CASTER	Steel
12	1	MOTOR	----
13	1	RODA	Steel
14	1	HALL EFFECT	Steel

Gambar 3.4 3D Komponen Mekanik

Penjelasan komponen mekanik pada robot adalah sebagai berikut:

a. Rangka Statis

Rangka statis merupakan struktur utama robot yang berfungsi sebagai penopang seluruh sistem mekanik. Rangka ini dirancang berbentuk rangka persegi yang memberikan kekakuan struktur serta menjaga kestabilan robot saat bergerak dan membawa beban.

b. Rangka Dinamis

Rangka dinamis merupakan bagian yang terhubung dengan mekanisme lifting dan dapat bergerak secara vertikal. Bagian ini berfungsi sebagai platform yang digunakan untuk mengangkat atau menurunkan beban.

c. Roda Penggerak

Robot menggunakan enam roda, di mana dua roda berfungsi sebagai roda

penggerak utama yang terhubung dengan motor, sedangkan roda lainnya berfungsi sebagai penopang untuk menjaga keseimbangan robot saat bergerak.

d. Mekanisme Linear (Linear Motion)

Mekanisme ini digunakan untuk menghasilkan gerakan naik dan turun pada sistem lifting. Gerakan linear memungkinkan platform pengangkut beban dapat bergerak secara vertikal dengan stabil.

e. Cover atau Pelindung Rangka

Cover berfungsi sebagai pelindung komponen yang berada di dalam rangka robot serta membantu menjaga kekakuan struktur mekanik.

Secara keseluruhan desain mekanik robot dibuat dengan mempertimbangkan beberapa aspek utama yaitu kekuatan struktur, stabilitas pergerakan, serta distribusi beban. Dengan konfigurasi differential drive dan rangka yang kokoh, robot diharapkan mampu bergerak secara stabil saat melakukan navigasi maupun ketika membawa beban.

3.4 Spesifikasi dan Fitur Alat

Spesifikasi dan fitur alat menjelaskan komponen utama yang digunakan dalam sistem robot Automated Guided Vehicle (AGV) yang dikembangkan pada penelitian ini. Spesifikasi pada Tabel 3.1 ini meliputi perangkat kontrol, sensor navigasi, sistem penggerak, serta sistem catu daya yang digunakan untuk mendukung kinerja robot.

Tabel 3.1 Spesifikasi Alat

No	Komponen	Spesifikasi	Tegangan Kerja	Power Consumption
1	Mikrokontroler	Arduino Mega 2560 (ATmega2560)	5 V	±0.25 W
2	Komputer Mini	Raspberry Pi 3 Model B	5 V	±6 W
3	Motor Penggerak	DC Gear Motor	12 V	±30 W / motor
4	Driver Motor	IBT-2 Motor Driver	5–12 V	±1 W
5	Sensor Obstacle	Proximity Sensor	5 V	±0.1 W

6	Sensor Orientasi	IMU (MPU6050)	3.3–5 V	±0.02 W
7	Sensor Jarak Tempuh	Rotary Encoder	5 V	±0.05 W

Sistem robot auto mobile yang dikembangkan pada penelitian ini memiliki beberapa fitur utama yaitu:

a. Navigasi Otonom Berbasis *Reinforcement learning*

   Robot mampu menentukan jalur navigasi secara mandiri menggunakan algoritma Q-*Learning* tanpa bergantung pada jalur fisik seperti garis atau marker.

b. Deteksi Obstacle

   Sensor proximity digunakan untuk mendeteksi keberadaan obstacle pada jalur navigasi robot sehingga dapat mencegah terjadinya tabrakan.

c. Pengukuran Orientasi Robot

   Sensor IMU digunakan untuk mengetahui orientasi robot selama proses navigasi sehingga pergerakan robot dapat dikontrol dengan lebih akurat.

d. Pengukuran Jarak Tempuh

   Rotary encoder digunakan untuk menghitung jarak tempuh roda robot sehingga sistem dapat memperkirakan posisi robot dalam lingkungan navigasi.

e. Sistem Pengangkatan Beban

   Robot dilengkapi dengan mekanisme lifting yang dapat mengangkat dan menurunkan beban secara otomatis.

3.5 Teknik Fabrikasi

Pada perancangan sistem auto mobile digunakan empat teksnik fabrikasi yaitu perancangan sistem elektrikal, perancangan wiring, perancangan perancangan pemogramman, dan perancangan mekanikal.

3.5.1 Perancangan Elektrikal

Untuk merancang sistem kelistrikan pada robot mobile, diperlukan analisis kebutuhan tegangan dan arus dari setiap komponen yang digunakan. Setiap komponen memiliki karakteristik konsumsi daya yang berbeda, sehingga perlu dilakukan identifikasi untuk memastikan sumber daya yang digunakan mampu menyuplai seluruh sistem secara stabil.

Pada penelitian ini, sistem elektrikal mencakup komponen utama seperti sensor, mikrokontroler, unit pemrosesan (*Raspberry Pi*), driver motor, serta motor penggerak. Perhitungan dilakukan dengan menggunakan nilai arus maksimum dari masing-masing komponen untuk mengantisipasi kondisi beban tertinggi selama operasi.

Hasil perhitungan kebutuhan tegangan dan arus tersebut digunakan sebagai dasar dalam menentukan spesifikasi sumber daya, seperti pemilihan baterai dan rangkaian regulator tegangan yang sesuai. Rincian kebutuhan tegangan dan arus dari setiap komponen ditunjukkan pada Tabel 3.2.

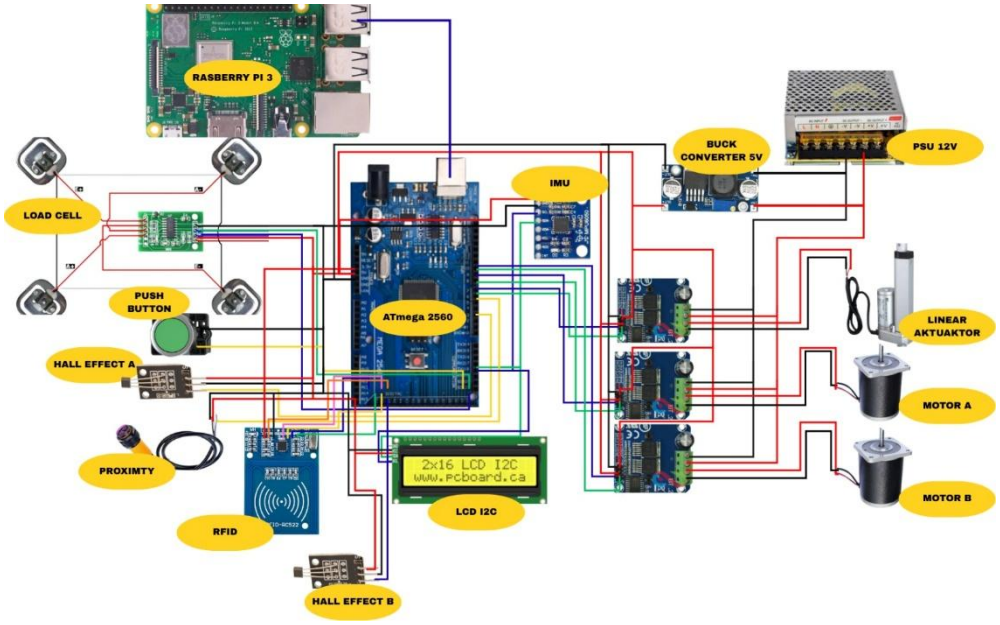
Tabel 3.2 Kebutuhan Tegangan dan Arus Alat

No	Komponen	Tegangan (V)	Arus Maks (A)
1	Hall Effect Sensor (2 buah)	5	0,04
2	Sensor Proximity	5	0,10
3	IMU MPU6050	3,3–5	0,02
4	Raspberry Pi 3	5	1,50
5	ATmega2560	5	0,10
6	Driver Motor (logic)	5	~0
7	Motor DC (2 buah)	12	13,00
Total			14,86 A

3.5.2 Perancangan Wiring

Perancangan *wiring* pada sistem robot mobile mencakup distribusi daya dan jalur komunikasi data antar komponen. Sumber daya utama berasal dari aki 12V yang digunakan untuk menyuplai motor DC melalui driver motor. Tegangan ini kemudian diturunkan menjadi 5V menggunakan *buck converter* untuk menyuplai ATmega2560, *Raspberry Pi*, serta sensor.

Selain distribusi daya, sistem *wiring* juga mengatur jalur komunikasi data. Sensor Hall Effect dan proximity terhubung ke pin digital ATmega2560, sedangkan modul HX711 menggunakan komunikasi digital melalui pin khusus. IMU MPU6050 dan LCD menggunakan komunikasi I2C (SDA dan SCL). Driver motor dikendalikan melalui sinyal PWM dari ATmega2560 untuk mengatur kecepatan dan arah putaran motor.



Gambar 3.5 Diagram Wiring Elektrikal

Gambar 3.5 menunjukkan diagram *wiring* elektrikal yang memperlihatkan jalur distribusi daya dari aki ke setiap komponen serta koneksi antar modul dalam sistem.

Tabel 3. 3 Konfigurasi Pin

No	Komponen	Pin Komponen	Pin ATmega2560
1	Hall Effect Sensor (Kiri)	VCC, GND, OUT	5V, GND, Pin 2
2	Hall Effect Sensor (Kanan)	VCC, GND, OUT	5V, GND, Pin 3
3	Push Button	Kaki 1, Kaki 2	Pin 26, GND
4	Sensor Proximity	VCC, GND, OUT	5V, GND, Pin 5
5	Modul HX711	VCC, GND, DT, SCK	5V, GND, Pin 24, Pin 25
6	IMU MPU6050	VCC, GND, SDA, SCL	5V, GND, Pin 20, Pin 21
7	LCD I2C	VCC, GND, SDA, SCL	5V, GND, SDA, SCL
8	Driver Motor Kiri	RPWM, LPWM	Pin 6, Pin 7
9	Driver Motor Kanan	RPWM, LPWM	Pin 10, Pin 11

Berdasarkan Tabel 3.3, sistem *wiring* membagi jalur daya dan data secara terpisah. Sumber 12V dari aki digunakan untuk motor, sedangkan tegangan 5V dari *buck converter* menyuplai mikrokontroler dan sensor.

Komunikasi data menggunakan pin digital, I2C, SPI, dan PWM sesuai kebutuhan masing-masing komponen, sehingga sistem dapat bekerja secara stabil.

3.5.3 Perancangan Pemrogramman

Perancangan pemrograman pada sistem robot mobile bertujuan untuk mengimplementasikan algoritma *Reinforcement learning* dalam proses navigasi robot pada lingkungan grid. Sistem pemrograman dibagi menjadi dua bagian utama yaitu program pada Raspberry Pi sebagai pengolah algoritma *Reinforcement learning* dan program pada mikrokontroler ATmega sebagai pengendali aktuator motor.

Raspberry Pi berfungsi untuk membaca kondisi robot, menentukan state, memilih aksi berdasarkan Q-table, dan mengirimkan perintah pergerakan robot ke mikrokontroler. Mikrokontroler ATmega kemudian mengubah perintah tersebut menjadi sinyal kontrol motor melalui driver motor sehingga robot dapat bergerak sesuai dengan aksi yang dipilih oleh algoritma *Reinforcement learning*.

a) Program *Reinforcement learning* pada Raspberry Pi

Program pada Raspberry Pi bertugas menjalankan algoritma *Reinforcement learning* dengan menggunakan Q-table hasil proses *training*. Q-table menyimpan nilai aksi terbaik untuk setiap kombinasi state robot.

Contoh struktur program *Reinforcement learning* pada Raspberry Pi ditunjukkan sebagai berikut:

```
class QAgent:
  def __init__(self, env: AGVEnv):
    self.env = env
    shape = (env.grid_h, env.grid_w, 4, len(env.goals), env.n_actions)
    self.Q = np.zeros(shape, dtype=np.float64)
  @staticmethod
  def _idx(state):
    gx, gy, d, g = state
    return (gy, gx, d, g)          # Q[y, x, dir, goal]
  def policy_at(self, state):
    """Return (best_action, q_values) — greedy, tanpa eksplorasi."""
    q = self.Q[self._idx(state)]
```

```
        return int(np.argmax(q)), q.copy()
    def load(self, path):
        if not os.path.exists(path):
            return False
        arr = np.load(path)
        if arr.shape != self.Q.shape:
            print(f"[QAgent] shape mismatch: {arr.shape} vs {self.Q.shape}")
            return False
        self.Q = arr.astype(np.float64)
        return True

# 3. CONTROLLER : baca state → lookup Q → kirim aksi

class Controller:
    def __init__(self, send):
        self.send = send
        self.env = AGVEnv()
        self.agent = QAgent(self.env)
        self.q_loaded = False
        self._goal = None
        self._waiting_action_done = False
        self._last_action_t = 0.0
    def load_q(self, path):
        self.q_loaded = self.agent.load(path)
        print(f"Q-table {'loaded' if self.q_loaded else 'FAIL'} <- {path}")
    def set_goal_by_idx(self, idx):
        gx, gy = self.env.goal_xy(idx)
        self._goal = (idx, gx, gy, GOAL_LABELS[idx])
    def on_state(self, cur):
        if self._goal is None or not self.q_loaded:
            return
        if self._waiting_action_done:
```

```
            return
        if time.time() - self._last_action_t < 0.1:     # rate-limit 100 ms
            return
        idx, gx, gy, label = self._goal
        if (cur.grid_x, cur.grid_y) == (gx, gy):
            self.send("ACT,3")          # STOP
            print(f"GOAL {label} tercapai di ({gx},{gy})")
            return
        s = (cur.grid_x, cur.grid_y, cur.direction, idx)
        a, q_vals = self.agent.policy_at(s)
        self._waiting_action_done = True
        self._last_action_t = time.time()
        # Kirim perintah ke Mega2560 via serial
        self.send(f"ACT,{a}")
        print(f"SEND ACT,{a} -> {['FWD','LEFT','RIGHT'][a]}"
f"state=({cur.grid_x},{cur.grid_y},{'NESW'[cur.direction]},{label}) "
              f"Q=[{q_vals[0]:.1f}, {q_vals[1]:.1f}, {q_vals[2]:.1f}]")
    def on_action_done(self):
        self._waiting_action_done = False
```

Pada program tersebut Raspberry Pi membaca kondisi robot, membentuk state, kemudian memilih aksi dengan nilai Q terbesar dari Q-table. Aksi yang dipilih kemudian dikirimkan ke mikrokontroler melalui komunikasi serial.

b) Program Kontrol Motor pada ATmega

Mikrokontroler ATmega berfungsi menerima perintah dari Raspberry Pi dan mengubahnya menjadi sinyal kontrol motor. Perintah yang diterima berupa kode karakter yang merepresentasikan aksi pergerakan robot. Kode perintah yang digunakan adalah

F : *Forward*

L : *Left*

R : *Right*

S : Stop

Contoh program pada ATmega ditunjukkan sebagai berikut:

```
void start(Cmd c) {
  g_cmd = c; g_run = true;
  switch (c) {
    case ACT_FWD:   g_target_ticks = 13; break;
    case ACT_LEFT:      g_targetYaw = wrap180(round(yaw/90)*90 +
g_turnAngleDeg); break;
    case ACT_RIGHT:  g_targetYaw = wrap180(round(yaw/90)*90 -
g_turnAngleDeg); break;
    case ACT_STOP:  motor::stop(); g_run = false; break; } }
bool step(float yaw_now, float dt) {
  switch (g_cmd) {
    case ACT_FWD: {
      long avgTick = (encoder::ticksLeft()+encoder::ticksRight())/2;
      if (avgTick >= g_target_ticks) { motor::stop(); odom::snapToGrid(); return
true; }
      float corr = g_pid.computeWrapped(g_targetYaw, yaw_now, dt);
      motor::setBoth(g_basePwm-(int)corr, g_basePwm+(int)corr);
      break;  }
    case ACT_LEFT:
    case ACT_RIGHT: {
      if (fabs(wrap180(g_targetYaw-yaw_now)) < 1.2f) { motor::stop(); return
true; }
      int dir = (wrap180(g_targetYaw-yaw_now) > 0) ? 1 : -1;
      motor::setBoth(-g_basePwm*dir, g_basePwm*dir);
      break;  }
    case ACT_STOP: motor::stop(); return true;  }
  return false; }
```

Program tersebut membaca data dari komunikasi serial dan menjalankan fungsi pergerakan robot sesuai perintah yang diterima.

c) Pengaturan Kecepatan Motor

Pengaturan kecepatan motor dilakukan menggunakan sinyal *Pulse Width Modulation* (PWM) yang dihasilkan oleh mikrokontroler ATmega. Sinyal PWM

digunakan untuk mengatur kecepatan motor sedangkan sinyal direction digunakan untuk menentukan arah putaran motor. Contoh implementasi fungsi pergerakan robot ditunjukkan sebagai berikut

```
        void setLeft(int pwm) {
           pwm = constrain(pwm, -255, 255);
           g_left = pwm;
           pwm = pwm * MOTOR_LEFT_INVERT;
           if (pwm > 0) {
              analogWrite(RPWM_L, pwm);
              analogWrite(LPWM_L, 0);
           } else if (pwm < 0) {
              analogWrite(RPWM_L, 0);
              analogWrite(LPWM_L, -pwm);
           } else {
              analogWrite(RPWM_L, 0);
              analogWrite(LPWM_L, 0);   } }
        void setRight(int pwm) {
           pwm = constrain(pwm, -255, 255);
           g_right = pwm;
           pwm = pwm * MOTOR_RIGHT_INVERT;
           if (pwm > 0) { analogWrite(RPWM_R, pwm); analogWrite(LPWM_R,
0); }
           else if (pwm < 0) { analogWrite(RPWM_R, 0); analogWrite(LPWM_R,
-pwm); }
           else { analogWrite(RPWM_R, 0); analogWrite(LPWM_R, 0); } }
        void setBoth(int l, int r) { setLeft(r); setRight(l); }
        void stop() { setBoth(0, 0); }
```

Pada fungsi tersebut kedua motor berputar maju dengan kecepatan yang sama sehingga robot bergerak lurus ke depan.

d) Pembacaan Encoder sebagai Feedback Sistem

Encoder Hall Effect digunakan untuk membaca putaran roda robot dan menghasilkan pulsa setiap kali roda berputar. Pulsa tersebut digunakan untuk

menghitung jarak tempuh robot pada arena navigasi. Contoh program pembacaan encoder adalah sebagai berikut

```
static volatile long g_ticksL = 0;
static volatile long g_ticksR = 0;
static volatile unsigned long g_lastIsrL_us = 0;
static volatile unsigned long g_lastIsrR_us = 0;
static void isrL() {
  unsigned long now = micros();
  if (now - g_lastIsrL_us < 40000) return;
  g_lastIsrL_us = now;
  g_ticksL++; }
static void isrR() {
  unsigned long now = micros();
  if (now - g_lastIsrR_us < 40000) return;
  g_lastIsrR_us = now;
  g_ticksR++; }
```

Jumlah pulsa encoder yang terbaca digunakan untuk menentukan kapan robot telah berpindah satu grid. Data tersebut kemudian digunakan untuk memperbarui posisi robot pada sistem navigasi.

3.5.4 Perancangan Mekanikal

Perancangan mekanikal dilakukan untuk menentukan struktur fisik robot mobile yang meliputi rangka utama, sistem penggerak, serta penempatan komponen elektronik dan sensor. Desain mekanik harus mampu menopang seluruh komponen robot sekaligus mempertahankan stabilitas robot selama proses navigasi dan pengangkutan beban.

3.5.4.1 Perancangan Rangka

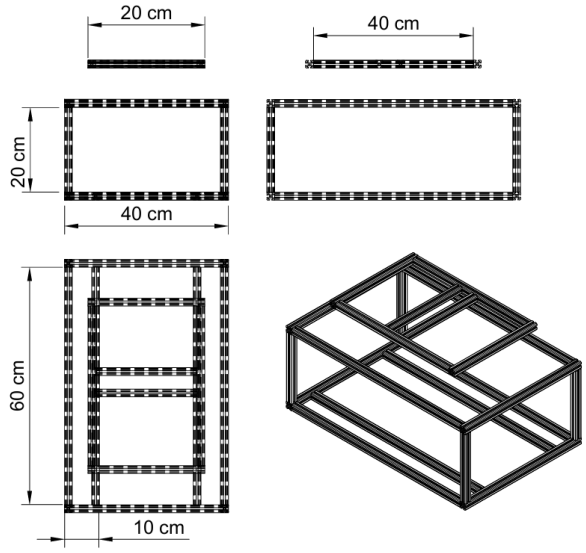
Perancangan rangka merupakan tahap awal dalam pembuatan struktur mekanik robot mobile. Rangka berfungsi sebagai struktur utama yang menopang seluruh komponen robot seperti motor penggerak, baterai, sistem kontrol, sensor, serta mekanisme tambahan lainnya. Oleh karena itu, rangka harus dirancang agar memiliki kekuatan struktural yang cukup, stabil, serta mampu menahan beban total robot selama proses operasi.

Pada penelitian ini rangka robot dirancang dengan bentuk persegi panjang untuk memberikan stabilitas yang baik saat robot bergerak. Dimensi rangka disesuaikan dengan kebutuhan ruang untuk pemasangan komponen elektronik, sistem penggerak, serta sistem mekanik lainnya. Dimensi rangka robot yang dirancang ditunjukkan pada Tabel berikut.

Tabel 3.4 Dimensi Rangka Robot

Parameter	Nilai
Panjang rangka	60 cm
Lebar rangka	40 cm
Tinggi rangka	20 cm

Dimensi tersebut dipilih agar robot memiliki ruang yang cukup untuk menempatkan komponen seperti baterai, mikrokontroler, sensor, serta motor penggerak tanpa mengganggu stabilitas robot saat bergerak.



Gambar 3.6 Dimensi Rangka Robot

3.5.4.2 Sistem Penggerak Robot

Sistem penggerak robot merupakan bagian mekanik yang berfungsi untuk menghasilkan pergerakan robot pada lingkungan operasinya. Pada penelitian ini robot mobile menggunakan konfigurasi dua roda penggerak yang dikontrol dengan prinsip differential drive. Konfigurasi ini umum digunakan pada mobile robot karena memungkinkan pengendalian arah dan kecepatan secara sederhana namun efektif.

Robot terdiri dari dua roda utama, yaitu roda kiri dan roda kanan, yang masing-masing digerakkan oleh motor secara independen. Dengan pengaturan kecepatan dan arah putaran kedua roda tersebut, robot dapat bergerak lurus, berbelok, maupun berputar pada tempat (*in-place rotation*). Prinsip ini sesuai dengan konsep differential drive, di mana perbedaan kecepatan antara roda kanan dan kiri menentukan arah gerak robot

a) Kinematika Differential Drive

Model kinematika differential drive digunakan untuk menggambarkan hubungan antara kecepatan roda kiri dan roda kanan terhadap pergerakan robot secara keseluruhan. Model ini banyak digunakan pada robot mobile karena memiliki struktur mekanik sederhana serta mudah dikontrol.

Hubungan antara kecepatan roda dan pergerakan robot mengacu pada Persamaan 2.3 dan Persamaan 2.4, yang menyatakan hubungan antara kecepatan roda kiri dan kanan terhadap kecepatan linear dan kecepatan sudut robot.

Parameter fisik robot yang digunakan dalam penelitian ini adalah sebagai berikut.

Tabel 3.5 Parameter Fisik Robot

Parameter	Nilai
Jarak roda kiri–kanan	27 cm = 0,27 m
Diameter roda	14,8 cm = 0,148 m
Jari-jari roda	7,4 cm = 0,074 m

Robot dirancang memiliki kecepatan operasi sebesar 0,5 m/s. Berdasarkan Persamaan 2.5, diperoleh kecepatan sudut roda sebesar 6,76 rad/s. Selanjutnya, pada kondisi robot berbelok digunakan kecepatan roda kanan sebesar 0,635 m/s dan kecepatan roda kiri sebesar 0,365 m/s. Berdasarkan Persamaan 2.4, diperoleh kecepatan sudut robot sebesar 1 rad/s. Berdasarkan hasil tersebut, radius belok robot yang dihitung menggunakan Persamaan 2.6 adalah sebesar 0,5 m. Ringkasan karakteristik pergerakan robot disajikan pada Tabel 3.6.

Tabel 3.6 Karakteristik Pergerakan Robot

Parameter	Nilai
Kecepatan linear robot	0,5 m/s
Kecepatan sudut roda	6,76 rad/s

Kecepatan sudut robot saat belok	1 rad/s
Radius belok robot	0,5 m

Nilai-nilai tersebut menjadi dasar dalam menentukan parameter kontrol motor serta implementasi navigasi robot.

b) Perhitungan Beban Robot

Perhitungan beban robot dilakukan untuk mengetahui gaya yang harus diatasi oleh sistem penggerak agar robot dapat bergerak dengan stabil. Beban robot terdiri dari massa rangka, baterai, komponen elektronik, motor penggerak, serta beban maksimum yang dapat diangkut oleh robot. Nilai massa total robot menjadi parameter utama dalam menentukan gaya yang bekerja pada robot serta kebutuhan torsi motor pada sistem penggerak.

Tabel 3.7 Massa Komponen

Komponen	Massa
Rangka robot	2 kg
Baterai	4 kg
Motor dan komponen elektronik	9 kg
Beban maksimum	10 kg
Total massa robot	25 kg

Berdasarkan hasil Tabel 3.7 tersebut, massa total robot mobile adalah 25 kg**.**

c) Perhitungan Gaya Berat Robot

Setelah massa robot diketahui sebesar 25 kg, gaya berat robot dihitung mengacu pada Persamaan 2.7:

$$W = 25 \times 9{,}81 = 245{,}25 \text{ N}$$

Pada kondisi permukaan datar, gaya normal memiliki nilai yang sama dengan gaya berat, sehingga diperoleh:

$$N = 245{,}25 \text{ N}$$

Gaya gesek dihitung menggunakan koefisien gesek $\mu = 0{,}03$mengacu pada Persamaan 2.9:

$$F_f = 0{,}03 \times 245{,}25 = 7{,}36 \text{ N}$$

Nilai tersebut merupakan gaya minimum yang harus diatasi agar robot dapat mulai bergerak.

Robot menggunakan konfigurasi 6 roda (4 caster dan 2 roda penggerak), sehingga beban diasumsikan terdistribusi merata pada setiap roda:

$$F_{roda} = \frac{245{,}25}{6} = 40{,}88 \text{ N}$$

Karena hanya 2 roda yang digerakkan, gaya per motor diperoleh:

$$F_{motor} = \frac{7{,}36}{2} = 3{,}68 \text{ N}$$

Dengan faktor keamanan sebesar 2, gaya desain menjadi:

$$F_{design} = 2 \times 7{,}36 = 14{,}72 \text{ N}$$

Sehingga gaya desain per motor adalah:

$$F_{motor} = \frac{14{,}72}{2} = 7{,}36 \text{ N}$$

Nilai gaya desain tersebut digunakan sebagai dasar dalam menentukan kebutuhan torsi motor pada sistem robot mobile.

d) Perhitungan Torsi Motor

Setelah diperoleh nilai gaya yang harus diatasi oleh robot, langkah selanjutnya adalah menghitung kebutuhan torsi motor pada sistem penggerak.

Dengan jari-jari roda sebesar 0,074 m dan gaya per motor sebesar 7,36 N, torsi dihitung mengacu pada Persamaan 2.10:

$$T = 7{,}36 \times 0{,}074 = 0{,}545 \text{ Nm}$$

Dengan faktor keamanan sebesar 3, kebutuhan torsi motor menjadi:

$$T_{design} = 3 \times 0{,}545 = 1{,}635 \text{ Nm}$$

Nilai tersebut kemudian dikonversi ke satuan kg·cm mengacu pada Persamaan 2.11:

$$T = 1{,}635 \times 10{,}197 \approx 16{,}67 \text{ kg.cm}$$

Dengan demikian, setiap motor minimal harus mampu menghasilkan torsi sekitar 16,67 kg·cm agar robot dapat bergerak dengan aman.

Berdasarkan spesifikasi motor pada Tabel 2.11, motor memiliki torsi kontinu sebesar 30,57 kg·cm dan torsi *stall* sebesar 65,74 kg·cm. Nilai tersebut lebih besar dibandingkan kebutuhan torsi hasil perhitungan, sehingga motor dinilai mampu menggerakkan robot secara aman dan stabil.

e) Kontrol PWM Motor dan Kecepatan Roda

Pengaturan kecepatan motor DC pada robot mobile dilakukan menggunakan metode *Pulse Width Modulation* (PWM) yang dihasilkan oleh mikrokontroler ATmega2560. Hubungan antara *duty cycle* dan tegangan efektif mengacu pada Persamaan 2.12 dan Persamaan 2.13. Sinyal PWM memiliki resolusi 8-bit dengan rentang nilai 0–255 yang digunakan untuk mengontrol driver motor BTS7960.

Berdasarkan spesifikasi motor pada Tabel 2.12, kecepatan maksimum motor adalah 12 RPM. Kecepatan sudut motor dihitung sebagai:

$$\omega = \frac{2\pi \times 12}{60} \approx 1{,}26 \text{ rad/s}$$

Dengan jari-jari roda sebesar 0,074 m, diperoleh kecepatan linear robot:

$$v = 1{,}26 \times 0{,}074 \approx 0{,}093 \text{ m/s}$$

Pada kondisi bergerak lurus, kedua motor menggunakan nilai PWM maksimum sehingga robot bergerak maju dengan kecepatan yang sama:

$$D = 1 \Rightarrow PWM = 255$$

Pada kondisi belok, perbedaan kecepatan roda diatur melalui perbedaan nilai *duty cycle*. Dengan pendekatan roda kiri sebesar 70% dari kondisi maksimum diperoleh:

$$D_{kiri} = 0{,}7 \Rightarrow PWM_{kiri} = 0{,}7 \times 255 \approx 179$$

Sedangkan roda kanan menggunakan nilai maksimum:

$$D_{kanan} = 1 \Rightarrow PWM_{kanan} = 255$$

Untuk gerakan belok ke kanan, konfigurasi nilai PWM dibalik. Perbedaan nilai PWM tersebut menghasilkan perbedaan kecepatan roda sehingga robot dapat melakukan manuver belok. Pengaturan PWM motor dirangkum pada Tabel 3.8.

Tabel 3.8 Pengaturan PWM Motor

Kondisi Robot	PWM Motor Kiri	PWM Motor Kanan	Kondisi Robot
Bergerak lurus	255	255	Bergerak lurus
Belok kiri	-179	255	Belok kiri
Belok kanan	255	-179	Belok kanan

Nilai PWM tersebut digunakan oleh mikrokontroler ATmega2560 untuk mengontrol driver motor sehingga robot dapat melakukan gerakan maju maupun

berbelok sesuai dengan perintah navigasi yang dihasilkan oleh algoritma *Reinforcement learning*.

3.6 Perancangan Sistem Navigasi

Perancangan sistem navigasi pada robot mobile dalam penelitian ini menggunakan pendekatan *Reinforcement learning* (RL) dengan algoritma *Q-Learning*. Metode ini memungkinkan robot mempelajari jalur navigasi secara mandiri melalui interaksi dengan lingkungan tanpa memerlukan model matematis yang kompleks.

Dalam metode ini robot diperlakukan sebagai agen yang berinteraksi dengan lingkungan untuk menentukan aksi terbaik pada setiap kondisi tertentu. Setiap aksi yang dilakukan robot akan menghasilkan nilai *reward* yang digunakan untuk memperbarui nilai pada *Q-Table*, sehingga robot secara bertahap dapat mempelajari jalur navigasi yang paling optimal menuju tujuan.

Untuk mempermudah proses pelatihan algoritma, lingkungan navigasi robot dimodelkan dalam bentuk grid environment dua dimensi. Representasi ini memungkinkan posisi robot, lokasi tujuan, serta obstacle dimodelkan secara diskrit sehingga proses pembelajaran dapat dilakukan secara sistematis.

Pada tahap implementasi, hasil pembelajaran berupa *Q-Table* digunakan sebagai dasar pengambilan keputusan navigasi robot. Raspberry Pi menjalankan algoritma navigasi dan membaca *Q-Table*, sedangkan ATmega2560 berfungsi sebagai pengendali aktuator dan pembaca sensor. Integrasi kedua perangkat ini memungkinkan robot mobile bergerak secara otonom menuju tujuan yang telah ditentukan.

3.6.1 Model Grid Environment

Pada tahap simulasi dan pelatihan algoritma *Reinforcement learning*, lingkungan navigasi robot dimodelkan dalam bentuk grid environment dua dimensi. Grid environment merupakan representasi diskrit dari ruang navigasi robot yang dibagi menjadi beberapa sel dengan ukuran yang sama. Setiap sel pada grid merepresentasikan posisi yang dapat ditempati oleh robot selama proses navigasi.

Dalam penelitian ini digunakan lingkungan grid berukuran: 5×7.

yang terdiri dari 5 baris dan 7 kolom, sehingga jumlah total sel pada lingkungan simulasi adalah:

$$N = 5 \times 7 = 35$$

Representasi lingkungan grid yang digunakan pada penelitian ini ditunjukkan sebagai berikut.

A	.	.	B	.	.	C
.	X	.	X	.	X	.
.	X	.	.	.	.	.
.	X	.	.	X	X	.
.	.	.	S	.	.	X

Gambar 3.7 *Grid Enviroment*

Keterangan:

A = titik tujuan pertama

B = titik tujuan kedua

C = titik tujuan ketiga

S = posisi awal robot (Home)

X = obstacle statis

. = area bebas

Berdasarkan Gambar 3.7, lingkungan pelatihan direpresentasikan dalam bentuk *grid environment* berukuran 5 × 7 yang terdiri atas posisi awal robot (*Home*), tiga titik tujuan, area bebas, dan beberapa *obstacle* statis. Robot memulai navigasi dari posisi Home (3,4) dan harus menentukan aksi yang tepat untuk mencapai salah satu tujuan dengan menghindari *obstacle* yang terdapat pada lingkungan.

Sedangkan lokasi tujuan robot berada pada koordinat:

$$A = (0,0)$$

$$B = (0,3)$$

$$C = (0,6)$$

Obstacle ditempatkan pada beberapa koordinat grid untuk mensimulasikan hambatan yang harus dihindari oleh robot selama proses navigasi.

A	.	.	B	.	.	C
.	X	.	X	.	X	.
.	X	.	.	.	.	.
.	X	.	.	X	X	.
.	.	.	S	.	.	X

(0,0)	(1,0)	(2,0)	(3,0)	(4,0)	(5,0)	(6,0)
(0,1)	(1,1)	(2,1)	(3,1)	(4,1)	(5,1)	(6,1)
(0,2)	(1,2)	(2,2)	(3,2)	(4,2)	(5,2)	(6,2)
(0,3)	(1,3)	(2,3)	(3,3)	(4,3)	(5,3)	(6,3)
(0,4)	(1,4)	(2,4)	(3,4)	(4,4)	(5,4)	(6,4)

Gambar 3.8 Koordinat Grid

Berdasarkan Gambar 3.8, setiap sel pada *grid environment* memiliki koordinat *(x,y)* yang digunakan sebagai representasi *state* pada algoritma Q-Learning. Koordinat tersebut menjadi acuan untuk menentukan posisi robot, posisi tujuan, dan lokasi *obstacle*, sehingga setiap perpindahan robot dari satu sel ke sel lainnya dapat direpresentasikan sebagai perubahan *state* selama proses navigasi. Model grid environment ini digunakan sebagai lingkungan pelatihan algoritma Q-*Learning*, di mana robot mencoba berbagai aksi pada setiap state untuk menemukan jalur navigasi yang paling optimal menuju tujuan. Meskipun model grid digunakan pada tahap simulasi, pada implementasi robot nyata lingkungan navigasi tidak menggunakan garis atau jalur fisik. Konsep grid hanya digunakan sebagai model logika navigasi, sedangkan posisi robot pada sistem nyata direpresentasikan menggunakan pembacaan sensor hall effect melalui *pulse* yang dibaca.

3.6.2 Penentuan State, *Action*, *Reward*

Dalam penerapan algoritma *Reinforcement learning*, proses pembelajaran agen ditentukan oleh tiga komponen utama yaitu *state, action*, dan *reward*. Ketiga komponen ini mendefinisikan bagaimana robot mobile berinteraksi dengan lingkungan serta bagaimana robot menentukan keputusan pergerakan pada setiap kondisi navigasi.

a) State

State merupakan representasi kondisi lingkungan yang diamati oleh robot pada suatu waktu tertentu dan digunakan sebagai dasar dalam menentukan aksi yang akan dilakukan.

Pada penelitian ini state robot didefinisikan berdasarkan posisi robot pada grid, arah hadap robot, serta tujuan navigasi yang sedang aktif. Secara matematis state dapat dinyatakan sebagai: $s = (x, y, d, g)$

dengan:

x, y = posisi robot pada grid

d = arah hadap robot

g = tujuan navigasi aktif

Orientasi robot terdiri dari empat kemungkinan arah:

$$d \in \{N, E, S, W\}$$

yang merepresentasikan North (Utara), East (Timur), South (Selatan), dan West (Barat).

Sedangkan tujuan navigasi robot dinyatakan sebagai:

$$g \in \{A, B, C, HOME\}$$

Pada lingkungan simulasi digunakan grid berukuran: 5×7

sehingga jumlah posisi yang mungkin ditempati robot adalah 35 posisi. Dengan mempertimbangkan empat orientasi robot dan empat kemungkinan tujuan, maka jumlah total state dalam sistem adalah:

$$N_{state} = 35 \times 4 \times 4$$

$$N_{state} = 560$$

Jumlah state tersebut digunakan sebagai dasar dalam pembentukan *Q-Table* pada algoritma *Q-Learning*.

b) *Action*

Action merupakan aksi yang dapat dilakukan robot pada setiap state yang diamati. Dalam penelitian ini robot memiliki tiga aksi utama yang berkaitan dengan pergerakan robot pada lingkungan navigasi.

$$a \in \{Forward, TurnLeft, TurnRight\}$$

Penjelasan aksi robot ditunjukkan pada tabel berikut.

Tabel 3.9 Aksi Robot

Action	**Deskripsi**

Forward	Robot bergerak maju satu grid
Turn Left	Robot berputar 90° ke kiri
Turn Right	Robot berputar 90° ke kanan

Aksi *Forward* akan mengubah posisi robot pada grid sesuai dengan arah hadap robot, sedangkan *Turn Left* dan *Turn Right* hanya mengubah orientasi robot tanpa mengubah posisi robot.

c) *Reward*

Reward merupakan nilai evaluasi yang diberikan kepada robot setelah melakukan suatu aksi. *Reward* berfungsi sebagai umpan balik untuk menentukan apakah aksi yang dilakukan menghasilkan kondisi yang baik atau tidak.

Tabel *reward* yang digunakan pada penelitian ini adalah sebagai berikut:

Tabel 3.10 Nilai *Reward Training*

Kondisi	***Reward***
Robot mencapai tujuan	+500
Robot menabrak obstacle atau keluar grid	-500
Langkah pergerakan normal	-1
Bergerak mendekati tujuan	+6
Bergerak menjauhi tujuan	-6
Aksi rotasi (*Turn Left* / *Turn Right*)	-5

Dengan mekanisme *reward* ini, robot akan belajar memilih aksi yang menghasilkan nilai *reward* paling optimal.

Penentuan nilai *reward* mengacu pada prinsip *reward shaping*, yaitu memberikan nilai berbeda untuk mengarahkan agen mencapai tujuan secara efisien. *Reward* besar pada *goal* (+500) diberikan agar kondisi tujuan menjadi prioritas utama, sedangkan penalti pada obstacle (-500) bertujuan menghindari kondisi yang tidak diinginkan [7].

Reward kecil pada setiap langkah (-1) digunakan untuk meminimalkan jumlah langkah, sementara tambahan *reward* (+6 dan -6) membantu mempercepat pembelajaran dengan mengarahkan robot menuju tujuan. Penalti rotasi (-5) diberikan untuk mengurangi gerakan yang tidak efisien. Secara keseluruhan, skema ini membantu agen menemukan jalur optimal melalui proses pembelajaran [3].

3.6.3 Implementasi Algoritma Q-*Learning*

Implementasi algoritma Q-*Learning* pada penelitian ini dilakukan dengan membangun sebuah Q-Table yang berfungsi untuk menyimpan nilai aksi pada setiap state yang mungkin terjadi pada lingkungan navigasi. Q-Table tersebut digunakan sebagai dasar pengambilan keputusan robot dalam menentukan aksi terbaik berdasarkan pengalaman yang diperoleh selama proses pembelajaran.

Struktur Q-Table pada sistem navigasi ini direpresentasikan sebagai:

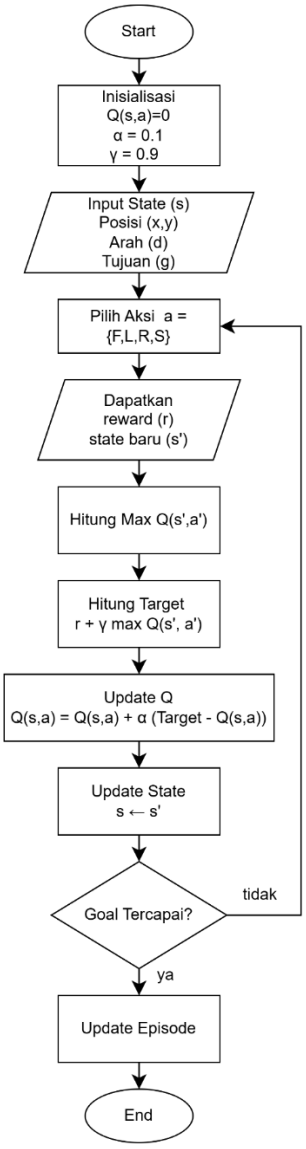
$$Q[x, y, d, g, a]$$

di mana x, ymerupakan posisi robot pada grid environment, $d$menunjukkan orientasi robot, $g$merupakan tujuan navigasi yang aktif, dan $a$merupakan aksi yang dapat dilakukan robot.

Pada awal proses *training* seluruh nilai Q pada Q-Table diinisialisasi dengan nilai 0. Hal ini menunjukkan bahwa robot belum memiliki informasi mengenai kualitas dari setiap aksi yang dapat dilakukan pada lingkungan navigasi.

Setelah robot melakukan suatu aksi pada state tertentu, robot akan menerima nilai *reward* dari lingkungan yang dilambangkan dengan r. Nilai *reward* ini diperoleh berdasarkan kondisi yang terjadi setelah robot melakukan aksi, di mana perancangan nilai *reward* mengacu pada Tabel 3.10 (nilai *reward training*). Secara umum, *reward* positif diberikan ketika robot bergerak mendekati tujuan atau berhasil mencapai target, sedangkan *reward* negatif diberikan ketika robot menabrak obstacle atau bergerak menjauhi tujuan. Nilai *reward* tersebut menggambarkan apakah aksi yang dilakukan memberikan hasil yang baik atau buruk. Oleh karena itu, *reward* digunakan sebagai indikator langsung terhadap kualitas aksi yang dilakukan robot pada setiap langkah dalam proses pembelajaran.

Selain mempertimbangkan *reward* saat ini, algoritma Q-*Learning* juga memperhitungkan kemungkinan *reward* yang dapat diperoleh pada langkah berikutnya. *Reward* masa depan dihitung dari nilai Q terbesar yang tersedia pada state berikutnya.



Gambar 3.9 Diagram Alir Algoritma Q-*Learning*

Flowchart pada Gambar 3.9 menggambarkan proses pembelajaran menggunakan algoritma *Q-Learning* pada sistem navigasi robot mobile. Proses dimulai dengan inisialisasi Q-table serta penentuan parameter pembelajaran, yaitu *learning rate* (α = 0,1) dan *discount factor* (γ = 0,9).

State awal ditentukan sebagai kombinasi posisi robot, orientasi, dan tujuan., sedangkan orientasi diperoleh dari sensor IMU. Berdasarkan state tersebut, robot memilih aksi yang tersedia, yaitu *forward*, *turn left*, *turn right*, atau *stop*.

Setelah aksi dilakukan, robot memperoleh *reward* dan berpindah ke state berikutnya. Pembaruan nilai Q dilakukan dengan mengacu pada Persamaan (2.23). Selanjutnya, state diperbarui dan proses pemilihan aksi diulang.

Proses ini berlangsung secara berulang hingga robot mencapai tujuan dalam satu episode. Setelah itu, sistem memeriksa jumlah episode dan mengulangi proses pembelajaran hingga batas yang ditentukan tercapai, sehingga diperoleh kebijakan navigasi yang optimal.

Sebagai contoh, untuk memperjelas proses pembaruan nilai Q, berikut diberikan ilustrasi perhitungan pada beberapa langkah dalam proses *training* yang dilakukan secara bertahap dalam bentuk episode.

Pada episode pertama, seluruh nilai Q masih diinisialisasi dengan nol.

Tabel 3.11 Data Langkah 1 (Episode 1)

Parameter	Nilai
State (s)	(4,3,N,*Goal*$_0$)
Aksi (a)	*Left*
State berikutnya (s')	(4,3,W,*Goal*$_0$)
Reward (r)	-5
max Q(s',a')	0

Berdasarkan Tabel 3.11, parameter pada langkah pertama episode pertama digunakan sebagai masukan dalam pembaruan nilai Q-value menggunakan Persamaan (2.23). Parameter tersebut meliputi *state* awal, aksi yang dipilih, *state* berikutnya, *reward*, dan nilai maksimum Q pada *state* berikutnya.

$$Q(s,a) = 0 + 0{,}1[-5 + 0{,}9(0) - 0]$$

$$Q(s,a) = -0{,}5$$

Tabel 3.12 Data Langkah 2 (Episode 1)

Parameter	Nilai
State (s)	(4,3,N,*Goal*$_0$)
Aksi (a)	*Forward*
State berikutnya (s')	(3,3,N,*Goal*$_0$)
Reward (r)	5
max Q(s',a')	0

Berdasarkan Tabel 3.12, parameter pada langkah kedua digunakan sebagai masukan dalam Persamaan (2.23) untuk menghitung pembaruan nilai Q-value berdasarkan perpindahan robot menuju *state* berikutnya.

$$Q(s,a) = 0 + 0{,}1[5 + 0{,}9(0) - 0]$$

$$Q(s,a) = 0{,}5$$

Tabel 3.13 Data Langkah 3 (Episode 1)

Parameter	Nilai
State (s)	(3,3,N,*Goal*$_0$)
Aksi (a)	*Forward*
State berikutnya (s')	(2,3,N,*Goal*$_0$)
Reward (r)	5
max Q(s',a')	0

Berdasarkan Tabel 3.13, parameter yang diperoleh pada langkah ketiga disubstitusikan ke dalam Persamaan (2.23) sehingga diperoleh nilai pembaruan Q-value untuk pasangan *state-action* yang dilalui robot.

$$Q(s,a) = 0 + 0{,}1[5 + 0{,}9(0) - 0]$$

$$Q(s,a) = 0{,}5$$

Tabel 3.14 Data Langkah 4 (Episode 1)

Parameter	Nilai
State (s)	(2,3,N,*Goal*$_0$)
Aksi (a)	*Forward*
State berikutnya (s')	*Goal*
Reward (r)	500
max Q(s',a')	0

Berdasarkan Tabel 3.14, parameter pada langkah keempat digunakan dalam Persamaan (2.23) untuk menghitung pembaruan Q-value ketika robot berhasil mencapai *goal* dan memperoleh *reward* terminal.

$$Q(s,a) = 0 + 0{,}1[500 + 0{,}9(0) - 0]$$

$$Q(s,a) = 50$$

Tabel 3.15 Data Langkah 4 (Episode 1)

State	*Forward*	*Left*	*Right*
(4,3,N)	0,5	-0,5	0

(3,3,N)	0,5	0	0
(2,3,N)	50	0	0

Berdasarkan Tabel 3.15, ditampilkan hasil pembaruan nilai Q-value setelah seluruh langkah pada episode pertama selesai dilakukan. Nilai-nilai tersebut merupakan hasil perhitungan menggunakan Persamaan (2.23) pada setiap pasangan *state-action* yang telah dikunjungi.

Berdasarkan contoh perhitungan pada episode pertama, terlihat bahwa pembaruan nilai Q hanya dipengaruhi oleh *reward* saat ini karena nilai pada state berikutnya masih bernilai nol. Nilai Q akan terus diperbarui seiring dengan proses pembelajaran yang dilakukan secara berulang pada episode berikutnya. Pada episode kedua, nilai Q pada beberapa state telah terisi dari hasil pembelajaran sebelumnya, sehingga pembaruan nilai Q mulai mempertimbangkan *reward* masa depan melalui nilai maksimum pada state berikutnya.

Tabel 3.16 Data Langkah 1 (Episode 2)

Parameter	Nilai
State (s)	(4,3,N,$Goal_0$)
Aksi (a)	*Forward*
State berikutnya (s')	(3,3,N,$Goal_0$)
Reward (r)	5
max Q(s',a')	0,5

Berdasarkan Tabel 3.16, parameter pada langkah pertama episode kedua digunakan sebagai masukan dalam Persamaan (2.23). Berbeda dengan episode pertama, nilai maksimum Q pada *state* berikutnya telah diperoleh dari hasil pembelajaran sebelumnya sehingga memengaruhi proses pembaruan Q-value.

$$Q(s,a) = 0{,}5 + 0{,}1[5 + 0{,}9(0{,}5) - 0{,}5]$$

$$Q(s,a) = 0{,}5 + 0{,}1(5{,}45 - 0{,}5)$$

$$Q(s,a) = 0{,}5 + 0{,}495 = 0{,}995$$

Tabel 3.17 Data Langkah 4 (Episode 1)

State	*Forward*	*Left*	*Right*
(4,3,N)	**0,995**	-0,5	0
(3,3,N)	0,5	0	0
(2,3,N)	50	0	0

Berdasarkan Tabel 3.17, ditampilkan hasil pembaruan nilai Q-value setelah proses perhitungan pada episode kedua dilakukan menggunakan Persamaan (2.23). Nilai Q-value pada pasangan *state-action* yang telah dikunjungi mengalami peningkatan sesuai dengan hasil pembelajaran dari episode sebelumnya.

Nilai tersebut kemudian disimpan pada Q-Table dan akan digunakan dalam proses pengambilan keputusan pada langkah navigasi berikutnya.

Proses pembelajaran dilakukan secara berulang selama sejumlah episode hingga nilai Q pada Q-Table mengalami konvergensi. Seiring bertambahnya jumlah episode *training*, robot akan semakin memahami aksi mana yang memberikan *reward* terbaik pada setiap state.

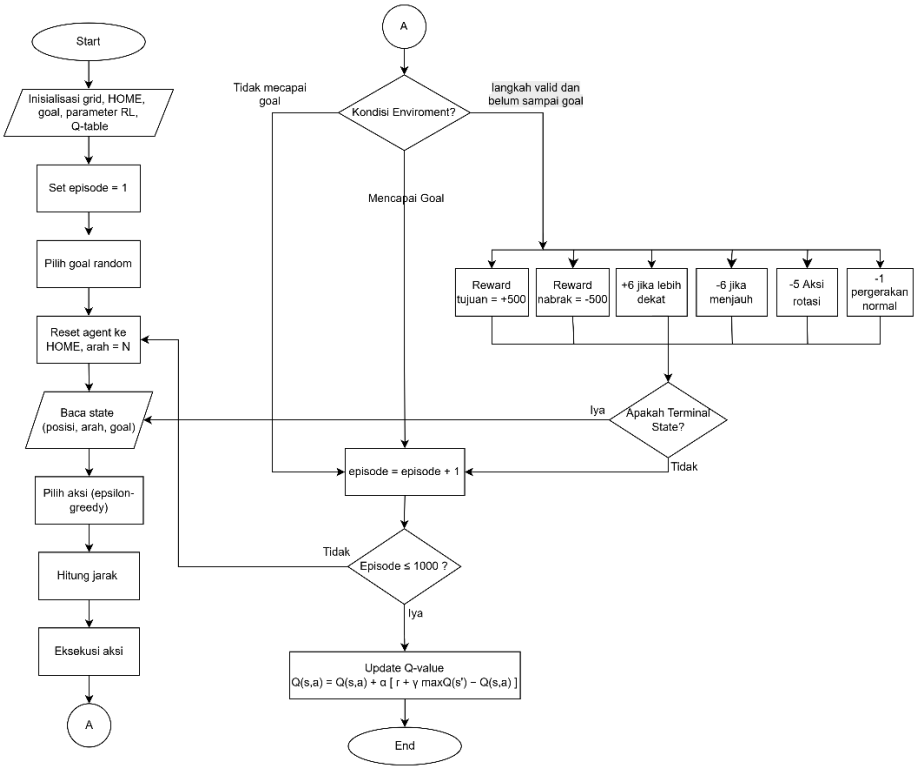
Setelah proses *training* selesai, Q-table yang dihasilkan digunakan sebagai dasar navigasi robot. Ketika robot berada pada suatu state tertentu, robot memilih aksi terbaik dengan mengacu pada Persamaan (2.22).

Dengan pendekatan tersebut, robot akan menentukan aksi berdasarkan nilai Q tertinggi yang diperoleh selama proses pembelajaran. Hal ini memungkinkan robot memilih jalur navigasi yang lebih efisien, menghindari obstacle, serta mencapai tujuan dengan jumlah langkah yang lebih optimal.

3.7 *Training* dan Simulasi *Reinforcement learning*

Pada tahap ini dilakukan proses simulasi dan *training algoritma Reinforcement learning* menggunakan bahasa pemrograman Python. Simulasi bertujuan untuk melatih robot mobile agar dapat menemukan kebijakan navigasi optimal sebelum diimplementasikan pada robot fisik.

Lingkungan simulasi menggunakan *grid environment* berukuran 5 × 7 yang telah dijelaskan pada subbab sebelumnyaelama proses *training* robot akan melakukan eksplorasi berbagai aksi untuk mencapai tujuan navigasi, kemudian nilai Q pada setiap pasangan state dan *action* akan diperbarui berdasarkan *reward* yang diterima.



Gambar 3.10 Diagram Alir *Training Reinforcement learning*

Hasil dari proses *training* ini berupa *Q-Table* yang menyimpan nilai kualitas aksi pada setiap kondisi lingkungan. *Q-Table* tersebut kemudian digunakan sebagai dasar pengambilan keputusan navigasi robot pada tahap implementasi.

3.7.1 Parameter *Training*

Proses *training* algoritma *Reinforcement learning* pada penelitian ini dilakukan menggunakan metode Q-*Learning* pada lingkungan simulasi berbentuk grid environment berukuran 5 × 7. Selama proses *training*, robot akan melakukan eksplorasi terhadap lingkungan dengan mencoba berbagai aksi pada setiap state dan menerima *reward* berdasarkan kondisi yang terjadi setelah aksi tersebut dilakukan. Nilai *reward* tersebut kemudian digunakan untuk memperbarui nilai pada Q-Table hingga diperoleh kebijakan navigasi yang optimal.

Proses pembelajaran dilakukan selama sejumlah episode dengan batas maksimum langkah pada setiap episode. Selain itu, beberapa parameter pembelajaran digunakan untuk mengatur proses pembaruan nilai Q serta keseimbangan antara eksplorasi dan eksploitasi selama proses *training* berlangsung.

Pada penelitian ini digunakan parameter *training* sebagai berikut.

Tabel 3.18 Parameter *Training Reinforcement learning*

Parameter	Nilai	Keterangan
Ukuran grid	5 × 7	Lingkungan simulasi
Episode *training*	1000	Jumlah iterasi pembelajaran
Maximum *step*	150	Langkah maksimum per episode
Learning rate (α)	0.1	Tingkat pembaruan nilai Q
Discount factor (γ)	0.9	Bobot *reward* masa depan
Initial epsilon	1.0	Probabilitas eksplorasi awal
Epsilon decay	0.9995	Penurunan eksplorasi
Minimum epsilon	0.05	Batas minimum eksplorasi

Parameter *learning rate* (α) menentukan seberapa besar nilai Q diperbarui pada setiap iterasi. Sedangkan *discount factor* (γ) menentukan seberapa besar pengaruh *reward* masa depan terhadap keputusan saat ini.

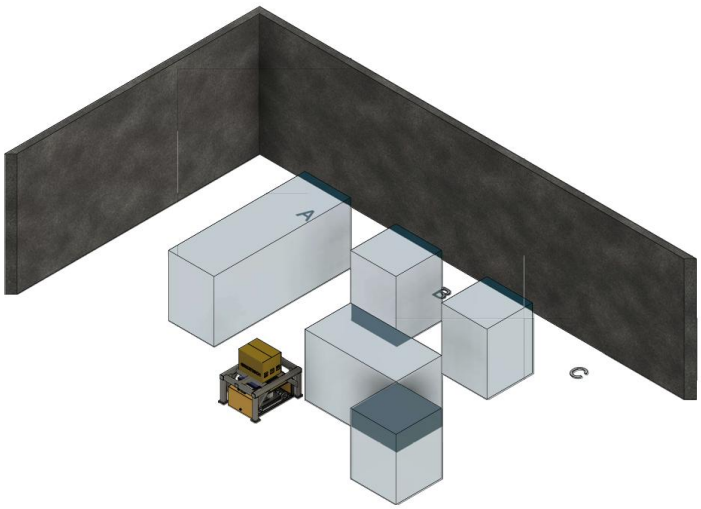
Parameter *epsilon* (ε) digunakan pada kebijakan eksplorasi *ε-greedy*. Pada awal *training* robot akan banyak melakukan eksplorasi dengan nilai ε yang tinggi. Seiring bertambahnya episode, nilai ε akan berkurang sehingga robot lebih banyak melakukan eksploitasi terhadap aksi terbaik yang telah dipelajari.

3.8 Implementasi *Hardware*

Implementasi *hardware* pada sistem robot dilakukan dengan mengintegrasikan mikrokontroler ATmega, Raspberry Pi, sensor, serta aktuator dalam satu sistem yang saling terhubung. Raspberry Pi dan ATmega berkomunikasi untuk mengirim dan menerima data yang diperlukan dalam proses pengendalian robot.

Mikrokontroler ATmega berperan dalam membaca data dari sensor seperti, IMU, dan sensor jarak, serta mengendalikan aktuator berupa motor DC melalui driver motor. Sementara itu, Raspberry Pi digunakan untuk memproses data yang diterima dan mengirimkan perintah gerakan ke mikrokontroler.

Seluruh komponen dihubungkan melalui sistem wiring yang mengatur distribusi daya dan jalur komunikasi, sehingga setiap bagian dapat bekerja secara terintegrasi dalam menjalankan sistem robot.



Gambar 3.11 3D Lingkungan *Warehouse*

Integrasi kedua perangkat tersebut memungkinkan robot untuk menentukan jalur navigasi secara mandiri berdasarkan kondisi lingkungan yang diperoleh dari sensor. Informasi tersebut kemudian digunakan untuk membentuk state pada algoritma *Reinforcement learning* sehingga robot dapat menentukan aksi navigasi yang paling optimal.

3.8.1 Lingkungan dan Arsitektur Sistem

Arsitektur sistem robot terdiri dari beberapa komponen utama yang bekerja secara terintegrasi untuk menjalankan sistem navigasi robot *mobile*. Sistem ini menggabungkan perangkat komputasi, sensor, serta aktuator untuk memungkinkan robot bergerak secara otonom menuju lokasi tujuan. Lingkungan navigasi robot dimodelkan menggunakan pendekatan *grid environment*, di mana area kerja robot dibagi menjadi beberapa sel dengan ukuran yang sama. Pada sistem ini, ukuran *grid* ditentukan berdasarkan karakteristik roda dan encoder yang digunakan pada robot.

Robot menggunakan roda dengan diameter 14,8 cm sehingga diperoleh jari-jari roda sebesar:

$$r = \frac{14{,}8}{2} = 7{,}4 \text{ cm}$$

Setiap roda dilengkapi encoder berbasis Hall Effect dengan 10 magnet, sehingga dalam satu putaran dihasilkan 10 pulsa. Berdasarkan Persamaan (2.18), sudut rotasi roda untuk setiap pulsa adalah:

$$\theta = \frac{2\pi}{10} \times 1 = 0{,}628 \text{ rad}$$

Selanjutnya, berdasarkan Persamaan (2.19), jarak tempuh roda untuk setiap pulsa diperoleh sebagai berikut:

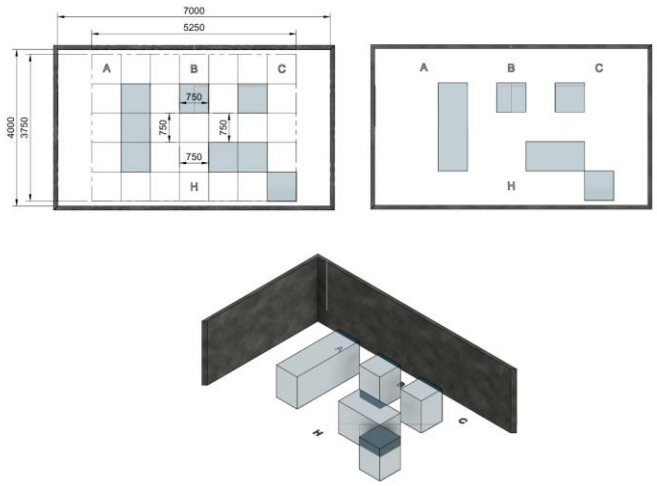
$$s = 7{,}4 \times 0{,}628$$

$$s = 4{,}65 \text{ cm/pulse}$$

Berdasarkan hasil kalibrasi dan pengujian pada robot, satu perpindahan *grid* ditetapkan sebesar 65 cm × 65 cm. Dengan demikian, jumlah pulsa yang diperlukan untuk berpindah satu *grid* dihitung sebagai berikut:

$$P = \frac{65}{4{,}65} = 13{,}98 \approx 14 \text{ pulse}$$

Sehingga pada implementasi penelitian ini, satu perpindahan *grid* direpresentasikan oleh 14 pulsa encoder. Ukuran lingkungan dapat dilihat pada Gambar 3.12.



Gambar 3. 12 *Layout Warehouse*

Untuk mendukung sistem navigasi robot, beberapa komponen sensor dan aktuator digunakan dalam sistem. Komponen-komponen tersebut berfungsi untuk memperoleh informasi mengenai kondisi robot serta mengendalikan pergerakan robot.

Tabel 3.19 Fungsi Komponen pada Penerapan Nyata

Komponen	Fungsi	Parameter Sistem
Raspberry Pi	menjalankan algoritma *Reinforcement learning* dan membaca Q-table	pengambilan keputusan navigasi

ATmega	membaca sensor dan mengendalikan aktuator	kontrol sistem robot
Encoder Hall Effect	menghitung jarak tempuh roda	menentukan posisi robot
IMU	memantau orientasi robot	koreksi arah robot
Motor DC	penggerak roda robot	pergerakan robot
Motor Driver	penguat daya motor	kontrol kecepatan dan arah

Data yang diperoleh dari sensor-sensor tersebut selanjutnya digunakan oleh sistem untuk menentukan kondisi robot pada lingkungan navigasi. Informasi tersebut kemudian digunakan untuk membentuk state pada algoritma *Reinforcement learning*, yang akan dijelaskan pada subbab berikutnya.

3.8.2 Integrasi Sensor dan State *Reinforcement learning*

Pada sistem navigasi robot yang dikembangkan, data dari berbagai sensor digunakan untuk membentuk state pada algoritma *Reinforcement learning*. State tersebut merepresentasikan kondisi robot pada lingkungan navigasi dan digunakan sebagai dasar dalam pengambilan keputusan pergerakan robot. Parameter state diperoleh dari integrasi beberapa sensor yang terpasang pada robot. Sensor-sensor tersebut memberikan informasi mengenai posisi robot, arah robot, serta tujuan navigasi yang harus dicapai.

Hubungan antara sensor dan parameter state pada sistem navigasi robot ditunjukkan pada Tabel berikut:

Tabel 3.20 Hubungan Sensor dan Parameter State

Sensor / Sistem	Parameter State	Fungsi
Encoder Hall Effect	X,Y	menghitung perpindahan robot pada grid
Memori sistem	D	menyimpan orientasi robot
IMU	D (koreksi)	memastikan rotasi robot mendekati 90°
Interface	G	menentukan tujuan navigasi robot

Posisi robot pada grid (X, Y)diperoleh dari perhitungan jarak tempuh robot menggunakan encoder roda. Ketika robot bergerak, jumlah *pulse* encoder dihitung untuk menentukan perpindahan robot pada lingkungan navigasi.

Orientasi robot (D) di representasikan dalam bentuk arah diskrit yaitu North, East, South, dan West. Nilai orientasi ini disimpan dalam memori sistem dan diperbarui setiap kali robot melakukan rotasi. Sensor IMU digunakan untuk mengoreksi sudut rotasi robot agar mendekati 90°, sehingga orientasi robot tetap sejajar dengan sistem koordinat grid.

Tujuan navigasi robot (G) diperoleh dari pembacaan Interface yang terdapat pada sistem Raspberry Pi.

Setelah seluruh parameter state diperoleh, Raspberry Pi membentuk state robot dalam bentuk:

$$S = (X, Y, D, G)$$

State tersebut kemudian digunakan untuk membaca nilai pada Q-table hasil *training* yang telah disimpan pada sistem.

Persamaan (2.24) menunjukkan bahwa robot akan memilih aksi yang memiliki nilai Q terbesar pada state yang sedang dihadapi.

Sebagai contoh, robot berada pada state:

$$S = (3{,}4, N, B)$$

yang berarti robot berada pada posisi Home (3,4), menghadap ke arah North, dan memiliki tujuan menuju lokasi B.

Nilai Q yang tersimpan pada Q-table untuk state tersebut ditunjukkan pada tabel berikut.

Tabel 3.21 Contoh Q-Table

Row	Col	Direction	*Goal*	Q *Forward*	Q *Left*	Q *Right*
4	3	0	0	12,8	4,1	3,6

Berdasarkan nilai Q tersebut, aksi *Forward* memiliki nilai terbesar. Dengan demikian aksi yang dipilih oleh sistem adalah:

$$a = Forward$$

Raspberry Pi kemudian mengirimkan perintah tersebut ke mikrokontroler ATmega melalui komunikasi serial untuk menggerakkan robot maju menuju grid berikutnya.

Proses ini akan terus berulang setiap kali robot mencapai posisi grid baru. State robot akan diperbarui berdasarkan data sensor, kemudian sistem kembali

membaca Q-table untuk menentukan aksi navigasi selanjutnya hingga robot mencapai lokasi tujuan.

3.8.3 Implementasi pada Robot Nyata

Setelah *state* robot terbentuk berdasarkan data sensor, sistem menentukan aksi pergerakan yang kemudian dikirim dari Raspberry Pi ke mikrokontroler ATmega melalui komunikasi serial. Aksi yang digunakan terdiri atas forward, turn left, dan turn right. Mikrokontroler ATmega menerjemahkan perintah tersebut menjadi sinyal PWM dan sinyal arah (*direction*) untuk mengendalikan driver motor sehingga robot dapat bergerak sesuai aksi yang dipilih.

Robot menggunakan konfigurasi *differential drive*, di mana aksi forward dilakukan dengan menggerakkan kedua motor searah, sedangkan aksi turn left dan turn right dilakukan dengan memberikan putaran berlawanan arah pada kedua motor sehingga robot berotasi pada tempat.

Perpindahan robot antar-*grid* dikendalikan menggunakan encoder Hall Effect yang dipasang pada roda penggerak. Berdasarkan Persamaan (2.18) dan Persamaan (2.19), dengan diameter roda 14,8 cm dan 10 magnet pada setiap roda, diperoleh jarak tempuh sebesar 4,65 cm untuk setiap *pulse* encoder. Berdasarkan hasil kalibrasi, satu perpindahan *grid* ditetapkan sebesar 65 cm, sehingga jumlah *pulse* yang diperlukan untuk berpindah satu *grid* adalah:

$$\frac{65}{4{,}65} = 13{,}98 \approx 14 \text{ pulse}$$

Robot akan terus bergerak hingga encoder membaca 14 pulse, kemudian mikrokontroler menghentikan motor sebagai penanda bahwa robot telah mencapai *grid* berikutnya. Ketika robot mencapai lokasi tujuan, sistem menghentikan pergerakan robot dan memberikan waktu tunggu selama sekitar 5 detik sebelum memulai siklus navigasi berikutnya.