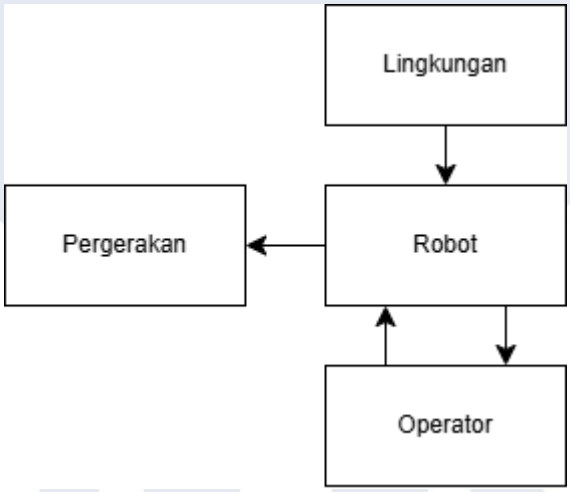
BAB III

PERANCANGAN DAN IMPLEMENTASI SISTEM

3.1 Tinjauan Desain Sistem

3.1.1 Desain Sistem Keseluruhan

Secara keseluruhan, bagian ini membahas mengenai desain sistem secara umum. Tinjauan desain akan menjelaskan mekanisme kerja sistem berdasarkan diagram blok sistem atau *data flow diagram* yan telah dirancang. Mekanisme kerja secara umum akan dijelaskan melalui *data flow diagram* (DFD) level 0.



Gambar 3.1 - DFD level 0 Mecanum Wheel Mobile Robot

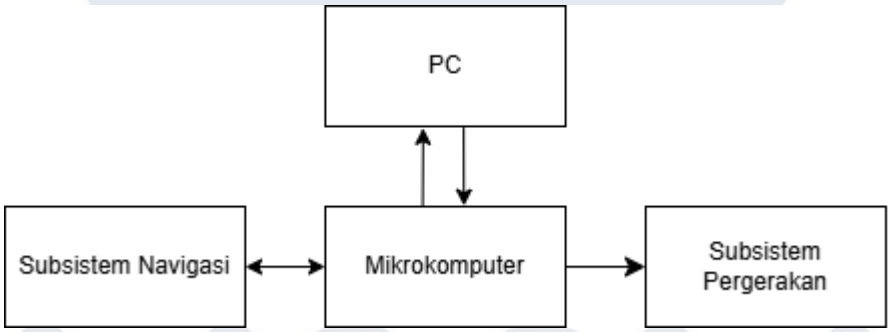
Tabel 3.1 - Penjelasan DFD Level 0 Mecanum Wheel Mobile Robot

Parameter	Keterangan
Input	● Konfigurasi sistem dari pengguna berupa koordinat *start point* dan *end point* ● Pembacaan data sensor terhadap lingkungan sekitar
Output	● Posisi robot ● Pergerakan robot ke tujuan ● Jalur algoritma perencanaan lintasan A*
Fungsi	● Mengatur pergerakan robot

	● Menampilkan data pembacaan sensor ● Menampilkan lingkungan kerja produk

Gambar 3.1 menunjukan bahwa *mecanum wheel mobile robot* memerlukan pengguna untuk memberikan perintah untuk menentukan tujuan (*end point*) ke mana robot akan bergerak. Dapat lebih jelas dilihat pada Table 3.2 terdapat sensor-sensor yang terdapat pada *mecanum wheel mobile robot* sehingga robot dapat menerima *input* dari lingkungannya dalam bentuk jarak dari sensor ke lingkungan sekitar. Pembacaan dari sensor serta perintah pergerakan dari pengguna akan menentukan pergerakan yang dihasilkan oleh *mecanum wheel mobile robot* sepanjang jalur pergerakan robot ke tujuan (*end point*). Pengoperasian *mecanum wheel mobile robot* dapat dinyatakan dalam sebuah diagram alir (*flowchart*) sebagai berikut.

3.1.2 Desain Subsistem



Gambar 3.2 - DFD Level 1 *Mecanum Wheel Mobile Robot*

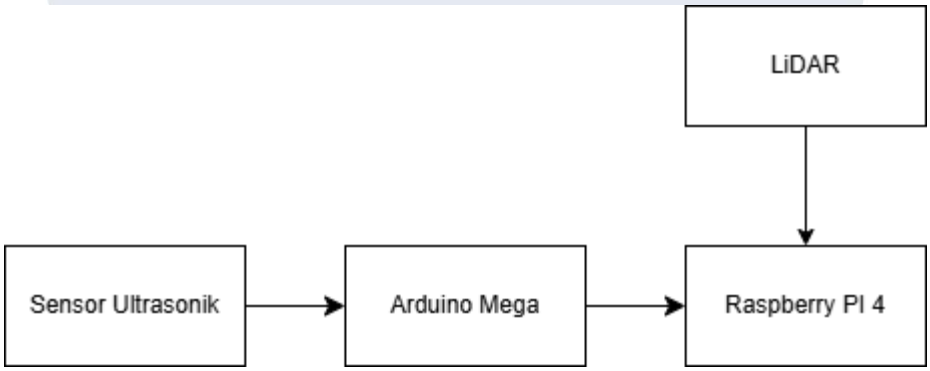
Tabel 3.2 - Penjelasan DFD Level 1 Produk

Parameter	Keterangan
Input	● Data sensor ultrasonik ● Data sensor LIDAR ● Tujuan pergerakan *Mecanum Wheel mobile robot* berupa koordinat *start point* dan *end point*
Output	● Pengendalian pergerakan robot. ● Perencanaan jalur A* dalam bentuk koordinat. ● *Obstacle avoidance* lokal

Fungsi	● Mengatur pergerakan *mecanum wheel mobile robo*t berdasarkan jalur perencanaan lintasan

Pada Gambar 3.2 terdapat dua subsistem utama, yaitu subsistem navigasi dan pergerakan. Subsistem navigasi pada produk ini bertujuan untuk memetakan lingkungan sekitar robot sehingga dapat menghasilkan visualisasi arena dalam bentuk *real time* dan dapat mendeteksi adanya halangan dalam jarak dekat. Subsistem pergerakan bertanggung jawab dalam mengendalikan pergerakan *mecanum wheel mobile robot* berdasarkan jalur yang telah direncanakan sebelumnya.

3.1.3 Diagram Subsistem Navigasi

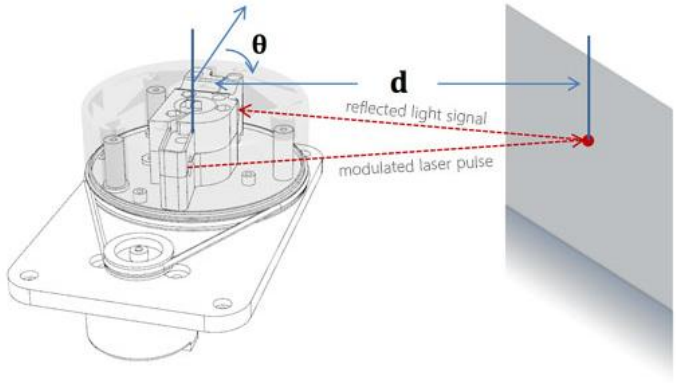


Gambar 3.3 - DFD Level 2 Subsistem Navigasi

Tabel 3.3 - Penjelasan DFD Level 2 Subsistem Navigasi

Parameter	Keterangan
Input	● Data jarak objek di sekitar robot dengan sensor ultrasonik dan LIDAR
Output	● Mendeteksi keberadaan objek di sekitar robot ● Menampilkan visualisasi 2D peta berdasarkan sensor LIDAR
Fungsi	● Membantu menentukan arah pergerakan robot dalam subsistem pergerakan ● Membantu robot menentukan jalur yang akan diambil

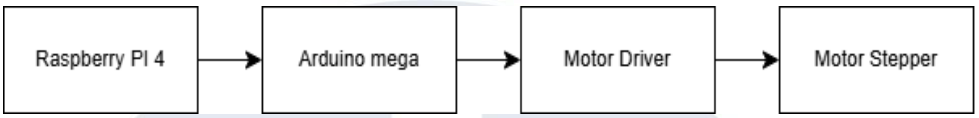
Pada subsistem ini terdapat data dari sensor LIDAR dan ultrasonik dengan pusat pengolahan datanya di Raspberry Pi 4. Sesuai dengan Tabel 3.3, Sensor LIDAR berfungsi untuk mendeteksi jarak dengan lingkungan sekitarnya yang kemudian datanya diolah sehingga dapat melakukan pendeteksian dan juga membantu robot dalam melakukan perencanaan lintasan A*. Cara kerja sensor LIDAR yaitu bekerja dengan metode *laser triangulation* yang memancarkan sinar laser ke lingkungan sekitarnya. Ketika sinar laser yang dipancarkan mengenai sebuah objek, sebagian dari cahaya tersebut akan dipantulkan kembali ke sensor. Dengan menghitung sudut rotasi (θ) yang terdeteksi oleh sistem *encoder* dan jarak yang didapatkan, serta posisi pantulan cahaya pada sensor. Prinsip dari sensor ini dapat dilihat lebih jelas pada Gambar 3.4



Gambar 3.4 - Prinsip Kerja Sensor LIDAR
Sumber : https://bucket-download.slamtec.com/d1e428e7efbdcd65a8ea111061794fb8d4ccd3a0/LD108_SLAMTEC_rplidar_datasheet_A1M8_v3.0_en.pdf

Sensor ultrasonik berfungsi untuk mengetahui ada tidaknya halangan di sekitar *mecanum wheel mobile robot* dalam jangkauan jarak yang cukup dekat yang tidak dapat dijangkau oleh sensor LIDAR. Sensor ultrasonik juga membantu mendeteksi objek yang tidak dapat dijangkau sensor LIDAR pada sisi bawah robot dikarenakan sensor LIDAR berada di atas robot.

3.1.4 Diagram Subsistem Pergerakan



Gambar 3.5 - DFD Level 2 Subsistem Pergerakan

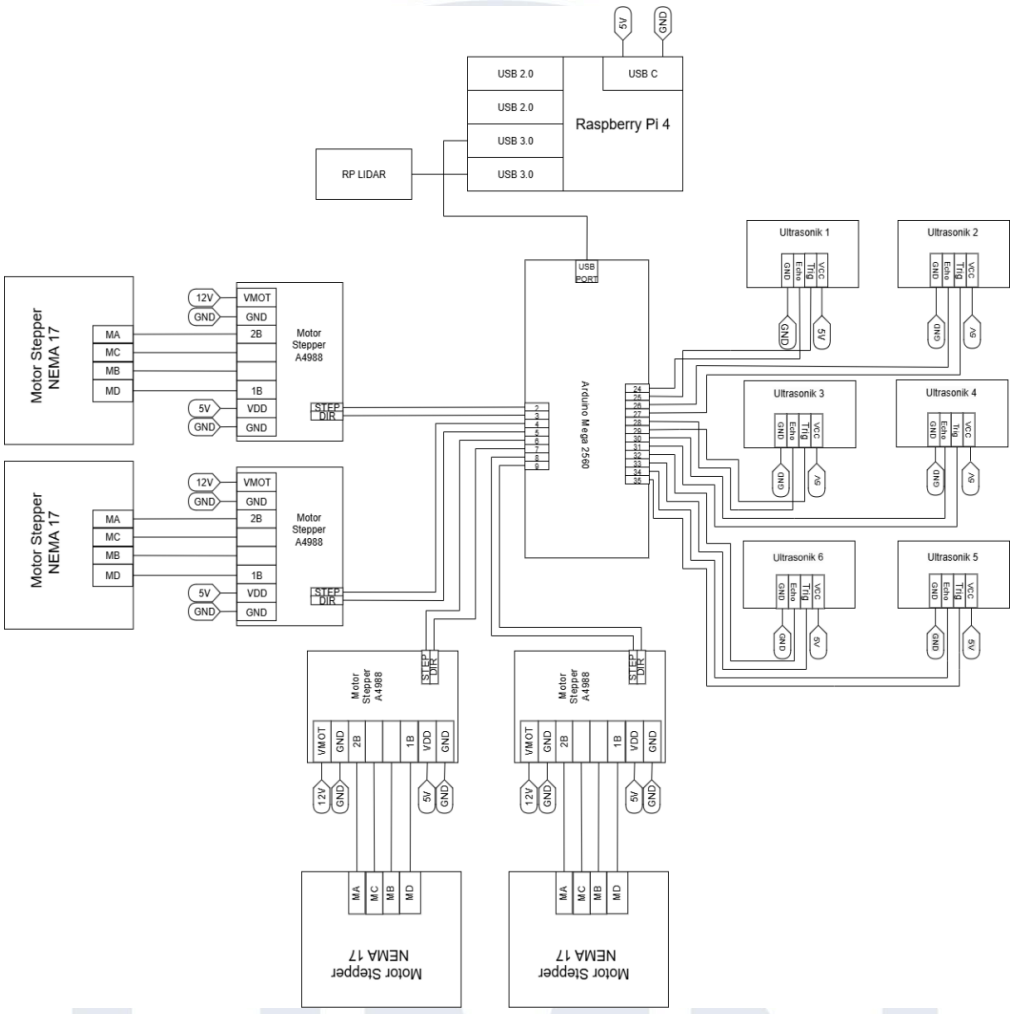
Tabel 3.4 - Penjelasan DFD Level 2 Subsistem Pergerakan

Parameter	Keterangan
Input	● Data dari subsistem navigasi
Output	● Pergerakan dari aktuator *motor stepper* dan roda mekanum
Fungsi	● Mampu mengatur perputaran keempat *motor stepper* untuk bergerak lurus, kanan, kiri, diagonal, dan berhenti

Pada subsistem ini terdapat empat *motor stepper* beserta *motor driver*. Kontrol perintah pergerakan dilakukan oleh mikrokontroler Arduino Mega. Raspberry Pi 4 memberikan perintah pada mikrokontroler Arduino Mega yang kemudian perintah itu dibaca sehingga dapat bergerak lurus, kanan, kiri, diagonal, dan berhenti.

3.1.5 Wiring Diagram Keseluruhan Sistem

Mecanum wheel mobile robot menggunakan sistem pengkabelan yang tidak menggunakan papan PCB, melainkan mayoritas koneksi pin pada *mecanum wheel mobile robot* terhubung secara langsung pada PCB buatan yang langsung menghubungkan antar kabel *power* 5V dan pin GND pada masing-masing sensor ultrasonik dan *chip motor driver* A4988. wiring diagram yang digunakan pada *mecanum wheel mobile robot* dapat dilihat pada Gambar 3.5.



Gambar 3.6 - *Wiring Diagram Mecanum Wheel Mobile Robot*

Pada sistem ini digunakan dua buah baterai *Lithium Polymer* (LiPo) terpisah, masing-masing dengan fungsi yang berbeda sesuai kebutuhan beban komponen. Baterai LiPo pertama berjenis 2 sel tegangan total 7,4 V dengan kapasitas 5200 mAh digunakan untuk menyuplai komponen, yaitu Raspberry Pi 4, Arduino Mega, sensor ultrasonik HC-SR04, dan LIDAR. Daya dari baterai ini diturunkan menggunakan modul step-down (*buck converter*) menjadi tegangan stabil 5 V sebelum diberikan ke Raspberry Pi 4 dan sensor lainnya. Sedangkan baterai LiPo kedua, yaitu 3 sel tegangan total 11,1 V dengan kapasitas 5400 mAh, digunakan khusus untuk mensuplai *motor stepper*

NEMA17 melalui *motor driver* A4988. Daya dari baterai ini dinaikin menggunakan modul step-up(*boost converter*) menjadi 12 V sebelum diberikan ke *motor stepper*.

Alasan penggunaan dua baterai ini untuk menjaga kestabilan suplai tegangan. *Motor stepper* memiliki karakteristik beban dinamis yang dapat menyebabkan lonjakan arus mendadak ketika mulai bergerak. Lonjakan arus ini menimbulkan *drop voltage* pada tegangan keluaran baterai, sehingga menghasilkan penurunan tegangan secara cepat. Penurunan ini terjadi di *input buck converter* yang menyuplai Raspberry Pi 4, sehingga menyebabkan *output buck converter* juga naik dan turun sesaat (*ripple voltage*). Karena Raspberry Pi 4 dan LIDAR memerlukan tegangan 5 V yang sangat stabil toleran terhadap tegangan yang fluktuasi kecil, *ripple voltage* tersebut dapat memicu terjadinya *undervoltage warning* pada Raspberry pi 4 sehingga menyebabkan *reboot* atau mati mendadak pada Raspberry pi 4 dan LIDAR. Dengan memisahkan catu daya motor dan *logic* menggunakan dua baterai, *ripple* dan penurunan tegangan akibat beban motor tidak lagi memengaruhi suplai ke Raspberry pi 4 dan sensor. Selain itu, dua baterai memperpanjang waktu operasi robot karena distribusi menjadi lebih efisien. Sehingga *motor stepper* dan *power logic* memiliki durasi lebih tahan lama dan efisien

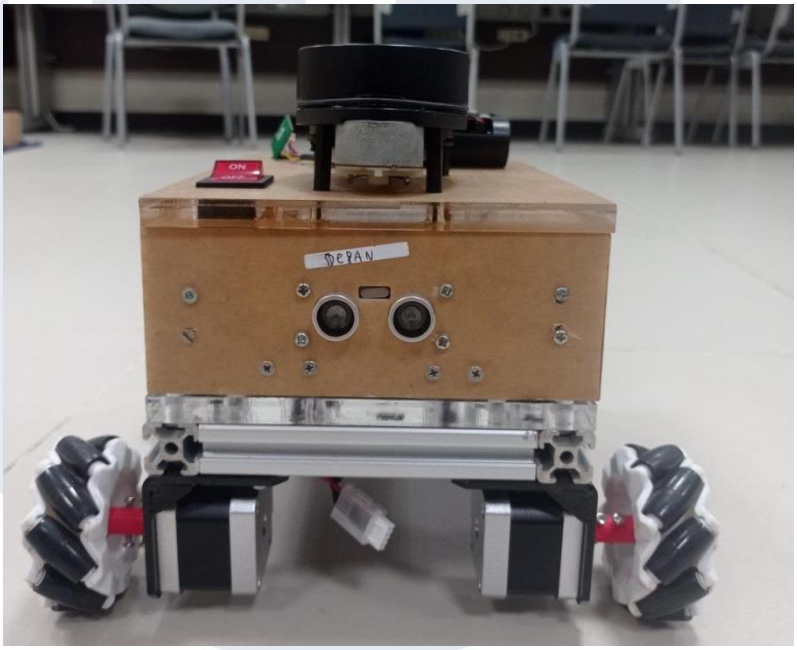
3.2 Implementasi Sistem

3.2.1 Hasil Implementasi

1. Hasil Implementasi Desain Fisik

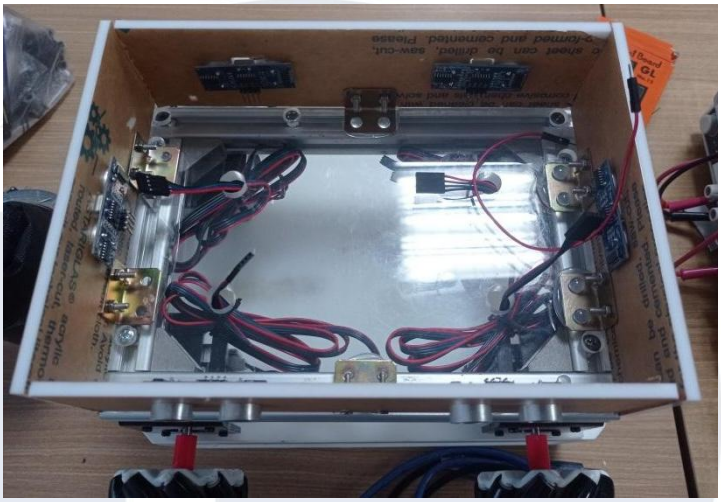
Mecanum wheel mobile robot ini terdiri dari dua bagian utama. Pertama bagian tengah yang merupakan badan utama, dan bagian bawah yang merupakan sasis. Badan utama ini berisi komponen-komponen yang cukup besar berupa 4 buah *motor driver*, 2 buah *buck converter* untuk 12V dan 5V, 6 buah sensor ultrasonik, 2 buah baterai salah satunya terletak di dalam robot dan 1 nya lagi terletak di sasis robot, dan Arduino Mega yang terhubung dengan PCB *motor driver*. Sasis *mecanum wheel mobile robot* hanya dipasangkan

komponen *motor stepper* yang sudah dipasangkan ke roda mekanum untuk dengan menggunakan *bracket* besi. Tampak depan dari keseluruhan *mecanum wheel mobile robot* dapat dilihat pada Gambar 3.7.



Gambar 3.7 - Tampak Depan *Mecanum Wheel Mobile Robot*

Bagian badan utama dari *mecanum wheel mobile robot* semuanya terbuat dari akrilik pada permukaan atas, bawah, maupun dinding nya. Pada bagian permukaan bawah dan atas menggunakan akrilik dengan ketebalan 10 mm, sedangkan untuk dinding hanyalah menggunakan akrilik dengan ketebalan 5 mm. Badan utama dari *mecanum wheel mobile robot* ini berbentuk persegi panjang berdimensi 23 x 16 cm. Permukaan bawah pada badan utama ini dihubungkan ke sasis dengan menggunakan *T-Nut* yang merupakan mur khusus untuk aluminium *profile*. Dinding *mecanum wheel mobile robot* terdiri dari 4 buah persegi panjang yang memiliki tinggi sebesar 6 cm. Bagian dinding ini dipasangkan ke samping robot dengan permukaan bawah menggunakan siku yang terbuat dari besi seperti pada Gambar 3.8.



Gambar 3.8 - Bagian Dalam Pada *Mecanum Wheel Mobile Robot*

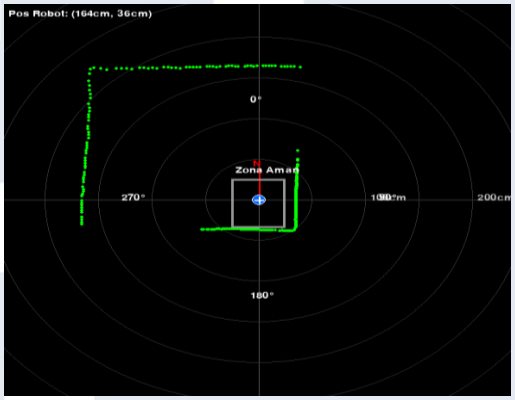
Pada Gambar 3.8 di bawah badan utama terdapat sasis yang terbuat dari bahan aluminium *profile* dengan tipe 2020 sehingga memiliki panjang dan lebar 20 mm. Aluminium *profile* ini terdiri dari 4 buah dengan 2 macam panjang, 23 cm 2 buah, dan 12 cm 2 buah yang dihubungkan dengan siku *profile* 2020 serta menggunakan *T-Nut*, dan sekrup ukuran M4.

2. Hasil Implementasi Subsistem Navigasi

Subsistem navigasi merupakan subsistem yang memiliki tanggung jawab dalam mendeteksi lingkungan sekitar robot. Subsistem ini memiliki fungsi untuk memberi robot kemampuan untuk memetakan lingkungan kerja serta dapat beradaptasi pada lingkungan yang dinamis karena dapat mengetahui adanya perubahan yang terjadi di lingkungan sekitar robot.

Implementasi awal pada subsistem ini adalah pembuatan dari *wiring diagram* yang dapat mempermudah pemasangan masing-masing komponen. Wiring diagram pada subsistem navigasi ini dapat dilihat pada Gambar 3.6 sebelumnya. Pada robot, sensor LIDAR diletakan di atas badan utama robot yang dihubungkan dengan menggunakan *fasteners* dan *spacers*. Sensor LIDAR dihubungkan dengan menggunakan *micro* USB ke Raspberry Pi 4. Fungsi sensor LIDAR diletakan di sisi atas agar sensor dapat dimanfaatkan dengan maksimal karena dapat tetap melakukan pembacaan secara 360°. Dalam

implementasikan ke robot, digunakan laptop/pc dari pengguna untuk menerima data LIDAR yang dikirimkan dari Raspberry Pi 4 yang nantinya divisualisasikan menggunakan library *Pygame*.



Gambar 3.9 - Visualisasi arena robot menggunakan sensor LIDAR

Pada Gambar 3.9 pada proses pengambilan data menggunakan sensor LIDAR yang divisualisasikan dengan library *Pygame*. hasil pemindaian dari sensor LiDAR yang dipasang di robot. Titik-titik hijau itu menunjukkan jarak antara robot dengan benda-benda di sekitarnya. Secara teori, kalau robot bergerak mengelilingi tembok atau objek yang bentuknya kotak, kita berharap hasil gambarnya juga terlihat seperti garis lurus atau sudut yang rapi. Namun, pada kenyataannya, titik-titik ini tidak selalu tampak sejajar atau benar-benar membentuk pola kotak yang sempurna. Hal ini terjadi karena data yang diambil oleh LIDAR aslinya disimpan dalam bentuk koordinat polar, yaitu pasangan antara sudut dan jarak, misalnya (sudut, jarak). Saat data ini diubah ke bentuk peta dengan sumbu x dan y, muncul sedikit ketidaktepatan karena terbatasnya ketelitian sudut dan jarak yang bisa diukur sensor. Selain itu, permukaan benda yang tidak rata atau miring juga bisa membuat pantulan laser sedikit meleset. Akibatnya, garis yang sebenarnya lurus jadi terlihat seperti kumpulan titik-titik yang sedikit bergelombang. Serta pada dinding juga terlihat adanya bagian dinding ruangan yang tidak terbaca. Ini terjadi karena saat robot berada di pojok ruangan, sudut datang sinar laser ke dinding yang dekat menjadi sangat miring, sehingga sinar pantul menjauh dari sensor dan tidak kembali ke penerima. Akibatnya, data jarak pada area tersebut tidak diperoleh dan visualisasi di *Pygame* menampilkan *gap* atau bagian

yang hilang. Fenomena ini merupakan karakteristik khas sensor berbasis triangulasi yaitu metode menghitung jarak dengan menggunakan sudut dan posisi pantulan cahaya (laser) untuk menghitung seberapa jauh objek itu berada, seperti LIDAR, ketika memindai bidang yang terletak sangat dekat dan sejajar dengan arah laser. Serta Pembacaan jarak jauh menghasilkan titik yang lebih sedikit karena LIDAR bekerja dengan resolusi sudut tetap, sehingga jarak antar titik pada dinding jauh menjadi semakin besar. Selain itu, pada jarak jauh sinyal pantul dari dinding lebih lemah sehingga sebagian data terfilter oleh *software*, menyebabkan titik yang terlihat pada visualisasi menjadi lebih sedikit.

Sensor kedua yaitu ultrasonik yang akan diimplementasikan pada dinding badan utama berjumlah 6 buah yang dipasang mengelilingi badan robot sehingga mampu mengurangi titik buta dari robot dan mendeteksi ke segala arah. Seluruh pin VCC dan GND sensor akan dihubungkan secara paralel pada bagian depan dan belakang PCB. Untuk pin *trig* dan *echo* sensor ultrasonik akan dipasangkan pada masing-masing GPIO pada Arduino Mega. Semua pin tersebut menggunakan kabel AWG 22 yang ujung nya disambungkan dengan *dupont connector male* untuk *trig* dan *echo* ke pin Arduino Mega dan *dupont connector female* yang terhubung pada *dupont connector male* pada PCB untuk VCC dan GND. *Power* yang digunakan untuk menghidupkan sensor ini menggunakan baterai Lipo dengan tegangan yang disesuaikan ke 5V dengan menggunakan *buck converter*. Setelah selesai dirangkai rangkaiannya, semua sensor ultrasonik dipasangkan ke lubang sudah dibuat sebelumnya di bagian dinding badan utama dengan menggunakan sekrup, *washer*, dan juga mur.

Implementasi algoritma A* pada subsistem navigasi digunakan untuk melakukan perencanaan lintasan dari titik awal (*start point*) menuju titik tujuan (*end point*) secara optimal. Algoritma ini bekerja dengan prinsip pencarian jalur berdasarkan evaluasi biaya total dari setiap node(n) dengan satuannya yaitu langkah *grid* (*cell*), yang dihitung menggunakan persamaan berikut:

$$f(n) = g(n) + h(n) \tag{3.1}$$

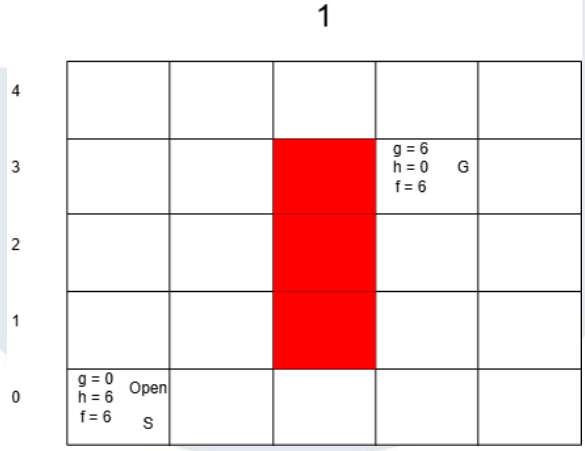
$f(n)$ = Total langkah *grid* (*cost*) dari titik awal menuju titik akhir pada node-n

$g(n)$ = Jarak yang sudah ditempuh dari titik awal menuju node-n

$h(n)$= perkiraan jarak pada node n ke titik tujuan

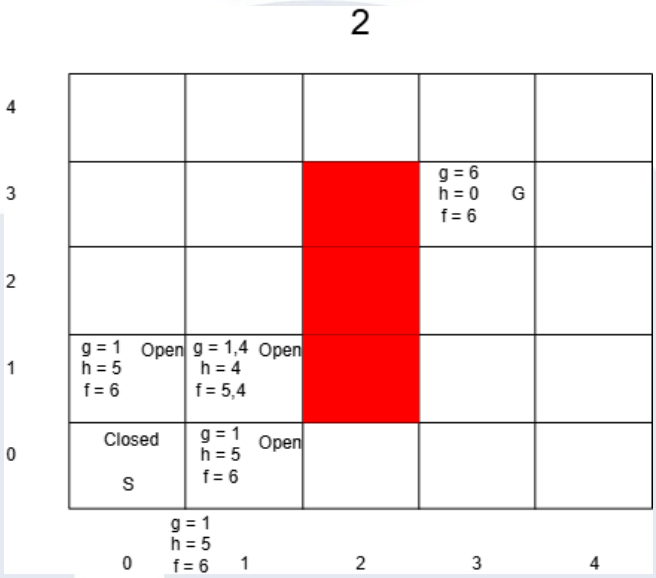
(n) = node merupakan titik yang sedang diperiksa oleh algoritma A*

Ilustrasi dari cara kerja algoritma A* dapat dilihat pada Gambar 3.9.



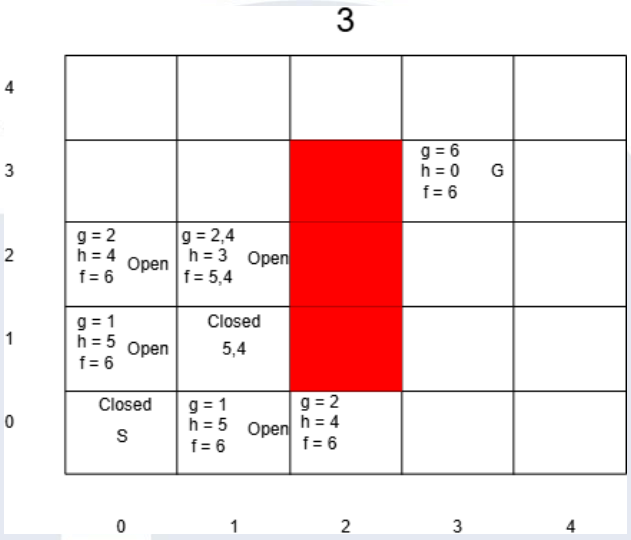
Gambar 3.10 - Ilustrasi Cara Kerja Algoritma A* dengan *Grid* 5 x 5

Pada Gambar 3.10 merupakan gambar cara kerja Algoritma A* dengan *grid* 5x5. Titik Awal dimulai pada node (0,0) dan titik akhirnya di node (3,3) dengan halangan nya diwarnai warna merah pada node (2,1) sampai node (2,3). Node awal yaitu (0,0) dimasukan ke *open list* (node yang akan dievaluasi) dengan nilai f_{cost} nya 6.



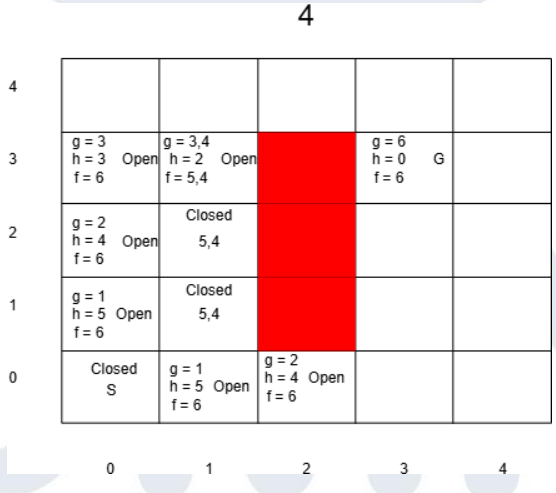
Gambar 3.11 - Iterasi ke-2 Eksplor Tetangganya

Pada Gambar 3.11 node (0,0) dimasukan ke *closed list* (node yang sudah dievaluasi) habis itu tetangganya akan juga di evaluasi sehingga akan dimasukan ke *open list*. Terlihat bahwa saat mengeksplor tetangganya $g(n)$ nya bernilai 1 untuk arah vertikal (atas dan bawah) dan untuk arah diagonal bernilai 1,4, itu terjadi karena pada saat ingin berjalan diagonal maka langkah yang ditempuh adalah 2 langkah, tapi 2 langkah itu mahal jika dibandingkan arah vertikal dan horizontal. Maka digunakan rumus pythagoras sehingga 2 langkah tersebut bisa diubah menjadi $\sqrt{2}$ = 1,4. Tetangganya node memiliki f_{cost}, yaitu node (0,1) memiliki f_{cost} 6, node (1,1) memiliki f_{cost} 5,4, dan node (1,0) memiliki f_{cost} 6.

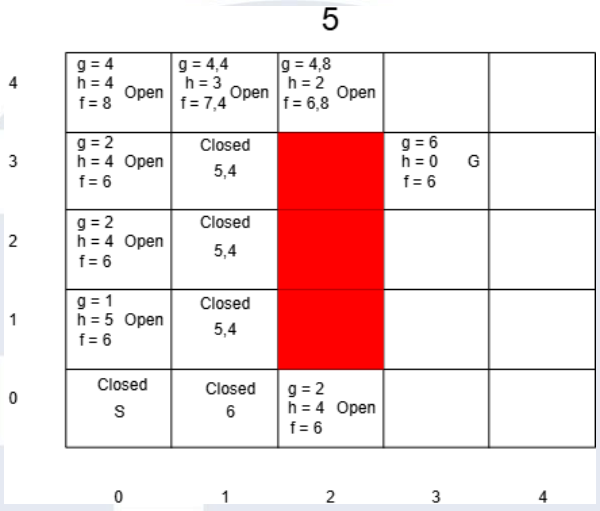


Gambar 3.12 - Iterasi ke-3 Eksplor Tetangganya

Pada Gambar 3.12 node yang memiliki f_{cost} paling kecil akan masuk kedalam *closed list* yaitu node (1,1) dengan f_{cost} 5,4. Setelah itu akan mevaluasi tetangga di atas dan di bawah yaitu node(0,2), (1,2), (2,0).

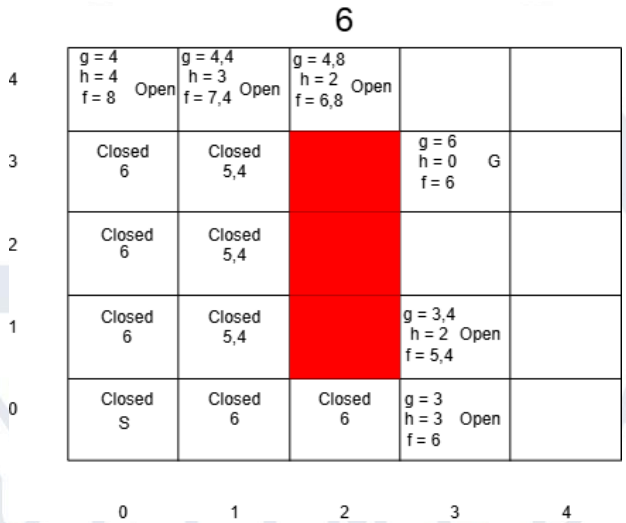


Gambar 3.13 - Iterasi ke-4 Eksplor Tetangganya



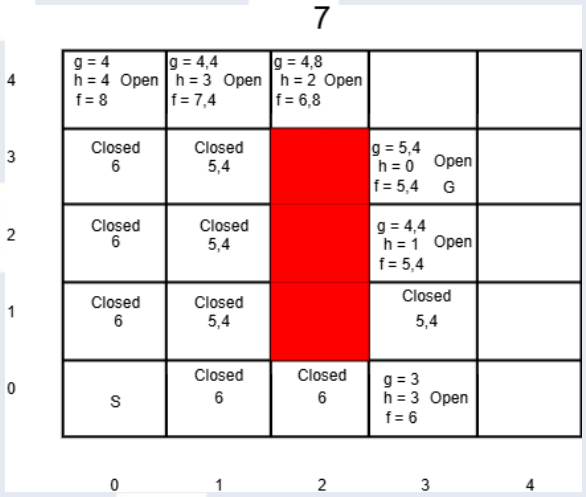
Gambar 3.14 - Iterasi ke-5 Eksplor Tetangganya

Pada Gambar 3.14 node yang memiliki f_{cost} paling rendah akan dimasukan kedalam *closed list* yaitu (1,3) dengan f_{cost} 5.4 setelah itu akan dievaluasi lagi tetangga-tetangganya sehingga akan dimasukan ke dalam *open list*. Didapatkan pada saat mengevaluasi terdapat f_{cost} yang mahal pada node (0,4) senilai 8, node (1,4) senilai 7,4, node (2,4) senilai 6,8, oleh karena itu algoritma akan memprioritas dengan f_{cost} paling kecil sehingga nantinya *parents* dan *child* nya akan dimasukan ke *closed list* yaitu node (0,1) sampai node (0,3).



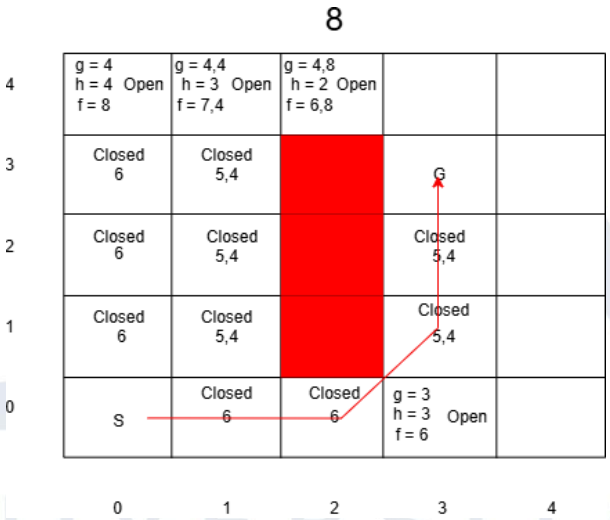
Gambar 3.15 - Iterasi ke-6 Eksplor Tetangganya

Pada Gambar 3.15 terlihat bahwa node (2,0) dimasukan kedalam *closed list* dan akan mengeksplor tetangga nya yaitu node (3,0) dan node (3,1), didapatkan f_{cost} nya yaitu 6 dan 5,4.



Gambar 3.16 - Iterasi ke-7 Eksplor Tetangganya

Pada Gambar 3.16 terlihat bahwa node (3,1) dimasukan ke *closed list* dan selanjutnya akan mengevaluasi node (3,2) yang berisikan f_{cost} 5,4.



Gambar 3.17 - Iterasi ke-8 Eksplor Tetangganya

Pada Gambar 3.17 terlihat bahwa node (3,2) dimasukan ke *closed list* dan akhirnya, jalur menuju titik tujuan ditemukan yaitu (0,0) → (1,0) → (2,0) → (3,1) → (3,2) → (3,3).

Dalam algoritma A* terdapat dua heuristik yang umum digunakan yaitu:

1. *Manhattan Distance*, metode ini memperhitungan jarak yang hanya memperbolehkan pergerakan ke arah vertikal dan horizontal di sepanjang *grid*. Konsep ini sesuai dengan pola jalan kota yang berbentuk kotak-kotak, di mana jalur terpendek dihitung berdasarkan jumlah langkah ke samping dan ke atas atau bawah. Pada perhitungan *Manhattan distance* digunakan operasi nilai mutlak untuk memastikan hasil jarak tetap positif, karena meskipun pergerakan menuju koordinat dengan nilai lebih kecil (arah negatif), jarak tersebut tetap dihitung sebagai panjang lintasan yang harus ditempuh.

$$h(n) = | x_{goal} - x_n | + | y_{goal} - y_n | \quad (3.2)$$

$h(n)$ = Nilai heuristik di node n (perkiraan jarak node n ke node tujuan)
x_{goal} = koordinat x dari node tujuan
y_{goal} = koordinat Y dari node tujuan
x_n = koordinat X dari node saat ini
y_n = koordinat Y dari node saat ini

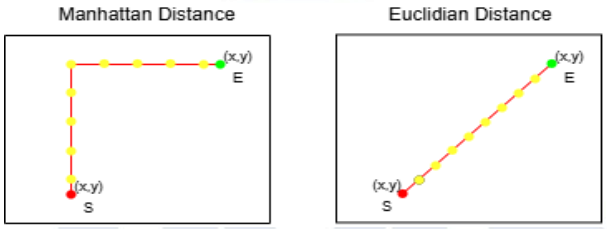
2. *Euclidean Distance*, heuristik ini digunakan untuk memperkirakan jarak sebagai garis lurus antara dua titik, sehingga memungkinkan pergerakan ke segala arah, termasuk diagonal. Perhitungan ini didasarkan pada teorema Pythagoras, dengan cara mengkuadratkan selisih koordinat x dan y untuk menghilangkan efek tanda negatif, lalu mengambil akar kuadrat dari hasil jumlah keduanya agar kembali ke satuan panjang yang sesuai. Metode ini menghasilkan nilai jarak untuk sistem atau robot yang memiliki kebebasan bergerak ke arah mana saja. *Euclidean distance* sering dipakai ketika pergerakan tidak dibatasi hanya pada jalur kotak-kotak, melainkan dapat langsung menuju tujuan secara diagonal, sehingga lebih efisien dalam kondisi tertentu.

$$h(n) = \sqrt{(x_{goal} - x_n)^2 + (y_{goal} - y_n)^2} \quad (3.3)$$

$h(n)$ = Nilai heuristik di node n (perkiraan jarak node n ke node tujuan)
x_{goal} = koordinat x dari node tujuan

y_{goal} = koordinat Y dari node tujuan
x_n = koordinat X dari node saat ini
y_n = koordinat Y dari node saat ini

Pada kedua heuristik seperti *Manhattan distance* dan *Euclidean distance* selalu menghitung selisih koordinat antara node saat ini dan goal. Nilai selisih ini bisa bernilai positif maupun negatif, tergantung posisi goal dibandingkan node tersebut. Misalnya jika goal berada di kanan node, selisihnya akan positif, tetapi jika goal ada di kiri, hasilnya negatif. Keberadaan angka negatif penting karena menandakan arah antara node dan goal. Namun pada saat menghitung jarak, yang kita perlukan hanya seberapa jauh, tanpa memperdulikan arah. Oleh karena itu dalam *manhattan distance*, kita menggunakan nilai mutlak agar hasil akhirnya selalu positif. Sedangkan dalam *euclidean distance*, kita mengkuadratkan selisih tersebut, lalu diakhiri dengan akar kuadrat agar memastikan hasil akhirnya positif. Implementasi dari kedua perhitungan heuristik tersebut dapat dilihat. Berikut merupakan Gambar 3.18 yang menunjukan hasil perencanaan lintasan A* pada kedua heuristik.



Gambar 3.18 - Ilustrasi perencanaan Lintasan A* dengan menggunakan heuristik *Manhattan Distance* dan *Euclidean Distance*

Berdasarkan Gambar 3.18, titik warna merah menggambarkan titik mulai (*start point*), sedangkan garis merah menunjukkan jalur yang dapat dilalui robot. Titik-titik kuning pada garis tersebut merepresentasikan koordinat yang harus dilalui atau dicapai oleh robot, sementara titik warna hijau merupakan titik akhir (*end point*). Ilustrasi hasil perencanaan dari kedua heuristik tersebut hanyalah contoh visualisasi sederhana. Pada *Manhattan distance*, heuristik menghitung jumlah total langkah horizontal dan vertikal dari posisi awal ke posisi tujuan, tanpa melebih-lebihkan jarak ke goal. Nilai heuristik tetap

sama meskipun pola pergerakan berbeda, karena yang dihitung hanyalah total selisih koordinat X dan Y, bukan jalur yang akan ditempuh robot. Sedangkan pada *Euclidean distance*, jarak heuristik dihitung sebagai garis lurus antara titik awal dan tujuan, sehingga biasanya menghasilkan estimasi jarak yang lebih pendek. Pemilihan pola pada kedua heuristik ditentukan oleh perbedaan nilai g(n), yaitu cost nyata yang dihitung dari titik awal menuju node yang sedang dievaluasi, serta oleh urutan pemrosesan node di dalam open list yang digunakan algoritma A*. Perbedaan g(n) muncul karena jalur yang lebih lurus cenderung memiliki jumlah langkah lebih sedikit atau cost yang lebih rendah dibandingkan jalur zig-zag atau jalur memutar, meskipun total nilai heuristic (h(n)) yang menggambarkan estimasi jarak ke goal tetap sama. Selain itu, algoritma A* memproses node di open list berdasarkan nilai f(n) yang paling kecil, dan apabila terdapat beberapa node dengan nilai f(n) yang sama, maka urutan masuknya node ke open list atau kebijakan prioritas dalam implementasi akan menentukan node mana yang dievaluasi lebih dulu.

Implementasi dari kedua perhitungan heuristik tersebut dapat dilihat pada Gambar 3.19 dan Gambar 3.20

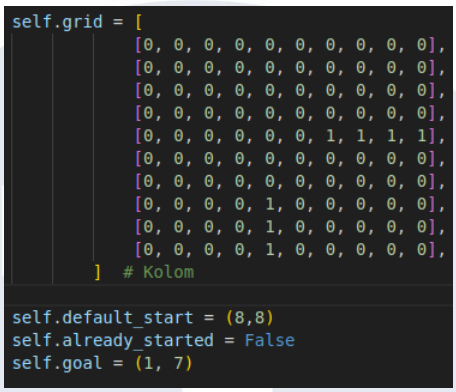
```
#Fungsi heuristic Manhattan
def heuristic(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])
```

Gambar 3.19 - Perhitungan Heuristik *Manhattan Distance*

```
#fungsi heuristic euclidean
def heuristic(a, b):
    return hypot(a[0] - b[0], a[1] - b[1])
```

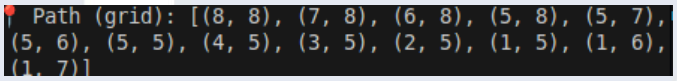
Gambar 3.20 - Perhitungan Heuristik *Euclidean Distance*

Sistem perencanaan lintasan dibuat berdasarkan peta *grid* yang diimplementasikan kedalam bentuk matriks 2 dimensi (2D). Berikut merupakan matriks 2D yang merepresentasikan peta *grid*.

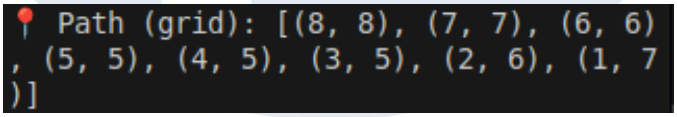


Gambar 3.21 - Matriks 2D yang mempresentasikan peta *grid*

Hasil dari perencanaan lintasan A* dengan peta *grid* akan menghasilkan koordinat *grid* yang dapat dilihat pada Gambar 3.22 dan Gambar 3.23



Gambar 3.22 - *Output* (Koordinat *Grid*) *Manhattan Distance*



Gambar 3.23 - *Output* (Koordinat *Grid*) *Euclidean Distance*

Kemudian hasil koordinat *grid* akan di konversi menjadi perintah pergerakan dalam format string. Dalam mengubah *output* koordinat *grid* ke perintah pergerakan dalam format string digunakan fungsi berikut:

```python
def convert_path_to_commands(self, path, grid_size_cm=20):
    commands = []
    if len(path) < 2:
        return commands

    dx = path[1][0] - path[0][0]
    dy = path[1][1] - path[0][1]
    direction = self.get_direction(dx, dy)
    count = 1

    for i in range(2, len(path)):
        dx_new = path[i][0] - path[i-1][0]
        dy_new = path[i][1] - path[i-1][1]
        new_dir = self.get_direction(dx_new, dy_new)

        if new_dir == direction:
            count += 1
        else:
            commands.append(f"{direction}_{count * grid_size_cm}")
            direction = new_dir
            count = 1

    commands.append(f"{direction}_{count * grid_size_cm}")
    return commands
```

Gambar 3.24 - Fungsi Untuk Merubah *Path* ke Pergerakan Robot

Fungsi di atas untuk mengubah *path* (hasil algoritma A*, berupa list koordinat *grid*) menjadi list perintah gerak. Caranya adalah kita cek perbedaan antara dua titik berurutan setelah itu cari arahnya kalau arahnya sama, tambahkan count. Kalau arah nya berubah, buat perintah baru dan reset count. Cara tau arahnya yaitu dengan fungsi berikut:

```
def get_direction(self, dx, dy):
    if dx == 1 and dy == 0:
        return "bawah"
    elif dx == -1 and dy == 0:
        return "maju"
    elif dx == 0 and dy == 1:
        return "kanan"
    elif dx == 0 and dy == -1:
        return "kiri"
    elif dx == 1 and dy == 1:
        return "diagonalkananbelakang"
    elif dx == 1 and dy == -1:
        return "diagonalkiribelakang"
    elif dx == -1 and dy == 1:
        return "diagonalkanandepan"
    elif dx == -1 and dy == -1:
        return "diagonalkiridepan"
    else:
        return "diam"
```

Gambar 3.25 - Fungsi Untuk Mengetahui Arah Pergerakan Robot

Pada fungsi get_direction yaitu mengubah perbedaan posisi pada grid (dx, dy) menjadi perintah arah / string arah yang dapat dipahami oleh mikrokontroller Arduino Mega. Digunakan dx yaitu untuk mengetahui perubahan di sumbu baris (*row*), jika naik keatas(maju) maka baris akan berkurang (-1) dan sebaliknya dan dy digunakan untuk mengetahui perubahan di sumbu kolom (*col*), jika ke kanan maka kolom akan bertambah (+1) begitu juga sebaliknya. Untuk mengetahui arah nya kemana berikut contoh penggunaan fungsi tersebut.

Contoh *path* dari (8,8) ke (7,8):

dx = 7-8 = -1, dy = 8-8 = 0

dx = -1, dy = 0 → Maju

Contoh *path* dari (7,8) ke (6,8):

dx = 6-7 = -1, dy = 8-8 = 0

dx = -1, dy = 0 → maju

Contoh *path* dari (6,8) ke (5,8):

dx = 5-6 = -1, dy = 8-8 = 0

dx = -1, dy = 0 → maju

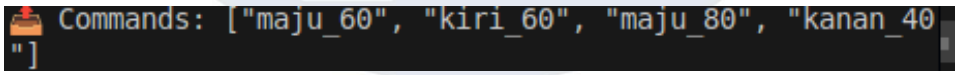
dari ketiga *path* tersebut yang sudah di konversi menjadi arah, didapatkan 3 langkah.Karena 1 koordinat atau *grid* 20 cm maka 3 langkah tersebut di kali dengan 20 yaitu hasilnya 60. Selanjutnya contoh pergerakan jika terdapat perubahan arah.

Contoh *path* dari (5,8) ke (5,7):

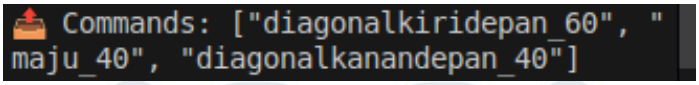
dx = 5-5 = 0, dy = 7-8 = -1

dx = 0, dy = -1 → kiri

Setelah terdapat perubahan arah maka dibuat perintah baru sehingga mereset count nya menjadi 0 karena sebelumnya count nya 3 yaitu 3 langkah. Dan seterusnya sehingga mendapatkan arah pergerakan robot, begitu juga yang menggunakan *euclidean distance*. Arah pergerakan robot menggunakan *path manhattan distance* dan *euclidean distance* sebelumnya dapat dilihat pada Gambar 3.18

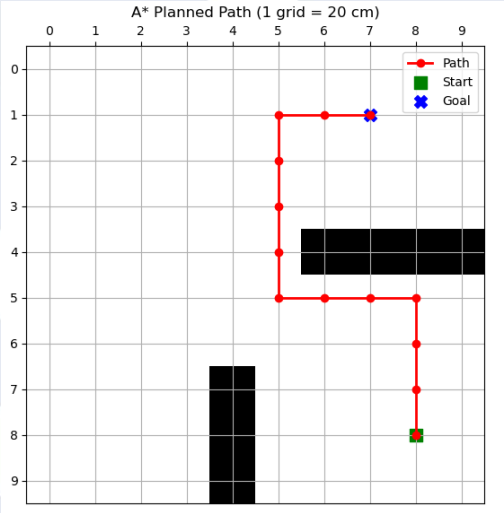


Gambar 3.26 - Arah Pergerakan Robot Menggunakan *Manhattan Distance*

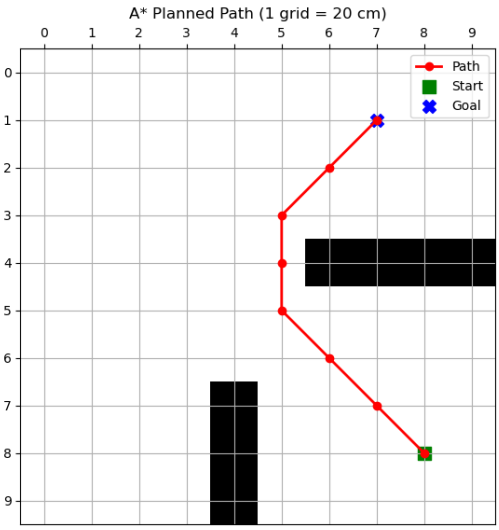


Gambar 3.27 - Arah Pergerakan Robot Menggunakan *Euclidean Distance*

Berikut merupakan Gambar 3.28 dan Gambar 3.29 yang menunjukkan visualisasi hasil *path planning* dari kedua heuristik.



Gambar 3.28 - Visualisasi Algoritma *path planning* A* dengan Menggunakan Heuristik *Manhattan Distance*



Gambar 3.29 - Visualisasi Algoritma *path planning* A* dengan Menggunakan Heuristik *Euclidean Distance*

3. Hasil Implementasi Subsistem Pergerakan

Subsistem pergerakan adalah subsistem yang memiliki peranan penting dalam pergerakan dari sistem produk. Subsistem pergerakan ini termasuk dalam subsistem yang membutuhkan data dari subsistem lainnya. Subsistem ini memerlukan data-data yang didapatkan dari subsistem navigasi. Subsistem pergerakan akan dikendalikan oleh mikrokontroler Arduino Mega dan subsistem ini akan berkomunikasi dua arah atau menerima dan mengirimkan data dari dan ke mikrokomputer Raspberry Pi 4 dengan menggunakan komunikasi USB

Implementasi awal pada subsistem ini adalah kurang lebih sama seperti subsistem navigasi yaitu membuat *wiring diagram* terlebih dahulu yang dapat dilihat pada Gambar 3.5 Subsistem pergerakan ini menggunakan beberapa komponen yaitu satu Arduino Mega, empat buah *motor driver* A4988, 4 *motor stepper*, 1 *buck converter* dengan tegangan keluarannya 5V untuk *chip* A4988, dan *boost converter* dengan tegangan keluarannya 12V untuk *motor stepper*. *Motor stepper* dipasangkan pada rangka bawah robot dengan menggunakan *bracket* besi dengan *fasteners* dan *T-Nut* M3 dan roda mekanum dipasangkan langsung ke besi as pada *motor stepper*

3.2.2 Hambatan dan Solusi Implementasi

a. Hambatan dan Solusi Desain Fisik

Dalam proses implementasi fisik robot, ditemukan beberapa kendala teknis. Salah satu permasalahan terjadi saat proses perakitan bagian-bagian akrilik, di mana tidak tersedia lubang untuk pemasangan *fastener*. Untuk mengatasi hal tersebut, solusi yang diterapkan adalah melakukan pengeboran secara manual menggunakan mata bor yang sesuai dengan ukuran *fastener* yang digunakan. Selain itu, pada bagian badan utama robot, permasalahan muncul terkait manajemen kabel. Jumlah kabel yang cukup banyak menyebabkan kondisi di dalam tubuh utama menjadi berantakan dan saling tumpang tindih. Hal ini tidak hanya mengganggu kerapihan kabel, tetapi juga menyebabkan beberapa kabel mudah terlepas dan menghambat pemasangan komponen lainnya karena keterbatasan ruang. Oleh karena itu, dilakukan penataan ulang kabel dengan mengarahkan

jalurnya ke sisi kiri dan kanan robot. Penataan ini dibantu menggunakan *cable ties* dan *cable organizer* agar kabel lebih rapi, tertata, dan tidak mengganggu komponen lain.

b. Hambatan dan Solusi Subsistem Navigasi

Dalam implementasikan subsistem navigasi, terdapat beberapa permasalahan yang ditemukan selama proses implementasinya. Pertama sensor LIDAR, kendala yang ditemukan adalah pada saat implementasikan pembacaan sensor LIDAR ke *Pygame* nya terdapat masalah sumbu x dan y nya kebalik sehingga perlu custom UI sendiri di *Pygame* nya sehingga pembacaan LIDAR dapat dimengerti oleh pengguna. Untuk sensor ultrasonik, kendala yang ditemukan adalah pada saat pemasangan sensor ke lubang pada dinding tidak dapat dipasangkan dengan benar dikarenakan pada saat pembelian pin header sudah terpasang pada sensor sehingga menonjol pinnya, sehingga perlu dilakukan desoldering pin header dan memasang pin tersebut ke belakang sedikit.