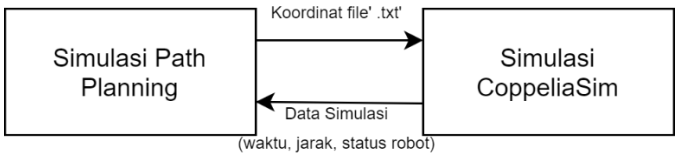
BAB III
METODOLOGI

3.1 Tinjauan Desain Sistem

3.1.1 Desain Sistem Keseluruhan

Secara keseluruhan, cara kerja simulasi yang dilakukan dan hubungan antara sistem program dengan robot pada simulator CoppeliaSim dapat dilihat pada Gambar 9. *DFD level 0* sistem simulasi.



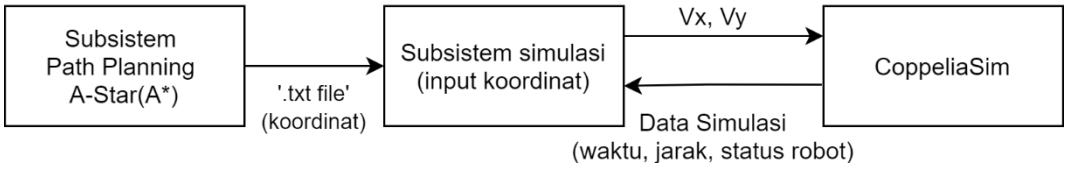
Gambar 3.1. *DFD level 0* sistem simulasi.

Tabel 3.1. Keterangan *DFD level 0* sistem simulasi.

Parameter	Keterangan
Input	● Koordinat *start point , intermediate point*, dan *end point.*
Output	● Waktu tempuh. ● Jarak tempuh. ● Posisi robot saat ini (koordinat). ● Total target dilewati.
Fungsi	● Mensimulasikan pergerakan *path planning A**.

Gambar 3.1 menunjukkan bahwa penelitian ini memerlukan penghubungan simulasi *path planning* dengan CoppeliaSim untuk menjalankan simulasi pergerakan robot menggunakan algoritma *path planning*. Sementara itu, Tabel 3.1 menjelaskan *input* yang digunakan dalam simulasi serta output yang dihasilkan berdasarkan *input* tersebut.

3.1.2 Desain Subsistem

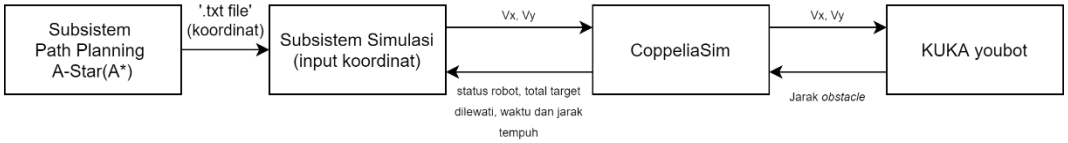


Gambar 3.2. *Data Flow Diagram level 1* subsistem simulasi.

Tabel 3.2. Penjelasan *DFD level 1* sistem simulasi.

Parameter	Keterangan
Input	● Koordinat *start point , intermediate point,* dan *end point.*
Output	● Kecepatan linear (*Vx*), Kecepatan transversal (*Vy*). ● Waktu tempuh. ● Jarak tempuh. ● Posisi robot saat ini (koordinat). ● Total target dilewati.
Fungsi	● Mensimulasikan jalur *path planning A*.* ● Menentukan jalur robot. ● Mengatur arah pergerakan robot. ● Menampilkan Status robot (Jarak tempuh, waktu tempuh robot, dan posisi robot saat ini).

Gambar 3.2 menunjukkan *data flow diagram level* 1 subsistem simulasi. Subsistem simulasi akan membaca hasil *path planning algoritma A** melalui '*file .txt* ' yang berisi x dan y. Tabel 3.2 merupakan penjelasan mengenai fungsi dari *input* dan *output* pada subsistem *path planning* hingga subsistem simulasi, sebagaimana ditunjukkan pada Gambar 3.2. Pada Gambar 3.2, subsistem *path planning A** menghasilkan koordinat lintasan hasil *path planning* menggunakan algoritma *A**, yang kemudian disimpan dalam *file '.txt'*. *File* tersebut selanjutnya dibaca oleh subsistem simulasi untuk menghasilkan perintah pergerakan robot pada lingkungan simulasi CoppeliaSim.



Gambar 3.3. *Data Flow Diagram Level 2* sistem simulasi.

Tabel 3.3. Penjelasan *DFD* level 2 sistem simulasi.

Parameter	Keterangan
Input	● Koordinat *start point* , *intermediate point*, dan *end point.*
Output	● Kecepatan linear (*Vx*), Kecepatan transversal (*Vy*). ● Waktu tempuh masing-masing *intermediate point*. ● Jarak tempuh masing-masing *intermediate point* . ● Koordinat *mobile robot* (*x, y*). ● Total target dilewati.
Fungsi	● Menentukan arah pergerakan robot. ● Menentukan jalur robot. ● Menampilkan status robot, jarak tempuh, waktu tempuh.

Gambar 3.3 menunjukkan *data flow diagram level 2* dari sistem simulasi. Tabel 3 menunjukan fungsi dari *input* yang diberikan dan *output* yang dihasilkan. Secara garis besar, alur proses pada tahap ini serupa dengan *Data Flow Diagram* (DFD) Level 1, namun terdapat sedikit penambahan pada struktur sistem, yaitu integrasi komponen KUKA youBot. KUKA youBot merupakan *mobile robot* yang dilengkapi dengan empat motor penggerak, masing-masing terhubung ke roda *mecanum*, serta dilengkapi dengan 12 buah sensor *proximity* yang terpasang pada setiap sisi robot untuk mendeteksi keberadaan objek di sekitarnya. *Input* yang diberikan kepada KUKA youBot berasal dari subsistem simulasi, yang mencakup perintah arah gerak robot, serta koordinat titik awal (*start point*) dan titik tujuan (*end point*). Sebagai respon balik terhadap *input*

tersebut, KUKA youBot mengirimkan data berupa hasil pembacaan jarak dari sensor *proximity* terhadap rintangan yang terdeteksi di lingkungan sekitarnya.

3.2 Implementasi Sistem

3.2.1 Hasil Implementasi

a. Implementasi subsistem *path planning A**.

Subsistem *path planning A** berfungsi untuk mensimulasikan jalur yang dihasilkan oleh algoritma *A**. Algoritma *A** merupakan metode *path planning* berbasis peta *grid* yang digunakan untuk menemukan lintasan optimal dengan memanfaatkan fungsi *heuristic* dalam mengevaluasi *cost* pada setiap *node* yang dikunjungi. Dalam konteks ini, *cost* mengacu pada total estimasi jarak dari titik awal menuju titik tujuan melalui simpul tertentu, yang dihitung berdasarkan kombinasi jarak aktual yang telah ditempuh *g(n)* dan estimasi jarak menuju tujuan *h(n)*. Berikut merupakan persamaan yang digunakan untuk melakukan pencarian jalur terpendek,

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

Manhattan distance

$$g(n) = |x_0 - x_n| + |y_0 - y_n| \tag{3.2}$$

$$h(n) = |x_{goal} - x_n| + |y_{goal} - y_n| \tag{3.3}$$

Euclidean distance

$$g(n) = \sqrt{(x_0 - x_n)^2 + (y_0 - y_n)^2} \tag{3.4}$$

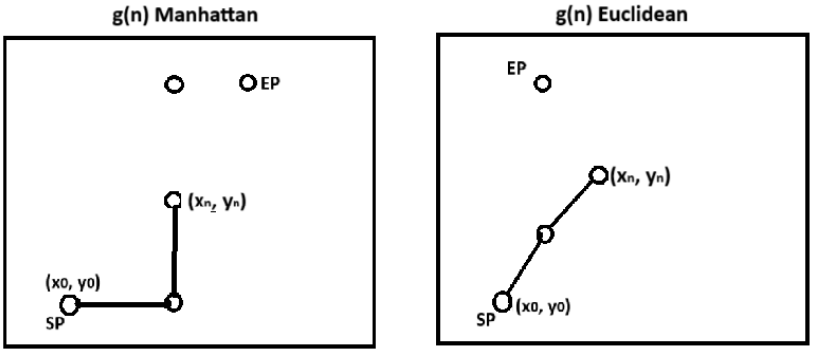
$$h(n) = \sqrt{(x_{goal} - x_n)^2 + (y_{goal} - y_n)^2} \tag{3.5}$$

f(n) = Total estimasi jarak dari titik awal menuju titik tujuan melalui *node n.*

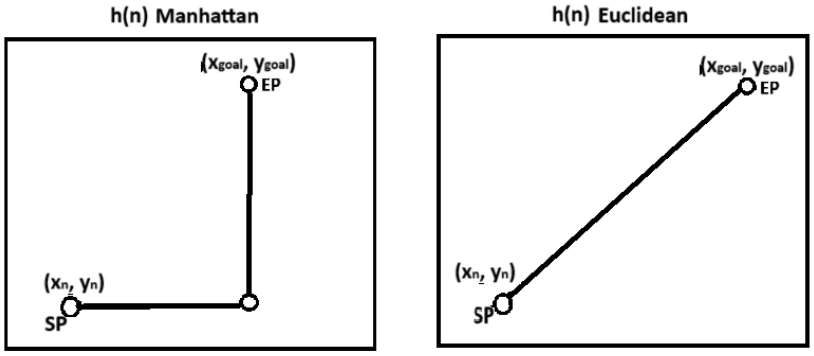
g(n) = Jarak aktual yang telah ditempuh dari titik awal menuju *node n,*

h(n) = estimasi jarak dari *node n* menuju titik tujuan, yang dihitung menggunakan fungsi heuristik.

Ilustrasi dari persamaan jarak aktual yang telah ditempuh *g(n)* dan estimasi jarak menuju tujuan *h(n)* dapat dilihat pada gambar 3.4 dan 3.5.



Gambar 3.4. Ilustrasi perhitungan jarak aktual yang telah ditempuh dari titik awal menuju *node n*



Gambar 3.5. Ilustrasi perhitungan estimasi jarak dari *node n* menuju titik tujuan.

Pada Algoritma *A**, digunakan perhitungan fungsi heuristik yang berbeda yaitu:

1. *Manhattan distance*, yang menghasilkan jalur vertikal dan horizontal. Perhitungan *Manhattan distance* dapat dilihat pada persamaan 3.2. Pada persamaan tersebut, digunakan operasi nilai mutlak untuk memastikan bahwa jarak antar koordinat tetap bernilai positif, karena dalam konteks gerakan *grid*, arah negatif tetap menghasilkan jarak yang harus dianggap sebagai positif.

2. *Euclidean distance*, memenghasilkan jalur vertikal, horizontal dan diagonal. Perhitungan *Euclidean distance* dapat dilihat pada persamaan 3.3. Pada persamaan tersebut, menggunakan operasi

kuadrat dan akar kuadrat berdasarkan teorema Pythagoras untuk menghitung jarak lurus antara dua titik. Penggunaan kuadrat menghilangkan efek tanda negatif, sedangkan akar kuadrat mengembalikan hasil ke satuan jarak yang sesuai.

Implementasi dari kedua perhitungan tersebut dapat dilihat pada Gambar 3.4 dan Gambar 3.5.

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

Gambar 3.6. Perhitungan *Manhattan distance* untuk melakukan estimasi jarak terpendek.

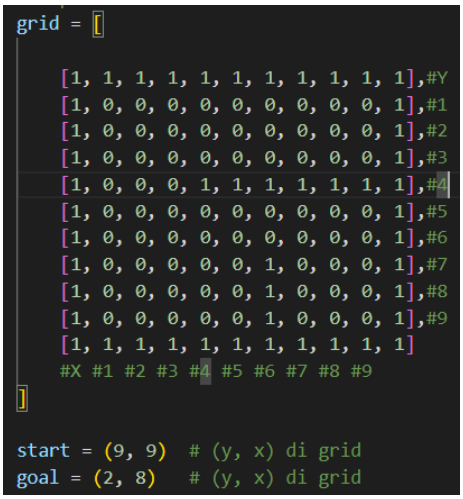
```
# Fungsi heuristic menggunakan jarak Euclidean
def heuristic(a, b):
    return ((a[0] - b[0])**2 + (a[1] - b[1])**2)**0.5
```

Gambar 3.7. Perhitungan *Euclidean distance* untuk melakukan estimasi jarak terpendek.

Perbandingan hasil implementasi dari kedua heuristik tersebut dapat dilihat pada Gambar 1.3.

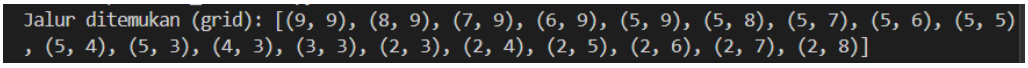
Heuristik yang digunakan akan di panggil kedalam *function* Algoritma *A** untuk menghitung *cost* setiap langkahnya.

Sistem *path planning* dibuat berdasarkan peta *grid* yang diimplementasikan dalam bentuk matriks 2 dimensi (2D) di mana *0* mempresentasikan area tanpa rintangan, sedangkan *1* mempresentasikan rintangan atau rintangan yang tidak dapat dilalui oleh robot. Berikut Gambar 3.6 menunjukkan peta *grid* yang dipakai pada algoritma *path planning.*



Gambar 3.8. matriks 2D yang mempresentasikan peta *grid*.

Hasil *path planning* dengan peta *grid* akan menghasilkan koordinat *grid* yang dapat dilihat pada Gambar 3.7.



Gambar 3.9. *Output* (koordinat *grid*).

Kemudian koordinat *grid* akan dikonversi agar sesuai dengan skala yang terdapat pada dunia CoppeliaSim.

Berikut merupakan persamaan untuk melakukan konversi *grid* menjadi koordinat dunia *(x, y)* pada lingkungan simulasi CoppeliaSim:

$$Scale\ factor = \frac{sim\ size}{grid\ size} \tag{3.4}$$

$$sim\ size = 5(5 \times 5)$$

$$grid\ size = 11(11 \times 11)$$

$$koordinat\ grid = (9 \times 9)$$

$$scale\ factor = \frac{5}{11}$$

Hitung skala faktor:

$$x = y \times scale\ factor - \frac{sim\ size}{2} \quad (3.5)$$

$$x = 9 \times 0{,}4545 \times \frac{5}{2} = \mathbf{1,5909}$$

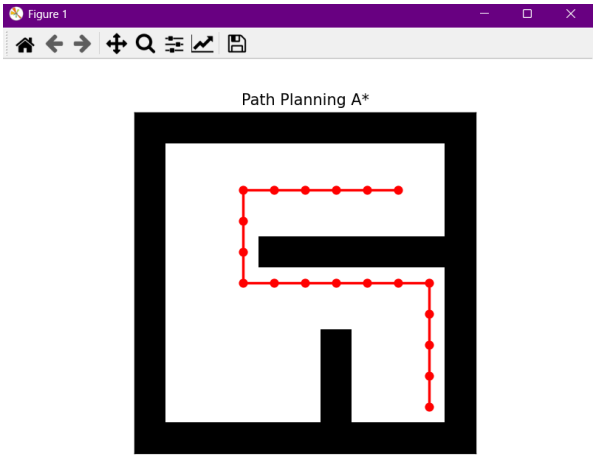
$$y = -x \times scale\ factor + \frac{sim\ size}{2} \quad (3.5)$$

$$y = -9 \times 0{,}4545 \times \frac{5}{2} = \mathbf{-1,5909}$$

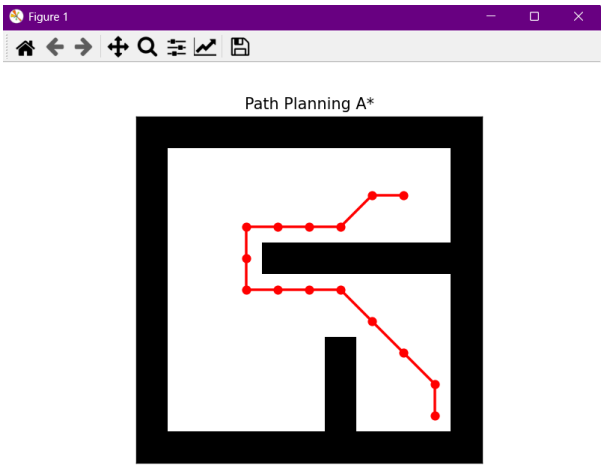
Dari perhitungan tersebut maka didapatkan skala koordinat dunia CoppeliaSim:

(x,y)= (1,5909, -1,5909)

Berikut merupakan Gambar 3.10 dan Gambar 3.11 menunjukkan visualisasi hasil *path planning* dari kedua heuristik.



Gambar 3.10. Visualisasi *path planning A** dengan heuristik *Manhattan*.



Gambar 3.11. Visualisasi *path planning A** dengan heuristik *Euclidean distance*.

Berdasarkan Gambar 3.11, garis merah pada gambar mempresentasikan jalur yang dapat dilalui robot sedangkan titik merah pada garis tersebut mempresentasikan titik koordinat yang harus dituju oleh robot. Berikut Gambar 3.12 yang menunjukkan koordinat yang telah di konversi dan disimpan kedalam ‘’*file .txt’*.

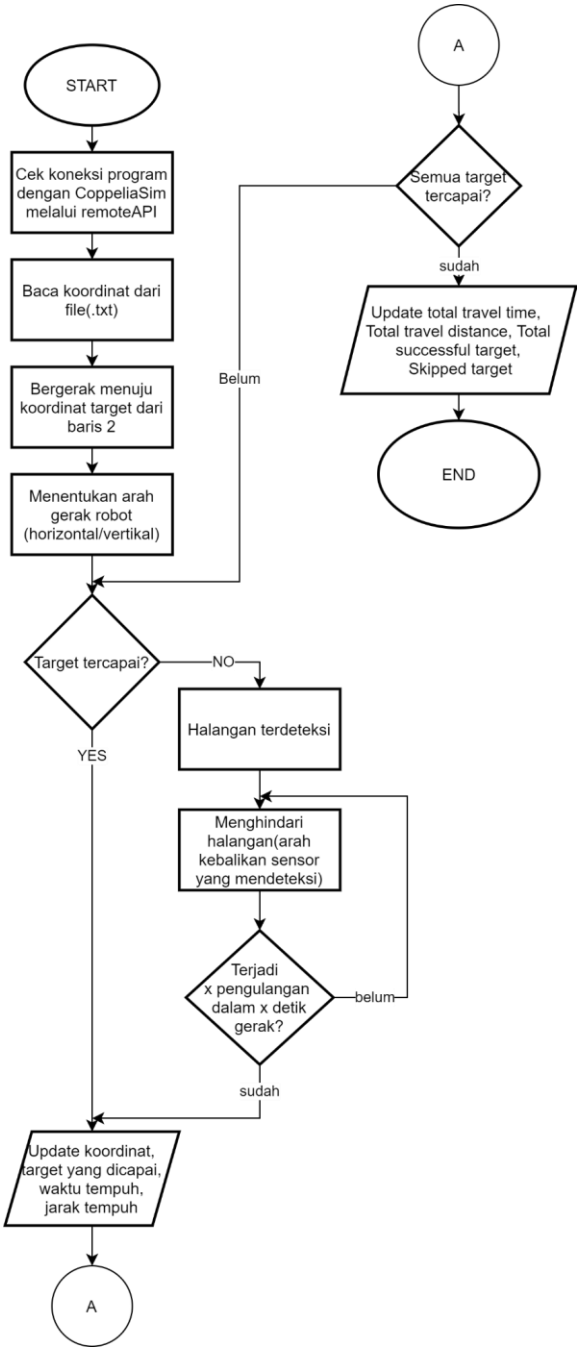
coorPath.txt > data

```
1   1.59, -1.59
2   1.59, -1.14
3   1.14, -0.68
4   0.68, -0.23
5   0.23, 0.23
6   -0.23, 0.23
7   -0.68, 0.23
8   -1.14, 0.23
9   -1.14, 0.68
10  -1.14, 1.14
11  -0.68, 1.14
12  -0.23, 1.14
13  0.23, 1.14
14  0.68, 1.59
15  1.14, 1.59
```

Gambar 3.12. Koordinat titik yang harus dituju oleh robot

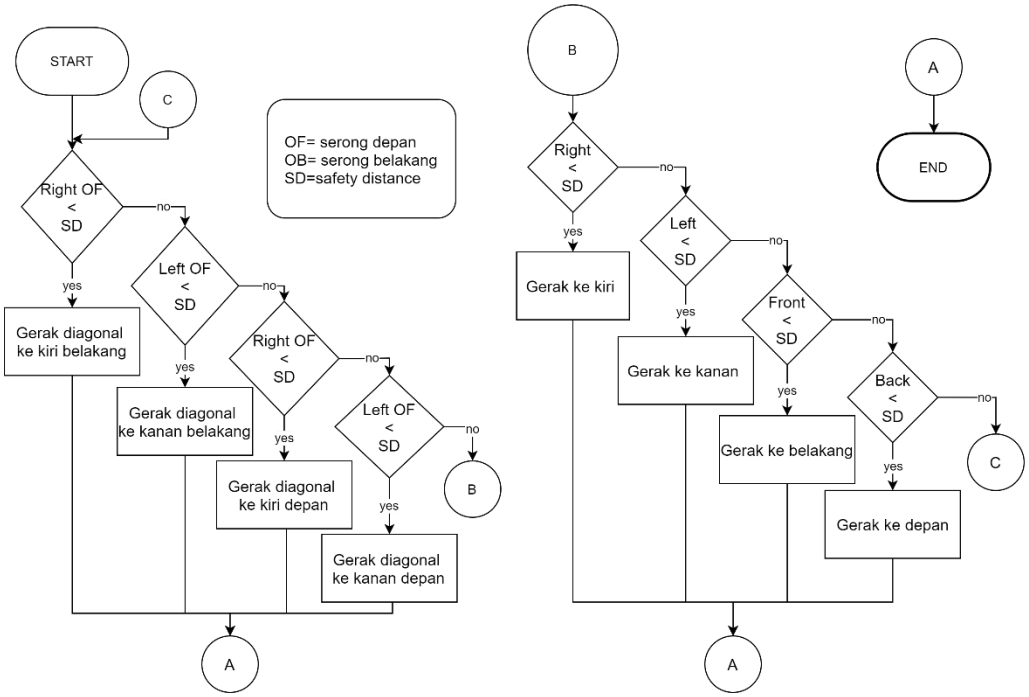
b. Implementasi Subsistem Simulasi

Subsistem simulasi berfungsi untuk mensimulasikan hasil implementasi algoritma *path planning* pada robot. *Flowchart* program dapat dilihat pada Gambar 3.13.



Gambar 3.13. *Flowchart* cara kerja robot.

Proses implementasi subsistem simulasi dimulai dengan menghubungkan program Python pada VS Code dengan CoppeliaSim untuk melakukan *remote simulation*. Setelah VS Code terhubung dengan CoppeliaSim. Program akan membaca *file .txt* yang berisi koordinat hasil *path planning*. Kemudian, program akan memerintahkan robot untuk bergerak menuju koordinat target mulai dari line 2 (*line 1= start point*) hingga koordinat *end point*. Ketika sensor mendeteksi ada nya rintangan, robot akan melakukan *obstacle avoiding* dengan melakukan gerakan yang berlawanan arah dari sensor yang mendeteksi. *Flowchart obstacle avoidance* robot dapat dilihat pada Gambar 3.14 dan 3.15.



Gambar 3.14. *Flowchart* algoritma *obstacle avoidance* robot.

Pada Gambar 3.14, robot akan melakukan gerakan *obstacle avoidance* ketika sensor mendeteksi jarak robot menuju rintangan lebih kecil dari nilai *safety distance.*

Penentuan nilai *safety distance* dapat dihitung menggunakan rumus seperti berikut:

$$SF = r - A \tag{3.6}$$

$$C = \sqrt{A^2 + B^2} \tag{3.7}$$

$$SO = r - C \tag{3.8}$$

SF= *Safety distance* tegak lurus robot

SO= *Safety distance* diagonal robot

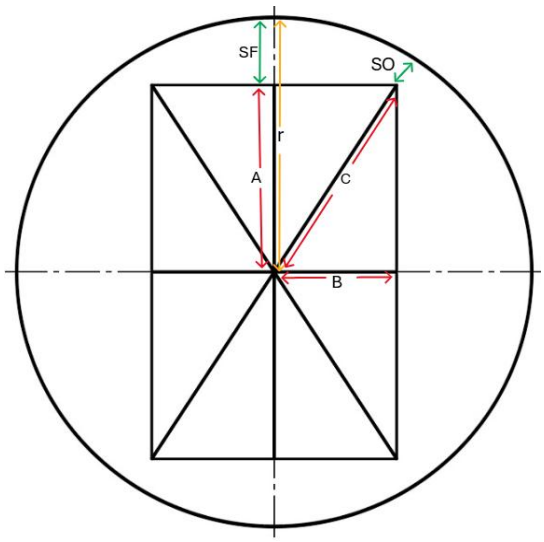
A= Jarak titik pusat-sisi depan robot

B= Jarak titik pusat-sisi samping robot

C= Sisi miring antara A dan B (jarak sisi diagonal- titik pusat robot)

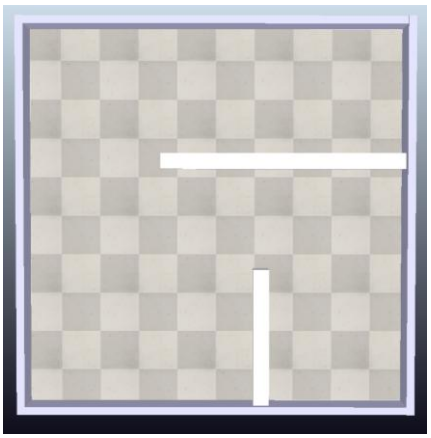
r= Jari-jari lingkaran

Berikut Gambar 3.15 merupakan ilustrasi dari *safety distance*.



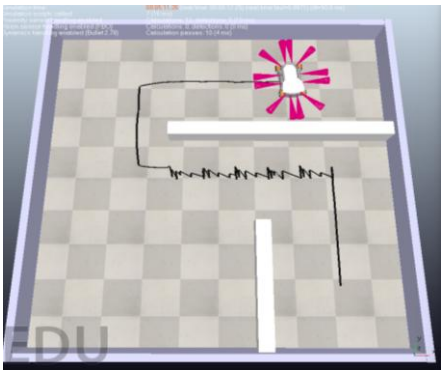
Gambar 3.15. Ilustrasi *safety distance* untuk *mobile robot.*

Pada Gambar 3.16 menunjukkan kondisi peta yang digunakan dalam CoppeliaSim. Peta tersebut merepresentasikan area logistik yang telah di sederhanakan.

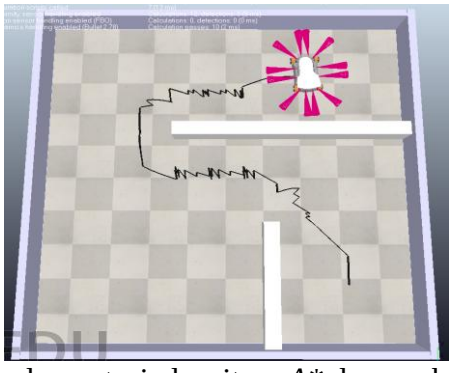


Gambar 3.16. Peta area logistik yang telah di sederhanakan

Hasil pergerakan robot dari *start point* menuju *end point* dapat dilihat pada Gambar 3.17 dan Gambar 3.18.



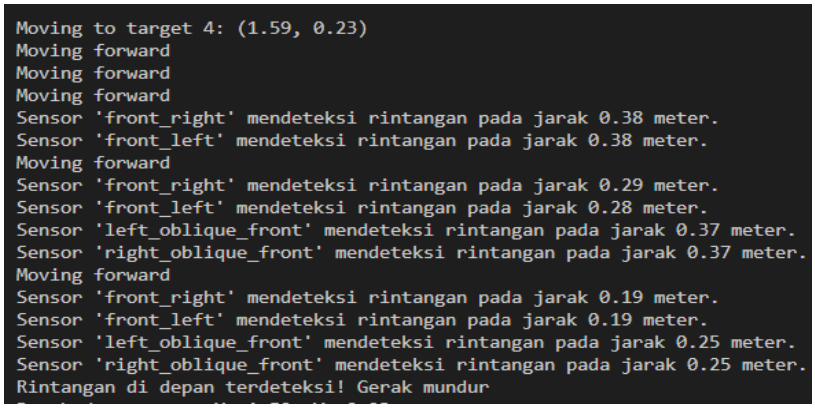
Gambar 3.17. Implementasi algoritma *A** dengan heuristik *Manhattan distance* pada KUKA youBot.



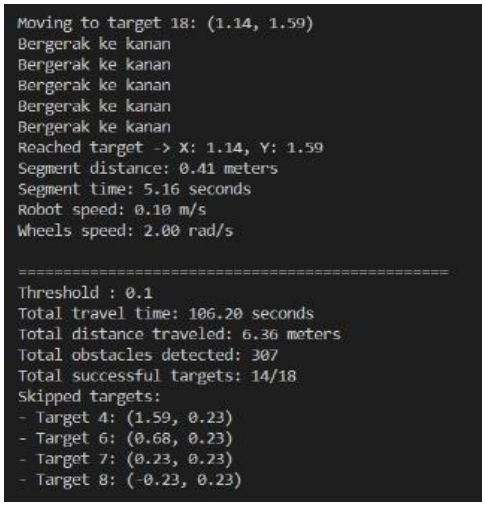
Gambar 3.18. Implementasi algoritma *A** dengan heuristik *Euclidean distance* pada KUKA youBot.

Pada Gambar 3.17 dan Gambar 3.18, terlihat bahwa robot telah berhasil mencapai koordinat *end point.* Garis hitam pada gambar

merepresentasikan *trajectory line* yang dilalui robot. *Output* hasil simulasi dapat dilihat pada Gambar 3.19 dan Gambar 3.20.



Gambar 3.19. Hasil baca pada terminal VS Code ketika mendeteksi



Gambar 3.20. Hasil baca akhir pada terminal VS Code.

3.2.2 Hambatan dan Solusi Implementasi

Implementasi hasil *path planning* pada simulasi robot di CoppeliaSim mengalami kendala ketika robot beroperasi di lingkungan sempit dengan rintangan yang menghalangi koordinat target. Pada pengujian tersebut, robot tidak mampu bergerak hingga mencapai *end point* karena terdapat koordinat target yang tidak tercapai, sehingga menyebabkan robot mengalami *infinite loop*. Pengujian dilanjutkan dengan mengubah nilai *threshold* robot dari 0.1 menjadi 0.5. Pemilihan nilai 0.5 dilakukan berdasarkan hasil eksplorasi nilai *threshold* yang lebih tinggi, dengan pertimbangan bahwa nilai tersebut memberikan hasil paling

stabil dan efektif dalam menyelesaikan lintasan tanpa mengalami kegagalan akibat *infinite loop*, dibandingkan dengan nilai lain seperti 0.2 yang sempat dipertimbangkan. Hasil pengujian menunjukkan bahwa robot mampu mencapai *end point*, meskipun sempat mengalami kesulitan ketika koordinat target terhalangi oleh rintangan. Untuk mengatasi hal tersebut, pada algoritma pergerakan robot dalam program simulasi ditambahkan mekanisme *timeout* yang memungkinkan robot keluar dari situasi yang dapat menyebabkan *infinite loop*.