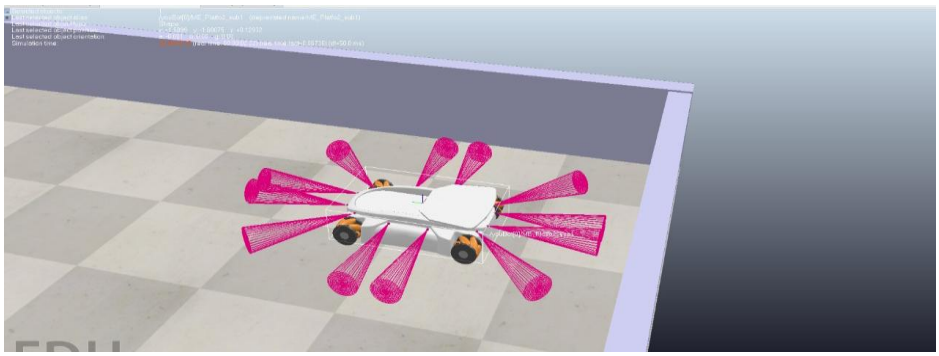
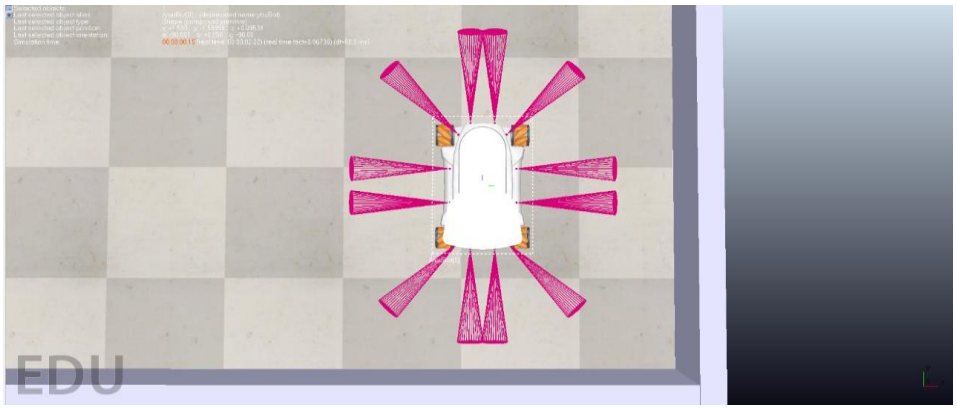
BAB II
KONSEP DESAIN & SPESIFIKASI SISTEM

2.1 Konsep Desain Sistem

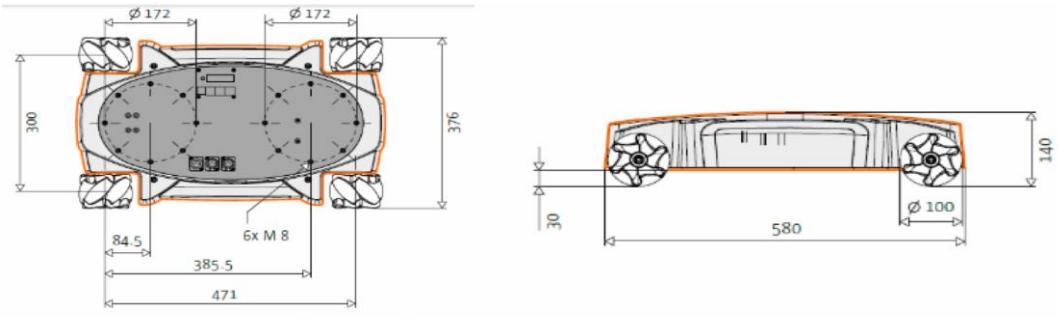
Produk sistem *mecanum wheel mobile robot* dirancang berdasarkan penelitian sebelumnya [24, 25]. Sistem *path planning* dirancang untuk mendukung MWMR *agar* dapat bergerak secara otonom pada lingkungan yang dinamis. Sistem *path planning* yang telah dirancang di implementasikan pada robot simulator CoppeliaSim, model robot dapat dilihat pada Gambar 2.1 dan Gambar 2.2.



Gambar 2.1. Model KUKA youBot yang telah disesuaikan dan digunakan untuk simulasi pada CoppeliaSim.



Gambar 2.2. Tampak atas Model KUKA youBot yang telah disesuaikan dan digunakan untuk simulasi pada CoppeliaSim.



Gambar 2.3. Dimensi KUKA youBot [26].

Mecanum wheel Mobile robot yang digunakan untuk simulasi pada CoppeliaSim adalah model KUKA youBot yang terdapat pada *library* CoppeliaSim. Model yang digunakan telah dilakukan penyesuaian seperti penambahan sensor *proximity* yang merepresentasikan sensor ultrasonik pada sisi tegak lurus dan sisi diagonal robot. Sensor tersebut akan terus bekerja dan berfungsi untuk mendeteksi rintangan pada lingkungan dinamis yang dilalui robot. Secara keseluruhan, dimensi model robot dapat dilihat pada Gambar 7. Robot ini memiliki badan berukuran 580mm×376mm×48mm, dilengkapi dengan 4 roda *mecanum* dengan diameter 100mm serta 12 sensor ultrasonik dengan jarak baca 400mm, radius 100mm, dan sudut baca 15°.

```
function sysCall_init()
    rolling=sim.getObject('.')
    slipping=sim.getObject('./slippingJoint_r1')
end

function sysCall_actuation()
    sim.setObjectOrientation(slipping|sim.handleflag_reljointbaseframe,rolling,{-math.pi/4,0,0})
end
```

Gambar 2.4. Potongan kode program pengaturan orientasi *joint* (motor) pada CoppeliaSim.

```
function coroutineMain()
    -- Put some initialization code here
    simRemoteApi.start(19999)

end
```

Gambar 2.5. Potongan program untuk memulai server *remoteAPI* yang ditulis pada CoppeliaSim.

Simulasi dilakukan menggunakan 2 bahasa program yakni, bahasa pemrograman LUA digunakan pada CoppeliaSim untuk melakukan inisialisasi

function pada masing-masing motor dan *function* untuk memulai *server remote API* agar dapat melakukan *remote simulation*, dan bahasa pemrograman Python yang dijalankan dengan Visual Studio Code *(*VS Code*)* untuk program simulasi *path planning* dan pergerakan robot.

Implementasi bahasa pemrograman LUA pada CoppeliaSim dapat dilihat pada Gambar 2.4 dan 2.5. Proses penghubungan antara CoppeliaSim dan VS Code dilakukan melalui *remote API* dengan memanfaatkan *port* yang sama, yaitu *port* 19999, sebagai jalur komunikasi. Dalam konfigurasi ini, CoppeliaSim berperan sebagai *server*, sedangkan VS Code berfungsi sebagai *client* yang meminta dan menerima data simulasi dari CoppeliaSim. Implementasi koneksi ini dapat dilihat pada Lampiran B, khususnya pada baris ke-8 yang menunjukkan inisialisasi koneksi ke *remote API server*.

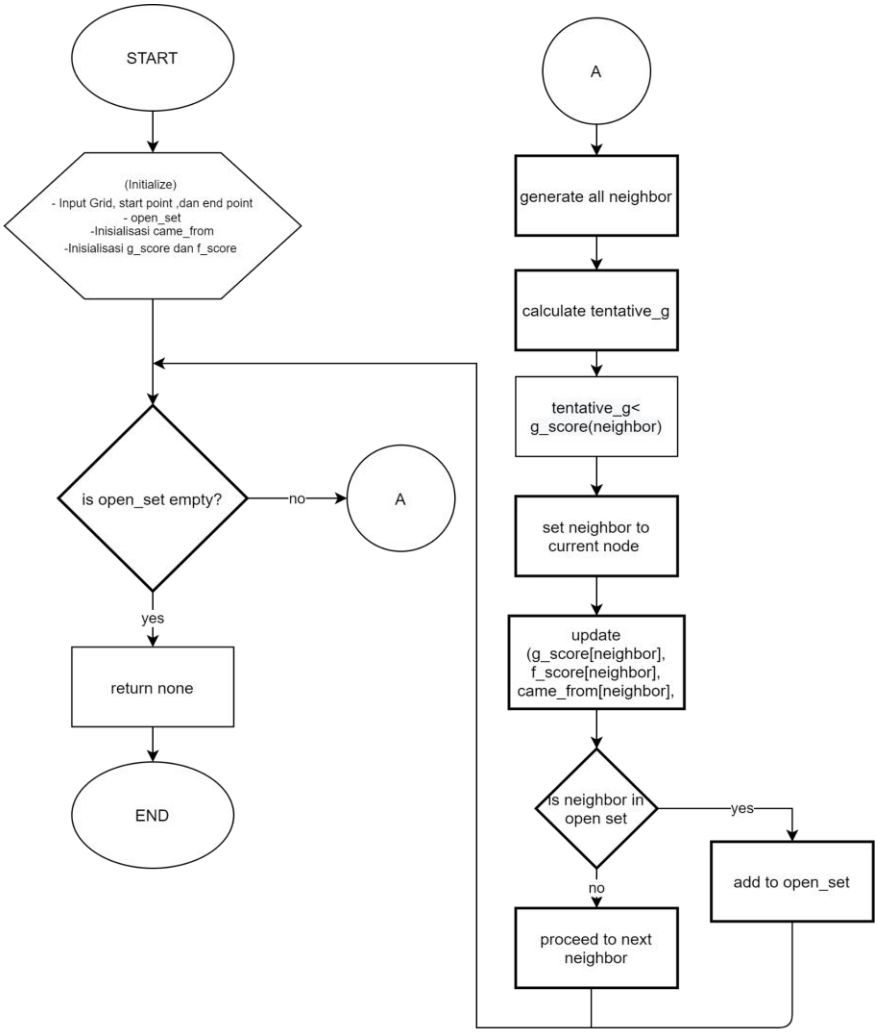
2.2 Algoritma *path planning A**

Path planning merupakan sistem perencanaan jalur yang memungkinkan bagi robot untuk menghindari tabrakan dengan rintangan saat robot bergerak dari *start point* menuju *end point* [27]. Algoritma *path planning* yang diterapkan untuk simulasi pada KUKA youBot adalah algoritma *path planning A-Star(A*).*

Algoritma A-Star (A)* merupakan salah satu metode *path planning* berbasis *grid atau graph* yang paling banyak digunakan karena kemampuannya dalam menemukan jalur terpendek secara efisien. *Algoritma A** menggunakan fungsi heuristik, yaitu fungsi estimasi yang digunakan untuk memperkirakan biaya atau jarak dari suatu titik ke tujuan akhir. Dengan bantuan fungsi heuristik, *A** dapat mengevaluasi setiap langkah pencarian jalur berdasarkan total *cost*, sehingga proses pencarian menjadi lebih cepat dan terarah dibandingkan dengan algoritma konvensional yang tidak menggunakan pendekatan heuristik [28].

Fungsi heuristik yang umum digunakan antara lain adalah *Euclidean Distance* dan *Manhattan Distance*. *Euclidean Distance* digunakan untuk mengestimasi jarak lurus antara dua titik, cocok untuk pergerakan bebas ke segala

arah. Sementara itu, *Manhattan Distance* menghitung jarak berdasarkan jumlah langkah horizontal dan vertikal, sesuai untuk lingkungan *grid* dengan gerakan tegak lurus (ortogonal). Selain kedua fungsi heuristik tersebut, terdapat juga *Chebyshev Distance* yang mempertimbangkan pergerakan diagonal secara maksimum, meskipun tidak digunakan secara utama dalam penelitian ini. Pada penelitian ini, perbandingan difokuskan pada penggunaan *Euclidean distance* dan *Manhattan Distance* sebagai fungsi heuristik utama dalam algoritma *A** [29].



Gambar 2.6. *Flowchart* Algoritma *A**

Flowchart algoritma *path planning A** dapat dilihat pada Gambar 2.6. *Flowchart* program dimulai dengan melakukan inisialisasi variabel serta membaca *input* berupa *Grid* peta, *start point* dan *end point*. Kemudian, program membuat struktur data utama yaitu, *open_set*, *came_from*, *g_score*, dan *f_score*. *Start point* kemudian

dimasukan kedalam *open_set* dengan nilai *f_score* yang dihitung berdasarkan heuristik yang digunakan.

Selanjutnya, program masuk ke dalam main *loop* yang berjalan selama *open_set* tidak kosong. Saat terjadi *looping*, *node* dengan nilai terkecil akan diambil sebagai *'current'.* Jika *node* saat ini (*current*) telah mencapai tujuan (*goal*), program akan melakukan rekonstruksi jalur dari tujuan kembali ke titik awal (*start*) dengan menggunakan *came_from* yang mencatat asal setiap *node*. Setelah itu, program akan menampilkan urutan jalur yang ditemukan dan selesai. Jika *node (current)* bukan *goal*, program akan mengecek *neighbor* yang dapat dikunjungi termasuk ke atas, bawah, kanan, kiri, dan diagonal (jika diperbolehkan).

Pada setiap *node*, program melakukan pengecekan untuk memastikan bahwa posisi tersebut masih berada dalam batasan area pencarian dan bukan merupakan rintangan. Pengecekan terhadap batasan posisi *neighbor* dapat dilihat pada Lampiran A, khususnya pada baris ke-48. Sementara itu, pengecekan apakah sel tersebut merupakan rintangan (yaitu memiliki nilai selain 0) dilakukan pada baris ke-51. Jika benar, program akan menghitung *cost* baru (*tentative_g_score*) berdasarkan jarak dari *start.* Jika *cost* tersebut lebih kecil dari *g_score* sebelumnya, program akan memperbaui *came_from, g_score* dan *f_score* untuk setiap *node*. Jika *neighbor* belum ada dalam *open_set*, program akan menambahkannya agar dapat di proses pada pengulangan berikutnya. Jika *open_set* kosong sebelum mencapai *goal*, program akan meyatakan bahawa tidak ada jalur yang ditemukan. Sementara itu, apabila jalur berhasil ditemukan, program akan melakukan proses rekonstruksi jalur dengan membaca nilai pada *came_from*. Rekonstruksi dilakukan dengan menelusuri kembali jalur dari *goal* menuju *start*, sebagaimana ditunjukkan pada baris ke-24 hingga ke-29 pada Lampiran A. Setelah jalur terbentuk, koordinat hasil jalur tersebut akan disimpan dan selanjutnya dapat digunakan untuk simulasi pada CoppeliaSim, serta divisualisasikan menggunakan *library matplotlib* untuk menggambarkan lintasan yang dihasilkan.

Implementasi dengan menggunakan *path planning algoritma A* diharapkan mampu meningkatkan efisiensi dalam pencarian rute dan mengurangi waktu tempuh. Untuk menguji hal tesebut, perangkat lunak seperti CoppeliaSim digunakan untuk melakukan simulasi implementasi Algoritma *path planning A** pada KUKA youBot *mecanum wheel mobile robot*.

2.3 Simulasi CoppeliaSim

CoppeliaSim, yang sebelumnya dikenal sebagai *Virtual Robot Experimentation Platform* (*V-REP*), merupakan perangkat lunak *robot simulator* yang banyak digunakan dalam bidang penelitian dan pendidikan karena kemampuannya dalam mendukung berbagai *physics engine* seperti *Bullet*, *ODE*, *Newton*, dan *Vortex*. Perangkat lunak ini menyediakan versi edukasi yang dapat digunakan secara gratis untuk keperluan non-komersial, seperti pendidikan dan penelitian akademik. Selain itu, CoppeliaSim mendukung berbagai bahasa pemrograman, antara lain LUA, Python, *C/C++*, dan *Java*, sehingga memberikan fleksibilitas yang tinggi dalam proses pengembangan, simulasi, serta integrasi sistem robotika. CoppeliaSim juga dapat dijalankan pada berbagai sistem operasi seperti *Windows* dan *Linux Ubuntu*. Perangkat lunak ini populer di kalangan pengguna karena fleksibilitasnya serta menyediakan berbagai opsi penyesuaian yang luas. Selain itu, CoppeliaSim menyediakan berbagai fitur yang memudahkan pengguna dalam proses pengembangan, antara lain penggunaan *built-in scripts*, *plug-in*, dan *remote API client* yang mendukung integrasi dengan berbagai bahasa pemrograman.

2.4 Metode Verifikasi Spesifikasi

2.4.1 Prosedur Pengujian dan Verifikasi

Prosedur pengujian sistem simulasi ini dibagi menjadi beberapa pengujian sebagai berikut:

1. Pengujian dengan menghubungkan VS Code dengan CoppeliaSim menggunakan *remote API* yang dilakukan untuk memastikan konfigurasi (*address* dan *port*) dari kedua perangkat lunak sama.

2. Pengujian presisi dari hasil simulasi gerak dan visualisasi *path planning* robot berdasarkan koordinat dan jalur yang telah ditentukan, dilakukan dengan menjalankan program *petaping* terlebih dahulu untuk menghasilkan koordinat yang disimpan pada *file* '.txt' yang nantinya akan dibaca oleh program simulasi Python dengan *input* koordinat. Kemudian, simulasi CoppeliaSim dijalankan secara bersamaan program simulasi Python dengan *input* koordinat.

3. Pengujian algoritma *obstacle avoidance* dilakukan dengan menganalisis jarak tempuh dan waktu tempuh robot pada setiap studi kasus. Verifikasi hasil pengujian mempertimbangkan sejumlah parameter penting dalam simulasi untuk mengevaluasi kinerja pergerakan robot. Parameter-parameter yang digunakan dalam pengujian ini meliputi:

    a. Jumlah manuver penghindaran rintangan yang dilakukan robot.
    b. Jarak tempuh rata-rata (m).
    c. Waktu tempuh rata-rata (s).