# Aplikasi Algoritma Dijkstra untuk Global Path Planning pada Swatantra (Autonomous) Unmanned Aerial Vehicle (UAV) dengan Masukan Peta Sederhana

Rizky Abdillah Rasyid – 13521109[1]
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia*
*[1]13521109@std.stei.itb.ac.id*

***Abstrak*—Autonomous Unmanned Aerial Vehicle (UAV) merupakan teknologi yang sedang dikembangkan untuk memenuhi kebutuan dan membantu manuisa. Aspek terpenting dalam pengembangan pesawat nirawak swatantra adalah sistem navigasi yang dikendalikan oleh kecerdasan buatan sehingga wahana dapat menentukan lintasannya tanpa ada kontrol langsung dari manusia. *Global Path-Planning* adalah salah satu sistem penentuan lintasan pada robot dengan prekondisi informasi lingkungannya yang berupa peta. Peta yang telah diproses menjadi graf dapat dikenali oleh wahana dan dapat ditentukan lintasan optimal yang dapat diambil untuk mencapai tujuan tersebut. Algortima Dijkstra dapat digunakan sebagai algoritma untuk menentuka lintasan terpendek yang perlu diambil wahana untuk mencapai tujuan. Algoritma ini juga termasuk unggul dibanding dengan algoritma dengan fungsionalitas yang sama.**

***Keywords*—Algortima, Dijkstra, *Autonomous*, graf, *Path-Planning*.**

## I. PENDAHULUAN

*Unmanned Aerial Vehicle* (UAV) adalah jenis pesawat (*aircraft*) yang dapat terbang tanpa kendali langsung dari pilot di dalam pesawat. Pada saat ini kemajuan teknologi dalam UAV semakin pesat, salah satunya penambahan rekayasa komputasi untuk membantu wahana untuk menyelesaikan misi. Rekayasa yang dilakukan salah satunya mengembangkan wahana swatantra (*autonomous*).

Wahana Swatantra adalah wahana yang beroperasi dengan menggunakan kecerdasan buatan (*Artificial intelligence*), dengan bantuan AI wahana dapat melakukan navigasi dan operasi perangkat lunak tanpa kontrol langsung dari pilot manusia. Salah satu penerapan kecerdasan buatan pada sistem navigasi wahana adalah pencarian laintasan (*Path Planning*). *Path planning* adalah pencarian lintasan terpendek, teroptimal atau yang dapat dilintasi untuk wahana Swatantra dari titik asal sampai titik tujuan. Permasalahan path planning pada wahana dipengaruhi pada dua faktor: (1) Lingkungan, dapat bersifat dinamik atau statik, (2) Pengetahuan wahana tentang informasi lingkunganya; Jika wahana telah mengetahui informasi tentang lingkungannya dan lingkungan bersifat statik, maka kasus tersebut termasuk ke dalam jenis *global path-planning* [8]. Pengetahuan wahana tentang lingkungannya dapat dibentuk dengan memberikan masukan sebuah peta (*map*) yang akan dilakukan proses pembacaan peta dengan program perangkat lunak menjadi suatu struktur data tertentu sehingga informasi peta dapat dikenali oleh wahana. Salah satu struktur data sederhana yang dapat digunakan adalah graf, karena dapat dibentuk sebagai sebuah matriks ketetanggaan (*adjacency matrix*).

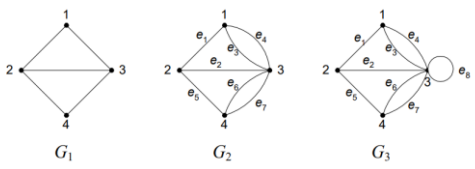
Penentuan lintasan pada graf memiliki beragam alternatif algortima sperti Algoritma Bellman-Ford, Algoritma Dijkstra, Algortima Floyd-Warshall, dan lainnya. Hanna dan Abdelfatah [9] telah melakukan riset perbandingan algoritma pencari lintasan terpendek, algortima dijkstra menjadi algoritma dengan kompleksitas waktu yang lebih cepat dari algoritma lain. Algoritma Dijkstra dikenal sebagai algoritma penyelesaian lintasan terpendek graf dari titik awal hingga titik tujuan.

Penerapan sistem path planning dengan struktur data graf dan algoritma dijkstra sebagai penyelesaian graf pada wahana diharapkan dapat membentuk wahana swatantra sederhana yang dapat dikembangkan lanjut untuk membantu penyelesaian misi wahana seperti pengiriman barang, penyiraman lahan, patroli, dan lainnya.

## II. TEORI DASAR

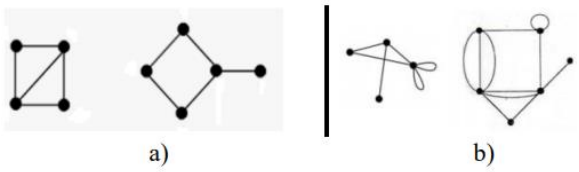
### A. Teori Graf

Graf adalah representasi dari kumpulan objek-objek disktrit dan hubungan antara objek-objek tersebut. Graf didefinisikan sebagai tuple G = (V, E), dengan V adalah himpunan tak kosong dari simpul-simpul (vertices) dan E adalah himpunan sisi (edges) yang menjadi penhubung antara dua simpul.




Gambar 1. Graf
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf, diakses pada 12/12/2022

Berdasarkan ada tidaknya gelang atau sisi ganda pada graf, graf digolongkan menjadi dua jenis: (1) Graf Sederhana, graf yang tidak memiliki gelang atau sisi ganda, (2) Graf tak-sederhana, graf yang memiliki gelang atau sisi ganda.

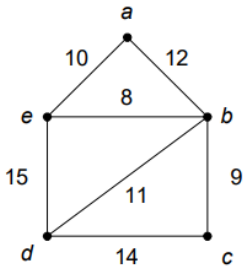



Gambar 2. Ilustrasi Graf sederhana (a) dan Graf Tidak Sederhana (b)
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf, diakses pada 12/12/2022

Berdasarkan orientasi arah pada sisi graf, graf dibagi menjadi dua jenis, yaitu graf tak-berarah dan graf berarah. Graf tak-berarah adalah graf yang tidak memiliki orientasi arah pada sisinya. Sedangkan, graf berarah adalah graf yang memiliki orientasi arah pada sisinya.

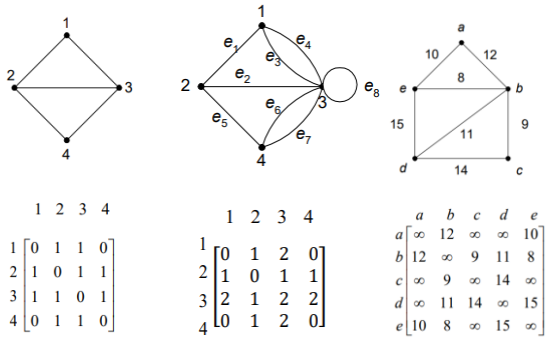
Pada graf dikenal istilah lintasan (*path*) dan sirkuit(*circuit*). Lintasan (*path*) dengan panjang n dari simpul awal $v_n$ ke simpul tujuan $v_n$ pada graf G adalah barisan berselang-seling simpul dan sisi yang berbentuk $v_0, e_1, v_1, e_2, v_2, \ldots, v_{n-1}, e_n, v_n$ sedemikian sehingga $e_1 = (v_0, v_1), e_2 = (v_1, v_2), \ldots, e_n = (v_{n-1}, v_n)$ adalah sisi-sisi dari graf G. Panjang lintasan adalah akumulasi nilai dari sisi pada suatu lintasan. Sirkuit adalah lintasan yang memiliki simpul awal dan simpul akhir yang sama.

Istilah lain pada graf adalah graf berbobot, yaitu graf yang memiliki nilai, harga atau bobot pada setiap sisinya.




Gambar 3. Graf Berbobot
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf, diakses pada 12/12/2022

Graf tidak hanya direpresentasikan sebagai gambar simpul-simpul dengan sisi-sisi, tetapi dapat juga direpresentasikan sebagai matriks. Matiks ini bernama matriks ketetanggaan (*Adjacency Matrix*), indeks baris dan kolom pada matriks ini merepresentasikan simpul-simpul pada graf. Elemen pada matriks ini bertipe *boolean* (0/1) yang mengindikasi apakah antara dua simpul (indeks baris dan kolom) terdapat sisi yang menghubungkannya. Selain menggunakan elemen bertipe boolean, matriks ketetanggaan dapat dibentuk dengan alternatif elemen lain, seperti elemen bertipe bilangan bulat yang menunjukkan jumlah sisi yang menghubungkan dua simpul pada graf tak-sederhana dan menggunakan elemen bertipe bilangan bulat atau real yang menunjukan nilai (bobot) dari sisi tersebut pada graf berbobot.



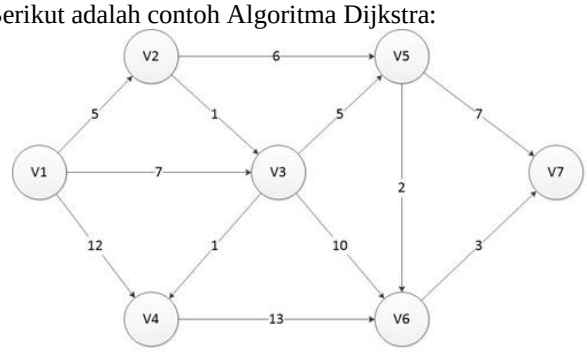

Gambar 4. Matriks Ketetanggaan dengan tipe data elemen yang berbeda
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian2.pdf, diakses pada 12/12/2022

## *B. Algoritma Dijkstra*

Algortima Dijkstra adalah algoritma pencarian lintasan terpendek dari *single source vertex* menuju ke semua *vertex* lain. Algoritma ini termasuk ke dalam algoritma *breadth-first search* (BFS). Algortima ini bekerja dengan membuat lintasan ke satu simpul optimal di setiap langkahnya. Jadi ketika mencapai langkah ke *n*, setidaknya ada *n* simpul yang sudah diketahui lintasan terpendeknya. Algoritma Dijkstra bekerja dengan Langkah-langkah berikut:

1. Terdapat graf berbobot terdefinisi dengan bobot yang merepresentasikan jarak dari sisi tersebut. Menentukan titik yang menjadi simpul awal. Dijkstra akan melakukan pencarian dari satu simpul ke simpul lain dengan dan ke simpul.
2. Pada simpul awal, nilai awal di set nilai 0 dan nilai tak hingga untuk bobot antar simpul yang tidak terhubung dengan sisi.
3. Pada simpul awal, bandingkan setiap bobot dari simpul tetangga yang belum dilalui dan akumulasi jarak dari simpul awal. Jika didapat jarak yang lebih kecil dari jarak sebelumnya (data yang sudah direkam) hapus jarak lama, lalu simpan informasi jarak yang baru.
4. Setelah membandingkan jarak terhadap simpul tetangga, tandai node yang telah dilalui. Hal ini dilakukan agar node yang telah di evaluasi tidak dipertimbangkan kembali, sehingga jarak yang disimpan adalah jarak terakhir yang paling minimum.

Berikut adalah contoh Algoritma Dijkstra:




Gambar 5. Graf berbobot
Sumber: https://mti.binus.ac.id/2017/11/28/algoritma-dijkstra/, diakses pada tanggal 12/12/2022

Berikut hasil dari algoritma dijkstra pada graf Gambar 5:

Tabel 1. Hasil Pemrosesan dengan Algoritma Dijkstra
Sumber: https://mti.binus.ac.id/2017/11/28/algoritma-dijkstra/, diakses pada 12/12/2022

| Iteration | Unvisited (Q) | Visited (S) | Current | Node : Min = (dist[node], prev[node]iteration | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | V1 | V2 | V3 | V4 | V5 | V6 | V7 |
| | Initialization {V1,V2,V3,V4,V5,V6,V7} | {-} | | (0, -)0 | (∞, -)0 | (∞, -)0 | (∞, -)0 | (∞, -)0 | (∞, -)0 | (∞, -)0 |
| 1 | {V2,V3,V4,V5,V6,V7} | {V1} | V1 | | (5,V1)1 | (7,V1)1 | (12,V1)1 | (∞,V1)1 | (∞,V1)1 | (∞,V1)1 |
| 2 | {V3,V4,V5,V6,V7} | {V1,V2} | V2 | | | (6,V2)2 | (12,V1)1 | (11,V2)2 | (∞,V2)2 | (∞,V2)2 |
| 3 | {V4,V5,V6,V7} | {V1,V2, V3} | V3 | | | | (7,V3)3 | (11,V3)3 | (16,V3)3 | (∞,V3)3 |
| 4 | {V5,V6,V7} | {V1,V2, V3,V4} | V4 | | | | | (11,V3)3 | (16,V3)3 | (∞,V3)3 |
| 5 | {V6,V7} | {V1,V2, V3,V4,V5} | V5 | | | | | | (13,V5)5 | (18,V5)5 |
| 6 | {V7} | {V1,V2, V3,V4,V5,V6} | V6 | | | | | | | (16,V6)6 |

Dari hasil pada **Tabel 1** didapat lintasan terpendek dari V1 ke V7 adalah 16 dengan lintasan yang melalui simpul V1, V2, V3, V5, V6 dan V7 secara berurutan. Physics

### *C. Robotic Operating System (ROS)*

*Robotic Operating System* (ROS) adalah sistem perangkat lunak yang membantu membangun aplikasi robot. *Robotic Operating System* merupakan sebuah *framework* yang sistemnya bekerja dengan komunikasi antar *node* dengan metode *publish* dan *subscribe* yang mengirimkan dan menerima informasi data yang disebut *message*. ROS dapat dikembangkan dengan bahasa pemrograman *C++* dan *Python* atau keduanya.

### *D. Gazebo Simulation*

Gazebo adalah simulasi 3D *open source*. Gazebo mengsimulasi fisik dunia nyata, simulasi ini membantu pengembangan robot untuk menguji algoritma dan merancang robot secara digital sebelum dilakukan uji coba pada dunia nyata. Dengan simulasi ini dapat menghindarkan kecelakaan atau crash pada pengujian robot, karena pada sistem yang dibuat dapat berpotensi memiliki *bug* yang berisiko.

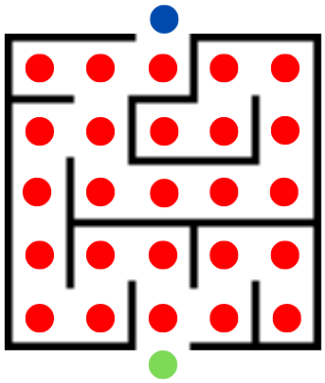
### *E. OpenCV*

OpenCV (*Open-Source Computer Vision Library*) adalah pustaka *computer vision* dan pembelajaran mesin yang bersifat *open source*. OpenCV dibuat untuk memenuhi kebutuhan dasar dalam membangun aplikasi *computer vision* dan mengakselerasi penggunaan presepsi mesin pada produk komersial. *Library* OpenCV dapat digunakan untuk mendeteksi objek dan melakukan manipulasi citra digital sehingga dapat memenuhi kebutuhan pengembangan robot dengan misi yang berkaitan dengan citra.

### *F. Peta Sederhana*

Peta sederhana yang dimaksud dalam makalah ini adalah peta dengan batasan tertentu yang disesuaikan dengan sistem yang diguanakan. Berikut adalah konfigurasi peta:

1. Peta merupakan gambar digital dengan ukuran sembarang dengan bentuk umum sebuah labirin berpetak.
2. Dinding (*Obstacle*) pada peta digambarkan sebagai garis berwarna hitam dengan lebar minimal 3 *pixels*.
3. Pada setiap petak ruang pada peta diberikan penanda berupa lingkaran merah tanpa garis luar. Titik awal wahana diberikan penanda lingkaran berwarna hijau tanap garis luar dan titik akhir atau tujan wahana diberikan penanda lingkaran berwanra biru.



Gambar 6. Peta Sederhana
Sumber: Dokumentasi Pribadi

## III. METODE

### *A. Pembacaan Peta*

Pembacaan peta sederhana dilakukan dengan pemrosesan citra dengan pustaka OpenCV. Berikut adalah Langkah-langkah pembacaan peta:

1. Menyiapkan peta sederhana yang telah didefinisikan pada teori dasar dan membaca peta dengan fungsi pada OpenCV.
2. Mendeteksi penanda merah pada peta dengan keluaran akumulasi data kontur penanda merah.
3. Lakukan deteksi pada penanda yang berwarna biru dan penanda berwarna hijau seperti deteksi yang dilakukan pada langkah 2.
   Gambar Kontur
4. Inisialisasi matriks *adjacency* dengan ukuran dari akumulasi jumlah kontur merah, jumlah kontur biru dan jumlah kontur hijau.
5. Untuk setiap kontur yang telah didapat akan ditentukan titik berat dari setiap kontur dengan bantuan pustaka OpenCV. Titik berat dari tiap kontur akan bertipe `cv::Point` yang terdiri dari posisi sumbu x dan y titik berat dalam satuan *pixel*.
6. Lakukan pengulangan pada setiap titik berat masing-masing kontur dengan validasi tidak adanya dinding diantara titik tersebut. Validasi dilakukan dengan mengambil data warna antara kedua titik yang akan dihubungkan dalam format gambar *gray*. Jika diantara titik terdapat dinding yang ditandai dengan nilai pada *pixel* kurang dari 10 (nilai kurang dari 10 menunjukan warna pada citra paling hitam (gelap) pada rentang 0-255) maka tidak dilakukan masukan ke dalam matriks *adjacency*. Jika tidak terdapat dinding maka melakukan masukan ke dalam elemen $[a_{ij}]$ (i adalah titik awal dan j adalah titik tujuan) dengan nilai jarak antara 2 titik. Jarak didapat dengan Pers. (1):

$$j = \sqrt{\Delta x^2 + \Delta y^2} \quad (1)$$

7. Jika pengulangan pada setiap titik telah dilakukan dan matriks *adjacency* telah terisi nilai setiap sisi graf.


Gambar 7. Matriks Ketetanggaan hasil pembacaan peta (NaN menunjukan bahwa tidak ada hubungan antara simpul)
Sumber: Dokumentasi Pribadi

8. Maka proses dilanjutkan pada penentuan lintasan terpendek pada graf.

### *B. Penentuan Lintasan Terpendek (Dijkstra Algorithm)*

Penentuan lintasan terpendek pada perencana lintasan wahana dilakukan dengan algoritma Dijkstra. Berikut adalah langkah-langkah penentuan lintasan terpendek:

1. Setelah didapat matriks ketetanggaan, simpan nama-nama simpul yang merupakan indeks (indeks dimulai dari nilai 1) dari matriks ke dalam suatu list bernama POS.
2. Lakukan pemrosesan graf dengan algoritma Dijkstra dengan metode yang telah dijelaskaan pada bagian Teori Dasar dengan tambahan. Tambahan berupa penyimpanan urutan simpul ke dalam *list* Bernama NODE.
3. Penyimpanan urutan simpul dilakukan bersamaan dengan dilakukannya pembaruan jarak pada langkah ketiga dalam pemrosesan Algoritma Djikstra.
4. Jika telah selesai dilakukan Algortima Dijkstra, akan dihasilkan Kumpulan urutan simpul pada *list* NODE yang akan digunakan dalam menentukan lintasan terpendek. *List* NODE merupakan informasi hubungan antara simpul dengan elemen pada indeks *list* merupakan simpul awal dan indeks *list* merupakan simpul setelah simpul awal (elemen *list* pada indeks).
5. Selanjutnya penentuan lokasi simpul yang dilewati dan penyesuaian data dari nilai jarak pada pixel menjadi nilai jarak pada lingkungan simulasi gazebo.

### *C. Penentuan Lokasi Simpul yang Dilewati*

Pada bagian penentuan lintasan terpendek telah didapat *List* NODE yang merupakan data informasi hubungan antar simpul. Karena tidak membutuhkan semua simpul untuk dilalui oleh wahana, maka akan ditentukan simpul yang perlu dilewati. Karena elemen indeks pada *list* NODE merupakan simpul awal (sebelum) dan indeks *list* adalah simpul setelah, maka dapat dilakukan pencarian dengan pendekatan yang sama dengan pencarian pada *linked list* dengan titik awal merupakan indeks tujuan (*goal*) dan penelusuran secara mundur (*backtrack*).


Gambar 8. Hasil Penentuan Lokasi secara mundur
Sumber: Dokumentasi Pribadi

Berikut adalah langkah-langkah penentuan lokasi simpul:

1. Inisialisai tipe data *vector* dengan tipe data elemen koordinat kartesian.
2. Penelusuran simpul dimulai melalui simpul tujuan akhir (indeks terbesar) dan disisipkan koordinat simpul pada indeks awal *vector*.
3. Selanjutnya akses simpul selanjutnya dengan menuju indeks selanjutnya yang sama dengan elemen indeks saat ini. Lalu, sisipkan koordinat pada indeks awal vector.
4. Lakukan langkah 3, hingga mencapai indeks dengan elemen yang bernilai -1 yang merupakan indeks titik awal (*start*).
5. Didapat *vector* yang merupakan kumpulan titik koordinat yang akan dilalui oleh wahana.

### *D. Penyesuaian Data*

Karena terdapat ketidaksesuaian data hasil perosesan citra dengan data pada lingkungan simulasi, perlu dilakukan penyesuaian data pada setiap lokasi simpul-simpul graf dengan melakukan translasi dan dilatasi. Translasi dilakukan degan melakukan pengurangan semua titik koordinat pada *vector* dengan titik koordinat awal (termasuk koordinat awal) sehingga akan didapat titik koordinat awal (0,0) dan titik koordinat simpul lain menyesuaikan. Dilatasi dilakukan dengan melakukan perkalian pada semua titik koordinat dengan rasio yang didapat dari Pers. (2):

$$rasio = \frac{Panjang\ Bangunan\ Gazebo\ (m)}{Panjang\ Bangunan\ pada\ Peta\ (px)} \quad (2)$$

## IV. HASIL

### *A. Program*

Program disusun dengan meliputi beberapa bagian:

1. UAV_Control
   Bagian ini berisi fungsi-fungsi yang berperan dalam kontrol wahana, seperti inisialisasi *publisher* dan *subscriber* ROS, fungsi perintah *take-off* dan fungsi perintah untuk mendarat. Bagian yang penting adalah *waypoint* yang merupakan fungsi untuk menunjukan destinasi yang akan dikunjungi oleh wahana.

2. ADT_Graf
   Bagian ini berisi fungsi-fungsi yang berfungsi untuk melakukan pemrosesan graf, seperti pembacaan graf dengan pemrosesan citra menggunakan pustaka OpenCV, melakukan validasi sisi yang menghubungkan antar simpul pada graf.

3. ADT_Dijkstra
   Bagian ini berisi fungsi-fungsi yang digunakan untuk menjalankan Algortima Dijkstra. Selain itu, terdapat fungsi bantuan `getMinimum` yang mencarikan sisi

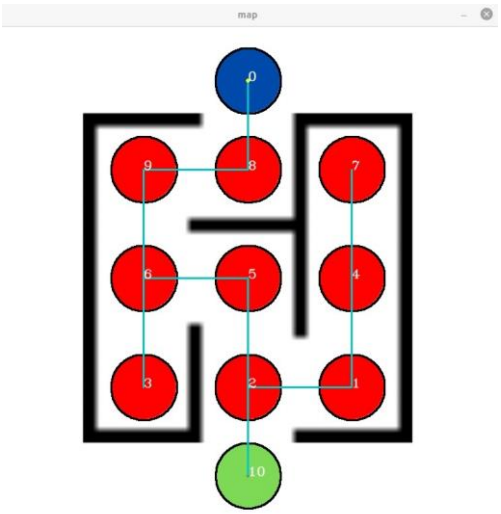
terpendek pada graf. Sebagai penunjang bagian UAV_Control pada bagian ini juga terdapat fungsi untuk melakukan penyesuaian dari koordinat citra menjadi koordinat lingkungan simulasi.

### *B. Graf*

Dari pengujian yang telah dilakukan didapat hasil berupa graf hasil pembacaan peta yang dilakukan dengan pemrosesan citra yang berupa graf dalam bentuk matriks adjacency.

```
rasyid@rasyid-kun-2280:~/Documents/Code/projec
ld$ /home/rasyid/Documents/Code/project-1/buil
ct-1
NaN NaN NaN NaN NaN NaN NaN NaN 133 NaN NaN
NaN NaN 155 NaN 163 NaN NaN 325 NaN NaN NaN
NaN 155 NaN NaN NaN 163 NaN NaN NaN NaN 132
NaN NaN NaN NaN NaN NaN 163 NaN NaN 325 NaN
NaN 163 NaN NaN NaN NaN NaN 162 NaN NaN NaN
NaN NaN 163 NaN NaN NaN 156 NaN NaN NaN 295
NaN NaN NaN 163 NaN 156 NaN NaN NaN 162 NaN
NaN 325 NaN NaN 162 NaN NaN NaN NaN NaN NaN
133 NaN NaN NaN NaN NaN NaN NaN NaN 156 NaN
NaN NaN NaN 325 NaN NaN 162 NaN 156 NaN NaN
NaN NaN 132 NaN NaN 295 NaN NaN NaN NaN NaN
Gtk-Message: 23:35:40.937: Failed to load modu
```

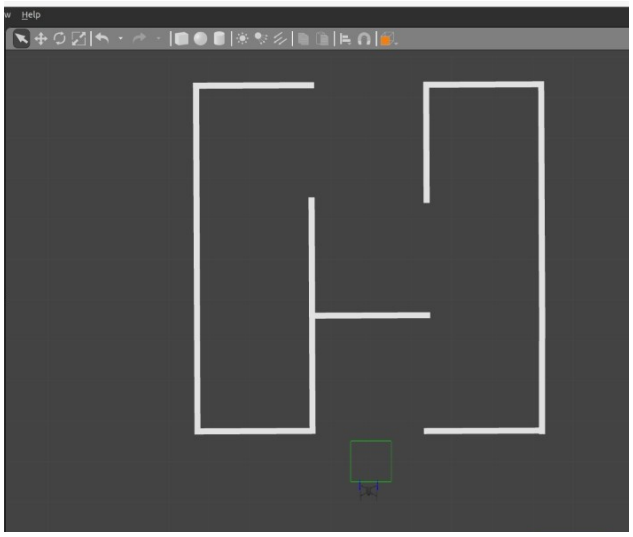
Gambar 9. Matriks Ketetanggaan hasil pembacaan peta (NaN menunjukan bahwa tidak ada hubungan antara simpul)
Sumber: Dokumentasi Pribadi



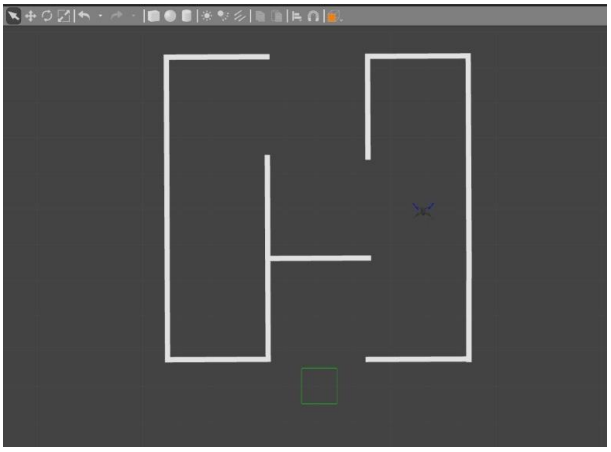

Gambar 10. graf peta dengan bantuan pustaka OpenCV
Sumber: Dokumentasi Pribadi

### *C. Simulasi Gazebo*

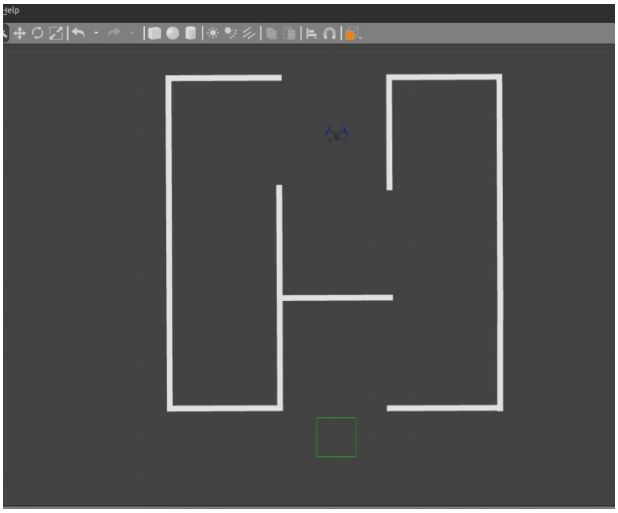
Telah berhasil dilakukan simulasi program dengan menggunakan ROS dan Gazebo yang ditunjukan pada Gambar 11-13.



Gambar 11. Posisi UAV mulai bergerak dari titik awal
Sumber: Dokumentasi Pribadi



Gambar 12. Wahana berada di dalam bangunan
Sumber: Dokumentasi Pribadi



Gambar 13. Wahana hampir mencapai titik akhir/tujuan
Sumber: Dokumentasi Pribadi

Untuk dokumentasi pengujian lebih lengkap dapat diakses pada pranala *repository*. Untuk pengujian ini kontrol wahana yang digunakan diadaptasi dari program yang dibuat oleh Intelligent-Quads (Pranala: https://github.com/Intelligent-Quads). Dari pengujian didapat bahwa wahana mampu

menentukan lintasan tercepat untuk mencapai titik akhir.

## V. KESIMPULAN

Berhasil dilakukan pembacaan peta dengan pemrosesan gambar dan mengubahnya menjadi graf dalam bentuk matriks adjacency. Algoritma Dijkstra dapat diaplikasikan dalam pengembangan pencarian lintasan terpendek (*Global Path Planning*) dengan masukan peta sederhana. Penggunaan ini juga bisa disimulasikan pada lingkungan simulasi 3D sehingga sangat memungkinkan untuk bisa diuji pada lingkungan dunia nyata.

## VI. SARAN

Saran untuk penulis selanjutnya adalah memberbanyak pengujian dan penyesuaian kontrol lebih lanjut pada wahana untuk menghindari kesalahan (*error*) pada pergerakan wahana.

## DAFTAR PUSTAKA


[1] A. S. Girsang, "Algoritma Dijkstra - MTI," 28 November 2017. [Online]. Available: https://mti.binus.ac.id/2017/11/28/algoritma-dijkstra/. [Accessed 12 Desember 2022].

[2] Robotic Simulation Services, "ROS Gazebo: Everything You Need To Know," 19 Januari 2021. [Online]. Available: https://roboticsimulationservices.com/ros-gazebo-everything-you-need-to-know/. [Accessed 12 Desember 2022].

[3] OpenCV, "About - OpenCV," [Online]. Available: https://opencv.org/about/. [Accessed 12 Desember 2022].

[4] geeksforgeeks, "Dijkstra's Shortest Path Algorithm | Greedy Algo-7," geeksforgeeks, 31 Agustus 2022. [Online]. Available: https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/. [Accessed 12 Desember 2022].

[5] E. A. Team, "How to implement Dijkstra's Algorithm in C++," educative, 2022. [Online]. Available: https://www.educative.io/answers/how-to-implement-dijkstras-algorithm-in-cpp. [Accessed 12 Desember 2022].

[6] Ros, "ROS: Home," Open Robotics, 2021. [Online]. Available: https://www.ros.org/. [Accessed 12 Desember 2022].

[7] R. G. L. Narayanan and O. C. Ibe, "6 - Joint Network for Disaster Relief and Search and Rescue Network Operations," *Wireless Public Safety Networks 1*, vol. 1, pp. 163-193, 2015.

[8] S. A. Fadzli, S. I. Abdulkadir, M. Makhtar and A. A. Jamal, "Robotic Indoor Path Planning using Dijkstra's Algorithm with Multi-Layer Dictionaries," Faculty of Informatics & Computing Universiti Sultan Zainal Abidin, Terengganu, 2014.

[9] H. M. Abu-Ryash and D. A. Tamimi, "Comparison Studies for Different Shortest path Algorithms," *INTERNATIONAL JOURNAL OF COMPUTERS & TECHNOLOGY*, vol. 14, pp. 5979-5986, 2015.

[10] R. Munir, "Homepage Rinaldi Munir," November 2022. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/. [Accessed 11 December 2022].


## PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 12 Desember 2022

Rizky Abdillah Rasyid NIM