Analisis Strategi Pencarian Jalur: Komparasi Heuristik A, Greedy Best-First Search, dan Dijkstra pada Nav2 Stack*

Studi Komparatif Optimasi Rute pada Robotika

Faris Wirakusuma Triawan - 13524130
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: fariswkt@gmail.com , 13524130@std.stei.itb.ac.id

Abstract—Navigasi robot otonom membutuhkan strategi pathfinding yang efisien di lingkungan statis. Makalah ini menganalisis kinerja algoritma A-Star (A*), Greedy Best-First Search (GBFS), dan Uniform Cost Search (UCS/Dijkstra) pada perencana jalur global Nav2. Implementasi menggunakan middleware ROS 2 yang terintegrasi dengan visualisasi real-time berbasis Bevy Engine via rosbridge. Parameter evaluasi mencakup total panjang jalur dan jumlah simpul yang dieksplorasi. Hasil eksperimen menunjukkan UCS konsisten menjamin lintasan terpendek absolut (4,26 m pada Peta 1; 7,11 m pada Peta 2), namun boros memori. Sebaliknya, GBFS sangat cepat namun gagal menghasilkan rute optimal akibat jebakan local minima. A* terbukti menjadi solusi kompromi terbaik dengan menghasilkan lintasan mendekati optimal (4,71 m pada Peta 1; 7,94 m pada Peta 2) sekaligus menekan ruang ekspansi simpul secara signifikan. Kode sumber dan data log tersedia terbuka di repositori BawalPathFinder .

Kata Kunci—Pathfinding, ROS 2, A, Dijkstra, GBFS, Robotika Otonom.

I. PENDAHULUAN

Di era sistem otonom saat ini, navigasi yang tangguh dan efisien merupakan persyaratan fundamental bagi robot seluler. Untuk beroperasi secara efektif dalam lingkungan yang dinamis, robot harus memiliki kemampuan untuk merencanakan lintasan bebas tabrakan dari posisi awal hingga ke tujuan secara real-time [1]. Tugas ini sangat bergantung pada algoritma pathfinding yang berfungsi sebagai "otak" dari sistem navigasi otonom.

Berbagai strategi pathfinding telah dikembangkan untuk mengatasi tantangan ini, dengan masing-masing memiliki trade-off komputasi yang berbeda. Algoritma Dijkstra atau Uniform Cost Search (UCS), misalnya, memberikan solusi optimal dengan menjelajahi seluruh kemungkinan node berdasarkan biaya akumulatif terkecil, namun membutuhkan biaya komputasi yang tinggi pada peta berskala besar [2], [3]. Sebaliknya, Greedy Best-First Search (GBFS) menekankan pada kecepatan komputasi dengan memprioritaskan node yang secara lokal paling dekat ke target menggunakan fungsi heuristik, meskipun seringkali mengorbankan optimalitas jalur [4]. Algoritma A* hadir sebagai titik tengah yang strategis, memanfaatkan kombinasi biaya aktual dan heuristik terinformasi untuk menyeimbangkan proses pencarian [5].

Implementasi praktis dari strategi pencarian ini dilakukan pada level framework robotika inti [1] menggunakan modul Navigation 2 (Nav2) [6]. Pemilihan strategi ini secara langsung mempengaruhi kemampuan robot untuk mengelola sumber daya komputasi yang terbatas. Untuk mengevaluasi kinerjanya secara objektif, diperlukan sistem visualisasi antarmuka pengguna (user interface) yang berkinerja tinggi berbasis Bevy Engine [7] yang terhubung secara asinkron melalui protokol bridge komunikasi data [8].

Makalah ini menyajikan perbandingan analisis antara algoritma A*, GBFS, dan Dijkstra/UCS yang diimplementasikan sebagai komponen modular pada Nav2. Penelitian ini mengevaluasi algoritma berdasarkan efisiensi komputasi, jumlah eksplorasi node, dan total biaya jalur. Dengan menganalisis metrik-metrik tersebut, penelitian ini bertujuan untuk memberikan wawasan strategis dalam pemilihan algoritma navigasi robot pada lingkungan dengan batasan sumber daya komputasi.

Sisa dari makalah ini disusun sebagai berikut: Bagian II membahas latar belakang teoritis dari strategi pathfinding. Bagian III merinci arsitektur simulasi dan metodologi eksperimental. Bagian IV menyajikan hasil dan analisis komparatif, dan Bagian V menyimpulkan temuan penelitian.

II. LATAR BELAKANG TEORITIS

A. Representasi Lingkungan Melalui *Costmap 2D* dan *Inflation Layer*

Dalam arsitektur navigasi ROS 2 Nav2, lingkungan fisik di sekitar robot direpresentasikan sebagai sebuah *Occupancy Grid Map* dua dimensi. Setiap sel dalam *grid* tersebut menyimpan nilai diskret antara 0 hingga 254 yang menunjukkan tingkat keterisian (*occupancy state*). Nilai 0 merepresentasikan ruang bebas (*free space*), 254 merepresentasikan rintangan mematikan (*lethal obstacle*), dan 253 menyatakan zona terlarang bagi pusat robot (*inscribed inflated obstacle*).

Proses penentuan jalur global tidak dapat mengabaikan dimensi fisik robot asli. Oleh karena itu, diterapkan mekanisme *Inflation Layer* yang secara matematis memancarkan nilai biaya dari pusat rintangan tegar ke sel-sel di sekitarnya menggunakan fungsi peluruhan eksponensial. Persamaan pembobotan nilai biaya ($cost$) pada sel berjarak (d) dari rintangan didefinisikan sebagai berikut:

$$cost = 252 * e^{-cost\_scaling\_factor * (d - inscribed\_radius)}$$

di mana *cost_scaling_factor* mengendalikan tingkat kecuraman penurunan biaya dan *inscribed_radius* adalah radius fisik robot. Melalui formula ini, algoritma pencarian rute tidak hanya melihat rintangan sebagai entitas biner (ada atau tidak ada), melainkan sebagai medan gradien biaya. Hal ini memaksa fungsi pencarian global untuk memprioritaskan sel dengan *cost* terendah demi menjaga margin keamanan aktuator robot fisik.

B. Karakteristik Mekanis *Min-Heap Priority Queue*

Ketiga algoritma yang dievaluasi dalam penelitian ini mengandalkan struktur data antrean prioritas (Priority Queue) jenis Min-Heap untuk mengelola daftar simpul terbuka (Open Set). Efisiensi algoritma perencanaan jalur sangat bergantung pada kompleksitas operasi pencarian nilai minimum pada antrean ini.

Pada setiap iterasi pencarian, simpul dengan nilai fungsi evaluasi $f(n)$ terkecil wajib diekstraksi dari puncak heap. Operasi pengambilan simpul minimum (pop) dan penyisipan simpul tetangga baru (push) ke dalam Min-Heap memiliki kompleksitas waktu sebesar $O(log\ N)$, di mana *N* adalah jumlah simpul yang berada dalam antrean terbuka. Perbedaan performa waktu komputasi antar-algoritma murni ditentukan oleh seberapa efisien fungsi evaluasi memangkas jumlah simpul (*N*) yang masuk ke dalam struktur heap tersebut.

C. A*lgoritma Dijkstra dan Uniform Cost Search (UCS)*

Algoritma Dijkstra merupakan strategi pencarian berbasis *uninformed search* yang menjamin penemuan rute terpendek dengan mengeksplorasi *node* berdasarkan akumulasi biaya terkecil dari titik awal [2]. Dalam konteks kecerdasan buatan dan navigasi, Dijkstra setara dengan *Uniform Cost Search* (UCS) jika biaya antar-simpul bersifat seragam atau bervariasi non-negatif [3]. Algoritma ini tidak menggunakan informasi estimasi jarak ke target (tanpa heuristik). Fungsi evaluasi untuk Dijkstra/UCS didefinisikan sebagai:

$$f(n) = g(n)$$

Di mana:

- $f(n)$ adalah total biaya evaluasi pada *node* (n).
- $g(n)$ adalah biaya aktual yang ditempuh dari *node* awal (*start_point*) menuju *node* (n) saat ini.

Algoritma ini bekerja dengan mengeksplorasi secara radial ke segala arah (melingkar), menjadikannya sangat andal untuk menjamin jalur terpendek, namun tidak efisien secara waktu dan ruang komputasi pada peta yang luas karena harus memeriksa banyak *node* yang tidak relevan.

D. *Greedy Best-First Search (GBFS)*

Greedy Best-First Search merupakan strategi *informed search* yang memprioritaskan kecepatan komputasi dengan mengeksplorasi jalur yang secara lokal dianggap paling dekat dengan target [4]. Berbeda dengan Dijkstra, GBFS sepenuhnya mengabaikan biaya masa lalu $g(n)$ dan hanya mengandalkan fungsi heuristik sebagai panduan arah. Fungsi evaluasi GBFS dirumuskan sebagai:

$$f(n) = h(n)$$

Di mana:

- $h(n)$ adalah fungsi heuristik, yaitu estimasi biaya atau jarak dari *node* (n) saat ini menuju titik target (*goal*).

Karena hanya mengejar nilai $h(n)$ terkecil, GBFS bekerja sangat cepat dan agresif mengarah langsung ke target. Namun, strategi *greedy* ini memiliki kelemahan fatal; algoritma tidak menjamin jalur terpendek (tidak optimal) dan rentan terjebak pada *local minima*, seperti rintangan berbentuk cekungan yang memaksa robot berputar jauh setelah buntu.

E. Algoritma A (A-Star)*

Algoritma A* dirancang untuk mengatasi kelemahan Dijkstra dan GBFS dengan menggabungkan keunggulan kedua strategi tersebut [5]. A* merupakan algoritma *informed search* yang optimal dan efisien karena memperhitungkan biaya aktual masa lalu sekaligus estimasi jarak masa depan. Fungsi evaluasi A* dinyatakan dalam persamaan:

$$f(n) = g(n) + h(n)$$

Dengan menggabungkan $g(n)$ dan $h(n)$, A* memastikan bahwa arah pencarian tetap fokus menuju target (berkat heuristik), tetapi tidak ceroboh mengambil jalur pintas yang mahal (berkat kontrol biaya aktual). A* dijamin menghasilkan jalur yang optimal jika fungsi heuristik yang digunakan bersifat *admissible* (tidak pernah menilai lebih tinggi dari biaya asli) dan *consistent*.

F. Perhitungan Fungsi Heuristik

Dalam sistem koordinat *grid* 2D (seperti visualisasi peta Bevy Engine dan Nav2 Costmap), terdapat dua fungsi heuristik utama yang umum diimplementasikan pada fungsi $h(n)$:

1. Euclidean Distance: Digunakan jika robot dapat bergerak bebas ke segala arah (sudut kontinu/bebas). Jarak dihitung berdasarkan garis lurus geometris menggunakan teorema Pythagoras:

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

2. Manhattan Distance: Digunakan jika pergerakan robot dibatasi hanya pada 4 arah mata angin (maju, mundur, kiri, kanan) di atas *grid*. Perhitungannya adalah absolute dari selisih koordinat:

$$h(n) = |x_{goal} - x_n| - |y_{goal} - y_n|$$

Pemilihan fungsi heuristik ini secara langsung mempengaruhi nilai $f(n)$ yang tercatat pada *Calculation Log* sistem navigasi saat mengevaluasi setiap nomor *node* di dalam antrian *Priority Queue*.

III. Arsitektur Simulasi Dan Metodologi

Bagian ini menguraikan arsitektur sistem *full-stack* BawalPathFinder yang digunakan sebagai lingkungan eksperimen. Sistem dirancang secara modular dengan memisahkan beban kerja komputasi navigasi robotik (*backend*) di dalam Docker Container dan antarmuka visualisasi performa tinggi (*frontend*) menggunakan Bevy Engine.

A. Lingkungan Container dan Standar Arsitektur ROS 2

Untuk menjamin reprodusibilitas penelitian, seluruh komponen komputasi ROS 2 Humble dan subsistem Nav2 diisolasi di dalam *Docker Container*. Struktur *repository backend* dikembangkan dengan mematuhi standar *best practice* ekosistem ROS 2, yang diorganisasikan ke dalam elemen-elemen berikut:

1. Package Manifest (package.xml): Mendefinisikan dependensi eksternal sistem seperti *rclcpp, nav2_core, std_msgs,* dan *nlohmann_json* untuk serialisasi data telemetri.
2. Sistem Kompilasi (CMakeLists.txt): Mengelola instruksi kompilasi untuk C++17, mengatur tautan pustaka dinamis (*shared libraries*), dan mengekspor kelas plugin agar dapat dikenali oleh *pluginlib* ROS 2.
3. Direktori Source (src/): Menjadi ruang lingkup isolasi bagi implementasi kode sumber C++ untuk masing-masing algoritma perencanaan jalur.
4. Launcher Node (launch/): Berisi skrip otomasi Python untuk menginisialisasi *map_server, lifecycle_manager*, dan peladen aksi *Nav2* secara sekuensial.

Komunikasi data antara *backend* ROS 2 dan *frontend* Bevy Engine dijembatani oleh protokol WebSocket asinkron melalui rosbridge_suite [8], memungkinkan transmisi *payload* JSON secara *bidirectional* tanpa hambatan blokir (*blocking*).

B. Pemodelan Ruang Keadaan dan Inflation Layer

Sebelum algoritma pencarian mengeksplorasi graf, Nav2 menerjemahkan peta biner statis menjadi representasi ruang rintangan *Costmap 2D* [6]. Di dalam memori utama, *costmap* ini disimpan sebagai larik satu dimensi (*1D array*) untuk mengoptimalkan efisiensi *cache* CPU (kompleksitas $O(1)$). Pemetaan indeks linier ke koordinat kartesian dua dimensi didefinisikan sebagai:

$$index = y * width + x$$

Selain pemetaan grid, ruang keadaan dimanipulasi secara dinamis oleh *InflationLayer* dalam nav2_params.yaml. Konfigurasi spesifik yang diterapkan adalah:

Parameter *inflation_radius: 0.45* menentukan jarak proksimitas di mana rintangan memancarkan hambatan. Sel bertepatan dengan rintangan bernilai *Lethal Cost* (254). Nilai hambatan menurun secara eksponensial di luar radius tersebut berdasarkan *cost_scaling_factor*. Manipulasi topografi ini memaksa algoritma pencarian untuk memperhitungkan *clearance* keamanan, mengubah definisi lintasan terpendek menjadi lintasan teraman. Jika target berada di dalam zona *lethal*, Nav2 secara otomatis membatalkan pencarian (*Planning Failed*).

C. Struktur Data dan Optimasi Antrian Prioritas

Strategi algoritma pencarian sangat bergantung pada mekanisme pemilihan simpul untuk diekspansi. Pengelolaan *Open Set* diimplementasikan menggunakan antrean prioritas (*Priority Queue*) bawaan dari *Standard Template Library* (STL) C++. Sebuah struktur data kustom *GridNode* dirancang untuk merangkum status evaluasi $f(n)$

Penggunaan *operator overloading* (>) memastikan bahwa *std::priority_queue* bertindak secara otomatis sebagai *Min-Heap*. Algoritma secara konsisten mengekstraksi simpul dengan nilai f(n) terkecil pada kompleksitas waktu logaritmik $O(\log V)$, menghindari pencarian linear $O(V)$yang menguras komputasi CPU.

D. Abstraksi dan Pewarisan Klas pada Nav2 Planner

Dalam membandingkan kinerja beberapa strategi algoritma secara objektif, sangat penting untuk memastikan bahwa setiap algoritma dieksekusi pada kondisi awal dan parameter lingkungan yang identik. Sistem ini mencapai hal tersebut melalui pemanfaatan arsitektur *polymorphism* yang memungkinkan injeksi algoritma kustom (A*, GBFS, Dijkstra) secara modular tanpa memodifikasi kode inti dari subsistem Nav2.

Secara arsitektural, setiap kelas algoritma diturunkan secara publik dari kelas abstrak *nav2_core::GlobalPlanner*. Pewarisan kelas (*class inheritance*) ini mewajibkan setiap kelas strategi pencarian untuk mengimplementasikan (*override*) metode virtual murni yang disediakan oleh *framework*. Fungsi paling krusial yang diimplementasikan secara mandiri oleh masing-masing kelas strategi adalah fungsi kalkulasi rute *createPlan()*.

Fungsi *createPlan()* menerima argumen berupa koordinat titik awal (*start*), koordinat titik tujuan (*goal*), serta pointer referensi menuju objek *Costmap2D*. Melalui antarmuka abstraksi ini, subsistem pengelola siklus hidup (*lifecycle manager*) Nav2 dapat memanggil fungsi *createPlan()* secara seragam, terlepas dari apakah yang sedang aktif adalah A*, GBFS, atau Dijkstra. Di dalam ruang lingkup fungsi inilah masing-masing algoritma mendefinisikan strategi evaluasi simpul dan antrean prioritasnya sendiri.

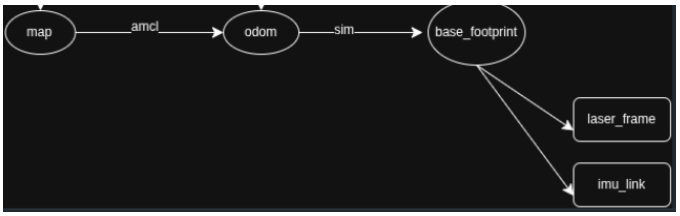
Penggunaan antarmuka abstrak yang konsisten ini mengisolasi proses algoritma dari intervensi eksternal dan menjamin bahwa ketiga strategi menerima masukan ruang keadaan (*costmap*) yang identik. Dengan demikian, setiap perbedaan performa (seperti waktu komputasi atau jumlah simpul yang diekspansi) yang tercatat pada sistem murni merupakan representasi dari efisiensi matematis algoritma tersebut, bukan diakibatkan oleh perbedaan perlakuan dari *framework* robotika.

E. Sinkronisasi Kinematika URDF dan Pohon Transformasi Spasial (TF2)

Untuk mengeksekusi lintasan hasil kalkulasi algoritma pencarian jalur (A*, GBFS, dan UCS) ke dalam pergerakan fisis, sistem navigasi membutuhkan kerangka acuan spasial yang konsisten. Hal ini dicapai dengan mengintegrasikan berkas deskripsi robot *Unified Robot Description Format* (URDF) dan subsistem pohon transformasi koordinat (*TF2 Tree*).

URDF mendefinisikan struktur kinematika robot fisis melalui representasi elemen kaki tegar (*Link*) dan sendi putar (*Joint*). Berdasarkan standar konvensi ROS 2, orientasi arah hadap (*heading*) robot searah dengan sumbu *X* positif (depan) dan sumbu *Y* positif (kiri).

Arah hadap robot di dalam ruang simulasi diekstrak secara kontinu melalui kalkulasi matriks transformasi homogen antar-kerangka acuan koordinat. Berdasarkan rekam struktur *TF Tree* sistem pada Gambar 3.1, rantai transformasi spasial diatur secara sekuensial dengan hierarki sebagai berikut:



Gambar 3.1. Diagram TF Tree

- Transformasi *map* ke *odom*: Dijembatani oleh subsistem lokalisasi amcl untuk mengompensasi penyimpangan (*drift*) posisi absolut robot di dalam peta global.
- Transformasi *odom* ke *base_footprint*: Dihitung oleh *node sim* (simulator) untuk menentukan estimasi posisi proyeksi dua dimensi sasis robot di atas permukaan tanah berdasarkan data *odometer* fisis.

- Transformasi *base_footprint* ke *laser_frame* dan *imu_link*: Menghubungkan titik koordinat sasis fisis terhadap posisi sensor *LiDAR* (*laser_frame*) dan sensor inersia (*imu_link*) sesuai dengan konstanta transformasi yang didefinisikan dalam berkas *URDF*.

Ketika kelas perencana jalur global mengekstrak titik-titik koordinat rute melalui fungsi *createPlan()*, luaran yang dikirimkan ke pengontrol lokal dikapsulasi dalam struktur data *geometry_msgs/msg/PoseStamped*. Setiap titik koordinat tidak hanya membawa posisi *(x, y),* melainkan juga nilai orientasi sudut rotasi (*yaw*) dalam format *Quaternion* untuk menghindari anomali *Gimbal Lock*.

Modul pengontrol lokal (*local controller*) *Nav2* secara kontinu membandingkan selisih orientasi *Quaternion* antara posisi base_footprint saat ini dengan target koordinat jalur terdekat berikutnya. Selisih sudut ini kemudian dikonversi menjadi perintah kecepatan sudut dinamis lewat topik /cmd_vel dalam format variabel kecepatan linier (*linear.x*) dan kecepatan angular (*angular.z*), yang mengontrol perbedaan traksi kecepatan roda kanan dan kiri pada model robot fisik.

F. Skenario Eksperimen dan Parameter Evaluasi

Untuk memenuhi standar pengujian algoritma komparatif yang empiris, ketiga strategi (*UCS/Dijkstra, GBFS,* dan *A**) dievaluasi pada lingkungan simulasi yang identik. Peta biner yang digunakan memiliki dimensi spasial 200 x 200 sel grid dengan topologi rintangan statis yang dirancang untuk menguji kelemahan masing-masing algoritma (seperti rintangan berbentuk cekungan atau lorong sempit).

Pengujian dilakukan dengan menetapkan koordinat awal (*start pose*) dan titik tujuan (*goal pose*) yang konstan untuk setiap iterasi eksperimen. Parameter *Inflation Layer* dikunci secara seragam guna memastikan bobot lintasan (*costmap*) tidak mengalami mutasi antar pengujian. Evaluasi kinerja algoritma didasarkan pada tiga metrik utama:

1. Waktu Komputasi (*Execution Time*): Durasi total (dalam milidetik) yang dibutuhkan algoritma dari inisialisasi awal hingga lintasan berhasil dikalkulasi atau dinyatakan gagal.
2. Efisiensi Ruang Pencarian (*Nodes Explored*): Total kumulatif simpul yang diekspansi dan dimasukkan ke dalam daftar tertutup (*closed list*) atau rekam *Calculation Log*. Metrik ini mengukur beban memori dari strategi pencarian.
3. Kualitas Lintasan (*Path Optimality*): Total jarak lintasan yang dihasilkan (dalam satuan meter). Metrik ini memvalidasi jaminan teoretis apakah algoritma mengembalikan solusi optimal atau suboptimal (*local minima*).

G. Ketersediaan Kode dan Reprodusibilitas Pengujian

Untuk mendukung prinsip reprodusibilitas ilmiah (scientific reproducibility), artefak perangkat lunak yang dikembangkan dalam eksperimen ini—termasuk berkas konfigurasi *Docker Container*, implementasi plugin *C++* dari ketiga algoritma perencana jalur global, serta arsitektur *frontend visualizer* berbasis *Bevy Engine*—telah dipublikasikan di bawah lisensi terbuka. Proyek utuh beserta dokumentasi replikasi lingkungan simulasi dapat diakses melalui repositori *GitHub*[9]

IV. Hasil Pengujian Dan Analisis

Bagian ini menguraikan hasil eksperimen komparatif secara empiris berdasarkan data visual dan telemetri dari kerangka BawalPathFinder. Pengujian difokuskan pada skenario peta dengan rintangan kompleks untuk mengamati secara langsung perilaku evaluasi simpul dari setiap algoritma (UCS, GBFS, dan A*).

A. Skenario Pengujian

Pengujian dirancang dalam dua lingkungan *grid* dua dimensi dengan karakteristik halangan statis yang berbeda, sebagaimana direpresentasikan pada Gambar 4.1 dan Gambar 4.2. Peta pertama dirancang dengan rintangan asimetris menyudut untuk menguji ketahanan algoritma terhadap kondisi jebakan *local minimal*. Sementara itu, peta kedua dirancang dengan struktur halangan yang lebih kompleks untuk menguji batas maksimal ekspansi simpul (*search space*) dari masing-masing antrian prioritas algoritma.

Sebagai catatan teknis, penamaan identitas atau judul peta pada antarmuka sistem dibangkitkan secara dinamis menggunakan pustaka *faker* (penghasil data acak). Oleh karena itu, nama peta yang mungkin terekam pada log sistem murni berfungsi sebagai pengenal sesi (*session identifier*) sementara dan tidak mendeskripsikan topologi fisik peta secara harfiah.

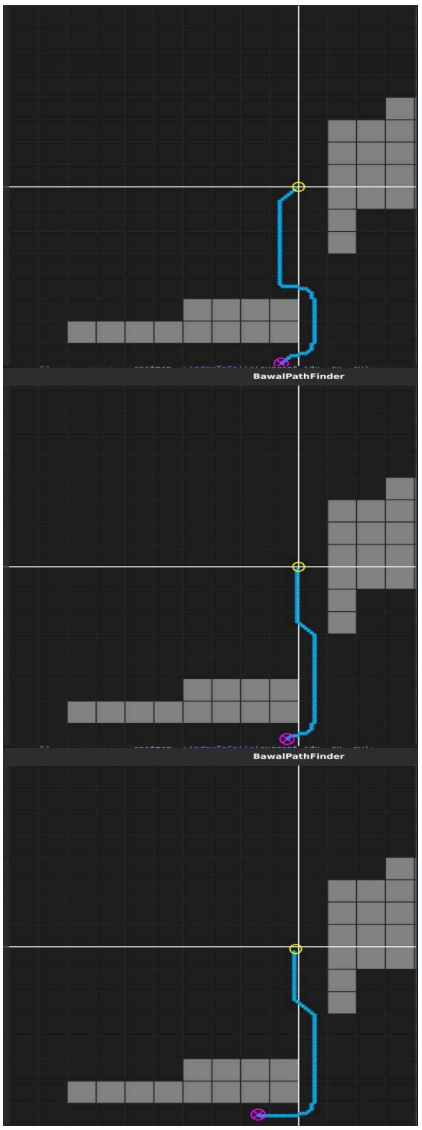
Gambar 4.1. Topologi rintangan asimetris (Peta 1) untuk pengujian *local minimal*.

Gambar 4.2. Topologi rintangan kompleks (Peta 2) untuk pengujian beban memori.

Untuk memastikan validitas komparasi, parameter *Costmap 2D* dikunci secara konstan pada kedua skenario pengujian. Nilai proksimitas hambatan diatur seragam dengan menetapkan *cost_scaling_factor* pada 10.0 dan *inflation_radius* pada jarak 0,25 meter, memastikan bahwa setiap algoritma mengevaluasi ruang keadaan (*state space*) yang identik.

B. Hasil Pengujian Skenario Peta 1 (Halangan Asimetris)

Pada skenario pertama, titik awal dan tujuan dipisahkan oleh struktur dinding menyudut. Berdasarkan log visualisasi, algoritma UCS (Dijkstra) yang beroperasi tanpa heuristik (f(n) = g(n)) berhasil menemukan jalur paling efisien secara geometris, menghasilkan panjang lintasan terpendek yaitu 7,11 m yang dibentuk oleh 131 *Path Nodes*. Jalur UCS memotong sudut rintangan secara agresif dengan mengevaluasi simpul secara merata.



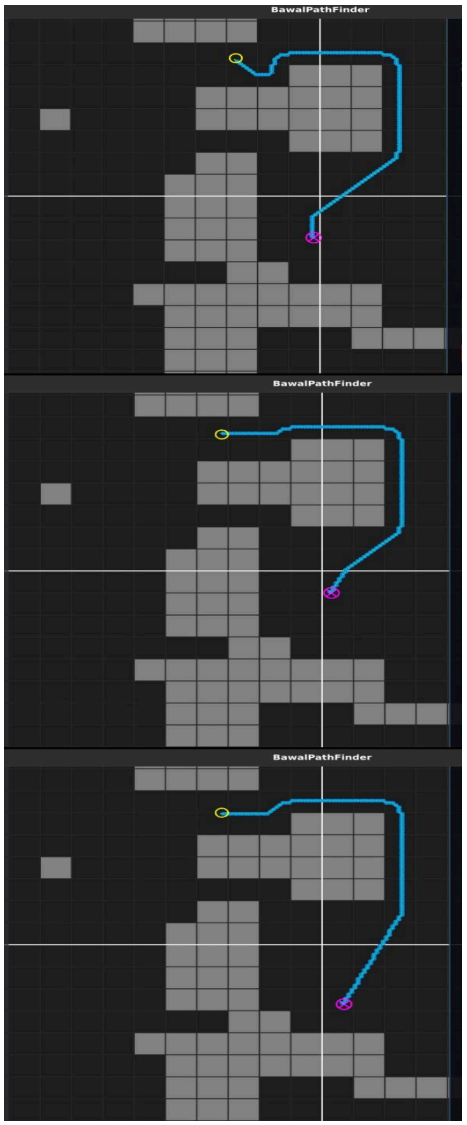
Gambar 4.3. Komparasi visual lintasan UCS, GBFS, dan A pada Peta 1.*

Sebaliknya, strategi GBFS yang hanya mengandalkan proksimitas heuristik $f(n) = h(n)$ menunjukkan pergerakan linier yang agresif di awal, namun terpaksa melambung jauh menyusuri dinding luar rintangan. Hal ini menghasilkan jalur suboptimal sepanjang 7,85 m dengan 140 *Path Nodes*. Algoritma A* $f(n) = g(n) + h(n)$ menyeimbangkan kedua metrik tersebut, menghasilkan jalur aman sepanjang 7,94 m dengan 148 *Path Nodes*. A* mengambil rute melambung yang lebih lebar dibandingkan UCS untuk menghindari penetrasi zona hambatan tegar.

C. Hasil Pengujian Skenario Peta 2 Rintangan Kompleks

Pada skenario kedua, kompleksitas ruang keadaan ditingkatkan. Topologi peta memaksa algoritma untuk

bermanuver melewati lorong sempit yang memaksa sistem mencari celah terkecil guna menghindari akumulasi biaya tinggi dari *inflation layer* di sekitar rintangan tegar. Hasil komparasi visual pada Gambar 4 memperlihatkan batasan visualisasi sebaran simpul yang diekstrak.



Gambar 4.4. Komparasi visual lintasan UCS, GBFS, dan A pada Peta 2.*

Pada pengujian ini, algoritma UCS menghasilkan lintasan sepanjang 7,11 m dengan meng ekspansi 131 *Path Nodes*. Karakteristik ekspansi radial UCS memaksa sistem memeriksa seluruh ruang bebas di sekitar rintangan menyudut, yang secara konsisten menjamin penemuan rute terpendek namun membebani memori antrian prioritas.

Algoritma GBFS menghasilkan lintasan sepanjang 7,85 m dengan 140 *Path Nodes*, kembali gagal menemukan rute terpendek karena sifat *greedy* lokal yang memaksanya menempel pada dinding luar rintangan demi mengejar target geometris terdekat.

Sementara itu, algoritma A* memitigasi kelemahan tersebut dengan mencetak lintasan sepanjang 7,94 m melalui 148 *Path Nodes*. Integrasi bobot heuristik dan biaya aktual pada A* menghasilkan kompromi yang seimbang, di mana lintasan akhir sengaja mengambil radius putar yang sedikit lebih lebar guna menjaga jarak aman dari sudut tajam rintangan *costmap*.

D. Analisis Tabel Komparasi dan data Telemetri

Seluruh parameter luaran utama dari pengujian dua peta ini dikompilasikan secara terpadu ke dalam Tabel I untuk mempermudah analisis komparatif performa makro.

Algoritma	Skenario Peta	Total Path Nodes	Total Jarak (m)	Konfigurasi Evaluasi
UCS	Peta 1 (Halangan Asimetris)	82	4,26	F = G
	Peta 2 (Rintangan Kompleks)	131	7,11	F = G
GBFS	Peta 1 (Halangan Asimetris)	91	4,86	F = H
	Peta 2 (Rintangan Kompleks)	140	7,85	F = H
A*	Peta 1 (Halangan Asimetris)	91	4,71	F = G + H

	Peta 2 (Rintangan Kompleks)	148	7,94	F = G + H

TABLE I. KOMPARASI PERFORMA JALUR AKHIR LINTAS SKENARIO

Algoritma	Node Index	F-Cost	G-Cost	H-Cost
UCS	N[4897]	86.0	83.1	-
	N[6508]	87.1	70.7	-
	N[15881]	86.1	27.5	-
	N[4896]	86.4	84.1	-
	N[4696]	86.0	84.6	-
GBFS	N[6105]	15.6	-	15.6
	N[8706]	26.4	-	26.4
	N[11099]	35.5	-	35.5
	N[15694]	58.0	-	58.0
	N[4095]	2.0	-	2.0
A*	N[18674]	96.8	26.9	69.9
	N[5305]	95.0	74.9	20.1
	N[21504]	96.6	11.5	85.1
	N[6910]	95.9	69.0	26.9
	N[4885]	94.6	94.6	0.0

TABLE II. LOG KALKULASI NAVIGASI LIMA SIMPUL TERAKHIR PADA PETA I

Algoritma	Node Index	F-Cost	G-Cost	H-Cost
UCS	N[22313]	144.2	119.6	-
	N[22113]	142.9	119.2	-
	N[37890]	144.3	43.6	-
	N[20104]	144.4	133.2	-
	N[18706]	142.4	136.8	-
GBFS	N[16496]	2.2	-	2.2
	N[16698]	3.2	-	3.2
	N[17896]	9.1	-	9.1

Algoritma	Node Index	F-Cost	G-Cost	H-Cost
	N[18701]	13.6	-	13.6
	N[19506]	19.2	-	19.2
A*	N[23731]	160.0	107.6	52.4
	N[27533]	159.8	89.5	70.4
	N[20028]	160.9	78.1	82.9
	N[25732]	159.8	98.0	61.7
	N[14707]	159.0	157.6	1.4

TABLE III. LOG KALKULASI NAVIGASI LIMA SIMPUL TERAKHIR PADA PETA II

V. Kesimpulan

Berdasarkan data makro yang dihimpun pada Tabel I serta rekam telemetri mikro pada Tabel II dan Tabel III, terdapat korelasi fisis yang kuat antara formulasi fungsi evaluasi f(n) dengan perilaku perencana jalur (*global planner*) dalam mengeksplorasi ruang keadaan *costmap*.

1. *Analisis Performa Makro dan Geometris Jalur:* Data pada Tabel I membuktikan jaminan teoretis bahwa *Uniform Cost Search* (UCS) selalu menghasilkan lintasan dengan kualitas paling optimal secara absolut (4,26 meter pada Peta 1 dan 7,11 meter pada Peta 2). Efek ketiadaan fungsi heuristik ($h(n) = 0$) memaksa UCS mengekspansi simpul secara radial melingkar. Secara geometris, hal ini memberikan keuntungan fisis di mana UCS mampu mendeteksi celah koordinat terkecil yang mepet dengan sudut rintangan statis. Namun, kompensasi dari optimalitas ini adalah membengkaknya beban memori karena *Min-Heap* harus menampung seluruh sel bebas sebelum target diekstrak.

Sebaliknya, *Greedy Best-First Search* (GBFS) menunjukkan anomali performa yang kaku. Pada Peta 1, GBFS menghasilkan jarak 4,86 meter (lebih panjang 0,6 meter dari UCS). Hal ini disebabkan oleh sifat *greedy* lokal yang hanya mengevaluasi jarak Euclidean langsung ke target ($h(n)$). Ketika dihadapi rintangan asimetris menyudut, antrean prioritas GBFS langsung terjebak pada *local minima* (cekungan halangan). Karena mengabaikan biaya akumulasi masa lalu ($g(n) = 0$), GBFS terpaksa berputar jauh menyusuri dinding luar rintangan costmap demi menurunkan nilai h(n) secara gradual.

Algoritma A* terbukti bertindak sebagai solusi kompromi paling efisien (*sub-optimal balanced*). Pada Peta 1, A* menghasilkan lintasan sepanjang 4,71 meter dengan jumlah simpul ekspansi (*Path Nodes*) sebanyak 91 simpul, menyamai efisiensi ruang pencarian GBFS namun dengan kualitas lintasan yang mendekati batas optimalitas UCS. Integrasi fungsi $f(n) = g(n) + h(n)$ mencegah A* untuk bertindak ceroboh memotong jalur berbahaya, sekaligus mencegah ekspansi radial yang sia-sia ke arah yang berlawanan dari target.

2. *Analisis Mikro Kalkulasi Log Lima Simpul Terakhir:* Melalui pencatatan instan *Calculation Log* pada Tabel II (Peta 1) dan Tabel III (Peta 2), perbedaan mekanis internal dari ketiga algoritma saat mendekati titik tujuan (*goal pose*) dapat dibedah secara transparan:

- Pada Blok Data UCS: Nilai *F-Cost* tercatat kembar identik dengan nilai *G-Cost*, sementara kolom *H-Cost* bernilai kosong (-). Hal ini mengonfirmasi secara empiris bahwa prioritas ekstraksi simpul pada *Min-Heap* UCS murni dikendalikan oleh fungsi akumulasi jarak internal dari titik mula. Simpul terakhir yang diekstrak sebelum target pada Peta 2 (N[18706]) mencatatkan biaya aktual *G* sebesar 136,2, merepresentasikan total bobot penalti *costmap* yang telah dilewati oleh robot sepanjang lorong sempit.
- Pada Blok Data *GBFS*: Nilai F-Cost selalu sama dengan nilai H-Cost, sedangkan kolom G-Cost tidak terdefinisi (-). Karakteristik ini memperlihatkan sifat *greedy* yang sangat agresif. Perhatikan simpul terakhir pada Peta I (N[4095]) dan Peta II (N[16496]), nilai *F* meluncur turun drastis hingga

menyentuh angka 2.0 dan 2.2. Angka kecil ini membuktikan bahwa pada iterasi akhir, antrean prioritas GBFS hanya didorong oleh jarak spasial sisa menuju target tanpa memedulikan seberapa mahal atau seberapa jauh robot telah berputar di belakang.

- Pada Blok Data *A**: Kolom telemetri menyajikan data paling ideal di mana komponen biaya historis (G) dan komponen estimasi geometris (H) berpadu secara dinamis. Pada Tabel II simpul N[4885], tercatat nilai $G = 94.6$ dan $H = 0.0$ dengan total $F = 94.6$. Nilai $H = 0.0$ menandakan simpul tersebut telah berimpit atau berada tepat pada koordinat *goal pose* (jarak sisa ke target sama dengan nol). Sebaliknya, pada simpul awal di antrean tersebut (N[18674]), nilai H masih tinggi (69.9) karena simpul berada di luar jangkauan langsung tujuan, namun diseimbangkan oleh nilai G yang rendah (26.9).

Fenomena fluktuasi nilai komponen pada blok *A** ini membuktikan adanya mekanisme penyeimbang adaptif: ketika posisi simpul masih jauh dari target, komponen heuristik H mendominasi jalannya arah pencarian; namun ketika simpul mulai memasuki lorong sempit berbiaya tinggi, komponen biaya aktual G mengambil alih kendali prioritas untuk memastikan robot tidak menabrak batas zona *inflation radius* (253) yang dapat memicu kegagalan sistem aksi *Nav2*.

Lampiran

Repository Program:
https://github.com/fariswirakusuma/BawalPathFinder.

Video Link at Youtube

https://www.youtube.com/watch?v=fu9y66EK7HA

Acknowledgment

Terima kasih banyak kepada semua asisten dan dosen yang telah tulus untuk mengajari serta mengarahkan mata kuliah ini sehingga saya dapat mengerjakan makalah ini dengan sebaik-baiknya

References

[1] Open Source Robotics Foundation, "ROS 2 Documentation: Humble Hawksbill," [Online]. Available: https://docs.ros.org/en/humble/index.html.

[2] E.CP-Algorithms, "Dijkstra Algorithm," [Online]. Available: https://cp-algorithms.com/graph/dijkstra.html.

[3] GeeksforGeeks, "Uniform Cost Search (UCS) in AI," [Online]. Available: https://www.geeksforgeeks.org/artificial-intelligence/uniform-cost-search-ucs-in-ai/.

[4] GeeksforGeeks, "Greedy Best-First Search Algorithm," [Online]. Available: https://www.geeksforgeeks.org/dsa/greedy-best-first-search-algorithm/.

[5] GeeksforGeeks, "A* Search Algorithm," [Online]. Available: https://www.geeksforgeeks.org/dsa/a-search-algorithm/.

[6] Open Navigation LLC, "Nav2 Documentation," [Online]. Available: https://docs.nav2.org/.

[7] Bevy Contributors, "Bevy Engine API Documentation," [Online]. Available: https://docs.rs/bevy/latest/bevy/index.html.

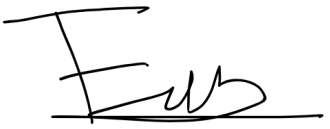
[8] ROS Wiki, "rosbridge_suite," [Online]. Available: https://wiki.ros.org/rosbridge_suite.

[9] fariswirakusuma,, "BawalPathFinder: Robot Navigation Simulation System using ROS 2 and Bevy Engine," GitHub Repository, 2026. [Online]. Available https://github.com/fariswirakusuma/BawalPathFinder.

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, 19 Juni 2026



Faris Wirakusuma Triawan
13524130