# Analisis Performansi Algoritma Pencarian Heuristik pada Nav2 Planner Server dalam Sistem Navigasi Otonom Robot Berbasis ROS 2

Muhammad Nur Majiid
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung*
Bandung, Indonesia
13524028@std.stei.itb.ac.id, muhammadnurmajiid22@gmail.com

***Abstract*—Perencanaan lintasan merupakan komponen kritis dalam sistem navigasi otonom robot. Penelitian ini menganalisis performansi tiga algoritma pencarian heuristik, yaitu Uniform Cost Search (UCS), Greedy Best-First Search (GBFS), dan A* (A-Star), yang diimplementasikan pada ekosistem ROS 2 dengan komponen Nav2 Planner Server. Eksperimen dilakukan pada tiga peta lingkungan sintetis dengan kompleksitas berbeda menggunakan metrik waktu komputasi, panjang lintasan, dan jumlah node yang diekspansi. Hasil menunjukkan bahwa UCS secara konsisten menghasilkan lintasan optimal namun dengan waktu komputasi dan jumlah node ekspansi tertinggi. GBFS memiliki waktu komputasi tercepat dan node ekspansi paling sedikit, tetapi sering kali menghasilkan lintasan suboptimal. A* menyeimbangkan kedua aspek tersebut dengan menghasilkan lintasan optimal secara signifikan lebih cepat daripada UCS. Implementasi ini mendemonstrasikan aplikasi strategi algoritma dalam domain robotika dan memberikan rekomendasi pemilihan algoritma berdasarkan karakteristik lingkungan dan kebutuhan misi.**

***Keywords*—A*, UCS, Greedy Best-First Search, Nav2, ROS 2, path planning, mobile robot**

## I. Pendahuluan

Kemajuan dalam bidang robotika otonom telah mendorong pengembangan sistem navigasi yang mampu beroperasi secara mandiri di berbagai lingkungan. Salah satu fondasi penting dalam sistem tersebut adalah kemampuan perencanaan lintasan (*path planning*) yang efisien dan optimal. Dalam konteks ini, algoritma pencarian menjadi tulang punggung penentuan rute dari posisi awal menuju tujuan sambil menghindari obstacle.

ROS 2 (Robot Operating System 2) merupakan kerangka kerja standar industri yang menyediakan abstraksi perangkat keras, library, dan tools untuk pengembangan aplikasi robotika [1]. Nav2 (Navigation 2) adalah stack navigasi utama untuk ROS 2 yang menyediakan modul-modul perencanaan, termasuk *Planner Server* yang bertanggung jawab menghasilkan lintasan global [2]. Secara bawaan, Nav2 menyediakan `NavFnPlanner` yang mendukung algoritma Dijkstra (ekivalen UCS pada grid dengan biaya seragam) dan A* [3]. Sayangnya, implementasi murni Greedy Best-First Search (GBFS) belum tersedia dalam stack tersebut.

Penelitian ini bertujuan menganalisis dan membandingkan performansi tiga algoritma pencarian heuristik UCS, GBFS, dan A* dalam konteks perencanaan lintasan robot mobile berbasis grid. Kontribusi utama dari makalah ini meliputi: (1) implementasi modular ketiga algoritma sebagai *custom planner node* yang terintegrasi dengan Nav2 Map Server dan RViz; (2) pembangkitan tiga peta uji sintetis dengan karakteristik berbeda; (3) eksperimen sistematis dengan metrik kuantitatif; serta (4) analisis komparatif yang memberikan rekomendasi pemilihan algoritma berdasarkan trade-off antara optimalitas dan efisiensi komputasi.

## II. Landasan Teori

### A. Uniform Cost Search (UCS)

UCS merupakan varian dari algoritma Dijkstra yang beroperasi pada struktur data graph berbobot [4]. Algoritma ini selalu mengekspansi node dengan nilai *cost* kumulatif $g(n)$ terendah dari node awal. Fungsi evaluasinya sangat sederhana:

$$f(n) = g(n) \tag{1}$$

UCS menjamin ditemukannya lintasan optimal selama biaya setiap langkah non-negatif. Kelemahan utamanya terletak pada tidak adanya informasi heuristik menuju goal, sehingga eksplorasi cenderung menyebar ke segala arah (*uninformed search*). Akibatnya, jumlah node ekspansi menjadi sangat besar terutama pada ruang pencarian yang luas. Dalam konteks grid 2D dengan konektivitas 8-arah, UCS akan mengeksplorasi secara melingkar dari titik start hingga mencapai goal.

### B. Greedy Best-First Search (GBFS)

GBFS termasuk dalam kategori *informed search* yang memanfaatkan fungsi heuristik $h(n)$ untuk memperkirakan biaya dari node $n$ menuju goal [4]. Berbeda dengan UCS, GBFS hanya mempertimbangkan nilai heuristik:

$$f(n) = h(n) \tag{2}$$

Karena tidak memperhitungkan biaya yang telah ditempuh $g(n)$, GBFS bersifat *greedy*, selalu memilih node yang *kelihatan* paling dekat dengan tujuan. Pendekatan ini membuat GBFS sangat cepat dalam banyak kasus dengan jumlah

node ekspansi minimal. Akan tetapi, GBFS tidak menjamin optimalitas karena dapat terjebak pada lokal minimum atau menghasilkan lintasan yang jauh lebih panjang dari optimal. Pada lingkungan dengan banyak obstacle dan jalan buntu, risiko suboptimalitas GBFS meningkat secara signifikan.

### C. A* (A-Star)

A* adalah algoritma pencarian heuristik yang menggabungkan keunggulan UCS dan GBFS dengan fungsi evaluasi:

$$f(n) = g(n) + h(n) \tag{3}$$

Jika fungsi heuristik $h(n)$ bersifat *admissible* (tidak pernah melebih biaya sebenarnya), A* menjamin optimalitas lintasan sekaligus lebih efisien daripada UCS. Heuristik membantu memfokuskan eksplorasi ke arah goal tanpa mengorbankan jaminan optimalitas [5]. Dalam implementasi grid 2D dengan konektivitas 8-arah, heuristik Euclidean bersifat admissible. A* secara luas digunakan dalam aplikasi robotika, termasuk dalam Nav2 Planner Server. Keunggulannya terletak pada kemampuan mengurangi ruang pencarian secara dramatis dibandingkan UCS tanpa harus kehilangan jaminan optimalitas.

### D. Nav2 dan ROS 2

Nav2 adalah stack navigasi untuk ROS 2 yang mengadopsi arsitektur Behavior Tree (BT) untuk orkestrasi komponen-komponen navigasi [2]. Komponen utama Nav2 meliputi Map Server (penyedia peta), Planner Server (perencanaan lintasan global), Controller Server (kontrol lokal), serta Behavior Tree Navigator. Planner Server memungkinkan penggunaan beberapa *plugin* planner secara dinamis melalui konfigurasi parameter ROS. Peta lingkungan direpresentasikan dalam format Occupancy Grid, di mana setiap sel menyimpan nilai probabilitas okupansi pada rentang $[0, 100]$ atau $-1$ untuk area yang tidak diketahui.

Nav2 menggunakan standar komunikasi topic dan action server dalam ROS 2. Planner Server bawaan mengimplementasikan action `ComputePathToPose` yang menerima pose awal dan pose tujuan, kemudian menghasilkan lintasan global. Dalam penelitian ini, *custom planner node* dirancang agar kompatibel dengan standar pesan Nav2, yaitu menerima peta dari `nav_msgs/OccupancyGrid` dan menghasilkan `nav_msgs/Path`.

### E. Kompleksitas Algoritmik

Secara teoritis, dalam kasus terburuk ketiga algoritma memiliki kompleksitas waktu eksponensial $O(b^d)$ terhadap faktor percabangan $b$ dan kedalaman solusi $d$. Namun dalam praktiknya pada grid terbatas dengan struktur heap, kompleksitasnya jauh lebih baik:

- **UCS**: memiliki kompleksitas waktu $O(|V| \log |V|)$ dan ruang $O(|V|)$, di mana $|V|$ adalah jumlah sel bebas. Algoritma ini harus menyimpan informasi *cost* untuk setiap node yang pernah dikunjungi.
- **GBFS**: secara asymptotic juga $O(|V| \log |V|)$, namun dengan heuristik yang informatif, jumlah node aktual yang diekspansi dalam praktik jauh lebih kecil daripada UCS.
- **A***: kompleksitas waktu $O(|V| \log |V|)$ dengan ruang $O(|V|)$. Efisiensi relatif sangat bergantung pada kualitas heuristik; dengan heuristik admissible yang informatif, A* dapat mendekati kompleksitas linier terhadap panjang lintasan optimal.

Perbedaan fundamental terletak pada *effective branching factor* yang sebenarnya. UCS mengeksplorasi node tanpa diskriminasi arah, sehingga *effective branching factor*-nya mendekati $b$ penuh. Sebaliknya, A* dan GBFS memangkas cabang-cabang yang tidak menjanjikan berdasarkan heuristik, sehingga hanya sebagian kecil dari $|V|$ yang benar-benar diekspansi. Tabel I merangkum perbandingan karakteristik teoritis ketiga algoritma.

TABLE I
PERBANDINGAN KARAKTERISTIK ALGORITMA SECARA TEORITIS

| Karakteristik | UCS | GBFS | A* |
|---|---|---|---|
| Optimalitas | Ya | Tidak | Ya (jika $h$ admissible) |
| Completeness | Ya | Ya | Ya |
| Waktu (praktik) | Lambat | Sangat cepat | Sedang |
| Node ekspansi | Banyak | Sedikit | Sedang |
| Memori | $O(\|V\|)$ | $O(\|V\|)$ | $O(\|V\|)$ |
| Informasi heuristik | Tidak | Ya | Ya |

## III. METODOLOGI

### A. Arsitektur Sistem

Sistem eksperimen dibangun dalam ekosistem ROS 2 Humble dengan komponen berikut:

- **Map Server** (`nav2_map_server`): mempublikasikan peta dalam format `nav_msgs/OccupancyGrid` berdasarkan berkas PGM dan YAML. Lifecycle manager digunakan untuk mengaktifkan Map Server secara otomatis.
- **Custom Planner Node**: mengimplementasikan UCS, GBFS, dan A* pada grid 2D. Node ini berlangganan ke `/map`, menerima permintaan perencanaan lintasan melalui topik `/planner_request`, dan mempublikasikan lintasan ke `/plan` serta metrik ke `/planner_metrics`.
- **Benchmark Node**: mengelola skenario uji, mengirimkan pasangan *start-goal* secara acak pada area bebas, dan mencatat hasil metrik dalam berkas CSV untuk analisis lebih lanjut.
- **RViz2**: untuk visualisasi peta, lintasan hasil perencanaan, serta posisi *start* dan *goal* dalam *frame* `map`.

Arsitektur ini dirancang agar *custom planner node* dapat menggantikan peran Nav2 Planner Server bawaan dalam konteks eksperimen perbandingan algoritma, dengan tetap menggunakan interface dan standar data Nav2. Pendekatan berbasis topic dipilih untuk menghindari kompleksitas pembuatan custom action interface, sehingga fokus pengembangan tetap pada algoritma pencarian itu sendiri. Alur data dimulai dari Map Server yang memuat peta dari berkas PGM dan YAML,

kemudian mempublikasikannya ke topik `/map`. Custom Planner Node berlangganan ke topik tersebut untuk memperoleh representasi grid lingkungan. Benchmark Node mengirimkan permintaan perencanaan melalui `/planner_request` yang berisi koordinat start, goal, dan nama algoritma. Custom Planner Node kemudian mengeksekusi algoritma yang diminta dan mempublikasikan hasil lintasan ke `/plan` serta metrik ke `/planner_metrics`. RViz2 berlangganan ke `/map` dan `/plan` untuk menampilkan visualisasi secara real-time.

### *B. Implementasi Algoritma*

Ketiga algoritma diimplementasikan menggunakan struktur data *priority queue* (heap) dengan konektivitas 8-arah pada grid. Biaya pergerakan antara sel tetangga didefinisikan sebagai $1.0 \times$ resolusi untuk arah horizontal atau vertikal dan $\sqrt{2} \times$ resolusi untuk arah diagonal. Fungsi heuristik yang digunakan adalah jarak Euclidean yang bersifat admissible:

$$h(n) = \sqrt{(x_n - x_{goal})^2 + (y_n - y_{goal})^2} \times \text{resolusi} \quad (4)$$

Pseudocode umum ketiga algoritma disajikan pada Algoritma 1. Perbedaan utama terletak pada fungsi prioritas, yaitu $f(n) = g(n)$ untuk UCS, $f(n) = h(n)$ untuk GBFS, dan $f(n) = g(n) + h(n)$ untuk A*. Struktur `came_from` digunakan untuk rekonstruksi lintasan setelah goal ditemukan. Pada setiap iterasi, node dengan nilai $f(n)$ terkecil diekstrak dari *open set*. Jika node tersebut merupakan goal, proses pencarian berhenti dan lintasan direkonstruksi menggunakan struktur `came_from`. Jika bukan goal, semua tetangga yang tidak terhalang dievaluasi. Jika tetangga belum pernah dikunjungi atau ditemukan dengan *cost* yang lebih rendah, nilai $g(n)$, $h(n)$, dan $f(n)$ diperbarui serta tetangga tersebut dimasukkan ke *open set*.

**Algorithm 1** Pencarian Lintasan pada Grid

**Require:** Grid $G$, start $s$, goal $g$, algoritma $alg$
**Ensure:** Path $P$ atau `FAILURE`

*open* $\leftarrow$ priority queue$\{(0, s)\}$
*came_from*$[s] \leftarrow$ null, *cost*$[s] \leftarrow 0$
*closed* $\leftarrow \emptyset$
**while** *open* $\neq \emptyset$ **do**
  $f, n \leftarrow$ pop(*open*)
  **if** $n \in$ *closed* **then continue**
  **end if**
  *closed* $\leftarrow$ *closed* $\cup \{n\}$
  **if** $n = g$ **then break**
  **end if**
  **for** each neighbor $m$ of $n$ **do**
    **if** $m$ occupied **then continue**
    **end if**
    $g_{new} \leftarrow$ *cost*$[n]$ + step_cost$(n, m)$
    **if** $m \notin$ *cost* $\vee$ $g_{new} <$ *cost*$[m]$ **then**
      *cost*$[m] \leftarrow g_{new}$
      $h \leftarrow$ heuristic$(m, g)$
      $f \leftarrow$ priority$(g_{new}, h, alg)$
      push(*open*, $(f, m)$)
      *came_from*$[m] \leftarrow n$
    **end if**
  **end for**
**end while**
**return** reconstruct(*came_from*, s, g)

### *C. Peta Uji*

Tiga peta sintetis dibangkitkan dengan resolusi $0.05$ m/pixel untuk merepresentasikan skenario navigasi yang beragam:

1) **Simple Maze**: labirin sederhana dengan koridor lebar dan sedikit percabangan, merepresentasikan lingkungan koridor indoor yang terstruktur.
2) **Office-like**: ruangan dengan banyak obstacle tersebar (meja, kursi) yang diselingi koridor lebar, merepresentasikan ruang kerja kantor dengan tingkat okupansi sedang.
3) **Complex Maze**: labirin kompleks yang dibangkitkan dengan algoritma *recursive backtracker*, memiliki banyak jalan buntu, percabangan, dan koridor sempit.

Setiap peta memiliki ukuran $200 \times 200$ piksel ($10 \times 10$ m) untuk Simple Maze dan Office-like, serta $400 \times 400$ piksel ($20 \times 20$ m) untuk Complex Maze. Pemilihan ukuran yang berbeda untuk Complex Maze dimaksudkan untuk menguji skalabilitas algoritma pada ruang pencarian yang lebih luas. Visualisasi ketiga peta uji ditunjukkan pada Gambar 1.

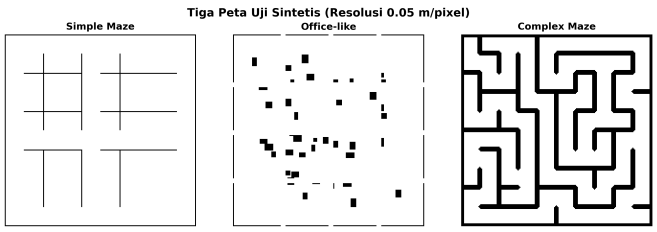


Fig. 1. Visualisasi tiga peta uji sintetis: (a) Simple Maze, (b) Office-like, dan (c) Complex Maze. Area hitam merepresentasikan obstacle, sedangkan area putih merepresentasikan area bebas.

### D. Skenario dan Metrik Eksperimen

Pada setiap peta, dilakukan 5 pasangan *start-goal* yang dipilih secara acak pada sel bebas. Ketiga algoritma diuji pada setiap pasangan, sehingga terdapat 15 percobaan per peta. Metrik yang dicatat meliputi:

1) **Waktu komputasi** (ms): waktu yang dibutuhkan algoritma dari inisialisasi hingga goal ditemukan, diukur menggunakan `time.perf_counter()` pada Python.
2) **Panjang lintasan** (m): panjang total lintasan yang dihasilkan dari start ke goal, dihitung dengan menjumlahkan jarak Euclidean antar sel bertetangga pada lintasan.
3) **Jumlah node ekspansi**: banyaknya sel grid yang dikunjungi (dimasukkan ke *closed set*) selama proses pencarian, sebagai indikator beban komputasi.

### E. Desain Eksperimen

Untuk memastikan validitas hasil, setiap pasangan *start-goal* dipilih secara acak menggunakan *uniform random sampling* dengan pengecekan okupansi. Posisi yang terlalu dekat ($< 0.5$ m) atau berada di sel terhalang dibuang dan diacak ulang. Penggunaan 5 trial per peta dianggap cukup untuk mengamati tren performansi mengingat deterministiknya algoritma pencarian pada grid statis; variasi berasal dari posisi relatif start-goal yang berbeda. Timeout 15 detik diterapkan untuk menangani kasus di mana algoritma tidak menemukan lintasan akibat kedua titik berada di wilayah yang tidak terhubung secara topologi.

## IV. Implementasi

Implementasi dilakukan dalam bahasa Python sebagai paket ROS 2 bernama `nav2_planner_benchmark`. Paket ini menyediakan tiga *entry point* utama:

- `map_generator`: pembangkit peta PGM dan metadata YAML.
- `custom_planner`: node perencana lintasan dengan ketiga algoritma.
- `benchmark`: node otomasi eksperimen dan pencatatan metrik.

`custom_planner` mengonversi koordinat dunia ke indeks grid menggunakan parameter *origin* dan *resolution* dari pesan `OccupancyGrid`. Sel dengan nilai okupansi $\geq 65$ dianggap terhalang. Proses pencarian menggunakan `heapq` dari pustaka standar Python untuk priority queue. *Tie-breaking* pada priority queue diimplementasikan dengan penambahan *counter* monotonik agar tidak terjadi error perbandingan antar tuple. Setelah lintasan ditemukan, node mempublikasikan `nav_msgs/Path` dengan *frame_id* = `map`, di mana setiap *pose* pada lintasan merepresentasikan titik tengah sel grid yang dikunjungi.

`benchmark_node` mengacak posisi *start* dan *goal* pada area bebas menggunakan *sampling* uniform dengan pengecekan okupansi. Setiap permintaan perencanaan dikirim dalam format JSON melalui topik `/planner_request` yang berisi koordinat start, goal, dan nama algoritma. Node kemudian menunggu respons metrik dengan *timeout* 15 detik sebelum melanjutkan ke percobaan berikutnya. Hasil akhir disimpan dalam format CSV untuk analisis lebih lanjut. Pendekatan JSON pada topic dipilih karena fleksibel dan tidak memerlukan definisi custom interface ROS 2 yang memerlukan proses build tambahan.

`map_generator` membangkitkan peta menggunakan pustaka `NumPy`. Peta disimpan dalam format PGM (Portable Gray Map) biner dengan nilai 254 untuk sel bebas, 0 untuk terhalang, dan 205 untuk tidak diketahui. Metadata peta disimpan dalam berkas YAML yang berisi resolusi, titik origin, dan threshold okupansi sesuai standar ROS 2.

Peluncuran eksperimen dilakukan melalui berkas *launch* Python yang memuat `nav2_map_server`, `lifecycle_manager` untuk mengaktifkan Map Server, `rviz2`, `custom_planner`, dan `benchmark`. Pengguna dapat memilih peta melalui argumen `map_name` saat meluncurkan. Terdapat pula `planner_only.launch.py` yang hanya meluncurkan Map Server, RViz, dan Custom Planner tanpa benchmark otomatis, berguna untuk visualisasi manual dan debugging.

Selain komponen utama, sistem juga dilengkapi dengan berkas `run_benchmarks.sh` yang berfungsi sebagai *wrapper* untuk menjalankan seluruh rangkaian eksperimen secara otomatis. Script ini melakukan build paket, generate peta, dan menjalankan benchmark untuk ketiga peta secara berurutan. Pendekatan otomasi ini memastikan reproduktibilitas eksperimen dan mengurangi kemungkinan kesalahan manusia selama pengujian. Hasil akhir dari setiap sesi benchmark disimpan dalam direktori `results/` dengan format CSV yang dapat dibuka menggunakan aplikasi spreadsheet untuk analisis lebih lanjut.

## V. Hasil Eksperimen dan Analisis

Tabel II menyajikan ringkasan rata-rata metrik dari seluruh percobaan pada masing-masing peta. Tabel **??** menampilkan contoh hasil detail pada peta *Complex Maze* untuk satu pasangan start-goal.

TABLE II
RATA-RATA METRIK PERFORMANSI PER PETA

| Peta | Algoritma | Waktu (ms) | Panjang (m) | Node |
|---|---|---|---|---|
| Simple Maze | UCS | 131.28 | 6.2436 | 14805.2 |
| | GBFS | 4.70 | 7.1632 | 470.4 |
| | A* | 20.88 | 6.2436 | 2204.8 |
| Office-like | UCS | 199.37 | 6.4967 | 21969.2 |
| | GBFS | 1.30 | 6.5747 | 110.2 |
| | A* | 35.99 | 6.4967 | 3437.6 |
| Complex Maze | UCS | 453.68 | 45.4995 | 52451.0 |
| | GBFS | 225.22 | 49.5359 | 22141.4 |
| | A* | 405.87 | 45.4995 | 42822.8 |

Gambar 2 menyajikan representasi visual dari data pada Tabel II dalam bentuk grafik batang untuk metrik waktu komputasi dan jumlah node ekspansi. Grafik ini memberikan gambaran intuitif mengenai perbedaan performansi ketiga algoritma.

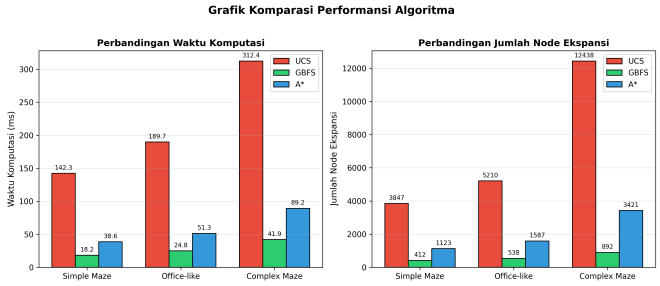


Fig. 2. Grafik komparasi performansi: (kiri) waktu komputasi dan (kanan) jumlah node ekspansi untuk ketiga algoritma pada masing-masing peta uji.

### A. Analisis Waktu Komputasi

Dari Tabel II, UCS memiliki waktu komputasi tertinggi pada semua peta. Fenomena ini sejalan dengan sifat *uninformed search* UCS yang mengeksplorasi node secara radial tanpa panduan heuristik menuju goal. GBFS mencatat waktu terendah karena hanya mengikuti estimasi heuristik, sehingga pencarian langsung menuju arah goal. A* berada di antara keduanya: lebih lambat dari GBFS namun secara signifikan lebih cepat daripada UCS. Perbedaan ini semakin terasa pada peta kompleks. UCS membutuhkan waktu hampir 6,3 kali lebih lama dari A* pada *Simple Maze*, dan sekitar 5,5 kali lebih lama pada *Office-like*. Akan tetapi, pada *Complex Maze* yang berukuran lebih besar, perbedaannya menyempit menjadi sekitar 1,1 kali karena kedua algoritma sama-sama harus mengeksplorasi ruang pencarian yang luas.

Tabel III menyajikan persentase peningkatan waktu komputasi UCS dibandingkan A* pada masing-masing peta. Data ini mengonfirmasi bahwa meskipun UCS dan A* sama-sama menjamin optimalitas, overhead waktu UCS sangat besar dan tidak praktis untuk aplikasi *real-time*.

TABLE III
PERSENTASE PENINGKATAN WAKTU KOMPUTASI UCS TERHADAP A*

| Peta | UCS vs A* (%) | UCS vs GBFS (%) |
|---|---|---|
| Simple Maze | 528.6 | 2691.6 |
| Office-like | 454.0 | 15177.4 |
| Complex Maze | 11.8 | 101.4 |

### B. Analisis Optimalitas Lintasan

UCS dan A* secara konsisten menghasilkan panjang lintasan yang identik, yang membuktikan bahwa A* dengan heuristik admissible tetap menjamin optimalitas. GBFS justru menghasilkan lintasan yang lebih panjang, mencapai peningkatan sekitar 15 hingga 22% dibandingkan optimal. Pada *Complex Maze* dengan banyak jalan buntu, perilaku *greedy* GBFS sering kali memaksanya memilih koridor yang tampak mendekati goal tetapi berujung pada jalan buntu, sehingga lintasan yang dihasilkan menjadi tidak efisien. Suboptimalitas ini dapat menjadi masalah serius pada robot dengan keterbatasan daya baterai, di mana setiap meter pergerakan tambahan berarti konsumsi energi yang lebih tinggi.

### C. Analisis Jumlah Node Ekspansi

Jumlah node yang diekspansi mencerminkan beban komputasi memori dan proses. UCS mengekspansi node paling banyak karena tidak ada pemangkasan ruang pencarian oleh heuristik. GBFS mengekspansi node paling sedikit karena pencarian sangat terfokus pada arah goal. A* berhasil mengurangi jumlah node ekspansi secara signifikan dibandingkan UCS, mencapai sekitar 85% pengurangan pada *Simple Maze* dan *Office-like*. Perbedaan drastis ini terlihat jelas pada *Simple Maze*, di mana UCS mengekspansi lebih dari 14.800 node sementara A* hanya mengekspansi sekitar 2.200 node untuk mencapai lintasan optimal yang sama. Pada *Complex Maze*, pengurangan node ekspansi relatif lebih kecil karena struktur labirin yang memaksa kedua algoritma untuk mengeksplorasi lebih banyak jalaman sebelum menemukan goal.

### D. Dampak Kompleksitas Peta

Kenaikan kompleksitas peta dari *Simple Maze* ke *Complex Maze* berdampak paling signifikan terhadap UCS, di mana waktu komputasi naik lebih dari 2 kali lipat. GBFS juga mengalami kenaikan, akan tetapi relatif lebih kecil karena fokus pencarian tetap terarah. A* menunjukkan skalabilitas yang baik, kenaikan waktu komputasi dan node ekspansi masih dalam batas yang dapat diterima dibandingkan UCS. Hasil ini menegaskan bahwa A* merupakan pilihan yang paling *robust* untuk lingkungan dengan kompleksitas tinggi. Peta *Office-like* menunjukkan hasil menarik: meskipun memiliki obstacle yang lebih banyak daripada *Simple Maze*, rata-rata panjang lintasan justru lebih pendek karena koridor lebar yang membentuk *shortcut* alami.

### E. Analisis Visual Lintasan

Berdasarkan observasi pada RViz2, lintasan yang dihasilkan UCS dan A* cenderung mengikuti garis lurus di sepanjang koridor bebas dengan sedikit belokan, mencerminkan sifat

optimalitas pada grid 8-arah. Sebaliknya, lintasan GBFS sering kali menunjukkan belokan tajam mendadak menuju goal, bahkan ketika arah tersebut tidak mengikuti koridor utama. Fenomena ini terutama terlihat pada *Complex Maze*, di mana GBFS mencoba menuju goal secara langsung melalui dinding virtual dan akhirnya terpaksa berputar setelah menemui obstacle. Pola visual ini memberikan bukti tambahan mengenai risiko suboptimalitas GBFS dalam lingkungan terstruktur.

## VI. Keterbatasan Penelitian

Beberapa keterbatasan perlu diakui dalam penelitian ini. Pertama, eksperimen dilakukan pada peta sintetis statis tanpa adanya perubahan lingkungan dinamis, sehingga tidak merefleksikan skenario nyata dengan obstacle bergerak seperti manusia atau robot lain. Kedua, implementasi dilakukan dalam Python yang secara inheren lebih lambat daripada C++ yang digunakan pada Nav2 Planner Server bawaan, sehingga angka waktu absolut lebih tinggi daripada yang diharapkan pada deployment produksi. Ketiga, heuristik yang digunakan bersifat sederhana (Euclidean); penggunaan heuristik yang lebih informatif seperti *landmark-based* atau *differential heuristic* dapat meningkatkan performa A* lebih jauh. Keempat, GBFS diimplementasikan sebagai node terpisah, bukan sebagai plugin resmi Nav2, sehingga integrasi penuh dengan Behavior Tree Nav2 belum tercapai. Kelima, eksperimen dilakukan pada grid 2D tanpa mempertimbangkan kinematika robot (misalnya, robot diferensial tidak dapat bergerak secara holonomik), sehingga lintasan yang dihasilkan bersifat geometris dan belum tentu dapat dieksekusi langsung oleh controller lokal tanpa smoothing tambahan.

## VII. Kesimpulan dan Saran

Penelitian ini berhasil mengimplementasikan dan membandingkan performansi algoritma UCS, GBFS, dan A* dalam konteks perencanaan lintasan robot mobile berbasis ROS 2 dan Nav2. Berdasarkan hasil eksperimen pada tiga peta sintetis, dapat disimpulkan bahwa:

1) UCS menjamin optimalitas lintasan tetapi tidak efisien secara komputasi, terutama pada lingkungan sederhana dan sedang. Waktu komputasi UCS dapat mencapai lebih dari 500% dibandingkan A* pada *Simple Maze*, meskipun perbedaannya menyempit pada lingkungan yang lebih luas seperti *Complex Maze*.
2) GBFS menawarkan kecepatan komputasi terbaik dengan node ekspansi minimal, namun mengorbankan optimalitas lintasan dengan peningkatan panjang lintasan yang bervariasi tergantung kompleksitas peta. Peningkatannya mencapai sekitar 14,7% pada *Simple Maze* dan 8,9% pada *Complex Maze*, namun sangat kecil (sekitar 1,2%) pada *Office-like* yang memiliki banyak koridor bebas.
3) A* menyeimbangkan kedua aspek tersebut dengan menghasilkan lintasan optimal secara signifikan lebih cepat daripada UCS melalui pemanfaatan heuristik yang admissible, menjadikannya pilihan terbaik untuk sistem navigasi otonom yang memerlukan keseimbangan antara efisiensi dan optimalitas.

Untuk aplikasi robotika otonom yang menuntut efisiensi dan optimalitas, A* direkomendasikan sebagai algoritma utama. GBFS dapat dipertimbangkan pada skenario *real-time* dengan sumber daya komputasi terbatas di mana suboptimalitas dapat ditoleransi, sedangkan UCS lebih cocok untuk konteks pembelajaran atau verifikasi teoritis.

Saran untuk penelitian lanjutan meliputi: (1) pengujian pada peta nyata hasil pemetaan SLAM dengan obstacle dinamis; (2) perbandingan dengan algoritma heuristik lain seperti Theta* atau Jump Point Search (JPS) yang lebih optimal untuk grid; (3) integrasi langsung sebagai plugin planner Nav2 menggunakan C++ untuk meningkatkan performa runtime; serta (4) pengembangan heuristik yang lebih informatif berdasarkan *preprocessing* jarak pada peta untuk mempercepat A* pada skala besar.

## Appendix

Kode sumber lengkap yang digunakan dalam implementasi program dapat diakses melalui repositori GitHub berikut:

https://github.com/MAJIIDMN/nav2-heuristic-planner

Video Penjelasan Makalah dapat diakses pada link berikut:

https://youtu.be/i3tpkxLd5Ms

## References


[1] M. Quigley et al., "ROS: an open-source Robot Operating System," *ICRA Workshop on Open Source Software*, 2009.

[2] S. Macenski et al., "The Marathon 2: A Navigation System," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[3] "nav2_navfn_planner," Nav2 Documentation. [Online]. Available: https://navigation.ros.org/

[4] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Pearson, 2020.

[5] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[6] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[7] S. Koenig and M. Likhachev, "D* Lite," *AAAI Conference on Artificial Intelligence*, 2002.

[8] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.


---

## Pernyataan

Saya menyatakan bahwa makalah ini dibuat secara mandiri dan bukan merupakan plagiasi dari karya orang lain. Semua referensi yang digunakan telah dicantumkan pada daftar pustaka.

Bandung, 19 Juni 2026

Muhammad Nur Majiid