Analisis ROS Week 5

- **A***
  Algoritma A* adalah metode pencarian jalur yang menggabungkan biaya aktual ke sebuah titik (g_score) dan perkiraan jarak ke tujuan (h_score). A* memilih node dengan nilai f_score terkecil, di mana f_score adalah penjumlahan g_score dan h_score. Algoritma ini memproses node dengan cara yang efisien, hanya mempertimbangkan jalur yang cenderung mendekati tujuan. Ketika tujuan tercapai, A* membangun jalur optimal dengan mengikuti jejak node dari tujuan ke titik awal. Kombinasi strategi ini membuat A* cepat dan akurat dalam menemukan rute terpendek, terutama pada peta yang kompleks.

- **Dijkstra**
  Algoritma Dijkstra adalah metode pencarian jalur terpendek yang menghitung jarak minimum dari titik awal ke semua titik dalam graf, bekerja efektif pada graf dengan bobot positif. Algoritma ini memilih node dengan jarak terpendek yang belum dikunjungi, kemudian memperbarui jarak ke node-node tetangganya. Proses ini diulang hingga semua node telah dikunjungi atau jarak ke tujuan ditemukan.
  Dalam setiap iterasi, Dijkstra menambah node terdekat ke jalur terpendek yang sudah dibangun, memperbarui jarak jika ditemukan jalur yang lebih pendek ke node tetangga. Algoritma ini menjamin solusi optimal, tetapi dapat lambat dalam peta besar karena memeriksa semua jalur tanpa heuristik untuk memprioritaskan jalur menuju tujuan, seperti pada A*.

- **Cell Decomposition**
  Algoritma cell decomposition adalah metode perencanaan jalur yang memecah ruang pencarian menjadi sel-sel (atau area kecil) yang lebih sederhana. Pendekatan ini cocok untuk lingkungan dengan rintangan yang kompleks, karena memetakan area yang bisa dilalui dan yang terhalang secara terstruktur. Ada dua jenis utama, namun yang digunakan saat ini hanyalah Approximate Cell Decomposition:

  Approximate Cell Decomposition: Ruang dibagi ke dalam grid dengan ukuran tetap, tanpa mempertimbangkan bentuk rintangan secara detail. Tiap sel grid dikategorikan sebagai bebas atau terhalang, dan jalur dibuat melalui sel-sel bebas yang saling terhubung. Pendekatan ini lebih cepat namun kurang presisi.

  Secara keseluruhan, cell decomposition mengubah masalah navigasi menjadi pencarian jalur di antara sel-sel yang bebas dari rintangan, mempermudah perhitungan dalam ruang kompleks dan memungkinkan robot untuk bergerak dengan aman di sekitar rintangan.

- **Simulasi ROS**
  Dalam menggunakan Rviz untuk implementasi path searching, saya melakukan analisis terhadap bagaimana algoritma seperti A* dan Dijkstra dapat digunakan untuk menavigasi robot melalui lingkungan yang kompleks. Pertama, saya memetakan area dengan mendefinisikan rintangan yang ada menggunakan representasi visual yang jelas, sehingga saya dapat memahami batasan navigasi. Selanjutnya, dengan

menggunakan algoritma A*, saya memanfaatkan fungsi heuristik yang membantu dalam memperkirakan jarak dari titik saat ini ke tujuan, yang memungkinkan pemilihan jalur yang lebih efisien. Di Rviz, jalur yang dihasilkan oleh algoritma ini divisualisasikan, menunjukkan rute optimal yang menghindari rintangan yang ada. Proses ini memungkinkan saya untuk menganalisis keefektifan jalur yang dipilih dan mengidentifikasi kemungkinan modifikasi yang diperlukan untuk meningkatkan navigasi robot. Dengan memantau jalur secara real-time dalam Rviz, saya dapat dengan mudah melakukan penyesuaian terhadap algoritma dan parameter, memastikan robot dapat menavigasi dengan aman dan efisien di dalam ruang yang telah didefinisikan. Analisis ini memberikan wawasan mendalam tentang bagaimana algoritma path searching berfungsi dalam konteks aplikasi dunia nyata, serta tantangan yang dihadapi dalam pengimplementasiannya.

- **Analisis Animasi**

1. GBFS:

- Algoritma yang menggunakan heuristik untuk menemukan jalur terpendek secara cepat, tetapi tidak menjamin optimalitas.

2. Dijkstra:

- Algoritma yang menemukan jalur terpendek dari satu titik ke semua titik lainnya dengan jaminan optimal, namun bisa lambat.

3. A*:

- Algoritma yang menggabungkan biaya perjalanan dan heuristik untuk menemukan jalur terpendek, optimal jika heuristik admissible.

4. JPS:

- Optimasi dari A* untuk grid yang mengurangi node yang dieksplorasi dengan cara melompati beberapa titik.

5. D*:

- Algoritma yang dirancang untuk lingkungan dinamis, memungkinkan pembaruan jalur secara efisien saat rintangan berubah.

6. LPA*:

- Varian dari A* yang mempertahankan solusi jalur dan memperbarui secara efisien ketika ada perubahan pada lingkungan.

7. *D Lite:**

- Versi sederhana dari D* yang memperbarui jalur dengan cepat dan lebih ringan dalam implementasi.

8. Voronoi:

- Menggunakan diagram Voronoi untuk menciptakan jalur optimal dengan mempertimbangkan jarak ke rintangan.

9. Theta*:

- Algoritma yang menghasilkan jalur yang lebih halus dan langsung, mengurangi jumlah sudut tajam.

10. Lazy Theta*:

- Versi Theta* yang menghitung jalur hanya saat diperlukan, mengurangi komputasi awal.

11. S-Theta*:

- Memadukan konsep Theta* dengan kontrol sudut, menghasilkan jalur yang lebih alami dengan sudut lebih lembut.

12. Hybrid A*:

- Menggabungkan pendekatan A* dengan teknik lain untuk mengoptimalkan kinerja dalam berbagai skenario.

13. RRT:

- Algoritma yang membangun pohon pencarian secara acak, efisien untuk lingkungan kompleks dan tinggi dimensi.

14. RRT*:

- Memodifikasi RRT untuk menghasilkan jalur yang lebih halus dan optimal dengan memperbaiki jalur yang sudah ditemukan.

15. Informed RRT:

- Versi RRT yang menggunakan informasi heuristik untuk memperbaiki pencarian jalur.

16. RRT-Connect:

- Menghubungkan dua pohon RRT secara efisien, mempercepat pencarian jalur antara dua titik.

17. ACO:

- Algoritma yang meniru perilaku semut untuk menemukan jalur optimal dengan memperkuat jalur yang sering dilalui.

18. GA:

- Menggunakan prinsip evolusi dan seleksi alam untuk menemukan solusi optimal melalui populasi kandidat.

19. PSO:

- Algoritma yang meniru perilaku kelompok burung untuk menemukan solusi optimal dengan memperbarui posisi berdasarkan pengalaman individu dan kelompok.