ROS Motion Planning

Dalam git ros_motion_planning terdapat algoritma yang dapat digunakan untuk melakukan simulasi motion planning. Yang pertama terdapat GBFS, GBFS digunakan untuk menemukan jalur yang paling menjanjikan menuju tujuan dengan menggunakan heuristik. Algoritma ini memilih jalur berdasarkan perkiraan terpendek dari posisi saat ini ke posisi tujuan, tanpa mempertimbangkan biaya yang telah dikeluarkan untuk mencapai posisi tersebut. Yang kedua ada Dijkstra, Dijkstra digunakan untuk menemukan jalur terpendek dari titik awal ke titik tujuan, mempertimbangkan semua kemungkinan jalur. Ini melacak biaya total dari titik awal ke setiap titik dalam ruang dan memperbarui jalur yang lebih pendek jika ditemukan. Dan yang terakhir terdapat A*, algoritma ini menggabungkan biaya dari titik awal ke titik saat ini (g(n)) dan estimasi biaya dari titik saat ini ke tujuan (h(n)). Dengan cara ini, A* dapat mengeksplorasi jalur yang lebih menjanjikan dengan lebih efisien dibandingkan Dijkstra. Algoritma-algoritma ini sangat membantu dalam melakukan simulasi motion planning dalam robot. Penggunaan algoritma motion planning ini tergantung pada kebutuhan aplikasi.

Berikut ini adalah analisis kelebihan dan kekurangan dari tiga algoritma yang kamu sebutkan: Cell Decomposition, Dijkstra Algorithm, dan A*

1. Cell Decomposition

Kelebihan:

- Sederhana untuk Diimplementasikan: Algoritma ini mudah dimengerti dan diimplementasikan, terutama untuk ruang dua dimensi.
- Efisien dalam Ruang Terbatas: Cell decomposition membagi area kompleks menjadi sel-sel lebih kecil yang memudahkan navigasi, terutama di lingkungan dengan batasan-batasan (misalnya, dinding atau rintangan).
- Fleksibel untuk Aplikasi Robotika: Umum digunakan dalam navigasi robot untuk rencana jalur karena dapat beradaptasi dengan bentuk dan ukuran sel yang fleksibel.

Kekurangan:

- Biaya Komputasi yang Tinggi untuk Lingkungan Kompleks: Jika lingkungan memiliki banyak rintangan, pembagian menjadi banyak sel dapat menjadi rumit dan menghabiskan memori.
- Keakuratan Terbatas: Ketika membagi ruang, solusi yang dihasilkan mungkin tidak optimal karena algoritma ini hanya mengandalkan jarak antar sel, bukan jalur langsung.
- Keterbatasan pada Area Besar: Jika diterapkan di ruang besar, terutama dalam tiga dimensi, proses dekomposisi menjadi sangat mahal dan kompleks.

2. Dijkstra Algorithm

Kelebihan:

- Optimalitas dan Keterjaminan: Algoritma Dijkstra menemukan jalur terpendek dari satu simpul ke simpul lain dalam graf berorientasi dan berbobot non-negatif. Hasilnya selalu optimal.

- Penerapan pada Graf Berbobot: Sangat berguna dalam graf yang memiliki bobot berbeda, seperti jaringan transportasi atau jaringan komunikasi.
- Stabil dan Konsisten: Hasilnya stabil di lingkungan yang dinamis karena akan tetap menemukan jalur terpendek selama graf tidak memiliki bobot negatif.

Kekurangan:

- Inefisiensi pada Graf Besar: Algoritma ini membutuhkan pemrosesan setiap simpul, sehingga waktu komputasinya tinggi pada graf besar (kompleksitasnya \(O(V^2)\) pada representasi matriks, atau \(O(E + V \log V)\) jika menggunakan heap).
- Tidak Optimal untuk Dinamisitas Lingkungan: Algoritma ini kurang efektif untuk situasi yang berubah-ubah, karena perlu dijalankan ulang setiap kali ada perubahan pada bobot graf.
- Kurang Efektif untuk Aplikasi Navigasi: Untuk kasus yang memerlukan pendekatan yang lebih heuristik, algoritma ini tidak mempertimbangkan jarak tujuan, yang dapat menyebabkan eksplorasi berlebihan.

3.A-Star

Kelebihan:

- Optimal dan Efisien dalam Pencarian Jalur Terpende: A* menggunakan heuristik untuk mempercepat pencarian jalur terpendek, sehingga lebih efisien dibandingkan Dijkstra di banyak kasus.
- Fleksibilitas dalam Penggunaan Heuristik: Algoritma A* memungkinkan penyesuaian heuristik, yang dapat disesuaikan untuk menghasilkan hasil lebih cepat atau lebih akurat.
- Efisiensi pada Graf Besar: Karena menggunakan heuristik, A* biasanya mengeksplorasi simpul yang lebih relevan dan mengabaikan simpul yang kurang relevan, menghemat waktu komputasi dibandingkan dengan Dijkstra.

Kekurangan:

- Kebergantungan pada Heuristi: A* sangat bergantung pada heuristik yang dipilih; jika heuristik kurang tepat, algoritma ini bisa kehilangan efisiensinya atau bahkan menjadi tidak optimal.
- Konsumsi Memori yang Tinggi: Pada graf yang kompleks, algoritma ini bisa menghabiskan banyak memori karena harus menyimpan semua simpul yang dieksplorasi.
- Keterbatasan pada Lingkungan yang Sangat Dinamis: A* tidak selalu efektif jika graf terus berubah, karena akan membutuhkan penghitungan ulang jalur ketika bobot berubah secara signifikan.