Langkah-langkah Selanjutnya untuk ROS Motion Planning

Untuk melanjutkan dengan perencanaan jalur menggunakan ROS (Robot Operating System), Anda dapat merujuk ke dokumentasi ROS dan mengikuti tutorial tentang MoveIt! atau Navigation Stack untuk merencanakan jalur robot.

Langkah dasar:

- Instalasi ROS dan Dependensi - Pastikan Anda memiliki ROS, Gazebo, dan Rviz.
- Set Up Robot Model dan Environmen - Gunakan model robot dalam format URDF untuk memvisualisasikan dan menjalankan simulasi pergerakan di Gazebo.
- Menggunakan MoveIt! untuk Motion Planning - Konfigurasi MoveIt! untuk memanipulasi robot dengan obstacle avoidance.
- Testing and Tuning - Setelah setup selesai, jalankan simulasi dan perbaiki konfigurasi agar jalur yang ditemukan lebih optimal dan efisien.

1. Implementasi Algoritma Perencanaan Jalur

a. Dijkstra's Algorithm

- Analisis Proses: Algoritma Dijkstra berhasil menemukan jalur terpendek dari titik awal ke tujuan dengan mempertimbangkan semua jarak antar node dalam graf berbobot. Setiap node diproses dengan mencari node yang memiliki nilai bobot terendah secara bertahap hingga mencapai tujuan.
- Kelebihan: Dijkstra adalah algoritma yang menjamin menemukan jalur terpendek karena memperhitungkan setiap node dengan akurat tanpa heuristic. Algoritma ini sangat efektif untuk jaringan graf yang kecil atau sedang.
- Kekurangan: Dalam graf besar atau grid dengan banyak rintangan, Dijkstra membutuhkan waktu yang cukup lama karena harus memeriksa semua node, menjadikannya kurang efisien dibandingkan algoritma dengan heuristic, seperti A*.

b. A Algorithm*

- Analisis Proses: A* Algorithm menggunakan kombinasi dari jarak sebenarnya yang sudah ditempuh dan estimasi jarak ke tujuan (heuristic) untuk mempercepat pencarian jalur. Pada grid dengan rintangan, A* lebih efisien karena heuristic memungkinkan pencarian yang lebih fokus.
- Kelebihan: A* dapat lebih cepat menemukan jalur terpendek dibandingkan Dijkstra karena menggunakan heuristic (dalam hal ini, jarak Manhattan), sehingga dapat mempersempit ruang pencarian, terutama pada graf/grid besar.
- Kekurangan: Hasil dari A* sangat tergantung pada heuristic yang digunakan. Jika heuristic tidak akurat, hasil yang ditemukan bisa tidak optimal.

c. Cell Decomposition

- Analisis Proses: Cell Decomposition membagi ruang atau grid menjadi cell-cell yang lebih kecil untuk mempermudah pencarian jalur. Algoritma ini bertujuan untuk menemukan jalur aman antar cell yang bebas rintangan dari titik awal hingga titik tujuan.
- Kelebihan: Algoritma ini efektif untuk lingkungan yang kompleks dengan banyak rintangan. Pembagian cell membantu dalam mengelola dan memetakan area yang luas serta mempercepat pencarian jalur aman.
- Kekurangan: Cell Decomposition bisa menjadi tidak efisien jika cell yang dibentuk sangat banyak atau memiliki ukuran kecil karena akan memakan waktu komputasi yang lebih besar.

2. Simulasi ROS Motion Planning

a. Path Searching

- Proses dan Analisis: Path Searching pada simulasi ROS dilakukan dengan berbagai algoritma untuk menemukan jalur optimal dari titik awal hingga titik tujuan. Dalam simulasi ini, modul seperti MoveIt! atau Navigation Stack membantu merencanakan jalur yang aman dan efisien dengan menghindari rintangan.
- Perbandingan Algoritma dalam ROS:

* GBFS (Greedy Best-First Search): Algoritma ini cenderung mencari jalur berdasarkan perkiraan jarak terpendek ke tujuan, tetapi seringkali tidak optimal karena tidak memperhitungkan seluruh bobot jalan.
* Dijkstra: Kembali membuktikan keandalannya dalam menemukan jalur terpendek tanpa heuristic, tetapi kurang efisien untuk grid besar.
* A*: A* menunjukkan kinerja yang efisien dengan memanfaatkan heuristic, memberikan hasil yang optimal dalam waktu yang lebih singkat dibandingkan Dijkstra.

Observasi Hasil: Pada animasi simulasi, A* terlihat lebih lancar dalam menghindari rintangan dan mencapai tujuan dibandingkan Dijkstra, terutama di lingkungan yang kompleks. GBFS, meski cepat, cenderung menghasilkan jalur yang tidak selalu optimal.

b. Trajectory Optimization

- Proses dan Analisis: Trajectory Optimization pada ROS memastikan jalur yang direncanakan tidak hanya aman tetapi juga sesuai dengan kinematika dan dinamika robot. Optimasi dilakukan untuk mengurangi belokan tajam dan memperhalus jalur.
- Kelebihan: Trajectory Optimization membantu menghindari belokan yang tidak mungkin atau tidak nyaman bagi robot dan menjadikan jalur lebih efisien serta aman dalam simulasi maupun aplikasi nyata.

- Observasi Hasil: Dari simulasi di Rviz, jalur yang sudah dioptimasi terlihat lebih halus dan terarah, memastikan pergerakan robot tetap konsisten. Hal ini mengurangi potensi ketidakstabilan selama perjalanan dan memberikan efisiensi energi.

3. Analisis Keseluruhan

a. Efisiensi dan Akurasi Jalur: A* dan Dijkstra terbukti lebih andal dalam memberikan jalur terpendek dibandingkan GBFS, namun dengan performa yang lebih cepat pada A*.

b. Kebutuhan Pemilihan Algoritma Berdasarkan Lingkungan: Lingkungan dengan grid yang besar atau banyak rintangan memerlukan algoritma dengan heuristic, seperti A*, untuk kecepatan. Namun, jika lingkungan kecil atau medium, Dijkstra tetap andal.

c. Pengaruh Terhadap Kendali Robot di Dunia Nyata: Simulasi ROS yang menggunakan Trajectory Optimization memastikan pergerakan robot menjadi lebih realistis, meningkatkan kinerja untuk aplikasi di dunia nyata.