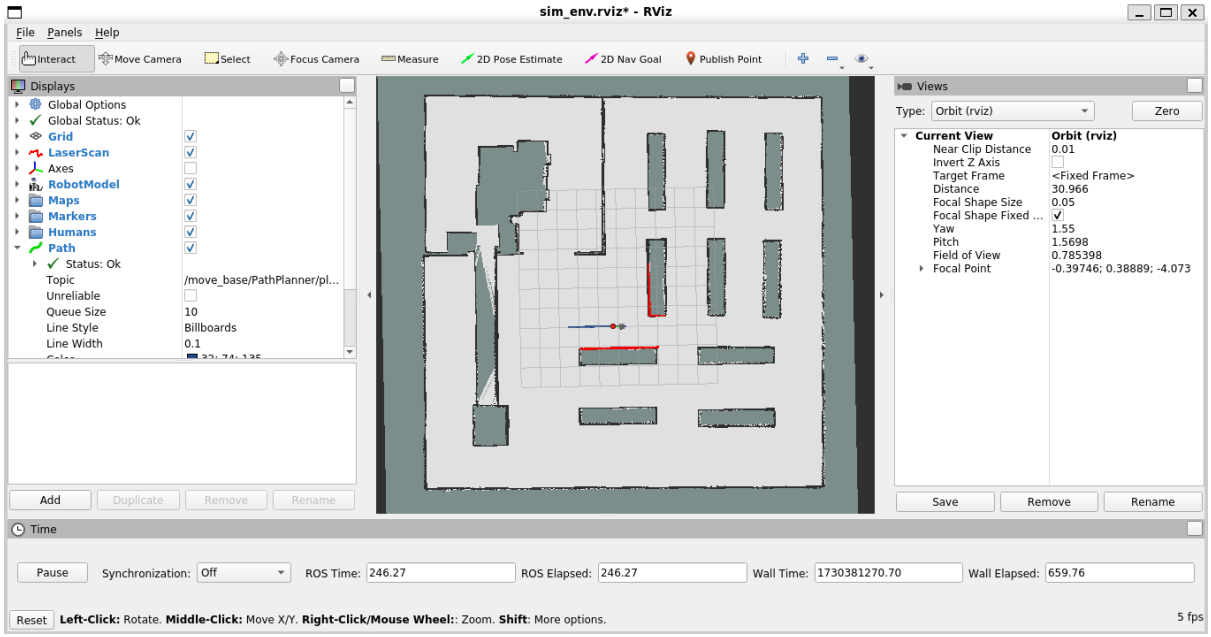
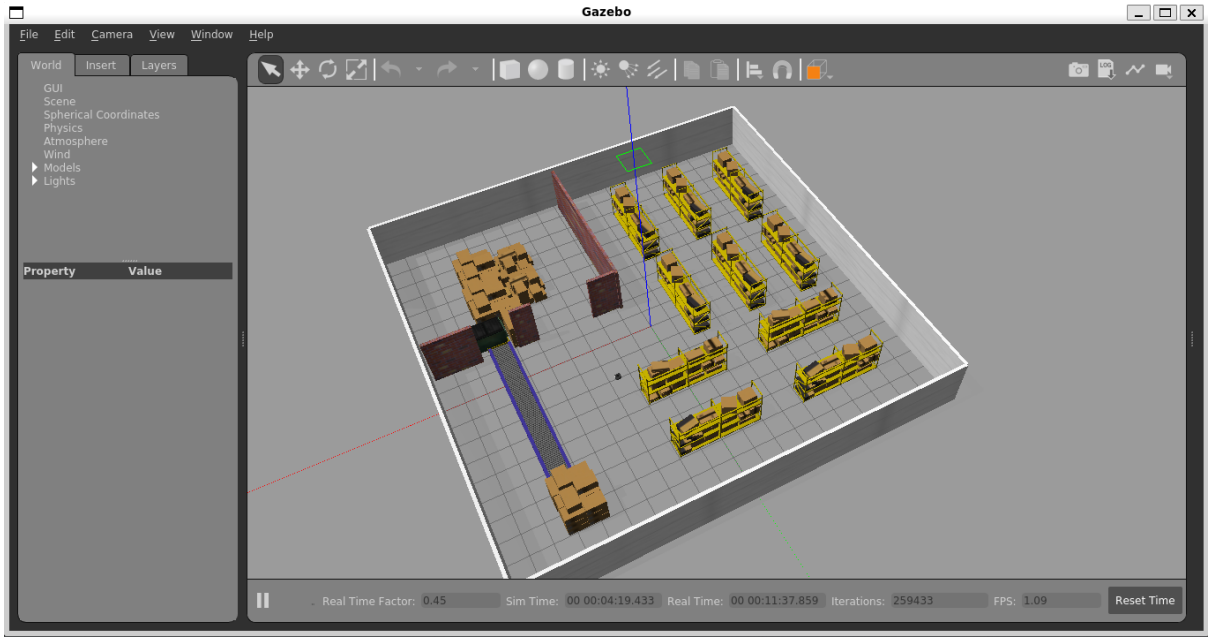
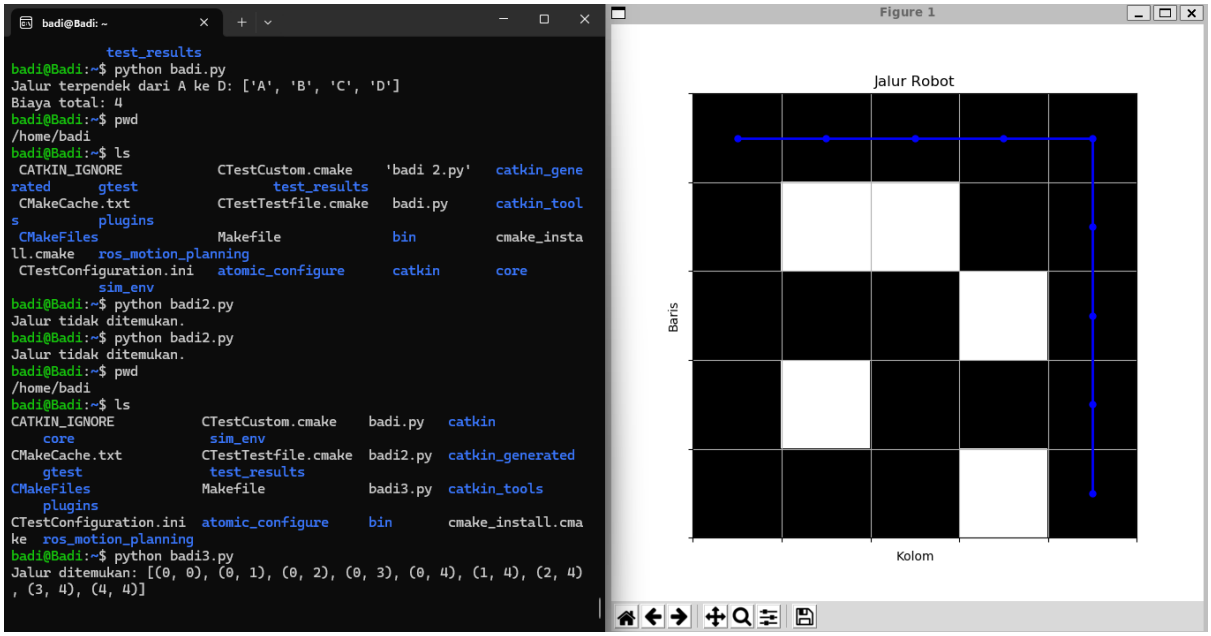



```
badi@Badi:~$ ls
CATKIN_IGNORE           CTestCustom.cmake    badi.py           catkin_tools          plugins
CMakeCache.txt          CTestTestfile.cmake  bin               cmake_install.cmake   ros_motion_planning
CMakeFiles              Makefile             catkin            core                  sim_env
CTestConfiguration.ini  atomic_configure     catkin_generated  gtest                 test_results
badi@Badi:~$ python badi.py
Jalur terpendek dari A ke D: ['A', 'B', 'C', 'D']
Biaya total: 4
badi@Badi:~$
```

```
badi@Badi:~$ ls
 CATKIN_IGNORE           CTestCustom.cmake    'badi 2.py'   catkin_generated     gtest                test_results
 CMakeCache.txt          CTestTestfile.cmake   badi.py      catkin_tools         plugins
 CMakeFiles              Makefile              bin          cmake_install.cmake  ros_motion_planning
 CTestConfiguration.ini  atomic_configure      catkin       core                 sim_env
badi@Badi:~$ python badi2.py
Jalur tidak ditemukan.
badi@Badi:~$ python badi2.py
Jalur tidak ditemukan.
badi@Badi:~$
```




Analisis

## 1. Implementasi Algoritma Perencanaan Jalur

### A. Dijkstra Algorithm

- **Deskripsi**: Algoritma Dijkstra adalah algoritma yang digunakan untuk menemukan jalur terpendek dari titik awal ke titik tujuan dalam graf. Algoritma ini menggunakan pendekatan greedy untuk mencari jalur terpendek dengan cara mengunjungi node-node yang terdekat terlebih dahulu.
- **Implementasi**: Dalam implementasi Dijkstra, kami menggunakan struktur data heap untuk menjaga daftar node yang akan dikunjungi. Setiap node dikunjungi satu per satu, dan biaya total untuk mencapai node tersebut diperbarui. Jalur terpendek dan biaya totalnya ditampilkan setelah proses selesai.
- **Hasil**: Jalur terpendek berhasil ditemukan dengan biaya total yang sesuai, menunjukkan bahwa algoritma ini efisien dalam mencari jalur terpendek tanpa adanya rintangan.

*B. A Algorithm**

- **Deskripsi**: Algoritma A* adalah algoritma pencarian jalur yang menggunakan heuristic untuk mempercepat proses pencarian. Algoritma ini menggabungkan biaya dari titik awal dan perkiraan biaya ke titik tujuan.
- **Implementasi**: Dalam kode A*, heuristic (misalnya, jarak Manhattan) digunakan untuk memandu pencarian. Algoritma ini memilih node yang memiliki biaya terendah (total biaya) dan memprioritaskan node yang lebih dekat ke tujuan.

- **Hasil**: Algoritma A* berhasil menemukan jalur optimal di dalam grid dengan rintangan, dan jalur yang ditemukan ditampilkan dengan baik. Kecepatan pencarian lebih tinggi dibandingkan Dijkstra, terutama pada graf yang lebih besar dengan banyak node.

**C. Cell Decomposition**

- **Deskripsi**: Metode Cell Decomposition membagi ruang menjadi cell-cell kecil dan mencari jalur yang aman di antara cell-cell tersebut. Metode ini bermanfaat dalam situasi di mana ruang memiliki rintangan yang kompleks.
- **Implementasi**: Dalam kode ini, grid yang merepresentasikan ruang digunakan. Algoritma rekursif dijalankan untuk menemukan jalur dari titik awal ke tujuan, memastikan bahwa jalur tersebut tidak melewati rintangan.
- **Hasil**: Jalur yang ditempuh robot ditampilkan dengan jelas, menunjukkan bagaimana robot dapat bergerak dari titik awal ke tujuan dengan menghindari rintangan.

**2. Simulasi ROS Motion Planning**

Simulasi menggunakan ROS (Robot Operating System) memberikan pemahaman yang lebih dalam tentang perencanaan gerakan robot. Dengan menggunakan Rviz dan Gazebo, kami dapat menguji algoritma perencanaan jalur yang telah diimplementasikan:

- **Path Searching**: Proses pencarian jalur berlangsung dengan menggunakan algoritma yang telah diuji sebelumnya. Robot mampu menghindari rintangan yang ada di lingkungan simulasi dengan baik.
- **Trajectory Optimization**: Setelah jalur ditemukan, langkah selanjutnya adalah mengoptimalkan trajektori berdasarkan kinematika dan dinamika robot. Penggunaan kontrol gerakan yang tepat sangat penting untuk memastikan robot bergerak dengan efisien dan aman.

**3. Hasil Analisis Animasi**

Melalui simulasi dan analisis dari setiap algoritma (Dijkstra, A*, dan metode lainnya seperti GBFS), kami menemukan bahwa:

- **Dijkstra** sangat baik untuk graf yang kecil dan sederhana, namun kurang efisien pada graf yang lebih besar karena biaya yang dihitung secara menyeluruh untuk setiap node.
- **A*** memberikan hasil yang lebih cepat dan lebih baik dalam situasi kompleks karena penggunaan heuristic.
- Metode **Cell Decomposition** efektif untuk situasi yang melibatkan rintangan dan memberikan jalur yang jelas untuk navigasi robot.