Dynamic Roadmaps

Soumik Saswat Patnaik
Robotics Engineering
Worcester Polytechnic Institute

Ameya Phadnis
Robotics Engineering
Worcester Polytechnic Institute

Pranjali Rangnekar
Robotics Engineering
Worcester Polytechnic Institute

I. Introduction: Research Question and Application Impacts

In this project, we aim to explore the development and implementation of Dynamic Roadmaps (DRM) for motion planning in environments where workspace configurations frequently change. Traditional roadmap-based planners, such as Probabilistic Roadmaps (PRM), offer efficient solutions by precomputing nodes and edges in the configuration space and reusing them for various start and goal queries (q_{start} and q_{goal}) within a fixed workspace (W). However, when the workspace changes—due to shifting obstacle locations or other dynamic factors—the precomputed roadmaps may become invalid, revalidating these roadmaps by performing collision checks on every edge is computationally prohibitive, with a time complexity of $O(n^2)$.

Dynamic Roadmaps offer a more efficient alternative. By mapping the workspace to roadmap nodes and edges in the configuration space, DRM allows for quick identification of affected areas. When changes occur in the workspace, only the impacted nodes and edges are marked as unsafe. This selective updating mechanism significantly reduces the computational cost associated with maintaining valid roadmaps.

For this project, we aim to develop a simple Dynamic Roadmap (DRM) capable of dynamically adapting to changes in the workspace. The DRM will efficiently update itself in response to obstacles and other environmental alterations, ensuring robust performance in real-time scenarios. Additionally, it will provide solutions for multiple motion-planning queries, such as pick-and-place tasks using a Fetch robot.

The core research question focuses on addressing the computational inefficiencies inherent in traditional roadmap-based planners when operating in dynamic environments. By leveraging DRMs, we aim to create solutions that enable real-time motion planning, meeting the demands of scenarios requiring rapid responses to environmental changes.

For example, in industrial automation, robots often face dynamically changing workspaces where tasks such as pick-and-place or assembly must be completed efficiently despite fluctuating obstacle locations. Our research will demonstrate how DRMs can handle these variations with minimal computational overhead, ensuring that motion-planning solutions remain feasible and efficient.

Ultimately, this research will highlight how DRMs provide a scalable and efficient framework for solving motion-planning problems in dynamic workspaces, advancing the capabilities of robotic systems in real-world applications.

II. Related Work

Leven and Hutchinson (2002)

This paper introduced a framework for real-time path planning in dynamic environments, addressing the challenge of maintaining feasible paths as the workspace changes. Their method combines precomputed roadmaps with dynamic updates, focusing on impacted regions rather than recalculating the entire roadmap. The framework employs the *Dynamic Region* concept to identify and update affected nodes and edges efficiently, minimizing computational overhead. Additionally, it integrates real-time collision-checking to ensure path validity within these dynamic regions. This approach, validated through simulations and experiments, serves as a foundation for our work on Dynamic Roadmaps (DRM), offering valuable insights into efficient path planning in changing environments.

Kallman and Mataric (2004)

This paper proposed a motion planning approach using Dynamic Roadmaps (DRM) to address the challenges of navigating in dynamic environments. Their method focuses on efficiently updating roadmaps as the workspace changes, enabling robots to maintain valid paths without the need for full recomputation. The key contribution of their work lies in dynamically linking roadmap nodes to regions of the workspace, allowing for selective updates when obstacles move or new ones are introduced. By focusing on local changes, their approach reduces the computational cost typically associated with global recalculations. This work directly aligns with our research on DRM, providing crucial insights into maintaining computational efficiency while ensuring robust motion planning in changing environments.

Yang et al. (2018)

This paper introduced HDRM, a high-resolution dynamic roadmap designed for real-time motion planning in complex and dynamic environments. Their approach ensures resolution completeness, meaning it can guarantee finding a solution if one exists, even in intricate workspaces. HDRM achieves this by maintaining a dense roadmap that is dynamically updated as the environment changes, with efficient collision-checking mechanisms to validate affected nodes and edges in real time. The authors demonstrated the effectiveness of HDRM in handling highly dynamic scenes, outperforming traditional planners in terms of both computational efficiency and success rate. This work is highly relevant to our research on Dynamic Roadmaps, providing a robust framework for ensuring path validity and adaptability in rapidly changing environments.

Guo et al. (2022)

This paper presented a dynamic free-space roadmap tailored for safe quadrotor motion planning in environments with dynamic obstacles. Their method continuously seeds and extracts free regions within the environment, incrementally decomposing polyhedra intersecting with obstacles into obstacle-free regions. This dynamic adaptation allows the roadmap to efficiently update in response to environmental changes, ensuring safe navigation for quadrotors. The authors validated their approach through extensive simulations and real-world experiments, demonstrating its practical applicability and efficiency in dynamic settings. This study offers valuable insights into maintaining safe and efficient motion planning in environments with moving obstacles.

Sivaramakrishnan et al. (2023)

This paper proposed a novel approach to improve the computational efficiency of motion planning for mobile robots with complex dynamics. Their method involves training a system-specific controller offline in an empty environment to handle the robot's dynamics. For a target environment, they construct a "Roadmap with Gaps," where nodes correspond to local regions and edges represent applications of the learned control policy that approximately connect these regions. This structure allows for efficient online planning, as the roadmap guides the robot towards the goal region, and random exploration is employed when the controller cannot reach a subgoal region. The authors' experimental evaluation demonstrated significant improvements in computational efficiency across various benchmarks, including physics-based vehicular models and quadrotors under varying conditions. This research contributes to developing efficient motion planning strategies that leverage learned controllers and adaptable roadmaps in dynamic environments.

III. Environment Setup

Environment Setup

The development and testing of the Dynamic Roadmap (DRM) algorithm require a well-configured environment to simulate and evaluate motion-planning scenarios. The following steps outline the setup process:

1) **Installation of OMPL Python Bindings:** The Open Motion Planning Library (OMPL) is installed with its Python bindings to provide essential functionality for sampling-based motion planning.
2) **Installation of Grapeshot API:** The Grapeshot API is installed to facilitate seamless communication with **PyBullet**. Once the Grapeshot project is built, it automatically installs the necessary dependencies such as `numpy`, `scipy`, and **PyBullet**.

Utilizing the Environment

The configured environment is used to simulate and implement the DRM algorithm effectively. The **PyBullet** simulation engine is employed to simulate our motion-planning scenarios, enabling realistic interactions with dynamic environments. The Grapeshot API acts as a bridge between the algorithm and **PyBullet**, allowing us to execute motion-planning algorithms and test their performance with various robotic platforms. Grapeshot provides versatile features, including support for different types of robots, environments, and obstacles, making it ideal for comprehensive testing.

In our experiments, we use the Fetch robot to evaluate the DRM framework in a table-pick scenario. This scenario consists of a table, a block obstacle, and a defined workspace. The motion-planning task involves navigating the robot's end-effector from a start configuration (q_{start}) to a goal configuration (q_{goal}) on the table. The setup enables us to test the efficiency and adaptability of the DRM algorithm in handling dynamic obstacles and achieving real-time path planning.

IV. Proposed Methods

This section outlines the steps involved in implementing the Dynamic Roadmap (DRM) algorithm for real-time motion planning in dynamic environments. The algorithm ensures efficient adaptation to workspace changes by selectively updating affected regions of the roadmap, thereby reducing computational overhead and maintaining performance in dynamic scenarios.

1. Workspace Representation

The algorithm begins by defining the workspace (W) and its corresponding configuration space (C). Obstacles within W are represented as dynamic entities that can change position, orientation, or shape over time. The configuration space (C) captures the valid and invalid states of the robot, ensuring safe navigation by avoiding collisions. This representation is crucial for addressing challenges posed by dynamic environments, where obstacle configurations are not fixed.

2. Roadmap Initialization

An initial roadmap $G = (V, E)$ is constructed using a sampling-based method such as Probabilistic Roadmaps (PRM). The nodes V represent collision-free configurations in C, while the edges E denote valid transitions between these configurations. This roadmap provides a precomputed structure that supports efficient reuse across multiple queries. By precomputing a connected graph, the algorithm enables quick responses to motion-planning tasks while minimizing redundant computations.

3. Dynamic Region Mapping

To facilitate efficient updates, each node and edge in the roadmap is associated with specific regions of the workspace. This mapping ensures that when a part of the workspace changes (e.g., a moving obstacle), the algorithm can quickly identify and localize the affected nodes and edges. This spatial mapping strategy is key to reducing the scope of updates, as only the impacted regions of the roadmap require revalidation.

4. Collision Checking

An efficient collision-checking mechanism is implemented to validate nodes and edges. Initially, all nodes and edges in the roadmap are validated against the workspace to ensure they are collision-free. During runtime, only nodes and edges associated with dynamically changing regions are rechecked, significantly reducing computational overhead. This selective approach ensures that the algorithm remains scalable and efficient, even in highly dynamic environments.

5. Dynamic Updates

When changes occur in the workspace, the algorithm performs the following steps:

1) Detect changes in the workspace (e.g., obstacle movements or additions).
2) Use the dynamic region mapping to identify the nodes and edges affected by these changes.
3) Mark affected nodes and edges as invalid if they are in collision with obstacles.
4) Repair or recompute edges to restore connectivity within the roadmap, ensuring it remains usable for motion planning.

This dynamic updating mechanism allows the roadmap to adapt to environmental changes without requiring a complete reconstruction.

6. Query Execution

For each motion-planning query (q_{start}, q_{goal}), the following steps are executed:

1) Validate the start and goal configurations to ensure they are collision-free and part of the roadmap.
2) Search for a valid path in the updated roadmap using a graph-based pathfinding algorithm, such as A*.
3) If no valid path exists, dynamically expand the roadmap by sampling new configurations and edges to find a feasible solution.

This process ensures that the algorithm can handle complex motion-planning queries even in challenging dynamic environments.

7. Path Retrieval and Repair

The solution path is obtained by running a search algorithm on the resulting roadmap after edge and vertex removal. If the solution path cannot be found in the roadmap, it implies that there are disconnected components in the roadmap that are not allowing us to reach the goal state from the start state. In this case, we use the RRTConnect planner to repair the roadmap.

7. Comparison Metric Setup with Naive Approach

We implemented a naive approach of this algorithm without discretization to compare the efficiency of the edge and node removal process with discretization. This approach simply iterates over all the edges and nodes of the constructed roadmap and removes the edges and nodes that collide with the object.

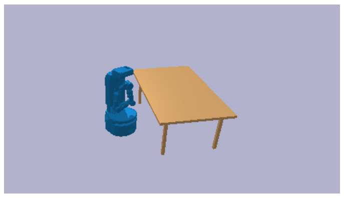


Fig. 1. Obstacle-free environment created for roadmap creation

V. Implementation

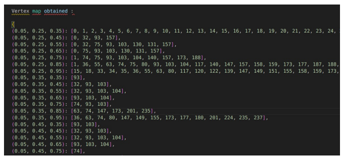
In this section, we explain how the Dynamic Roadmaps algorithm is implemented with **PyBullet** and the **Grapeshot API**. Along with the algorithm, we have also provided a short description of implementing the naive approach for the algorithm, which we have used for comparing the efficiency of our algorithm with performing it the naive way.

1. Initializing the Obstacle-Free Workspace

- **Obstacle-Free Roadmap Construction:** The roadmap is constructed using OMPL's `plan` and `constructRoadmap` functions. The use of `plan` ensures that the start and goal states are explicitly added to the roadmap, while `constructRoadmap` grows the roadmap by sampling configurations and connecting them with valid edges.
- **Static Elements:** The table is considered a static element and is not part of the dynamic updates. The workspace assumes that all dynamic obstacles will be placed on the table surface. Hence, when we perform `plan` and `constructRoadmap`, the table is considered while sampling, and then the edges and vertices are removed based on this roadmap.
- Figure 1 shows the obstacle-free workspace on which the construction of roadmaps will be done.

2. Discretization of the Workspace

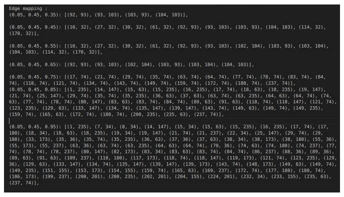
- **Purpose of Discretization:** The workspace is divided into discrete cells to efficiently localize the regions affected by dynamic obstacles. This mapping allows the roadmap's vertices and edges to be associated with specific grid cells.
- **Implementation:**
  - A cube is moved systematically across the workspace grid using `Pose` and the `set_joint_positions` function from Grapeshot.
  - Collision checks are performed for each cell using PyBullet to test if:
    - **Vertices:** The robot configurations represented by the roadmap vertices collide with the cube using the `isValid` function of the planner, which calls the Grapeshot's collision-checking function internally.
    - **Edges:** The robot's motion between two vertices (approximated as a swept volume) intersects with



```
Vertex map obtained :

{
(0.05, 0.25, 0.35): [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
(0.05, 0.25, 0.45): [0, 32, 93, 157],
(0.05, 0.25, 0.55): [0, 32, 75, 93, 103, 130, 131, 157],
(0.05, 0.25, 0.65): [0, 75, 93, 103, 130, 131, 157],
(0.05, 0.25, 0.75): [1, 74, 75, 93, 103, 104, 140, 157, 173, 188],
(0.05, 0.25, 0.85): [1, 36, 55, 63, 74, 75, 80, 93, 103, 104, 117, 140, 147, 157, 158, 159, 173, 177, 187, 188,
(0.05, 0.25, 0.95): [15, 18, 33, 34, 35, 36, 55, 63, 80, 117, 120, 122, 139, 147, 149, 151, 155, 158, 159, 173,
(0.05, 0.35, 0.35): [93],
(0.05, 0.35, 0.45): [32, 93, 103],
(0.05, 0.35, 0.55): [32, 93, 103, 104],
(0.05, 0.35, 0.65): [93, 103, 104],
(0.05, 0.35, 0.75): [74, 93, 103],
(0.05, 0.35, 0.85): [63, 74, 147, 173, 201, 235],
(0.05, 0.35, 0.95): [36, 63, 74, 80, 147, 149, 155, 173, 177, 180, 201, 224, 235, 237],
(0.05, 0.45, 0.35): [93, 103],
(0.05, 0.45, 0.45): [32, 93, 103],
(0.05, 0.45, 0.55): [32, 93, 103, 104],
(0.05, 0.45, 0.65): [93, 103, 104],
(0.05, 0.45, 0.75): [74],
```

Fig. 2. Vertex Mapping obtained



```
Edge mapping :
(0.05, 0.45, 0.35): [(92, 93), (93, 103), (103, 93), (104, 103)],

(0.05, 0.45, 0.45): [(10, 32), (27, 32), (30, 32), (61, 32), (92, 93), (93, 103), (103, 93), (104, 103), (114, 32),
(170, 32)],

(0.05, 0.45, 0.55): [(10, 32), (27, 32), (30, 32), (61, 32), (92, 93), (93, 103), (102, 104), (103, 93), (103, 104),
(104, 103), (114, 32), (170, 32)],

(0.05, 0.45, 0.65): [(92, 93), (93, 103), (102, 104), (103, 93), (103, 104), (104, 103)],

(0.05, 0.45, 0.75): [(17, 74), (21, 74), (29, 74), (35, 74), (63, 74), (64, 74), (77, 74), (78, 74), (83, 74), (84,
74), (118, 74), (121, 74), (134, 74), (143, 74), (149, 74), (159, 74), (172, 74), (180, 74), (237, 74)],
(0.05, 0.45, 0.85): [(1, 235), (14, 147), (15, 63), (15, 235), (16, 235), (17, 74), (18, 63), (18, 235), (19, 147),
(21, 74), (25, 147), (29, 74), (35, 74), (35, 235), (36, 63), (37, 63), (63, 74), (63, 235), (64, 63), (64, 74), (74,
63), (77, 74), (78, 74), (80, 147), (83, 63), (83, 74), (84, 74), (89, 63), (91, 63), (118, 74), (118, 147), (121, 74),
(123, 235), (129, 63), (133, 147), (134, 74), (135, 147), (139, 147), (143, 74), (149, 63), (149, 74), (149, 235),
(159, 74), (165, 63), (172, 74), (180, 74), (200, 235), (235, 63), (237, 74)],

(0.05, 0.45, 0.95): [(1, 235), (7, 34), (8, 34), (14, 147), (15, 34), (15, 63), (15, 235), (16, 235), (17, 74), (17,
180), (18, 34), (18, 63), (18, 235), (19, 34), (19, 147), (21, 74), (21, 237), (22, 34), (25, 147), (29, 74), (29,
180), (33, 173), (35, 36), (35, 74), (35, 235), (36, 63), (37, 36), (37, 63), (38, 34), (38, 173), (38, 180), (55, 36),
(55, 173), (55, 237), (63, 36), (63, 74), (63, 235), (64, 63), (64, 74), (70, 36), (74, 63), (74, 180), (74, 237), (77,
74), (78, 74), (78, 237), (80, 147), (82, 173), (83, 34), (83, 63), (83, 74), (84, 74), (86, 237), (88, 36), (89, 36),
(89, 63), (91, 63), (109, 237), (110, 180), (117, 173), (118, 74), (118, 147), (119, 173), (121, 74), (123, 235), (129,
36), (129, 63), (133, 147), (134, 74), (135, 147), (139, 147), (139, 173), (143, 74), (148, 173), (149, 63), (149, 74),
(149, 235), (151, 155), (153, 173), (154, 155), (159, 74), (165, 63), (169, 237), (172, 74), (177, 180), (180, 74),
(186, 173), (199, 237), (200, 201), (200, 235), (202, 201), (204, 155), (224, 201), (232, 34), (233, 155), (235, 63),
(237, 74)],
```

Fig. 3. Edge Mapping obtained

the cube using the `checkMotion` function of OMPL.

∗ Fig 2 and 3 show an example of the vertex and edge mapping generated when the algorithm is executed. In `Python`, we get this by using the `Dictionary` data structure, such that the center coordinates of the cube are the **keys** of the hashmap, and this is mapped to a list of edges or vertices that are contained in the discrete cell centered at those coordinates.

3. Handling Dynamic Obstacles

- **Obstacle Addition:** Dynamic obstacles are introduced into the workspace, placed on the table surface. The code uses Grapeshot's skeleton management functions to add obstacles dynamically. Figure 4 shows the final environment in which the algorithm is implemented. The cube is the obstacle that is added in the environment.
- **Intersection Testing:** The intersection of the obstacle with the roadmap's vertices and edges is determined based on their mapped grid cells. The Axis-Aligned Bounding Box (AABB) Intersection test is used to identify which cells collide with the obstacle.
- **Grapeshot Functions Used:**
  - `World.create_headless_clone`: Clones the workspace without rendering for efficient collision checking. In this cloned world, the robot moves to various different poses according to the vertices in the roadmap to perform collision checking.

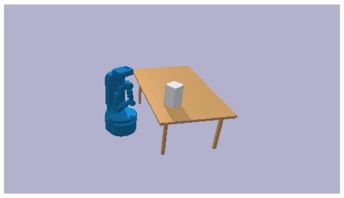


Fig. 4. Environment with the obstacle added

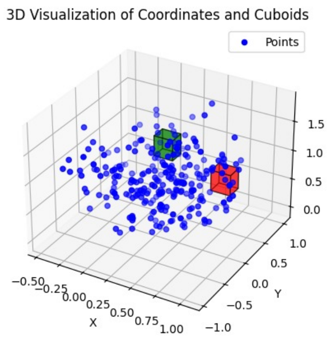


Fig. 5. Vertices of the roadmap before collision checking. The points represent the end-effector positions

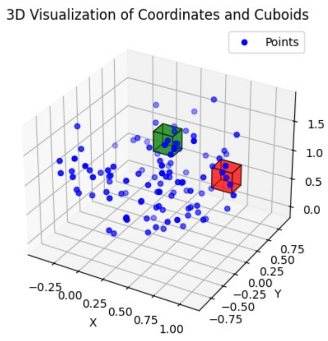


Fig. 6. Vertices obtained after collision-checking through discretization

4. Removing Colliding Nodes and Edges

- **Vertex Removal:** For each colliding grid cell, the vertices mapped to that cell are invalidated and removed from the roadmap using `PlannerData.removeVertex`.
- **Edge Removal:** Similarly, edges mapped to colliding grid cells are invalidated and removed using `PlannerData.removeEdge`. This ensures that no paths traverse through the obstacle-affected areas.
- **Efficiency:** By using the discretized mapping, the code avoids redundant collision checks across the entire roadmap, focusing only on the affected regions.
- Figures 5 and 6 show the result of edge and vertex removal. **Figure 5** shows the end-effector positions of the robot and represents every vertex constructed in the roadmap in the obstacle-free environment. The obstacles in this figure are added for comparison purposes and do not exist when the actual roadmap is being constructed. **Figure 6** shows the end-effector positions after performing edge-removal through the discretization process. We can see from the figure that the density of points behind the obstacle has decreased, as a result of the addition of the obstacles, which end up blocking some configurations of the robot. The figure shows that the algorithm was successfully able to remove the affected vertices and edges.

5. Finding the Final Path

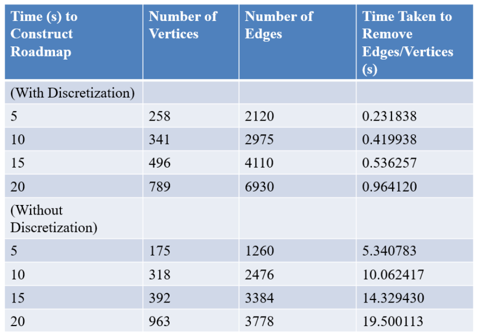
- **Shortest Path Search:** After updating the roadmap, the shortest path between the start and goal states is computed using NetworkX's Dijkstra algorithm. For this purpose, the vertices and edges of the roadmap are converted into a graph.
- **Handling Disconnected Components:** If no valid path exists in the updated roadmap, the code employs RRT-Connect to bridge the disconnected subgraphs. Partial paths from the reachable subgraphs are merged with the RRTConnect path to form a complete solution.
- **Path Optimization:** The final path is converted into a geometric path using Grapeshot's trajectory functions, and smoothing is applied to ensure the path is feasible and efficient.
- **Visualization:** The updated path is visualized in PyBullet to demonstrate the robot's motion planning in the simulated environment.

6. Key Features of the Code

- **Dynamic Updates:** The roadmap is updated dynamically by removing invalid nodes and edges rather than reconstructing it from scratch, ensuring computational efficiency.
- **Fallback Mechanisms:** The use of RRTConnect ensures that disconnected components can still be connected, providing robustness in complex scenarios.
- **Integration of Grapeshot and PyBullet:** The code seamlessly integrates Grapeshot's planning utilities and PyBullet's simulation capabilities for collision checking, discretization, and visualization.
- **Efficient Discretization:** By associating roadmap elements with workspace cells, the code minimizes the computational cost of handling dynamic obstacles.

A. Naive Implementation Without Discretization

This approach demonstrates a naive implementation of dynamic roadmaps for motion planning, handling environmental changes without discretizing workspace or configuration space updates. Initially, a roadmap is constructed in an obstacle-free configuration space using PRM, representing sampled configurations as vertices and feasible motions as edges. After adding a new obstacle, such as a cube, the roadmap is updated by revalidating vertices and edges; invalid configurations are removed, and edges passing through the obstacle are pruned. A graph search algorithm, such as Dijkstra's, is then used to find a path in the updated roadmap. If no direct path is found due to disconnected components, the approach identifies reachable states from the start and goal and uses RRTConnect to bridge the disjoint regions. The resulting path is constructed by merging three segments: the roadmap-based path from the start to a reachable state, the RRTConnect-generated path connecting the components, and the roadmap-based path from the reachable state to the goal. The final path is converted into a time-parameterized trajectory for visualization. This method partially reuses pre-existing information while dynamically adjusting to environmental changes but relies on brute-force validation, making it straightforward yet computationally naive. By combining PRM for broad coverage and RRTConnect for focused reconnection, the approach offers a hybrid solution to dynamic motion planning challenges.



Time (s) to Construct Roadmap	Number of Vertices	Number of Edges	Time Taken to Remove Edges/Vertices (s)
(With Discretization)			
5	258	2120	0.231838
10	341	2975	0.419938
15	496	4110	0.536257
20	789	6930	0.964120
(Without Discretization)			
5	175	1260	5.340783
10	318	2476	10.062417
15	392	3384	14.329430
20	963	3778	19.500113

Fig. 7. Table comparing times for Fetch robot

Summary

This code demonstrates the **Dynamic Roadmap (DRM)** algorithm's capability to handle dynamic environments efficiently. It uses Grapeshot for roadmap management, trajectory generation, and workspace setup, while **PyBullet** provides accurate simulations and collision detection. The combination of these tools ensures a modular, efficient, and scalable implementation suitable for various motion-planning tasks.

VI. Results and Comparisons

Comparison with Naive Approach

To evaluate the efficiency of our Dynamic Roadmap (DRM) algorithm, we conducted a comparative analysis against a naive approach. The naive approach involves traversing the entire graph to check for collisions, making it computationally expensive and time-consuming. In contrast, the DRM algorithm selectively updates only the affected portions of the roadmap, showcasing its adaptability and efficiency.

The comparison was performed by introducing a single cuboidal obstacle into the environment. The obstacle was placed on the table at coordinates $(0.4, 0.8, 0.8)$ in the *x, y, z* space, with dimensions $0.2 \times 0.3 \times 0.35$. This scenario allows us to analyze the performance of both approaches in handling dynamic changes to the workspace.

The metric for comparison was the time taken to remove invalid nodes and edges from the roadmap due to the addition of the obstacle. The DRM algorithm's ability to localize updates using dynamic region mapping significantly reduces the computational overhead compared to the naive approach. This comparison highlights the advantages of the DRM algorithm in efficiently adapting to changes in the environment while maintaining robust performance in motion planning.

The table demonstrates the significant efficiency improvements achieved through discretization in the Dynamic Roadmap (DRM) algorithm. With discretization, the time taken to remove invalid edges and vertices is drastically reduced compared to the naive approach. For example, at 5 seconds, the algorithm with discretization processes 258 vertices and 2120 edges in just 0.231 seconds, whereas the naive approach takes 5.34 seconds to handle 175 vertices and 1260 edges. As the roadmap size increases, the scalability of discretization becomes evident, with only a slight rise in computational time (e.g., 0.964 seconds for 789 vertices and 6930 edges at 20 seconds), whereas the naive approach becomes increasingly inefficient, requiring 19.50 seconds for 963 vertices and 3778 edges. These results highlight that discretization significantly enhances the real-time adaptability and efficiency of DRM in dynamic environments.

Visualization of the path taken by the robot for the sample run is given in the following link: Path Visualization Link All of the implementations of the problems present in grapeshot can be visualized through the naive approach whereas the dynamic roadmaps are only implemented with the cube obstacle owing to the time taken in the discretization process.

VII. Limitations of This Project

The Dynamic Roadmap (DRM) framework presents significant advantages in handling dynamic environments, offering robustness and reduced computational overhead by reusing portions of the roadmap instead of rebuilding it from scratch. However, the framework's performance is influenced by the resolution of discretization, which governs the granularity of edge and vertex mapping to the workspace. A low-resolution roadmap might fail to detect subtle environmental changes, leading to disconnected components or inefficient paths, while a high-resolution roadmap increases the computational cost and memory consumption. Balancing these trade-offs is essential to optimize the framework's performance.

Another challenge lies in the real-time processing of workspace changes. The DRM framework assumes that dynamic changes, such as obstacle movements, can be detected and integrated seamlessly. However, in highly cluttered or rapidly changing environments, the time complexity of vertex and edge removal—though improved through discretization—can still pose issues, especially for scenarios requiring instantaneous path updates. While the integration of subgraph extraction and RRT-Connect provides a fallback for disconnected components, this method introduces additional computational steps, potentially affecting time-sensitive applications.

Finally, the framework's current implementation has been tested primarily in simulation, such as 2D and 3D workspaces with robots like Fetch and Panda. Although these simulations demonstrate significant improvements in execution times and pathfinding success rates, real-world applications may face additional constraints. Hardware limitations, sensor inaccuracies, and unpredictable dynamics in physical environments could hinder the robustness of the DRM framework. Moreover, tuning discretization resolution for diverse environments remains a key challenge, as it directly impacts computational efficiency and path quality. Addressing these limitations through further testing in real-world scenarios is critical to validate and refine the framework for practical use.

VIII. Project Task Distribution

The project involves several key tasks to implement and evaluate the Dynamic Roadmap (DRM) framework in both two-dimensional and three-dimensional workspaces. Below is a detailed description of the tasks and their distribution among the team members: Ameya, Soumik, and Pranjali.

1. 2D DRM Implementation and Evaluation

Task: Implement the initial DRM framework. Develop basic Python-based plotting to visualize and evaluate the roadmap in a 2D workspace. **Assigned to: Ameya** Ameya will lead the implementation of the 2D DRM and ensure the accuracy of the initial roadmap construction. He will also handle the plotting and basic performance evaluation of the DRM in this simplified setting.

2. Transition to 3D Workspace and Integration with Grapeshot API

Task: Set up the PyBullet simulation environment for 3D workspaces and integrate the DRM framework. Additionally, understand and implement the Grapeshot API for seamless communication between the motion planner and the PyBullet simulation. This involves exploring the Grapeshot interface, its interaction with PyBullet, and leveraging OMPL functionalities to perform tasks in pre-built simulations. **Assigned to: Soumik and Pranjali** Soumik and Pranjali will collaboratively configure the PyBullet environment and implement the DRM framework within the 3D workspace. They will work together to understand the Grapeshot interface, ensuring its smooth integration with PyBullet and OMPL for task execution in dynamic scenarios.

3. Performance Evaluation and Comparative Analysis

Task: Collect performance metrics (e.g., computational efficiency, success rate, path quality) from both 2D and 3D simulations. Compare the results of the DRM framework with traditional planners. **Assigned to: Ameya and Pranjali** Ameya will analyze the 2D simulation results, while Pranjali will handle the evaluation for the 3D workspace. Both will collaborate to compare the results and present insights.

4. Documentation and Reporting

Task: Prepare detailed documentation and final reports, including methodology, results, and conclusions. **Assigned to: Soumik** Soumik will compile the final report, ensuring that all aspects of the project are thoroughly documented. He will also coordinate input from Ameya and Pranjali to ensure comprehensive coverage of all tasks.

5. Presentation and Final Review

Task: Create and deliver a presentation summarizing the project outcomes, including key findings and future directions. **Assigned to: Entire Team** All three members will collaborate on creating the presentation, with each member presenting their respective contributions and findings.

By distributing tasks evenly, each team member plays a crucial role in the successful implementation, evaluation, and documentation of the DRM framework.

References

[1] Leven, P., Hutchinson, S. (2002). A framework for real-time path planning in changing environments. *The International Journal of Robotics Research, 21*(12), 999–1030. https://doi.org/10.1177/0278364902021012001

[2] Kallmann, M., Mataric, M. (2004). Motion planning using dynamic roadmaps. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2004, 4399–4404. https://doi.org/10.1109/ROBOT.2004.1302373

[3] Yang, W., Merkt, W., Ivan, V., Li, Z., Vijayakumar, S. (2018). HDRM: A resolution complete dynamic roadmap for real-time motion planning in complex scenes. *IEEE Robotics and Automation Letters, 3*(1), 551–558. https://doi.org/10.1109/LRA.2017.2773669

[4] Guo, J., Xun, Z., Geng, S., Lin, Y., Xu, C., Gao, F. (2022). Dynamic free-space roadmap for safe quadrotor motion planning. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, 10523–10528. https://doi.org/10.1109/IROS47612.2022.9981775

[5] Sivaramakrishnan, A., Tangirala, S., Granados, E., Carver, N. R., Bekris, K. E. (2023). Roadmaps with gaps over controllers: Achieving efficiency in planning under dynamics. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023. https://doi.org/10.1109/IROS2023.00123