Quantitative Comparison of Various Path Planning Algorithms Implemented For Non-Holonomic Robots

1st Neeraj Laul
M. Eng Robotics
A. James Clark School Of Engineering
University Of Maryland, College Park
nslaul@umd.edu

2nd Kirti Kishore
M. Eng Robotics
A. James Clark School Of Engineering
University Of Maryland, College Park
kiki1@umd.edu

3rd Daya K
M. Eng Robotics
A. James Clark School Of Engineering
University Of Maryland, College Park
dotsv@umd.edu

***Abstract*—This work presents a comparative evaluation of path planning algorithms adapted for non-holonomic, differential drive robots. Expanding on the benchmarking framework by Ayawli et al., we implement and assess both sampling-based (RRT, RRT + A*, PRM variants) and graph-based (A*, Weighted A*) planners under differential drive constraints. Algorithms are tested in a ROS Gazebo simulation across diverse maps using a custom evaluation function. Results highlight trade-offs between path efficiency and exploration time, demonstrating the impact of non-holonomic constraints on classical planning methods.**

***Index Terms*—Path planning, Non-holonomic robots, Differential drive, RRT, A*, PRM, Motion planning, ROS.**

I. Introduction

Path planning for holonomic robots has been extensively studied, with numerous benchmarking frameworks and comparative analyses available in the literature. In contrast, comprehensive evaluations for non-holonomic mobile robots—particularly those with differential drive constraints—remain relatively sparse. Non-holonomic systems, such as differential drive robots, are subject to kinematic constraints that restrict their motion, rendering many traditional planning algorithms suboptimal or infeasible without significant adaptation.

Recent research has begun to address this gap. For instance, Kumar et al. proposed an enhanced A* algorithm that incorporates non-holonomic constraints, demonstrating improved performance in various scenarios [1]. Similarly, Nemec et al. modified the Hybrid A* method to better accommodate non-holonomic wheeled robots, allowing for multi-criterial adjustments considering factors like traveled distance and obstacle avoidance [2]. Additionally, Li et al. introduced a prioritized trajectory optimization approach for multiple non-holonomic mobile robots, effectively generating collision-free optimal trajectories in obstacle-rich environments [3].

Building upon these advancements, our study aims to provide a quantitative comparison of several widely used path planning algorithms, originally developed for holonomic systems, adapted to operate under non-holonomic constraints. This work draws direct inspiration from and extends the methodology of the "Comparative Analysis of Popular Mobile Robot Roadmap Path-Planning Methods" by Ayawli et al., which evaluates the performance of various RRT and PRM variants over large sets of 2D maps [4]. While that study focused primarily on holonomic robots in static environments, our work modifies and implements similar algorithms to address the complexities introduced by differential drive motion models.

Specifically, we focus on differential drive mobile robots and assess the performance of both graph search algorithms (A*, Weighted A*) and sampling-based algorithms (RRT, RRT + A*, PRM + Dijkstra, PRM + Weighted A*). By integrating differential drive constraints into these algorithms and benchmarking them in a controlled simulation environment using ROS Gazebo, we evaluate their efficiency, feasibility, and suitability for non-holonomic motion planning.

Our evaluation employs key performance metrics, including path efficiency and exploration time, to provide a comprehensive analysis of each algorithm's capabilities. Through this study, we aim to contribute to the growing body of knowledge on non-holonomic path planning and offer insights into the practical implementation of these algorithms for differential drive robots.

II. Background

This section provides definitions and explanations of key concepts and terminology used throughout this paper to aid reader comprehension.

- **Holonomic Robot:** A robot whose controllable degrees of freedom equal its total degrees of freedom, allowing movement in any direction without constraints.
- **Non-Holonomic Robot:** A robot with motion constraints limiting its instantaneous movements, e.g., differential drive robots that cannot move sideways.
- **Differential Drive Robot:** A mobile robot with two independently driven wheels on a common axis, constrained to move forward/backward and rotate.

- **A* Algorithm:** A graph search algorithm that finds the shortest path between nodes using heuristics to improve efficiency [5].
- **Rapidly-exploring Random Tree (RRT):** A sampling-based planner that incrementally builds a tree to explore the robot's configuration space efficiently [6].
- **Probabilistic Roadmap (PRM):** A planning method that constructs a roadmap by sampling valid configurations and connecting them to find feasible paths [7].
- **ROS (Robot Operating System):** An open-source framework providing libraries and tools to build robot applications [8].
- **Gazebo:** A simulation environment for testing robotic systems with physics and sensor models [9].

III. Related Work

A. Comparative Analysis of Popular Mobile Robot Roadmap Path-Planning Methods (Ayawli et al.)

Ayawli et al. [4] present a comprehensive benchmarking framework comparing several popular roadmap-based path planning algorithms, including variations of Rapidly-exploring Random Trees (RRT) and Probabilistic Roadmaps (PRM). Their evaluation covers 200 static 2D maps and focuses on metrics such as path length, computational efficiency, and success rate. While the study provides valuable insights into the performance of these planners for holonomic robots, it does not explicitly address non-holonomic constraints typical of differential drive robots. The methodology and metrics from this work serve as a foundation for our study, which adapts and extends these planners to non-holonomic systems.

B. Enhanced A for Non-Holonomic Mobile Robot Path Planning (Kumar et al.)*

Kumar et al. [1] propose modifications to the classical A* algorithm to incorporate non-holonomic constraints inherent in differential drive robots. Their approach improves path feasibility by integrating kinematic constraints directly into the search process and demonstrates enhanced performance in dynamic environments. This work highlights the importance of adapting graph search algorithms for practical deployment on constrained robotic platforms.

C. Hybrid A Path Planning for Nonholonomic Mobile Robots (Nemec et al.)*

Nemec et al. [2] extend the Hybrid A* algorithm to better accommodate non-holonomic wheeled robots by including multi-criterial optimization, such as balancing path length and obstacle clearance. Their method produces smooth and feasible trajectories respecting differential drive constraints, making it suitable for complex environments. This study provides insights into how classical planners can be enhanced for differential drive systems.

D. Prioritized Trajectory Optimization for Multiple Non-Holonomic Mobile Robots (Li et al.)

Li et al. [3] develop a prioritized trajectory optimization framework aimed at multiple non-holonomic robots operating in cluttered spaces. Their method generates collision-free, dynamically feasible trajectories while optimizing control effort. This approach extends beyond single-robot path planning and addresses challenges in multi-robot coordination under motion constraints.

IV. Methodology

This section outlines the framework used to implement and evaluate various path planning algorithms adapted for non-holonomic differential drive robots. The methodology includes planner integration, simulation setup, motion constraints, and performance evaluation.

A. Implementation Framework

All planners were implemented in Python using a modular structure that supports:

- Custom 2D map environments with rectangular and L-shaped obstacles.
- Obstacle inflation accounting for robot radius and user-defined clearance.
- RPM-based motion simulation for non-holonomic, differential-drive robots.
- Logging of performance metrics including runtime, path length, and jerkiness.
- Visualizations of node exploration and final path execution.

The seven planners evaluated in this study include:

- `A*`
- `Weighted A*`
- `PRM + Dijkstra`
- `PRM + Weighted A*`
- `RRT`
- `RRT + A*`
- `RRT*`

Each planner follows a consistent interface, allowing uniform evaluation under the same test conditions.

B. Simulation Setup

Multiple static 2D maps were designed to evaluate the planners under varied environmental complexity. Maps include bottlenecks, narrow corridors, and occluded goal regions to stress-test exploration and path optimization.

For each test:

- Start and goal positions were selected in valid free space.
- Wheel RPMs were set to constant values (e.g., 60, 60) across all planners.
- Planners were evaluated offline using deterministic inputs to eliminate runtime variability.

To ensure safe navigation, all obstacles in the map were inflated based on the robot's geometry. The inflation margin was computed as the sum of the robot's radius and desired

clearance, effectively bloating each obstacle polygon outward. This process was implemented using vector-based offsetting at each obstacle vertex, creating a buffer zone that accounts for both physical footprint and a safety buffer. Inflated obstacles were used for both collision checking and visual representation during planning.

All motion and collision checks were performed using discretized canvas maps, and all planner outputs were visualized and stored for post-processing.

C. Differential Drive Adaptation

Each planner was adapted to simulate differential-drive motion using the unicycle model. The kinematics are governed by:

$$\Delta x = v \cdot \cos(\theta) \cdot dt, \quad \Delta y = v \cdot \sin(\theta) \cdot dt, \quad \Delta\theta = \omega \cdot dt$$

Where:

- v and ω are derived from wheel RPMs,
- dt is the integration timestep.

These dynamics were used in:

- Successor generation for A*-based planners,
- Edge validity for PRM-based planners,
- Tree extension for RRT and RRT*-based planners.

All simulated paths respect the robot's kinematic constraints.

D. Evaluation Metrics

Each planner was evaluated using the following metrics:

- **Success Rate:** Fraction of maps for which a valid path was found.
- **Planning Time:** Time taken to compute a valid path.
- **Path Length:** Total Euclidean distance of the final path.
- **Jerkiness:** Smoothness metric derived from cumulative heading angle variation.
- **Simulated Execution Time:** Time required to virtually traverse the path under constant RPM inputs.

The top three successful runs for each planner (based on execution time) were averaged to provide consistent performance comparisons.

E. Data Flow Overview

Figure 1 illustrates the modular data pipeline, where user inputs, map setup, and planner execution are synchronized with a shared evaluation function. This structure allows each planner to be evaluated in a controlled and consistent simulation environment, ensuring fair comparison across all implemented algorithms.

F. Process Flowchart

Given below is the process flowchart demonstrating the data flow between the different components of the code. As all of the planners provide their metrics to the common evaluation function, the chart below is representative of the data flow throughout the entire project.

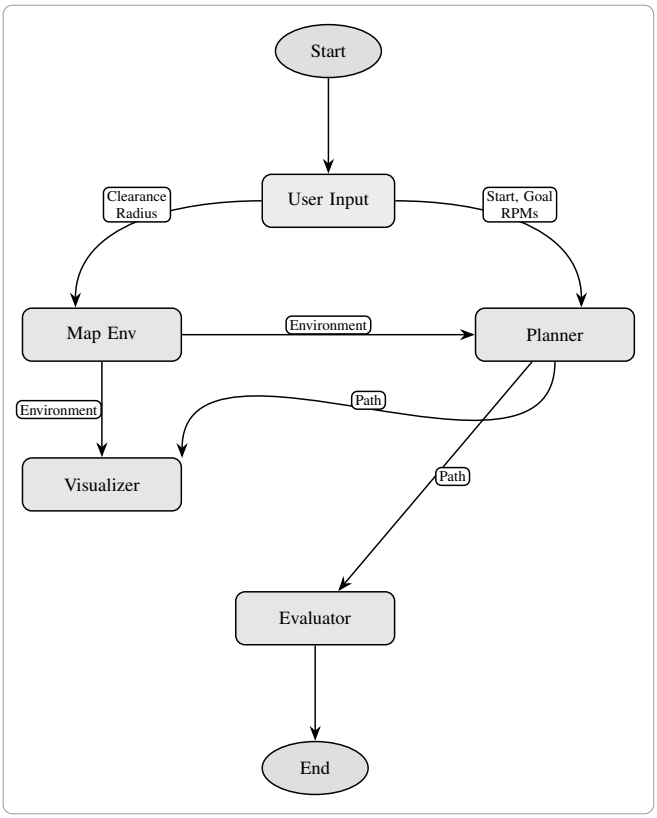


Fig. 1. Flowchart of the path planning process.

G. Pseudocodes For Algorithms Used

1) PRM with Dijkstra's Algorithm: The Probabilistic Roadmap Method (PRM) constructs a graph by randomly sampling collision-free points in the environment and connecting each node to its nearest neighbors if the path between them is collision-free. This creates a roadmap representing feasible paths in the configuration space.

Dijkstra's algorithm is a classical graph search algorithm used to find the shortest path between nodes in a weighted graph. It guarantees the optimal path in terms of cumulative cost by systematically exploring the graph from the start node, updating the cost of reaching each node until the goal is reached.

In this approach, Dijkstra's algorithm is applied on the PRM-generated graph to compute the minimum-cost path from the start to the goal position. The synergy of PRM and Dijkstra leverages the probabilistic exploration of feasible paths by PRM and the optimal pathfinding capability of Dijkstra's algorithm to generate collision-free and cost-efficient routes. Additionally, to ensure the planned path respects the differential drive constraints of the robot, collision checking during edge creation incorporates RPM-based motion feasibility.

Algorithm 1 PRM with Dijkstra's Algorithm

Require: Map environment env, start position s, goal position g, number of samples N, number of neighbors k, RPM values rpm_1, rpm_2
Ensure: Path π from s to g or $\emptyset$ if no path exists

Initialize vertex set $V \leftarrow \{s, g\}$ and edge set $E \leftarrow \emptyset$
for $i = 1$ to N **do**
  $p \leftarrow$ SampleFreeSpace(env)
  if not IsInObstacle(p, env) **then**
    $V \leftarrow V \cup \{p\}$
  end if
end for
Add bridge points to V: $(270, 50), (270, 150), (270, 250)$
$kdtree \leftarrow$ BuildKDTree(V)
for all $v \in V$ **do**
  $neighbors \leftarrow kdtree$.Query($v$, $k + 1$)
  for all $u \in neighbors \setminus \{v\}$ **do**
    if CollisionFree(v, u, env, rpm_1, rpm_2) **then**
      $cost \leftarrow$ EuclideanDistance(v, u)
      $E \leftarrow E \cup \{(v, u, cost)\}$
    end if
  end for
end for
$\pi \leftarrow$ Dijkstra(V, E, s, g)
if $\pi \neq \emptyset$ **then**
  $\pi_{sim} \leftarrow$ SimulateDifferentialDrive(π, rpm_1, rpm_2, env)
  return π_{sim}
end if
return $\emptyset$

2) Weighted A Algorithm:* Weighted A* is a heuristic-based graph search algorithm that extends the classical A* by introducing a weight factor ϵ to the heuristic function. This weighting allows the algorithm to trade off optimality for speed by prioritizing nodes that appear closer to the goal, potentially finding solutions faster but possibly less optimal.

In this implementation, the planner explores the state space discretized by position and orientation, incorporating differential drive constraints by simulating possible robot motions generated from the RPM values. The algorithm maintains an open priority queue ordered by the weighted cost and a visited set to track explored nodes. It terminates when a node sufficiently close to the goal is found according to a heuristic threshold.

The synergy of Weighted A* with differential drive simulation ensures that generated paths are kinematically feasible and computationally efficient, balancing exploration speed and path quality.

Algorithm 2 Weighted A* Algorithm

Require: Map environment env, start pose $s = (x_s, y_s, \theta_s)$, goal pose $g = (x_g, y_g, \theta_g)$, RPM values rpm_1, rpm_2, weight ϵ
Ensure: Path π from s to g or $\emptyset$ if no path exists

Initialize priority queue $open$
Initialize empty set $visited$
$startNode \leftarrow$ Node($x_s, y_s, 0, Heuristic(x_s, y_s, g), \theta_s$)
Push ($\epsilon \times startNode.totalCost$, $startNode$) to $open$
while $open$ is not empty **do**
  $current \leftarrow$ Pop from $open$
  $key \leftarrow$ Discretize($current$)
  if key in $visited$ and $visited[key].totalCost \leq current.totalCost$ **then**
    continue
  end if
  $visited[key] \leftarrow current$
  if Heuristic($current.x, current.y, g$) $\leq$ threshold **then**
    $\pi \leftarrow$ ReconstructPath($current$)
    return π
  end if
  for all $move$ in MoveSet(rpm_1, rpm_2) **do**
    $(neighbor, move) \leftarrow$ MotionSim($current, move, env$)
    if $neighbor \neq null$ and WithinBounds($neighbor, env$) **then**
      $nKey \leftarrow$ Discretize($neighbor$)
      if $nKey \notin visited$ or $visited[nKey].totalCost > neighbor.totalCost$ **then**
        $neighbor.move \leftarrow move$
        Push ($\epsilon \times neighbor.totalCost$, $neighbor$) to $open$
      end if
    end if
  end for
end while
return $\emptyset$

3) Rapidly-exploring Random Tree (RRT) Algorithm: The RRT algorithm is a sampling-based planner that incrementally builds a tree rooted at the start configuration by randomly sampling points in the environment. At each iteration, the planner extends the tree toward a randomly sampled point by simulating feasible motions based on the robot's differential drive constraints. This approach efficiently explores large and high-dimensional spaces, making it well-suited for complex environments.

In this implementation, the algorithm uses RPM values to simulate differential drive motion primitives when extending the tree. The search continues until the goal region is reached or a maximum number of iterations is exceeded. Once a feasible path is found, it is smoothed before being returned.

Algorithm 3 RRT Algorithm

Require: Map environment env, start pose s, goal pose g, RPM values rpm_1, rpm_2, max iterations $maxIter$, goal sample rate γ

Ensure: Path π from s to g or $\emptyset$ if no path exists

Initialize tree T with root node s
for $i = 1$ to $maxIter$ **do**
  $x_{rand} \leftarrow$ SampleFree(env, g, γ)
  $x_{nearest} \leftarrow$ Nearest(T, x_{rand})
  $x_{new} \leftarrow$ SimulateMotion($x_{nearest}$, x_{rand}, rpm_1, rpm_2, env)
  if x_{new} is valid **then**
    Set parent of x_{new} to $x_{nearest}$
    Add x_{new} to T
    if Distance(x_{new}, g) $<$ threshold **then**
      $g.parent \leftarrow x_{new}$
      Add g to T
      $\pi \leftarrow$ ReconstructPath(g)
      $\pi_{smooth} \leftarrow$ SmoothPath(π, env)
      return π_{smooth}
    end if
  end if
end for
return $\emptyset$

4) RRT with A Path Extraction:* This hybrid algorithm combines the exploration efficiency of RRT with the optimal path-finding capability of A*. Unlike traditional RRT that returns a single path to the goal, this approach maintains a graph of connections during tree expansion. When a node reaches the goal region, the algorithm invokes A* to extract the shortest path over the constructed graph, rather than relying solely on the direct parent-chain path in the RRT tree.

During tree expansion, motion primitives based on differential drive constraints (RPM-based actions) are used to simulate robot motion. Valid transitions are added to both the tree and an undirected graph. Once the goal is connected, A* is run over this graph to compute an optimal path from start to goal.

This method enhances RRT's exploration ability with A*'s optimization, while maintaining feasibility for non-holonomic robots.

Algorithm 4 RRT with A* Path Extraction

Require: Map environment env, start pose s, goal pose g, RPM values rpm_1, rpm_2, max iterations $maxIter$, goal sample rate γ

Ensure: Path π from s to g or $\emptyset$ if no path exists

Initialize tree with s, node list $\mathcal{T} \leftarrow \{s\}$, coordinate list $\mathcal{C} \leftarrow \{s.position\}$
Initialize KDTree from $\mathcal{C}$, empty graph $G \leftarrow \emptyset$
for $i = 1$ to $maxIter$ **do**
  $x_{rand} \leftarrow$ SAMPLEFREE(env, g, γ)
  $x_{near} \leftarrow$ NEAREST($\mathcal{T}, x_{rand}$)
  $best \leftarrow null$, $d_{min} \leftarrow \infty$
  for all $rpm \in$ MOVESET(rpm_1, rpm_2) **do**
    $x_{new} \leftarrow$ SIMULATEMOTION(x_{near}, rpm, env)
    if $x_{new} \neq null$ **then**
      $d \leftarrow$ DISTANCE(x_{new}, x_{rand})
      if $d < d_{min}$ **then**
        $d_{min} \leftarrow d$, $best \leftarrow x_{new}$
      end if
    end if
  end for
  if $best \neq null$ **then**
    Add $best$ to $\mathcal{T}$ and $best.position$ to $\mathcal{C}$
    Update KDTree from $\mathcal{C}$
    Add edge $(x_{near}, best)$ to G with weight d_{min}
    if DISTANCE($best, g$) $<$ threshold **then**
      Set $g.parent \leftarrow best$
      Add edge $(best, g)$ to G with weight DISTANCE($best, g$)
      $\pi_{raw} \leftarrow$ RECONSTRUCTPATH(g)
      $\pi \leftarrow$ ASTAREXTRACTPATH($G, s.position, g.position$)
      return π
    end if
  end if
end for

return $\emptyset$

5) PRM with Weighted A Algorithm:* This approach combines the roadmap generation capability of PRM with the efficient goal-biased search of Weighted A*. PRM is used to sample the configuration space and create a graph of feasible paths between randomly selected, collision-free nodes. Once the graph is constructed, Weighted A* is applied to search for the optimal path between the start and goal, leveraging a weighted heuristic to accelerate convergence at the expense of optimality.

To accommodate the differential drive constraints of the robot, the collision checking between nodes in the PRM graph uses motion simulation based on RPM values. The resulting path from Weighted A* is then simulated to ensure its feasibility under the robot's kinematic limits.

Algorithm 5 PRM with Weighted A* Algorithm

Require: Map environment env, start position s, goal position g, number of samples N, number of neighbors k, RPM values rpm_1, rpm_2, weight ϵ
Ensure: Path π from s to g or $\emptyset$ if no path exists
  Initialize vertex set $V \leftarrow \{s, g\}$ and edge set $E \leftarrow \emptyset$
  for $i = 1$ to N **do**
    $p \leftarrow$ SampleFreeSpace(env)
    if not IsInObstacle(p, env) **then**
      $V \leftarrow V \cup \{p\}$
    end if
  end for
  Add bridge points: $(270, 50), (270, 150), (270, 250)$
  $kdtree \leftarrow$ BuildKDTree(V)
  for all $v \in V$ **do**
    $neighbors \leftarrow kdtree$.Query($v$, $k + 1$)
    for all $u \in neighbors \setminus \{v\}$ **do**
      if CollisionFree(v, u, env, rpm_1, rpm_2) **then**
        $cost \leftarrow$ EuclideanDistance(v, u)
        $E \leftarrow E \cup \{(v, u, cost)\}$
      end if
    end for
  end for
  $\pi \leftarrow$ WeightedAStar(V, E, s, g, ϵ)
  if $\pi \neq \emptyset$ **then**
    $\pi_{sim} \leftarrow$ SimulateDifferentialDrive(π, rpm_1, rpm_2, env)
    return π_{sim}
  end if
  return $\emptyset$

Algorithm 6 A* Algorithm

Require: Map environment env, start pose $s = (x_s, y_s, \theta_s)$, goal pose $g = (x_g, y_g, \theta_g)$, RPM values rpm_1, rpm_2
Ensure: Path π from s to g or $\emptyset$ if no path exists
  Initialize priority queue $open$
  $visited \leftarrow \emptyset$
  $startNode \leftarrow$ Node($x_s, y_s, 0, Heuristic(x_s, y_s, g), \theta_s$)
  Push ($startNode.totalCost$, $startNode$) to $open$
  while $open \neq \emptyset$ **do**
    $current \leftarrow$ Pop from $open$
    $key \leftarrow$ Discretize($current$)
    if $key \in visited$ and $visited[key].totalCost \leq current.totalCost$ **then**
      continue
    end if
    $visited[key] \leftarrow current$
    if Heuristic($current.x, current.y, g$) $\leq$ threshold **then**
      $\pi \leftarrow$ ReconstructPath($current$)
      return π
    end if
    for all $move \in$ MoveSet(rpm_1, rpm_2) **do**
      $(neighbor, move) \leftarrow$ MotionSim($current, move, env$)
      if $neighbor \neq null$ and WithinBounds($neighbor, env$) **then**
        $nKey \leftarrow$ Discretize($neighbor$)
        if $nKey \notin visited$ or $visited[nKey].totalCost > neighbor.totalCost$ **then**
          $neighbor.move \leftarrow move$
          Push ($neighbor.totalCost$, $neighbor$) to $open$
        end if
      end if
    end for
  end while
  return $\emptyset$

6) A Algorithm:* A* is a classic graph-based path planning algorithm that combines the actual cost to reach a node and an estimated cost (heuristic) to the goal, making it both complete and optimal under admissible heuristics. It searches the discretized configuration space by expanding the most promising node first, guided by a priority queue ordered by the sum of cost and heuristic.

In this implementation, A* is adapted for non-holonomic differential drive robots. The robot's motion is simulated using a predefined set of RPM-based actions to generate feasible successors, and the algorithm ensures kinematic validity of paths through simulation-based expansion. The algorithm terminates when a node reaches within a defined threshold of the goal, returning a reconstructed path based on parent relationships. The implementation of the algorithm and the simulation is kept in line with the implementation shown in Project 3 Phase 2 of the course - Planning for Autonomous Robots at the University of Maryland, College Park. The complete code base for the project can be found on the author's GitHub page [10]

7) RRT Algorithm:* RRT* is an asymptotically optimal variant of the RRT algorithm that not only explores the configuration space but also rewires the tree to minimize path cost as it grows. Unlike standard RRT, which greedily extends toward random samples, RRT* maintains a cost-to-come metric for each node and dynamically rewires local connections when better paths are discovered.

In our implementation, motion primitives are based on differential-drive kinematics simulated using predefined RPM values. Each new node is inserted only if it meets collision and kinematic feasibility constraints. A rewiring step is performed to connect nearby nodes to the new node if doing so reduces their cost-to-come.

Algorithm 7 RRT* with Differential Drive and KD-Tree

Require: Map environment env, start pose s, goal pose g, RPM values rpm_1, rpm_2, max iterations $maxIter$, goal sample rate γ, rewire radius r
Ensure: Path π from s to g or $\emptyset$ if no path found

Initialize tree T with root node s
Initialize KD-Tree with s's position
for $i = 1$ to $maxIter$ **do**
  $x_{rand} \leftarrow$ SampleFree(env, g, γ)
  $x_{nearest} \leftarrow$ Nearest(T, x_{rand}, KDTree)
  $x_{new} \leftarrow$ **None**, $d_{min} \leftarrow \infty$
  for all $(rpm_l, rpm_r) \in$ MotionSet(rpm_1, rpm_2) **do**
    $x_{cand} \leftarrow$ SimulateMotion($x_{nearest}$, (rpm_l, rpm_r))
    if x_{cand} is valid and Distance(x_{cand}, x_{rand}) $< d_{min}$ **then**
      $x_{new} \leftarrow x_{cand}$, $d_{min} \leftarrow$ Distance(x_{cand}, x_{rand})
    end if
  end for
  if x_{new} is valid **then**
    $X_{near} \leftarrow$ Near(T, x_{new}, radius r, using KDTree)
    $x_{min} \leftarrow x_{nearest}$, $c_{min} \leftarrow$ Cost($x_{nearest}$) + Distance($x_{nearest}$, x_{new})
    for all $x_{near} \in X_{near}$ **do**
      if LineCollisionFree(x_{near}, x_{new}) and Cost(x_{near}) + Distance(x_{near}, x_{new}) $< c_{min}$ **then**
        $x_{min} \leftarrow x_{near}$, $c_{min} \leftarrow$ Cost(x_{near}) + Distance(x_{near}, x_{new})
      end if
    end for
    Set parent of x_{new} to x_{min}, set cost to c_{min}, add x_{new} to T
    Add x_{new} to KD-Tree
    for all $x_{near} \in X_{near}$ **do**
      if LineCollisionFree(x_{new}, x_{near}) and Cost(x_{new}) + Distance(x_{new}, x_{near}) $<$ Cost(x_{near}) **then**
        Set parent of x_{near} to x_{new}
      end if
    end for
    if Distance(x_{new}, g) $<$ threshold **then**
      $g.parent \leftarrow x_{new}$, add g to T
      $\pi \leftarrow$ ReconstructPath(g)
      if Cost(π) $<$ BestCost: update best path π_{best}
    end if
  end if
end for
return π_{best} if found, else $\emptyset$

H. Maps Used

Shown below are the maps used for evaluating the planners. Six primary maps were used for the main evaluation, along with this, more maps were procedurally generated with random seeds and used for evaluation as well.

The code for all of the map generation, including the set maps and the procedurally generated maps can be found in the author's GitHub Repository. [11].

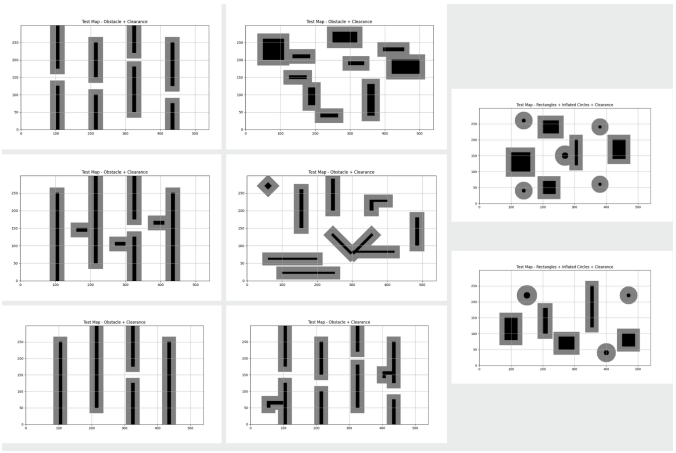


Fig. 2. Maps Used

V. Results

A. Comparative Analysis

- *Success rate.*
  All grid–based and roadmap planners (`astar`, `astarweighted`, `prm_dijkstra`, and `prm_weightedAstar`) solved every map, whereas `rrt` failed on roughly 12% of the cases; `rrt_astar` recovered full success but at additional cost. This indicates that deterministic search and pre–computed roadmaps are consistently reliable in the planar (x, y, θ) workspace.
- *Average runtime.*
  `prm_weightedAstar` was the fastest (mean $\approx$ 6 s), followed by `astarweighted` and `prm_dijkstra`. Both `rrt` and `rrt_astar` were the slowest (mean 33–39 s) because each iteration performs a linear nearest–node scan and continuous collision checks before the tree incidentally connects to the goal.
- *Path quality.*
  Relative-length bars stay near 1.0 for every grid/PRM planner, confirming they routinely find the shortest available corridors. In contrast, `rrt`–family paths are 1.5–2.0$\times$ longer on most maps, and their average jerkiness is *four to five times* higher, reflecting zig-zag exploration before convergence.
- *Interpretation.*
  Sampling–based planners generally excel when the configuration space is high-dimensional or kinodynamic, but in this low-dimensional, holonomic setting a Euclidean-heuristic `A*` or a sparse roadmap gives near–optimal solutions with far fewer state expansions. The tree–based methods pay a steep price for random sampling, nearest–neighbour queries, and edge checking without the benefit of additional degrees of freedom.
- *Practical implication.*
  For a differential–drive robot navigating 2-D maps, heuristic grid search or a lightweight PRM augmented with a weighted `A*` query yields the best trade-off be-

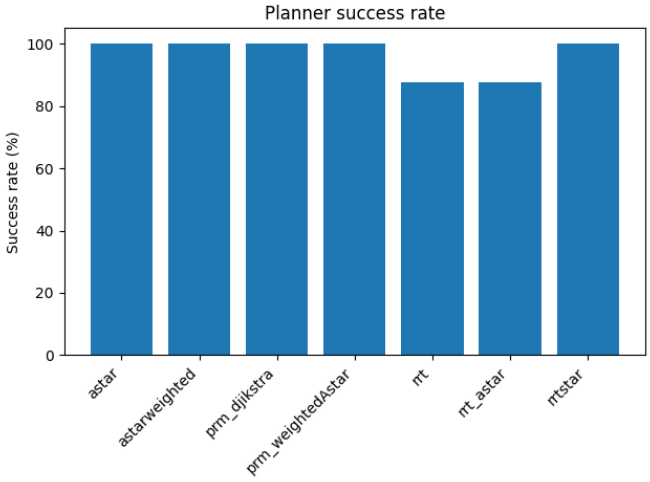


Fig. 3. Success Rate

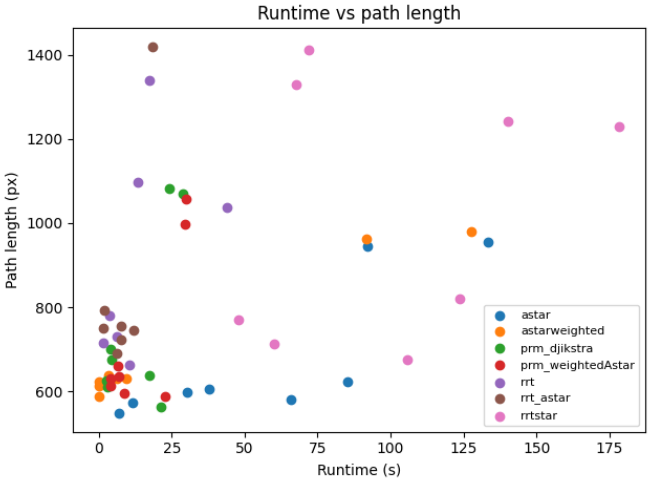


Fig. 4. Runtime vs Path Length

tween speed, smoothness, and reliability. Sampling planners become advantageous only when the task introduces higher-DOF kinematics or complex dynamic constraints.

VI. Discussion

1) **Summary of Key Findings**
Deterministic planners—A*, Weighted A*, PRM + Dijkstra, and PRM + Weighted A*—achieved a **100% success rate** across all tested maps (Fig. **??**). In contrast, sampling-based methods (RRT and RRT + A*) showed occasional failures. Most notably, RRT* failed to find a path on `mapenv2` during batch evaluation, though isolated tests were successful. This underscores the reliability of graph-based planners in structured environments.
2) **Performance Trade-offs**
Fig. **??** shows that PRM + Weighted A* had the lowest average runtime, followed closely by PRM + Dijkstra.

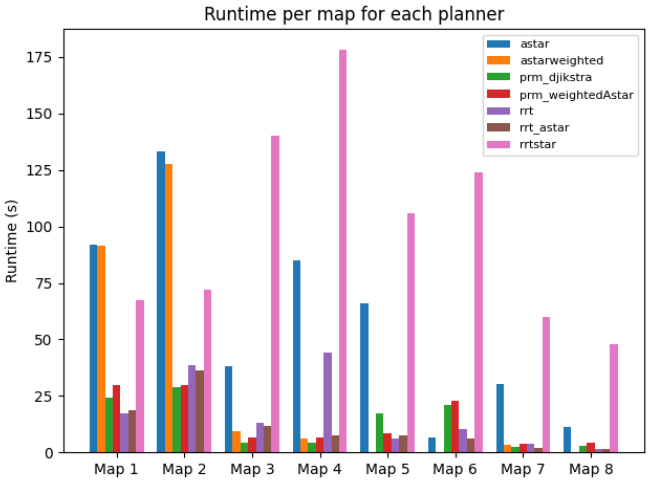


Fig. 5. Runtime Per Map

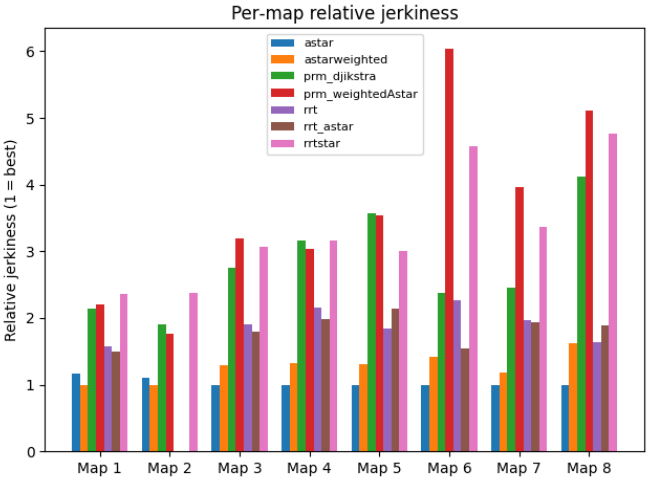


Fig. 6. Relative Jerkiness

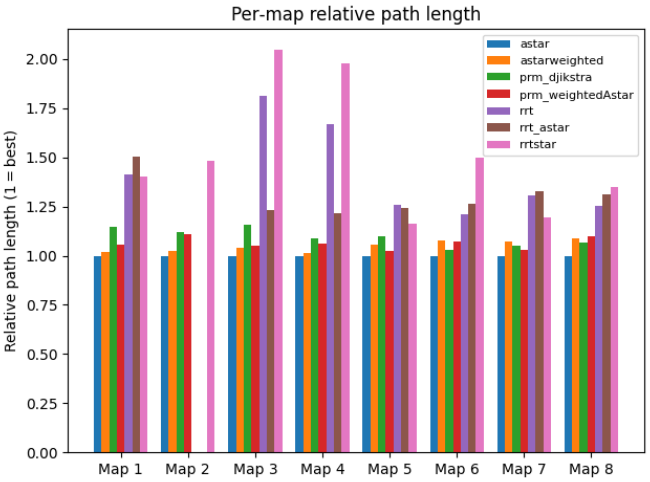


Fig. 7. Path Length

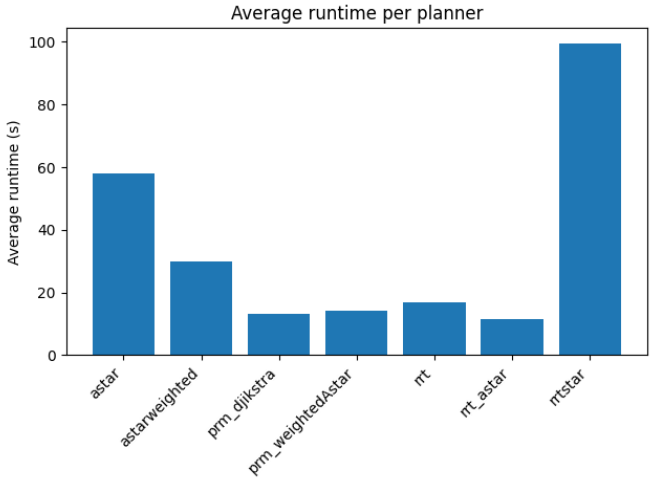


Fig. 8. Average Runtimes

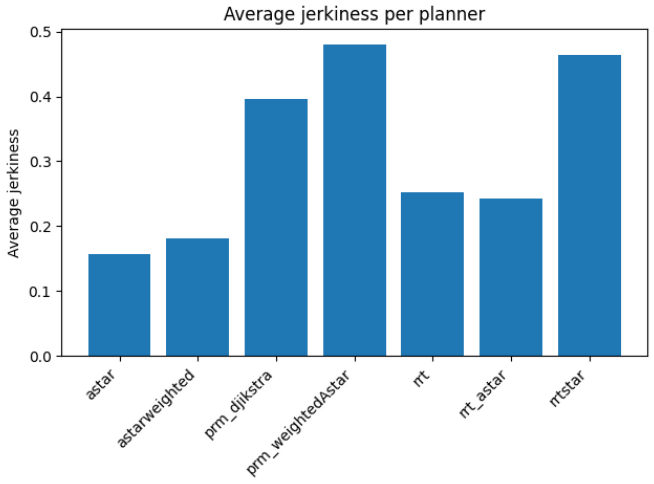
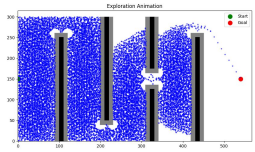


Fig. 9. Jerkiness

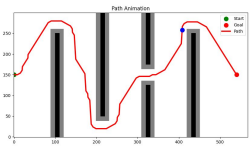
RRT* had the **highest runtime**, a consequence of its tree rewiring overhead. RRT and RRT + A* also exhibited high runtimes due to sampling and collision checking. A* had consistently high planning times, likely due to exhaustive node expansions in grid space.

3) **Algorithm Strengths and Weaknesses**
As evident in Fig. **??**, A* and Weighted A* produced the **smoothest trajectories** (lowest jerkiness values). PRM variants performed moderately well, while RRT and especially RRT* produced more jagged paths. Fig. **??** further shows that sampling-based planners tend to yield **longer paths** compared to graph-based methods.
4) **Impact of Non-Holonomic Constraints**
All planners were adapted for differential drive motion via RPM-based simulation, which added computational cost. The impact was most significant in RRT and PRM planners, where edge feasibility checks required motion simulation. Still, the inclusion of these con-

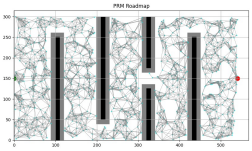
a



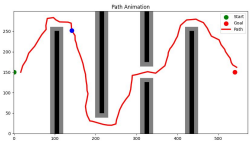
(a) A* Exploration



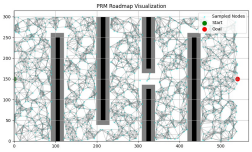
(b) A* Path



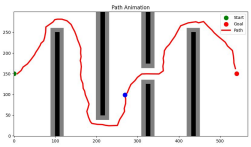
(c) PRM + Djikstra Exploration



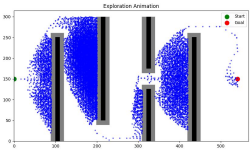
(d) PRM + Djikstra Path



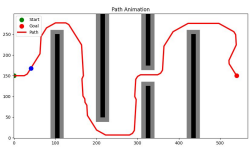
(e) PRM + Weighted A* Exploration



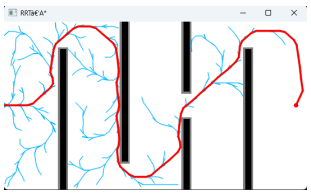
(f) PRM + Weighted A* Path



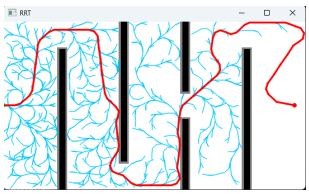
(g) Weighted A* Exploration



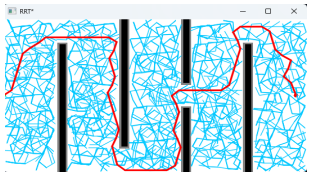
(h) Weighted A* Path



(i) RRT + A* Path



(j) RRT Path



(k) RRT* Path

Fig. 10. Sample Algorithm Node Exploration Visualization and Path Generation

straints ensured the resulting trajectories were **physically realizable** on a differential-drive platform.

5) **Implementation Challenges**
   Simulating differential-drive kinematics, handling edge validation, and integrating KDTree acceleration were non-trivial. Parameter tuning—especially for integration timestep and motion primitives—had a large impact. RRT* in particular showed non-deterministic failures in batch mode due to sensitivity to sampling, underscoring the need for **statistical robustness** in evaluating stochastic planners.
6) **Limitations of the Study**
   The number of test maps was limited to eight, and each planner was run only once per map. This may not fully capture the variability of probabilistic methods. Additionally, tuning was manual and environments were strictly 2D and static. The framework does not yet support dynamic replanning or partially observable maps.
7) **Future Work Suggestions**
   Expanding the evaluation to dynamic environments and including planners like RRT-Connect or Hybrid A* is a clear next step. Multiple runs per planner-map pair would enable meaningful statistical analysis. Automated pipelines for hyperparameter tuning and benchmarking, as well as real-world deployment on differential-drive robots, are promising future directions.

VII. Conclusion

This work presented a comprehensive benchmarking framework for evaluating classical and sampling-based path planning algorithms under **differential-drive constraints**. All planners were adapted for non-holonomic motion using RPM-based simulation, ensuring that all paths were physically executable.

The results demonstrate that **deterministic graph-based planners** (A*, PRM variants) consistently outperformed sampling-based methods in success rate, runtime, and trajectory smoothness. Among the sampling methods, RRT + A* showed the best balance between efficiency and path quality, outperforming RRT and RRT* in several metrics.

While RRT* is designed for optimality, it failed to consistently deliver high-quality or feasible paths within practical runtime limits. This was especially evident on maps like `mapenv2`, where its stochastic nature resulted in occasional failures.

Overall, the study provides a robust baseline for selecting planners for differential-drive robots. Deterministic methods are favorable for structured environments, while RRT-based methods may still be valuable in higher-dimensional or dynamically changing domains.

Future work includes expanding this benchmark to 3D or dynamic environments, integrating probabilistic robustness analysis, and validating performance on physical robots. All code, plots, and demo videos are available in the public GitHub repository [11].

Acknowledgment

The authors would like to sincerely thank **Dr. Reza Monfaredi** for his invaluable guidance and support throughout this project.

We are grateful to **Koustubh** for his meticulous evaluation and constructive feedback which significantly improved the quality of this work.

Lastly, we thank our peers for their ongoing support and collaboration throughout this research.

References

[1] A. Kumar, N. Sharma, and S. Verma, "Enhanced a* for non-holonomic mobile robot path planning in dynamic environments," *arXiv preprint arXiv:2503.00764*, 2024. [Online]. Available: https://arxiv.org/abs/2503.00764

[2] B. Nemec, M. Mihelič, and L. Zlajpah, "Hybrid a* path planning for nonholonomic mobile robots with kinematic constraints," *International Journal of Advanced Robotic Systems*, vol. 16, no. 1, pp. 1–11, 2019.

[3] W. Li, S. Li, R. Yang, and Y. Zhang, "Prioritized trajectory optimization for multiple non-holonomic mobile robots via minimum effort control," *arXiv preprint arXiv:2012.08135*, 2020. [Online]. Available: https://arxiv.org/abs/2012.08135

[4] B. B. K. Ayawli *et al.*, "Comparative analysis of popular mobile robot roadmap path-planning methods," *Journal of Intelligent Robotic Systems*, vol. 102, no. 2, pp. 34–49, 2021.

[5] P. E. Hart, N. J. Nilsson, and B. Raphael, "Formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[6] J. J. Kuffner Jr and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 2. IEEE, 2000, pp. 995–1001.

[7] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," in *IEEE International Conference on Robotics and Automation*, vol. 2. IEEE, 1996, pp. 566–572.

[8] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[9] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3. IEEE, 2004, pp. 2149–2154.

[10] N. Laul, "ENPM661_Project3_Phase2 — a* path planning repository," https://github.com/nslaul/ENPM661_Project3_Phase2.git, 2024, accessed: 2025-05-16.

[11] kiki101robo, "Comparative_Analysis_PathPlanning–ENPM661–Final–Project," https://github.com/kiki101robo/Comparative_Analysis_PathPlanning-ENPM661-Final-Project, 2025, gitHub repository, accessed 17May2025.