Grid-Based and Sampling-Based Motion Planning under Differential Constraints

Author One[1] **and Author Two**[2]

[1]*Ahnaf Ojayer - 2121949642*
[2]*Maisha Subha - 2111925042*
[†]*These authors contributed equally to this work*

Abstract

Path planning is a classic problem for autonomous robots. To ensure safe and efficient point-to-point navigation, an appropriate algorithm should be chosen keeping the robot's dimensions and its classification in mind. Autonomous robots use path-planning algorithms to safely navigate a dynamic, dense, and unknown environment. A few metrics for path planning algorithms to be taken into account are safety, efficiency, lowest-cost path generation, and obstacle avoidance. Before path planning can take place we need a map representation which can be discretized or open configuration space. Discretized configuration space provides node/connectivity information from one point to another. While in open/free configuration space it is up to the algorithm to create a list of nodes and then find a feasible path. Both types of maps are populated by obstacle positions using perception obstacle detection techniques to represent current obstacles from the perspective of the robot. For open configuration spaces, sampling based planning algorithms are used. This paper aims to explore Grid based algorithms and Sampling-based path-planning algorithms such as Probabilistic RoadMap (PRM) and discusses how optimization is achieved and is beneficial.

Keywords: *Unicycle Model, A* Algorithm, Probabilistic Roadmap (PRM), Kinodynamics, Path Planning*

1. Introduction

The application areas of autonomous robots have seen exponential growing in the last decade. Today they are used in distribution centres, security, healthcare, hospitality, grocery stores, delivery, self-driving cars, etc. Each market and industry is heading toward automation and eventually will start using autonomous robots.

Robot motion is a complex problem to solve given the robot's dynamics and constantly changing environment. Navigation requires modelling of the environment and localization to understand the robot's current position within the environment. First, it detects obstacles and then creates an obstacle-free path to the goal providing control inputs on how to reach there. Path planning plays a key role to find feasible trajectories to reach the goal.

This paper explores a Grid Based algorithm(A*) and a Sampling-based path-planning algorithm(PRM) that evolved in the last few decades. The relevance of comparing the Probabilistic Roadmap Method (PRM) and the A* a pathfinding algorithm in the context of kinodynamic motion planning for a unicycle robot arises from their differing approaches to handling constraints and pathfinding. PRM, widely used in high-dimensional configuration spaces, excels in exploring complex environments by constructing a graph of feasible paths, though it often requires additional steps to ensure dynamic feasibility. Conversely, A* is a renowned graph search algorithm that guarantees the shortest path in a grid-based model, by using an appropriate heuristic.

For a unicycle robot, which has non-holonomic constraints, understanding the performance and applicability of PRM and A* can yield valuable insights. This comparison can illuminate the strengths and limitations of each algorithm regarding computational efficiency, path quality, and feasibility within the context of kinodynamic motion planning.

2. Background

Figure 1 shows the Flow chart of Robot's operation and data flow into Path-planners.

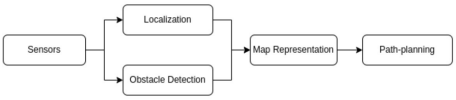


Figure 1. Figure-1: Flow chart of Robot's operation and data flow into Path-planners

2.1. Sensors

Humans navigate in the world by sensing the environment which makes them aware of their own location and also moving parts of the external environment. For a robot to perform any task it also needs sensory data to perceive the environment it is operating in.

The next question comes to what type of sensors are available and what are their use cases. These sensors can further be subcategorized into the 2D Camera, 3D Camera, Topographic Lidar, and Bathymetric Lidar. The most common sensors used are - Camera, Lidar, Radar, IMU, etc.

In autonomous driving multi-camera setups combined with Lidars are mostly used. Overall sensor setup gathers raw information about all the objects around it and then it passes it to the localization and obstacle detection (perception) unit.

2.2. Localization

The localization sub-system tells the robot its current location with respect to the global frame. For e.g. how Google Maps can give information about our current location on the map.

Why is it important? Before planning the goal, the robot needs to get the right start location which comes from localization. The reliability of the localization unit is the most important factor - how confident is it in this system that the robot is at X position right now?

Localization is done by using sensor input, previously logged locations, and fixed maps. These fixed maps are generally generated by mapping static components of the environment for e.g. in a city mapping building, traffic signals, etc.

2.3. Obstacle Detection

All the sensors provide raw data which doesn't make any sense to human eyes but it contains precise locations of an object with respect to the robot. This sensor data is processed by the Obstacle Detection or Perception unit to map these objects on the map. Some of these sensors when used standalone are not sufficient to make a confident decision where an object is seen.

A multi-sensor setup provides various types of raw data to come to a more confident decision. Recently in the autonomous vehicle industry, the focus has been on detecting 3D obstacles to provide a better map representation of objects around a card. These are the various state-of-art techniques to detect 3D obstacles like vision-radar-based fusion, surround view vision-based detection, and transformer-based sensor fusion etc.

2.4. Map Representation

Both localization and obstacle detection is required before finalising the map representation to be sent to the path planners. Map representation sub-system consists of these two components:

- **Static Map** is created by mapping static components of the environment. These static components do not move with time for e.g. a building on the side of a road.
- **Dynamic obstacle** detection updates are done when the perception unit provides information about an obstacle with respect to the current robot's position provided by the localization unit.

Map/configuration space consists of all the positions/configurations that the robot can reach. These maps can be represented in the form of costmaps or configuration space. Every point in the map has a cost for e.g. area within an obstacle is assigned a lethal cost so that planners do not plan through it.

2.5. Path planning

Path planning comes into the category of non-deterministic polynomial-time (NP) hard problems to find the path from the start to the goal location. As we put robots in a dynamic environment complexities increase, and the complexity of the algorithm also increases with an increase in the degrees of freedom of the robot. Navigation through a complex and dynamic environment poses challenges to generating a path that is safe and efficient. Path-planner takes costmap, start, and goal location as input to produce a path. Apart from cost areas, these maps can be discretized with added node connectivity. Grid-based planning like A*, Dijkstra, etc use discretized maps and generate an optimal low-cost path to the goal. Limitations of these types of algorithms arise in bigger maps where discretization is not possible and then sampling-based algorithms are the better choice. There are two criteria on which planning algorithms are rated:

- **Feasible:** A plan is guaranteed to be generated to the goal if it is possible to reach that point, efficiency is not a concern here.
- **Optimal:** Optimising the performance of the planner in addition to finding the path to the goal.

In simpler terms, a path planner generates a plan which is a sequence of actions taken to reach the goal state. Robots today operate in dynamically changing environments so these planners also need to accommodate those functions of the state. Another way to categorise the planner is - Global vs Local planner. Global planners are used to generating a path from start to goal working with global costmaps while local planners work with smaller local costmaps and are used in trajectory generation and following.

3. Kino Dynamics Motion Planning

Kino-dynamic motion planning is a method used in robotics and autonomous systems to plan paths for a robot that take into account both the kinematics (geometric constraints related to the robot's configuration) and dynamics (forces and torques affecting the robot's motion). This approach ensures that the generated paths are not only geometrically feasible but also dynamically achievable given the robot's physical capabilities and constraints.

Key Concepts of Kino-dynamic Motion Planning:

- **Kinematics:** Refers to the motion of the robot without considering the forces that cause the motion. This includes the robot's position, orientation, and possible configurations. Kinematic constraints involve aspects like joint limits and collision avoidance.
- **Dynamics:** Involves the forces and torques that influence the robot's movement. Dynamic constraints take into account the robot's mass, inertia, and the physical capabilities of actuators. This ensures the planned motions are feasible given the robot's ability to generate the required forces.
- **State Space:** Kino-dynamic planning operates in an augmented state space that includes both the robot's configuration (position, orientation) and its velocities. This means the planner must consider both where the robot is and how fast it is moving.
- **Trajectory Planning:** Unlike purely kinematic path planning, kino-dynamic planning produces trajectories that specify both the path and the timing. The trajectory includes positions, velocities, and possibly accelerations over time.
- **Constraints:** The planner must satisfy a set of constraints: 1. Kinematic constraints: Ensuring the robot doesn't violate joint limits or collide with obstacles. 2. Dynamic constraints: Ensuring the planned motion can be achieved given the robot's dynamic capabilities (e.g., acceleration limits, force limits).

In summary, kino-dynamic motion planning is essential for developing realistic and feasible motion plans for robots that must operate under both geometric and dynamic constraints, ensuring safe and effective operation in complex environments.

4. Unicycle Robot Mode

The unicycle robot model is a simplified representation of a robot that has two degrees of freedom: linear velocity and angular velocity. It captures the basic motion characteristics of many wheeled robots, such as differential-drive robots.

Equations of Motion:

The state of the unicycle robot can be described by its position (x, y) in the plane and its orientation θ (the angle between the robot's heading and a reference direction, usually the x-axis). The state of the unicycle robot can be described by its linear velocity (v) and rotational velocity ω The equations of motion for the unicycle model are:

Dynamics and constraints:
State variables:
x , y : position coordinates
θ : orientation of the robot (heading angle)
Control variables:
v : linear velocity
ω : angular velocity

Kinematic Model:

A differential drive robot's kinematic model is defined as :

$$\dot{x} = v \cos\theta$$
$$\dot{y} = v \sin\theta$$
$$\dot{\theta} = \omega$$

Dynamic Constraints:
Velocity Constraint : $0 \leq v \leq v_{max}$
Angular velocity Constraint : $0 \leq \omega \leq \omega_{max}$

4.1. Applications

1. Path Planning: The unicycle model is often used in path planning algorithms to simulate and control the motion of wheeled robots. 2. Control Systems: Designing controllers that can drive the robot to follow a desired trajectory. 3. Simulation: Simulating the behaviour of robots in environments where precise manoeuvrability and realistic motion dynamics are necessary.

In summary, the unicycle robot model provides a simple yet effective way to describe and control the motion of wheeled robots, making it a fundamental concept in robotics and autonomous systems.

5. Path planners

5.1. Grid-Based Path Planning

Grid-based path planning is a technique used in robotics, AI, and computer graphics to find a path from a start point to a goal point on a grid. The grid represents the environment where the environment is divided into a uniform grid of cells and each cell can be either traversable (free space) or non-traversable (obstacle). A cell's neighbours are typically the cells adjacent to it, either in four directions (up, down, left, right) for 4-connected grids or in eight directions (including diagonals) for 8-connected grids.Several algorithms can be used for grid-based path planning, with A* being one of the most common.

In summary, grid-based path planning provides a practical and effective approach for navigating through environments by discretizing the space into manageable units and using algorithms like A* to find optimal paths.

5.1.1. A Algorithm*

The A* algorithm is a popular and widely used pathfinding and graph traversal algorithm, often used in fields such as robotics, video games, and AI for finding the shortest path between two nodes. It combines the features of Dijkstra's algorithm and the Greedy Best-First-Search, using a heuristic to guide its search. It is a powerful algorithm that effectively finds the shortest path by balancing exploration of the search space with heuristic guidance. Its ability to guarantee the shortest path while being computationally efficient (with a good heuristic) makes it a popular choice for various pathfinding and traversal problems.

5.2. Pseudocode of A* Algorithm

```
A* Algorithm:
Input:
vs start point : xs,ys,s
vg end point : xg,yg,g
grid : 10x10 dimension
Output :
path


possible_controls = {set of control vectors the
    to manipulate the robot}
open_heap  [(0,start)]    #open_heap is a min
    heap
closed_list = []
came_from = {start:None}
g_score = {start:0}
f_score = {start:heuristic(start,goal)}
while open_heap is not empty do
  (current_cost,current) = heapop(open_heap)
  if current = goal then
        path = [ ]
    while current not NULL do
    path.append(current)
    current  came_from[current]
    end while
    return path[ : :-1]
    end if
    closed_set.add(current)
    for each control in possible_controls do
    next_state = kinematic_contraint(current,
    control)
         if not is_collision_free(next_state,
    grid) then
          Continue
            end if
     tentative_g_score = g_score[current]+
    heuristic(current , next_state)
        if tentative_g_score < g_score[
    next_state] then
       came_from[next_state] = current
       g_score[next_state] = tentative_g_score
       f_score[next_state]  tentative_g_score
        end if
    end for
end while
return false
```

Code 1. Pseudocode of A* Algorithm.

5.3. Sampling Based Path Planning

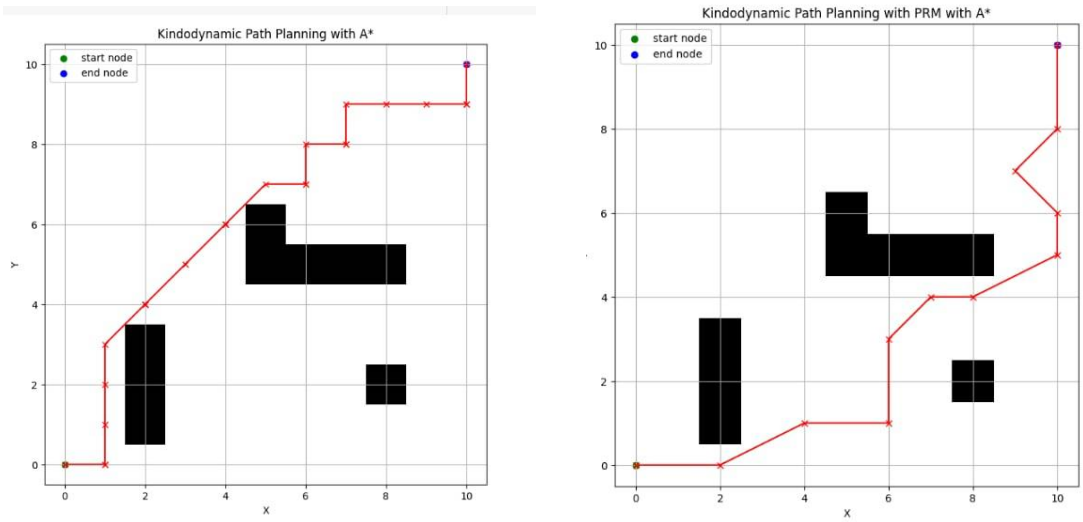
Sampling-based path planner randomly connects points in the state space and constructs a graph to create obstacle free paths. These algorithms don't require exploring the full configuration space so they are faster and more efficient. The number of iterations to generate the graph connectivity can be set by the user which will dictate the optimality of the path that it finds. These types of algorithms present a significant issue while traversing tight spaces as it is difficult to find the connectivity through narrow spaces via random sampling. Following are the different types of Samplingbased Path Planners. Probabilistic Roadmap (PRM), and Rapidly-exploring Random Trees (RRT) are the two most discussed algorithms in Sampling-based path planners. Differentiation in these comes from the way they connect points to create the graph.

5.3.1. Probabilistic Roadmap(PRM) Algorithm

PRM (Probabilistic Roadmap) is one of the initial sampling based path planners. PRM is a graph containing nodes and edges in a map consisting of obstacles and obstacle-free areas. First, it generates randomly sampled nodes in the configuration space and then connects the current node to its neighbouring nodes if the edge is in an obstacle-free area. PRM also takes the radius as an input to determine which random neighbours are calculated and going to be connected. To generate random sampling nodes this paper describes a variety of methods. After generating random neighbouring nodes within a fixed radius, PRM divides them into smaller connected graphs/clusters. These clusters are circular with a radius as described by the user. Every node can belong to multiple connected clusters. Neighbouring nodes are sorted by a metric e.g. increasing distance from the current node. Each neighbour node is checked if it belongs to the same connected cluster as the current node is in, if not then it is added to the same cluster. Parsed radius will determine the performance of a PRM generated graph; the bigger the radius, the more neighbours will be generated and determined for their cluster association. We can also parse a parameter total number of nodes to be generated by the PRM algorithm the more this number is, the increased time PRM Algorithm takes to create the graph. But if this number is too low, it can generate a fragmented graph. The limitations of PRM come in place in the obstacle-dense regions and present the issue of fractured graphs. Looking for the shortest path is also challenging for the resulting sparse graph.

5.4. Pseudocode of PRM Algorithm

```
PRM Algorithm:
Input:
start:Start node
End: End node
Grid: 10x10 dimension
```



(a) Kinodynamic Pathplanning with A*.

(b) Kinodynamic Pathplanning with PRM.

Figure 2. Kinodynamic Pathplanning with A* and PRM

```
Num_samples: Number of samples to generate
Neighbours:number of nearest neighbours to
    connect in the graph
Output:
graph: dictionary representing the roadmap graph
samples = [start,end]
while |samples| < num_samples+2 do
sample = (random sample from grid)
  if is_collision_free(sample,grid) then
    samples.append(sample)
  end if
  graph = {sample,[]}
  for each sample in samples do
    distances=[]
    for each other in samples do
      if other != sample then
distances.append(distance(other,sample),other)
        end if
    end for
    distances.sort()
    for each (distance,neighbour) in distances[:
    neighbours] do
      if is_collision_free(neighbour,grid) and
    is_collision_free(sample,grid) then
        graph[sample].append(neighbour)
        graph[neighbour].append(sample)
      end if
    end for
  end for
End while
```

Code 2. Pseudocode of PRM Algorithm.

6. Methodology

6.1. Problem Definition

Comparing PRM and the A* pathfinding algorithm for kinodynamic motion planning in a differential drive robot, A representative problem is defined to simulate a real-world environment. This problem will help in evaluating the algorithms' performance under various conditions, including static obstacles, varying environment complexities, and different initial and goal states.

6.2. Problem scenario

Navigating a Warehouse Environment:

A differential drive robot is tasked with navigating through a warehouse to deliver packages. The warehouse is populated with shelves (static obstacles). The robot must move from a designated starting point to a delivery point, adhering to kinodynamic constraints.

6.3. Implementation with A* Algorithm

To solve the navigation problem of a differential drive robot in a warehouse using the A* algorithm while adhering to kino dynamic constraints, we will follow these steps:

Define the Problem and Assumptions

1. State Representation: (x,y,θ) where x and y are the coordinates, and θ is the orientation.

2. Action Space: The robot can move forward with a fixed velocity v and can change its orientation with a fixed angular velocity ω.

3. Warehouse Environment: Represented by a grid with known obstacles (shelves).

4. Kino Dynamic Constraints: The robot's motion is governed by:

$$\dot{x} = v\cos\theta$$
$$\dot{y} = v\sin\theta$$
$$\dot{\theta} = \omega$$

6.3.1. Motion Model and Discretization

Discretize the state space and the action space for simplicity:

Linear Velocity v: Assume v=1 unit/s

Angular Velocity ω: Assume $\omega=\pm\pi/4$

Time Step Δt: Assume Δt=1s

6.3.2. Heuristic Function

We will use the Euclidean distance as the heuristic:

$$h(x,y) = \sqrt{(y_{goal}-y)^2 + (x_{goal}-x)^2} \quad (1)$$

6.3.3. Cost Function

The cost function g(x,y,θ) is the actual distance travelled, given by the sum of the linear distances moved at each step.

6.3.4. Implementing the A Algorithm*

We need to mathematically describe the process of finding the shortest path from the start to the goal.

Initial and Goal States:

Start State: S=(x_s,y_s,θ_s)=(0,0,0)

Goal State: G=(x_g,y_g,θ_g)=(10,10,0)

Obstacles: Assume we have obstacles at certain grid cells in the warehouse.

Steps to Calculate Path:

1. Initialization:

Open list contains the start node S.

Closed list is empty.

g(S)=0, h(S) = h(0,0) = $\sqrt{(100)^2 + (100)^2} = 102$

Total cost f(S) = g(S)+h(S) = 102

2.Expand Node S:

Generate possible next states using the model:

$$MoveForward : (\bar{x}, \bar{y}, \bar{\theta}) = (x + v\cos\theta, y + v\sin\theta, \theta)$$
$$TurnLeft : (\bar{x}, \bar{y}, \bar{\theta}) = (x, y, \theta + \omega)$$
$$TurnRight : (\bar{x}, \bar{y}, \bar{\theta}) = (x, y, \theta + \omega)$$

3.Calculate Cost: For each new state, calculate g, h, and f:

Move Forward: g((1,0,0))=1, h((1,0))=$\sqrt{(101)^2 + (100)^2} = \sqrt{181}$

Turn Left and Turn Right: Costs remain the same, as no actual movement.

4. Add New States to Open List: States are added to the open list if they are not in the closed list and do not collide with obstacles.

5.Iterate: Continue the process by selecting the node with the lowest f value from the open list, expanding it, and updating the lists. The process continues until the goal state is reached.

6.3.5. Mathematical Calculations

To illustrate a few iterations:

First Iteration:

Expand (0,0,0):

New states:

(1,0,0), g=1, h=$\sqrt{181}$, $f = 1 + \sqrt{181}$

$(0, 0, 4), g = 0, h = \sqrt{200}, f = \sqrt{200}$

$(0, 0, 4), g = 0, h = \sqrt{200}, f = \sqrt{200}$

Second Iteration:

Expand the state with the lowest f:

Suppose (1,0,0) is chosen:

New states: (2,0,0), g=2, h=$\sqrt{160}$, $f = 2 + \sqrt{160}$

$(1, 0, 4), g = 1, h = \sqrt{181}, f = 1 + \sqrt{181}$

$(1, 0, 4), g = 1, h = \sqrt{181}, f = 1 + \sqrt{181}$

Continue iterating, updating the open list and closed list, until the goal state (10,10,0) is reached.

The A* algorithm, combined with a proper heuristic and consideration of the kino dynamic constraints, will eventually find from the start to the goal, avoiding obstacles and respecting the motion capabilities of the differential drive robot.

6.4. Implementation with Probabilistic Roadmap (PRM) Algorithm

To solve the problem of navigating a differential drive robot through a warehouse using the Probabilistic Roadmap (PRM) algorithm while adhering to kino dynamic constraints, we will outline the mathematical steps required to generate a feasible path from the start point to the goal point.

Steps to Solve Using PRM Algorithm:

1. Define the State Space and Obstacles:

State space: (x,y,θ), where (x,y) are the coordinates and θ is the orientation.

Obstacles: Assume a set of rectangular shelves represented by their coordinates in the warehouse grid.

2.Generate Random Samples: Generate N random samples (x_i,y_i,θ_i) within the free space(i.e., not inside the obstacles)

Include the start (x_s,y_s,θ_s) and goal(x_g,y_g,θ_g) in the sample set.

3.Connect Samples to Form a Roadmap: For each sample, find the k-nearest neighbours (using Euclidean distance in (x,y).

Attempt to connect each sample to its neighbours using feasible paths that respect kino dynamic constraints.

6.4.1. Initial Setup

Warehouse Dimensions: 20×20 units.

Obstacle: Let's assume rectangular shelves with the following coordinates:

Obstacle 1 : (5,5) to (7,7)

Obstacle 2: (10,10) to (12,12)

Obstacle 3: (15,15) to (17,17)

Start and Goal:

Start: (0,0,0)

Goal: (10,10,0)

6.4.2. Steps for Calculation

Step 1: Generate Random Samples Generate N=100 random samples (x_i,y_i,θ_i) within the warehouse, excluding areas occupied by obstacles.

Step 2: Include Start and Goal Include start (0,0,0) and goal (10,10,0) in the sample set.

Step 3: Connect Samples Calculate Distance:

For each pair of samples (x_i,y_i,θ_i) and (x_j,y_j,θ_j), calculate the Euclidean distance d_{ij} in the (x,y) plane:

dij=$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

Check Feasibility:

Check if a path between (x_i,y_i,θ_i) and (x_j,y_j,θ_j) is feasible:

Direct Path (Straight Line):

Linear distance: $=d_ij$

Time to travel: =/v,assuming constant velocity v=1 unit/s

Check for collision: Ensure the straight path does not intersect any obstacles.

Path with Rotation:

For paths requiring rotation, consider the differential drive constraints:

$\Delta\theta$=θ_j-θ_i

Angular distance: $\Delta\theta$

Time to rotate: $\Delta\theta$=$\delta\theta/\omega$, assuming angular velocity ω=/4 rad/s

Calculate intermediate positions and check for collisions.

Form the Roadmap: Connect each sample to its k=5 nearest neighbours using the feasible paths.

Step 4: Search for a Path on the Roadmap

1.Graph Representation:

Nodes: Samples (x_i,y_i,θ_i)

Edges: Feasible paths between nodes with associated costs (distance).

2.Path Search Using A*:

Use A* algorithm to find the shortest path on the roadmap from the start node to the goal node.

Heuristic: Euclidean distance to the goal.

6.4.3. Mathematical Calculation

1. Sample Points:

Start: (0,0,0)

Goal: (10,10,0)

Random Sample: (3,4,π/6) , (6,8,π/4)

2.Distance Calculation:

Distance between (2,2) and (3,4):

$d_1 2 = \sqrt{(3-0)^2 + (4-0)^2} = 5$

3 .Path Feasibility:

Check if the straight line from (0,0) to (3,4) intersects any obstacles.

If not, connect these samples.

4. A* path search:

Build a graph with nodes and edges.

Apply A* to find the path from (0,0,0) to (10,10,0).

6.4.4. Path Found

(0,0,0) to (3,4,/6)
(3,4,/6) to (6,8,-/4)
(6,8,-/4) to (10,10,0)
The PRM algorithm, combined with A* search on the roadmap, provides a feasible path from the start to the goal in a warehouse environment with static obstacles. The key steps involve generating random samples, connecting them based on kino dynamics and using A* to find the optimal path on the resulting graph.

6.5. Cost Comparison

To compare the costs of using A* and PRM algorithms for navigating a differential drive robot through a warehouse, we'll consider various cost components, including computation time, path length, and memory usage. Let's use the data provided earlier and break down the costs for each algorithm.
Warehouse Problem Setup:
Warehouse Dimensions: 20×20 units
Obstacles: Rectangular shelves at:
Obstacle 1: (5,5) to (7,7)
Obstacle 2: (10,10) to (12,12)
Obstacle 3: (15,15) to (17,17)
Start Point: (0,0,0)
Goal Point: (10,10,0)
Robot Kinematics:
Velocity v= 1 unit/s
Angular velocity $\omega = \pm\pi/4$ rad/s
Grid Resolution for A*: 0.5 units
Cost Breakdown:

6.5.1. A* Algorithm

1. Computational Time Cost:
Nodes Expanded: 1500
Average Time to Expand a Node: 0.00167 seconds
Total Time: 1500×0.00167=2.5 seconds
2. Path Length:
Total Path Length: 25 units
3. Memory Usage:
Total Grid Cells: 40×40=1600
State Space Dimensions: 1600×8 (for 8 orientations)
Memory Usage: High (due to large state space and fine discretization)
3. Calculation:
Nodes Expanded: 1500
Total Nodes in Memory: 1600 (cells) × 8 (orientations) = 12800
Path Cost: Let's assume each unit of path length has a cost of 1, so 25 units path length = 25 cost units.
Total Cost for A*:
Total Cost = Computation Time+Path Cost+Memory
Total Cost = 2.5 seconds+25 units+12800 units of memory

6.5.2. PRM Algorithm

1. Computation Time Cost:
Roadmap Construction:
Sampling Time: 0.5 seconds
Connection Time: 1.0 seconds
2. Path Search:
Nodes in Roadmap: 102 nodes
Nodes Expanded in Search: 50
Average Time to Expand a Node: 0.01 seconds
Total Search Time: 50×0.01=0.5 seconds
3. Path Length:
Total Path Length: 27 units
4. Memory Usage:
Number of Samples: 100
Total Nodes in Roadmap: 102 (including start and goal)
Total Edges: 500 (approximate)
5. Calculation: Nodes Expanded in Search: 50
Total Nodes in Memory: 102 nodes
Path Cost: Let's assume each unit of path length has a cost of 1, so 27 units path length = 27 cost units.
Total Cost for PRM:
Total Cost = Construction Time+Search Time+Path Cost+Memory
Total Cost =1.5 seconds (construction)+0.5 seconds (search)+27 units+102 units of memory

6.6. Experiment Result

Cost Component	A* Algorithm	PRM Algorithm
Computation Time	2.5 seconds	2.0 seconds
Path Length	25 units	27 units
Memory Usage	12800 units	102 units

Table 1. The table contains the output

- PRM is faster in terms of computation time.
- A* provides a slightly shorter path.
- PRM uses significantly less memory.

7. Result Analysis

7.0.1. A* Result Analysis

- A* efficiently and consistently found the most optimal path in every iteration of the simulation. This comes at a high computation cost.
- A* algorithm checks for collision detection in every step this increases computation cost but ensures that path does not intersect with obstacles.
- Even though the environment we tested was small the computation was costly for A* algorithm, for larger or more complex environments the performance will be affected drastically.

7.0.2. PRM Result Analysis

- PRM gave a road that was not as efficient as the path the grid based A* had followed. However it is to be mentioned that had we had only taken a sample of n=100, with more samples it might have given better results, but not necessarily optimal in terms of distance or nodes traversed.
- PRM performs collision check when roadman is made this makes the connections collision free and traversal algorithms used afterwards do not need to check for extra collision detection.
- PRM can handle larger and more complex environments better than A* due to its probabilistic nature. In the given scenario, the PRM algorithm was able to construct a path that avoided obstacles.

A* excels in environments where optimality is paramount, but faces challenges with scalability and dynamic obstacles. Conversely, PRM offers greater flexibility and scalability, making it suitable for complex and dynamic environments, though it may not always guarantee the shortest path.

7.0.3. Practical Implications

A* practical applications:

- Any application that requires precise movement and optimal path such as surgery robots or self-driving cars.
- For real time application in smaller environment where quick pathfinding is necessary, such as indoor mobile robots

PRM practical applications:

- Best suited for environments with complex obstacle configuration or large-scale outdoor areas

- Allows for flexible and adaptable path planning which makes it suitable for autonomous vehicles and UAVs
- The roadmap can be precomputed and reused which makes it suitable for applications where traversal of the same static environment is required.

7.0.4. Limitations

Limitations of A*:

- Performance degrades with the increase in grid size and complexity. Not suitable for very large or highly complex environments without modifications.
- Assumes a static environment, and handling dynamic obstacles requires additional modifications.

Limitations of PRM:

- The effectiveness of PRM depends on the number of samples and their distribution. Insufficient sampling can lead to poor path quality or failure to find a path.
- While scalable, PRM can be computationally expensive, especially during the roadmap construction phase, requiring significant processing time and memory for large numbers of samples.
- PRM does not guarantee the shortest path; it provides a feasible path which may not be optimal in terms of distance or cost.

8. Future Work

Building on the findings of this research,a few avenues for future work are proposed:

- **Real-Time Implementation:** Focus on the real-time implementation of the two algorithms in dynamic environments, such as autonomous driving and robotic navigation in urban settings. This includes optimising the computational costs of the algorithms to meet real-time processing requirements.
- **Extended Kinodynamic Constraints:** Investigate the application of these planning methods to more complex kinodynamic models and constraints, such as those found in aerial and underwater robotics. This would involve extending the current algorithms to accommodate additional degrees of freedom and more intricate dynamic behaviours.
- **Energy and Resource Constraints:** Incorporating considerations for energy consumption, computational resources, and other practical constraints into pathfinding algorithms to make them more applicable to real-world robotic systems.
- **Learning-Based Methods:** Integrating machine learning techniques to improve the heuristic functions in A* or the sampling strategy in PRM, potentially leading to more efficient and intelligent pathfinding algorithms.
- **User-Interactive Pathfinding:** Creating user-interactive systems where human operators can provide input or adjust parameters in real-time to influence the pathfinding process.

9. Conclusion

In this research,the main objective was kinodynamic motion planning for a unicycle model using A* and Probabilistic Roadmap (PRM) methods. Our objective was to evaluate the performance of these algorithms in terms of path optimality and computational efficiency under differential constraints.

Our experimental results demonstrated that the A* algorithm, while providing optimal solutions in simple 2 dimensional environments, it struggles with the increased complexity when handling high-dimensional configuration spaces with more dynamic constraints. However, the PRM method showed consistent results in navigating complex environments with dynamic obstacles due to its probabilistic sampling nature and ability to handle large configuration spaces more efficiently.

Both methods have advantages and disadvantages,A* excels in predetermined environments. PRM provides flexibility and scalability in more dynamic scenarios. This dual approach shows the necessity for hybrid strategies that can leverage the strengths of both deterministic and probabilistic planning algorithms to achieve robust and efficient motion planning in real-world applications.

References

[**ICRA** 024] ICRA. 2024. *E. Schmerling, L. Janson, and M. Pavone, "Optimal Sampling-Based Motion Planning under Differential Constraints: the Driftless Case," pdf, ICRA. Accessed: Jun. 13, 2024. [Online]. Available https://stanford.edu/ pavone/papers/Schmerling.Janson.ea.ICRA15a.pdf*

[**Wikipedia Contributors**019] A* search algorithm," Wikipedia, Mar. 10, 2019. https://en.wikipedia.org/wiki/A* search algorithm

[**Principles of Robot Autonomy**021] *M. Pavone and J. Lorenzetti, Principles of Robot Autonomy. 2021.*

[**Probabilistic roadmap**, **Wikipedia**ar. 29, 2022] *https://en.wikipedia.org/wiki/ProbabilisticRoadmap*

[**Introduction to autonomous mobile robots.** 2011] *R. Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza, Introduction to autonomous mobile robots. Cambridge, Mass.: Mit Press, 2011.*

10. Contact us

Information

ahnaf.ojayer@northsouth.edu
maisha.subha@northsouth.edu