Robotic Systems

7CCEMROB

Motion Planning

Coursework 2

Date: 28/03/2023

Abstract

Motion Planning is a computational problem involving seamless integration of robotic systems and sensing for optimization of movement by the process initiating breaking down of a desired movement task into discrete motions. The task to plan collision-free path from start to goal without encountering numerous obstacles is a fundamental one. This report provides a deep insight for getting an understanding of methodologies revolving around motion planning. These three methods are namely, Randomized Rapidly-Exploring Random Trees (RRT), Probabilistic Roadmap (PRM) and A* for 2D motion planning. The first task involved implementation of a Randomized Rapidly-Exploring Random Trees (RRT) planner to generate a resolution of the path. The second task implements a Probabilistic Roadmap (PRM) planner by modification of the existing RRT code for a generating a roadmap of nodes and edges for employing it to plan a path from the start to the goal using Dijkstra's algorithm. The third task implements A* algorithm, a search based algorithm used for map traversal to find the shortest path with the usage of additional heuristics for a better performance in a 2D environment. The final task provides a critical analysis of the pros and cons for each of these three methodologies concerned with motion planning demonstrations. Case studies for solving different types of motion planning problems are provided. Overall, this report is an insightful presentation to equip readers with profound knowledge and practical applications of RRT, PRM and A* methods.

Table of Contents

1. Introduction…….4
2. Literature Review……....5
3. Methodologies……6
4. Task 1……...7
   4.1 Pseudo Code……7
   4.2 Result……8
5. Task 2…….. 9
   5.1 Pseudo Code……..9
   5.2 Result……...10
6. Task 3……..11
   6.1 Pseudo Code……11
   6.2 Result……...12
7. Task 4……..13
8. Conclusion……..15
9. References……..17
10. Appendix……17

List of Figures

1. The RRT (Randomized Rapidly-Exploring Random Trees) method result…….8
2. The PRM (Probabilistic Roadmap) along with Dijkstra's algorithm result……10
3. The A* method result……12

1. Introduction

Robotics motion planning refers to a series of methods and techniques used to determine the path/sequence that a robot can move mapping the robot from the initial state it is in to the desired goal state, whilst avoiding objects and obstacles in the environment. This could be a robot that is a robot manipulator with an end-effector (such as a robotic arm) or a mobile robot, all of which move through something known as a configuration space, which is a key idea in robotics motion planning. A configuration space (C-space) is the mathematical representation of all possible configurations of a robot, in regards to the orientation and position of that robot, for the purpose of this report a mobile robot is deployed onto a grid-like environment, where each cell would represent a unique combination of x and y positions and the orientations (the configuration space). Furthermore, the configuration space also helps in allowing the robot to avoid obstacles (C-obstacles), marking them and further allowing effective planning for the robots motion. Thus, the mathematical description of the configuration space and C-space is as follows:

$$configuration\colon q \in C \subseteq R^n$$

$$C - space\colon C = C_{free} \cup C_{obs}$$

There are several methods for robotics motion planning, including:

- Sampling-based methods: These methods randomly sample the robot's configuration space to generate a set of feasible paths. Two common sampling-based methods are the Probabilistic Roadmap (PRM) and Rapidly-exploring Random Tree (RRT).
- Optimization-based methods: These methods formulate the motion planning problem as an optimization problem, where the goal is to minimise a cost function that takes into account factors such as path length, smoothness, and obstacle avoidance. Common optimization-based methods include A* search, Dijkstra's algorithm, and gradient-based methods.
- Heuristic-based methods: These methods use heuristics or rules of thumb to guide the robot towards the goal while avoiding obstacles. Examples include potential fields and artificial potential fields.
- Hybrid methods: These methods combine different motion planning techniques to take advantage of their strengths and overcome their weaknesses. An example of a hybrid method is the PRM-RRT algorithm, which combines the PRM and RRT methods.

This report will solely use sampling-based methods alongside path-finding algorithms.

2. Literature Review

Motion Planning forms a critical core in the field of robotic systems and automation. Several methods have been proposed for solving motion planning problems involving Randomized Rapidly-Exploring Random Trees (RRT), Probabilistic Roadmap (PRM) and A* for 2D motion planning. There have been studies being conducted to present a comparison on the performances of each of these methodologies.

Zhang in 2022 [1] focused on the significance of motion planning methods. Converting a subgoal into a sequence of parameters is what motion planning takes into focus so that the software links with the hardware for reaching the subgoal. He also provided emphasis on classical approaches like sampling-based methods and optimisation-based methods. The classical motion planning methods like A* and RRT-Connect have been applied for changing states upto the goal. Major limelight is shown in how a great deal of progress has been made in task and motion planning (TAMP) methods.

S. Lavelle in 1998 [2] discussed several desirable properties and implementation of Randomized Rapidly-Exploring Random Trees (RRT). He also discussed how RRTs have similar properties as that of a probabilistic roadmap designed with few heuristics and arbitrary parameters. The advantage RRTs have over probabilistic roadmap is that RRTs don't have a requirement of connections between pairs of states which make them more efficient. Path planning consists of collision detection as its key bottleneck for which RRT is completely suited.

Matthias Hüppi in 2022 [3] provides an extension on the original Probabilistic Roadmap (PRM) for generating an augmented graph-like structure. Importance is shown in how the development of algorithms is essential for planning safe trajectories while accounting for moving obstacles. Temporal-PRM (T-PRM) for real-time obstacle avoidance in static and dynamic environment, and an improved A* for solving queries in graphs augmented in time dimension have been implemented. Demonstration of an approach is done such that scenarios can be handled wherein obstacles change velocities and direction of travel unexpectedly. Smoother paths are generated with a lower computation time in comparison to the existing algorithms.

Daniel Foead in 2021 [4] focused on the current state and potential future developments of the A* search algorithm for future projects. Better overall performance is ensured by the usage of this algorithm when dealing with large roaadmaps. This makes use of a heuristic function form start to goal for efficient decision making. The stability and quickness of A* algorithm

has room for improvement for successful completion of tricky tasks involving pathfinding problems. Variants like Hierarchical Pathfinding A* (HPA*) have found their base due to A* lacking in solving complex problems with less overhead.

There have been many studies over the years comparing the performance of these methods in different environments. As per the deduced outcomes, these methods have proven to be effective for motion planning in 2D environments. Each method has its advantages and disadvantages with the choice depending on specific application. Researchers are looking to combine such methodologies to formulate hybrid methods for vital improvements in performance and efficiency.

3. Methodologies

Configuration Space: The configuration space in robotic systems is the space of all possible configurations. It is a key concept of motion planning wherein it is defined as a space that captures all the possible positions and orientations that a robot can take while navigating through an environment. Considering a high dimensional space, each dimension is a representation of the robot's degree of freedom in a given environment.

Randomized Rapidly-Exploring Random Trees (RRT): RRT is a sampling based algorithm for generating a tree structure by exploring the search space with the use of random samples. This algorithm can quickly explore large areas of the search space by sampling random configurations. It proves to be an important method in motion planning due to its effectiveness in various types of environments.

Probabilistic Roadmap (PRM): PRM is a motion planning algorithm that builds a graph structure for representation of the connectivity of the search space. This algorithm i used to find a path from the start node to the goal by using a search based method. A search based algorithm such as Dijkstra's or A* is used for performing a graph search.

A* algorithm: A* algorithm is an optimal pathfinding algorithm which uses additional heuristics to find the shortest path in a 2D environment. The heuristic function is used for prioritising the search for the algorithm to explore the most essential nodes first. This algorithm also finds its use cases for various types of graphs, meshes and roadmaps.

4. Task 1 : Implementing RRT

4.1 Pseudo Code

```
Initialize the tree T with a single node at the start configuration q_start
Repeat until a goal configuration q_goal is reached:
    Randomly generate a new configuration q_rand in the configuration space
    Find the nearest configuration q_near in the tree to q_rand
     Compute a new configuration q_new that is a small step in the direction of q_rand from q_near
        If the path from q_near to q_new is collision-free:
            Add q_new as a new node to the tree T with q_near as its parent
            If q_new is close to the goal configuration q_goal:
                If the path from q_new to q_goal is collision-free:
                Add q_goal as a new node to the tree T with q_new as its parent
                Return the path from q_start to q_goal
```

The algorithm starts with the creation of a tree with the start position as the root node. In each iteration, a random point is sampled from the environment, and the nearest node to that point in the tree is identified. A new node is added to the tree by moving from the nearest node towards the random point. The new node is added to the tree if it satisfies two conditions: (1) it is not in collision with any obstacle in the environment, and (2) it is within the boundaries of the play area (if provided). If the end point is reachable from the newly added node, then a path is generated by tracing back from the end node to the start node.

The implementation consists of a class named RRT, which has several member functions. The most important member function is planning, which performs the RRT algorithm and returns a collision-free path if found. Other member functions are *get_random_node*, which returns a random node sampled from the environment, *get_nearest_node_index*, which returns the index of the nearest node in the tree to a given node, and steer, which moves from a given node towards a target node. The *generate_final_course* function generates the final path by tracing back from the end node to the start node.

The implementation also visualises the progress of the algorithm by setting the animation flag to True. The *draw_graph* function is called to draw the tree in each iteration of the algorithm.

The technique described here is a rudimentary version of the RRT algorithm for motion planning. Depending on the specific situation, numerous upgrades and adjustments may be possible to increase performance and/or the quality of the solution, such as biassing the random configuration generation towards the goal or employing a heuristic to steer the tree expansion.

4.2 Result

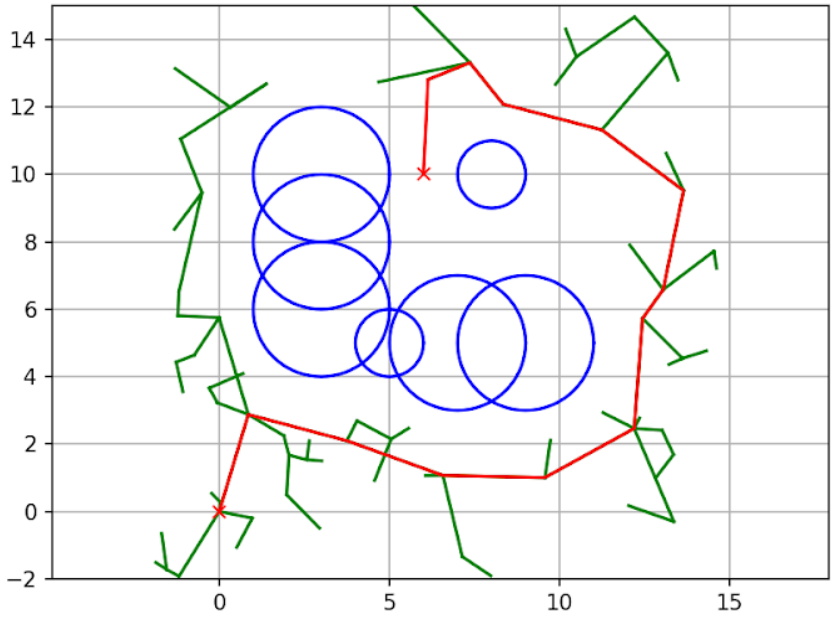

Fig 1. The RRT (Randomized Rapidly-Exploring Random Trees) method result

The blue circles represent the obstacles, Green lines are the RRT, Red line represents the final shortest path.

Run	Time taken (2 d.p.)	Final path length (2 d.p.)
1	32.76	22.34
2	7.05	35.13
3	7.50	22.20
4	21.07	28.57
5	4.61	25.88

Table 1. Path length and time of the RRT algorithm (including animation time)

5. Task 2 : Implementing PRM along with Dijkstra's algorithm

5.1 Pseudo Code

```
(Creating the Probabilistic roadmap)
Define the number of nodes to be generated num_nodes
    Generate a random sample of nodes in the configuration space (Cfree)
        while len(nodes) < num_nodes:
        x = np.random.uniform(x_min, x_max)
        y = np.random.uniform(y_min, y_max)
            If  node is not colliding with obstacles
            then nodes.append([x, y])
Define the number of nearest neighbours to consider
Compute the Euclidean distance between each pair of points
Find the k-nearest neighbours for each point
Initialise an empty list to hold the edges
Add edges between each point and its k-nearest neighbours (To get the final PRM)

(Implementing Dijkstra's to find the smallest distance between start and goal)
Find the closest nodes to the start and end points
Initialize distances to all nodes as infinity except the start node, which is 0
 Create a priority queue with the start node
   While queue is not empty
        get the node with the smallest distance from the queue
        if we have reached the end node,
            return the shortest path
       for each neighbour of the current node, update its distance if necessary
            distance = distances[current] + distance_between(nodes[current], nodes[neighbor])
if we reach this point, there is no path from the start to the end node
```

The code implements a Probabilistic Roadmap (PRM) and Dijkstra's algorithm to find the shortest distance between a start and a goal position for a mobile robot in a 2D environment. The PRM algorithm generates a random sample of nodes in the configuration space (C-free) and connects each node to its k-nearest neighbours to form a roadmap. The Dijkstra's algorithm then finds the shortest path between the start and goal positions based on the roadmap generated by the PRM algorithm. The functions used in the code include,

is_collision(point) which returns True if the given point collides with any of the obstacles in the environment, False otherwise. *euclidean_distance(node1,node2)*, returns the Euclidean distance between two given nodes. *is_edge_collision_free(node1, node2)* returns True if the edge between the two given nodes does not collide with any of the obstacles in the environment, False otherwise.

The function Dijkstra uses Dijkstra's algorithm for finding the shortest path between two points in a graph. It is a well-known graph search algorithm that solves the single-source shortest path problem for a weighted graph with non-negative edge weights, producing a shortest path tree. The main function takes in 'nodes' and 'connections' lists representing the graph, as well as the start and end points. It returns a list of nodes representing the shortest path. The code uses a priority queue to efficiently find the next closest node, and updates the distance to each node as it searches for the shortest path.

The code also visualises the nodes, obstacles, edges and the final shortest path of the PRM using Dijkstra's algorithm using matplotlib.

5.2 Result

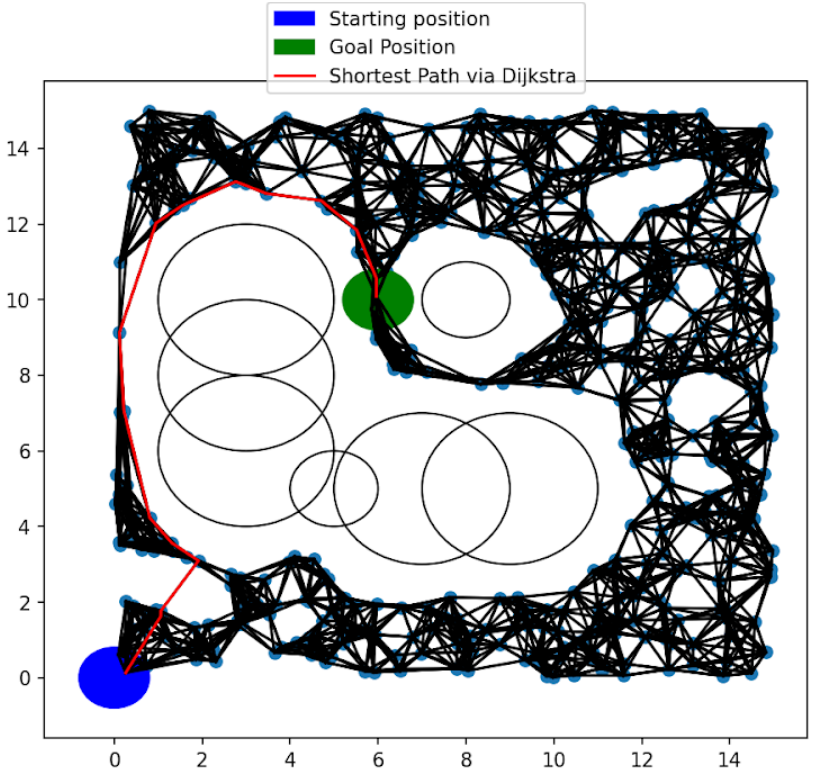

Fig 2. The PRM (Probabilistic Roadmap) along with Dijkstra's algorithm result.
The blue and green circles represent, start and goal points, respectively.
The blue dots are the randomly generated nodes joined by black edges which from the roadmap. The red line represents the final shortest path.

Run	Time taken (sec)	Final path length
1	0.3621	23.51
2	0.1109	20.44
3	0.218	21.10
4	0.1084	19.35
5	0.3211	20.35

Table 2. Path length and time of the PRM algorithm (excluding animation time)

6. Task 3 : Implementing A* algorithm

6.1 Pseudo Code

```
function A_star(start, goal, cost, heuristic):
    open_set = priority queue containing start node with priority 0
    came_from = empty dictionary
    g_score = dictionary with start node as key and value 0
    f_score = dictionary with start node as key and value heuristic(start, goal)

    while open_set is not empty:
        current = node in open_set with lowest f_score
        if current == goal:
            return reconstruct_path(came_from, goal)

        open_set.remove(current)

        for neighbor in get_neighbors(current):
            tentative_g_score = g_score[current] + cost(current, neighbor)
            if neighbor not in g_score or tentative_g_score < g_score[neighbor]:
                came_from[neighbor] = current
                g_score[neighbor] = tentative_g_score
                f_score[neighbor] = tentative_g_score + heuristic(neighbor, goal)
                if neighbor not in open_set:
                    add neighbor to open_set with priority f_score[neighbor]
```

```
    return failure

function reconstruct_path(came_from, current):
    path = [current]
    while current in came_from:
        current = came_from[current]
        path.append(current)
    return reverse(path)
```

In the preceding pseudocode, ‘start’ and ‘goal’ are the start and goal configurations, ‘cost’ is a function that returns the cost of moving from one configuration to another, and ‘heuristic’ is a function that estimates the remaining cost from a given configuration to the goal. The ‘get_neighbors’ method returns a list of neighbouring configurations from a given configuration. The A* algorithm is used to discover the best path from ‘start’ to ‘goal’ by minimising the sum of the path cost and the expected remaining cost from current configuration to target. The algorithm keeps an open node priority queue, where the priority is the node's f-score (sum of g-score and heuristic estimate). The algorithm begins by expanding the nodes with the lowest f-score and then updates the g-score and f-score of its neighbours. ‘The reconstruct_path’ function is used to retrace the final path by traversing the came from dictionary, which holds the parent of each node in the optimal path.

The following paragraph will mention the main functions and link them to the pseudocode provided above. The ‘A_star’ function in the pseudocode corresponds to the ‘planning’ method of the Python code's ‘AStarPlanner’ class. Both functions take the start and goal nodes, as well as a cost function and a heuristic function, as inputs. In a graph or grid map, they employ the A* algorithm to find the shortest path from the start node to the goal node. The ‘get_neighbours’ function in the pseudocode corresponds to the ‘verify_node’ method of the ‘AStarPlanner’ class in Python code. Both functions accept a node as an input and return the nodes that can be reached in a single step from the input node. The ‘reconstruct_path’ function in the pseudocode corresponds to the ‘calc_final_path’ method in the ‘AStarPlanner’ class in the Python code. Both functions take the dictionary of came from nodes and the goal node as parameters. By traversing the came from dictionary from the goal node back to the

start node, they rebuild the path from the start node to the goal node. The ‘calc_xy_index’, ‘calc_grid_index’, ‘calc_grid_position’, and ‘verify_node’ methods of the ‘AStarPlanner’ class correspond to the indexing and conversion processes in the pseudocode. They convert x-y coordinates to grid map indices and assess whether or not a node is safe to travel.

6.2 Result

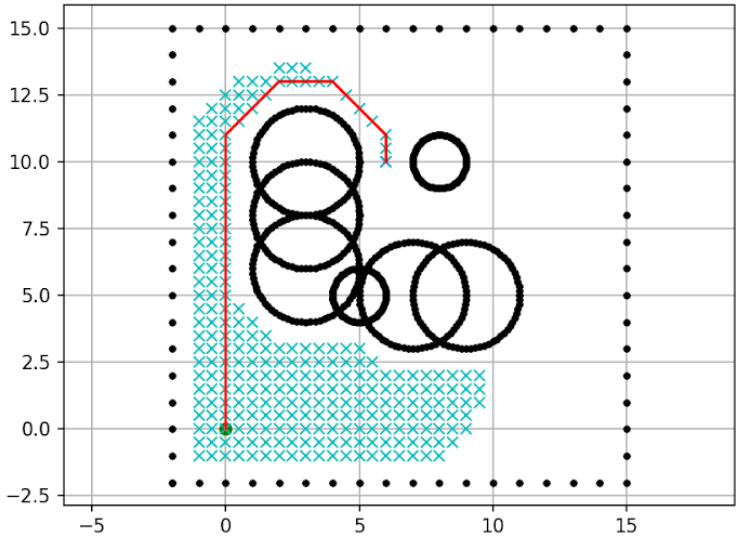

Fig 3. The A* method result

The black circles represent obstacles and the red line represents the final shortest path.

Run	Time taken (2 d.p.)	Final path length (2 d.p.)
1	1.84	19.67
2	1.87	19.67
3	1.83	19.67
4	1.86	19.67
5	1.85	19.67

Table 3. Path length and time of the A* algorithm (including animation time)

7. Task 4 : Comparison of the RRT planner, PRM and A* search methods

The RRT (Randomised Rapidly-Exploring Random Trees) planner, PRM (Probabilistic Roadmap) and A* search methods are amongst the most popular path-planning techniques used in Robotics. Through the tasks of this coursework, each one of these methods were explored and hence, their pros and cons were also highlighted. The critical analysis below compares and contrasts each method by highlighting their pros and cons.

The RRT planner is a randomised algorithm used for motion planning in robotics. It works by randomly sampling the robot's configuration space and building a tree of potential states in that space. The algorithm then extends the tree by connecting new configurations to existing ones until it finds a path from the starting state to the goal state. The algorithm repeats this process until a path is established from the start to the goal.

Pros:

- RRTs can handle complex and high-dimensional spaces. It can explore spaces with obstacles, narrow passages, and complicated geometries.
- The RRT planner can adapt to changing environments, and it does not require a pre-defined map of the environment.
- Given enough time, RRT planner will find a path from the start to the goal with a high probability of success.

Cons:

- It provides sub-optimal solutions. RRT planner can find a feasible solution but not necessarily the shortest one.
- Although various methods can be used to enable fast convergence, depending on the size and complexity of the space, RRT planner can take a long time to converge to a solution.
- The quality of RRT is highly sensitive to the available parameters without which it may lead to a failure in pathfinding.

PRM Planner on the other hand, also being a sampling based method constructs a graph for capturing the connectivity of the robot in its configuration space for finding a collision-free path from the start to the goal. Each node is connected to its nearest node by checking the

existence of a path without obstacles which is how a whole path is laid out for getting the final result.

Pros:

- PRM algorithm finds its suitability in environments wherein a large number of obstacle free configurations are available for a robot.
- This algorithm is also quick in building roadmaps and efficiently search for a collision-free path from start to goal.
- For some instances, it has the capability to provide multiple solutions for different runs which makes it very valuable. .

Cons:

- The efficiency of the overall algorithm is challenging to optimise due to its dependency on the sampling and connectivity algorithms.
- High curvature or presence of narrow passages can hamper the roadmap growth in the case of PRM algorithm.
- The computational cost is directly proportional to an increase in the number of samples, wherein a large number of samples can also improve the quality. This is the reason why large number of samples aren't as feasible.

A* algorithm is a deterministic algorithm for optimal pathfinding from the starting configuration to the goal configuration. It uses additional heuristics for guiding its search for a better performance as every node doesn't need to be visited. Prioritzing the node search at every iteration forms a necessary component for finding an optimal path. The estimated cost of the cheapest solution while performing node traversal is taken into consideration.

Pros:

- The additional heuristics in A* algorithm result in very less iterations which avoids searching at high cost regions.
- A* is applicable to any graph structure with its suitability for low dimensional environments.
- Overestimation is never done in case of the final cost in reaching from the start to the goal.

Cons:

- High dimensional environments consisting of numerous nodes can lead to high computational complexity.
- Any inconsistency in the heuristic function can lead to A* being computationally high in terms of cost.
- A* doesn't entertain changes in between the runtime due to which re-computation is essential every time the environment changes.

Therefore, all these three methodologies namely RRT, PRM and A* have some pros and cons with different approaches to solving a motion planning problem based in different environments. While RRT consists of identification of the nearest node from the random samples until the end point is reachable followed by backtracking of this path; PRM makes use of Dijkstra's algorithm to generate a roadmap from start to goal returning a list of nodes concerning with the shortest path which then visualises the final shortest path using PRM algorithm. On the other hand, A* practices heuristics for minimising the the total cost of the path and the anticipated remaining cost from the start to the start to the goal for finding the best route. Therefore, it can be seen that RRT and PRM are sampling based techniques for motion planning where A* is a search based technique for computation of paths over a discrete graph representation. Fig 1 and Fig 2 are evident of sampling based pathfinding by overcoming obstacles whereas Fig 3 depicts a guided search for obtaining the optimal path as required.

When calculating the time taken to execute the code, the RRT and A* algorithms factor in the time needed to produce animation, whereas the PRM algorithm does not. Prior to analysing the results, it is observed that A* produces the shortest path length among the three tables (1, 2, and 3). Additionally, when comparing the programs that consider animation-based code, A* takes the shortest time. While we cannot directly compare the PRM code with RRT and A* algorithms because it is not written in a similar fashion, we can compare the paths. Overall, the results indicate that A* is the most consistent algorithm in terms of both path length and time taken. Due to its high level of consistency and low variation in results, A* is deemed to provide superior performance for motion planning.

8. Conclusion

The first task of this coursework involved implementing a Randomized Rapidly-Exploring Random Trees (RRT) planner which is one of the sampling based algorithms for motion

planning in robot configurations. Traversal to the random point is done in an environment by identification of the nearest nodes until the path reaches that point. This path is backtracked from start to goal. Valid configurations were obtained if the nearest node generated doesn't result in collision. Hence, this algorithm is used for generating a tree of connected nodes in its configuration space. It was found out that one of the key features of RRTs is that it quickly explores large areas of the search space which makes it a powerful method for motion planning in robotics.

The second task of this coursework involved implementing another sampling based algorithm for motion planning, i.e. , a Probabilistic Roadmap (PRM) planner. This algorithm builds a roadmap for representation of the connectivity of the work space. It randomly samples the search space to introduce only valid configurations for connecting each node. In this methodology, Dijkstra's algorithm has been used to find the shortest path between the start and the goal based on the generated roadmap. Such results also show RRT as an efficient method for handling complex environments.

The third task of this coursework involved implementing A* algorithm for finding an optimal path from the start configuration to the goal configuration. Unlike RRT and PRM, this algorithm uses a heuristic function for estimation of the cost for reaching the goal from each node which avoids visiting every node for a better search. Hence, A* search algorithm exhibited its upper hand in finding an optimal path with flexibility due to the additional heuristics, finding its application in various types of environments.

The fourth task which is the final task is a detailed comparison on the above mentioned algorithms for motion planning. The pros and cons of each methodology were laid out with each algorithm having strengths and weaknesses in specific areas. Thus, the choice of the algorithm is dependent on the type of environment, the type of robot, number of samples and the available computational resources.

Overall, the various methodologies involved in motion planning for robotic systems were explored through this coursework which involved Randomized Rapidly-Exploring Random Trees (RRT), Probabilistic Roadmap (PRM) and A* algorithms. With every algorithm having pros and cons, the decision to choose a method completely lies on the motion planning problem assigned with its features. Motion planning finds its extensive application in fields

like automation, gaming, robotic surgery, architectural design and the study of biological molecules.

9. References

[1] Kai, Zhang & Lucet, Eric & Alexandre dit Sandretto, Julien & Kchir, Selma & Filliat, David. (2022). Task and motion planning methods: applications and limitations. 10.5220/0000163600003271.

[2] LaValle, Steven M.. "Rapidly-exploring random trees : a new tool for path planning." *The annual research report* (1998): n. Pag.

[3] M. Hüppi, L. Bartolomei, R. Mascaro and M. Chli, "T-PRM: Temporal Probabilistic Roadmap for Path Planning in Dynamic Environments," *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Kyoto, Japan, 2022, pp. 10320-10327, doi: 10.1109/IROS47612.2022.9981739.

[4] Foead, Daniel & Ghifari, Alifio & Kusuma, Marchel & Hanafiah, Novita & Gunawan, Eric. (2021). A Systematic Literature Review of A* Pathfinding. Procedia Computer Science. 179. 507-514. 10.1016/j.procs.2021.01.034.

10. Appendix

RRT-Connect is one of the variants of RRT algorithm for motion planning which finds its usage in solving problems consisting of high dimensional environments. This algorithm is an extension of RRTs wherein two trees are implemented which grow towards each other for

pathfinding. A collision-free path is generated once a connection is laid out between these trees. The probability in finding a feasible solution compared to RRTs is more in the case of RRT-Connect as it has better convergence properties. These advantages lead to its widespread use in robotics, computer graphics and computational biology. Some other notable variants of RRTs are Rapidly-exploring random graph (RRG), RRT*, RRT*-Smart, etc.