Torquee: Minimise Torque In Robots

Samreen Ansari, Kincent Lan, Daniel Carlson
Northeastern University Boston

Abstract

This project aims to minimize torque needed by a robot arm (Puma560) to move from start to goal position using various algorithms such as A*, PRM, RRT, RRT* and genetic algorithms. Motion planning is utilized to optimize the system for achieving the desired task while minimizing torque. The Robotics Toolbox is used for finding the path, and PyBullet is used for rendering the simulation. The results show the strengths and weaknesses of each algorithm in terms of torque minimization and motion planning.

GitHub link to the code: Torquee

Contents

1 Introduction

Robotic systems have become ubiquitous in various industries due to their ability to perform complex tasks with high accuracy and speed. However, these systems require significant amounts of energy to operate, resulting in high torque and power consumption. The main objective of this research is to optimize robot motion planning to achieve the desired task while minimizing the torque required.

To achieve this, motion planning techniques are employed, and Puma560 model from the Robotics Toolbox is used for finding the torque and forward kinematics of the robotic arm. PyBullet is used to render the results along with obstacles. By analyzing the simulation results, we can compare the effectiveness of each algorithm in minimizing torque and optimizing motion planning.

The insights from this research can aid in developing more energy-efficient and cost-effective robotic systems that are better suited for specific applications. Overall, this project contributes to ongoing research in the optimization of robotic systems, making them more sustainable and efficient in various industries.

2 Problem Statement

Why are we doing this?

Robots are a costly investment, and their operation requires a significant amount of energy, especially when moving payloads using the robotic arm, as high torque can result in increased energy consumption and reduced system lifespan. Hence, the aim of this project is to develop a motion planning strategy that minimizes energy consumption and torque while enabling the robot to move the payload accurately and efficiently.

One of the key challenges in achieving this goal is to optimize the motion planning of robots. The goal of motion planning is to generate a sequence of robot configurations that move the robot from an initial state to a desired goal state, while avoiding collisions with obstacles in the environment. While motion planning has been extensively studied in the past, minimizing the torque required to achieve a desired task has not been a major focus.

The main objective of this research project is to investigate the effectiveness of various algorithms in minimizing torque while achieving a desired task through motion planning. To achieve this objective, we utilize different algorithms, including A*, PRM, RRT, RRT*, and genetic algorithms, to plan the motion of robots. These algorithms are evaluated based on their ability to minimize torque, as well as their performance in achieving the desired task while avoiding collisions.

The results of this study can provide guidance for developing more energy-efficient and cost-effective robotic systems that are better suited for specific applications.

3 Methodology

How did we do it?

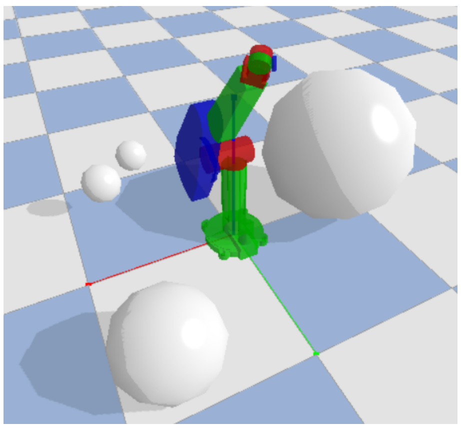


Figure 1: PyBullet sim with Puma560 and obstacles.

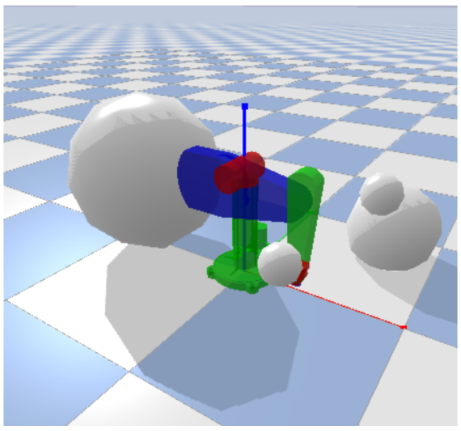


Figure 2: The starting position.

3.1 Simulator

We used the Robotics Toolbox by Peter Corke to train the Puma560 robot to find the path from start joint angle configurations to the goal joint angle configurations while minimizing distance and cost. We used the Swift and PyBullet simulators for displaying the final results of the path planning with torque and cost minimization.

We faced a lot of issues with using the Robotics Toolbox for Python as it still has open issues and is under development, and hence is buggy. For example, it has two kinds of models, the robotics toolbox models, and the DH models, and they both support different operations and are not compatible. For example, the DH model supports functions for finding the torque values, but cannot be rendered in the Swift simulator. There is also a lack of documentation for the updated Swift library, which makes the needed functions like Sphere() and Box() deprecated, but no valid replacements for those.

Another inconsistency we discovered was that the rtb acceps radians for some functions, and degrees for others, instead of using one for all the functions, which required some debugging to find out that this was the problem.

This caused us to use a different simulator for training and finding the paths, and a separate simulator for rendering the motion of the robot. This lead to us requiring more time to develop and run our algorithm, as we had to take into consideration the inconsistencies of the robotics toolbox simulator.

3.2 Algorithms

We explored several algorithms, including A*, greedy, genetic, PRM, and RRT, to minimize torque and displacement for a robotic arm in the context of path planning. Other potential algorithms we planned to explore were neural networks and wavefront planning. However, we excluded neural networks due to a lack of computation power and training data, and we also excluded wavefront planning because it was too computationally expensive. Our goal was to identify the algorithm that can best optimize the path planning process while considering the constraints and limitations of the system. In this paper, we will explore the advantages and disadvantages of each algorithm and evaluate their potential effectiveness in minimizing torque and displacement in robotic arm path planning.

3.2.1 Calculating Torque

In our algorithms, we calculated the torque by using the current position, velocity, and acceleration. We calculated the velocity by finding the delta between current position and the last position over a one-second time interval. The same process was used to calculate acceleration by finding the delta between current velocity and the last velocity over the same time interval. This approach allowed us to accurately compute the necessary torque required to move the robot along its designated path.

3.2.2 Obstacles

We experimented with various positions and sizes for the obstacles while making sure that we do not give the algorithms an impossible problem, but which is still challenging enough to demonstrate the working of the algorithms.

3.2.3 Weighted A* Search

Initially, we implemented the A* search algorithm for optimizing only the torque cost of the robotic arm to the goal[2]. However, we soon discovered that the search space was too large, and having torque as the only cost and heuristic took too long to find a feasible path. To address this issue, we decided to include displacement into the cost and heuristic functions along with torque.

To achieve our objectives of torque and distance optimization, we assigned different weights to the cost of displacement and torque and utilized a weighted A* search algorithm to expand the search space more efficiently. We opted for more weight on displacement, assigning 95% weight to the displacement cost and 5% weight to the torque cost. This was because if we had assigned equal weights, the A* algorithm would have taken a long time to find a feasible path. If given more time, we could have run the simulation for longer with torque as a bigger value for cost, ideally 60-40% weights for the costs. However, the chosen weightings allowed us to find a feasible path in a reasonable amount of time given our project's time constraints and computational resources.

This is similar to what the research paper titled "Multi-objective path planning of an autonomous mobile robot using hybrid PSO-MFB optimization algorithm" did (Ajiel). Specifically, their assumption for multi-objective optimization in section 3.1.2 regarding the relationship between displacement and path-smoothness was similar to ours in regards to the weighted sum of torque and displacement.

Although the weighted sum approach did help speed up the process, it quickly became apparent that the process was still too slow given our time constraints. To expedite the search, we implemented a weighted A* search algorithm, which involved multiplying the heuristic by an epsilon value between $[1, \infty)$. This allowed for A* to quickly converge to the goal location while still optimizing distance and torque.

It's important to note that our approach utilizing weighted A* search algorithm is suboptimal. By assigning different weights to the cost of displacement and torque, we were able to efficiently expand the search space and find a feasible path, but it may not necessarily be the optimal path. This is because the weights were chosen based on our specific needs and constraints, and may not be the true optimal solution for all scenarios. However, we deemed this trade-off acceptable given our project's time constraints and computational resources.

3.2.4 Greedy Search

Initially, we employed a greedy search algorithm in an effort to minimize torque. However, this approach proved to be insufficient, as the robot failed to move along a designated path. This is most likely because the minimum torque from all given neighbors might not necessarily lead to the goal configuration, resulting in the robot arm stagnating.

To address this challenge, we modified our optimization criteria to include both distance and torque, but running it also resulted in high compute times and stagnation. To resolve this issue, we implemented a greedy search by selecting the neighbor with the minimum torque among the 50 closest neighbors to the goal. This allowed us to converge closer to the goal point and minimize torque to an extent. While this approach helped to reduce torque and allowed the robot to move along a designated path, the resulting path was not optimal due to the nature of the greedy algorithm. In particular, the path tended to be jagged as it tried to minimize torque at a specific point rather than the path as a whole.

To improve the quality of the path and achieve a smoother trajectory, we applied a path smoothing algorithm. This approach helped to reduce the jagged effect of the greedy search algorithm and resulted in a more direct path with reduced torque. Overall, the combination of the modified greedy search algorithm and the path smoothing algorithm helped us to achieve a better balance between minimizing torque and optimizing the path for effective robot movement.

3.2.5 Probabilistic Road Map

The probabilistic road map (PRM) algorithm we employed used random sampling to generate a set number of nodes on a directional graph. Next, the k closest neighbors with clear edge paths by distance were connected. Normally this edge would be bidirectional however, the PRM algorithm was modified to create two one directional edges between the two connecting nodes. Each of these edges used the configuration of the node the edge was entering and the distance to calculate the torque cost. Lastly, Dijkstra's algorithm was run on the graph and the shortest path by torque was retrieved.

Due to the nature of this graph having a random number of nodes with edges entering it, creating continuously tracked velocities between a given path from the start configuration to the goal configuration was not possible. Thus the algorithm was blind to the cost of the acceleration of the final path.

3.2.6 Rapidly Exploring Random Tree

The rapidly exploring random tree (RRT) algorithm we employed also used random sampling to generate nodes on a directional graph. However, with the tree structure, each time a node was created it was connected to the closest node by distance only and only in a single direction. This allowed for the velocity of the arm at each position to be tracked and used when determining the torque cost for a given edge.

3.2.7 Rapidly Exploring Random Tree*

The rapidly expanding random tree star (RRT*) algorithm was an algorithm we attempted to implement but ended up being unsuccessful. However, we believe it is still worth highlighting as the idea seems very promising in finding very good minimized torque paths.

The idea of the algorithm was to record the maximum torque along a path in the cost of the graph edges. When a new node is created it would be connected to the neighbor with the smallest cost in a neighborhood. The cost for that move would be found and then the max between the cost of the original node and the new edge cost would be saved within the edge and new node. Then the new node would be checked against all nodes in the neighborhood to see if it could be used to reduce the cost of reaching that node. This would be done by calculating the cost of a move to each of those nodes and then deciding to create a new edge with a given cost by comparing the new node cost, new edge cost, and node that resides within the neighborhood. It would then be necessary to remove the original edge that entered this neighborhood node so that each node only has one input edge. The single input edge allows velocity tracking and makes it so that once the algorithm reaches the goal, there is only a single path that is connected.

Once this algorithm reaches the goal, finding this single path should return a torque optimized path with the final node cost being the maximum torque that is experienced by the arm along the path.

3.2.8 Genetic Algorithms

The genetic algorithm used in this project was designed to find the best path from the starting configuration to the goal configuration for the Puma560 robot, while minimizing the torque. Each individual in the population was represented by a randomly generated path, consisting of a sequence of robot arm configurations, where most of them go from start configuration to goal configuration following a random path. The fitness of each individual was evaluated using a combination of distance to the goal and torque.

To evaluate the fitness, the inverse of the sum of distance to the goal and torque was used. This was done to give higher fitness to individuals that were closer to the goal and required less torque to reach it. Specifically, the fitness function was defined as follows:

fitness = 1 / (alpha * distance + beta * torque)

where alpha and beta are weighting factors that control the relative importance of distance and torque in the fitness function. In this case, alpha was set to 0.9 and beta was set to 0.5.

The genetic algorithm proceeded through a series of evolutionary steps: selection, crossover, and mutation. Selection was performed using the roulette wheel method, in which individuals with higher fitness values had a greater chance of being selected for reproduction. Crossover involved randomly selecting a few parents from the population and generating new offsprings by combining portions of their paths. Mutation involved sometimes changing a configuration randomly, or removing a configuration from the path.

To prevent the paths from becoming too long and unwieldy, we limited their size to a maximum of n configurations. Any paths that exceeded this limit were cut off, as they would have a low fitness value due to the increased torque required.

The genetic algorithm continued for a fixed number of generations, with the fittest individual from each generation being kept as the best path found so far. At the end of the algorithm, the best path was returned as the solution to the problem.

4 Results & Analysis

What did we find out?

4.1 Results

We performed many experiments on the various parameters for each of the algorithms, along with the goal and positions of spheres. The results on the experiments have been listed in 4.1, and the analysis has been mentioned in the next section.

Torque, time, and distance				
Algorithm	Parameter	Torque (N-m)	Time (s)	Distance (rads)
Weighted A* search	Sphere config 1	472	37.35	2.59
	Sphere config 2	486	51.28	2.63
	Sphere config 3	596	60.37	3.29
	Goal 2	499	21.36	2.51
	Goal 3	315	13,32	1.63
	$\epsilon = 3.5$	435	24.07	2.42
Greedy search	Sphere config 1	1241	8.26	11.6
	Sphere config 2	CNS	CNS	CNS
	Sphere config 3	CNS	CNS	CNS
	Goal 2	673	2.35	4.15
	Goal 3	374.10	1.45	2.11
	0.8Dist + 0.2Torq	800	3.79	5.37
	0.2-0.6Dist + 0.8-0.4Torq	1241	9.95	11.67
	0.1Dist + 0.9Torq	1321	9.78	13.16
PRM	Sphere config 1	352	7.79	6.2
	Sphere config 2	471	6.43	7.8
	Sphere config 3	672	6.32	8.9
	Goal 2	381	8.06	5.09
	Goal 3	352	8.12	3.34
RRT	Sphere config 1	2232	0.23	2.86
	Sphere config 2	2155	0.23	2.74
	Sphere config 3	2147	0.31	2.65
	Goal 2	1862	0.24	2.22
	Goal 3	1736	0.31	2.32

*CNS = Could Not Solve

4.2 Analysis

After implementing our algorithms and running them on various problems including collision, we found out the following about each of the algorithms. From figures 9(a), 9(b), and 9(c), we can see that RRT takes the least amount of time and finds the shortest path, but produces high torque; A* produces the least torque, but takes the most time in all of the experiments. Weighted greedy and PRM give average performance on both torque and distance, with PRM being much faster. We explain the analysis of each of the algorithms in detail in the following sub-sections.

4.2.1 Weighted A* Search

As stated before, weighted A*, which extends the A* algorithm, is used because it allows for a faster convergence to a path. The trade-off of using weighted A* versus A* is that it sacrifices optimality for efficiency. This is achieved by assigning a weight the heuristic function of a node.

In the case where the algorithm is optimizing for torque, using only the cost of torque resulted in slow convergence. To address this, a weighted sum of distance and torque was used, in which 95% of the weight was assigned to the displacement cost and 5% weight to the torque cost. Additionally, a weight was re-

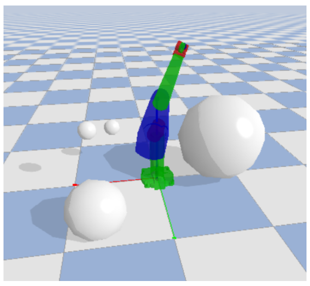


Figure 3: Goal 1
[80, 40, 40, 40, 40, 40]

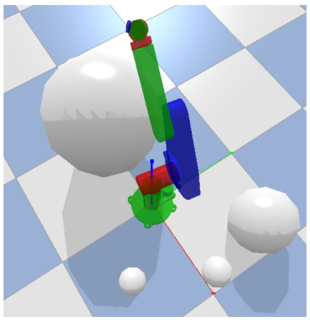


Figure 4: Goal 2
[85, 30, 40, 30, 30, 20]

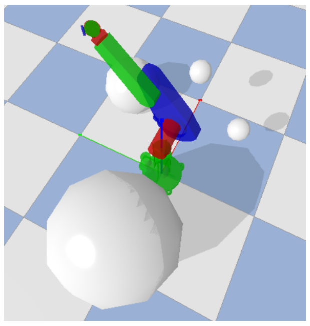


Figure 5: Goal 3
[45, 30, 40, 30, 30, 20]

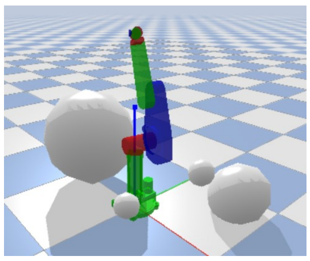


Figure 6: Sphere Config. 1

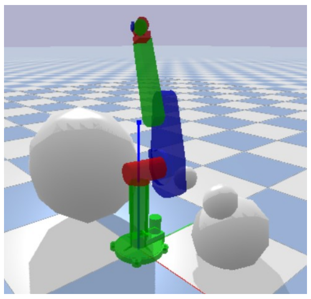


Figure 7: Sphere Config. 2

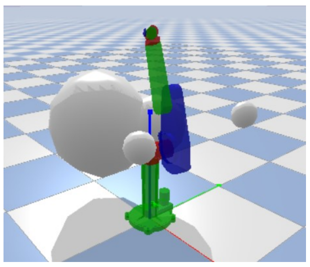


Figure 8: Sphere Config. 3

quired to multiply the heuristic function between the ranges of [1, infinity) to guide the search towards the goal. In this case, the value of 3 has been settled on for this particular problem.

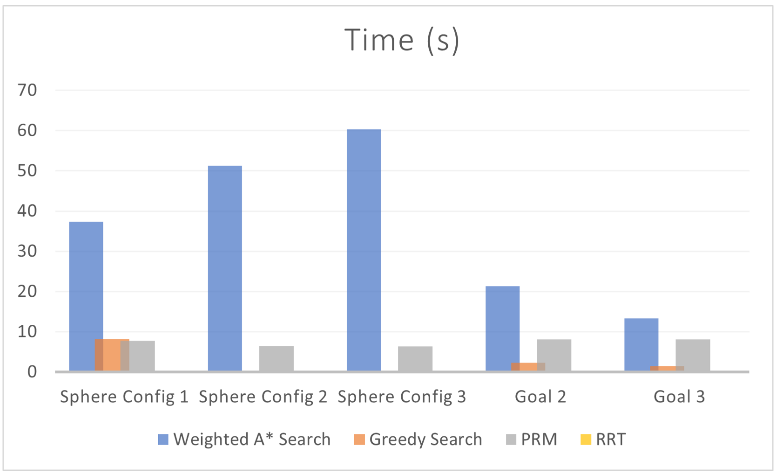
Running weighted A* on this particular problem showed that A* was moderately fast, but not as fast as PRM or RRT. The resulting path generated by Weighted A* is decently smooth and efficient, but may not be the true optimal path. Comparing the total torque of A* to other algorithms, A* had the lowest torque cost. Overall, weighted A* returns a good path that minimizes torque and distance with a relatively moderate compute time.

4.2.2 Greedy Search

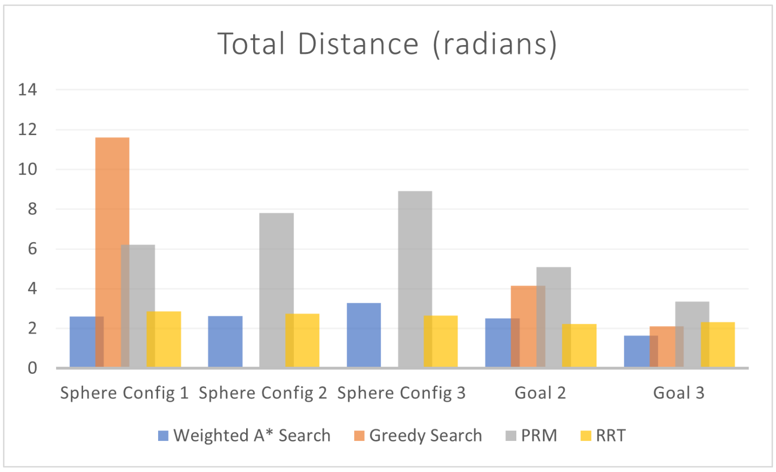
The greedy search algorithm was chosen to minimize torque, which is known for its simplicity and efficiency. However, as we found in our study, a greedy search algorithm can lead to suboptimal results and a jagged path, especially when only optimizing for torque.

To address these limitations, we modified the optimization criteria to include both distance and torque, but running this modified algorithm also resulted in high compute times and stagnation. We then implemented a modified greedy search algorithm by selecting the neighbor with the minimum torque among the 50 closest neighbors to the goal. This allowed us to converge closer to the goal point and minimize torque to some extent. However, the resulting path was still suboptimal due to the nature of the greedy algorithm, which tends to prioritize minimizing torque at specific points rather than optimizing the path as a whole.

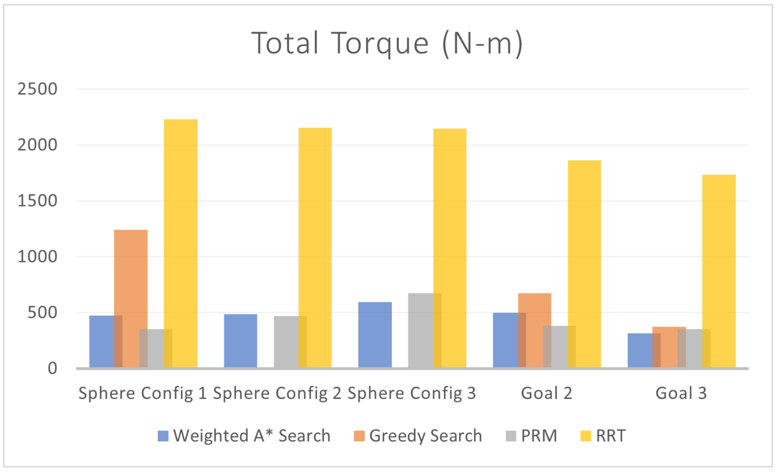
Running greedy on this particular problem showed that greedy performed as fast as PRM or RRT. The resulting path, however, was jagged and had high torque values compared to A*, but almost the same level compared to PRM and RRT. Overall, greedy returns a jagged path that somewhat minimizes torque and distance with a relatively low compute time.



(a) Time Chart



(b) Distance Chart



(c) Torque Chart

Figure 9: Charts

4.2.3 Probabilistic Road Map

The PRM algorithm was average in its run speed and had good success and finding successful paths through obstacles. However, it did create very long and random moves. This meant that the final path was very long but contained far less configuration than other solutions. Due to the usage of a fixed time of one second to move from configuration to configuration, the overall torque was quite low. The PRM output could be significantly improved by path smoothing but that was not included in this implementation.

4.2.4 Rapidly Exploring Random Tree

The RRT algorithm was extremely fast at finding a solution and those solutions were very short, equaling the output of the A* algorithm. However, these paths contained a lot of configurations to move between due to the small step size and lack of path optimization. Due to the fixed time between configurations these high configuration count paths used a lot more power. Had we opted to instead use a set velocity instead of a set time step between configurations the RRT algorithm may have been much more successful in creating reduced energy paths.

4.2.5 Genetic Algorithms

Genetic algorithms can be a powerful method for optimizing the robot arm's path planning and torque minimization. However, we found that it did not work as well in problems involving higher degrees of freedom due to the exponential increase in the search space. It could possibly work better with more optimization an a non-Python faster programming language.

We found a convincing path when we run the algorithm without the obstacles, but with them, it would run for multiple tens of minutes, and still get stuck at a position, that is not close to the goal, even when we tried with 10,000 intermediate configurations. Hence, we have left genetic algorithms out of the comparison for collision checking.

Hence, we concluded that genetic might not be a good algorithm for this problem.

5 Conclusions

From our research, we found out that a motion planning algorithm such as RRT works quite fast in finding the path from start to goal, while minimising torque, but uses a high amount of power to get to the goal configuration when using a fixed time step between configurations. The A* algorithm may take longer than RRT, but gives the least torque, and a distance almost as small as RRT. Other algorithms such as greedy and PRM both give similar performance, with low time taken, but average torque and distance. Machine learning algorithms such as genetic take the longest, and do not optimize distance or torque. This is mostly likely due to lack of computation time, and suboptimal optimization with the code. We found that many research papers actually use genetic and we believe that given enough time in the future, the evaluation function and speed could be optimized[1].

We conclude that if minimising torque and distance are the highest priority, without limits on time and computational capacity, A* is the best algorithm. If the shortest path needs to be found in the quickest time from start to goal, without considering torque, then RRT should be used. If the requirement is to give equal weightage to torque and distance, along with no strict time limits, PRM would be best.

References

[1] D. P. Garg and M. Kumar, 'Optimization techniques applied to multiple manipulators

for path planning and torque minimization', Engineering Applications of Artificial Intelligence, vol. 15, no. 3, pp. 241–252, 2002.
[2] F. H. Ajeil, I. K. Ibraheem, M. A. Sahib, and A. J. Humaidi, 'Multi-objective path planning of an autonomous mobile robot using hybrid PSO-MFB optimization algorithm', Applied Soft Computing, vol. 89, p. 106076, 2020.