Optimized motion planning of manipulators in partially-known environment using modified D* Lite algorithm

AMIR FEIZOLLAHI and RENE V. MAYORGA
Department of Industrial Systems Engineering
University of Regina
3737 Wascana Parkway, Regina, Saskatchewan
CANADA
Feizolla@uregina.ca and Rene.Mayorga@uregina.ca

Abstract: Optimized motion planning of the manipulators with regards to their energy consumption level is a challenging problem in robotics which requires a combination of interdisciplinary studies to offer a solution. In this article, a framework is developed to design a motion planner implementing a proposed search algorithm and simulate the robot motion in different environments. The superiority of the search algorithm is investigated and the development of the MATLAB framework is totally discussed accompanying the simulation results.

Key-Words: Motion Planning, Manipulator, Optimization, Graph Search Algorithm, Mathematical Modeling

1 Introduction

In most industrial and daily applications, manipulation is the main or part of the task. Saving labor and reducing the cost of operation has been always the main concern of the designers to minimize the time and effort needed to perform these tasks. In many applications such as exploration in the dangerous environment [1] and rescue operation in natural disasters [2], to dexterous manipulation [3], and point-to-point placement in industrial workspaces [4], using human operator can be either inefficient or dangerous. Robot manipulators, as one of the main group of robots, are designed from the sequence of rigid or/and continuum links, actuators, and end-effector to minimize or entirely eliminate human interaction with the surrounding environment in different workspaces and conditions.

Developing an optimized algorithm to convert a high-level task from human into a low-level description understandable by the robot has been always one of the most interesting research topics in robotics. Motion planning is a step-by-step procedure that generates the most appropriate motion for the robot based on the modeling, analyzing the possible motions (robot's workspace), and the specified criteria. Motion planning can be generally classified into two groups: motion planning in static environment and motion planning in a dynamic environment.

In a static environment, all the information about the obstacles, avoiding criteria, workspace features, and all other variables in the defined cost function are obtainable in pre-processing phase. The motion planning procedure in this type of problems involves analyzing the workspace in order to reach the goal node while avoiding the obstacles.

Another type of motion planning algorithms deals with the problems in which part of the working environment is unknown (partially-known environment), or the robot is exploring the environment for the first time (unknown environment), or some of the environment features like location of the obstacles and the goal node can change (dynamic environment). The algorithms that are developed to solely solve the dynamic environment motion planning problems are mainly based on adaptive path planning where the prior information about the environment is used to generate the best path [5]. In an attempt to deal with the problems of partially-known environment, D* algorithm was proposed in [6]. D* is an informed incremental graph search algorithm that has been widely used for the automatic navigation of the mobile robots. D* Lite [7] which is also an incremental heuristic search algorithm, is built on LPA* to determine the best path from the start node to the goal node while the path costs change due to discovering new obstacles or changes in obstacle features. More details about this algorithm, the procedure of its implementation, and the proposed modification to enhance its performance can be found in the following sections of this article.

Optimized motion planning of the manipulators with regards to the consumed energy of the actuators takes this problem to a new level in which modeling the robot and its actuators as a uniform system play a big role in the motion planning procedure. In this article, a novel method is used to combine the dynamics modeling of the robot and its actuators to develop a single state space describing the motion of the robot in order to minimize the energy consumption of the manipulator during its motion from the start node to the goal node while avoiding the obstacles.

In this study, a modification to D* Lite algorithm is proposed, the proposed algorithm is implemented using MATLAB and the corresponding results are presented. This article is organized as follows: the governing equation of the robot-actuator system is derived in section 2 following by a discussion on the development of modified D* Lite search algorithm in section 3. The procedure of implementing the graph search algorithm using MATLAB is discussed and the corresponding results are presented in sections 4 and 5.

2 Mathematical Modeling of the Robot

As mentioned in the introduction section of this article, the mathematical model of the robot and its equation of motion should be developed to calculate the energy consumption during its motion along the desired trajectory. Using Hamilton's principle [8] for a conservative system between two states from t_1 to t_2, there is a stationary value for the integral in (1) where L is a calculable form (2) in which T and V are the kinetic and potential energy of the system, respectively.

$$I=\int_{t_2}^{t_1} L\,dt \qquad (1)$$

$$L(q,\dot{q},\mathrm{t})=T(q,\dot{q})-V(q) \qquad (2)$$

Taking the derivative of both sides in (1) yields the Lagrange equation of motion for the system:

$$\frac{d}{dt}\frac{\partial T}{\partial \dot{q}_i}-\frac{\partial T}{\partial q_i}+\frac{\partial V}{\partial q}=0 \qquad (3)$$

Rewriting (3) in existence of external forces such as the applied torques by the actuators at the robot's joints,

$$D(q)\ddot{q}+\dot{q}^T C(q)\dot{q}+G(q)=\tau \qquad (4)$$

In which D and C matrices can be determined using (5) and (6), respectively.

$$D=\sum_{i=1}^{n}[\mathrm{m}_i\, J_{V_i}{}^T J_{V_i}+J_{\omega_i}{}^T R_{0_i} I_i R_{0_i}^T J_{\omega_i}] \qquad (5)$$

$$C_{ijk}=\left\{\frac{\partial}{\partial q_i}(d_{kj})+\frac{\partial}{\partial q_j}(d_{ik})+\frac{\partial}{\partial q_k}(d_{ij})\right\} \qquad (6)$$

Where R is the rotation matrix and J is the Jacobian matrix that relates the joint angular velocities to the velocity of any point along the robot's links and vice versa.

DC motors are widely used as the actuator in many control systems. They can be directly connected to the wheels or cables to provide rotary or linear motion.

The applied voltage to the armature (V) is the input of the DC motor and the angular velocity of the shaft is the output. Assuming that the rotor and the shaft are rigid and also with the assumption of proportional viscous friction to the shaft's angular velocity, using Newton's second law and Kirchhoff's voltage law yield

$$J\ddot{\theta}+B\dot{\theta}+\tau=K_t i \qquad (7)$$

$$L\frac{d}{dt}i+Ri=V-K_e\dot{\theta} \qquad (8)$$

Combining (4) and (7), isolating in (8), and rewriting the equations in matrix form:

$$\frac{d}{dt}\begin{bmatrix}\theta\\ \dot{\theta}\\ i\end{bmatrix}=\begin{bmatrix}0 & 1 & 0\\ 0 & \left(\frac{C(\theta)+\mathrm{B}}{J+D(\theta)}\right) & \left(\frac{K_t}{J+D(\theta)}\right)\\ 0 & -\frac{K_e}{L} & -\frac{R}{L}\end{bmatrix}\begin{bmatrix}\theta\\ \dot{\theta}\\ i\end{bmatrix}+\begin{bmatrix}0\\ 0\\ 1/L\end{bmatrix}V+\overrightarrow{NGT} \qquad (9)$$

In which (Nonlinear Gravitational Terms) can be calculated using (10).

$$\overrightarrow{NGT}=\begin{bmatrix}0\\ -\left(\frac{G(\theta)}{J+D(\theta)}\right)\\ 0\end{bmatrix} \qquad (10)$$

Solving the differential equations of motion in (9) that are associated with the dynamics of the robot results in the calculation of energy consumption as a variable in the cost function of the graph search algorithm.

The governing equations of motion of the robot are solved simultaneously using the fourth order of Runge-Kutta method in MATLAB. This solution will give a great close-to-reality approximation of the system's behavior and returns the needed variables for computation of energy consumption of the robot.

3 Graph Search Algorithm

The graph search algorithms can be generally categorized into two groups of incremental search algorithms and heuristic search algorithms. The first one uses the information in the previous searches to solve the current search problem while the latter one uses is mainly based on A* algorithm, in which the heuristic information of each node is taken into consideration to focus the search on reaching the goal node.

D* Lite is an incremental heuristic search algorithm that uses the main features of both methods to speed up the search while putting the priority on reaching the goal node. D* Lite is basically the incremental form of A* algorithm for searching in an unknown environment. D* Lite algorithm uses the same navigation method as D* but considerably shorter. It is as efficient as D* but different in terms of the algorithm. These features make D* Lite one of the best search algorithms for unknown/partially-known environment [7].

D* Lite assigns two estimates of cost in each node, g and rhs. The value of g corresponds to the objective function and rhs is the one-step look-forward value of the objective function. This algorithm also defines the status of “consistency” for each node. The node is

- Consistent if g=rhs
- Inconsistent if g≠rhs

The “Inconsistent” nodes on the “open list” have the priority to be checked. The rhs value of each node is computable using the g value of its successor nodes and the path cost to those nodes using (11).

$$rhs(u) = \min_{p\in Succ(u)} \left(\text{cost}(u,p) + g(p)\right) \tag{11}$$

The key value of a node sets the priority of each node on the “open list”. This value is the summation of the heuristic value of the node, h, and minimum of its g and rhs value. According to (12), in the case of tie, the secondary key value is considered to break the tie.

$$key(u) = \begin{bmatrix} \min\left(g(u), rhs(u)\right) + h(start, u) \\ \min\left(g(u), rhs(u)\right) \end{bmatrix} \tag{12}$$

D* Lite algorithm consists of five procedures: Initialization, Main Procedure, Key Value Calculation, Node Update, and Best Path Generation [7].

Initialization includes creating the open list, assigning a relatively big value to the g and rhs of all the nodes and finally inserting the goal node to the “open list”. The process of finding the best path in D* Lite algorithm, unlike A*, starts with the goal node.

Key Value Calculation procedure involves all the computations that are needed to assign the key value to the input node. The formulation for this procedure is based on (12). Node Update procedure basically sets the rhs value of the node under process and returns it to the appropriate list of nodes.

In Best Path Computation, the consistency status of the nodes with the higher key values on the open list is determined and the appropriate function is called accordingly to change the status of the node if it is needed. If the node is under-consistent then a function is called to change the node to over-consistent status. These changes will be later propagated to the neighboring nodes of the node under process.

The core procedure of D* Lite is Main that has to be run repeatedly to execute the best-path-finding and scanning functions until the best path between the start node and the goal node is generated.

Running D* Lite algorithm, best path can be obtained by following the gradient of g values from the start node. This means that the g values of all the neighboring nodes to each node should be compared to each other and sorted from the start node to the goal node to make sure that the path with the minimum cost is chosen as the best path. D* Lite is an algorithm that is capable of solving the graph search problems with considerably high level of efficiency. However, a simple modification to its best path generation method increases the efficiency of this algorithm significantly.

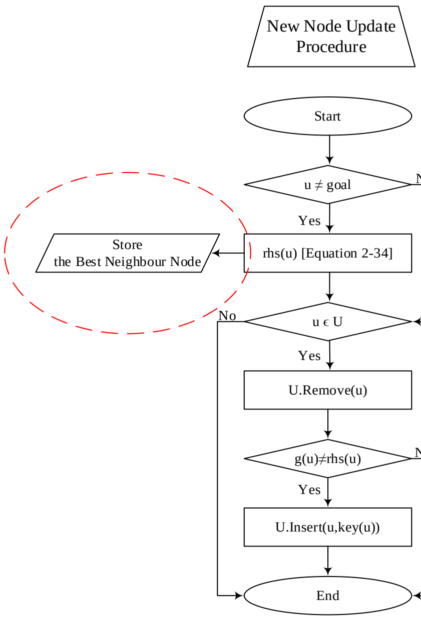


Fig.1 - Modification to D* Lite Algorithm, Node Update procedure

Adding a “store” function in the Node Update procedure of D* Lite, there would be no need to execute an extra procedure for generating the best path in the post-processing phase of the graph search. In Fig. 1, an event (the dashed ellipse) is added to the original procedure under the name of “Store the Best Neighbor Node”. By adding this function to the procedure, after calculating the rhs value of the under process node, a new property will be created to store the best neighbor of the node in terms of rhs calculation. In new Node Update Procedure, generation of the best path is simply piling up the nodes from the start node to the goal node based on the new property added to the nodes.

4 Developing the Simulation Framework

The previous sections provided the foundations for modeling the robot and also delivered the theoretical structure for the proposed search algorithm in partially-known environment. In order to implement the search algorithm on a robot with user-defined characteristics, a MATLAB framework is designed to model the robot, apply the search algorithm, and simulate the robot motion in different scenarios.

The framework consists of thousands of lines of MATLAB script in several classes with different types of functions and properties to execute the above-mentioned tasks. The overall steps for designing such framework can be summarized as developing the below classes:

* Robot dynamics, a class that contains several functions and properties for defining the robot, developing the corresponding equations and doing the needed calculation based on the formulation provided in section 2.
* Motor dynamics, a class that is responsible for developing the equation of motion of the user-defined actuator, solving the coupled differential equations and calculating the energy consumption upon running the graph search algorithm
* Occupancy analysis, group of classes and sub-classes for generating random configuration of the manipulators and identifying the workspace accordingly
* Search algorithm, the class that is developed to implement the proposed search algorithm and find the best path with minimum energy consumption
* Robot simulation, a class that deals with the robot motion simulation and prepares an appropriate graphic output to present the results

There are some assumptions associated with the development of the robot dynamics and motor dynamics classes. It is assumed that the actuators acting on each joint are single function adding only one degree of freedom to the manipulator. In another word, the number of degrees of freedom of the robot is equal to the number of the robot’s links. Although the user can deliberately change the robot characteristics, this assumption converts the motion planning of the robot to a more complicated problem since the robot has to reach the goal node with more restricted limits. It is also assumed that the magnetic field of the DC motors at the joints is constant, the flux is assumed to flow straight into the motor and it can be formulated as indicated in (7). It is also assumed that the actuators have a self-lock mechanism that causes the full stop in actuators motion once the robot reaches the intended configuration. In Motor Dynamics class, all the links features such as length, mass, and configuration are stored in the associated cell arrays and sent to the symbolic method in Robot Dynamics class in which the corresponding equations of motion are developed. The differential equations are then returned to Motor Dynamics class to do the final calculations for the energy consumption.

The main process in the graph search procedure using D* Lite algorithm is to identify the nodes that are to be checked, labeling them to be listed under “Open List” or “Free Nodes List”, and sorting them based on the defined cost function. The process starts with calculating the key value of the Goal node’s neighbors and putting them on the Open List. Based on the key value, the next node will be selected to be checked. The Goal node is then transferred to Free Nodes List and all its neighboring nodes are on the Open List. Following this procedure, removing the processed nodes from the Open List and adding them to the Free Nodes List, the graph search algorithm reaches End once the Start node is added to the Free Nodes List (see Fig. 2).

A MATLAB class, Plot_2D, is developed to deal with the simulation of the robot motion and displaying the output result from the previously discussed classes. Upon finding the best path using the search algorithm and the required MATLAB objects, the plotting class simulates the robot motion. This class also provides a graphic output showing the joints rotation and energy consumption graphs during the robot motion.

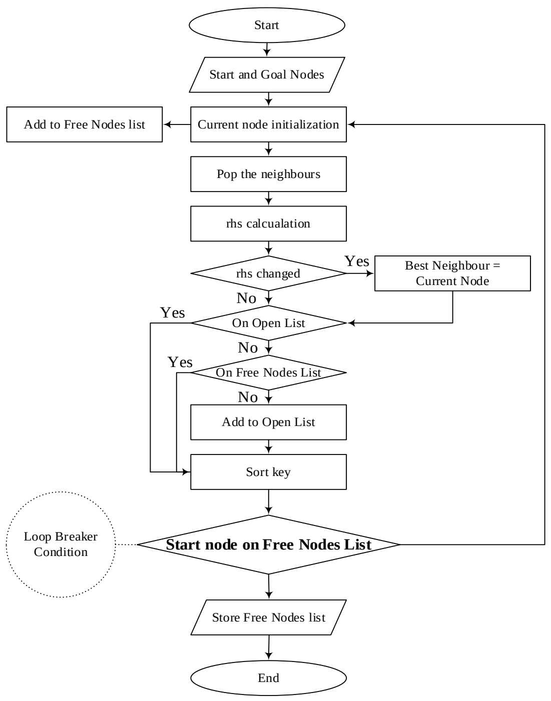


Fig. 2 - Workflow of the modified D* Lite algorithm

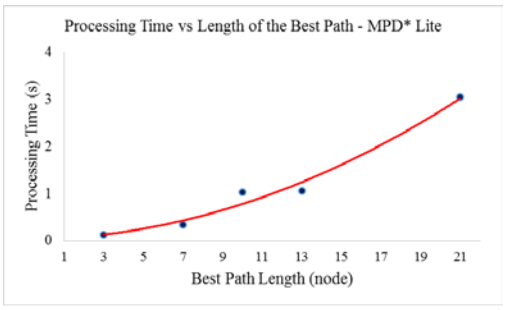
5 Results

In the previous sections, the mathematical modeling, implementation of the search algorithm, and developing the MATLAB framework have been discussed. In this section, some typical manipulation problems are defined and the search algorithm is applied to find the optimized path for the manipulation task. The performance of A* and modified D* Lite algorithms are compared with each other and the re-planning procedure results are discussed.

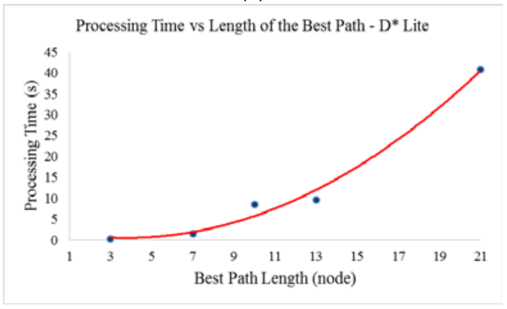
One of the best and easiest ways to investigate the efficiency of two search algorithms is comparing their processing time for finding the best path. For this purpose, the proposed algorithm (modified D* Lite), D* Lite and A* are applied to exactly the same problems. Five different scenarios with five different start-to-goal path length are defined and both algorithms are applied to generate the best path between two given nodes. The outputs, as the best path are exactly the same.

Table. 1 - Comparison between MPD* Lite, D* Lite, and A* algorithms: processing time

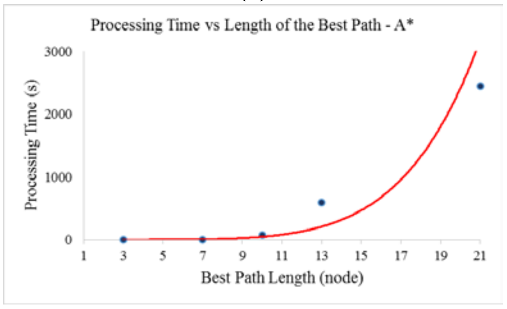
Scenario	Total Nodes	Best Path Length	Processing Time (s)		
			MPD Lite*	*D* Lite*	*A**
1	91	3	0.125	0.348	0.094
2	91	7	0.344	1.553	0.891
3	91	10	1.031	8.564	78.125
4	182	13	1.063	9.678	593.218
5	273	21	3.047	40.849	2450.375



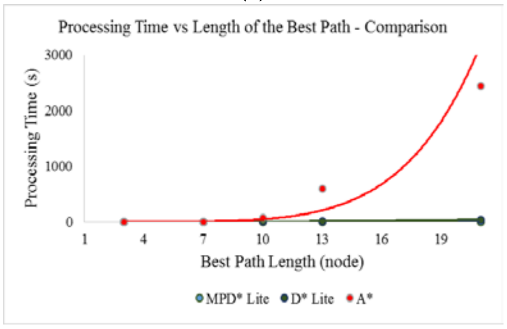
(a)



(b)



(c)



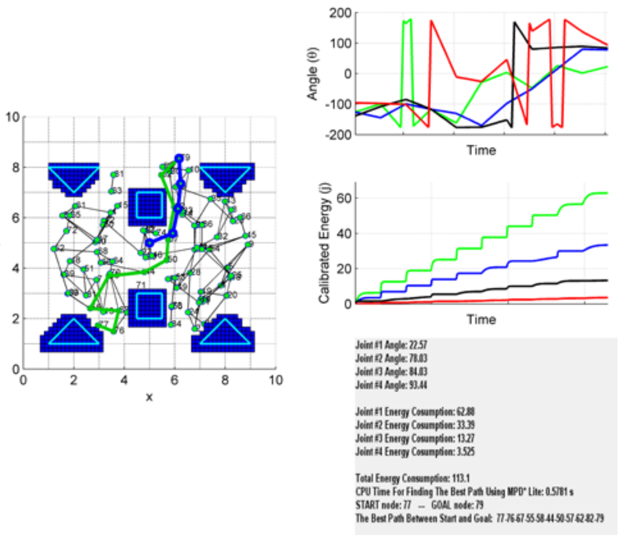
(d)

Fig. 3 - MPD* Lite, D* Lite, and A* graphs:
a) MPD* Lite algorithm: processing time graph
b) D* Lite algorithm: processing time graph
c) A* Lite algorithm: processing time graph
d) Comparative graph for five simulated scenarios

According to the results presented in Table. 1 and Fig. 3, it can be inferred that both open list size and processing time, which are basically the governing factors that affect computational complexity of the algorithms, have exponential relationship with the number of nodes (namely number of all nodes and the length of the best path between the start node and the goal node) in A* algorithm. On the other hand, the computational complexity of D* Lite and MPD* Lite algorithms, according to the same results, has a polynomial relationship with the number of nodes. Although this polynomial relationship can be different for these two algorithms (e.g. different order of polynomial relationship), they both perform much faster than A* graph search algorithm.

For modeling the robot's actuators, DC motors are used as discussed in section 2. It is assumed that all the joints have the same actuators with similar characteristics.

One of the typical manipulation problems for robotic arms is their maneuverability in a tightly crowded workspace. To verify the capability of the proposed search algorithm in finding the optimized path, a workspace is defined as the one in Fig. 4. Six obstacles in different shapes and sizes are located to limit the robot's workspace. The robot's mission is to safely manipulate an object from the right corner of the workspace and drop it in another corner between two obstacles while avoiding any collision with them. The manipulator is defined to have four links of the same length. The energy consumption graph, the processing time and the best path are depicted in Fig. 4.



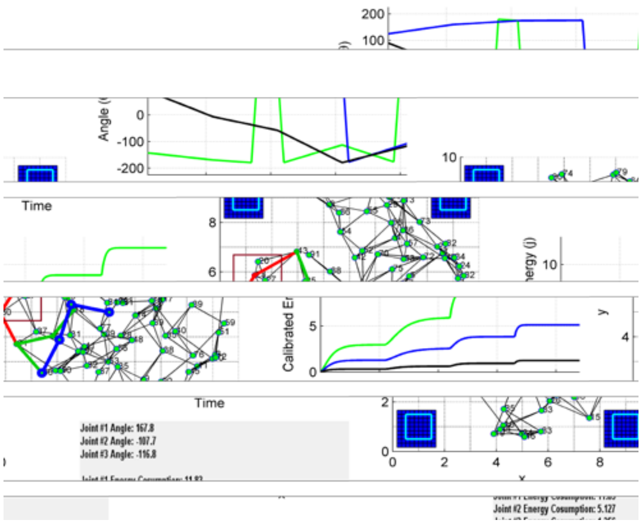
```
Joint #1 Angle: 22.57
Joint #2 Angle: 78.03
Joint #3 Angle: 84.03
Joint #4 Angle: 93.44

Joint #1 Energy Cosumption: 62.88
Joint #2 Energy Cosumption: 33.39
Joint #3 Energy Cosumption: 13.27
Joint #4 Energy Cosumption: 3.525

Total Energy Consumption: 113.1
CPU Time For Finding The Best Path Using MPD* Lite: 0.5781 s
START node: 77  --  GOAL node: 79
The Best Path Between Start and Goal: 77-76-67-55-58-44-50-57-62-82-79
```

Fig. 4 - Energetically optimized path for the manipulator in a crowded workspace

To verify the re-planning procedure of the search algorithm, the robotic arm is located in a partially-known environment where some of the obstacles are pre-defined. The manipulator detects a new obstacle during its motion toward the goal node. As a result, all the nodes neighboring the new obstacle will have an update on their cost and the search algorithm takes the re-planning procedure to find another path with minimum energy consumption. The corresponding result and more details on this scenario can be found in Fig. 5.



```
Joint #1 Angle: 167.8
Joint #2 Angle: -107.7
Joint #3 Angle: -116.8
```

```
Joint #2 Energy Cosumption: 5.127
```

Fig. 5 - Re-planning using the modified D* Lite algorithm in case of detecting new obstacle

6 Conclusions

Using Manipulators in different sizes and shapes for delivering a wide range of tasks from simple repetitive manipulation to performing the maintenance procedure in hazardous workspaces, has been always an interesting and at the same time challenging problem in the field of robotics. In this study, the optimized motion planning of the manipulators in partially-known environment was addressed and a search algorithm was proposed and applied as the solution to this problem. The mathematical modeling of the robot and its actuators was done and the corresponding formulation was derived. D* Lite algorithm as one of the most well-known search algorithms was discussed, its superiority over A* algorithm was studied and a modification was proposed to enhance its efficiency. This algorithm was implemented using a MATLAB framework for generating the best path for the user-defined manipulators in different scenarios. Although there may be much more possible scenarios to study the efficiency of the proposed motion planner, the selected results from tens of investigated scenarios are the most concise and informative cases.

References:

[1] K. S. Senthilkumar and K. K. Bharadwaj, “Multi-robot exploration and terrain coverage in an unknown environment,” Robotics and Autonomous Systems, vol. 60, no. 1, pp. 123–132, 2012.

[2] K. Nagatani, S. Kiribayashi, Y. Okada, S. Tadokoro, T. Nishimura, T. Yoshida, E. Koyanagi, and Y. Hada, “Redesign of rescue mobile robot Quince,” IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), 2011, pp. 13–18.

[3] P. Sabetian, A. Feizollahi, F. Cheraghpour, and S. A. A. Moosavian, “A compound robotic hand with two under-actuated fingers and a continuous finger,” IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), 2011, pp. 238–244.

[4] M. Hvilshøj, S. Bøgh, O. S. Nielsen, and O. Madsen, “Autonomous industrial mobile manipulation (AIMM): past, present and future,” Industrial Robot, vol. 39, no. 2, pp. 120–135, 2012.

[5] J. van den Berg, D. Ferguson, and J. Kuffner, “Anytime path planning and replanning in dynamic environments,” IEEE International Conference on Robotics and Automation (ICRA), 2006, pp. 2366–2371.

[6] A. Stentz, “Optimal and efficient path planning for partially-known environments,” IEEE International Conference on Robotics and Automation (ICRA), 1994, pp. 3310–3317.

[7] S. Koenig and M. Likhachev, “D*Lite,” Eighteenth National Conference on Artificial Intelligence, American Association for Artificial Intelligence, 2002, pp. 476–483.

[8] H. Goldstein, Classical mechanics. Pearson Education India, 1965.