A Framework for Kinematic Modeling and Trajectory Planning of Hyper-Redundant Manipulators Using a Modified PRM

MAHDI F. GHAJARI and RENE V. MAYORGA
Department of Industrial Systems Engineering
University of Regina
3737 Wascana Parkway, Regina, Saskatchewan
CANADA
mfallahinejad@gmail.com and Rene.Mayorga@uregina.ca

Abstract: - Trajectory planning for robotic manipulators can be defined as a set of a step-by-step procedure to break down an arbitrary movement task into discrete motions while satisfying pre-defined constraints and optimizing a cost function. In spite of the fact that various aspects of trajectory planning for robotic manipulators have been investigated; the problem of providing a time-wise efficient collision-free path for hyper-redundant manipulators in cluttered environments, have not been specifically addressed. This research has developed a comprehensive computationally tractable collision-free path planner for several user-defined degrees of freedom (DOF) robot manipulators without using inverse kinematics (IK) which is computationally expensive. This study introduces a novel efficient multiple-query based sampling approach for obstacle avoidance, and 2D trajectory planning, for N-DOF robot arms. A MATLAB based motion planner is proposed to investigate this approach for different and diverse types of manipulators, with various joint types, and cost functions. Various scenarios with different pre-defined highly constraining obstacles have been simulated in the proposed motion planner and the results demonstrate the fast computation of collision free motions.

Key-Words: - Trajectory Planning, Hyper-redundant Manipulators, Collision-free Motion Planning

1 Introduction

According to the definition of Lozano [1], motion planning can be defined as the process of converting the robot motion task into a set of computed discrete movements in order to satisfy some constraints and optimize some aspects of the robot motion between two locations.

The objective of motion planning is giving the robot the ability of automatically deciding and guiding the order of motions execution in an environment filled with obstacles. Use of cost functions will be mandatory in economic considerations, such as minimum time, energy or operating in the shortest path [2]. Our goal in a motion planning problem is to find a collision-free motion for an object from start configuration to a goal configuration. This is proved as a hard problem [3], since the complexity of motion planning increases dramatically as the number of DOFs of moving object grows.

The history of motion planning for robotic arms is as old as that of industrial robot invention. George Devol was the creator of Unimate, the first industrial robot in the 1950s that worked on a General Motors assembly line in New Jersey, in 1961 [4].

The humanlike robot arms normally have 7-DOF which are typically referred as redundant manipulators and the ones with greater than 7-DOF are considered as hyper-redundant manipulators. Sampling-based motion planning algorithms have been proven to be successful in high dimensional C-space that have a probability of failure which decreases to zero as more time is spent. Sampling-based motion planning methods are currently considered as state of art for motion planning.

Although there is a wide range of different trajectory designing algorithms, currently there exists no general method, which outperforms all others for all problem instances. In fact, each method has various advantages and drawbacks that make it best suited for specific types of

problems. Furthermore, since a workspace can contain vast various regions with different characteristics, there may not be a single planner that will perform well in all these types of regions [5].

Bohlin et al [6] described an algorithm based on PRMs that is called Lazy PRM. This approach is designed to minimize the running time of the planner by minimizing the number of collision checks performed during planning. The planner constructs a roadmap assuming that all the nodes and edges in the roadmap are in the obstacle-free region. Then the planner searches for the shortest path between the start and the goal node. The planner removes the nodes and edges which are in the obstacle space along the path as it is searching for the shortest path. Experiments indicate that the introduced Lazy PRM is efficient in practice.

Song et al [7] presented a customized version of PRMs method that postpones some of the validation checks such as collision check to the query stage which yields an efficient outcome for many motion planning problems. This approach enables the user to customize the same roadmap according to multiple, variable, query preferences. The results indicate a huge improvement in performance of the motion planner.

Branicky et al [8] by implementing an innovative algorithm, highlighted the advantages of their algorithm in comparison with random sampling. Quasi-random variants based on PRM planner are developed as (1) classical PRM with quasi-random sampling, and (2) a quasi-random lazy-PRM.

Bertram et al [9] proposed a novel integrated approach to the problem of inverse kinematics for redundant manipulators in manipulation tasks. The proposed solution is based on exploring a connected collision free configuration with RRT and evaluating the candidate configurations by heuristic workplace metric, which measures the ability to reach to the desired pose. The goal distance along with the obstacle distance information is later used for guiding the configuration space search.

Karaman et al [10] described an anytime method based on RRT that quickly computes an initial feasible plan, but unlike the RRT, converges to an optimal path. Two key extensions to the RRT committed trajectories and branch-and-bound tree adaptation were presented to enable the technique of having an efficient use of calculation time online, resulting in an anytime method for real-time implementation.

Gomez-Bravo et al [11] proposed a new approach to the problem of collision-free trajectory planning for hybrid robots in the presence of the obstacles. This method is based on a combination of random search algorithm (RRT) with an optimization method that is solved by a genetic algorithm.

None of the abovementioned studies address the following issues:

- A general trajectory planner for robot manipulators that can support unlimited degrees of freedom, different joint types (revolute and prismatic), different cost functions, and all types of obstacles.
- Lack of multiple queries in most algorithms
- Most of the previous studies have been conducted for specific scenarios.

Accordingly, the following objectives are defined as the contribution of this study for addressing the hyper-redundant manipulators motion planning problem:

- Developing a comprehensive trajectory planner capable of handling different scenarios including non-redundant, redundant, and hyper-redundant manipulators with different joint types.
- The capability of mapping all types of obstacles and workspaces.
- Handling various cost functions based on the scenarios.
- User-control accuracy and resolution based on the scenarios.
- High reliability in obstacle avoidance application.

2 Methodology

In this section, a new methodology for developing a collision-avoiding trajectory for redundant and hyper-redundant robot arms in different types of environments is discussed. This methodology is based on sampling-based motion planning that can be divided into two phases: preprocessing phase (or off-line phase) and query phase (or on-line phase).

2.1. The Preprocessing Phase (off-line)

During the preprocessing phase as an off-line stage, a data structure (or roadmap) is constructed in a probabilistic way for a given workspace. The graph contains nodes which are chosen over C_{free} and the edges that correspond to the feasible trajectories. A proper sampling method is needed to generate collision-free configurations in C_{free}. A uniform generation of nodes continues until the desired density is achieved. Each generated random configuration needs to be checked by the collision detector method, whether the randomly generated configuration is in a collision with obstacles. Once random nodes are created, they are connected to their neighboring nodes. The local planner computes a collision-free trajectory between a node and its neighbors. Then, randomly created configurations and their connections are added to the graph. In this stage, a roadmap is generated in a way that can quickly handle future queries [12].

The Initial empty graph includes only occupied regions. Then, repeatedly, a random free configuration is created and added to the number of nodes. As the planner generates nodes, the local planner connects the generated node to the neighboring nodes with a straight line representing the feasible trajectory between these two nodes. For mapping the obstacle the occupancy grid has to be created. The workflow of pre-processing phase is depicted in Figure 1.

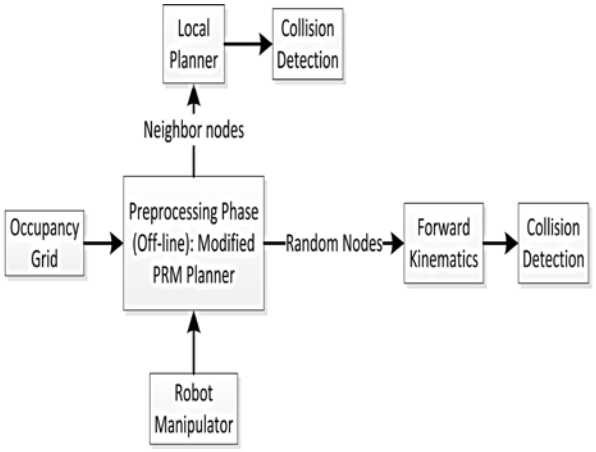

Figure 1. Workflow of the pre-processing phase

According to [13], the pseudocode of learning phase (or PRM planner algorithm) can be outlined as follows:

1 ***G***.*init(**AllNodes**={}, **Edges**={}),i←0;*
2 **while** *i<N*
3 ***if*** $q_{new}(i) \epsilon C_{free}$ **and** *((**not** **G**.same_comp* $(q_{new}(i),q))$ **then**
4 ***G***.*add_vertex*$(q_{new}(i))$*;i←i+1*
5 **for each** $q \epsilon$ *NEIGHBOURHOOD*$(q_{new}(i),$***G***$)$
6 **if** ***G***, *trajectory*$(q_{new}(i),q) \epsilon C_{free}$ **then**
7 *CONNECT*$(q_{new}(i),q)$ **then**
8 ***G***.*add_edge*$(q_{new}(i),q)$

The preprocess phase of the trajectory planning can be accordingly divided into following steps:

1) Robot Manipulator

The first step in the off-line phase is defining the robot manipulator with all possible details. These details include the number of degrees of freedom (DOF), Denavit-Hartenberg D-H parameters, joint type, joint limit, and link cost values (pre-defined weight for each link).

2) Occupancy Grid Mapping

Trajectory planning requires both manipulator and workspace information. The

manipulator information can be safely defined as the number of degrees of freedom, number of links and their dimension. The workspace information consists of obstacles location and their dimension. To create occupancy grid, a numerical value is assigned to each cell that indicates the probability of the cell existence and also the difficulty of its reachability [13].

3) Forward Kinematics Calculation

In this Paper, the D-H convention for forward kinematics (FK) calculation is used to compute the position of the links and the end-effector [14]. It is prohibitively difficult to explicitly calculate the shape of C_{free}; however, detecting whether a given configuration is in C_{free} is an efficient solution to this problem. First, the manipulator configuration is calculated and then the collision detector recognizes if the manipulator intersects any of the obstacles.

Equation 1 expresses the location and orientation of the end-effector frame with respect to the base frame as:

$$T_{0n} = A_{01}(q_1)A_{12}(q_2)A_{23}(q_3)\ldots A_{n-1,n}(q\_n) \quad (1)$$

4) Random Sampling

According to Lavalle [13], C_{space}, the configuration space, is "uncountably" infinite while a sampling-based motion planning method can consider at most a countable number of random samples. This method can run forever then it may be "countably" infinite, but practically the expectation is to terminate the algorithm after considering a finite number of random samples. Therefore, the planner generates dense random configurations for any bounded C_{space}. The maximum number of nodes for each scenario and needed roadmap can be equated to the mass, therefore, the total number can be computed as follows:

$$\text{Number of Total Nodes} = \text{Density} \times \text{Volume of } C_{free} \quad (2)$$

5) Tip Distribution Grid

Although the Beta distribution with tuned parameters yields to a more uniform distribution of manipulator's end-effector, there is still a dense region of generated nodes around center towards the right side of the map.

Therefore, the tip distribution grid (TDG) solution is applied to satisfy the distribution problem. The TDG controls the random generation of configurations by controlling the position of manipulator end-effector to uniformly distribute it over the entire workspace. Each TDG cell has a desired and predefined limit of number of nodes. Hence, if the limit is exceeded by the generated random configurations, the generated node will be discarded and the iteration will continue to spread nodes until all TDG cells reach the maximum number of nodes. The maximum number of nodes for each TDG cell is considered as a "mass", thus it can be assumed that the density multiplied volume can be calculated as follows:

$$\text{Maximum Nodes per Cell} = \text{Density} \times \text{TDG Cell Volume} \quad (3)$$

6) Collision Detection

Once a random configuration is generated and the location of the created sample is determined, the collision status of the configuration should be investigated. Hence, collision checking is a critical component of sampling-based planning [8]. The collision detector checks if the generated configuration intersects any part of C_{obs}. If it is collision-free, it will be added to the total generated collision-free nodes; otherwise, it will be discarded. The first step is the calculation of the end-effector's position. Then, this position is sent as a point to the occupancy grid. If the point lies in C_{obs} that point will automatically be discarded. Once the collision detection process is done and the generated configuration's end-effector position is in C_{free}; then we need to check if any part of the robot collides with an obstacle. The assumption during this process is that the robot cannot intersect itself or the links have an appropriate offset to avoid this collision.

7) Finding Neighboring Samples

Once all random nodes are created and the collision detection is performed for each one of them, they can be considered as the valid nodes and their configuration is placed in the C_{free}. Next crucial step is to find the candidate neighbors around each node. Each connection (edge) in the graph indicates a pair of neighbor nodes (Figure 7). One of the critical factors in sample-based path planning algorithms is the process of pairing close nodes. A criterion (i.e. distance) is required to determine a neighborhood region for each node and then for their connection. The strategy that has been used to find the neighbor nodes in the proposed methodology is the Tip Distribution Grid (TDG). Not only the applied TDG technique solves the problem of random nodes distribution, but also it can be used as a criterion to recognize the neighboring nodes in the workspace.

8) Local Planner

The local planner is in charge of connecting the neighbor nodes. The straight-line segments in Figure 7 shows the connections between two nodes. However, the trajectory between two neighbor nodes, which is not necessarily a straight line, is generated by the local planner while satisfying the collision-free condition.

The quality of the collision detection for the trajectories depends on the workspace and obstacles' geometry. For instance, if there are obstacles with sharp edges and vertices the local planner must generate paths with higher resolution.

9) Cost Function

Path planning for highly redundant manipulators involves cost functions that have to be optimized since there are large numbers of degrees of freedom performing a task. Hereby, one of the main contributions of the proposed technique for hyper-redundant robotic arms motion planning is handling and optimization of various cost functions simultaneously. In this study the cost function is defined to be a function of following parameters:

- Distance: finding the shortest Euclidean distance between the end-effector positions from initial to goal configuration.
- Time: finding the shortest trajectory between initial and goal configuration.
- Total joints displacement: the objective is having minimum joints displacement in accordance to the assigned costs for each link.

2.2. The Query Phase (real-time)

By using the constructed roadmap in the off-line preprocessing phase, paths are to be found between any initial and goal configurations during the query phase. That is, once the roadmap is generated, it contains occupancy grid, all valid nodes, and all feasible obstacle-avoiding trajectories between neighbor nodes and initial robot properties. The strategy that has been used in the query phase for connecting any arbitrary pairs of nodes is finding the shortest/least cost path between any desired start and goal nodes that involves the graph searching algorithm A*. The workflow of this phase is shown in Figure 2.

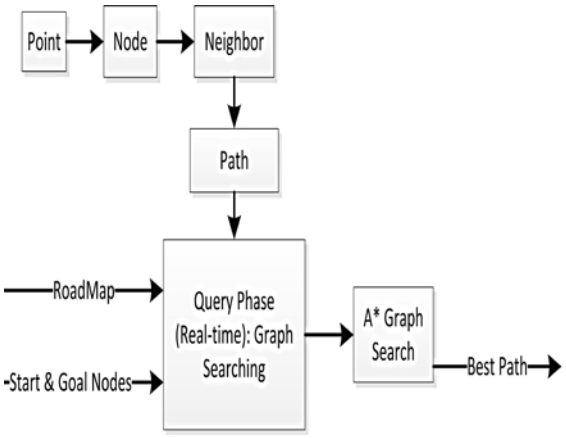

Figure 2. Real-time query phase workflow

A* is the best-first graph search algorithm that has been widely used in pathfinding because of its performance and accuracy to find the efficient path. Additionally, other studies found A* to be superior to other methods [15]. Detailed information about A* can be found in [16].

3 Implementation

The proposed approach is implemented in a MATLAB platform because this platform is widely accepted for academic purposes as well as in

industry. The overall procedure of implementation of the proposed algorithm can be listed as follows:

The workflow of the developed modified PRM planner is depicted in Figure 3 According to this workflow, the off-line phase starts with the generation of obstacle-avoiding random configurations followed by the robot and environment initialization to create a roadmap to begin the query phase, which needs a pair of initial and goal nodes to find the best path.

To efficiently develop and implement the proposed approach in MATLAB, one should use the advantages of Object-Oriented Programming (OOP) since the planner is dealing with a large number of nodes and numerous properties for each node as well as the edges and the trajectories between each pair of neighbor nodes. Figure 4, demonstrates the structure overview of this planner including the classes, relations, and connections.

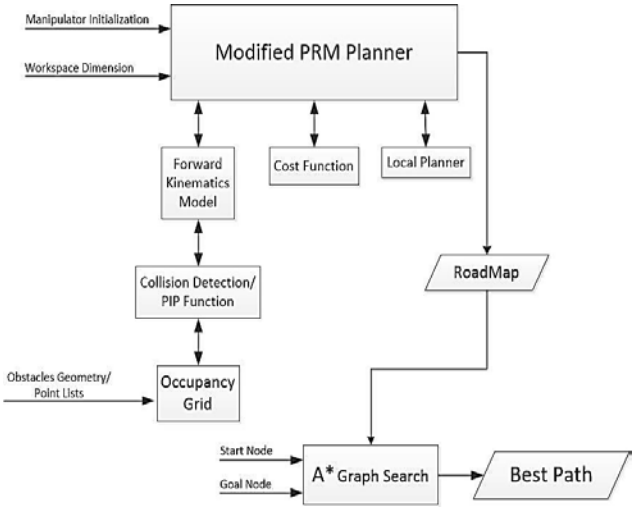

Figure 3. Overview of presented planning algorithm

Implementing an OOP programming would also solve the problem of handling the numerous assigned properties to the nodes, edges and randomly generated trajectories. Defining different classes with different properties is an appropriate approach in dealing with such problems. These classes function like a template for creating specific instances of the classes which are called objects. These objects contain all or part of the data for a particular entity that is designed base on that class. Objects can actively manage the set of data by calling certain functions and methods to be executed. Figure 4 illustrates the workflow of MATLAB implementation. This workflow includes all the classes and their relationship to each other and how they feed data to following classes and objects to perform a trajectory planner for a user-defined environment and robot.

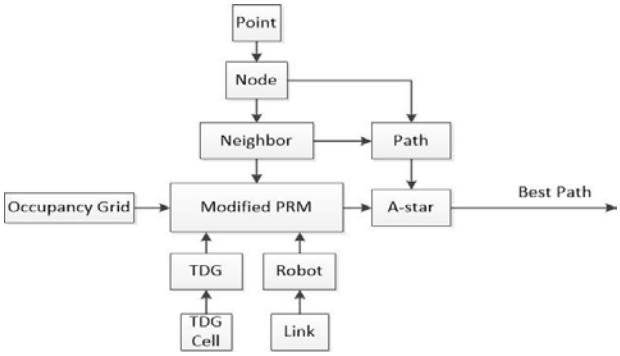

Figure 4. Relations of the user-defined classes in MATLAB for simulation

1) Robot Manipulator Class

To develop a model of the manipulator that the motion planning is being defined on, the robot has to be initialized in MATLAB. This MATLAB class consists of few sub-classes to set the robot properties including its number of DOF, the links and their properties, the robot's origin, and its other physical properties. Link class can be considered as a sub-class to this class which defines some specific link properties such as their ID number, D-H parameters, joint types, their weight cost (as one of the parameters in the *Cost Function*).

2) Node and Point Classes

As discussed in the previous sections, creating random nodes to develop a motion planner for the robot, the corresponding random configuration of the robot have to be defined. This step can be done by assigning random values to the pre-defined properties in the *Robot Manipulator* class. In the process of generating random configurations, one of the most important issues that should be handled properly is to determine that the nodes are uniformly distributed.

In Figure 5.a, a 7-dof manipulator with the maximum length of 8.25 units was chosen to illustrate the end-effector position distribution in black dots. Not only the normal random

generator has densely generated samples around the center but also the manipulator end-effector has barely reached the workspace boundaries. Hereby, the shape of the distribution needs to be controlled. Using Beta distribution as a family of continuous probability distribution that is parameterized by two positive shape parameters, denoted by α and β, the result is much closer to a uniform randomized node generation. Figure 5.b indicates the effect of Beta distribution that allows a uniform distribution all over the workspace and reachability of the boundaries.

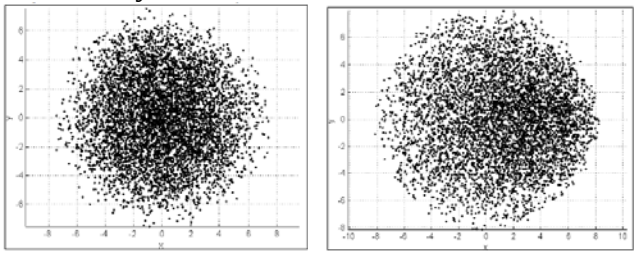

Figure 5.
(a) End-effector position distribution of 6000 generated random configurations for 7-dof manipulator (maximum length 8.25 units) using simple random generator.
(b) End-effector position distribution of 6000 generated random configurations for 7-dof manipulator (maximum length 8.25 units) using Beta distribution, $\alpha, \beta=5$

3) Occupancy Grid Class

Occupancy Grid class is another class that has to be defined for the initialization process of the motion planner. In this class, the environment properties such as its size, the obstacle locations, their size, and also the grid resolution to map the obstacle in the environment. Discretization resolution of manipulator workspace defines the number of cells per axis and the quality of approximation. The occupied cells are listed in C_{obs} and marked with color blue on the roadmap (Figure 6).

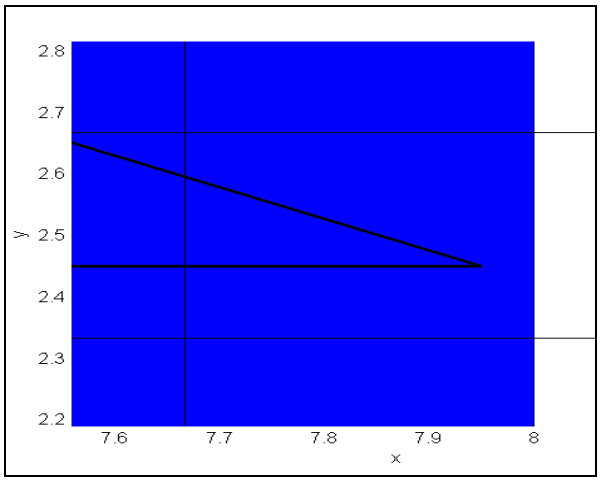

Figure 6. Securely occupancy grid mapped obstacle, resolution: 30 cells per 10-unit-axis

The Point in Polygon (PIP) method in general uses repeated geometric queries. Given multiple polygons and a sequence of query points, PIP finds the status for each query point. Each cell calls for PIP function and checks if any vertex of the cell is in the defined polygon. If the answer value is "1" then that cell is considered as an occupied cell and marked with color blue. An extension has been added to this method that adds all of the surrounded cells of each occupied cell to the blocked area.

4) Distribution Grid and TDG Cell Classes

As depicted in Figure 5, although the controlling parameters of α and β makes the distribution of randomly generated nodes as uniform as possible, there is still a rather dense region in the center of the map. *TDG* class is defined to control the number of nodes generated in each cell of the map. This class uses the *Occupancy Grid* class and some of its defined properties (e.g. *Density*, *Walking Distance*, *Max. Nodes per Cell*) as the input to construct a new instance of TDG.

5) Neighbor Class

After generating all the random nodes and detecting collision by performing collision avoiding procedure explained in the previous section, the existing nodes on the maps are the valid nodes that the robot can pass its motion through them. Next step is assigning the neighboring nodes to each generated node. According to TDG Cell method, each cell contains the nodes that are within the *Walking*

Distance to the other nodes in that cell. Therefore, the local planner is called by the origin class to connect these nodes and generate a linear trajectory between them (Figure 7).

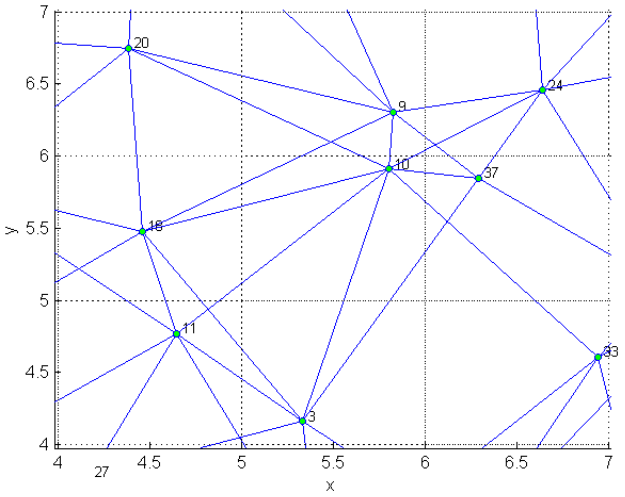

Figure 7. A 7-dof manipulator neighbor nodes selection; workspace dimension= 10 unit by 10 units, robot origin= (5, 5), density= 1

6) PRM Class

Once all the necessary classes for initialization are started then the trajectory planner is called to construct the roadmap by repeatedly generating random nodes and checking their collision free feature. The modified PRM planner runs by calculating an estimation of needed total nodes with the desired density. The verified nodes are then added to an array in this class called *All Nodes* which basically contains all the eligible nodes for the roadmap. Every single trajectory generated between the neighboring nodes are checked by collision detector to determine it does not collide any of the pre-defined obstacles.

7) A-Star and Path Classes

A* is a graph search algorithm that has been used in this study to handle the defined cost function and develop the trajectory planner. A* is widely used in the graph search applications and many studies have shown its superiority to the other approaches. A* is a best-first search algorithm and finds the path from the start node to the goal node with the least cost value.

A* traverses the graph, it follows a path of the lowest expected total cost, keeping sorted priority queue of alternate path segments along the way. An amount of cost computed for each edge in the graph is equal to the summation of displacement of each joint during local planning for neighbor nodes, multiplied by the link cost assigned by the user.

All Nodes array is fed into this class as an input to construct a new instance of *A-Star* and its objects are created as the roadmap and an empty *open list*. There are several methods (functions) in this class that are designed to use the previous classes output and generated the best path. These methods include: *Is In Path*, *Cost to Node*, *Path Length*, *Add Node*, *Sort* and *Clone Path*.

4 Results and Discussion

The following scenarios are executed in MATLAB framework. In the resultant figures, the manipulator and its joints are shown in bold black line, and green circles, respectively. The obstacles are shown in cyan line segments.

4.1. Scenario No.1

The first scenario contains a scene populated with large obstacles and a redundant fixed-base 6-DOF manipulator. The mission is designed for a situation that a redundant manipulator operates in a tight area. Figure 8 demonstrates the capability of the presented general planner in designing motions for a redundant robot arm moving along tight workspace from start configuration at the bottom left corner to top right corner keeping safe clearance with obstacles.

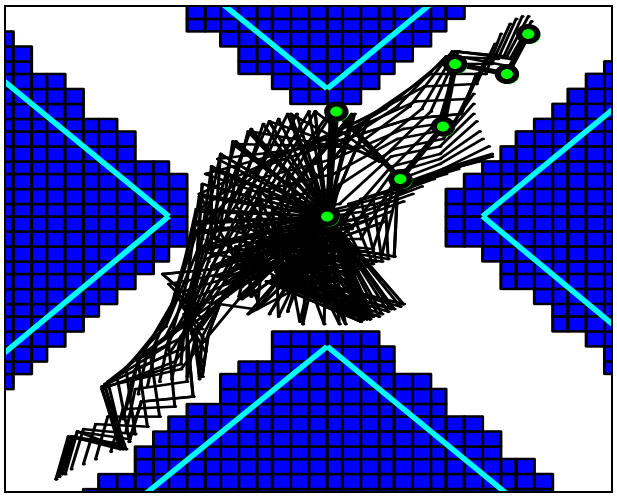

Figure 8. The designed trajectory of a 6-DOf manipulator; joints are shown in green in final configuration

4.2. Scenario No.2

The second scenario consists of a humanoid robot arm, which is a 7-DOF redundant manipulator operating in a crowded workspace populated with many small obstacles. Figure 9 illustrates the maneuverability of the presented planner in crowd workspaces. As can be seen in the bottom right corner of this figure, the manipulator folds itself perfectly to avoid any collision whereas the planner is random-based.

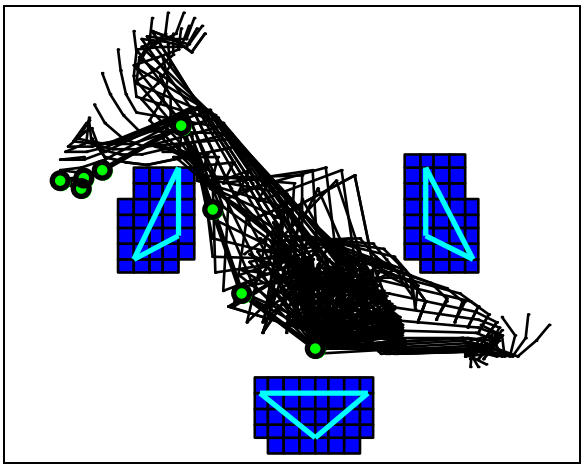

Figure 9. The designed trajectory for a 7-DOf manipulator; joints are shown in green in final configuration

4.3. Scenario No.3

The third scenario is presenting a simulation for the Canadarm2, manipulating an object form an initial configuration and move along the free spaces and then places the object in a tight location surrounded by obstacles like a box. Figure 10 presents one of the challenges that may Canadarm2 can encounter while operating at ISS which has simply done by the proposed planner. The final configuration is exactly the best-obtained configuration to reach that end-effector position deep inside a narrow space.

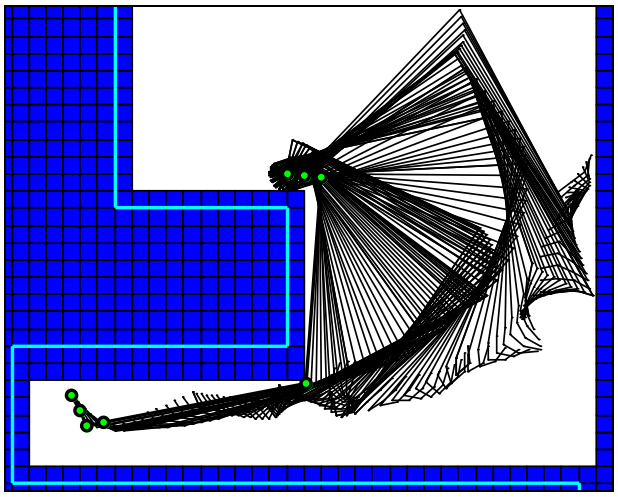

Figure 10. The designed trajectory for Canadarm2; joints are shown in green at the goal configuration

4.4. Scenario No.4

In the 4th scenario the performance and capability of the proposed motion planner is going to be checked for an RRP manipulator having a prismatic joint. Figure 11 shows one of challenging features of the proposed method which has not been addressed reasonably in the past works. This scenario designed to illustrate the perfect capability of using prismatic joint for end-effector reaching the farthest boundaries of the workspace by only sliding.

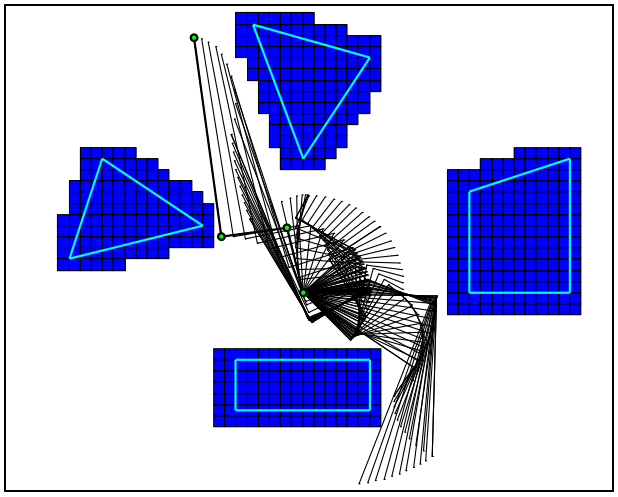

Figure 11. The designed trajectory for the RRP manipulator (end-effector is slider); joints are shown in green in goal configuration

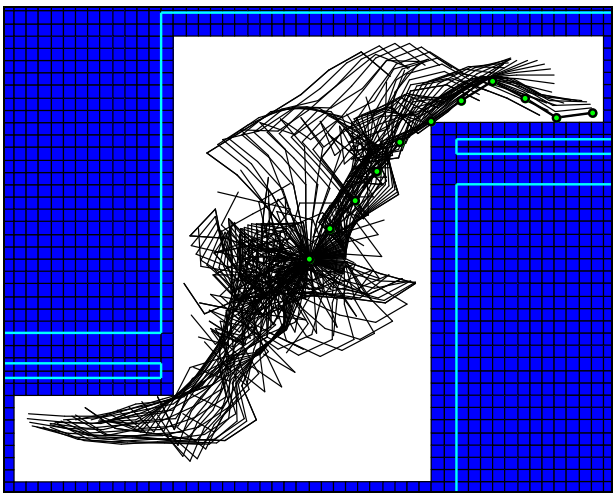

Figure 12. The designed trajectory for a 10-DOF manipulator; joints are shown in green at the goal configuration

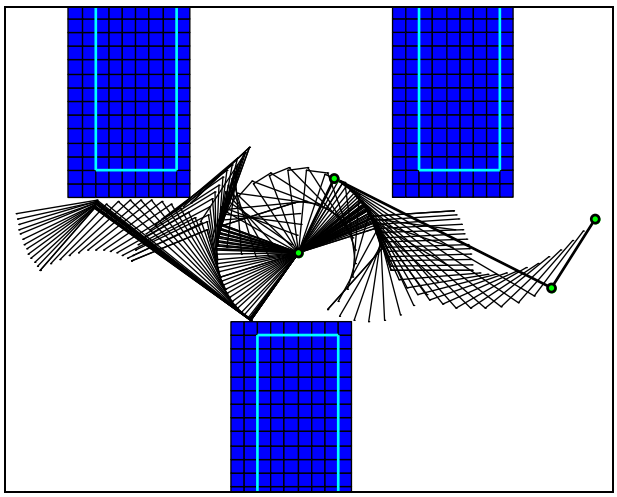

Figure 13. The designed trajectory for the RPR manipulator (Second link is slider); joints are shown in green in goal configuration

4.5. Scenario No.5

The fifth scenario is presenting the capability of planning obstacle avoidance trajectory of the proposed motion planner for a 10-DOF robot arm in a crowded environment or highly constrained workspace. Figure 12 demonstrates a hyper-redundant 10-DOF robot arm manipulates an object deep inside a box located at the bottom left corner and placed deep inside another box located at the top right corner.

4.6. Scenario No.6

The planner plans an obstacle avoiding path for an RPR (Revolute-Prismatic-Revolute) manipulator which means there is a slider sliding for the second link. The designed robot arm has 90-degree prismatic joint for the second link which makes it more constrained for obstacle avoidance maneuver. As can be seen in 0 Figure 13, the designed collision-free trajectory employed the feature of sliding joint specifically to avoid collisions.

4.7. Scenario No.7

The last scenario is presenting the capability of the proposed planner in collision-free trajectory planning for a 16-DOF hyper-redundant manipulator which has not been properly addressed comprehensively in the past studies. Sharp-edge obstacle avoidance of an unlimited-DOF robot arm is one of the boldest features of the introduced efficient and quick path planner. Figure 14 shows an example for scenarios that consists of an unlimited-DOF manipulator avoiding collisions in highly constrained workspaces including sharp-edge obstacles.

Table 1 shows the performance of the proposed trajectory planner for selected scenarios. Its columns represent the total number of nodes which is one of the most important parameters in defining the difficulty of the problem, off-line CPU-time (secs), and Real-time CPU-time (secs).

According to this table, due to the length of roadmap trajectories, density and occupancy grid resolution, the CPU-time (secs) for off-line phase increases up to nine minutes (scenario No.7). However, the real-time phase is done in a quite short period of computation time (secs) for a 16-dof robotic arm in comparison to other methods and non-sampling based approaches. More detailed results can be found in [18].

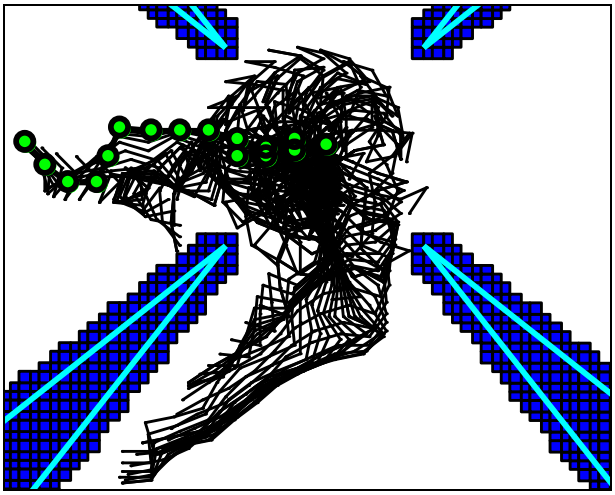

Figure 14. The designed trajectory for a 16-DOF manipulator; joints are shown in green at the goal configuration

Table 1. Comparison between the results of defined scenarios

	Number of Nodes	Off-line CPU Time	On-line CPU Time
Scenario 1	90	28.65	3.20
Scenario 2	187	74.15	0.80
Scenario 3	109	276.4	34.80
Scenario 4	46	39.31	2.70
Scenario 5	65	157.8	26.59
Scenario 6	50	45.96	4.10
Scenario 7	150	539.3	40.30

* All the simulations are executed using Windows 8.1 on a Lenovo ThinkPad W530 processing with Intel Core i7-3820QM 2.70 GHz (64-bit) processor and 8 GB of RAM.

Please take notice that motion planning and obstacle avoidance in 2-D; in general, can be more difficult and restrictive than in the 3-D space. This is widely acknowledged worldwide. Still, the proposed approach can be easily extended to motion planning and obstacle avoidance in 3-D, for redundant and hyper-redundant manipulators [18].

5 Conclusions

In this paper, a novel approach has been presented for setting a MATLAB framework and developing an efficient collision-free trajectory planner for redundant and hyper-redundant manipulators. An innovative method has been presented based on a modified multiple query PRM which is capable of dealing with a wide range of variety of the problem; including robots with different number of DOF, joint types, and cost functions. A MATLAB framework has been developed to get the user defined robot characteristics, and simulation environment features. The trajectory planner has been applied to seven scenarios using an A* search algorithm and the corresponding results have been presented. This study can be expanded by implementing a 3D collision detection function and including the dynamics of the manipulators.

Acknowledgements

The authors would like to express their appreciation to Mr. Mehrdad Bakhtiari for his valuable suggestions and advice on the path planner programming.

This paper research has been supported by a grant (No: 155147-2013) from the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] T. Lozano-Perez, "A simple motion-planning algorithm for general robot manipulators," Robotics and Automation, IEEE Journal of, 3(3), pp. 224-238, 1987.

[2] A. Feizollahi and R. V. Mayorga, "Optimized Motion Planning of Manipulators in Partially-Known Environment Using Modified D* Lite Algorithm", *WSEAS Transactions on Systems*, vol. 16, no. 10, pp. 69-75, 2017.

[3] J. H. Reif and H. Wang, "Social potential fields: A distributed behavioral control for autonomous robot," Robotics and Autonomous Systems, vol. 27, no. 3, pp. 171-194, 1999.

[4] J. N. Pires, "Industrial robots programming: building applications for the factories of the future" *Springer*, 2007.

[5] M. Morales, L. Tapia, R. Pearce, S. Rodriguez, and N. M. Amato, "A machine learning approach for feature-sensitive motion planning," In Algorithmic Foundations of Robotics VI, pp. 361-376, 2005.

[6] R. Bohlin and E. E. Kavraki, "Path planning using lazy PRM," in ICRA'00. IEEE International Conference on Vol. 1, 2000, pp. 521-528.

[7] G. Song, S. Miller, and N. M. Amato, "Customizing PRM roadmaps at query time," in Proceedings 2001 ICRA. IEEE International Conference on Vol. 2, 2001, pp. 1500-1505.

[8] M. S. Branicky, S. M. LaValle, K. Olson, and L. Yang, "Quasi-randomized path planning," in Proceedings 2001 ICRA. IEEE International Conference on Vol. 2, 2001, pp. 1481-1487.

[9] D. Bertram, J. Kuffner, R. Dillmann, and T. Asfour, "An integrated approach to inverse kinematics and

path planning for redundant manipulators," in Proceedings 2006 IEEE International Conference, 2006, pp. 1874-1879.

[10] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," The International Journal of Robotics Research, 30(7), pp. 846-894, 2011.

[11] F. Gómez-Bravo, G. Carbone, and J. C. Fortes, "Collision free trajectory planning for hybrid manipulators," Mechatronics, 22(6), pp. 836-851, 2012.

[12] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," Robotics and Automation, IEEE Transactions on, 12(4), pp. 566-580, 1996.

[13] S. M. LaValle, Planning algorithms. Cambridge university press, 2006.

[14] F. C. Samavati, A. Feizollahi, P. Sabetian, and S. A. A. Moosavian. "Design, Fabrication and Control of a Three-Finger Robotic Gripper." In Robot, Vision and Signal Processing (RVSP), 2011 First International Conference on, pp. 280-283. IEEE, 2011.

[15] W. Zeng and R. L. Church, "Finding shortest paths on real road networks: the case for A*," International Journal of Geographical Information Science, pp. 531-543, 2009.

[16] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," Systems Science and Cybernetics, pp. 100-107, 1968.

[17] R. V. Mayorga, F. Janabi-Sharifi, and A. K. Wong, "A Fast Approach For The Robust Trajectory Planning Of Redundant Manipulators," Journal of Robotic Systems, vol. 12, no. 2, pp. 147-161, Feb. 1995.

[18] M. F. Ghajari, "Trajectory Planning for Hyper-Redundant Manipulators in Constrained Workspaces", M.A.Sc. Thesis, Industrial Systems Engineering, University of Regina, Canada, March, 2015.