IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 21, 2020, accepted September 10, 2020, date of publication September 18, 2020,
date of current version September 30, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3024630

Control Framework for Trajectory Planning of Soft Manipulator Using Optimized RRT Algorithm

AMEER TAMOOR KHAN[1], SHUAI LI[1], (Senior Member, IEEE), SEIFEDINE KADRY[2], (Senior Member, IEEE), AND YUNYOUNG NAM[3], (Member, IEEE)

[1]Department of Computing, The Hong Kong Polytechnic University, Hong Kong
[2]Department of Mathematics and Computer Science, Faculty of Science, Beirut Arab University, Beirut 11072809, Lebanon
[3]Department of Computer Science and Engineering, Soonchunhyang University, Asan 31538, South Korea

Corresponding authors: Shuai Li (shuaili@polyu.edu.hk) and Yunyoung Nam (ynam@sch.ac.kr)

This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ICAN (ICT Challenge and Advanced Network of HRD) program (IITP-2020-0-01832) supervised by the IITP (Institute of Information & Communications Technology Planning & Evaluation) and the Soonchunhyang University Research Fund.

ABSTRACT This paper proposes a model-free control framework for the path planning of the rigid and soft robotic manipulator using an intelligent algorithm called Weighted Jacobian Rapidly-exploring Random Tree (WJRRT). The optimization approach is used to model the path planning problem, which is independent of the robotic model, and then used the WJRRT algorithm to solve it. WJRRT algorithm not only explores the cartesian space for the end-effector of the robotic manipulator randomly but also directs it towards the goal-position when required. It is robust enough to tackle the uncertainties in the manipulator and make the computation of path planning more efficient. WJRRT assigned a fitness value to each node of the tree. Based on the fitness values algorithm computes the final path, which is a trade-off between efficiency and safety of the path. The simulation results of two, three, and seven degrees of freedom (DOF) robotic manipulators are presented and compared with JT-RRT, Bi-RRT, and TB-RRT algorithms. Experimental results are verified using a soft manipulator made from flexible materials, i.e., polypropylene and polychloroprene. Their flexible structure makes their control complex and creates uncertainties in the model. The simulation and experimental results demonstrate that WJRRT can efficiently and accurately control the motion of manipulators.

INDEX TERMS Rapidly-exploring Random Tree, robotic manipulator, redundant manipulator, robust path planning, soft robotics.

I. INTRODUCTION

Soft robotic manipulators draw inspiration from animals like, arthropods, starfish, and snakes and attracts immense attention from researchers and engineers [1]–[4]. However, a soft robotic manipulator's path planning has always been a challenging and intricate task for the researchers. They deal with the high dimensional complex state-space, noisy signals from the surroundings, and other stringent constraints because of the highly flexible structure of soft robot [5]–[7]. Unlike rigid mechanical robots, it is impossible to model forward and inverse kinematics for the soft robot, due to their infinite degrees of freedom [8]–[10]. Furthermore, as the soft robotic manipulator's joints increase the complexity of the system four-fold [11]–[13]. For such complicated systems, researchers have produced some insightful work that allows the robotic system to navigate through search space and reach the goal position [14]–[16].

Researchers found it immensely challenging to generate a simple controlled motion in a soft robotic manipulator, whereas path planning is highly intricate. Deimel *et al.* [17] used an open-loop control for the three-finger soft robotic manipulator is used to reach out for an object and grasp it. The controlled pressurized air injected in pneumatic actuators in a scripted manner to inflate and deflate the fingers of the soft robotic manipulator. Likewise, Ilievski *et al.* [18] employed another open-loop motion control for a starfish-shaped soft robotic manipulator made of silicone chambers used to grasp

The associate editor coordinating the review of this manuscript and approving it for publication was Xujie Li.

delicate objects like an egg. Stokes *et al.* [19] mounted a soft robotic manipulator on a mobile robot to pick and place objects; it used Electro-Pneumatic Control (EPS) consisted of eight microprocessors to inflate and deflate the chambers. Brown *et al.* [20] further enhanced the controllability by increasing the inflatable joints. Likewise, for medical use, Ikuta *et al.* [21] have developed a slender tube to move through the blood vessel under a controlled pressure control system. Furthermore, Calisti *et al.* [22] used another open-loop framework to control the octopus Vulgaris arm to grasp objects. All these control algorithms exhibit limitations in performing a simple motion and require delicate fabrication, high stiffness, and tensile strength.

The path planning of the soft robotic manipulator is several folds complicated and complex task. As for the same end-effector position in cartesian-space, the robot can have several configurations in joint-space because of its infinite degrees of freedom [23], as shown in FIGURE 1. Researchers come around different techniques to model the path planning framework for the soft robotic manipulator. Hahnel *et al.* [24] used the graphical models, and Khatib *et al.* [25] employed potential fields as classical techniques to model path planning, but as the integrity of robotic material compromises, the robotic parameters change, and the performance of these techniques deteriorate. Likewise, the sampling technique [26] and the grid-based algorithm [27] are computationally expensive, time-consuming, and become more exhaustive as the degrees of freedom (DOFs) increases. Furthermore, the key-point interpolation technique [28] faces difficulty in selecting the key point.

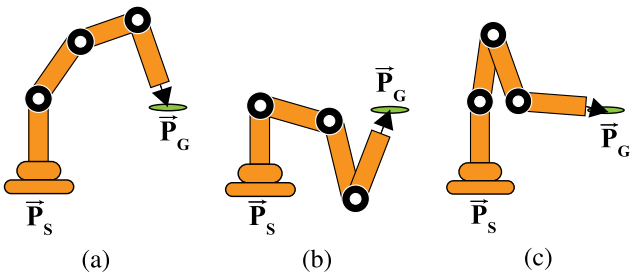

FIGURE 1. (a), (b), and (c) show that for the same cartesian coordinates P_G of the end-effector, the manipulator has three different joint-space configurations.

Advanced methods include learning from the demonstration (LFD) [29], a novel technique incorporated with learning algorithms to train the manipulator in path planning. LFD has its limitations of pre-determination of the path, which is time-consuming, and sometimes the manipulator can not trace the path [30]. Rapidly-exploring Random Tree (RRT) [31] is another known method for path planning. It works on a tree's principle with branches (nodes) rapidly growing and exploring the space. The tree initially includes a Start-node (initial-position) and a Goal-node (goal position). The start-node rapidly grows out in a random direction, fills the search space until it reaches the goal-node. Every node is a Parent-node for the subsequent node, and the subsequent node is known as a Child-node. In RRT, the generation of parent-child node continuous until the tree reaches the goal-node. The abstract idea of the RRT is manifested in FIGURE 2. The algorithm then traces back from the goal-node to the start-node to determine the path. Furthermore, the biased variant of the RRT algorithm [32] allows the rapid expansion of tree until it reaches the goal-node by giving more weight to those nodes and branches that are directed towards goal-node and thus increases the accuracy and efficacy. As it can navigate through the complex space, it has already been employed in different path planning schemes [32]–[37]. There are various state of the art variants of RRT, e.g., RRT*, B-RRT, RRT*-Smart, A*-RRT, and RRT^X. RRT* is an optimized version of RRT. It not only randomly explores the search space to the goal-node but also finds the optimal path on the addition of each node to the tree. Reference [38] is another variant of RRT*, which not only optimizes RRT but also plans the path in a dynamic environment. However, RRT* does not ensure the convergence even when $t \rightarrow \infty$. To come around this issue RRT*-Smart [39] was introduced, It not only accelerates the convergence rate but also reduced the convergence time. Likewise, B-RRT (Bidirectional RRT) [29] explores the search space from both ends, i.e., start-node and goal-node. The expansion of the tree from both directions further consolidates the effectiveness and robustness of RRT. Later, the optimized variant of B-RRT [40] was also proposed. A*-RRT [41] is a two-phase method, in the first phase, the algorithm searches in a lower dimension, and in the second phase it works as RRT* to search the path in higher-dimension. Furthermore, RRT^X, [42] is another extensions of RRT* for the path planning in a dynamic environment.

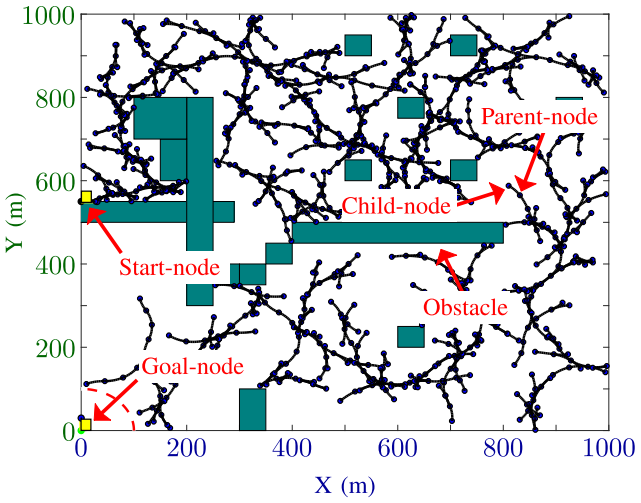

FIGURE 2. It shows the typical path planning technique of RRT algorithm. It starts from the Start-node and reaches the Goal-node while avoiding obstacles. It also gives an illustration of Parent-node and Child-node.

In this paper, a model-free control for the path planning of a soft robotic manipulator is presented. The algorithm includes

RRT integrated with Jacobian-transpose and weighted feasible paths known as Weighted Jacobian Rapidly-exploring Random Tree (WJRRT). The proposed algorithm is computationally efficient since it computes Jacobian-transpose instead of Jacobian-inverse, which is far more challenging. WJRRT controls the manipulator in forward kinematics so, it does not require the inverse kinematics model. Furthermore, weighted feasible paths are included to safely avoid the obstacles and overcome the system's uncertainties. RRT computes all feasible paths from parent-node to child-node with different weights. Based on the weights, the algorithm intelligently decides the path while considering the route's safety and efficiency. Unlike traditional techniques, our proposed algorithm WJRRT is robust enough to avoid the uncertainties in the system and surroundings, e.g., changes in system-model and the environment [43]–[48].

Furthermore, experimental validation of the algorithm are also included. Soft robotic manipulator are made of polypropylene tubes as links and polychloroprene as joints, then employed WJRRT to compute the path. The soft robotic manipulator successfully and robustly followed the path and completed the task. The highlights of the proposed method are as follow:

1) A model-free control framework for the path planning of a soft robotic manipulator includes obstacle avoidance.
2) Proposed algorithm is a robust variant of RRT integrated with Jacobian-transpose and weighted feasible paths to design a path planning framework for a soft robotic manipulator while avoiding obstacles.
3) The theoretical analysis shows that WJRRT is a stable and convergent algorithm.
4) The time and space complexity of WJRRT is polynomial in time.
5) The simulation results show that WJJRT is capable of controlling the redundant manipulator as well. Furthermore, WJRRT successfully accomplished the path-planning of a soft robotic manipulator.

The rest of the paper is implemented as follows. Section II includes the formulation of the optimization problem to solve the path planning of the soft robotic manipulator while avoiding obstacles. Section III includes the WJRRT algorithm, its theoretical analysis, and time-space complexity. The simulation results of two-arm, three-arm, and seven-arm redundant manipulators are discussed in Section IV. Section V includes the manufacturing and the path planning of a soft robotic manipulator. Lastly, Section VI concludes the paper with final remarks.

II. PROBLEM FORMULATION

In this section, two sub-optimization problems are formulated, i.e., path planning and obstacle avoidance, and then two sub-optmization problems are unified into a single objective function.

A. TRACKING CONTROL

Before path planning, the control framework of soft robotic manipulator needs to be understand. The mechanics of mechanical robots are applicable on soft robotic manipulator, if they have rigid links, but soft joints. There are two models to describe the motion of rigid manipulators, i.e., Forward-kinematics (FK) and Inverse-kinematics (IK). In FK, the input is provided in joint-space $\boldsymbol{\theta}$ and output is in cartesian-coordinates $\boldsymbol{X}$ of the end-effector. The mathematical model of FK is given as,

$$\boldsymbol{X} = F(\boldsymbol{\theta}), \tag{1}$$

where $\boldsymbol{\theta} \in \mathbf{R}^m$ and $\boldsymbol{X} \in \mathbf{R}^n$, m and n are total links and cartesian-coordinates respectively of the manipulator. Since the manipulator is in 3D space so $n = 3$. $F(.)$ is a non-linear transformation from joint-space to the cartesian-space. The classical method to determine the FK of a manipular is known as Denavit–Hartenberg [49]. However, the real-world robotic applications are based on IK, where the input to the manipulator is cartesian-coordinates of the end-effector, and output is the joint-space of the links. The mathematical model is given as,

$$\boldsymbol{\theta} = F^{-1}(\boldsymbol{X}), \tag{2}$$

where $F^{-1}(.)$ is again a non-linear function. As mentioned earlier, a manipulator can have multiple joint-space configurations that result in the same end-effector configuration, i.e., no closed-form solution, as shown in FIGURE 1. Therefore, the viable model is to optimize FK, as IK is immensely challenging to solve. For the path planning, It is required to minimize the error between the current position of the end-effector $\boldsymbol{X_c}$ and the goal position $\boldsymbol{X_G}$. The input to the manipulator is joints angle and the output will be its end-effector coordinates, i.e., $\boldsymbol{X} \in \mathbf{R}^3$. The formulation of the objective function is as follow,

$$\min_{\boldsymbol{\theta_c}} H_t(\boldsymbol{X_G}, \boldsymbol{\theta_c}), \tag{3}$$

where $H_t(.)$ is an optimization function to minimize the error between the current position of the end-effector and the goal position. θ_c is the current joint-configuration of the manipulator. it can be expanded (3) as,

$$\min_{\boldsymbol{\theta_c}} H_t(\boldsymbol{X_G}, \boldsymbol{\theta_c})) = ||\boldsymbol{X_G} - F(\boldsymbol{\theta_c})||_2^2. \tag{4}$$

From (1) it can be seen that $\boldsymbol{X_c} = F(\boldsymbol{\theta_c})$, so it can replace $F(\boldsymbol{\theta_c})$ with $\boldsymbol{X_c}$ in (4), which is given as,

$$\min_{\boldsymbol{\theta_c}} H_t(\boldsymbol{X_G}, \boldsymbol{\theta_c}) = ||\boldsymbol{X_G} - \boldsymbol{X_c}||_2^2. \tag{5}$$

Thus, the optimization problem for path planning of the soft robotic manipulator is formulated, from the start position of the end-effector $\boldsymbol{X_S}$ to the goal position $\boldsymbol{X_G}$. Now the objective of the WJRRT algorithm is to input angles θ_c to the joints of the manipulator. The FK will compute the end-effector configuration $\boldsymbol{X_c}$ of the manipulator based on θ_c. The (5) will estimate the error between the current configuration $\boldsymbol{X_c}$, and

the goal configuration $\boldsymbol{X_G}$. The ultimate objective of WJRRT is to minimize this error so, that the end-effector reaches the goal position, i.e., $\boldsymbol{X_c} \approx \boldsymbol{X_G}$.

B. OBSTACLE AVOIDANCE

The optimization problem (3) does not include the obstacle avoidance of soft robotic manipulator. To come around the problem, maximizing the minimum distance technique is used. It means that the links of the manipulator will maintain a certain distance from the obstacles *Obs*. The formulation is given as follows,

$$\min_{\boldsymbol{\theta}} H_o(Obs, \boldsymbol{\theta_c}), \tag{6}$$

where $H_o(.)$ is another objective function to account for obstacle avoidance. The function depends on the obstacles *Obs* in the environment and the current configuration $\boldsymbol{\theta_c}$ of the manipulator. On expanding the formulation becomes,

$$H_o(Obs, \boldsymbol{\theta_c}) = \frac{1}{\min\limits_{l \in \{1,2,3...m\}} dist_l(Obs, {}^{l}\boldsymbol{\theta_c})}, \tag{7}$$

where l represents the l^{th} link of manipulator. The $dist_l(.)$ is a non-linear function that calculates the distance between the obstacle and the l^{th} link. For the implementation of $dist_l(.)$, a popular technique known as Gilbert–Johnson–Keerthi (GJK) [50] algorithm is used. It takes two 3D bodies as an input and first computes their geometry, then calculates the distance between their vertices, and finally finds the shortest distance between two vertices, i.e., obstacle and the link of the manipulator. The formulation of $dist_l(.)$ is given as,

$$dist_l(Obs, {}^{l}\theta_c) = GJK(obs, R({}^{l}\theta_c)) \;\; l \in \{1, 2 \ldots m\}, \tag{8}$$

where $R(.)$ is a non-linear function to compute the 3D structure of the manipulator. The optimization problem for obstacle avoidance is completed. Next, the unification of (3) and (6) will be introduce.

C. UNIFICATION OF THE OBJECTIVE FUNCTIONS

The problem formulation is completed in two parts, i.e., path planning and obstacle avoidance. Here it is worth mentioning that the mechanical and joint angle limitations for soft manipulators are not required to be considered because of their flexible nature. The soft robotic manipulator can endure the high fluctuations in joints because of its flexible nature. Now combine the sub-objective functions (3) and (6), which is given as,

$$H = H_t(\boldsymbol{X_G}, \boldsymbol{\theta_c}) + \lambda H_o(Obs, \boldsymbol{\theta_c}) \tag{9}$$

$$H = \min_{\boldsymbol{\theta_c}} ||\boldsymbol{X_G} - F(\boldsymbol{\theta_c})||_2^2 + \frac{\lambda}{\min\limits_{l \in \{1,2,3...m\}} dist_l(Obs, {}^{l}\boldsymbol{\theta_c})} \;\; l \in \{1, 2 \ldots m\}, \tag{10}$$

where $\lambda \in \mathbb{R}$ is weight-operator. It decides the trade-off between the path tracking and the obstacle avoidance. From (10) it can be seen that the final objective function consists of two sub optimization problems, i.e., tracking control and obstacle avoidance. The goal is to minimize (10), which is 1) Decrease the error between the end-effector position $\boldsymbol{X_c}$ and the reference trajectory $\boldsymbol{X_G}$ and 2) Maximize the minimum distance between the obstacle and the manipulator.

III. WEIGHTED JACOBIAN RAPIDLY-EXPLORING RANDOM TREE ALGORITHM

In this section, the formulation of WJRRT algorithm will be discussed.

A. PATH PLANNING USING RAPIDLY-EXPLORING RANDOM TREE (RRT)

The path planning of robotic manipulator is described as the navigation of the end-effector from the start-coordinates $\boldsymbol{X_S}$ to the goal-coordinates $\boldsymbol{X_G}$ while avoiding the obstacles. RRT is a valuable tool to compute complex trajectories efficiently. RRT works in both configurations, i.e., cartesian-configuration and joint-configuration. In our proposed method, RRT will search all the viable joint-configurations in search space from initial joint-configuration $\boldsymbol{\theta_S}$ to goal joint-configuration $\boldsymbol{\theta_G}$. The general framework of RRT is shown in FIGURE 3. The FIGURE 3 (a) shows a single node generation. Let us say a node $\boldsymbol{N_0}$ and a random node $\boldsymbol{N_r}$ are created in the search space. $\boldsymbol{N_0}$ should extend towards $\boldsymbol{N_r}$, but there is no definite distance between them, so RRT generates $\boldsymbol{N_1}$ between $\boldsymbol{N_0}$ and $\boldsymbol{N_r}$ at a distance d and bridges $\boldsymbol{N_0}$ with $\boldsymbol{N_1}$. There are total $k \in \mathbf{N}$ nodes in search-space and the k–th node is a Goal Node, i.e., $\boldsymbol{N_k}$ Likewise, the FIGURE 3(b) shows that generated node of $\boldsymbol{N_{k\text{-}q}}$ is connected to $\boldsymbol{N_4}$ instead of $\boldsymbol{N_2}$ because $d_1 > d_2$. It means that every generated node will connect with the nearest node in the search space. The generation of nodes will continue as shown in FIGURE 3(c) until reaches the goal-node. At the end, RRT will trace back the path from goal-node $\boldsymbol{N_k}$ to initial-node $\boldsymbol{N_0}$. For further elaboration the pseudocode of RRT is shown in Algorithm 1.

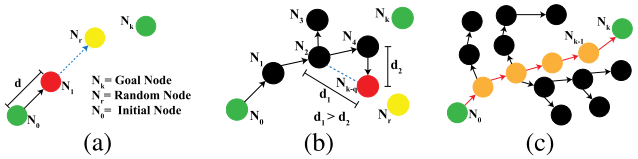

FIGURE 3. (a) shows how RRT starts building the tree, (b) shows that how $\boldsymbol{N_{k-q}}$ (node) connect with the nearest node $\boldsymbol{N_4}$, and (c) gives the idea of how finally RRT track its path from goal-node $\boldsymbol{N_k}$ back to initial-node $\boldsymbol{N_0}$.

In Section II, It is discussed that the computation of a closed-form IK solution is challenging, especially when the number of links of a manipulator increase. The different numerical approximations were utilized to come around this problem, but those approximations were failed to produce promising results. To tackle the issue, Bertram *et al.* [37] made an extension to RRT, and instead of extending tree in joint-space configuration, extend it in

Algorithm 1 RRT Algorithm

RRT(N_0, N_k)
  T.add(N_0) % Add node to the list
  $N_{new} \leftarrow N_0$
  While(Distance(N_{new}, N_k)> d_x)
    N_r = Random_node()
    $N_{nearest}$ = T.Nearest_node(N_r)
    N_{new} = extend($N_{nearest}$, N_r)
    if (N_{new}! = Null)
      N_{new}.set_parent($N_{nearest}$)
      T.add(N_{new}) % Add new node to the list
    end if
  Resulting_Path ← T.Trace_Back(N_{new})
  return Resulting_Path

workspace-configuration (Cartesian-coordinates). The proposed approach has some advantages:

1) It does not require inverse-kinematics (IK) to solve the path planning problem.
2) Improved method generates only feasible and admissible configurations of the manipulator.
3) The convergence rate of RRT boosts as the algorithm works in workspace configuration.

B. TRANSPOSE JACOBIAN REPLACED INVERSE JACOBIAN

Our proposed approach shares the motivation of Bertram *et al.* [37], but instead of extending a tree randomly, it directs the robotic arm to move towards the goal-node if subsequent configurations are feasible. Let us say a robotic arm has a joint-configuration of $\boldsymbol{\theta}$ with the end-effector position $\boldsymbol{X} \in \mathbf{R}^3$. The extension of the manipulator from $\boldsymbol{\theta} \rightarrow \boldsymbol{X}$ is a non-linear mapping, which is complex and computationally expensive. However, there exists a linear mapping between $\dot{\boldsymbol{\theta}}$ and $\dot{\boldsymbol{X}}$ through Jacobian J is a linear mapping. It is given as,

$$J\dot{\boldsymbol{\theta}} = \dot{\boldsymbol{X}}. \tag{11}$$

Here the assumption includes obstacles *Obs* in the environment as well. The $\dot{\boldsymbol{\theta}}$ is given as,

$$\dot{\boldsymbol{\theta}} = \rho J^{-1} H, \tag{12}$$

where $\rho > 0$ determines the step-size, H is the objective function (9), and the goal is to minimize it. In the absence of obstacles, uncertainties in the system, or the environment, the simple controller will be enough to reach the goal position. However, the real world environment includes obstacles and the computation of J^{-1} on each iteration is a challenging task. There are different methods like coordinate descent techniques, function approximation, and inverse mapping to approximate the Jacobian Inverse [51]–[53], but all are computationally expensive and time-consuming. Another viable approach presented in [27], instead of computing inverse, computes the transpose of Jacobian. The computational time and cost in calculating Jacobian transpose is much less than Jacobian inverse. Based on [27] the (12) becomes,

$$\dot{\boldsymbol{\theta}} = \rho J^T H. \tag{13}$$

The transpose controller works the same way as an inverse controller. The rigorous proof of the concept is presented in [27], but the basic idea is as follows: the instantaneous motion of the end-effector is given as,

$$\dot{\boldsymbol{X}} = J\dot{\boldsymbol{\theta}} = J(\rho J^T H). \tag{14}$$

Multiple both sides with objective function H, (14) becomes,

$$H^T \dot{\boldsymbol{X}} = \rho H^T J J^T H \geq 1, \tag{15}$$

since it is always positive definite, so it shows that the manipulator always move towards the goal-node.

The transpose controller made a fundamental assumption that any directed joint velocity is achievable, so this assumption breaks in the presence of obstacles and uncertainties in the system or the environment. These challenges are incorporated by further exploiting Jacobian controller, which will help the soft robotic manipulator navigate around in search-space. The proposed idea includes two scenarios; random search with probability P_R, and goal-directed search, $P_G = 1 - P_R$. In the absence of the obstacles, the motion of the manipulator will be directed towards goal $\boldsymbol{X_G}$, but once the manipulator comes across the obstacles, the algorithm will compute all the valid configurations. Then it weighs them based on how challenging they are. Finally, it obtains a trade-off between the efficient and the safe path.

Algorithm 2 WJRRT Algorithm

initialize environment()
initialize robotic arm()
N_{new} = [] %Initialize Nodes
$P_G = 0.5$ %Threshold for goal directed extension
P = rand()
for < n iterations > **do**
  if $P < P_G$ **then**
    N_{new} = extend_Towards_Goal()
  else
    N_{new} = extend_Randomly()
  end if
  if $N_{new} \neq$ NULL **then**
    N = add_Node(N_{new})
  end if
end for

As mentioned earlier, the WJRRT algorithm is divided into two scenarios: Extend randomly, i.e., extend_Randomly() and Extend towards the goal, i.e., extend_Towards_Goal(). The addition of two scenarios made the path planning more efficient, robust, and less time-consuming. The reason is, if the tree grows out randomly based on random nodes, i.e., extend_Randomly(), then it will be time-consuming and less efficient in search of goal-node. However, extend_Towards_Goal() is incorporated, then the

algorithm purposely generates those nodes which moves the manipulator towards goal-node until it comes across an obstacle. And then, the extend_Randomly() function comes in handy, which will search the space randomly, and it will connect the tree with the configuration (node) that avoids the obstacle. The efficiency of WJJRT is further elaborated in simulation section. The basic framework of WJRRT is given in Algorithm 2. Here it is worth mentioning, $\boldsymbol{N_{new}}$ is a node which includes the configuration of the manipulator, i.e., $\boldsymbol{N_{new}} \rightarrow \{\boldsymbol{X_{new}}, \boldsymbol{\theta_{new}}\}$. In the framework of WJRRT, the key function is the extension of the tree towards the goal, i.e., extend_Towards_Goal(). To understand the function in more detail, the pseudocode is provided in Algorithm 3. Out of all the nodes so far generated in search space, WJRRT looks for the node close to goal, i.e., $\boldsymbol{N_c}$. To extend that node further in joint-space, the formulation of closest which is also the current joint-configuration $\boldsymbol{\theta_c}$ is given as,

$$\boldsymbol{\theta_c} = \boldsymbol{\theta_c} + \Delta\boldsymbol{\theta_c}, \tag{16}$$

where $\Delta\boldsymbol{\theta_c}$ is a small random change in $\boldsymbol{\theta_c}$. After calculating the Jacobain transpose at $\boldsymbol{\theta_c}$, WJRRT calculates the joint-configuration $\boldsymbol{\theta_{new}}$ for $\boldsymbol{N_{new}}$, which is given as,

$$\boldsymbol{\Delta X_c} = \text{delta\_X}(\boldsymbol{\theta_c}, \boldsymbol{\theta_G}) \tag{17}$$

$$\Delta\boldsymbol{\theta_c} = J^T \boldsymbol{\Delta X_c} \tag{18}$$

$$\boldsymbol{\theta_{new}} = \boldsymbol{\theta_c} + \Delta\boldsymbol{\theta_c}, \tag{19}$$

where $\boldsymbol{\Delta X_c}$ is a small change towards goal. Now, the next step is to check the obstacles in the path. WJRRT avoids obstacles by solving the optimization problem mentioned in (7). Obstacle avoidance is an intricate issue, especially for the manipulators because of two reasons: 1) The mechanical constraints of manipulators and 2) Uncertainty in the system and environment. Imagine there is an *obs* obstacle between $\boldsymbol{\theta_c}$ and the $\boldsymbol{\theta_{new}}$. Considering above mentioned issues, WJRRT will generate m different configurations from $\boldsymbol{\theta_c}$ towards $\boldsymbol{\theta_{new}}$, and the configurations avoid obstacle. All m trajectories will be assigned two weights, i.e., to choose the safest path from $\boldsymbol{\theta_c}$ to $\boldsymbol{\theta_{new}}$ and to make an efficient and smooth transition towards $\boldsymbol{\theta_{new}}$ (considering mechanical constraints). Two weights parameters, i.e., W_s and W_d are defined. Here, W_s deals with the obstacle and the manipulators' current configuration. It sums up the distances between the obstacles and the links of the manipulator. It is given as,

$$W_s = \sum_{i=1}^{m} dist(obs, \boldsymbol{\theta}[\boldsymbol{l_i}]_{\boldsymbol{new}}). \tag{20}$$

where $dist(.)$ is given in (8) and $W_s > C_1$, where $C_1 > 0$. The threshold of W_s is to maintain a reasonable distance between the obstacle and the manipulator. The value of C_1 is set through hit and trial. In an environment with several obstacles, it is better to keep its value high because higher W_s means the reasonable distance from the obstacle. Likewise, W_d is to control angular step-size based on admissible configurations, which is given as,

$$W_d = \sum_{i=1}^{m} dist(\boldsymbol{\theta}[\boldsymbol{l_i}]_{\boldsymbol{c}}, \boldsymbol{\theta}[\boldsymbol{l_i}]_{\boldsymbol{new}}), \tag{21}$$

where $W_d > C_2$ and $C_2 > 0$. The value of C_2 is also set through hit and trial. The trade-off between safe (far from obstacles) and efficient (far from current configuration) lies somewhere in between W_s and W_d. The function extend_Randomly() is similar to above except that the algorithm does not calculate Jacobian. WJRRT simply explore the configuration space randomly. Finally, the contributions of WJRRT algorithm are as follows:

- It is bias towards the direction of goal-node, which makes It fast and efficient.
- It tackles the uncertainties in soft robotic manipulator and make trajectory exploration more efficient.

Algorithm 3 extend_Towards_Goal()

```
initialize_zero(Count, Ws, Wd)
set_threshold(k, C1, C2)
H_best = ∞
N_c = closest_Node_to_Goal()
while N_c ≠ N_G do
  Δθ_c = k * rand(1, m)
  θ_c = θ_c + Δθ_c
  J^T = J_Tanspose(θ_c)
  ΔX_c = delta_X(θ_c, θ_G)
  Δθ_c = J^T ΔX_c
  θ_new = θ_c + Δθ_c
  for l = 1 : m do
    if H_o(Obs, θ[l]_new), as in (7) < 0.5 then
      Count = Count + 1
      W_s = W_s + dist(Obs, θ[l]_new), as in (4)
      W_d = W_d + dist(θ[l]_c, θ[l]_new)
    end if
  end for
  if Count < m then
    return N_new = NULL
  end if
  if W_s > C1 && W_d < C2 then
    Compute H, as given in 9
    if H < H_best then
      H_best = H
      return N_new
    end if
  end if
end while
```

C. THEORETICAL ANALYSIS

The section includes the theoretical analysis if WJRRT, stability, convergence, random and towards goal extension, time and space complexity.

1) WJRRT IS STABLE AND CONVERGENT

Theorem 1: The objective function "H" of Weighted Jacobian Rapidly-exploring Random Tree (WJRRT) will continuously decrease monotonically with iterations "n" thus makes it stable. It is given as,

$$H_1 > H_2 \quad n_1 < n_2. \tag{22}$$

Proof: See Lemma 1 of [54].

Theorem 2: The objective function "H" of Weighted Jacobian Rapidly-exploring Random Tree (WJRRT) will drive the system towards optimal solution N_G, as the iteration "n" approaches to infinity, makes WJRRT convergent. It is given as,

$$H \rightarrow \boldsymbol{N_G} \quad n \rightarrow \infty. \tag{23}$$

where $\boldsymbol{N_G}$ is the goal-node in path planning of soft robotic manipulator.

Proof: See Lemma 2 of [54].

2) EXTEND RANDOMLY VS. EXTEND TOWARDS GOAL

Here it is worth mentioning that the number of nodes towards the goal, i.e., the configurations that lead the manipulator towards the goal, are higher in extend_Towards_Goal() than extend_Randomly() function. Assume $P_G = 0.5$ and $P < P_G$, the algorithm will enter in extend_Towards_Goal(). As mentioned earlier, this function is biased towards the Goal-node because it generates the series of joint-configurations until it comes across the obstacle. After the generation of one admissible node, the Transpose Jacobian (within extend_Towards_Goal()) takes control and linearly moves the manipulator towards the goal and, in this process, all the generated nodes are goal nodes. Once it comes across the obstacle the subsequent node breaks the loop and returns the series of generated nodes, i.e., N_{new} to the main function. On the other hand, when $P > P_G$, the algorithm calls extend_Randomly() function, and it generates only a single node on each call. Despite the equal probability for the occurrence of both functions, extend_Towards_Goal() has the higher potential to generate more nodes. The two functions are elaborated in Algorithm 2 and Algorithm 3.

Finally, the selection of parameters, i.e., k, C_1, C_2, depends on the manipulator and the environment. Here, k is a step-size in configuration space, i.e., from θ_c towards θ_{new}. The smaller range of [0, 0.1] will ensure the smooth transition of manipulator from $\theta_c \rightarrow \theta_{new}$. For the values of C_1 and C_2, as mentioned earlier, in case of a complex environment with several obstacles keep the value of C_1 higher, e.g., ≥ 10, and C_2 lower, e.g., ≤ 1. In that case, the algorithm includes only those nodes that ensures the safe route for manipulator towards goal-node.

D. TIME AND SPACE COMPLEXITY

1) TIME COMPLEXITY

The WJRRT algorithm will last for n iterations, as shown in Algorithm 2. It includes two functions, extend_Towards_Goal() and extend_Randomly(). Primarily, extend_Towards_Goal() function contributes to the time-complexity of the algorithm. Let us say, each statement consumes a unit of time. The time-complexity of extend_Towards_Goal() is shown in Algorithm 3. Consider there are N nodes then the time complexity of closest_Node_to_Goal() will be n. The complexity of Jacobian depends on the number of joints in manipulator, which is given as m^2. The time complexity of obstacle check is linear because of the GJK algorithm [49], so the time-complexity of **for** is m. The rest of the statements take a unit of time each so they can ignored. The total complexity of the function becomes $O(m^3n)$. The overall time complexity of WJRRT becomes $O(m^3n^2)$. Since, $n^2 \gg m^3$, so m^3 can be eliminated and the time-complexity becomes $O(n^2)$, which is polynomial.

2) SPACE COMPLEXITY

The WJRRT algorithm will last for n iterations, as shown in Algorithm 2. Again, focus on extend_Towards_Goal() alone as it contributes to the space-complexity of the algorithm mainly, it is shown in Algorithm 3. Let us say, each node consumes unit space. The function closest_Node_to_Goal() maximum will consume n, the jacobian transpose will consume m^2. Assume the obstacles consume p space. The time-complexity of extend_Towards_Goal() is $O(pnm^2)$. The space-complexity of WJRRT becomes, $O(pn^2m^2)$. Since, $pn^2 \gg m^2$, so m^2 can be ignored to make the expression simple. The final space-complexity is $O(pn^2)$, which is polynomial.

IV. SIMULATION RESULTS

In this section, simulation results and the comparison of WJRRT with JT-RRT (Jacobian Transpose RRT), Bi-RRT (Bidirectional RRT) and TB-RRT (Tangent Bundle RRT) are presented. WJRRT is tested on three different manipulators, i.e., two-arm, three-arm, and seven-arm redundant manipulators.

A. TWO-ARM ROBOTIC MANIPULATOR

First, WJRRT is tested on the two-arm robotic manipulator and compared the results with the JT-RRT [55], Bi-RRT [56], and TB-RRT [57] algorithms. The algorithms were tested on the same PC with specifications: Intel-i7, 3.41GHz processor, and 16GB RAM. To end the simulation two cases are used, either a certain number of iterations or if end-effector of manipulator reaches within the threshold radius around goal position $\boldsymbol{X_G}$. In case of two-arms, the number of iterations are 500, and the threshold radius is $r = 0.001cm$. Safety parameters are: $C_1 = 4$ and $C_2 = 0.5$. The simulation ran for ten times and evaluated the performance of all four algorithms based on the following parameters:

- **Success:** The number of times algorithm succeed in accomplishing the task.
- **Total Time:** The time taken by the algorithm to accomplish single task.

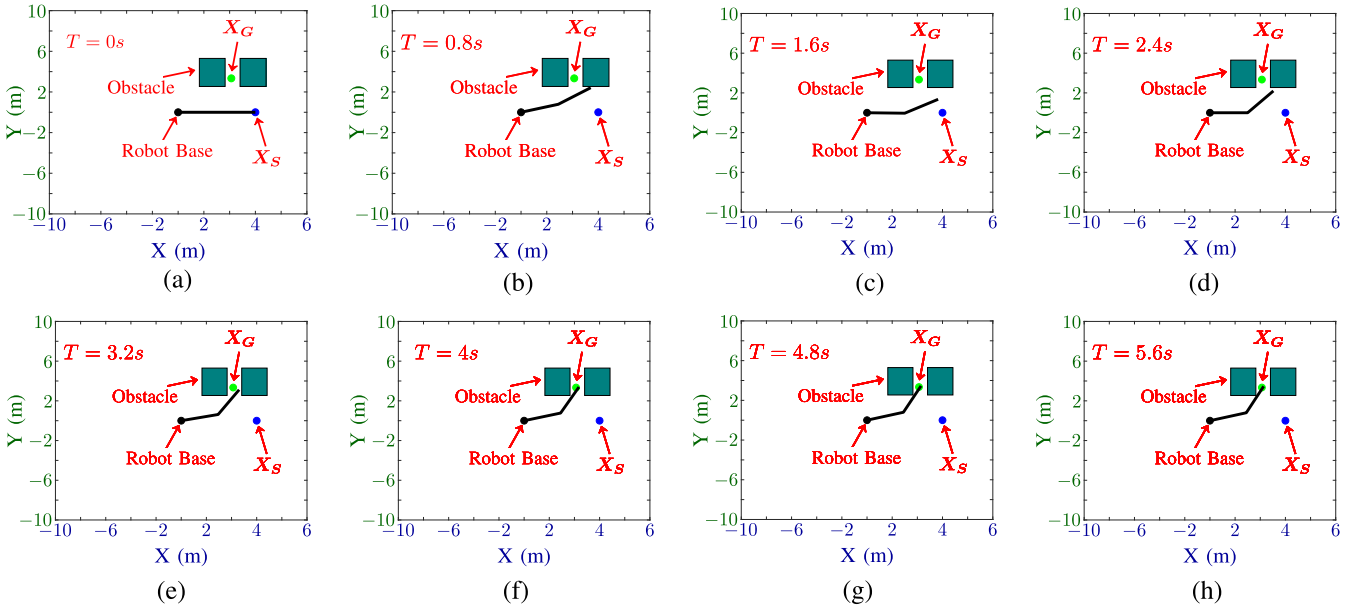

FIGURE 4. It is an implementation of the WJRRT algorithm on the two-arm manipulator. It shows the time-lapse of trajectory following by the manipulator starting from X_S to X_G, and the task completion time is 5.63 *sec*.

TABLE 1. Comparison between WJRRT, JT-RRT, Bi-RRT, and TB-RRT algorithms.

Approach	Arms	Success	Iter.	Total Time	Nodes	Track Nodes	Goal Nodes	Random Nodes	Obstacle Check	Objective Function (H)
JT-RRT [55]	2	8/10	500	7.63	376	43	23	20	121	-
Bi-RRT [56]	2	8/10	500	7.10	398	57	-	57	102	-
TB-RRT [57]	2	9/10	500	6.23	352	90	-	90	113	-
WJRRT	2	10/10	500	5.63	345	93	70	23	98	0.001
JT-RRT [55]	3	7/10	1000	10.43	753	181	138	43	234	-
Bi-RRT [56]	2	9/10	1000	11.02	749	190	-	190	274	-
TB-RRT [57]	2	10/10	1000	10.27	645	213	-	213	334	-
WJRRT	3	9/10	1000	9.43	674	209	163	46	359	0.005
JT-RRT [55]	7	6/10	2000	18.43	1290	932	503	429	1023	-
Bi-RRT [56]	2	7/10	2000	17.63	1323	1078	-	1078	1092	-
TB-RRT [57]	2	7/10	2000	12.23	1054	870	-	870	1287	-
WJRRT	7	8/10	2000	13.21	1583	1231	903	328	1340	0.002

- **Nodes:** The number of nodes generated by the algorithm during a single task.
- **Track Nodes:** Out of total nodes those which are included in path planning.
- **Goal Nodes:** The nodes in track nodes which are obtained from extend_Towards_Goal() function.
- **Random Nodes:** The nodes in track nodes which are obtained from extend_Randomly() function.
- **Obstacle Check:** The total number of times algorithm avoided obstacles.
- **Objective Function:** The objective function value which shows how close end-effector gets to the Goal-Node.

The first two parameters are self explanatory, but Nodes, Track Nodes, Track Nodes, and Random Nodes are related to each other. Nodes represent the percentage of exploration done by the algorithm in search space, more nodes mean more exploration. Track Nodes are derived from the Nodes, those nodes which directs the manipulator from Start-node towards Goal-node, more Track Nodes. Within Track Nodes there exist two types of nodes, i.e., extend_Towards_Goal() (Goal Nodes) and extend_Randomly() (Random Nodes). More nodes in Goal Nodes mean that manipulator's trajectory less abrupt changes because of Transpose Jacobian controller. All these parameters are given in TABLE 1. For the two-arm manipulator, the scenario includes obstacles. The goal is to plan a path from $\boldsymbol{X_S}$ to $\boldsymbol{X_G}$, using the WJRRT framework while avoiding obstacles. The time-lapse of the simulation is given in FIGURE 4(a)-(h). In this case, $P_G = 0.5$. There are no angular constraints except for self-collision.

The results manifested in TABLE 1 shows that the success rate and simulation time of WJRRT outsmarts the other algorithms. Likewise, more nodes generated under extend_Towards_Goal() function makes WJRRT

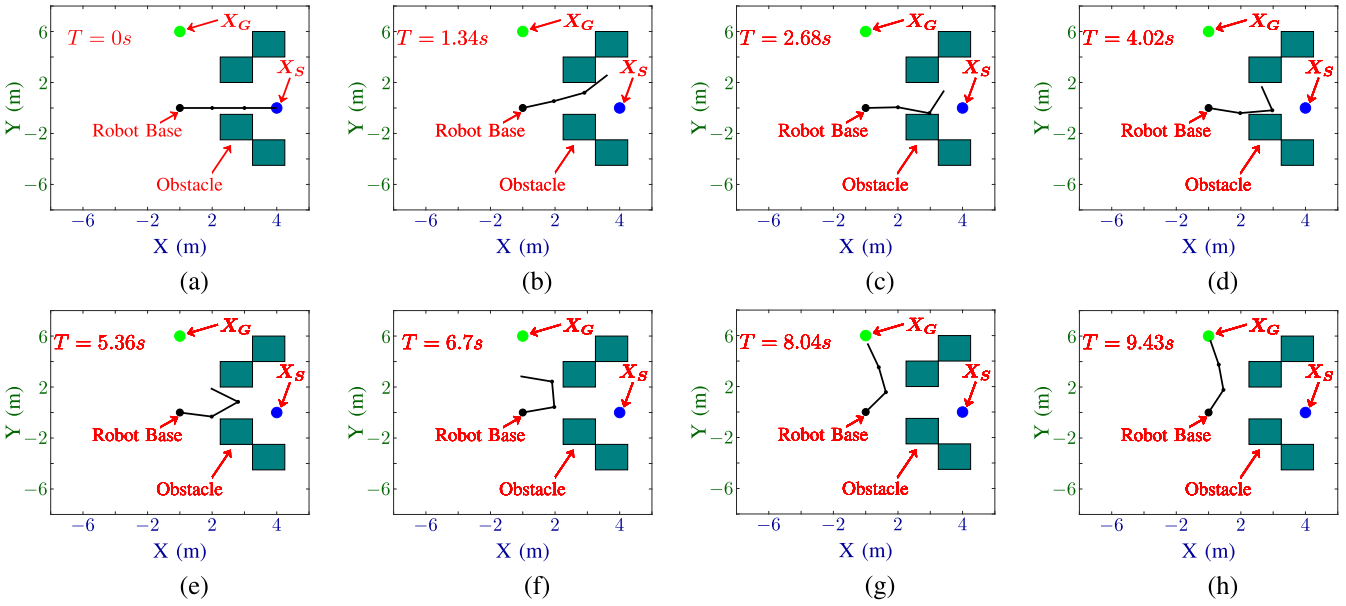

FIGURE 5. It is an implementation of the WJRRT algorithm on the three-arm manipulator. It shows the time-lapse of trajectory following by the manipulator starting from X_S to X_G, and the task completion time is 9.43 *sec*.

more efficient. Lastly, the objective function value of WJRRT converged to the optimal solution which is 0.001.

B. THREE-ARM ROBOTIC MANIPULATOR

WJRRT algorithm is applied on the three-arm manipulator and compared it with other algorithms, the results are shown in FIGURE 5 and TABLE 1. The kinematics of a three-arm manipulator is quite complicated as it has no closed-form solution. Furthermore, the complexity increases when there are obstacles in the environment. The number of iterations for the accomplishment of the task is 1000, and there are no angular constraints. Safety parameters are: $C_1 = 3$ and $C_2 = 0.7$.

The detailed results are shown in TABLE 1. Again, WJRRT outperformed other algorithms in the success rate and in simulation time. Our proposed algorithm-generated more nodes under extend_Towards_Goal() function. Lastly, WJRRT algorithm converges to the optimum solution of 0.005. The time-lapse of WJRRT generated path traced by a three-arm manipulator is shown in FIGURE 5(a)-(h).

C. SEVEN-ARM REDUNDANT MANIPULATOR

Finally, the algorithm is applied on seven-arm redundant manipulator and compared it with other algorithms, and the results are shown in FIGURE 6 and TABLE 1. Redundant manipulators are widely used in industries and the medical field because of the dextrous nature. The extra degree of freedom allows them to perform secondary tasks like obstacle avoidance along with primary tasks like tracking paths. The control of redundant rigid-manipulator is an intricate and challenging task and requires complex controlling frameworks to control them. The challenge increases several folds in the case of soft robotic manipulators because of the flexible nature. The path planning scenario for the redundant manipulator is shown in FIGURE 6. The task for the redundant manipulator is to follow the WJRRT generated path from $\boldsymbol{X_S}$ to $\boldsymbol{X_G}$. The objective is to plan a path in a narrow tube without colliding the walls. Since the manipulator is surrounded by more obstacles as compare to Two-arm and Three-arm robotic manipulators, so the safety parameters are: $C_1 = 10$ and to further ensure the safety the $C_2 = 0.01$ so that manipulator takes smaller step.

The results are shown in TABLE 1, and it shows that once again, WJRRT performed way better than other algorithms. However, in some aspects, WJRRT has a close contest with TB-RRT. TB-RRT has a higher success rate, but it is more time-consuming because the nodes are not biased towards goal-node. WJRRT algorithm converged to its optimal solution 0.002, the time-lapse of the simulation is shown in FIGURE 6(a)-(h).

V. SOFT ROBOTIC MANIPULATOR PLATFORM

In this section, the implementation of WJRRT is presented on the soft robotic prototype. The proposed design of soft robotic manipulator is hybrid in nature. It includes rigid links and soft joints made of polypropylene and polychloroprene, respectively. These materials are widely available in market and are used in sea for carrying cables which ensures their durability [58]. The soft-manipulator has a fuzzy behavior that makes its path planing a difficult task than a rigid robot. The weight or fitness functions in the WJRRT algorithm accommodate the fuzzy nature of the manipulator. W_s ensures the safest path, i.e., Obstacle avoidance, which can be seen from Obstacle Check mentioned in Table 1 where WJRRT mostly has more obstacles checks than the other

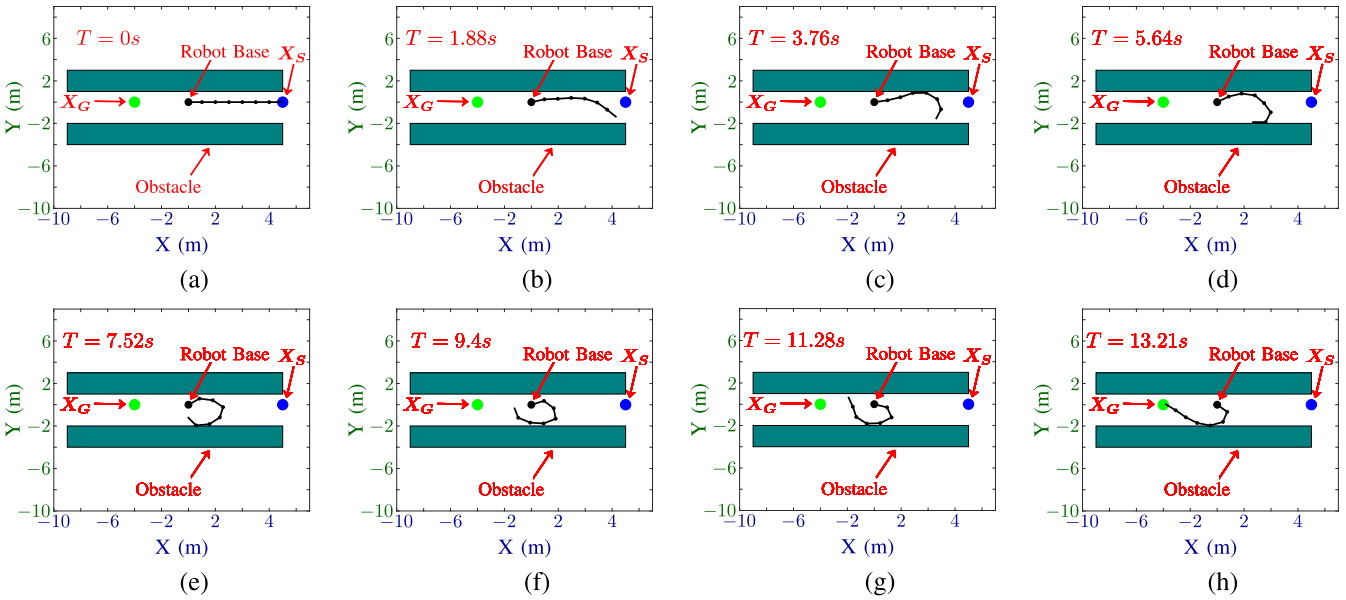

FIGURE 6. It is an implementation of the WJRRT algorithm on the seven-arm redundant manipulator. It shows the time-lapse of trajectory following by redundant manipulator starting from X_S to X_G, and the task completion time is 13.21 *sec*.

three algorithms. Likewise, W_d ensures that manipulator has a smooth transition from one joint space configuration to other. The simulation results in TABLE 1 shows that the Track Nodes of WJRRT are more in number than the rest especially in case of Seven-Arm Manipulator.

A. FABRICATION OF SOFT ROBOTIC MANIPULATOR

As mentioned earlier, our soft robotic manipulator is the combination of two materials, i.e., polypropylene tubes as links and polychloroprene as joints. Here it is worth mentioning that traditionally the fabrication of soft robotic manipulator includes silicone elastomer because of its durability and flexibility. However, it requires hours to develop into the designed shape, and small mistakes can force to start the fabrication process from the beginning. Silicone elastomer is replaced with polychloroprene, as it is cheap and easy to use.

The fabrication of the soft robotic manipulator is as follows. The polypropylene tubes are flexible, so take a tube of the length of two links, bend it from the middle, and cut a notch, as shown in FIGURE 7(a). Take tube size polychloroprene as it is soft and flexible like a ballon and insert it inside the polypropylene tube. Now, take an air pipe and insert it inside polychloroprene from one end and glue or tape both ends of the tube, as shown in FIGURE 7(a). To test the working, inject air inside the air-pipe using the pump, when the polychloroprene inflates the tube bends from the center, as shown in FIGURE 7(b)-(c).

The concept is further extended from two-arms soft robotic manipulator to the three-arms soft robotic manipulator. For three-arms, take a larger tube of polypropylene and cut three notches; each represents a joint, so with three notches, it has three links, i.e., three arms. Now insert three different polychloroprenes inside the tube to control three joints of the soft robotic manipulator.

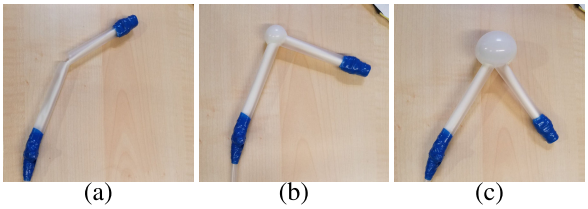

FIGURE 7. (a), (b), and (c) show deflated, semi-inflated, and fully inflated form of the soft-muscle, respectively.

B. SOFT ROBOTIC MANIPULATOR PROTOTYPE FOR WJRRT ALGORITHM

A platform of a soft robotic manipulator is build to test the performance of the WJRRT algorithm on real-world applications. The fabrication of the soft robotic manipulator is discussed in the previous section. For the pump mechanism, three syringes of $100mL$, were used to drive them. Three $12V$ linear-motors with motor driver $L298$, and Arduino-mega controller is used. The system is shown in FIGURE 8. This prototype is limited to support only the three-arm robotic system, but by increasing the number of syringes and motors, it can be further extended.

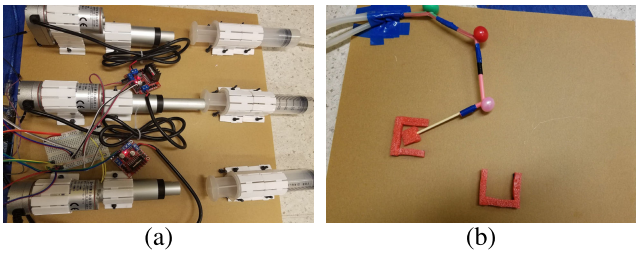

FIGURE 8. It is a prototype for the soft robotic manipulator manipulator, (a) shows the hardware to drive the manipulator, and (b) is the three-arm soft robotic manipulator.

For controlling mechanism, three syringes of $100mL$ is used and three linear motors because of the simple control. Motors are used to control the back and forth motion of the syringes. As the motor moves forward, it pushes the syringe along as a result an air blows out of the syringe. Likewise, when the motor moves backward, the syringe sucks the air inside. The openings of all three syringes are attached with one of the joints of the system, respectively. Now, when motors move forward, the syringes blow the air inside the soft robotic manipulator and bends it. Likewise, when motors move backward, the syringes suck the air causing a manipulator to achieve the initial position.

1) BENDING SENSOR TO MEASURE THE JOINTS ANGLE LIMIT

The mechanism mentioned above is for the free movement of all three joints, but to make it more controlled, the maximum angles that these joints can provide are measured. A bending sensor is attached with the tube. For three joints, three bending sensors are used. On calculation, the maximum bending angle from base to the end-effector joints turned out to be, 40 to 45 degrees, 52 to 55 degrees, and 55 to 60 degrees. And the minimum angle for all the joints was between 0 to 5 degrees.

2) PROPORTIONAL CONTROLLER TO CONTROL THE SOFT ROBOTIC MANIPULATOR

Now the platform of soft robotic manipulator is ready, and the next step is to control the bending of the manipulator. The problem with the polychloroprenes is that it inflates abruptly and deflates smoothly. To control the joint-configuration of the manipulator, a simple proportional controller (P) is used. The P controller formulation for the single joint soft robotic manipulator is given as,

$$E = K(\boldsymbol{\theta_r} - \boldsymbol{\theta_c}), \tag{24}$$

where E is the error between reference joint-configuration $\boldsymbol{\theta_r}$ and the current joint-configuration $\boldsymbol{\theta_c}$ of the manipulator. K is a proportional gain. The general trend between the pump air and the inflation and deflation of a soft robotic manipulator is shown in FIGURE 9(a). As mentioned earlier, inflation requires some extra force at the beginning; however, deflation happens smoothly. It can be seen in FIGURE 9(b), that how the P controller reduced the error between $\boldsymbol{\theta_r}$ and $\boldsymbol{\theta_c}$, and this is done on a single joint soft robotic manipulator.

C. EXPERIMENTAL RESULTS

In this section, the implementation of WJRRT on the soft robotic manipulator is elaborated. Experiment is performed on two-arm and three-arm soft robotic manipulator. The results were promising, and manipulators successfully and robustly completed the tasks.

1) TWO-ARM SOFT ROBOTIC MANIPULATOR

First, WJRRT is implemented on the two-arm soft robotic manipulator, the specifications of the experiment are as follows: the angle constraints for both the manipulator were 0 to 30 degrees for base-joint and 0 to 55 degrees for end-effector joint. The threshold or bearable error E for the P controller was 2 degrees.

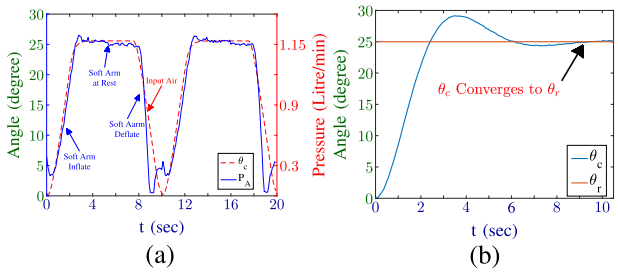

FIGURE 9. (a) shows the relation between the pump air pressure P_A and the bending angle of the soft robotic manipulator, (b) shows the error E deterioration as θ_c converges to θ_r using P controller.

An environment is generated in simulation and replicated that environment in real-world, during the simulation the number of iterations were 2000, and recorded all the nodes generated during the simulation that leads the manipulator from initial position $\boldsymbol{X_S}$ to the goal position $\boldsymbol{X_G}$. The total generated nodes were 1643, and the nodes for the final path were 367, out of them selected 20 nodes for experimentation including the start and the goal nodes, i.e., $\boldsymbol{N_S}$ and $\boldsymbol{N_G}$ Each node consists of angles for both joints of the manipulator. Although 20 out of 367 nodes made the motion of manipulator discretized, our purpose was only to test the implementation of WJRRT on the soft robotic manipulator. The completion of the track for a two-arm soft robotic manipulator using WJRRT is shown in FIGURE 10(a)-(f).

The working of the proportional controller becomes prominent in FIGURE 10(a)-(f), when the end-effector arm passes the required angle and proportional control brings it back to its required position by minimizing the error between its current position and its required position. But the working of the WJRRT algorithm was a success for the two-arm soft robotic manipulator. The next task is to integrate another arm and test the implementation of the WJRRT algorithm on the three-arm soft robotic manipulator.

2) THREE-ARM SOFT ROBOTIC MANIPULATOR

Now, WJRRT is implemented on the three-arm soft robotic manipulator. The angle constraints for the manipulators were 0 to 30 degrees for base joint, 0 to 55 degrees for middle-joint, and 0 to 60 degrees for the end-effector joint. The error $E = 4$ degrees, as it is more challenging to control the three-arm soft robotic manipulator.

WJRRT is implemented twice, first to move the manipulator from its initial position to inside the pit, as shown in FIGURE 10(g)-(l), and then from the pit to the goal position $\boldsymbol{X_G}$. The implementation technique is same, simulation is performed and collected some output nodes for the path. The P controller controlled the motion of the manipulator from the initial-node $\boldsymbol{N_S}$ to the final node $\boldsymbol{N_G}$. The experimental results are shown in FIGURE 10(g)-(l).

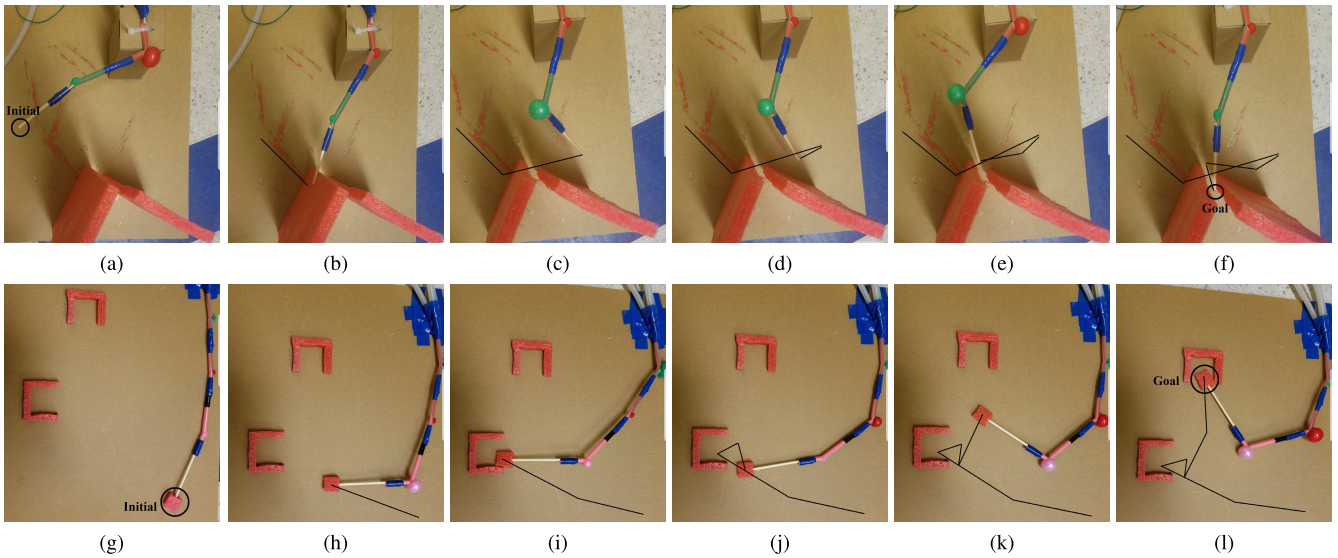

FIGURE 10. Experimental results for the implementation of WJRRT on two-arm and three-arm soft robotic manipulator.

The results obtained from both the experiments are promising, which shows that in the future it can be further extended to a soft robotic manipulator with more arms, and by incorporating integral (I), Differential (D) controller, the accuracy of path planning can be further improved.

VI. CONCLUSION

This paper used an optimization approach to present a model-free control for the path planning of the robotic manipulator. It uses an intelligent framework that integrates the Jacobian controller with Rapidly-exploring Random Tree known as Weighted Jacobian Rapidly-exploring Random Tree (WJRRT). No inverse kinematic model of the manipulator is required as WJRRT solves the path planning problem in forward kinematics. WJRRT is robust as it does not depend on the model of the manipulator. It explores the search space randomly as well as it directs the manipulator towards goal position when required. It also tackles the obstacles in the environment using an approach known as maximizing the minimum distance. It assigns a fitness value to each node and, based on the values algorithm, intelligently decides the final path, which is a trade-off between the path's efficiency and safety. The simulation and experimental results are presented on soft robotic manipulators of different degrees of freedom (DOF) and compared with JT-RRT, Bi-RRT, and TB-RRT algorithms. The results demonstrate that WJRRT can efficiently and accurately control the motion of manipulators.

REFERENCES

[1] D. Rus and M. T. Tolley, "Design, fabrication and control of soft robots," *Nature*, vol. 521, no. 7553, pp. 467–475, May 2015.

[2] H. Lipson, "Challenges and opportunities for design, simulation, and fabrication of soft robots," *Soft Robot.*, vol. 1, no. 1, pp. 21–27, Mar. 2014.

[3] M. Cianchetti, C. Laschi, A. Menciassi, and P. Dario, "Biomedical applications of soft robotics," *Nature Rev. Mater.*, vol. 3, no. 6, pp. 143–153, Jun. 2018.

[4] C. Yang, K. Huang, H. Cheng, Y. Li, and C.-Y. Su, "Haptic identification by ELM-controlled uncertain manipulator," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 47, no. 8, pp. 2398–2409, Aug. 2017.

[5] K.-J. Cho, J.-S. Koh, S. Kim, W.-S. Chu, Y. Hong, and S.-H. Ahn, "Review of manufacturing processes for soft biomimetic robots," *Int. J. Precis. Eng. Manuf.*, vol. 10, no. 3, pp. 171–181, Jul. 2009.

[6] Z. Li, C. Yang, and Y. Tang, "Decentralised adaptive fuzzy control of coordinated multiple mobile manipulators interacting with non-rigid environments," *IET Control Theory Appl.*, vol. 7, no. 3, pp. 397–410, Feb. 2013.

[7] S. Walker, O. Yirmibeşoğlu, U. Daalkhaijav, and Y. Mengüç, "Additive manufacturing of soft robots," in *Robotic Systems and Autonomous Platforms*. Amsterdam, The Netherlands: Elsevier, 2019, pp. 335–359.

[8] H. Wang, W. Chen, X. Yu, T. Deng, X. Wang, and R. Pfeifer, "Visual servo control of cable-driven soft robotic manipulator," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Nov. 2013, pp. 57–62.

[9] Q. Wang, S. Chen, and X. Luo, "An adaptive latent factor model via particle swarm optimization," *Neurocomputing*, vol. 369, pp. 176–184, Dec. 2019.

[10] A. D. Marchese, K. Komorowski, C. D. Onal, and D. Rus, "Design and control of a soft and continuously deformable 2D robotic manipulation system," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2014, pp. 2189–2196.

[11] M. A. Robertson and J. Paik, "New soft robots really suck: Vacuum-powered systems empower diverse capabilities," *Sci. Robot.*, vol. 2, no. 9, Aug. 2017, Art. no. eaan6357.

[12] C. Yang, Y. Jiang, J. Na, Z. Li, L. Cheng, and C.-Y. Su, "Finite-time convergence adaptive fuzzy control for dual-arm robot with unknown kinematics and dynamics," *IEEE Trans. Fuzzy Syst.*, vol. 27, no. 3, pp. 574–588, Mar. 2019.

[13] R. F. Shepherd, F. Ilievski, W. Choi, S. A. Morin, A. A. Stokes, A. D. Mazzeo, X. Chen, M. Wang, and G. M. Whitesides, "Multigait soft robot," *Proc. Nat. Acad. Sci. USA*, vol. 108, no. 51, pp. 20400–20403, Dec. 2011.

[14] M. K. Soltani, S. Khanmohammadi, F. Ghalichi, and F. Janabi-Sharifi, "A soft robotics nonlinear hybrid position/force control for tendon driven catheters," *Int. J. Control, Autom. Syst.*, vol. 15, no. 1, pp. 54–63, Feb. 2017.

[15] T. George Thuruthel, Y. Ansari, E. Falotico, and C. Laschi, "Control strategies for soft robotic manipulators: A survey," *Soft Robot.*, vol. 5, no. 2, pp. 149–163, Apr. 2018.

[16] D. Chen, S. Li, Q. Wu, and X. Luo, "New disturbance rejection constraint for redundant robot manipulators: An optimization perspective," *IEEE Trans. Ind. Informat.*, vol. 16, no. 4, pp. 2221–2232, Apr. 2020.

[17] R. Deimel and O. Brock, "A compliant hand based on a novel pneumatic actuator," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2013, pp. 2047–2053.

[18] F. Ilievski, A. D. Mazzeo, R. F. Shepherd, X. Chen, and G. M. Whitesides, "Soft robotics for chemists," *Angew. Chem. Int. Ed.*, vol. 50, no. 8, pp. 1890–1895, 2011.
[19] A. A. Stokes, R. F. Shepherd, S. A. Morin, F. Ilievski, and G. M. Whitesides, "A hybrid combining hard and soft robots," *Soft Robot.*, vol. 1, no. 1, pp. 70–74, Mar. 2014.
[20] E. Brown, N. Rodenberg, J. Amend, A. Mozeika, E. Steltz, M. R. Zakin, H. Lipson, and H. M. Jaeger, "Universal robotic gripper based on the jamming of granular material," *Proc. Nat. Acad. Sci. USA*, vol. 107, no. 44, pp. 18809–18814, Nov. 2010.
[21] K. Ikuta, H. Ichikawa, and K. Suzuki, "Safety-active catheter with multiple-segments driven by micro-hydraulic actuators," in *Proc. Int. Conf. Med. Image Comput. Comput.-Assist. Intervent.* Berlin, Germany: Springer, 2002, pp. 182–191.
[22] M. Calisti, A. Arienti, M. Elena Giannaccini, M. Follador, M. Giorelli, M. Cianchetti, B. Mazzolai, C. Laschi, and P. Dario, "Study and fabrication of bioinspired octopus arm mockups tested on a multipurpose platform," in *Proc. 3rd IEEE RAS EMBS Int. Conf. Biomed. Robot. Biomechatron.*, Sep. 2010, pp. 461–466.
[23] B. Bhandari, G.-Y. Lee, and S.-H. Ahn, "A review on IPMC material as actuators and sensors: Fabrications, characteristics and applications," *Int. J. Precis. Eng. Manuf.*, vol. 13, no. 1, pp. 141–163, Jan. 2012.
[24] D. Hahnel, W. Burgard, D. Fox, and S. Thrun, "An efficient fastSLAM algorithm for generating maps of large-scale cyclic environments from raw laser range measurements," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, vol. 1, Oct. 2003, pp. 206–211.
[25] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Autonomous Robot Vehicles*. New York, NY, USA: Springer, 1986, pp. 396–404.
[26] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, Jun. 2011.
[27] W. Wolovich and H. Elliott, "A computational technique for inverse kinematics," in *Proc. 23rd IEEE Conf. Decis. Control*, Dec. 1984, pp. 1359–1363.
[28] A. Vakanski, I. Mantegh, A. Irish, and F. Janabi-Sharifi, "Trajectory learning for robot programming by demonstration using hidden Markov model and dynamic time warping," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 42, no. 4, pp. 1039–1052, Aug. 2012.
[29] S. M. LaValle and J. J. Kuffner, Jr., "Randomized kinodynamic planning," *Int. J. Robot. Res.*, vol. 20, no. 5, pp. 378–400, May 2001.
[30] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, "Survey: Robot programming by demonstration," in *Handbook of Robotics*, vol. 59. Berlin, Germany: Springer, 2008.
[31] S. M. LaValle and J. J. Kuffner, Jr., "Rapidly-exploring random trees: Progress and prospects," in *Algorithmic Comput. Robot. New Directions*, vol. 5, no. 5, pp. 293–308, 2001.
[32] P. Dario and R. Chatila, *Robotics Research: The Eleventh International Symposium*, vol. 15. Berlin, Germany: Springer, 2005.
[33] J. Kim and J. P. Ostrowski, "Motion planning a aerial robot using rapidly-exploring random trees with dynamic constraints," in *Proc. IEEE Int. Conf. Robot. Autom.*, vol. 2, 2003, pp. 2200–2205.
[34] L. Jin, S. Li, H. M. La, and X. Luo, "Manipulability optimization of redundant manipulators using dynamic neural networks," *IEEE Trans. Ind. Electron.*, vol. 64, no. 6, pp. 4710–4720, Jun. 2017.
[35] G. Oriolo, M. Vendittelli, L. Freda, and G. Troso, "The SRT method: Randomized strategies for exploration," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 5, Jun. 2004, pp. 4688–4694.
[36] H. Wang and S. Kang, "Adaptive neural command filtered tracking control for flexible robotic manipulator with input dead-zone," *IEEE Access*, vol. 7, pp. 22675–22683, 2019.
[37] D. Bertram, J. J. Kuffner, Jr., R. Dillmann, and T. Asfour, "An integrated approach to inverse kinematics and path planning for redundant manipulators," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2006, pp. 1874–1879.
[38] O. Adiyatov and H. A. Varol, "A novel RRT*-based algorithm for motion planning in dynamic environments," in *Proc. IEEE Int. Conf. Mechatron. Autom. (ICMA)*, Aug. 2017, pp. 1416–1421.
[39] F. Islam, J. Nasir, U. Malik, Y. Ayaz, and O. Hasan, "RRT*-smart: Rapid convergence implementation of RRT* towards optimal solution," in *Proc. IEEE Int. Conf. Mechatronics Autom.*, Aug. 2012, pp. 1651–1656.
[40] M. Jordan and A. Perez, "Optimal bidirectional rapidly-exploring random trees," CSAIL, MIT, Cambridge, MA, USA, Tech. Rep. MIT-CSAIL-TR-2013-021, Aug. 2013.
[41] M. Brunner, B. Brüggemann, and D. Schulz, "Hierarchical rough terrain motion planning using an optimal sampling-based method," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2013, pp. 5539–5544.
[42] M. Otte and E. Frazzoli, "*RRT*^{x}: Real-time motion planning/replanning for environments with unpredictable obstacles," in *Algorithmic Foundations of Robotics XI*. Springer, 2015, pp. 461–478.
[43] N. A. Melchior and R. Simmons, "Particle RRT for path planning with uncertainty," in *Proc. IEEE Int. Conf. Robot. Autom.*, Apr. 2007, pp. 1617–1624.
[44] B. Luders, M. Kothari, and J. How, "Chance constrained RRT for probabilistic robustness to environmental uncertainty," in *Proc. AIAA Guid., Navigat., Control Conf.*, p. 8160, 2010.
[45] H. Wang, Y. Zou, P. X. Liu, and X. Liu, "Robust fuzzy adaptive funnel control of nonlinear systems with dynamic uncertainties," *Neurocomputing*, vol. 314, pp. 299–309, Nov. 2018.
[46] J. J. Craig, *Introduction to Robotics: Mechanics and Control, 3/E*. New Delhi, India: Pearson, 2009.
[47] C. A. Klein and C.-H. Huang, "Review of pseudoinverse control for use with kinematically redundant manipulators," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-13, no. 2, pp. 245–250, Mar. 1983.
[48] S. Li, H. Wang, and M. U. Rafique, "A novel recurrent neural network for manipulator control with improved noise tolerance," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 5, pp. 1908–1918, May 2018.
[49] C. Jin Ong and E. G. Gilbert, "The Gilbert-Johnson-Keerthi distance algorithm: A fast version for incremental motions," in *Proc. Int. Conf. Robot. Autom.*, vol. 2, 1997, pp. 1183–1189.
[50] P. Lindemann, "The Gilbert–Johnson–Keerthi distance algorithm," *Algorithms Media Informat.*, to be published.
[51] M. Šoch and R. Lórencz, "Solving inverse kinematics—A new approach to the extended jacobian technique," *Acta Polytechnica*, vol. 45, no. 2, 2005.
[52] K. Tchoń, J. Karpińska, and M. Janiak, "Approximation of jacobian inverse kinematics algorithms," *Int. J. Appl. Math. Comput. Sci.*, vol. 19, no. 4, pp. 519–531, Dec. 2009.
[53] H. Bass, E. H. Connell, and D. Wright, "The jacobian conjecture: Reduction of degree and formal expansion of the inverse," *Bull. Amer. Math. Soc.*, vol. 7, no. 2, pp. 287–330, 1982.
[54] Y. Zhang, S. Li, and B. Xu, "Convergence analysis of beetle antennae search algorithm and its applications," 2019, *arXiv:1904.02397*. [Online]. Available: http://arxiv.org/abs/1904.02397
[55] M. Vande Weghe, D. Ferguson, and S. S. Srinivasa, "Randomized path planning for redundant manipulators without inverse kinematics," in *Proc. 7th IEEE-RAS Int. Conf. Humanoid Robots*, Nov. 2007, pp. 477–482.
[56] X. Zang, W. Yu, L. Zhang, and S. Iqbal, "Path planning based on Bi-RRT algorithm for redundant manipulator," in *Proc. Int. Conf. Electr., Autom. Mech. Eng.*, 2015.
[57] B. Kim, T. T. Um, C. Suh, and F. C. Park, "Tangent bundle RRT: A randomized algorithm for constrained motion planning," *Robotica*, vol. 34, no. 1, pp. 202–225, Jan. 2016.
[58] P. Y. Le Gac, "Durability of polychloroprene rubber for marine application," Ph.D. dissertation, ENSAM, Paris, France, 2014.

AMEER TAMOOR KHAN received the B.S. degree in electrical engineering from the Pakistan Institute of Engineering and Applied Sciences, in 2017. He is currently pursuing the Ph.D. degree with the Department of Computing, The Hong Kong Polytechnic University. From 2017 to 2018, he was with the Cyber Physical Lab, The Hong Kong Polytechnic University, as a Research Assistant. His research interests include control system, optimization, and robotics path planning.

SHUAI LI (Senior Member, IEEE) received the B.E. degree in precision mechanical engineering from the Hefei University of Technology, Hefei, China, in 2005, the M.E. degree in automatic control engineering from the University of Science and Technology of China, Hefei, in 2008, and the Ph.D. degree in electrical and computer engineering from the Stevens Institute of Technology, Hoboken, NJ, USA, in 2014. He is currently a Professor, leading the Robotic Team, working on neural networks, multiple robot coordination, and distributed control. He is the General Co-Chair of 2018 International Conference on Advanced Robotics and Intelligent Control and the Founding Editor-in-Chief of the *International Journal of Robotics and Control*.

SEIFEDINE KADRY (Senior Member, IEEE) received the bachelor's degree in applied mathematics from Lebanese University, in 1999, the M.S. degree in computation from Reims University, France, and EPFL, Lausanne, in 2002, the Ph.D. degree from Blaise Pascal University, France, in 2007, and the H.D.R. degree in engineering science from Rouen University, in 2017. He is currently working as an Associate Professor with Beirut Arab University, Lebanon. His current research interests include education using technology, smart cities, system prognostics, stochastic systems, and probability and reliability analysis. He is a Fellow of IET, ACSIT, and ABET (Program Evaluator).

YUNYOUNG NAM (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer engineering from Ajou University, South Korea, in 2001, 2003, and 2007, respectively. He was a Senior Researcher with the Center of Excellence in Ubiquitous System, Stony Brook University, Stony Brook, NY, USA, from 2007 to 2010, where he was a Postdoctoral Researcher, from 2009 to 2013. He was a Research Professor with Ajou University, from 2010 to 2011. He was a Postdoctoral Fellow with the Worcester Polytechnic Institute, Worcester, MA, USA, from 2013 to 2014. He has been the Director of the ICT Convergence Rehabilitation Engineering Research Center, Soonchunhyang University, since 2017, where he is currently an Assistant Professor with the Department of Computer Science and Engineering. His research interests include multimedia database, ubiquitous computing, image processing, pattern recognition, context-awareness, conflict resolution, wearable computing, intelligent video surveillance, cloud computing, biomedical signal processing, rehabilitation, and healthcare systems.