ASTES

Advances in Science, Technology and Engineering Systems Journal
Vol. 4, No. 6, 386-396 (2019)
www.astesj.com
Special Issue on Multidisciplinary Sciences and Engineering

ASTES Journal
ISSN: 2415-6698

Novel Cost Function based Motion-planning Method for Robotic Manipulators

Dániel Szabó*, Emese Gincsainé Szádeczky-Kardoss

Department of Control Engineering and Information Technology, Budapest University of Technology and Economics, H-1117, Hungary

ARTICLE INFO

Article history:
Received: 09 October, 2019
Accepted: 01 December, 2019
Online: 30 December, 2019

Keywords:
robotic manipulator
motion-planning
Rapidly-exploring Random Tree
Fuzzy-approximation

ABSTRACT

In this paper an offline motion-planning algorithm is presented for robotic manipulators. In this method to solve the path-planning task, the Transition-based Rapidly-exploring Random Tree (T-RRT) algorithm was applied, that requires a cost-function over the search space. The goal of this cost-function is to keep distance between the actual position and configurations that cause collisions. The paper presents two possible cost function generation methods. The first one is based on multidimensional Gauss functions and the second solution uses fuzzy function-approximation to determine the cost all over the configuration space. At last, the trajectory generator method will be introduced that calculates the desired shape of the function of joint variables and their derivatives. The algorithm is universal, the only restriction is that the manipulator and the environment are modelled by their bounding polyhedra. To demonstrate the presented approach, simulation were performed in MATLAB Simulink environment using the Mitsubishi RV-2F-Q robotic manipulator.

1 Introduction

Nowadays, automation is playing an increasingly important role in our daily lives, therefore is a growing need for the autonomous motion-planning of robotic manipulators.

Two types of path-planning methods can be defined, the offline and the online methods. In case of offline path-planning method, the calculation time does not mean a bottleneck, so more time can be spent on improving the found path. For example, in [1] an offline path-planning method is introduced, that can find an optimal collision-free path, where the optimality is based on the energy usage of the manipulator. These kind of methods can ensure the collision-freeness only with static obstacles or obstacles with known motions.

Online path-planning methods are used to avoid collisions with dynamic obstacles having unpredictable movements. These algorithms can be used in human-robot collaboration systems for example [2]. The environment of the robot can be observed with a depth camera. Besides, the image processing can be calculated with GPU, so the real-time calculation can be ensured [3]. In this paper, an offline path planning method will be described.

In our case, the minimization of cost-function is needed during the motion-planning algorithm, while avoiding configurations leading to collisions. There are numerous solutions in robotic applications to this problem.

Many of these are founded on the classical grid-based methods like A* or D* which can be used to find an optimal path limited by the resolution of the grid [4]. The main disadvantage of these algorithms is that the computation time grows exponentially with the number of dimensions. This effect is called as the "Curse of dimensionality".

Another type of path-planning methods are the sampling-based algorithms like the Rapidly-exploring Random Tree (RRT). These algorithms can be used effectively with higher number of dimensions as well, but the optimality of the solution is not guaranteed [5].

These problems can be solved by the Transition-based RRT (T-RRT) algorithm, that incorporates the advantages of the grid-based and the sampling-based methods [6]. Thus, it can be used to find a path that minimizes a cost-function defined in the high dimensional search space.

Heuristics can be used to determine the meaning of cost. For example, in case of a human-collaboration, to ensure the safe operation and the security, the cost has to be high near by the person [7].

Another method could be that, when the dynamics of the system is used to minimize the time or the performed work during the motion.

In this paper, the goal is to maximize the distance between the path of the manipulator and the configurations that cause collisions with the help of defined cost function. In this case the result of the offline path-planning algorithm can be used by a reactive mo-

*Corresponding Author: Dániel Szabó, szabo.daniel0604@gmail.com

tion planning method, that will have more space to avoid dynamic obstacles.

The paper is organized as follows. Section 2 introduces the base of the kinematics of robotic manipulators. In Section 3, the T-RRT algorithm is presented, that was used for path-planning in this work. Then the collision-detection method is described, based on investigation of the feasibility of inequalities of bounding polyhedra, that are describing the manipulator and the obstacles (Section 4). Then, Section 5 describes two cost function approximation algorithms with Gauss-functions and with fuzzy function-approximation. The latter is more detailed. Subsection 5.5 describes the method that is used in this work for cost function evaluation. The trajectory generation method is explained in Section 6. The simulation results are presented in Section 7. Finally, Section 8 contains conclusions and possible developments in the future.

2 Robotic manipulators

Robotic manipulators are usually modeled as open kinematic chains. A manipulator contains links, which are connected by joints. Two basic types of joints exist: Revolute joints are used to apply relative rotation, and prismatic joints allow linear motion between two adjacent links. The relative movement between adjacent links is given by the joint variables q_i. In case of revolute joints, q_i gives the angular displacement, while for prismatic joints, q_i equals the amount of linear motion. The actual configuration q of the robot is defined by the vector containing actual q_i values of all joints.

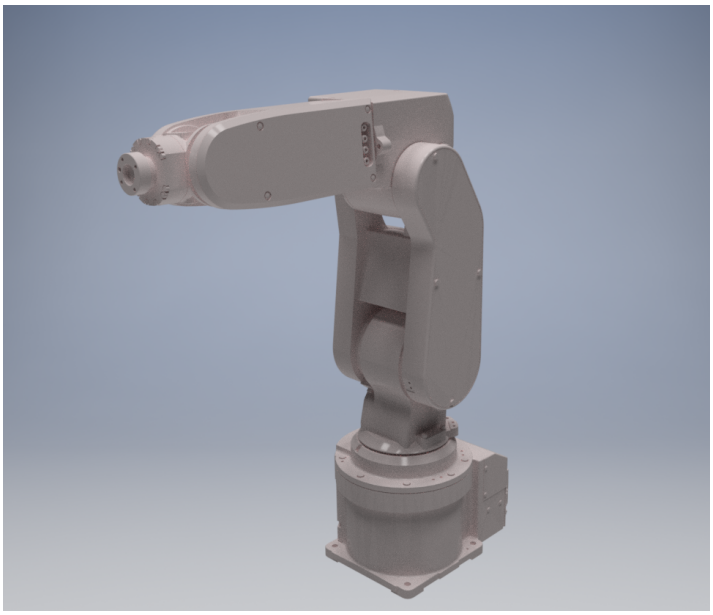

Figure 1: Mitsubishi RV-2F-Q manipulator (used in the simulations)

An end effector is usually attached to the last segment of the manipulator. To determine the position and the orientation of the end effector in the workspace of the robot, the equations of forward kinematics can be used. A homogeneous transformation matrix belongs to each robotic joints. The position and orientation of the coordinate frame attached to the end effector (relative to the base frame) can be determined by multiplying these matrices.

The main parameters of a robotic manipulator can be defined by the Denavit-Hartenberg parameters [8]. Homogeneous transformation matrices can be calculated from these parameters using actual joint variables.

The task of the path-planning is to find a path, which moves the robot to a desired goal configuration. (This goal configuration is typically given by the desired orientation and position of the end-effector at the end of the motion.) The path-planning problem can be defined in the Euclidean space, but usually, the planning is performed in the configuration space $\mathcal{C}$ of robot. The configuration space is spanned by the admissible configuration vectors (q), i.e. joint variables (q_i) are used as coordinates. For example the configuration space of the Mitsubishi RV-2F-Q manipulator with six degrees of freedom is a six-dimensional space. This robot has six revolute joints. Its 3D model is depicted in Fig. 1. The simulations presented in Section 7 were performed using this manipulator.

3 The Path-planning Algorithm

The traditional RRT algorithm is a stochastic sampling path-planning algorithm. In this method a search tree is grown from the initial q_{init} point to the q_{goal} goal configuration. In a given iteration, a randomly selected configuration is inserted to the tree [9].

In these days, the planning algorithms like the RRT method are very popular, due to their efficiency in exploration of the search space. There exist several improvements of the traditional method, for example the RRT* described in [10], which checks the available corrections in every iteration cycle, so it is able to find an optimal path to the goal point. The method presented in [11] called RRT$^+$ can be used in very high dimensional spaces, such as the configuration space of hyper-redundant manipulators. In this work, the T-RRT algorithm is used, that is able to find a suboptimal solution defined by a cost function in high dimensional configuration spaces as well. This solution is called as high quality path in [6]. In our case the goal of cost function is to minimize the risk of collision.

Algorithm 1: Rapidly-exploring Random Tree (RRT)

Input: the configuration space $\mathcal{C}$
the root q_{init} and the goal q_{goal}
Output: the tree $\mathcal{T}$

1: $\mathcal{T} \leftarrow$ InitTree(q_{init})
2: **while not** StopCondition($\mathcal{T}, q_{goal}$) **do**
3: $q_{rand} \leftarrow$ SampleConf($\mathcal{C}$)
4: $q_{near} \leftarrow$ NearestNeighbor($q_{rand}, \mathcal{T}$)
5: $q_{new} \leftarrow$ Extend($\mathcal{T}, q_{rand}, q_{near}$)
6: **if** NoCollision(q_{new}) **then**
7: AddNewNode($\mathcal{T}, q_{new}$)
8: AddNewEdge($\mathcal{T}, q_{near}, q_{new}$)
9: **end if**
10: **end while**
11: **return** $\mathcal{T}$

In Algorithm 1 the traditional RRT algorithm is described. The inputs of the algorithm is the configuration space, where the path-planning method is evaluated, the initial and the goal configuration of the path-planning. The output of the method is the search tree. At first, the search tree is initialized with the initial configuration as its root node.

After that a uniformly distributed random configuration q_{rand} is generated in function SampleConf.

This configuration is used by the NearestNeighbor function to determine the nearest configuration (called q_{near}) to it in the search tree. In this method, distance from the edges is not examined, due to the higher computational demand, it is determined only from the previously added nodes.

A new point q_{new} is created in Extend, which is in the same direction as q_{rand} to q_{near}, but the distance between q_{new} and q_{near} is maximized with the parameter δ. In this method, the collision-freeness of the created q_{new} configuration is not examined.

The function NoCollision returns true, if the q_{new} configuration does not lead to a collision. In this case, a new node and edge will be added to the tree.

The algorithm is succeeded, if the distance between the last inserted point and the goal configuration is less than a given parameter, or failed, if a maximum number of iterations has been reached. In the function StopCondition are these conditions inspected.

Algorithm 2 describes the T-RRT method. As it can be seen, the method is based on the traditional RRT algorithm described in Algorithm 1. The main differences are the TransitionTest and MinExpandControl functions (see Subsections 3.1-3.2), which are the conditions to extend the search tree with a new node and edge.

The cost of the found path can be evaluated by several methods, e.g. maximal cost, average cost or total cost along the path. However it will lead to a path, that avoids the areas of search space with higher costs, if the performed work is minimized along the path, as it is introduced in [6]. In this case, the positive variations of the cost function will be minimized.

Algorithm 2: Transition-based RRT

Input: the configuration space $\mathcal{C}$
the cost function $c : \mathcal{C} \mapsto \mathbb{R}_+^*$
the root q_{init} and the goal q_{goal}
Output: the tree $\mathcal{T}$

$\mathcal{T} \leftarrow$ InitTree(q_{init})
while not StopCondition($\mathcal{T}, q_{goal}$) **do**
  $q_{rand} \leftarrow$ SampleConf($\mathcal{C}$)
  $q_{near} \leftarrow$ NearestNeighbor($q_{rand}, \mathcal{T}$)
  $q_{new} \leftarrow$ Extend($\mathcal{T}, q_{rand}, q_{near}$)
  if $q_{new} \neq NULL$ **then**
    $d_{near-new} \leftarrow$ Distance(q_{near}, q_{new})
    if TransitionTest($c(q_{near}), c(q_{new}), d_{near-new}$)
    and MinExpandControl($\mathcal{T}, q_{near}, q_{rand}$)
    and NoCollision(q_{new}) **then**
      AddNewNode($\mathcal{T}, q_{new}$)
      AddNewEdge($\mathcal{T}, q_{near}, q_{new}$)
    end if
  end if
end while
return $\mathcal{T}$

3.1 Transition-test

Algorithm 3 describes the TransitionTest function. In the first step, the number, how many times the transition-test failed previously, is queried by the GetCurrentNFail function.

Thereafter if the cost in q_{new} point is higher than a c_{max} parameter, then the test will not be accepted, so the search tree will not be extended. In case of negative variation of cost function the transition test will return true.

After that in case of positive slope of the cost, the transition test will be successful in accordance with a probability based on the Metropolis criterion [12], defined as

$$p_{ij} = \exp(-\frac{\Delta c_{ij}}{K \cdot T}) \tag{1}$$

where $\Delta c_{ij} = \frac{c_j - c_i}{d_{ij}}$ is the variation of the cost, K is a normalization constant, it can be taken as the average cost of the query configurations.

The difficulty of success for a given transition is defined by the temperature parameter denoted by T. The goal of this parameter is to minimize the amount of positive slopes in the cost function along the path, but it allows to explore the whole reachable configuration space by its adaptive tuning. In case of higher temperature the acceptance of transition test is more probable for a transition with higher positive slope. To accept only very low positive variations at the start, T is initialized with a very low value. This functions similar to the simulated annealing algorithm.

The function Rand(0,1) gives a random number between 0 and 1 based on uniform distribution.

The $\alpha > 1$ variable is used to change the temperature adaptively. If the transition was successful then the temperature is decreased. And if the test failed $nFail_{max}$ times then the value of T is increased by multiplying with α. Thus if the $nFail_{max}$ parameter is high then the path will be more likely to go into the valleys, however finding a path over a region with high cost will be more difficult.

Algorithm 3: TransitionTest(c_i, c_j, d_{ij})

$nFail$ = GetCurrentNFail();
if $c_j > c_{max}$ **then**
  return False
end if
if $c_j < c_i$ **then**
  return True
end if
$p = \exp(\frac{-(c_j - c_i)/d_{ij}}{K \cdot T})$
if Rand(0, 1) $< p$ **then**
  $T = T/\alpha$
  $nFail = 0$
  return True
else
  if $nFail > nFail_{max}$ **then**
    $T = T \cdot \alpha$
    $nFail = 0$
  else
    $nFail = nFail + 1$
  end if
  return False
end if

3.2 Minimal Expansion Control

The transition-test can slow the exploration rate of high cost hills and the new configurations only refine the already known regions.

By specifying a minimum level of exploration, this can be prevented. Accordingly, a new node close to the closest search tree configuration will only be accepted if the refining node ratio is smaller than the ρ parameter. The minimal exploration control method will accept the exploring nodes automatically.

This function is introduced in Algorithm 4. The function UpdateNbNodeTree increases the number of nodes, while the UpdateNbRefineNodeTree function updates the number of refining nodes in the search tree.

The Euclidean distance between the q_{rand} (not the extended q_{new}) and the q_{near} configurations is used to determine the type of a given point. A point is an exploring node if the distance is high, otherwise it will be a refining node.

The path-planning will discover the entire configuration space very effectively with this complement, while trying to stay in the valleys of the cost function owing to the transition-test.

Algorithm 4: MinExpandControl($\mathcal{T}, q_{near}, q_{rand}$)

```
if Distance(q_near, q_rand) > δ then
    UpdateNbNodeTree(T)
    return True
else
    if NbRefineNodeTree(T+1) / NbNodeTree(T+1) > ρ then
        return False
    else
        UpdateNbRefineNodeTree(T)
        UpdateNbNodeTree(T)
        return True
    end if
end if
```

4 Collision detection

The effectiveness of a path planning method depends highly on the applied collision detection function, since it is called often (see Step 6 of Algorithm 1 or Step 8 of Algorithm 2) and it has to ensure that only collision-free configurations are used for planning. To calculate in real time, which robot configurations are in collisions, is a difficult problem. The task is not only to determine if there is any collision between the robot and the surrounding objects, but also the self-collision configurations of the robot has to be avoided.

Moreover, the goal could be not only to plan a collision-free path, but also to get a time-optimal motion. A solution for such a problem is presented in [1], but the computational demand of that algorithm is high, due to the solution of the optimization problem. However, the collision detection algorithm used by that method can be applied by other applications as well. The algorithm presented in this paper is also based on this collision detection.

4.1 Collision detection with polyhedrons

The basic idea of this collision detection method is that the geometry of the robot and the obstacles is approximated such that they are represented by unions of convex polyhedrons. Fig. 2 and Fig. 3 illustrate how the bounding polyhedrons can be determined around the robot and the obstacles. For the sake of simplicity, it is supposed that the workspace of the robot contains only one obstacle. The presented collision detection method can be easily extended if more obstacles are present.

Let P denote the union of polyhedrons that represent the robot:

$$P = \bigcup_{i=1}^{n_P} P^{(i)} = \bigcup_{i=1}^{n_P} \{y \in \mathbb{R}^3 | A^{(i)} y \leq b^{(i)}\} \tag{2}$$

where n_P shows the number of polyhedrons in P. A polyhedron is bounded by faces. A face $P^{(i)}$ is defined by $A^{(i)}$ and $b^{(i)}$ parameters and p_i is the number of faces. Then $A^{(i)} \in \mathbb{R}^{p_i \times 3}$ and $b^{(i)} \in \mathbb{R}^{p_i}$ for $i = 1, \ldots, n_p$.

Similarly, an obstacle Q can also be described by the union of polyhedrons:

$$Q = \bigcup_{j=1}^{n_Q} Q^{(j)} = \bigcup_{j=1}^{n_Q} \{y \in \mathbb{R}^3 | C^{(j)} y \leq d^{(j)}\} \tag{3}$$

where n_Q denotes the number of polyhedrons in Q. q_j shows the number of faces in $Q^{(j)}$, hence $C^{(j)} \in \mathbb{R}^{q_j \times 3}$ and $d^{(j)} \in \mathbb{R}^{q_j}$ for $j = 1, \ldots, n_Q$.

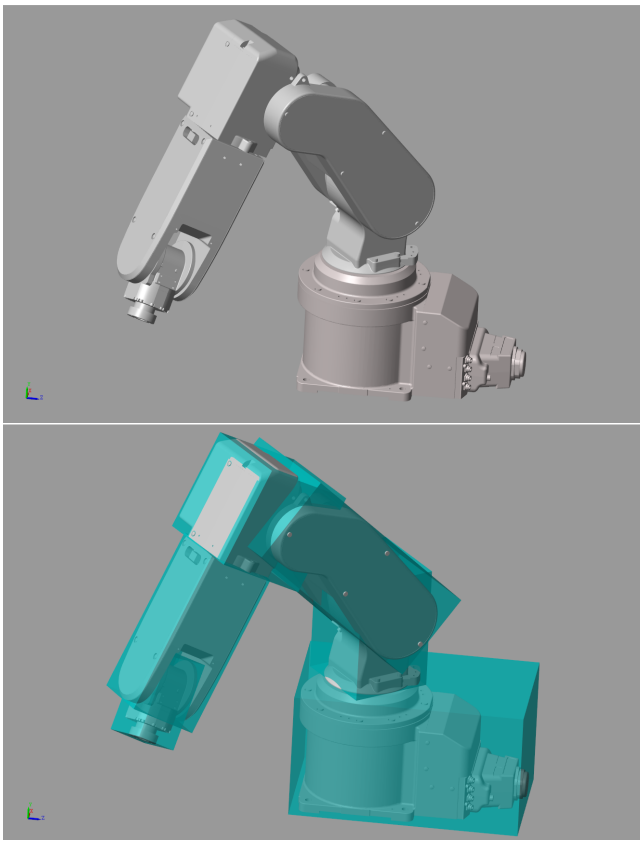

Figure 2: Bounding boxes of each segments were defined as the bounding polyhedrons of the manipulator

To avoid self-collisions, pairs of polyhedrons $P^{(i)}$ have to be checked, whether they collide with each-other. There exist such

pairs of $P^{(i)}$ where the collision detection is not necessary, since they cannot collide physically (e.g. adjacent segments). Therefore, the set of pairs $(k, l), k \neq l$ has to be defined, such that k and l denote two polyhedrons $(P^{(k)}, P^{(l)})$ of the robot which could collide. This set is denoted by I. If $(k, l) \in I$, the collision check has to be performed for the corresponding two polyhedrons.

Suppose, that P is already calculated for a given configuration and Q is determined as well, then the configuration is collision-free if

$$P^{(i)} \cap Q^{(j)} = \emptyset \quad \wedge \quad P^{(k)} \cap P^{(l)} = \emptyset \tag{4}$$

$\forall i = 1, \ldots, n_P$ and $\forall j = 1, \ldots, n_Q$ and $\forall (k, l) \in I$.

One polyhedron of the robot $P^{(i)}$ and one of the obstacle $Q^{(j)}$ do not collide with each-other if the union of their system of inequalities has no solution. Formally, there exists no such $y^{(i,j)} \in \mathbb{R}^3$ point where

$$\begin{pmatrix} A^{(i)} \\ C^{(j)} \end{pmatrix} y^{(i,j)} \leq \begin{pmatrix} b^{(i)} \\ d^{(j)} \end{pmatrix} \tag{5}$$

Similar inequalities can be used to check self-collision.

To determine, whether an inequality similar to (5) has no solution, Farkas's lemma can be applied. The lemma says that there exists no solution for the linear system if and only if a vector $w^{(i,j)} \in \mathbb{R}^{(p_i+q_j)}$ can be found such that

$$w^{(i,j)} \geq 0 \quad \text{and} \quad \begin{pmatrix} A^{(i)} \\ C^{(j)} \end{pmatrix}^{\top} w^{(i,j)} = 0 \quad \text{and} \tag{6}$$

$$\begin{pmatrix} b^{(i)} \\ d^{(j)} \end{pmatrix}^{\top} w^{(i,j)} < 0$$

Consequently, if given is a configuration q and a $w^{(i,j)}$ solution can be found for (6), then the q configuration can be marked as collision-free.

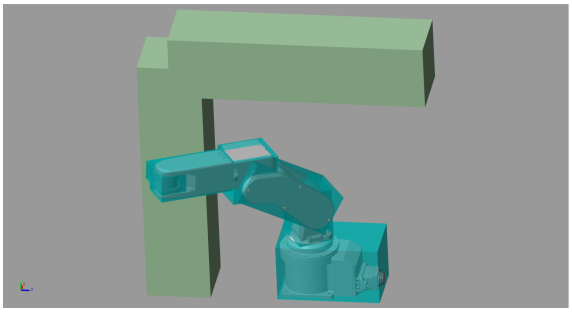

Figure 3: Bounding polyhedrons around the robot and obstacles (as used in the simulation)

4.2 Construction of polyhedrons

The face parameters $(A^{(i)}, b^{(i)})$ for some robot-polyhedron $P^{(i)}$ depend on the actual robot configuration q. This subsection describes how to calculate them.

First, $A_r^{(i)}$ and $b_r^{(i)}$ have to be determined for each segment of the robot in a reference coordinate system that is attached to the same segment. For example, the CAD model of the manipulator (e.g. shown in Fig. 1) can be used to measure the corresponding parameters in this frame. (For obstacles, the same process has to be performed.)

Then, a transformation is required to get these parameters in the base frame. Denote the resultant transformation matrix by $T^{(i)}(q)$. $T^{(i)}(q)$ can be used for the transformation from the base coordinate system to the reference frame of the i_{th} segment, if the actual configuration of the robot is q.

The parameters in the base frame can be determined as follows

$$\begin{bmatrix} A^{(i)} & b^{(i)} \end{bmatrix} = \begin{bmatrix} A_r^{(i)} & b_r^{(i)} \end{bmatrix} \begin{bmatrix} T^{(i)}(q) \end{bmatrix}^{-1} \tag{7}$$

During the collision-check algorithm, these parameters have to be used.

5 Cost-function approximation

The goal of cost-function is to keep distance from configurations leading to a collision. The approximation of this function is necessary, to decrease the computational demand of the algorithm. This cost-function is used by the T-RRT path-planning method, introduced in Section 3.

5.1 Cost-funcion approximation with Gauss-functions

At first, the approximation of cost-function was evaluated by taking enough sample of the configuration space and Gauss-functions were summed in every sampling point. The centers of Gauss-functions are the configurations causing a collision. There are several disadvantages of this method.

As it can be seen in the Fig. 4 the result is very hilly and a maximal value of the cost function can not be predefined, so the c_{max} parameter of the TransitionTest function in Algorithm 3 is difficult to determine.

In addition, the cost function has to be stored in every sampling point, moreover the interpolation is slow at any other point.

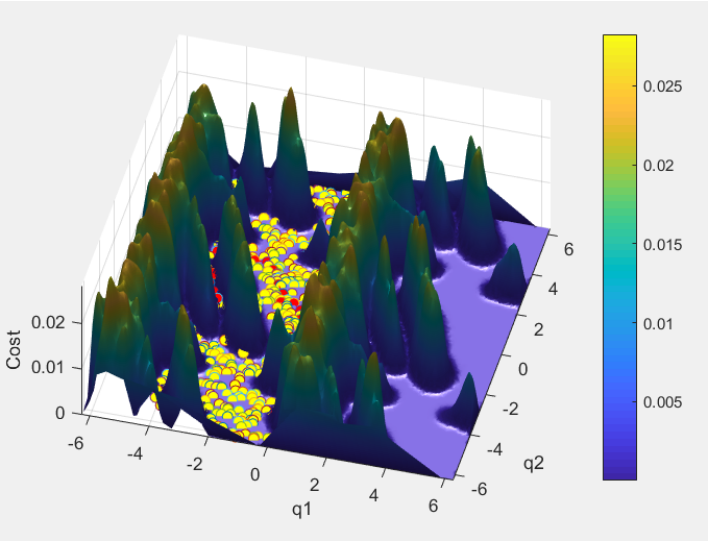

Figure 4: Cost-function approximation with Gauss-functions with the search tree found by the T-RRT algorithm

5.2 The base of fuzzy approximation

Another solution could be the fuzzy approximator systems.

Fuzzy systems have several uses in many areas such as decision making [13], system identification [14], control theory [15] etc. In addition, the approximation of nonlinear functions is possible as well, if enough teaching points are available. Teaching points are defined as $y_j = f(x_j)$, where f is the function that needs to be approximated, x_j is the position of the given teaching point and y_j is the value of f in x_j.

There are several advantages of fuzzy estimator systems. Depending on which type of inference method is used, their rules can be tuned adaptively and the approximation time is one of the benefits of these systems as well.

Singleton fuzzyfication, center average defuzzyfication, product inference and zero-order Sugeno model fuzzy systems with Gaussian membership functions are therefore the most widely used approaches [16].

Thus the fuzzy estimation of a nonlinear function is defined as

$$\hat{f}(x) = \frac{\sum_{l=0}^{M} \bar{y}^l \prod_{i=1}^{n} \exp(-(\frac{x_i - \bar{x}_i}{\sigma_i^l})^2)}{\sum_{l=0}^{M} \prod_{i=1}^{n} \exp(-(\frac{x_i - \bar{x}_i}{\sigma_i^l})^2)} \tag{8}$$

where n is the number of dimensions in the configuration space, $\bar{y}^l$ is the output in the l_{th} rule of the fuzzy inference system, $\bar{x}_i$ is the location of the maximum value of the Gaussian membership function and with σ_i^l the width of this function can be modified.

The adaptive tuning of the fuzzy rules is feasible by modifying the $\bar{y}^l$, $\bar{x}_i$ and σ_i^l values.

According to the Universal approximation theorem described in [16], for any given real continuous function f on the compact set $U \in \mathbb{R}^n$ and arbitrary $\epsilon > 0$ there exists $\hat{f}$ in the form given in (8) such that

$$\sup_{x \in U} |f(x) - \hat{f}(x)| < \epsilon \tag{9}$$

Henceforth, two adaptive tuning method for fuzzy estimator systems will be introduced. In which bounded domain intervals of input vector x and output y are assumed.

5.3 Generating fuzzy rules by learning from examples

Details of the algorithm can be found in [17].

The input and output domain is divided into intervals, in each Gaussian membership functions are defined.

The possible fuzzy relations are determined exactly by the number of input membership functions. For example, in case of input variables x_1 and x_2, if the number of membership functions are N_1 and N_2 then the number of possible rules is $N_1 \cdot N_2$. For each teaching point the relation and output membership function has to be selected that mostly fits the given teaching point.

In k_{th} teaching step a generic form of fuzzy rules can be defined:

$$R^k : \text{IF } x_1 \text{ IS } F_1^k \text{ AND } x_2 \text{ IS } F_2^k \text{ AND } \ldots \text{ AND } x_n \text{ IS } F_n^k$$

$$\text{THEN } y \text{ IS } G^k$$

where F_i^k is one of the input variable membership functions and G^k is one of the output variable membership functions.

In the k_{th} step a given rule will be chosen, if it maximizes the following expression:

$$D(R^k) = \prod_{i=1}^{n} (\mu_{F_i^k}(x_i^k)) \cdot \mu_{G^k}(y^k) \tag{10}$$

where μ is the firing strength of the given membership function for the actual value of the teaching point.

There exist more possible solutions to that problem, if a new relation has to be added with the same condition part, but a different inference part as an already added relation. In this case, the relation with higher value can be selected by evaluating (10), or a weighted value of both inference parts can be specified.

The main disadvantage of this technique is that (10) has to be evaluated $N = N_1 \cdot N_2 \cdot \ldots \cdot N_n$ times for every teaching points (where n is the number of dimensions of the space) to find the maximum value. As a consequence, with the number of dimensions the computational time grows exponentially.

5.4 Nearest neighborhood clustering

In this algorithm, teaching points are clustered depending on their position in the configuration space. A given cluster can be described by three parameter, the center of it x_0^l, the number of teaching points in it B^l and the sum of values of these points A^l [18].

A new point x_j ($y_j = f(x_j)$) is inserted to the cluster l, if it is closer to x_0^l than a given r radius. After that the other cluster parameters A^l and B^l will be updated. If the distance of the teaching point x_j is too high from all of the clusters then a new cluster will be created as $\{x_0^l = x_j; A^l = y_j; B^l = 1\}$. M represents the number of clusters that have been defined during the teaching process.

Therefore, the estimation of the nonlinear function can be defined as

$$\hat{f}(x) = \frac{\sum_{l=0}^{M} A^l \exp(-(\frac{x - \bar{x}_0^l}{\sigma^l})^2)}{\sum_{l=0}^{M} B^l \exp(-(\frac{x - \bar{x}_0^l}{\sigma^l})^2)} \tag{11}$$

5.5 Cost-function evaluation

Eventually, the cost-function for the T-RRT method was approximated with the Nearest neighborhood clustering algorithm, because it has a higher computational efficiency.

To determine the position of the teaching points, random sampling of the configuration space is needed, owing to the high number of dimensions.

The collision detection method described in Section 4 is evaluated in all of teaching points. If the x_j configuration causes collision, then one will be assigned as the value of the cost-function y_j in x_j, otherwise zero.

After that the rules of the fuzzy approximator are tuned with the Nearest neighborhood clustering algorithm. This method can be used for arbitrary robotic manipulators. An example for the cost function and path can be seen in Fig. 5 in case of 2-DoF manipulator (since it is not possible to depict the cost function in higher dimensions).

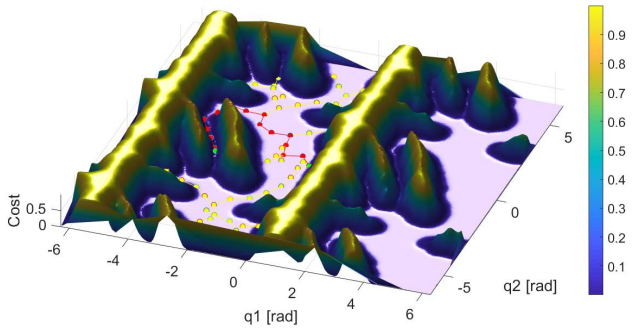

Figure 5: T-RRT method is able to find high quality path based on a cost-function. The cost-function is evaluated for a two degrees of freedom manipulator with the Nearest Neighborhood Clustering algorithm.

Table 1 shows the comparison of approximation with Gauss-function and the Nearest neighborhood clustering algorithm. As can be seen, both the training time and the evaluation time for one configuration in the case of Gauss-function approximation grow very fast by increasing the number of teaching points. In the case of Nearest neighborhood clustering, these values increased to a lesser extent. By using clusters, the evaluation time of the fuzzy function-approximation method did not increase significantly, which is ideal for applying the method in a path-planning algorithm.

Table 1: Comparison of fuzzy function-approximation and approximation with Gauss-functions

Num. of teaching points	Nearest Neighborhood clustering			Gauss	
	Training time	Number of clusters	Eval. time	Training time	Eval. time
1000	0.0481s	714	22μs	0.0402s	7.8μs
5000	0.207s	1649	45μs	0.3027s	28.8μs
10000	0.4554s	1974	52μs	1.4151s	58.2μs
50000	3.0962s	2431	65μs	22.88s	137.9μs
100000	7.045s	2521	76μs	155.26s	1632μs

6 Trajectory generation

The already presented methods can be used to find points in the configuration space, that by moving the robot between these points, the possible collisions can be avoided.

6.1 Trajectory generation for scalar values

In the following method y denotes one coordinate of the q configuration, in other words, one joint variable of the robot.

Let $\{y_k\}$ be the set of points founded by the path-planning algorithm and $\{t_k\}$ denotes the desired absolute time of reaching these points [19].

$$\begin{aligned} \{y_k\} &= \{\dots, A, B, C, D, \dots\}, \\ \{t_k\} &= \{\dots, t_A, t_B, t_C, t_D, \dots\} \end{aligned} \tag{12}$$

Due to the limits of torques, boundaries have to be defined for the acceleration of joints: $|\ddot{y}|_{max}$.

At first, $y(t)$ trajectory can be defined as a series of linear functions between adjacent points. In this case, the first derivative $\dot{y}(t)$ is constant between two points and it steps to another constant value in a given point. The acceleration $\ddot{y}(t)$ is zero in every point except the points where the slope of $y(t)$ trajectory is changing. In these points, the second derivative contains Dirac-delta functions, which leads to an unbounded control value, so this solution is not applicable. To overcome this problem, the acceleration of joints have to be changed continuously.

To reach the next point two kinds of phases can be defined: traveling phase and acceleration phase. In the former, we use a constant velocity and zero acceleration, while in the latter a continuous acceleration function has to be applied, to reach the desired constant velocity of the next traveling phase. The quadratic function can be chosen as the form of the function in the acceleration phase (Fig. 6).

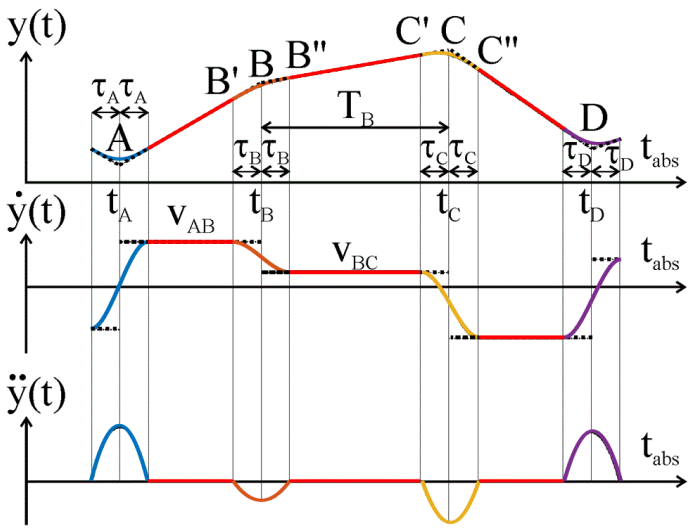

Figure 6: To ensure the continuity of the acceleration function, quadratic function can be applied in the acceleration phase. The form of the trajectories can be seen in the figure.

Let B and C be two adjacent points, while B' and C' are the beginning, B'' and C'' are the end of their acceleration phase. The relative time t can be defined as

$$t = t_{abs} - t_B \in [-\tau_B, T_B], \tag{13}$$

where t_B is the absolute time of reaching the point B, τ_B is the half of duration of acceleration phase in B and T_B denotes the time-interval of reaching point C from point B. This can be seen in Fig 7. The time-intervals between configurations can be determined by the boundaries of joint-velocities, for example $T_B = \frac{C-B}{|\dot{y}|_{max}}$.

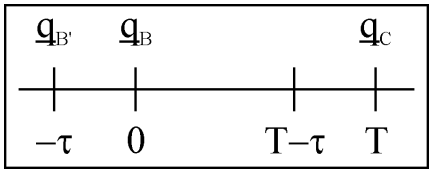

Figure 7: Relative time can be defined, where the zero value is in the q_B configuration. The zero point will be shifted with T, when the $T - \tau$ relative time is reached.

By integrating the quadratic acceleration during the acceleration phase, the velocity and position can be determined:

$$\begin{aligned} B'B'': \quad \ddot{y}(t) &= a_0 t^2 + a_1 t + a_2 \\ \dot{y}(t) &= a_0 \frac{t^3}{3} + a_1 \frac{t^2}{2} + a_2 t + a_3 \\ y(t) &= a_0 \frac{t^4}{12} + a_1 \frac{t^3}{6} + a_2 \frac{t^2}{2} + a_3 t + a_4 \end{aligned} \tag{14}$$

To determine the $a_0 \dots a_4$ parameters of the trajectory of position, five independent conditions have to be defined:

$$\begin{aligned} \ddot{y}(-\tau_B) &= 0 \\ \ddot{y}(\tau_B) &= 0 \\ \dot{y}(-\tau_B) &= v_{B'B} = \frac{B - B'}{\tau_B} \\ \dot{y}(\tau_B) &= v_{BC} = \frac{C - B}{T_B} \\ y(\tau_B) &= B + v_{BC} \cdot \tau_B \end{aligned} \tag{15}$$

The following linear equation system can be defined:

$$\begin{bmatrix} \ddot{y}(-\tau_B) \\ \ddot{y}(\tau_B) \\ \dot{y}(-\tau_B) \\ \dot{y}(\tau_B) \\ y(\tau_B) \end{bmatrix} = \begin{bmatrix} \tau_B^2 & -\tau_B & 1 & 0 & 0 \\ \tau_B^2 & \tau_B & 1 & 0 & 0 \\ -\frac{\tau_B^3}{3} & \frac{\tau_B^2}{2} & -\tau_B & 1 & 0 \\ \frac{\tau_B^3}{3} & \frac{\tau_B^2}{2} & \tau_B & 1 & 0 \\ \frac{\tau_B^4}{12} & \frac{\tau_B^3}{6} & \frac{\tau_B^2}{2} & \tau_B & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} \tag{16}$$

$$\mathbf{y} = \mathbf{C} \cdot \mathbf{a}$$

The solution of this linear equation system can be calculated as follows:

$$\mathbf{a} = \mathbf{C}^{-1} \cdot \mathbf{y} \tag{17}$$

In the traveling phase ($t \in [\tau_B, T_B - \tau_C)$), the form of the path of the scalar value will be a linear function:

$$y(t) = B + v_{BC} \cdot t \tag{18}$$

6.2 Trajectory generation for joint variables

The presented method for scalar values can be used for each joint variable separately, by taking into account the difference between the boundaries of velocities ($|\dot{q}_i|_{max}$) and accelerations ($|\ddot{q}_i|_{max}$) of some joints.

In addition, the minimal time needed to reach q_{Ci} joint value from q_{Bi} can be different as well, so just that joint have to be moved in maximal speed, that needs the most time to reach the next configuration.

The same τ parameter can be used for each joint variable and it is calculated as follows [19]:

$$\tau = \max_i \frac{3}{2} \frac{|\dot{q}_i|_{max}}{|\ddot{q}_i|_{max}} \tag{19}$$

The trajectory generation method can be seen in the Algorithm 5. The input of the method denoted by Q is the series of configurations contained by the found path.

Algorithm 5: TrajectoryGeneration(Q, τ)

```
q_C = GetFirstConfiguration(Q)
T = τ
while Size(Q) > 0 do
    q_B' = q(T − τ)
    q_B = q_C
    q_C = GetFirstConfiguration(Q)
    T_i = |q_Ci − q_Bi| / |q̇_i|_max
    T = max_i{max{T_i}, 2τ}
    v_AB = (q_B − q_B') / τ
    v_BC = (q_C − q_B) / T
    a = C^{-1} · y
    t = −τ
    while t < T − τ do
        q(t) = CalculateTrajectory(a, t)
        t = t + Δ
    end while
end while
return q(t)
```

The GetFirstConfiguration function gives the first element of the configuration queue and this element will be removed from the queue as well.

At first the variable T is initialized with τ calculated by (19), so in the first iteration the $q_{B'}$ variable will be the root configuration.

The equations in (14) are evaluated in the CalculateTrajectory function for each joint variable, consequently the dimension of the q variable is equal to the dimension of the configuration space.

Trajectories calculated with this method can be seen in Fig. 9.

7 Simulation results

The comparison of several path-planning algorithms can be seen in Table 2 in case of two degrees of freedom robotic manipulator. The result of planning with the T-RRT method can be seen in Fig. 8. The RRT* algorithm is a variant of traditional RRT method, that refines the search tree in every iteration to find the actual shortest path to the goal configuration [10]. In case of multi-directional RRT method an other search tree is started from the goal configuration [5]. The calculation time can be reduced by this method.

Table 2: Comparison of path-planning algorithms

Algorithm	Calculation time [s]	Length	Total cost	Performed work
RRT	0.039	10.54	0.415	0.896
Multi-directional RRT	0.019	10.54	0.396	0.822
Multi-directional RRT*	3.34	6.92	0.171	0.449
T-RRT$_g$	0.55	10.21	0.028	0.121
T-RRT$_t$	13.466	10.91	0.01	0.062

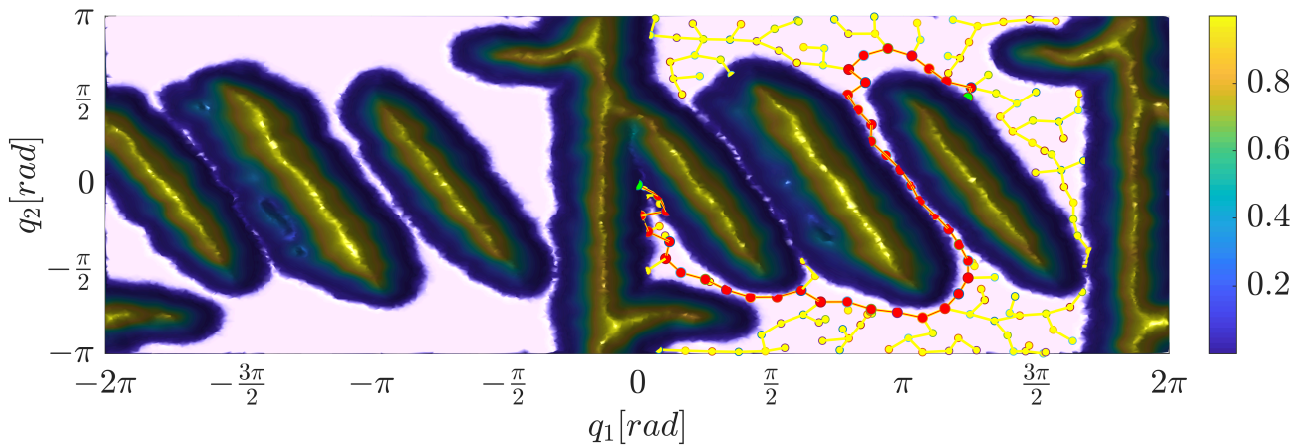

Figure 8: The result of path-planning for two degrees of freedom manipulator with the tempered version of T-RRT

Two parameter settings are examined for the T-RRT method, the first one (T-RRT$_g$) is a greedy version with $nFail_{max} = 10$, in the other case (T-RRT$_t$) $nFail_{max} = 100$, which will lead to a tempered version of T-RRT. The computation time in the first case is lower, but the found path has a lower quality as well. The other parameters were: $T_{init} = 10^{-5}$, $\alpha = 1.5$, $\rho = 0.2$, $c_{max} = 0.4$, $\delta = 0.3$. The parameters were chosen empirically.

As it can be seen, the path of the T-RRT algorithm is not equal with the shortest collision-free path.

To demonstrate the presented method, simulations were performed using the Mitsubishi RV-2F-Q manipulator as well. The task of the robot was to move to the other side of an obstacle, which is modeled by two polyhedrons. For implementation and simulation, MATLAB Simulink and Simscape Multibody toolbox were used. The running times were measured on a computer with Intel Core i7-7700HQ Processor.

For the cost function evaluation, the nearest neighborhood clustering method was applied. The parameters were empirically selected as follows: number of teaching points is $n = 20000$, radius of the cluster is $r = 1$ *rad* and deviation of the Guassian membership functions is $\sigma = 0.5$. To determine the approximated cost function, $t \approx 120s$ computational time was required, what is appropriate, since these calculations have to be performed only once for a given robot and workspace.

The parameters of the T-RRT method were selected as $T_{init} = 0.01$, $\alpha = 5$, $nFail_{Max} = 10$, $\rho = 0.1$, $c_{max} = 0.8$ and $\delta = 0.4$. To get a solution for a path-planning problem, the average calculation time was $t \approx 7.3s$.

One solution is presented in Fig. 9–Fig. 10. Fig. 9 depicts the trajectories of the joint variables. The whole movement can be seen in a video: `https://youtu.be/t6GO8LKG5js`. Some key configurations of the robot are also presented in Fig. 10. It can be seen, that the presented algorithm was able to plan such a motion, where colliding configurations can be avoided.

based on the T-RRT method, which is able to calculate the reference motion for the robot, such that collision avoidance is guaranteed and suboptimal path is selected according to some cost-function.

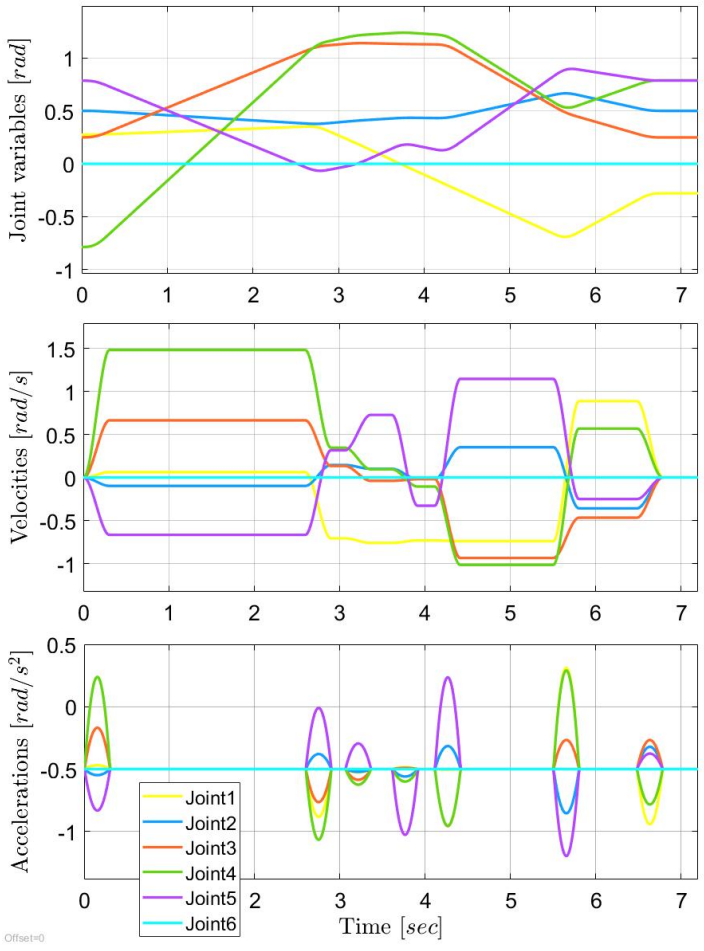

Figure 9: Trajectories of the joint variables

8 Conclusion

The method presented in this paper can be used to solve the offline path-planning problem for robotic manipulators. The algorithm is

In this work such a cost-function was selected, which can ensure, that the distance between the path of the robot and the colliding configurations is large. Instead of the time-consuming determination of

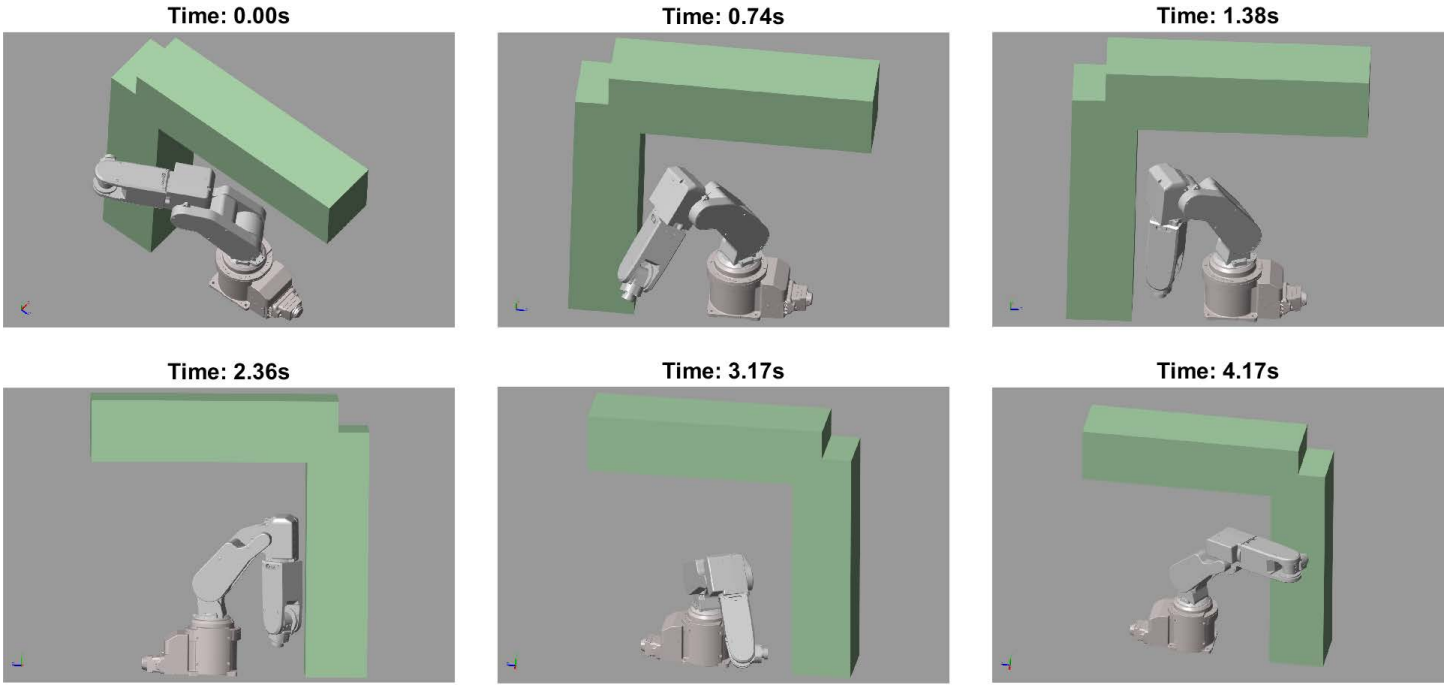

Figure 10: Robot configurations along the collision-free path

the exact cost values, a nonlinear-fuzzy approximation algorithm was suggested which uses the Nearest neighborhood clustering method.

For the collision checking of robot configurations, another approximation was applied. The segments of the robot and the obstacles can be modeled by sets of bounding polyhedrons. The collision of these polyhedrons was checked by the feasibility analysis of inequality systems.

Trajectory planning was also presented in the paper. To produce an input for a trajectory tracking and control method, time functions were determined for the joint variables and their derivatives using known bounds of the joint velocities and accelerations.

In the sequel, the goal is to develop an effective online motion-planning method, which can be used to plan a collision-free motion in such a workspace which contains obstacles with unpredictable movements as well. First, the reference motion can be determined by the presented T-RRT based method for the static environment, and additionally, a reactive planning method (similar to [20]) can be applied to ensure the avoidance of dynamic obstacles as well.

Conflict of Interest The authors declare no conflict of interest.

Acknowledgment The research reported in this paper was supported by the Higher Education Excellence Program in the frame of Artificial Intelligence research area of Budapest University of Technology and Economics (BME FIKP-MI/SC).

References

[1] Chantal Landry, Matthias Gerdts, René Henrion, and Dietmar Hömberg. Path-planning with collision avoidance in automotive industry. In *IFIP Conference on System Modeling and Optimization*, pages 102–111. Springer, 2011.

[2] Fabrizio Flacco, Torsten Kröger, Alessandro De Luca, and Oussama Khatib. A depth space approach to human-robot collision avoidance. In *2012 IEEE International Conference on Robotics and Automation*, pages 338–345. IEEE, 2012.

[3] Massimo Cefalo, Emanuele Magrini, and Giuseppe Oriolo. Parallel collision check for sensor based real-time motion planning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1936–1943. IEEE, 2017.

[4] Anthony Stentz. Optimal and efficient path planning for partially known environments. In *Intelligent Unmanned Ground Vehicles*, pages 203–220. Springer, 1997.

[5] Steven M LaValle. *Planning algorithms*. Cambridge University Press, 2006.

[6] Léonard Jaillet, Juan Cortés, and Thierry Siméon. Sampling-based path planning on configuration-space costmaps. *IEEE Transactions on Robotics*, 26(4):635–646, 2010.

[7] Ran Zhao. *Trajectory planning and control for robot manipulations*. PhD thesis, Université Paul Sabatier-Toulouse III, 2015.

[8] Mark W Spong, Seth Hutchinson, Mathukumalli Vidyasagar, et al. *Robot modeling and control*. John Wiley & Sons, Inc., 2006.

[9] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical report, Department of Computer Science, Iowa State University, 1998.

[10] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[11] Marios Xanthidis, Ioannis Rekleitis, and Jason M O'Kane. Rrt+: Fast planning for high-dimensional configuration spaces. *arXiv preprint arXiv:1612.07333*, 2016.

[12] Peter JM Van Laarhoven and Emile HL Aarts. Simulated annealing. In *Simulated annealing: Theory and applications*, pages 7–15. Springer, 1987.

[13] Dug Hun Hong and Chang-Hwan Choi. Multicriteria fuzzy decision-making problems based on vague set theory. *Fuzzy sets and systems*, 114(1):103–113, 2000.

[14] Robert Babuška and Henk Verbruggen. Neuro-fuzzy methods for nonlinear system identification. *Annual reviews in control*, 27(1):73–85, 2003.

[15] L-X Wang. Stable adaptive fuzzy control of nonlinear systems. *IEEE Transactions on fuzzy systems*, 1(2):146–155, 1993.

[16] L-X Wang. Fuzzy systems are universal approximators. In *[1992 Proceedings] IEEE International Conference on Fuzzy Systems*, pages 1163–1170. IEEE, 1992.

[17] L-X Wang and Jerry M Mendel. Generating fuzzy rules by learning from examples. *IEEE Transactions on systems, man, and cybernetics*, 22(6):1414–1427, 1992.

[18] L. . Wang. Training of fuzzy logic systems using nearest neighborhood clustering. In *[Proceedings 1993] Second IEEE International Conference on Fuzzy Systems*, pages 13–17 vol.1, March 1993.

[19] Lantos Béla. *Robotok irányítása (Robot Control)*. Akadémiai K., 2002.

[20] MG Mohanan and Ambuja Salgoankar. A survey of robotic motion planning in dynamic environments. *Robotics and Autonomous Systems*, 100:171–185, 2018.