GPU Accelerated Collision Detection for Robotic Manipulators

Dániel Szabó
Department of Control Engineering and Information Technology
Budapest University of Technology and Economics
Budapest, Hungary
dszabo@iit.bme.hu

Emese Gincsainé Szádeczky-Kardoss
Department of Control Engineering and Information Technology
Budapest University of Technology and Economics
Budapest, Hungary
szadeczky@iit.bme.hu

2022 8th International Conference on Automation, Robotics and Applications (ICARA) | 978-1-6654-8383-4/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICARA55094.2022.9738587

***Abstract*—This paper presents a collision detection method for robotic manipulators. In this work, three methods are compared with each other. In the first one, both the segments of the manipulator and the obstacles are modeled with polyhedrons, and the collision between them is detected by checking the feasibility of the union of inequality systems describing each of them. The second one is solving the collision detection problem between meshes and polyhedrons, while the third one uses the same process, but parallelizing it by using GPU. These are general methods to determine which robot configurations are causing collisions, so they can be used in path-planning or in collision-avoidance algorithms as well. The presented GPU-based parallelized method overperformed the previously examined algorithms and it is fast enough to use in online applications.**

***Index Terms*—robotic manipulator, collision detection, GPU**

I. Introduction

Nowadays, with the help of industrial automation, it is possible to create highly automated manufacturing processes. Thanks to the trends of Industry 4.0, there is an ever-growing need for robotic manipulators, which are capable to modify their predefined trajectories, in order to avoid collisions in a dynamically changing environment.

One of the most researched use-cases of such manipulators is the human-collaborative robot [1]. This type of robot is working with the human operator in the same environment, thus it is crucial to be aware of the safety aspects of these systems. It is important, to have a fast and reliable collision detection method, with which the harmful collisions with the moving objects can be eliminated. There are two techniques, how the collisions can be taken into consideration during the dynamic trajectory planning.

Before the collision occurs, the main task is to avoid the possible impacts. In this case, the actual position of the dynamic objects needs to be known by the collision avoidance algorithm, which means that it is important to observe them with some external sensors (e.g. depth camera). On the other hand, when a contact already exists between the obstacles and the manipulator, detection of this state is required, and after that, the controller has to react to this event.

The latter is a necessary feature in the case of dynamic motion-planning. In [2] a collision detection algorithm is introduced, in which no external sensors are required, to detect the collisions with the dynamic objects. This method determines if a collision exists based on the total energy and the generalized momentum of the manipulator.

Another solution to this problem is presented in [3], where an extended state observer is used to detect already existing contacts between the robotic arm and the objects in the environment.

In our paper, the presented collision detection algorithm is meant to be used in path planning or in a collision avoidance algorithm. Thus, in this case, it is necessary to decide whether a particular configuration involves a collision or not.

Some of the existing algorithms are capable to detect collisions between deformable objects as well [4], [5]. Although these methods are providing an opportunity to simulate these types of objects, they can not be used in real-time yet, due to their computational expensiveness. Therefore in our case, the segments of the robot and the obstacles were modeled with rigid bodies.

There are methods, where some of the objects are modeled in a probabilistic way [6], [7]. In this case, because the environment is modeled in a stochastic manner, the probability of collisions can be estimated. After that, this can be used in a higher level of the control system to modify the robot motion based on this information.

It is possible to approximate the segments and the obstacles with some bounding polyhedrons, e.g. with oriented bounding boxes (OBB) or with axis-aligned bounding boxes (AABB), and the collision can be determined by using these simpler models [8].

This paper is organized as follows. In Section II the structure of manipulators is described. Section III introduces how the collisions can be determined between polyhedrons, based on the feasibility of the union of inequality systems describing these shapes. Then, Section IV describes how the collisions can be noticed if the segments of the robot or the environment are modeled with meshes, while the other one remains modeled with the polyhedron. Section V presents the simulation results. Finally, the possible developments and the conclusions are contained in Section VI.

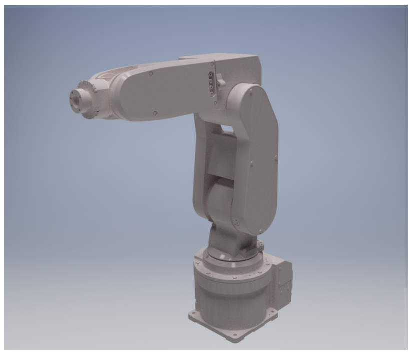

Fig. 1. The 3D model of Mitsubishi RV-2F-Q manipulator that is used in the simulations

II. Kinematics of robotic manipulators

A robotic arm is described by an opened kinematic chain, where typically the adjacent links are connected by revolute or prismatic joints. With revolute joints, relative rotational movements can be applied between the neighboring links, while the prismatic one is used to apply translational movements. The actual relative displacement applied by the i_{th} joint is called a joint variable and it is denoted by q_i. The dimension of this variable is angle or distance, related to the two different types of joints.

The number of joints is often referenced as the degrees of freedom of the manipulator, while the robot configuration is the vector composed of the joint variables: $\mathbf{q} = \{q_i\}$.

By solving the direct geometry problem, the state of the end-effector (position and orientation) can be calculated by the actual joint variables. This can be performed by transforming the world coordinate system to the frame attached to the tool, with a chain of homogeneous transformations defined by the structure of the robot and its actual configuration.

The manipulator structure can be defined by a standard called the Denavit-Hartenberg description, which defines four parameters for all of the robot links. The exact process can be found in [9].

The simulations were performed by using the Mitsubishi RV-2F-Q manipulator, a six degrees of freedom robotic arm. The 3D model of this robot is depicted in Fig. 1.

III. Collision detection between polyhedrons

For most dynamic trajectory planning algorithms, it is necessary to know if a configuration is collision-free or not. It is not a trivial problem, how the objects in the environment and the segments of the robot are modeled during the collision detection.

It is possible, to use a mesh representation for both the robot and the obstacles. However this choice can give the most exact representation of the object structures, but the calculations are computationally expensive.

Another solution could be the modeling of the robot segments and the obstacles with some simpler bounding polyhedrons.

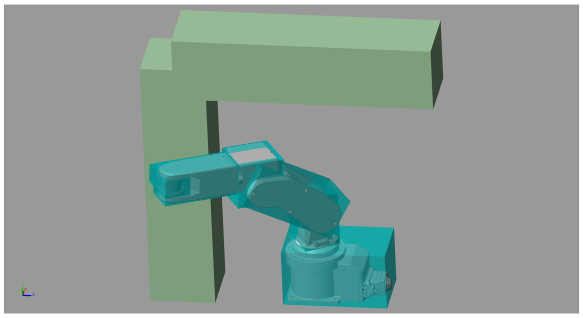

Fig. 2. The Mitsubishi RV-2F-Q manipulator, two object and the OBBs around them

In [10], a constrained optimization process was performed to find collision-free and time-optimal motion, but the resulting algorithm is computationally expensive, due to the optimization method. In [11], a path-planning method was introduced, where a similar collision detection was used, and by using a fuzzy inference system, the algorithm became faster.

Unions of polyhedrons can be used to model the manipulator and obstacles in the workspace as can be seen in Fig. 2, where oriented bounding boxes (OBB) were defined around the robot segments and the objects. Let us suppose that only one obstacle exists in the environment – the algorithm can be easily extended in the case of more objects.

Let $P(\mathbf{q})$ denote the union of polyhedrons describing the segments of the robot:

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

where n_P is the number of polyhedrons in P. $\mathbf{A}^{(i)}$ and $\mathbf{b}^{(i)}$ are defining the plane parameters overlapping with the faces of the i_{th} polyhedron $P^{(i)}(\mathbf{q})$, be p_i the number of planes. Then $\mathbf{A}^{(i)} \in \mathbb{R}^{p_i \times 3}$ and $\mathbf{b}^{(i)} \in \mathbb{R}^{p_i}$ for $i = 1, \ldots, n_p$. The $\mathbf{y}$ point is inside $P^{(i)}(\mathbf{q})$ polyhedron if the all of the inequalities defined by $\mathbf{A}^{(i)}$ and $\mathbf{b}^{(i)}$ in (1) are satisfied.

Similarly, the union of obstacles Q can be defined as follows:

$$Q = \bigcup_{j=1}^{n_Q} Q^{(j)} = \{\mathbf{y} \in \mathbb{R}^3 | \mathbf{C}^{(i)}\mathbf{y} \leq \mathbf{d}^{(i)}\} \qquad (2)$$

where n_Q is the number of polyhedrons in Q. If q_j is the number of planes in $Q^{(j)}$, then $\mathbf{C}^{(i)} \in \mathbb{R}^{q_j \times 3}$ and $\mathbf{d}^{(i)} \in \mathbb{R}^{q_j}$ for $j = 1, \ldots, n_Q$.

Let us define I as a set of index pairs (k, l), $k \neq l$, where each of the two polyhedrons of the manipulator indexed by k and l need to be examined if they are collide or not. Since not all of the segments pairs can physically collide (e.g. adjacent segments), the I does not contain all the possible index pairs, therefore the investigation of the self-collision is unnecessary in these cases.

If Q is given and P is calculated in the given robot configuration, then no collision exists if

$$P^{(i)} \cap Q^{(j)} = \emptyset \quad \wedge \quad P^{(k)} \cap P^{(l)} = \emptyset \qquad (3)$$

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

$P^{(i)}$ and $Q^{(j)}$ does not collide (similarly in case of self-collision) if the union of their system of inequalities has no solution. In other words, there is no such $\mathbf{y}^{(i,j)} \in \mathbb{R}^3$ point that

$$\begin{pmatrix} \mathbf{A}^{(i)} \\ \mathbf{C}^{(i)} \end{pmatrix} \mathbf{y}^{(i,j)} \leq \begin{pmatrix} \mathbf{b}^{(i)} \\ \mathbf{d}^{(i)} \end{pmatrix} \tag{4}$$

According to Farkas's lemma, an inequality system is unfeasible if and only if there exists a vector $\mathbf{w}^{(i,j)} \in \mathbb{R}^{(p_i+q_j)}$ that satisfies

$$\mathbf{w}^{(i,j)} \geq 0 \quad \text{and} \quad \begin{pmatrix} \mathbf{A}^{(i)} \\ \mathbf{C}^{(i)} \end{pmatrix}^{\top} \mathbf{w}^{(i,j)} = 0 \quad \text{and} \tag{5}$$

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

A. Construction of polyhedrons

At first, the $\mathbf{A}_r^{(i)}$ and $\mathbf{b}_r^{(i)}$ parameters describing the parameters of the i_{th} segment should be determined. These should be defined in a reference frame attached to the given segment, which can be done by using the CAD model of the manipulator (Fig.1). In case of obstacles, the same method can be applied.

To transform the parameters from the reference frame of the i_{th} segment to the base coordinate system, a $\mathbf{T}^{(i)}(\mathbf{q})$ resultant homogeneous transformation can be defined by knowing the actual $\mathbf{q}$ configuration and the robot structure.

Then the parameters in the base frame can be calculated as

$$\begin{bmatrix} \mathbf{A}^{(i)} & \mathbf{b}^{(i)} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_r^{(i)} & \mathbf{b}_r^{(i)} \end{bmatrix} \begin{bmatrix} \mathbf{T}^{(i)}(q) \end{bmatrix}^{-1} \tag{6}$$

IV. Collision detection with meshes and polyhedrons

A mesh is a network, which is composed of polygon cells and points. It can represent 3D models in almost any shape and size. Two types of meshes can be distinguished, the structured and the unstructured meshes (Fig. 3).

In the case of the former one, the cell structures are rectangular, spherical, or in other simple forms, which makes it easier to determine the adjacent cells. These types of meshes are used for example in finite-difference time-domain simulations [12].

In the case of the unstructured meshes, there is no simple way to find the neighboring cells, but a much more detailed model can be represented by using them.

It is computationally expensive, to determine if a given configuration for complex meshes is causing a collision or not. Therefore we solved the collision detection problem between meshes and polyhedrons.

It is not trivial if the robot segments or the obstacles need to be modeled with meshes, the Sections IV-A and IV-B will discuss about this decision.

If one mesh and one polyhedron intersect each other, then there exists some point in the mesh that satisfies the inequality system of the polyhedron. The pseudo-code of the collision detection can be seen in Algorithm 1.

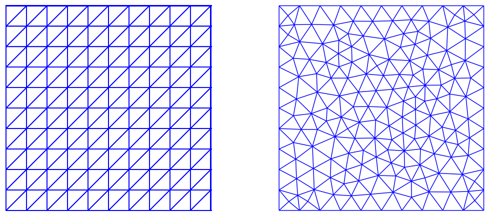

Fig. 3. Structured (left) and unstructured (right) two-dimensional meshes

Algorithm 1 Collision detection between meshes and polyhedrons

```
Input:
 1: the mesh points at the initial configuration: $\mathbf{M}(\mathbf{0})$
 2: the set of polyhedrons at the initial configuration: $P(\mathbf{0})$
 3: the actual configuration: $\mathbf{q}$
Output: inCollision
 4: $\mathbf{M}(\mathbf{q}) \leftarrow$ TransformMesh($\mathbf{M}(\mathbf{0}), \mathbf{q}$)
 5: $P(\mathbf{q}) \leftarrow$ TransformPolyHedrons($\mathbf{P}(\mathbf{0}), \mathbf{q}$)
 6: for each $P^{(i)} \in P(\mathbf{q})$ do
 7:     for each $\mathbf{M}^{(j)} \in \mathbf{M}(\mathbf{q})$ do
 8:         for each $\mathbf{x}_R^{(k)}(\mathbf{q}) \in \mathbf{M}^{(j)}$ do
 9:             if All($\mathbf{A}^{(i)}\mathbf{x}_R^{(k)}(\mathbf{q}) \leq \mathbf{b}^{(i)}$) then
10:                 return True
11:             end if
12:         end for
13:     end for
14: end for
15: return False
```

The inputs of this algorithm are the meshes ($\mathbf{M}(\mathbf{0})$) and polyhedrons ($P(\mathbf{0})$) interpreted in the initial configuration of the robot arm between which the investigation of collision need to be performed. The last input is the robot configuration to be investigated for collision, while the output is a boolean value indicating the presence of a collision.

At first, the meshes and the polyhedrons need to be transformed to the desired $\mathbf{q}$ configuration (lines 4-5).

The mesh transformations are defined in Section IV-A and IV-B for the case of the robot and the obstacles respectively.

The transformation for the robot polyhedrons is defined in (6), but if we want to model the obstacles with polyhedrons, then the same type of $\mathbf{T}$ homogeneous transformation can be defined for all objects.

After the transformations into the current configuration, for all polyhedrons and meshes, we need to examine whether the given mesh has a point that satisfies the inequality system of the given polyhedron. The All function returns True if all of the inequalities are satisfied in the actual iteration.

A. Modeling the robot segments with meshes

One solution could be, to model the segments of the robot with unstructured meshes, while the obstacles are modeled with the same terminology defined in (2). These meshes can be exported from the CAD model of the robot in its initial $\mathbf{q} = \mathbf{0}$ configuration.

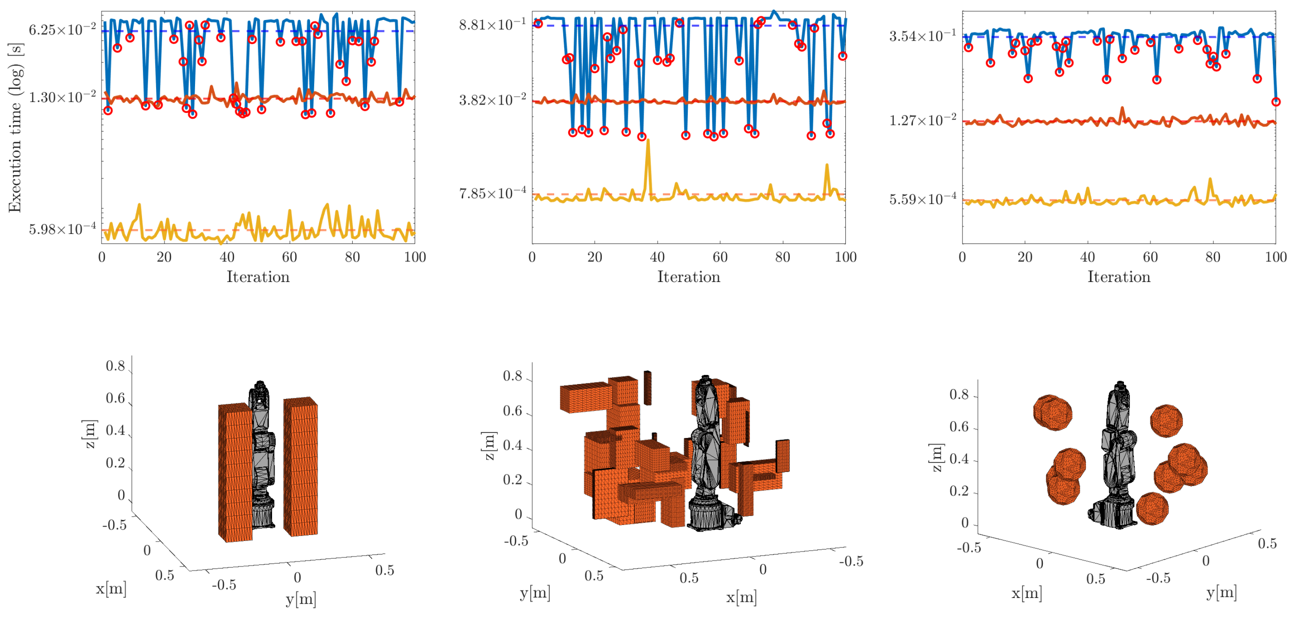

Fig. 4. The three environment configuration and the related execution times for the simulations. The blue line depicts the execution times for the method described in Section III. The orange and the yellow lines depict the results of the method from Section IV, the former is the execution time on the CPU, while the latter is the run time on the GPU. The red circles are showing in which iteration a collision has been found.

Let n be the number of the joints and take the mesh describing the k_{th} segment ($1 \leq k \leq n$). Let $\mathbf{x}_R^k(\mathbf{0})$ be one point in this mesh defined in the world coordinate system and $\mathbf{T}_{k-1,k}$ be the homogeneous transformation matrix between the frames of the $(k-1)_{\text{th}}$ and the k_{th} segments.

The points of the mesh in the actual $\mathbf{q}$ configuration can be determined as:

$$\mathbf{x}_R^k(\mathbf{q}) = \mathbf{T}_{M_k}(\mathbf{q}) \cdot \mathbf{x}_R^k(\mathbf{0}), \tag{7}$$

where $\mathbf{T}_{M_k}$ is the model matrix belonging to the k_{th} segment:

$$\begin{aligned}\mathbf{T}_{M_k}(\mathbf{q}) =& \mathbf{T}_{0,1} \cdot \mathbf{R}_1(q_1) \cdot \mathbf{T}_{1,2} \cdot \mathbf{R}_2(q_2) \cdot \ldots \cdot \\ & \cdot \mathbf{T}_{k-1,k} \cdot \mathbf{R}_k(q_k) \cdot \mathbf{T}_{k-1,k}^{-1} \cdot \ldots \cdot \mathbf{T}_{1,2}^{-1} \cdot \mathbf{T}_{0,1}^{-1},\end{aligned} \tag{8}$$

where $\mathbf{R}_i(q_i)$ is the homogeneous transformation of the rotation around the z axis of the i_{th} segment in case of revolute joint. In case of prismatic joints, the homogeneous transformation is a translation by q_i along the z axis of the i_{th} segment.

The model matrix in (8) is calculated by taking a chain of transformations, to get to the coordinate system of the k_{th} joint. These transformations ($\mathbf{T}_{i-1,i}^{-1}$) are required, because the mesh points of the segments $\mathbf{x}_R^k(0)$ are given in the robot initial configuration. After that, the actual configuration is taken into account by applying rotations around the rotational axis of the joints ($\mathbf{R}_i(q_i)$), while transforming back to the world coordinate system.

The disadvantage of this method is that if in a given configuration a small object is inside one of the segments then the collision cannot be determined. Because none of the points at the boundary of the segment will satisfy the inequality system of the polyhedron, the small obstacles can cause unexpected behavior.

This problem could be solved by taking points inside the manipulator as well, but it can increase the number of points required to describe one segment, hence it increases the computational expensiveness and the memory usage.

B. *Modeling the obstacles with meshes*

Another solution could be, to model the obstacles with meshes, while the segments of the robot are represented as it is described in (1).

In this case, the polyhedrons need to be transformed to the actual robot configuration with (6). The obstacles can be modeled with their bounding polyhedrons and subdivision can be applied on these meshes, to get more points at the boundaries.

Assume that the size of the objects in the environment does not exceed the size of the entire robot arm, or if it does, the base of the manipulator is not included in that object. Under this condition, the object modeling meshes cannot contain the polyhedrons of all segments. Also, since the polyhedrons describing the robot overlap, it is not possible in this type of modeling to leave a collision undetected due to the small size of the obstacle.

C. *GPU acceleration*

On a CPU, the run time of the previously described method in Algorithm 1 is directly proportional with the number and the size of the meshes and the polyhedrons either.

To accelerate this process, the linear inequalities can be evaluated on a GPU. A GPU can contain thousands of processing units, whose performance is not reaching the performance of a CPU core, but they are supporting the Simple-Instruction-Multiple-Data (SIMD) type of commands. That is, the different iterations in Algorithm 1 can be evaluated simultaneously.

Data movement between the GPU and the CPU is a bottleneck in the performance, so it should be minimized as much as possible. The largest variable in this problem is the array of points in the meshes. To minimize the data transfer, these points at the initial robot configuration can be sent to the GPU memory only once.

After that, these points can be transformed by the GPU itself if needed, only the required $\mathbf{T}_{M_k}$ model matrix has to be transferred to the GPU memory. If the meshes are modeling the static object in the environment then no transformation is needed.

V. Simulation results

The simulations were performed on the Mitsubishi RV-2F-Q six degrees-of-freedom manipulator. The method was executed on the Intel Core i7-7700HQ Processor with a GeForce GTX 1050ti GPU. The algorithm was implemented in the MATLAB environment, while the GPU programming was performed in CUDA.

To compare the presented three methods with each other, three environment configurations were taken. These configurations and the run times for 100 iterations can be seen in Fig. 4.

Because the method described in Section IV-A can have mistakes in case of small obstacles, the other method was used during the simulations, which was presented in Section IV-B. Another advantage of this technique is that because static objects was used, the meshes do not need to be transformed in every iterations. This method was compared to the collision detection method described in Section III.

In all of the test cases the manipulator was modeled with six oriented bounding boxes. In the first scenario, only two object were created, with 1204 mesh points altogether. On the second scenario, 30 boxes were used with 18060 mesh points. And in the last scenario, 10 more complicated polyhedrons were created with 5240 points.

As it can be seen in Fig 4, in every scenario, the GPU accelerated version outperformed the other two methods with approximately two decades of decreasing in the execution time.

If the collision is investigated between polyhedrons with the help of Farkas's Lemma, the execution time is directly proportional with the number of the polyhedrons. An advantage of the GPU accelerated version is that its run time did not increase significantly, even with much more mesh points.

VI. Conclusion

In this paper, a new collision detection method was presented to detect collisions with the obstacles, where polyhedrons and meshes were used to model the robot segments and the environment.

At first, we have shown a method, based on the feasibility checking of some linear inequality system, which was capable to determine if a given configuration is collision-free or not.

Although the previous method did work, it was computationally expensive to use it in a complex system. That is why this method was reshaped to be capable of running on a GPU, which significantly accelerated the collision detection method.

Future work will address, how this method can be used to not just determine the collision-freenes, but calculate the minimal distances between the robot segments and the obstacles. After that, this knowledge can be used in a dynamic collision avoidance algorithm.

Acknowledgement

The research reported in this paper and carried out at the Budapest University of Technology and Economics was supported by the "TKP2020, Institutional Excellence Program" of the National Research Development and Innovation Office in the field of Artificial Intelligence (BME IE-MI-SC TKP2020).

References

[1] R. R. Galin and R. V. Meshcheryakov, "Human-robot interaction efficiency and human-robot collaboration," in *Robotics: Industry 4.0 Issues & New Intelligent Control Paradigms*. Springer, 2020, pp. 55–63.

[2] A. De Luca, A. Albu-Schaffer, S. Haddadin, and G. Hirzinger, "Collision detection and safe reaction with the dlr-iii lightweight manipulator arm," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2006, pp. 1623–1630.

[3] T. Ren, Y. Dong, D. Wu, and K. Chen, "Collision detection and identification for robot manipulators based on extended state observer," *Control Engineering Practice*, vol. 79, pp. 144–153, 2018.

[4] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser *et al.*, "Collision detection for deformable objects," in *Computer graphics forum*, vol. 24, no. 1. Wiley Online Library, 2005, pp. 61–81.

[5] M. Macklin, K. Erleben, M. Müller, N. Chentanez, S. Jeschke, and Z. Corse, "Local optimization for robust signed distance field collision," *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, vol. 3, no. 1, pp. 1–17, 2020.

[6] J. Pan, S. Chitta, and D. Manocha, "Probabilistic collision detection between noisy point clouds using robust classification," in *Robotics Research*. Springer, 2017, pp. 77–94.

[7] J. S. Park, C. Park, and D. Manocha, "Efficient probabilistic collision detection for non-convex shapes," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 1944–1951.

[8] P. Jiménez, F. Thomas, and C. Torras, "3d collision detection: a survey," *Computers & Graphics*, vol. 25, no. 2, pp. 269–285, 2001.

[9] M. W. Spong, S. Hutchinson, M. Vidyasagar *et al.*, *Robot modeling and control*. John Wiley & Sons, Inc., 2006.

[10] C. Landry, M. Gerdts, R. Henrion, and D. Hömberg, "Path-planning with collision avoidance in automotive industry," in *IFIP Conference on System Modeling and Optimization*. Springer, 2011, pp. 102–111.

[11] D. Szabó and E. G. Szádeczky-Kardoss, "Robotic manipulator path-planning: Cost-function approximation with fuzzy inference system," in *2019 24th International Conference on Methods and Models in Automation and Robotics (MMAR)*, 2019, pp. 259–264.

[12] M. K. Berens, I. D. Flintoft, and J. F. Dawson, "Structured mesh generation: Open-source automatic nonuniform mesh generation for fdtd simulation." *IEEE Antennas and Propagation Magazine*, vol. 58, no. 3, pp. 45–55, 2016.