The algorithm for generating the initial trajectory in TrajOpt

Elena Rubleva[1], Konstantin Mironov[1,2], and Aleksandr Panov[2,3]

[1] Moscow Institute of Physics and Technology, Moscow, Russia
[2] AIRI Artificial Intelligence Research Institute, Moscow, Russia
[3] Federal Research Center "Computer Sciences and Control", Moscow, Russia
rubleva.ea@phystech.edu

Abstract. In the current work, we present an algorithm for generating an initial trajectory for use in TrajOpt. Our algorithm enables the generation of high-quality trajectories in an environment with obstacles. Experiments demonstrate that the implementation of the algorithm enhances the efficiency of TrajOpt planning.

Keywords: Mobile Manipulator · Trajectory Optimization · Collision Avoidance.

1 Introduction

Robotic manipulators that act in collaborative human-oriented environments (e.g. office rooms) face a number of challenging tasks. These task may include the motion between the opposite regions of the workspace. Embedded controllers may not provide such a motion in the task space due to singularity effects. Therefore advanced trajectory planning methods are applied to define joint trajectory, which provide the required motion in the task space. Sampling-based methods based on RRT (Rapidly-exploring Random Tree) [1] or PRM (Probabilistic Roadmap Method) [2] are very popular for such planning. Disadvantage of these methods consist in non-optimality non-stability of generated trajectories due to use of random sampling for planning. Contrary optimization-based planners provide smooth and stable solutions. However the collision avoidance for trajectory optimizers is more challenging.

Collision avoidance requirements maybe inserted into trajectory optimizers either as a set of hard constraints or as a set of soft penalties. Popular CHOMP optimizer (Covariant Hamiltonian Optimization for Motion Planning) [3] exploit the first approach. Constraint-based collision-aware optimization require collision-free initial guess. This mean that we primarily plan non-optimal path (e.g. with sampling-based planner) and than optimize it with constraint-based method. Contrary penalty-based TrajOpt (Trajectory Optimization) method [4] may converge to suitable solution even when the initial guess include collision. However, it require mesh of the obstacles, while CHOMP may work with pointclouds or voxel representation. This is because TrajOpt require information about collision depth not only the fact of collision.

1.1 Related Work

Algorithms RRT and PRM are sampling-based algorithms that efficiently explore the search space by randomly sampling configurations to plan paths. CHOMP is an optimization-based planner focusing on generating smooth solutions through constraint-based collision-aware optimization. TrajOpt is another optimization-based method that can converge to solutions even with initial collisions, requiring collision depth information for effective collision avoidance in trajectory planning for robotic manipulators.

Planning the movement of robotic manipulators to bypass obstacles requires the use of algorithms that are able to take into account the environment in which the robot operates. In research, the TrajOpt algorithm is often used as a no-line. Below is an overview of the works that contain certain improvements to the basic TrajOpt algorithm that are significant for the obstacle avoidance task.

Text hints [5] can be used for planning using TrajOpt. They are used to describe the location of obstacles, as well as information about whether the specified obstacle can be touched. For example, a soft toy can be dropped and therefore it can be taken into account when planning the trajectory.

Planning in an environment with dynamic obstacles deserves special attention. In this case, it is necessary to take into account the speed of the obstacles and avoid collisions in the future [6].

Another task of improving planning is to work on smoothing the trajectory. The paper [7] describes a planning method that uses finite Fourier series to represent continuous motion in time and proposes a method that regulates the harmonics of movement of each joint of the manipulator. Collision fields are used to avoid obstacles.

The approach of dividing the task of optimizing motion into 2 subtasks helps to achieve better planning results: the subtask of robot dynamics, the subtask of collision avoidance [8]. Planning using TrajOpt along the collision objects is calculated in the immediate vicinity of them.

To improve planning, approaches based on improvements in the representations of the robot and the environment are proposed. In particular, the paper [9] proposes a trajectory planner for the receding horizon, which uses a combination of a new representation of a set of achievable objects and a signed accurate distance function. During execution, the algorithm uses parameterized trajectories to calculate sets of achievable objects consisting entirely of spheres that overestimate the volume of movement of the robot.

Neural network approaches can be used to improve planning results. The work [10] uses graph neural networks (GNNs) and an attention mechanism to estimate the distance to a collision. In this case, the method is used for planning in conditions when the target is surrounded by obstacles (for example, an object in an open bedside table). And the paper [11] describes the approaches of using Energy Based Models (EBM) neural networks in the tasks of optimizing the trajectory of robotic manipulators. The authors consider priority modeling for optimization using implicit functions. The approach allows you to make the trajectories smoother.

Another neural network approach was presented by the authors of the LeTO method [12]. LeTO uses the CNN and LSTM neural network, which is trained based on human demonstrations, while also adjusting trajectory optimization parameters. LeTO makes it possible to generate smooth trajectories. It has robustness and the ability to deal effectively with abnormal data.

To generate an initial approximation of the trajectory, the authors of the paper [13] use Reinforcement learning (RL). The RL-ITG method presents a learning-based approach for efficiently generating high-quality initial trajectories for redundant manipulators, significantly improving the performance of trajectory optimization (TO) in path-following problems. By leveraging example-guided reinforcement learning and null-space projected imitation reward, RL-ITG demonstrates enhanced optimality, efficiency, and applicability over various setups, addressing the critical challenge of selecting high-quality initial trajectories to drive better optimization outcomes.

The authors of the papers [14] and [15] propose a framework where planners and ML are combined. The problem is solved when the information is partially unavailable or noisy. Bayesian blind learning is used. A potentially unknown environment is being considered. They use feedback – learning with reverse reinforcement (IRL). Plans the movement using Trajopt, then gives the person a trajectory to choose from (the person accepts or rejects). If a person rejects, then in the algorithm he must mark each segment (the volume passed) as good or bad.

Based on the extensive research and developments in the field of robotic manipulator trajectory planning using the TrajOpt algorithm, it is evident that various approaches have been proposed to address critical challenges such as obstacle avoidance, dynamic environments, trajectory smoothing, and the incorporation of neural network methodologies. The works have demonstrated significant advancements, encompassing techniques like the implementation of finite Fourier series, collision fields, graph neural networks, Energy Based Models (EBM), Reinforcement Learning (RL), and Bayesian blind learning. Moreover, the combination of planners and machine learning (ML) has shown promise, especially in scenarios with partially unavailable or noisy information. These innovations collectively contribute to enhancing the performance, efficiency, and adaptability of trajectory planning using TrajOpt, thereby paving the way for more effective and robust solutions in real-world robotic manipulator applications.

Thus, in the field of trajectory planning for manipulators using TrajOpt, numerous approaches have been proposed to address collision avoidance and trajectory smoothing, which contribute to improved performance and adaptability of solutions.

However, many authors of articles point out that TrajOpt is sensitive to the initial trajectory approximation, and the success and speed of the algorithm's calculations depend on this initial trajectory. The main contribution of our work is to propose a neural network algorithm that generates a better initial trajectory than any other classical algorithms. We present a learning-based method that enables the generation of an initial approximation of the robotic manipu-

lator's trajectory, significantly enhancing trajectory optimization performance. Our method is based on reinforcement learning (RL) and allows for trajectory generation based on demonstrations while taking obstacles into account.

In particular, we utilize a Recurrent Neural Network (RNN) to model the complex temporal dependencies inherent in trajectory generation. The RNN's ability to process sequences of observations makes it well-suited for predicting viable trajectories in dynamic environments. By training the RNN on a diverse dataset of trajectory examples, including both successful and challenging configurations, we can effectively leverage its predictive capabilities to produce an initial trajectory that is more aligned with the constraints of the task. This integration not only improves the starting point for TrajOpt but also enhances the overall efficiency and success rates of the trajectory optimization process, leading to faster and more reliable planning outcomes in robotic manipulation tasks.

1.2 Contribution

We present a learning-based method that allows you to generate an initial approximation of the trajectory of the robot manipulator, which significantly improves the performance of trajectory optimization. Our method is based on reinforcement learning (RL), allows you to generate trajectories based on demonstrations, taking into account obstacles.

2 Background

TrajOpt is a trajectory planning algorithm that optimizes robot paths to avoid obstacles. It uses penalties to prevent collisions by adjusting the trajectory to steer clear of objects. It calculates pushing distances to avoid collisions between the robot link and obstacles.

The robot avoids collision at every time step. A robot and obstacles are objects (solids) that should not collide and intersect. To avoid collisions, the TrajOpt algorithm identifies the fact that the link of the robot intersects with an obstacle. If there is an intersection, finds how much the shapes penetrate each other. Assigns a penalty for this robot configuration, which, when the penetration depth changes at the next stages of trajectory calculation, will gradually decrease until it becomes close to zero.

The signed distance is used as the basis for calculating the penalty for collisions.

The collision penalty method can be described as follows:

$$sd_{AB}(\theta) \approx \widehat{n}(P_B - P_A(\theta)),$$

where A and B is objects (solids), T is distance between objects and sd is distance sign sd is the signed distance, θ – time step, $\widehat{n}$ is the normal vector, P_A and P_B – the points of contact of objects.

To determine the penalty, it is necessary to know the distance between the objects (sets). The distance between two sets $A, B \subset R^3$ other than zero for non-intersecting sets, is defined as:

$$dist(A,B) = inf\{\|T\| \mid (T+A) \cap B \neq \varnothing\}$$

To calculate the penalty for collisions, use:

1. Gilbert-Johnson-Kirti algorithm (GJK) to determine the fact of collision between a link of the robot and an obstacle [16];
2. The expanding polyhedron algorithm (EPA) [17] to determine the depth of penetration.

a) Gilbert-Johnson-Kirti algorithm (GJK)

The algorithm is used for calculating the distance between objects and identifies the fact of a collision (collision).

The GJK algorithm is used to define the configuration space of obstacles. For a pair of convex hull (A and B) of their configuration space, the distance will be equal to:

$$d(A,B) = min\{\|x\| : x \in A - B\}$$

Objects A and B intersect if their obstacle configuration space contains the origin of the coordinate system:

$$A \cap B \neq \emptyset \equiv 0 \in A - B$$

GJK uses the Minkowski difference [18]. This algorithm calculate as the difference of coordinates of shapes A and B.

The Minkowski difference of sets A and B is the maximal set C such that $C + B \subset A$. Each vertex of the each shape is subtracted from each vertex of other shape. The coordinates of this difference of sets make up a polyhedron C.

If the Minkowski difference contains the origin, then shapes A and B intersect.

Calculating the Minkowski difference can be time-consuming, especially when checking intersections of shapes with many points. The complexity of the calculation $A - B$ is $O(n^2)$, because you need to iterate over all the vertices of each shapes. The GJK algorithm aims to efficiently determine if the origin is within the Minkowski difference by finding the farthest vertices in a direction to speed up the process. It involves iteratively seeking the extreme vertices and creating a simplex around the origin. This enables quick convergence and collision detection. GJK algorithm handles collision detection, while EPA algorithm deals with determining the separation distance between objects to prevent collisions.

The expanding polyhedron algorithm (EPA)

The EPA algorithm takes the last simplex containing the origin (which was obtained during the implementation of the GJK algorithm) and finds the collision normal, that is, the shortest vector to push the shapes away from each other. EPA is called the expanding polyhedron algorithm because it expands the simplex by adding vertices to it until we find the shortest normal from the face that is on

the original polygon. Simply put, a simplex becomes a polyhedron after we start adding additional vertices to it.

3 Baseline

As a baseline for planning solution, we used the RRTConnect and TrajOpt algorithm. RRTConnect we use from MoveIt framework. It is known that the success of the TrajOpt algorithm depends on the initial trajectory. In this case, the RRTConnect algorithm provides the initial trajectory. And TrajOpt optimizes it. If the trajectory for initialization is not set, then a straight line is taken from the initial to the target position of the manipulator.

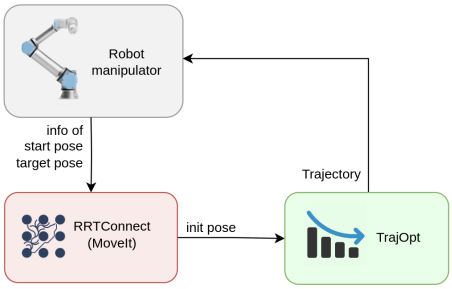

Fig. 1: Architecture

4 Trajectory Planning Model

In this paper, we consider the formal statement of the trajectory planning problem for a manipulator in the configuration space. Our objective is to minimize the functional $F(T)$, defined as follows:

$$\min_T F(T) = \begin{cases} T(0) = q_{\text{start}}, \\ T(1) = \text{N}(q_{\text{start}}, q_{\text{goal}}), \\ \forall t \in [0,1], T(t) \in C_{\text{free}} \setminus M, \\ S(T(t)) = 0 \end{cases}$$

Where:

- C denotes the configuration space of the manipulator. In this space, we distinguish between two subsets: C_{free}, representing configurations without obstacles, and C_{obs}, representing configurations with obstacles. Thus, the relation $C = C_{\text{free}} \cup C_{\text{obs}}$ holds.

- q_{start} is the initial position of the manipulator in the configuration space, while q_{goal} represents the target position of the manipulator in the same space.

- N is a neural network that takes as input the initial configuration q_{start} and the target configuration q_{goal}, generating an initial approximation of the manipulator's trajectory q_{init}.

- M represents the obstacle representation, which is formed by aggregating cubes defined by their coordinates and sizes using the following expression:

$$M = \text{aggregation}_{Z,Y,X}\left(\{\text{cube}(p, s(\|p - p_{\text{init}}\|)) \mid p \in \{p \in C \mid I(p_x, p_y, p_z) = 1\}\}\right)$$

The function $s(d) = s_{\min} + k(d - d_{\min})$ defines the size of the cube based on the distance, where $s_{\min}$ is the minimum cube size, k is the growth coefficient, and $d_{\min}$ is the minimum distance for $s_{\min}$. The function $I(x, y, z)$ takes the value 1 if the point (x, y, z) is within the reachable zone, and 0 otherwise.

- S represents a set of data obtained from sensors, which can be represented as a mapping $S : C \to \{0, 1\}$. Here, $S(q) = 0$ indicates that a given configuration is free from obstacles, while $S(q) = 1$ indicates that the configuration is unavailable due to an obstacle.

- $T : [0, 1] \to C$ is the trajectory, which is a continuous function in the configuration space defining the manipulator's path from the initial configuration to the target.

Thus, the problem is to find a trajectory $T(t)$ that starts from the position q_{start}, reaches the position q_{goal}, and does not intersect or enter the obstacle region M, while ensuring that the sensor data $S(T(t))$ confirm the availability of the path at each step t.

5 Common approach

The architecture of the solution is shown in figure 1.

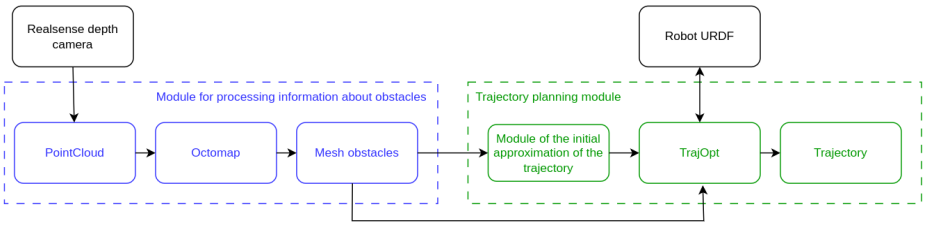

Fig. 2: Architecture

We get information about obstacles from the data of the RealSense depth camera. The camera is mounted on the gripper of the Universal Robots UR5 robot. The obstacle information processing module receives data in the form of a point cloud. Points located beyond the range of the manipulator are filtered out. Next, the point cloud is transformed into an Octomap. The width of the Octomap cube is set manually and depends on the required accuracy.

In the planning module, information about obstacles is received by the initial trajectory initialization module. The initial trajectory initialization module, based on information about an obstacle, predicts points in space through which the robot's trajectory can be laid.

The resulting trajectory is transmitted to the TrajOpt algorithm. The TrajOpt algorithm calculates the final trajectory of the robot based on information about obstacles, the initial trajectory and the description of the robot in URDF format.

6 Suggested improvements to the algorithm

The following enhancements to the TrajOpt algorithm are proposed:

Adjust, refine, or redefine the parameters within the TrajOpt algorithm to enhance its performance for planning the motion of the robotic manipulator in environments with obstacles. The goal is to increase the success rate of calculations and reduce planning time.

We develop a method for finding initial trajectories that avoid collisions with obstacles, thereby ensuring safe navigation.

In this paper we improve the representation of obstacles to strike a balance between the algorithm's performance and the speed of data processing from the robot's sensors.

These improvements aim to optimize the overall effectiveness and efficiency of the trajectory planning process.

7 Generation of the initial trajectory guess

In this section, we present the generation of an initial guess for the trajectory planning problem. The baseline approach utilizes the classical solution known as RRTConnect, while the neural network-based solution employs a Recurrent Neural Network (RNN).

To accomplish this, we develop and train an RNN with Long Short-Term Memory (LSTM) units, which are particularly effective for processing sequential data.

7.1 Network Parameters

- Input: A tensor containing sequences of arbitrary length with 6 features.
- Masking Layer: This layer is included to exclude values equal to 0 from the input data.
- LSTM Layer: Composed of 64 neurons, this layer processes the sequential inputs.
- TimeDistributed Layer: This layer applies a dense layer with 6 outputs across all time steps of the output from the LSTM layer.
- Optimizer: The Adam optimizer is utilized, with Mean Squared Error (MSE) serving as the loss function.

7.2 Training Data

The training data consists of:

- A scene represented as an octomap.
- The initial and target positions of the robot, each specified by 6 joints.
- A trajectory generated using the RRTConnect algorithm.
- The results from the optimization performed by the TrajOpt algorithm.

Additionally, we incorporate Reinforcement Learning (RL) techniques based on the actor-critic model to enhance the trajectory generation process. This combination allows for improved initial guesses that facilitate smoother and more effective navigation through complex environments.

8 Experiments

In this section, we present the results obtained from the Recurrent Neural Network (RNN) used for trajectory prediction.

8.1 Details

- The dataset consists of 390 positions, with each position tested across 10 experiments, resulting in a total of 3900 data points.
- The diversity of positions has been significantly increased to enhance the model's adaptability.
- Additionally, positions that are initially infeasible (where the robot intersects an obstacle in either the start or target configuration) have been included in the dataset.
- Notably, the prediction results from the RNN align closely with the trajectories generated by RRTConnect, demonstrating the effectiveness of the neural network approach.

8.2 Visualization

The results of our experiments are illustrated in Figure 3, showcasing the performance of the RNN in generating viable trajectories.

This analysis underscores the potential of using RNNs for trajectory prediction in robotic manipulation tasks and emphasizes the successful integration of complex, diverse positions in the training process.

8.3 Arm and environment modeling

It is planned to experiment with our planning method on the UR5 robot in the Gazebo simulation environment.

Experiments were conducted in the Robot Operating System (ROS) environment using the UR5 robot model. rViz was used for visualization of the

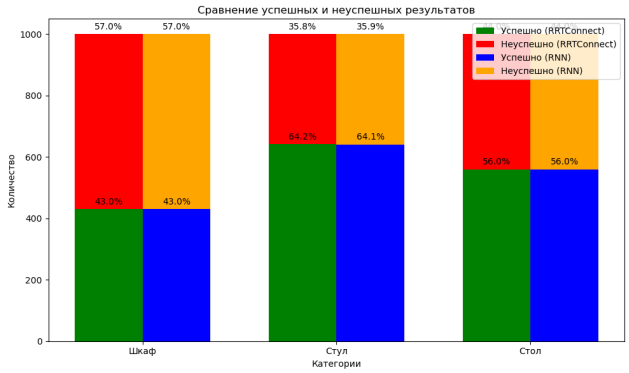

Fig. 3: Results of the RNN experiments (pictire for example)

experiments. Obstacles were taken from a created library (link to library description) or obtained from the Gazebo simulator. Additionally, a PointCloud was generated in Isaac Sim, and testing of the proposed method was performed on it.

As a baseline, we used the RRTConnect algorithm, which calculates the trajectory and feeds it into TrajOpt. Additionally, we conducted testing of our method using only TrajOpt.

The results are presented in the table below. In this experiment, we set the task of planning from a position where the robot's gripper is close to the ground under the table (preparing to pick something up). The target position for the robot was set to a location above the table. An example of the initial and final positions is illustrated in the figure below.

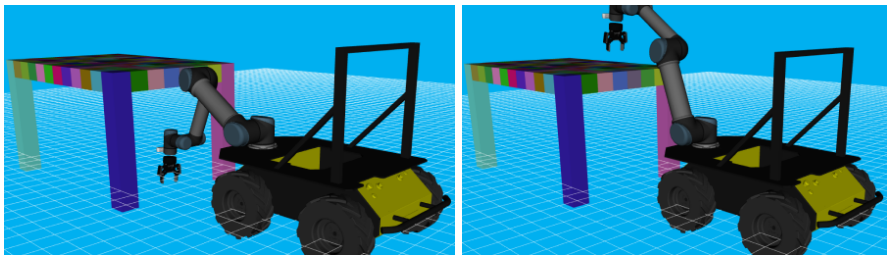

Fig. 4: Initial and final positions of the robot.

The initial and final positions of the robot when planning between the shelves of the cabinet are shown in the figure below.

Table 1: Description of the Experiments with RNN, RL and TrajOpt

Experiment	Success Rate	Time, s
Planning with Table, 10 cm Cube (RRTConnect + TrajOpt)	%	0
Planning with Table, 10 cm Cube (RL + TrajOpt)	%	0
Planning with Table, 10 cm Cube (RNN + TrajOpt)	%	0
Planning with Table, 5 cm Cube (RRTConnect + TrajOpt)	%	0
Planning with Table, 5 cm Cube (RL + TrajOpt)	0%	0
Planning with Table, 5 cm Cube (RNN + TrajOpt)	%	0
Planning with Cabinet, 10 cm Cube (RRTConnect + TrajOpt)	0%	0
Planning with Cabinet, 10 cm Cube (RL + TrajOpt)	0%	0
Planning with Cabinet, 10 cm Cube (RNN + TrajOpt)	0%	0
Planning with Cabinet, 5 cm Cube (RRTConnect + TrajOpt)	0%	0
Planning with Cabinet, 5 cm Cube (RL + TrajOpt)	0%	0
Planning with Cabinet, 5 cm Cube (RNN + TrajOpt)	0%	0
Planning with Chair, 10 cm Cube (RRTConnect + TrajOpt)	%	0
Planning with Chair, 10 cm Cube (RL + TrajOpt)	%	0
Planning with Chair, 10 cm Cube (RNN + TrajOpt)	%	0
Planning with Chair, 5 cm Cube (RRTConnect + TrajOpt)	%	0
Planning with Chair, 5 cm Cube (RL + TrajOpt)	%	0
Planning with Chair, 5 cm Cube (RNN + TrajOpt)	%	0

8.4 Real-world experiments

It is planned to put experiments on a real UR5 robot at the MIPT area for various planning tasks.

9 Conclusion

As a result of the conducted experiments, it was found that the Recurrent Neural Network (RNN) method performs poorly in the trajectory planning task, showing low success rates and significant execution times. The Rapidly-exploring Random Tree Connect (RRTConnect) method displayed average results, managing the task better than RNN, but still not achieving high efficiency. The best results in terms of success and execution time were demonstrated by the Reinforcement Learning (RL) method, although this method is still far from perfect performance in the task. Thus, the results indicate the need for improvements across all approaches, but especially for the RNN and RRTConnect methods, to achieve more optimal solutions in the field of trajectory planning.

References

[1] Steven M. LaValle and Jr. James J. Kuffner. "Randomized Kinodynamic Planning". In: The International Journal of Robotics Research 20.5 (2001), pp. 378–400. DOI: 10.1177/02783640122067453. eprint: https://doi.org/10.1177/02783640122067453. URL: https://doi.org/10.1177/02783640122067453.

[2] L.E. Kavraki et al. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In: IEEE Transactions on Robotics and Automation 12.4 (1996), pp. 566–580. DOI: 10.1109/70.508439.

[3] Nathan Ratliff et al. "CHOMP: Gradient optimization techniques for efficient motion planning". In: 2009 IEEE International Conference on Robotics and Automation. 2009, pp. 489–494. DOI: 10.1109/ROBOT.2009.5152817.

[4] John Schulman et al. "Motion planning with sequential convex optimization and convex collision checking". In: The International Journal of Robotics Research 33 (2014), pp. 1251–1270.

[5] Stephen James Amber Xie1 Youngwoon Lee Pieter Abbeel. "Language-Conditioned Path Planning". In: arXiv preprint arXiv:2308.16893 (2023).

[6] Fang Xu Liu Hongyan Daokui Qu. "Real-Time and Efcient Collision Avoidance Planning Approach for Safe Human-Robot Interaction". In: Journal of Intelligent Robotic Systems 105.93 (2022).

[7] Lu Guodong Feng Yichang Wang Jin. "Optimal Motion Planning using Finite Fourier Series in a Learning-based Collision Field". In: arXiv preprint arXiv:2312.09073 (2023).

[8] Jeffrey Bingham JChanghao Wang and Masayoshi Tomizuka. "Trajectory Splitting: A Distributed Formulation for Collision Avoiding Trajectory Optimization". In: arXiv preprint arXiv:2111.01899 (2021).

[9] Qingyi Chen Jonathan Michaux Adam Li. “Safe Planning for Articulated Robots Using Reachability-based Obstacle Avoidance With Spheres”. In: arXiv preprint arXiv:2402.08857 (2024).

[10] Jinwoo Kim Yeseung Kim and Daehyung Park. “GraphDistNet: A Graph-based Collision-distance Estimator for Gradient-based Trajectory Optimization”. In: IEEE Robotics and Automation Letters (2023).

[11] Alexander Lambert Julen Urain An T. Le. “Learning Implicit Priors for Motion Optimization”. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2022).

[12] Yu She Zhengtong Xu. “LeTO: Learning Constrained Visuomotor Policy with Differentiable Trajectory Optimization”. In: arXiv preprint arXiv:2401.17500 (2024).

[13] Daehyung Park Minsung Yoon Mincheul Kang. “Learning-based Initialization of Trajectory Optimization for Path-following Problems of Redundant Manipulators”. In: IEEE Transactions on Industrial Informatics (2022).

[14] Vaibhav Unhelkar Carlos Quintero-Pena Constantinos Chamzas. “Motion Planning via Bayesian Learning in the Dark”. In: ICRA: Workshop on Machine Learning for Motion Planning (2021).

[15] Zhanyi Sun Carlos Quintero-Pena Constantinos Chamzas. “Human-Guided Motion Planning in Partially Observable Environments”. In: 2022 International Conference on Robotics and Automation (ICRA) (2022).

[16] Keerthi S. Gilbert E. Johnson D. “A Fast Procedure for Computing the Distance between Complex Objects in ThreeDimensional Space”. In: IEEE Journal of Robotics and Automation 4.2 (1988), pp. 193–203.

[17] G van den Bergen. “Proximity Queries and Penetration Depth Computation on 3D Game Objects”. In: In Game Developers Conference(GDC) (2001).

[18] Hugo Hadwiger. “Minkowskische Addition und Subtraktion beliebiger Punktmengen und die Theoreme von Erhard Schmidt”. In: Math. Z. 53.3 (1950), pp. 210–218.