Stabilizing Manipulator Trajectory via Collision-Aware Optimization

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

[1] Moscow Institute of Physics and Technology, 9 Institutsky lane, Dolgoprudny, Moscow 141701, Russia
mironovconst@gmail.com
[2] AIRI Artificial Intelligence Research Institute, 6/2 Presnenskaya Emb., Moscow, Russia
[3] Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences, 44/2 Vavilova St., Moscow 119333, Russia

Abstract. In this work, we develop an optimization-based solution for a manipulation planning among obstacles. This task is particularly challenging for collaborative manipulators when the operations include movements through the singular configurations. Trajectory optimization for this case requires a collision model, which allows converging to a valid solution from an invalid initial guess. By improving the initial trajectory approximation and optimizing collisions with advanced obstacle representation, it significantly enhances trajectory planning accuracy and efficiency. In our work, we propose a method based on the combining of obstacles. We use Octomap as the baseline (obstacles are represented as a set of cubes). This creates the need to calculate each cube when calculating the trajectory. Enlarging cubes to parallelepipeds allows one to obtain a collision-free trajectory faster. We have conducted a set of experiments with Octomap and with the representation of obstacles in the form of enlarged cubes. Experiments have shown that enlarging cubes to parallelepipeds reduces planning time and increases the success rate.

Keywords: Manipulator · Trajectory Optimization · Collision Avoidance · Initializing Trajectory · Octomap

1 Introduction

Robotic manipulators that act in collaborative human-oriented environments (e.g. office rooms) face a number of challenging tasks. These tasks 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 and 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.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024
A. Ronzhin et al. (Eds.): ICR 2024, LNAI 14898, pp. 30–44, 2024.
https://doi.org/10.1007/978-3-031-71360-6_3

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 then 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 not only the fact of collision, but also information about collision depth. Therefore collision modeling for TrajOpt is challenging task, which is of our interest in this work.

1.1 Related Work

RRT [1] and PRM [2] are sampling-based algorithms that efficiently explore the search space by randomly sampling configurations to plan paths. CHOMP [3] is an optimization-based planner focusing on generating smooth solutions through constraint-based collision-aware optimization. TrajOpt [4] 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. Text hints [5] can be used for planning with TrajOpt in order 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]. 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 were 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.

1.2 Contribution

We present a novel method for handling the obstacles: based on the information received from the depth camera, we convert PointCloud to Octomap [16] and then combine Octomap cubes in the direction from the initial trajectory. The initial trajectory is a straight line from the initial position of the gripper work to the target position of the gripper.

At the end, we run the optimization of the TrajOpt with a refined initial approximation trajectory (we use it for this purpose RRTConnect algorithm) near the obstacle.

2 Background

TrajOpt is a trajectory planning algorithm that optimizes robot paths to avoid obstacles. Optimized trajectory it then passed to the manipulator controller, which execute it according to dynamic constraints [17]. 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:

$$sdAB(\theta) \approx \hat{n}(P_B - P_A(\theta)), \tag{1}$$

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, $\hat{n}$ 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 nonintersecting sets, is defined as:

$$dist(A, B) = inf\{\|T\| (T + A) \cap B = \emptyset\}. \tag{2}$$

To calculate the penalty for collisions, use:

- Gilbert-Johnson-Kirti algorithm (GJK) to determine the fact of collision between a link of the robot and an obstacle [18];
- The expanding polyhedron algorithm (EPA) [19] to determine the depth of penetration.

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:

$$d(A, B) = min\{\|x\| : x \in A - B\}. \tag{3}$$

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

$$A \cap B \neq 0 \equiv 0 \in A - B. \tag{4}$$

GJK uses the Minkowski difference [20]. 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 [21] and TrajOpt [4] algorithms. RRTConnect implementation is taken from MoveIt[1] framework (see Fig. 1). 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 and is intrapolated along the entire length.

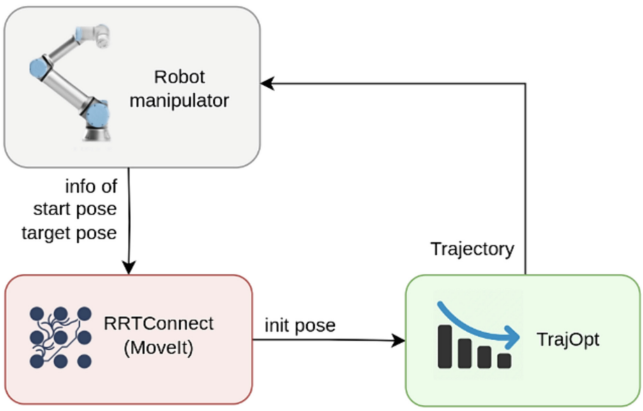

Fig. 1. Tools for trajectory planning and optimization

[1] https://moveit.ros.org/, last accessed 2024/05/31.

For each experiment, one trajectory is calculated in MoveIt and then optimized once using TrajOpt.

4 Proposed Method

We get information about obstacles from the data of the gripper-mounted depth camera. It provide obstacle 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. Next, the cubes are combined into parallelepipeds according to the principle: first, the cubes located next to the initial trajectory are enlarged. The result of the obstacle handling module is fed into the planning module (see Fig. 2).

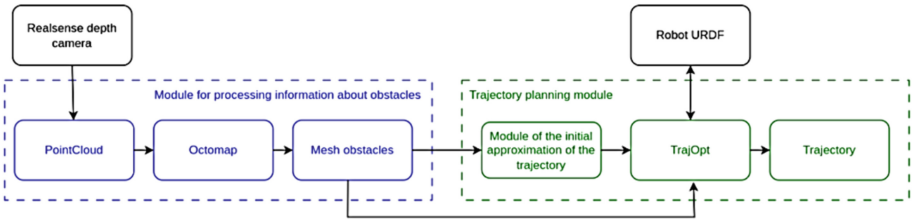

Fig. 2. Architecture of the planning module

In the planning module, information about obstacles is received by the initial trajectory initialization module, that predicts points, through which the robot's trajectory can be laid. The resulting trajectory is transmitted to the TrajOpt algorithm, which optimize the final trajectory of the robot based on obstacle model, nitial trajectory and robot model.

5 Representation of Obstacles

The existing options for representing obstacles (Mesh, Octomap, PointCloud) have a number of disadvantages that reduce the effectiveness of planning the trajectory of a robotic manipulator in an environment with obstacles due to the following reasons:

- reduction of planning accuracy (costs of approximating obstacles to a more convenient form for processing - representation in the form of cubes, ellipsoids, Octomap);
- increase in planning time (meshes require more calculations).

Example of combining cubes is shown in Fig. 3, 4 and 5.

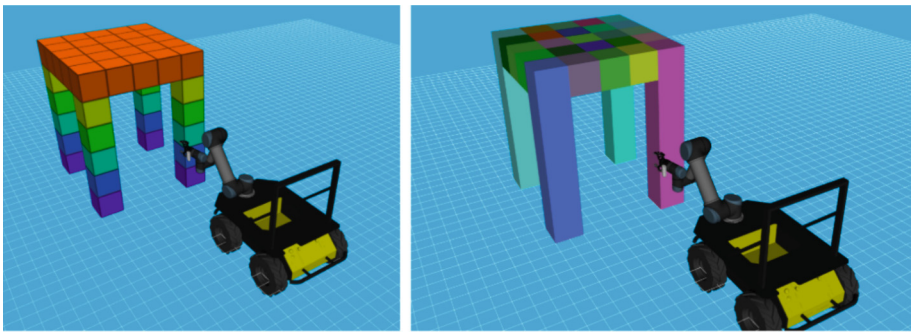

Fig. 3. Combining cubes (right) along the Z axis from Octomap (left)

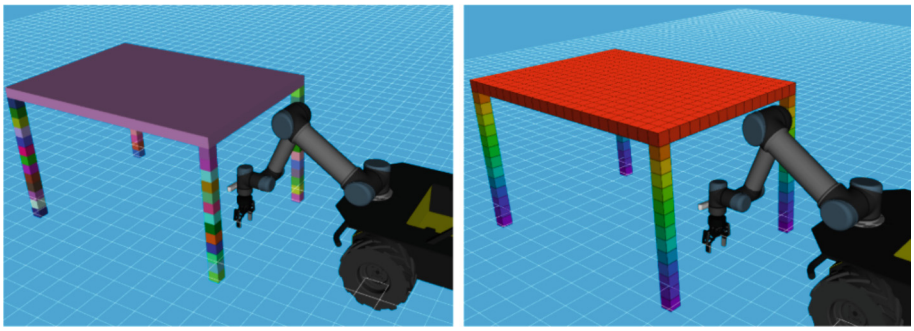

Fig. 4. Combining cubes (left) along the X, Y axises from Octomap (right)

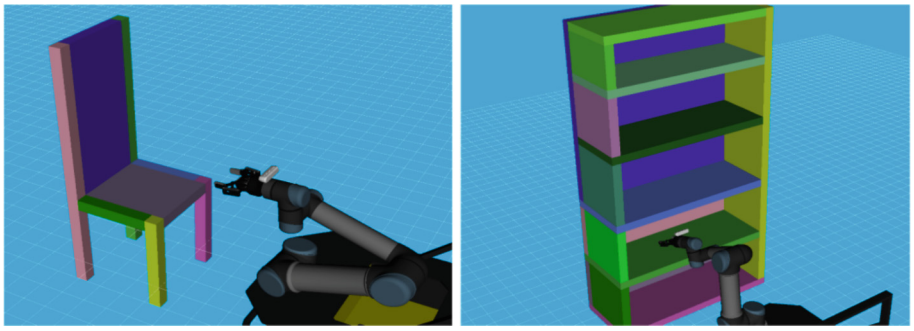

Fig. 5. Combining cubes along the X, Y, Z axises from Octomap for chair (left) and closet (right)

Disadvantages of using existing representations of obstacles:

- meshes have high accuracy due to the high detail of objects, however, its parsing requires a lot of calculations;

- approximating obstacles to cubes and ellipsoids (1 obstacle = 1 cube or 1 ellipsoid) entails a large loss of accuracy, which can give incorrect results when planning (not finding the trajectory where it is, and vice versa);
- Octomap is the most common way to represent obstacles. By specifying different cube sizes, it allows one to achieve different approximation accuracy, but it is difficult to calculate when planning a trajectory.

We propose to improve the method of representing obstacles in the form of Octomap and adapt it for trajectory planning tasks with a manipulator. To do this, you need to filter out data from sensors that are beyond the reach of the robot manipulator. Then convert PointCloud to Octomap and combine the clubs into mesh parallelepipeds.

Combining the entire task into one formal formula, taking into account the operations of filtering, converting and creating meshes:

$$M = agg_{x,z,y}(cube(p,\ s(\|p - p_{init}\|))|p \in p \in PC|I\big(p_x,\ p_z, p_y\big) = 1), \tag{5}$$

where:

- $s(d) = s_{min} + k(d - d_{min})$, $s_{\min}$ – minimum cube size, k is the magnification factor, d_{min} is the minimum distance for smin;
- p_{init} – the initial position of the robot arm.

And I(x, z, y) is:

$$I(x, z, y) = \begin{cases} 1 if nisthepoint(x, y, z) \in thereachzone \\ 0 elsen \end{cases}. \tag{6}$$

We use Octomap as the baseline.

To get information about obstacles, you can use simulator data. An example of a scene in a Gazebo simulator and a picture with the RealSense camera installed on the gripper is shown in Fig. 6. The transformation of data from the simulator camera first into an Octomap, and then into enlarged obstacles is shown in Fig. 7.



Fig. 6. The scene in the Gazebo: data from the camera RealSense (left) and fragment of the scene (right)

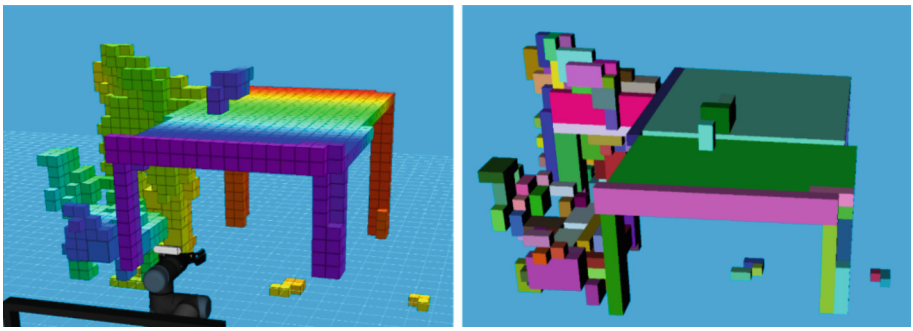

Fig. 7. Octomap for table (left) and combining cubes along the X, Y, Z axises for table (right)

We suggest using a different representation of obstacles instead of Octomap by combining cubes into parallelepipeds. To conduct experiments, an obstacle library was developed, which included such obstacles as: a table, a chair, a basket, a wall, a closet (see Fig. 8).

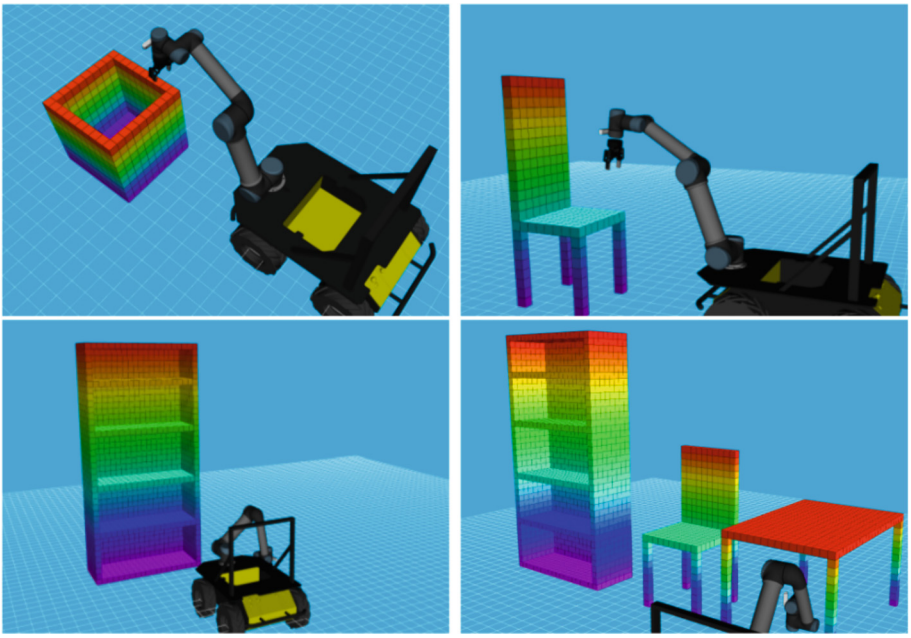

Fig. 8. Combining cubes along the x, y axes

6 Experiments

We conducted the experiments in the Robot Operating System (ROS)[2] environment with the Universal Robots (UR5)[3] robot model, rViz was used to visualize the experiments. The obstacles were taken from the created obstacle library or were obtained from the Gazebo simulator. Also, PointCloud was assembled in Isaac Sim[4] and the proposed method was tested on it.

We used 2 scenarios to work out the planning method. In the first scene there is only a table, the initial position of the gripper robot is under the table and directed to the floor. It is necessary to build a trajectory so that the gripper is above the countertop. An illustration of the initial and target position of the robot is shown in Fig. 9. In this experiment, we set the task of planning from a position where the gripper job is close to the floor under the table (going to take something).

There is a closet on the second stage. The work must be planned between the shelves. An example of such planning is shown in Fig. 10.

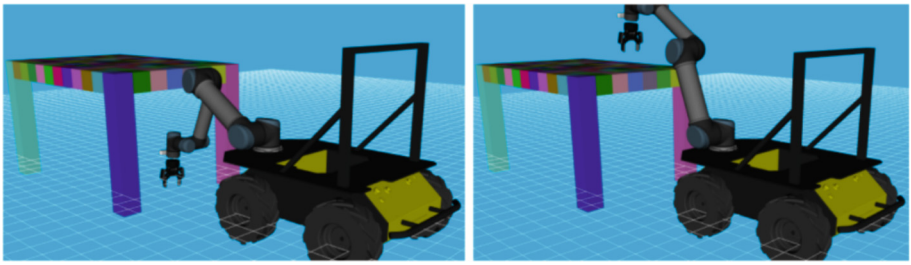

Fig. 9. The initial and target position of the robot

The experiments were carried out under the following conditions:

- The cube size is 5 and 10 cm;
- The Cubes were united and not united;
- Experiments were conducted when the initial trajectory calculated using RRTConnect and was used into the TrajOpt algorithm, and when only TrajOpt was used.

[2] ROS - Robot Operating System Homepage, https://www.ros.org/, last accessed 2024/05/25.
[3] UR5e, Universal Robots Homepage, https://www.universal-robots.com/products/ur5-robot/, last accessed 2024/04/20.
[4] NVIDIA Isaac Sim Homepage, https://developer.nvidia.com/isaac/sim, last accessed 2024/05/20.

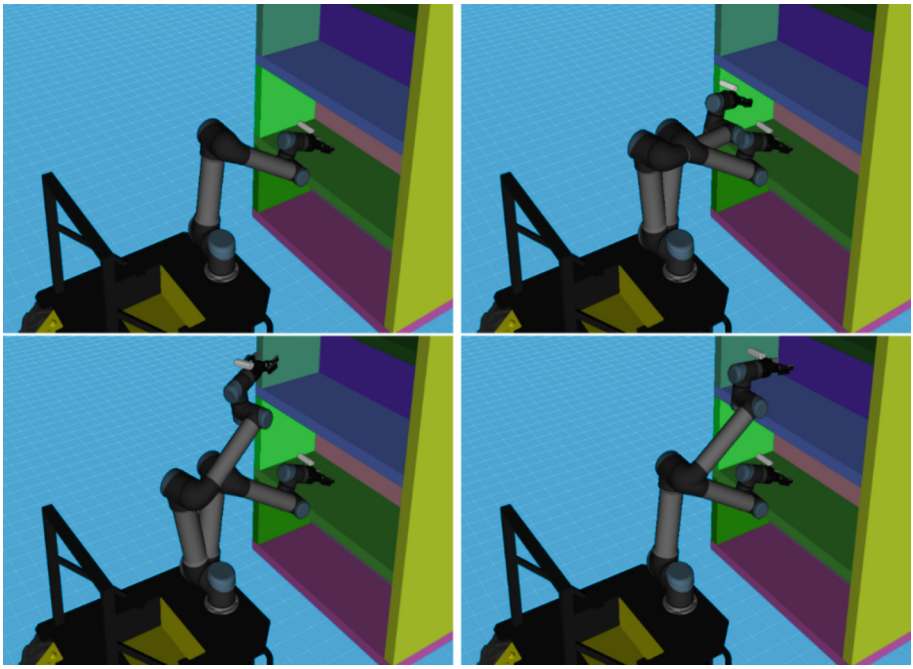

Fig. 10. Planning between cabinet shelves

We have done 20 tests for each situation. The average values for all launches can be seen in the Table 1. For one test case, one RRTConnect trajectory was built, which was optimized once in TrajOpt. The success rate and execution time of the TrajOpt algorithm were measured as metrics (the time it took the algorithm to find a trajectory or generate an error). If RRTConnect was used, then the results of the calculation of the RRTConnect algorithm were submitted to TrajOpt. If only TrajOpt was used, then an interpolated trajectory along the arc from the initial gripper pose to the target gripper pose was taken as the initial trajectory. The length of the initial trajectory in both cases was 11 steps.

The Table 1 shows the results of the experiments. The scheduling time in table is given only for TrajOpt. As can be seen from the table, the enlargement of Octomap cubes allows you to reduce planning time and increase the success rate. For example, when planning between the shelves of the cabinet, when each shelf was represented by one mesh, TrajOpt built a trajectory. And in a similar scene with octomama, he was able to build a trajectory only with RRTConnect. A comparison of the planning time is shown in Fig. 11.

Table 1. Table with a description of the experiments

Scene for experiment	Octomap, success rate	Octomap, time, sec	Cubes are united, success rate	Cubes are united, time, sec
Planning from under the table, cube 10cm, (RRTConnect and TrajOpt)	80%	7.48	80%	4.68
Planning from under the table, cube 10cm, (TrajOpt only)	0%	3.58	0%	2.66
Planning from under the table, cube 5cm, (RRTConnect and TrajOpt)	75%	5.68	80%	4.68
Planning from under the table, cube 5cm, (TrajOpt only)	0%	6.7	0%	2.15
Planning between shelves in closet, 10 cm cube (RRTConnect and TrajOpt)	95%	2.97	100%	2.78
Planning between shelves in closet, 10 cm cube (TrajOpt only)	0%	2.54	100%	1.36
Planning between shelves in closet, 5 cm cube (RRTConnect and TrajOpt)	100%	2.94	100%	2.6
Planning between shelves in closet, 5 cm cube (TrajOpt only)	0%	3.8	100%	1.29

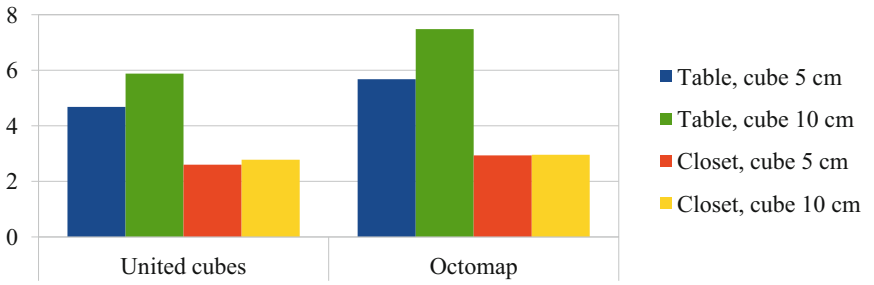

Fig. 11. Planning time, sec

7 Conclusion

In this work, we leveraged the RRTConnect and TrajOpt algorithms as a baseline for trajectory planning. The RRTConnect algorithm, sourced from the MoveIt framework, was utilized to generate the initial trajectory, which was subsequently optimized by the TrajOpt algorithm. The success of the TrajOpt algorithm is highly dependent on the initial trajectory; if not provided, a straight line from the starting to the target position of the manipulator is used.

We obtained obstacle information in the simulator Gazebo using the RealSense depth camera mounted on the UR5 robot's gripper. Obstacles from the created obstacle library were also used for planning. The obstacle data was processed into an Octomap, with a manual setting for the cube width based on the required accuracy. Cubes were then combined into parallelepipeds to enhance trajectory planning efficiency. This processed data was fed into the planning module, where the initial trajectory was initialized and subsequently optimized by the TrajOpt algorithm.

Experiments were conducted in the ROS environment using a UR5 robot model, visualized with rViz. We analyzed various scenarios, including planning under a table and between cabinet shelves, with different cube sizes and combinations. The results revealed that enlarging Octomap cubes reduced planning time and increased success rates.

Specifically, when planning trajectories between cabinet shelves, representing each shelf with a single mesh allowed TrajOpt to successfully generate a trajectory, unlike in similar scenes using Octomap, where RRTConnect was necessary. The experiments demonstrated that integrating different cube sizes and combining them into parallelepipeds significantly improved planning efficiency.

In conclusion, the proposed method of enhancing obstacle representation using Octomap and optimizing it for trajectory planning tasks has proven to be effective. This method can be incorporated into future developments of robotic manipulator trajectory planning systems (e.g. utilizing reinforcement learning [22, 23]), providing increased efficiency and accuracy in environments with obstacles.

Acknowledgments. This work was supported by the Ministry of Science and Higher Education of the Russian Federation under Project 075-15-2024-544.

References

1. LaValle, S., Kuffner, J.: Randomized kinodynamic planning. Int. J. Robot. Res. **20**(5), 378–400 (2001)
2. Kavraki, L., Svestka, P., Latombe, J.-C., Overmars, M.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Trans. Robot. Autom. **12**(4), 566–580 (1996)
3. Ratliff, N., Zucker, M., Bagnell, J., Srinivasa. S.: CHOMP: gradient optimization techniques for efficient motion planning. In: IEEE International Conference on Robotics and Automation, pp. 489–494. IEEE, Kobe (2009)
4. Schulman, J., et al.: Motion planning with sequential convex optimization and convex collision checking. Int. J. Robot. Res. **33**(9), 1251–1270 (2014)
5. James, S., Xie, A., Lee Y., Abbeel P.: Language-conditioned path planning. arXiv preprint arXiv:2308.16893 (2023)
6. Liu, H., et al.: Real-time and efficient collision avoidance planning approach for safe human-robot interaction. J. Intell. Robot. Syst. **105**(4) (2022)
7. Guodong, L., Yichang, F., Jin, W.: Optimal motion planning using finite Fourier series in a learning-based collision field. arXiv preprint arXiv:2312.09073 (2023)
8. Bingham, J., Wang, C., Tomizuka, M.: Trajectory splitting: a distributed formulation for collision avoiding trajectory optimization. arXiv preprint arXiv:2111.01899 (2021)
9. Michaux, J., Li, A., Chen, Q., Zhang, B., Vasudevan, R.: Safe planning for articulated robots using reachability-based obstacle avoidance with spheres. arXiv preprint arXiv:2402.08857 (2024)
10. Kim, Y., Kim, J., Park, D.: GraphDistNet: a graph-based collision-distance estimator for gradient-based trajectory optimization. IEEE Robot. Autom. Lett. **7**(4), 11118–11125 (2023)
11. Urain, J., Le, A., Lambert, A., Chalvatzaki, G., Boots, B., Peters, J.: Learning implicit priors for motion optimization. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2022, pp. 7672–7679 (2022)
12. She, Y., Xu Z.: LeTO: learning constrained visuomotor policy with differentiable trajectory optimization. arXiv preprint arXiv:2401.17500 (2024)
13. Yoon, M., Kang, M., Park, D., Yoon, S.-E.: Learning-based initialization of trajectory optimization for path-following problems of redundant manipulators. In: IEEE International Conference on Robotics and Automation, ICRA 2023, pp. 9686–9692. IEEE, London (2023)
14. Quintero-Pena, C., Chamzas, C., Unhelkar, V., Kavraki, L.: Motion planning via bayesian learning in the dark. In: ICRA: Workshop on Machine Learning for Motion Planning (2021)
15. Quintero-Pena, C., Chamzas, C., Sun, Z., Unhelkar, V., Kavraki, L.: Human-guided motion planning in partially observable environments. In: International Conference on Robotics and Automation, ICRA 2022, pp. 7226–7232. IEEE, Philadelphia (2022)
16. Hornung, A., Wurm, K., Bennewitz, M., Stachniss, C., Burgard, W.: OctoMap: an efficient probabilistic 3D mapping framework based on octrees. Auton. Robot. **34**, 189–206 (2013)
17. Mironov, K., Mambetov, R., Panov, A., Pushkarev, D.: Model predictive control with torque constraints for velocity-driven robotic manipulator. In: 20th International Conference on Advanced Robotics, ICAR 2021, pp. 107–112. IEEE, Ljubljana (2021)
18. Keerthi, S., Gilbert, E., Johnson, D.: A fast procedure for computing the distance between complex objects in three dimensional space. IEEE J. Robot. Autom. **4**, 193–203 (1988)
19. Van den Bergen, G.: Proximity queries and penetration depth computation on 3D game objects. In: Proceedings of the Game Developers Conference, GDC 2001, vol. 170. Hilton Head, USA (2001)
20. Hadwiger, H.: Minkowskische addition und subtraktion beliebiger punktmengen und die theoreme von erhard schmidt. Math. Z. **53**(3), 210–218 (1950)

21. Kuffner, J., LaValle, S.: RRT-connect: an efficient approach to single-query path planning. In: Proceedings 2000 ICRA, Millennium Conference, IEEE International Conference on Robotics and Automation, Symposia Proceedings, vol. 2, pp. 995–1001. IEEE, San Francisco (2000)
22. Gorodetsky, A., Mironov, K., Pushkarev, D., Panov, A.: Goal and force switching policy for DMP-based manipulation. In: Ronzhin, A., Meshcheryakov, R., Xiantong, Z. (eds.) ICR 2022. LNCS, vol. 13719, pp. 105–116. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-23609-9_10
23. Gorodetskiy, A., Mironov, K., Panov, A.: Model-based policy optimization with neural differential equations for robotic arm control. In: Ronzhin, A., Sadigov, A., Meshcheryakov, R. (eds.) ICR 2023. LNCS, vol. 14214, pp. 258–266. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-43111-1_23