2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

Grasping Trajectory Optimization with Point Clouds

Yu Xiang, Sai Haneesh Allu, Rohith Peddi, Tyler Summers, Vibhav Gogate

***Abstract*— We introduce a new trajectory optimization method for robotic grasping based on a point-cloud representation of robots and task spaces. In our method, robots are represented by 3D points on their link surfaces. The task space of a robot is represented by a point cloud that can be obtained from depth sensors. Using the point-cloud representation, goal reaching in grasping can be formulated as point matching, while collision avoidance can be efficiently achieved by querying the signed distance values of the robot points in the signed distance field of the scene points. Consequently, a constrained nonlinear optimization problem is formulated to solve the joint motion and grasp planning problem. The advantage of our method is that the point-cloud representation is general to be used with any robot in any environment. We demonstrate the effectiveness of our method by performing experiments on a tabletop scene and a shelf scene for grasping with a Fetch mobile manipulator and a Franka Panda arm.** [1]

I. Introduction

In robot manipulation, planning a robot trajectory to grasp an object is a fundamental research problem. The problem is challenging since it requires motion planning to avoid obstacles in the task space and grasp planning to decide how to grasp a target object. Traditionally, the motion planning problem and the grasp planning problem are tackled separately. Motion planning approaches focus on finding a collision-free path to reach a given end-effector goal. For example, sampling-based motion planning methods such as Rapidly exploring Random Trees (RRTs) [1], [2], [3] and Fast Marching Tree (FMT) [4] find robot trajectories by incrementally building configuration space filling trees through directed sampling. Optimization-based motion planning methods [5], [6], [7], [8] solve optimization problems to find robot trajectories that minimize some loss functions and obey certain constraints, such as joint limits.

Since these motion planning algorithms need to have a given goal, they cannot be applied directly to robot grasping unless a grasping goal is given. On the other hand, grasp planning methods such as GraspIt! [9], 6D GraspNet [10] and SE(3)-DiffusionFields [11] aim to synthesize grasps of robot grippers given 3D models or 3D point clouds of objects. These methods focus on planning the poses of robot grippers to grasp various objects. However, they do not consider the motion of the robotic arm to reach the planned grasps.

Yu Xiang, Sai Haneesh Allu, Rohith Peddi and Vibhav Gogate are with the Department of Computer Science, and Tyler Summers is with the Department of Mechanical Engineering, University of Texas at Dallas, Richardson, TX 75080, USA {yu.xiang, saiHaneesh.allu, rohith.peddi, tyler.summers, vibhav.gogate}@utdallas.edu

[1]Code and videos for the project are available at https://irvlutd.github.io/GraspTrajOpt

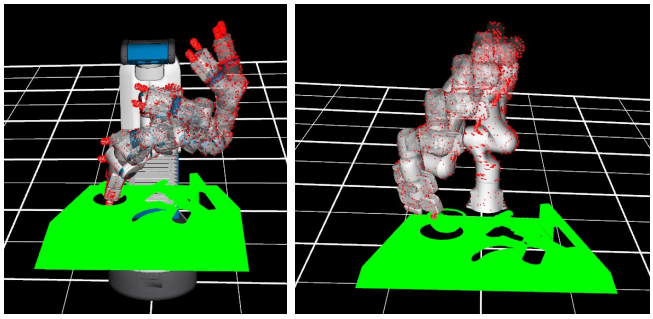


Fig. 1. We represent robots and the task space with point clouds, and solve a trajectory optimization problem for joint motion and grasp planning.

Combining grasp planning and motion planning can address the robot grasping problem. A straightforward approach is first to utilize a grasp planning method to generate grasps of a target object and then employ a motion planning method to plan a robot trajectory to reach one of the grasps. A naive way is to loop over all the planned grasps until the motion planner finds a collision-free path to reach one of the grasps. This naive approach is complete, that is, as long as there is one plausible grasp from the grasp planner, the method can find a path to reach it. However, it is very slow, especially when the number of planned grasps is large. Therefore, a number of approaches are proposed to address the problem of joint motion and grasp planning.

Similarly to motion planning methods, these joint motion and grasp planning methods can be categorized into sampling-based and optimization-based ones. Sampling-based methods [12], [13], [14] bias a motion planner to sample nodes that are closer to better grasps, and the grasps are synthesized online. The main limitation of these approaches is that the online synthesized grasps may not be accurate enough for precise grasping, especially for objects with complicated shapes. To overcome this limitation, several goal-set-based trajectory optimization methods are proposed [15], [16]. These methods first utilize an offline grasp planner to generate grasps of objects, where well-designed grasp planners can be used, such as grasps synthesized from physics simulation [9], [17]. These generated grasps are treated as goals in a goal set. The joint motion and grasp planner optimizes a collision-free trajectory that can reach one of the goals in the goal set. The goal set introduces a constraint on the last configuration of the robot trajectory. Using high-quality grasps as goals, these approaches can handle various objects in grasping.

In this work, motivated by the goal-set-based trajectory optimization framework for joint motion and grasp planning, we introduce a new trajectory optimization method

for robotic grasping. Compared to previous methods [15], [16], our method has the following advantages. First, we introduce a point-cloud representation of robots and task spaces for goal reaching and obstacle avoidance. Point clouds of robots are generated using the 3D meshes of the robot links, whereas point clouds of the task space can be obtained from depth sensors such as RGB-D cameras. Figure 1 shows the point cloud representation with the planned trajectories of a Fetch robot and a Franka Panda arm. This representation is general and can be used with any robot and any task space. Second, we formulate a constrained trajectory optimization problem using point-cloud representation for joint motion and grasp planning. Given a set of grasping goals, solving the optimization problem generates a trajectory to reach one of the goals that minimizes the objective function subject to certain constraints, such as joint limits. Instead of converting the constrained optimization problem into an unconstrained one and solving with first-order gradient descent-based techniques as in [15], [16], we utilize the Interior Point OPTimizer (Ipopt) [18] to solve the large-scale nonlinear optimization problem for trajectory planning, which can find better solutions compared to first-order solvers. Finally, we empirically verify our method on two robot grasping environments in the PyBullet simulator [19], i.e., a tabletop scene and a shelf scene, and demonstrate a significant improvement over the OMG-Planner [16] in terms of metrics on grasping success and collision avoidance. In addition, we conducted real-world grasping experiments according to the SceneReplica benchmark [20]. Our method improves over a sampling-based baseline in real-world experiments.

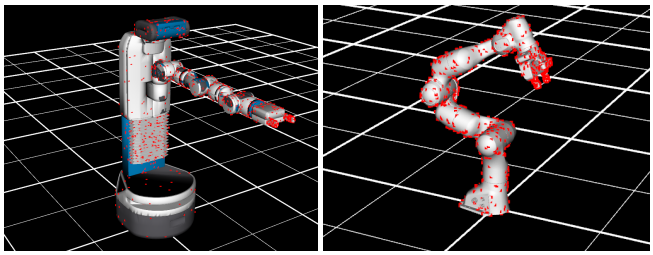
II. Related Work

A. Manipulation Trajectory Optimization

Trajectory optimization techniques have been successfully applied to robot manipulation. Early work such as CHOMP [5] and related methods [15], [21] optimize a cost functional using covariant gradient descent. STOMP [22] uses stochastic sampling of noisy trajectories to optimize nondifferentiable costs. TrajOpt [6] solves a sequential quadratic program, while GPMP2 [7] formulates the problem as inference on a factor graph and finds the maximum a posteriori trajectory by solving a nonlinear least-squares problem. More recently, various trajectory optimization methods have been proposed to solve specific manipulation problems. For example, TORM [23] is introduced to follow given end-effector paths. [24] solves a trajectory optimization problem for the manipulation of deformable objects. [25] solves a whole-body trajectory optimization for mobile manipulation. The advantage of trajectory optimization lies in its flexibility in introducing different cost functions and constraints for various problems. In this work, we solve a trajectory optimization problem for joint motion and grasp planning in robotic grasping.

B. Joint Motion and Grasp Planning

Traditionally, arm motion planning and grasp planning are tackled separately, which can result in suboptimal grasping trajectories. Since jointly optimizing trajectories and grasps is challenging, several approaches are proposed to solve a goal-constrained trajectory optimization problem for joint motion and grasp planning [15], [16], [26], where grasps from a grasp planner such as GraspIt! [9] are used as goals. For example, [15] projects the robot configuration of the last time step in the goal set during trajectory optimization. OMG-Planner [16] iterates between goal selection and trajectory optimization based on CHOMP. Recently, SE(3)-DiffusionFields [11] learns a cost function for grasp planning based on a diffusion model and then solved a joint optimization problem for grasp and motion planning. Unlike these methods, we introduce a cost function for goal reaching using our point-cloud representation and solve a constrained optimization problem for joint motion and grasp planning.



(a) A Fetch Mobile Manipulator (b) A Franka Panda Arm

Fig. 2. Surface points (red points in the figure) are sampled as a representation for robots.

III. Method

A. A Point-Cloud Representation for Robots and Task Spaces

In robot motion planning, the goal is to generate a robot trajectory to reach a goal location while avoiding obstacles in the task space. The geometric representation of robots and the task space is a critical component of robot motion generation. A natural choice is to use 3D meshes of robots and objects in the task space. However, the limitation of using 3D meshes is that we cannot always obtain 3D meshes of objects, and collision checking between meshes is expensive. Another choice is to approximate robot links and obstacles in the task space with 3D shape primitives such as spheres, boxes, or cylinders [5], [27]. Using 3D shape primitives simplifies collision checking, but results in inaccurate collision checking, where motion plans can be conservative. In this work, we utilize a simple geometric representation of objects and the task space, i.e., point clouds, for robot motion planning based on trajectory optimization.

Given a robot description using the Unified Robotics Description Format (URDF), each robot link has an associated 3D mesh model. To obtain a point-cloud representation of the robot, we simply sample 3D points from the vertices of the 3D meshes of the links. Figure 2 shows two examples of a Fetch mobile manipulator and a Franka Panda arm with their 3D points sampled, respectively. The number of points for each link is a parameter to set. Using more points requires more computation in goal reaching and obstacle avoidance, but it can achieve more accurate collision checking. We simply sample 100 points for each link in our experiments.

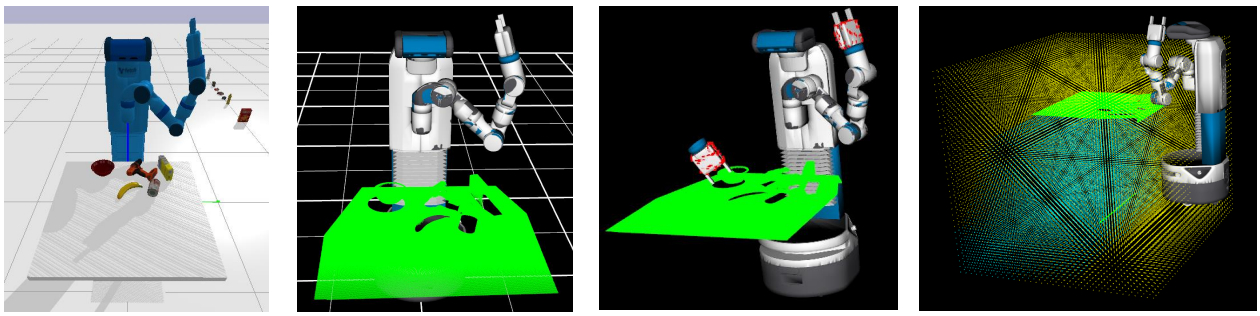


Fig. 3. (a) A tabletop scene for grasping with a Fetch robot. (b) A 3D point cloud of the scene computed using a depth image from the camera on the robot. (c) Reaching a grasping goal can be formulated as matching 3D points on the robot gripper. (d) Visualization of the signed distance field of the task space. Cyan points are with negative distances, and yellow points are with positive distances.

For objects in the robot task space, we cannot obtain 3D models of them if we want the robot to work in arbitrary environments. Therefore, we rely on depth sensing to obtain a point-cloud representation of the task space. By equipping a RGB-D camera with a robot, the robot can capture a depth image of the scene. Depth pixels can be back-projected to the camera frame using the intrinsic parameters of the camera. Then we can obtain a point cloud of the scene. Given the camera extrinsic parameters, i.e., 3D rotation and 3D translation of the camera in the robot base frame, the point cloud can be transformed into the robot base frame. Figure 3(a) shows a tabletop scene and a Fetch robot in the PyBullet simulator, and Figure 3(b) illustrates the computed point cloud using a depth image captured by the robot camera. Since RGB-D cameras are commonly used in robotic applications, using 3D scene points makes our approach generalizable to various scenarios. Next, we describe how to use the point-cloud representation in our grasping trajectory optimization method.

B. Point Cloud-based Cost Function for Goal Reaching

In grasping trajectory optimization, we need to generate a trajectory for a robot from its current joint configuration to a goal configuration for grasping a target object. The task-space goal is defined as an end-effector configuration to grasp the target object. For two-finger grippers, a goal can be simplified to be a homogeneous transformation $\mathbf{T}_g = (\mathbf{R}_g, \mathbf{t}_g) \in \mathbb{SE}(3)$, where $\mathbf{R}_g$ and $\mathbf{t}_g$ are the 3D rotation and the 3D translation of the gripper link with respect to the robot base frame, respectively.

In our method, we optimize for a trajectory that is discretized into T time steps. The trajectory is parameterized by T joint positions $\mathcal{Q} = (\mathbf{q}_1, \dots, \mathbf{q}_T)$, where $\mathbf{q}_i, \in \mathbb{R}^n$ for $i = 1, \dots, T$, and n is the degree of freedom of the robot. The last configuration of the trajectory $\mathbf{q}_T$ must reach the goal $\mathbf{T}_g$ in the task space. We can use forward kinematics to compute the end-effector pose of the robot at time step T: $\mathbf{T}(\mathbf{q}_T) = (\mathbf{R}_T, \mathbf{t}_T) \in \mathbb{SE}(3)$. We wish to define a cost function $c_{\text{goal}}(\mathbf{T}(\mathbf{q}_T), \mathbf{T}_g)$ to measure the distance between the gripper pose of the robot at the time step T and the grasping goal. Consequently, minimizing this cost function can find a robot configuration to reach the goal.

Usually, the cost function is defined based on the distance between the two 3D rotations $(\mathbf{R}_T, \mathbf{R}_g)$ and the distance between the two 3D translations $(\mathbf{t}_T, \mathbf{t}_g)$. However, a weight must be adjusted to balance the two distances. Motivated by work on 6D object pose estimation [28], we utilize the point matching loss function as our cost function for goal reaching. Let $\mathcal{E} = \{\mathbf{x}_i\}_{i=1}^m$ be a set of m 3D points on the end-effector of the robot (see Figure 3(c)). Our cost function for goal reaching is defined as

$$c_{\text{goal}}(\mathbf{T}(\mathbf{q}_T), \mathbf{T}_g) = \sum_{i=1}^{m} \|(\mathbf{R}_T \mathbf{x}_i + \mathbf{t}_T) - (\mathbf{R}_g \mathbf{x}_i + \mathbf{t}_g)\|^2, \quad (1)$$

which minimizes the distance between two sets of point clouds undergone two homogeneous transformations. The advantage of using this cost function is that it eliminates the need to use a hyperparameter to balance rotation and translation. This cost function can also be generalized to grippers with high degrees of freedom, such as multi-finger grippers. Note that $\mathbf{T}(\mathbf{q}_T)$ is a function of $\mathbf{q}_T$ in the loss function according to forward kinematics.

C. Point Cloud-based Cost Function for Collision Avoidance

In addition to reaching the grasping goal, another requirement in robotic grasping is to avoid obstacles in the task space. We hope that the robot will not hit any object before grasping the target. For the example in Figure 3, the robot should avoid hitting the table and objects on the table during grasping. Instead of using 3D meshes or 3D shape primitives to represent obstacles, our method only has access to a point cloud of the scene. Therefore, we propose to compute a Signed Distance Field (SDF) of the robot task space using the point cloud for collision avoidance.

First, the extent of the task space is determined by the extent of the point cloud in the task space, where we add some margin to the point cloud space. Second, the SDF is constructed by densely sampling a 3D grid within the extent of the task space. The resolution of the grid is a parameter that can be tuned as a trade-off between computational efficiency and accuracy of collision checking. Third, we compute the signed distance value for each vertex of the 3D grid, which is approximated by the distance between the vertex and the closest point in the point cloud of the

scene. The sign of the distance is determined by checking if the vertex is behind the point cloud or not. Specifically, we project the vertex to the depth image using the camera parameters and compare the depth values of the vertex and the projected pixel to obtain the distance sign. Figure 3(d) illustrates the SDF of the task space, where the cyan vertices have negative distances. Finally, using the computed SDF, we can check the collision between the robot and the scene by checking the signed distance values of the 3D points on the surface of the robot (Figure 2) in the task space.

In addition, we can define a cost function for collision avoidance using the SDF. For each joint configuration in the robot trajectory $\mathbf{q}_t \in \mathbb{R}^n, t = 1, \ldots, T$, let $\mathbf{x}(\mathbf{q}_t) \in \mathbb{R}^3$ be a surface point on the robot transformed into the task space according to the joint configuration $\mathbf{q}_t$ using forward kinematics. Then we can define a cost function for the 3D point $\mathbf{x}(\mathbf{q}_t)$ as in CHOMP [5]:

$$c_{\text{collision}}(\mathbf{x}) = \begin{cases} -d(\mathbf{x}) + \frac{1}{2}\varepsilon & \text{if } d(\mathbf{x}) < 0 \\ \frac{1}{2\varepsilon}(d(\mathbf{x}) - \varepsilon)^2 & \text{if } 0 \leq d(\mathbf{x}) \leq \varepsilon \\ 0 & \text{otherwise} \end{cases}, \tag{2}$$

where ε is a margin parameter and $d(\mathbf{x})$ is the signed distance of the 3D point. When the signed distance $d(\mathbf{x})$ is greater than ε, there is no cost in collision. Note that in our implementation, the SDF is precomputed using a 3D grid to speed up computation. Therefore, we simply find the voxel in which the 3D point $\mathbf{x}$ falls and use the signed distance value of the voxel as $d(\mathbf{x})$.

D. Constrained Trajectory Optimization for Joint Motion and Grasp Planning

With the designed cost functions for goal reaching and collision avoidance, we describe our trajectory optimization framework for joint motion and grasp planning. The task of a robot is to grasp a target object in a cluttered scene. We assume that there exists a grasp planner that can be used to synthesize grasps of the target. For example, in model-based grasping, GraspIt! [9] can be used to synthesize grasps given the 3D model of the target object. In model-free grasping, learning-based approaches such as 6DGraspNet [10] or Contact-GraspNet [29] can be used to synthesize grasps of the target object given the segmented point cloud of the target. We denote the set of synthesized grasps as a goal set $\mathcal{G} = \{\mathbf{T}_i\}_{i=1}^K$, where $\mathbf{T}_i \in \mathbb{SE}(3)$ is a homogeneous transformation of the robot gripper and K is the number of planned grasps. Our goal is to find a collision-free trajectory for the robot to reach one of the grasps.

We optimize for a trajectory that is discretized into T time steps. The trajectory is parameterized by T joint positions $\mathcal{Q} = (\mathbf{q}_1, \ldots, \mathbf{q}_T)$ and T joint velocities $\dot{\mathcal{Q}} = (\dot{\mathbf{q}}_1, \ldots, \dot{\mathbf{q}}_T)$, where $\mathbf{q}_i, \dot{\mathbf{q}}_i \in \mathbb{R}^n$ for $i = 1, \ldots, T$. Therefore, we optimize both the joint positions and the joint velocities in our method. Intuitively, we want to have the last joint position $\mathbf{q}_T$ reach one of the grasps in the goal set $\mathcal{G}$. Meanwhile, the trajectory should be collision-free and subject to constraints of the robot dynamics and joint limits. Formally, we solve the following constrained optimization problem to find the trajectory:

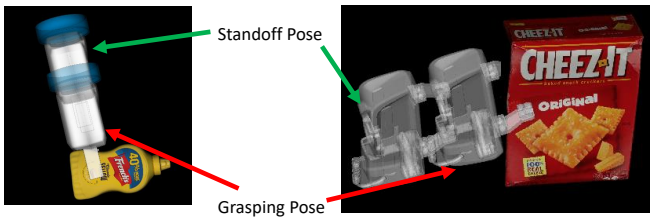


Fig. 4. Illustration of the grasping pose and the standoff pose for grasping.

$$\arg\min_{\mathcal{Q},\dot{\mathcal{Q}}} \Big(\min_{i=1}^{K} \left(c_{\text{goal}}(\mathbf{T}(\mathbf{q}_T), \mathbf{T}_i) + c_{\text{standoff}}(\mathbf{T}(\mathbf{q}_{T-\delta}), \mathbf{T}_i\mathbf{T}_\Delta) \right) + \lambda_1 \sum_{t=1}^{T} c_{\text{collision}}(\mathbf{q}_t) + \lambda_2 \sum_{t=1}^{T} \|\dot{\mathbf{q}}_t\|^2 \Big) \tag{3}$$

$$\text{s.t.,} \mathbf{q}_1 = \mathbf{q}_0 \tag{4}$$

$$\dot{\mathbf{q}}_1 = \mathbf{0}, \dot{\mathbf{q}}_T = \mathbf{0} \tag{5}$$

$$\mathbf{q}_{t+1} = \mathbf{q}_t + \dot{\mathbf{q}}_t dt, t = 1, \ldots, T-1 \tag{6}$$

$$\mathbf{q}_l \leq \mathbf{q}_t \leq \mathbf{q}_u, t = 1, \ldots, T \tag{7}$$

$$\dot{\mathbf{q}}_l \leq \dot{\mathbf{q}}_t \leq \dot{\mathbf{q}}_u, t = 1, \ldots, T, \tag{8}$$

where we minimize an objective function of $\mathcal{Q}$ and $\dot{\mathcal{Q}}$ subject to a set of constraints. Note that the objective function computes the minimum cost among all the grasping goals in the goal set $\mathcal{G}$. Consequently, solving the optimization problem will select the best goal from the goal set.

First, the term $c_{\text{goal}}(\mathbf{T}(\mathbf{q}_T), \mathbf{T}_i)$ is the goal reaching cost described in Eq. (1) for the ith goal $\mathbf{T}_i$ in the goal set, where forward kinematics is used to compute the gripper pose given the robot configuration at the last time step $\mathbf{q}_T$. Second, in addition to reaching the goal in the last step, we introduce a cost term $c_{\text{standoff}}(\mathbf{T}(\mathbf{q}_{T-\delta}), \mathbf{T}_i\mathbf{T}_\Delta)$ to ensure that the robot reaches a standoff pose for grasping before the goal. The standoff pose $\mathbf{T}_i\mathbf{T}_\Delta$ is computed by a displacement $\mathbf{T}_\Delta \in \mathbb{SE}(3)$ of the grasping pose $\mathbf{T}_i$ along the forward axis of the gripper as illustrated in Figure 4. The main reason of introducing the standoff pose is because optimizing the trajectory directly to reach the grasping pose may result in a collision between the robot and the target object. In these cases, the robot will knock down the target object and cannot grasp it. Adding the standoff pose in the trajectory optimization makes the problem simpler. In our objective function, we require that the robot gripper pose $\mathbf{T}(\mathbf{q}_{T-\delta})$ at time step $T - \delta$ to reach the standoff pose, where δ is a parameter to set. Finally, the objective function contains a cost term for collision avoidance and a cost term to penalize large velocities, where λ_1 and λ_2 are two weights to balance the costs. The collision cost for the time step t is defined as

$$c_{\text{collision}}(\mathbf{q}_t) = \sum_{i=1}^{M} c_{\text{collision}}(\mathbf{x}_i(\mathbf{q}_t)), \tag{9}$$

where $\mathbf{x}_i(\mathbf{q}_t) \in \mathbb{R}^3$ is a 3D point on the robot at the robot configuration $\mathbf{q}_t$ and M is the total number of points on the

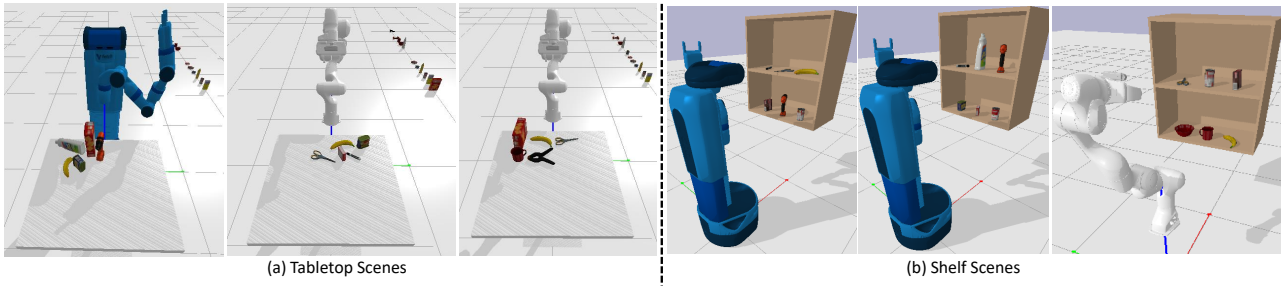


Fig. 5. Examples of (a) tabletop scenes and (b) shelf scenes for grasping in PyBullet.

robot. The collision cost is computed according to Eq. (2) using our SDF representation.

Next, we describe the constraints in the optimization problem. 1) $\mathbf{q}_1 = \mathbf{q}_0$, where $\mathbf{q}_0$ denotes the current configuration of the robot. This constraint ensures that the trajectory starts from the current configuration of the robot. 2) $\dot{\mathbf{q}}_1 = \mathbf{0}, \dot{\mathbf{q}}_T = \mathbf{0}$ ensure that the starting velocity and the ending velocity of the robot are zero. 3) $\mathbf{q}_{t+1} = \mathbf{q}_t + \dot{\mathbf{q}}_t dt$ ensures that the robot state follows the kinematics of the robot, where dt is the time interval between two time steps. 4) The last two constraints in Eqs. (7) and (8) ensure that the joint positions and the joint velocities are within the lower bounds $(\mathbf{q}_l, \dot{\mathbf{q}}_l)$ and upper bounds $(\mathbf{q}_u, \dot{\mathbf{q}}_u)$.

E. Initialization for Grasping Trajectory Optimization

The optimization problem in Eq. (3) is a large-scale constrained nonlinear programming problem. For example, a Franka panda arm has $n = 7$ DOFs. If we set the number of time steps of the trajectory $T = 50$, the optimization problem has $7 \times 2 \times 50 = 700$ variables. We utilize the Interior Point OPTimizer (Ipopt) [18] interfaced with the CasADi framework [30] to solve it. Ipopt can only find local solutions that are sensitive to the initialization of the variables. To obtain a good local solution and speed up the optimization, we use the following strategy to initiate the optimization. 1) Given a set of grasping poses $\mathcal{G} = \{\mathbf{T}_i\}_{i=1}^K$ of a target, we first filter out grasps that are in-collision with other objects in the scene. This collision checking can be achieved by checking the signed distance values of the 3D points on the robot gripper of a given pose as described in Section III-C. 2) For the remaining grasps, we check if an inverse kinematics (IK) solution exists. We solve a simplified optimization problem to find an IK solution of a grasping goal $\mathbf{T}_\text{g}$:

$$\arg\min_{\mathbf{q}_T} \; c_\text{goal}(\mathbf{T}(\mathbf{q}_T), \mathbf{T}_\text{g}) \tag{10}$$

$$\text{s.t.,} \mathbf{q}_l \leq \mathbf{q}_T \leq \mathbf{q}_u, \tag{11}$$

where we use $\mathbf{q}_T$ to denote the variable in IK, and the objective function is the point matching cost function defined in Eq. (1). After finding a local solution $\mathbf{q}_T^*$, we compute the pose error between $\mathbf{T}(\mathbf{q}_T^*)$ and the goal $\mathbf{T}_g$ using a rotation error and a translation error. If both errors are smaller than some pre-defined thresholds, we claim that an IK solution is found. Otherwise, there is no IK solution for $\mathbf{T}_g$. In this way, we can filter out grasps without IK solutions. 3) For each remaining grasp with an IK solution, we interpolate a trajectory of the robot from the current configuration of the robot to the IK configuration. We then compute the collision cost of the trajectory $\sum_{t=1}^T c_\text{collision}(\mathbf{q}_t)$ to rank these trajectories. 4) Finally, we initialize the optimization with the trajectory that has the minimum collision cost. In the case of tie-breaking, e.g., multiple non-collision trajectories, we use the trajectory whose last configuration is closer to the current configuration of the robot. We empirically found that the above initialization process can speed up the convergence of the optimization to find a good local solution.

IV. Experiments

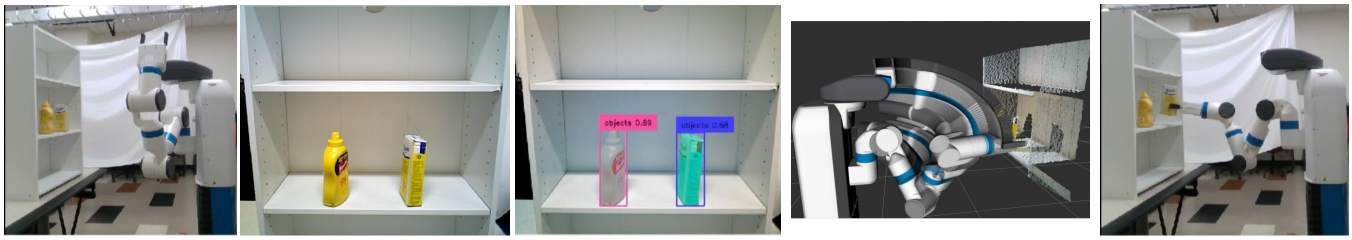
We conducted experiments on 6DoF robotic grasping to evaluate our method in both simulation and in the real world. Two types of scenes are used for evaluation: a tabletop scene and a shelf scene as illustrated in Figure 5 in the Pybullet simulator [19]. In these scenes, 16 YCB objects [31] are used for grasping. The objects in the tabletop scenes are arranged according to the SceneReplica benchmark [20], and we sample object locations for the shelf scenes with 6 objects in each scene. Two robots, i.e., a Fetch mobile manipulator and a Franka Panda arm, are used for evaluation. The main evaluation metric is the success rate of grasping. If an object is successfully lifted by the robot, we count it as a success. In addition, we evaluate collisions during grasping.

A. Implementation Details

First, grasps of the 16 YCB objects are generated using GraspIt! [9], with 100 grasps for each object. Therefore, the size of the goal set is 100. Second, the trajectory optimization is implemented based on the OpTaS library [32], which provides an interface to Ipopt solver using the CasADi framework [30]. Third, the hyper-parameters in the method are set as follows. For each robot link, we sample 100 surface points. The margin $\varepsilon = 0.02$ in computing the collision cost (Eq. (2)). The grid resolution of the signed distance field is 5cm. In the optimization problem Eq. (3), $\lambda_1 = 10$, $\lambda_2 = 0.01$ and $\delta = 10$. The standoff pose for grasping is set as 10cm and 20cm from the grasping pose for the tabletop scenes and the shelf scenes, respectively. The number of time steps is $T = 50$, and the time span of a trajectory is set to 10 seconds. Therefore, $dt = 0.2$ in Eq. (6).

B. The Effect of Point Matching for Goal Reaching

We evaluated the effectiveness of our point cloud-based representation for goal reaching. We solve the inverse kine-



(a) A shelf grasping scene (b) Robot view (c) Object segmentation (d) Grasping trajectory optimization (e) Trajectory execution

Fig. 6. Illustration of the model-free grasping in the real world.

TABLE I

COMPARISON BETWEEN DIFFERENT LOSS FUNCTIONS FOR IK. COUNT IS THE TOTAL NUMBER OF GRASPS TESTED FOR IK. THE NUMBERS OF FOUND IK SOLUTIONS ARE PRESENTED FOR THREE LOSS FUNCTIONS.

	Tabletop (success ↑)				Shelf (success ↑)			
	Tabletop (success ↑ / collision ↓)				Shelf (success ↑ / collision ↓)			
Robot	Count	Point matching	Quaternion	Euler angle	Count	Point matching	Quaternion	Euler angle
Fetch	18,758	**9,665**	9,600	8,288	10,917	**2,911**	2,824	2,364
Panda	20,000	**16,397**	15,905	10,596	12,000	**3,925**	3,867	3,172

TABLE II

STATISTICS OF GRASPING EXPERIMENTS IN THE PYBULLET SIMULATOR

	Tabletop (success ↑ / collision ↓)				Shelf (success ↑ / collision ↓)			
Object	Count	Ours Fetch	Ours Panda	OMG [16] Panda	Count	Ours Fetch	Ours Panda	OMG [16] Panda
cracker box	12	7 / 0	7 / 0	5 / 2	6	3 / 0	2 / 0	4 / 0
sugar box	10	10 / 0	10 / 0	10 / 0	5	4 / 0	5 / 0	4 / 0
tomato soup can	14	13 / 2	14 / 0	12 / 2	7	7 / 1	2 / 1	2 / 4
mustard bottle	14	12 / 0	11 / 0	11 / 0	5	3 / 0	4 / 0	4 / 0
tuna fish can	12	0 / 5	0 / 3	0 / 12	6	5 / 5	0 / 0	0 / 6
pudding box	10	7 / 0	6 / 0	6 / 2	7	5 / 1	4 / 0	2 / 3
gelatin box	14	11 / 0	10 / 0	4 / 2	7	6 / 1	1 / 1	2 / 3
potted meat can	14	10 / 0	14 / 0	12 / 0	10	8 / 0	10 / 0	6 / 1
banana	14	12 / 6	11 / 4	9 / 4	7	3 / 4	1 / 2	1 / 7
bleach cleanser	10	9 / 0	5 / 0	8 / 0	7	5 / 0	6 / 0	4 / 4
bowl	14	11 / 2	9 / 0	8 / 1	11	8 / 3	8 / 2	2 / 7
mug	10	8 / 1	4 / 1	6 / 0	10	3 / 4	6 / 2	5 / 4
power drill	14	14 / 2	12 / 0	12 / 0	7	4 / 0	2 / 1	1 / 4
scissors	14	2 / 0	1 / 6	1 / 13	11	5 / 7	1 / 5	0 / 10
large marker	12	1 / 3	4 / 5	4 / 9	5	3 / 2	0 / 3	0 / 5
extra large clamp	12	5 / 6	2 / 5	4 / 11	9	6 / 2	1 / 2	1 / 9
ALL	200	**132** / 27	120 / **24**	112 / 58	120	**78** / 30	53 / **19**	38 / 67

matics optimization problem in Eq. (10) with three different cost functions and compare their performance. The first one is the point-matching cost function in Eq. (1) to measure the difference between two transformations $\mathbf{T}_T$ and $\mathbf{T}_g$. For the other two cost functions, we use quaternions $(\tilde{\mathbf{q}}_T, \tilde{\mathbf{q}}_g)$ and Euler angles $(\mathbf{e}_T, \mathbf{e}_g)$ to represent the 3D rotations, and compute costs for rotation and translation separately:

$$c_{\text{goal}}^{\text{quat}}(\mathbf{T}_T, \mathbf{T}_g) = \|\mathbf{t}_T - \mathbf{t}_g\|^2 + 1 - (\tilde{\mathbf{q}}_T \cdot \tilde{\mathbf{q}}_g)^2, \quad (12)$$

$$c_{\text{goal}}^{\text{euler}}(\mathbf{T}_T, \mathbf{T}_g) = \|\mathbf{t}_T - \mathbf{t}_g\|^2 + \|\mathbf{e}_T - \mathbf{e}_g\|^2, \quad (13)$$

where the distance between two quaternions measures the angular distance between the two rotations.

Using the three cost functions, we solve IK for each grasp of each object in the tabletop scenes and the shelf scenes. Table I presents the statistics of this experiment, where we count the number of successful IK solutions among all the trials. We consider an IK solution to be found after optimization if the translation error is less than 1 cm and the rotation error is less than 5 degrees. From the table, we can see that using the point matching cost function finds the maximum number of IK solutions, which validates the effectiveness our point-cloud based representation. Using distances between Euler angles is not a good choice due to the discontinuity between $-\pi$ and π.

C. Simulation Results

The results of our grasping experiments in PyBullet are presented in Table II, where we compare our approach to the OMG-Planner [16]. The OMG-Planner is a trajectory optimization method based on first-order gradient descent for joint motion and grasp planning. It alternates between goal selection and fixed-goal trajectory optimization. Grasps of the 16 YCB objects are generated using GraspIt! [9]. In the simulation, we query the object poses directly and then transform the grasps according to the object poses. We evaluate the number of successful grasps and the number of collisions during grasping. Using our point-cloud representation, we treat a grasping trajectory as in collision if there are 5 surface points of the robot with negative signed distances.

For the table, we can see that 1) our method improves over the OMG-Planner in both the tabletop scenes and the shelf scenes. Our method achieves higher grasping success rates and lower collision rates. The main advantage of our method is that we solve a constrained nonlinear optimization problem with an advanced solver (Ipopt) compared to a gradient descent-based optimization. In addition, our point-cloud representation enables more accurate goal reaching and collision avoidance. 2) The Fetch robot achieves higher success rates compared to the Panda robot, largely due to its greater reachability and wider gripper. We cannot run the OMG-Planner for the Fetch robot since its implementation is tightly coupled with the Panda robot. In contrast, our implementation can be easily applied to different robots, where it only requires an URDF of a robot as input. 3) Some objects are more difficult to grasp. These are small or flat objects such as the tuna fish can, the scissors, the large marker, and the extra large clamp. Nonprehensile grasping strategies might be needed to grasp these objects successfully, which can be explored in future work.

D. Model-free Grasping in the Real World

Lastly, we conduct grasping experiments in the real world to evaluate our trajectory optimization method. We consider the task of model-free grasping, where we do not have 3D models of objects for perception and motion planning. Model-free grasping is applicable to diverse environments, and our approach does not rely on 3D object models. Figure 6 illustrates the perception, planning, and control pipeline for model-free grasping.

TABLE III

COMPARISON BETWEEN OUR GRASPING TRAJECTORY OPTIMIZATION (GTO) AND THE OMPL PLANNING IN [20] FOR MODEL-FREE GRASPING.

Method #	Perception	Grasp Planning	Motion Planning	Control	Ordering	Pick-and-Place Success	Grasping Success
Model-free Grasping							
1	MSMFormer [33]	Contact-graspnet [29] + Top-down	OMPL [34]	MoveIt	Near-to-far	57 / 100	65 / 100
2	MSMFormer [33]	Contact-graspnet [29] + Top-down	**GTO (Ours)**	MoveIt	Near-to-far	**65 / 100**	**71 / 100**

Fig. 7. Examples of real-world grasping: (a) tabletop scenes and (b) shelf scenes

TABLE IV

STATISTICS OF OUR GRASPING EXPERIMENTS FOR EACH YCB OBJECT. S: #PICK-AND-PLACE SUCCESS, P_EF: #PERCEPTION FAILURE, P_LF: #PLANNING FAILURE, EF: #EXECUTION FAILURE

Object	Count	Method 1 (OMPL-based)				Method 2 (GTO-based Ours)			
		S	P_EF	P_LF	EF	S	P_EF	P_LF	EF
Order: Near-to-Far									
cracker box	6	4	1	-	1	4	-	-	2
sugar box	5	5	-	-	-	5	-	-	-
tomato soup can	7	2	2	3	-	4	1	1	1
mustard bottle	7	6	-	1	-	2	1	3	1
tuna fish can	6	5	1	-	-	6	-	-	-
pudding box	5	4	1	-	-	5	-	-	-
gelatin box	7	6	-	1	-	7	-	-	-
potted meat can	7	5	2	-	-	2	1	4	-
banana	7	6	-	-	1	7	-	-	-
bleach cleanser	5	-	1	2	2	2	1	-	2
bowl	7	6	-	-	1	7	-	-	-
mug	5	2	-	2	1	2	-	3	-
scissors	7	-	2	2	3	3	3	-	1
power drill	7	3	3	-	1	2	3	1	1
large marker	6	1	2	2	1	3	1	2	-
extra large clamp	6	2	1	2	1	4	-	2	-
ALL	100	57	16	15	12	65	11	16	8

We utilized the MSMFormer [33] to segment unseen objects in an input RGB-D image for tabletop scenes. For shelf scenes, we found that MSMFormer cannot successfully segment objects in the shelf since it is not trained with similar scenes. Therefore, we used Grounding DINO [35] with text prompt "objects" to detect generic objects, and then used SAM [36] to segment objects inside the bounding boxes from Grounding DINO. To synthesize grasps for a target object, we used Contact-GraspNet [29], which takes a segmented point cloud of an object as input and generates grasping poses of a parallel jaw gripper. These planned grasps are treated as goals in the goal set for joint motion and grasp planning. To execute a planned trajectory on a real robot, we also need to generate accelerations of the robot joints on the trajectory. Since our method does not solve for joint accelerations, we apply the path parameterization method [37] to reparameterize the planned trajectory.

We compare our method with an OMPL [34]-based planning baseline in the SceneReplica benchmark [20]. This baseline algorithm simply loops over all goals in the goal set and checks if there is a collision-free motion plan to reach a goal. The comparison results are presented in Table III. Our method achieves a better grasping success rate and a better pick-and-place success rate. Detailed evaluation statistics for each YCB object are presented in Table IV, where we classify pick-and-place failures into perception failures, planning failures, and execution failures. A detailed description of these failure types can be found in [20]. Most failures are due to errors in object segmentation, grasp planning, and grasping goal selection. Because stable grasp is critical for pick-and-place success. By solving the trajectory optimization problem, our method benefits from better goal selection compared to the baseline algorithm. Figure 7 shows some examples of successful grasping in the real world. Grasping videos can be found on the project page and in the supplementary material.

E. Planning Time

Our approach has demonstrated a significant improvement in planning efficiency over the OMPL-based baseline in the experiments conducted using the SceneReplica Benchmark [20]. On average, our method achieves a planning time of 15.4 seconds, which includes the computation time for the grasp collision checking, the IK checking, and the trajectory optimization. However, the OMPL-based baseline takes 45.6 seconds to find a grasp trajectory for a target object. In contrast, the OMG-Planner achieves 3.2 seconds planning time by solving parallel IKs and using GPUs for acceleration. We consider speeding up our method for future work.

V. CONCLUSION AND DISCUSSION

We introduce a new trajectory optimization method for joint motion and grasp planning. The core component of our method is a point cloud-based representation for robots and task spaces. This representation is generalizable to different robots and different environments. We formulate goal reaching and collision avoidance in the trajectory optimization using the point-cloud representation. By solving a constrained nonlinear optimization problem using the Ipopt solver, our method can generate robot trajectories for grasping. Experiments are conducted in simulation and in the real world to demonstrate the effectiveness of our method.

One limitation of our method is that trajectory optimization is slow when relying on an external solver. Future work includes speeding up the optimization. One direction

is to explore using GPUs for parallel computing. Another direction is to explore model predictive control with our point-cloud representation for robotic grasping. To further improve the grasp success rate, a grasp planner that considers force closure or grasp stability will be helpful.

Acknowledgement. This work was supported in part by the DARPA Perceptually-enabled Task Guidance (PTG) Program under contract number HR00112220005 and the Sony Research Award Program. The work of T. Summers was supported by the United States Air Force Office of Scientific Research under Grant FA9550-23-1-0424 and the National Science Foundation under Grant ECCS-2047040.

REFERENCES

[1] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Technical Report. Computer Science Department, Iowa State University*, 1998.

[2] J. Kuffner and S. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2, 2000, pp. 995–1001.

[3] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research (IJRR)*, vol. 30, no. 7, pp. 846–894, 2011.

[4] L. Janson, E. Schmerling, A. Clark, and M. Pavone, "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions," *The International Journal of Robotics Research (IJRR)*, vol. 34, no. 7, pp. 883–921, 2015.

[5] N. Ratliff, M. Zucker, A. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[6] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research (IJRR)*, vol. 33, no. 9, pp. 1251–1270, 2014.

[7] M. Mukadam, J. Dong, X. Yan, F. Dellaert, and B. Boots, "Continuous-time gaussian process motion planning via probabilistic inference," *The International Journal of Robotics Research (IJRR)*, vol. 37, no. 11, pp. 1319–1340, 2018.

[8] T. Stouraitis, L. Yan, J. Moura, M. Gienger, and S. Vijayakumar, "Multi-mode trajectory optimization for impact-aware manipulation," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 9425–9432.

[9] A. Miller and P. Allen, "Graspit! A versatile simulator for robotic grasping," *IEEE Robotics & Automation Magazine*, vol. 11, no. 4, pp. 110–122, 2004.

[10] A. Mousavian, C. Eppner, and D. Fox, "6-DOF GraspNet: Variational grasp generation for object manipulation," in *International Conference on Computer Vision (ICCV)*, 2019, pp. 2901–2910.

[11] J. Urain, N. Funk, J. Peters, and G. Chalvatzaki, "Se (3)-diffusionfields: Learning smooth cost functions for joint grasp and motion optimization through diffusion," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 5923–5930.

[12] N. Vahrenkamp, M. Do, T. Asfour, and R. Dillmann, "Integrated grasp and motion planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2010, pp. 2883–2888.

[13] J. Fontanals, B.-A. Dang-Vu, O. Porges, J. Rosell, and M. Roa, "Integrated grasp and motion planning using independent contact regions," in *IEEE-RAS International Conference on Humanoid Robots*, 2014, pp. 887–893.

[14] J. Haustein, K. Hang, and D. Kragic, "Integrating motion and hierarchical fingertip grasp planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 3439–3446.

[15] A. Dragan, N. Ratliff, and S. Srinivasa, "Manipulation planning with goal sets using constrained trajectory optimization," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 4582–4588.

[16] L. Wang, Y. Xiang, and D. Fox, "Manipulation trajectory optimization with online grasp synthesis and selection," *arXiv preprint arXiv:1911.10280*, 2019.

[17] C. Eppner, A. Mousavian, and D. Fox, "A billion ways to grasp: An evaluation of grasp sampling schemes on a dense, physics-based grasp data set," in *International Symposium on Robotics Research (ISRR)*, 2019.

[18] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical programming*, vol. 106, pp. 25–57, 2006.

[19] "PyBullet," https://pybullet.org/wordpress/.

[20] N. Khargonkar, S. H. Allu, Y. Lu, B. Prabhakaran, Y. Xiang, *et al.*, "Scenereplica: Benchmarking real-world robot manipulation by creating reproducible scenes," *arXiv preprint arXiv:2306.15620*, 2023.

[21] A. Dragan, G. Gordon, and S. Srinivasa, "Learning from experience in manipulation planning: Setting the right goals," in *International Symposium on Robotics Research (ISRR)*, 2011.

[22] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 4569–4574.

[23] M. Kang, H. Shin, D. Kim, and S.-E. Yoon, "Torm: Fast and accurate trajectory optimization of redundant manipulator given an end-effector path," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 9417–9424.

[24] S. Jin, D. Romeres, A. Ragunathan, D. K. Jha, and M. Tomizuka, "Trajectory optimization for manipulation of deformable objects: Assembly of belt drive units," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 10 002–10 008.

[25] M. Spahn, B. Brito, and J. Alonso-Mora, "Coupled mobile manipulation via trajectory optimization with free space decomposition," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 12 759–12 765.

[26] J. Ichnowski, M. Danielczuk, J. Xu, V. Satish, and K. Goldberg, "Gomp: Grasp-optimized motion planning for bin picking," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 5270–5277.

[27] B. Sundaralingam, S. K. S. Hari, A. Fishman, C. Garrett, K. Van Wyk, V. Blukis, A. Millane, H. Oleynikova, A. Handa, F. Ramos, *et al.*, "Curobo: Parallelized collision-free robot motion generation," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 8112–8119.

[28] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes," *arXiv preprint arXiv:1711.00199*, 2017.

[29] M. Sundermeyer, A. Mousavian, R. Triebel, and D. Fox, "Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 13 438–13 444.

[30] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[31] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. Dollar, "Benchmarking in manipulation research: The YCB object and model set and benchmarking protocols," *IEEE Robotics and Automation Magazine*, vol. 22, no. 3, pp. 36–52, 2015.

[32] C. E. Mower, J. Moura, N. Z. Behabadi, S. Vijayakumar, T. Vercauteren, and C. Bergeles, "Optas: An optimization-based task specification library for trajectory optimization and model predictive control," *arXiv preprint arXiv:2301.13512*, 2023.

[33] Y. Lu, Y. Chen, N. Ruozzi, and Y. Xiang, "Mean shift mask transformer for unseen object instance segmentation," *arXiv preprint arXiv:2211.11679*, 2022.

[34] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012.

[35] S. Liu, Z. Zeng, T. Ren, F. Li, H. Zhang, J. Yang, C. Li, J. Yang, H. Su, J. Zhu, *et al.*, "Grounding dino: Marrying dino with grounded pre-training for open-set object detection," *arXiv preprint arXiv:2303.05499*, 2023.

[36] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, *et al.*, "Segment anything," *arXiv preprint arXiv:2304.02643*, 2023.

[37] H. Pham and Q.-C. Pham, "A new approach to time-optimal path parameterization based on reachability analysis," *IEEE Transactions on Robotics*, vol. 34, no. 3, pp. 645–659, 2018.