CONFIDENTIAL. Limited circulation. For review only.

Geometric Reachability Analysis for Grasp Planning in Cluttered Scenes for Varying End-Effectors

Vahid Azizi[1], Andrew Kimmel[1], Kostas Bekris[1], Mubbasir Kapadia[1]

***Abstract*— This paper presents a reachability analysis in cluttered scenes based on computational geometry principles for identifying a complete set of subsurfaces on objects, which allow an end-effector to approach and grasp the corresponding object. The proposed methodology builds on top of the concept of a *motion constraint graph* (MCG), which graphically represents the capabilities of end-effectors as a set of constraints. This representation efficiently encodes the continuous space of possible end-effector configurations. Types of constraints that can be expressed through the MCG include: (a) End-effector constraints that consider contact point geometry and end-effector kinematic; as well as (b) collision constraints, between objects in the scene and the end-effector. Then, for an arbitrary scene, a constraint satisfaction problem is defined so as to compute the set of reachable subsurfaces that permit valid grasps. The resulting set of graspable subsurfaces can be used to support grasp planning, or can be used to efficiently prune grasps from a predefined database or it could be used to select the right type of end-effector in a complex scene when we have access to multiple alternatives. The approach is applicable to any kind of scene geometry (arbitrarily cluttered scenes, curved surfaces), and complex end-effectors with multiple degrees of freedom. Simulated experiments indicate that the proposed geometric reachability analysis increases the success rate of grasp planning and consequently motion planning processes, while also reducing total online planning time, at the cost of a small amount of precomputation.**

I. Introduction

Emerging automation application will involve less structured environments, where robotic equipment may need to interface with a large variety of objects in unpredictable configurations. Picking and manipulating objects with varying geometries in cluttered scenes is, however, a challenging problem for robotic systems. One aspect of this challenge corresponds to perception issues, another relates to the design and development of effective robotic end-effectors, while another corresponds to grasp and motion planning. The focus of this work is on the last component and relates to the properties of different types of end-effectors. In particular, algorithms for grasp planning often have difficulty dealing with highly cluttered or complex scenes, such as the one shown in Fig. 1. This work seeks to mitigate the computational overhead of grasp planning by quickly analyzing the environment geometry by taking advantage of computational geometry primitives and identify the subsurfaces on the objects' geometry in the scene, which afford grasps for a given end-effector.

[1]Vahid Azizi, Andrew Kimmel, Kostas Bekris, and Mubbasir Kapadia are with the Department of Computer Science, Rutgers University, NJ, USA {vahid.azizi, andrew.kimmel, kostas.bekris, mubbasir.kapadia} @cs.rutgers.edu

Fig. 1: A cluttered tabletop scene.

A straightforward way to approach grasp planning in the related literature is to build offline a "grasping database" for each target object type and end-effector [3], [6]. Such databases contain a discrete set of feasible grasps, which can also be evaluated in terms of their physical robustness given appropriate metrics [19]. Online, these grasps are used to define arm states with the aid of an inverse kinematic solver. If the corresponding arm state is reachable with a collision-free trajectory for the robotic arm, then the object can be grasped. Nevertheless, some critical challenges arise in this context:

- The database efficacy depends on its resolution, which implies a sacrifice in completeness for efficiency.
- Time and space complexity increase with the complexity of objects and with clutter.
- A database is needed for each end-effector/object pair.

In cluttered scenarios, as in Fig. 1, the desired object could be heavily occluded by other objects, resulting in a narrow range of viable grasps. In certain automation setups, it may be undesirable for the robot to move other objects in the scene. Thus, a grasp database may not provide any collision-free, reachable grasp, causing the planner to terminate without a solution. Ideally, the grasp generation method would provide a rich set of viable grasps for the motion planner to consider.

Furthermore, in automation a variety of end-effectors may be available, where each one has different capabilities and allows to pick up a variety of items. This can further complicate, however, grasping reasoning as the decision of which end-effector to use may also depend on the scene configuration. Thus, the robot needs to quickly determine which end-effector can be used given the current scene and minimize the number of grasps that need to be considered until a solution is found.

In this context, this paper proposes a method for precomputing grasp contact semantics, or *graspable surfaces*, for a given set of end-effectors and objects, which can be effectively utilized online in novel scenes. The proposed method abstracts the capabilities of an end-effector through a graphical representation of interval constraints, which is referred to as a *motion constraint graph* (MCG).

Preprint submitted to 13th Conference on Automation Science and Engineering.
Received March 20, 2017.

Given the MCG and an arbitrary scene, an interval constraint satisfaction problem is defined to compute a complete set of subsurface combinations that give rise to all possible grasp configurations for a particular end-effector. The following constraints can be satisfied as part of the solver: (a) End-effector constraints, which include the distance ranges between contact points of the end-effector, as well as the surface area of each contact point. (b) Collision constraints between objects in the scene and the end-effector.

The resulting set of graspable subsurfaces on the objects' geometry can be used to: 1) recommending permissible grasps in the context of grasp planning, or 2) efficiently pruning grasps from a grasp database given a scene (e.g., such as those generated with GraspIt [6]), or 3) selecting between alternative end-effectors. The proposed approach is applicable to any kind of scene geometry, including arbitrarily cluttered scenes and curved surfaces. It also allows reasoning for a variety of end-effectors with multiple degrees of freedom. Experiments in simulation indicate that the proposed geometric reachability analysis increases the success rate of grasp and motion planning processes, while also reducing online planning time, at the cost of a small amount of precomputation.

II. Related Work

This section reviews a small portion of the significant literature on grasp generation and planning. The focus is on work related to grasping given object models, model-free grasping approaches and alternative kinematic abstractions of end-effector for use in grasp generation. There are detailed surveys that provide a more comprehensive coverage on work related to computing grasps [3], [20].

Model-based Approaches for Grasping Known Objects: Computing grasps that optimize desirable metrics, such as stability [14], task coverage [11], or contact point uncertainty [15], can help to safely grasp and manipulate objects in real-life. This can be an expensive and time-consuming procedure, which often means precomputation of such grasps is desirable. If both the objects and type of end-effector are known beforehand, then the grasps can be precomputed by sampling and optimizing offline [16]. Such "grasp databases" can still be used for objects that they were not necessarily originally constructed for, provided that the objects share some similarity in shape [6]. The idea of Agglomerative Clustering [7] has been introduced to group points that share similar normals and position values into a hierarchical data structure, which is then utilized to adapt precomputed, reachable grasps online using optimization methods.

The method proposed in this work also assumes the model of the object is known but it does not depend on extensive preprocessing. It is able to work directly over a geometric representation of a scene in order to generate a set of "graspable surfaces", which satisfy a set of constraints imposed by: 1) the end-effector kinematics, 2) the geometry of the contact points, as well as 3) the proximity of other objects in the scene. The resulting surfaces can then be used for several different applications, with grasp generation being one possible utilization.

Grasping Unknown Objects: Without knowing the type of object beforehand, grasp generation methods typically focus on finding "graspable" features directly from sensor data [18]. One method uses the swept volumes of the geometries of the end-effector, along with a bounding box decomposition of the sensed object, to generate grasps [24]. It generates pregrasps by sampling a placement of contact points that are "reachable" by the hand on each approximate bounding box. Deep-learning has also been utilized successfully for detecting graspable points over a point cloud [10]. Machine learning has also been integrated with geometric reasoning to detect grasps on unknown objects in cluttered scenes by establishing a set of necessary and sufficient conditions for grasping with a parallel-jaw gripper, which can be used to guide the sampling of grasp hypotheses [21].

In automation setups where the objects that can appear may be known, the benefit of the proposed approach is that: 1) it does not require extensive training once an object model has been built, 2) it can easily generalize to different and new types of end-effectors. Nevertheless, integration of the current work with machine learning methods is interesting towards specifying robust grasps directly over sensing data.

End-Effector Abstraction: For general end-effector motion constraints, the notion of Task Space Regions [2] specifies various constraints both on the end-effector and the target object. It can be used to guide the grasp generation and motion planning processes. If the end-effectors are known beforehand, it is possible to define the contact points and motion capability of the end-effector, so as to enforce constraints on grasp generation [4], [17]. The current work similarly aims to abstract end-effector kinematics but takes advantage of fast computational geometry primitives, which have been shown effective in analyzing geometric surfaces to compute affordances for locomotion behaviors [8]. The idea is to be able to quickly operate over complex, cluttered scenes with respect to parametric representations of robotic end-effectors.

III. Problem Formulation

Consider a situation similar to the one shown in Figure 1, where an n-dimensional manipulator $\mathbf{R}$ is placed in a workspace $\mathbf{W}$ with a set of known movable objects $\mathbf{M}$ and a set of static obstacles $\mathbf{O}$. The set $\mathbf{O}$ includes objects like the table in a table-top scenario, while the set $\mathbf{M}$ corresponds to the manipulatable objects placed on top of the table.

The assumption for the current work is that the type of objects that can appear are known and their mesh is provided. The underlying techniques can be extended to operate directly with sensing data and cases where the objects are unknown but this is not in the focus in the current paper. Thus, given the observation of the scene and available end-effectors, the framework loads the meshes of all detected objects and obstacles at estimated poses assuming access to a vision-based solution for pose estimation.

The manipulator $\mathbf{R}$ is equipped with a set of end-effectors $\mathbf{F}$, where each end-effector $f \in \mathbf{F}$ can achieve a set of

Preprint submitted to 13th Conference on Automation Science and Engineering.
Received March 20, 2017.

grasping configurations, expressed through a motion graph MCG_f . Each MCG_f is a graph, where the nodes are contact points (i.e., fingertips or lexical parts), while edges specify the relative constraints between the contact points. Motion graphs are representative of the end-effector kinematics, which can be abstracted through the definition of constraints on the motion graphs [8].

In this context, a grasping mode can be abstracted as $MCG_f = \{\mathbf{C}, \mathbf{E}\}$, which defines the contact points $c \in \mathbf{C}$ and constraints $\mathbf{e}(i,j) \in \mathbf{E}$ between pairs of contacts $(c_i, c_j) \in \mathbf{C}$. Each contact point defines a relative position on the end-effector, a normal pointing outward, and an approximate bounding radius. Each spatial constraint defines the range of possible distance between the contact points. If a contact point can rotate, the MCG can specify the rotation axis and a possible range of rotations for that contact point. An example MCG is shown in Figure 2.

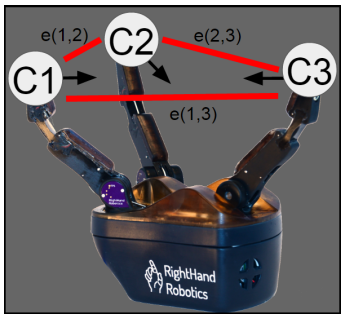

Fig. 2: A motion constraint graph (MCG) defining a 3-fingered grasp for the Reflex hand. Arrows are contact normal vectors. Edges are distance constraints between pairs of contact points (min & max).

These grasping modes define distinctly different configurations an end-effector can achieve to grasp objects, e.g., a hand-like end-effector switching from an open-palm grasp to a pinch grasp. An object $m \in \mathbf{M}$ is said to be **graspable** $\iff \{\exists\, f \in \mathbf{F} \wedge \exists\, g \in MCG_f \mid m \cap g(M) \neq \emptyset\}$. In other words, this means that for the object m, the end-effector f has some MCG_f whose constraints are satisfied, such that f can achieve a valid grasping configuration on m using this motion constraint graph. An object is **reachable** if there is a collision-free trajectory for the arm to the object.

IV. Geometric Reachability Analysis

The proposed Geometric Reachability Analysis (GRA) consists of three major components, as shown in Figure 3:

- **Adaptive Surface Clustering:** Given a set of meshes, the approach clusters all triangles that belong to the same object and have a similar normal into a "surface set". During this process, the motion graph of each end-effector is used to adaptively tune the area of surfaces based on the area of the end-effector's contact points.
- **Surface Pruning:** For objects that are close to one another, compute the swept volume of their corresponding surfaces and remove any intersecting portions.
- **Constraint Satisfaction:** For each surface set and MCG pair, apply constraint satisfaction to find the graspable surfaces for the end-effector.

Details of these components are explained in the following subsections.

A. Adaptive Surface Clustering

The primary element of this algorithm is the *surface*, so accordingly we first describe how surfaces are created out of object meshes. Our method differentiates between *surfaces*, which represent initial sets of triangles, and *sub-surfaces*, which are surfaces who have been altered as a consequence of any operation in our algorithm. The focus of this work is not on object detection or mesh reconstruction, therefore we assume that the objects are known, and their corresponding meshes are available. There are several works which can deal with handling unknown objects, such as 3D surface reconstruction by visual data using volumetric scene reconstruction [5], or surface scene reconstruction [12], [23], which is more appropriate for grasp planning. In future work, surface reconstruction could be applied as a preprocessing step in our method if the objects are unknown, after which point our method would have some model of the object and could proceed with the rest of the algorithm.

Adaptive Surface Clustering processes the meshes of all movable objects $\mathbf{M}$ and static obstacles $\mathbf{O}$ in the scene, incrementally constructing a surface from each mesh. The main idea is to iterate over all triangles in each mesh, and for a particular triangle, create a candidate list of all other triangles which share a common normal (up to an ϵ difference). Then, by applying BFS (Breadth First Search) over the selected triangle, we can construct the surface incrementally. Specifically, we put the selected triangle into a queue, and while the queue is not empty, we pop the front triangle and add it to the current *candidate surface*. Then, we find its children by examining the candidate list. Each candidate triangle is added to the queue if and only if it has at least one common vertex with one of the triangles already in the current surface. Once the queue is empty, the candidate surface is added to a set, and this process continues until all triangles in the mesh belong to at least one candidate surface.

The candidate surfaces are now evaluated based on an area constraint, which relates to the geometric representation of the contact points. For this work, we approximate this geometry using a circle, and define a specific radius for the contact points of each different end-effector. In order to do adaptive clustering, the normal threshold ϵ is initially set to zero. Next, we calculate how many candidate surfaces satisfy the area requirement - if this ratio is too low, then we *increase* the normal threshold ϵ by some increment. This does not mean, however, that we ignore candidate surfaces from the last step; instead, we rerun this process only for the candidate surfaces whose area is less than the minimum contact area.

After having the final set of candidate surfaces with an acceptable ratio, we check each surface for whether a contact point can be placed on it. This is to remove any irregularly shaped surfaces, such as a long narrow strips, from being considered in further steps. This final step converts the candidate surfaces into the final set of *surfaces* for use by the next steps of the method. As for all of the described parameters, they are all controllable and depend on the

Preprint submitted to 13th Conference on Automation Science and Engineering.
Received March 20, 2017.

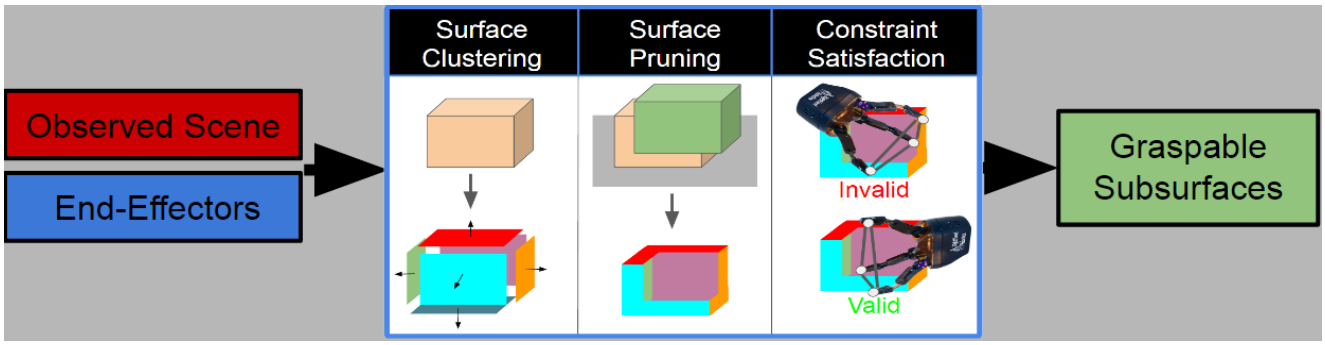

Fig. 3: The mesh of an object is extracted and clustered into surfaces as part of **Adaptive Surface Clustering**. Any other objects or obstacles that are proximal to this object alter the target surfaces in **Surface Pruning**. Given an end-effector, **Constraint Satisfaction** extracts the graspable subsurfaces of the object for each given motion constraint graph.

desired resolution. Finer resolution gives us greater precision at the expense of higher computation time and vice versa, because with finer resolution the number of surfaces will be increased, and as mentioned all operations in this method are over surfaces. Depending on the application, we could prioritize our preferences and set parameters to get desired result.

B. Surface Pruning

The primary focus of this work is to produce a set of surfaces which account for the complete set of permissible grasps capable by the end-effector, while accounting for geometric constraints imposed by the scene - in particular, due to clutter from other objects. This step in our method handles the potential collisions caused by the end-effector when attempting to grasp a particular object. We remove surfaces or parts of surfaces in which there is no possibility of placing contact points over them. This condition happens when two surfaces are so close, that regardless of the orientation of the end-effector, it is physically impossible to place one of the fingers. This can occur, for example, if one object is resting on top of another object, since the resting surfaces of the objects will be unreachable.

The algorithm for surface pruning is shown in Alg. 1. The inputs are the surface set $\mathbf{S}$ and the contact point width υ, which is end-effector specific. In order to do surface pruning for each available end-effector, only the corresponding contact point width must be specified. Additionally, this process could be called for a single target object, or it could be used on all objects in the scene by merging their surface sets into $\mathbf{S}$.

We first compute the closest distance between each pair of surfaces $s, s' \in \mathbf{S}$. If this distance is less than or equal to contact point width υ, then the two surfaces are proximal and must be pruned. First, the proximal surface s' is swept in the reverse normal direction $\hat{s}$ of $\mathbf{s}$, with the magnitude υ, resulting in a swept surface s''.

Finally, we compute the intersection between the swept surface s'' and the target surface s, and remove the intersecting portions from s and s''. This in turn creates a *subsurface* for s and s', as they have had their original surfaces altered by this procedure. If there are no intersections between the surfaces, then the resulting subsurface after intersection is same as s, and if they have complete overlap then the final subsurface is empty.

Inside the intersection function, the defined contact points constraints are applied over the resulting subsurface. This is used to check whether the resulting subsurfaces satisfy both the area and contact point geometric constraints. If the subsurface does not satisfy these constraints, it is removed from further consideration. For any point on the resulting subsurfaces, we are guaranteed that the end-effector will not collide with any neighboring objects when attempting to place contact points over these subsurfaces.

Algorithm 1: Surface Pruning

```
1 Function Surface_Pruning(S, υ)
2     foreach s ∈ S do
3         foreach s' ∈ (S − s) do
4             if (DISTANCE(s, s') < υ) then
5                 s'' ← SWEEP(s', −ŝ, υ)
6                 s, s' ← INTERSECT(s'', s)
```

C. Constraint Satisfaction

After all triangles in the observable scene have been clustered, with any proximal subsurfaces removed, the algorithm then proceeds to apply constraint satisfaction over the remaining subsurfaces using the motion constraint graphs $\mathbf{MCG}$ defined for each end-effector. This results in a mapping from end-effector grasping mode to valid graspable surfaces. The algorithm is shown in Alg. 2, The input is surface set after surface pruning S', defined motion graphs for target end-effector and normal threshold δ. This function could be run for all of the available end-effectors.

In order to apply constraint satisfaction, we first find all possible permutations for contact points over surfaces. We iterate over the defined contact points $\mathbf{C}$ of the current $\mathbf{MCG}$ and over each surface. Then the current surface is set to be the corresponding surface $\mathbf{CS}$ of the current contact point, and the rotation r between the normal of $\mathbf{CS}$ and the normal of the current contact point is calculated. Then consequently we need to rotate all other fingers with r and find corresponding surfaces for them if there is any(`Match_Surface` fucntion). Finding a corresponding

Preprint submitted to 13th Conference on Automation Science and Engineering.
Received March 20, 2017.

surface for other fingers follows a similar procedure, with the only difference being that we consider their permissible rotations only if it is specified in the **MCG**. This allows us to account for all the degrees of freedom of the end-effector while search for valid surfaces.

Algorithm 2: Constraint Satisfaction

```
1  Function Constraint_Satisfaction(S', MCG, δ, v)
2      foreach m ∈ MCG do
3          foreach c ∈ C(m) do
4              foreach s ∈ S' do
5                  CS ← CS ∪ s
6                  r ← COMPUTE_ROTATION(ĉ, ŝ)
7                  if (MATCH_SURFACE(S', MCG, δ, r)) then
8                      IDS ← IDS ∪ ids
9              foreach ids ∈ IDS do
10                 foreach ci ∈ C(ids) do
11                     foreach cj ∈ (C(ids) − ci) do
12                         sd ← p_ci − p_cj
13                         s' ← SWEEP(CS_cj, ŝd, v)
14                         CS_ci ← INTERSECT(s', CS_ci)
```

Once we are done with finding the corresponding surfaces for each contact point, and if all contact points have corresponding surface, then constraint satisfaction can be applied to find the valid surfaces. The result of the first step is a vector of IDS (intermediate data structure) which stores the contact points with their corresponding surfaces. Next, we iterate over all IDSs in order to find valid surfaces. The `SWEEP` operation is applied and then we do intersection to find any intersection between each pair of corresponding surfaces **CS** with each pair of contact points. For sweeping we need a sweep direction $\hat{\mathbf{sd}}$, which is calculated using the normalized direction between position **p** of two contact points.

If there is any intersection, the resulting subsurface after intersection is a valid area for placing corresponding contact points. Finally the result of the this step is a set of "graspable surfaces" which satisfy the geometric and kinematic constraints of the end-effector subject to the constraints of the scene, which can then be used by other processes (e.g. a grasp planner).

V. Use Cases of Our Approach

This section describes some of the potential use cases, as shown in Figure 4, for using the output of our geometric environment analysis in the context of grasping and manipulation. The validity of these use cases are later evaluated in the experimental section.

A. Grasp Generation

Although the focus of this paper is not on the grasp generation method, it is nevertheless important to demonstrate how the "graspable subsurface sets" could be leveraged. In order to grasp a particular object *m* in the scene, the framework must now convert these surfaces into a set of grasps **G** for use by the manipulation planner. Given parameters N (total number of sampled grasps) and Θ (rotation resolution), we describe a straightforward sampling-based procedure to generate grasps for evaluation.

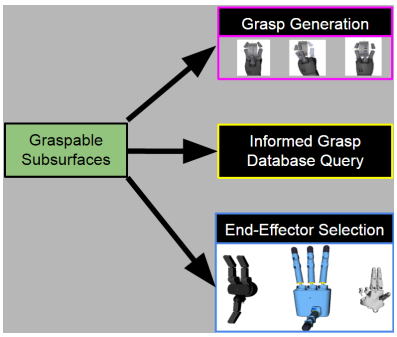

Fig. 4: Different use cases of graspable subsurfaces.

The procedure behaves in the following way: first, a surface $s_{\texttt{rand}}$ is uniformly randomly sampled from the surface set $\mathbf{S}(m)$. Then, a contact point from the corresponding *MCG* is randomly selected and then placed randomly onto $s_{\texttt{rand}}$. This automatically fixes the placements of all other remaining contact points, and defines a grasping configuration g. The end-effector is placed at g and rotated around the normal of the fixed contact point in increments of Θ. Each rotation generates a new grasping configuration g_Θ. If g_Θ is collision-free for the end-effector alone (disregarding the rest of the robot), it is then added to the set of grasping configuration $\mathbf{G}_{\texttt{RAND}}$.

This sampling procedure repeats until N grasps have been sampled and added to $\mathbf{G}_{\texttt{RAND}}$ Each grasp $g \in \mathbf{G}_{\texttt{RAND}}$ is given as input to an IK solver [1] and the resulting state of the manipulator is collision checked. Collision-free grasps are added to the final set of grasps **G**. After grasp generation, the planning framework receives a set of valid grasps, **G**, along with their corresponding IK solutions. The reachability of each grasp $g \in \mathbf{G}$ is evaluated through a standard manipulation planning framework, such as the Grasp-RRT [22]. The first collision-free trajectory to any of the grasps in **G** is returned to the robot for execution.

B. Database Pruning

The advantage of using a grasp database [6], [16], over an online grasp generation method, is that additional time can be spent optimizing various criteria (e.g. stability [14], task coverage [11], or contact point uncertainty [15]), which can greatly increase the grasp success rate of the robot. If the robot must deal with clutter in its workspace, these databases must be sufficiently large enough so that the probability of finding a successful grasp is high.

The downside with such a large database is that the robot could potentially spend a large amount of time evaluating the different grasps in the database, until eventually a valid grasp is found or the robot runs out of alloted time.

Rather than blindly searching through the database, the graspable surfaces proposed in this paper could be used to search a focused subsection of the database. This could be accomplished by using the aforementioned grasp generation method on the graspable surfaces, and performing a nearest-neighbor query on the database to extract similar grasps.

Preprint submitted to 13th Conference on Automation Science and Engineering.
Received March 20, 2017.

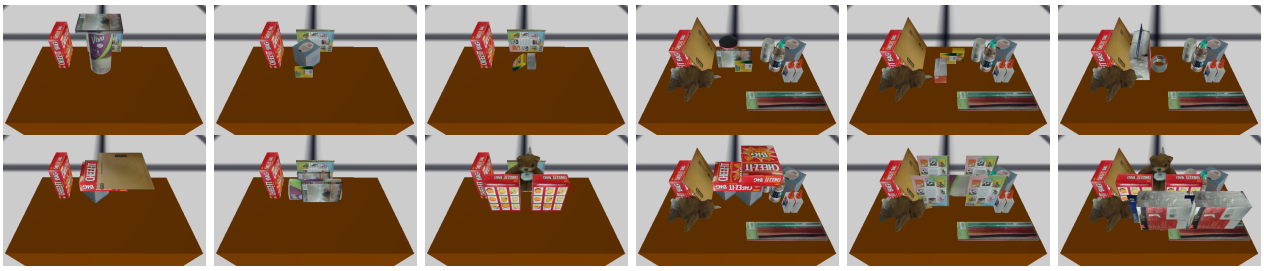

Fig. 5: (Top) Parallel Gripper Benchmarks: DVD, Crayola, and Duct Tape. (Bottom) ReFlex Hand Benchmarks: Cheezit, Kleenex Paper Towels, and Bear. Left 3 Images (S)parse Clutter, Right 3 Images (D)ense Clutter.

C. End-Effector Selection

For multiple different end-effectors, the graspable surfaces provide an evaluation criteria that a task planner could use in order to select which end-effector of the robot would be used for manipulation. One of the more straight-forward procedures would be to compute the graspable surfaces for each available end-effector. If a particular end-effector has no viable grasp surfaces returned by the method, then the planner can avoid spending any extra time checking for collision-free grasps or even motion-planning for that end-effector. Being able to skip planning on an end-effector becomes critical as the number and complexity of available end-effectors increases for the robot.

VI. Experiments

This section evaluates the applicability of the proposed method in various domains through evaluation in simulation. To accomplish this, three different types of simulations were performed, demonstrating the proposed method being used as: 1) a grasp generation method; 2) a database pruning method; and 3) an end-effector selection method.

Setup: Each experiment was conducted on a single computer with an Intel(R) Xeon(R) E5-1650 3.50GHz CPU using a motion and task planning simulation framework [13]. The robot used in the experiments was a single dual-arm Yaskawa Motoman with 7 independent DOFs per arm and 1 shared DOF at the torso. Each arm was equipped with a different end-effector. The left arm with a parallel gripper and the right arm with a 3-finger Reflex hand. Each benchmark has a designated "goal object", which corresponds to the object that the planning framework computes a trajectory to grasp, and two variations of clutter in the scene: sparse and dense.

The environments used are shown in Figure 5.

Motion Constraint Graphs: The MCGs for both end-effectors were created by checking the distance and relative position constraints between the fingers. Since the parallel gripper only has two contact points, a single MCG was sufficient for representing its geometric and kinematic constraints. For the 3-finger Reflex hand, a single MCG was used, which consisted of one vertex per finger, positioned at the fingertips, with rotation and distance constraints imposed between each pair of fingers (similar to Figure 2). This MCG was sufficient in representing the kinematics of the Reflex.

Computational Overhead: Table I reports the amount of time spent by the proposed method for computing "graspable surfaces" for each benchmark, as well as the complexity of each benchmark's scene. The computation time increases as a function of the geometric complexity in the scene, expressed by the number of surfaces generated by MCG and the total number of triangles in the scene's meshes. It should be noted that the parameters used to generate the surfaces in all benchmarks were kept static and not changed - with some tuning, it would be possible to achieve faster times on the more complex objects (such as the Bear).

ID	G	Object	C	GOC	EC	CT(s)
S0	P	DVD	S	6/12	58/172	1.95798
D0	P	DVD	D	6/12	1049/2276	2.78367
S1	P	Crayola	S	6/12	24/48	0.565691
D1	P	Crayola	D	6/12	1021/2164	1.7192
S2	P	Tape	S	66/256	24/48	4.6155
D2	P	Tape	D	66/256	1028/2179	11.4858
S0	R	Cheezit	S	6/12	30/60	1.5875
D0	R	Cheezit	D	6/12	1027/2176	4.63645
S1	R	Towel	S	34/124	30/60	3.2792
D1	R	Towel	D	34/124	1033/2188	11.734
S2	R	Bear	S	77/604	36/72	18.939
D2	R	Bear	D	77/604	1065/2254	30.4325

TABLE I: Computation time for generating "graspable surfaces", relative to the geometric complexity of the scene. **G**: Gripper: Parallel (P) or ReFlex (R). **C**: Clutter: Sparse (S) or Dense (D). **GOC**: Goal Object Complexity in # of surfaces and triangles triangles. **EC**: Total Scene Complexity in # of surfaces and triangles. **CT**: Computation Time.

A. Grasp Generation

This experiment evaluates the use of MCG-based surfaces for generating grasps to manipulate a target object. As before, the environments used are shown in Figure 5. We utilize the grasp generation method described in Section 5A, with parameters N chosen to be 500 and Θ is set to 15. RAND represents the comparison method, which does not take the "graspable surfaces" into account. The purpose of such a comparison is to establish a potential use case of the MCG surfaces.

The metrics used to evaluate both methods were: *number of valid grasps*, *success rate*, *grasp generation time* and *motion planning time*. The number of valid grasps measures how many collision-free grasps were found by the grasp generator. The success rate corresponds to whether a collision-free trajectory for grasping the goal object was found within a time limit of 60 seconds. Grasp generation time measures how long it took to compute collision-free IK

Preprint submitted to 13th Conference on Automation Science and Engineering.
Received March 20, 2017.

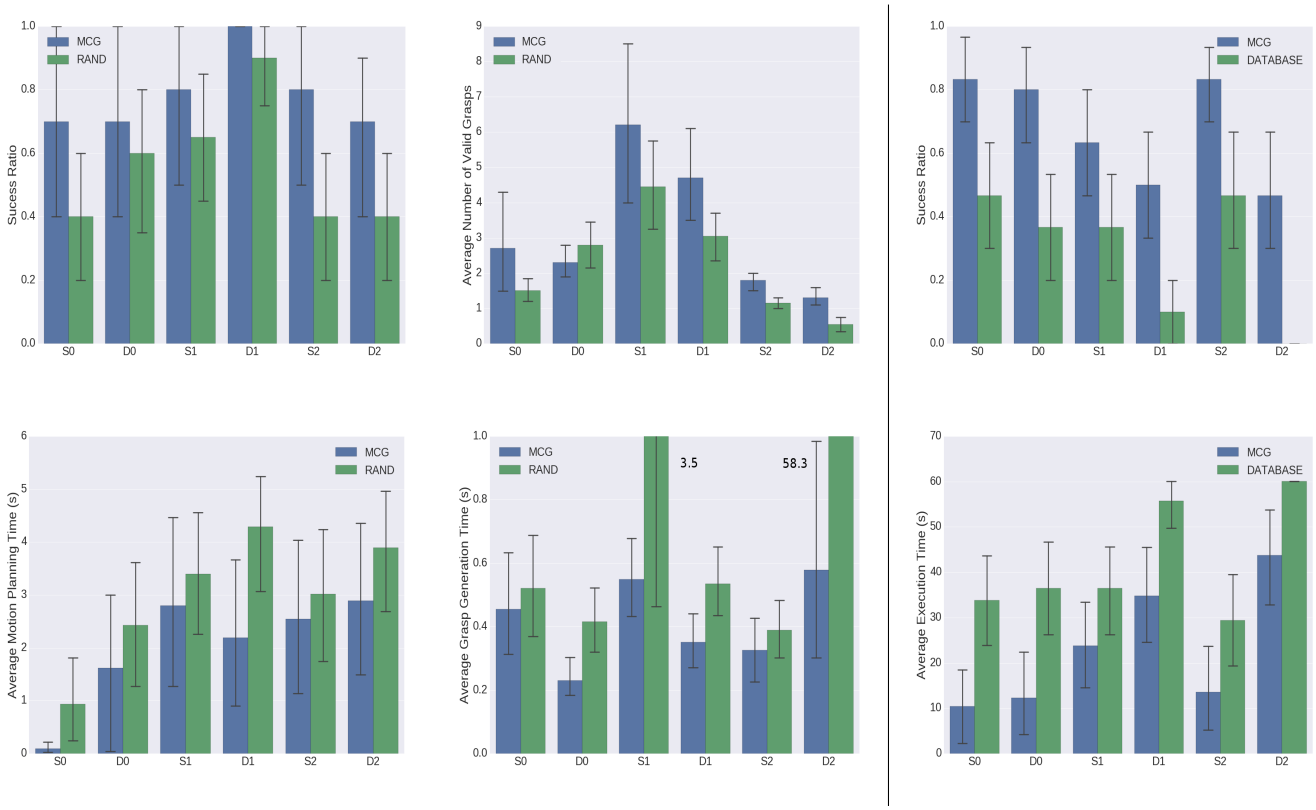

TABLE II: Experimental Results for **Parallel (Left 2 Columns)** and **Reflex (Right Column)** End-Effectors. In both cases, **Planning Success Rate** corresponds to whether a collision-free trajectory for grasping the goal object was found within a time limit of 60 seconds. For the Parallel-Jaw, the **Number of Valid Grasps** is how many collision-free grasps were computed, the **Average Grasp Generation Time** is how much time the grasp planner spent to compute IK-solutions and approach motion plans, and the **Average Motion Planning Time** is how much time the motion planner took to compute a collision-free trajectory. For Reflex, **Average Execution Time** represents the total accumulated time by all components (grasp database validation, motion planning, and when applicable, **MCG** surface construction.)

solutions for each valid grasp. If grasp generation fails to produce a collision-free solution, it continues to sample and collision-check new grasps run up to 60 seconds, after which point the run is reported as a failure. Motion planning time measures how long it took the motion planner to compute a solution trajectory for the character using a precomputed PRM* roadmap [9].

Analysis: The results are shown in Table II (the four leftmost graphs). In all of the benchmarks, the MCG method provided a larger number of valid grasps, spent less time in grasp generation, and improved the overall success rate of the manipulation planning framework. For some of the easier benchmarks, the difference between the methods was not as significant. This is to be expected, as the benefit of using MCG arises when the goal object is severely occluded by other objects in the scene.

It was originally expected that both methods would spend a similar amount of time motion planning, as both methods share the same planning framework. This was not the case, however, as MCG spent overall a smaller amount of time for motion planning purposes. This result seems to indicate that the grasps generated from MCG allow the motion planner to find a solution quicker.

B. Database Pruning

This experiment examines how effective the MCG surfaces can be for decreasing the amount of time it takes for finding a valid grasp in a precomputed grasp database. As before, the environments used are shown in Figure 5. As described in Section 5B, the MCG surfaces can be used to reduce the number of grasps evaluated from the database. The comparison method, DATABASE, does not utilize the MCG surfaces, and evaluates grasps incrementally until a collision-free grasp is found or the amount of allotted time runs out.

The metrics used to evaluate both methods were: *success rate* and *execution time*. Success rate similarly corresponds to whether a collision-free trajectory for grasping the goal object was found within a time limit of 60 seconds. Execution time represents the total time taken by the method, but only for the instances where the method was successful in solving the problem. Of important note here is that since grasp databases are being used, there is no time spent by either method in grasp generation. To account for the randomness inherit in the motion planning framework, each benchmark was executed 30 times and the resulting averages and standard deviation are reported.

Analysis: The results shown in Table II (the two rightmost graphs) indicate that the MCG approach was able to

significantly reduce the execution time in the benchmarks, as well as improve upon the overall success rate of the motion planner. In terms of where the time was spent, the bottleneck in the experiments was during grasp validation, which evaluated IK-solutions for the grasps, as well as grasp approach plans for the manipulator.

C. End-Effector Selection

By examining the time efficiency of the MCG method, which is shown in Table I, there is an indication that the proposed approach could be used as a selection criteria for end-effectors. The computation time spent by MCG scales with the complexity of the scenes; in all cases, it was much faster to query the MCG than to wait for a failure criterion (i.e. evaluating grasps for 60 seconds). The method could therefore be used by a task planning framework to first evaluate whether or not an end-effector can grasp the target object, without resorting to more expensive processes.

VII. Conclusion

This paper introduces a method for annotating an unstructured environment with graspable surfaces given a set of constraints defined for end-effectors carried by a robotic manipulator. The proposed method utilizes geometric reasoning and constraint satisfaction, in order to evaluate the geometry in the scene, and quickly extract a continuous representation of all viable grasp surfaces. The approach can be used to generate candidate grasps, prune existing grasp databases or alternatively to select end-effectors, which are most suitable for specific environment configurations.

Although the current work assumed that the object models and pose of the object were available, this was mostly associated to the framework's implementation rather than a limitation of the underlying methodology. An immediate next step is to utilize mesh-approximation methods, which will allow to handle receiving point clouds as direct input.

Currently, the motion constraint graphs for each end-effector are defined by the user. There has been work in learning effective grasping shapes for specific end-effectors [10], [17]. Extending such work to general types of end-effectors, and developing a method for creating MCG representations of these shapes, would allow the proposed method to not require user input.

Currently for the generation of grasps, a sampling process is used to generate them given the detected grasp surfaces. Nevertheless, an important benefit for the proposed technique is that it is able to operate over the continuous geometry and provide a complete description of the viable grasp surfaces. This indicates a promising direction, where rather than sampling grasps given the continuous surfaces, the motion planner would ideally incorporate the entire surface in order to generate a manipulation trajectory that results in an effective grasp. One way to do this is to utilize the grasp surface as a way to define goal sets in the context of trajectory optimization methods, such as CHOMP [25].

References

[1] P. Beeson and B. Ames. Trac-ik: An open-source library for improved solving of generic inverse kinematics. In *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*, pages 928–935. IEEE, 2015.

[2] D. Berenson, S. Srinivasa, and J. Kuffner. Task space regions: A framework for pose-constrained manipulation planning. *IJRR*, 30(12):1435–1460, 2011.

[3] J. Bohg, A. Morales, T. Asfour, and D. Kragic. Data-driven grasp synthesisa survey. *IEEE Transactions on Robotics*, 30(2):289–309, 2014.

[4] M. Ciocarlie, C. Goldfeder, and P. Allen. Dimensionality reduction for hand-independent dexterous robotic grasping. In *IROS '07*, pages 3270–3275. IEEE, 2007.

[5] C. R. Dyer. Volumetric scene reconstruction from multiple views. In *Foundations of image understanding*, pages 469–489. Springer, 2001.

[6] C. Goldfeder, M. Ciocarlie, H. Dang, and P. K. Allen. The columbia grasp database. In *ICRA '09*, pages 1710–1716. IEEE, 2009.

[7] K. Hang, J. A. Stork, and D. Kragic. Hierarchical fingertip space for multi-fingered precision grasping. In *IROS '14*, pages 1641–1648. IEEE, 2014.

[8] M. Kapadia, X. Xianghao, M. Nitti, M. Kallmann, S. Coros, R. W. Sumner, and M. Gross. Precision: Precomputing environment semantics for contact-rich character animation. In *SIGGRAPH '16*, pages 29–37. ACM, 2016.

[9] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *IJRR '11*, 30(7):846–894, 2011.

[10] I. Lenz, H. Lee, and A. Saxena. Deep learning for detecting robotic grasps. *IJRR*, 34(4-5):705–724, 2015.

[11] Y. Lin and Y. Sun. Grasp planning to maximize task coverage. *The International Journal of Robotics Research*, 34(9):1195–1210, 2015.

[12] V. Lippiello, F. Ruggiero, B. Siciliano, and L. Villani. Visual grasp planning for unknown objects using a multifingered robotic hand. *IEEE/ASME Transactions on Mechatronics*, 18(3):1050–1059, 2013.

[13] Z. Littlefield, A. Krontiris, A. Kimmel, A. Dobson, R. Shome, and K. E. Bekris. An extensible software architecture for composing motion and task planners. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, pages 327–339. Springer, 2014.

[14] S. Liu and S. Carpin. A fast algorithm for grasp quality evaluation using the object wrench space. In *CASE '15*, pages 558–563. IEEE, 2015.

[15] S. Liu and S. Carpin. Kinematic noise propagation and grasp quality evaluation. In *CASE '16*, pages 1177–1183. IEEE, 2016.

[16] A. T. Miller and P. K. Allen. Graspit! a versatile simulator for robotic grasping. *IEEE Robotics & Automation Magazine*, 11(4):110–122, 2004.

[17] A. T. Miller, S. Knoop, H. I. Christensen, and P. K. Allen. Automatic grasp planning using shape primitives. In *ICRA '03*, volume 2, pages 1824–1829. IEEE, 2003.

[18] M. Popović, G. Kootstra, J. A. Jørgensen, D. Kragic, and N. Krüger. Grasping unknown objects using an early cognitive vision system for general scene understanding. In *IROS '11*, pages 987–994. IEEE, 2011.

[19] M. A. Roa and R. Suárez. Grasp quality measures: review and performance. *Autonomous Robots*, 38(1):65–88, 2015.

[20] A. Sahbani, S. El-Khoury, and P. Bidaud. An overview of 3d object grasp synthesis algorithms. *Robotics and Autonomous Systems*, 60(3):326–336, 2012.

[21] A. ten Pas and R. Platt. Using geometry to detect grasp poses in 3d point clouds. In *Intl Symp. on Robotics Research*, 2015.

[22] N. Vahrenkamp, T. Asfour, and R. Dillmann. Simultaneous grasp and motion planning: Humanoid robot armar-iii. *IEEE Robotics & Automation Magazine*, 19(2):43–57, 2012.

[23] C. Xu and J. L. Prince. Snakes, shapes, and gradient vector flow. *IEEE Transactions on image processing*, 7(3):359–369, 1998.

[24] Z. Xue and R. Dillmann. Efficient grasp planning with reachability analysis. *International Journal of Humanoid Robotics*, 8(04):761–775, 2011.

[25] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa. Chomp: Covariant hamiltonian optimization for motion planning. *IJRR*, 32(9-10):1164–1193, 2013.