2017 IEEE-RAS 17th International Conference on
Humanoid Robotics (Humanoids)
Birmingham, UK, November 15-17, 2017

Affordance Detection for Task-Specific Grasping Using Deep Learning

Mia Kokic Johannes A. Stork Joshua A. Haustein Danica Kragic

***Abstract*— In this paper we utilize the notion of affordances to model relations between task, object and a grasp to address the problem of task-specific robotic grasping. We use convolutional neural networks for encoding and detecting object *affordances*, *class* and *orientation*, which we utilize to formulate *grasp constraints*. Our approach applies to previously unseen objects from a fixed set of classes and facilitates reasoning about which tasks an object affords and how to grasp it for that task. We evaluate affordance detection on full-view and partial-view synthetic data and compute task-specific grasps for objects that belong to ten different classes and afford five different tasks. We demonstrate the feasibility of our approach by employing an optimization-based grasp planner to compute task-specific grasps.**

I. INTRODUCTION

Most of the tools that humans use consist of multiple functional parts. Part attributes such as shape, size, material, etc. are often indicators of their function, i.e., a task they *afford*. Blades, for example, are sharp and rigid and therefore afford the task cutting. In the context of robotic manipulation, using these tools to execute such a task, requires grasping them first. However, for a given task we cannot just grasp in any manner. For instance, when grasping a knife for cutting, a grasp should be applied on the handle and the hand should be aligned with the main axis of the knife. Obviously, the object part that affords a task as well as the object pose condition a grasp applied to it.

For a robot, task-specific grasping is a challenging problem. In addition to *what*, a robot also needs to know *how* to grasp so that the desired post-grasp action, i.e., task, can be executed. Task-specific grasping, therefore, requires modeling relations between object, task and a grasp so that the reasoning system is able to generalize well to novel scenarios.

In our previous work [1], we address this problem by encoding relations of different grasp, task and object variables in a probabilistic manner. Although this approach naturally deals with uncertainties, it required a discretization of multiple variables, which is infeasible for such a high-dimensional problem.

In contrast, in this work we propose to model relations between object, task and a grasp by utilizing the notion of affordances. We describe *how* to grasp in terms of *grasp constraints*, namely contact locations and a hand approach direction. We assume that contact locations depend on a location of a part that affords the task and argue that detecting part affordances facilitates reasoning about both *what* and *how* to grasp for a given task. While *what* to grasp depends only on local information—the presence of a part—*how*, in addition, depends on global information—the presence of other parts of the object and spatial relations between them as well as the object pose. Therefore, we encode and detect object affordances (see first row, Fig. 1), class and orientation in a data-driven way and use these quantities to formulate grasp constraints (see second row, Fig. 1). A grasp planner then computes a stable grasp that fulfills these constraints (see third row, Fig. 1). By utilizing deep learning we avoid the need for feature engineering. In addition, formulating these quantities as grasp constraints allows for a flexible integration with any optimization-based grasp planner.

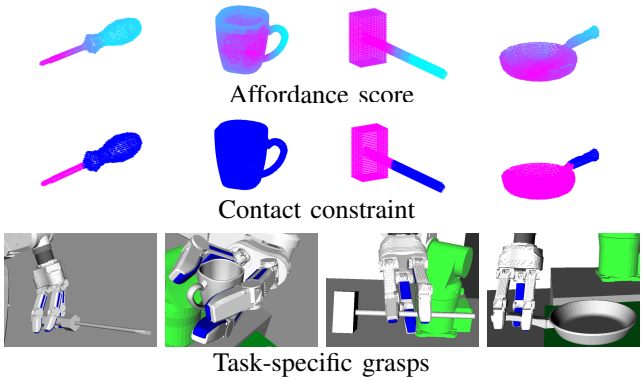


Fig. 1: Given the shape of an object and a task, we detect object part affordances. From these we formulate grasp constraints, such as a contact location constraint. These constraints are then utilized to compute task-specific grasps as shown here for example tasks poke, pour, pound and support on the objects screwdriver, mug, hammer and pan respectively. Magenta color indicates high affordance score (top) and contact avoidance constraint for grasping (middle).

Overall, this paper makes the following contributions:

1) Using convolutional neural networks (CNNs) for object affordance detection for task-specific grasping on full and partial point clouds.
2) Encoding object affordances, class and, orientation to formulate grasp constraints and modeling the relations between task, object, and grasp based on human knowledge.
3) Showing proof of concept results for task-specific grasping using optimization-based fingertip grasp planning.

All authors are with the Robotics, Perception, and Learning lab, School of Computer Science and Communication, KTH Royal Institute of Technology, Stockholm, Sweden {mkokic|jastork|haustein|dani}@kth.se

This work was partially supported by Wallenberg Autonomous Systems and Software Program (WASP), the Swedish Research Council and the Swedish Foundation for Strategic Research.

II. RELATED WORK

A. Learning Affordances from 2.5D and 3D data

The concept of affordances has been used in many studies in robotics. Earlier works focus on hand-crafted features to extract geometric properties from point clouds such as size, convexity and eccentricity of an object. Song et al. [1] encode these in a Bayesian network while Ten Pas and Platt [2] detect graspable object parts from 3D point clouds by searching for cylindrical shells that satisfy certain criteria with respect to local neighborhoods of the point cloud. Myers et al. [3] construct a large corpus of RGB-D images and infer affordances of 105 tools from geometric features. All of these approaches rely on manually designed features and therefore suffer from a lack of generalizability.

Nguyen et al. [4] demonstrate the power of deep learning on an RGB-D dataset, setting a new benchmark for affordance detection. Since learning the important depth properties from RGB-D is challenging on small datasets, they combine multiple modalities such as depth, RGB channels, horizontal disparity, etc.

Recent successes in deep learning have shown promising results ranging from shape completion to recognition by employing a fully volumetric representation to learn complex shape distributions from large scale 3D CAD datasets. The first to deploy CNNs on 3D data were Wu et al. [5]. To study 3D shape representations for objects, they propose a convolutional deep belief network to represent a geometric 3D shape as a probability distribution of binary variables on a 3D voxel grid for which they construct a large-scale 3D CAD model dataset. Similarly, Maturana and Scherer [6] use a volumetric architecture and represent objects as binary voxel grids. The benefits of these approaches are that they can process 3D data from different sources such as CAD models, LiDAR, point clouds, etc.

In this work we follow a similar approach by adopting a volumetric representation which allows us to make full use of the geometric information in the data. Additionaly, such a representation of an object allows for flexible integration with a fingertip grasp planner when computing a task-specific grasp.

B. Task-specific Grasping

Numerous methods have been suggested for generating stable grasps on objects [7–9]. However, grasping in a goal-oriented manner requires reasoning about task requirements and satisfying task-specific constraints. Previous attempts to represent these requirements by grasp wrench analysis [10] or detection of graspable part [11] do not consider task-specific constraints such as contact locations and hand approach directions.

Dang and Allen [12] present an examplar-based planning framework to generate task-specific grasps. They introduce a semantic affordance map, which relates local geometry to a set of predefined semantic grasps that are appropriate for different tasks. Given a map, the pose of a robot hand with respect to an object can be estimated so that the hand is adjusted to achieve the ideal approach direction required by a particular task. A semantic affordance map on each object class is built using a representative object of that class, which limits the capability of their method to deal with objects with large shape variance inside the class.

Vahrenkamp et al. [13] argue that extracting semantic information is necessary for task-specific grasping. In their approach objects are segmented into parts labeled with semantic information (affordances) which are used for generating and transfering robotic grasps. This approach, however, relies on hand crafted features to detect affordances and requires offline grasp planning for a chosen hand.

In our previous work [1], we encoded relations between different variables in Bayesian networks (BNs) to represent and model robot-grasping tasks. To interpret the "goals" of a given task we defined constraints which depend on both the object and the action features. This approach relies on hand-crafted features to extract physical attributes of an object and requires training of one BN for each hand.

In this paper, we learn affordances, class and orientation in a data-driven way by using CNNs. Formulating these as grasp constraints facilitates reasoning about *where* and *how* to grasp for a task.

III. PROBLEM FORMULATION

We propose a system for generating a stable, task-specific grasp on an object for a robotic hand. Our system aims to model relations between object, task and a grasp by encoding and detecting object attributes and utilizing them to compute task-specific grasps.

A. Notation and Definitions

Before going into the details, we first clarify the use of notations. We use $\mathbf{T}$ to represent a set of tasks. The tasks we consider are $\mathbf{T} = \{\text{cut, poke, pound, pour, support}\}$. We describe an object by its surface $\mathbf{O}$ which we represent by a point cloud, i.e., a set of points $\mathbf{p} \in \mathbb{R}^3$. We denote a set of object classes $\mathbf{C} = \{\text{bottle, can, hammer, knife, mug, pan, pen, screwdriver, spoon, wine glass}\}$ and use $\mathbf{g}$ to represent a grasp.

To describe relations between tasks and objects we define affordance labels that correspond to given tasks $\mathbf{T}$. Each affordance label characterizes a set of object parts that have certain physical attributes. We say that an object affords a task if it has such a part. We define those relations as follows:

- **cut** — thin parts with a flat edge such as blades
- **poke** — long pointy parts such as shanks of tools
- **pound** — parts of compact solid mass such as hammer heads or tool handles
- **pour** — parts such as containers typically found on mugs, cups and wine glasses
- **support** — thin and flat vessels usually used to serve food such as plates, spoon heads etc.

IV. SYSTEM OVERVIEW

The input to our system is a surface of an object $\mathbf{O}$ in the form of a point cloud and a task $t \in \mathbf{T}$. The output is

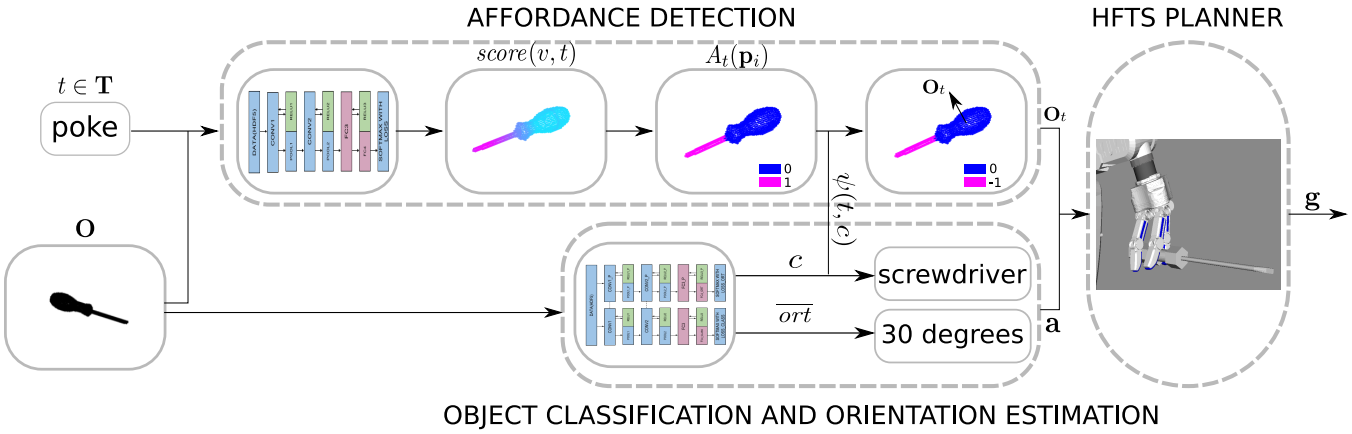


Fig. 2: Pipeline of the process. The input to our system is a point cloud $\mathbf{O}$ and a task $t \in \mathbf{T}$. In the first stage, we detect object affordances (first row) and in parallel we classify objects and estimate their orientation (second row). The output of the affordance detection is a binary map $\mathbf{O}_t$, which indicates contact locations for grasping. The class and orientation estimation is used to determine a desired approach direction $\mathbf{a}$. In the second stage, these quantities are used by a grasp planner to compute a task-specific grasp on the object.

a stable grasp $\mathbf{g}$ which fulfills requirements posed by the task. We model a grasp as a tuple $\mathbf{g} = (\phi, T)$, where ϕ is a hand configuration and $T \in SO(3)$ the pose of the hand with respect to the grasped object.

To generate a task-specific grasp on an object we propose a two stage process, see Fig. 2. In the first stage, we detect which parts of the input object afford the given task. In parallel, we classify the object and estimate its orientation $\overline{ort}$ with respect to a nominal class reference frame. To encode and detect affordances, class and orientation, we employ two CNNs: one for affordance detection, *AFF-CNN*, and one for joint object classification and orientation estimation, *CO-CNN*, described in detail in Sec. V. The input to both networks is a binary voxel grid V of an object, where a voxel $v \in V$ is assigned value 1, if there is $\mathbf{p} \in \mathbf{O}$ that lies in the voxel's volume, and else 0.

We assume that the task influences a grasp both in contact locations and approach direction. Hence, the results of the first stage are used to formulate grasp constraints, which consist of a *contact constraint* and an *approach direction constraint*. The *contact constraint* defines the part of the object's surface $\mathbf{O}_t$ the robot is allowed to grasp and we define it with respect to the part that affords the task. The *approach direction constraint* describes the 3D angle that the robot hand approaches the object with, $\mathbf{a}$, and depends on the detected orientation $\overline{ort}$. In addition to the task, both of these constraints require knowledge about the full object geometry. Although one object can afford multiple tasks we define these constraints for the most common combinations of object classes and tasks they afford, based on human knowledge. In the second stage, we use an optimization-based grasp planner to compute a grasp that fulfills these constraints.

The complete process is summarized as pseudocode in Algorithm 1. In the following sections we will first provide a conceptual overview over the individual components before explaining each component in more detail in Sec. V.

Algorithm 1: Task-specific grasping.

Data: Object point cloud $\mathbf{O}$, task $t \in \mathbf{T}$
Result: Grasp $\mathbf{g}$

```
/* Affordance Detection                                   */
V ← VOXELIZE(O);
V ← SLIDING-WINDOWS(V);
for each sub-volume Ṽ ∈ V do
    p(t | Ṽ) ← FORWARD-PASS(Ṽ, AFF-CNN);
end
for each voxel v ∈ V do
    score(v,t) ← Σ_{Ṽ∈V} p(t|Ṽ) δ(Ṽ,v,t);
end
for each point p_i ∈ O do
    A_t(p_i) ← score(GET-VOXEL(p_i), t) > ε;
end
/* Object Classification and Orientation
   Estimation                                             */
V ← VOXELIZE(O);
ort, c ← FORWARD-PASS(V, CO-CNN);
/* Grasp Planning                                         */
O_t ← CONTACT-LOCATIONS(A_t(p_i) ψ(t,c));
a ← APPROACH-DIRECTION(ort, a_n);
g ← PLAN-GRASP(O_t, a, O);
```

A. From Affordances to Contact Locations

The output of *AFF-CNN* is used to compute a score function $score(v,t)$ that assigns an affordance score to each voxel $v \in V$, where a larger score indicates that the points represented by this voxel afford the task t.

This scoring function is then used to segment the object into voxels that afford the task t and voxels that do not afford this task. After segmentation, we convert the voxel grid back to the point cloud and for each point $\mathbf{p} \in \mathbf{O}$ obtain the binary value based on the segmentation. We denote this segmentation as predicate $A_t(\mathbf{p}) \in \{0,1\}$ on the set of points, where the value $A_t(\mathbf{p}) = 1$ indicates that the point $\mathbf{p}$ affords the task t. An example for this is shown in Fig. 2, where a screwdriver is segmented for the task *poke*.

In order to model the *contact constraint*, we define a preference for fingertip placement $\psi(t,c)$. If an object that

Object class	Affordance for Task				
	cut	poke	pound	pour	support
bottle				0	
can				0	
hammer			-1		
knife	-1		-1		
mug				0	
pan					-1
pen		0			
screwdriver		-1	-1		
spoon		-1			-1
wine glass				0	

Fig. 3: Preference $\psi(t, c)$ for fingertip placement depending on affordance labels. The preference can be positive or neutral (0) and negative (-1). Empty entries indicate that $\psi(t, c)$ is not defined, meaning that the object class is not suited for that particular task.

belongs to the class c has a part that affords the task t, i.e., $A_t(\mathbf{p}) = 1$, the function $\psi(t, c)$ defines if the fingers are allowed to touch that part (positive or neutral preference) or if they should avoid it (negative preference):

$$\psi(t, c) = \begin{cases} 0, & \text{positive or neutral preference} \\ -1, & \text{negative preference} \end{cases}. \quad (1)$$

The definition of $\psi(t, c)$ is given in Fig. 3. With this definition the part of the object that should be grasped for a given task is:

$$\mathbf{O}_t = \{\mathbf{p} \in \mathbf{O} | A_t(\mathbf{p}) \cdot \psi(t, c) = 0\}. \quad (2)$$

For example, although a screwdriver and a pen both have a part that affords poking, $\psi(\textit{poke}, \textit{pen}) = 0$ for a pen indicates that the fingers can be placed on the affording part since there is no other way to grasp a pen for poking. On the other hand, $\psi(\textit{poke}, \textit{screwdriver}) = -1$ expresses our preference for placing our fingers on the handle of a screwdriver rather than on its shank when poking with it.

B. From Orientation to Approach Direction

In order to define the *approach direction constraint*, we require a reference frame. For this, we manually align all objects from a training set that belong to the same class and define one prefered nominal approach direction $\mathbf{a}_n(c, t) \in \mathbb{S}_2$ for each task in the local frame of the aligned objects.

During online detection, we assume that the reference frame of the input point cloud $\mathbf{O}$ is aligned with the nominal class reference frame, apart from an unknown rotation around the z-axis, see Fig. 4. Hence, we utilize *CO-CNN* to classifiy the object and to estimate its rotation along this z-axis.

Once classified, the preferred approach direction for an input object is a function of the estimated orientation $\overline{ort}$ with respect to the nominal frame, and the nominal approach direction for the task-class pair, i.e. $\mathbf{a}(\overline{ort}, \mathbf{a}_n(c, t))$, see Fig. 4. While there may be many approach directions suitable for a particular task, we formulate the *approach direction constraint* such that the set of valid approach directions is

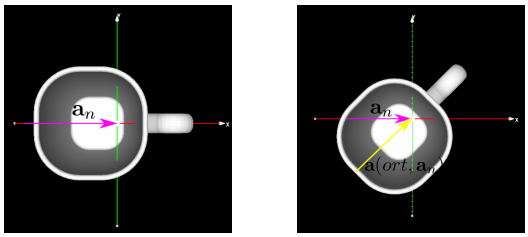


Fig. 4: An example of $\mathbf{a}$ (colored yellow) for a mug rotated by 45° around the z-axis with respect to the nominal frame. The magenta arrow indicates $\mathbf{a}_n$ for the class mug for task pour.

limited to the set $\mathbf{D} = \{\mathbf{d} \in \mathbb{S}_2 \mid \measuredangle(\mathbf{a}, \mathbf{d}) \leqslant \theta\}$, where the bounding angle θ is a modeling parameter. An example of the preffered approach direction $\mathbf{a}$ on a mug is shown in Fig. 4 and examples of nominal approach directions $\mathbf{a}_n(c, t)$ for different objects and tasks are shown in Fig. 5.

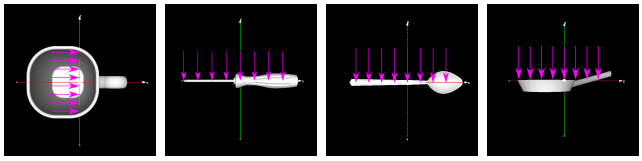


Fig. 5: Examples of $\mathbf{a}_n(c, t)$ (colored magenta) for a mug for pour, for a screwdriver for poke and for spoon and pan for support. A set of arrows on objects corresponds to one approach direction (x on a mug, -y on a screwdriver, -y on a spoon and -z on a pan). For simplicity, we represent a set of arrows with only one arrow in the rest of the paper.

V. METHOD

A. Data Preparation

For training the CNNs we use a dataset consisting of 3D CAD models from ShapeNet and ModelNet40. For each polygon mesh model we first extract coordinates of the mesh's vertices and use these to construct a dense point cloud $\mathbf{O}$ from which we then generate a binary voxel grid V. All objects are scaled to fit inside a $50 \times 50 \times 50$ grid of fixed size regardless of their relative real-world size. Each point $\mathbf{p} \in \mathbf{O}$ is therefore, represented as a voxel in this grid that has a value of 1.

In most real-world scenarios the object point clouds we get from cameras are noisy, sparse or unsegmented and the object may be in clutter or occluded. To that end we prepare the data for training on full and incomplete point clouds. We generate partial point clouds from different camera perspectives by randomly sampling viewpoints on a sphere positioned at the center of the object's local coordinate frame. *AFF-CNN* was trained only on full point clouds and *CO-CNN* on both full and partial point clouds.

B. Affordance Detection

1) Training Phase: We construct the training data for *AFF-CNN* from parts of the objects in our 3D dataset. For this, we select for each task $t \in \mathbf{T}$ the parts of the objects that afford the task t independent of object class, see Fig. 7. Although one part can have multiple affordances we mark only one, dominant affordance. The training set of *AFF-CNN*

DATA	CONV1			POOL1		CONV2			POOL2		FC3
INPUT	FILTER SIZE	PAD	OUTPUT	FILTER SIZE	STRIDE	FILTER SIZE	PAD	OUTPUT	FILTER SIZE	STRIDE	OUTPUT
$50 \times 50 \times 50$	$5 \times 5 \times 5$	3	32	$4 \times 4 \times 4$	4	$5 \times 5 \times 5$	3	64	$4 \times 4 \times 4$	4	64

Fig. 6: Layer parameters the both networks. The parameters are the same up to the last fully connected layer *fc4* which has different number of outputs. For affordance detection it is $k = 5$, for object class $m = 10$ and orientation estimation $n = 36$.

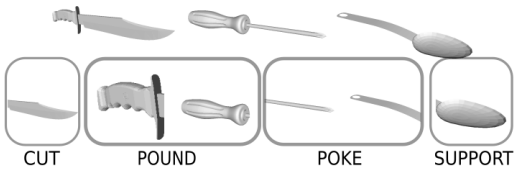


Fig. 7: Segmented objects and examples of parts labeled with cut, pound, poke and support affordance on a knife, screwdriver and a spoon.

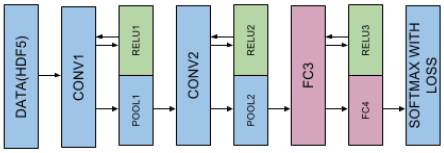


Fig. 8: *AFF-CNN*: The network consists of two convolutional layers and two fully connected layers with ReLU and pooling layers in between. The last layer is Softmax with loss.

is then the set of object parts labeled with the respective task. The object parts are represented as sub-volumes of the complete voxel grid of an object and are upscaled to maximal expansion inside the $50 \times 50 \times 50$ grid. To increase the size of our training set we augment it by scaling, rotating and translating each part.

2) Network Architecture: The network consists of two convolutional layers and two fully connected layers with pooling and ReLU (Rectified Linear Unit) layers in between. The output of the last layer corresponds to the number of affordance labels, $k = 5$. The network architecture is similar to LeNet-5 [14] and the choice of architecture was based on the ratio of the number of training examples and classes. The parameters are described in Fig. 6 and the architecture is shown in Fig. 8.

3) Testing Phase: During testing, we use a sliding-window technique on the complete voxel grid of an object to obtain a set of candidate sub-volumes which we then classify. We slide the 3D window in *x,y* and z direction to obtain a set of 3D sub-volumes. The sub-volumes are $\lambda \times \lambda \times \lambda$ voxel grids where $\lambda \in \{15, 20, 25, 30, 35, 40\}$ and are obtained with at least 30% overlap. Each sub-volume $\tilde{V} \in \mathbf{V}$ is scaled to a $50 \times 50 \times 50$ voxel grid and then fed to the network (Algorithm 1 l. 3-5).

The score for each voxel-task pair (v, t) is calculated as a sum over all sub-volumes obtained by sliding the window. The value for each sub-volume $\tilde{V}$ is equal to the probability $p(t, \tilde{V})$ of the respective sub-volume affording the task t multiplied by 1, if $\tilde{V}$ contains the voxel v and if t is the most probable task afforded by $\tilde{V}$, and otherwise 0:

$$score(v, t) \leftarrow \sum_{\tilde{V} \in \mathbf{V}} p(t \mid \tilde{V}) \delta(\tilde{V}, v, t), \tag{3}$$

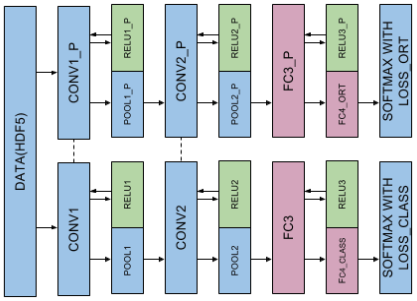


Fig. 9: *CO-CNN*. Both towers consist of two convolutional layer and two fully connected layers with ReLU and pooling layers in between. The last layer is Softmax with loss.

where

$$\delta(\tilde{V}, v, t) = \begin{cases} 1, & \text{if } v \in \tilde{V} \text{ and} \\ & t \in \arg\max_{t' \in \mathbf{T}} p(t' \mid \tilde{V}) \ . \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

In the end, all the voxel scores are normalized to sum up to 1.

C. Object Classification and Orientation Estimation

1) Training Phase: For the training set of *CO-CNN* we first manually label each object with a class $c \in \mathbf{C}$. We align all meshes so that the local coordinate frame of an object is aligned with the nominal class reference frame. As we want *CO-CNN* to be able to estimate the orientation of an object, we further rotate each instance for $10^\circ \times \overline{ort}$ around the z-axis and label each rotation with a bin $\overline{ort}$, where $\overline{ort} \in \{0, 1 \ldots, 35\}$. The network was trained on both full and partial point clouds and the results of the experiments are shown in Sec. VI-C.

2) Network Architecture: Considering that object class and orientation share similar information, we train two networks jointly to avoid having to learn one network per class. The network has similar structure to *AFF-CNN* except it consists of two towers: one for the class and one for the orientation. Since bottom layers of the network are more general, we share weights up to the first fully connected layer. This weight sharing mechanism also helps in reducing the complexity of the system. The outputs of the network are two vectors of m and n probabilities, where $m = 10$ is the number of classes and $n = 36$ the number of orientation bins. The parameters are described in Fig. 6 and the structure of the network is shown in Fig. 9.

D. Grasp Planning

We wish to verify that the *contact* and the *approach direction constraints* computed in Algorithm 1 can be utilized in planning stable task-specific grasps. For this, we

adopt the integrated fingertip grasp and motion planner presented in [15]. The algorithm is based on [16–18] and simultaneously searches stable fingertip grasps and hand-arm approach motions for an n_f-fingered robotic hand. The grasps are planned by performing stochastic optimization of grasp contacts $\gamma \in \mathbf{O}^{n_f}$ under an objective function $O(\gamma)$ using a multilevel refinement metaheuristic.

The input to the algorithm is the point cloud $\mathbf{O}$ of an object, the *approach direction constraint* in form of the preferred approach direction $\mathbf{a}$, the tolerance angle θ, and the *contact constraint* in form of the subset of contact points $\mathbf{O}_t \subseteq \mathbf{O}$. We limit the search space of grasp contacts to $\mathbf{O}_t^{n_f}$, thus implicitly adopting the *contact constraint*. To address the *approach direction constraint*, we formulate the objective function

$$O(\gamma) = \frac{Q(\gamma)}{R(\gamma) + \beta(1 - \mathbf{a} \cdot \mathbf{e}(\gamma)) + \alpha}, \tag{5}$$

where Q denotes a Ferrari-Canny grasp quality function [19], R the grasp reachability function described in [17, 18] and $\mathbf{e}(\gamma) \in \mathbb{S}_2$ the approximated end-effector orientation for a fingertip grasp achieving contacts γ. The parameters $\alpha, \beta \in \mathbb{R}^{\geq 0}$ serve as scaling factors between the different sub-objectives. The end-effector orientation $\mathbf{e}(\gamma)$ is computed using an approximate hand configuration that is provided by the reachability heuristic R as described in [17, 18].

By stochastically maximizing Eq. 5 the algorithm searches for contacts that achieve high grasp quality and are likely to be feasible and reachable with the desired hand orientation. If the found contacts $\gamma^* \in \mathbf{O}_t^{n_f}$ achieve a sufficient grasp quality, i.e., achieve a stable grasp, a post-optimization procedure is applied to compute a hand configuration ϕ and pose T, for which the hand reaches γ^*. For this, the approximate hand configuration and pose which are provided by the reachability heuristic R are adjusted to minimize the error to the planned contact locations γ^* and preferred approach direction $\mathbf{a}$. If the resulting grasp (ϕ, T) is infeasible or violates any of the constraints, we rerun the stochastic algorithm.

VI. EXPERIMENTAL EVALUATION

A. Training and Testing Data

There are about $10,000$ instances spanning over 5 task and 10 object classes. Each object class consists of $\sim$1000 instances. The training and testing sets for *AFF-CNN* consist of object parts and we use 70% of these instances for training and 30% for testing. Fig. 3 indicates to which objects the parts used as training data for *AFF-CNN* belong to.

During evaluation, for each object labeled with a certain affordance label, we first obtain the scores $score(v, t)$ for each voxel and the input task as described in Sec. V-B.3. We then apply a threshold to segment the object. This gives us voxel coordinates of points that afford the task t. We calculate the overlap with ground truth values (parts used in training) to compute F_1 scores.

	CO-CNN	*AFF-CNN*
Optimizer	SGD (Stochastic Gradient Descent)	
Learning policy	step	
Base learning rate	0.001	
Step size	1000	200
Gamma	0.95	
Momentum	0.9	
Loss	cross entropy	
Batch size	50	

Fig. 10: Training parameters for both networks.

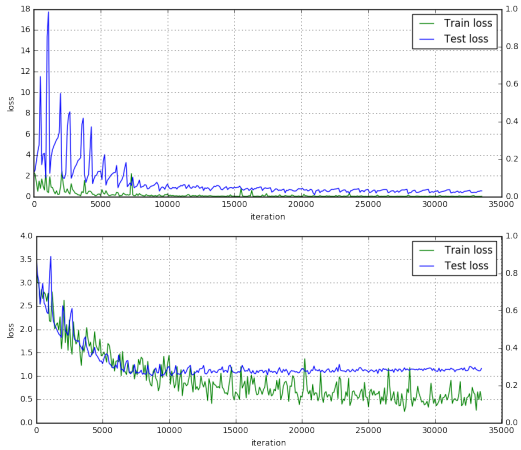


Fig. 11: Loss over time for object classification (top) and orientation estimation (bottom) network (*CO-CNN*).

For *CO-CNN* network we use 70% of the objects belonging to each class for training and 30% for testing. We provide F_1 scores, precision, recall and accuracy[1] results on training and testing set for both classification and orientation.

For training we use a modified version of the Caffe library [21], which supports 3D convolution and pooling. We train *CO-CNN* from scratch and use the learned weights to initialize the *AFF-CNN*. The training parameters are shown in Fig. 10. The networks were trained until convergence on an NVIDIA GeForce GTX TITAN and the training curves for *CO-CNN* are shown in Fig. 11. The loss for *CO-CNN* is computed as:

$$L^{joint} = L^{class} + L^{ort} \tag{6}$$

where L^{class} is classification and L^{ort} orientation loss. Both losses are computed as cross entropy losses.[2]

B. Affordance Detection

We evaluate the performance of the network on full and partial point clouds. The network trained on full point clouds and tested on partial data did perform worse as shown in Fig. 12. To plot the ROC curve we vary the threshold parameter when converting score values $score(v, t)$ to binary values A_t. For each threshold $\varepsilon \in \{0, 0.1, 0.2, \ldots, 1.0\}$ we calculate precision and recall based on overlap with ground truth. We find the optimal threshold to be 0.5 based on

[1] see the definition of these terms in [20]
[2] see the definition of this term in [22]

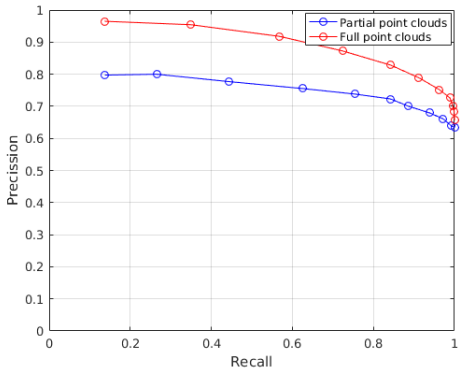


Fig. 12: ROC curves for network trained on full and tested on full and partial point clouds.

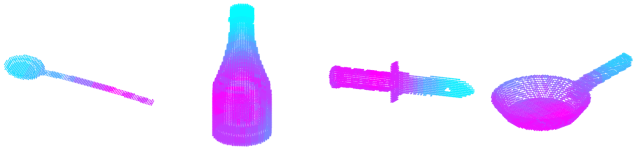


Fig. 13: Examples of part affordance detection on full point clouds for spoon (poke), bottle (pour), knife (pound) and pan (support).

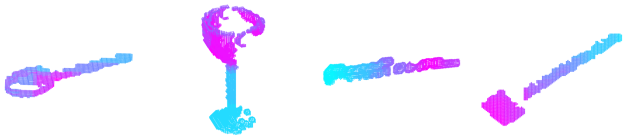


Fig. 14: Examples of part affordance detection on partial point clouds for spoon (support), wine glass (pour), screwdriver (poke) and hammer (pound).

Fig. 15: Detection results on objects which do not belong to the training set for drill (poke), spatula (support) and ladle (pour).

Youden's index J = sensitivity + specificity $-$ 1 for which J is maximum.

The F_1 scores for each affordance label on full and partial point clouds are shown in Table Fig. 16 and qualitative results are shown in Fig. 13 and Fig. 14. Note that the F_1 scores for the *cut* task are low for full and partial point clouds. This is because we only test the performance on the objects from the *knife* class and for many instances the shape of a blade and a handle are very similar.

The *AFF-CNN* can be applied to affordance detection on objects whose parts were not used in the training process. We show qualitative results in Fig. 15.

C. Object Classification and Orientation Estimation

We trained *CO-CNN* on full point clouds and tested it on full and partial point clouds. The results after $30,000$ training iterations show that when testing the network on partial point clouds the accuracy decreases by 23% for classification and 13% for orientation in comparison to the results on full point clouds.

	F_1 score	
Affordance Label	full	partial pc
cut	0.7262	0.6471
poke	0.8569	0.8244
pound	0.8946	0.7405
pour	0.9111	0.8249
support	0.8925	0.8327
overall	0.8461	0.7792

Fig. 16: Performance on full and partial point clouds.

Training	Testing	Precision	Recall	F_1 score	Accuracy
full	full	0.8932	0.8500	0.8242	0.8500
full	partial	0.7487	0.6200	0.6231	0.6200
partial	partial	0.7580	0.7800	0.7297	0.7800

Fig. 17: Object Classification Performance on Full and Partial Point Clouds

Training	Testing	Precision	Recall	F_1 score	Accuracy
full	full	0.8205	0.8393	0.8155	0.9000
full	partial	0.6871	0.7024	0.6616	0.7700
partial	partial	0.7128	0.6962	0.6813	0.8300

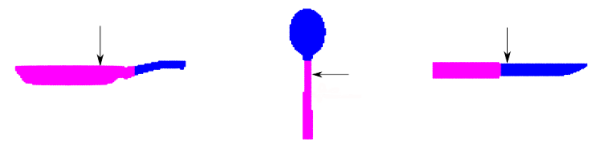
Fig. 18: Orientation Estimation Performance on Full and Partial Point Clouds.

To improve the performance on partial point clouds we trained the same network on partial point clouds. Since the dataset which consists of partial point clouds is similar to the original—it consists of the same objects—higher-level features learned in the first network are relevant for the new dataset as well. Thus, we initialized the network with the weights from the network trained on full point clouds and trained until convergence. The results show an increase in accuracy for classification to 78% and orientation to 83%. Tables in Fig. 17 and Fig. 18 summarize the results of *CO-CNN*.

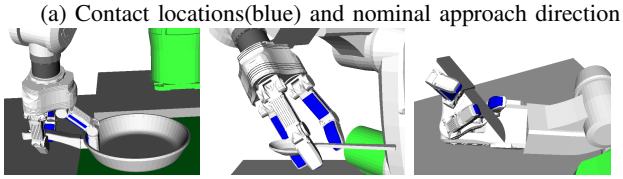
D. Grasp Planning

We qualitatively demonstrate the usability of the *contact* and *approach direction constraint* by applying grasp planning as described in Sec. V-D for a Schunk-SDH hand mounted on a KUKA KR5 sixx 850 manipulator using a simulation environment [23]. The imported meshes of the objects are rotated randomly around the z-axis in their local coordinate frame.

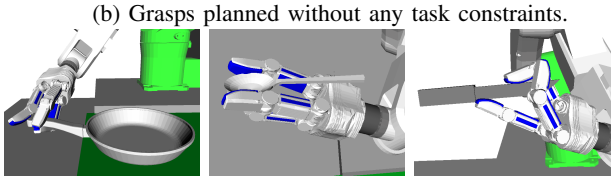
As shown in Fig. 19e, the grasp planner successfully computes stable grasps for which the robot hand only contacts $\mathbf{O}_t$ while being sufficiently aligned with the preferred approach direction $\mathbf{a}$. For comparison we plan grasps in the same environment without any of the constraints (see Fig. 19b), without *approach direction constraint* (see Fig. 19c), and without *contact constraint* (see Fig. 19d). This often leads to grasps that do not allow the task.



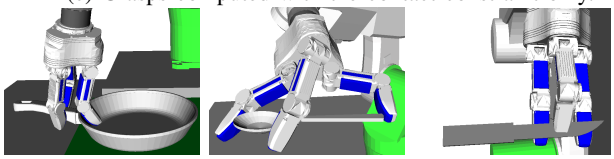
(a) Contact locations(blue) and nominal approach direction



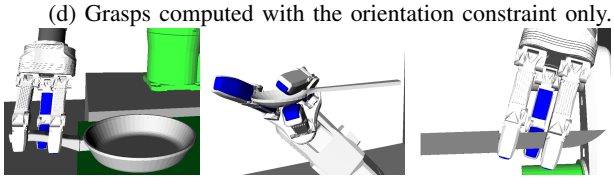
(b) Grasps planned without any task constraints.



(c) Grasps computed with the contact constraint only.



(d) Grasps computed with the orientation constraint only.



(e) Grasps computed with both constraints.

Fig. 19: Example grasps computed with the grasp planner on three objects for *support, poke* and *pound*.

VII. CONCLUSION AND FUTURE WORK

In this paper we proposed a system for computing a stable, task-specific grasp on an object by encoding and detecting part affordances, class and orientation of an object and formulating these quantities as grasp constraints. We evaluated the affordance detection, object classification and orientation estimation performance on full and partial point clouds and showed examples for grasps in simulation using a standard optimization-based planner. We used a fingertip grasp planner as a proof of concept. In the future we plan on considering the grasp type as an additional constraint induced by a task. Further, we plan on deploying an encoder-decoder network architecture for semantic segmentation of part affordances and extending our method to real-world data.

REFERENCES

[1] D. Song, C. H. Ek, K. Huebner, and D. Kragic, "Task-based robot grasp planning using probabilistic inference," *IEEE Transactions on Robotics*, vol. 31, no. 3, pp. 546–561, 2015.

[2] A. Ten Pas and R. Platt, "Localizing handle-like grasp affordances in 3d point clouds," in *Experimental Robotics*. Springer, 2016, pp. 623–638.

[3] A. Myers, C. L. Teo, C. Fermüller, and Y. Aloimonos, "Affordance detection of tool parts from geometric features," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 1374–1381.

[4] A. Nguyen, D. Kanoulas, D. G. Caldwell, and N. G. Tsagarakis, "Detecting object affordances with convolutional neural networks," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 2765–2770.

[5] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1912–1920.

[6] D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 922–928.

[7] A. Bicchi and V. Kumar, "Robotic grasping and contact: A review," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2000.

[8] A. Sahbani, S. El-Khoury, and P. Bidaud, "An overview of 3d object grasp synthesis algorithms," *Robotics and Autonomous Systems*, 2012.

[9] J. Bohg, A. Morales, T. Asfour, and D. Kragic, "Data-driven grasp synthesisa survey," *IEEE Transactions on Robotics*, vol. 30, no. 2, pp. 289–309, 2014.

[10] R. Haschke, J. J. Steil, I. Steuwer, and H. Ritter, "Task-oriented quality measures for dextrous grasping," in *2005 IEEE International Symposium on Computational Intelligence in Robotics and Automation, 2005. CIRA 2005. Proceedings.* IEEE, 2005, pp. 689–694.

[11] R. Detry and J. Piater, "Unsupervised learning of predictive parts for cross-object grasp transfer," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2013, pp. 1720–1727.

[12] H. Dang and P. K. Allen, "Semantic grasping: Planning robotic grasps functionally suitable for an object manipulation task," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2012, pp. 1311–1317.

[13] N. Vahrenkamp, L. Westkamp, N. Yamanobe, E. E. Aksoy, and T. Asfour, "Part-based grasp planning for familiar objects," in *Humanoid Robots (Humanoids), 2016 IEEE-RAS 16th International Conference on*. IEEE, 2016, pp. 919–925.

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

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

[16] K. Hang, J. A. Stork, F. T. Pokorny, and D. Kragic, "Combinatorial optimization for hierarchical contact-level grasping," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 381–388.

[17] K. Hang, J. A. Stork, and D. Kragic, "Hierarchical fingertip space for multi-fingered precision grasping," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014)*. IEEE, 2014, pp. 1641–1648.

[18] K. Hang, M. Li, J. A. Stork, Y. Bekiroglu, F. T. Pokorny, A. Billard, and D. Kragic, "Hierarchical fingertip space: A unified framework for grasp planning and in-hand grasp adaptation," *IEEE Transactions on Robotics*, vol. 32, no. 4, pp. 960–972, 2016.

[19] C. Ferrari and J. Canny, "Planning optimal grasps," in *1992 IEEE International Conference on Robotics and Automation, 1992. Proceedings.* IEEE, 1992, pp. 2290–2295.

[20] D. M. Powers, "Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation," 2011.

[21] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 2014, pp. 675–678.

[22] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[23] R. Diankov and J. Kuffner, "Openrave: A planning architecture for autonomous robotics," *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34*, vol. 79, 2008.