Learning How a Tool Affords by Simulating 3D Models from the Web

Paulo Abelha[1] and Frank Guerin[1]

***Abstract*— Robots performing everyday tasks such as cooking in a kitchen need to be able to deal with variations in the household tools that may be available. Given a particular task and a set of tools available, the robot needs to be able to assess which would be the best tool for the task, and also where to grasp that tool and how to orient it. This requires an understanding of what is important in a tool for a given task, and how the grasping and orientation relate to performance in the task. A robot can learn this by trying out many examples. This learning can be faster if these trials are done in simulation using tool models acquired from the Web. We provide a semi-automatic pipeline to process 3D models from the Web, allowing us to train from many different tools and their uses in simulation. We represent a tool object and its grasp and orientation using 21 parameters which capture the shapes and sizes of principal parts and the relationships among them. We then learn a 'task function' that maps this 21 parameter vector to a value describing how effective it is for a particular task. Our trained system can then process the unsegmented point cloud of a new tool and output a score and a way of using the tool for a particular task. We compare our approach with the closest one in the literature and show that we achieve significantly better results.**

I. INTRODUCTION

Service robots will face many challenges when working in unconstrained environments such as the home. Many of the tasks they need to accomplish require the use of tools which come in a great variety of shapes. On top of that the usual tool for a task might not be available and the robot will need to adapt what it knows in order to use a different tool to achieve its goals.

Humans are able to exploit visual and physical similarities to assess how good an unknown tool is for a task and how to use it: where to grasp and how to orient. For example: a wine bottle can roll dough; a broad-bladed kitchen knife can lift a small pancake; the handle of a wooden spoon can retrieve something that falls in the gap below a fridge.

This 'everyday creativity' with tools is the inspiration behind our work as it makes us think of a tool's affordance and different ways of using it as open to interpretation given an interplay between the goal task and the candidate tool's physical characteristics. We tackle the problem of how to assess the affordance of a tool by considering many different possible ways of using it given a goal task. A robot should be able to learn what is important in a tool for a given task and then apply this knowledge to an unknown tool to find the best way of using it for the task.

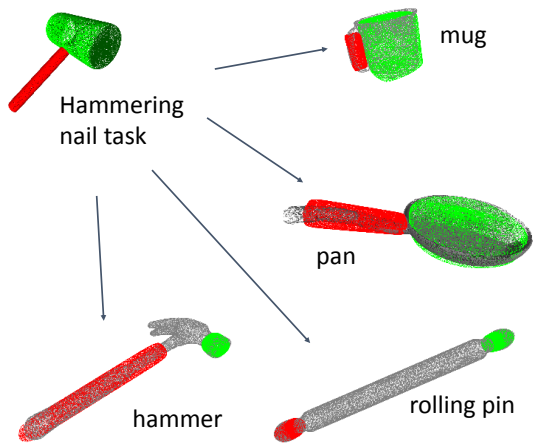


Fig. 1: If the usual tool is not available for a task the robot should be able to find the best way of grasping and orienting new tools. Here in red is the grasping part and in green, the end effector; in grey are points not used in the tool representation.

Work in the area of tool affordance for Robotics has been mainly focused on RGB-D input and per-pixel [1] or per-part [2] labelling of affordances. We believe it is important to also consider where the tool is grasped and its pose when learning/assessing affordances since the *way in which* a tool affords is a big part of *why* it affords. Returning to the examples above, the knife is only usable for lifting if its blade is horizontal, and not vertical as it would be in cutting; the wooden spoon might only fit in the gap if grasped by its bowl with the handle becoming the end effector.

The main contribution of this paper is providing a pipeline and framework to process 3D models for automatically learning through simulation both a tool's affordance and how the tool should be held and oriented. The issue of providing the appropriate grasp position and orientation is crucial if we want to train semi-autonomously. For this a robot needs to be able to assess the affordances of a set of tools (or models in simulation) to label them itself. This requires an automated method to set the grasp region and orientation in the best possible way. There are usually many ways to make a tool fail, but if there is one way to make it work then it should be labelled positive for the affordance.

At test time, given the point cloud of an unsegmented object and a task to accomplish, we output an 'affordance score' for how good it is for the task and also how to grasp

*Paulo Abelha is on a PhD studentship supported by the Brazilian agency CAPES through the program Science without Borders. Frank Guerin received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors

[1]Department of Computing Science, University of Aberdeen, King's College, AB24 3UE Aberdeen, Scotland p.abelha@abdn.ac.uk

and orient the tool. We assume the robot can gather a full 3D point cloud from the object - for instance by rotating it to see all sides [3], [4], [5], and also measure the weight by picking it up. The main limitation of this work is not considering agency, for instance, whether or not a robot could grasp a tool at a certain place, and swing it, depends on its gripper and arm strength.

We introduce three datasets: *ToolWeb* (Sec. III-D); *ToolArtec* (Sec. IV-A); and *ToolKinect* (Sec. IV-A). ToolWeb is made of 70 synthetic point clouds gathered from the web and annotated with ground truth for 4 tasks. The other two comprise scanned common household tools with annotated ground truth for the same 4 tasks tried in the real-world. ToolArtec has 50 point clouds scanned using Artec Eva 3D and ToolKinect, 13 point clouds using Kinect v2. We compare our approach with the closest approach in the literature [2] and show that we achieve significantly better results. We also show results for ToolArtec and ToolKinect in order to assess how the performance of the system changes with different scanner quality.

II. RELATED WORK

There are many approaches tackling the problem of assessing tool affordance for tasks. We can see them as differing in their inputs, representation, grounding, and outputs. Concerning inputs the approaches can use RGB [6], [7], RGB-D [1]; or point clouds [2] and also added physical information such as weight and material. Regarding representations they can use purely Machine Learning to learn features [8], [6], [1] or combine it with some hand-engineered features (e.g. histograms [2]; model fitting [9]). The grounding of the system concerns whether the approach is based on learning from hand-labelled data [2]; simulation of tasks [6], [10]; or a real robot trying out the tools [11]. As for outputs they can be scores for pixels/regions of an image (RGB, RGB-D) [6], [1] (in what is sometimes called *pixel-wise labeling task*); or for 'parts' of a point cloud [2], or a score for the object as a whole [9]. The outputs also vary in giving out a binary score (affords or does not afford) [12] or a graded score [1] [2], [11], [13], [14], or additionally providing manipulation cues that the robot could use to grasp [8], [1] and orient the tool. This final output (how the tool would be grasped oriented) is something we tackle in this paper, and is rarely considered in the existing literature; Mar et al. [10] implicitly considers how the tool is grasped and oriented as part of its input, hence learning a function that accounts for how the affordance differs depending on how it is used.

Given this varied landscape we situate our work closest to that of Schoeler and Wörgötter [2] as we both get as input a point cloud, use some form of hand-engineered feature (histogram and geometric models respectively) combined with Machine Learning, and output where to grasp and give it a graded score. Nonetheless, we go beyond current approaches in our ability to output a graded score and both where to grasp and also how to orient the tool for a task, while considering both the visual and physical (weight) aspects of an object. Relative to our past work [9] the major difference is that we now learn a function relating tool parameters to affordance by simulating different ways of using tools that we gather from 3D Web models (whereas previously the function was hand coded).

III. SYSTEM

A. Overview

There are four main parts to our approach (coloured in Fig. 2): preparing training data; simulating a task; learning a task function (that maps from our tool representation to a real-valued score); and assessing a new tool. Below we first explain our tool representation, the *p-tool*, and then we have one section for each of the four parts.

We gather models from the web and construct a training dataset that is labeled by trying the tool in simulation. The training set is then used for learning a function from a p-tool (Sec. III-B) to an affordance score. At test time the system gets as input a task, a point cloud, and tool weight. It outputs an affordance score for the task and what we call a *grasp-pose*: how to grasp and orient the point cloud for the task. At test time we are able to abstract the candidate tool in different ways, by fitting different p-tools, and find the best one for the task.

B. Tool Representation (p-tool)

For our tool representation we always assume a tool with two parts: Grasp and action. We represent a tool as a p-tool: a 21-dimensional vector storing information about the *grasp* and *action parts*; the relationship between these parts and the tool's mass. The grasp pose is contained in the p-tool because the first tool-part is the grasping place, and the orientation of the second part is given relative to this grasping part. We use *superquadrics* and *superparaboloids* to represent each part due to their geometrical flexibility (Fig. 3). An important aspect of our representation is that we can go from a p-tool to a rendered mesh that can be used in simulation. We render the mesh by first uniformly sampling from the superquadric/superparaboloid and then applying convex hull to the obtained point cloud. In the case of superparaboloids we remove the faces that 'close' the superparaboloid at the top.

We are able to extract p-tools from a point cloud by fitting superquadrics and superparaboloids to its segments. Superquadrics have been used in Computer Vision for their ability to represent many different shapes and recovery of superquadrics from range data has been widely studied [15]. We fit superquadrics as in our previous work [9] and extend the idea to also recover superparaboloids. We refer the reader to [15] as the main source of our approach to superquadrics. To the best of our knowledge there is no recovery of superparaboloids from point clouds and we derived the equation to get the cost function required for optimisation in a similar form as used for superquadrics in [15]:

$$F(\mathbf{x}) = \left(\left(\frac{x}{a_1} \right)^2 + \left(\frac{y}{a_2} \right)^2 \right)^{\frac{1}{2\epsilon_1}} - \left(\frac{z}{a_3} \right) \qquad (1)$$

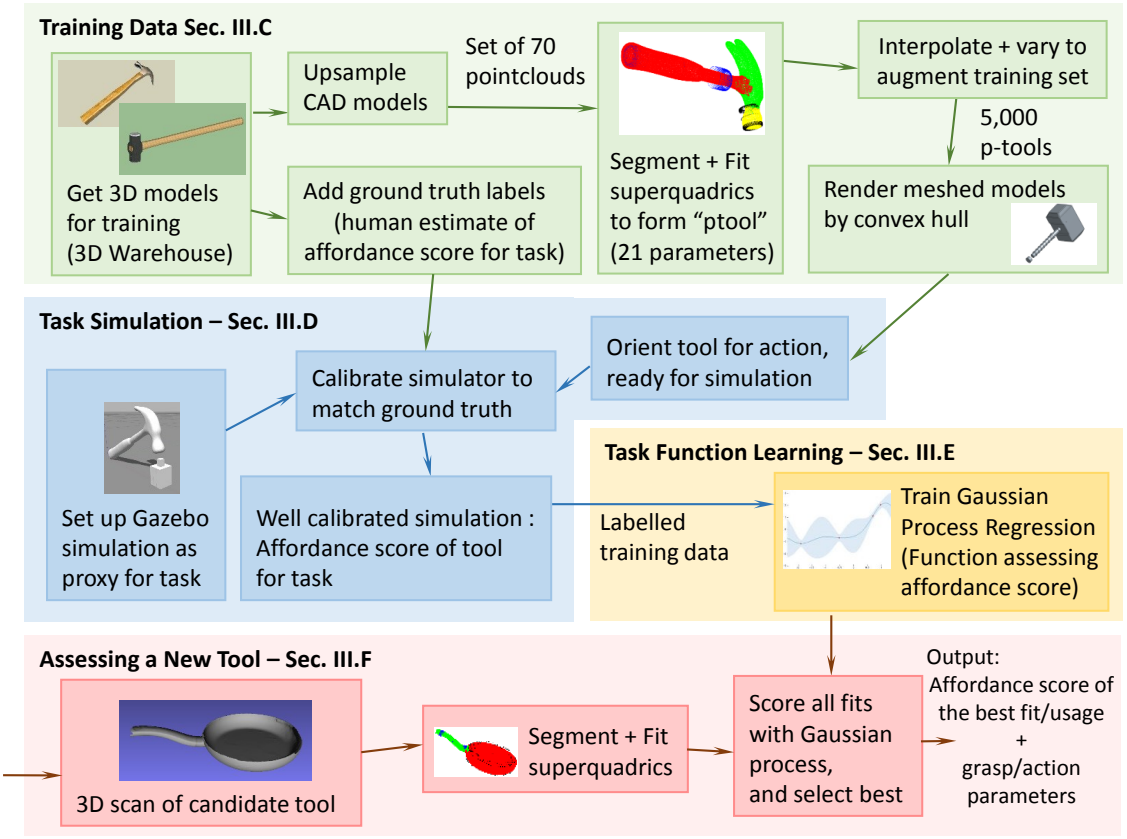


Fig. 2: Overview of the system; how it is trained and used.

Similarly to the one in [15], this is an inside-outside function to measure how close to the surface a given point $\mathbf{x} = (x, y, z)$ is. The parameters a_1, a_2 and a_3 control the scale and ϵ_1 the curvature. Each superquadric/paraboloid takes 13 parameters to represent: 5 for scale and shape, 2 for tapering 6 more for general orientation and position in space [9].

Once the superquadrics/superparaboloids are fitted to the point cloud's segments we run the p-tool extraction, which generates several possible p-tools for the pair of segments (corresponding to the possible ways to use the object). For each pair of segments, we alternately pick one of the point clouds to be the grasp and the other the action part. Then we get a vector that goes from the centre of the grasping part to the centre of the action part and also get the Euler angles representing the orientation of the action part: this requires 6 parameters, 3 for the vector and 3 for the action orientation. Additionally, we rotate the action superquadric/superparaboloid in steps of 90 degrees in each axis, and if the fitting score remains below a threshold, we also accept that fit as one possible way of orienting the action part. This way, we can get many (typically 12) possible good orientations of the action part in an efficient manner. The choice of 90 degrees is arbitrary and could be less, which would generate more p-tools for each point cloud. Finally, this orientation is relative to a canonical frame of reference for p-tools.

There are 7 parameters for the geometry of each tool-part (scale, shape and tapering of the superquadric), 6 for the geometric relationship between the parts, and 1 for the tool's weight leaving us with 21 parameters to represent a great variety of two-part tools. Note that if the point cloud has more than two segments we will extract every possible combination of two segments and use each for the grasp or action part.

C. Training Data

We gather *ToolWeb* (Fig. 5), a dataset of 70 3D synthetic meshes from 3DWarehouse[1]. Each point cloud goes through an automatic pipeline script in MeshLab[2] to up-sample the number of points and re-generate faces. In MeshLab we run two filters: Poisson-disk sampling to up-sample the points with input 5000; and Ball-Pivoting, with default options, to re-generate the faces. On top of that we run an automatic re-scaling of each point cloud to bring it within the minimum and maximum size expected for each class name (e.g. 'spatula', 'knife', 'hammer' etc.). We segmented the tools automatically (see Sec. IV-A.4) but also manually inspected and adjusted parameters to get a perfect segmentation. This manual step is not strictly necessary, but improves results and is feasible on a set of size 70. We then label the dataset with a discrete ground truth affordance score for each task: 1 - no good; 2 - could be used; 3 - good with effort; 4 - good.

[1]https://3dwarehouse.sketchup.com/
[2]http://www.meshlab.net/

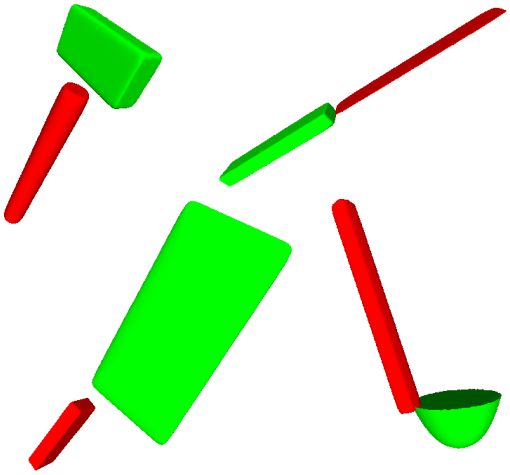


Fig. 3: Different p-tools: red and green represent different segments than can be used as grasp or action part. We can map a point cloud to one or more p-tools and, conversely, render a p-tool into a point cloud/mesh. Here shown: hammer; bread knife; Chinese knife and ladle (using a paraboloid for the bowl part)

For labelling we consider the best possible grasping and orientation for the tool. Once a starting set of tools is hand-labelled and those examples used to calibrate the simulator (see Sec. III-D.2) the system can autonomously label further tools.

Our system is trained on each task by simulating 5000 p-tools, which we get by augmenting the ToolWeb dataset; this augmentation step is described here. Extracting p-tools from ToolWeb (see Sec. III-B) resulted in 825 p-tools for 70 point clouds. We run a simple interpolation sampling to get more p-tools from the extracted ones. This works by sampling 100 points between two p-tools *iff* the Euclidean distance between them is smaller than the mean distance between each p-tool and its closest neighbour; a p-tool's closest neighbour is the one with smaller Euclidean distance to the p-tool (if there is more than one closest neighbour than a random one is chosen). Finally, we filter the sampled p-tools so that their values fall into a pre-specified range of valid tools.

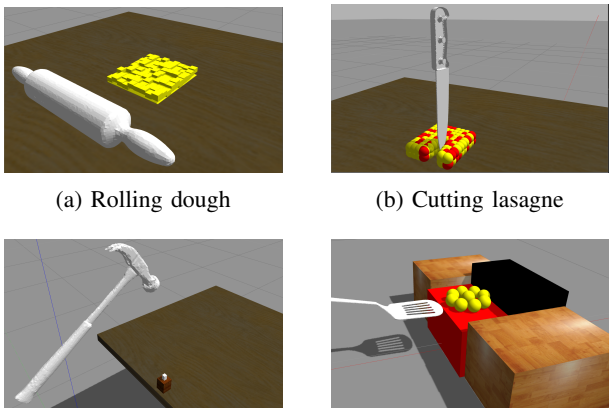
D. Task Simulation

We have four tasks that we simulate using Gazebo[3]: Rolling dough; cutting lasagne; hammering nail; and lifting pancake (Fig. 4).

At the end of a simulation the output is always a single real number, e.g. relating to: how much the nail went down for hammering nail; the variance of the dough in rolling dough; the median difference in distance for moved pieces in cutting lasagne; and for lifting pancake: how close the pancake is to a plane and how close the action part is to the pancake.

All p-tools go through our generic procedure for positioning and orienting for each task. For each task, we use a

[3]http://gazebosim.org/



(a) Rolling dough (b) Cutting lasagne

(c) Hammering nail (d) Lifting pancake

Fig. 4: Task simulation in Gazebo. Each simulation outputs a single real-valued score for the outcome.

cluster of 20 machines to run 3 simulations for each of the 5000 training p-tools (Sec. III-C). The 15000 simulations for each task took between $1-3$ hours to complete; tasks that simulate many elements, such as cutting lasagna, are slower

1) Tool Positioning: It is possible to orient and position any p-tool in the simulation environment by using a generic positioning procedure for each task. This allows us to run any number of required simulations automatically for a given set of p-tools. The p-tool parameters such as size and orientation of the action part and relationship between the parts determine, for each task, how to position it in the simulator. Each point cloud produces several p-tools both due to switching of grasp and action for each segment and orientation possibilities. Therefore, each point cloud is put in simulation in several different grasp-poses.

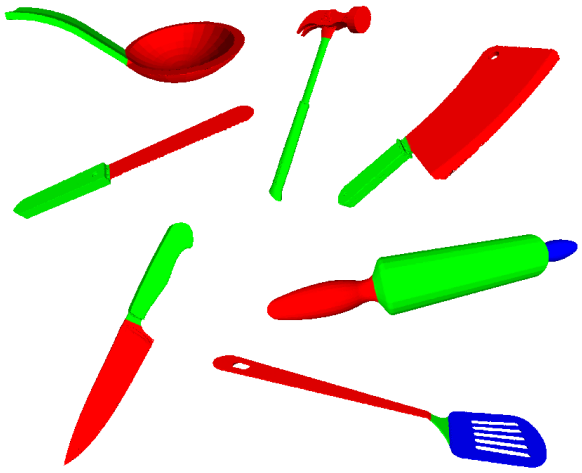


Fig. 5: ToolWeb dataset examples. Different colours represent different segments (colours here do not code any grasp or action part). Point clouds in ToolWeb are automatically up-sampled, re-meshed, re-scaled and segmented.

2) Automatic Calibration: ToolWeb (Sec. III-C) is used to calibrate the simulation of each task. We extract p-tools from

the segmented ToolWeb and try all of them in simulation getting a real-valued score for each one. The best score for each tool is used as a label for automatic calibration

In order to automatically calibrate the simulator we need to compare its output on ToolWeb with the labelled ground truth scores. We run a brute-force algorithm to discover the best thresholds for the real values to be discretised to a $[1, 4]$ range of affordance scores; we can do this by trying 3 threshold parameters (in between the minimum and maximum possible values for the task simulation) and making the value fall into each discrete score if it is less than one of the threshold parameters in order. This is run for all possible thresholds between the minimum and maximum possible real values from the simulation and given a pre-specified step size.

E. Training

In order to improve training, we perform a balancing of our training data to have an approximately uniform distribution over affordance scores. To accomplish this we have a pre-training step where we learn from the 5000 p-tools a function to map from a p-tool to a real-valued score. We then keep generating more p-tools (with the method in Sec. III-C), assessing them with the learned function and throwing away excess p-tools until we achieve 5000 p-tools with an approximate uniform distribution over affordance scores (i.e. no affordance score has more than 5% more p-tools than another one). These balanced 5000 p-tools are then simulated. After simulation we learn the final task function (Sec. III-A) on the balanced 5000 p-tools.

For learning the function in both pre-training and training we use Gaussian Process Regression with the well-known ARD squared exponential kernel, using Matlab's Statistics and Machine Learning implementation[4].

F. Affordance Assessment

Our system is able to assess the affordance of a new point cloud by extracting p-tools and assessing them using the trained GP for that task. We can either deal with segmented or unsegmented point clouds. When the point cloud is unsegmented we plant 10 random seeds used for starting points for fitting the superquadrics/superparaboloids and constrain the fitting to be at $\pm 20\%$ of the 'ideal p-tool' parameters. We get the ideal p-tool as one at random from all p-tools that have maximum value on the trained GP for the task. If the point cloud is segmented we extract the p-tools as in Sec. III-B and use these together with the ones from planting seeds. We pick the p-tool with the best score on the GP to be the one selected for the point cloud being assessed.

IV. EXPERIMENTAL EVALUATION

A. Test Sets

We evaluate our system on three datasets: ToolArtec, SmallArtec and ToolKinect. The first two obtained with the high-end Artec Eva 3D scanner and ToolKinect with the low-end Kinect 2. Artec Eva 3D is a structured light scanner and provides a much higher resolution than Kinect 2. It also performs much better on shiny objects such as glossy mugs and metal knives etc. We constructed two datasets using different scanners in order to better evaluate our technique and its dependence on high-resolution and full visual input. Both datasets contain only point clouds of real objects and also labels for each task. Labels were obtained by trying out many of the objects in the real world tasks, and for the remainder, extrapolating from very similar tools that had been tried.

1) ToolArtec Dataset: We scanned 50 full-view point clouds of real household objects with the Artec Eva 3D scanner, using real-time fusion and moving the scanner around all accessible sides of the object while it was fixed on the ground, often in an upright orientation. All tools are hand-labelled for the four different tasks (Sec. III-D). More details about this dataset can be found in [9].

2) ToolKinect Dataset: We scanned a subset of 13 tools present in ToolArtec. Each tool was positioned alone in a table and scanned by moving Kinect 2 in an arch over the tool to simulate what a robot could do by moving a mounted sensor (e..g looking over the tool with its head or arm-mounted scanner). We then applied a RANSAC method for fitting and removing the plane (i.e. table) and also everything below it. Labels are inherited from ToolArtec. More details about this dataset can be found in [9]

3) SmallArtec Dataset: This is simply the subset of the ToolArtec that has the 13 tools of ToolKinect. This allows us to compare the results on the same tools scanned by two different scanners.

ToolKinect is small because Kinect is unable to obtain models of any shiny tools, e.g. metal or glossy ceramic, or any small parts, e.g. chopstick or thin tool shaft. We gave up on trying to get a large set of Kinect tools and include a small set just to test if our technique also works on lower quality sensor data. ToolKinect is too small to provide a thorough evaluation on a representative range of kitchen tools. As sensor technology improves and costs reduce we expect that higher quality data will be the norm, and hence the ToolArtec results are more relevant for roboticists.

4) Automatic Segmentation for Test Sets: We segment all point clouds in each test set using *CPC* (Constrained Planar Cuts) segmentation [16] which is part of the PCL library [17]. CPC makes use of local concavities to perform cuts through the object and partition it in segments; it requires a set of three main parameters: seed resolution, voxel resolution and smoothing. We perform the automatic segmentation in two steps: segmentation and filtering.

During the first step we modify the original CPC code to loop over the parameters: seed resolution going from 0.01 to 0.015 in a 0.001 step size; voxel resolution going from 0.001 to 0.005 in a 0.0001 step size; and smoothing going from 0 to 50 in a 2 step size.

The brute-force stepping may produce bad segmentation outcomes since it is searching over a large space of parameter

[4]https://uk.mathworks.com/help/stats/gaussian-process-regression-models.html

TABLE I: Comparison of our system with ScW [2] on ToolArtec

	Metric 1		Random ratings			Accuracy Aff. score 1		Accuracy Aff. score 2		Accuracy Aff. score 3		Accuracy Aff. score 4	
	ours	ScW	μ	σ	$p < 0.05$	ours	ScW	ours	ScW	ours	ScW	ours	ScW
ToolArtec	0.89	0.84	0.72	0.04	0.77	0.55	0.54	0.56	0.27	0.33	0.43	0.52	0.23

values. To overcome this, for each segmentation outcome we perform a superquadric fitting to each segment and keep only those outcomes below a pre-specified fitting score threshold. Segmentation is necessary for running Schoeler and Wörgötter's system[5] [2] with which we compare our results. Here we call this system *ScW* and it needs the segments to consider them as parts of the object.

B. Metrics

In all of our experiments we are dealing with four possible scores for the affordance (Sec. III-D) going from 1 to 4. We have two metrics for measuring performance: accuracy and Metric 1. Accuracy is calculated separately for each of the four affordance scores 1 to 4. Accuracy is the proportion of tools that a system guessed precisely the right affordance score for. Metric 1 measures how well the system estimates affordances and is calculated as follows:

$$m_1 = \frac{(c-1)^2 - \frac{1}{n}\sum_{i=1}^{n} |\mathbf{s}_i - \mathbf{g}_i|^2}{(c-1)^2} \tag{2}$$

where $c > 1$ is the number of possible discrete scores (in our case 4), n is the number of scores (size of $\mathbf{s}$ and $\mathbf{g}$), $\mathbf{s}$ is the vector containing the system's scores, $\mathbf{g}$ is the vector containing the ground truth scores. Metric 1 is always between 0 and 1 and quadratically penalises the distance between the scoring vector and the ground truth.

C. Experiments

We compare our system with that of ScW [2]. We train it for each task with 15 objects[6] of affordance score 4 taken from the ToolWeb dataset (Sec. III-D) and test in on ToolArtec, ToolKinect and SmallArtec, comparing with our own results. The ScW system trains an SVM on ideal tools for each task and evaluates newly presented tools by scoring how close they are to the model it has generalised of the ideal tool for that task.

In our comparisons we also include a baseline of random ratings of the tools. For each task, we randomly score (1-4) each tool in the dataset 10^5 times, each time computing Metric 1 for the random ratings and ground truth for the task. This results in 10^5 Metric 1 values for each task that roughly follow a normal distribution. We take the mean and standard deviation of the values and, assuming a normal distribution, calculate the p-value ($p < 0.05$) for each task. That is, the Metric 1 value above which there is only a 5% or less chance of achieving that by randomly scoring each tool.

We present the mean (μ), standard deviation (σ) and the p-value ($p < 0.05$) in all tables (I, II and III). In the case of random ratings for an entire dataset (Tables I and II), we consider the mean and standard deviations for the dataset to be the respective average for the means and standard deviations of all tasks. The average and standard deviation over all tasks are used for the normal distribution that we use to calculate the p-value for a whole dataset. Finally, for Table I we also present the accuracy in each affordance score value in the $[1-4]$ range, that is, the number of tools guessed correctly for the value over the total number of tools.

[5]The authors kindly provided us with the full Matlab code.

[6]This number was recommended as a minimum by Markus Schoeler.

TABLE II: Overall Metric 1 on all test sets

	Metric 1		Random ratings		
	our	ScW	μ	σ	$p < 0.05$
ToolArtec	0.89	0.84	0.72	0.04	0.77
SmallArtec	0.88	0.85	0.79	0.05	0.85
ToolKinect	0.84	0.78	0.75	0.03	0.80

TABLE III: Metric 1 on ToolArtec per task

	Metric 1		Random ratings		
	ours	ScW	μ	σ	$p < 0.05$
Hammering Nail	0.94	0.79	0.77	0.03	0.82
Lifting Pancake	0.87	0.87	0.73	0.05	0.81
Rolling Dough	0.91	0.86	0.66	0.04	0.72
Cutting Lasagne	0.81	0.83	0.70	0.05	0.78

V. DISCUSSION

Looking at the overall results for the three datasets in Table II we see that our system always performs significantly better than chance, and ScW does so only on ToolArtec. The drop in performance when moving from Artec to Kinect (i.e. compare Kinect to ArtecSmall) is significant, but Kinect data is still good enough for reasonable affordance assessments with our technique. Superquadric fitting is more robust to noise and missing data (e.g. underside of tool) than the histogram approach. As can be seen in Table III we are better or equal to ScW at every task except for Cutting Lasagne, which may be explained by the superquadric/superparaboloid fitting generating p-tools that are too thick, but this requires further study. As shown in Table I we are better than ScW1 on the accuracy for all affordance scores except score 3.

It is important to note that ScW is only trained on canonical uses of tools for tasks. ScW can sometimes map parts and find a creative use (e.g. in cutting lasagne with several tools such as scraper or spatula), however it more often misses creative uses such as frying pan or rolling pin for hammering nail, or rice spoon for cutting lasagne. This is understandable since it only considers the shape histogram using angles between normals, it does not consider weight nor absolute size. This is evident in Table I where we see that ScW has a very poor success at affordance score 4.

One interesting observation out of the training was that many times the Gaussian Process Regression could identify a small number of parameters that dominated the prediction. In future work we intend to devise more complex tasks on which to further test the generalising capability of our system. Another possible direction is making the data augmentation more principled by using recent methods of manifold learning [18] on the 21-dimensional space of p-tools. Finally, we also intend to make our pipeline for gathering a training set fully automatic by having automatic segmentation such as the one we used for the test set.

VI. CONCLUSION

In this paper we presented a system capable of finding good ways of grasping and orienting unknown tools for a task and giving them an affordance score. We achieved this by training from Web models. We gathered and segmented a dataset ToolWeb of 70 point clouds of common household tools from the Web and fitted geometric models to each segment in order to abstract different ways of grasping and orienting the tools. Our tool abstraction, p-tool (Sec. III-B), captures the grasp and action part, the relationship between them (orientation and distance) and also the tool's weight. We augment the data of extracted tool abstraction by interpolating between similar p-tools and train 5000 of them in simulations for each task.

We introduced three datasets, ToolArtec, ToolKinect and ToolWeb of common household tools for tasks. Our results showed a significant performance improvement relative to the closest competitor approach, but more than this we output information that the competing system does not: about how to orient the tool for the task. This additional information is crucial to facilitate our semi-automatic pipeline to process Web point clouds. This pipeline is at the core of our system; it can semi-automatically re-scale, re-sample and re-mesh Web models and abstract their parts into possible ways of grasping and orienting. These different ways of using the tools can be then be all simulated in order to get a large number of labeled tools for training. In our current times of data-hungry approaches we believe this part of our system could be of significant interest to the community.

VII. ACKNOWLEDGEMENT

Thanks to: UoAs ABVenture Zone, N. Petkov, K. Georgiev, B. Nougier, S. Fichtl, S. Ramamoorthy, M. Beetz, A. Haidu, J. Alexander, M. Schoeler, N. Pugeault, D. Cruickshank, M. Chung and N. Khan.

REFERENCES

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

[2] M. Schoeler and F. Wörgötter, "Bootstrapping the Semantics of Tools: Affordance analysis of real world objects on a per-part basis," *IEEE Trans. on Autonomous Mental Development*, vol. 8, no. 2, pp. 84–98, 2016

[3] M. Krainin, P. Henry, X. Ren, and D. Fox, "Manipulator and object tracking for in-hand 3D object modeling," *The International Journal of Robotics Research*, vol. 30, no. 11, pp. 1311–1327, 2011.

[4] D. Kraft, R. Detry, N. Pugeault, E. Baseski, F. Guerin, J. H. Piater, N. Kruger, E. Baseski, F. Guerin, J. H. Piater, and N. Kruger, "Development of Object and Grasping Knowledge by Robot Exploration," *Autonomous Mental Development, IEEE Transactions on*, vol. 2, no. 4, pp. 368–383, dec 2010

[5] K. Welke, J. Issac, D. Schiebener, T. Asfour, and R. Dillmann, "Autonomous acquisition of visual multi-view object representations for object recognition on a humanoid robot," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, may 2010, pp. 2012–2019

[6] A. Dehban, L. Jamone, A. R. Kampff, and J. J. Santos-Victor, "Denoising auto-encoders for learning of objects and tools affordances in continuous space," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2016-June, pp. 1–6, 2016.

[7] C. Wang, K. V. Hindriks, and R. Babuska, "Effective transfer learning of affordances for household robots," in *4th International Conference on Development and Learning and on Epigenetic Robotics*, no. 1. Genoa: IEEE, oct 2014, pp. 469–475

[8] 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, oct 2016, pp. 2765–2770

[9] P. Abelha, F. Guerin, and M. Schoeler, "A model-based approach to finding substitute tools in 3D vision data," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, may 2016, pp. 2471–2478

[10] T. Mar, V. Tikhanoff, G. Metta, and L. Natale, "Multi-model approach based on 3D functional features for tool affordance learning in robotics," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, vol. 270273, no. 270273. IEEE, nov 2015, pp. 482–489

[11] J. Sinapov and A. Stoytchev, "Detecting the functional similarities between tools using a hierarchical representation of outcomes," in *2008 7th IEEE International Conference on Development and Learning*. IEEE, aug 2008, pp. 91–96

[12] V. Chu, T. Fitzgerald, and A. L. Thomaz, "Learning Object Affordances by Leveraging the Combination of Human-Guidance and Self-Exploration," in *The Eleventh ACM/IEEE International Conference on Human Robot Interaction*. New Zeland: IEEE Press, 2016, pp. 221–228.

[13] A. Agostini, M. Javad Aein, S. Szedmak, E. E. Aksoy, J. Piater, and F. Worgotter, "Using structural bootstrapping for object substitution in robotic executions of human-like manipulation tasks," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 2015-Decem. IEEE, sep 2015, pp. 6479–6486

[14] W. Mustafa, M. Waechter, S. Szedmak, A. Agostini, D. Kraft, T. Asfour, J. Piater, F. Wörgötter, and N. Krüger, "Affordance Estimation For Vision-Based Object Replacement on a Humanoid Robot," in *Proceedings of ISR 2016: 47st International Symposium on Robotics*, jun 2016, pp. 1–9.

[15] A. Jaklič, A. Leonardis, and F. Solina, *Segmentation and Recovery of Superquadrics*, ser. Computational Imaging and Vision. Dordrecht: Springer Netherlands, 2000, vol. 20

[16] M. Schoeler, J. Papon, F. Wörgötter, and F. Worgotter, "Constrained planar cuts - Object partitioning for point clouds," in *IEEE Conference on Computer Vision and Pattern Recognition CVPR*. IEEE, jun 2015, pp. 5207–5215

[17] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, may 2011, pp. 1–4

[18] J. Wang, Z. Zhang, and H. Zha, "Adaptive Manifold Learning," in *Proceedings of the 17th International Conference on Neural Information Processing Systems*, ser. NIPS'04. Cambridge, MA, USA: MIT Press, 2004, pp. 1473–1480