Learning End-Effector Orientations for Novel Object Grasping Tasks

Benjamin Balaguer Stefano Carpin

***Abstract*— We present a new method to calculate valid end-effector orientations for grasping tasks. A three-layered hierarchical supervised machine learning framework is developed that is both fast and accurate. The algorithm is trained with a human-in-the-loop in a learn-by-demonstration procedure where the robot is shown a set of valid end-effector rotations. Learning is then achieved through a multi-class support vector machine, orthogonal distance regression, and nearest neighbor searches. We provide results acquired both offline and on a humanoid torso and demonstrate the algorithm generalizes well to objects outside the training data.**

I. Motivation and statement of contribution

In this paper we consider the problem of grasping an unknown object with a robotic hand mounted on an anthropomorphic arm. Due to its fundamental nature, this problem has received considerable attention in the past. However, advances in robot hardware and machine learning techniques have given a new perspective to this challenge. In particular, a relatively recent paper by Saxena et al. [9] presented an innovative machine learning method to identify good grasping points starting from the picture of a previously unseen object. The method is feature-based: given an object's image, feature vectors are used to classify each pixel as being a good or bad grasping point. Classification is performed using logistic regression, where the logistic regression coefficients are learned from a labeled training set containing thousands of images. More recently, we have demonstrated that this technique can be accelerated using dimensionality reduction and executed in real time [1]. In this contribution, we extend this feature-based approach and consider *plan execution*. More precisely, after a pixel is selected as a good grasping point, one is left with the problem of correctly executing the grasp at the chosen point. The pixel is associated with a point $p_g = (x, y, z)^T$ whose three dimensional Cartesian coordinates are obtained through stereo camera processing. Given p_g, we address the problem of determining the end-effector orientation to grasp the object. More formally, indicating with T_E the 4×4 transformation matrix of the end effector, the problem is cast as determining a function $f : \mathbb{R}^3 \rightarrow SE(3)$, where $SE(3)$ is the Special Euclidean group of dimension 3. Evidently, the function f maps points from a three-dimensional space into elements of a six-dimensional manifold. Once T_E is calculated, inverse kinematics can be applied to get one or more configurations giving the desired end-effector pose. Therefore T_E is the missing element to map images of objects to be grasped into joint angles for the robot to grasp the object. If the geometry of the object to be grasped is known a-priori one may pre-compute good grasping points and the associated end-effector poses. However we are here interested in situations where the robot is lacking this knowledge. For this reason, and in the same spirit of [1], [9], we propose a learning algorithm to compute T_E. However, since the position component of T_E is given by p_g, the paper focuses on the 3×3 rotation component of T_E.

The rest of the paper is organized as follows. In Section II we shortly revise related literature. Our proposed algorithm is detailed in Section III. Section IV describes the robotic system used to validate our method, and a battery of experiments substantiating its performance. Finally, in section V we outline the lessons learned and summarize possible directions for further research.

II. Related work

Research on grasp planning is divided between model-based and feature-based algorithms. Model-based techniques capture the geometric model of the object and match it against a database labeled with grasps. Conversely, feature-based techniques attempt to extract features from object views and match them against a database of features. Model-based approaches have captivated much of the grasp planning research but do not scale well to objects not encompassed by the database, so we will not further discuss them here.

SIFT [7] and SURF [2] are popular feature extraction methods popular for their scale and rotation invariance. These feature extraction methods have been applied to robotic grasping, where a database of objects, encoded by their corresponding features, is used to determine the location and rotation of the object the robot wants to grasp. The success of SIFT/SURF is directly dependent on the object's texture and, as such, difficult to extend to novel objects. In [8] Remazeilles et al. present a feature-based method for visual servoing. The authors devise a solution for novel objects (i.e. no information about the objects is known in advance). This method, however, requires an operator to draw a box around the object to be grasped. Another approach to feature-based grasping is presented by Kroemer et al. [4], where an online learning method is initialized with prior knowledge. Objects are represented by Early Cognitive Vision descriptors. The robot is then taught a series of grasps for a given object by a human. These grasps can be tried by the real robot and new knowledge about the outcome is incorporated. Experimental results show the robot quickly dismisses its observations and comes up with its own grasps, i.e. the active learning method presented does not put enough importance

The authors are with the School of Engineering, University of California, Merced, USA. This work is partially supported by the NSF under grant BCS-0821766.

on human teaching, which can be thought of as the ultimate set of positive examples. In our view, the most interesting publication to solve problems associated with feature-based grasp planners is by Saxena et al. [9]. The authors introduce the idea of image features as a representation of good grasping points learned from massive amounts of synthetic training data. Fueling an interesting synergy with machine learning, grasp planners have recently seen some interest in dimensionality reduction due to the high number of DOFs encompassed by robots. In [3] the authors exploit the finding that a two-dimensional subspace accounts for 80 percent of the variance in hand posture. This lower subspace can then be exploited to find an accurate pre-grasp, which, in turns, is used to grasp the object. Their method however is concerned with finger posture and not hand placement. In [1], we have ourselves looked into the power of dimensionality reduction for grasping tasks by modifying [9] to make it more efficient using feature selection.

III. Learning how to grasp

A. Design Choices

The most severe drawback of feature-based methods proposed so far comes from the fact that they solve half of the grasping problem. More specifically, given a single image from a robot camera, the aforementioned algorithm [9] finds the best grasping point $p_g = (x, y, z)^T$. It does not, however, consider the problem of correctly orientating the end-effector to maximize the chance of a successful grasp. Consequently, in this paper we propose a method to compute an appropriate end-effector rotation. We note the difficult constraint of developing the algorithm with limited sensory information, namely the fact that we are using a single stereo image and do not look at the object from different angles. Additionally, we emphasize a manipulator-independent algorithm that is scale, rotation, and translation invariant for the objects - algorithmic properties that are central to the algorithm development. Conversely to model-based techniques, feature-based algorithms solely rely on images and lack model databases, thus making it very difficult to use grasp quality measures to predict the goodness of a grasp plan. We use a supervised machine learning algorithm, trained using a learn-by-demonstration technique. We note the inherent difficulty in solving a continuous multi-variable problem with many-to-many relationships. Indeed, we have many-to-many mappings since a given image pixel location can map to different valid wrist orientations, while the same wrist orientations might map to different image pixel locations. The biggest obstacle to learning in this context comes from the many-to-many mapping in continuous space, where most learning algorithms (e.g. radial basis functions, support vector machines, decision trees, etc...) deteriorate to a slow nearest neighbor search due to the high data dimensionality. Therefore we introduce a hierarchical learning method comprised of an object classification layer, followed by the calculation of the object's rotation, and finalized by a nearest neighbor search to find the best end-effector orientation. While the details are left to the next subsections, it is important to note the hierarchical nature of the approach allows the algorithm to aggressively prune the search space at run time, which results in a much faster 3-dimensional nearest neighbor search. Furthermore, each layer of the algorithm was chosen for its attractive tradeoff between speed and accuracy.

B. Acquiring Training Data

We have made the training data acquisition, which requires a human-in-the-loop, as automatic as possible. A diverse set of 6 objects shown in Figure 1 is used for training. For each object, we account for rotation-invariance by taking 36 images, with each image representing a different rotation around the axis perpendicular to the plane where they lay. We take the images uniformly around the full 360-degree spectrum, thus having one image every 10 degrees. It is important to mention that when we use the expression *rotation invariance*, we are specifically referring to a one-dimensional rotation (the one around the axis perpendicular to the table). Even though such definition might seem to be an over-constraint at first, we point to the fact that typical household objects can stably stand in just a few ways. Full 3-dimensional rotation-invariance can then be achieved by acquiring additional images for each of the stable object positions, a step that we omit for sake of clarity.

Fig. 1. The six objects used to generate training data. In the results, we refer to each object, from left to right, as a number between 1 and 6.

For each object class c, we have a set of stereo images, S_I^c, such that $S_I^c = \left[I_1^c, I_2^c \cdots I_n^c\right]$ where c is an integer between 1 and 6 and $n = 36$. The use of a stereo camera allows for a conversion of images into point clouds which we label S_P^c, where $S_P^c = \left[P_1^c, P_2^c \cdots P_n^c\right]$. Each point cloud P_i^c is inherently noisy while including both the object and the surrounding environment (i.e. the table). We remove the table from the point cloud by using orthogonal distance regression to find the best fit plane and removing the points close to the plane. We then remove any additional noise using a quartile range denoiser [6]. The two-step denoising process, an example of which is shown in Figure 2, is fast, requires no human supervision, and works well in practice. From now onwards P_i^c refers to a denoised point cloud.

We rely on a human showing the robot how to grasp some objects, the knowledge of which can be exploited by the robot to grasp both trained and novel objects. We refer to a trained object as an object used to train the robot and a novel object as an object not used in the training stage. Specifically, we put the robotic manipulator in a gravity-compensated state such that a human can freely move it as he/she pleases. For each object view, i, for which we have an image, I_i^c, the human moves the manipulator to a valid grasp position and the robot configuration, q, is recorded. The

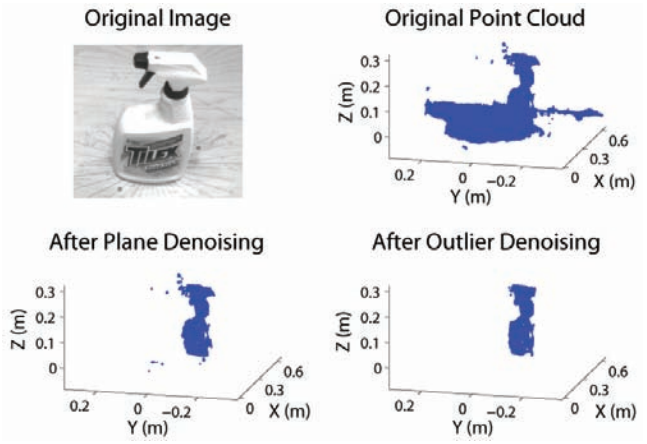

Fig. 2. Graphical representation of the denoising process. The upper-left image shows the object. The upper-right, lower-left, and lower-right plots show the unmodified, plane denoising, and outlier denoising point clouds, respectively.

process is repeated m times, yielding a set of configurations $q^c_{i,j}$ where $j = 1, 2, \cdots m$. We introduce the j variable to denote the fact that for a given image, I^c_i, there exists many robot configurations with valid grasps throughout the surface of the object. From a practical standpoint, the human moves the manipulator into k good grasping positions, covering as much of the object as possible. This step is the most time consuming and, consequently, we only record this information for 12 of the 36 views, uniformly spaced by 30-degree increments. Readers are invited to view videos of this process on our website[1], where the entire data set used for this paper can also be downloaded.

C. Classifying Objects

We start the hierarchical process by classifying the object to grasp. We approach the classification problem with a multi-class SVM [5], for their speed and accuracy. In this paper, we have 6 discrete categories, one for each of our trained objects. More specifically, for k classes ($k = 6$ in our case) two strategies can be used: 1) one-against-all, where k SVMs are trained to separate one class from all the others, or 2) one-against-one, where $\binom{k}{2}$ SVMs are trained to separate each class from another. We chose to use the one-against-one method, after empirically determining that it produced more accurate results. For the sake of completeness, we mention that we use a polynomial kernel for its speed and accuracy (e.g. 10 times faster than a Gaussian Radial Basis Function kernel).

The SVM requires a constant-sized feature vector for training and classifying. Since we want to use point clouds to differentiate between objects, and each will have a different number of points, we re-factor the original point clouds, P^c_i, into fixed-sized feature vectors, F^c_i. Accounting for translation-invariance, we generate a new point cloud, P'^c_i, by subtracting the mean from every point in the cloud. next, we generate our feature vector by encompassing the object, in image space, into a two-dimensional matrix of fixed sized. Since objects will vary in size, the matrix is scaled uniformly to maintain its given size. To be specific, we use a matrix size of 50×50 which automatically and uniformly scales to any object size. Each cell in the matrix encompasses a number of pixels, each of which being a 3-dimensional Cartesian coordinate in P'^c_i. Our feature vector, F^c_i, is the average Cartesian coordinate of all the pixels within each cell. For our 50×50 grid, we have a feature vector size of 7500 (2500 cells, each comprised of the average Cartesian coordinate). This encoding accounts for two properties. First, each point cloud can be encoded with a same-sized feature vector. Second, the feature encoding indirectly achieves scale invariance since the grid automatically adjusts to changes in size. Since the next components of the hierarchical method depend on the classification accuracy, we provide results showing the validity of the SVM classification, before describing the next stage. In our first experiment we train our algorithm with all training data except for one, and try to classify the one that was not included. We repeat this process removing and classifying a different view every time, until all views have been classified. With this method we obtained a classification accuracy of 97.69%.

The next set of experiments are performed to test the scale, translation, and rotation invariance properties of the algorithm. This time we acquire new data, with each of the 6 objects at 10 random locations and rotations. More specifically, we train the multi-class SVM on the full training data set and classify the new images. Evidently, this is a harder experiment due to the translation changes, different viewpoints from the robot's perspective which create different scales for the objects, and various rotations. Results are shown in Table I, with an overall classification accuracy of 88.33%. The emphasis of our work is on grasping novel

		Actual					
	Object #	**1**	**2**	**3**	**4**	**5**	**6**
Predicted	**1**	**9**	1	0	0	0	0
	2	0	**9**	0	2	0	0
	3	0	0	**9**	0	0	1
	4	0	0	0	**8**	0	0
	5	0	0	1	0	**9**	0
	6	1	0	0	0	1	**9**

TABLE I

CONFUSION MATRIX FOR THE EXPERIMENT PERFORMED ON TRAINED OBJECTS OF DIFFERENT SCALES, TRANSLATIONS, AND ROTATIONS.

objects, so it is crucial for the algorithm to grasp objects that are not in the training set. The idea behind novel object grasping stems from the observation that different objects can be grasped similarly based on shared geometry. With our SVM classification we should inherently be able to detect similar objects and we test our algorithm with the novel objects shown in Figure 3. For this experiment, we have chosen novel objects that are fairly similar to the trained objects, in order to come up with a quantitative classification success rate. The objects are indeed different from the trained ones by their texture, size, or geometry. We

[1]https://robotics.ucmerced.edu/Robotics/Humanoids2010

point the readers to Section IV for experiments involving highly different novel objects. For each novel object, we

Fig. 3. The six objects used as novel objects. The row order in which they are presented corresponds to our expected classification in Figure 1. We refer to each object, from left to right, as a number between 1 and 6.

acquire 10 images, placed at random locations and rotations, further testing scale, rotation, and translation invariance. The SVM is trained on all of the training data and each image is classified. The result of this experiment is shown in Table II, which shows an overall classification rate of 85%. Even though we present the results in a confusion matrix, we emphasize that results presented are based on our prior belief of how the novel objects should be classified. In some cases, this prior belief is evidently correct (e.g. the drill, the spray bottle, the water bottle, the mug) while, in other cases, it is not obvious (e.g. the DVD case matching the coffee can and the shampoo bottle matching the bottle). This observation is especially evident with the DVD case, which is classified 60% of the time as a coffee can (i.e. our belief, due to the rectangular nature of both objects) and 40% of the time as a drill. It is important to note that while our expected classification rate is at 85%, our grasping success rate will be better because misclassifying an object will not necessarily result in a wrong wrist orientation. It simply means that a different object will be used to find the best wrist orientation.

	Actual (Trained)						
	Object #	**1**	**2**	**3**	**4**	**5**	**6**
Predicted (Novel)	**1**	**6**	1	0	1	0	0
	2	4	**9**	0	1	0	0
	3	0	0	**9**	0	1	0
	4	0	0	0	**8**	0	0
	5	0	0	0	0	**9**	0
	6	0	0	1	0	0	**10**

TABLE II

CONFUSION MATRIX FOR THE EXPERIMENT PERFORMED ON NOVEL OBJECTS OF DIFFERENT SCALES, TRANSLATIONS, AND ROTATIONS.

D. Determining Object Rotation

Thanks to the object classification we can focus on a single object (i.e. 36 images, 12 of which having the robot configurations), rather than all of them (i.e. 216 images, 72 of which having hundreds of robot configurations). In the second stage of the algorithm, we determine the correct object orientation using the closest matching object in our training data. We recall that we are only interested in a one dimensional rotation for reasons described in the Design Choices section. Based on this assumption, we came up with a solution involving plane fitting to deduce the most likely object rotation. More specifically, we exploit the point clouds to find the best-fit plane, through orthogonal distance regression, for all of our data. Given two planes with their normal vectors, N_1 for an object in our training data and N_2 for the object to grasp, we calculate the angle between two, $\theta = \arccos\left(\frac{N_1^T N_2}{||N_1||||N_2||}\right)$. This process, an example of which is shown in Figure 4, is efficient since the plane fitting can be done offline for the training data. We use this plane fitting process to find the nearest neighbor, in terms of rotation, in our training data. We note that we cannot differentiate objects that are rotated by 180 degrees, since the best fit planes is the same. We address this issue with image moments calculated for the left and right sides of our object image. More specifically, we calculate the first and second degree image moments, for each side, and use the Root Mean Square Error (RMSE) between an object in our training set and the object to grasp. Summarizing, we determine the object rotation as follows. First, find the angles between the best-fit planes in our training data and the one to grasp. Second, find the 6 training objects with the lowest angles and calculate the RMSE image moments. Then, we choose the one with the lowest RMSE. We again run an experiment

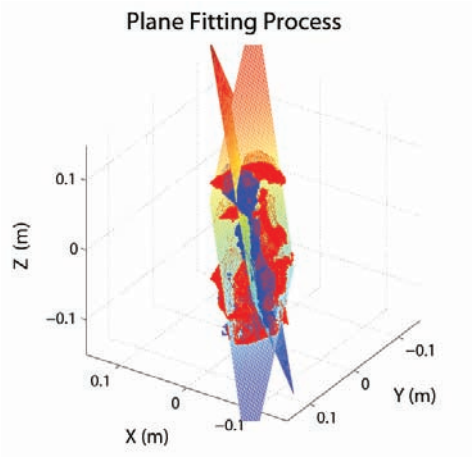

Fig. 4. Example of the plane fitting process, with point clouds for two spray bottles (blue and red) along with their equivalent best-fit planes.

to determine the validity of this algorithmic stage before moving to the next one. For each object class we remove one object from our training data and try to find its closest neighbor as per the aforementioned technique. We repeat this process removing a different object for each class until all of the views have been processed. Before analyzing the results encompassed by the histogram displayed in Figure 5, we note that different object symmetries will result in different outcomes. There are three possible cases of symmetry to take into account. First, some objects will be fully symmetric (e.g. water bottle), and rotations around the axis perpendicular to the table will not alter the object view. For such objects, the experiment is meaningless since any rotation would be dealt the same way by the manipulator. For this reason, we do not include the water bottle as part of our results. Second, some objects will be partially symmetric (e.g. coffee can, bottle, mug), where only rotations that are 180 degrees apart

will result in the same object view. In that case, we cannot differentiate between rotations that are 180 degrees apart and can equally use either. Third, some objects will be completely asymmetric (e.g. drill, mug, spray bottle), where they never look the same under different rotations and we need to find the closest match in our training set. We note that the mug can be either partially symmetric or completely symmetric depending on whether or not the handle is occluded. Keeping this information in mind, Figure 5 shows a 87.5% rate of finding the closest neighbor (the one within a 10 degree difference) for completely asymmetric objects and a 75% rate of finding the closest neighbor for partially symmetric objects (taking into account that a 180 degree rotation yields the same object view).

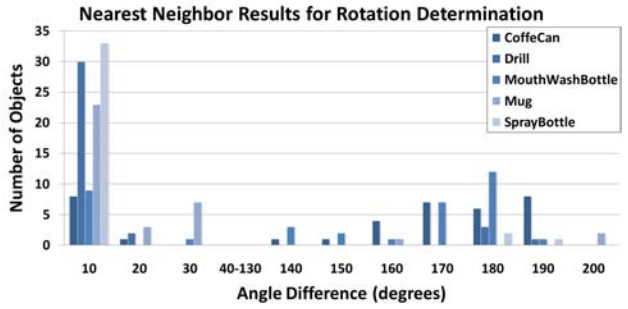

Fig. 5. Histogram showing the results in finding the nearest neighbor, in terms of rotation. The x-axis measures how far, in degrees, the closest neighbor is from the actual value (i.e. it should be 10 or 180 degrees for the algorithm to work). The y-axis is the number of trials for which that value occurred (36 is the maximum value).

E. Calculating End-Effector Rotation

At this point we have found both the closest object class and its rotation in our training data. We now need to determine the correct end-effector orientation using the input pixel location in image space. The difficulty of this step lies in the fact that we only have example grasps for 30-degree rotation steps (i.e. for 12 of the 36 views). First, we convert the pixel location into a Cartesian coordinate, thanks to our stereo camera, and offset it by the point cloud's mean M to yield a zero-meaned point L. We then rotate the point, using a rotation matrix R_z that takes into account the angle calculated in the previous step. This process generates a point, $L' = R_z(L - M) + M$ that is aligned with the closest object in our training set with grasp examples. After converting all of the configurations from our training, $q^c_{i,j}$, to Cartesian coordinates, using forward kinematics, we use a neighbor search in 3 dimensions to find the closest match to our input point, which will be labeled with a configuration q_l. Using forward kinematics, we convert q_l to the end-effector rotation matrix, R_{nn}. Last but not least, we need to rotate our end-effector to match the original object, straightforwardly achieved by $R_e = R'_z \times R_{nn}$, where R'_z is the rotation matrix rotating in the inverse direction of R_z. The resulting end-effector rotation R_e can be used with the input pixel location p_g to form T_E. Finally, T_E is fed to an inverse kinematics solver to get the manipulator to grasp the object.

IV. Experimental results

The robotic platform used to evaluate the proposed algorithm is shown in Figures 6, 7, 8, and 9. *George* is a humanoid robotic torso composed of two Barrett arms mounted sideways. Each arm is equipped with a Barrett Hand, and the torso is completed by a BumbleBee stereo camera mounted on two servos allowing for motions about the yaw and pitch angles. Each arm has 7 degrees of freedom (four in the arm and three in the wrist), whereas the three fingered hand provides additional 4 degrees of freedom (one commanding the spread of the fingers and three controlling the closure of each finger). All the software computing inverse and forward kinematics and image acquisition and processing has been developed in house and is written in C++. The rest of the algorithm is implemented in MatLab. We have performed a large amount of experiments, which we highlight in this section through accuracy measures and representative pictures. We encourage readers to visit our aforementioned website for videos showing the experiments running on our robot in real time. For all the experiments presented herein, we simply close the fingers of the hand to grasp the object and count any form-closed grasp as being correct, regardless of whether or not the object can physically be picked up (e.g. due to slippage or payload restrictions). In our first experiment, with examples shown in Figure 6, we place each of our 6 trained objects at 10 random rotations, manually input a pixel location, and let the algorithm try to grasp the object. The overall success rate is 81.66%, where the bottle, drill, and coffee can had the highest (90%) and the mug and bottle had the lowest (70%). We repeat our

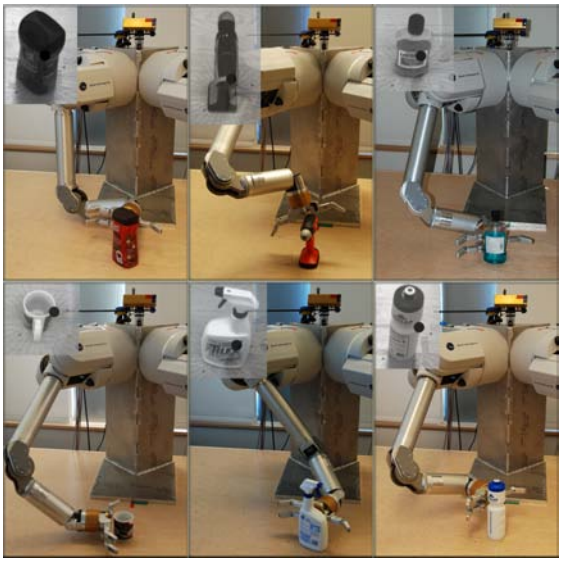

Fig. 6. Robot grasping trained objects with camera view in each corner.

first experiment (i.e. 6 objects, each with 10 trials) with novel objects that are fairly similar to those we trained on, some example of which are shown in Figure 7. Under those conditions, the accuracy drops to 76.66%, with the highest accuracy of 100% achieved by the drill and lowest accuracy of 60% for the shampoo bottle and the DVD case. In an

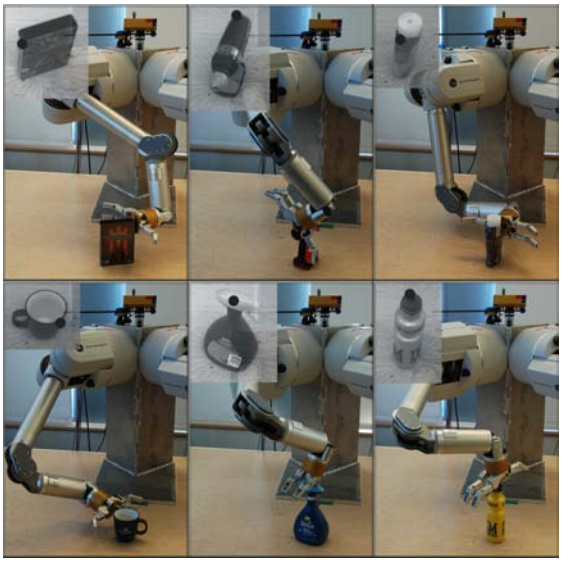

Fig. 7. Robot grasping novel objects with camera view in each corner.

attempt to further test our algorithm, we have also performed some experiments with novel objects that are completely different than those we trained on, as seen in Figure 8. While we only ran this experiment on 6 objects and 4 trials, we were impressed with the overall results of 83.33%. Last but not

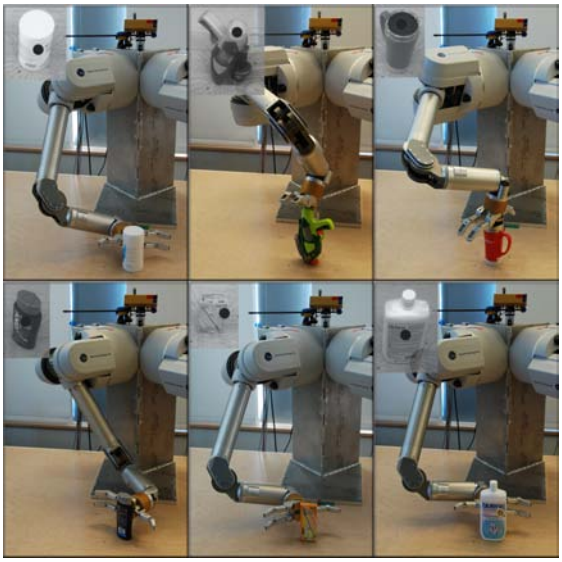

Fig. 8. Robot grasping novel objects with camera view in each corner.

least, even though all training was done with the right arm, we ran some experiments grasping with the left arm, to test the manipulator-independence property of our algorithm. We only ran the algorithm on 3 objects with 2 trials, as a proof of concept rather than a full experimental setup, and show some sample results in Figure 9. We conclude the experimental section of the paper with Table III, which shows the speed of the algorithm.

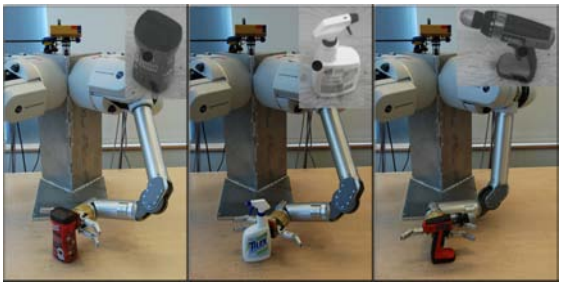

Fig. 9. Left arm grasping objects with camera view in each corner.

Algorithmic Part	Time (ms)
Outlier Removal	487.5
Object Classification	103.1
Nearest Neighbor Search	6.5
Wrist Calculation	4.4
Inverse Kinematics	20.0
Total Time	**621.5**

TABLE III

ALGORITHM SPEED, DIVIDED BY PARTS.

V. CONCLUSIONS

We have presented a novel algorithm aimed at computing wrist orientations for feature-based grasping algorithms. In addition to speed, one of the most desirable properties of the algorithm is that it is manipulator-independent, solely requiring forward and inverse kinematic solvers. Training data acquired with one manipulator can even be transferred to another provided that the forward kinematics of the original manipulator is known. Finally, the algorithm is capable of grasping unseen object parts, usually due to self-occlusions, and works with a single image taken from a stereo camera. The most important extension would be to incorporate a grasp quality metric as part of the algorithm to make sure that stable grasps are generated when the fingers close.

REFERENCES

[1] B. Balaguer and S. Carpin. Efficient grasping of novel objects through dimensionality reduction. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1279–1285, 2010.

[2] H. Bay, T. Tuytelaars, and L. Gool. Surf: Speeded up robust features. In *European Conference on Computer Vision*, pages 404–417, 2006.

[3] M. Ciocarlie and P. Allen. Hand posture subspaces for dexterous robotic grasping. *The International Journal of Robotics Research*, 28(7):851–867, 2009.

[4] O. Froemer, R. Detry, J. Piater, and J. Peters. Active learning using mean shift optimization for robot grasping. In *Int. Conference on Intelligent Robots and Systems*, pages 2610–2615, 2009.

[5] A. Karatzoglou, D. Meyer, and K. Hornik. Support vector machines in r. *Journal of Statistical Software*, 15(9):1–28, 2006.

[6] J. Laurikkala, M. Juhola, and E. Kentala. Informal identification of outliers in medical data. In *Fifth International Workshop on Intelligent Data Analysis in Medicine and Pharmacology IDAMAP*, 2000.

[7] D. Lowe. Object recognition from local scale-invariant features. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[8] A. Remazeilles, C. Dune, E. Marchand, and C. Leroux. Vision-based grasping of unknown objects to improve disabled people autonomy. In *Robotics: Science and Systems Manipulation Workshop: Intelligence in Human Environments*, 2008.

[9] A. Saxena, J. Driemeyer, and A.Y. Ng. Robotic grasping of novel objects using vision. *International Journal of Robotics Research*, 27(2):157–174, 2008.