Generating Multi-Fingered Robotic Grasps via Deep Learning

Jacob Varley, Jonathan Weisz, Jared Weiss, Peter Allen

Abstract— **This paper presents a deep learning architecture for detecting the palm and fingertip positions of stable grasps directly from partial object views. The architecture is trained using RGBD image patches of fingertip and palm positions from grasps computed on complete object models using a grasping simulator. At runtime, the architecture is able to estimate grasp quality metrics such as force closure without the need to explicitly calculate the given metric. This ability is useful as the exact calculation of these quality functions is impossible from an incomplete view of a novel object without any tactile feedback. This architecture for grasp quality prediction provides a framework for generalizing grasp experience from known to novel objects.**

I. Introduction

Despite the existence of a vast amount of literature on robotic grasping, grasp planning for objects from partial views remains a very challenging task. This paper presents a system relying on deep learning methods in order to generate grasps for objects that may or may not have been seen before and whose current full geometry is unknown. The proposed system uses a Convolutional Neural Network to quickly calculate grasp affordances such as force closure [3] directly from RGBD images. These methods make our system efficient, allow the use of a large context window for determining grasp quality, and enable the system to effectively generalize precomputed grasps of known models to novel 3D objects.

The proposed approach involves training a deep learning network on thousands of quality grasps generated using known object models from BigBIRD[16], the Berkeley Instance Recognition Dataset. Training grasps are generated in an environment where the full object geometry is known using GraspIt![12], a grasp simulator. For each grasp, the locations of the fingertip positions and palm are recorded along with the grasp quality score representing the stability of the given grasp for this known environment. Then, RGBD images of the grasp can be rendered, and RGBD image patches centered on the fingertip and palm locations can be extracted. These patches paired with the recorded grasp quality score are then used to train a deep network to quickly recognize stable fingertip and palm locations for different canonical grasp types directly from RGBD images.

At runtime, both an RGBD image and a pointcloud for a scene are generated from a single point of view. The pointcloud is segmented and meshes are created for the segmented objects. Since the pointcloud was generated from a single view, these meshes are incomplete, and represent only the faces of the object that were visible to the camera. At the same time that the pointcloud is being processed, the RGBD image is passed through the deep learning network in order to generate heatmaps or dense per pixel labels revealing how well every location in the image would act as a fingertip or palm location for a number of different canonical hand configurations. The incomplete meshes are then imported into GraspIt!, and Simulated Annealing is run to find stable grasps that both align well with the visible portions of the mesh model to be grasped, and whose fingertips and palm positions project into low energy locations in the heatmaps produced by the deep network.

II. Related Work

There is a significant body of work related to grasping with incomplete sensor data as well as different metrics for generating grasps for known objects. For a more detailed summary, we refer the reader to [1]. The following is a brief overview of some of the work most relevant to the presented system.

1) Data Driven Grasping: In the approach taken by [5][4], a grasping simulator plans grasps that maximize a given quality metric offline to generate grasp for known 3D objects. Then, when attempting to grasp an object in a scene, the scene is segmented, and the segmented object's closest model is found in the database and the precomputed grasps from the closest matching model are used on the new object. This approach works reliably when the object to be grasped can easily be segmented out of the scene and is not markedly different than models in the database. A problem with this approach is that it is difficult to fine tune a retrieved grasp in this situation especially when the full geometry of the object is not known.

2) Deep Learning for Parallel Jaw Grippers: In another approach [10]. The grasping system learns grasps for a parallel jaw gripper from manually labeled RGBD images. The labels consist of rectangles with two of the edges representing where the gripper should be placed. This approach, while able to learn arbitrary grasp types based on whatever training data is given, has three major drawbacks. First each image has to be hand labeled, which can be burdensome. Second, the grasp rectangles that are evaluated to determine grasp quality only contain the region of the image that will lie inside of the gripper. There is no wider context in which to evaluate the grasp. Third, and most importantly, the grasping rectangles concept is not easily extensible to more complicated robotic hands and fingers.

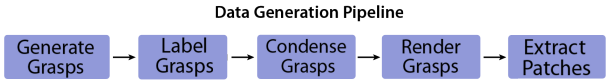


Fig. 1: Overview of the grasp training data generation stages.

3) Fingertip Space and Precision Grasping: The concept of a Fingertip Space and an efficient algorithm for producing stable grasps within this space has been proposed by a number of researchers including [7][13][11]. For example, in [7], their approach provides a useful grasping strategy when there is a proficient amount of information about the geometry of the object to be grasped. This stands in contrast to this work which provides a framework enabling the transfer of knowledge from grasps generated with object geometries known to situations where only a partial view of the object is available.

III. Method

The description of the presented system is decomposed into three main sections, grasp training data generation, the deep learning model, and grasping from partial views. The first section describes the generation of grasp data that will be used to train the deep model to recognize good locations for the placement of the palm and fingertips of the end effector. The next section overviews the deep model that was used, how it was trained, and an interpretation of the output. The final section describes how the output of the deep model can be used to find stable grasps in coordination with a segmented mesh of the scene generated from a single viewpoint.

A. Grasp Training Data Generation

The generation of training data is a multi-stage process. Grasps are generated in GraspIt! using complete object models. The locations of the palm and fingertips for grasps in prototypical hand configurations are saved along with the energy of the given grasp. RGBD images of these palm and fingertip locations are then captured in simulation, and patches around these locations are saved. An overview of the data generation process is shown in Fig. 1, and the individual steps are explained in more detail below.

1) Grasp Generation: Training of the grasp generation system required the generation of quality grasps on known objects. In order to generate these grasps, models from BigBIRD, the Berkeley Instance Recognition Dataset, were uploaded into GraspIt!. The dataset contains mesh models for approximately 125 everyday objects. Several of the models had large holes due to transparent regions and were not used as valid training grasps could not be generated for them. In GraspIt! each model was uploaded along with a Barrett Hand model as shown in Fig. 2. From each object model, a set of points was chosen uniformly over the mesh surface and GraspIt!'s Simulated Annealing planner was run for 30 seconds at a time seeded over the chosen points on the object mesh. For the grasp data generation, GraspIt!'s Contact and Potential Energy function was used to score the generated grasps. The Potential Energy function ensures that the generated grasps have force closure. Only the grasps with low-energy quality scores below a threshold were kept. This generated 272,380 grasps in all, and of the generated grasps, 41,708 had a quality score below our threshold and were used for training.

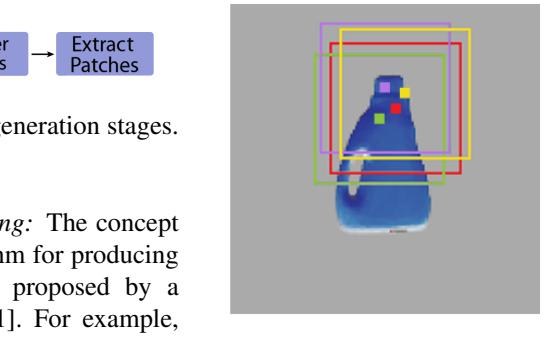


Fig. 2: Barrett Hand with virtual contacts shown in red.

Fig. 3: Image Patches extracted from a grasp on an All brand detergent bottle. The center of each image patch corresponds to where the palm and fingertips from the grasp project into the image plane.

2) Grasp Type Labeling: The generated grasps were then categorized into different grasp types based on the 7 joint values and the roll of the wrist. Each joint value of the hand and the wrist roll were given $nBins = 5$ bins. This provided $nBins^{nJoints} = 5^8 = 390,625$ different grasp types. The grasp type is simply a integer between 0 and 390,624 that is determined by the coarse configuration of the hand. The exact formulation is given by:

$$joints = \{j0, j1, j2, j3, j4, j5, j6, wrist\_roll\}$$

$$grasp\_type = \sum_{i=0}^{nJoints-1} bin(joint[i]) * nBins^i$$

3) Grasp Type Reduction: The vast majority of the above grasp types had no grasp examples. For instance, the grasp type corresponding to the hand being completely open never results in a stable grasp. The frequency of each grasp type within the training data was computed, and only the $n = 8$ most common grasps types were kept as the set of canonical grasps. The set of 41,708 quality grasps from above contained 3,770 grasps of the 8 most common grasp types. Only the top 8 were kept as the number of examples per grasp type was no longer sufficient past the 8th most common grasp type.

4) Grasp Rendering: For each grasp, the target object was placed into Gazebo[9]. The position of the palm with relation to the object was calculated and a simulated Kinect sensor

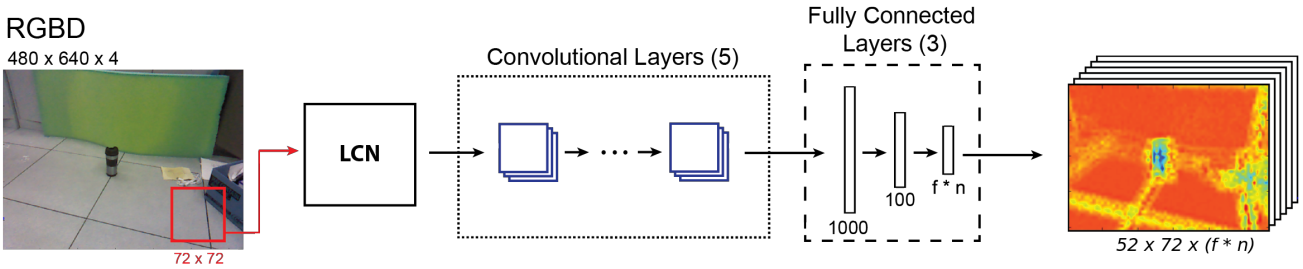


Fig. 4: Deep Model consisting of a local contrast normalization (LCN) stage, a 5 layered convolutional stage, and 3 fully connected layers.

Layer	1	2	3	4	5	6	7	8
Stage	conv + max pool	conv + max pool	conv	conv	conv	full	full	full
# channels	96	256	256	256	256	1000	100	$f * n$
Filter Size	11x11	3x3	3x3	3x3	3x3	-	-	-
Filter Stride	2x2	-	-	-	-	-	-	-
Pool Size	2x2	2x2	-	-	-	-	-	-
Pool Stride	2x2	2x2	-	-	-	-	-	-
Input Size	480x640	118x158	58x78	56x76	54x74	52x72	52x72	52x72

TABLE I: Architecture for the model. The final number of channels in the final stage is the number of grasp types (n) multiplied by the number of interest points on the hand (f) i.e the palm and fingertips.

was positioned 2 meters backed off from the object along the approach direction of the hand and an RGBD image was captured.

5) Training Patch Extraction: From each RGBD image, patches around the location of the palm and fingertips are extracted. These training patches allow the deep learning model to learn what good fingertip locations look like for specific grasp types, for a known object, from a specific viewing angle. Fig. 3 demonstrates how 4 patches are generated from a single grasp of a known model.

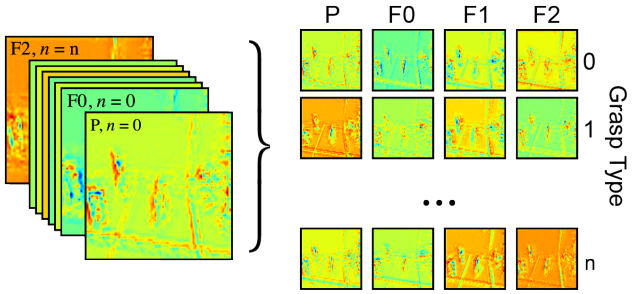


Fig. 6: The output of the Deep Network can be thought of as a set of heatmaps representing how well each location in the image acts as a fingertip or palm location for each grasp type.

B. Deep Learning Model

An overview of the Deep Network Architecture is shown in Fig. 4

1) Training Data Labels: The deep learning model is trained on the RGBD patches generated using the process described above. For each patch, a label vector the length of the number of canonical grasp types (n) multiplied by the number of palm and finger tip locations (f) is created.

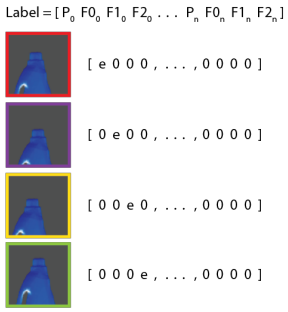


Fig. 5: Patches and label vectors for a single grasp with energy e.

$$len(\text{label\_vec}) = n * f = 8 * 4 = 32$$

The vector contains all zeros except at the entry which represents the corresponding grasp and finger type of the patch. This entry is filled with the grasp quality score for the grasp that this patch was extracted from. Fig. 5 shows how the training labels for a set of patches from the same grasp are composed. At runtime, the model takes a full RGBD image as input and returns a set of heatmaps, one for each of the fingertips and palm for each grasp type. These heatmaps represent how well a given pixel location works as a fingertip or palm location for the given grasp type.

2) Model Description: Our model was implemented using Pylearn2 [6] and consists of a local contrast normalization stage(LCN) [8] followed by five Convolutional Rectified Linear layers and then by three fully connected Linear layers. The model was trained using an Nvidia GTX 780 GPU. The parameters of the model are shown in Table I.

3) Heatmaps: While the model was trained on image patches of 72x72, when running it takes as input an entire RGBD image (480x640) and outputs a label vector for every pixel in the image with the exception of the border pixels where a full 72x72 patch is not available. The output

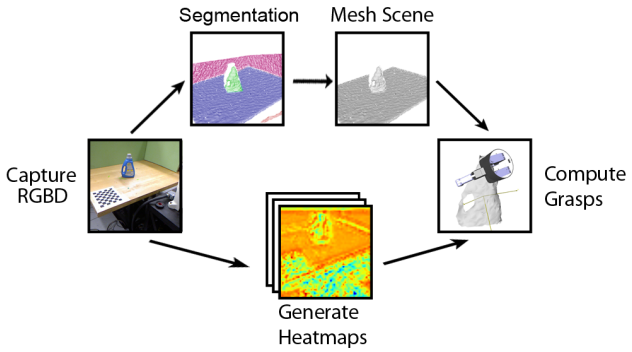


Fig. 7: Overview of the Grasp Generation System. After an image is captured, the scene is segmented and meshed while the heatmaps are generated in parallel. Both the meshed scene and the heatmaps are then used within GraspIt! to plan grasps.

matrix is downsampled due to the Filter Strides and Max Pooling in the first two Convolutional Layers. In order to account for this, we rescale the output so that it is of the same dimensionality as the input image with a border corresponding to the patch size removed. This output matrix can be thought of as a set of potentials showing how well every pixel would work as a fingertip location for every fingertip type for each of the canonical grasp types. Fig. 6 demonstrates how the output of the deep learning model can be interpreted.

4) Training: The model is trained using stochastic gradient descent (SGD) with minibatches of size 20, and was shown 30,200 examples in 151 epochs. The learning rate was .1, and an L1 term was used to induce sparsity in the weights. In addition, small amounts of Gaussian noise was randomly added to the training patches.

C. Grasping from Partial Views

In order to realize a grasp of an object using the model above, several additional steps must be taken. An RGBD image must be captured, the scene must be segmented, and meshes must be generated for the segmented objects in the scene. Then Simulated Annealing can be run within GraspIt! on the mesh of the scene in order to find an optimal grasp that is well aligned with the visible portions of the object and is stable according to the deep network. An overview of this process is shown in Fig. 7.

1) Scene Segmentation: The system segments the point cloud associated with the RGBD image following the Euclidean Cluster Extraction tutorial from PCL[15]. The largest plane in the pointcloud is found using RANSAC and then removed. Euclidean clustering is then run on the remaining points, and any objects that were previously resting on the largest plane in the scene now form separate clusters. This method, while naive, works well for any scene with models that are not touching and resting on a dominant plane such as the floor or a table.

2) Mesh Generation: The segmented clusters from the previous step are then meshed and the set of meshed partial objects is returned. It is important to ensure that the normals for the mesh are correctly oriented as they are used later to determine how well the mesh is aligned with the hand.

3) Heatmap Generation: In parallel with the generation of the meshes as described above, the RGBD image captured from the Kinect is sent through the previously described deep learning network. For a given input RGBD image this creates a set of $f * n$ heatmaps, one for each of the f finger and the palm locations for each of the n canonical grasp types the system was trained on.

4) Grasp Planning: Within GraspIt!, the meshes are loaded into a planning scene, and a partial mesh is selected for grasp planning. A partial mesh must be selected as the planning scene may be crowded and an automated selection of which object to manipulate is outside the scope of this work. Once a target object is selected, GraspIt!'s Simulated Annealing planner is started [2]. At each step, the planner calculates the energy associated with the current hand configuration as a weighted sum of a Contact Energy term and the Heatmap Energy term, both of which are described in the following section. The Contact Energy term encourages grasps to align well with the visible portions of the meshed object, while the Heatmap energy term encourages placement of the palm and fingertips into positions for a stable grasp as predicted by the deep learning network. The grasp planner stores a list of the top 20 grasps ranked by energy that are calculated over a pre-specified number of iterations of the planner. At this point, either the lowest energy grasp that is reachable can be executed automatically or a user can select a preferred grasp from the list generated by the planner.

D. Grasp Energy Calculation

1) Contact Energy: The Contact Energy function makes use of Virtual contacts, or user-specified points of interest on the hand as shown in Fig. 8. In our case, we spread a total of 16 virtual contacts uniformly over the Barrett Hand. The Contact Energy metric penalizes grasps whose virtual contacts are either distant from the object or whose normals do not align well with the object. The algorithm is fully described in Algorithm 1. The two components come from the distance between the closest point on the object to be grasped, and the dot product of the normalized vector extending from the virtual contact location to the closest point on the model with the normal of the virtual contact and the hand. γ is used to scale how large of a contribution

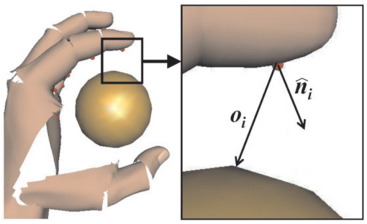


Fig. 8: Minimizing Contact Energy is equivalent to minimizing both $o_i \cdot n_i$ and the length of o_i in order to keep the virtual contact close to and well aligned with the object.

the alignment term makes relative to the distance term. Two versions of the isVisible function were explored. For forward facing grasps using only the 4 virtual contacts on the palm was sufficient, while for all cases, the orientation of the vector extending from the virtual contact to the nearest point on the mesh was compared to the vector extending from the camera location to the same point. The sign of the dot product was used to ensure that contact was being made on a visible face of the mesh.

Algorithm 1 Contact Energy Calculation

```
1:  procedure CONTACTENERGY()
2:      numVisible = 0
3:      energy = 0
4:      energy = 0
5:      for vc in virtualContacts do
6:          ip = getIntersectionPoint(vc, body)
7:          if isVisible(ip) then
8:              numVisible ++
9:              energy += dist(vc, ip)
10:             energy += γ* (1 - (vc.norm · ip.norm))
11:     return energy / numVisible
```

Algorithm 2 Heatmap Energy Calculation

```
1:  procedure HEATMAPENERGY()
2:      energy = 0
3:      graspType, distance = getGraspType()
4:      if distance > α then
5:          return MAX_ENERGY
6:      for vc in virtualContacts do
7:          fingerType = getFingerType(vc)
8:          if fingerType in [Palm, F0, F1, F2] then
9:              u,v = contactToImagePlane(vc.x, vc.y, vc.z)
10:             index = getIndex(graspType, fingerType)
11:             energy += heatmaps[index][u][v]
12:             energy += μ*distance
13:     return energy
```

2) Heatmap Energy: The second component of the energy function comes from the heatmaps generated by the deep learning network. The distance from the current hand configuration to the nearest canonical grasp type is computed. If there are no nearby grasp types, then the Heatmap Energy is returned as a large value. This implies that the current hand configuration is not well associated with any of the learned grasp types. If there is a nearby grasp type, then the corresponding heatmaps are used to determine the Heatmap Energy. The current locations of the palm and finger tips of the hand are projected back into the plane of the image captured by the Kinect, and the heatmap values for the individual fingers and the palm for the current grasp type are added to the energy score for the current hand configuration. In addition, the distance to the nearest grasp type is multiplied by μ and added to the energy. This penalizes grasps for moving away from the canonical hand configurations. Higher μ values keep the hand in configurations close to the canonical grasp hand configuration. A full description of this algorithm is shown in Algorithm 2. Overall, the Heatmap Energy function favors hand configurations where the palm and fingertips project into low energy areas of the heatmaps, areas that the deep model has associated with stable grasps.

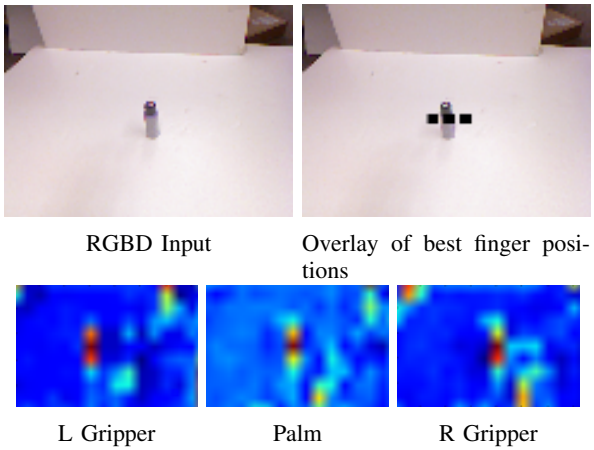


Fig. 9: Example for a scene from the Grasping Rectangles Dataset[10]. For this dataset, positive grasp examples where scored with a +1 and negative grasp examples with a score of -1, so for these heatmaps red corresponds to good positions to place the gripper or palm. The argmax from each heatmap is displayed on the overlay of the original image.

3) Combined Energy: The Contact Energy and Heatmap Energy terms are combined using a mixing parameter ρ. This parameter determines how heavily to weight the above terms relative to each other as shown in the following equation:

$$\rho * heatmapEnergy + (1 - \rho) * contactEnergy$$

Higher values of ρ lead to grasps where the fingertips are able to move into lower energy positions at the expense of increasing the contact energy. It is a balance for how much a grasp should be rewarded for aligning nicely with the visible portions of the object against how the current hand configuration compares to the learned idea of what a stable grasp should look like.

IV. Experimental Results

A. Grasping Rectangle Dataset

In order to demonstrate the ability of the system to learn arbitrary grasp quality functions, we trained our system on the Grasping Rectangle Dataset from [10]. In the Grasping Rectange Dataset, two of the edges of a rectangle represent where the gripper should be placed. Image patches were extracted from the center of each edge of the rectangle to represent each finger, and from the center to represent the palm. The grasps were discretized based on the rotation of the rectangle, so that grasp types related to the rotation of the wrist. The system was able to reproduce results similar to those found by[10]. An example scene along with a set

Real World Grasps								
Trial #	Scene	Target Object	Success	Scene Segmentation (s)	Mesh Generation (s)	Heatmap Generation (s)	Simulated Annealing (s)	Total Planning Time (s)
1	All bottle	All bottle	yes	0.168	3.014	3.122	10.982	14.164
2	water bottle	water bottle	yes	0.159	2.977	3.141	10.471	13.612
3	drill	drill	yes	0.157	2.991	3.119	12.745	15.893
4	shampoo	shampoo	yes	0.339	3.277	3.122	12.253	15.869
5	tool case	tool case	no	0.175	3.026	3.242	14.328	17.570
6	crowded	shampoo	yes	0.181	3.307	3.127	14.328	17.816
7	crowded	drill	no	0.181	3.167	3.118	15.940	19.288
8	crowded	All bottle	yes	0.188	3.338	3.118	15.507	19.033
		Success:	6/8				Average Total Time:	16.655

TABLE II: Results of grasp attempts for varying scenes and objects. $\rho = .1$, $\mu = 10$, and $\gamma = 500$

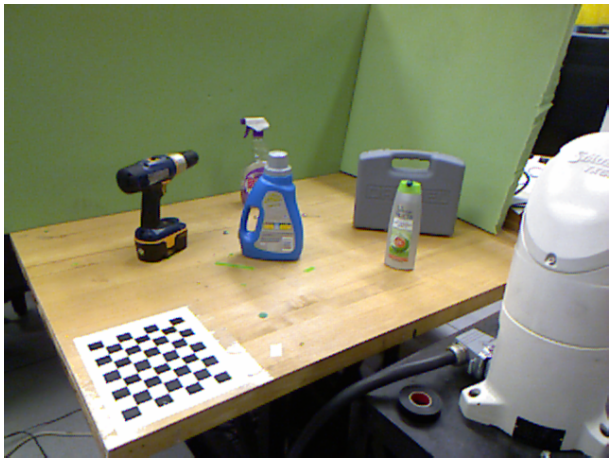
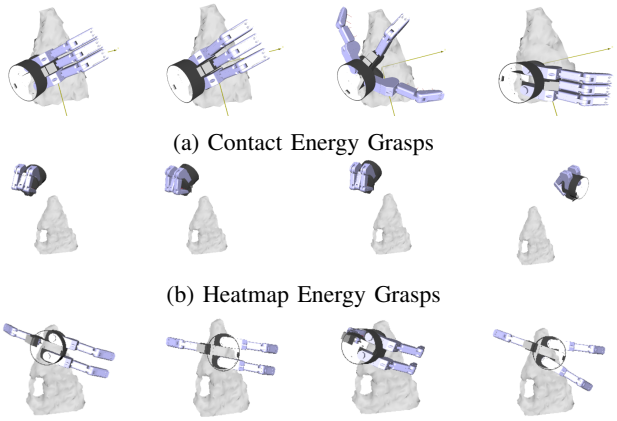


Fig. 10: Crowded scene from trial 8. This scene contains all other objects used in the experiments.



(a) Contact Energy Grasps

(b) Heatmap Energy Grasps

(c) Combined Energy Grasps

Fig. 11: The Contact Energy function aligns the palm well with the object, but fails to move the fingers into meaningful positions. The Heatmap Energy moves the hand to low energy positions that may not be relevant for grasping the current object. The Combined Energy uses the Contact Energy to stay close to the object and the Heatmap Energy to refine the fingertip positions.

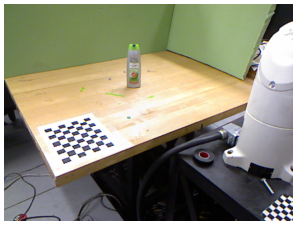
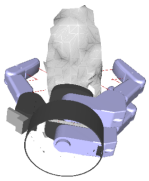
of heatmaps produced by the deep learning network and the best fingertip positions found by our system is shown in Fig. 9.

B. BigBird Generated Dataset

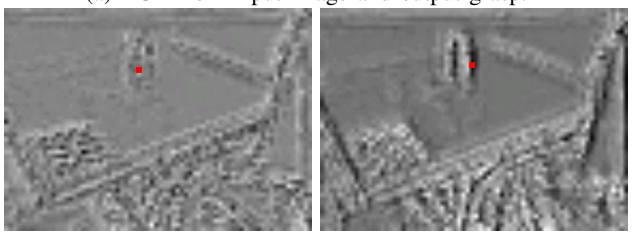
Using the deep learning model trained on the BigBird dataset, grasps were generated for several different real world scenes, and executed using a Barrett Hand attached to a StaubliTX60 robotic arm. None of the objects in the scenes were present in the training data for the deep learning model. The models used for this experiment are pictured in Fig. 10. For each trial, a single RGBD image and pointcloud of the scene was captured. The RGBD image was sent through the deep learning network while the pointcloud was used to segment the scene and mesh the segmented objects. Then the Simulated Annealing Planner was run for 10,000 steps using the Combined Energy function and the lowest energy reachable grasp generated within that time was executed by the arm. Success was determined by whether or not the object was contained in the hand and lifted off of the table. The total time is generated with the following formula as the heatmap generation and scene segmentation are run in parallel:

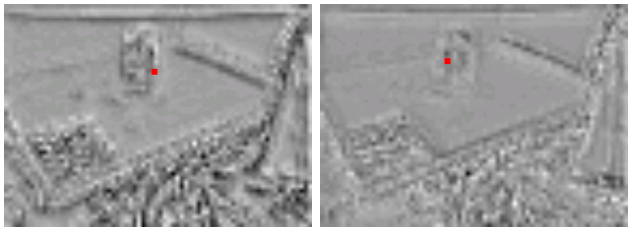
$$t_{total} = max(t_{segment} + t_{mesh}, t_{heatmap}) + t_{sim.ann.}$$

The results for the experiments are shown in Table II. While $\rho = .1$ sets more weight to the Contact Energy, the unweighted Contact Energy is smaller than the unweighted Heatmap Energy, so this value balances the two. The experimental results demonstrate that our system is able to generalize grasps generated on the BigBird models to these new objects. Several example grasps are shown in Figures 12,13,14,15. These experiments reveal not only that our system can generalize well, but that it is also effective in crowded scenes and with semi transparent objects. As long as some portion of the object is visible and able to be meshed, then this system is able to generate grasps around that portion of the object. While the current system failed to generate a stable grasps for the tool case, we attribute this partially to the fact that very few of the objects in the training set have handles, and none of them had horizontal handles like the tool case. For the failure case involving the drill, when the partial mesh of the drill was created, it was missing portions of the handle. While the hand was closing, it collided with a

(a) RGB from input image and output grasp.

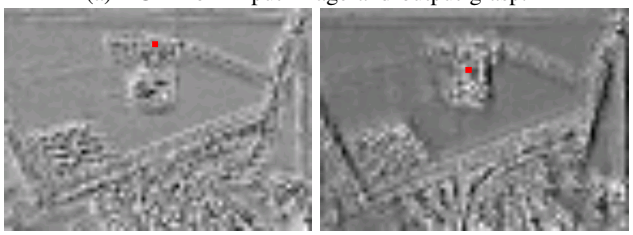


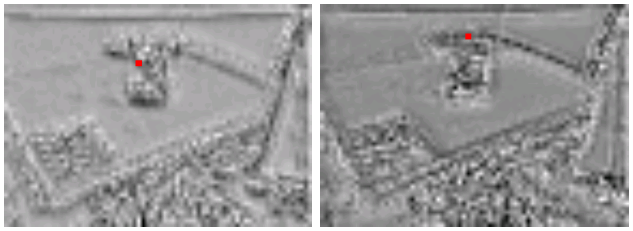


(b) Heatmaps with the virtual contacts projected into them. The red rectangles show where the given fingertip projects into its appropriate heatmap. The darker the regions of the heatmaps represent low energy regions corresponding to good fingertip positions.

Fig. 12: Example grasp generated for a shampoo bottle.


(a) RGB from input image and output grasp.





(b) Heatmaps with the virtual contacts projected into them. The red rectangles show where the given fingertip projects into its appropriate heatmap. The darker the regions of the heatmaps represent low energy regions corresponding to good fingertip positions.

Fig. 13: Example grasp generated for a drill.

portion of the drill that had not been meshed, this is difficult to avoid, as we can only plan trajectories and generate hand configurations that avoid collisions with portions of objects that are successfully meshed.

C. Comparison of Grasp Energy Components

Using a single scene, grasps were planned using Contact Energy, Heatmap Energy, and Combined Energy to demonstrate how the different components of the Combined Energy function work in isolation. The Contact Energy function aligns the palm to the object to be grasped, but does not place the fingers into any specific configuration. The Heatmap Energy alone places the fingers into positions that project into low energy locations within the heatmaps, but the hand will often find low energy regions of the heatmaps that are for other objects to be grasped, or may be either to far into the scene, or too close to the camera. Nothing is ensuring that the hand stay near the object to be grasped. The Combined Energy function uses the Contact Energy to ensure that the hand is near the target object, while the heatmap energy refines the fingertip and palm locations locally. Fig. 11 shows several low energy grasps computed with the individual energy functions as well as for the Combined Energy function.

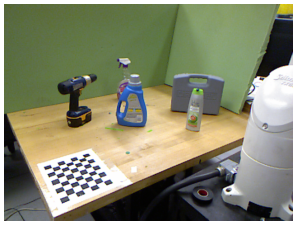
V. Conclusion

It is worth noting that this deep learning network is on the small end compared to networks built for the ImageNet challenge [14], which have 1000 output categories compared to our 32. This implies that our methodology could reasonably scale to 250 grasp types with 4 fingertip positions given the appropriate volume of training data.

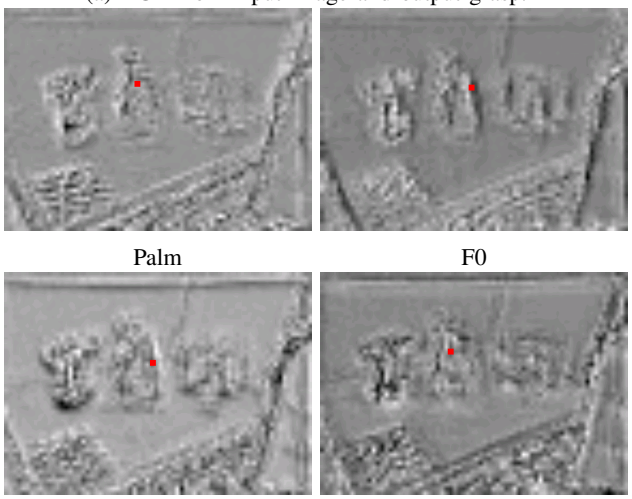
We have shown that traditional methods for calculating grasps on known 3D objects can be paired with a deep learning architecture in order to efficiently estimate arbitrary grasp affordances from RGBD images. The deep architecture can learn grasp qualities that may take arbitrarily long to calculate on the training models enabling the use of thorough and meaningful grasp quality scores to generate training grasps. The generated grasps can then be used to train a deep model to quickly recognize quality grasps without explicitly calculating the grasp quality or even requiring a 3D model of the object to be grasped.

VI. Acknowledgments

We would like to thank Y. Li for many thoughtful discussions. This work is supported by NSF Grant IIS-1208153.

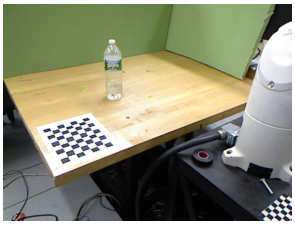


(a) RGB from input image and output grasp.

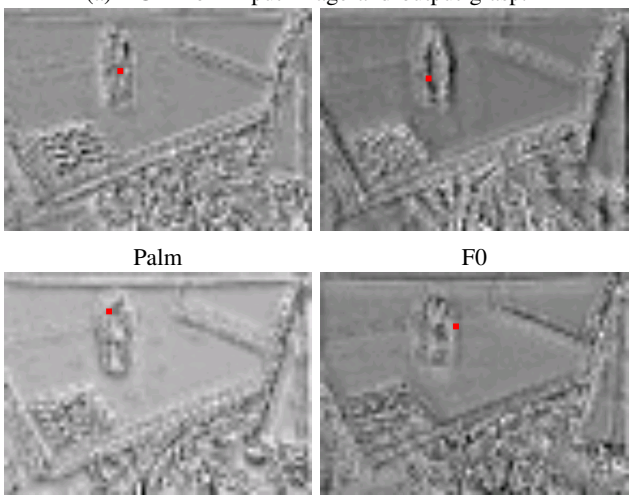


(b) Heatmaps with the virtual contacts projected into them. The red rectangles show where the given fingertip projects into its appropriate heatmap. The darker the regions of the heatmaps represent low energy regions corresponding to good fingertip positions.

Fig. 14: Example grasp generated for an All Detergent bottle.



(a) RGB from input image and output grasp.



(b) Heatmaps with the virtual contacts projected into them. The red rectangles show where the given fingertip projects into its appropriate heatmap. The darker the regions of the heatmaps represent low energy regions corresponding to good fingertip positions.

Fig. 15: Example grasp generated for a water bottle.

REFERENCES

[1] Jeannette Bohg, Antonio Morales, Tamim Asfour, and Danica Kragic. Data-driven grasp synthesisa survey. *Robotics, IEEE Transactions on*, 30(2):289–309, 2014.

[2] Matei T Ciocarlie and Peter K Allen. Hand posture subspaces for dexterous robotic grasping. *The International Journal of Robotics Research*, 28(7):851–867, 2009.

[3] Carlo Ferrari and John Canny. Planning optimal grasps. In *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, pages 2290–2295. IEEE, 1992.

[4] Corey Goldfeder and Peter K Allen. Data-driven grasping. *Autonomous Robots*, 31(1):1–20, 2011.

[5] Corey Goldfeder, Matei Ciocarlie, Jaime Peretzman, Hao Dang, and Peter K Allen. Data-driven grasping with partial sensor data. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 1278–1283. IEEE, 2009.

[6] Ian J. Goodfellow, David Warde-Farley, Pascal Lamblin, Vincent Dumoulin, Mehdi Mirza, Razvan Pascanu, James Bergstra, Frédéric Bastien, and Yoshua Bengio. Pylearn2: a machine learning research library. *arXiv preprint arXiv:1308.4214*, 2013.

[7] Kaiyu Hang, Johannes A Stork, and Danica Kragic. Hierarchical fingertip space for multi-fingered precision grasping. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 1641–1648. IEEE, 2014.

[8] Kevin Jarrett, Koray Kavukcuoglu, M Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2146–2153. IEEE, 2009.

[9] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2149–2154. IEEE.

[10] Ian Lenz, Honglak Lee, and Ashutosh Saxena. Deep learning for detecting robotic grasps. *IJRR*, 2014.

[11] Yun Lin and Yu Sun. Robot grasp planning based on demonstrated grasp strategies. *The International Journal of Robotics Research*, 34(1):26–42, 2015.

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

[13] Carlos Rosales, Llus Ros, Josep M. Porta, and Ral Surez. Synthesizing grasp configurations with specified contact regions. *The International Journal of Robotics Research*, 30(4):431–443, 2011.

[14] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge, 2014.

[15] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[16] Arjun Singh, James Sha, Karthik S Narayan, Tudor Achim, and Pieter Abbeel. Bigbird: A large-scale 3d database of object instances. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 509–516. IEEE, 2014.