Shape Completion Enabled Robotic Grasping

Jacob Varley, Chad DeChant, Adam Richardson,
Avinash Nair, Joaquín Ruales, and Peter Allen

***Abstract*— This work provides an architecture to enable robotic grasp planning via shape completion. Shape completion is accomplished through the use of a 3D convolutional neural network (CNN). The network is trained on an open source dataset of over 440,000 3D exemplars from varying viewpoints. At runtime, a 2.5D pointcloud captured from a single point of view is fed into the CNN, which fills in the occluded regions of the scene, allowing grasps to be planned and executed on the completed object. Runtime shape completion is very rapid because most of the computational costs of shape completion are borne during offline training. We explore how the quality of completions vary based on several factors. These include whether or not the object being completed existed in the training data and how many object models were used to train the network. The ability of the network to generalize to novel objects allows the system to complete previously unseen objects at runtime, representing a potentially significant improvement over purely database-driven completion approaches. Finally, experimentation is done both in simulation and on actual robotic hardware to explore the relationship between completion quality and the utility of the completed mesh model for grasping.**

I. INTRODUCTION

Grasp planning based on raw sensory data is difficult due to occlusion and incomplete information regarding scene geometry. This work utilizes 3D convolutional neural networks (CNNs)[1] to enable stable robotic grasp planning via shape completion. The 3D CNN is trained to do shape completion from a single pointcloud of a target object, essentially filling in the occluded portions of objects. This ability to infer occluded geometries can be applied to a multitude of robotic tasks. It can assist with path planning for both arm motion and robot navigation where an accurate understanding of whether occluded scene regions are occupied or not results in better trajectories. It also allows a traditional grasp planner to generate stable grasps via the completed shape.

The proposed framework consists of two stages: a training stage and a runtime stage. During the training stage, the CNN is shown occupancy grids created from thousands of synthetically rendered depth images of different mesh models. Each of these occupancy grids is captured from a single point of view, and occluded portions of the volume are marked as empty. For each training example, the ground truth occupancy grid (the occupancy grid for the entire 3D volume) is also generated for the given mesh. From these pairs of occupancy grids the CNN learns to quickly complete mesh models at runtime using only information from a single point of view. Several example completions are shown in Fig. 1. This setup is beneficial for robotics applications as the majority of the computation time takes place during offline training, so that at runtime an object's pointcloud can be run through the CNN in under a tenth of a second on average and then quickly meshed.

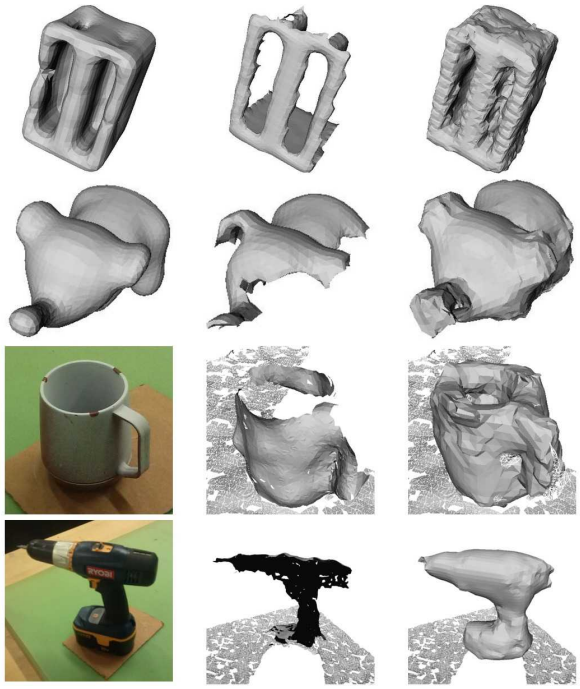


Fig. 1: Ground Truth, Partials, and Completions. The toaster and toy are completions of synthetic depth images of Grasp Dataset holdout models. The mug and drill show completions of Kinect-captured depth images of physical objects.

During the runtime stage, a pointcloud is captured using a depth sensor. A segmentation algorithm is run, and regions of the pointcloud corresponding to graspable objects are extracted from the scene. Occupancy grids of these regions are created, where all occluded regions are incorrectly labeled as empty. These maps are passed separately through the trained CNN. The outputs from the CNN are occupancy grids, where the CNN has labeled all the occluded regions of the input as either occupied or empty for each object. These new occupancy grids are either run through a fast marching cubes algorithm[2], or further post-processed if they are to be grasped. Whether the object is completed or completed and post-processed results in either 1) fast completions

Authors are with Columbia University Robotics Group, Columbia University, New York, NY 10027, USA jvarley@cs.columbia.edu, dechant@cs.columbia.edu, ajr2190@columbia.edu, asn2129@columbia.edu, jar2262@columbia.edu, allen@cs.columbia.edu

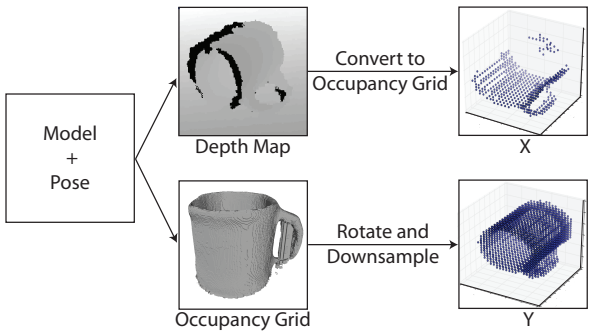


Fig. 2: Training Data Generation. For each training example, X is the input into the CNN, and Y is the expected output. In X, the occupancy grid only marks visible portions of the model, while Y has all voxels occupied by the model marked.

suitable for path planning and scene understanding or 2) refined meshes suitable for grasp planning. This framework is extensible to crowded scenes with multiple objects as each object is completed individually. It is also applicable to different domains because it can learn to reproduce objects from whatever dataset it is trained on, and further shows the ability to generalize to unseen views of objects or even entirely novel objects. This applicability to multiple domains is complemented by the thousands of 3D models available from datasets such as ModelNet[3] and the rapidly increasing power of GPU processors.

The contributions of this work include: 1) a novel CNN architecture for shape completion; 2) Complementary completion methods, one resulting in meshes to quickly fill the planning scene, and the second creating smoothed meshes that are combined with the observed pointcloud for grasp planning; 3) A large open-source dataset of over 440,000 40^3 voxel grid pairs used for training. This dataset and the related code are freely available at http://shapecompletiongrasping.cs.columbia.edu. In addition, the website makes it easy to browse and explore the thousands of completions and grasps related to this work; 4) Results from both simulated and live experiments comparing our method to other approaches and demonstrating its performance in grasping tasks.

II. Related Work

There is a significant body of prior work relating to the various intersections of shape completion, deep learning, and grasp planning. But this is the first work that the authors are aware of that integrates all three.

This framework makes heavy use of both the YCB[4] and Grasp Database[5] mesh model datasets. We augmented the YCB set with models from the Grasp Database which contains 590 mesh models.

Prior work for shape completion includes [6][3]'s use of a deep belief network and Gibbs sampling for 3D shape reconstruction. In addition, work by [7] uses an exemplar based approach for the same task. [8][9] have explored shape completion for robotic grasping, but use symmetry and extrusion based completion approaches which work well for objects well represented by geometric primitives.

Other related approaches to grasping include [10] which uses Gaussian processes to plan under uncertainty rather than explicitly completing the target object. In addition [11][12] where deep learning has been used to find optimal manipulator configurations based on prior training grasps. These works have attempted to plan using only the visible portions of the scene paired with affordances produced as output by a deep network. These approaches differ from our work where the deep network is used to reconstruct a scene and then a planner is run on the reconstructed scene. It is plausible that these approaches could be used concurrently.

III. Training

A. Data Generation

In order to train a network to reconstruct a diverse range of objects, meshes were collected from the YCB and Grasp Database. The models were run through binvox[13] in order to generate 256x256x256 occupancy grids. In these occupancy grids, both the surface and interior of the meshes are marked as occupied. In addition, all the meshes were placed in Gazebo[14], and 726 depth images were generated for each object subject to different rotations uniformly sampled (in roll-pitch-yaw space, 11*6*11) around the mesh. The depth images are used to create occupancy grids for the portions of the mesh visible to the simulated camera, and then all the occupancy grids generated by binvox are transformed to correctly overlay the depth image occupancy grids. Both sets of occupancy grids are then down-sampled to 40x40x40 to create a large number of training examples. The input set (X) contains occupancy grids that are filled only with the regions of the object visible to the camera, and the output set (Y) contains the ground truth occupancy grids for the space occupied by the entire model. An illustration of this process is shown in Fig. 2.

B. Model Architecture and Training

The architecture of the CNN is shown in Fig. 3. The model was implemented using Keras[15], a Theano[16][17] based deep learning library. Each layer used rectified linear units as nonlinearities except the final fully connected (output) layer which used a sigmoid activation to restrict the output to the range $[0, 1]$. We used the cross-entropy error $E(y, y')$ as the cost function with target y and output y':

$$E(y, y') = -\left(y \log(y') + (1 - y) \log(1 - y')\right)$$

This cost function encourages each output to be close to either 0 for unoccupied target voxels or 1 for occupied. The optimization algorithm Adam[18], which computes adaptive learning rates for each network parameter, was used with default hyperparameters (β_1=0.9, β_2=0.999, ϵ=10^{-8}) except for the learning rate, which was set to 0.0001. Weights were initialized following the recommendations of [19] for rectified linear units and [20] for the logistic activation layer. The model was trained with a batch size of 32. Each of the

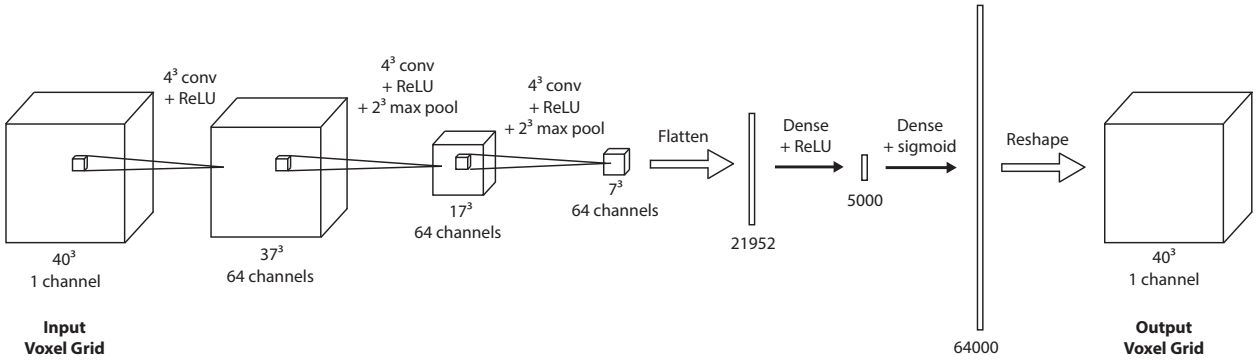
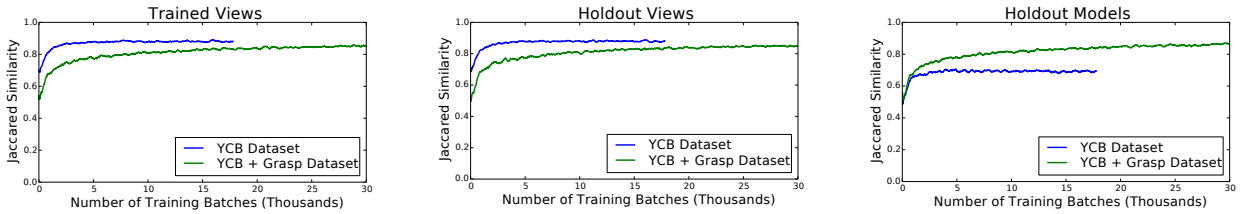


Fig. 3: CNN Architecture. The CNN has three convolutional layers and two dense layers. The final layer has 64000 nodes, and is reshaped to form the resulting $40 \times 40 \times 40$ occupancy grid. The numbers on the bottom edges show the input sizes for each layer of the CNN. Every layer uses a ReLU activation except for the last dense layer, which uses a sigmoid activation.



(a) (b) (c)

Fig. 4: Jaccard similarity for two CNNs, one trained with 14 mesh models (YCB Dataset), and the other trained with 486 mesh models (YCB dataset + Grasp Database). For each plot, the same two CNNs are evaluated on inputs they were trained on (trained views, left), novel inputs from meshes they were trained on (holdout views, center) and novel inputs from meshes they have never seen before (holdout models, right).

32 examples in a batch was randomly sampled from the full training set with replacement.

We used the Jaccard similarity to evaluate the similarity between a generated voxel occupancy grid and the ground truth. The Jaccard similarity between sets A and B is given by:

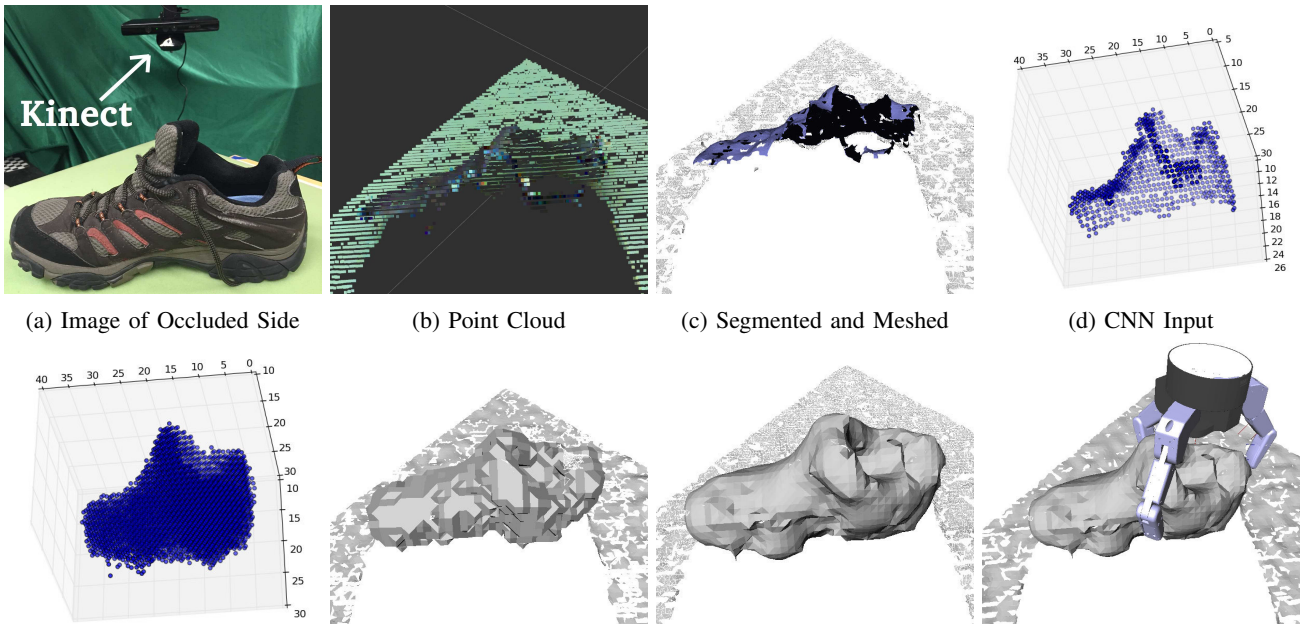
$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

The Jaccard similarity has a minimum value of 0, where A and B have no intersection and a maximum value of 1 where A and B are identical. During training, this similarity measure is computed for input meshes that were in the training data (trained views), meshes from objects within the training data but from novel views (holdout views), and for meshes of objects not in the training data (holdout models). The CNNs were trained with an NVIDIA Titan X GPU.

C. Training Results

Fig. 4 shows how the Jaccard similarity measures vary as the networks' training progresses. In order to explore how the quality of the reconstruction changes as the number of models in the training set is adjusted, we trained two networks with identical architectures using variable numbers of mesh models. One was trained with partial views from 14 YCB models, and the other was trained with the same 14 YCB models in addition to 472 Grasp Database mesh models. The remaining 4 YCB and 118 Grasp Dataset models were kept as a holdout set. Results are shown in Fig. 4. We note that the networks trained with fewer models perform better shape completion when they are tested on views of objects they have seen during training than networks trained on a larger number of models. This suggests that the network is able to memorize the training data for the smaller number of models but struggles to do so when trained on larger numbers. Conversely, the models trained on a larger number of objects perform better than those trained on a smaller number when asked to complete novel objects. Because, as we have seen, the networks trained on larger numbers of objects are unable to memorize all of the models seen in training, they may be forced to learn a more general completion strategy that will work for a wide variety of objects, allowing them to better generalize to objects not seen in training.

Fig. 4(a) shows the performance of the two CNNs on trained views. In this case, the fewer the mesh models used during training, the better the completion results. Fig. 4(b) shows how the CNNs performed on novel views of the mesh objects used during training. The fact that there is very little difference in performance on holdout views compared to training views leads us to believe that the objects have been



(a) Image of Occluded Side (b) Point Cloud (c) Segmented and Meshed (d) CNN Input

(e) CNN Output (f) Mesh (g) Smoothed Mesh (h) Grasp Planning

Fig. 5: Stages to the Runtime Pipeline. These images are not shown from the angle in which the data was captured in order to visualize the occluded regions. (a): An object to be grasped is placed in the scene. (b): A pointcloud is captured. (c): The pointcloud is segmented and meshed. (d): A partial mesh is selected by the user and then voxelized and passed into the 3D shape completion CNN. (e): The output of the CNN. (f): The resulting occupancy grid can be run through a marching cubes algorithm to obtain a mesh quickly. (g): Or, for better results, the output of the CNN can be combined with the observed pointcloud and preprocessed for smoothness before meshing. (h): Grasps are planned on the smoothed completed mesh. Note: this is a novel object not seen by the CNN during training.

sampled densely enough that the specific viewpoint no longer has much bearing on completion quality for the objects used in training. Fig. 4(c) shows the completion quality of the two CNNs on objects they have not seen before. In this case, as the number of mesh models used during training increases, performance improves as the system has learned to generalize a wider variety of inputs. The model trained on only the YCB dataset peaked at .707 Jaccard Similarity, while the model trained on the YCB + Grasp Dataset peaked with a Jaccard Similarity of .869.

IV. Runtime

At runtime the pointcloud for the target object is acquired from a 3D sensor, scaled, voxelized and then passed through the CNN. The output of the CNN, a completed voxel grid of the object, goes through a post processing algorithm that returns a mesh model of the completed object. Finally, a grasp can be planned and executed based on the completed mesh model. Fig. 5 demonstrates the full runtime pipeline on a novel object never seen before.

1) Acquire Target Pointcloud: First, a pointcloud is captured using a Microsoft Kinect, then segmented using Euclidean cluster extraction. A segment is selected either manually or automatically[21] corresponding to the object to be completed and passes it to the shape completion module.

2) Complete via CNN: The selected pointcloud is then used to create an occupancy grid with resolution 40x40x40. This occupancy grid is used as input to the CNN whose output is an equivalently sized occupancy grid for the completed shape. In order to fit the pointcloud to the 40x40x40 grid, it is scaled down uniformly so that the bounding box of the pointcloud fits in a 32x32x32 voxel cube, and then centered in the 40x40x40 grid such that the center of the bounding box of the pointcloud is at point (20, 20, 18) in the zero-indexed voxel grid. Finally, an element-wise floor function is applied to the point cloud point coordinates in order to snap points to voxels, and the corresponding voxels are marked as occupied. Placing the pointcloud slightly off-center in the z dimension leaves more space in the back half of the grid for the network to fill.

3a) Create Mesh: At this point, if the object being completed is not going to be grasped, then the voxel grid output by the CNN is run through the marching cubes algorithm to turn it into a mesh, and the resulting mesh is added to the planning scene, filling in the occluded regions.

3b) Create Smoothed Mesh: Alternatively, if this object is going to be grasped, then post-processing occurs. The output of the CNN is converted to a point cloud and its density is compared to the density of the partial view point cloud. The densities are computed by randomly sampling points and averaging the distances to the nearest neighbors. In general, the density of the CNN output is much lower

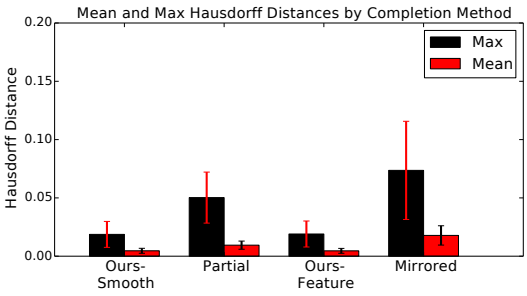


Fig. 6: Hausdorff Distance comparing meshes generated via various completion methods to the ground truth meshes.

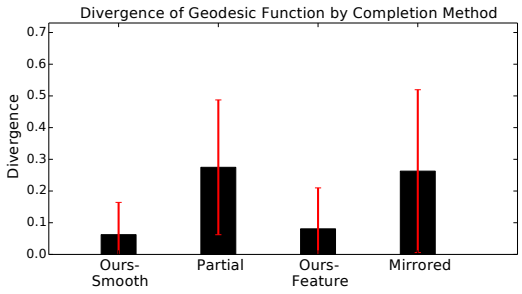


Fig. 7: Geodesic Divergence comparing meshes generated via various completion methods to the ground truth meshes.

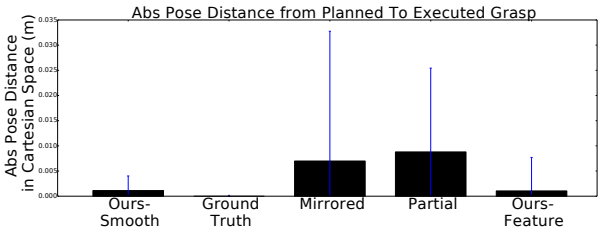


Fig. 9: Variation in Cartesian space of Barrett Hand base from planned to executed grasps by completion method.

than that of the partial view. The CNN output is up-sampled by an appropriate integral value to match the density of the partial view. The combined point cloud is then voxelized at the new higher resolution.

This voxel grid is smoothed using [22], as described in Alg. 1, which weights the voxels, minimizing the Laplacian on the boundary. The weighted voxel grid is run through either standard marching cubes, or a variant that detects edge and corner features, and estimates new feature points with linear approximations [23]. In this case, the features are detected pre-smoothing and the weights of the feature points are reset post-smoothing. This feature preserving variant was used because the sharp features can have a significant impact on grasp planning.

4) Grasp completed mesh: The reconstructed mesh is then loaded into GraspIt![24] where the simulated annealing grasp planner is run using a Barrett Hand model, The reachability of the planned grasps are checked using MoveIt![25], and the highest quality reachable grasp is then executed.

Algorithm 1 Mesh Reconstruction

```
procedure MeshReconstruction
    vox_raw ← upsampleMerge(cnn_out, observed_pc)
    vox_smooth ← smooth(vox_raw)
    if useFeatures then
        features ← detectCornersEdges(vox_raw)
        vox ← restoreFeatures(vox_smooth, features)
        mesh ← mCubesFeatures(vox)
    else
        mesh ← mCubes(vox_smooth)
    return mesh
```

V. Experimental Results

To compare our framework to other approaches, meshes were generated using four completion methods: smoothed marching cubes of the partial view (Partial), mirroring completion[8] (Mirror), our CNN completion with smoothing and feature detection (Ours-Feature), and our CNN completion with only smoothing (Ours-Smooth). The CNN was trained on the YCB + Grasp Dataset. 10 Randomly sampled views for each of 18 YCB objects were completed using the different completion methods. This resulted in 720 (18 object * 4 methods * 10 views) different completions. These meshes were evaluated by comparing them to the ground truth meshes using three metrics: Hausdorff distance, geodesic divergence, and the distance between planned and realized grasps in simulation.

A. *Hausdorff Distance*

The Hausdorff distance is a one-directional metric computed by sampling points on one mesh and computing the L2 distance of each sample point to its closest point on the other mesh. The mean value of a completion is the average distance from the sample points on the completion to their respective closest points on the ground truth mesh, and the max value is the single largest such distance. The symmetric Hausdorff distance was computed by running Meshlab's[26] Hausdorff distance filter in both directions. Fig. 6 shows the mean and standard deviation of the mean and max values of the symmetric Hausdorff distance for each completion method. In this metric, the CNN completions are significantly closer to the ground truth than both the partial and the mirrored completion. The feature detection has little effect because the number of feature points and the difference in error at those points is small. The mirrored completion's accuracy has a high variance as the quality is dependent on its ability to accurately capture a useful plane of symmetry.

B. *Geodesic Divergence*

The completions are also compared using a measure of Geodesic Divergence[27]. A geodesic shape descriptor is computed for each mesh. Then, a probability density function is computed for each mesh by considering the shape

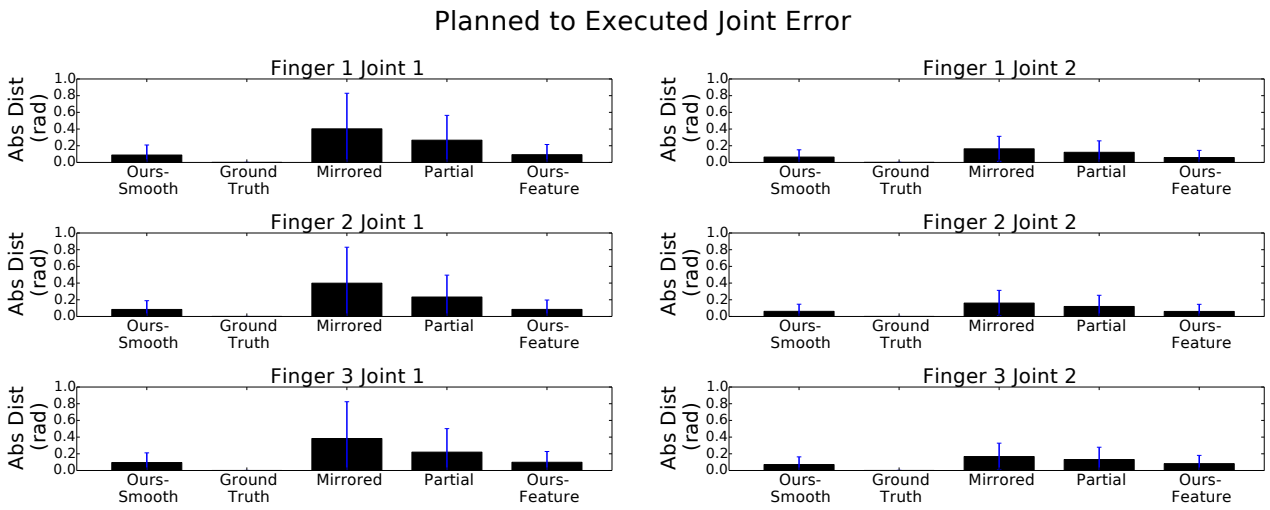


Fig. 8: Variation in joint space from planned to executed grasps for different completion methods. The Spread angle is not included, since it is set before approaching the object, there is no change in its value from planned to realized grasp.

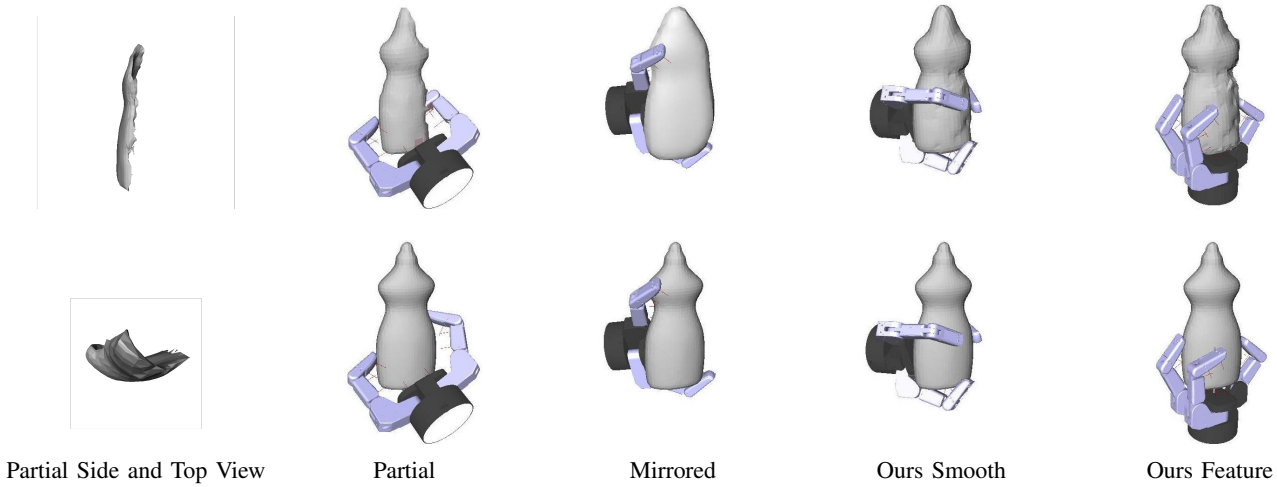


Fig. 10: Top Row: Planned Grasps using variety of completions methods. Bottom Row: Grasps from the top row executed on the Ground Truth object. Notice the partial completion is missing the entire right side of the object, causing the large discrepancy between the planned and executed grasp.

descriptor as a random distribution and approximating the distribution using a Gaussian Mixture Model. The probability density functions for each completion are compared with that of the ground truth mesh using the Jenson-Shannon probabilistic divergence. Fig. 7 shows the mean and standard deviation of the divergences for each completion method. The CNN completions show an improvement over both the partial views and the mirrored completions.

C. Simulation Based Grasp Comparison

In order to evaluate our framework's ability to enable grasp planning, the system was tested in simulation. While there are clear limitations in the ability of simulators to correctly mirror the real world, simulation does allow us to plan and evaluate thousands of grasps quickly. GraspIt!'s Multi-threaded planner was used with default settings to plan grasps on all of the completions of the objects. These grasps were then executed on the ground truth meshes in GraspIt!. In order to simulate a grasp execution, the completion was removed from GraspIt! and the ground truth object was inserted in its place using the original camera transform. Then the hand was placed 20cm backed off from the ground truth object along the approach direction of the grasp. The spread angle of the fingers was set, and the hand was moved along the approach direction of the planned grasp either until contact was made, or the grasp pose was reached. At this point, the fingers closed to the planned joint values. Then each finger continued to close until either contact was made with the object, or until the joint limits were reached. Fig. 10 shows several grasps and their realized executions for different completion methods. Visualizations of the simulation

Fig. 11: Barrett Hand(BH8-280), StaubliTX60 Arm, and objects used in experiments.

results for the entire YCB and Grasp Datasets are available at http://shapecompletiongrasping.cs.columbia.edu

Fig. 8 and Fig. 9 show the differences between the planned and realized joint states as well as the difference in pose of the base of the end effector between the planned and realized grasps. Both of these figures demonstrate that our completion method works better for enabling grasp planning than the partial and mirrored completions. Using our method caused the end effector to end up closer to where it intended to be in terms of both joint space and the palm's cartesian position. We found that the partial is not unreasonable in this metric given that the planner only interacts with regions of the object which are observed, of which we know the geometry. This does limit the approach direction of the grasps planned using the partial as the majority come from the same direction that the partial mesh was capture from, while our method and the mirroring tended to produce grasps whose approach directions were less dependent on viewing angle.

D. Performance on Real Hardware

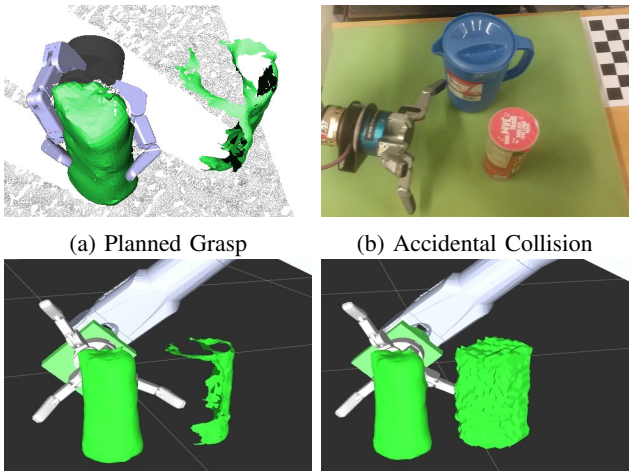
In order to further evaluate our framework, the system was used in an end-to-end manner using actual robotic hardware to execute grasps planned via the shape completion method described above. The object used are shown in Fig. 11. They were selected because they were sized to work well with the Barrett Hand. In this experiment, for each object, shape completions were generated and grasps were planned and attempted. The results are shown in Table I. One failure case involved the wooden doll which is a non-rigid object. The other two involved the detergent bottle and joystick, where the planned grasps enveloped small portions of the objects giving them good scores from the planner, despite not being stable in reality. Examples can be seen in the video attachment as well as at http://shapecompletiongrasping.cs.columbia.edu.

E. Crowded Scene Completion

Often objects in the scene are not going to be manipulated, and only require shape completion without smoothing and grasp planning in order to be successfully avoided. In this case, the output of the CNN can be run directly through marching cubes rather than our post processing to quickly create a mesh of the object. Fig. 12(a) shows a grasp planned using only the partial mesh for the object near the grasp target. Fig. 12(b)(c) show the robotic hand crashing into one of the nearby objects when attempting to execute the grasp. The failure is caused by an incomplete planning scene. Fig. 12(d) shows the scene with the nearby objects completed, though without smoothing. With this fuller picture, the planner accurately flags this grasp as unreachable. The time requirement for the scene completion process is given by:

$$T_{completion} = T_{segment} + T_{target} + T_{non\_target} * n$$

with Segmentation Time ($T_{segment}$), Target Completion Time (T_{target}), Non Target Completion Time(T_{non_target}) and Number of Non Target Objects (n). The system has the ability to both quickly fill in occluded regions of the scene, and selectively spend more time generating smoothed completions optimized for grasping on selected objects of interest. Several crowded scene configurations were created and each scene completion step was run 5 times. Average completion times are shown in Table. II



(a) Planned Grasp (b) Accidental Collision

(c) Unfilled Planning Scene (d) Filled Planning Scene

Fig. 12: The system can be used to quickly complete obstacles that are to be avoided. The arm fails to execute the grasp planned in (a). The resulting failure is shown in (b). The collision with the non-target object occurred due to a poor planning scene as shown in (c), but the CNN without post-processing can be used to fill the planning scene allowing the configuration to be correctly marked as invalid as shown in (d).

TABLE II: Crowded scene completions timing results.

Completion Stage	Average Times (s)
Scene Segmentation	2.680
Non-Target Completion (average/object)	0.198
Target Completion Smooth	4.917
Target Completion Feature	4.860

VI. Conclusion

This work presents a framework to train and utilize a CNN to complete and mesh an object observed from a single point of view, and then plan grasps on the completed object. The completion system is fast, with completions

Target	Novel	Success	Segment (s)	Preprocess (s)	CNN (s)	Postprocess (s)	Total Scene Processing (s)	Grasp Planning (s)	Total (s)
flashlight	Yes	Yes	1.512	0.083	0.074	5.592	7.26	38.44	45.70
upright drill	Yes	Yes	1.795	0.121	0.077	6.250	8.24	38.11	46.35
biplane	Yes	Yes	1.500	0.167	0.083	7.844	9.59	40.34	49.93
pringles	No	Yes	1.784	0.083	0.083	5.838	7.79	32.64	40.43
wooden doll	Yes	No	1.797	0.109	0.173	17.621	19.70	37.30	57.00
milk jug	Yes	Yes	1.801	0.066	0.072	4.816	6.755	37.04	43.80
joystick	Yes	No	1.605	0.153	0.080	6.910	8.75	32.73	41.48
detergent bottle	Yes	No	1.444	0.088	0.074	5.947	7.55	31.19	38.74
shoe	Yes	Yes	1.733	0.114	0.076	9.106	11.03	37.56	48.59
small drill	Yes	Yes	1.602	0.058	0.075	5.780	7.56	36.08	43.64
soccerball	No	Yes	1.532	0.038	0.078	2.553	4.20	35.73	39.93
coffee can	Yes	Yes	1.465	0.084	0.073	6.185	7.81	37.62	45.43
cardboard box	Yes	Yes	1.556	0.108	0.072	5.294	7.03	34.21	41.24
	Success:	10/13				Average Scene Processing Time:	8.71	Average Total Time:	44.79

TABLE I: Results from end to end robotic grasping task. The CNN was trained with 14 YCB objects and 472 Grasp Database objects. Only the soccerball, and pringles were both in the YCB dataset and used to train the network.

available in a matter of milliseconds, and post processed completions suitable for grasp planning available in several seconds. The dataset and code are open source and available for other researchers to use. It has also been demonstrated that our completions are better than more naive approaches in terms of Hausdorff and Geodesic Distances. In addition, grasps planned on completions generated using the proposed framework result in executed grasps closer to the intended hand configuration than grasps planned on completions from the other methods.

Acknowledgements: This work is supported by NSF Grant IIS-1208153. Thanks to the NVIDIA Corporation for the Titan X GPU grant.

REFERENCES

[1] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," in *Handbk. of brain theory & neural networks*, 1995.

[2] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," in *ACM siggraph computer graphics*, vol. 21, no. 4. ACM, 1987, pp. 163–169.

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

[4] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar, "The ycb object and model set: Towards common benchmarks for manipulation research," in *Advanced Robotics (ICAR), 2015 International Conference on*. IEEE, 2015, pp. 510–517.

[5] D. Kappler, J. Bohg, and S. Schaal, "Leveraging big data for grasp planning," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 4304–4311.

[6] Z. Wu, S. Song, A. Khosla, X. Tang, and J. Xiao, "3D shapenets for 2.5D object recognition and next-best-view prediction," *arXiv preprint arXiv:1406.5670*, 2014.

[7] J. Rock, T. Gupta, J. Thorsen, J. Gwak, D. Shin, and D. Hoiem, "Completing 3d object shape from one depth image," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 2484–2493.

[8] J. Bohg, M. Johnson-Roberson, B. León, J. Felip, X. Gratal, N. Bergström, D. Kragic, and A. Morales, "Mind the gap-robotic grasping under incomplete observation," in *Robotics and Automation (ICRA), IEEE Int. Conference on*. IEEE, 2011, pp. 686–693.

[9] A. H. Quispe, B. Milville, M. A. Gutiérrez, C. Erdogan, M. Stilman, H. Christensen, and H. B. Amor, "Exploiting symmetries and extrusions for grasping household objects," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2015, pp. 3702–3708.

[10] J. Mahler, S. Patil, B. Kehoe, J. van den Berg, M. Ciocarlie, P. Abbeel, and K. Goldberg, "GP-GPIS-OPT: Grasp planning with shape uncertainty using gaussian process implicit surfaces and sequential convex programming," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[11] J. Varley, J. Weisz, J. Weiss, and P. Allen, "Generating multi-fingered robotic grasps via deep learning," in *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*, 2015, pp. 4415–4420.

[12] I. Lenz, H. Lee, and A. Saxena, "Deep learning for detecting robotic grasps," *arXiv preprint arXiv:1301.3592*, 2013.

[13] P. Min, "Binvox, a 3d mesh voxelizer," 2004.

[14] N. Koenig and A. 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*, vol. 3. IEEE, 2004, pp. 2149–2154.

[15] F. Chollet, "Keras," https://github.com/fchollet/keras, 2015.

[16] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: a cpu and gpu math expression compiler," in *Proceedings of the Python for scientific computing conference (SciPy)*, vol. 4. Austin, TX, 2010.

[17] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley, and Y. Bengio, "Theano: new features and speed improvements," *arXiv:1211.5590*, 2012.

[18] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *arXiv preprint arXiv:1502.01852*, 2015.

[20] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the 13th AISTATS*, 2010, pp. 249–256.

[21] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[22] V. Lempitsky, "Surface extraction from binary volumes with higher-order smoothness," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 1197–1204.

[23] L. P. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel, "Feature sensitive surface extraction from volume data," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 2001, pp. 57–66.

[24] A. T. Miller and P. K. Allen, "Graspit! a versatile simulator for robotic grasping," *Robotics & Automation Magazine, IEEE*, vol. 11, no. 4, pp. 110–122, 2004.

[25] I. A. Sucan and S. Chitta, "Moveit!" *Online Availabl e: http://moveit.ros.org*, 2013.

[26] P. Cignoni, M. Corsini, and G. Ranzuglia, "Meshlab: an open-source 3d mesh processing system," *Ercim news*, vol. 73, pp. 45–46, 2008.

[27] A. B. Hamza and H. Krim, "Geodesic object representation and recognition," in *International conference on discrete geometry for computer imagery*. Springer, 2003, pp. 378–387.