Dex-Net AR: Distributed Deep Grasp Planning Using an Augmented Reality Application and a Smartphone Camera

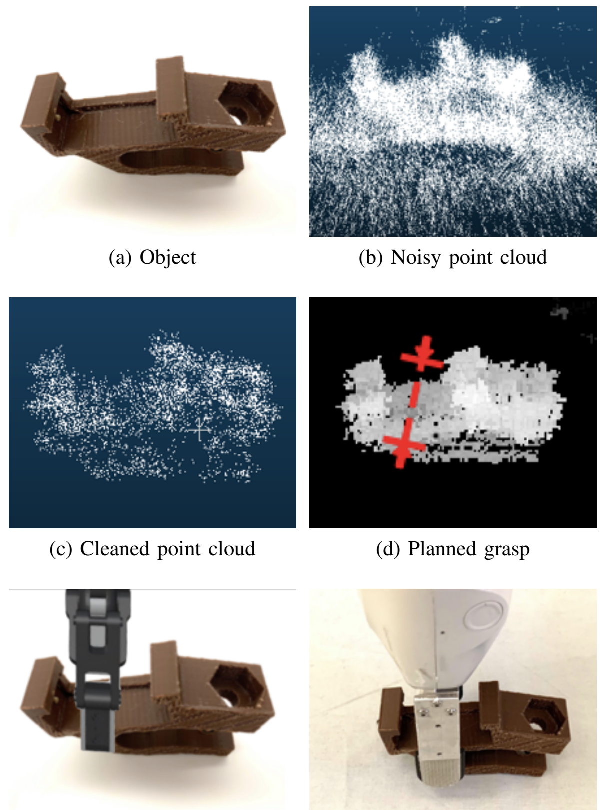
Harry Zhang, Jeffrey Ichnowski, Yahav Avigal, Joseph Gonzales, Ion Stoica, and Ken Goldberg[1]

***Abstract*— Recent consumer demand for augmented reality in mobile phone applications has accelerated performance of structure from motion methods that build a point cloud from a sequence of RGB images taken by the camera on a mobile phone as it is moved around an object. Smartphone apps, such as the Apple ARKit, have potential to expand access to deep grasp planning systems such as Dex-Net. However, the resulting point clouds are often noisy due to estimation errors. We present a distributed pipeline, Dex-Net AR, that allows point clouds to be sent to our lab, cleaned, and evaluated by Dex-Net grasp planner to generate a grasp axis that is returned and displayed as an overlay on the original object. We implement Dex-Net AR using the iPhone and ARKit to generate point clouds and compare results with those generated with high-performance depth sensors.**

I. Introduction

Grasping real-world objects using a robot gripper is complicated due to the limitations of sensing modalities. In highly controlled environments, such as industrial warehouses, sensing modalities such as depth sensors can be calibrated to their environment. However, in everyday life, such controlled circumstances are unlikely. To address this problem, we propose using structure from motion (SfM) to generate point clouds that can be used to generate grasps. To use and generate the data for SfM, one needs to move an RGB camera (such as one found on most smartphones) through 3D space to create a representative image set. Then, SfM extracts points from the images, reflecting the location of objects located in a 3D space. As the camera moves through space, the density of the point cloud increases, better detecting and defining the object's surfaces for grasping.

In this paper, we propose a system called Dex-Net AR that collects images, generates and cleans a point cloud, feeds it to a deep learning system, and generates high-quality grasp points for a robot such as Dex-Net 2.0 [1]. In our testing, we used an iPhone and ARKit [2], a popular consumer smartphone and a free software developer kit from Apple Inc., to generate a point cloud from which Dex-Net AR can compute grasp points and a mobile manipulator robotic grasp. Dex-Net AR can generate grasps with accuracy similar to state-of-the-art systems that rely on expensive, industry-grade depth sensors. Compared to depth camera systems that capture images from a fixed view, usually top-down, Dex-Net AR allows the user to move the smartphone camera all around the object, collecting three-dimensional point cloud



(a) Object (b) Noisy point cloud (c) Cleaned point cloud (d) Planned grasp (e) AR grasp overlay (f) ABB YuMi physical grasp

Fig. 1. Distributed deep grasp planning applied on a 3D printed object. **Top row:** Original object, and the visualization of raw point cloud data which exhibit a large amount of noise, collected by iPhone ARKit. **Middle row:** point cloud data after removing ground plane using RANSAC and a kNN-based cleaning process, and the planned grasp simulated by GQ-CNN. **Bottom row:** Augmented reality grasp overlay and real grasp by an ABB YuMi robot.

data. Such data are able to reflect more geometric details and features from other facets of the objects that are less likely to be captured by any depth camera from a top-down view, which can potentially reveal better grasp points in areas that are usually occluded or poorly inferred from a single fixed view.

This paper contributes:

1) A pipeline to record point cloud data of objects using commodity smartphones, clean up the collected data using an outlier removal algorithm based on Random Sample Consensus (RANSAC) and k-Nearest Neighbors (kNN) to clean up the point cloud data, and feed the point cloud data to a deep learning grasp planning

[1]The authors are with the University of California at Berkeley, Berkeley, CA, USA 94720 {harryhzhang, jeffi, yahav_avigal, jegonzal, istoica, goldberg}@berkeley.edu

tool, GQ-CNN, to plan grasps on the objects.

2) A method to show multiple facets of the objects to the robot in order to generate grasps with higher qualities.
3) Experiments measuring the advantage of 3D point clouds as input over traditional depth maps, that point clouds reveal more geometric information of the objects.

II. Related Work

a) Augmented Reality: Augmented reality (AR) is an interactive experience of a real-world environment where the objects in the real-world can be enhanced by perceptual information generated by a computer. It was first introduced by USAF Armstrong Labs [3] in order to create a virtual augmentation of a real environment to improve human performance in physical tasks. Recently, researchers have combined AR with computer vision techniques to recognize and classify objects in a real-world environment [4], [5], [6].

b) Structure from Motion: In 3D reconstruction, Structure from Motion (SfM) is used when 3D point positions are not known in advance [7]. SfM simultaneously recovers the 3D structure and pose of the camera from image correspondences given multiple frames of RGB images. In this way, SfM estimates the 3D locations of points on the object's notable geometric features from continuous frames. One limitation of SfM is that the objects being reconstructed must have notable geometric features such as contours, edges, and vertices. Thus, the objects need to be non-reflective and chromatic for the feature points to be detected and recorded.

c) Point Cloud Cleaning: To clean the SfM-generated 3D point cloud, we use a k-Nearest Neighbors (kNN) based approach which removes remote and isolated outliers. Ning et al. [8] developed a method to locally fit a plane using kNN and then project the near-surface, non-isolated outliers to the plane, further making the surface smoother and cleaner. In addition, Rakotosaona et al. [9] suggested a learning-based approach to denoise dense point cloud data. We build on this line of research by cleaning point cloud recorded by a smartphone to generate better quality grasps.

d) Grasp Planning: Grasp planning considers the problem of finding a gripper configuration that maximizes the Q value of grasp. There are several different approaches. Analytic approaches typically assume knowledge of the object and gripper state and consider the capability of constraining the object's motion [10] under perturbations and noises. Examples include GraspIt! [11], OpenGRASP [12], and the Dexterity Network (Dex-Net) 2.0 [1]. In order to fully satisfy the assumption of known state, analytic methods use a registration-based perception system: matching sensor data to known 3D models available in the existing database [13], [14], [15], [16], [17]. Empirical approaches [18] use learning-based methods to develop models that map sensor readings to success labels from humans or physical trials [19]. Both classes of approaches often use depth images taken from high-end depth cameras for both training and real data. In contrast, we explore planning grasps from relatively low-cost point cloud data taken from commodity devices, such as iPhones.

III. Problem Definition

We wish to take a sequence of images of an object from moving the camera of a commodity smartphone, and plan a grasp on the object. Suppose we move a camera around an object to scan it, during the recording process, which we define as a session, n frames are recorded. In each frame i, the camera captures an RGB image f_i, where $f_i \in \mathbb{R}^{W \times H \times 3}$. W and H are the width and height of f_i, and they vary depending on the camera we are using. Therefore, the input $\mathcal{F}$ is a sequence of captured RGB images:

$$\mathcal{F} = \{f_1, f_2, ..., f_n\}$$

In each frame, SfM can detect notable geometric features of the object in the RGB image, and record the features as points, where each point is a 3D vector in $\mathbb{R}^3$, representing the (x, y, z) coordinates in the camera's local coordinates system. Multiple features detected in a frame can then be recorded as a point cloud of this frame. Thus, from $\mathcal{F}$, we use SfM to generate point cloud data:

$$\mathcal{C}_{raw} = c_1 \cup c_2 \cup ... \cup c_n$$

where c_i is the set of points extracted from image f_i. In each frame, we also record the frame number, a camera transformation matrix, and the points' unique identifiers. However, for SfM to generate point clouds with higher qualities, the scanned objects should not be monochromatic, reflective, or small. Thus, the objects that perform better in a session are those with a decent amount of texture variation. Points extracted by SfM also contain a large amount of noisy points, so $\mathcal{C}_{raw}$, as an aggregation of all point clouds that are recorded, contains both point cloud of the object of interest and points from noise. Let $Q \subset \mathbb{R}^3$ be the actual points of the objects' surfaces captured in F, and let $X \subset \mathbb{R}^3 \setminus Q$ be the noise points that are captured. We want to clean $\mathcal{C}_{raw}$ by removing X from it to obtain a point cloud with less noise:

$$\mathcal{C}_{clean} = f(\mathcal{C}_{raw})$$

where f is our cleaning method applied to the aggregation of all point clouds from different frames.

With a cleaner aggregated point cloud of the object, we transform the data to a depth map $\mathcal{D}$:

$$\mathcal{D} = g\left(\mathcal{C}_{clean}, d_{plane}, \mathbf{n}_{plane}, Int, T\right)$$

where d_{plane} and $\mathbf{n}_{plane}$ are the depth and the normal vector, respectively, of the ground plane or desktop on which the object is placed, and Int is the known camera intrinsics matrix of a depth camera, and $T \in \mathbb{R}^{4 \times 4}$ is the camera pose transformation matrix.

To plan a grasp, we feed $\mathcal{D}$ into a convolutional neural network architecture $\mathcal{N}$ called GQ-CNN [1], [20], [21]:

$$Q = \mathcal{N}(\mathcal{D})$$

where the output value $Q \in [0, 1]$ represents the quality of the grasp planned. The objective is to generate a robust grasp while maintaining a relatively high Q value, which largely relies on high-quality point cloud data.

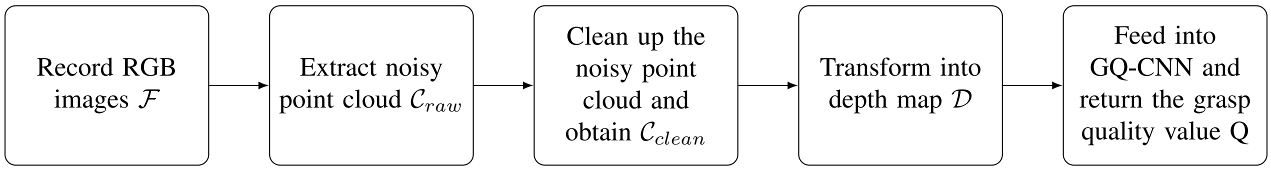


Fig. 2. Pipeline of Dex-Net AR

IV. Method

Our distributed system is divided into five steps. Fig 2 shows the pipeline of our method.

A. Point Cloud Data Recording

The purpose of the first and second steps of Fig 2 is to record the point cloud data of the object of interest. The input data $\mathcal{F}$ is a set of RGB images. SfM extracts the point cloud data, and points' unique identifiers from the RGB images. The aggregated point cloud generated from $\mathcal{F}$ contains large amounts of noise from the ground plane that the object is placed on. As a result, after extracting $\mathcal{C}_{raw}$ from $\mathcal{F}$, we have an extremely noisy and dense point cloud that contains large amounts of noise (X) such as from the ground plane which is not usable for the experiments because in this case grasp planning tool is likely to grasp the noise.

B. Point Cloud Data Cleaning

We observed a noise problem that we refer to as "drifting". During a session, the same geometric feature of an object is likely to be recorded multiple times across different frames, resulting in multiple points of the same geometric feature. Such points share the same unique identifier, but have different (x, y, z) coordinates. The points that share the same unique identifier tend to fit a straight line, hence the notion of a drift. However, different lines don't share the same direction, so the drift effect does not appear to be uniform across the point cloud.

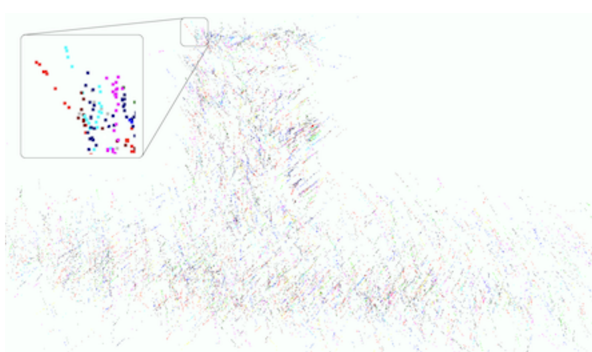


Fig. 3. Drifted points in a point cloud. Points that are extracted by the ARKit from the same feature and are from different frames are colored by the same color. A magnified window at the upper left corner exhibits the drift and shows that the drifting points roughly lie on the same line.

To clean up $\mathcal{C}_{raw}$ we average out points with same unique identifiers as the first cleaning technique. Suppose we record geometric feature k of the object and m drifts occur in total. We have m points with the same identifier, but different (x, y, z) coordinates. Each point $\mathbf{p}$ of geometric feature k has its coordinates (p_x, p_y, p_z), and the set of all points for geometric feature k is $\mathcal{P}_k$, so $m = |\mathcal{P}_k|$. We update feature k's points as follows:

$$\mathbf{p}' = \frac{1}{m} \sum_{\mathbf{p} \in \mathcal{P}_k} \mathbf{p}$$

As illustrated in Fig 3, when a point drifts, it roughly creates a straight line, and in most cases, the actual geometric feature point of the object lies right in the middle of the line. Thus, we are able to recover the actual geometric feature points by taking an average over drifted points. We iterate through every point of geometric feature based on its unique identifier. After this step, each geometric feature only has one point. While we have reduced the drifting problem, we now have a sparser point cloud, which we denote as $\mathcal{U}$. Our assumption, that the drifting problem originates from a linear transformation (rotation and translation), is reinforced by different point cloud registration methods we have tried (iterative closest point and bundle adjustment [7]). However, the results are similar to the averaging method while these methods are much more time-consuming.

Having reduced the drifts, we proceed to further clean up and denoise the new point cloud. First, we need to eliminate the dense ground plane that camera captures during a session. The points on the ground plane are a major source of noise in the point cloud. They are as dense as the feature points from the object, so it is essential to get rid of the noise points from the plane. We fit and remove the plane using RANSAC. After running RANSAC [22], [23] on $\mathcal{U}$, we obtain two output values $d_{plane}, \mathbf{n}_{plane}$, where d_{plane} is the threshold value in locally fit z-direction, representing the approximate z-coordinate of the plane that we are trying to get rid of, and $\mathbf{n}_{plane} \in \mathbb{R}^3$ is the approximate normal vector of the plane calculated by RANSAC. Using d_{plane} and $\mathbf{n}_{plane}$, we can cut off the points on the ground plane by rejecting any point $\mathbf{p}$ with:

$$p_z \leqslant d_{plane} \left(\mathbf{n}_{plane} \cdot \hat{\mathbf{k}} \right)$$

The above criterion separates the object from the plane in the point cloud $\mathcal{U}$. We call the separated object point cloud without plane $\mathcal{U}_{obj}$.

After rejecting the points from the ground plane noise in the point cloud, outliers still exist, and they are either near-surface or isolated. Such outliers are easier to remove because they are usually sparser than regular data points.

Manuscript 1559 submitted to 2020 International Conference on Robotics and Automation (ICRA). Received September 15, 2019.

Algorithm 1 kNN-Based Outliers Removal

Require: $\mathcal{U}_{obj}$, k, and ϵ
for $\mathbf{p} \in \mathcal{U}_{obj}$ **do**
  knn = k-Nearest Neighbors of $\mathbf{p}$
  dist = 0
  for $\mathbf{nn} \in$ knn **do**
    dist $\leftarrow$ dist + $\|\mathbf{p} - \mathbf{nn}\|$
  dist $\leftarrow$ dist / k
  if dist $> \epsilon$ **then**
    $\mathcal{U}_{obj} \leftarrow \mathcal{U}_{obj} \setminus \{\mathbf{p}\}$
return updated $\mathcal{U}_{obj}$ as $\mathcal{C}_{clean}$

We propose an algorithm based on k-Nearest Neighbor (kNN) to rid the point cloud of the sparse outliers by iterating through every point $\mathbf{p}$ in $\mathcal{U}_{obj}$.

In Algorithm 1, for each point $\mathbf{p}$, we calculate its kNN, and we reject the selected point if the average distance from it to its kNN is above a threshold value ϵ, meaning that the point is potentially a sparse outlier. Here, k and ϵ are hyperparameters.

The updated $\mathcal{U}_{obj}$ point cloud contains substantially less noise from the ground and isolated outliers. As a result, we obtain a better-quality point cloud data, and we can convert the updated $\mathcal{U}_{obj}$ into a depth map that is compatible with GQ-CNN. We denote the updated $\mathcal{U}_{obj}$ as $\mathcal{C}_{clean}$.

C. Transformation to a Depth Map

We want to transform the point clouds to depth maps because most grasp planning tools are based on the depth images captured by a depth camera. To generate a depth map, we first create an artificial depth camera, and we fix the camera at depth 0. Then to create an artificial "bin" to emulate regular robot grasping and bin-picking tasks, we use the output from RANSAC. First, since $d_{plane}\left(\mathbf{n}_{plane} \cdot \hat{\mathbf{k}}\right)$ represents the approximate z-coordinate of the ground plane in camera's local coordinates, we use this value as the depth of the bin or the desk in grasping scenarios, which should be the farthest from the camera.

One of the potential advantages of point clouds over traditional depth images is that a point cloud contains richer geometric information about the object: depth images only contain top-down views. Since we artificially create a depth camera, we can manipulate the camera pose in order to view the object from different angles, thus revealing more geometry of the objects which is potentially useful to generate more robust grasps. We use a camera pose matrix T to adjust the view orientations of the object, which can potentially reveal more information about grasp locations on other facets of the object.

One caveat about changing view orientation is that the system is not aware of the ground after we change view angle. Under this circumstance, the robotic arm might interfere with the plane when it is trying to grasp from the side. Even though such grasps have high Q values, interfering with the ground plane makes such grasps not applicable. To address this, we introduce a constraint function c, where c takes in a grasp, analyzes its pose, and outputs a boolean value. If the parallel jaws try to grasp some point beyond the plane limit, we reject the grasp, and c outputs False. Since GQ-CNN samples all possible grasps and outputs the grasp with the highest Q, we will return the grasp G with the highest Q such that it does not interfere with the ground plane and $c(G)$ returns True.

After setting the camera pose and defining the grasp constraint function, we obtain a depth map converted from the point cloud. Note that this depth map may have some holes in it because of the sparsity of the point cloud data. The resulting depth map is likely to be porous, where each hole is a group of zero-valued pixels. So one last step we do before feeding the image into GQ-CNN is to inpaint [24] the image. This step fills in the zero-valued pixels in the holes based on the values of surrounding pixels using OpenCV [25]. Having reduced the number of holes, we can then feed the image into GQ-CNN to plan a grasp.

D. Feeding into GQ-CNN

In this step, the pre-trained network GQ-CNN takes in a depth image and generates 100 potential grasps, where each potential grasp should satisfy the constraint function c. The output grasp will be the grasp with the highest Q value. We visualize the grasp with the overlaid grasp vector onto the depth map and record the Q value of the grasp.

V. Results

A. Simulation

To record the point cloud, we use an iPhone X with ARKit. ARKit uses SFM to extract feature points from an RGB image sequence [2]. As shown in Fig 1 and 3, the points collected by ARKit is extremely noisy. We use the default setting of the iPhone's camera, which is 60fps. In the simulation, we use parallel-jaw grippers with jaw widths of 5 and 10 centimeters. Therefore, the objects are chosen according to that size limit for the grippers to grasp successfully. We use $\epsilon = 0.005$ and $k = 10$, and this combination of hyperparameters gives us the best point cloud cleaning result. To add an artificial depth camera, we use the intrinsics of a Photoneo PhoXi camera, which is the camera that was used in GQ-CNN's training process, and the intrinsics are: $fx = 1122.0, fy = 1122.0, cx = 511.5, cy = 384.0$.

To test the proposed pipeline, we run the method on nine different objects. Other than the size limit, the objects should have relatively complex physical shapes in order to reflect discrepancy in geometry when viewed from different orientations. Object 9 in Fig 4 is actually a failure case. Such an object demonstrates the drawback of SfM, which is that it does not recognize features on reflective, monochromatic, or small objects.

In the trials, for each object, we record 5 point clouds of the object separately. For each point cloud, we plan 9 grasps based on 9 different view orientations of the camera: one grasp from traditional top-down view, four grasps from 45

Manuscript 1559 submitted to 2020 International Conference on Robotics and Automation (ICRA). Received September 15, 2019.

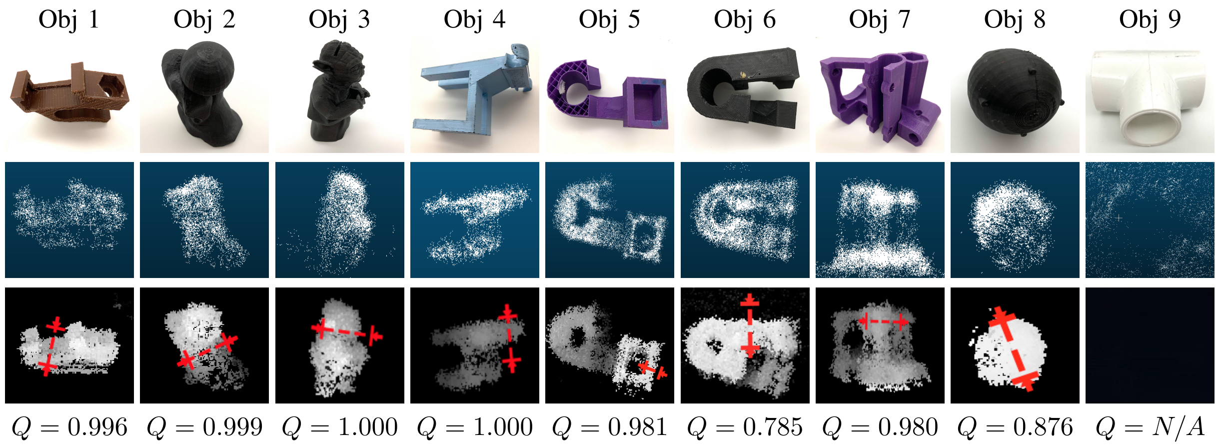
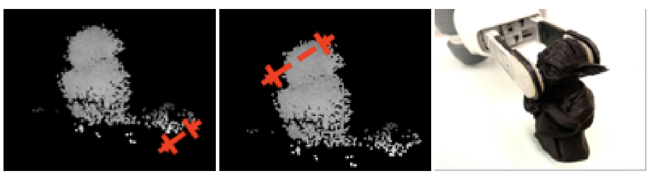


Fig. 4. Results with nine objects and their Q values. **Top Row:** photos. **Middle Row:** the cleaned and denoised point clouds. **Bottom Row:** planned grasps on the transformed depth images. Note that Obj 9 is a failure case due to the reflective and monochromatic feature of the object.



(a) Without constraint (b) With constraint (c) Physical grasp

Fig. 5. The usage of constraint functions prevents GQ-CNN from grasping ground noise. When the ground noise exists, we set the constraint function such that the gripper will not grasp any point near the ground surface

degrees camera poses, with each pose 90 degrees apart, and four grasps from 90 degrees camera poses, with each pose 90 degrees apart. In total, we end up having a dataset with 360 different grasps. From this dataset, we choose the best grasp of each object based on point cloud data quality, Q value, and grasp location. In the cases where a top-down view depth image cannot reveal enough geometric information, the generated grasps on those objects are not physically robust. Generally, when one view limits geometric information from view, the planned grasp is possibly bad, despite being the best grasp from that view. Therefore, viewing it from other directions such as 90 degrees or 45 degrees is likely to reveal more geometry of the objects, thus generating grasps with higher robustness.

In Fig 4, the grasps planned on Obj 2, Obj 3, and Obj 7 are not based on top-down view depth maps. In contrast, Obj 2's grasp is based on 45 degrees view orientation, and Obj 3 and 7's grasps are based on 90 degrees view orientation. The result grasps have better qualities than traditional grasps based on top-down view depth maps and are inaccessible from top-down views.

Good point cloud quality is an essential element to satisfactory grasps. When the cleaning process is not sufficiently aggressive, some noise on the ground plane fails to be removed. In such a case, GQ-CNN is likely to grasp noisy points on the ground plane. We resolve this problem by using the RANSAC procedure with higher threshold values and decreasing ϵ in the kNN-based outliers removal method. Meanwhile, we try to add in and fine-tune the constraint function that we create in order to prevent the parallel jaws from grasping any point on the ground plane surface or interfering with the ground plane. In most cases, when the parallel jaws do not collide with the ground plane, they are less likely to grasp noises. Hence, making use of and adjusting the constraint functions also alleviates the noise problem if we are planning grasps on a point cloud with bad quality. Fig. 5 shows a failure case when planning a grasp on Obj 3 from a 90 degrees view orientation that without a constraint function, GQ-CNN is planning to grasp the ground noise when it persists after RANSAC, and with a constraint function that keeps the gripper from grasping any point near the ground surface, even though the noise remains the same, the gripper now tries to grasp the object instead.

The pipeline takes approximately 3 minutes for an object to plan a grasp from any angle. The majority of the time is allocated to data collecting part of the pipeline. In order to sufficiently collect the geometry of the object, we need to carefully scan each facet three to four times, and we also need to reduce the speed of movement in order to keep the drifting minimal and stable. Therefore, the scanning process takes 2 minutes, and we fix this amount of time to all objects. Cleaning a recorded point cloud using RANSAC and Algorithm 1 typically takes 30 seconds, and converting the point cloud into a depth image and feeding the resulting depth image to GQ-CNN to plan a grasp takes less than 10 seconds.

B. Physical Experiments

To test Dex-Net AR on a physical robot, we execute it on an ABB YuMi robot. We reroute YuMi robot's input from depth images taken from a PhoXi depth camera to the artificial depth map converted from the point clouds recorded by an iPhone. For each object, we run the algorithm on 5 depth maps converted from 5 point clouds in top-down view orientation. For each depth map, we grasp the corresponding object twice. Therefore, we grasp each object 10 times in

Manuscript 1559 submitted to 2020 International Conference on Robotics and Automation (ICRA). Received September 15, 2019.

Objects										
	Obj 1	**Obj 2**	**Obj 3**	**Obj 4**	**Obj 5**	**Obj 6**	**Obj 7**	**Obj 8**	**Obj 9**	**Total**
Success Rate (%)	100	100	90	100	90	100	80	100	N/A	95
Processing Time (sec)	25	37	49	30	28	48	33	24	N/A	34.25

Fig. 6. Physical experiments result on 8 different objects, that were successfully recorded by SFM. Ten trials on each object.

total. To measure the performance, we use metrics in [1]:

1) **Success Rate:** as the percentage of grasps that we were able to lift, transport, and hold a desired object without collision when approaching the object.
2) **Processing Time:** as the amount of time spent to clean up the point cloud using RANSAC and Algorithm 1.

Since we keep the scanning time for all objects identical (2 minutes), the major difference of running time for the objects comes from the cleaning process. As the geometric complexity of the object increases, the number of points that are recorded by SfM also increases correspondingly, so the running time of running Algorithm 1 on the point cloud also increases. For example, Obj 8 in Fig 4 has the simplest geometry among all objects, so it requires the least cleaning time as shown in Fig 6.

From Fig 6, the average success rate for all eight objects (we have excluded Obj 9 whose point cloud failed to be recorded) is 95%.

VI. Conclusion

We present Dex-Net AR: a pipeline to plan grasps from data taken from commodity smartphones. With appropriate post-processing and cleaning methods, the point clouds collected by a smartphone can be used to plan robust grasps from different view orientations using Dex-Net, and used as input to pass into a physical robot to grasp the objects.

However, the time spent on data collection is exceedingly high: one needs to spend at least 120 seconds to scan the object in order to record sufficient data. Therefore, one potential improvement is that we can try to bring down the amount of time in video capturing using a learning-based method to augment and complete the point cloud data given that only limited data are available. Emerging smartphones may also have depth cameras [26], so we are planning on utilizing such smartphones to collect cleaner point clouds.

Furthermore, reconstructing the meshes from the point clouds is also another option to generate better grasps. Cleaner point clouds are also expected to be beneficial to grasp planning, so we are working on some faster algorithms to further clean the recorded point cloud.

Acknowledgements

This research was performed at the AUTOLAB at UC Berkeley in affiliation with the Berkeley AI Research (BAIR) Lab, Berkeley Deep Drive (BDD), the Real-Time Intelligent Secure Execution (RISE) Lab, and the CITRIS "People and Robots" (CPAR) Initiative. This research was supported in part by: the Scalable Collaborative Human-Robot Learning (SCHooL) Project, NSF National Robotics Initiative Award 1734633 and by a Google Cloud Focused Research Award for the Mechanical Search Project jointly to UC Berkeley's AUTOLAB and the Stanford Vision Learning Lab. The authors were supported in part by donations from Siemens, Google, Toyota Research Institute, Autodesk, Honda, Intel, Hewlett-Packard and by equipment grants from PhotoNeo, NVidia, and Intuitive Surgical. We thank our colleagues who provided helpful feedback and suggestions, in particular Priya Sundaresan, Jackson Chui, Michael Danielczuk, Kate Sanders, and Ajay Tanwani.

References

[1] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg, "Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics," 2017.

[2] (2019) ARKit: Apple Developer Documentation. [Online]. Available: https://web.archive.org/web/20190912200131/https://developer.apple.com/documentation/arkit

[3] L. B. Rosenberg, "The use of virtual fixtures as perceptual overlays to enhance operator performance in remote environments." Stanford Univ Ca Center for Design Research, Tech. Rep., 1992.

[4] T. Lee and T. Hollerer, "Handy ar: Markerless inspection of augmented reality objects using fingertip tracking," in *2007 11th IEEE International Symposium on Wearable Computers*. IEEE, 2007, pp. 83–90.

[5] M. Billinghurst, A. Clark, G. Lee *et al.*, "A survey of augmented reality," *Foundations and Trends® in Human–Computer Interaction*, vol. 8, no. 2-3, pp. 73–272, 2015.

[6] T. Starner, S. Mann, B. Rhodes, J. Levine, J. Healey, D. Kirsch, R. W. Picard, and A. Pentland, "Augmented reality through wearable computing," *Presence: Teleoperators & Virtual Environments*, vol. 6, no. 4, pp. 386–398, 1997.

[7] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[8] X. Ning, F. Li, G. Tian, and Y. Wang, "An efficient outlier removal method for scattered point cloud data," *PloS one*, vol. 13, no. 8, p. e0201280, 2018.

[9] M.-J. Rakotosaona, V. La Barbera, P. Guerrero, N. J. Mitra, and M. Ovsjanikov, "Pointcleannet: Learning to denoise and remove outliers from dense point clouds," in *Computer Graphics Forum*. Wiley Online Library, 2019.

[10] A. Rodriguez, M. T. Mason, and S. Ferry, "From caging to grasping," *The International Journal of Robotics Research*, vol. 31, no. 7, pp. 886–900, 2012.

[11] C. Goldfeder, M. Ciocarlie, H. Dang, and P. K. Allen, "The columbia grasp database," in *2009 IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 1710–1716.

[12] B. León, S. Ulbrich, R. Diankov, G. Puche, M. Przybylski, A. Morales, T. Asfour, S. Moisio, J. Bohg, J. Kuffner *et al.*, "Opengrasp: a toolkit for robot grasping simulation," in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 2010, pp. 109–120.

[13] P. Brook, M. Ciocarlie, and K. Hsiao, "Collaborative grasp planning with multiple object representations," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 2851–2858.

[14] M. Ciocarlie, K. Hsiao, E. G. Jones, S. Chitta, R. B. Rusu, and I. A. Şucan, "Towards reliable grasping and manipulation in household environments," in *Experimental Robotics*. Springer, 2014, pp. 241–252.

[15] C. Hernandez, M. Bharatheesha, W. Ko, H. Gaiser, J. Tan, K. van Deurzen, M. de Vries, B. Van Mil, J. van Egmond, R. Burger *et al.*, "Team delft's robot winner of the amazon picking challenge 2016," in *Robot World Cup*. Springer, 2016, pp. 613–624.

[16] B. Kehoe, A. Matsukawa, S. Candido, J. Kuffner, and K. Goldberg, "Cloud-based robot grasping with the google object recognition engine," in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 4263–4270.

[17] S. Hinterstoisser, S. Holzer, C. Cagniart, S. Ilic, K. Konolige, N. Navab, and V. Lepetit, "Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes," in *2011*

international conference on computer vision. IEEE, 2011, pp. 858–865.

[18] J. Bohg, A. Morales, T. Asfour, and D. Kragic, “Data-driven grasp synthesis—a survey,” *IEEE Transactions on Robotics*, vol. 30, no. 2, pp. 289–309, 2013.

[19] D. Wang, D. Tseng, P. Li, Y. Jiang, M. Guo, M. Danielczuk, J. Mahler, J. Ichnowski, and K. Goldberg, “Adversarial grasp objects.”

[20] J. Mahler and K. Goldberg, “Learning deep policies for robot bin picking by simulating robust grasping sequences,” in *Proceedings of the 1st Annual Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, S. Levine, V. Vanhoucke, and K. Goldberg, Eds., vol. 78. PMLR, 13–15 Nov 2017, pp. 515–524.

[21] V. Satish, J. Mahler, and K. Goldberg, “On-policy dataset synthesis for learning robot grasping policies using fully convolutional deep networks,” *IEEE Robotics and Automation Letters*, 2019.

[22] M. A. Fischler and R. C. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[23] M. Y. Yang and W. Förstner, “Plane detection in point cloud data,” in *Proceedings of the 2nd int conf on machine control guidance, Bonn*, vol. 1, 2010, pp. 95–104.

[24] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester, “Image inpainting,” in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 2000, pp. 417–424.

[25] G. Bradski, “The OpenCV Library,” *Dr. Dobb’s Journal of Software Tools*, 2000.

[26] (2019) Samsung Galaxy S10. [Online]. Available: https://www.samsung.com/us/mobile/galaxy-s10/camera/

Manuscript 1559 submitted to 2020 International Conference on Robotics and Automation (ICRA). Received September 15, 2019.