Dex-Net as a Service (DNaaS): A Cloud-Based Robust Robot Grasp Planning System

Pusong Li[1], Bill DeRose[2], Jeffrey Mahler[1], Juan Aparicio Ojea[3], Ken Goldberg[1,2]

***Abstract*— Accessing software resources via the Cloud has become increasingly popular as a means to configure and manage automation systems with reduced infrastructure overhead. Dex-Net as a Service (DNaaS) is a cloud-based grasp planning system for parallel-jaw grippers that provides a graphical user interface and API access to Dex-Net, a robust grasp planning system based on wrench mechanics and stochastic sampling. DNaaS allows anyone online to compute grasps on triangular meshes using parametric parallel-jaw grippers and visualize the results at** `http://automation.berkeley.edu/dex-net`.

We analyze grasps planned by DNaaS for multiple parallel-jaw grippers and adversarial object meshes, report system timing benchmarks, and present failure modes encountered during the development of DNaaS. Our experiments find that DNaaS takes under 75 seconds to process grasp requests on adversarial meshes using a parameterized gripper model.

I. Introduction

The ability to compute robust robot grasps remains a grand challenge for robotics in manufacturing, agriculture, and home care. Today, most robots and automation systems operate independently using onboard computation and memory. The development of Cloud Robotics [30] highlights the role that collective robot learning, Cloud Computing, and open-source software can play in achieving robust robotic manipulation of everyday objects [36]. Whether leveraging analytic methods or learning based approaches to achieve dextrous robotic manipulation, the increasing ubiquity of Cloud resources suggests new approaches to robot grasping, where processing is performed remotely with access to large shared datasets, can increase the reliability, performance, and cross-platform flexibility of robotic systems.

Robotics and Automation as a Service (RAaaS) [36] can play an important role in a Cloud Robotics framework by avoiding complex software installation and maintenance, allowing remote robots to scale beyond their onboard hardware limitations, and facilitating the sharing of robot trajectories and outcomes. RAaaS benefits users building robotics applications by making state-of-the-art algorithms and training datasets available without the need to perform data collection or algorithm implementation themselves.

The Dexterity-Network 1.0 (Dex-Net) [43] is an algorithm for robust grasp planning which relies on Unix-based libraries and operating systems. We present Dex-Net as a Service (DNaaS), an HTTP API which uses Dex-Net [43] to compute and rank parallel-jaw grasps on triangular-faced, 3D object meshes using parametric parallel-jaw grippers under user-specified physics and robustness parameters. Grasps generated by DNaaS can be viewed in a web browser where users may choose their desired gripper parameters, explore grasps on example watertight 3D object meshes, and upload custom objects to evaluate DNaaS on their own meshes. Users interact with mesh objects and candidate grasps in a 3D scene with the ability to filter grasps to specific stable poses. This allows industrial practitioners to easily access Dex-Net through the Web.

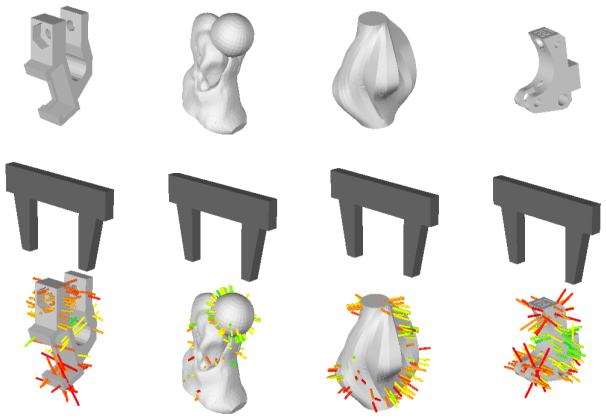


Fig. 1: DNaaS grasp generation examples. Adversarial mesh objects (top) and the corresponding grasps (bottom) generated by DNaaS for the pictured parallel-jaw gripper. Candidate grasps are represented by oriented line segments along the grasp axis, colored by their robustness (green more robust, red less robust).

This paper makes three contributions:

1) DNaaS, a RAaaS architecture for Dex-Net and public HTTP API which takes as input a 3D object mesh (in `.obj` format with triangular faces) and computes stable poses, grasps, and robust grasp quality metrics for parallel-jaw grippers under uncertainty in object pose, gripper pose, and friction,
2) An implemented graphical interface where users upload object meshes and visualize the quality of candidate grasps to better understand or debug grasping models, and
3) Experiments evaluating failure modes of the system, the real-time performance of the DNaaS API across multiple parallel-jaw grippers and adversarial meshes, and the effect of gripper width on predicted grasp quality.

[1] Dept. of Electrical Engineering and Computer Science;
[2] Dept. of Industrial Engineering and Operations Research;
University of California, Berkeley, USA {alanpusongli, bderose, jmahler, goldberg}@berkeley.edu
[3] Siemens Corporation; juan.aparicio@siemens.com

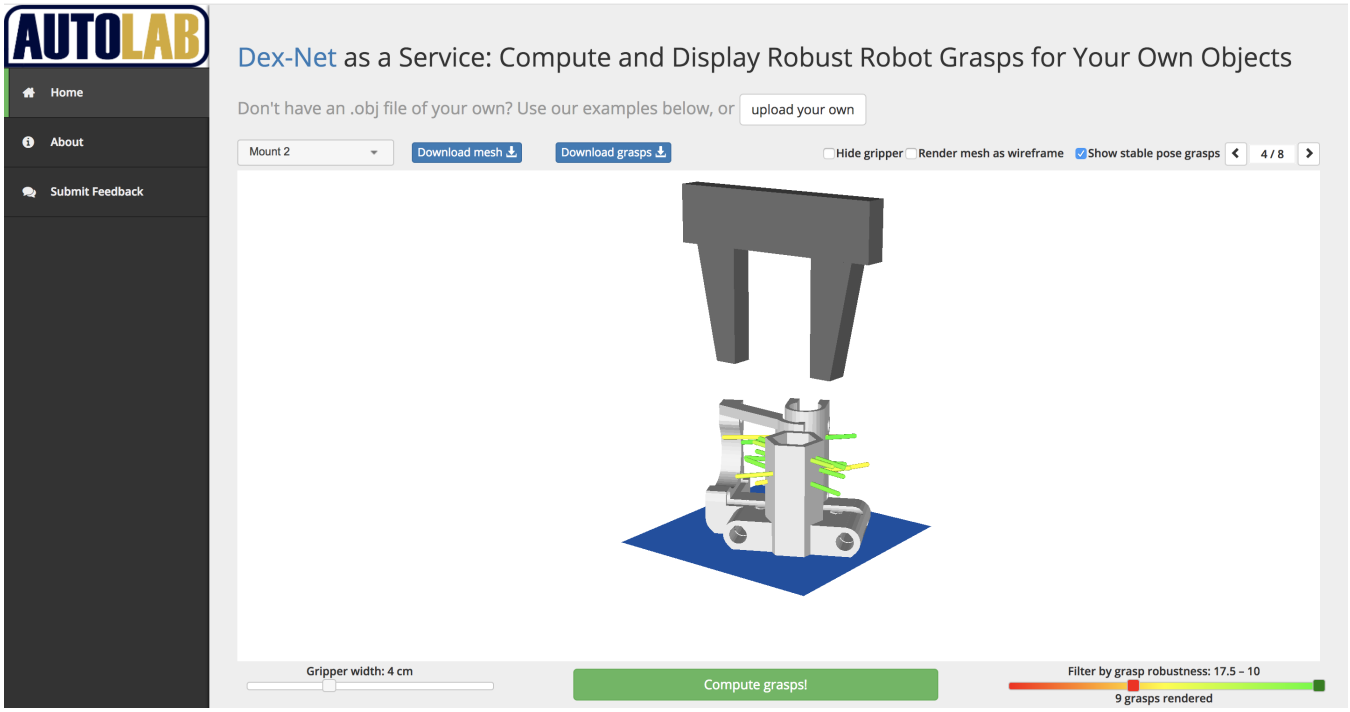


Fig. 2: DNaaS user interface. Parametric gripper model is rendered along the z-axis. Candidate grasps (green more robust, red less robust) are super-imposed on the target object. The gripper width can be adjusted manually (bottom left). Grasps may be filtered by their grasp quality (bottom right) or by their feasibility for a given stable pose (top right).

II. Related Work

Cloud robotics. The term "Cloud Robotics" was coined in 2010 [37] to describe systems that allow robots to offload compute and storage requirements from local hardware to the Cloud [30]. Cloud Robotics, spurred on by the general availability of Cloud Computing resources, has had an impact on robotics efforts in industry [3], [8], [9], [15]. Though a complete survey of Cloud Robotics is beyond the scope of this paper, we direct interested readers to works which cover the topic in more depth [26], [36].

Of particular interest are grasp planning and simulation systems, such as GraspIt! [29] and OpenGRASP [41], which allow users to interact with objects in a virtual world and compute quantitative grasp quality metrics to evaluate candidate grasps. These simulators have collected novel datasets [44] of thousands of distinct object models and associated grasps to bootstrap the study of grasp planning at scale. More recently, Dex-Net [42], [43] has addressed the issue of robust grasp planning and dataset generation for learning-based grasping algorithms.

Programs like GraspIt! may be downloaded and run by end-users locally, or used with ROS [11], [18]. RAaaS [36] introduces a layer of abstraction on top of systems like ROS to isolate end-users from manual management of Cloud Robotics software. RAaaS relies on software developers to deploy their projects to the Cloud where they may be used directly by end-users, each with possibly different robots and applications, without knowing the details of a specific robotic software package.

Brass [51] implements a RAaaS grasp-planning solution similar to DNaaS. DNaaS accepts gripper-parameterized grasp requests via a public HTTP REST API to enable the computation of robust grasps over a wide variety of parallel-jaw grippers. These grasps may then be visualized in a web user interface. Our experiments examine candidate grasps generated by DNaaS and provide univariate distributions of our grasp quality metric for different parallel-jaw grippers.

Grasp Planning. Given an object, gripper parameters, and reachability constraints due to the environment, grasp planning considers finding a gripper configuration that maximizes a certain metric. Methods fall into one of two categories based on success criteria: *analytic* methods [47], which consider performance according to physical models such as the ability to resist external wrenches [46], and *empirical* (or data-driven) methods [20], which typically use human labels [19] or the ability to lift the object in physical trials [45].

Analytic Methods. Analytic approaches typically assume that object and contact locations are known exactly and consider either the ability to resist external wrenches [47] or the ability to constrain the object's motion [49]. To execute grasps on a physical robot, one approach is to precompute a database of known 3D objects labeled with grasps and quality metrics using software like GraspIt! [29]. Precomputed grasps are indexed at execution time using *point cloud registration*: matching point clouds to known 3D object models in the database using visual and geometric similarity [20], [21], [22], [28], [31], [33], [35].

Robust grasp planning methods maximize grasp *robustness*, or the expected value of an analytic metric under

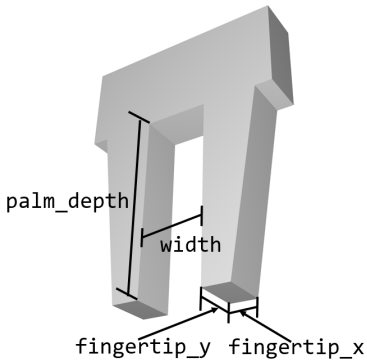


Fig. 3: User specifiable parameters for parallel-jaw gripper in DNaaS.

uncertainty in sensing and control. This involves labeling grasps on a database of 3D object models with robust metrics such as the probability of force closure [35] or the pose error robust metric [53]. Recent research has demonstrated that the sampling complexity of robust grasp planning can be improved using Multi-Armed Bandits [39] and datasets of prior 3D objects and robust grasps, such as the Dexterity Network (Dex-Net) 1.0 [43].

Empirical Methods. Empirical approaches typically use machine learning to develop models that map from robotic sensor readings directly to success labels from humans or physical trials. Human labels may be expensive to acquire for large datasets and irregular objects. Research in this area has largely focused on associating human labels with graspable regions in RGB-D images [40] or point clouds [23], [32], [34]. Lenz et al. [40] created a dataset of over 1k RGB-D images with human labels of successful and unsuccessful grasping regions, which has been used to train fast CNN-based detection models [24], [38], [48].

III. Problem Statement

The goal of Dex-Net as a Service (DNaaS) is to provide a public API which computes a set of robust parallel-jaw grasps for a given 3D object using a Cloud-based implementation of the Dex-Net grasp computation pipeline [43]. Specifically, DNaaS takes as input an object specified as 3D triangular mesh and outputs a set of collision-free parallel-jaw grasps ranked by their robustness to perturbations in object pose, gripper pose, and the Coulomb friction coefficient. Optionally, users can set application-specific parameters of the robust quasi-static analysis engine: the parallel-jaw gripper geometry, the Coulomb friction coefficient, and the grasp quality metric (either force closure [53] or the epsilon metric [25]). DNaaS can also optionally compute the subsets of grasps for the 3D object that have an axis parallel to the table plane for each stable resting pose of the object on a planar worksurface [27], which may be useful in industrial applications.

A. Assumptions

The robust quasi-static grasp analysis engine of DNaaS assumes quasi-static physics, a rigid object with uniform

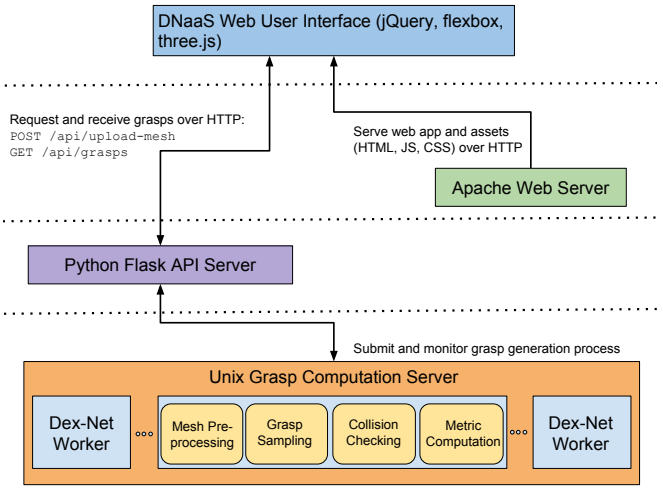


Fig. 4: DNaaS architecture. DNaaS consists of a front-end website, served statically by an Apache web-server, and a backend Python Flask API which communicates with a worker pool of Dex-Net processes to orchestrates grasp generation.

mass density, and a known friction coefficient. We assume that the parallel-jaw gripper geometry can be approximated with four parameters: the gripper width, the length of the fingers, and the cross-sectional dimensions of the fingertip, (see Figure 3). DNaaS assumes that the input mesh has triangular faces and fewer than 70k total faces to ensure grasp computation latency remains under two-minutes.

IV. Dex-Net as a Service (DNaaS) Architecture

Dex-Net as a Service (DNaaS) is comprised of multiple distinct layers, depicted in Figure 4. The frontend of the system is a web-based graphical user interface based on jQuery [12], [13] that parses user mesh models and grasp computation requests from a web browser. The frontend uploads mesh models and makes requests for grasp computations via DNaaS's public grasping API [5]. Requests are forwarded to the robust grasp analysis backend using a Python-based Flask API. The backend spawns worker processes which analyze the input mesh model using the robust grasp analysis engine from Dex-Net 1.0. Each worker process returns a set of parallel-jaw grasps with robustness metrics. The grasps are retrieved from the worker by a monitor process on the server, which relays the JSON encoded grasps to the DNaaS frontend via HTTP. Finally, the frontend renders the grasps on the 3D object model in the browser. Readers interested in using the DNaaS API may consult our Python example [6] as a guide to computing candidate grasps and stable-poses for object meshes.

A. Grasp Computation

The DNaaS backend uses an updated version of the robust grasp analysis engine of Dex-Net 1.0 that represents the object as an explicit surface (3D triangular mesh) rather than an implicit surface (3D Signed Distance Function), as was used in the original system [43]. When a user uploads a

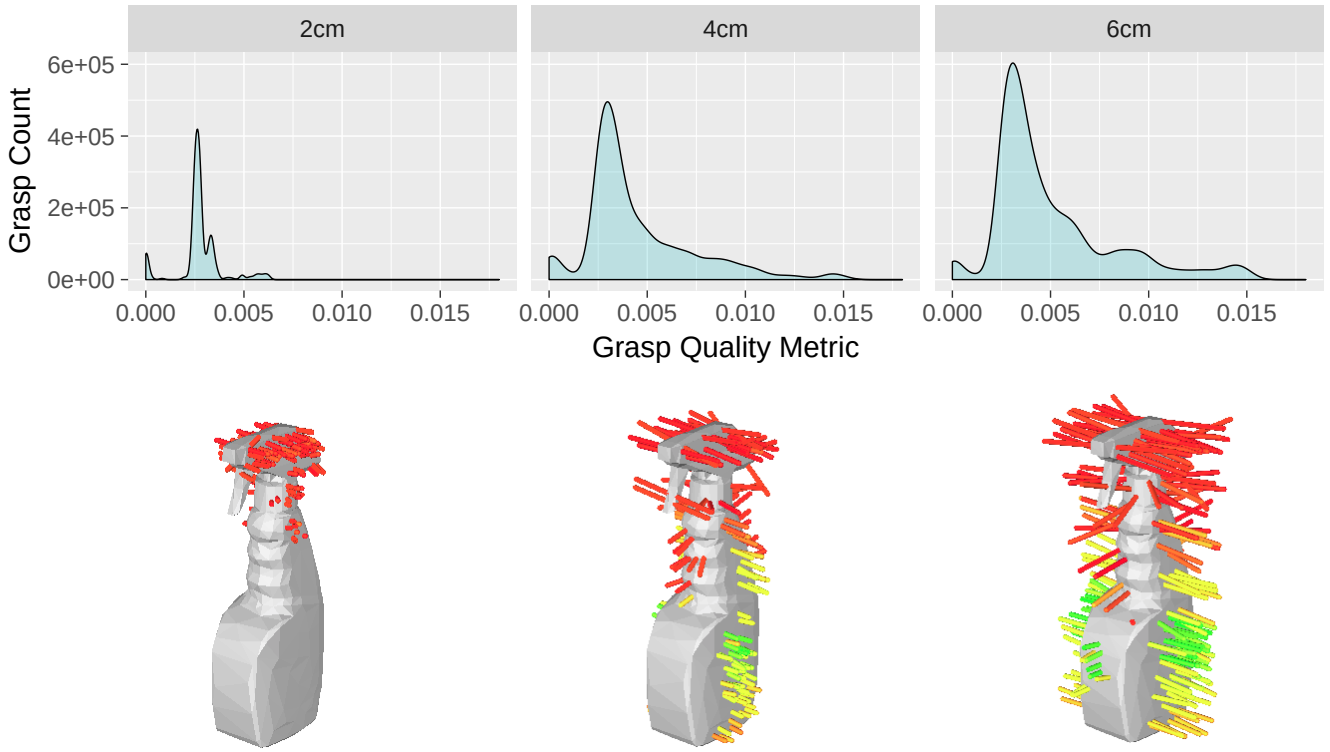


Fig. 5: Grasp quality and count for three gripper widths (2, 4, and 6 cm) generated by DNaaS on the spray bottle mesh object depicted in the bottom series. The gripper width increases from left to right. The images of the grasps superimposed on the spray bottle are taken from the DNaaS web interface directly. The dimensions of the spray bottle are 12.7cm tall, 3.1cm deep, and 5.0cm wide.

3D object mesh to DNaaS, the service first preprocesses the mesh for grasp computation. We align the principal axes of the object with the x, y, and z axes. We estimate the center of mass for the object by assuming a uniform density when the mesh is watertight and taking the centroid of the mesh bounding box otherwise. Next, we compute the stable resting poses of the object on a planar worksurface under quasi-static physics and a uniform initial object orientation [27].

The second stage in our pipeline samples an initial set of antipodal grasps using an implementation of the grasp sampling algorithm of Dex-Net 1.0 that operates on triangular meshes. First, we sample a set of candidate contact points from the surface of the mesh that are approximately evenly spaced using the trimesh library's [17] implementation of the triangle point picking algorithm [52]. For each candidate contact point, we search for a second contact point to form an antipodal pair by sampling a direction uniformly at random from the friction cone around the contact normal (inward-pointing surface normal) and tracing a ray along the sampled direction to find the most distant point of intersection with the mesh surface that is within the maximum opening width of the gripper. If no such intersection point exists, the candidate contact point is discarded. If one exists, we compute the surface normal and friction cone at the point of intersection and determine whether the candidate contact point and point of intersection form an antipodal pair. If the pair is antipodal, then we construct a candidate grasp with center at the midpoint between the pairs and an axis along the line between the points and add the candidate grasp to the set.

The set of antipodal contact points is then pruned by checking for collision-free configurations of the gripper relative to the object that reach the contact points. We search over all rotations of the gripper about the grasp axis (line between the contact points). For each grasp and stable resting pose, we also check whether or not the grasp axis is parallel to the planar worksurface to mark valid crane grasps.

Finally, we compute grasp robustness for the set of candidate grasps using Monte-Carlo sampling. For each grasp we iteratively sample an object pose, a gripper pose, and a friction coefficient from Gaussian distributions using the graphical model of [50]. We then compute the contact points for the perturbed grasp and evaluate the grasp quality metric. The backend implements force closure using a soft finger contact model by computing the angle between the line between the contacts and the friction cone, and it also implements the epsilon metric by Ferrari and Canny [25] using a Python implementation based on pyhull [14]. We estimate the mean and standard deviation of the quality metric over all samples. We stop sampling when either (a) the 95% upper confidence bound on the quality metric is less than a threshold value or (b) a maximum number of samples have been reached. The final set of grasps and metrics are JSON-encoded and returned to the end user when queried.

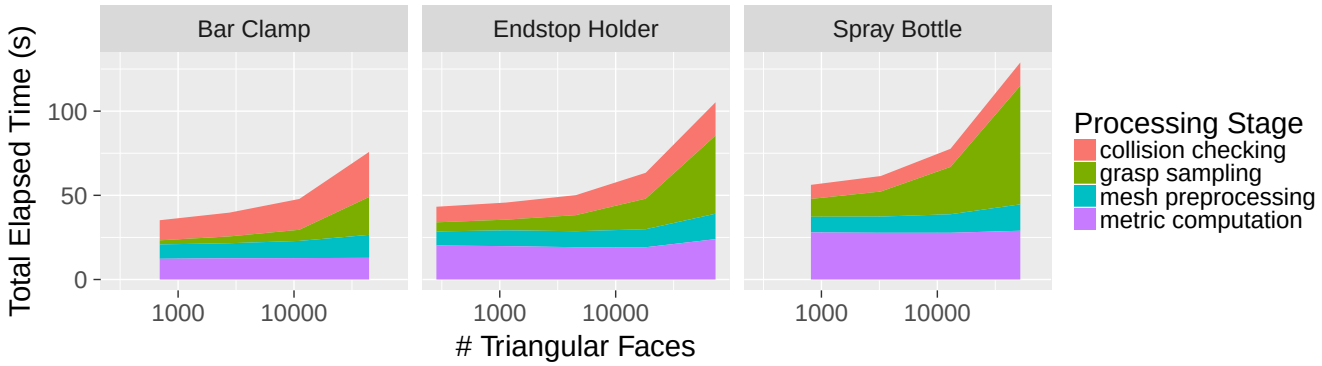


Fig. 6: Time analysis. Measuring the elapsed time of the major stages in the grasp generation pipeline, segmented by mesh, and colored by the processing stage. Timing measurements are given for three adversarial object meshes: a bar clamp, an endstop holder, and a spray bottle. Note: x-scale is log-transformed.

B. HTTP API

The full API specification implemented by DNaaS is available online [5] with an accompanying example of how to query DNaaS for grasp candidates [6].

Each request to process a mesh is associated with a globally unique identifier. Once computation is complete, any future calls to the `grasps` endpoint associated with this identifier bypass the computational pipeline and immediately return the previously computed grasps, allowing for efficient reuse of prior computation.

C. Web Interface

Figure 2 shows the DNaaS web user interface. When users visit the site, they can explore the grasps DNaaS predicts on a example set of 3D object meshes from the Dex-Net database. The quality of different grasps is visually distinguished on a spectrum from green (more robust) to red (less robust) by using the grasp quality metrics DNaaS computes. The interface allows users to cycle through an object's stable poses and filter grasps based on their robustness or feasibility for a given pose. Users may also compute and visualize grasps on objects they upload themselves and, for any object, the mesh and the candidate grasps are downloadable through the website.

The client-side user interface and server-side Flask API run, alongside Dex-Net itself, on a quad-core Intel(R) Xeon(R) CPU E3-1220 v3 with a clockrate of 3.10GHz and 16GB of RAM. The website is written using HTML, JavaScript, and CSS served statically by an Apache web server [1]. We use three.js [16] to render a 360° 3D scene in the browser where candidate grasps are super-imposed on the target object mesh. The page is designed using a flexible box layout [10] for easy accessibility across modern web-browsers (Chrome, Safari, Firefox) and on mobile devices. The website uses the latest version of jQuery [12] for DOM [7] manipulation, event handling, and Promise-based [2] asynchronous HTTP requests. The graphical user interface combines elements from jQuery UI [13], Bootstrap [4], and custom CSS.

V. Experiments

We present grasp examples generated by DNaaS for multiple parallel-jaw grippers and adversarial object meshes. We report detailed system timing measurements and present failure modes encountered during the development of DNaaS.

A. Grasp Quality and Gripper Width

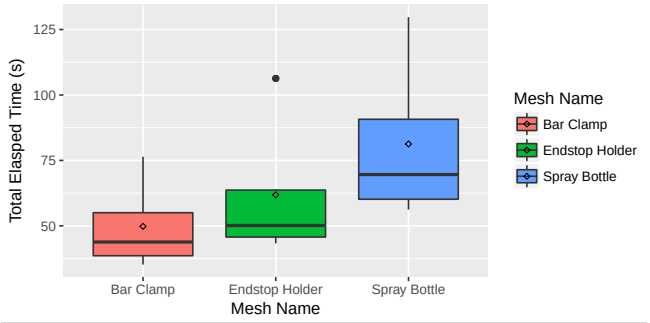
DNaaS's ability to process grasp requests using parametric parallel-jaw grippers allows us to describe the differences between sets of grasps generated for different gripper parameterizations.

We measure the distribution of DNaaS's grasp quality metric (the epsilon metric under uncertainty in object pose, gripper pose, and friction coefficients) for gripper models with different widths (see Figure 5). Intuitively, as the width of the gripper increases from 0 to the longest dimension of the target object's bounding box, we expect the distribution of the grasp quality metric to shift from a point mass at grasp quality 0 out to the right tail. The positive relationship between parallel-jaw width and grasp quality should hold up to the point when the width of the parallel-jaw gripper reaches the longest dimension of the target object's bounding box. Beyond that, any additional width of the parallel-jaws does not enable additional grasps that were previously infeasible.

Concretely, we can see these three phases in Figure 5 where we generate grasps on a spray bottle object using parallel-jaw grippers of different widths. On the far left we begin with a gripper too narrow to grasp the spray bottle robustly. As the width is increased from 2cm to 4cm, the parallel-jaw gripper begins to fit around parts of the bottle that allow for more robust grasps. The maximum width of the bottle is approximately 5cm, so although the increase in width from 4cm to 6cm does continue to shift the distribution of grasp quality out to the tail, it does not have the same magnitude of effect as the jump from 2cm to 4cm does.

B. Processing Time Analysis

We gather system timing measurements to evaluate the performance of DNaaS as a software system and benchmark



Mesh	Avg. Processing Time (s)	Std. Dev.	Avg. Num. Faces
Bar Clamp	49	16	14747
Endstop Holder	61	23	19368
Spray Bottle	81	30	17255

Fig. 7: Measuring the elapsed time of the major stages in the grasp generation pipeline, segmented by mesh. Timing measurements are again given for three adversarial object meshes: a bar clamp, an endstop holder, and a spray bottle.

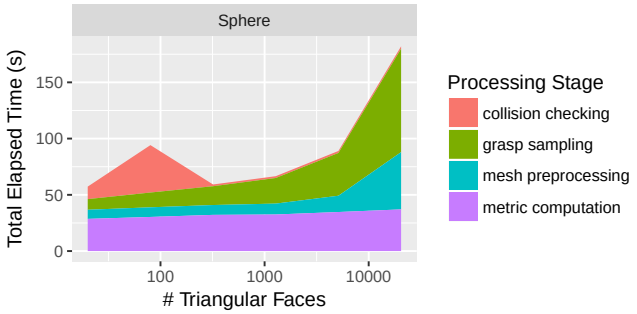


Fig. 8: Timing measurements for a spherical mesh, which has many equally likely stable poses. This leads to a peak in the time it takes to collision check around 100 triangular faces.

the speed of our gasping service. A single trial measures the time from the initial incoming request for grasps on a given mesh until DNaaS has completed all stages of the the grasp generation pipeline and is ready to return a set of candidate grasps to the user. When measuring the performance of DNaaS, we consider two criteria: overall timing and timing of the individual steps in the grasp generation pipeline. We conduct 25 trials across 4 objects whose triangular faces are subdivided to simulate increased mesh complexity. The resulting meshes ranged from 284 to 72704 triangular faces. We find that on average DNaaS takes under 75 seconds to process grasp requests on adversarial meshes using a parameterized gripper model.

The individual steps of the DNaaS grasp generation pipeline are dependent on both object geometry and mesh complexity (measured by the number of triangular faces in a mesh). Figure 7 depicts the difference in timing across object geometries by splitting out the distributions of processing time across three example meshes. Figure 6 shows the positive relationship between the number of triangular faces in the target object's mesh and grasp generation runtime.

Although there is an upfront cost in the initial grasp computation step, the grasps computed by DNaaS are cached on the server and can be queried quickly thereafter. This suggests that DNaaS could be a feasible solution to grasping scenarios where the target objects are known in advance so grasps can be pre-computed and cached for faster on-the-fly access.

C. Failure Modes

We present a series of failure modes encountered during the development of DNaaS in hopes of galvanizing users to further test the system's capabilities.

Perturbation During metric robustness computation, we perturb candidate grasps with random noise. For parallel-jaw grippers with small widths, the contact points of the grasp can be close to the surface of the object. When we attempt to perturb such grasps, sometimes the contact point is moved inside the object mesh, resulting in an infeasible grasp whose metric score should be zero. However, as collision checking each perturbation would take a prohibitive amount of time, these grasps receive non-zero scores. This means that even though this type of grasp is not robust, they can potentially receive relatively high metric scores.

Stable Poses During our performance testing, we noticed an anomalous result on a spherical adversarial object. For low face-count spherical meshes, one stable pose exists for every face. This results in a large number of symmetric stable poses. Due to collision checking being run for each stable pose, this causes low triangle count spheres to have significantly longer runtimes when compared to other meshes with a similar number of triangular faces. As the number of faces increases and the mesh more closely approximates a true sphere, the faces become smaller and the probabilities of the associated stable poses decrease. Once this probability becomes small enough, the poses associated with each face are no longer considered stable, and the object has zero stable poses. This means that once the spherical mesh has enough faces, its collision checking runtime drops instead of increasing. Figure 8 depicts this effect.

Uniform Density Center of Mass When computing the center of mass (COM) for an object mesh, we assume uniform density. This can lead to counter-intuitive grasps, for example the ones in Figure 5, where the most robust grasps are around the uniform-density COM. Humans may see the spray bottle and infer that there is likely liquid inside which leads to a lower COM and an entirely different set of robust grasps. Thus, DNaaS may counter-intuitively rank certain grasps as robust when given objects of non-uniform density.

VI. Conclusion and Future Work

We present DNaaS, a RAaaS architecture, which combines a public HTTP API and graphical web user interface to provide programmatic and GUI access to Dex-Net. We examine grasps generated by DNaaS across three parallel-jaw gripper hardware configurations and four adversarial objects. The distribution of grasp quality presented for the experimental hardware configurations allow us to make qualitative

observations about the space of grasp-enabling parallel-jaw grippers.

System timing measurements suggest that DNaaS could be used in industrial settings such as multi-item, single-line production lines where the up-front cost of grasp computation for objects can be pushed to the change-over times between items and subsequent computations can make use of DNaaS's grasp caching.

By making the Dex-Net algorithm widely accessible, we invite research and industrial users to experiment with the system, evaluate performance, and identify new failure modes. DNaaS hopes to motivate future data-driven grasping algorithms by including objects submitted to DNaaS in the growing collection of 10,000 unique 3D objects models and 2.5 million associated parallel-jaw grasps in the Dex-Net database. In future works we will extend DNaaS to DexNet 2.0 [42], which uses Grasp Quality Convolutional Neural Networks (GQ-CNNs) to predict successful grasps.

ACKNOWLEDGMENTS

This research was performed at the AUTOLAB at UC Berkeley in affiliation with the Berkeley AI Research (BAIR) Lab, the Real-Time Intelligent Secure Execution (RISE) Lab, and the CITRIS "People and Robots" (CPAR) Initiative. The authors were supported in part by the U.S. National Science Foundation under NRI Award IIS-1227536: Multilateral Manipulation by Human-Robot Collaborative Systems, the Department of Defense (DoD) through the National Defense Science & Engineering Graduate Fellowship (NDSEG) Program, the Berkeley Deep Drive (BDD) Program, and by donations from Siemens, Google, Cisco, Autodesk, IBM, Amazon Robotics, and Toyota Robotics Institute. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Sponsors. We thank our colleagues who provided helpful feedback, code, and suggestions, in particular Roy Fox, David Gealy, Sanjay Krishnan, Animesh Garg, Michael Laskey, Matt Matl, and Vishal Satish.

REFERENCES

[1] "Apache web server." [Online]. Available: https://httpd.apache.org/
[2] "Asynchronous operations in javascript using promises." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise
[3] "Autonomous solutions (asi)." [Online]. Available: https://www.asirobots.com/research/
[4] "Bootstrap front-end component library." [Online]. Available: http://getbootstrap.com/
[5] "Dnaas api documentation." [Online]. Available: https://gist.github.com/bderose/61d70384d159af3715865b35b75e96e0
[6] "Dnaas api example." [Online]. Available: https://gist.github.com/bderose/0c1c2cad2d4291b16a9a24d4b271da1f
[7] "Document object model." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction
[8] "Dropblet home automation." [Online]. Available: http://smartdroplet.com/
[9] "Emerald cloud labs." [Online]. Available: https://www.emeraldcloudlab.com/about/
[10] "Flexbox." [Online]. Available: https://www.w3.org/TR/css-flexbox-1/
[11] "graspit - ros wiki." [Online]. Available: http://wiki.ros.org/graspit
[12] "Jquery javascript library." [Online]. Available: https://jquery.com/
[13] "Jquery ui library." [Online]. Available: https://jqueryui.com/
[14] "Pyhull." [Online]. Available: http://pythonhosted.org/pyhull/
[15] "Tempo automation." [Online]. Available: https://www.tempoautomation.com/capabilities
[16] "three.js." [Online]. Available: https://threejs.org/
[17] "Trimesh." [Online]. Available: https://github.com/mikedh/trimesh
[18] R. Arumugam, V. R. Enti, L. Bingbing, W. Xiaojun, K. Baskaran, F. F. Kong, A. S. Kumar, K. D. Meng, and G. W. Kit, "Davinci: A cloud computing framework for service robots," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 3084–3089.
[19] R. Balasubramanian, L. Xu, P. D. Brook, J. R. Smith, and Y. Matsuoka, "Physical human interactive guidance: Identifying grasping principles from human-planned grasps," *IEEE Trans. Robotics*, vol. 28, no. 4, pp. 899–910, 2012.
[20] J. Bohg, A. Morales, T. Asfour, and D. Kragic, "Data-driven grasp synthesisa survey," *IEEE Trans. Robotics*, vol. 30, no. 2, pp. 289–309, 2014.
[21] P. Brook, M. Ciocarlie, and K. Hsiao, "Collaborative grasp planning with multiple object representations," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*. IEEE, 2011, pp. 2851–2858.
[22] 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.
[23] R. Detry, C. H. Ek, M. Madry, and D. Kragic, "Learning a dictionary of prototypical grasp-predicting parts from grasping experience," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*. IEEE, 2013, pp. 601–608.
[24] K. Fang, Y. Bai, S. Hinterstoisser, and M. Kalakrishnan, "Multi-task domain adaptation for deep learning of instance grasping from simulation," *CoRR*, vol. abs/1710.06422, 2017. [Online]. Available: http://arxiv.org/abs/1710.06422
[25] C. Ferrari and J. Canny, "Planning optimal grasps," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 1992, pp. 2290–2295.
[26] K. Goldberg and B. Kehoe, "Cloud robotics and automation: A survey of related work," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2013-5*, 2013.
[27] K. Goldberg, B. V. Mirtich, Y. Zhuang, J. Craig, B. R. Carlisle, and J. Canny, "Part pose statistics: Estimators and experiments," *IEEE Trans. Robotics and Automation*, vol. 15, no. 5, pp. 849–857, 1999.
[28] C. Goldfeder and P. K. Allen, "Data-driven grasping," *Autonomous Robots*, vol. 31, no. 1, pp. 1–20, 2011.
[29] C. Goldfeder, M. Ciocarlie, H. Dang, and P. K. Allen, "The columbia grasp database," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*. IEEE, 2009, pp. 1710–1716.
[30] E. Guizzo, "Cloud robotics: Connected to the cloud, robots get smarter," Jan 2011. [Online]. Available: https://spectrum.ieee.org/automaton/robotics/robotics-software/cloud-robotics
[31] 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," *arXiv preprint arXiv:1610.05514*, 2016.
[32] A. Herzog, P. Pastor, M. Kalakrishnan, L. Righetti, J. Bohg, T. Asfour, and S. Schaal, "Learning of grasp selection based on shape-templates," *Autonomous Robots*, vol. 36, no. 1-2, pp. 51–65, 2014.
[33] 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 *Proc. IEEE Int. Conf. on Computer Vision (ICCV)*. IEEE, 2011, pp. 858–865.
[34] D. Kappler, J. Bohg, and S. Schaal, "Leveraging big data for grasp planning," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2015.
[35] B. Kehoe, A. Matsukawa, S. Candido, J. Kuffner, and K. Goldberg, "Cloud-based robot grasping with the google object recognition engine," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*. IEEE, 2013, pp. 4263–4270.
[36] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Transactions on automation science and engineering*, vol. 12, no. 2, pp. 398–409, 2015.
[37] J. Kuffner, "Cloud-enabled robots," in *IEEE-RAW International Conference on Humanoid Robots*, 2010.

[38] S. Kumra and C. Kanan, "Robotic grasp detection using deep convolutional neural networks," *CoRR*, vol. abs/1611.08036, 2016. [Online]. Available: http://arxiv.org/abs/1611.08036

[39] M. Laskey, J. Mahler, Z. McCarthy, F. T. Pokorny, S. Patil, J. van den Berg, D. Kragic, P. Abbeel, and K. Goldberg, "Multi-armed bandit models for 2d grasp planning with uncertainty." in *Proc. IEEE Conf. on Automation Science and Engineering (CASE)*. IEEE, 2015.

[40] I. Lenz, H. Lee, and A. Saxena, "Deep learning for detecting robotic grasps," *Int. Journal of Robotics Research (IJRR)*, vol. 34, no. 4-5, pp. 705–724, 2015.

[41] 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 *Proc. IEEE Int. Conf. on Simulation, Modeling, and Programming of Autonomous Robots (SIMPAR)*. Springer, 2010, pp. 109–120.

[42] 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," in *Proc. Robotics: Science and Systems (RSS)*, 2017.

[43] J. Mahler, F. T. Pokorny, B. Hou, M. Roderick, M. Laskey, M. Aubry, K. Kohlhoff, T. Kröger, J. Kuffner, and K. Goldberg, "Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*. IEEE, 2016.

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

[45] L. Pinto and A. Gupta, "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2016.

[46] F. T. Pokorny and D. Kragic, "Classical grasp quality evaluation: New algorithms and theory," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*. IEEE, 2013, pp. 3493–3500.

[47] D. Prattichizzo and J. C. Trinkle, "Grasping," in *Springer handbook of robotics*. Springer, 2008, pp. 671–700.

[48] J. Redmon and A. Angelova, "Real-time grasp detection using convolutional neural networks," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*. IEEE, 2015, pp. 1316–1322.

[49] A. Rodriguez, M. T. Mason, and S. Ferry, "From caging to grasping," *Int. Journal of Robotics Research (IJRR)*, p. 0278364912442972, 2012.

[50] D. Seita, F. T. Pokorny, J. Mahler, D. Kragic, M. Franklin, J. Canny, and K. Goldberg, "Large-scale supervised learning of the grasp robustness of surface patch pairs," in *Proc. IEEE Int. Conf. on Simulation, Modeling, and Programming of Autonomous Robots (SIMPAR)*. IEEE, 2016.

[51] N. Tian, M. Matl, J. Mahler, Y. X. Zhou, S. Staszak, C. Correa, S. Zheng, Q. Li, R. Zhang, and K. Goldberg, "A cloud robot system using the dexterity network and berkeley robotics and automation as a service (brass)," in *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017*, 2017, pp. 1615–1622. [Online]. Available: https://doi.org/10.1109/ICRA.2017.7989192

[52] E. W. Weisstein. Triangle point picking. From MathWorld–A Wolfram Web Resource. Last visited 3/14/18. [Online]. Available: http://mathworld.wolfram.com/TrianglePointPicking.html

[53] J. Weisz and P. K. Allen, "Pose error robust grasping from contact wrench space metrics," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*. IEEE, 2012, pp. 557–562.