Robotic gripper design with Evolutionary Strategies and Graph Element Networks

Ferran Alet[*] **Maria Bauza**[*] **Adarsh K. Jeewajee**[*] **Max Thomsen**[*]

Alberto Rodriguez **Leslie Pack Kaelbling**

Tomás Lozano-Pérez
MIT
`{alet,bauza,jaks19,mthomsen,albertor,lpk,tlp}mit.edu`

Abstract

Robots are increasingly pervasive in manufacturing. However, most robotic grippers are still simple parallel-jaw grippers with flat fingers, which for many objects leads to sub-optimal manipulations. One way to solve this problem relies on having engineers design a new gripper for every object, however, this is expensive and inefficient. We instead propose to automatically design them using machine learning. First, we use evolutionary strategies in simulation to get a good initial gripper. We pair this approach with an automatic curriculum method that increases the difficulty of the manipulation task to ease the design process. Once the gripper is designed in simulation we propose to fine-tune it via back-propagation on a Graph Neural Network model trained on real data for many grippers and objects. By sharing real-world data across grippers and objects we can be more data-efficient in the real world. We show that our method improves the default flat gripper by significant margins on multiple datasets of varied objects.

1 Introduction

Many robotic applications in the industry involve picking and placing objects at fast speeds and with high reliability. However, in contrast to human hands, which can adjust to every object, most robots only have parallel-jaw grippers with a single degree of actuation. This highlights the importance of customizing the finger morphology to account for the properties of the manipulated object such as size, shape, weight, and texture. However, currently, gripper design is mostly manual: when a factory has to accommodate a new piece into the production pipeline, someone is tasked to design a custom gripper (which is slow and expensive). Otherwise, the automated solution is likely to use a sub-optimal default gripper, which hinders the efficiency and the flexibility of the manufacturing pipeline.

In this work, we propose to automate the design of new grippers by combining evolutionary search in simulation with meta-learned adaptation from small amounts of real data. Given an object or a collection of objects, we can design a gripper to increase the probability of correctly performing a given primitive. We exemplify that when the task considered is robust picking.

In particular, our contributions are:

1. <u>We use Evolutionary Strategies in simulation to optimize grippers in a wide variety of scenarios</u>. By using a simple model-free method, our approach can apply to many different objects and primitives.

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.
[*] these authors contributed equally; sorted alphabetically.

2. We leverage curriculum learning techniques to design grippers for hard tasks on which the original gripper fails or is far from solving the final task. By slowly increasing the difficulty of the task, we can get enough signal to enable Evolutionary Strategies to make progress.
3. We achieve data-efficiency in the real world by fine-tuning grippers on gradients coming from a meta-learned Graph Element Network. This approach allows us to leverage all the data generated in simulation while remaining data-efficient in the real world.

2 Related work

There have been multiple works on shape optimization for contact in the robotics community, such as optimizing the fence of a conveyor belt to reorient objects [1] or end-effector shapes for 1 degree-of-freedom actuators [2, 3]. In these works, authors assume that motions are quasi-static, and reduce the problem to solving a differential equation. There have also been analytical approaches to gear design [4], use optimization to solve trajectory planning through contact [5], and designing trajectory and shape in tandem for 1-D throwers [6]. In contrast to these theoretical works, our approach does not rely on an explicit model of the world, allowing us to design grippers in more varied and complex situations.

There have also been computer-based morphology designs back from the pioneering work of Karl Sims [7], who efficiently evolved virtual creatures to swim, jump, run or compete from a cube. Chenney et al. [8] later optimized soft robots made from different materials to move them in simulated environments. More recently, [9] used reinforcement learning to co-optimize policy and morphology in creatures moving in simulated environments. Pathak et al. also used graph neural networks and reinforcement learning to train a collection of self-assembling robots [10] as well as a policy that worked well across many morphologies [11]. In contrast to these works, whose creatures are relatively low resolution, we optimize finger meshes where both the design and the task are very detail-sensitive. Moreover, we care about bringing the designs to the real world, which forces our approach to be data-efficient.

Recently, machine learning has also been used for design in robotics. Liao et al. [12] use Batched Bayesian Optimization to design micro-robots; however, the approach heavily restricts the number of variables they can optimize, describing the morphology with only 3 variables. Closer to our work, Wang et al. [13] designed a set of objects that are extremely hard to grasp for regular grippers. Their approach relies on combining an analytical model with GANs [14] to generate objects that are close to a specific given object while being much harder to grasp. Their motivation is related to ours, but can only tackle grasping and not arbitrary primitives. However, while they only focus on a single object for a single gripper, our method also tackles the case where a gripper has to work with a collection of objects and primitives.

Graph Neural Networks [15, 16, 17] have been already applied to design problems, most notably in molecule design [18, 19], with edges describing chemical bonds. Closer to our work, GNNs have also been applied to designing the structure of buildings [20], with edges defined by beams and columns. In contrast to both works, we optimize meshes and leverage a meta-learning approach to be data-efficient in the real world (a problem that neither method addresses).

Finally, our goal is similar to the concurrent work from Ha et al. [21] that uses a generative network that, given the mesh of the object to grasp, produces a gripper. In contrast, we use evolutionary strategies, which is computationally slower, but can easily optimize a single gripper for multiple objects. In [21], they also use an evaluation network for the grasps, similar to our use of GENs. However, they only use it at training-time in simulation to propagate gradients back to the generative network. We instead use it to fine-tune our gripper to the real-world dynamics.

3 Method

3.1 Evolutionary Strategies

We start with the default parallel-jaw gripper with planar fingers used in most mid-size robots. We then collect multiple datasets of objects and resize them appropriately so that they have at least one direction that fits within the gripper. This step allows us to have a diverse dataset of objects (including even chairs) without having to worry about their original graspability. Next, we simulate grasps on the pybullet engine [22]. For each grasp, we first place the object at a random orientation (conditioned

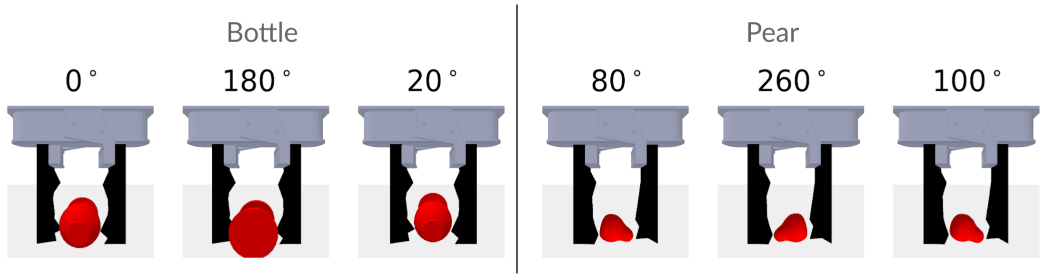

Figure 1: Example of the design setup in simulation. We simulate different grippers grasping a diverse set of object objects at multiple angles. We record whether each gripper manages to perform the grasp and whether the object remains stable after the gripper moves up and shakes.

on fitting inside the gripper), move down the gripper vertically, close its fingers, move up enough to lift the object completely from the floor, and shake the gripper; all in an open-loop sequence. For each grasp, we can record at any point in time if the grasp is succeeding by checking whether the gripper is opened or closed.

Given the simulated data, we run CMA-ES [23] to optimize the grippers. First, we take a series of points representing the original mesh and repeatedly add noise to the current mesh and obtain a new mesh by taking a weighted mean of the randomized grippers, weighted according to their simulated success metric. In our case, we compute success by adding whether the gripper managed to grasp the object, and whether the grasp was stable under shaking. However, we note that our approach is not limited to grasping, and can be applied to other manipulation primitives as long as their success can be automatically assessed on the simulator.

3.2 Automated curriculum design

In machine learning, it has often been observed that, when learning a very complex task, it helps to create a curriculum for the learner [24]. Most relevant to our case, curriculum learning has been used with domain randomization for learning to manipulate a Rubik's cube [25], by slowly increasing the amount of randomization faced by the policy. In our case, we can apply the curriculum to our design problem by taking into account that if a gripper is far from solving the primitive, there will be no local improvement that can guide CMA-ES towards a better gripper.

Therefore, we propose to parametrize the manipulation primitive and create an automated curriculum that goes from simple tasks to the desired one. For now, we assume there is a single parameter d that describes the difficulty of the task and goes from 0 to D. We also assume that performance on the task, for a fixed gripper, is monotonically decreasing with d. In our case d parametrizes the speed of the gripper shake so that 0 implies no shaking and higher d implies higher shaking speed, making it harder for the grasp to remain stable.

Typically, automatic curricula aim at increasing the difficulty of the task while maintaining a fixed level of success. However, this does not align with our objective: we want to maximize success for a given task difficulty, not difficulty given a fixed success. However, we do not know how often we will solve a task at the desired difficulty, d, i.e. we do not know what is the probability of solving a given task. This can prevent us from performing an effective curriculum because if we set the target probability too low we will quickly reach the target task difficulty D but at an underwhelming performance. Instead, if we set the task success probability too high, we might get stuck training at a different difficulty $d < D$ than the desired.

A simple, yet effective, approach to mitigate this issue is to start with a target success probability of $p = 0$ and have a simple controller that regulates d to keep p constant. While the gripper gets optimized, this controller will eventually lead to $d = D$. To keep improving after reaching $d = D$, we increase the target success p, making the controller automatically decrease d to ensure that the new target probability is satisfied. As a result, we keep arriving at the target difficulty D with an increasing amount of success until, at some point, we cannot increase it further.

3.3 Graph Element Networks for data-efficient design

While simple, CMA-ES has the problem of being hopelessly data-inefficient. Before obtaining a useful parameter update, it needs to try many gripper variations. However, in practice, we would

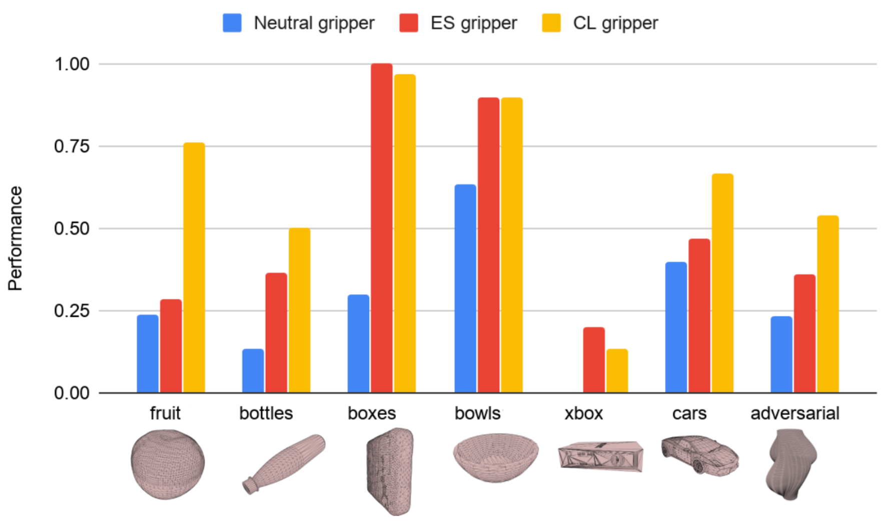

Figure 2: Success metric, accounting for both grasping and shake stability. We evaluate the neutral gripper, the ES gripper, and the curriculum learning gripper on multiple datasets. We can see that ES and CL perform significantly better than the neutral gripper for all tasks. Curriculum learning often leads to big gains, although on some occasions it performs slightly worse than not using it. Adversarial objects were designed in [13].

like to 3-D print grippers and have them be effective in the real world. We thus need a data-efficient approach both in terms of data collection and the number of different grippers created.

We leverage Graph Element Networks [26](GENs), which take in any spatial function described with a set of sampled points in the 3D space, and return another function (in our case, a single scalar). GENs work by first creating a spatial mesh around the region of interest, and mapping the input points to the mesh via attention. Once the input points are embedded in the mesh, we treat the mesh as a regular graph neural network and perform message-passing propagation [27]. After message passing, we can query any point in the space by again using attention mechanisms to turn it into a few queries into the mesh. GENs have been used to model Partial Differential Equations and, more relevant to our setting, predicting the result of a robot pushing a planar object.

Also relevant to our setting, we showed that we can fine-tune the meshes with few data and increase the accuracy of the predictions made by the GENs, by focusing the mesh computation where it is most needed. This affects the node positions without modifying the GEN weights, which are the bulk of the parameters and the computation. This adaptation can be done because GENs are end-to-end differentiable and thus we can train the position of the mesh nodes via back-propagation. Given a grasp, the input spatial functions are the point-clouds of the gripper and object (appropriately rotated). We then train the GEN to predict the result of the grasp (and its success after shaking). In practice, we train the GEN using many (gripper, object) pairs, with grippers coming from those found by CMA-ES. As a result, the GEN provides a differentiable approximation to our simulator. This allows us to optimize the gripper mesh by maximizing the probability of success predicted by the GEN w.r.t. the input gripper mesh.

For an unseen collection of objects and manipulation tasks in the real world, we would like to use back-propagation through GENs since it is very data-efficient. However, back-propagating a model w.r.t. its input typically leads to overly optimistic results, a phenomenon well observed in model-based RL and similar to that of adversarial examples [28]. To avoid relying too much on the back-propagation, we can perform most of the gripper optimization in simulation using CMA-ES. Then, to bridge the reality gap between simulation and real-world, we fine-tune both the GEN and the input-mesh on a small amount of real data. Note that the GEN itself can pool real data from many different real grippers and objects. This allows us to design a better version of the grippers efficiently, as GENs have learned a good model from prior tasks; analogous to the meta-learning setting [29, 30, 31, 32].

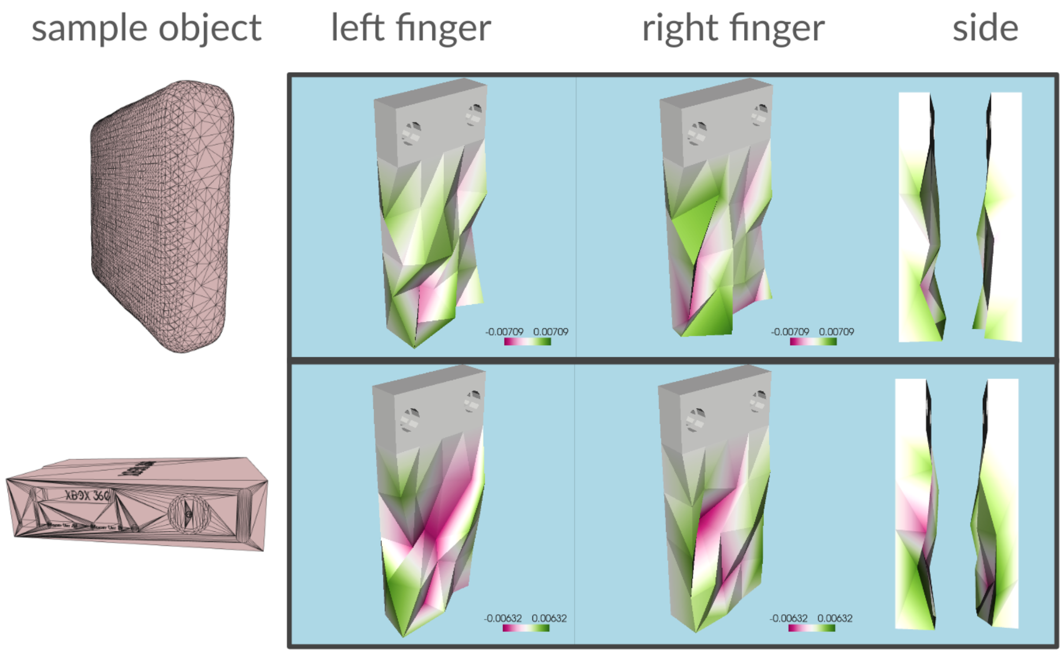

Figure 3: Comparison of designed grippers with default grippers. Designed grippers tend to have rugosity and external protuberances on the bottom to help funnel and lift the objects. There are also differences between grippers trained on different sets of objects, as they adapt to the characteristics of each set.

4 Experiments

Our current experiments involve CMA-ES in simulation, as well as training GENs on the simulated data coming from many different grippers and objects. We plan to 3D print and test these grippers in the real world, once the coronavirus situation allows it.

Figure 2 shows how we can reliably improve the success of the neutral gripper across all datasets. Notably, Xboxes perform very poorly and boxes perform very well; this is mainly due to boxes being in a vertical configuration and Xboxes being flat on the floor. Figure 3 shows the grippers for both boxes and Xboxes, and their difference w.r.t. to the original gripper.

Curriculum learning also helps in almost all circumstances, with a 200% gain on the fruits dataset. For two of the datasets, the success decreases. We believe this happens because learnability and success are related, but not identical. Sometimes it can be easier to learn a gripper in a hard, but discriminative task, than an easier but less-discriminative one. We do observe, however, that the decreases are very minor and curriculum design is useful overall.

We also successfully trained a Graph Element Network on data coming from CMA-ES, predicting grasp success with 72% accuracy for new grippers and 90% for known grippers and new objects; while training on 300 different bottles and more than 50 grippers simultaneously. These results make us optimistic that we will be able to leverage the GENs to fine-tune grippers in the real world.

5 Conclusion

We have shown a method for automatically designing new grippers combining ideas from evolutionary strategies, curriculum learning, and graph neural networks. This allows us to generate promising grippers in simulation and fine-tune them in the real world in a data-efficient manner. Our method has direct applications in the industry, where robots are increasingly pervasive, but deploying them still requires a lot of effort. By automating part of the design process, we can lower the cost of setting a new robot chain, making manufactured products cheaper.

References

[1] M Brokowski, M Peshkin, and Kenneth Goldberg. Optimal curved fences for part alignment on a belt. 1995.

[2] Alberto Rodriguez and Matthew T Mason. Grasp invariance. *The International Journal of Robotics Research*, 31(2):236–248, 2012.

[3] Alberto Rodriguez and Matthew T Mason. Effector form design for 1dof planar actuation. In *2013 IEEE International Conference on Robotics and Automation*, pages 349–356. IEEE, 2013.

[4] Jen-Yu Liu and Yen-Chuan Chen. A design for the pitch curve of noncircular gears with function generation. In *Proceedings of the international multiconference of engineers and computer scientists*, volume 2, 2008.

[5] Michael Posa, Cecilia Cantu, and Russ Tedrake. A direct method for trajectory optimization of rigid bodies through contact. *The International Journal of Robotics Research*, 33(1):69–81, 2014.

[6] Orion Taylor and Alberto Rodriguez. Optimal shape and motion planning for dynamic planar manipulation. *Autonomous Robots*, 43(2):327–344, 2019.

[7] Karl Sims. Evolving virtual creatures. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 15–22. ACM, 1994.

[8] Nick Cheney, Robert MacCurdy, Jeff Clune, and Hod Lipson. Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 167–174, 2013.

[9] David Ha. Reinforcement learning for improving agent design. *Artificial life*, 25(4):352–365, 2019.

[10] Deepak Pathak, Christopher Lu, Trevor Darrell, Phillip Isola, and Alexei A Efros. Learning to control self-assembling morphologies: a study of generalization via modularity. In *Advances in Neural Information Processing Systems*, pages 2295–2305, 2019.

[11] Wenlong Huang, Igor Mordatch, and Deepak Pathak. One policy to control them all: Shared modular policies for agent-agnostic control. *arXiv preprint arXiv:2007.04976*, 2020.

[12] Thomas Liao, Grant Wang, Brian Yang, Rene Lee, Kristofer Pister, Sergey Levine, and Roberto Calandra. Data-efficient learning of morphology and controller for a microrobot. *arXiv preprint arXiv:1905.01334*, 2019.

[13] David Wang, David Tseng, Pusong Li, Yiding Jiang, Menglong Guo, Michael Danielczuk, Jeffrey Mahler, Jeffrey Ichnowski, and Ken Goldberg. Adversarial grasp objects.

[14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[15] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[16] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[17] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[18] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. *arXiv preprint arXiv:1802.04364*, 2018.

[19] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Hierarchical generation of molecular graphs using structural motifs. *arXiv preprint arXiv:2002.03230*, 2020.

[20] Kai-Hung Chang and Chin-Yi Cheng. Learning to simulate and design for structural engineering. *arXiv preprint arXiv:2003.09103*, 2020.

[21] Huy Ha, Shubham Agrawal, and Shuran Song. Fit2Form: 3D generative model for robot gripper form design. In *Conference on Robotic Learning (CoRL)*, 2020.

[22] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. 2016.

[23] Nikolaus Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[24] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[25] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[26] Ferran Alet, Adarsh Keshav Jeewajee, Maria Bauza Villalonga, Alberto Rodriguez, Tomas Lozano-Perez, and Leslie Kaelbling. Graph element networks: adaptive, structured computation and memory. In *International Conference on Machine Learning*, pages 212–222, 2019.

[27] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017.

[28] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[29] Jürgen Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

[30] Samy Bengio, Yoshua Bengio, and Jocelyn Cloutier. On the search for new learning rules for anns. *Neural Processing Letters*, 2(4):26–30, 1995.

[31] Sebastian Thrun and Lorien Pratt. *Learning to learn*. Springer Science & Business Media, 1998.

[32] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.