Gradient-Based Shape Optimization for Engineering Using Machine Learning

Xingyuan Sun

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
by the Department of
Computer Science
Adviser: Szymon Rusinkiewicz and Ryan P. Adams

March 2023

Abstract

Shape design problems are important in engineering, *e.g.*, trajectory planning for robot arms, material distribution optimization, *etc*. However, existing works usually solve these tasks without the help of gradients, whose efficiency can be limited. We formalize design problems as constrained optimization tasks and propose to use gradient-based optimizers with automatic differentiation to solve them. Specifically, we use the adjoint method when the underlying physical process can be characterized by PDEs. In Chapter 2, we solve for extruder paths of 3D printing that can compensate for the deformation caused by the fiber printing process. As the printing process is complex and difficult to model, we create a synthetic dataset and fit it using a neural network to get a differentiable surrogate of the printing simulator. We further speed up the optimization process by using a neural network to amortize it, sacrificing a bit of accuracy but getting much faster, real-time inferences. In Chapter 3, we study the task of fiber path planning, figuring out where to lay reinforcing fibers in plastic for 3D printing, maximizing the stiffness of the composite. We build a simulator by solving the linear elastic equations and use the adjoint method for gradient calculation and BFGS for fiber path optimization. In Chapter 4, we investigate the problem of dovetail joint shape optimization for stiffness. To model the contact between two parts of a joint, we build a simulator by alternatively solving one side of the joint while fixing the other side. We use the adjoint method for gradient computation and gradient descent for optimization. All methods across the projects are tested both in simulation and real-world experiments, showing our approach produces high-quality designs, and also the amortized approach provides real-time inference while achieving a comparable design quality.

Acknowledgements

First and foremost, I would like to thank both of my advisors, Szymon Rusinkiewicz and Ryan P. Adams, and it is my greatest honor to be advised by them throughout my PhD study. Ryan and Szymon taught me how to do research—how to solve problems in a scientific approach: formalizing problems, reading existing works, designing methods, conducting experiments, communicating results with the community, *etc*. I believe the methods and principles I learned from them will guide me through my whole life.

I would also like to thank my thesis committee members: Olga Russakovsky, Sigrid Adriaenssens, and Felix Heide. Their feedback and comments are always inspiring, and this thesis cannot be done without their help.

I am very grateful to work with my excellent collaborators on the research projects presented in this thesis: Chenyue Cai, Tianju Xue, and Geoffrey Roeder. Their discussion, insights, and contributions are essential to this thesis.

I appreciate the summers spent outside of Princeton at Google Research and Citadel Securities. The experiences there are enriching and provided me with new knowledge and perspectives.

Many thanks to other amazing collaborators during my PhD study: Runzhe Yang, Jimmy Wu, Sulin Liu, Andy Zeng, Shuran Song, Karthik Narasimhan, and Thomas Funkhouser. Our joint research is an important part of my PhD study, which provided me with new perspectives and always inspired me.

I am very thankful to Joseph Vocaturo and the Department of Civil and Environmental Engineering for providing the universal testing machine and the testing environment. I sincerely appreciate Eder Medina's help in setting up the resin printer. I greatly appreciate the help I received from Nicki Mahler and Louis Riehl, the graduate coordinators from the Computer Science Department, and Barbara Varga and Pamela DelOrefice for their help on various reimbursement requests.

I am fortunate to spend time with all other amazing labmates in Szymon's and Ryan's lab: Yuanqiao Lin, Vivien Nguyen, Christina Shatford, Fangyin Wei, Qiang Zhang, Maxine Perroni-Scharf, Xinyi Fan, Linguang Zhang, Joshua Aduol, Sam Barnett, Diana Cai, Alex Guerra, Mehran Mirramezani, Deniz Oktay, Nicholas Richardson, Olga Solodova, Kathryn Wantlin, Jenny Ni Zhan, Jordan Ash, Alex Beatson, Jad Rahme, Ari Seff, Jose Garrido Torres, and Yaniv Ovadia. Although I have not directly collaborated with them, their presentations during group meetings always brought new ideas to me, showing me exciting approaches to solving problems.

I would like to extend my thanks to the collaborators and advisors during my undergraduate research: Jiajun Wu, Xiuming Zhang, Zhoutong Zhang, Chengkai Zhang, Shaoxiong Wang, Wenzhen Yuan, Yifan Wang, Tianfan Xue, Yonglong Tian, Andrew Luo, Hongtao Lu, Yong Yu, William T. Freeman, and Joshua B. Tenenbaum. Without their guidance and research experience with them, I will not be able to start my PhD study.

I am deeply grateful to my roommates and friends at Princeton, not mentioned above: Zeyu Xiong, Wei Zhan, Ze Han, Dingli Yu, Genyue Liu, Sinong Geng, Zhuqi Li, Qinghua Liu, Zhenyu Song, Yu Wu, Ziyang Xu, Yuling Yan, Shunyu Yao, Ethan Tseng, Zexuan Zhong, and Shaowei Zhu. I sincerely appreciate everyone who ever helped me during my PhD study.

Lastly, I would like to thank my parents for their unconditioned love.

To my parents.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Manufacturing is a significant part of the secondary sector of the economy, and it is related to a great variety of industries, including aerospace, automobile, furniture, *etc*. Design is one of the most important parts of manufacturing —*e.g.*, distributing material properly to reduce material usage while achieving the required strength. In this dissertation, we formalize design tasks as a type of optimization problem. Design refers to the parameters that we have control over and what we are optimizing. Given the design parameters, a known or unknown underlying physical process happens, which produces a resulting physical realization. A design goal is given, which describes desired properties of the physical realization. An objective function is then evaluated using the design parameters, the physical realization, and the design goal, and we would like to minimize the value of this function.

Existing research works solve design problems mostly using three types of approaches: optimization (*e.g.*, evolutionary algorithms (EA), constrained optimization algorithms), sampling (*e.g.*, simulated annealing (SA)), and search (*e.g.*, A* search). However, most existing works do not calculate and use the gradient of the objective function, which is essential for gradient-based optimizers. With the development of automatic differentiation, gradient calculation has become much easier and more

efficient. We, therefore, propose to solve design problems using automatic differentiation and gradient-based optimizers (*e.g.*, gradient descent, BFGS). If the physical realization process can be characterized as PDEs, we use the adjoint method for gradient calculation; otherwise, we can generate synthetic datasets and fit them using neural networks, which provides us with a differentiable surrogate of the simulator. In this dissertation, three design problems related to "shape" are studied: 3D printer extruder path planning, 3D printer fiber path planning, and dovetail joint shape optimization.

3D printing is powerful in rapid prototyping, aerospace, *etc*. To increase the strength of polymers, adding continuous fibers is a popular option. However, fibers are stiff, which deform during the printing process and results in fiber paths that are different from the extruder paths. In Chapter 2, we study extruder path planning, aiming at finding an extruder path that can compensate for the deformation caused by the printing process for any desired fiber path, and Chapter 2 is based on a joint work [Sun et al., 2021] with Tianju Xue, Szymon Rusinkiewicz, and Ryan P. Adams. As the printing process is complex and difficult to be modeled directly, we create a simulator of the fiber printing process and use it to generate a synthetic dataset. We build a differentiable surrogate of the simulator by fitting a neural network to the dataset. To further speed up the optimization process, we propose to train an "encoder" (consider the surrogate as a decoder) that directly produces a design from the goal, which amortizes the optimization process. We test the amortized approach as well as the optimization approach both synthetically and in real tests, demonstrating that the optimization approach can produce extruder paths that result in high-quality fiber paths, and the amortized approach generates extruder paths of comparable qualities and is much faster, providing real-time inferences.

Another task in continuous fiber 3D printing is fiber path planning, *i.e.*, given a shape and some external loads, figuring out where to lay fibers, and we study this

task in Chapter 3. Chapter 3 is based on a joint work [Sun et al., 2022] with Geoffrey Roeder, Tianju Xue, Ryan P. Adams, and Szymon Rusinkiewicz. Given a shape, loads, and a fiber layout, we create a simulator by solving the linear elastic equations, and the task becomes a PDE-constrained optimization task. We use the adjoint method and automatic differentiation to compute gradients and optimize fiber paths using the BFGS optimizer. We test our algorithm as well as commercial software, Eiger, both in simulation and in real experiments, which proves our method provides fiber paths that are shorter and the corresponding composites are stiffer.

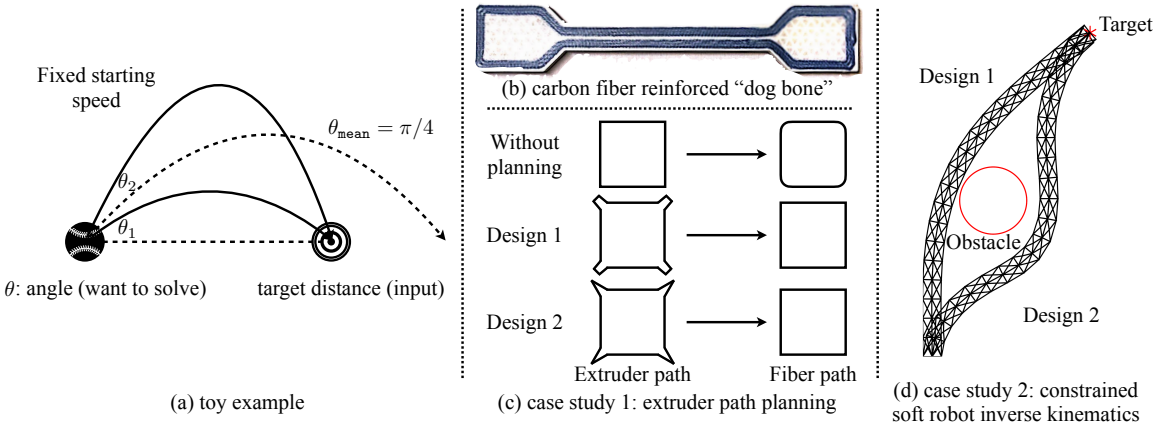
In manufacturing, objects are manufactured in several parts for reasons including machine size limit, easier to transport, *etc*. Joints are commonly used structures to connect parts, and the dovetail joint is one of them. In Chapter 4, we study the task of dovetail joint shape optimization, and Chapter 4 is based on a joint work with Chenyue Cai, Ryan P. Adams, and Szymon Rusinkiewicz. To effectively apply our approach, we cannot simply use existing simulators for contact simulation as they are usually not differentiable. We thus build our own simulator by using the penalty approach, *i.e.*, solve one side of the joint while considering the other side as rigid. We alternatively solve on the two sides for the solution to converge. Similarly, we test our approach both synthetically and in real experiments, proving the optimized joint shape from our algorithm is stiffer.

In summary, this dissertation studies the use of automatic differentiation and the adjoint method on shape design tasks, solving them using gradient-based optimizers, as well as an extension that amortizes the optimization process that sacrifices a bit of accuracy but provides much faster real-time inferences. All methods are tested in both simulation and real-world experiments.

Chapter 2

Amortized Synthesis of Constrained Configurations Using a Differentiable Surrogate

In design, fabrication, and control problems, we are often faced with the task of *synthesis*, in which we must generate an object or configuration that satisfies a set of constraints while maximizing one or more objective functions. The synthesis problem is typically characterized by a physical process in which many different realizations may achieve the goal. This many-to-one map presents challenges to the supervised learning of feed-forward synthesis, as the set of viable designs may have a complex structure. In addition, the non-differentiable nature of many physical simulations prevents efficient direct optimization. We address both of these problems with a two-stage neural network architecture that we may consider to be an autoencoder. We first learn the decoder: a differentiable surrogate that approximates the many-to-one physical realization process. We then learn the encoder, which maps from goal to design, while using the fixed decoder to evaluate the quality of the realization. We evaluate the approach on two case studies: extruder path planning in additive manufacturing



(a) toy example (c) case study 1: extruder path planning (d) case study 2: constrained soft robot inverse kinematics

Figure 2.1: (a) For a fixed and large-enough starting speed, there exist exactly two angles such that the ball will hit the target, where the mean of these two angles is $\pi/4$. (b) Some 3D printers utilize fibers to reinforce the thermoplastic print. (c) For such printers, fiber is laid out along an extruder path but deforms into a smoothed version due to the fiber's high stiffness and low stretch. Our goal is to generate extruder paths that compensate for the smoothing, but multiple extruder paths can result in the same target shape, such as a square. (d) In soft robot inverse kinematics, we control the stretch ratios of both the left- and right-hand sides of a snake-like robot. Our goal is to reach the target while avoiding an obstacle but, as is illustrated, the solution is not unique – two different designs are shown.

and constrained soft robot inverse kinematics. We compare our approach to direct optimization of the design using the learned surrogate, and to supervised learning of the synthesis problem. We find that our approach produces higher quality solutions than supervised learning, while being competitive in quality with direct optimization, at a greatly reduced computational cost.

2.1 Introduction

One of the ambitions of artificial intelligence is to automate problems in design, fabrication, and control that demand efficient and accurate interfaces between machine learning algorithms and physical systems. Whether it is optimizing the topology of a mechanical structure or identifying the feasible paths for a manufacturing robot, we can often view these problems through the lens of *synthesis*. In a synthesis task, we seek configurations of a physical system that achieve certain desiderata while satisfying given constraints; *i.e.*, we must optimize a physically-realizable design.

In this work, *design* refers to the parametric space over which we have control and in which, *e.g.*, we optimize. A *realization* is the object that arises when the design is instantiated, while *goal* refers to its desired properties. For example, in fabrication, the design might be a set of assembly steps, the realization would be the resulting object, while the goal could be to match target dimensions while maximizing strength. Synthesis, then, refers to finding a design whose realization achieves the goal.

Synthesis problems are challenging for several reasons. The physical realization process may be costly and time-consuming, making evaluation of many designs difficult. Moreover, the realization process—or even a simulation of it—is generally not differentiable, rendering efficient gradient-based methods inapplicable. Finally, there may be a many-to-one map from the parametric space of feasible and equally-desirable designs to realizations; *i.e.*, there may be multiple ways to achieve the goal.

Surrogate modeling is widely used to address the first two challenges, though it can still be expensive because of the need for optimization, sampling, or search algorithms to find a feasible design. More seriously, the third challenge—lack of uniqueness—creates difficulties for naïve supervised learning approaches to synthesis. Specifically, consider generating many design/realization pairs, evaluating the constraints and objectives on the realizations, and attempting to learn a supervised map from goal back to design. When multiple designs lead to the same realization, or multiple realizations achieve the same goal, the supervised learner is penalized for producing designs that are valid but happen to not be the ones used to generate the data. Moreover, this approach may learn to produce an "average" design that is actually incorrect. Figure 2.1a shows a cartoon example: if the goal is to throw a ball to reach some target distance, there are two possible launch angles (designs) resulting in landing points (realizations) at the correct spot. Performing least-squares regression

from distance to angle on the full set of distance/angle pairs, however, learns an average angle that does not satisfy the goal.

To address these challenges, we propose to use a two-stage neural network architecture that resembles an autoencoder. One stage (the decoder) acts as a differentiable surrogate capturing the many-to-one physical realization process. The other stage (the encoder) maps from a goal back to a design but, critically, it is trained end-to-end with a loss in the space of realizations that flows back through the decoder. Thus the encoder—our central object of interest for synthesis—is not constrained to match a *specific* design in a training dataset, but instead is tasked with finding *any* design that meets the desiderata of the realized output. The result is a neural network that performs *amortized* synthesis: it is trained once, and at run time produces a design that is approximately optimal, using only a feed-forward architecture. Note that our method is not an autoencoder, as the design is not a lower-dimensional representation of the goal, and the encoder and the decoder are trained in separate stages.

Our method places a number of requirements on the synthesis problem. First, to train the surrogate, we need data pairs of designs and realizations. Commonly, this would require us to generate designs, in which a substantial amount of designs are viable, and simulate them on a simulator. Second, given our current setting, the physical realization process needs to be deterministic. Third, to train the encoder, the synthesis problem should have a clear objective function, or at least we can quantify the objective. Finally, given our current feed-forward setting, we consider synthesis problems that need only one valid design, although we may further extend our method by using a generative encoder.

In this work, we demonstrate this two-stage approach on a pair of specific design tasks. The first case study is extruder path planning for a class of 3D printers (the Markforged Mark Two) that can reinforce polymer layers with discrete fibers (Figure 2.1b). Since the fibers are stiff, their shape is deformed after extrusion

(Figure 2.1c, top row), and our task is to find an extruder path that results in a given fiber shape. As shown in Figure 2.1c, this problem has the many-to-one nature described above: for a small error tolerance on fiber path, there exist infinitely many extruder paths, which may even look very different. The second case study is constrained soft robot inverse kinematics. In this work, we use a simulation of a snake-like soft robot as in Xue et al. [2020a], in which we can control the stretch ratios of each individual segment on both sides of the robot. The robot has to reach a target while avoiding an obstacle, and the locations of both are input goals. As before, there may be multiple solutions for a given goal (*i.e.*, locations of target and obstacle), as shown in Figure 2.1d.

For both case studies, we compare to two baseline algorithms. In `direct-learning`, a neural network for the synthesis problem (*i.e.*, from goal to design) is trained in a supervised manner on a set of designs. Since this effectively averages designs in the training dataset, as argued above, our method outperforms it significantly. The second baseline is `direct-optimization`, which uses a gradient-based method (BFGS) to optimize for each new design separately, given access to the trained differentiable surrogate for the realization process (decoder). Our method is competitive with this rough "performance upper bound" while using dramatically lower computational resources.

2.2 Related work

Machine learning applications in synthesis problems. In synthesis tasks, the aim is to find a design solution such that its realization achieves one or more given goals. Usually, the solution is non-unique, and only one is needed. In molecule discovery, one would like to find a molecule which has some desired properties (*e.g.*, minimal side effects, efficacy, metabolic stability, growth inhibition) [Coley et al., 2018,

Jin et al., 2020, Polykovskiy et al., 2018, Hawkins-Hooker et al., 2021, Stokes et al., 2020]. See Vamathevan et al. [2019], Chen et al. [2018] for surveys on machine/deep learning in drug discovery. Challenges of molecule discovery include discrete design space [Gómez-Bombarelli et al., 2018, Seff et al., 2019] and limited data [Jin et al., 2021] due to difficulty in simulation. In materials synthesis, researchers seek materials with specific properties (*e.g.*, Xue et al. [2020c], Xue [2022]). See Bhuvaneswari et al. [2021] for a review. Similarly, to obtain enough training data, researchers have put efforts into parsing and learning from scientific literature in natural language [Kim et al., 2017, Huo et al., 2019]. In 3D shape generation, one wants to find a 3D shape that has some desired properties: properties like 2.5D sketches [Wu et al., 2017, Wang et al., 2018] that can be easily calculated and properties like functionality [Guan et al., 2020] that need to be human-labeled. In topology optimization, researchers aim to maximize the system's performance by optimizing the material layout given boundary conditions, constraints, external loads, etc [Sigmund and Maute, 2013, Bendsoe and Sigmund, 2013, Wang et al., 2003]. Machine learning has been used to infer properties [Sasaki and Igarashi, 2019], find representations of designs [Kallioras et al., 2020, Yu et al., 2018], and directly generate designs [Kollmann et al., 2020]. In program synthesis, the goal is to find programs that realize given intentions (*e.g.*, generating 3D shapes, answering visual questions) [Gulwani, 2014, Tian et al., 2018, Yi et al., 2018], where machine learning has been used to generate programs and execute programs. In this work, we propose an approach to learn a feed-forward neural network that can directly and efficiently produce a feasible solution to a synthesis problem that satisfies the requirements mentioned above.

Surrogate/oracle-based synthesis. Surrogate/oracle-based synthesis uses an auxiliary model—a surrogate (or sometimes called oracle)—that can evaluate qualities of a design without time-consuming laboratory experiments, while still being reasonably

accurate [Koziel et al., 2011, Brookes et al., 2019]. Surrogate models can be physics-based or approximation-based [Koziel and Ogurtsov, 2014] (*i.e.*, empirical), and there are different modeling techniques for approximation-based surrogates [Koziel et al., 2011, Han et al., 2012], including polynomial regression, radial basis functions, Gaussian processes, and neural networks (*e.g.*, [Xue et al., 2020b, Beatson et al., 2020, Xue et al., 2023]). Bhosekar and Ierapetritou [2018] provide a review of surrogate-based methods, and see Koziel and Leifsson [2013] for a general review of surrogate models in engineering. There are two major approaches: optimization and sampling. The most common approach is surrogate-based optimization; see Forrester and Keane [2009] for a survey. Some application examples include: optimizing the parameters of a CPU simulator [Renda et al., 2020]; solving partial differential equations in service of PDE-constrained optimization [Xue et al., 2020a]; and optimizing stochastic non-differentiable simulators [Shirobokov et al., 2020]. Researchers have also developed methods to deal with the challenge that inputs can be out-of-distribution for the oracle [Fannjiang and Listgarten, 2020, Fu and Levine, 2020, Trabucco et al., 2021]. On the other hand, sampling-based methods have several advantages: the design space can be discrete and can generate multiple designs [Brookes et al., 2019, Brookes and Listgarten, 2018, Engel et al., 2018]. Researchers have successfully applied sampling methods to problems in chemistry [Gómez-Bombarelli et al., 2018] and biology [Gupta and Zou, 2019, Killoran et al., 2017, Rives et al., 2021]. In this work, we use surrogate-based optimization as one of our baseline algorithms, and our proposed method uses a surrogate during training to optimize for a feed-forward network for the design problem.

Differentiable surrogate of losses in machine learning. Since some loss functions in machine learning are not differentiable (*e.g.*, IoU for rotated bounding boxes, 0-1 loss in classification), researchers have proposed to learn surrogates for them.

Grabocka et al. [2019] provided a formulation of surrogate loss learning and compared several learning mechanisms on some commonly used non-differentiable loss functions. Liu et al. [2020] proposed a general pipeline to learn surrogate losses. Bao et al. [2020], Hanneke et al. [2019] provided theoretical analyses of surrogates for 0-1 loss in classification. Patel et al. [2020], Nagendar et al. [2018], Yuan et al. [2020] explored the use of surrogate losses in various real-world tasks, including medical image classification, semantic segmentation, and text detection and recognition. In this work, due to the non-uniqueness of design solutions, we further extend this idea from loss functions to more complex, physical realization processes.

Robot motion planning. Robot motion planning [Latombe, 2012] can also be viewed as a form of synthesis, and there are different types of approaches to it. One popular strategy is *optimization*. For example, researchers have used evolutionary algorithms (EA) [Leger and Bares, 1999, Cabrera et al., 2011, Hornby et al., 2001], including variants of genetic algorithms (GA) [Chung et al., 1997, Chen and Burdick, 1995, Tabandeh et al., 2016, Cabrera et al., 2002, Khorshidi et al., 2011, Kim and Khosla, 1993, Farritor et al., 1996] and covariance matrix adaptation evolution strategy (CMA-ES) [Ha et al., 2016]. Another example is constrained optimization [Whitman and Choset, 2018, Subramanian et al., 1995, Coros et al., 2013, Dogra et al., 2021], which includes extensions like sequential quadratic programming (SQP) [Campos de Almeida et al., 2020, Ha et al., 2016] and the DIRECT algorithm [Van Henten et al., 2009]. Other examples of optimization applied to robot motion planning include reinforcement learning [Singh et al., 1994, Everett et al., 2018, Wu et al., 2020, 2021, 2022] and teaching-learning-based optimization [Sleesongsom and Bureerat, 2017]. Another popular class of methods is *sampling*, including simulated annealing (SA) [Baykal and Alterovitz, 2017, Patel and Sobh, 2015, Zhu et al., 2012], probabilistic roadmaps (PRM) [Karaman and Frazzoli, 2011], rapidly exploring random trees

(RRT) [Campos et al., 2019, Baykal and Alterovitz, 2017, Campos and Kress-Gazit, 2021], *etc*. Finally, *search* methods, *e.g.*, A* search [Ha et al., 2018], have been used to solve motion planning problems. These works usually model the robot's physics directly without using a surrogate model and solve the design using methods including optimization, sampling, and search, which can be time-consuming. Our method amortizes the cost of inference, and can be used to solve planning problems with more complex physics that cannot be directly modeled. In our first case study, we plan for a trajectory of the extruder, and our design space is not the speeds of the motors but rather the coordinates of points along the trajectory. In the second case study, rather than solving a dynamic planning problem, we solve a static planning task on a soft robot, with the stretch ratios of all the controllable segments of the robot as the design space.

Path planning in 3D printing. Path planning is one of the most important problems in 3D printing, and people plan for different objectives: minimizing printing time, avoiding collision, faster planning, etc. Shembekar et al. [2018] proposed a planning algorithm to build complex shapes with multiple curvatures and can avoid collisions. Ganganath et al. [2016] minimized the printing time by modeling the task as a traveling salesman problem. Xiao et al. [2020] speeded up path planning algorithms by introducing efficient topology reconstruction algorithms. Stragiotti [2020] provided an optimization-based algorithm that minimizes compliance of a printed part. Asif [2018] introduced a planning algorithm for continuous fiber and can generate a continuous deposition path. See Huang et al. [2020] for a review of existing work in 3D printing path design. In this work, instead of the aforementioned objectives, we plan an extruder path to compensate for the deformation caused by the printing process. We demonstrate increased run-time efficiency by amortizing the

cost of physical simulation of the printing process, learning a feed-forward network to output the extruder path.

2.3 Method

We formalize synthesis as a constrained optimization problem, denoting the set of allowed designs as Θ and the set of possible realizations as $\mathcal{U}$. There is a physical process that maps from design to realization that we denote as $U : \Theta \to \mathcal{U}$. Our goal may be a function of both the realization and the design, as designs may differ in, *e.g.*, ease of manufacturing. Moreover, it may be appropriate to specify a parametric family of goals to accommodate related tasks, *e.g.*, different target locations in inverse kinematics. The user expresses a ($\boldsymbol{g}$-indexed) family of design goals via a cost function denoted $\mathcal{L}_{\boldsymbol{g}} : \Theta \times \mathcal{U} \to \mathbb{R}$. The problem of interest is to optimize the cost function with respect to the design:

$$\min_{\boldsymbol{\theta} \in \Theta} \mathcal{L}_{\boldsymbol{g}}(\boldsymbol{\theta}, \boldsymbol{u}) \quad \text{s.t.} \quad U(\boldsymbol{\theta}) = \boldsymbol{u}\,. \tag{2.1}$$

We can view this problem as a generalization of PDE-constrained optimization problems, where we have allowed for broader types of realizations than PDE solutions. Revisiting the challenges of synthesis problems articulated earlier, U may be expensive, non-differentiable, and non-injective, and $\mathcal{L}_{\boldsymbol{g}}$ may not a have a unique minimum.

Recalling the toy problem of Figure 2.1a: $\boldsymbol{\theta}$ is the angle at which the ball is thrown, $U(\boldsymbol{\theta}) = \boldsymbol{u}$ is where it lands, the goal $\boldsymbol{g}$ is a desired distance, *i.e.*, a target realization, and a (design-independent) cost function might be the squared difference between the desired and actual realizations: $\mathcal{L}_{\boldsymbol{g}}(\boldsymbol{\theta}, \boldsymbol{u}) = ||\boldsymbol{g} - \boldsymbol{u}||^2$.

Since the realization $\boldsymbol{u} = U(\boldsymbol{\theta})$ is unique for a specific design $\boldsymbol{\theta}$, we propose using a two-stage approach, which can be viewed as an autoencoder. We first learn a differentiable surrogate (decoder) $\hat{U}(\cdot)$ for the physical realization process from $\boldsymbol{\theta}$ to $\boldsymbol{u}$,

and then learn an encoder $\phi(\cdot)$ from goal $\boldsymbol{g}$ to design $\boldsymbol{\theta}$, evaluating the design quality with the trained decoder. To build a dataset, we have to randomly sample designs and calculate realizations and goals from them, since the physical realization process $U(\cdot)$ is known, and the reverse direction (*i.e.*, goal to design) is difficult as discussed before and is what we are trying to learn. Thus we first sample D designs $\boldsymbol{\theta}_1, \cdots, \boldsymbol{\theta}_D$ and build our dataset as $\mathcal{D} \coloneqq \big\{(\boldsymbol{\theta}_1, \boldsymbol{u}_1, \boldsymbol{g}_1), \cdots, (\boldsymbol{\theta}_D, \boldsymbol{u}_D, \boldsymbol{g}_D)\big\}$, where $\boldsymbol{u}_i \coloneqq U(\boldsymbol{\theta}_i)$ and $\boldsymbol{g}_i$ is the goal calculated from the realization $\boldsymbol{u}_i$. The calculation of the goal $\boldsymbol{g}_i$ summarizes properties that we care about in $\boldsymbol{u}_i$, and this process depends on the synthesis problem itself and how the cost function $\mathcal{L}_\cdot(\cdot, \cdot)$ is designed. We split $\mathcal{D}$ into $\mathcal{D}_{\texttt{train}}$, $\mathcal{D}_{\texttt{val}}$, and $\mathcal{D}_{\texttt{test}}$. We then train our surrogate $\hat{U}(\cdot)$ such that

$$\hat{U}^*(\cdot) \coloneqq \arg\min_{\hat{U}(\cdot)} \mathbb{E}_{(\boldsymbol{\theta}, \boldsymbol{u}, \cdot) \sim \mathcal{D}_{\texttt{train}}} \big[||\hat{U}(\boldsymbol{\theta}) - \boldsymbol{u}||^2\big], \tag{2.2}$$

so that $\hat{U}^*(\cdot)$ serves as a differentiable surrogate of the physical realization function $U(\cdot)$. We then use the trained decoder $\hat{U}^*(\cdot)$ to train an encoder $\phi(\cdot)$:

$$\phi^*(\cdot) \coloneqq \arg\min_{\phi(\cdot)} \mathbb{E}_{(\cdot, \cdot, \boldsymbol{g}) \sim \mathcal{D}_{\texttt{train}}} \big[\mathcal{L}_{\boldsymbol{g}}\big(\phi(\boldsymbol{g}), \hat{U}^*(\phi(\boldsymbol{g}))\big)\big]. \tag{2.3}$$

Note that although our method resembles an autoencoder, it is not an autoencoder, since we learn the decoder first and then the encoder rather than jointly. Besides, the design space is usually not a lower-dimensional representation of the goal space, and the cost function is usually not simply reconstruction loss.

2.4 Empirical evaluation approach

In this work, we use our method to solve two real problems as case studies: a task of optimizing the extruder path in additive manufacturing, and a task of actuating a soft robot in an inverse kinematics setting. While the details of the two differ, we

evaluate them in the same way. In each case, we compare our algorithm with two baselines—`direct-learning` and `direct-optimization`—and evaluate our method in terms of relative quality and run time.

`direct-learning`: One natural baseline is to directly learn a model from the goal $\boldsymbol{g}$ to the design $\boldsymbol{\theta}$. Thus we introduce the `direct-learning` model $\phi_{\texttt{dl}}(\cdot)$:

$$\phi_{\texttt{dl}}(\cdot) \coloneqq \arg\min_{\phi(\cdot)} \mathbb{E}_{(\boldsymbol{\theta},\cdot,\boldsymbol{g})\sim\mathcal{D}_{\texttt{train}}}\big[||\boldsymbol{\theta} - \phi(\boldsymbol{g})||^2 + \mathcal{R}_{\texttt{dl}}(\phi(\boldsymbol{g}))\big]. \tag{2.4}$$

Note that, since there is no surrogate, we do not have access to the realization $\boldsymbol{u}$. We thus have to slightly adjust the cost function into a squared Euclidean distance part and a regularizer part $\mathcal{R}_{\texttt{dl}}(\cdot)$ (if there is one), the latter of which comes from the original cost function $\mathcal{L}_{\cdot}(\cdot,\cdot)$. We expect our method to perform much better than `direct-learning`.

`direct-optimization`: Since our surrogate is a neural network and therefore differentiable, another natural baseline is to directly optimize the design $\boldsymbol{\theta}$ with respect to the goal $\boldsymbol{g}$ by using a gradient-based optimizer (*e.g.*, BFGS), which provides us a rough performance "upper bound" on our method:

$$\phi_{\texttt{do}}(\boldsymbol{g}) \coloneqq \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{g}}(\boldsymbol{\theta}, \hat{U}^*(\boldsymbol{\theta})). \tag{2.5}$$

Note that AmorFEA [Xue et al., 2020a] is state-of-the-art method for PDE-constrained optimization, which uses the same approach as the `direct-optimization` baseline. We expect our method to have performance close to `direct-optimization`, while running orders of magnitude faster.

To evaluate our method and the baselines, we iterate over all $\boldsymbol{g}$ from $\mathcal{D}_{\texttt{test}}$, and evaluate the quality of each tuple $\big(\phi(\boldsymbol{g}), U(\phi(\boldsymbol{g})), \boldsymbol{g}\big)$, where we use the physical

realization function $U(\cdot)$ rather than the learned surrogate $\hat{U}^*(\cdot)$. We run every experiment 3 times with random initialization of neural networks. More details are provided in each case study and in the appendix.

2.5 Case study: extruder path planning

3D printing (a.k.a. additive manufacturing) is the process of creating a 3D object from a 3D model by successively adding layers of material. It has a wide variety of applications in areas including aerospace, automotive, healthcare, and architecture [Shahrubudin et al., 2019]. Popular 3D printers use thermoplastic polymers (PLA, ABS, nylon, etc.) as the printing material, but these have limited strength. To address this issue, some recent printers support using strong fibers to reinforce the composite. In this work, we explore a 3D printer (the Markforged Mark Two) capable of extruding discrete fibers (fiberglass, kevlar, or carbon fiber) along a controllable path. However, since the fibers are stiff and non-stretchable, the printed fiber path will be "smoothed" compared to the extruder path. As shown in Figure 2.1c, without path planning, the fiber path will be severely deformed. We seek to find a general method that, given any desired fiber path, plans an extruder path to compensate for the deformation caused by the printing process. To the best of our knowledge, there is no existing automated method for this task, so it is worthwhile to tackle it using machine learning.

2.5.1 Cost function

Following the notation in Section 2.3, we denote the target fiber path as $\boldsymbol{g} \in \mathbb{R}^{n\times 2}$: a path is represented as a series of n points, and n varies from path to path (at the scale of hundreds in our experiments). We denote the extruder path (the design) as $\boldsymbol{\theta} \in \mathbb{R}^{n\times 2}$ and its realization fiber path as $\boldsymbol{u} \in \mathbb{R}^{n\times 2}$. Our cost function $\mathcal{L}_\cdot(\cdot,\cdot)$ is

defined as:

$$\mathcal{L}_{\boldsymbol{g}}(\boldsymbol{\theta}, \boldsymbol{u}) \coloneqq ||\boldsymbol{g} - \boldsymbol{u}||_2^2 + \lambda \cdot \mathcal{R}(\boldsymbol{\theta}), \tag{2.6}$$

where λ is a hyper-parameter and $\mathcal{R}(\cdot)$ is a smoothing regularizer that calculates the sum of squared empirical second-order derivatives of the extruder path:

$$\mathcal{R}(\boldsymbol{\theta}) \coloneqq \sum_{i=2}^{n-1} \left(\left(\frac{\boldsymbol{\theta}_{i+1} - \boldsymbol{\theta}_i}{||\boldsymbol{\theta}_{i+1} - \boldsymbol{\theta}_i||_2} - \frac{\boldsymbol{\theta}_i - \boldsymbol{\theta}_{i-1}}{||\boldsymbol{\theta}_i - \boldsymbol{\theta}_{i-1}||_2} \right) \Big/ \left(\frac{||\boldsymbol{\theta}_{i+1} - \boldsymbol{\theta}_i||_2 + ||\boldsymbol{\theta}_i - \boldsymbol{\theta}_{i-1}||_2}{2} \right) \right)^2, \tag{2.7}$$

where $\boldsymbol{\theta}_i \in \mathbb{R}^2$ is the i-th row of $\boldsymbol{\theta}$.

2.5.2 Evaluation metric

The most intuitive way to evaluate the quality of extruder path $\boldsymbol{\theta}$ is to measure the distance between $\boldsymbol{g}$, the desired fiber path, and $\boldsymbol{u}$, the fiber path we get by printing $\boldsymbol{\theta}$. Note that it is likely the model under-estimates or over-estimates the deformation of fiber introduced by printing, so both cases can happen when we print with path $\boldsymbol{\theta}$: we run out of fiber before we finish $\boldsymbol{\theta}$, or there is still some fiber remaining in the nozzle after we finish $\boldsymbol{\theta}$ (the total length of fiber is fixed given a desired fiber path $\boldsymbol{g}$). In other words, the $\boldsymbol{g}_i$'s and $\boldsymbol{u}_i$'s might not be synchronized. Thus, directly measuring the distance between $\boldsymbol{g}_i$ and $\boldsymbol{u}_i$ does not necessarily reflect the difference between desired fiber path and the fiber path we get. Therefore, to better measure the distance between $\boldsymbol{g}$ and $\boldsymbol{u}$ during testing, we use Chamfer distance [Fan et al., 2017], which was first proposed by Barrow et al. [1977] as an image matching technique, later developed as a commonly used (semi)metric to measure the difference between two sets, and has been shown to have a higher correlation with human judgment compared to intersection over union and earth mover's distance [Sun et al., 2018]:

$$d_{\mathtt{CD}}(\boldsymbol{g}, \boldsymbol{u}) \coloneqq \frac{1}{2} \left(\frac{1}{n} \sum_{i=1}^{n} \min_{j \in [1,n]} \|\boldsymbol{g}_i - \boldsymbol{u}_j\|_2 + \frac{1}{n} \sum_{j=1}^{n} \min_{i \in [1,n]} \|\boldsymbol{g}_i - \boldsymbol{u}_j\|_2 \right). \tag{2.8}$$

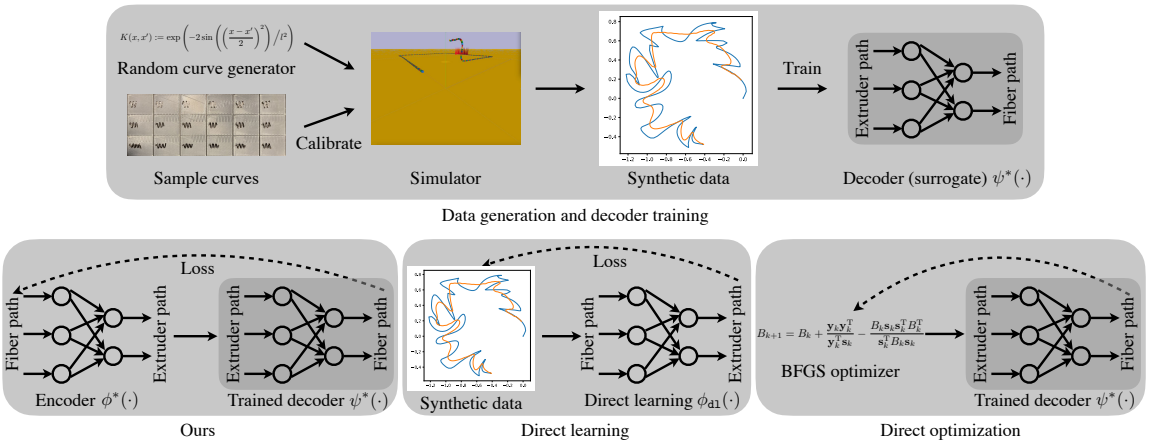


Figure 2.2: The pipeline for our method and two baselines for extruder path planning. We first generate data by building a simulator and calibrating it using a real printer, and we train the decoder and `direct-learning` using the synthetic dataset. We then train our method and build `direct-optimization` using the trained decoder.

2.5.3 Implementation

The pipeline is shown in Figure 2.2. To generate the dataset for calibrating the decoder, we first use elliptical slice sampling [Murray et al., 2010] (New BSD License) to sample random extruder paths from a Gaussian process. We then use a physical simulator built using Bullet [Coumans, 2010] (zlib License), calibrated to a real printer, to predict the realization (fiber path) for each extruder path. We generate 10,000 paths, split into 90% training, 5% validation, and 5% testing. For decoder, encoder, and `direct-learning`, we use an MLP with 5 hidden layers and ReLU as the activation function. The MLP takes 61 points as input and produces 1 point as output, and is applied in a "sliding window" fashion over the entire path (details in appendix). We train every model with a learning rate of 1×10^{-3} for 10 epochs using PyTorch [Paszke et al., 2019b] and Adam optimizer [Kingma and Ba, 2015]. For `direct-optimization`, we use the BFGS implementation in SciPy [Virtanen et al., 2020a]. More implementation details are included in the appendix.

Table 2.1: Path-planning evaluation of the average Chamfer distance on the test set evaluated in simulation

Regularizer weight	0.1	0.3	0.6	1.0	1.5
`direct-learning`	0.0314±0.0008	0.0319±0.0016	0.0502±0.0072	0.1007±0.0292	0.1457±0.0216
Ours	**0.0180±0.0004**	**0.0157±0.0003**	**0.0164±0.0004**	**0.0158±0.0002**	**0.0156±0.0002**
`direct-optimization`			0.0155±0.0002		

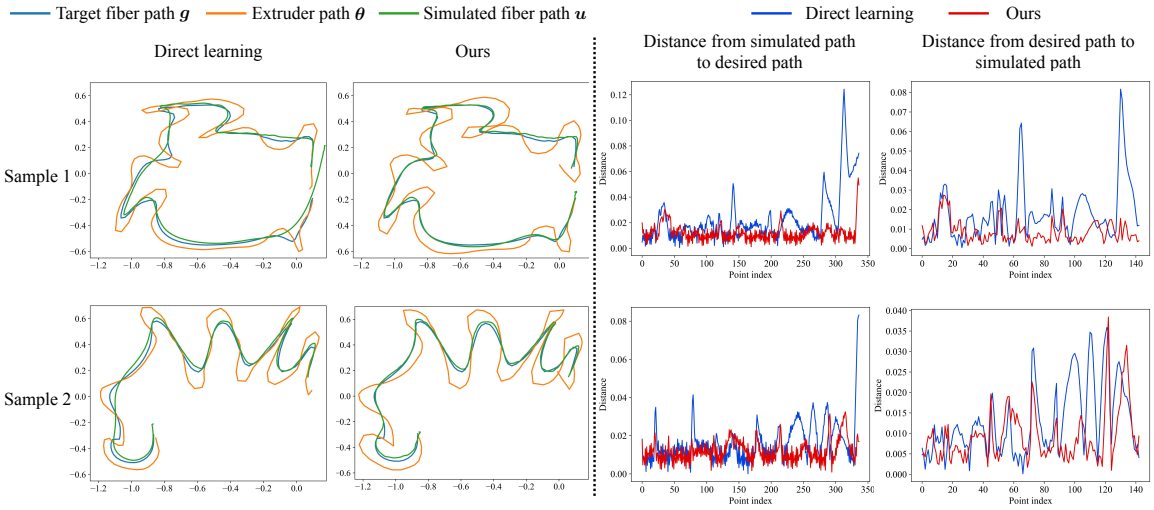


Figure 2.3: Path-planning evaluation of `direct-learning` *vs.* ours on the test set evaluated in simulation. We also visualize the Chamfer distance for each point on both the simulated fiber path and the desired fiber path.

2.5.4 Experiments

Fiber path quality evaluated in simulation. To quantitatively evaluate the effectiveness of our approach, the most straightforward way is to run its prediction on the simulator and see how close the simulated fiber path is to the input fiber path. We compare the performance of our approach (encoder), `direct-learning`, and `direct-optimization` on the test set of 500 paths, for different values of the regularization parameter λ. We report the average Chamfer distance (§ 2.5.2) with the standard error among 3 runs in Table 2.1. Note that `direct-optimization` runs very slowly, so we instead tune its regularizer weight on the first 40 test samples, select the best regularizer weight, and report its performance on the whole test set using the selected regularizer weight (more details in the appendix). The results demonstrate

Table 2.2: Path-planning evaluation of the average running time on the first 50 samples in the test set

	Avg. time (s)
`direct-learning`	7.96×10^{-4}
Ours	7.96×10^{-4}
`direct-optimization`	1.17×10^{4}

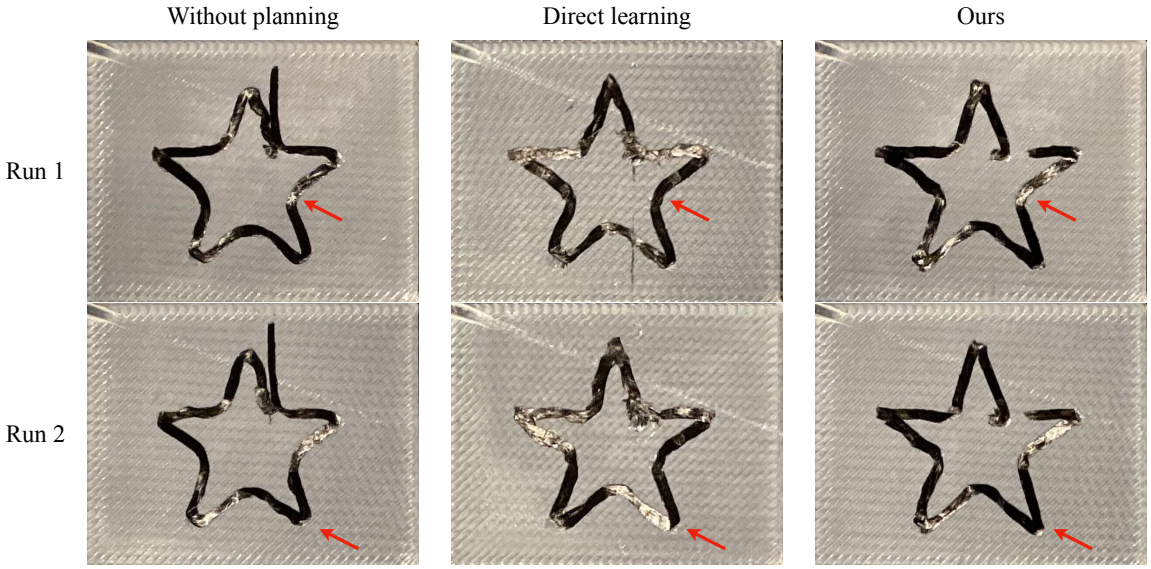


Figure 2.4: Path-planning evaluation of "without planning" *vs.* `direct-learning` *vs.* ours on Markforged Mark Two, with a star as the desired fiber. For "without planning", we have the same extruder path for the 2 runs; for `direct-learning` and ours, the 2 runs are predictions from neural networks trained with different initializations.

that our method significantly outperforms `direct-learning`, and the performance is comparable to `direct-optimization`.

As a qualitative evaluation, Figure 2.3 shows the predictions of both `direct-learning` and our method on samples from the test set. We select the run with minimum average Chamfer distance over the test set for both `direct-learning` and our method, respectively. The results indicate that our method is better than `direct-learning` on handling details in the fiber path. We also visualize the Chamfer distance from each individual point on one path to the other, and observe that the distances for our method are generally lower than for `direct-learning`.

Running time comparison. To train needed neural networks, `direct-optimization` takes roughly 10 minutes, `direct-learning` takes roughly 1 hour, and our method takes roughly 5 days. Note that training costs are amortized, since we only need to train once. We then evaluate the inference time of the three algorithms on a server with two Intel(R) Xeon(R) E5-2699 v3 CPUs running at 2.30GHz. Since small neural networks generally run faster on the CPU, we run all of the tests solely on CPU. We run every algorithm on the first 50 paths in the test set and report the average inference time in Table 2.2. As we can see, both ours and `direct-learning` achieve a running time below 1 millisecond, while `direct-optimization` runs orders of magnitude slower.

Fiber path quality evaluated on a real printer. Lastly, we test our extruder path solutions on a real Markforged Mark Two 3D printer. We set the desired fiber path to a star, and print the star itself (without planning) as well as solutions from both `direct-learning` and our method. We select regularizer weights based on Table 2.1, *i.e.*, 0.1 for `direct-learning`, 1.5 for our method, and we show two of the trained models. The results are visualized in Figure 2.4, confirming that our method successfully improves the quality of the printed fiber.

2.6 Case study: constrained soft robot inverse kinematics

Partial differential equations (PDEs) are a powerful tool for describing complex relationships between variables and have been used widely in areas including physics, engineering, and finance. In PDE-constrained optimization [Biegler et al., 2003], the goal is to optimize a cost function such that the constraints can be written as PDEs, *i.e.*, the solutions are consistent with the relationships specified by the PDE. In the

context of synthesis problems, we can consider the boundary conditions of the PDE as the design, and the solution of the PDE as the realization. Similar to other synthesis problems, different boundary conditions can result in the same PDE solution, and the cost function may not have a unique minimum. In the above situations, we propose to apply our method. We test our method on a specific PDE-constrained optimization problem—constrained soft robot inverse kinematics, which serves as a representative use case of our method in this large category of problems.

Soft robots made of elastic materials, have received significant recent attention because of their reduced potential harm when working with humans [Rus and Tolley, 2015]. Researchers have explored a variety of applications, including surgical assistance [Cianchetti et al., 2014], bio-mimicry [Li et al., 2021a], and human-robot interaction [Pang et al., 2020]. In this case study, as in Xue et al. [2020a], we use a snake-like soft robot with a fixed bottom, in which we can control the stretch ratios on both sides of the robot. As shown in Figure 2.1d, the objective is for the midpoint at the top of the robot to reach a target, while making sure that the robot does not collide with a fixed-size circular obstacle. The relationship between the robot's shape and the stretch ratios can be written as a PDE, and as shown in Figure 2.1d, there are different solutions to achieve the goal.

2.6.1 Cost Function

We adopt the soft robot from Xue et al. [2020a], which has an original height of 10 and an original width of 0.5, with its bottom is fixed. The goal vector $\boldsymbol{g}$ is in $\mathbb{R}^{2\times 2}$, where $\boldsymbol{g}_1 \in \mathbb{R}^2$ indicates the target location and $\boldsymbol{g}_2 \in \mathbb{R}^2$ indicates the obstacle location. We denote the radius of the obstacle as r, which we set to 0.9. The design (control) vector $\boldsymbol{\theta}$ is in $\mathbb{R}^n$ with $n = 40$, with $\theta_i \in \mathbb{R}$ indicating the stretch ratio of the i-th segment (*e.g.*, $\theta_i = 0.95$ indicates the i-th segment is contracted by 5%). The physical realization vector $\boldsymbol{u}$ is in $\mathbb{R}^{m\times 2}$ with $m = 103$, where $\boldsymbol{u}_i \in \mathbb{R}^2$ indicates the location of

the i-th vertex on the soft robot's mesh. The location of the top midpoint is denoted as $\boldsymbol{u}_{\texttt{tm}} \in \mathbb{R}^2$. Implicitly, the relationship between $\boldsymbol{\theta}$ and $\boldsymbol{u}$ obeys a PDE that governs the deformation of the robot, as detailed in Xue et al. [2020a].

The cost function $\mathcal{L}_{\cdot}(\cdot, \cdot)$ is defined as

$$\mathcal{L}_{\boldsymbol{g}}(\boldsymbol{\theta}, \boldsymbol{u}) \coloneqq \frac{1}{2}||\boldsymbol{g}_1 - \boldsymbol{u}_{\texttt{tm}}||_2^2 + \lambda_1 \cdot \mathcal{B}(\boldsymbol{u}, \boldsymbol{g}_2) + \lambda_2 \cdot \mathcal{R}(\boldsymbol{\theta}). \tag{2.9}$$

The first term $||\boldsymbol{g}_1 - \boldsymbol{u}_{\texttt{tm}}||_2^2$ is the squared Euclidean distance between the top midpoint of the robot and the target. The second term, weighted by a hyper-parameter λ_1 (which we fix at 0.5), enforces the constraint via a barrier function [Nesterov et al., 2018, Nocedal and Wright, 2006] for the obstacle $\mathcal{B}(\boldsymbol{u}, \boldsymbol{g}_2)$:

$$\mathcal{B}(\boldsymbol{u}, \boldsymbol{g}_2) \coloneqq \frac{1}{m} \sum_{i=1}^{m} \big(\max(r + \Delta r - ||\boldsymbol{u}_i - \boldsymbol{g}_2||_2, 0) \big)^2, \tag{2.10}$$

where Δr is a hyper-parameter (which we fix at 0.1). A positive Δr provides a penalty as well as nonzero gradients when the robot gets close to the obstacle. The last term contains another hyper-parameter λ_2, varied in our experiments, weighting a smooth regularization term $\mathcal{R}(\boldsymbol{\theta})$, with

$$\mathcal{R}(\boldsymbol{\theta}) \coloneqq \frac{1}{n-4} \sum_{\substack{1<i<n, i \neq n/2, \\ i \neq n/2+1}} \left(\frac{\theta_{i+1} - \theta_i}{2} - \frac{\theta_i - \theta_{i-1}}{2} \right)^2, \tag{2.11}$$

where θ_i for $i = 1, 2, \cdots, n/2$ corresponds to stretch ratios on the left-hand side of the robot, and θ_i for $i = n/2 + 1, \cdots, n$ corresponds to stretch ratios on the right-hand side of the robot. This regularizer prevents unphysical deformations with strong discontinuities.

Table 2.3: Soft-robot evaluation of the number of successful cases (over 1,000) on test set

Regularizer weight	0.03	0.05	0.07	0.09
`direct-learning`	907.7±3.1	918.3±3.4	910.7±3.4	912.3±3.8
Ours	**986.3±0.5**	**975.0±3.9**	**981.7±5.0**	**984.7±4.5**
`direct-optimization`	997.0±0.5	998.0±0.0	998.3±0.7	997.7±0.5

2.6.2 Evaluation metric

Since there are two objectives—"reach" and "avoid"—in this task, we have two evaluation metrics. The first metric is the number of cases that successfully avoid the obstacle (*i.e.*, have all vertex positions outside the obstacle circle). The second metric is the average Euclidean distance of the robot's top midpoint to the target for successful cases.

2.6.3 Implementation

Using the finite element method [Hughes, 2012] and the code from Xue et al. [2020a] (MIT license), we randomly generate 40,000 data samples, and split them into 90% training, 7.5% validation, 2.5% testing. For encoder, decoder, and `direct-learning`, we use an MLP with 3 hidden layers and ReLU activation. We train every model for 200 epochs with a learning rate of 1×10^{-3} using PyTorch [Paszke et al., 2019b] and Adam optimizer [Kingma and Ba, 2015]. For `direct-optimization`, we use the BFGS implementation in SciPy [Virtanen et al., 2020a]. More implementation details are included in the appendix.

2.6.4 Experiments

Design quality evaluation. We experiment with different regularizer weights λ_2 for our method and the two baselines. During training, we randomly sample the location of the obstacle, and we ensure the robot never collides with the obstacle for

Table 2.4: Soft-robot evaluation of the average distance to the target on successful cases on test set

Regularizer weight	0.03	0.05	0.07	0.09
`direct-learning`	0.2171±0.0016	0.2200±0.0028	0.2236±0.0006	0.2179±0.0036
Ours	**0.0657±0.0093**	**0.0464±0.0018**	**0.0599±0.0097**	**0.0691±0.0224**
`direct-optimization`	0.0233±0.0003	0.0240±0.0004	0.0241±0.0004	0.0242±0.0003

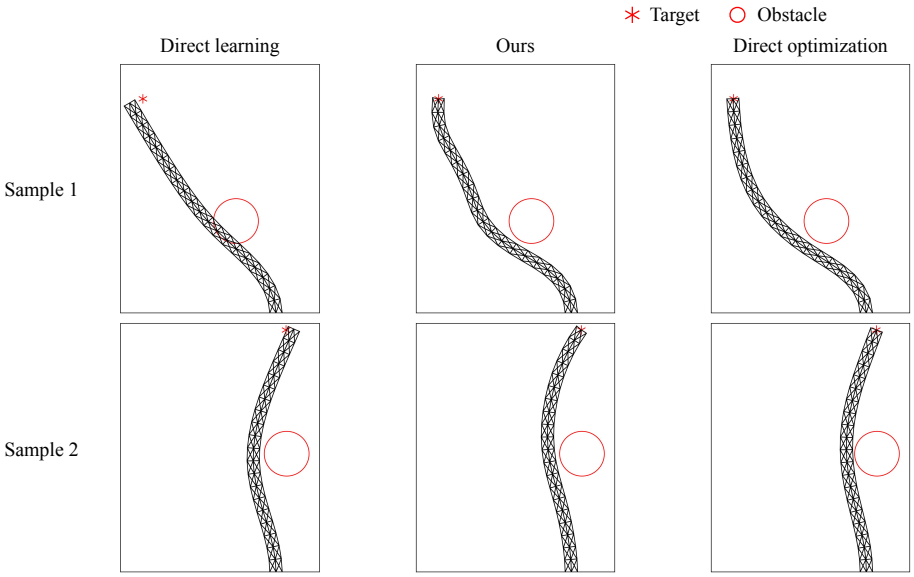


Figure 2.5: Soft-robot evaluation of `direct-learning` *vs.* ours *vs.* `direct-optimization` on examples from the test set. The direct learning baseline both violates the constraints (Sample 1) and fails to reach the target (Samples 1 and 2), while the run time of `direct-optimization` is 4000 times that of our method.

`direct-learning`, since it does not have access to the realization vector and thus its loss function cannot contain the barrier function term for the obstacle (more details about `direct-learning` are in the appendix). For a fair comparison, during testing, we set the random seed to 0 such that for the same test sample, the obstacle will appear at the same location for all algorithms. The number of cases in which the robot successfully avoids the obstacle, with standard error for 3 runs, is shown in Table 2.3. The average Euclidean distance to the target for successful cases, with its standard error, is shown in Table 2.4. As the numbers show, our method is competitive to `direct-optimization`, and performs much better than `direct-learning`. Samples from the test set are shown in Figure 2.5 (for both algorithms, we select the best run

Table 2.5: Soft-robot evaluation of the average running time on the test set

	Avg. time (s)
`direct-learning`	3.12×10^{-4}
Ours	3.34×10^{-4}
`direct-optimization`	1.33×10^{0}

with a regularizer weight of 0.5). We can see that our method collides less frequently while reaching the target more accurately than `direct-learning`.

Running time. To train the neural networks, `direct-optimization` takes roughly 2 hours, `direct-learning` takes roughly 4 hours, and our method takes roughly 4.5 hours. Note that training costs are amortized, since we only need to train once. We then test all algorithms on a server with two Intel(R) Xeon(R) E5-2699 v3 CPUs running at 2.30GHz. Everything runs solely on the CPU, maximizing efficiency for the small neural networks we use. The results are shown in Table 2.5, showing that we successfully reduce the running time from over 1 second to less than 1 millisecond. Note that the soft robot is relatively small (40 control variables), and the time complexity of BFGS grows quadratically w.r.t. the number of parameters. Therefore, for more complex soft robots or PDE-constrained optimization problems with a larger number of variables, the running time advantage of our method can be of even greater importance.

2.7 Discussion

In this work, we provided an amortized approach to synthesis problems in machine learning. To tackle the non-differentiability of physical system realizations, the huge computational cost of realization processes, and the non-uniqueness of the design solution, we designed a two-stage neural network architecture, where we first learn the decoder, a surrogate that approximates the realization processes, and then learn

the encoder, which proposes a design for an input goal. We tested our approach on two case studies on fiber extruder path planning and constrained soft robot inverse kinematics, where we demonstrated that our method provides designs with much higher quality than supervised learning of the design problem, while being competitive in quality to and orders of magnitude faster than direct optimization of the design solution.

Although the experiments in both case studies show the effectiveness of our approach, we would like to mention some limitations of our method. First, to effectively learn a differentiable surrogate for the realization process, we need to be able to generate a substantial number of viable designs. We also need a simulator to calculate physical realizations of them, and the realization process has to be deterministic, although extensions might consider probability distributions over realizations. Also, to train the encoder, we need the objective ("goal") to be quantifiable. Our method provides the greatest gains if the realization is computationally expensive and/or non-differentiable, or if our encoder can exploit the non-uniqueness of designs to choose one good option where supervised learning would have learned a poor "average" solution. Additionally, due to the cost of neural network training, amortization is only a good idea when we need to solve one design problem many times with different goals, or we need fast inference. From a societal point of view, the primary negative consequence is the potential for replacing human labor in design. We view the present approach, however, as part of larger human-in-the-loop design processes in line with other software tools for modeling and fabrication.

Chapter 3

More Stiffness with Less Fiber: End-to-End Fiber Path Optimization for 3D-Printed Composites

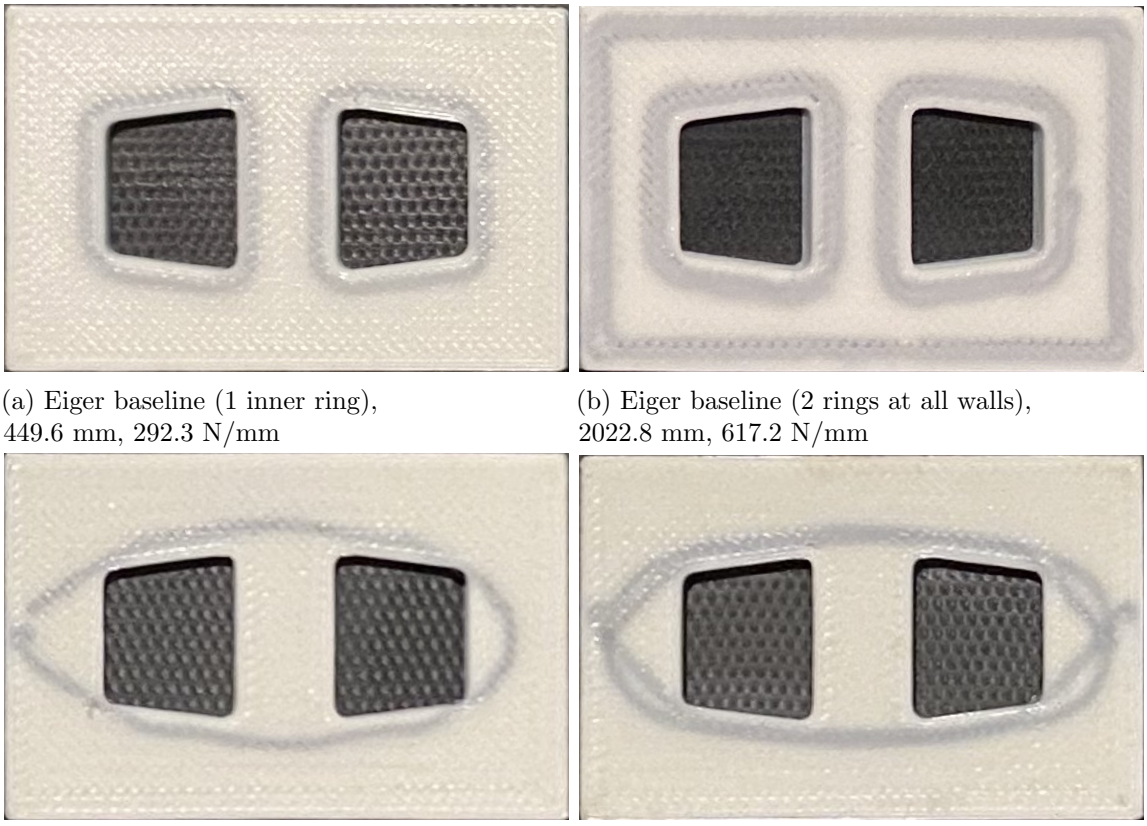
In 3D printing, stiff fibers (*e.g.*, carbon fiber) can reinforce thermoplastic polymers with limited stiffness. However, existing commercial digital manufacturing software only provides a few simple fiber layout algorithms, which solely use the geometry of the shape. In this work, we build an automated fiber path planning algorithm that maximizes the stiffness of a 3D print given specified external loads. We formalize this as an optimization problem: an objective function is designed to measure the stiffness of the object while regularizing certain properties of fiber paths (*e.g.*, smoothness). To initialize each fiber path, we use finite element analysis to calculate the stress field on the object and greedily "walk" in the direction of the stress field. We then apply a gradient-based optimization algorithm that uses the adjoint method to calculate the gradient of stiffness with respect to fiber layout. We compare our approach,

in both simulation and real-world experiments, to three baselines: (1) concentric fiber rings generated by Eiger, a leading digital manufacturing software package developed by Markforged, (2) greedy extraction on the simulated stress field (*i.e.*, our method without optimization), and (3) the greedy algorithm on a fiber orientation field calculated by smoothing the simulated stress fields. The results show that objects with fiber paths generated by our algorithm achieve greater stiffness while using less fiber than the baselines—our algorithm improves the Pareto frontier of object stiffness as a function of fiber usage. Ablation studies show that the smoothing regularizer is needed for feasible fiber paths and stability of optimization, and multi-resolution optimization help reduce the running time compared to single-resolution optimization.

3.1 Introduction

Additive manufacturing has revolutionized the ability to fabricate three-dimensional objects of high geometric complexity, with a variety of applications including in healthcare, automotive, and aerospace industries [Shahrubudin et al., 2019]. However, the increasing flexibility in manufacturing has outstripped our ability to produce designs that optimally take advantage of 3D printers. This has motivated research on *computational fabrication* pipelines that augment human specification of goals with computational optimization of designs that best realize those goals, for problems ranging from ensuring structural integrity through controlling appearance and fine-tuning the fabrication process Attene et al. [2018].

In this work, we address the problem of producing structurally-sound parts that are capable of bearing nontrivial load. We aim to exploit the capabilities of devices such as the Markforged Mark Two Markforged [2022b], which is based on conventional fused deposition modeling (FDM) using thermoplastic nylon, but augments this with the ability to extrude and deposit continuous fibers. Options for the latter include carbon



(a) Eiger baseline (1 inner ring), 449.6 mm, 292.3 N/mm

(b) Eiger baseline (2 rings at all walls), 2022.8 mm, 617.2 N/mm

(c) Optimized fiber path (1 ring), 372.7 mm, 483.8 N/mm

(d) Optimized fiber path (2 rings), 799.5 mm, 815.0 N/mm

Figure 3.1: Planned and 3D printed fiber paths with fiber lengths and average stiffness measured over four batches annotated, for a part with external tension applied between two holes. (a) (b): Concentric fiber rings generated by the Eiger baseline only consider geometry. (c) (d): Our optimized fiber paths, tuned for the applied loads, yield greater stiffness at lower fiber lengths.

fiber, Kevlar, fiberglass, and HSHT (High Strength High Temperature) fiberglass, all of which offer the ability to selectively strengthen printed parts with respect to tensile loads. In effect, this process creates fiber-reinforced plastic (FRP) composites [Kabir et al., 2020], but with the ability to control fiber placement to achieve specific tradeoffs in strength, weight, and cost.

The optimization of fiber layout is similar to problems traditionally considered in computational fabrication, such as topology optimization (*i.e.*, removing material from certain regions) and spatially-varying assignment of different materials. Systems for these latter tasks are typically based on Eulerian analysis and optimization, in which

a quantity (density, material choice, *etc.*) is determined for each location in space (*e.g.*, on a voxel grid). Similarly, almost all existing methods for optimizing carbon fiber composites focus on the spatially-varying fiber direction field, then use variants of greedy extraction or ODE solvers to extract the fiber paths themselves [Wang et al., 2021, Schmidt et al., 2020].

In contrast, we are inspired by a Lagrangian point of view: we characterize the strength of the part as a function of the fiber path, compute gradients with respect to changes in fiber coordinates, and optimize the fiber path directly using gradient descent. This strategy is based on the adjoint method [Errico, 1997, Cao et al., 2003], commonly used for PDE-constrained optimization, and exploits modern systems for automatic differentiation [Griewank and Walther, 2008], which have evolved considerably in recent years to support a range of machine learning and general optimization problems. Our end-to-end optimization approach has the benefit of focusing directly on the final goal—maximizing stiffness with respect to external loads—rather than on indirect objectives such as minimizing strain throughout the object.

We incorporate our gradient descent-based optimization into a complete system that addresses three key challenges: (1) solving for the stress field of the object given external loads, (2) computing an optimization objective and its gradient based on the stress field, and (3) providing a good initialization of fiber layout for our local optimizer. To address the first challenge, we model the composite material using the linear elastic model, and approximately solve the PDE using the finite element method. Without loss of generality and for the sake of simplicity, we model the composite material in two dimensions under the assumption of in-plane stress (*i.e.*, we only consider laminates). We also simplify the problem by considering Dirichlet (fixed-displacement) boundary conditions. To address the second challenge, we design an objective function based on total strain energy given the boundary conditions: under the assumption of linear elasticity, maximizing this energy is equivalent to maximizing the object's

stiffness. We regularize the objective to ensure that the optimized fiber paths are feasible. Finally, to address the last challenge, we initialize each fiber path by greedily following the directions of maximum tensile stress (or perpendicular to the direction of maximum compressive stress). We further use a multi-resolution approach inspired by multigrid methods, to reduce the running time of the optimization.

We show designs produced by our method on a number of illustrative case studies, demonstrating that our method yields higher stiffness with less fiber as compared to baseline paths produced by the Eiger software by Markforged [Markforged, 2022a]. We compare our results to greedy extraction based on either the stress field or optimized fiber direction field, as well as other ablations including omitting regularization or multi-resolution optimization. We print our designs (see Figure 3.1), using the method of Sun et al. [2021] to compute fiber extruder paths that compensate for fiber stiffness. Finally, we test our printed parts to verify that our method matches the predicted stiffness in the real world (subject to inherent print-to-print variations in material strength).

3.2 Related work

3.2.1 Fiber orientation optimization in 3D printing

A task that is similar to fiber path planning is fiber orientation optimization, where researchers discretize space into elements and optimize fiber orientations in them. Additional steps, such as greedy extraction, ODE solvers, or geometric methods, must be performed to produce fiber paths from the orientation field. Thus fiber orientation optimization can serve as the first step of fiber path planning, which we will discuss in § 3.2.2. See Hu [2021] for a survey (called "free material optimization"). The most common approach for orientation optimization is to set density and orientation as design variables and optimize an objective such as compliance [Chu et al., 2021,

da Silva et al., 2020, Jung et al., 2022] or the Tsai–Wu failure criterion [Ma et al., 2022] with a gradient-based optimizer. To address the checkerboard pattern issue (periodicity of the orientation variables), researchers usually use filtering [Andreassen et al., 2011] to smooth the design variable field (*e.g.*, through a weighted average of neighboring elements). Another choice of design variable is the lamination parameters: Shafighfard et al. [2019] and Demir et al. [2019] proposed to first optimize lamination parameters, search for the best fitting fiber orientations from the optimized lamination parameters, and then perform an optimization on the orientation field while considering manufacturing constraints (*e.g.*, curvature). There are also iterative variants. For example, Caivano et al. [2020] proposed iterating between calculating the principal stress direction and updating the material distribution until convergence. While mainly concentrating on orientation optimization, some approaches do ultimately generate fiber paths. For example, Fedulov et al. [2021] first optimized density and orientation and then used third-party software for printing trajectory generation; Schmidt et al. [2020] performed density and orientation optimization and generated streamlines using the 4th-order Runge-Kutta integrator for visualization.

3.2.2 Fiber path planning in 3D printing

A variety of path planning algorithms have been proposed for continuous fiber-reinforced plastics—see Zhang et al. [2020] for a survey. The most common approach is to first perform an optimization (topology, orientation, *etc.*), and then extract fiber paths from the result. As discussed, orientation optimization is one choice of the optimization (*i.e.*, use density and orientation as the design variables), but there are different methods for path extraction. Wang et al. [2021] proposed to "walk" in the field along with the stress direction and consider the angle turned in every move to produce smoothed paths. Papapetrou et al. [2020] described three methods for path extraction: the offset method and the EQS (Equally-Space) method use the geometry

of the optimized layout, and the streamline method fits the orientation field with streamlines. Safonov [2019] proposed to alternate between topology optimization and fiber orientation updates using an evolutionary heuristic method.

There are also more potential choices for the design variable. For example, one choice is to only optimize the density. Li et al. [2021b] performed topology optimization of material density (without orientation) using regularizers that force the fiber material to form lines. However, they did not extract fiber paths explicitly at the end, so it is unclear whether the fibers are directly printable. Li et al. [2020] and Chen and Ye [2021] proposed to lay fibers along with the load transmission trajectories. Almeida Jr. et al. [2019] proposed to perform the SIMP (Solid Isotropic Material with Penalization) method first, designed the fiber pattern manually, and then used a genetic algorithm to determine the number of fiber rings/paths that would minimize compliance (defined as mass divided by stiffness). Sugiyama et al. [2020] proposed to calculate the stress field and update fiber paths so that they follow the direction of maximum principal stress, repeating this process until convergence. Apart from these two-stage approaches, there are also end-to-end approaches based on genetic algorithms. For example, Yamanaka et al. [2016] modeled fiber paths as streamlines and optimized them directly using a genetic algorithm.

In summary, most existing works perform fiber planning in two stages (topology/orientation optimization followed by path extraction). In contrast, our method performs an end-to-end optimization of the fiber layout, maximizing the regularized object stiffness via a gradient-based optimizer.

3.2.3 PDE-constrained optimization

Also related to the problem of optimizing geometry to maximize stiffness is the area of PDE-constrained optimization, in which an optimization problem is subjected to physical constraints expressed via partial differential equations (PDEs) [De los

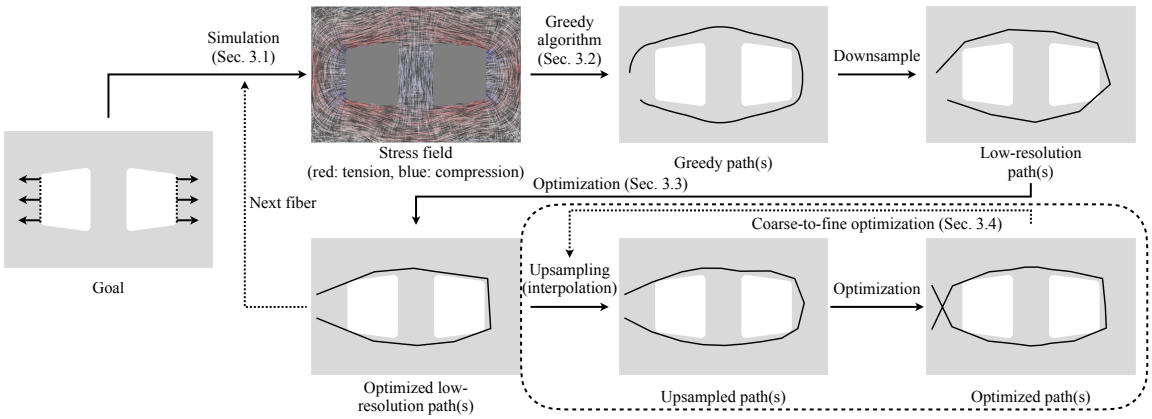


Figure 3.2: We repeatedly use the finite element method to calculate the stress field of the object (§ 3.3.1), extract a new fiber path by greedily "walking" on the stress field (§ 3.3.2), optimize the downsampled fiber path with an objective function designed to maximize stiffness and regularize fiber paths to be manufacturable (§ 3.3.3), and finally upsample and optimize all the fiber paths several times to perform coarse-to-fine optimization (§ 3.3.4).

Reyes, 2015]. There are two common types of algorithms to solve PDE-constrained optimization problems: *all-at-once* and *black-box* [Herzog and Kunisch, 2010]. *All-at-once* treats both the design variable and the state variable as independent variables, so the method may not satisfy the constraints before the optimization finishes. A common *all-at-once* algorithm is SQP (sequential quadratic programming) [Boggs and Tolle, 1995]. A disadvantage of the *all-at-once* approach is the dimension of the state variable can be very large, which makes the optimization costly. *Black-box* solves the problem in reduced form, by treating the design variable as the only independent variable, so that a gradient-based optimizer can be applied (*e.g.*, gradient descent, Newton's method). We formalize the fiber path planning task as a PDE-constrained optimization problem and use the *black-box* approach, specifically the adjoint method, to solve it.

3.3 Method

The pipeline of our method is shown in Figure 3.2. Starting from a goal (a shape with some external loads), we first simulate the stress field using the finite element method

(§ 3.3.1). We apply a greedy fiber extraction algorithm by "walking" in the stress field, and then downsample the greedy path (§ 3.3.2). We build and optimize an objective function based on the object's stiffness and regularity conditions of the fiber paths, using the adjoint method to calculate the gradients of the objective (§ 3.3.3). These steps can be repeated until a desired number of fiber paths are extracted and optimized. We then perform a coarse-to-fine optimization by upsampling and optimizing the fiber paths a specified number of times (§ 3.3.4).

3.3.1 Simulation

In this subsection, we describe how we solve the stress field given a shape, some external loads, and a specified fiber layout. We denote the body as Ω, the stress tensor as $\boldsymbol{\sigma}$, the strain tensor as $\boldsymbol{\varepsilon}$, the displacement vector as $\mathbf{u}$, and the stiffness tensor as C. The linear elastic model can be written as

$$\begin{aligned} -\nabla \cdot \boldsymbol{\sigma} &= f, \\ \boldsymbol{\varepsilon} &= \frac{1}{2}\big(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\intercal}\big), \\ \boldsymbol{\sigma} &= \mathsf{C} : \boldsymbol{\varepsilon}, \end{aligned} \tag{3.1}$$

where f is the body force and we set it to 0. For certain regions on the boundary of Ω (*i.e.*, $\partial\Omega$), the value of $\mathbf{u}$ is given as input (*i.e.*, Dirichlet boundary condition). For the remaining regions, we have $\boldsymbol{\sigma} \cdot \mathbf{n} = T$ (*i.e.*, Neumann boundary condition), where $\mathbf{n}$ is the outward unit normal vector, and T is the tractive force which we set to 0.

The constitutive equations $\boldsymbol{\sigma} = \mathsf{C} : \boldsymbol{\varepsilon}$ can also be written in a matrix product form; under the assumption of in-plane stress, we have

$$\begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{12} \end{bmatrix} = \begin{bmatrix} \frac{E_1}{1-\nu_{21}\nu_{12}} & \frac{E_1\nu_{21}}{1-\nu_{21}\nu_{12}} & 0 \\ \frac{E_2\nu_{12}}{1-\nu_{12}\nu_{21}} & \frac{E_2}{1-\nu_{12}\nu_{21}} & 0 \\ 0 & 0 & \mu \end{bmatrix} \begin{bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ 2\varepsilon_{12} \end{bmatrix}, \tag{3.2}$$

where E_1 and E_2 are Young's moduli, ν_{12} and ν_{21} are the Poisson's ratios, and μ is the shear modulus. For simplicity, we assume both plastic and fiber are isotropic materials, and they have different Young's moduli E_{plastic} and E_{fiber} and identical Poisson's ratio ν.

The next issue is to calculate the Young's modulus field. Consider a laminate of height h_{object}, with some layers filled with just plastic and others containing both plastic and fiber. We assume that all layers with fiber, adding up to a total height of h_{fiber}, have identical fiber paths, and omit plastic where fiber is present. The set of fiber paths is denoted as P, and every path p in it is a sequence of vertices on the fiber path. For a point $x \in \Omega$, for the purpose of differentiability, we define its "soft" Young's modulus as

$$E(x) \coloneqq E_{\text{plastic}} \cdot \alpha_{\text{plastic}}(x) + E_{\text{fiber}} \cdot \alpha_{\text{fiber}}(x), \tag{3.3}$$

where

$$\alpha_{\text{fiber}}(x) \coloneqq \sum_{p \in P} \exp\left(-\left(\frac{\text{dis}(p,x)}{w_{\text{fiber}}/2}\right)^2\right) \cdot h_{\text{fiber}}, \tag{3.4}$$

where $w_{\text{fiber}} = 0.9$ mm is the width of the fiber, $\text{dis}(\cdot,\cdot)$ measures the distance between a point and a path, and

$$\alpha_{\text{plastic}}(x) \coloneqq h_{\text{object}} - \min(\alpha_{\text{fiber}}(x), h_{\text{fiber}}). \tag{3.5}$$

We allow fiber paths to overlap in this setting, as even in real prints from the Markforged Mark Two, we do not observe any problems. We then have $\mu(x) = \frac{E(x)}{2(1+\nu)}$. Finally, we solve the PDE in Equation 3.1 using FEniCS with DOLFIN [Logg and Wells, 2010] by solving its first-order condition. Figure 3.2 visualizes an example of the calculated stress field, using line integral convolution Cabral and Leedom [1993].

3.3.2 Greedy fiber path extraction

In this subsection, we describe how we greedily extract a fiber path from a stress field along the directions of maximum tensile stress or perpendicular to the direction of maximum compressive stress. With the stress tensor $\boldsymbol{\sigma}$ calculated from § 3.3.1, we first calculate the stress on plastic:

$$\boldsymbol{\sigma}_{\text{plastic}} \coloneqq \boldsymbol{\sigma} \cdot \frac{E_{\text{plastic}} \cdot \alpha_{\text{plastic}}}{E_{\text{plastic}} \cdot \alpha_{\text{plastic}} + E_{\text{fiber}} \cdot \alpha_{\text{fiber}}}. \tag{3.6}$$

For any point $x \in \Omega$, we can calculate the eigenvalue with the largest absolute value $\lambda(x)$ and its corresponding eigenvector $\mathbf{v}(x)$. We then build a scalar field with $|\lambda(x)|$ and randomly sample a starting point x_0 with the field as sampling weights. From the starting point, we walk in both directions along with $\pm\mathbf{v}(x_0)$ (or perpendicular to $\mathbf{v}(x_0)$ if $\lambda(x_0)$ is negative) at a fixed step size of 0.5 mm. If we walk outside Ω or within 1.3 mm to $\partial\Omega$ (number measured from prints from Eiger), we retry at most 19 times with a random rotation uniformly sampled between $-\pi/12$ to $\pi/12$. The algorithm stops when a preset length limit is reached, or we cannot walk in both directions even after retries.

We then downsample the extracted fiber path by keeping 1 of every 20 vertices. We iterate every subsequence of the downsampled path and select the one that minimizes the objective function we will define in Section 3.3.3. We repeat this process 10 times (sampling 10 starting points) and keep the one that minimizes the objective function.

3.3.3 Gradient calculation and optimization

In this subsection, we describe how we design an objective function and optimize it using a gradient-based optimizer. We denote the optimized strain energy in Equation 3.1

as U, and the set of fiber paths as P. The objective $\mathcal{L}(P)$ is defined as

$$-U + \sum_{p \in P} \left(w_{\text{lap}} \cdot \mathcal{L}_{\text{lap}}(p) + w_{\text{min\_l}} \cdot \mathcal{L}_{\text{min\_l}}(p) + w_{\text{bdy}} \cdot \mathcal{L}_{\text{bdy}}(p) \right), \tag{3.7}$$

where w_{lap}, $w_{\text{min\_len}}$, and w_{bdy} are hyper-parameters. The Laplacian regularizer $\mathcal{L}_{\text{lap}}$ penalizes non-smooth fiber paths:

$$\mathcal{L}_{\text{lap}}(p) \coloneqq s(P)^3 \cdot \sum_{i=2}^{|p|-1} \left|\left| p_i - \frac{p_{i-1} + p_{i+1}}{2} \right|\right|^2, \tag{3.8}$$

where $s(P)$ is a count of the total number of segments in all fiber paths (*i.e.*, $\sum_{p \in P} |p| - |P|$). The reason to apply the $s(P)^3$ multiplier is because the Laplacian regularizer is sensitive to upsampling, which we discuss in § 3.3.4, and this multiplier keeps our Laplacian regularizer scale-invariant. The minimum-length regularizer $\mathcal{L}_{\text{min\_l}}$ penalizes infeasibly-short fiber paths:

$$\mathcal{L}_{\text{min\_l}}(p) \coloneqq \max \left(l_{\min} - \sum_{i=2}^{|p|} ||p_i - p_{i-1}|| \, , 0 \right)^2, \tag{3.9}$$

where $l_{\min}$ is the minimum fiber length that can be printed by the 3D printer. The boundary regularizer $\mathcal{L}_{\text{bdy}}$ penalizes fiber paths outside Ω or too close to $\partial\Omega$:

$$\mathcal{L}_{\text{bdy}}(p) \coloneqq \sum_i \max(d_{\min} - \text{dis}(p_i, \Omega), 0)^2, \tag{3.10}$$

where $\text{dis}(p_i, \Omega)$ measures the distance from p to $\partial\Omega$ (positive for $p_i \in \Omega$, negative otherwise) and $d_{\min}$ is the lower limit of distance from fiber to the boundary.

The next step is to calculate $\frac{\mathrm{d}\mathcal{L}(P)}{\mathrm{d}P}$. Here we apply the adjoint method. Denote the first-order condition of Equation 3.1 as $F(\mathbf{u}, P) = 0$. By the implicit function theorem (under proper regularity conditions) $\mathbf{u}$ can be thought of a function of P, and the

derivative $\frac{\mathrm{d}\mathbf{u}}{\mathrm{d}P}$ is well-defined. Taking the derivative of F with respect to P, we have

$$\frac{\mathrm{d}F}{\mathrm{d}P} = \frac{\partial F}{\partial \mathbf{u}} \frac{\mathrm{d}\mathbf{u}}{\mathrm{d}P} + \frac{\partial F}{\partial P} = 0, \tag{3.11}$$

which leads to

$$\frac{\mathrm{d}\mathcal{L}(P)}{\mathrm{d}P} = -\frac{\partial \mathcal{L}(P)}{\partial \mathbf{u}} \left(\frac{\partial F}{\partial \mathbf{u}}\right)^{-1} \frac{\partial F}{\partial P} + \frac{\partial \mathcal{L}(P)}{\partial P}. \tag{3.12}$$

We implement this end-to-end differentiation automatically using dolfin-adjoint [Mitusch et al., 2019] and PyTorch [Paszke et al., 2019a]. We use the BFGS implementation in SciPy [Virtanen et al., 2020b] to minimize $\mathcal{L}(P)$, and again we iterate every subsequence of the optimized path and select the one that minimizes $\mathcal{L}(P)$. We can repeat the steps in § 3.3.1, § 3.3.2, and § 3.3.3 several times to extract multiple fiber paths.

3.3.4 Coarse-to-fine optimization

To speed up the optimization, we perform multigrid optimization. As described in § 3.3.2, we initially downsample all the fiber paths. Then, for every fiber path p, we insert midpoints between every p_i and p_{i+1} by B-spline interpolation, using SciPy, and optimize all the fiber paths. This process can be repeated several times to generate the final fiber paths for 3D printing.

3.4 Fabrication and experimental setup

In this section, we describe how we manufacture real 3D prints and measure their position-load curves. We use a Markforged Mark Two printer with nylon as the plastic material and carbon fiber as the reinforcing fiber material. We print laminates with a height of 2 mm and 16 layers, from which the 4th, 7th, 10th, and 13th layers are fiber layers. All layers without fiber and regions in fiber layers without fiber are filled with nylon (solid fill).

Figure 3.3: A 3D printed part being tested on a universal testing machine (Instron 600DX), with square nuts in the trapezoidal holes. The machine moves at a speed of 20 mm/min and stops when the object breaks or by a manual stop.

For the 2D shape, we use a 46 mm $\times$ 30 mm rectangle with two rounded isosceles trapezoid holes, the same shape as shown in Figure 3.2. The isosceles trapezoids have two sides of 11 mm and 14 mm and a height of 11 mm, with every corner smoothed by an arc with a radius of 1 mm. We will reuse this shape in § 3.5, § 3.6.3, and § 3.7.

To measure the position-load curve of a print, we insert two square nuts into both its holes and apply tension to them using a universal testing machine (Instron 600DX), as shown in Figure 3.3. The machine is programmed to move at a speed of 20 mm/min until the object breaks or by a manual stop when we believe enough data is collected. A position-load curve is recorded for every print.

3.5 Modulus calculation

In this section, we describe how we determine the effective Young's moduli of nylon and carbon fiber. We print composites with different (baseline) fiber layouts, measure

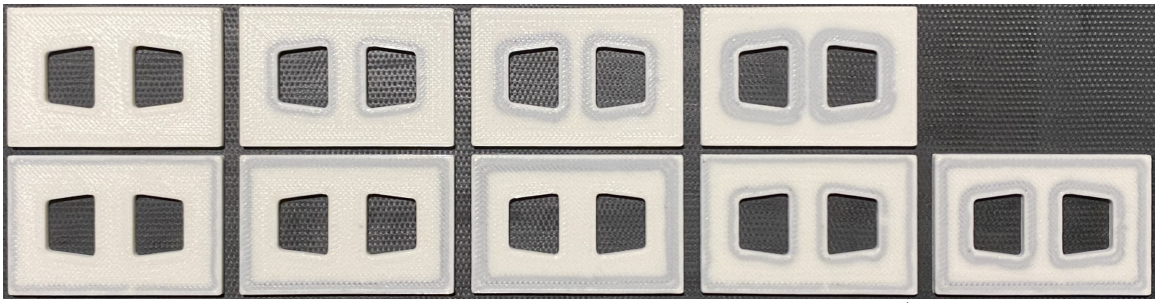


Figure 3.4: Nine different fiber layouts printed for moduli calculation (left to right, top to bottom): no fiber path, 1 to 3 inner rings, 1 to 3 outer rings, 1 to 2 rings for all walls.

their stiffness, then optimize for moduli such that their stiffness in simulation best matches the real-world measurements.

3.5.1 3D prints for testing

We use Eiger to generate nine different layouts of carbon fiber paths: no fiber path, 1 to 3 inner rings, 1 to 3 outer rings, 1 to 2 rings for all walls. To reduce the bias introduced by the non-uniformity of the material, we print all of them in one batch, as shown in Figure 3.4. Due to the variability of the printing process, we print three batches of these nine prints and pick the batch with the best printing quality.

3.5.2 Stiffness measurement

As described in Section 3.4, we test the prints and record their position-load curves (Figure 3.5). Note that the beginning of every curve can be noisy as the part is not perfectly vertical, *etc*. Additionally, a large load can cause the part to buckle out of the 2D plane, which violates our in-plane stress assumption. We therefore measure the position change between a load of 150 N and a load of 300 N for every print, and calculate the stiffness by dividing load change (150 N) by position change (in mm). The results are shown in Figure 3.6, marked as "X". Note that there is a factor of 0.5 when converting stiffness in N/mm to energy in N·mm at 1 mm displacement (*e.g.*,

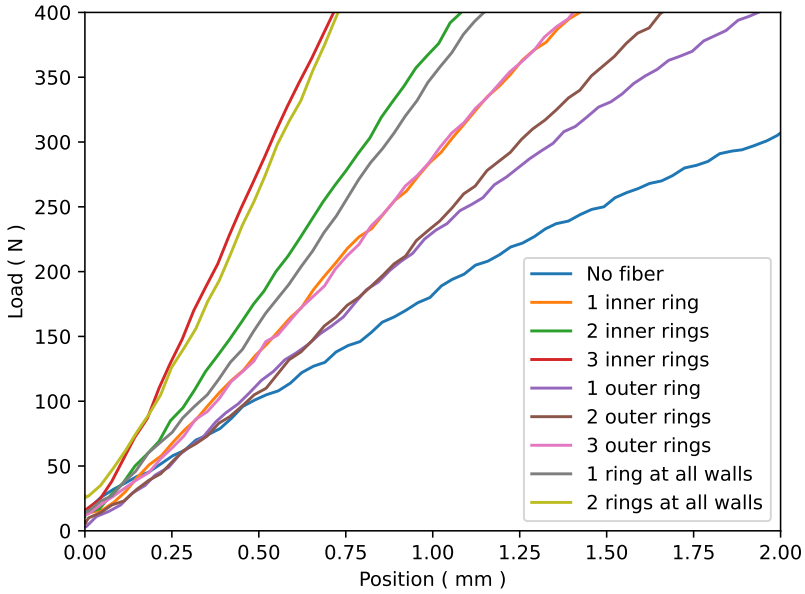


Figure 3.5: Position-load curves recorded from the testing machine. The beginning parts of the curves are noisy due to parts not being perfectly vertical, *etc.*, and a too-large load can cause the object to buckle, violating our in-plane stress assumption. We therefore use the middle parts of the curves, with loads between 150 N and 300 N, to calculate the stiffness.

a stiffness of 500 N/mm corresponds to having strain energy of 250 N·mm at 1 mm displacement).

3.5.3 Simulation and modulus calculation

For each measured data point, we apply Dirichlet boundary conditions corresponding to 1 mm displacement on the two shorter sides of the two holes on the rectangle. We calculate the strain energy of the object, then do a grid search for the values of the moduli of nylon and carbon that minimize the sum of squared distances between measured and simulated stiffness. The search yields moduli of 0.40 GPa for nylon and 20.1 GPa for carbon, with results shown in Figure 3.6. As we can see, the simulation results mostly match the real results, with small residuals relative to the energy.

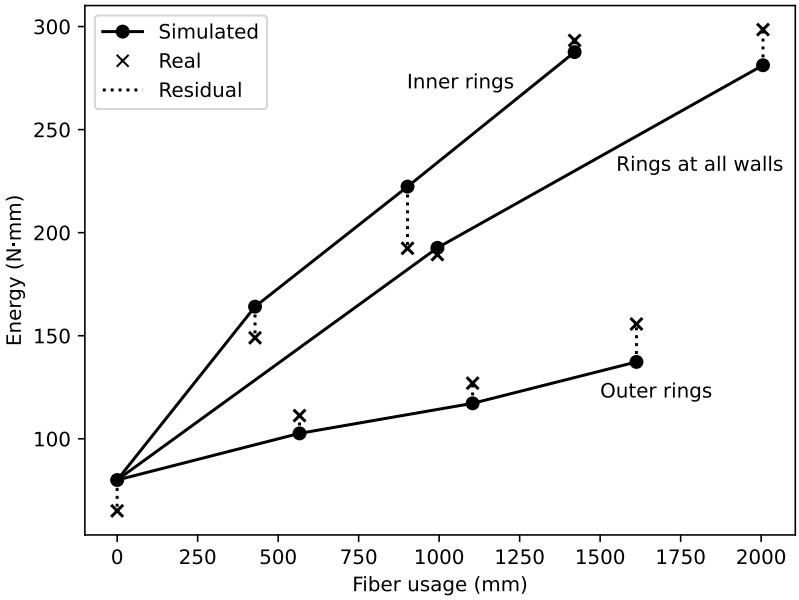
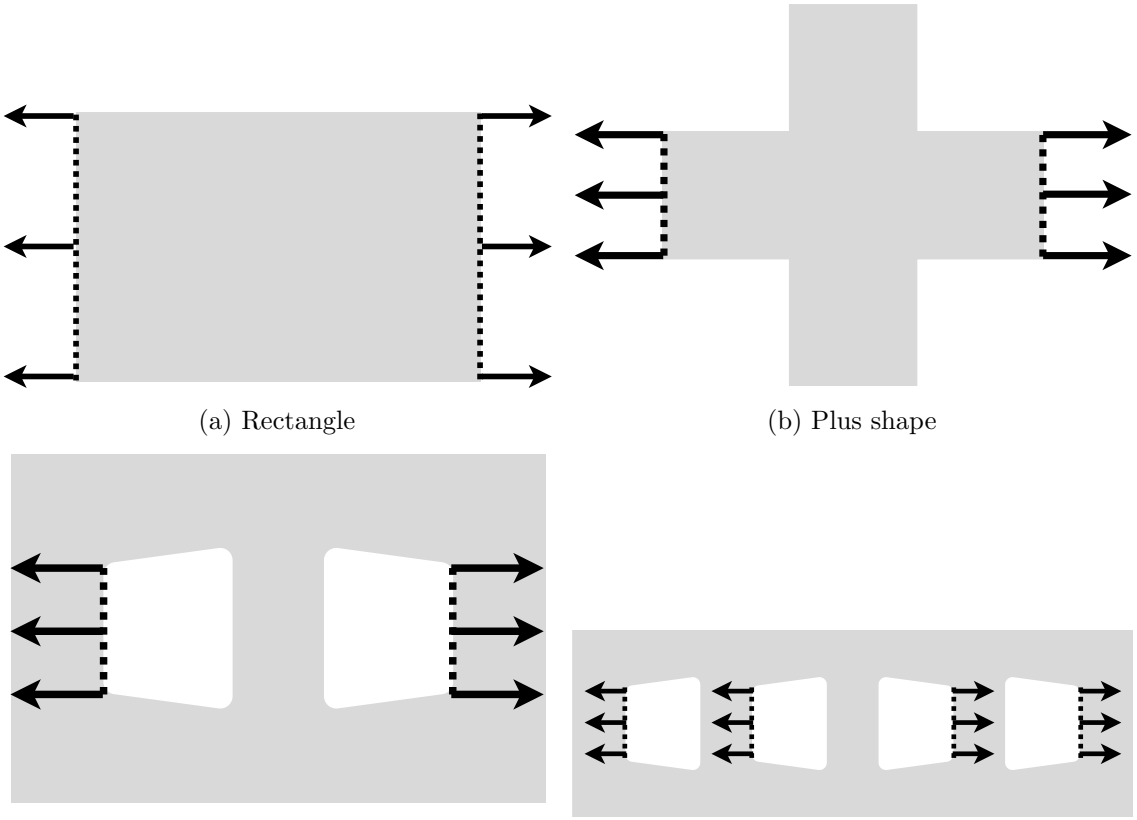


Figure 3.6: We calibrate the moduli of nylon and carbon fiber using nine prints with three different types of fiber layouts: inner rings, outer rings, and rings at all walls. The solid lines connect datapoints sharing the same fiber layout strategy. The real results are marked as "X", the simulated results are marked as small solid circles, and the residuals are shown as dotted lines. The energy numbers are calculated at 1 mm displacement.

3.6 Experiments

In this section, we present detailed evaluations of the performance of our method in both simulation and real experiments on four case studies (§ 3.6.1–§ 3.6.4), then show several additional results in § 3.6.5. We start with two simple shapes—a rectangle and a "plus" shape—then move to more complex shapes: rectangles with two and four holes (Figure 3.7). The first baseline we use is *concentric* fiber rings from Eiger, which have three different types: *inner*, *outer*, and *all walls*. For the next two case studies, to better illustrate the effectiveness of our algorithm on complex shapes and loads, we include two additional baselines: (1) *greedy*, simplifying our algorithm by removing all the optimization components and directly generating results using the greedy algorithm; (2) *field-opt-greedy*, similar to *greedy* but with an additional step of field optimization before running the greedy algorithm. The latter baseline, intended to represent the approach of previous work on fiber orientation optimization (see § 3.2.1),



(a) Rectangle (b) Plus shape

(c) Rectangle with two holes (d) Rectangle with four holes

Figure 3.7: Four shapes we use in our case studies. (a) and (b) are relatively simple shapes, and the loads are applied on the two sides. For (c), a rectangle with two holes, tension is applied on the two shorter sides of the holes. (d) is designed to be a multi-functional rectangle with four holes, and the user can choose one hole from the left two holes and another hole from the right two holes to apply tension.

optimizes a vector field that aligns to the stress direction, with a smoothing regularizer. Additional details about the field optimization can be found in the appendix. We refer to the results from our method as *optimized*. For all the experiments (unless otherwise specified), we use the BFGS optimizer and limit the maximum number of iterations to 500 and a gradient tolerance of 3×10^{-9}. The objective function is set with $w_{\text{lap}} = 1 \times 10^{-8}$, $w_{\text{min\_l}} = 1$, and $w_{\text{bdy}} = 1$.

3.6.1 Case 1: rectangle

In this case study, we show how our algorithm works step by step on a rectangle (45 mm $\times$ 30 mm), with tension applied to its two shorter sides (Figure 3.7a). As we can

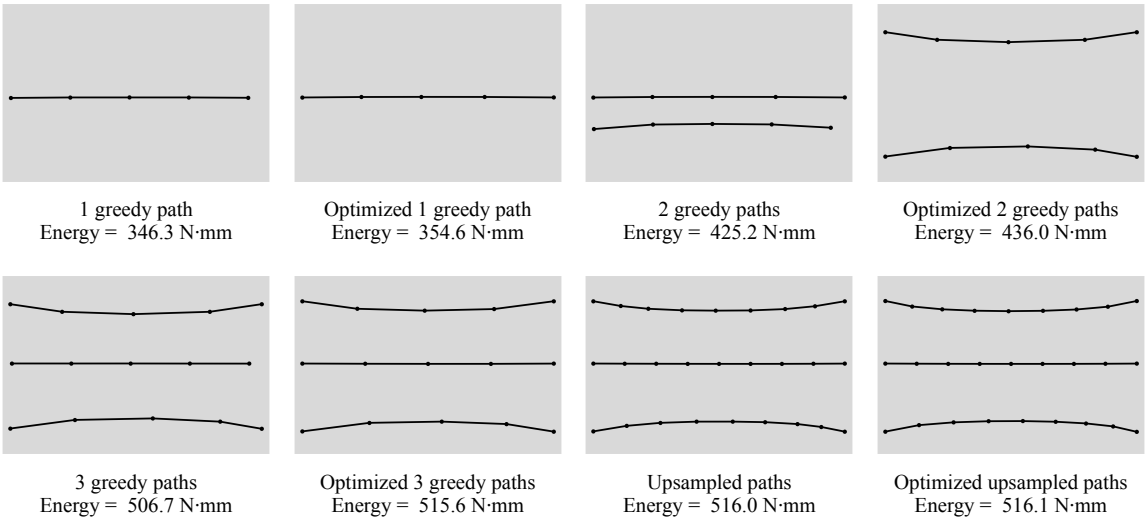


Figure 3.8: Step-by-step visualization of how our method extracts three fiber paths, optimizes them, and performs coarse-to-fine optimization on the *rectangle* shape. In the first row, we extract the first fiber path, optimize it, extract the second fiber path, and optimize both paths. The two paths curve and move up and down after the optimization, respectively. In the second row, we extract a third fiber path, optimize all three paths, upsample them, and finally optimize them. The energy numbers are calculated at 1 mm displacement.

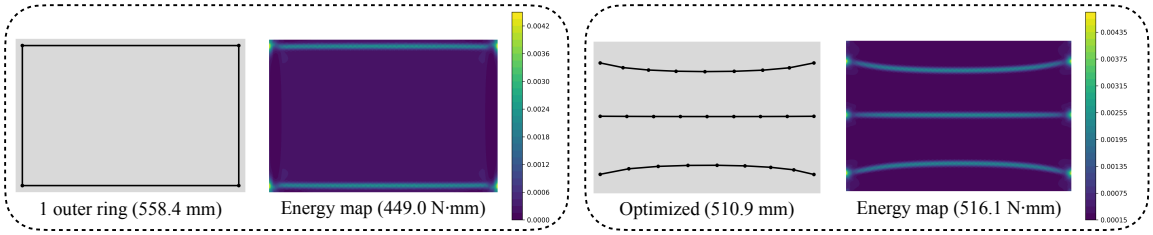


Figure 3.9: Fiber paths and energy maps of *outer* and *optimized* at 1 mm displacement on the *rectangle* shape. We use less fiber while achieving higher energy, as the baseline lays vertical fibers that are much less useful than horizontal fibers.

see in the first row in Figure 3.8, we first greedily extract a fiber path and optimize it. When we add and optimize a second fiber path, the two paths separate and curve. In the second row, we extract a third greedy path and optimize the three paths together. Finally, we double the number of points of both fiber paths and optimize the three paths together. The last step does not help much since the task is relatively simple. As shown in Figure 3.9, for a fixed displacement of 1 mm, our algorithm uses less fiber while achieving higher energy in simulation, compared to 1 *outer* concentric fiber ring, which lays fiber in vertical directions that are much less useful than fibers in horizontal directions.

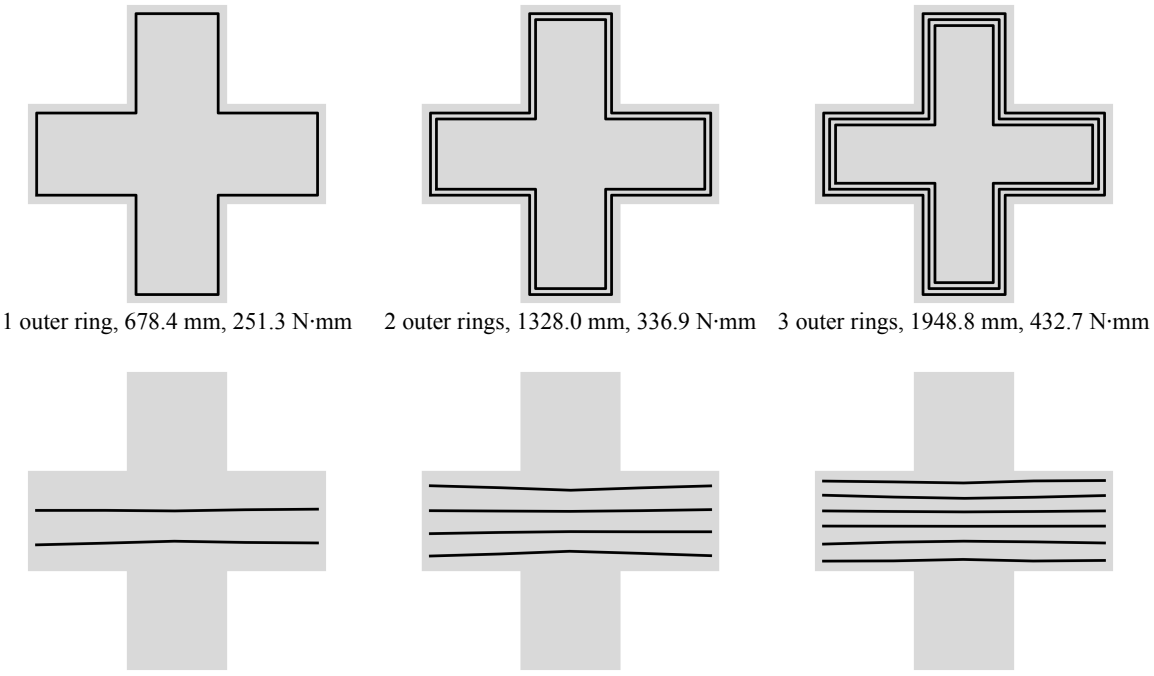


Figure 3.10: Fiber paths, lengths, and strain energy at 1 mm displacement of *outer* and *optimized* on the *plus* shape. With the help of optimization, fiber paths automatically distribute themselves uniformly in the space as we increase the number of fiber paths. By laying slightly bending fibers in horizontal directions, we save fiber while increasing the energy, compared to *outer*, which lays fiber in unrelated regions.

3.6.2 Case 2: "plus" shape

As shown in Figure 3.7b, we use a "plus" shape whose edges are all of length 15 mm, and we apply tension to two sides of the shape. We compare fiber paths of *outer* and *optimized* in Figure 3.10, with three solutions from each strategy. As we can see, *outer* lays fibers in regions of low relevance to the loads applied, in contrast to *optimized* which prioritizes regions of high relevance to the loads. We also observe that the optimization process automatically distributes fiber paths uniformly as we extract more fiber paths. Based on our simulation, for a fixed displacement at 1 mm, the fiber paths of *optimized* improve upon the Pareto front of *outer*, as shown in Figure 3.11.

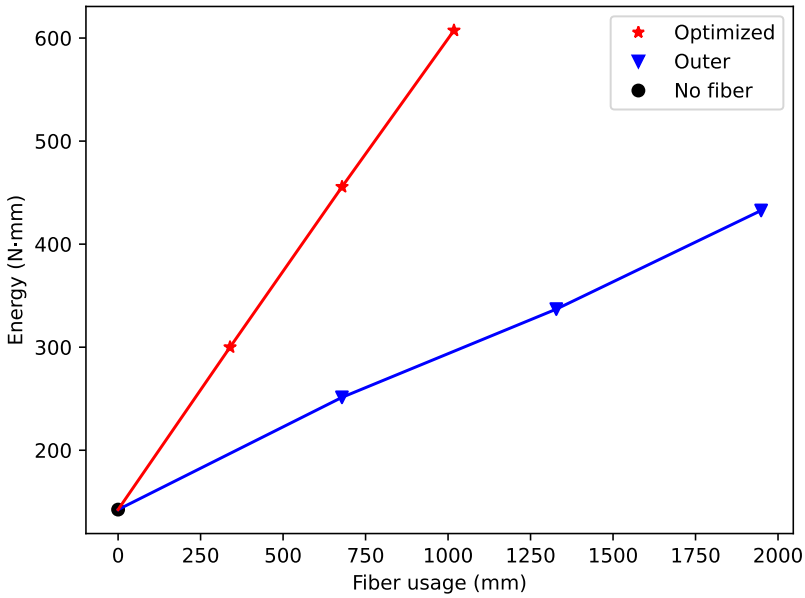


Figure 3.11: Energy-fiber usage plot of *outer* and *optimized* at 1 mm displacement on the *plus* shape. Our method improves over the Pareto front of *outer* by laying fibers according to the external loads.

3.6.3 Case 3: rectangle with two holes

As shown in Figure 3.7c, we also tested a rectangle (46 mm × 30 mm) with two rounded isosceles trapezoid holes, with external forces applied to the two sides of the holes.

Planned fiber paths and simulation results The fiber paths generated from all methods are shown in Figure 3.12. We set the maximum greedy fiber path length so that fiber lengths of *greedy*, *field-opt-greedy*, and *optimized* are comparable. As we can see, the baseline methods use only geometric information; both *greedy* and *field-opt-greedy* generate similar fiber paths along stress directions, but paths from *field-opt-greedy* are smoother; *optimized* wraps fiber paths tightly around the holes while aligned with stress direction, yielding larger strain energy when using a similar amount of fiber.

Real experiment results To evaluate the quality of fiber paths, we perform real-world experiments by applying tension to 3D prints on a universal testing system

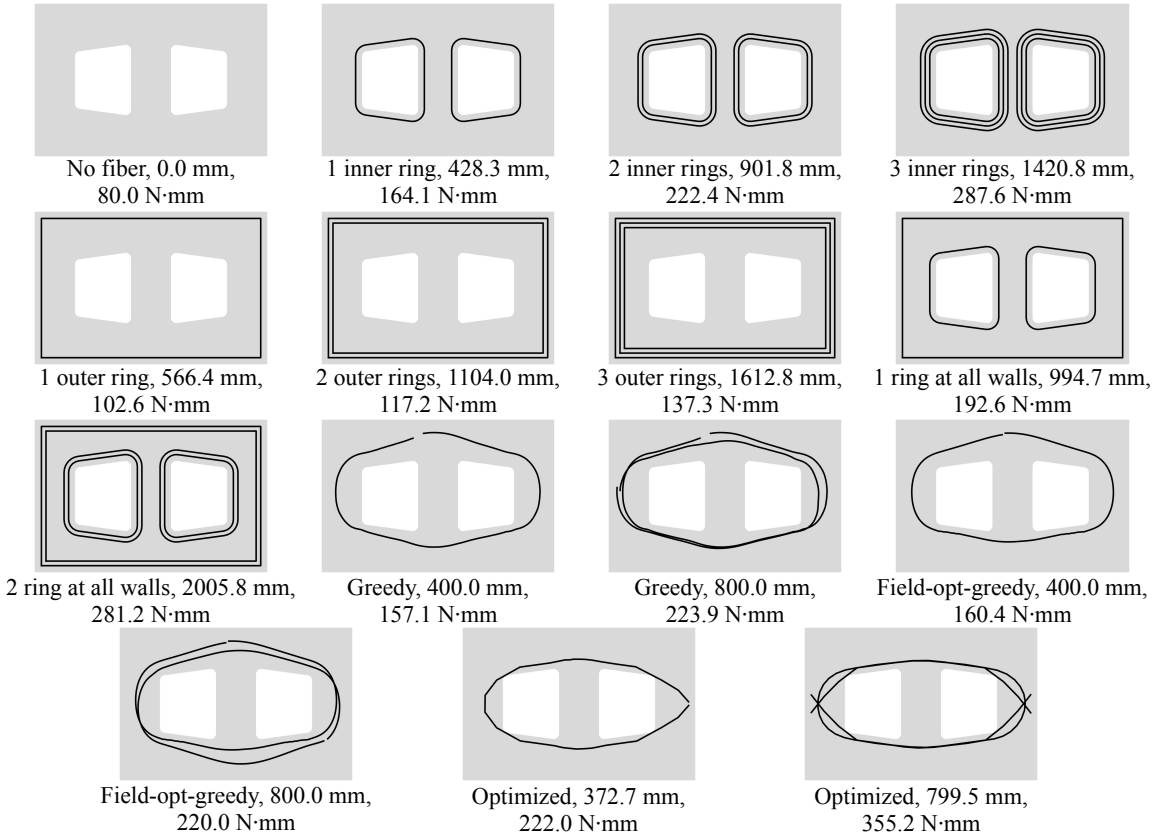
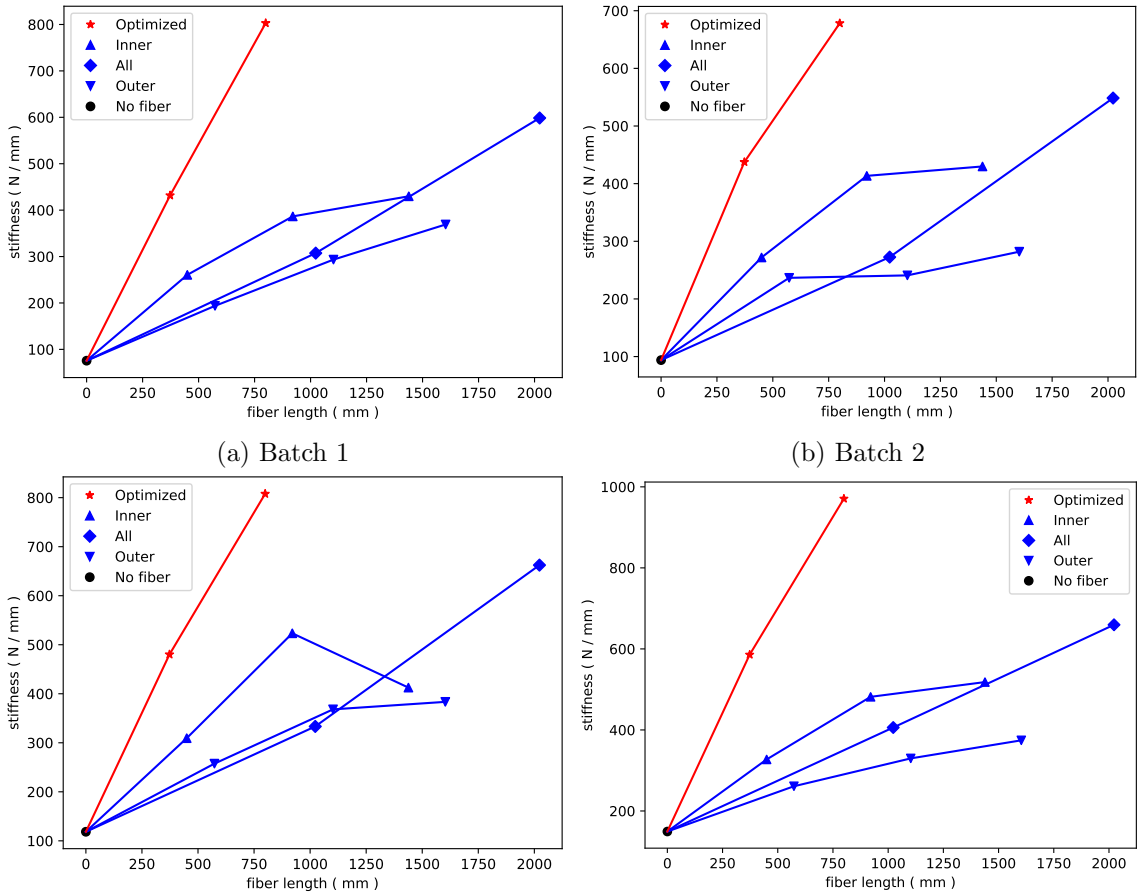


Figure 3.12: Fiber paths, lengths, and strain energy at 1 mm displacement of *inner*, *outer*, *all walls*, *greedy*, *field-opt-greedy*, and *optimized* on the *rectangle with two holes* shape. *field-opt-greedy* provides similar but smoother paths compared to *greedy*, and *optimized* provides more effective fiber paths. Note that there is a factor of 2 when converting the strain energy in N·mm at 1 mm displacement to stiffness in N/mm which we will use in real experiments (*e.g.*, strain energy of 250 N·mm at 1 mm displacement corresponds to having a stiffness of 500 N/mm).

(600DX from Instron). Due to the limited space on the printer bed, two sets of comparisons are performed separately: (1) *inner*, *outer*, and *all walls vs. optimized*; (2) *greedy* and *field-opt-greedy vs. optimized*. We thus printed eight batches, four for each set of comparisons. Again, as in Section 3.4, we measure the stiffness of a print by calculating the slope of its position-load curve, picking two points that have loads of 150 N and 300 N. The results of *inner*, *outer*, and *all walls vs. optimized* are shown in Figure 3.13. As we can see, our algorithm consistently provides significantly higher stiffness than the concentric baselines when using a similar or lower amount of fiber. Note that the fiber lengths may have slight discrepancies between simulation



(a) Batch 1

(b) Batch 2

(c) Batch 3

(d) Batch 4

Figure 3.13: **Real** stiffness-fiber length plots of *inner*, *outer*, *all walls*, and *optimized* on the *rectangle with two holes* shape, measured between 150 N and 300 N (4 batches). By laying fibers tightly around the holes, *optimized* consistently performs better than all others.

and real-world experiments since they are from different path generation algorithms (one from our re-implementation of Eiger, another from Eiger directly). The results of *greedy* and *field-opt-greedy vs. optimized* are shown in Table 3.1. Again, our algorithm consistently improves the stiffness over the two baselines while using a similar or lower amount of fiber.

3.6.4 Case 4: rectangle with four holes

As shown in Figure 3.7d, we also tested a rectangle (84 mm × 28 mm) with four rounded isosceles trapezoid holes. We design the shape to be multi-functional—if we label the holes from 1 to 4 from left to right, we assume the user uniformly chooses

Table 3.1: **Real** (measured) stiffness of *greedy* (g), *field-opt-greedy* (f), and *optimized* (o) on the *rectangle with two holes* shape, measured between 150 N and 300 N (4 batches). *Optimized* performs consistently better than the baselines when using a similar or less amount of fiber.

Stiffness (N/mm)	Solution 1			Solution 2		
	g	f	o	g	f	o
Batch 1	490.1	574.6	625.5	745.0	741.3	992.6
Batch 2	584.3	692.0	756.0	801.0	741.3	985.2
Batch 3	483.5	485.0	656.7	671.6	603.4	953.5
Batch 4	491.7	481.7	603.0	670.0	670.9	970.4
Average	512.4	558.3	**660.3**	721.9	689.2	**975.4**
Length (mm)	400.0	400.0	372.7	800.0	800.0	799.5

one of the four settings: 1) hole 1 and hole 3; 2) hole 1 and hole 4; 3) hole 2 and hole 3; 4) hole 2 and hole 4. To support this multi-functional shape, we simulate all four cases and calculate the average strain energy.

The fiber paths from all the methods are shown in Figure 3.14. Again, both *greedy* and *field-opt-greedy* produce fibers along stress directions with fiber paths from *field-opt-greedy* being slightly smoother. *Optimized* lays the first fiber over all holes and lays the second fiber around the middle two holes, due to the multi-functional nature of the shape. The energy-fiber usage plot is shown in Figure 3.15, where *optimized* improves upon the Pareto front of every baseline.

3.6.5 Results on additional shapes

In this subsection, we provide results from our method and baselines on several additional shapes. The shape designs are inspired by sketches from SketchGraphs [Seff et al., 2020], a large-scale dataset of sketches of real-world CAD models, as well as shapes from existing works [Shafighfard et al., 2019, Ma et al., 2022]. We use a Laplacian regularizer weight $w_{\text{lap}} = 5 \times 10^{-7}$, and the results are shown in Figure 3.16, with every dotted line a Dirichlet boundary condition. For the first shape, all methods use a similar amount of fiber but *optimized* achieves much higher energy than others.

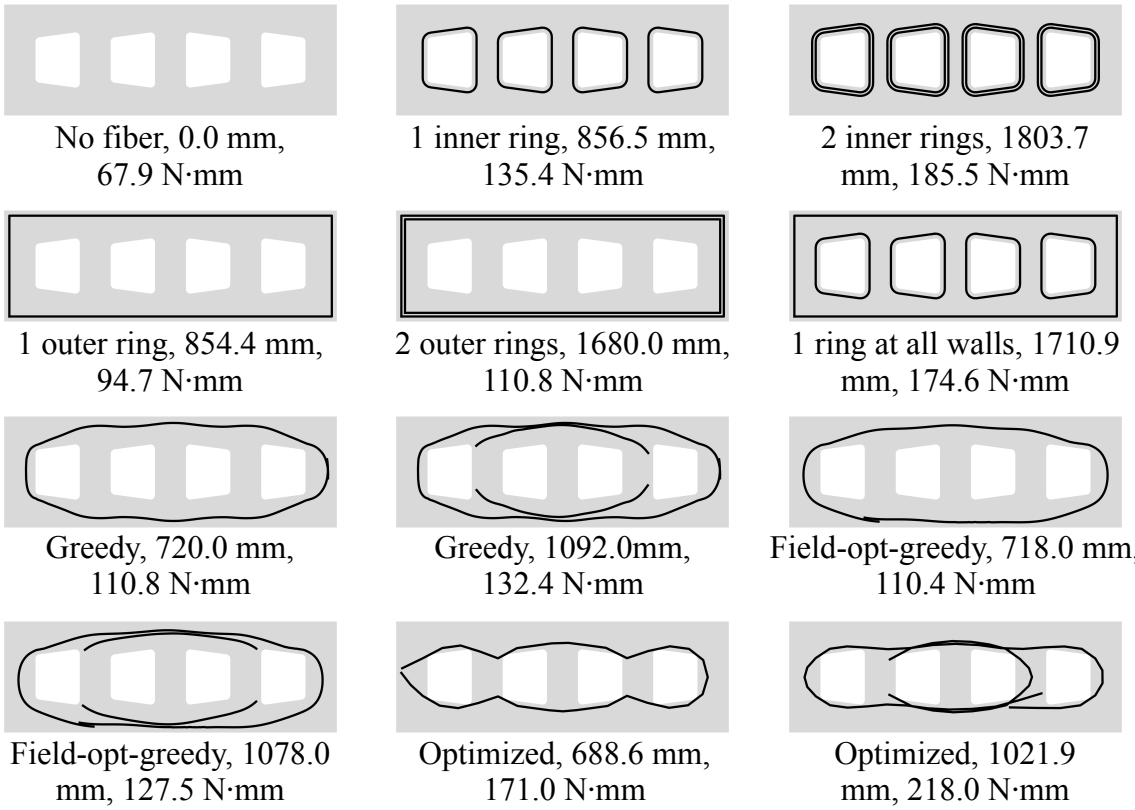
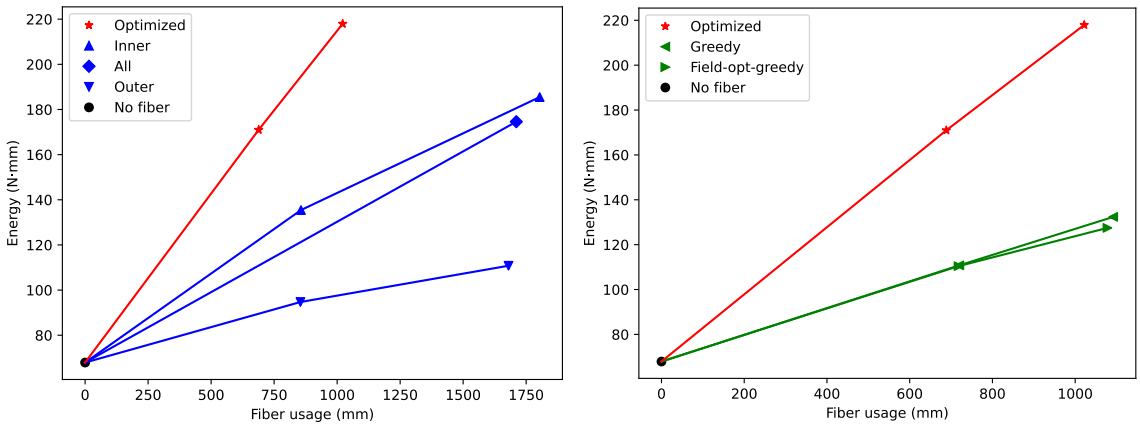


Figure 3.14: Fiber paths, lengths, and strain energy at 1 mm displacement of *inner*, *outer*, *all walls*, *greedy*, *field-opt-greedy*, and *optimized* on the multi-functional *rectangle with four holes* shape. Similarly, *greedy* and *field-opt-greedy* lay fibers along stress directions, and *field-opt-greedy* provides slightly smoother fiber paths. *Optimized* lays the first fiber over all the holes, and the second fiber around the middle two holes, with paths tightly around the holes.

For the second shape, *optimized* uses a similar amount of fiber as *concentric*, less fiber than *greedy* and *field-opt-greedy* but achieves higher energy. For the third shape, *optimized* achieves comparable energy as *concentric* but saves approximately 70% of fiber. Compared to *greedy* and *field-opt-greedy*, *optimized* achieves much higher energy while using slightly more fiber. For the fourth and fifth shapes, *optimized* uses less fiber or comparable fiber as other baselines while achieving significantly higher energy.



(a) Concentric *vs. optimized* (b) Greedy-based *vs. optimized*

Figure 3.15: Energy-fiber usage plot (at 1 mm displacement) of all methods on the *rectangle with four holes* shape. The comparison between concentric fiber rings and *optimized* is shown on the left, and the comparison between greedy-based baselines and *optimized* is shown on the right. *Optimized* improves the Pareto front of all the baselines by laying fibers tightly around the holes.

3.7 Ablation studies

Our algorithm without optimization has been studied in Section 3.6 as the *greedy* baseline. In this section, we study the effects of removing two other components of our method: the Laplacian regularizer and the multi-resolution optimization, using the shape *rectangle with two holes* (Figure 3.7c).

3.7.1 Ablation study of the Laplacian regularizer

As both the minimum-length regularizer and the boundary regularizer are intuitively necessary for fiber paths to be long enough for printing purposes and within the object boundary, we study the effect of removing the Laplacian regularizer from the optimization. We run our algorithm with the same hyper-parameter setting except for $w_{\text{lap}} = 0$. We extract one fiber path and upsample for one time. As shown in Figure 3.17, the optimizer successfully optimizes the low-resolution path as the number of points is still small (Figure 3.17a), but introduces jagged results with

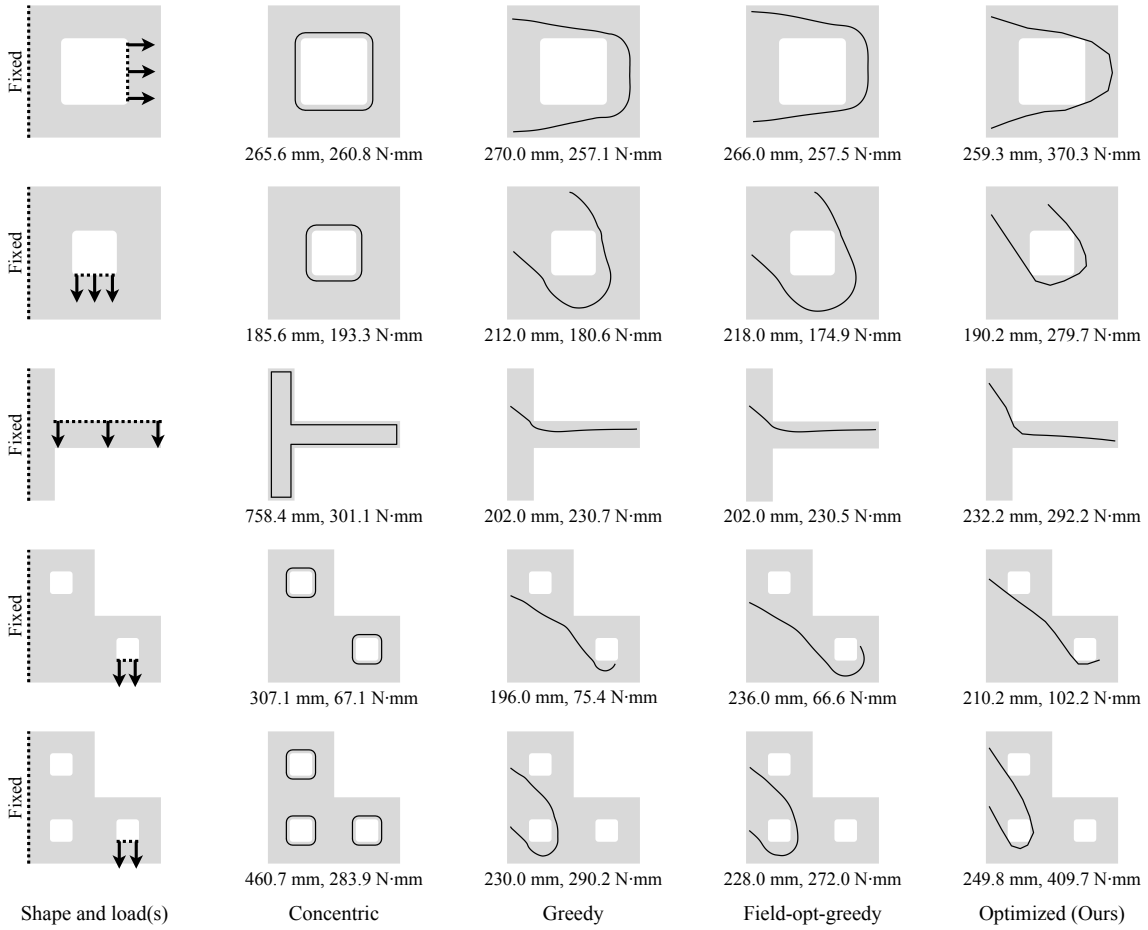
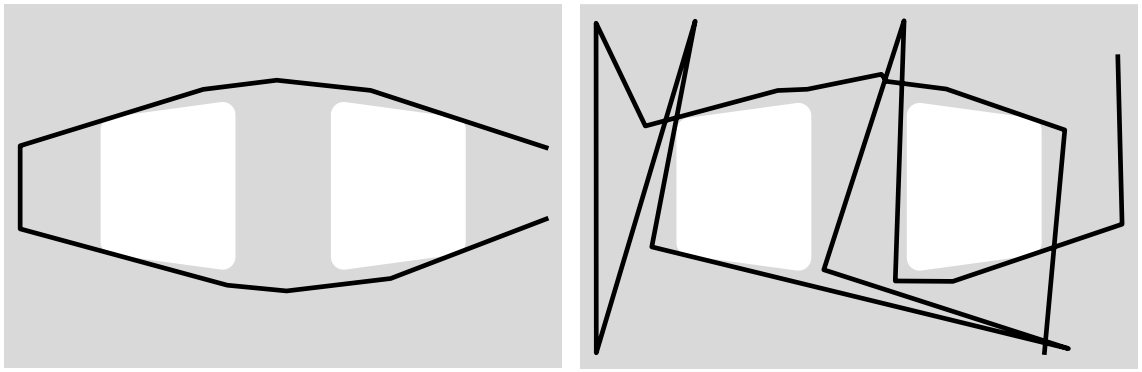


Figure 3.16: Fiber paths and energy at 1 mm displacement of all methods on additional shapes. Every dotted line indicates a Dirichlet boundary condition. For the first shape, *optimized* achieves significantly higher energy while using slightly less fiber than all baselines. For the second shape, *optimized* achieves higher energy while using a similar amount of fiber as *concentric* and less fiber than *greedy* and *field-opt-greedy*. For the third shape, *optimized* saves approximately 70% of fiber usage while achieving similar energy as *concentric*. It also achieves much higher energy than *greedy* and *field-opt-greedy* while using slightly more fiber. For the last two shapes, compared to other baselines, *optimized* achieves significantly higher energy while using a less or comparable amount of fiber.

more degrees of freedom (Figure 3.17b), demonstrating the need for some form of regularization.

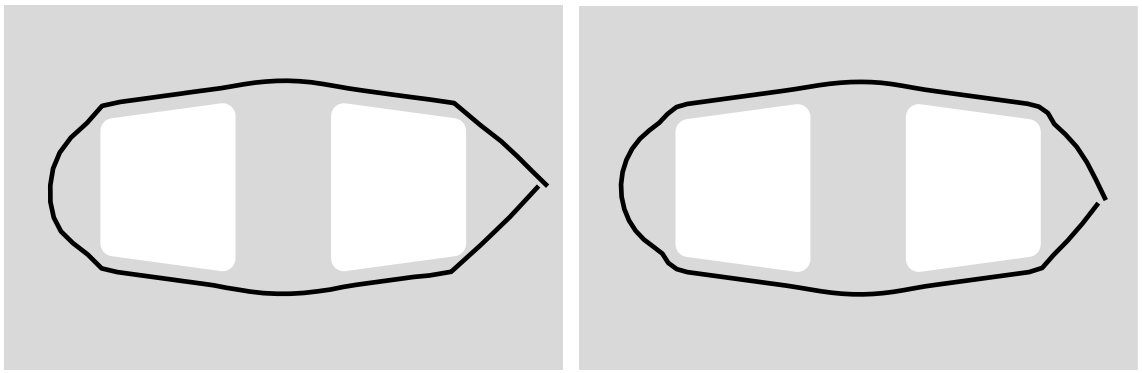
3.7.2 Ablation study of multi-resolution optimization

In this subsection, we study how the multi-resolution approach speeds up the optimization process. For the multi-resolution case, we extract one fiber path, downsample its



(a) Optimized low-resolution path (b) Optimized upsampled path

Figure 3.17: Optimization results with the Laplacian regularizer disabled. As shown on the left, the optimizer successfully optimizes the low-resolution path. It fails to optimize the fiber path after upsampling, as shown on the right.



(a) Multi-res, 293 s, 195.8 N·mm (b) Single-res, 485 s, 190.0 N·mm

Figure 3.18: Running time and the strain energy at 1 mm displacement for both single-resolution and multi-resolution optimization. In this case, we save approximately 40% of running time by multi-resolution optimization.

resolution by a factor of 20, optimize it, and upsample and optimize it three times, with every optimization limited to 100 iterations. For the single-resolution case, we also extract one fiber path, downsample its resolution by a factor of 2, optimize it and limit the maximum number of optimization iterations to 400. For a fair comparison, we use the same random seed for both cases when sampling starting points of the greedy path extraction algorithm. As shown in Figure 3.18, both cases get similar fiber paths with similar strain energy, but multi-resolution optimization reduces the running time by approximately 40%.

3.8 Discussion

In this work, we studied the task of fiber path planning in 3D printing for given external loads, aiming at maximizing the stiffness. We proposed an end-to-end optimization approach that optimizes regularized object stiffness directly to the fiber layout, rather than an intermediate fiber orientation field, with the help of the adjoint method and automatic differentiation. We perform planning by extracting fiber paths using a greedy algorithm that lays fiber paths along stress directions, followed by coarse-to-fine optimization. To apply our method, we first measure the effective moduli of plastic and fiber by manufacturing and testing real 3D prints. We then study our method with three baselines on four case studies and several additional shapes. The first baseline is concentric fiber rings from Eiger, a leading digital manufacturing software package developed by Markforged. The second baseline is our method with the optimization part removed, producing fiber paths from the greedy path extraction algorithm. The third baseline includes a fiber field optimization part which smooths the stress field before using it in the greedy algorithm. We demonstrated that, both in simulation and real experiments, our method could generate shorter fiber paths while achieving greater stiffness (*i.e.*, we improved the Pareto front). We also studied the effects of removing the Laplacian regularizer and the multi-resolution optimization, showing the Laplacian regularizer is necessary for the optimization to be stable and multi-resolution optimization helps reduce the running time.

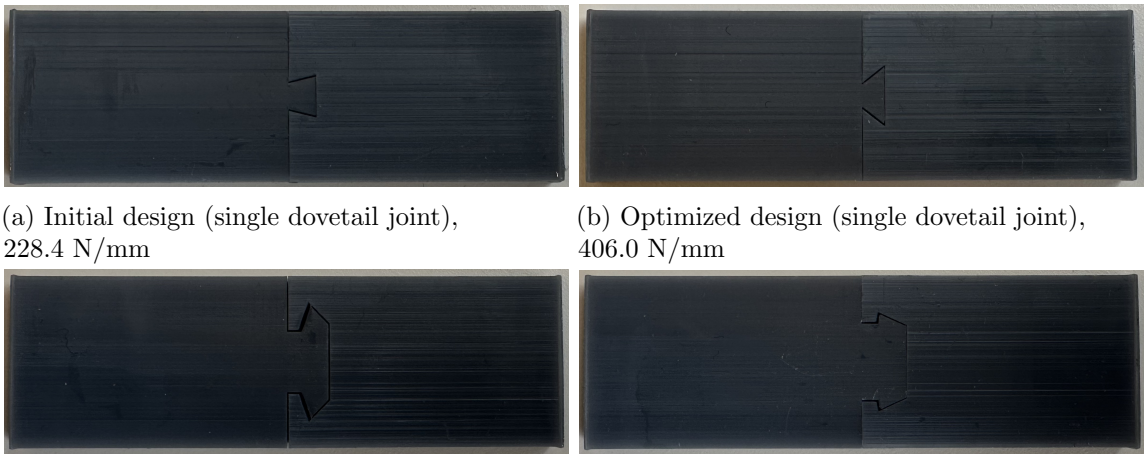
We would also like to mention some limitations of our method. First, our simulation simplifies the task by assuming linear elasticity, restricting to in-plane stress, and treating both plastic and fiber as isotropic materials with different Young's moduli and identical Poisson's ratio. Lifting these assumptions would introduce greater mathematical complexity, but would require no conceptual changes to our approach. Additionally, the planning is not performed in real time. For example, to plan fiber paths for the shape *rectangle with two holes*, our method uses 10 minutes and 18

minutes to generate the two studied solutions, respectively. Relative to the time required to design and print a part, this represents only a small increase. In addition, the hyper-parameters may have to be tuned when the task changes. For example, if we switch to a much larger shape, the scale of strain energy and the lengths of fiber paths will change. We may have to adjust the weight of the Laplacian regularizer, balancing the optimization stability and the variety of fiber paths, though this is usually easy to tune in a few tries. Lastly, as our optimizer is gradient-based, the optimization may be trapped in a local minimum. Thus a good initialization is important for our method, and we may have to sample greedy paths several times to obtain a good one.

Chapter 4

Gradient-Based Dovetail Joint Shape Optimization for Stiffness

In manufacturing, people manufacture an object in several parts for various reasons, including the size limit of the machine, the complex structure of the shape, *etc.*, after which they can be assembled using joints. In this project, we study the task of dovetail-joint shape optimization for stiffness. Instead of using search algorithms or gradient-free optimizers, we use a gradient-based optimizer. To perform the optimization, a simulator is needed to model the contact between the two parts of a joint, and gradients on shape parameters are needed for efficient optimization. To address these challenges, we propose to perform contact simulations using a penalty approach alternatively on two sides of the joint, and use the adjoint method to calculate gradients with respect to the shape parameters. We test our method by optimizing the joint shapes in three different joint shape spaces, and evaluate optimized joint shapes in both simulation and real tests. The experiments show that optimized joint shapes achieve higher stiffness, both synthetically and in real tests.



(a) Initial design (single dovetail joint), 228.4 N/mm

(b) Optimized design (single dovetail joint), 406.0 N/mm

(c) Initial design (complex dovetail joint), 518.1 N/mm

(d) Optimized design (complex dovetail joint), 1036.3 N/mm

Figure 4.1: Initial and optimized designs of single and complex dovetail joints with average stiffness measured over three batches and external forces applied on the two sides. (a) (c): initial (randomly chosen) single and complex dovetail joints. (b) (d): optimized single and complex dovetail joints, which provide greater stiffness.

4.1 Introduction

Manufacturing is one of the essential activities of humans [Groover, 2020]. It involves the design and assembly of different parts of an object, which is a standard and common practice in the industry. For example, a wooden record crate can be made and then assembled from five wood pieces connected by dovetail joints. Dovetail joint [Ruiz et al., 1984], as shown in Figure 4.1, is a connector commonly used in woodworking joinery, turbine blades, 3D printing, *etc*. It is, therefore, an important question to ask: what is the optimal dovetail joint shape?

In this work, we study the problem of efficiently finding the optimal dovetail joint design that maximizes the stiffness, given a dovetail joint design space and specified external loads. We choose to study the dovetail joint for simplicity reasons—it is a representative type of joint, and the method we develop should work for any joint shape that is piecewise linear. To find the optimal design, we formalize the task as an optimization problem, with the stiffness as a function of design parameters that characterize the shape of the dovetail joint in the design space, calculate the gradient

with respect to the design parameters, and directly optimize them using gradient descent. However, optimizing the shape of a dovetail joint is challenging. First, we need a simulator that can simulate the deformation of the joint given external loads, while considering the contact between two parts of the joint. Second, we need to compute the gradient of stiffness with respect to the design parameters.

To address the first challenge, we alternatively perform contact simulations using the penalty approach [Huněk, 1993] on two sides of a joint. In every iteration, we compute the deformation of one side of the joint while considering the other side as rigid, and we penalize the elastic side penetrating the rigid side. By iteratively and alternatively applying this approach on the two sides, we have a reasonably accurate simulator for joint deformation simulation. As the simulation can be written as PDEs, the stiffness maximization task is now a PDE-constrained optimization [De los Reyes, 2015]. We thus use the adjoint method [Errico, 1997, Cao et al., 2003] to calculate the gradients of stiffness with respect to design parameters, which addresses the second challenge.

To demonstrate the effectiveness of our method, we test it both in simulation and in real tests. We first perform a gradient check by computing the gradients of stiffness with respect to mesh vertex coordinates, both using the adjoint method and the finite difference method, and compare the difference between these two. We then run optimization on three different dovetail joint design spaces, each using two different initial designs. Experiments show that optimized designs provide greater stiffness compared to the initial ones, both in simulation and in real tests by 3D printing them. We finally study the sensitivity of the optimization result with respect to material parameters by optimizing using different Poisson's ratios. Experiments show that the optimization result is not sensitive to different Poisson's ratios.

4.2 Related work

4.2.1 Dovetail joint shape optimization

Researchers optimize the shapes of dovetail joints using different approaches. The most straightforward approach is simply testing different design parameters. Kogo et al. [2002] and Kogo et al. [2019] conducted tensile and shear tests on carbon-carbon composite dovetail joints with different dovetail angles. Miyauchi et al. [2006] tested wooden dovetail joints with different inclinations and base widths. Jeong et al. [2012] tested different wooden dovetail joints for maximum tension load. Estenlund et al. [2022] studied the dovetail design for mounting coils on rotors by building a simulator and enumerating different dovetail angles. Another approach for dovetail optimization is applying a gradient-free optimizer on a simulator. Hu et al. [2022] tested dovetails with different combinations of tenon length, width, thickness, and angle, and studied the effect of each design parameter using the linear model. Yang et al. [2018] used commercial FEM software for simulation and optimized dovetail shapes for aero-engines using several different gradient-free optimizers. Some researchers build (differentiable) surrogate models for the simulators and optimize the surrogate models. For example, Hahn and Cofer IV [2012] first optimized design parameters using a surrogate model and further used other gradient-free optimizers to optimize design parameters that are sensitive. In this work, instead of enumerating, using gradient-free optimizers or surrogate models, we directly optimize dovetail design parameters on the FEM simulator, using the adjoint method to compute the gradients.

4.2.2 PDE-constrained optimization

PDE-constrained optimization is a type of constrained optimization problems whose constraints can be written as PDEs. See De los Reyes [2015] for a textbook. Specifically, the objective function depends on both the design variable and the state variable, and

the constraints between them can be written as PDEs. There are mainly two different types of approaches [Herzog and Kunisch, 2010]: *black-box* and *all-at-once*. *all-at-once* treats the design variable and the state variable as independent variables during optimization, and researchers may use algorithms including Sequential quadratic programming (SQP) Boggs and Tolle [1995] for this approach. *black-box* treats only the design variable as the independent variable during optimization, and researchers may use the adjoint method to calculate the total derivative of the objective with respect to the design variable. See Givoli [2021] for a tutorial on the adjoint method. In this work, we formalize the dovetail joint shape optimization problem as a PDE-constrained optimization task and use the adjoint method with gradient descent to solve it.

4.3 Method

The pipeline of our approach is shown in Figure 4.2. For an initial set of shape parameters, we first calculate the displacement of the corresponding dovetail joint given a fixed load size on the two sides. We simulate by alternatively applying a contact solver with the penalty approach—specifically, we alternatively solve one side of the joint, assuming the other side is rigid. We then use the adjoint method to calculate the derivative of the displacement with respect to design parameters and use line search to find the optimum step size for optimization using gradient descent.

4.3.1 Alternating penalty contact simulator

In this subsection, we describe how we simulate the deformation of a specific joint given external loads. Assuming the material is isotropic, we use the linear elastic model from Langtangen and Logg [2017]. Denoting the body as Ω, we have the

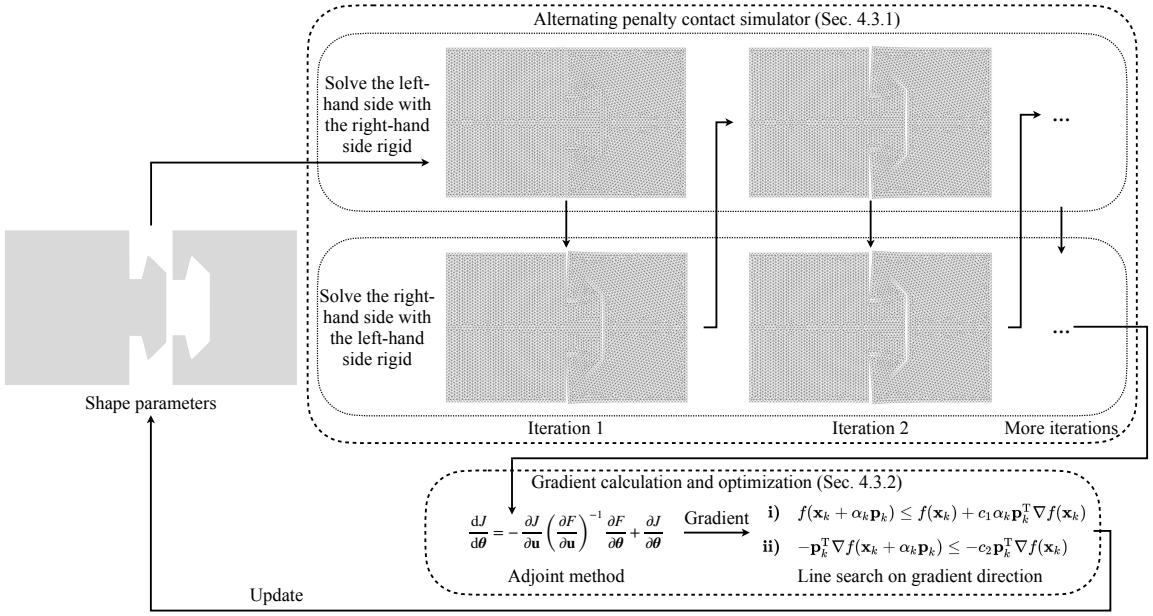


Figure 4.2: Given a set of shape parameters, we first use the alternating penalty contact simulator (§ 4.3.1) to calculate the deformation of the joint, and then use the adjoint method to calculate the gradient and use line search to find the step size for gradient descent (§ 4.3.2).

equations governing the deformation on Ω as

$$\begin{aligned} -\nabla \cdot \boldsymbol{\sigma} &= f, \\ \boldsymbol{\varepsilon} &= \frac{1}{2}\big(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\intercal}\big), \\ \boldsymbol{\sigma} &= \lambda \mathrm{tr}(\boldsymbol{\varepsilon}) I + 2\mu \boldsymbol{\varepsilon}, \end{aligned} \tag{4.1}$$

where $\boldsymbol{\sigma}$ is the stress tensor, f is the body force (0 in our case), $\boldsymbol{\varepsilon}$ is the strain tensor, $\mathbf{u}$ is the displacement vector, λ and μ are Lamé parameters, and I is the identity matrix. Under the assumption of plane stress, we have $\lambda = \frac{E\nu}{1-\nu^2}$ and $\mu = \frac{E}{2(1-\nu^2)}$ where E is Young's modulus of the material (we set it to 1 GPa), and ν is the Poisson's ratio of the material (we set it to 0.4). Equivalently, we are minimizing the total potential energy Π which is defined as Alnæs et al. [2015]:

$$\Pi \coloneqq \int_{\Omega} \frac{1}{2} \boldsymbol{\varepsilon} : \boldsymbol{\sigma} \mathrm{d}A - \int_{\Omega} f \cdot \mathbf{u} \mathrm{d}A - \int_{\partial\Omega} T \cdot \mathbf{u} \mathrm{d}x, \tag{4.2}$$

where the colon is the dot product between tensors and T is the traction force. For reasonable deformations, we set T to 0.001 GPa for single dovetail joints and 0.003 GPa for complex and double dovetail joints (see Figure 4.3). We apply equal traction forces on two sides of the joint as Neumann boundary conditions.

One difficulty is to model the contact between the two sides of the joint. As Bleyer [2018], we use a penalty approach—solving one side while assuming the other side is rigid, and we apply penalty directly on the displacement field $\mathbf{u}$. Denote two sides of the joint as Ω_L and Ω_R, and consider the case that we want to solve the deformation on Ω_L while considering Ω_R rigid. For simplicity and the piecewise-linearity nature of the boundary of dovetail joints, we fit lines to all the contacting edges (see § 4.4.1 for more details) of Ω_R and penalize $\mathbf{u}$ on Ω_L if collide with the fitted lines. We have our penalized total potential energy Π_L for Ω_L as

$$\Pi_L \coloneqq \int_{\Omega_L} \frac{1}{2}\boldsymbol{\varepsilon} : \boldsymbol{\sigma} \mathrm{d}A - \int_{\Omega_L} f \cdot \mathbf{u} \mathrm{d}A - \int_{\partial\Omega_L} T \cdot \mathbf{u} \mathrm{d}x + w_{\mathsf{pen}} \cdot \int_{\partial\Omega_L} \mathtt{softplus}^2(-\mathtt{sdf}(\mathbf{u}; \Omega_L, \Omega_R)) \mathrm{d}x, \tag{4.3}$$

where w_{pen} is the weight of the penalization term, which we set to 1, $\mathtt{softplus}(x) = (\ln(1 + \exp(kx))/k)^2$ and k is a scale factor that we set to 50, $\mathtt{sdf}$ is the signed distance function and $\mathtt{sdf}(\mathbf{u}; \Omega_L, \Omega_R)$ measures the signed distance from the deformed left-hand side to the fitted lines of the deformed right-hand side (positive if outside and negative if inside). We create meshes with a mesh step size of 0.5 mm using pygmsh [Schlömer] and implement the simulator using FEniCS [Alnæs et al., 2015]. We alternatively solve the two sides for four iterations, as it is usually enough for $\mathbf{u}$ to converge. Note that the simulator works for any piecewise linear joints, not only for dovetail joints.

4.3.2 Gradient calculation and optimization

In this subsection, we describe, given the simulator, how we optimize the shape design parameters θ. With the simulator, we can compute the corresponding displacement field $\mathbf{u}(\theta)$. We define the length change of the joint $d(\mathbf{u}(\theta))$ as the difference of average

displacement in the horizontal direction between the left and right edges of the joint. Note that maximizing the stiffness is equivalent to minimizing the displacement given fixed traction. We define the optimization objective function $\mathcal{L}(\theta)$ as

$$\mathcal{L}(\theta) \coloneqq d(\mathbf{u}(\theta)) + w_{\text{min\_l}} \cdot \mathcal{L}_{\text{min\_l}}(\theta) + w_{\text{min\_w}} \cdot \mathcal{L}_{\text{min\_w}}(\theta), \tag{4.4}$$

where $\mathcal{L}_{\text{min\_l}}(\cdot)$ and $\mathcal{L}_{\text{min\_w}}(\cdot)$ are regularizers, and $w_{\text{min\_l}}$ and $w_{\text{min\_w}}$ are weights for the regularizers, both of which we set to 1. The minimum contact length regularizer penalizes too short edges that are contacting:

$$\mathcal{L}_{\text{min\_l}}(\theta) \coloneqq \sum_{l \in \texttt{contact}(\theta)} \left(\max(\texttt{min\_len} - |l|, 0)\right)^2, \tag{4.5}$$

where $\texttt{contact}(\theta)$ is the set of all contacting edges (see § 4.4.1 for more details), $|\cdot|$ measures the length of an edge, and we set `min_len` to 1.5 mm. The minimum width regularizer penalizes too small width of the joint:

$$\mathcal{L}_{\text{min\_w}}(\theta) \coloneqq \left(\max(\texttt{min\_width} - \texttt{width}(\theta), 0)\right)^2, \tag{4.6}$$

where $\texttt{width}(\cdot)$ measures the width of a joint (see § 4.4.1 for more details), and we set `min_width` to 3.5 mm.

To calculate the gradient of the objective function with respect to the design parameters ($\frac{\mathrm{d}\mathcal{L}(\theta)}{\mathrm{d}\theta}$), we use the adjoint method, which provides the following result:

$$\frac{\mathrm{d}\mathcal{L}(\theta)}{\mathrm{d}\theta} = -\frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{u}} \left(\frac{\partial F}{\partial \mathbf{u}}\right)^{-1} \frac{\partial F}{\partial \theta} + \frac{\partial \mathcal{L}(\theta)}{\partial \theta}, \tag{4.7}$$

where $F(\mathbf{u}, \theta) = 0$ is the PDE corresponding to the simulator. We implement the automatic gradient calculation using dolfin-adjoint [Mitusch et al., 2019, Dokken et al., 2020] and PyTorch [Paszke et al., 2019b].

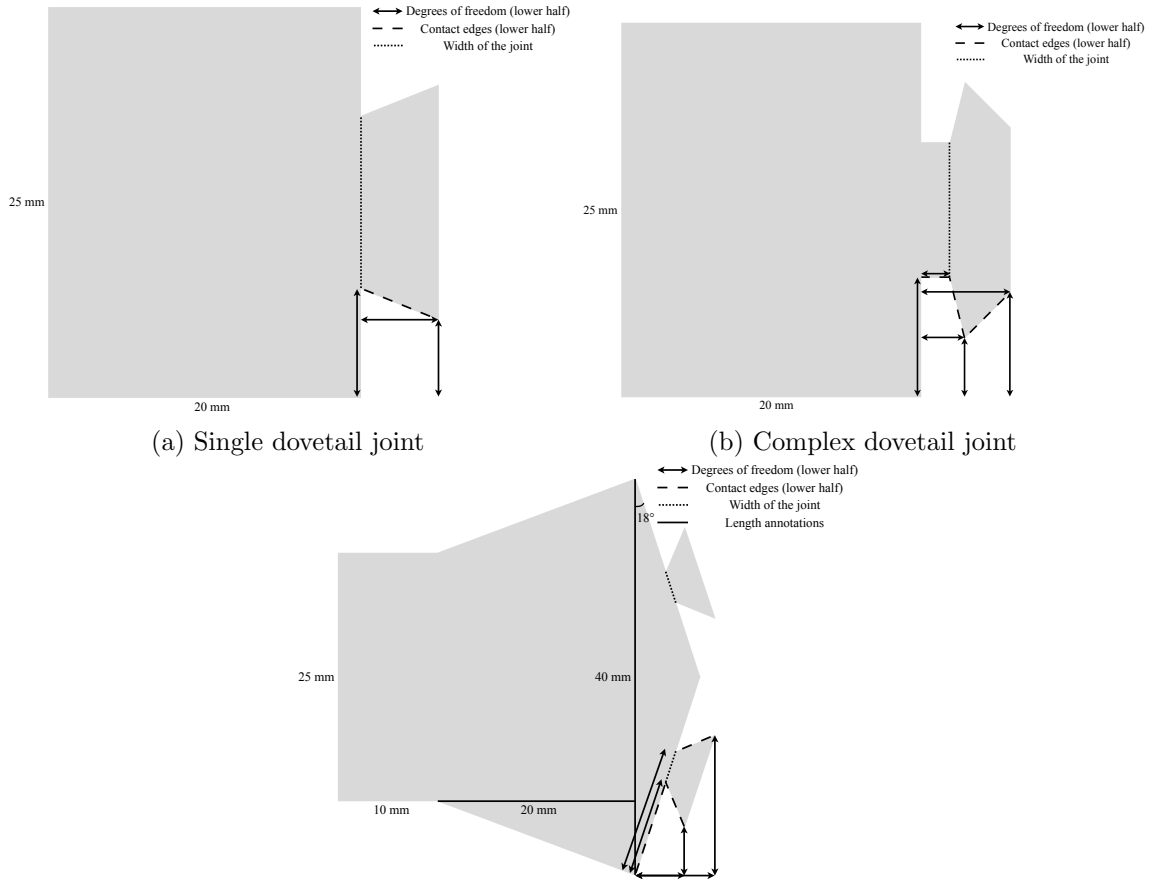
To optimize θ, we use gradient descent. For every step, we perform a line search along the gradient direction using SciPy [Virtanen et al., 2020b], finding step size that satisfies strong Wolfe conditions [Wolfe, 1969, 1971]. If the line search fails, we randomly sample a step size from $\mathcal{N}(0, 0.5^2)$. We perform 15 optimization steps and keep the step with minimum $d(\mathbf{u}(\theta))$. Lastly, to prevent landing in design parameters that are sensitive to manufacturing errors, every time we evaluate the objective function or its gradient, we apply independent random noises sampled from $\mathcal{N}(0, 0.01^2)$ to every dimension of θ three times and take the average.

4.4 Experiments

In this section, we study the effectiveness of our method. We present different dovetail joint design spaces for all the experiments in § 4.4.1. We then provide simulated results from the simulator (§ 4.4.2) and the correctness check of gradients on mesh vertex coordinates (§ 4.4.3). We show the main result—the optimization results in § 4.4.4, evaluating them both synthetically and in real experiments. Finally, to better understand the effect of material parameters, we show the sensitivity test of optimization results with respect to the Poisson's ratio in § 4.4.5.

4.4.1 Shapes for experiments

As shown in Figure 4.3, we use three different joint design spaces for all the experiments. We only visualize the left part of the joint, as the right part is complementary to it. All joints are horizontally symmetric, so all simulation is only performed on the lower half of the mesh for lower computational cost. The first design space is named *single dovetail joint* (Figure 4.3a), which contains the simplest form of a dovetail joint and three degrees of freedom. The second design space is called *complex dovetail joint* (Figure 4.3b), which still contains only one dovetail shape but with a structure that



(a) Single dovetail joint

(b) Complex dovetail joint

(c) Double dovetail joint

Figure 4.3: Three dovetail joint shape design spaces that are used in the experiments. All designs are horizontally symmetric. (a) *single dovetail joint* with the simplest design of dovetail; three degrees of freedom. (b) *complex dovetail joint* with a design that is more complex; six degrees of freedom. (c) *double dovetail joint* with two dovetails and a non-vertical boundary in the middle; six degrees of freedom. All contact edges and the width are annotated for every design space.

is more complex and has six degrees of freedom. The third design space is *double dovetail joint* (Figure 4.3c), which contains two dovetail structures, a non-vertical boundary in the middle, and six degrees of freedom. We annotate all the contacting edges for each design space, which are used in the contact penalizer and the minimum contact length regularizer. We also label the width of every design space, which is used in the minimum width regularizer.

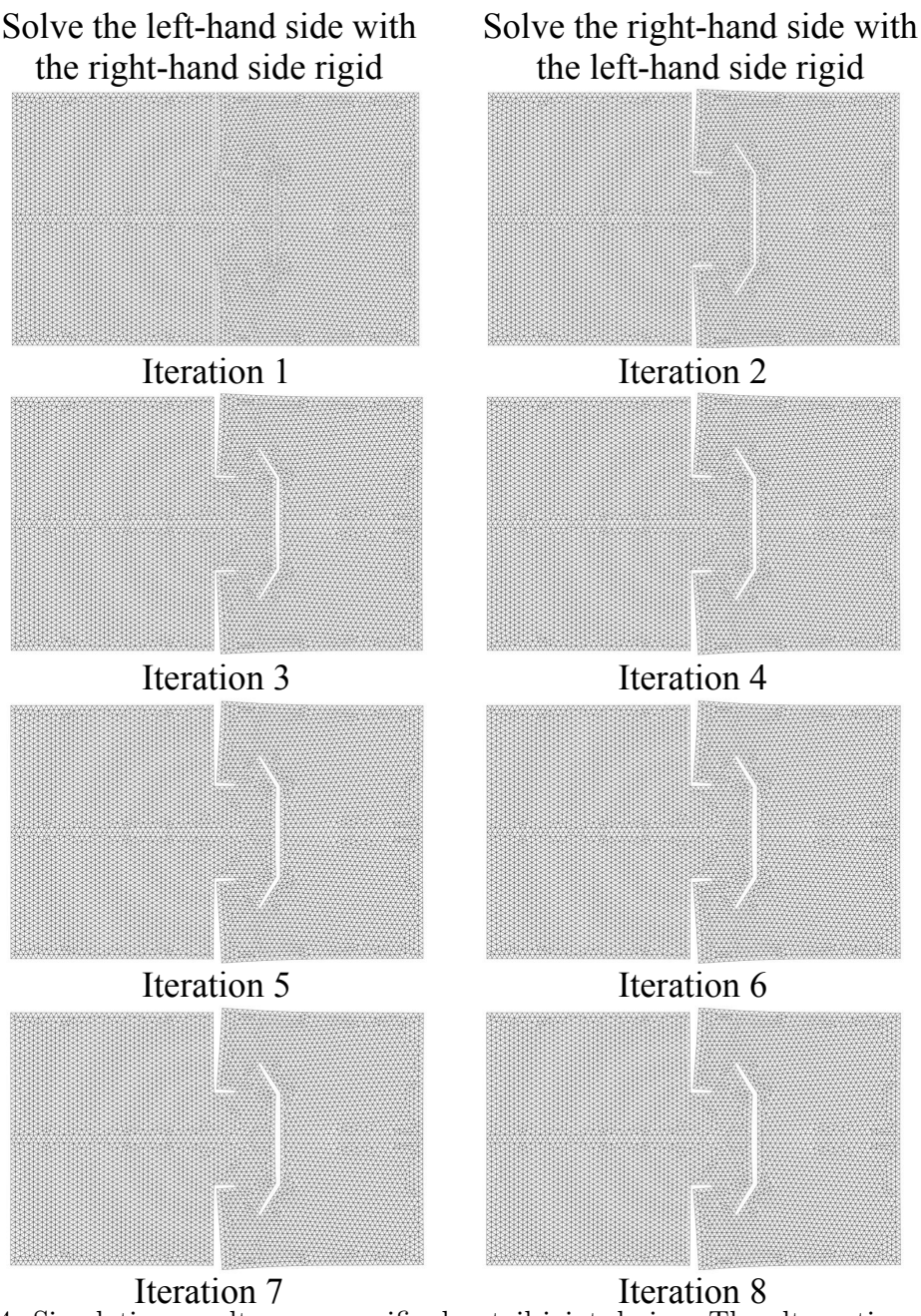
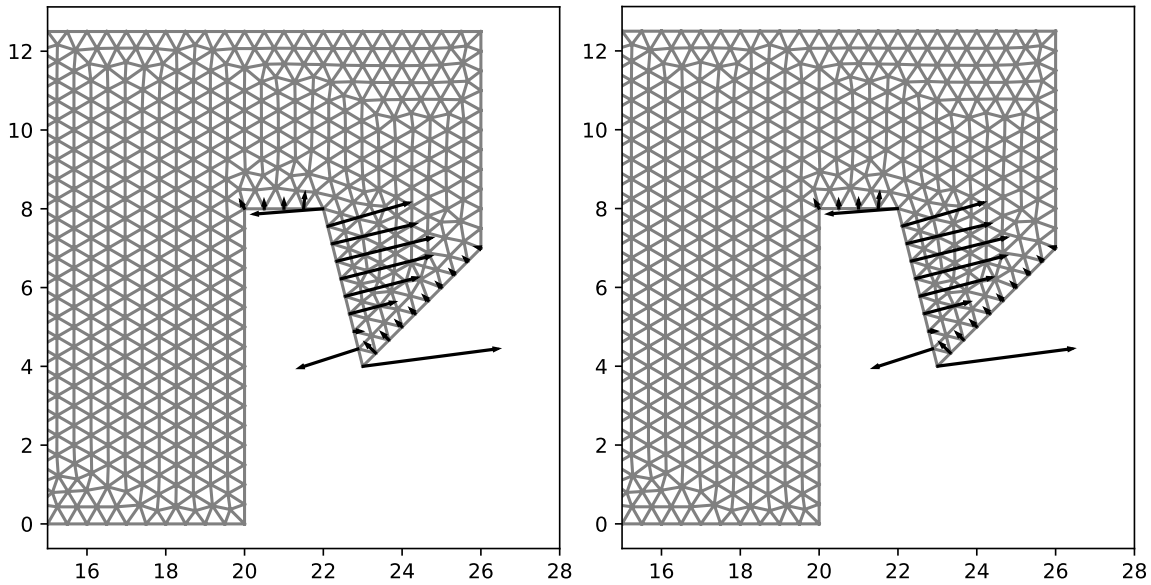


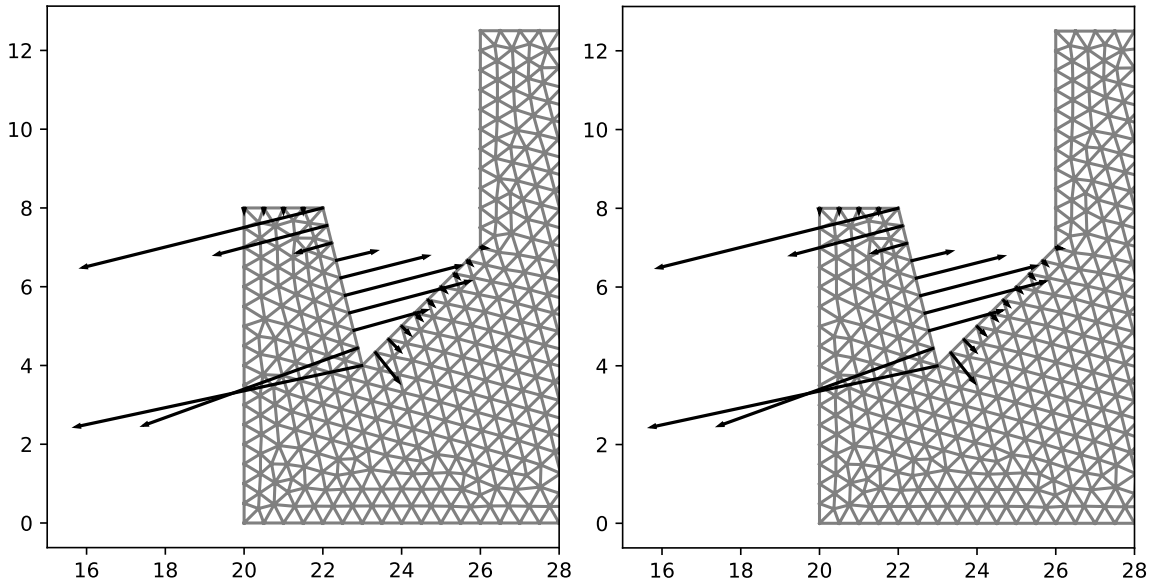
Figure 4.4: Simulation results on a specific dovetail joint design. The alternating simulator produces reasonable results as the number of iterations increases, and the results converge in the end.

4.4.2 Alternating penalty contact simulator results

In this subsection, we visualize simulation results from the alternating penalty contact simulator to demonstrate it is producing reasonable results, and the results are shown



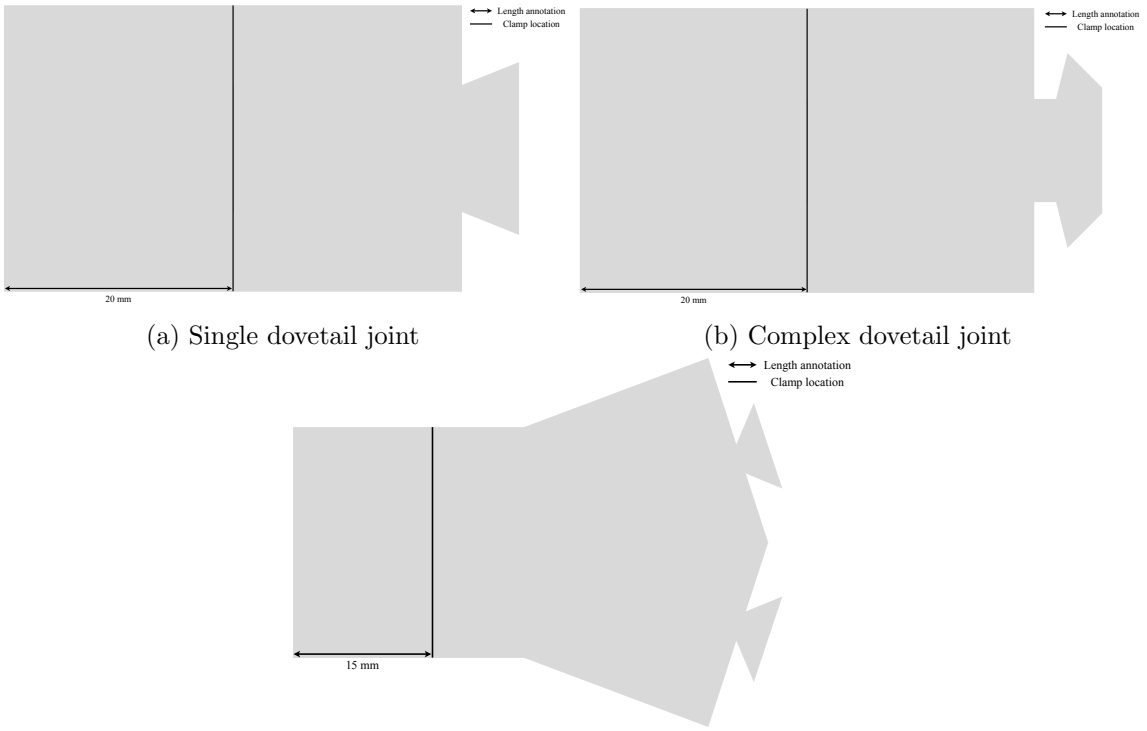
(a) Gradients on the left half; adjoint method (b) Gradients on the left half; finite difference



(c) Gradients on the right half; adjoint method (d) Gradients on the right half; finite difference

Figure 4.5: Gradient directions on three contacting edges calculated using the adjoint method and the finite difference method. The two results are indistinguishable, which indicates the gradient calculation is correct.

in Figure 4.4. As the number of iterations increases, the results evolve and converge in the end.



(a) Single dovetail joint

(b) Complex dovetail joint

(c) Double dovetail joint

Figure 4.6: Tabs are added in real printings such that the universal testing machine can clamp the printed parts.

4.4.3 Gradient correctness check

In this subsection, we check the correctness of our gradient computation. We calculate the derivative of the displacement with respect to the coordinates of vertexes from the mesh in two different approaches: the adjoint method and the finite difference method, as shown in Figure 4.5. For the finite difference method, we use a step size of 10^{-4}. We only visualize gradients on three edges, as interior gradients should be all zero. The two sets of gradients are indistinguishable, which infers our gradient calculation is consistent with the finite difference method. The average relative difference from the adjoint method results to the finite difference results is 1.82×10^{-4}, which is negligibly small.

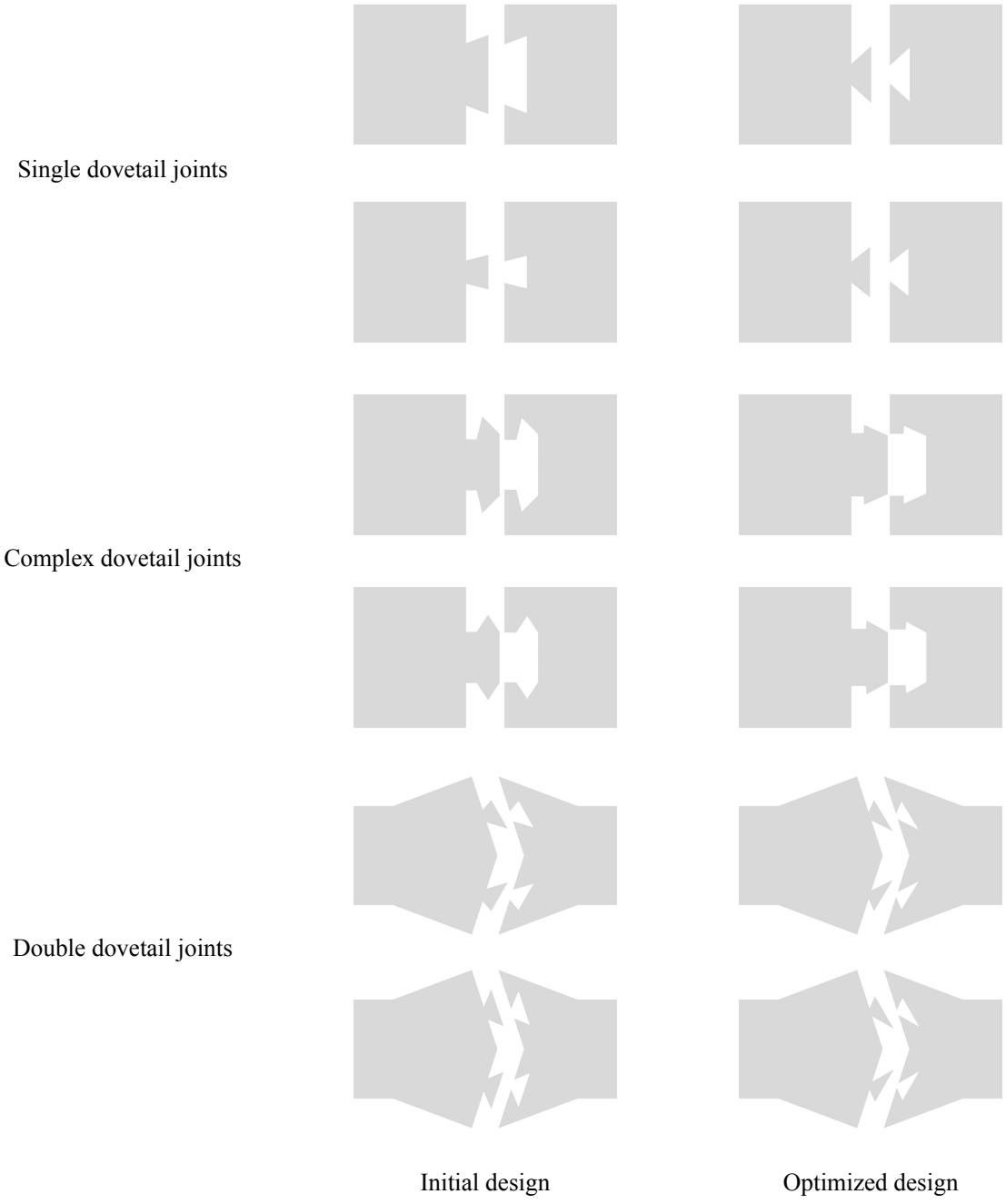


Figure 4.7: Initial and optimized designs of dovetail joints. The optimized designs are similar for the same design space though the initializations are very different.

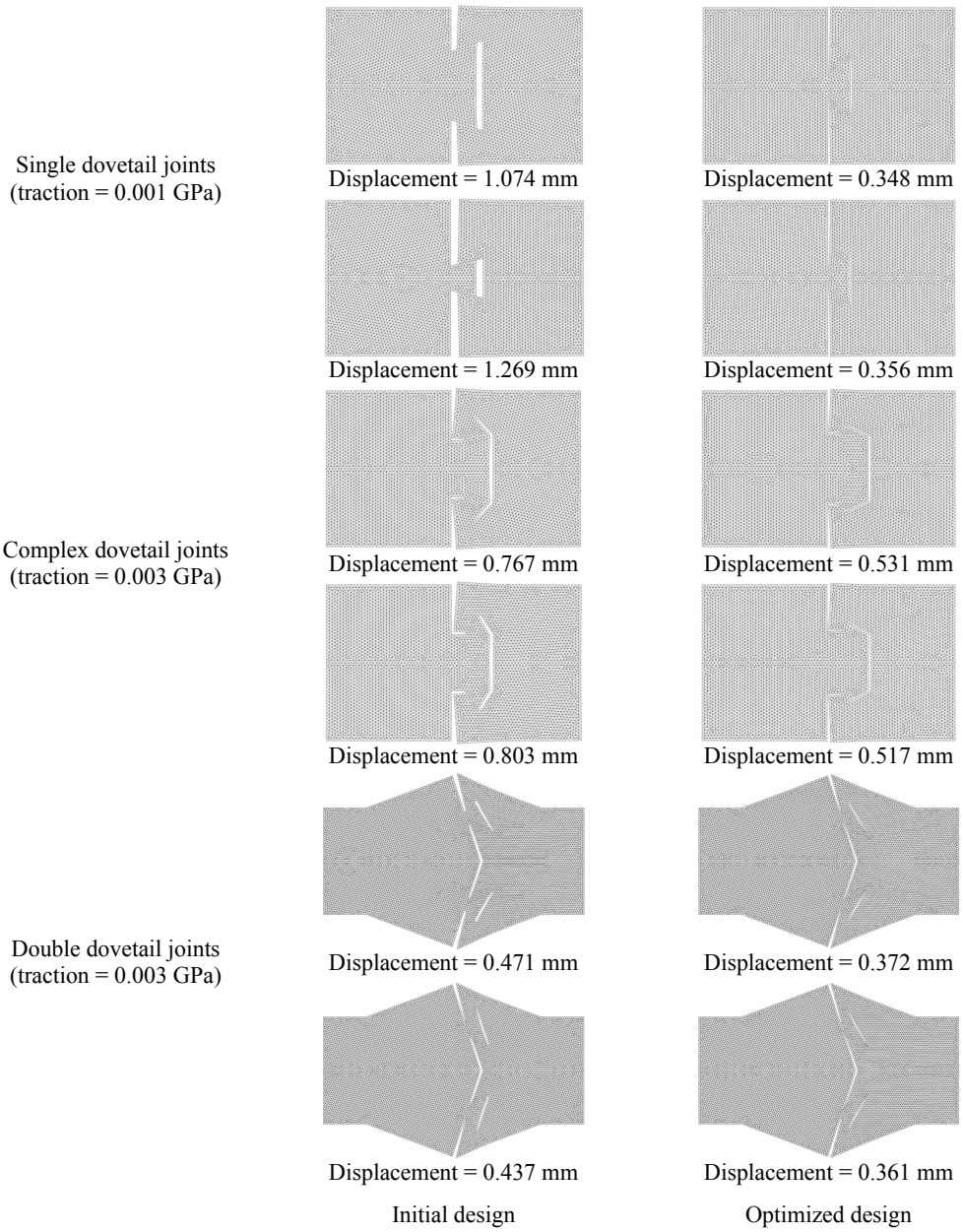


Figure 4.8: Simulated results and displacements of initial and optimized designs. The displacements of the optimized designs are much smaller than those of the initial designs, which indicates the optimization algorithm is working effectively.

4.4.4 Shape optimization for stiffness

Fabrication and experimental setup. We use Fusion 360[1] to draw 3D shapes and CHITUBOX[2] and UVtools[3] for slicing. We use the ELEGOO Saturn S resin 3D

[1] https://www.autodesk.com/products/fusion-360

[2] https://www.chitubox.com/

[3] https://github.com/sn4k3/UVtools

Table 4.1: Real measured stiffness of initial and optimized designs over three batches. The significant increase in stiffness indicates our algorithm successfully optimizes the joint shape design.

Joint type	Initial design	Optimized design
Single dovetail joint	219.9±22.6	**362.3±26.9**
	228.4±53.2	**406.0±15.1**
Complex dovetail joint	518.1±29.2	**1036.3±61.6**
	553.3±55.7	**1120.0±102.4**
Double dovetail joint	380.6±23.8	**527.4±48.3**
	360.2±12.1	**611.8±64.0**

printer to print laminates of ABS-like resin with a height of 5 mm. For the printed parts to be assemblable, we introduce a gap of 0.1 mm between the joints. To measure the stiffness of a joint, we use a universal testing machine Instron 600DX, which is set to move at a speed of 20 mm/min until the tested object breaks and produces a position-load curve. We also include tabs on two sides of the joint for the machine to clamp, as shown in Figure 4.6: the parts on the left-hand side of the solid line are the tabs, and the solid lines indicate where we clamp at.

Optimization and test results. For each design space, we randomly select two different initial sets of shape parameters and optimize them for fifteen gradient descent steps. We select the iteration with the smallest simulated displacement, and the results are shown in Figure 4.7. The optimized results are similar though the initializations are quite different. The simulated results are shown in Figure 4.8. We print three copies of each design and test them on the universal testing machine with position-load curves recorded. We measure the stiffness using the position change between a load of 30 N and a load of 60 N, and the results are listed in Table 4.1. The significant differences between the initial and optimized designs indicate the effectiveness of our algorithm.

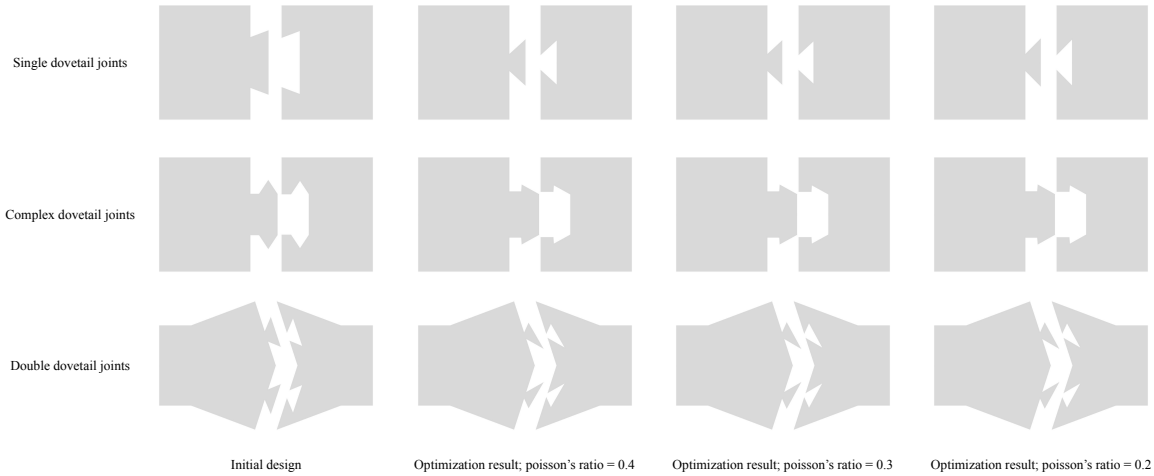


Figure 4.9: Optimization results from the same initial design but using different Poisson's ratios. As the difference is negligible, we observe that the optimization results are not sensitive to Poisson's ratio.

4.4.5 Poisson's ratio sensitivity test

This subsection studies the effect of material parameters on the optimization results. As Young's modulus is obviously not affecting our optimization process (if simultaneously increasing the traction), we study whether changing Poisson's ratio would change the results, and the results are shown in Figure 4.9. As different Poisson's ratios produce almost the same optimization result, we observe that the results are not sensitive to Poisson's ratio.

4.5 Discussion

In this project, we studied the task of dovetail joint shape optimization to maximize its stiffness, and, as existing works, we formalized the task as an optimization problem, viewing the stiffness of the joint as a function of shape design parameters. Existing works use search algorithms, gradient-free optimizers, or surrogate models for optimization, which have limited efficiency. To use gradient-based optimizers, we first built our own contact simulator by alternatively simulating the deformation of one side of the joint while considering the other side as rigid, using the penalty

approach. We then use the adjoint method and gradient descent for optimization. For experiments, we first verified the gradients were correct by comparing them to the gradients calculated from the finite difference method. We then tested the optimized joint shapes on different initializations in different dovetail shape design spaces, both synthetically and in real experiments, showing that the optimized joints are much stiffer than the initial ones. Note that our method is not restricted to dovetail joint shape optimization but works for joints with piecewise linear joint boundaries.

We would also like to discuss some limitations and future directions of our approach. First, the simulator has limited accuracy as several assumptions and simplifications are made, *e.g.*, plane stress, fitting boundaries using lines for the penalty term, *etc*. Real experiments showed that the simulator is still reasonably accurate, but future work can be done on more accurate simulators. Besides, the optimization is not in real-time, and most optimization (15 gradient descent steps) in this project takes 10 to 20 minutes to finish on a laptop computer. However, compared to the manufacturing time, this is acceptable, and there are amortized approaches that can significantly reduce the running time. Finally, the optimized results from gradient-based optimizers can be local minima. Possible solutions include using different initializations, introducing randomness during optimization, *etc*.

Chapter 5

Discussion

In this dissertation, we studied three shape design tasks with the help of gradient-based optimizers. Compared to existing works that mostly rely on gradient-free optimizers, sampling algorithms, or search algorithms, we proposed to use automatic differentiation to compute the derivative of the objective function with respect to the design parameters and use gradient-based optimizers to optimize them. In the first project (Chapter 2), we aimed to find extruder paths for continuous fibers that can compensate for the deformation caused by the fiber printing process. As the process is difficult to be directly modeled, we build a simulator and a differentiable surrogate of it using neural networks. We further reduce the running time by using a neural network to amortize the cost of optimization. In the second project (Chapter 3), we worked on finding locations to lay reinforcing fibers in plastic, aiming at maximizing the stiffness of the 3D-printed composite. A linear elastic simulator was built to calculate the deformation of the object given a specific fiber layout. We used the adjoint method to compute the gradient and BFGS optimizer to find the optimal fiber layout. In the third project (Chapter 4), we investigated the task of dovetail joint shape optimization, where a simulator was built by alternatively solving the deformation of one side of the joint while considering the other side rigid. We used the

adjoint method for gradient calculation and gradient descent for optimization. In the three projects, we tested our method in both simulation and real-world experiments, showing that our approach is producing designs of high quality and that the amortized approach provides real-time inferences and designs of comparable quality.

There are limitations to our approach, and we would also like to mention some future directions. First, the accuracy of our simulator is limited, as several assumptions and simplifications were made (*e.g.*, plane stress). Thus a future direction would be creating simulators with higher quality. Additionally, the optimization is not real-time, and the amortized approach sacrifices some accuracy. The optimization usually takes 10 to 20 minutes to finish—though acceptable, can be improved. Future directions can continue to speed up the optimization or increase the accuracy of the amortized approach. Besides, hyperparameters may need to be tuned if the design goal changes. For example, the weights of regularizes need to be adjusted if the design is happening on a different scale, though usually, a few tries would be enough. Lastly, gradient-based optimizers can be trapped in local minima. Future directions can be introducing randomness to the optimization process or trying different initializations.

Appendix A

Appendix for Chapter 2

A.1 Details about extruder path planning

A.1.1 Methods

Ours. As we discussed before in Section 2.3, we train a decoder (surrogate) using Equation 2.2 and an encoder using Equation 2.3, with the cost function defined in Equation 2.6. We take the trained encoder $\phi^*(\cdot)$ as our final model in use.

`direct-learning`. As we discussed in Section 2.4, we train `direct-learning` using Equation 2.4, with a regularizer as defined in Equation 2.7. Note that this is equivalent to training to minimize the cost function $\mathcal{L}_{\cdot}(\cdot,\cdot)$ in Equation 2.6.

`direct-optimization`. As described in Section 2.4, we build `direct-optimization` as in Equation 2.5, with a trained surrogate of the physical realization process. Here, to enforce that points $\boldsymbol{\theta}_i$ are evenly distributed along the extruder path, we use a slightly different cost function $\mathcal{L}_{\mathsf{do},\cdot}(\cdot,\cdot)$:

$$\mathcal{L}_{\mathsf{do},\boldsymbol{g}}(\boldsymbol{\theta},\boldsymbol{u}) \coloneqq d_{\mathsf{do}}(\boldsymbol{g},\boldsymbol{u}) + \lambda_{\mathsf{do}} \cdot \mathcal{R}_{\mathsf{do}}(\boldsymbol{\theta}), \tag{A.1}$$

where we have a distance function $d_{\mathsf{do}}(\cdot,\cdot)$ and a smooth regularizer $\mathcal{R}_{\mathsf{do}}(\cdot)$. The smooth regularizer $\mathcal{R}_{\mathsf{do}}(\cdot)$ is derived from Equation 2.7 by requiring $||\boldsymbol{\theta}_{i+1}-\boldsymbol{\theta}_i||_2$ to be the same for all i:

$$\mathcal{R}_{\mathsf{do}}(\boldsymbol{\theta}) \coloneqq \frac{1}{S_{\boldsymbol{\theta}}}\sum_{i=2}^{n-1}\left(\frac{\boldsymbol{\theta}_{i+1}-\boldsymbol{\theta}_i}{2}-\frac{\boldsymbol{\theta}_i-\boldsymbol{\theta}_{i-1}}{2}\right)^2, \tag{A.2}$$

where $S_{\boldsymbol{\theta}}$ is the length of the extruder path $\boldsymbol{\theta}$; this intrinsically enforces points in $\boldsymbol{\theta}$ to be evenly spaced. To measure the distance between $\boldsymbol{g}$ and $\boldsymbol{u}$, we first map them into two functions $\boldsymbol{f}_{\boldsymbol{g}}(\cdot)$ and $\boldsymbol{f}_{\boldsymbol{u}}(\cdot)$, such that $\boldsymbol{f}_{\boldsymbol{g}}(s)\in\mathbb{R}^2$ is the location if we walk a distance s along the path $\boldsymbol{g}$ (assuming the path is piecewise linear), and similarly for $\boldsymbol{f}_{\boldsymbol{u}}(\cdot)$. Then the distance function is defined as

$$d_{\mathsf{do}}(\boldsymbol{g},\boldsymbol{u}) \coloneqq \int_0^1 \left\|\boldsymbol{f}_{\boldsymbol{g}}(x\cdot S_{\boldsymbol{g}})-\boldsymbol{f}_{\boldsymbol{u}}(x\cdot S_{\boldsymbol{u}})\right\|^2 \mathrm{d}x, \tag{A.3}$$

where $S_{\boldsymbol{g}}$ and $S_{\boldsymbol{u}}$ denote the lengths of paths $\boldsymbol{g}$ and $\boldsymbol{u}$, respectively.

A.1.2 Data generation

Random curve generation. To build the dataset, we first need to generate some random 2D curves, which can be used as extruder paths later. The curves should be smooth and non-intersecting. For each path, we take both the x and y to be Gaussian processes whose kernel function $K(\cdot,\cdot)$ is

$$K(x,x') \coloneqq \exp\left(-\frac{\sin^2\big((x-x')/2\big)}{2\,l^2}\right), \tag{A.4}$$

with the two axes independent and $l=0.1$. We use elliptical slice sampling [Murray et al., 2010] (New BSD License): for each path, we start from 1,000 points on a unit circle, and sample 1,000 times. To avoid intersections, we use a log-likelihood of $-\infty$

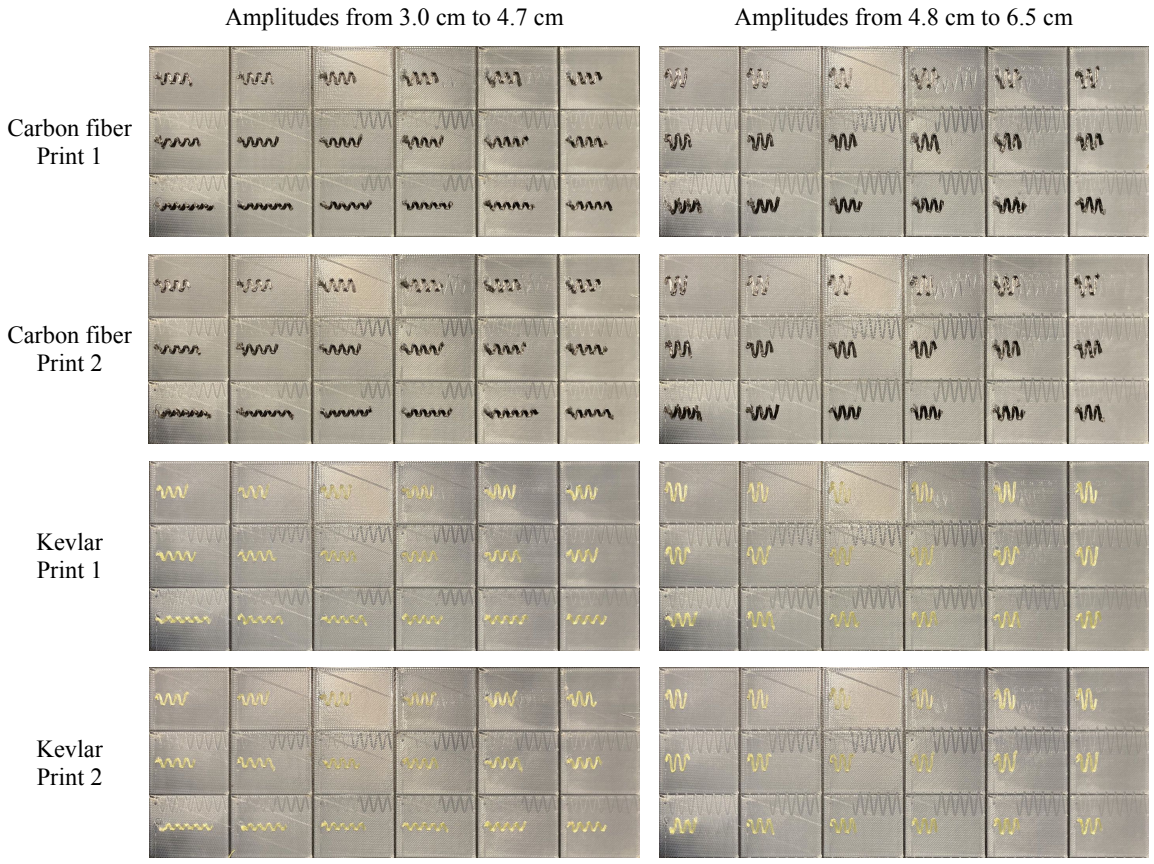
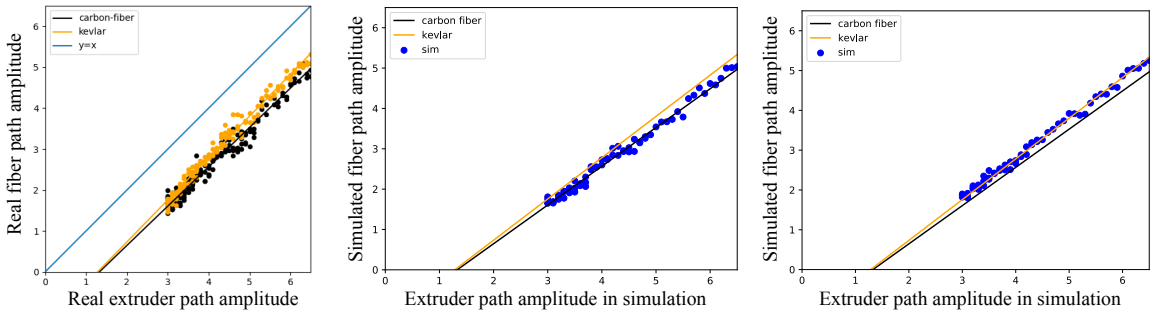


Figure A.1: We print paths shaped as sine functions with amplitudes from 3.0 cm to 6.5 cm on the Markforged Mark Two printer, using carbon fiber and Kevlar, respectively. We print everything twice.

for a self-intersecting path, and a log-likelihood of 0 for a non-intersecting path. We generate 10,000 paths using this approach.

Simulator. Since it is time-consuming to print every extruder path we generate in the last step on a real printer, we build a simulation system by using Bullet [Coumans, 2010] to help us generate fiber paths. The simulator is also used in our evaluation. We calibrate the simulator on two materials—carbon fiber and Kevlar, respectively. To calibrate, we print paths shaped as sine functions with amplitudes ranging from 3.0 cm to 6.5 cm on the Markforged Mark Two printer (Figure A.1). We then measure the amplitudes of the printed fiber paths, which will be lower than the amplitudes of the extruder paths because of smoothing, and fit lines to actual amplitude vs. extruder



a) lines fitted on real data b) simulator tuned for carbon fiber c) simulator tuned for Kevlar

Figure A.2: (a) We print paths shaped as sine functions with different amplitudes, collect amplitudes of the printed fiber paths, and fit lines through the data we collected. We experiment with two materials—carbon fiber and Kevlar, and the identity line is also visualized. (b) and (c) We select two sets of simulator hyper-parameters with their simulation results closest to the lines we get from the previous step for carbon fiber and Kevlar, respectively.

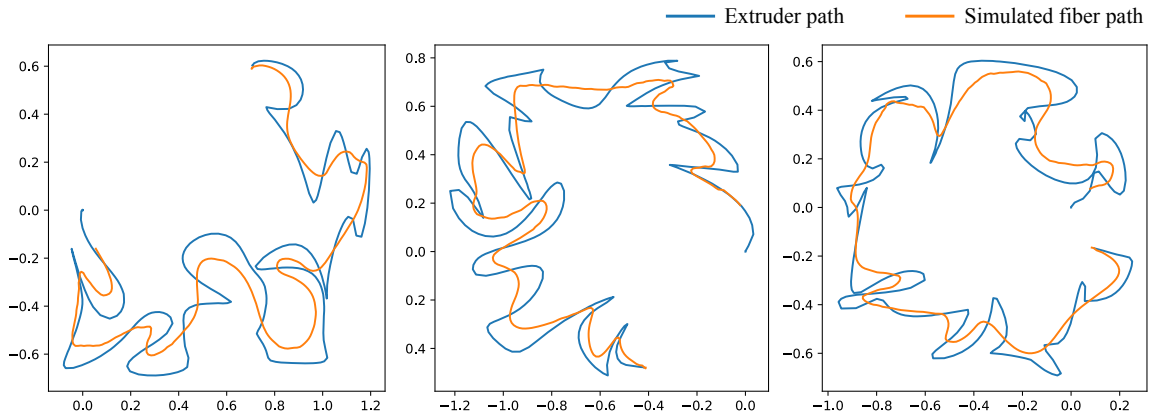


Figure A.3: Samples from our dataset. We plot both the extruder paths and the simulated carbon fiber paths.

amplitude (Figure A.2a). After that, we perform a grid search on the parameters of the simulator (such as stiffness and friction), run simulations with the sine functions as extruder paths, and collect data pairs of extruder path amplitudes and simulated fiber path amplitudes. We end up with calibrated simulators for carbon fiber (Figure A.2b) and Kevlar (Figure A.2c), by selecting the set of parameters for each having the minimum sum of squared distances between the fitted line from Figure A.2a) and the collected simulation data points. Finally, we run the tuned simulators on the generated extruder paths. Though we have conducted experiments with both materials, due to space constraints, the experiments in the paper use data generated from the carbon

fiber simulator. We visualize some extruder paths and simulated carbon fiber paths in Figure A.3.

A.1.3 Mapping from a path to another

We have to design a neural network that can take in an input path ($\mathbb{R}^{n\times 2}$) and output another path ($\mathbb{R}^{n\times 2}$), which can be used for encoder, decoder, and `direct-learning`. Both paths are sequences of n 2D coordinates. For the purpose of illustration, we use decoder as an example here. Now the input is the extruder path $\boldsymbol{\theta}$, and the output is the resulting fiber path $\boldsymbol{u}$. Remember $\boldsymbol{\theta}_i \in \mathbb{R}^2$ is the i-th row of $\boldsymbol{\theta}$. Due to the intrinsic equivariant property of the problem, one natural idea is to have a neural network that takes in a certain number of points near $\boldsymbol{\theta}_i$ in $\boldsymbol{\theta}$ and outputs the corresponding point in $\boldsymbol{u}$ (*i.e.*, $\boldsymbol{u}_i$), and we iterate over every i, as a window sliding over $\boldsymbol{\theta}$. Note that we used the same neural network for all i's.

We thus use a multilayer perceptron (MLP), which takes $2m+1$ points (we set $m = 30$) and outputs one point. We take $\boldsymbol{\theta}_i$ as the starting point and resample m points both forward and backward along the path $\boldsymbol{\theta}$. To be specific, as in § A.1.1, we first map $\boldsymbol{\theta}$ into a function $\boldsymbol{f}_{\boldsymbol{\theta}}(\cdot)$ such that $\boldsymbol{f}_{\boldsymbol{\theta}}(s) \in \mathbb{R}^2$ is the location if we start from $\boldsymbol{\theta}_1$ and walk a length of s on the path. We further set $\boldsymbol{f}_{\boldsymbol{\theta}}(s) \coloneqq \boldsymbol{f}_{\boldsymbol{\theta}}(0)$ for $s < 0$ and $\boldsymbol{f}_{\boldsymbol{\theta}}(s) \coloneqq \boldsymbol{f}_{\boldsymbol{\theta}}(S_{\boldsymbol{\theta}})$ for $s > S_{\boldsymbol{\theta}}$, where $S_{\boldsymbol{\theta}}$ is the length of extruder path $\boldsymbol{\theta}$. We denote the distance of walking from $\boldsymbol{\theta}_1$ to $\boldsymbol{\theta}_i$ as s_i, *i.e.*, $\boldsymbol{f}_{\boldsymbol{\theta}}(s_i) = \boldsymbol{\theta}_i$. The input to the MLP is:

$$[\boldsymbol{f}_{-m}, \boldsymbol{f}_{-m+1}, \cdots, \boldsymbol{f}_0, \cdots, \boldsymbol{f}_m]^\intercal, \tag{A.5}$$

where

$$\boldsymbol{f}_i \coloneqq \boldsymbol{f}_{\boldsymbol{\theta}}(s_i + i \cdot s_0) - \boldsymbol{f}_{\boldsymbol{\theta}}(s_i), \tag{A.6}$$

and $s_0 = 0.03$ is the step size. Since the problem is intrinsically translation-equivariant, we normalize every $\boldsymbol{f}_i$ by subtracting $\boldsymbol{f}_{\boldsymbol{\theta}}(s_i)$, as shown in the above equation.

Table A.1: The architecture of the MLP used in extruder path planning

Type	Configurations
Fully connected	$4m$+2 to 500
ReLU	N/A
Fully connected	500 to 200
ReLU	N/A
Fully connected	200 to 100
ReLU	N/A
Fully connected	100 to 50
ReLU	N/A
Fully connected	50 to 25
ReLU	N/A
Fully connected	25 to 2

Table A.2: Path-planning evaluation of `direct-optimization` of the average Chamfer distance on the first 40 samples in the test set evaluated in simulation

Regularizer weight	0.0001	0.0003	0.0006	0.001
`direct-optimization`	0.0171	0.0161	**0.0153**	0.0167

A.1.4 Hyper-parameters and neural network training

The architecture of the MLP is shown in Table A.1, and we implement it in PyTorch [Paszke et al., 2019b]. We coarsely tuned the architecture, including the number of hidden layers (from 1 to 6) and the size of each hidden layer. We noticed that the accuracy is not largely affected by the architecture, as long as there is at least one hidden layer. We split the dataset into 90% training (9,000 paths), 5% validation (500 paths), and 5% testing (500 paths), and we use the Adam optimizer [Kingma and Ba, 2015] with a learning rate of 1×10^{-3}, a learning rate exponential decay of 0.95 per epoch, and a batch size of 1 (path). We train every model—the decoder, the encoder, and `direct-learning`—for 10 epochs. We use our internal cluster with 7 servers with 14 Intel(R) Xeon(R) CPUs. For our method and `direct-learning`, we train them with different regularizer weights $\lambda = 0.1$, 0.3, 0.6, 1.0, and 1.5. For `direct-optimization`, we use the BFGS implementation in SciPy [Virtanen et al.,

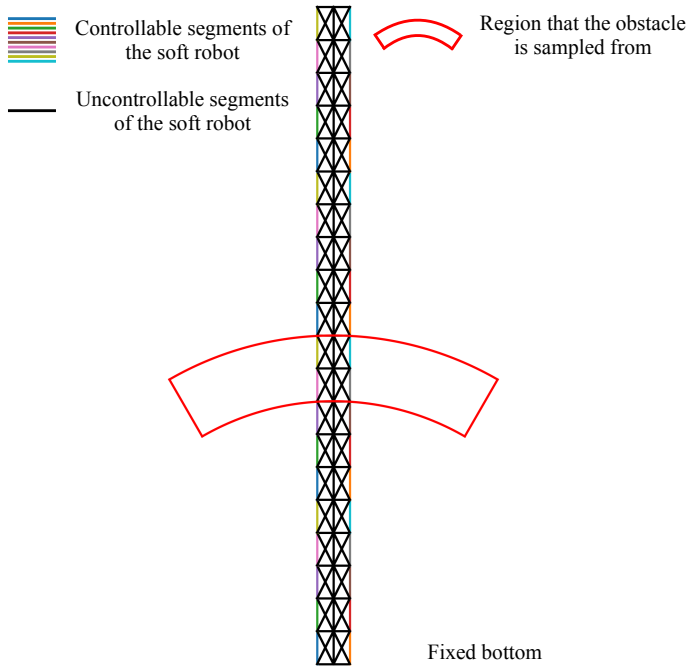


Figure A.4: We visualize the soft robot with controllable segments in color, uncontrollable segments in black. The red region shows where we sample the center of the obstacle—a sector region with an angle of 60°, inner and outer radiuses of 4 and 5, respectively.

2020a], with a gradient tolerance of 1×10^{-7}. Since its running time is extremely long, we tune its regularizer weight on the first 40 test samples (Table A.2) and select the one with the best performance. To train the needed neural networks, it takes approximately 10 minutes for `direct-optimization`, approximately 1 hour for `direct-learning`, and approximately 5 days for our method. Note that we only need to train once so that the training costs are amortized.

A.2 Details about constrained soft robot inverse kinematics

A.2.1 Robot setting

We adopt the snake-like soft robot that was used in Xue et al. [2020a]. The robot has an original height of 10 and an original width of 0.5, and its bottom is fixed.

We can control the stretch ratio of 40 segments (20 on the left-hand side and 20 on the right-hand side), as visualized in colors in Figure A.4. The stretch ratios are restricted to be between 0.8 and 1.2. The physical realization of the robot consists of the locations of its 103 vertexes, as shown in Figure A.4.

A.2.2 Methods

Ours. We follow Section 2.3 with the cost function defined as in Equation 2.9, and the obstacle location is randomly sampled.

`direct-learning`. We follow Section 2.4 with $\mathcal{R}_{\texttt{dl}}(\cdot)$ defined as in Equation 2.11. Note that since we do not have access to the physical realization $\boldsymbol{u}$, we cannot have a barrier function term for the obstacle. Thus, for `direct-learning`, we still randomly sample the obstacle location, but we guarantee during training, the obstacle does not collide with the robot in every specific training sample.

`direct-optimization`. We follow Section 2.4 with the cost function defined as in Equation 2.9 and the obstacle location randomly sampled. Note that this baseline is similar to the approach used in Xue et al. [2020a]. The major difference is that we train the surrogate using supervised loss (as shown in Equation 2.2), and Xue et al. [2020a] trained their surrogate using a physically informed loss that minimizes the total potential energy.

A.2.3 Data generation

We first randomly sample the design vector $\boldsymbol{\theta}$ with each dimension i.i.d. uniformly between 0.8 and 1.2. For each design vector $\boldsymbol{\theta}$, we solve the governing PDE with the finite element method [Hughes, 2012] to obtain the corresponding physical realization $\boldsymbol{u}$ of the robot. Note that the obstacle location is randomly sampled during training

Table A.3: The architecture of the MLP used in constrained soft robot inverse kinematics

`direct-learning`		Decoder		Encoder	
Type	Configurations	Type	Configurations	Type	Configurations
Fully connected	4 to 128	Fully connected	40 to 128	Fully connected	4 to 128
ReLU	N/A	ReLU	N/A	ReLU	N/A
Fully connected	128 to 256	Fully connected	128 to 256	Fully connected	128 to 256
ReLU	N/A	ReLU	N/A	ReLU	N/A
Fully connected	256 to 128	Fully connected	256 to 128	Fully connected	256 to 128
ReLU	N/A	ReLU	N/A	ReLU	N/A
Fully connected	128 to 40	Fully connected	128 to 206	Fully connected	128 to 40
Sigmoid	N/A	/	/	Sigmoid	N/A
Linear map	$0.2(2x-1)$	/	/	Linear map	$0.2(2x-1)$

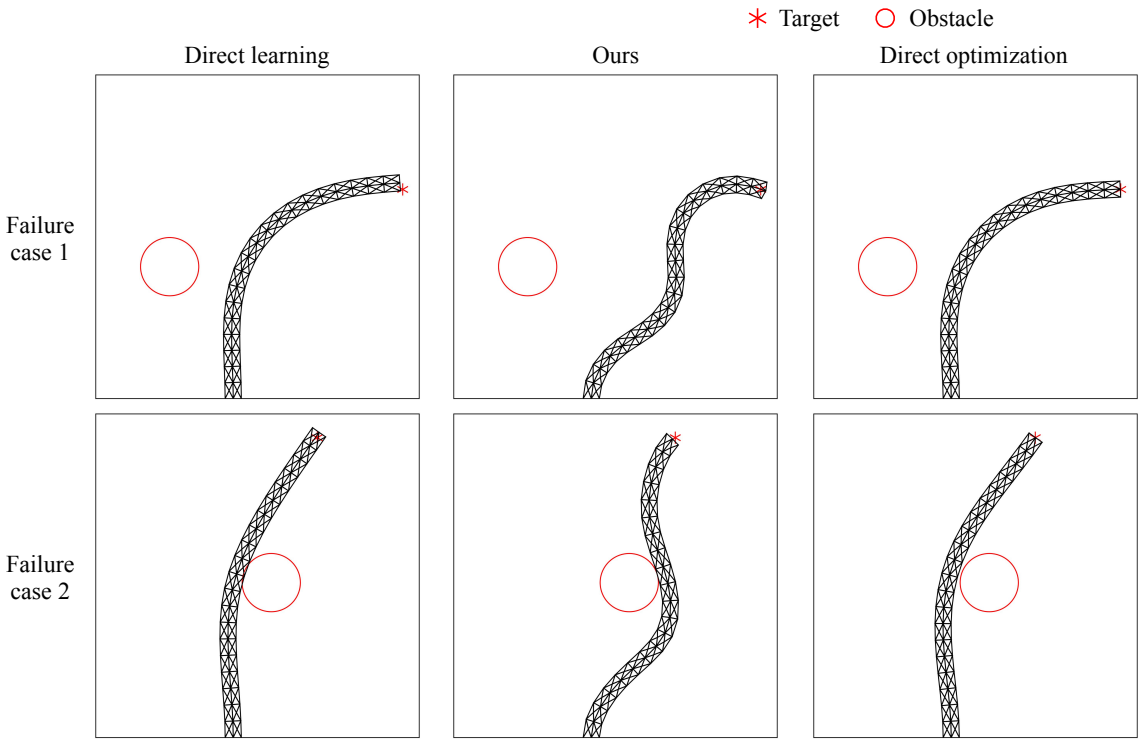


Figure A.5: Failure cases of our method in test set on soft robot. In rare cases, our method might miss the target by a short distance (Samples 1 and 2) or touch the obstacle (Sample 2).

and randomly sampled with a fixed random seed (we set to 0) during testing. The center of the obstacle is uniformly sampled from a sector region, as shown in Figure A.4, with an angle of 60°, an inner radius of 4, and an outer radius of 5. We altogether generate 40,000 data samples.

A.2.4 Hyper-parameters and neural network training

We use MLP for all models (encoder, decoder, and `direct-learning`) with ReLU as the activation function and 3 hidden layers of sizes 128, 256, 128, respectively (Table A.3). We coarsely tuned the architecture, including the number of hidden layers (from 1 to 4) and the size of each hidden layer. Similarly, we noticed that the accuracy is not largely affected by the architecture, as long as there is at least one hidden layer. Note that for the input and output of the neural network, we subtract 1 from all stretch ratios such that they are always between -0.2 and 0.2, and we use displacement of each vertex rather than its absolute location since displacement values are mostly centered around 0. In addition, to ensure that the encoder and `direct-learning` always output stretch ratios (minus one) between -0.2 and 0.2, we apply a sigmoid layer at the end of both the encoder and `direct-learning`, and linearly map the sigmoid output to be between -0.2 and 0.2 (as in Table A.3). We use the same trick in `direct-optimization` to ensure the stretch ratios never fall out of range.

We implement all neural networks in PyTorch [Paszke et al., 2019b]. We split the dataset into 90% training (36,000 samples), 7.5% validation (3,000 samples), and 2.5% testing (1,000 samples), and we use the Adam optimizer [Kingma and Ba, 2015] with a learning rate of 1×10^{-3}, a learning rate exponential decay of 0.98 per epoch, and a batch size of 8. We train every model—the decoder, the encoder, and `direct-learning`—for 200 epochs. For our method and all baselines, we experiment with different regularizer weights $\lambda_2 = 0.03$, 0.05, 0.07, and 0.09. For `direct-optimization`, we use the BFGS implementation in SciPy [Virtanen et al., 2020a], with a gradient tolerance of 1×10^{-7}. We use our internal cluster with 7 servers with 14 Intel(R) Xeon(R) CPUs. To train the needed neural networks, it takes approximately 2 hours for `direct-optimization`, approximately 4 hours for `direct-learning`, and approximately 4.5 hours for our method. Note that since we only need to train once, the training costs are amortized.

Table A.4: Ablation study: linear encoder vs. non-linear encoder for soft-robot evaluated on test set. All encoders are trained on non-linear decoders with a regularizer weight of 0.05

	#successful cases (over 1,000)	Avg. distance to target on successful cases
Non-linear encoder	**975.0±3.9**	**0.0464±0.0018**
Linear encoder	710.7±24.8	0.1324±0.0276

A.2.5 Failure cases

Since our encoder and decoder are both neural networks, there might be some generalization errors. We show two failure cases of our method in test set in Figure A.5. The design proposed by our method misses the target by a short distance in the first example, and both touches the obstacle and misses the target by a short distance in the second example.

A.2.6 Ablation study: linear encoder

To demonstrate the non-linearity in our encoder is necessary, we train a linear encoder on the pre-trained non-linear decoder, following exactly the same training procedure mentioned in § A.2.4. We show the number of cases the robot successfully avoids the obstacle and the average Euclidean distance to the target for successful cases in Table A.4. Linear encoder violates the obstacle constraints approximately 11.6 times as much as the non-linear encoder, and the average Euclidean distance for successful cases is approximately 2.9 times as much as the non-linear encoder. Two samples are shown in Figure A.6. In both samples, the linear encoder misses the target by a short distance, and in the second sample, the linear encoder violates the obstacle constraint.

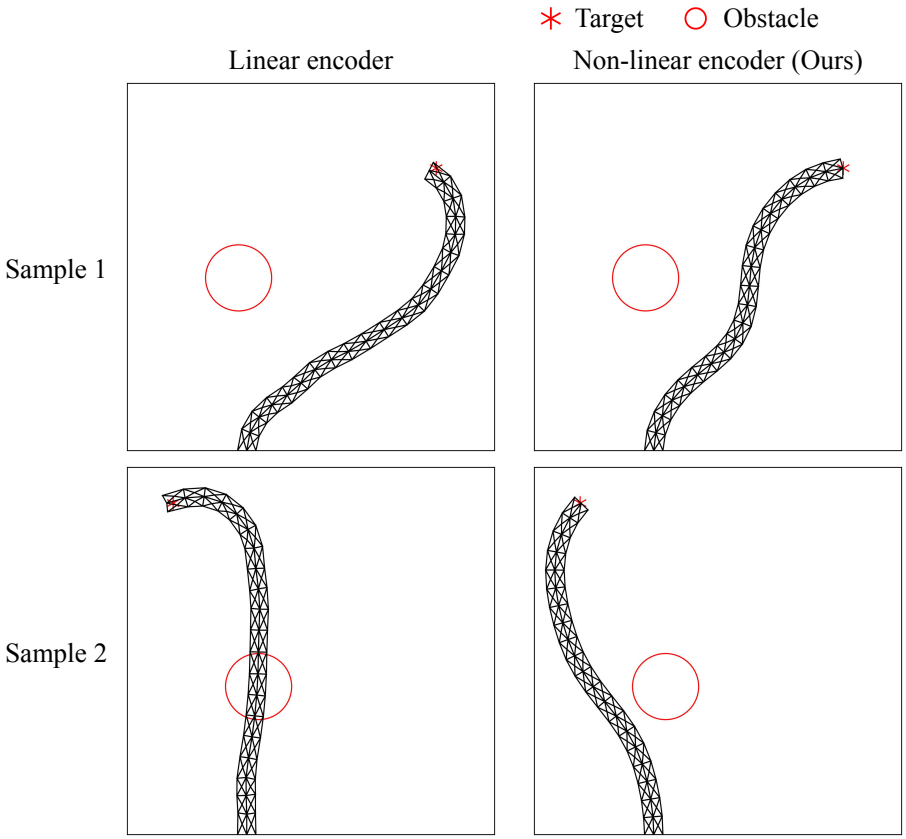


Figure A.6: Soft-robot evaluation of linear encoder *vs.* non-linear encoder (ours) on examples from the test set. Linear encoder both violates the constraints (Sample 2) and misses the target (Samples 1 and 2).

Appendix B

Appendix for Chapter 3

B.1 Field optimization

Given a stress field $\boldsymbol{\sigma}$, we would like to find a fiber field $\boldsymbol{v} : \Omega \to \mathbb{R}^2$ such that (1) its direction is aligned with $\boldsymbol{\sigma}$; (2) it is smooth. We solve $\boldsymbol{v}$ by minimizing an objective function that reflects both proprieties:

$$\mathcal{L}(\boldsymbol{v}; \boldsymbol{\sigma}) \coloneqq \alpha_{\texttt{stress}} \cdot \mathcal{L}_{\texttt{stress}}(\hat{\boldsymbol{v}}; \boldsymbol{\sigma}) + \alpha_{\texttt{smooth}} \cdot \mathcal{L}_{\texttt{smooth}}(\hat{\boldsymbol{v}}), \tag{B.1}$$

where $\alpha_{\texttt{stress}}$ and $\alpha_{\texttt{smooth}}$ are hyper-parameters, and

$$\hat{\boldsymbol{v}}(x, y) \coloneqq \boldsymbol{v}(x, y)/||\boldsymbol{v}(x, y)|| \tag{B.2}$$

is the normalized $\boldsymbol{v}$, as the objective function should be invariant regardless of the length of $\boldsymbol{v}(x, y)$. Note that the objective function should also be invariant if we randomly flip some $\boldsymbol{v}(x, y)$'s, which needs some special handling, as we will discuss below.

Consistent with $\boldsymbol{\sigma}$. For a specific point $(x, y) \in \Omega$, we calculate the tension in the stress field $\boldsymbol{\sigma}$ along $\hat{\boldsymbol{v}}(x, y)$, which is $\hat{\boldsymbol{v}}(x, y)^\intercal \boldsymbol{\sigma}(x, y)\hat{\boldsymbol{v}}(x, y)$. We then integrate it over Ω and get

$$\mathcal{L}_{\texttt{stress}}(\hat{\boldsymbol{v}}; \boldsymbol{\sigma}) \coloneqq -\iint_{\Omega} \hat{\boldsymbol{v}}(x, y)^\intercal \boldsymbol{\sigma}(x, y)\hat{\boldsymbol{v}}(x, y)\mathrm{d}x\mathrm{d}y, \tag{B.3}$$

where the negative sign indicates we would like to maximize the tension along the field direction.

Smoothness. We penalize the squared Frobenius norm of the gradient of $\hat{\boldsymbol{v}}$:

$$\mathcal{L}_{\texttt{smooth}}(\hat{\boldsymbol{v}}) \coloneqq \iint_{\Omega} ||\nabla \hat{\boldsymbol{v}}(x, y)||_F^2 \mathrm{d}x\mathrm{d}y. \tag{B.4}$$

Note that the penalty should be invariant to flips of $\hat{\boldsymbol{v}}(x, y)$'s, so we handle this invariance when calculating the finite difference:

$$\begin{aligned} ||\nabla \hat{\boldsymbol{v}}(x, y)||_F^2 \coloneqq &\min\left(\left|\left|\frac{\hat{\boldsymbol{v}}(x+h, y) - \hat{\boldsymbol{v}}(x, y)}{h}\right|\right|^2, \left|\left|\frac{\hat{\boldsymbol{v}}(x+h, y) + \hat{\boldsymbol{v}}(x, y)}{h}\right|\right|^2\right) \\ &+ \min\left(\left|\left|\frac{\hat{\boldsymbol{v}}(x, y+h) - \hat{\boldsymbol{v}}(x, y)}{h}\right|\right|^2, \left|\left|\frac{\hat{\boldsymbol{v}}(x, y+h) + \hat{\boldsymbol{v}}(x, y)}{h}\right|\right|^2\right), \end{aligned} \tag{B.5}$$

where h is the step size.

In the experiments, we set $\alpha_{\texttt{stress}}$ to 1 and $\alpha_{\texttt{smooth}}$ to 0.02. We use the BFGS optimizer with a gradient tolerance of 1×10^{-6} and set the maximum number of iterations to 100.

Bibliography

José Humberto S Almeida Jr., Lars Bittrich, Tsuyoshi Nomura, and Axel Spickenheuer. Cross-section optimization of topologically-optimized variable-axial anisotropic composite structures. *Composite Structures*, 225:111150, 2019.

Martin Alnæs, Jan Blechta, Johan Hake, August Johansson, Benjamin Kehlet, Anders Logg, Chris Richardson, Johannes Ring, Marie E Rognes, and Garth N Wells. The fenics project version 1.5. *Archive of Numerical Software*, 3(100), 2015.

Erik Andreassen, Anders Clausen, Mattias Schevenels, Boyan S Lazarov, and Ole Sigmund. Efficient topology optimization in matlab using 88 lines of code. *Structural and Multidisciplinary Optimization*, 43(1):1–16, 2011.

Suleman Asif. *Modelling and path planning for additive manufacturing of continuous fiber composites*. PhD thesis, 2018.

Marco Attene, Marco Livesu, Sylvain Lefebvre, Thomas Funkhouser, Szymon Rusinkiewicz, Stefano Ellero, Jonàs Martínez, and Amit Haim Bermano. Design, representations, and processing for additive manufacturing. *Synthesis Lectures on Visual Computing*, 10(2):1–146, June 2018.

Han Bao, Clay Scott, and Masashi Sugiyama. Calibrated surrogate losses for adversarially robust classification. In *Conference on Learning Theory*, pages 408–451. PMLR, 2020.

Harry G Barrow, Jay M Tenenbaum, Robert C Bolles, and Helen C Wolf. Parametric correspondence and chamfer matching: Two new techniques for image matching. In *IJCAI*, 1977.

Cenk Baykal and Ron Alterovitz. Asymptotically optimal design of piecewise cylindrical robots using motion planning. In *Robotics: Science and Systems*, volume 2017, 2017.

Alex Beatson, Jordan Ash, Geoffrey Roeder, Tianju Xue, and Ryan P Adams. Learning composable energy surrogates for pde order reduction. *Advances in neural information processing systems*, 33:338–348, 2020.

Martin Philip Bendsoe and Ole Sigmund. *Topology optimization: theory, methods, and applications*. Springer Science & Business Media, 2013.

Atharv Bhosekar and Marianthi Ierapetritou. Advances in surrogate based modeling, feasibility analysis, and optimization: A review. *Computers & Chemical Engineering*, 108:250–267, 2018.

V Bhuvaneswari, M Priyadharshini, C Deepa, D Balaji, L Rajeshkumar, and M Ramesh. Deep learning for material synthesis and manufacturing systems: a review. *Materials Today: Proceedings*, 2021.

Lorenz T Biegler, Omar Ghattas, Matthias Heinkenschloss, and Bart van Bloemen Waanders. Large-scale pde-constrained optimization: an introduction. In *Large-Scale PDE-Constrained Optimization*, pages 3–13. Springer, 2003.

Jeremy Bleyer. *Numerical Tours of Computational Mechanics with FEniCS*, 2018.

Paul T Boggs and Jon W Tolle. Sequential quadratic programming. *Acta numerica*, 4:1–51, 1995.

David Brookes, Hahnbeom Park, and Jennifer Listgarten. Conditioning by adaptive sampling for robust design. In *International conference on machine learning*, pages 773–782. PMLR, 2019.

David H Brookes and Jennifer Listgarten. Design by adaptive sampling. *arXiv preprint arXiv:1810.03714*, 2018.

Brian Cabral and Leith Casey Leedom. Imaging vector fields using line integral convolution. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, page 263–270, 1993.

JA Cabrera, A Simon, and M Prado. Optimal synthesis of mechanisms with genetic algorithms. *Mechanism and machine theory*, 37(10):1165–1177, 2002.

JA Cabrera, A Ortiz, F Nadal, and JJ Castillo. An evolutionary algorithm for path synthesis of mechanisms. *Mechanism and Machine Theory*, 46(2):127–141, 2011.

R Caivano, A Tridello, D Paolino, and G Chiandussi. Topology and fibre orientation simultaneous optimisation: A design methodology for fibre-reinforced composite components. *Proceedings of the Institution of Mechanical Engineers, Part L: Journal of Materials: Design and Applications*, 234(9):1267–1279, 2020.

Thais Campos and Hadas Kress-Gazit. Synthesizing modular manipulators for tasks with time, obstacle, and torque constraints. *arXiv preprint arXiv:2106.09487*, 2021.

Thais Campos, Jeevana Priya Inala, Armando Solar-Lezama, and Hadas Kress-Gazit. Task-based design of ad-hoc modular manipulators. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6058–6064. IEEE, 2019.

Thais Campos de Almeida, Samhita Marri, and Hadas Kress-Gazit. Automated synthesis of modular manipulators' structure and control for continuous tasks around obstacles. *Robotics: Science and Systems 2020*, 2020.

Yang Cao, Shengtai Li, Linda Petzold, and Radu Serban. Adjoint sensitivity analysis for differential-algebraic equations: The adjoint dae system and its numerical solution. *SIAM journal on scientific computing*, 24(3):1076–1089, 2003.

Hongming Chen, Ola Engkvist, Yinhai Wang, Marcus Olivecrona, and Thomas Blaschke. The rise of deep learning in drug discovery. *Drug discovery today*, 23(6): 1241–1250, 2018.

I-Ming Chen and Joel W Burdick. Determining task optimal modular robot assembly configurations. In *proceedings of 1995 IEEE International Conference on Robotics and Automation*, volume 1, pages 132–137. IEEE, 1995.

Yuan Chen and Lin Ye. Topological design for 3d-printing of carbon fibre reinforced composite structural parts. *Composites Science and Technology*, 204:108644, 2021.

Sheng Chu, Mi Xiao, Liang Gao, Yan Zhang, and Jinhao Zhang. Robust topology optimization for fiber-reinforced composite structures under loading uncertainty. *Computer Methods in Applied Mechanics and Engineering*, 384:113935, 2021.

Wan Kyun Chung, Jeongheon Han, Youngil Youm, and SH Kim. Task based design of modular robot manipulator using efficient genetic algorithm. In *Proceedings of International Conference on Robotics and Automation*, volume 1, pages 507–512. IEEE, 1997.

Matteo Cianchetti, Tommaso Ranzani, Giada Gerboni, Thrishantha Nanayakkara, Kaspar Althoefer, Prokar Dasgupta, and Arianna Menciassi. Soft robotics technologies to address shortcomings in today's minimally invasive surgery: the stiff-flop approach. *Soft robotics*, 1(2):122–131, 2014.

Connor W Coley, William H Green, and Klavs F Jensen. Machine learning in computer-aided synthesis planning. *Accounts of chemical research*, 51(5):1281–1289, 2018.

Stelian Coros, Bernhard Thomaszewski, Gioacchino Noris, Shinjiro Sueda, Moira Forberg, Robert W Sumner, Wojciech Matusik, and Bernd Bickel. Computational design of mechanical characters. *ACM Transactions on Graphics (TOG)*, 32(4): 1–12, 2013.

Erwin Coumans. Bullet physics engine. *Open Source Software: http://bulletphysics.org*, 2010.

Andre Luis Ferreira da Silva, Ruben Andres Salas, Emilio Carlos Nelli Silva, and JN Reddy. Topology optimization of fibers orientation in hyperelastic composite material. *Composite Structures*, 231:111488, 2020.

Juan Carlos De los Reyes. *Numerical PDE-constrained optimization*. Springer, 2015.

Eralp Demir, Pouya Yousefi-Louyeh, and Mehmet Yildiz. Design of variable stiffness composite structures using lamination parameters with fiber steering constraint. *Composites Part B: Engineering*, 165:733–746, 2019.

Anubhav Dogra, Srikant Sekhar Padhee, and Ekta Singla. An optimal architectural design for unconventional modular reconfigurable manipulation system. *Journal of Mechanical Design*, 143(6):063303, 2021.

Jørgen S Dokken, Sebastian K Mitusch, and Simon W Funke. Automatic shape derivatives for transient pdes in fenics and firedrake. *arXiv preprint arXiv:2001.10058*, 2020.

Jesse Engel, Matthew Hoffman, and Adam Roberts. Latent constraints: Learning to generate conditionally from unconditional generative models. In *International Conference on Learning Representations*, 2018.

Ronald M Errico. What is an adjoint model? *Bulletin of the American Meteorological Society*, 78(11):2577–2592, 1997.

Samuel Estenlund, Alexandra Tokat, Jonas Engqvist, and Mats Alaküla. Dovetail design for direct cooled rotor: Design and manufacturing. In *2022 International Conference on Electrical Machines (ICEM)*, pages 2121–2127. IEEE, 2022.

Michael Everett, Yu Fan Chen, and Jonathan P How. Motion planning among dynamic, decision-making agents with deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3052–3059. IEEE, 2018.

Haoqiang Fan, Hao Su, and Leonidas Guibas. A point set generation network for 3d object reconstruction from a single image. In *CVPR*, 2017.

Clara Fannjiang and Jennifer Listgarten. Autofocused oracles for model-based design. *Advances in Neural Information Processing Systems*, 33, 2020.

Shane Farritor, Steven Dubowsky, Nathan Rutman, and Jeffrey Cole. A systems-level modular design approach to field robotics. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 4, pages 2890–2895. IEEE, 1996.

Boris Fedulov, Alexey Fedorenko, Aleksey Khaziev, and Fedor Antonov. Optimization of parts manufactured using continuous fiber three-dimensional printing technology. *Composites Part B: Engineering*, 227:109406, 2021.

Alexander IJ Forrester and Andy J Keane. Recent advances in surrogate-based optimization. *Progress in aerospace sciences*, 45(1-3):50–79, 2009.

Justin Fu and Sergey Levine. Offline model-based optimization via normalized maximum likelihood estimation. In *International Conference on Learning Representations*, 2020.

Nuwan Ganganath, Chi-Tsun Cheng, Kai-Yin Fok, and K Tse Chi. Trajectory planning for 3d printing: A revisit to traveling salesman problem. In *2016 2nd International Conference on Control, Automation and Robotics (ICCAR)*, pages 287–290. IEEE, 2016.

Dan Givoli. A tutorial on the adjoint method for inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 380:113810, 2021.

Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

Josif Grabocka, Randolf Scholz, and Lars Schmidt-Thieme. Learning surrogate losses. *arXiv preprint arXiv:1905.10108*, 2019.

Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008.

Mikell P Groover. *Fundamentals of modern manufacturing: materials, processes, and systems*. John Wiley & Sons, 2020.

Yanran Guan, Han Liu, Kun Liu, Kangxue Yin, Ruizhen Hu, Oliver van Kaick, Yan Zhang, Ersin Yumer, Nathan Carr, Radomir Mech, and Hao Zhang. FAME: 3d shape generation via functionality-aware model evolution. *IEEE Transactions on Visualization and Computer Graphics*, 2020.

Sumit Gulwani. Program synthesis. *Software Systems Safety*, pages 43–75, 2014.

Anvita Gupta and James Zou. Feedback gan for dna optimizes protein functions. *Nature Machine Intelligence*, 1(2):105–111, 2019.

Sehoon Ha, Stelian Coros, Alexander Alspach, Joohyung Kim, and Katsu Yamane. Task-based limb optimization for legged robots. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2062–2068. IEEE, 2016.

Sehoon Ha, Stelian Coros, Alexander Alspach, James M Bern, Joohyung Kim, and Katsu Yamane. Computational design of robotic devices from high-level motion specifications. *IEEE Transactions on Robotics*, 34(5):1240–1251, 2018.

Youngwon Hahn and John I Cofer IV. Design study of dovetail geometries of turbine blades using abaqus and isight. In *Turbo Expo: Power for Land, Sea, and Air*, volume 44731, pages 11–20. American Society of Mechanical Engineers, 2012.

Zhong-Hua Han, Ke-Shi Zhang, et al. Surrogate-based optimization. *Real-world applications of genetic algorithms*, 343, 2012.

Steve Hanneke, Liu Yang, et al. Surrogate losses in passive and active learning. *Electronic Journal of Statistics*, 13(2):4646–4708, 2019.

Alex Hawkins-Hooker, Florence Depardieu, Sebastien Baur, Guillaume Couairon, Arthur Chen, and David Bikard. Generating functional protein variants with variational autoencoders. *PLoS computational biology*, 17(2):e1008736, 2021.

Roland Herzog and Karl Kunisch. Algorithms for pde-constrained optimization. *GAMM-Mitteilungen*, 33(2):163–176, 2010.

Gregory S Hornby, Hod Lipson, and Jordan B Pollack. Evolution of generative design systems for modular physical robots. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, volume 4, pages 4146–4151. IEEE, 2001.

Wengang Hu, Runzhong Yu, Mengyao Luo, and Arif Caglar Konukcu. Study on tensile strength of single dovetail joint: experimental, numerical, and analytical analysis. *Wood Material Science & Engineering*, pages 1–9, 2022.

Zheng Hu. A review on the topology optimization of the fiber-reinforced composite structures. *Aerospace technic and technology*, (3):54–72, 2021.

Jiaqi Huang, Qian Chen, Hao Jiang, Bin Zou, Lei Li, Jikai Liu, and Huangchao Yu. A survey of design methods for material extrusion polymer 3d printing. *Virtual and Physical Prototyping*, 15(2):148–162, 2020.

Thomas JR Hughes. *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation, 2012.

I Huněk. On a penalty formulation for contact-impact problems. *Computers & structures*, 48(2):193–203, 1993.

Haoyan Huo, Ziqin Rong, Olga Kononova, Wenhao Sun, Tiago Botari, Tanjin He, Vahe Tshitoyan, and Gerbrand Ceder. Semi-supervised machine-learning classification of materials synthesis procedures. *npj Computational Materials*, 5(1):1–7, 2019.

GY Jeong, MJ Park, JS Park, and KH Hwang. Predicting load-carrying capacity of dovetail connections using the stochastic finite element method. *Wood and Fiber Science*, pages 430–439, 2012.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Multi-objective molecule generation using interpretable substructures. In *International Conference on Machine Learning*, pages 4849–4859. PMLR, 2020.

Wengong Jin, Jonathan M Stokes, Richard T Eastman, Zina Itkin, Alexey V Zakharov, James J Collins, Tommi S Jaakkola, and Regina Barzilay. Deep learning identifies synergistic drug combinations for treating covid-19. *Proceedings of the National Academy of Sciences*, 118(39), 2021.

Taehoon Jung, Jaewook Lee, Tsuyoshi Nomura, and Ercan M Dede. Inverse design of three-dimensional fiber reinforced composites with spatially-varying fiber size and orientation using multiscale topology optimization. *Composite Structures*, 279: 114768, 2022.

SM Fijul Kabir, Kavita Mathur, and Abdel-Fattah M Seyam. A critical review on 3d printed continuous fiber-reinforced composites: History, mechanism, materials and properties. *Composite Structures*, 232:111476, 2020.

Nikos Ath Kallioras, Georgios Kazakis, and Nikos D Lagaros. Accelerated topology optimization by means of deep learning. *Structural and Multidisciplinary Optimization*, 62(3):1185–1212, 2020.

Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

M Khorshidi, M Soheilypour, M Peyro, A Atai, and M Shariat Panahi. Optimal design of four-bar mechanisms using a hybrid multi-objective ga with adaptive local search. *Mechanism and Machine Theory*, 46(10):1453–1465, 2011.

Nathan Killoran, Leo J Lee, Andrew Delong, David Duvenaud, and Brendan J Frey. Generating and designing dna with deep generative models. *arXiv preprint arXiv:1712.06148*, 2017.

Edward Kim, Kevin Huang, Adam Saunders, Andrew McCallum, Gerbrand Ceder, and Elsa Olivetti. Materials synthesis insights from scientific literature via text extraction and machine learning. *Chemistry of Materials*, 29(21):9436–9444, 2017.

J-O Kim and Pradeep K Khosla. A formulation for task based design of robot manipulators. In *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'93)*, volume 3, pages 2310–2317. IEEE, 1993.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Y Kogo, H Hatta, T Sugibayashi, and M Toyoda. Application of carbon-carbon composites to dovetail joint structures. In *Proceedings of the Eighth Japan-US Conference on Composite Materials*, pages 693–702. CRC Press, 2019.

Yasuo Kogo, Hiroshi Hatta, Masaji Toyoda, and Toshio Sugibayashi. Application of three-dimensionally reinforced carbon–carbon composites to dovetail joint structures. *Composites science and technology*, 62(16):2143–2152, 2002.

Hunter T Kollmann, Diab W Abueidda, Seid Koric, Erman Guleryuz, and Nahil A Sobh. Deep learning for topology optimization of 2d metamaterials. *Materials & Design*, 196:109098, 2020.

Slawomir Koziel and Leifur Leifsson. *Surrogate-based modeling and optimization*. Springer, 2013.

Slawomir Koziel and Stanislav Ogurtsov. Surrogate-based optimization. In *Antenna Design by Simulation-Driven Optimization*, pages 13–24. Springer, 2014.

Slawomir Koziel, David Echeverría Ciaurri, and Leifur Leifsson. Surrogate-based methods. In *Computational optimization, methods and algorithms*, pages 33–59. Springer, 2011.

Hans Petter Langtangen and Anders Logg. *Solving PDEs in python: the FEniCS tutorial I*. Springer Nature, 2017.

Jean-Claude Latombe. *Robot motion planning*, volume 124. Springer Science & Business Media, 2012.

Chris Leger and John Bares. Automated task-based synthesis and optimization of field robots. 1999.

Guorui Li, Xiangping Chen, Fanghao Zhou, Yiming Liang, Youhua Xiao, Xunuo Cao, Zhen Zhang, Mingqi Zhang, Baosheng Wu, Shunyu Yin, et al. Self-powered soft robot in the mariana trench. *Nature*, 591(7848):66–71, 2021a.

Hang Li, Liang Gao, Hao Li, Xiaopeng Li, and Haifeng Tong. Full-scale topology optimization for fiber-reinforced structures with continuous fiber paths. *Computer Methods in Applied Mechanics and Engineering*, 377:113668, 2021b.

Nanya Li, Guido Link, Ting Wang, Vasileios Ramopoulos, Dominik Neumaier, Julia Hofele, Mario Walter, and John Jelonnek. Path-designed 3d printing for topological optimized continuous carbon fibre reinforced composite structures. *Composites Part B: Engineering*, 182:107612, 2020.

Lanlan Liu, Mingzhe Wang, and Jia Deng. A unified framework of surrogate loss by refactoring and interpolation. In *European Conference on Computer Vision*, pages 278–293. Springer, 2020.

Anders Logg and Garth N Wells. Dolfin: Automated finite element computing. *ACM Transactions on Mathematical Software (TOMS)*, 37(2):1–28, 2010.

Guowei Ma, Wenwei Yang, and Li Wang. Strength-constrained simultaneous optimization of topology and fiber orientation of fiber-reinforced composite structures for additive manufacturing. *Advances in Structural Engineering*, page 13694332221088946, 2022.

Markforged. Eiger 3d printing software. `https://markforged.com/software`, 2022a.

Markforged. Carbon fiber composite 3d printer: Markforged mark two. `https://markforged.com/3d-printers/mark-two`, 2022b.

Sebastian K Mitusch, Simon W Funke, and Jørgen S Dokken. dolfin-adjoint 2018.1: automated adjoints for fenics and firedrake. *Journal of Open Source Software*, 4 (38):1292, 2019.

Kazunari Miyauchi, Minoru Masuda, and Koji Murata. Analysis of strain distributions of wooden dovetail joints using digital image correlation method. *JOURNAL-SOCIETY OF MATERIALS SCIENCE JAPAN*, 55(4):367, 2006.

Iain Murray, Ryan Adams, and David MacKay. Elliptical slice sampling. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 541–548. JMLR Workshop and Conference Proceedings, 2010.

Gattigorla Nagendar, Digvijay Singh, Vineeth N Balasubramanian, and CV Jawahar. Neuro-iou: Learning a surrogate loss for semantic segmentation. In *BMVC*, page 278, 2018.

Yurii Nesterov et al. *Lectures on convex optimization*, volume 137. Springer, 2018.

Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

Gaoyang Pang, Geng Yang, Wenzheng Heng, Zhiqiu Ye, Xiaoyan Huang, Hua-Yong Yang, and Zhibo Pang. Coboskin: Soft robot skin with variable stiffness for safer human–robot collaboration. *IEEE Transactions on Industrial Electronics*, 68(4): 3303–3314, 2020.

Vasileios S Papapetrou, Chitrang Patel, and Ali Y Tamijani. Stiffness-based optimization framework for the topology and fiber paths of continuous fiber composites. *Composites Part B: Engineering*, 183:107681, 2020.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019a. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019b.

Sarosh Patel and Tarek Sobh. Task based synthesis of serial manipulators. *Journal of advanced research*, 6(3):479–492, 2015.

Yash Patel, Tomáš Hodaň, and Jiří Matas. Learning surrogates via deep embedding. In *European Conference on Computer Vision*, pages 205–221. Springer, 2020.

Daniil Polykovskiy, Alexander Zhebrak, Dmitry Vetrov, Yan Ivanenkov, Vladimir Aladinskiy, Polina Mamoshina, Marine Bozdaganyan, Alexander Aliper, Alex Zhavoronkov, and Artur Kadurin. Entangled conditional adversarial autoencoder for de novo drug discovery. *Molecular pharmaceutics*, 15(10):4398–4405, 2018.

Alex Renda, Yishen Chen, Charith Mendis, and Michael Carbin. Difftune: Optimizing cpu simulator parameters with learned differentiable surrogates. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 442–455. IEEE, 2020.

Alexander Rives, Joshua Meier, Tom Sercu, Siddharth Goyal, Zeming Lin, Jason Liu, Demi Guo, Myle Ott, C Lawrence Zitnick, Jerry Ma, et al. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proceedings of the National Academy of Sciences*, 118(15), 2021.

C Ruiz, PHB Boddington, and KC Chen. An investigation of fatigue and fretting in a dovetail joint. *Experimental mechanics*, 24(3):208–217, 1984.

Daniela Rus and Michael T Tolley. Design, fabrication and control of soft robots. *Nature*, 521(7553):467–475, 2015.

Alexander A Safonov. 3d topology optimization of continuous fiber-reinforced structures via natural evolution method. *Composite Structures*, 215:289–297, 2019.

Hidenori Sasaki and Hajime Igarashi. Topology optimization accelerated by deep learning. *IEEE Transactions on Magnetics*, 55(6):1–5, 2019.

Nico Schlömer. pygmsh: A python frontend for gmsh, 2021. *GPL-3.0 License*.

Martin-Pierre Schmidt, Laura Couret, Christian Gout, and Claus BW Pedersen. Structural topology optimization with smoothly varying fiber orientations. *Structural and Multidisciplinary Optimization*, 62(6):3105–3126, 2020.

Ari Seff, Wenda Zhou, Farhan Damani, Abigail Doyle, and Ryan P Adams. Discrete object generation with reversible inductive construction. In *Advances in Neural Information Processing Systems*, volume 32, pages 10353–10363, 2019.

Ari Seff, Yaniv Ovadia, Wenda Zhou, and Ryan P. Adams. SketchGraphs: A large-scale dataset for modeling relational geometry in computer-aided design. In *ICML 2020 Workshop on Object-Oriented Learning*, 2020.

Torkan Shafighfard, Eralp Demir, and Mehmet Yildiz. Design of fiber-reinforced variable-stiffness composites for different open-hole geometries with fiber continuity and curvature constraints. *Composite Structures*, 226:111280, 2019.

Nurhalida Shahrubudin, Te Chuan Lee, and Rhaizan Ramlan. An overview on 3d printing technology: technological, materials, and applications. *Procedia Manufacturing*, 35:1286–1296, 2019.

Aniruddha V Shembekar, Yeo Jung Yoon, Alec Kanyuck, and Satyandra K Gupta. Trajectory planning for conformal 3d printing using non-planar layers. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 51722, page V01AT02A026. American Society of Mechanical Engineers, 2018.

Sergey Shirobokov, Vladislav Belavin, Michael Kagan, Andrei Ustyuzhanin, and Atilim Gunes Baydin. Black-box optimization with local generative surrogates. In *Workshop on Real World Experiment Design and Active Learning at International Conference on Machine Learning*, 2020.

Ole Sigmund and Kurt Maute. Topology optimization approaches. *Structural and Multidisciplinary Optimization*, 48(6):1031–1055, 2013.

Satinder P Singh, Andrew G Barto, Roderic Grupen, Christopher Connolly, et al. Robust reinforcement learning in motion planning. *Advances in neural information processing systems*, pages 655–655, 1994.

Suwin Sleesongsom and Sujin Bureerat. Four-bar linkage path generation through self-adaptive population size teaching-learning based optimization. *Knowledge-Based Systems*, 135:180–191, 2017.

Jonathan M Stokes, Kevin Yang, Kyle Swanson, Wengong Jin, Andres Cubillos-Ruiz, Nina M Donghia, Craig R MacNair, Shawn French, Lindsey A Carfrae, Zohar Bloom-Ackermann, et al. A deep learning approach to antibiotic discovery. *Cell*, 180(4):688–702, 2020.

Enrico Stragiotti. *Continuous Fiber Path Planning Algorithm for 3D Printed Optimal Mechanical Properties.* PhD thesis, Politecnico di Torino, 2020.

Devika Subramanian et al. Kinematic synthesis with configuration spaces. *Research in Engineering Design*, 7(3):193–213, 1995.

Kentaro Sugiyama, Ryosuke Matsuzaki, Andrei V Malakhov, Alexander N Polilov, Masahito Ueda, Akira Todoroki, and Yoshiyasu Hirano. 3d printing of optimized composites with variable fiber volume fraction and stiffness using continuous fiber. *Composites Science and Technology*, 186:107905, 2020.

Xingyuan Sun, Jiajun Wu, Xiuming Zhang, Zhoutong Zhang, Chengkai Zhang, Tianfan Xue, Joshua B Tenenbaum, and William T Freeman. Pix3d: Dataset and methods for single-image 3d shape modeling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Xingyuan Sun, Tianju Xue, Szymon Rusinkiewicz, and Ryan P Adams. Amortized Synthesis of Constrained Configurations Using a Differentiable Surrogate. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Xingyuan Sun, Geoffrey Roeder, Tianju Xue, Ryan P Adams, and Szymon Rusinkiewicz. More stiffness with less fiber: End-to-end fiber path optimization for 3d-printed composites. *arXiv preprint arXiv:2205.16008*, 2022.

Saleh Tabandeh, William Melek, Mohammad Biglarbegian, Seong-hoon Peter Won, and Chris Clark. A memetic algorithm approach for solving the task-based configuration optimization problem in serial modular and reconfigurable robots. *Robotica*, 34(9): 1979–2008, 2016.

Yonglong Tian, Andrew Luo, Xingyuan Sun, Kevin Ellis, William T Freeman, Joshua B Tenenbaum, and Jiajun Wu. Learning to infer and execute 3d shape programs. In *International Conference on Learning Representations*, 2018.

Brandon Trabucco, Aviral Kumar, Xinyang Geng, and Sergey Levine. Conservative objective models for effective offline model-based optimization. In *International Conference on Machine Learning*, pages 10358–10368. PMLR, 2021.

Jessica Vamathevan, Dominic Clark, Paul Czodrowski, Ian Dunham, Edgardo Ferran, George Lee, Bin Li, Anant Madabhushi, Parantu Shah, Michaela Spitzer, et al. Applications of machine learning in drug discovery and development. *Nature Reviews Drug Discovery*, 18(6):463–477, 2019.

EJ Van Henten, DA Van't Slot, CWJ Hol, and LG Van Willigenburg. Optimal manipulator design for a cucumber harvesting robot. *Computers and electronics in agriculture*, 65(2):247–257, 2009.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020a. doi: 10.1038/s41592-019-0686-2.

Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3):261–272, 2020b.

Michael Yu Wang, Xiaoming Wang, and Dongming Guo. A level set method for structural topology optimization. *Computer methods in applied mechanics and engineering*, 192(1-2):227–246, 2003.

Shaoxiong Wang, Jiajun Wu, Xingyuan Sun, Wenzhen Yuan, William T Freeman, Joshua B Tenenbaum, and Edward H Adelson. 3d shape perception from monocular

vision, touch, and shape priors. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1606–1613. IEEE, 2018.

Ting Wang, Nanya Li, Guido Link, John Jelonnek, Jürgen Fleischer, Jörg Dittus, and Daniel Kupzik. Load-dependent path planning method for 3d printing of continuous fiber reinforced plastics. *Composites Part A: Applied Science and Manufacturing*, 140:106181, 2021.

Julian Whitman and Howie Choset. Task-specific manipulator design and trajectory synthesis. *IEEE Robotics and Automation Letters*, 4(2):301–308, 2018.

Philip Wolfe. Convergence conditions for ascent methods. *SIAM review*, 11(2):226–235, 1969.

Philip Wolfe. Convergence conditions for ascent methods. ii: Some corrections. *SIAM review*, 13(2):185–188, 1971.

Jiajun Wu, Yifan Wang, Tianfan Xue, Xingyuan Sun, Bill Freeman, and Josh Tenenbaum. Marrnet: 3d shape reconstruction via 2.5 d sketches. *Advances in neural information processing systems*, 30, 2017.

Jimmy Wu, Xingyuan Sun, Andy Zeng, Shuran Song, Johnny Lee, Szymon Rusinkiewicz, and Thomas Funkhouser. Spatial action maps for mobile manipulation. In *Proceedings of Robotics: Science and Systems (RSS)*, 2020.

Jimmy Wu, Xingyuan Sun, Andy Zeng, Shuran Song, Szymon Rusinkiewicz, and Thomas Funkhouser. Spatial intention maps for multi-agent mobile manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

Jimmy Wu, Xingyuan Sun, Andy Zeng, Shuran Song, Szymon Rusinkiewicz, and Thomas Funkhouser. Learning pneumatic non-prehensile manipulation with a mobile blower. *IEEE Robotics and Automation Letters*, 2022.

Hong Xiao, Wei Han, Wenbin Tang, and Yugang Duan. An efficient and adaptable path planning algorithm for automated fiber placement based on meshing and multi guidelines. *Materials*, 13(18):4209, 2020.

Tianju Xue. *Computational modeling and design of mechanical metamaterials: A machine learning approach*. PhD thesis, Princeton University, 2022.

Tianju Xue, Alex Beatson, Sigrid Adriaenssens, and Ryan Adams. Amortized finite element analysis for fast pde-constrained optimization. In *International Conference on Machine Learning*, pages 10638–10647. PMLR, 2020a.

Tianju Xue, Alex Beatson, Maurizio Chiaramonte, Geoffrey Roeder, Jordan T Ash, Yigit Menguc, Sigrid Adriaenssens, Ryan P Adams, and Sheng Mao. A data-driven computational scheme for the nonlinear mechanical properties of cellular mechanical metamaterials under large deformation. *Soft matter*, 16(32):7524–7534, 2020b.

Tianju Xue, Thomas J Wallin, Yigit Menguc, Sigrid Adriaenssens, and Maurizio Chiaramonte. Machine learning generative models for automatic design of multi-material 3d printed composite solids. *Extreme Mechanics Letters*, 41:100992, 2020c.

Tianju Xue, Sigrid Adriaenssens, and Sheng Mao. Learning the nonlinear dynamics of mechanical metamaterials with graph networks. *International Journal of Mechanical Sciences*, 238:107835, 2023.

Yusuke Yamanaka, Akira Todoroki, Masahito Ueda, Yoshiyasu Hirano, Ryosuke Matsuzaki, et al. Fiber line optimization in single ply for 3d printed composites. *Open Journal of Composite Materials*, 6(04):121, 2016.

Tian Yuan Yang, Duo Qi Shi, and Zhen Cheng. 2d geometrical parameters optimization design method of cmc/metal dovetail joint. In *Materials Science Forum*, volume 923, pages 156–163. Trans Tech Publ, 2018.

Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli, and Joshua B Tenenbaum. Neural-symbolic vqa: disentangling reasoning from vision and language understanding. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 1039–1050, 2018.

Yonggyun Yu, Taeil Hur, and Jaeho Jung. Deep learning for topology optimization design. *arXiv preprint arXiv:1801.05463*, 2018.

Zhuoning Yuan, Yan Yan, Milan Sonka, and Tianbao Yang. Robust deep auc maximization: A new surrogate loss and empirical studies on medical image classification. *arXiv preprint arXiv:2012.03173*, 2020.

Leen Zhang, Xiaoping Wang, Jingyu Pei, and Yu Zhou. Review of automated fibre placement and its prospects for advanced composites. *Journal of Materials Science*, 55(17):7121–7155, 2020.

Lifeng Zhu, Weiwei Xu, John Snyder, Yang Liu, Guoping Wang, and Baining Guo. Motion-guided mechanical toy modeling. *ACM Transactions on Graphics (TOG)*, 31(6):1–10, 2012.