Preliminary Exploration of Automated Robot Design: Implementation Based on Black-Box Function Optimization

Authors: Zhiyang Xiang, Zhiyang Xiang

Date: 2019-05-05T00:00:00+00:00

Abstract

Robots possess significant potential for reducing physical labor demands and represent a prominent focus of current research. Robot design is an intellectually labor-intensive endeavor, making research on automated design methodologies for robots of considerable practical importance. As existing research seldom addresses automated design, this paper investigates the fundamental framework for automation in robot design. We abstract robot design as a black-box parameter optimization process, wherein a specific robot design is evaluated through physical simulation to ascertain its capability to accomplish a given task. By modifying design parameters, the capability to accomplish tasks is optimized. Specifically, the inputs to the automated design methodology consist of a set of components, a task to be performed by the robot, and typically a robot control training approach based on reinforcement learning, while the output is the configuration of these components. As physical simulation or real-world experimental validation entails substantial time and labor costs, the black-box optimization process must minimize the number of sampling iterations. This paper employs Bayesian parameter optimization to accomplish the design task. Experimental results demonstrate that the proposed method can accomplish an automated design task for a two-joint robot within approximately five simulations.

Full Text

Preamble

TOWARDS AUTOMATIC ROBOT DESIGN: A BLACK-BOX FUNCTION OPTIMIZATION APPROACH

A PREPRINT

Zhiyang Xiang
School of Information Science and Engineering, Jishou University
Hunan Province, China, 416000
sbxzy@foxmail.com
z_xiang@hnu.edu.cn

May 5, 2019

ABSTRACT

Robotics is one of the most heavily funded research areas worldwide, and robots have the potential to significantly reduce human labor. However, robot design is intellectually labor-intensive. Surprisingly, automatic robot design remains a largely unexplored research area to date.

In this paper, we propose an automatic design framework that models the robotics design problem as a black-box function optimization process. Given robot parts and a specific purpose for the robot to accomplish, the framework automatically assembles the parts in an optimal configuration for the task. The framework operates through an interactive simulation and optimization process, where a physics engine (or even real-world simulations) evaluates an assembly configuration based on its ability to perform the given task, and a black-box optimization method maximizes this score. Bayesian hyperparameter optimization is employed for its advantage of requiring fewer samples than other methods. Experiments demonstrate that the proposed method can find acceptable designs for a two-joint system within only five simulations.

Keywords: Robotics · Automatic design · Bayesian optimization

Introduction

It is always desirable for Artificial Intelligence (AI) to carry out more work for humans. As the physical counterpart to AI, automated machines have the potential to substantially reduce human labor. It is particularly desirable that AI can automatically create artifacts for both entertainment and industrial purposes. Robotics, as a combination of mechanics and computer science, represents one of the most funded research areas in AI globally. Despite the maturity of robot control as a field, the problem of automatic robot design has received far less attention. In this work, we propose a simulation-based automatic design method whose results can be evaluated in real-world situations using 3D printers.

Previous studies have considered the automatic design of control logic—the "software" component of robots [**?**]. In [**?**], the auto-assembly problem was investigated; however, this study focused on construction applications where almost all part contacts are fixed in position, making it fundamentally different from robot construction where most parts are not fixed.

Two major challenges arise in automatic robot design: data representation and

optimization efficiency. Robots can be represented as tree structures of individual parts (as in ROS [**?**]), which are discrete data structures, while parts themselves have continuous properties. The fusion of discrete and continuous data representations enables the construction of a continuous search space. Because evaluating robot designs is expensive, the search algorithm in this constructed space should be able to reach optima with minimal evaluations.

In this paper, we study the automatic design of robots. Rather than designing control software or hardware structures for a specific purpose, we devise auto-assembly procedures for general-purpose robots.

2 Methodology

The design of physical structures in the real world has a long evolutionary history spanning billions of years. One reason for this slow pace is that evaluating the fitness of a design (in this case, the physical structure of a species) is time-consuming. The evolution speed of physical structures is partly dependent on evaluation speed, as evidenced by fruit flies widely used in animal experiments. Due to their high reproduction rate, evolutionary processes that normally require generations can be accomplished in a short time.

In the computational world, the evolution of physical structures can be nearly infinitely accelerated by the seemingly unlimited computing power available. In this work, robot design is assisted by accelerated physical simulations. The framework is illustrated in Fig. 1 [Figure 1: see original paper]. The optimization process generates a vector, which is then translated into a physical robot representation. This representation is trained with reinforcement learning according to a specified task. Subsequently, the model is evaluated to determine how well it can accomplish the task, and this evaluation score is optimized. The optimizer should be able to reach the optimum with minimal evaluations, similar to active sampling to minimize sample counts. Thus, the framework constitutes an active design procedure.

Figure 1: Active design: automatic robot design with simulation and optimization interaction

The input to active design consists of a set of parts and their specifications S, a reinforcement learning method L, an evaluation function $\text{Score}(S, J, L)$ that determines how well the robot can accomplish a specified task when parts are connected with joints J, and an optimizer $\arg\max \text{Score}(S, L)$. Due to the representation and efficiency challenges described in the introduction, two problems must be addressed in this framework. First, how to transform the vector representation (the operating objective of black-box optimizers) into the tree structure recognized by physical simulators or real robot constructors. Second, which optimizer to choose to minimize evaluations of the Score function.

2.1 Representation

The representation must be bidirectional: from tree-structured data to vector representation, and from vector to tree-structured parts and their specifications. First, consider the forward transformation, which combines a specification similarity matrix C and a similarity matrix G defined by the tree structure. If structure a differs from structure b, despite similarities in part specifications, their distance in the similarity space will be larger than that of structure c, which shares the same tree structure as a.

The fusion of these two similarity matrices should preserve C (Euclidean distances) as much as possible while following the constraints in $G = \{g_{ij}\}$ calculated from graph similarities. This matrix fusion problem is formalized as:

$$\begin{aligned} &\min_{C'} \sum_{i,j} (c'_{ij} - c_{ij})^2 \\ &\text{s.t.} \quad \forall g_{ij}, g_{mn} \in G, \text{ if } g_{ij} \leq g_{mn}, c'_{ij} \geq c'_{mn} \end{aligned}$$

Because similarity matrices are symmetric, the upper half of G is extracted and stored in a vector $M = \{m_k\}$ sorted in decreasing order. Two additional vectors are created: $D = \{d_k\}$ and $K = \{k_k\}$, where for each element $m_k = g_{ij}$, we have $d_k = p_{ij}$ and $k_k = c'_{ij}$.

The optimization task from Eq. (2) is then transformed as:

$$\min_{K} \sum_{i} (k_i - d_i)^2 \quad \text{s.t. } 0 \leq k_1 \leq k_2 \leq \cdots \leq k_j \leq \cdots$$

which is equivalent to [**?**]. This constitutes a quadratic programming problem. The reverse transformation can be accomplished in two steps. First, find the best structure through nearest neighbor search in the transformed space, since structure similarity is fully preserved. After that, determining the specifications presents no further difficulty.

2.2 Optimization

Bayesian optimization [**?**] is chosen to perform the main work of active design due to its ability to reach optima in few search steps. This optimization model operates under the assumption that the parameters to be selected and the score to be maximized form a Gaussian process. The optimization process combined with simulation steps is given in Algorithm 1.

Algorithm 1: Active design of robots with Bayesian optimization

Input: Specifications $\{S\}$, a reinforcement learning method L, an evaluation function $\text{Score}(S, J, L)$, maximum evaluations N.

Output: Robot structure and joint specifications.

1. Set GP priors.
2. Generate two space-filling joint and structure settings, calculate their vector representations using procedures from Section 2.1, and initialize GP with generated vectors and their scores.
3. Add the calculated vectors $v_1 \in V$, $v_2 \in V$ and their scores to an observation set as $O = \{[v_1^T, \text{Score}(v_1)]^T, [v_2^T, \text{Score}(v_2)]^T\}$.
4. Update GP posterior with observed vector representations and their scores in O.
5. From GP, calculate a position $v' \in V$ in the space of robot vector representations that maximizes the GP posterior.
6. Evaluate Score(v') and add it to O as $[v_n^T, \text{Score}(v_n)]^T$.
7. While $n \leq N$ do:
   - Continue optimization loop
8. End while
9. Output $\arg\max_{[v_n^T, \text{Score}(v_n)]^T \in O} \text{Score}(v_n)$

3 Experiments

Two toy problems were studied using ROS and PyBullet [**?**] simulation tools. In the first problem, the input consists of tree links with cylindrical shapes and joint positions at the ends. The task is to "stand up," meaning the system must lift one piece while keeping two other pieces in contact with the ground. Random initialization with a simple reinforcement learning setup results in the situation shown in Fig. 2 Figure 2: see original paper. After five evaluations with active design, the robot achieves the configuration that yields the highest score, where the central link is higher than in other configurations, as illustrated in Fig. 2(b). The deterministic policy gradient reinforcement learning method [**?**] was selected for the experiments.

In the second problem, the system attempts to form a diagonal gesture. Random setup results in Fig. 3 Figure 3: see original paper because the rotation axes of joints do not permit a diagonal gesture. After seven steps of active design, the gesture is successfully accomplished as shown in Fig. 3(b).

Comparisons were conducted using SciPy' s [**?**] Nelder-Mead optimization method [**?**]. The SciPy module can achieve similar results to Bayesian optimization but requires more samples. The results are presented in Table 1 .

(a) Random setup
(b) Joint optimized with active design

Figure 2: Problem one–optimize joint configurations so that the central link can stay as high as possible.

(a) Random setup
(b) Joint optimized with active design

Figure 3: Problem two–optimize joint configurations so that a diagonal gesture can be performed.

Table 1: Function evaluation times comparison between Bayesian optimization and SciPy' s optimization module.

Optimization method	Problem one	Problem two
Proposed		
SciPy		

Each simulation takes approximately 10 seconds on a mainstream PC. Consequently, the proposed method can complete design in minutes, while hours are required using the SciPy module. This is because many optimization methods are designed under the assumption that function evaluations are fast, which is not the case for robot simulations or evaluations.

References

[1] Gianpiero Francesca, Manuele Brambilla, Arne Brutschy, Vito Trianni, and Mauro Birattari. Automode: A novel approach to the automatic design of control software for robot swarms. *Swarm Intelligence*, 8(2):89–112, 2014.

[2] Y. Terada and S. Murata. Automatic assembly system for a large-scale modular structure—hardware design of module and assembler robot. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE Cat. No.04CH37566), volume 3, pages 2349–2355 vol.3, Sep. 2004.

[3] Anis Koubaa. *Robot Operating System (ROS)*. 2016.

[4] Zhiyang Xiang, Zhu Xiao, Yourong Huang, Dong Wang, Bin Fu, and Wenjie Chen. Unsupervised and semi-supervised dimensionality reduction with self-organizing incremental neural network and graph similarity constraints. In James Bailey, Latifur Khan, Takashi Washio, Gill Dobbie, Joshua Zhexue Huang, and Ruili Wang, editors, *Advances in Knowledge Discovery and Data Mining*, pages 191–202, Cham, 2016. Springer International Publishing.

[5] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS' 12, pages 2951–2959, USA, 2012. Curran Associates Inc.

[6] pybullet: Bullet real-time physics simulation, available at https://pybullet.org.

[7] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML' 14, pages I–387–I–395. JMLR.org, 2014.

[8] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 2019.05.05].

[9] Fuchang Gao and Lixing Han. Implementing the nelder-mead simplex algorithm with adaptive parameters. *Computational Optimization and Applications*, 51(1):259–277, Jan 2012.

Note: Figure translations are in progress. See original paper for figures.