Hardware as Policy: Mechanical and Computational Co-Optimization using Deep Reinforcement Learning

Tianjian Chen[*] **Zhanpeng He**[*] **Matei Ciocarlie**
Columbia University
{tianjian.chen, zhanpeng.he, matei.ciocarlie}@columbia.edu

Abstract: Deep Reinforcement Learning (RL) has shown great success in learning complex control policies for a variety of applications in robotics. However, in most such cases, the hardware of the robot has been considered immutable, modeled as part of the environment. In this study, we explore the problem of learning hardware and control parameters together in a unified RL framework. To achieve this, we propose to model the robot body as a "hardware policy", analogous to and optimized jointly with its computational counterpart. We show that, by modeling such hardware policies as auto-differentiable computational graphs, the ensuing optimization problem can be solved efficiently by gradient-based algorithms from the Policy Optimization family. We present two such design examples: a toy mass-spring problem, and a real-world problem of designing an underactuated hand. We compare our method against traditional co-optimization approaches, and also demonstrate its effectiveness by building a physical prototype based on the learned hardware parameters. Videos and more details are available at `https://roamlab.github.io/hwasp/`.

Keywords: Mechanical-Computational Co-Optimization, Reinforcement Learning

1 Introduction

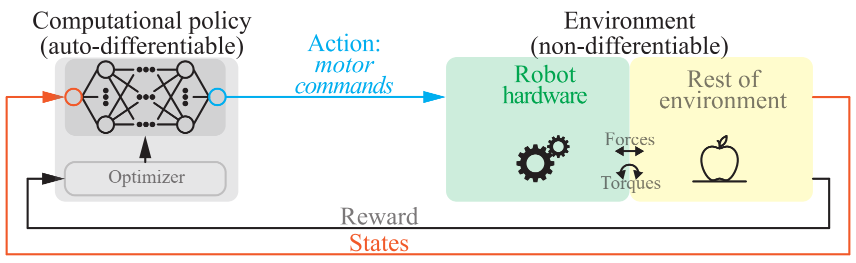
Human "intelligence" resides in both the brain and the body: we can develop complex motor skills, and the mechanical properties of our bones and muscles are also adapted to our daily tasks. Numerous motor skills exhibit this phenomenon, from running (where the stiffness of the Achilles tendon has been shown to maximize locomotion efficiency [1]) to grasping (where coordination patterns between finger joints emerge from both synergistic muscle control and mechanical coupling of joints [2]). Mechanical adaptation and motor skill improvement can happen simultaneously, both over an individual's lifetime (e.g. [3]) and at evolutionary time scales — for example, it has been suggested that, as early hominids practiced throwing and clubbing, hand morphology also changed accordingly, e.g. the thumb got longer to provide better opposition [4].

In robotics, the idea of jointly designing/optimizing mechanical and computational components has a long track record with remarkable advances, exploiting the fact that the morphology, transmissions, and control policies are tightly connected by the laws of physics and co-determine robot behavior. If the control policy and dynamics can both be modeled analytically, traditional optimization can derive the desired values for hardware and policy parameters. When such an approach is not feasible (e.g. due to complex policies or dynamics), evolutionary computation has been used instead. However, these methods still have difficulty learning sophisticated motor skills in complex environments (e.g. with partially observable states, dynamics with transient contacts), or are sample-inefficient.

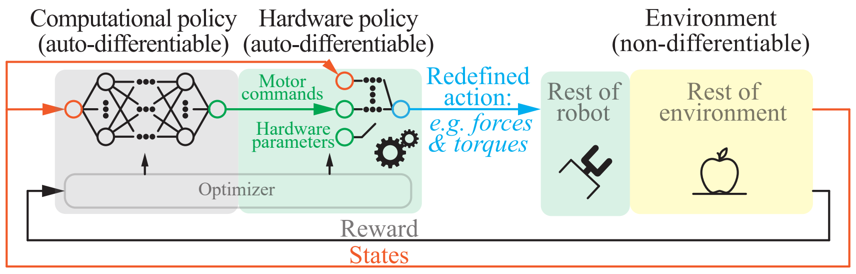
In contrast, recent advances in Deep Reinforcement Learning (Deep RL) have shown great potential for learning difficult motor skills despite having only partial information of complex, unstructured environments (e.g. [5, 6, 7]). Traditionally, the output of a Deep RL policy in robotics consists of motor commands, and the robot hardware converts these motor commands to effects on the external world (usually through forces and/or torques). In this conventional RL perspective, robot hardware is considered given and immutable, essentially treated as part of the environment (Fig 1a).

In this work we bring a different perspective for hardware in RL. Consider the concrete example of an underactuated robot hand. Motor forces are converted into joint torques by a transmission mechanism

[*]Authors have contributed equally.



(a) Traditional perspective — Reinforcement Learning with a purely computational policy



(b) Hardware as Policy — computational graph implementation

Figure 1: Hardware as Policy overview. From the traditional perspective (a), all robot hardware is part of the simulated environment. In the proposed method (b), aspects of robot hardware are formulated as a "hardware policy" implemented as a computational graph, then optimized jointly with the computational policy.

(gears, tendons or linkages). Through careful design of the hardware parameters, such a transmission can provide desired grasping behavior for a wide range of objects (e.g. [8, 9]). Such a transmission is conceptually akin to a policy, mapping an input (motor forces) to an output (joint torques) with carefully tuned parameters leading to beneficial effects for overall performance.

Can we leverage the power of Deep RL for co-optimization of the computational and mechanical components of a robot? High-fidelity physics simulation and effective sim-to-real transfer (where a policy is trained on a physics simulator and only then deployed on real hardware [10]) provide such an opportunity, since they allows modifications of design parameters during training without incurring the prohibitive cost of re-building hardware. A straightforward option to optimize hardware parameters during training is to treat them as hyperparameters of the RL algorithm. However, this approach usually carries a massive computational cost.

In this study, we propose to consider *hardware as policy*, optimized jointly with the traditional computational policy. As is well known, a model-free Policy Optimization (e.g. [11, 12]) or Actor-critic (e.g. [13]) algorithm can train using an auto-differentiable agent/policy and a non-differentiable black-box environment. The core idea we propose is to move part of the robot hardware from the non-differentiable environment into the auto-differentiable agent/policy (Fig 1b). In this way, hardware parameters[2] become parameters in the policy graph, analogous to the neural network weights and biases. Therefore, the optimization of hardware parameters can be directly incorporated into the existing RL framework, and can use existing learning algorithms with changes to the computational graphs as we will describe here. We summarize our major contribution as follows:

- To the best of our knowledge, we are the first to express hardware and computation as a unified RL policy which allows the optimization algorithm to propagate gradients of the actions w.r.t both hardware and computational parameters simultaneously.
- Via case studies comprising both a toy problem and a real-world design challenge, we show that such gradient-based methods are superior to hyperparameter tuning as well as gradient-free evolutionary strategies for hardware-policy co-optimization.
- To the best of our knowledge, we are the first to build a physical prototype and deploy the trained policy on it to validate a Deep RL-based hardware-policy co-optimization approach.

[2]This paper primarily focuses on the mechanical aspect of hardware, we use the terms "hardware" and "mechanics/mechanical" interchangeably. However, we believe that, in the future, the proposed idea could be extended to electrical or sensorial aspects of a physical device.

2 Related Work

A first category of related work comprises studies using analytical dynamics and classical control [14], a number of which optimized mechanical and control or planning parameters for legged locomotors [15, 16, 17]. All studies above require an analytical model of the complete mechanical-control system, which is non-trivial in complex problems. More recent work uses classical control but evaluates and iterates on real hardware [18], applying Bayesian Optimization to micro robots. However, the goal is different from ours: this study aims to decrease the number of real-world design evaluations, which is avoided in our work by training in simulation and performing sim-to-real transfer.

Evolutionary computation provides another way to approach this problem. This research path originated from studies on the evolution of artificial creatures [19], where the morphology and the neural systems are both encoded as graphs and generated using genetic algorithms. Lipson and Pollack [20] introduced the automatic lifeform design technique using bars, joints, and actuators as building blocks of the morphology, with neurons attached to them as controllers. A series of works from Cheney et al. [21, 22] studied the morphology-computation co-evolution of cellular automata, in the context of locomotion. Nygaard et al. [23] presented a method that optimizes the morphology and control of quadruped robot using real-world evaluation of the robot. Evolutionary strategies, which are gradient-free, have significant promise, but also exhibit high computational complexity and data-inefficiency compared to recent gradient-based optimization methods.

The recent influx of reinforcement learning provides a new perspective on the co-optimization problem. Ha [24] augmented the REINFORCE algorithm with rewards calculated using the mechanical parameters. Schaff et al. [25] proposed a joint learning method to construct and control the agent, which models both design and control in a stochastic fashion and optimizes them via a variation of Proximal Policy Optimization (PPO). Vermeer et al. [26] perform two-dimensional linkage mechanism synthesis using a Decision-Tree-based mechanism representation fused with RL. Luck et al. [27] presented a method for data-efficient co-adaptation of morphology and behaviors based on Soft Actor-Critic (SAC), leveraging previous information to estimate the performance of new candidates. In all the studies above, hardware parameters are still optimized iteratively and separately from the computational policies, whereas we aim to optimize both together in a unified framework. In addition, none of these works show physical prototypes based on the co-optimized agent.

Recent work on auto-differentiable physics [28, 29, 30, 31] is also relevant to us, as we rely on modeling (part of) the robot hardware as an auto-differentiable computational graph. We hope to make use of advances in general differentiable physics simulation in future iterations of our method.

3 Preliminaries

We start from a standard RL formulation, where the problem of optimizing an agent to perform a certain task can be modeled as a Markov Decision Process (MDP), represented by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{F}, \mathcal{R})$, where $\mathcal{S}$ is state space, $\mathcal{A}$ is the action space, $\mathcal{R}(\boldsymbol{s}, \boldsymbol{a})$ is the reward function, and $\mathcal{F}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$ is the state transition model ($\boldsymbol{s}, \boldsymbol{s}' \in \mathcal{S}$, $\boldsymbol{s}'$ is for the next time step, $\boldsymbol{a} \in \mathcal{A}$). Behavior is determined by a computational control policy $\pi_{\boldsymbol{\theta}}^{comp}(\boldsymbol{a}|\boldsymbol{s})$, where $\boldsymbol{\theta}$ represents the parameters of the policy. Usually, $\pi_{\boldsymbol{\theta}}^{comp}$ is represented as a deep neural network, with $\boldsymbol{\theta}$ consisting of the network's weights and biases. The goal of learning is to find the values of the policy parameters that maximize the expected return $\mathbb{E}[\sum_{t=0}^{T} \mathcal{R}(\boldsymbol{s}_t, \boldsymbol{a}_t)]$ where T is the length of episode.

We start from the observation that, in robotics, in addition to the parameters $\boldsymbol{\theta}$ of the computational policy, the design parameters of the hardware itself, denoted here by $\boldsymbol{\phi}$, play an equally important role for task outcomes. In particular, hardware parameters $\boldsymbol{\phi}$ help determine the output (the effect on the outside world) that is produced by a given input to the hardware (motor commands). This is analogous to how computational parameters $\boldsymbol{\theta}$ help determine the output of the computational policy (action $\boldsymbol{a}$) that is produced by a given input (state or observations $\boldsymbol{s}$).

Even though this analogy exists, traditionally, these two classes of parameters have been treated very differently in RL. Computational parameters can be optimized via gradient-based methods: for example, in Policy Optimization methods such as Trust Region Policy Optimization (TRPO) [11] and Proximal Policy Optimization (PPO) [12], the parameters of the computational policy are optimized by computing and following the policy gradient: $\boldsymbol{g} = \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}^{comp}}[\sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}^{comp}(\boldsymbol{a}_t|\boldsymbol{s}_t) A_t(\boldsymbol{s}_t, \boldsymbol{a}_t)]$ (A_t is the advantage function). In contrast, hardware is generally considered immutable, and modeled as part of the environment. Formally, this means that hardware parameters $\boldsymbol{\phi}$ are considered as parameters of the transition function $\mathcal{F} = \mathcal{F}_{\boldsymbol{\phi}}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$ instead of the policy. This is the concept illustrated in Fig. 1a. Such a formulation is grounded in the most general RL framework, where $\mathcal{F}$ is not modeled analytically, but only observed by execution on real hardware. In such a case, changing $\boldsymbol{\phi}$ can only be done by building a new prototype, which is generally impractical.

However, in recent years, the robotics community has made great advances in training via a computational model of the transition function $\mathcal{F}$, often referred to as a physics simulator (e.g. [32]). The main drivers have been the need to train using many more samples than possible with real hardware, and ensure safety during training. Recent results have indeed shown that it is often possible to train exclusively using an imperfect analytical model of $\mathcal{F}$, and then transfer to the real world [10].

In our context, training with such physics simulator opens new possibilities for hardware design: we can change the hardware parameters $\boldsymbol{\phi}$ and test different hardware configurations on-the-fly inside the simulator, without incurring the cost of re-building a prototype.

4 Hardware as Policy

The Hardware as Policy method (HWasP) proposed here largely aims to perform a similar optimization for hardware parameters as we do for computational policy parameters, i.e. by computing and following the gradient of action probabilities w.r.t such parameters.

The core of the HWasP method is to model the effects of the robot hardware we aim to optimize separately from the rest of the environment. We refer to this component as a "hardware policy", and denote it via $\pi^{hw}_{\boldsymbol{\phi}}(\boldsymbol{a}^{new}|\boldsymbol{s}, \boldsymbol{a})$. The input to the hardware policy consists of the action produced by the computational policy (i.e. a motor command) and other components of the state; the output is in a redefined action space $\mathcal{A}^{new}$ further discussed below.

In the traditional formulation outlined so far, the "hardware policy" and its parameters $\boldsymbol{\phi}$ are included in the transition function $\mathcal{F}_{\boldsymbol{\phi}}$. With HWasP, $\pi^{hw}_{\boldsymbol{\phi}}$ becomes part of the agent. The new overall policy $\pi_{\boldsymbol{\theta},\boldsymbol{\phi}} = \pi^{hw}_{\boldsymbol{\phi}}(\boldsymbol{a}^{new}|\boldsymbol{s}, \boldsymbol{a})\pi^{comp}_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})$ comprises the composition of both computational and mechanical policies, while the new transition probability $\mathcal{F}^{new} = \mathcal{F}^{new}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a}^{new})$ encapsulates the rest of the environment. In other words, we have split the simulation of the environment: one part consists of the mechanical policy, now considered part of the agent, while the other simulates the rest of the robot, and the external environment. The reward function, $\mathcal{R}(\boldsymbol{s}, \boldsymbol{a})$, is redefined to be associated with the new action space: $\mathcal{R}^{new}(\boldsymbol{s}, \boldsymbol{a}^{new})$. Once this modification is performed, we run the original Policy Optimization algorithm on the new tuple $(\mathcal{S}, \mathcal{A}^{new}, \mathcal{F}^{new}, \mathcal{R}^{new})$ as redefined above. However, in order for this to be feasible, two conditions have to be met:

Condition 1: The redefined action vector $\boldsymbol{a}^{new}$ must encapsulate the interactions between the mechanical policy and the rest of the environment. In other words, this new action interface must comprise all the ways in which the hardware we are optimizing effects change on the rest of the environment. Furthermore, the redefined action vector must be low-dimensional enough to allow for efficient optimization. Such an interface is problem-specific. Forces / torques between the robot and the environment make good candidates, as we will exemplify in the following sections.

Condition 2: To use Policy Optimization algorithms, we need to efficiently compute the gradient of the redefined action probability w.r.t. hardware parameters. We further discuss this condition next.

Computational Graph Implementation (HWasP). In order to meet Condition 2 above, *we propose to simulate the part of hardware we care to optimize as a computational graph.* In this way, gradients can be computed by auto-differentiation and can flow or back-propagate through the hardware policy. Similar to the computational policy, the gradient of log-likelihood of actions w.r.t mechanical parameters $\boldsymbol{\phi}$ can be computed as $\nabla_{\boldsymbol{\phi}} \log \pi^{hw}_{\boldsymbol{\phi}}(\boldsymbol{a}^{new}|\boldsymbol{a}, \boldsymbol{s})$.

Critically, since the computational policy is also expressed as a computational graph, the gradient can back-propagate through both the hardware and computational policies, and the hardware and computational parameters are jointly optimized. This general idea is illustrated in Fig. 1b.

However, this approach is predicated on being able to simulate the effects of the hardware being optimized as a computational graph. Once again, the exact form of this simulation is problem-specific, and can be considered as a key part of the algorithm. In the next sections, we illustrate how this can be done both for a toy problem, and for a real-world design problem, and regard these implementations as an intrinsic part of the contribution of this work.

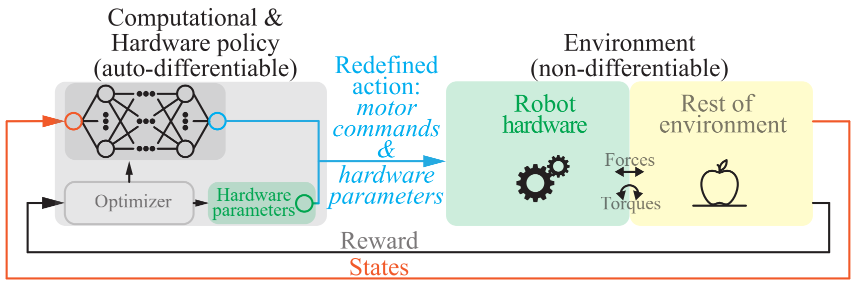


Figure 2: Hardware as Policy-Minimal

Minimal Implementation (HWasP-Minimal). In the general case, where should the split between the (differentiable) hardware policy and the (non-differentiable) rest of the environment simulation be performed? In particular, what if the hardware we care to optimize does not lend itself to a differentiable simulation using existing methods?

Even in such a case, we argue that a "minimal" hardware policy is always possible: we can simply put the hardware parameters into the output layer of the original computational policy. In this case, $\boldsymbol{a}^{new} = [\boldsymbol{a}, \boldsymbol{\phi}]^T$. Here, the policy gradient with respect to the hardware parameters is trivial but can be still useful to guide the update of parameters. When this case in implemented in practice, the transition function $\mathcal{F}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a}^{new})$ typically operates in two steps: first, it sets the new values of the hardware parameters to the underlying simulator, then advances the simulation to the next step.

We illustrate HWasP-Minimal in Fig. 2, which can be compared to HWasP as illustrated in Fig. 1b. HWasP-Minimal is simple to implement since it does not require a physics-based auto-differentiable hardware policy. As shown in our results, we found that HWasP-Minimal performs at least as well as or better than our baselines, but still below HWasP.

Comparison Baselines. We compare HWasP and HWasP-Minimal with the following baselines:

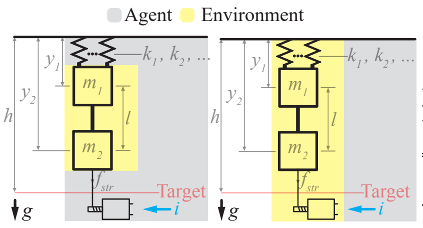
- CMA-ES/ARS with RL inner loop: we treat hardware parameters as hyperparameters, optimized in an outer loop using an evolutionary algorithm while the computational policy is learned (by RL algorithms such as PPO or TRPO) in an inner loop, for each set of hardware parameters. We tested two evolutionary algorithms: Covariance Matrix Adaptation - Evolution Strategy (CMA-ES) [33], and Augmented Random Search (ARS) [34]. While ARS was originally tested on linear models, it can also be applied to non-linear systems (like our computational and mechanical policies) where it still represents a useful baseline.
- CMA-ES/ARS: we use CMA-ES/ARS as gradient-free evolutionary strategies to directly learn both computational policy and hardware parameters, without a separate inner loop.

5 A Mass-spring Toy Problem

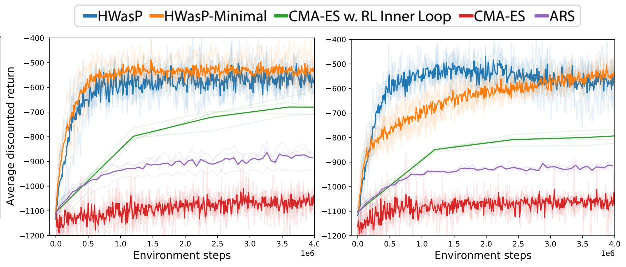
We present a one-dimensional example on the mass-spring system in Fig. 3a. Two point masses, connected by a massless bar, are hanging under n parallel springs with stiffnesses $k_1, \ldots, k_n$. A motor can pull the lower mass by a string. The behavior is governed by a computational policy regulating the motor current, and by the hardware parameters (spring stiffnesses). We note that only the sum of spring stiffnesses matters since they are in parallel, but we still consider each stiffness as an individual parameter to test how our methods scale up for higher-dimensional problems.

The input to the computational policy consists of y_2 and $\dot{y_2}$, and the output is motor current i. The goal is to optimize both the computational policy and the hardware parameters $k_1, \ldots, k_n$ such that the lower mass goes to the red target line ($y_2 = h$) and stay there with minimum motor effort. (The exact formulation for the reward function we use is presented in Supplementary Materials A.)

Hardware as Policy. In this case, we include the effect of the parallel springs in the mechanical policy. Using Hooke's Law, we model spring effects as a computational graph, with $\boldsymbol{k} = [k_1, \ldots, k_n]$ as parameters. The redefined action $\boldsymbol{a}^{new}$ consists of the total resultant force $f_{total} = f_{str} - f_{spr}$. The transition function $\mathcal{F}$ (rest of the environment) implements Newton's Law for the two masses, assuming f_{total} as an external force. Details about the implementation of the computational graph can be found in Fig. 7 in the Supplementary Materials.



(a) Problem descriptions, left: HWasP, right: HWasP-Minimal.



(b) Training curves, left: 10 parameters, right: 50 parameters.

Figure 3: The mass-spring toy problem.

Hardware as Policy — Minimal. In this method, we simply re-define the action vector to also include spring stiffnesses: $\boldsymbol{a}^{new} = [i, k_1, \cdots, k_n]^T$. The transition function $\mathcal{F}$ is responsible for modeling the dynamics of the springs and the two masses.

Results. Fig. 3b shows the comparison of the training curves for both implementations of our method, as well as other baselines, for two cases: 10 and 50 parallel springs. In both cases, HWasP learns an effective joint policy that moves the lower mass to the target position. HWasP-Minimal works equally well for the smaller problem, but suffers a drop in performance as the number of hardware parameters increases. CMA-ES with RL inner loop also learns a joint policy to some extent, but learns slower than our method, especially for the larger problem. CMA-ES by itself does not exhibit any learning behavior over the number of samples tested. ARS achieves better performance than CMA-ES but is still worse than HWasP, HWasP-Minimal and CMA-ES with RL inner loop. For the numerical results of the optimized stiffnesses, please refer to the Supplementary Materials.

To test if our method can also optimize geometric parameters, we also performed co-optimization of the bar length and the control policy. This additional design case is detailed in Supplementary Materials.

6 Co-Design of an Underactuated Hand

In this section we show how HWasP can be applied to a real-world design problem: optimizing the mechanism and the control policy for an underactuated robot hand. The high-level design goal, inspired by previous work [35], is to design a robot hand that is compact, but still versatile (able to grasp different shaped objects, as shown in Fig 4). To achieve the stated compactness goal, all joints are driven by a single motor, via an underactuated transmission mechanism: one motor actuates all joints by tendons in the flexion direction (see Fig 4). Finger extension is passive, via preloaded torsional springs. The mechanical parameters that govern the behavior of this mechanism consist of tendon pulley radii in each joint, as well as stiffness values and preload angles for restoring springs.

Here, we look to simultaneously optimize the hardware parameters along with a computational policy that determines how to position the hand and use the hand motor. The input to the computational policy consists of palm and object positions, object size, and current motor travel and torque. Its output contains a relative position setpoint for hand motor travel as well as palm position commands.

From a hardware perspective, we aim to optimize all the underactuated transmission parameters listed above. We note that, in this work, we do not try to optimize the kinematic structure or topology for the hand. Unlike the underactuated transmission, these aspects do not lend themselves to parameterization and implementation as computational graphs, preventing the use of the HWasP method in its current form. While HWasP-Minimal could still be applied, we leave that for future investigations.

We tested our method with two grasping tasks: top-down grasping with only z-axis motion for the palm movement (Z-Grasp), and top-down grasping with 3-dimensional palm motion (3D-Grasp). The former is a simplified problem version of the latter, and easier to train. Since hardware parameters can be large in scale comparing to weights and biases in the neural network, a small change can lead to a large shift of the joint policy output distribution during training, which in turn can result in local optimum in the reward landscape. To alleviate this issue, we add scaling factors for parameters in the hardware policy computational graph. We use TRPO [11] as the underlying RL algorithm as it allows for hard constraints on action distribution changes. Additional details on problem formulation and training can be found in Supplementary Materials.

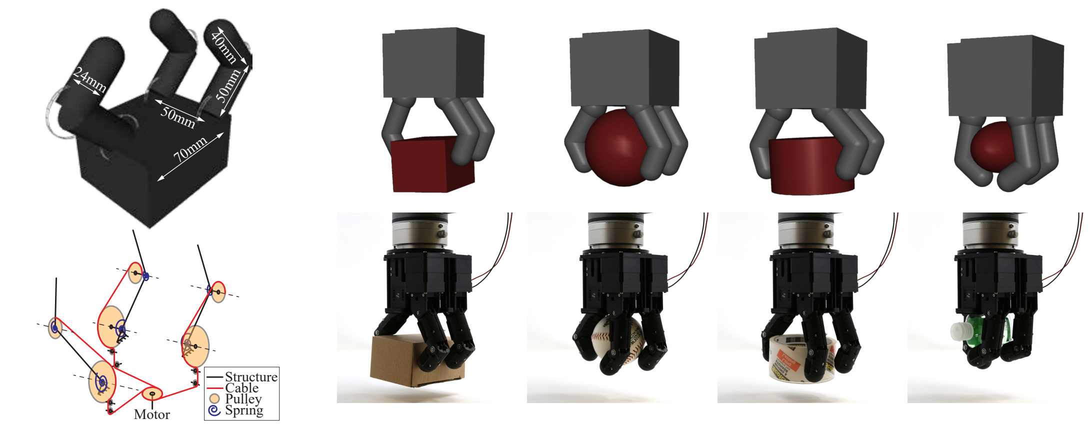


Figure 4: Hand design optimization problem. Left: hand kinematics, dimension, and tendon routing. Right: successful grasps executed in simulation and on a real hand prototype.

We also apply Domain Randomization [10] in the training to increase the chance of successful sim-to-real transfer. We randomized object shape, size, weight, friction coefficient and inertia, injected sensor and actuation noise, and applied random disturbance wrenches on the hand-object system.

Hardware as Policy. In this case, we model the complete underactuated transmission as a computational graph and include it in our mechanical policy. The input to the mechanical policy consists of the commanded motor travel (output by the computational policy), as well current joint angles. Its output consists of hand joint torques. To perform this computation, we use a tendon model that computes the elongation of the tendon in response to motor travel and joint positions, then uses that value to compute tendon forces and joint torques. Details of this model as well as its implementation as an auto-differentiable computational graph can be found in Supplementary Materials B.

The redefined action $\boldsymbol{a}^{new}$ contains the palm position command output by the computational policy, and the joint torques produced by the mechanical policy. The rest of the environment comprises the hand-object system without the tendon underactuation mechanisms, i.e. with independent joints.

Hardware as Policy — Minimal. In this case, all hardware parameters are simply appended to the output of the computational policy. The underactuated transmission model is part of the environment, along with the rest of the hand as well as the object.

Results. Our results are shown in Fig. 5. For Z-Grasp (left plot), HWasP learns an effective computational/hardware policy. HWasP-Minimal does the same, but at a slower pace. Neither evolutionary strategy shows any learning behavior over a similar number of training steps.

To gain additional insight, we also tried an easier version of the same problem with the search range for the hardware parameters reduced by a factor of 8 (middle plot). Here, some of our baselines can also learn effective policies, particularly with an RL inner loop, but HWasP is still the most efficient.

Finally, we investigated performance for the more complex 3D-Grasp task. With a large search range, neither method was able to learn. However, with a reduced search range, HWasP learned an effective policy, while neither CMA-ES-based method displayed any learning behavior over a similar timescale. The values of the optimized hardware parameters are shown in the Supplementary Materials.

Validation with Physical Prototype. To validate our results in the real world, we physically built the hand with the parameters resulted from the co-optimization. The hand is 3D printed, and actuated by a single position-controlled servo motor. Fig. 4 shows grasps obtained by this physical prototype, compared to their simulated counterparts. All shown grasps are stable and allow object lift and transport. We note that, as expected, the hand is highly versatile and can perform a wide range of both fingertip and enveloping grasps, for objects of varying shape and size. This shows that the optimized hardware policy is indeed effective in the real world.

Furthermore, we tested the combined hardware and computational policies for both Z-Grasp and 3D-Grasp on the real hand. Here, the computational policy determined how to position the hand (implemented in practice using a UR5 robot with position control) and also issued all commands to the hand servo. While driving the optimized hand, the computational policy achieved 100% success rate for Z-Grasp with boxes and balls and 90% with cylinders. For 3D-Grasp, the success rate was

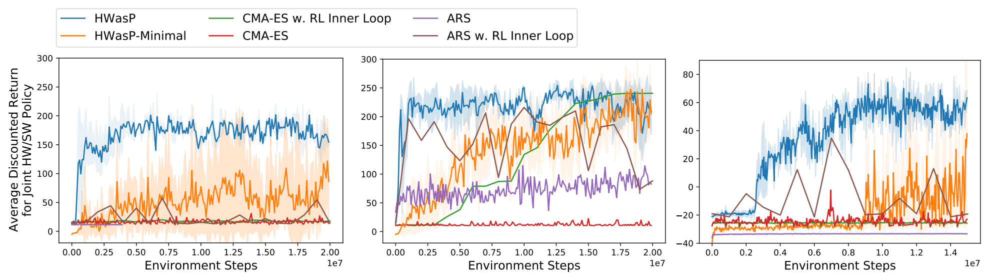


Figure 5: Training curves for the grasping problem. Left: Z-Grasp with a large hardware parameter search range. Middle: Z-Grasp with a small hardware search range. Right: 3D-Grasp with a small search range.

100% on boxes, 80% on cylinders and 50% on balls. We note however that the computational policy expects object pose as input, which was not available on our experimental setup; instead, we started each run with the object in a known location, which was provided as part of the observation. However, on the ball object, early contacts occasionally caused untracked object displacement, which lowered the success rate. While we hope to train and test more complex computational policies in the future, including with real sensor feedback, we believe these experiments underlined the effectiveness of the hardware policy, and its ability to operate together with the jointly optimized computational policy.

7 Disscussion and Conclusion

Our results show that the HWasP approach is able to learn combined computational and mechanical policies. We attribute this performance to the fact that HWasP connects different hardware parameters via a computational graph based on the laws of physics, and can provide the physics-based gradient of the action probability w.r.t the hardware parameters. The HWasP-Minimal implementation does not provide such information, and the policy gradient can only be estimated via sampling, which is usually less efficient, particularly for high-dimensional problems. In consequence, HWasP-Minimal also shows the ability to learn effective policies, but with reduced performance.

Compared to gradient-free evolutionary baselines for joint hardware-software co-optimization, HWasP always learns faster, while HWasP-Minimal is at least as effective as the best baseline algorithm. We note that combining an RL inner loop for the computational policy with a CMA-ES outer loop for hardware parameters proved more effective than directly using CMA-ES for the complete problem. Still, HWasP outperforms both methods.

The biggest advantage of HWasP-Minimal is that, like gradient-free methods, it does not depend on auto-differentiable physics, and is widely applicable with straightforward implementations to various problems using existing non-differentiable physics engines. We believe that our methods represent a step towards a framework where an algorithm designer can ”tune the slider” to decide how much physics to include in the computational policy, based on the trade-offs between computation efficiency, ease of development, and the availability of auto-differentiable physics simulations.

In its current stage, our work still presents a number of limitations. The computational aspects of the policies we have explored so far are relatively simple (e.g. limiting hand motion to 1- or 3-DOF). We hope to explore more challenging robotic tasks, including 6-DOF hand positioning. Here, we also used HWasP with a single task, but believe it is possible to learn more robust hardware parameters by trying to generalize to multiple tasks. Finally, we aim to include additional hardware aspects in the optimization, such as mechanism kinematics, morphology, or link dimensions.

We believe the proposed idea of considering hardware as part of the policy can enable co-design of hardware and software using existing RL toolkits, with new computational graph structures but little change in the learning algorithms. We hope this work can open new opportunities for task-based hardware-software co-design of intelligent systems, for researchers in both RL and hardware design.

Acknowledgments

The authors would like to thank Roberto Calandra for insightful discussions and suggestions. This work was supported in part by ONR Grant N00014-16-1-2026 and by a Google Faculty Award.

References

[1] G. Lichtwark and A. Wilson. Is achilles tendon compliance optimised for maximum muscle efficiency during locomotion? *Journal of biomechanics*, 40(8):1768–1775, 2007.

[2] M. Santello, G. Baud-Bovy, and H. Jörntell. Neural bases of hand synergies. *Frontiers in computational neuroscience*, 7:23, 2013.

[3] R. Hammami, A. Chaouachi, I. Makhlouf, U. Granacher, and D. G. Behm. Associations between balance and muscle strength, power performance in male youth athletes of different maturity status. *Pediatric Exercise Science*, 28(4):521–534, 2016.

[4] R. W. Young. Evolution of the human hand: the role of throwing and clubbing. *Journal of Anatomy*, 202(1):165–174, 2003.

[5] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[6] M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*, 2018.

[7] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine. Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103*, 2018.

[8] L. Birglen and C. M. Gosselin. Kinetostatic analysis of underactuated fingers. *IEEE Transactions on Robotics and Automation*, 20(2):211–221, 2004.

[9] L. U. Odhner, L. P. Jentoft, M. R. Claffee, N. Corson, Y. Tenzer, R. R. Ma, M. Buehler, R. Kohout, R. D. Howe, and A. M. Dollar. A compliant, underactuated hand for robust manipulation. *The International Journal of Robotics Research*, 33(5):736–752, 2014.

[10] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. IEEE, 2017.

[11] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *Intl. Conf. on machine learning*, pages 1889–1897, 2015.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[13] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[14] J.-H. Park and H. Asada. Concurrent design optimization of mechanical structure and control for high speed robots. *Journal of dynamic systems, measurement, and control*, 116(3):344–356, 1994.

[15] C. Paul and J. C. Bongard. The road less travelled: Morphology in the optimization of biped robot locomotion. In *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180)*, volume 1, pages 226–232. IEEE.

[16] T. Geijtenbeek, M. Van De Panne, and A. F. Van Der Stappen. Flexible muscle-based locomotion for bipedal creatures. *ACM Transactions on Graphics (TOG)*, 32(6):206, 2013.

[17] S. Ha, S. Coros, A. Alspach, J. Kim, and K. Yamane. Computational co-optimization of design parameters and motion trajectories for robotic systems. *The International Journal of Robotics Research*, 37(13-14):1521–1536, 2018.

[18] T. Liao, G. Wang, B. Yang, R. Lee, K. Pister, S. Levine, and R. Calandra. Data-efficient learning of morphology and controller for a microrobot. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2488–2494. IEEE, 2019.

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

[20] H. Lipson and J. B. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406(6799):974, 2000.

[21] N. Cheney, R. MacCurdy, J. Clune, and H. Lipson. Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding. *ACM SIGEVOlution*, 7(1):11–23, 2014.

[22] N. Cheney and H. Lipson. Topological evolution for embodied cellular automata. *Theoretical Computer Science*, 633:19–27, 2016.

[23] T. F. Nygaard, C. P. Martin, E. Samuelsen, J. Torresen, and K. Glette. Real-world evolution adapts robot morphology and control to hardware limitations. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 125–132, 2018.

[24] D. Ha. Reinforcement learning for improving agent design. *arXiv preprint arXiv:1810.03779*, 2018.

[25] C. Schaff, D. Yunis, A. Chakrabarti, and M. R. Walter. Jointly learning to construct and control agents using deep reinforcement learning. In *IEEE Intl. Conf. on Robotics and Automation*, pages 9798–9805. IEEE, 2019.

[26] K. Vermeer, R. Kuppens, and J. Herder. Kinematic synthesis using reinforcement learning. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 51753, page V02AT03A009. ASME, 2018.

[27] K. S. Luck, H. Ben Amor, and R. Calandra. Data-efficient co-adaptation of morphology and behaviour with deep reinforcement learning. In *Conf. on Robot Learning*, 2019.

[28] F. de Avila Belbute-Peres, K. Smith, K. Allen, J. Tenenbaum, and J. Z. Kolter. End-to-end differentiable physics for learning and control. In *Advances in Neural Information Processing Systems*, pages 7178–7189, 2018.

[29] J. Degrave, M. Hermans, J. Dambre, and F. Wyffels. A differentiable physics engine for deep learning in robotics. *Frontiers in neurorobotics*, 13:6, 2019.

[30] Y. Hu, J. Liu, A. Spielberg, J. B. Tenenbaum, W. T. Freeman, J. Wu, D. Rus, and W. Matusik. Chainqueen: A real-time differentiable physical simulator for soft robotics. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6265–6271. IEEE, 2019.

[31] Y. Hu, L. Anderson, T.-M. Li, Q. Sun, N. Carr, J. Ragan-Kelley, and F. Durand. Difftaichi: Differentiable programming for physical simulation. *arXiv preprint arXiv:1910.00935*, 2019.

[32] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[33] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[34] A. Kumar Tiwari and S. V. Nadimpalli. Augmented random search for quadcopter control: An alternative to reinforcement learning. *International Journal of Information Technology and Computer Science*, 11(11):24–33, Nov 2019. ISSN 2074-9015. doi:10.5815/ijitcs.2019.11.03. URL `http://dx.doi.org/10.5815/ijitcs.2019.11.03`.

[35] T. Chen, L. Wang, M. Haas-Heger, and M. Ciocarlie. Underactuation design for tendon-driven hands via optimization of mechanically realizable manifolds in posture and torque spaces. *IEEE Transactions on Robotics*, 2020.

[36] T. garage contributors. Garage: A toolkit for reproducible reinforcement learning research. `https://github.com/rlworkgroup/garage`, 2019.

Supplementary Materials

A Mass-spring Toy Problem - Implementation Details

A.1 Optimizing Spring Stiffnesses and Computational Policy

Problem Formulation. We have presented the problem formulation in the paper (except for the exact form of the reward function), and we list these bullet points again for better readability here.

- The *observations* we can measure are the mass position y_1 and velocity $\dot{y}_1$.
- The *input* to the hardware policy is motor current i, which is the result of the computational policy.
- The *variables* to optimize are the weights and biases in the computational policy neural network, as well as all the spring stiffnesses $k_1, k_2, \cdots, k_n$.
- The *goal* is to make the mass m_2 go to the red target line in Fig 6 and stay there, with the minimum input effort. We designed a two-stage reward function that rewards smaller position and velocity error when the mass m_2 is far from the goal or moving fast, and in addition rewards less input current when the mass is close to the goal and almost still.

$$R = \begin{cases} -\alpha|y_2 - h| - \beta|\dot{y}_2| - \gamma|i_{max}|, & if\ \alpha|y_2 - h| + \beta|\dot{y}_2| > \epsilon \\ -\alpha|y_2 - h| - \beta|\dot{y}_2| - \gamma|i|, & if\ \alpha|y_2 - h| + \beta|\dot{y}_2| < \epsilon \end{cases} \tag{1}$$

where the α, β, and γ are the weighting coefficients, i_{max} is the upper bound of the motor current, and ϵ is a hand-tuned threshold.

Shared Implementation Details. We implemented HWasP, HWasP-Minimal, as well as our two baselines: CMA-ES with RL inner loop, and CMA-ES. In order to have a fair comparison between them, we intentionally made different cases share common aspects wherever possible. The physics parameters not being optimized are the same for all cases: $m_1 = m_2 = 0.1kg$, $l = 0.1m$, $h = 0.2m$, $g = 9.8m/s^2$, $k_T = 0.001Nm/A$, $r_{shaft} = 0.001m$. The initial conditions are random within the feasible range. The initial values of the total spring stiffness are sampled from 0 to $100N/m$. We used PPO and CMA-ES in the Garage package [36] for all cases. We implemented the computational graphs for HWasP in TensorFlow and the dynamics of the rest of the environment (non-differentiable) by ourselves using mid-point Euler integration. In the computational policies the neural network sizes are set to be 2 layers and 32 nodes each layer. The episode length is $1,000$ environment steps and the total number of steps is 4×10^6.

Hardware as Policy. We use Hooke's Law for the parallel springs

$$f_{spr} = \sum_{i=1}^{n} k_i y_1 \tag{2}$$

and current-torque relationship for the motor (ignoring rotor inertia and friction)

$$f_{str} = \frac{k_T i}{r_{shaft}} \tag{3}$$

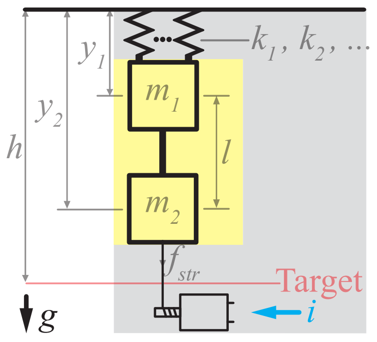
to model the mechanical part of our agent. We implement the computational graph of this hardware policy and combine it with a computational policy expressed as a neural network computational, as shown in Fig 7a.

Hardware as Policy — Minimal. In this case (Fig. 7b), we only add the hardware parameters (spring stiffnesses vector $\boldsymbol{k}$) to the action vector of the computational policy. The environment is governed by the physics of the spring-mass system, and takes the new values of $\boldsymbol{k}$ into account when simulating of the next time step.

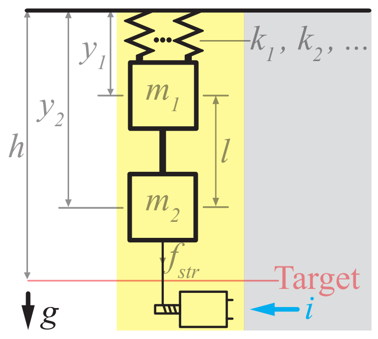
Numerical Results. If we ignore the transition phase of the task in this toy problem and make a quasi-static assumption, and assuming total spring stiffness equals the following:

$$k^* = \frac{(m_1 + m_2)g}{h - l} \tag{4}$$

then gravity will drag the mass m_2 exactly to the target, and the steady-state input current i can be zero, which minimizes the return on i in a long enough horizon. In the real world, the system has dynamic effects, but optimized total stiffness k should still be close to this value given a long enough horizon. After training, we indeed find the optimized total stiffness close to k^*, as shown in Table 1.

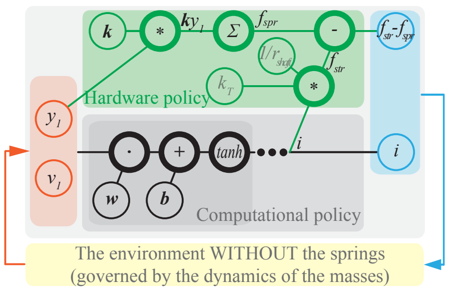


(a) HWasP



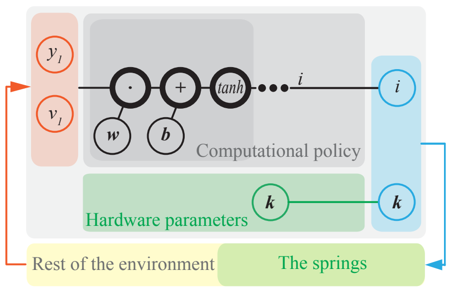
(b) HWasP-Minimal

Figure 6: The mass-spring system — optimizing spring stiffnesses and computational policy (yellow: environment, gray: agent/policy). (This is already presented in the paper, and we include it again for better readability.)



Hardware policy
Computational policy
The environment WITHOUT the springs
(governed by the dynamics of the masses)

(a) HWasP



(b) HWasP-Minimal

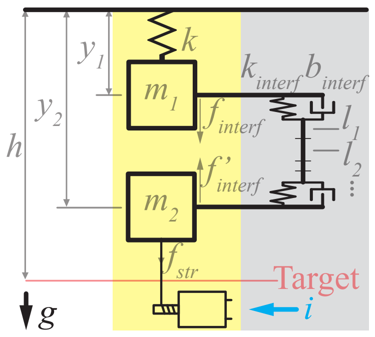
Figure 7: The (simplified) computational graphs of the proposed method for the toy problem — optimizing spring stiffnesses and computational policy. Bold circles represent tensor operations, light black or green circles with black fonts represent the variables to be optimized, and light circles with translucent fonts represent constants that are not part of the optimization.

Table 1: Optimization results of total stiffness [N/m]

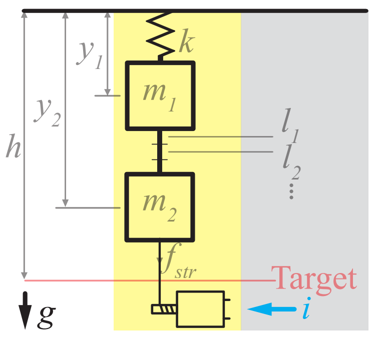
	$n = 10$	$n = 50$
k^*	19.6	
HWasP	19.9	20.1
HWasP-Minimal	18.7	21.9

A.2 Optimizing Bar Length and Computational Policy

Another case we tried for the toy problem is to optimize the bar length connecting two masses. Unlike the previous case, the interface between the redefined policy and environment is no longer an element generating forces, but a "hard" geometric relationship. For HWasP, we propose to use springs and dampers to "soften" such interfaces, and still use the spring-damper forces as the redefined action. For HWasP-Minimal, the bar length can directly be used a part of the action vector. Similar to the previous case, we divide the bar into multiple segments and treat the length of each segment as an individual parameter to test the scalability to higher-dimensional problems. The spring-mass system is illustrated in Fig.8.

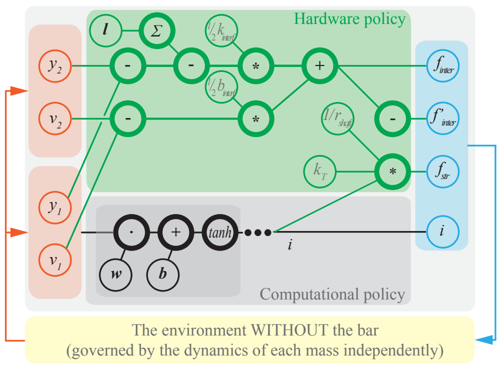


(a) HWasP



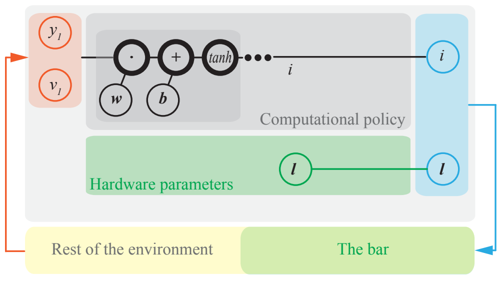
(b) HWasP-Minimal

Figure 8: The mass-spring system — optimizing the bar lengths and computational policy (yellow: environment, gray: agent/policy).



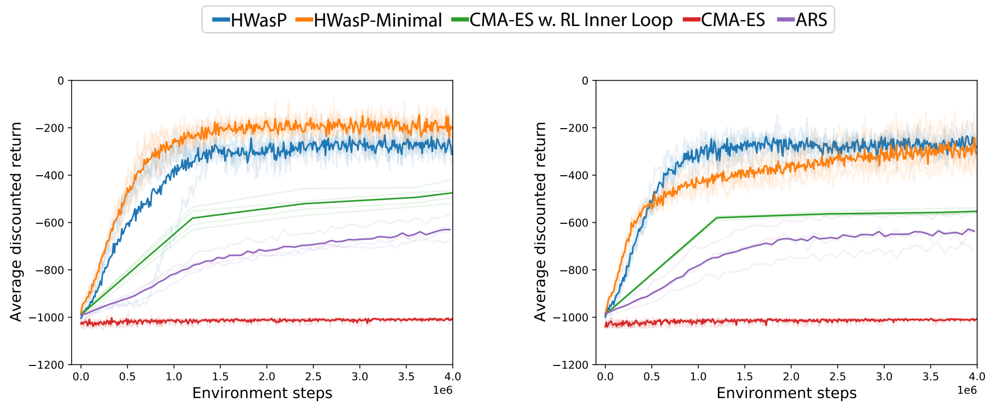
Hardware policy
Computational policy
The environment WITHOUT the bar
(governed by the dynamics of each mass independently)

(a) HWasP



(b) HWasP-Minimal

Figure 9: The (simplified) computational graphs of the proposed method for the mass-spring toy problem — optimizing bar lengths and computational policy.

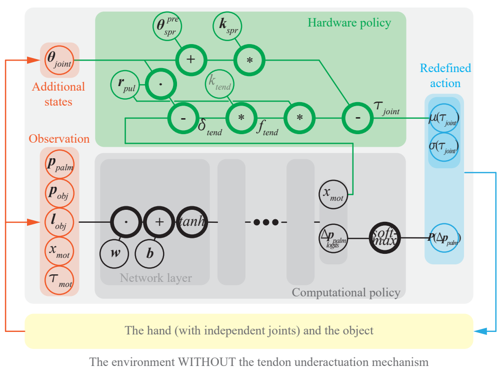


(a) 10 parameters (b) 50 parameters

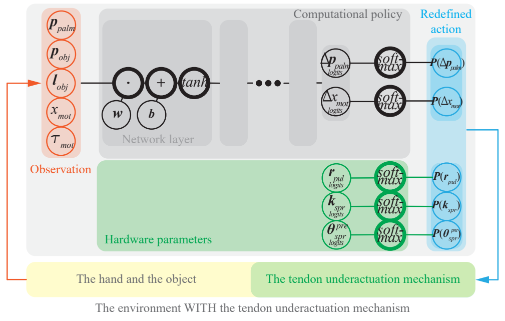
Figure 10: The training curves of the toy problem — optimizing bar lengths and computational policy.

Since we assume the bar is weightless, the dynamics is infinitely fast. Assuming all interfaces have the same k_{interf} and b_{interf}, we have:

$$\begin{aligned} f_{interf} &= k_{interf}((\frac{1}{2}(y_1+y_2) - \frac{1}{2}\sum_{i=1}^{n} l_i) - y_1)) + b_{interf}(\frac{1}{2}(\dot{y}_1+\dot{y}_2) - \dot{y}_1) \\ &= \frac{1}{2}k_{interf}(y_2 - y_1 - \sum_{i=1}^{n} l_i) + \frac{1}{2}b_{interf}(\dot{y}_2 - \dot{y}_1), \end{aligned} \tag{5}$$


Hardware policy
Redefined action
Additional states
Observation
Network layer
Computational policy
The hand (with independent joints) and the object
The environment WITHOUT the tendon underactuation mechanism

(a) HWasP



(b) HWasP-Minimal

Figure 11: The (simplified) computational graphs of both proposed methods for the hand mechanical-computational co-design problem. The plotting conventions are the same as the toy problem.

$$f'_{interf} = -f_{interf}. \tag{6}$$

We implement this relationship as a computational graph, as shown in Fig.9a.

Except for the variables to optimize and the computational graphs, other aspects (such as the reward function) are the same as the previous case. The training curves are shown in Fig.10. We can see that the performance of both our methods and the baselines are very similar to the previous case.

B Co-Design of an Underactuated Hand - Implementation Details

Tendon Underactuation Model. In a tendon-driven underactuated hand, the tendon mechanism acts as the transmission that converts motor forces to joint torques, based on the overall state of the system. In the model we constructed, it assumes an elastic tendon (with stiffness k_{tend}) goes through multiple revolute joints by wrapping around circular pulleys (radii $\boldsymbol{r}_{pul}$), and each joint is closed by the tendon and opened by a restoring spring (stiffness $\boldsymbol{k}_{spr}$ and preload angle $\boldsymbol{\theta}^{pre}_{spr}$). In order to make this problem determinate, our model thus assumes a nominal (finite) tendon stiffness, takes in the commanded relative motor travel Δx_{mot}, the motor position reading x_{mot} and the joint angles $\boldsymbol{\theta}_{joint}$ in the previous time step, and computes the joint torques $\boldsymbol{\tau}_{joint}$ for the current time step. Such torques are then commanded to the joints in the physics simulation.

The tendon elongation can be calculated as:

$$\delta_{tend} = (x_{mot} + \Delta x_{mot}) + \boldsymbol{r}^T_{pul}\boldsymbol{\theta}^{ref}_{joint} - \boldsymbol{r}^T_{pul}\boldsymbol{\theta}_{joint} \tag{7}$$

where $\boldsymbol{\theta}^{ref}_{joint}$ is the joint angle when the motor is in zero-position, and usually we define it to be zero. Then the tendon force can be calculated as:

$$f_{tend} = k_{tend}\delta_{tend}. \tag{8}$$

Hence, the torques applied to joints are:

$$\boldsymbol{\tau}_{joint} = f_{tend}\boldsymbol{r}_{pul} - \boldsymbol{k}_{spr} * (\boldsymbol{\theta}_{joint} + \boldsymbol{\theta}^{pre}_{spr}) \tag{9}$$

where $*$ means element-wise multiplication.

This model is built as an auto-differentiable computational graph in HWasP, and a non-differentiable model on top of the physics simulation in HWasP-Minimal and the baselines.

Problem Formulation.

- The *observations* are the position vector of the palm $\boldsymbol{p}_{palm}$, the position vector of the object $\boldsymbol{p}_{obj}$, the size vector of the object bounding-box $\boldsymbol{l}_{obj}$, and the current hand motor travel x_{mot} and torque τ_{mot}. We can also send joint angles $\boldsymbol{\theta}_{joint}$ to the hardware policy (only used in training time), but not the computational policy, because there are no joint encoders in such an underactuated hand, and the computational policy (which will serve as a controller of the real robot in run time) does not have access to joint angles.

- The *input* to the hardware policy also includes the relative motor travel command Δx_{mot} produced by the computational policy. The output of the overall system includes joint torques as well as palm motion $\Delta \boldsymbol{p}_{palm}$.
- The *variables* to optimize are the parameters in the computational policy neural network, and the hand underactuation parameters: pulley radii $\boldsymbol{r}_{pul}$, the joint restoring spring stiffnesses $\boldsymbol{k}_{spr}$ and the joint restoring spring preload angles $\boldsymbol{\theta}_{spr}^{pre}$, where each vector has a dimension of four corresponding to the proximal and distal joints in the thumb and the opposing fingers (the two fingers share the same parameters).
- The *goal* is to grasp the object and lift it up. Formally, the reward function is:
$$R = \alpha \left\| \boldsymbol{p}_{palm} - \boldsymbol{p}_{obj} \right\| + \beta C + f(z_{obj}) \tag{10}$$
where the α, and β are the weighting coefficients, $\boldsymbol{p}_{palm}$ and $\boldsymbol{p}_{obj}$ are the positions of the palm and the object, C is the number of contacts between the distal links and the object, z_{obj} is the height of the object, and $f(z_{obj})$ is a hand-tuned non-decreasing piecewise-constant function of z_{obj}.

Shared Implementation Details. Similar to the toy problem, we used the Garage package [36] and TensorFlow for HWasP, HWasP-Minimal and the two baselines in this design case. We use MuJoCo [32] for the physics simulation of the hand. The simulation time step is $0.001s$, the environment step is $0.01s$, and there are 500 environment steps per episode and 2×10^7 steps for the entire training. The initial height of the hand, as well as the type, weight, size, and friction coefficient of the object are randomly sampled within a reasonable range. We also added random perturbations forces and torques on the hand-object system to encourage more stable grasps. We use TRPO to explicitly limit the shift of action distributions. The computational policy is a fully-connected neural network with 2 layers and 128 hidden units on each layer. To stabilize the training, we scale the parameters in the hardware model by changing their units. Specifically, we scale all tendon stiffness and k_{spr} by $1.0e^{-3}$. Then, we multiply the torques by $1.0e^{3}$ when we apply them to the environment.

Hardware as Policy. Shown in Fig. 11a, we implement the underactuation model as a hardware policy computational graph, which allows the hardware parameters to be optimized via auto-differentiation and back-propagation. We redefine the action in the RL formulation to be the relative palm position command as well as the joint torques. The environment then becomes the hand-object system without the tendon underactuation mechanisms, i.e. with independent joints.

Hardware as Policy — Minimal. We also implemented the "HWasP-Minimal" method by incorporating all hardware parameters (pulley radii $\boldsymbol{r}_{pul}$, joint restoring spring stiffnesses $\boldsymbol{k}_{spr}$ and preload angle $\boldsymbol{\theta}_{spr}^{pre}$) into the original control (relative hand position $\Delta \boldsymbol{p}_{palm}$ and motor command Δx_{mot}), shown in Fig. 11b. The environment (containing underactuated hand and the object) takes in all these actions, sets the hardware parameters to the tendon model and performs simulation.

Numerical Results. Our results show that we can learn effective hardware parameters. The resulting pulley radii, spring stiffnesses and preload angles are shown in Table 2, using HWasP and HWasP-Minimal respectively. We note that the resulting parameters do not necessarily need to be identical: the optimal set of underactuation parameters is not unique by nature (for example, scaling them does not change the grasping behavior; for another example, a higher spring stiffness and a higher spring preload have similar effects), the evaluation is also noisy since we intentionally injected noise, and the gradient-based training process may also settle in local optima in the optimization landscape.

Table 2: The optimized pulley radii, joint spring stiffnesses and preload angles. In each cell, the first number is from HWasP, and the second number is from HWasP-Minimal.

	Pulley radius [mm]		Spring stiffness [Nmm/rad]		Spring preload [rad]	
Thumb proximal	10.0	8.1	6.2	7.0	2.0	3.2
Thumb distal	7.5	6.1	6.1	9.2	1.5	3.1
Finger proximal	3.0	5.0	6.1	8.2	1.9	3.4
Finger distal	2.6	4.0	5.9	9.2	1.1	3.1

Validation with Physical Prototype. Here we present additional details of the physical prototype we built according to optimization results. This hand is 3D printed using polylactide (PLA). All eight joints in three fingers are actuated by a single servo motor (DYNAMIXEL XM430-W210-T) equipped with 12-bit encoder and current sensing. The tendons are made of Ultra-high-molecular-weight polyethylene (commercially named Spectra®). In each finger joint, there is a pulley with the

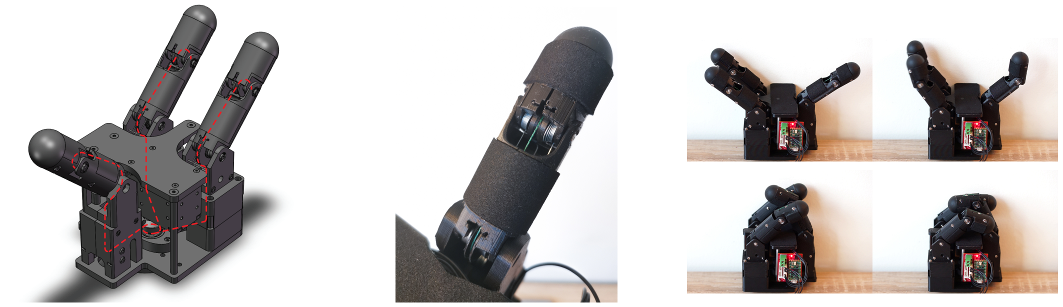

(a) The CAD design and tendon routing (red lines). (b) A close-up showing the joints, tendons and springs. (c) The finger trajectory.

Figure 12: The physical hand prototype.

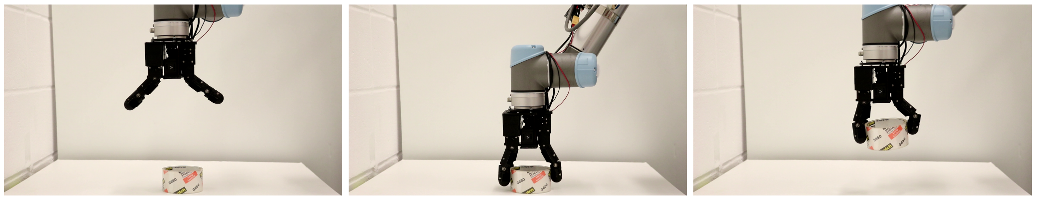


Figure 13: Policy deployment on real robot.

radius prescribed by the optimization, and two parallel springs whose stiffnesses also add up to the optimized value prescribed by our method. The distal joint pulleys are fixed to the finger structures, and proximal joint pulleys are free-rotating. The CAD model, tendon routing scheme, joint design, and finger trajectory are shown in Fig. 12.

Sim-to-Real Transfer. In order to transfer the trained policy from the simulation to the physical world, we used the following techniques:

- We applied the *Domain Randomization* techniques to a lot of physical parameters and processes. We randomized object shape (among sphere, box, cylinder, ellipsoid), size (bounding box size uniformly sampled from 40 to $100mm$), weight (uniformly sampled from 100 to $500g$), friction coefficient (uniformly sampled from 0.5 to 1.0) and inertia (each principal component uniformly sampled from 0.0001 to $0.005kg \cdot m^2$). We also injected sensor and actuation noise (Gaussian noise with $1mm$ and $0.01rad$ standard deviation for translational and rotational joints respectively), and applied random disturbance wrenches (Gaussian disturbance with $0.02N$ and $0.002Nm$ standard deviation for force and torque respectively) on the hand-object system.
- In simulation, we limited the hand motion close to quasi-static, and use position control to drive the palm and hand joints. This control scheme is not sensitive to inaccurate parameters, unmodeled dynamics and can effectively reject disturbances.

Policy Deployment on Real Robot. We used a UR5 robot and our optimized hand prototype as the hardware platform and deployed the trained computational policy to control the arm and hand motion. Fig. 13 shows an example grasp in the 3D-Grasp case. Videos are available at `https://roamlab.github.io/hwasp/`.