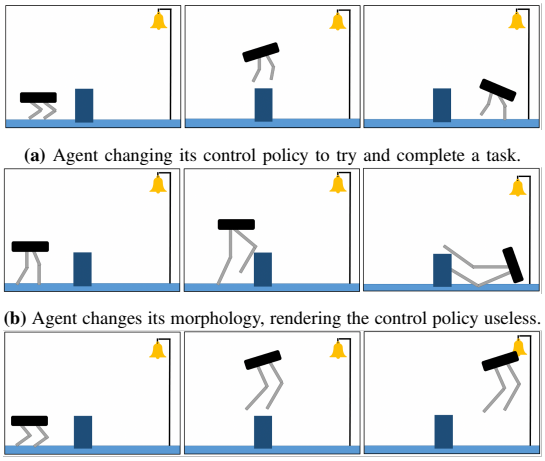
ORCHID: Optimisation of Robotic Control and Hardware In Design using Reinforcement Learning

Lucy Jackson[1], Celyn Walters [1], Steve Eckersley[2], Pete Senior [2] and Simon Hadfield[1]

***Abstract*— The successful performance of any system is dependant on the hardware of the agent, which is typically immutable during RL training. In this work, we present ORCHID (Optimisation of Robotic Control and Hardware In Design) which allows for truly simultaneous optimisation of hardware and control parameters in an RL pipeline. We show that by forming a complex differential path through a trajectory rollout we can leverage a vast amount of information from the system that was previously lost in the 'black-box' environment. Combining this with a novel hardware-conditioned critic network minimises variance during training and ensures stable updates are made. This allows for refinements to be made to both the morphology and control parameters simultaneously. The result is an efficient and versatile approach to holistic robot design, that brings the final system nearer to true optimality. We show improvements in performance across 4 different test environments with two different control algorithms - in all experiments the maximum performance achieved with ORCHID is shown to be unattainable using only policy updates with the default design. We also show how re-designing a robot using ORCHID in simulation, transfers to a vast improvement in the performance of a real-world robot.**



(a) Agent changing its control policy to try and complete a task.

(b) Agent changes its morphology, rendering the control policy useless.

(c) Agent is able to complete the task by changing both its morphology and control policy.

Fig. 1: Illustration of the impact of the ORCHID system on an agents ability to achieve a task.

I. INTRODUCTION

The reinforcement learning (RL) paradigm suffers from an inherent limitation, caused by the intrinsically coupled physical laws that underpin motion. The successful performance of any agent in a given environment is dependant on both its physical design and control capabilities. In some cases a poor mechanical design may cause an impossible control problem, such as a walker with legs too short to reach a target, see Figure 1a. In less extreme cases, designs may be inefficient, with excess material or joints, leading to high energy consumption and/or poor movement patterns. Deep RL and other data-driven learning techniques are now capable of learning complex control polices with exceptional results [1]–[3]. However, these techniques treat hardware as immutable and learn no useful representation for how changes may affect the environment or task outcome. No amount of RL training can overcome a fundamentally flawed hardware design.

Analytical methods to optimise robotic design do exist and are based around a combination of computational software, multi-objective optimisation techniques and human in the loop prototyping and testing [4], [5]. The use of these techniques makes the design process a long, arduous and expensive one. They attempt to serve as a proxy for optimizing the final performance of the agent in a real task. However, the growing complexity of learned control algorithms is rendering these proxies less and less valid.

Unfortunately, it is impossible to decouple what constitutes a good hardware design from the control policy used. Approaches to automated morphology optimization generally assume a simplistic control scheme. This means that once the hardware is finalized and a more custom control policy is developed, the previously chosen hardware may no longer be optimal, as shown in Figure 1b. A biological example of the interconnectedness of mechanical design and motor skills, can be seen in human athletes. Sprinters aim to develop stronger hip muscles and a technique that involves higher frequency, shorter strides, while long distance runners require stronger ankle muscles and longer strides at a lower frequency [6]. These solutions are only possible thanks to the joint consideration of both mechanical design and motor control. This raises the question of whether we can effectively automate the full design of the agent. The approach proposed in this paper is designed to unify the design of both the control policy and the robotic morphology bringing the final system nearer to true optimality while providing a more time efficient and less costly design process.

To date, most RL techniques that have touched on this co-optimzation problem rely on intelligent sampling and sheer computational power to find an optimal combination. They utilise high fidelity simulators that allow for refinement of the hardware design during simulation, therefore sidestepping the need to re-build hardware in between each trial.

*This work was supported by Surrey Satellite Technology Ltd.

[1] University of Surrey, GU2. 1.jackson@surrey.ac.uk, s.hadfield@surrey.ac.uk, celyn.walters@surrey.ac.uk

[2]Surrey Satellite Technology Limited. Tycho House, GU2. s.eckersley@sstl.co.uk, p.senior@sstl.co.uk

Solving the problem in this manner cannot be considered as genuine co-optimisation since the problem is decoupled into design generation and policy optimisation and solved iteratively. Furthermore, the joint control and hardware space is very complex: each hardware design can only be refined up to a point before changes to the fixed control policy are necessary for further improvement. The dual problem of refining control policies for a collection of fixed, randomly sampled hardware designs, suffers from severe scalability issues, since the high dimensional nature of modern control policies make their repeated optimization very costly.

We propose a truly unified approach where a single RL system optimises both the morphology of the robot and the control policy simultaneously, as shown in Figure 1c. Our method takes inspiration from the concept that fundamentally underpins all forms of deep learning – hierarchical partial differentiation. By using a modern simulator during optimisation, consisting of a differentiable transition function, we are able to form a complex differential path throughout a rollout. This allows our system to leverage vast amounts of information that was otherwise lost in the 'black box' environment. We exploit this information to quantify the coupled relationship between physical design and control of a robot-agent pair, leading to improved designs and increased performance. In addition, the hardware parameters can be directly modelled without the need for function approximators. This makes it possible to transfer the resultant designs to the real world, as we demonstrate with a balance robot.

To summarise, the contributions of this paper are 1) The first work that, to the best of our knowledge, allows a truly unified general approach to hardware and software co-optimisation capable of dealing with kinematic design and complex control policies. 2) The first work to integrate fully differentiable simulation into an RL pipeline. 3) A new hardware conditioned critic to improve the stability of reinforcement learning across changing hardware designs.

II. RELATED WORK

Co-optimisation of morphology and control has been of interest for decades in the field of robotics. Early approaches used a full set of dynamic equations to solve for both the morphology and control [7]–[9]. However, this becomes non-trivial for any complex system and is impossible to implement in the more common model-free scenario. Evolutionary computation has been used to solve the design problem [10]–[12]. While the results are promising these methods are computationally expensive and data inefficient.

More recently the problem has been approached in an RL setting. Ha [13] bridged the gap between evolutionary computation and RL by using a population based policy gradient method to sample the optimal robot-agent pair. Although their method yielded a higher reward in fewer training iterations when compared to the 'out-of-the-box' agent, the method is computationally expensive since a population of designs must be maintained throughout training. Schaff et al. [14] propose a similar approach whereby they treat the parameterised design as an additional input to the control policy. Throughout training they maintain a distribution of designs and continually shift this distribution towards higher performing regions of the design space. Fundamental to the success of their method is that their control policy maintains generalization over morphologies in the distribution – a widely documented challenge in the field of RL [15].

Luck et al. [16] noted the need to maintain a distribution of designs as a limitation and instead decompose the problem as a bi-level optimisation. They alternate optimizing the morphology and the control policy. By learning Q-values over the design distribution and using this to pre-evaluate agent performance prior to testing they reduced size of design distribution that must be maintained. This method is reliant on learning Q-values over a wide distribution of robotic designs, or risk missing out on the optimal design due to poor function approximation. Our work proposes a novel hardware conditioned critic to mitigate this issue, and also avoids the explicit decoupling of the two design elements.

Avila Belbute-Peres et al. [17] propose an approach to directly optimise robotic morphologies and solve simple control problems. They achieve this by re-defining the systems using differentiable mathematical concepts on a case-by-case basis (similar methods are shown in [18], [19]). Application of their method is limited since they require differntiable reward functions, which are typically sparse step functions based on the final state of a system – and reward functions almost universally hold no information about the robotic morphology. In contrast to these, our proposed approach only requires the system dynamics to be differentiable, while solving for standard non-differentiable reward functions.

The most similar prior work to our proposed approach is HWasp [20]. In this approach they re-define the effects of the robotic hardware needing to be optimised separately from the rest of the environment as a differentiable computational graph. The control policy and hardware policy are then optimised in a standard RL framework. This approach lacks generality since it requires a redefinition of the action space, unique to each problem, as well as the inclusion of bespoke computations for the graph implementation. In contrast our proposed approach can be employed in standard RL benchmark environments [21] since no change to the action or state space is required. HWasp also suffers from an inability to fully optimise changes in morphology or link dimensions. Instead, modifications are made to kinematic structure (inclusion of a mass damper system at either end of a link) in order to allow for the necessary modifications to the action space to be implemented. Our proposed approach does not suffer from such limitations and the physical dimensions of robots can be optimised directly.

III. METHODOLOGY

ORCHID provides a truly unified approach to optimising robotic morphologies and control policies. Figure 2a shows the standard RL feedback loop used to update an actor-critic architecture based on experiences in an environment, for all time steps in T. Visible in red is the simple differential path required for network updates to improve agent performance.

This is dependant only on the action taken (a_t) at time t and the subsequent reward (r_t), leaving the agent with little information to exploit during updates. Figure 2b shows three time steps in an environment following ORCHID. The use of a differentiable transition function ($\mathcal{P}$) hugely expands the capability of our system, via the introduction of a complex differential path throughout the rollout. From this, it becomes apparent how information on the impact/quality of a change to the morphology parameters (v) can improve performance. During training we allow the optimiser to update the morphology parameter and/or the agent parameters in a way that best improves performance throughout the system. The exact robotic morphology parameterisation vector (v) is environment specific and is sized by the number of designable parameters, i.e if the mass and width of each leg on a quadrouped robot is to be optimised: $v \in \mathbb{R}^{8\times1}$.

A. Reinforcement Learning Formulation

We solve the co-optimisation problem in an RL setting, in which the state is fully observable and episodes/transitions are modeled as a finite Markov Decision Process (MDP). The MDP is represented by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}(s_{t+1}|s_t, a_t)$ is the state transition function, $r(s_t, a_t)$ is the reward function and γ is the reward discount rate. The control of the agent in this environment is determined by policy $\pi_\theta(a_t|s_t)$, which uses an agent paramtereised by θ to take action a_t from state s_t at time t. During optimisation, parameters θ are updated such that the expected return, $R = \sum_{t=0}^{T} \gamma^t r(s_t, a_t)$, over trajectories induced by $\pi_\theta(a_t|s_t)$, is maximised, where T is the length of an episode. The optimal values (θ^*) are determined by solving:

$$\theta^* = \underset{\theta}{arg\,max} \mathbb{E}_{s\sim\mathcal{P}, a\sim\pi_\theta} R \tag{1}$$

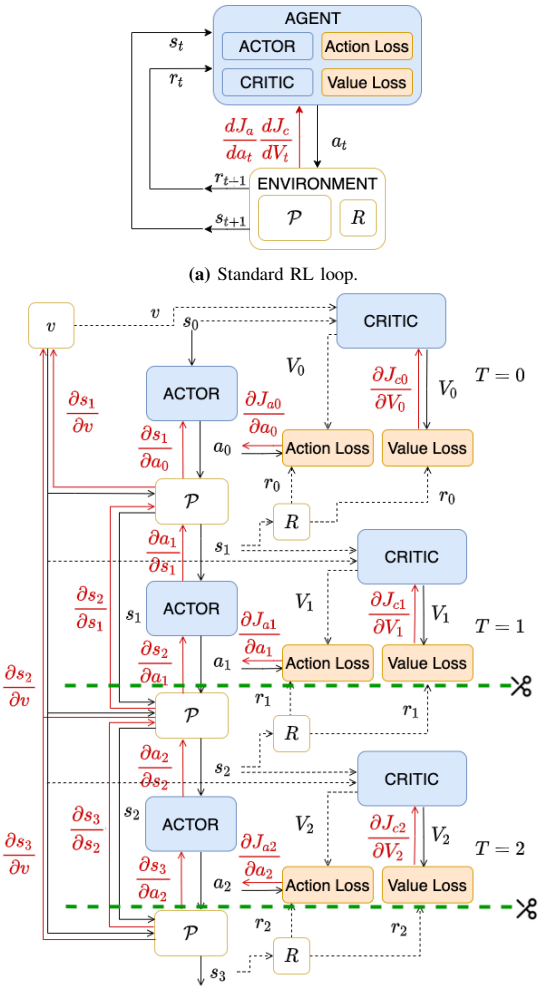
B. Conditioning RL on morphology

During standard RL, the morphology of a robot (v) is treated as immutable. However, in this work we allow it to change during training. We make $\mathcal{P}$ a function of v such that $\mathcal{P}(s_{t+1}|a_t, s_t, v_t)$. This allows for the derivative of the future state with respect to the morphology to be computed. Note that v_t is only updated between episodes and is therefore referred to as v in the remainder of this paper. Exploiting the fact that the performance of an agent is now dependant on v and θ, we introduce it as a trainable parameter and equation 1 becomes:

$$(\theta^*, v^*) = \underset{\theta, v}{arg\,max} \mathbb{E}_{s\sim\mathcal{P}, a\sim\pi_\theta} R \tag{2}$$

Successful learning with modern RL algorithms is usually dependant on an accurate estimation of the value of each state ($V(s_t)$). In this setting the value becomes dependant on v, giving $V(s_t, v)$. The value is used to calculate the advantage ($\hat{A}_t$), defined as:

$$\hat{A}_t = \sum_{t}^{T} \gamma^t r_t - V(s_t, v), \tag{3}$$



(a) Standard RL loop.

(b) Three unrolled time steps in ORCHID.

Fig. 2: In both images the forward pass is shown in black and the backward pass in red. Non-differentiable paths are shown as dashed lines.

The use of the advantage during actor updates reduces variance during training in a way that stabilises learning and prevents exploding gradients. The value function is usually learnt as a distribution over states by the critic network, (μ_ϕ), parameterised by ϕ. In order to maintain stability throughout training we require an accurately learnt value function approximation. Given this we require μ_ϕ to learn a distribution which does not lose reliability every time the morphology is modified. In order to facilitate this, we condition μ_ϕ on v during training and optimse for:

$$\phi^* = \underset{\phi}{arg\,min} \mathbb{E}_{s\sim\mathcal{P}, a\sim\mu_\phi} (R - V(s, v))^2 \tag{4}$$

The result is a general network, $\mu_\phi(V_t|s_t, v)$ that is capable of approximating V_t based on the morphology at hand.

C. Parametric control optimisation

Our approach is agnostic to the control policy architecture, given it is differentiable and can be parameterised. In our experiments we test two types of control policies. The first is a classical PID controller, where the policy is defined as:

$$a_t = \omega_1 e_t + \omega_2 \int e_t + \omega_3 \frac{de_t}{dt}, \tag{5}$$

and parameterised as $\theta = \{\omega_0, \omega_1, \omega_2\}$. Here e_t is the difference between s_t and a reference vector. Secondly we test a deep multi-layer-perceptron control network parameterized as $\theta = \{w_i, b_j, w_{i+1}, b_{j+1}, ..., w_I, b_J\}$ where w are the weights, and b are the biases of the network.

D. Parameter Co-Optimisation

Optimising parameters θ, v and ϕ according to equations 2 and 4 is nontrivial. These equations are highly nonlinear and include many complex inter-related functions. To deal with this, we make use of optimization techniques that have proved highly successful in the field of deep-learning. Specifically, we formulate the entire system including the actor, critic and environment updates (but not the reward function) as a chain of partial derivatives leading back from the final loss functions. We use the composition of these partial derivatives to make small updates to the parameters based on the value of the loss functions, and repeat this across many different explorations of the input space.

The applications of the chain rule for back-propagation within a multi-layer perceptron (i.e. Figure 2a) are well understood. Likewise, the optimization of parameters for a PID controller are trivial to compute and will not be repeated here. Instead, we focus on the propagation of information along the other routes shown in red in Figure 2b.

The introduction of v into the optimisation results in a number of new partial derivative terms necessary for backpropagation, compared to the standard formulation. $\frac{\partial s}{\partial v}$, $\frac{\partial s}{\partial a}$ and $\frac{\partial s_t}{\partial s_{t-1}}$. $\frac{\partial s}{\partial a}$ can be solved explicitly from the forward pass, while $\frac{\partial s}{\partial v}$ and $\frac{\partial s_t}{\partial s_{t-1}}$ require the use of a differentiable transition function. We achieve the state transition function from the rest of the environment, giving a split environment consisting of $\mathcal{P}$ and r, as seen in Figure 2b.

The unified approach to co-optimisation means a single loss function (J_a) drives the refinement of θ and v, based on its differential such that:

$$\frac{\partial J_a}{\partial v} = \frac{\partial J_a}{\partial a}\frac{\partial a}{\partial s}\frac{\partial s}{\partial v}, \tag{6}$$

and

$$\frac{\partial J_a}{\partial \theta} = \frac{\partial J_a}{\partial a}\frac{\partial a}{\partial \theta}. \tag{7}$$

However, careful examination of the full differential path shown in Figure 2b reveals the dependency of a single point in the state space on all previous actions and states. This requires gradients to flow across a timestep meaning a single step trajectory cannot optimize v because no gradient flows through the transition function. For example we can see that the update for v from a 3 step rollout should be:

$$\begin{aligned}\frac{\partial J_{a_1} + \partial J_{a_2}}{\partial v} = &\frac{\partial J_{a_1}}{\partial a_1}\frac{\partial a_1}{\partial s_1}\frac{\partial s_1}{\partial v} + \frac{\partial J_{a_2}}{\partial a_2}\frac{\partial a_2}{\partial s_2}\frac{\partial s_2}{\partial v} \\ &+ \frac{\partial J_{a_2}}{\partial a_2}\frac{\partial a_2}{\partial s_2}\left(\frac{\partial s_2}{\partial s_1} + \frac{\partial s_2}{\partial a_1}\frac{\partial a_1}{\partial s_1}\right)\frac{\partial s_1}{\partial v}\end{aligned} \tag{8}$$

Indeed the number of routes for gradient flow grows exponentially with the length of the rollout, a full derivation can be found in the supplementary material. To counteract this, it is possible to limit the gradient flow between pairs of time steps as shown by the green line. This also improves computational efficiency of our method since the computational graphs that need to be stored during training are smaller.

E. Differentiable State Transitions

In this work we employ a method similar to [19] for defining our simulators. In order to use ORCHID the environment must have a differentiable transition function as detailed above, necessary for the calculation of $\frac{\partial s}{\partial v}$. Although this could theoretically preclude certain environments, the advancement of differentiable simulators means behaviours such as collisions [18], grasping and soft deformable structures [22] can now be modelled using fully differentiable functions. It is also important to highlight that gradients do not flow between v and μ_ϕ, shown by the dashed black line in Figure 2b. This is because updates to the robot design should be driven by optimising the performance on the given task, not to improve the system's knowledge of reward distribution. Instead, the interaction between v and μ_ϕ is limited to the conditional training explained above.

F. Losses and Design Constraints

The simultaneous nature of the optimisation in our method means all parameters (θ, v, ϕ) are updated at the same time. As mentioned previously we present two implementations of ORCHID corresponding to two different parameterisations of theta. Both use the same loss functions (A2C [23]), critic architecture and overall set-up. However, they differ in the actor network architecture. The first uses three fully connected layers with 64 hidden nodes. The second implementation is a classical PID controller, demonstrating the generality of our approach across control policies. The actor takes an error signal (e_t) as input and outputs a_t, as seen in equation 5.

The loss function that constrains θ and v is calculated as:

$$J_{at} = -\hat{A}_t log(\pi_\theta(a_t|s_t)) - H_t(\pi(a_t|s_t)), \tag{9}$$

where $\hat{A}_t$ is the advantage, as defined in equation 3 and H_t is an entropy term, added to encourage exploration. The loss used to update the critic network is calculated as:

$$J_{ct} = (\gamma^t r_t + \mu_\phi(V_{t+1}|s_{t+1}, v) - \mu_\phi(V_t|s_t, v))^2. \tag{10}$$

At every training iteration J_a and J_c are summed. Algorithm 1 outlines the system training. [1]

During robotic hardware design it is normal to have bounds on the allowable values of parameters. These may be due to physical limitations, such as non-negative weights and lengths or known environmental stipulations such as storage or operating facility sizes. ORCHID implements such constraints by passing parameters through a smooth differentiable bounding function such as CELU [24]:

$$CELU(v, \alpha) = \begin{cases} v, & \text{if } v \geq 1 \\ \alpha\, exp\left(\frac{v}{\alpha}\right) - 1, & \text{otherwise} \end{cases} \tag{11}$$

[1] All code can be found at `https://gitlab.eps.surrey.ac.uk/lj00304/orchid.git`

Algorithm 1 Implementation of ORCHID

Randomly initialise parameters θ, v and ϕ
for $i = 1, 2, \ldots$ total number of updates **do**
  Initialise memory $M \leftarrow \varnothing$
  Sample starting state $s_0^i \sim \mathcal{S}$
  for $j = 1, 2, \ldots$ number of steps per update **do**
    Sample action $a_j \sim \pi_\theta(a_j|s_j)$
    Determine next state $s_{j+1} = \mathcal{P}(s_{j+1}|a_j, s_j, v_i)$
    Determine reward $r_j = r(s_{j+1}, a_j, v_i))$
    Store transition $M_\mu \leftarrow M_\mu \cup (s_j, a_j, r_j, v_i, s_{j+1})$
  end for
  Calculate J_c
  Calculate J_a
  $\theta \leftarrow \theta + \alpha \nabla J_a$, $v \leftarrow v + \alpha \nabla J_a$
  $\phi \leftarrow \phi + \alpha \nabla J_c$
end for
return θ, v and ϕ

IV. Experiments and Results

We demonstrate the generality of our method by performing all evaluations across four standard RL problems (CartPole, Pendulum, Mountain Car and Balancer). Three of these are OpenAi gym environments [21]. The fourth is a newly developed simulation environment[1] with a corresponding physical robot system. Unless stated otherwise we keep motor specifications and material choice/density the same as the default. In all cases, updates to the morphology maintained the symmetry of parts and the overall configuration of the robot. To further illustrate the generality of our work, we repeat all experiments in all environments using both a classical control policy, and an RL control policy.

A. Environments

In the **CartPole** environment the task involves balancing a pole above a cart attached to an un-actuated joint where motion occurs along a friction less track. A reward of 1 is given for each time step that the pole remains above the cart ($\pm 15°$). Additionally a penalty is applied based on the distance moved from the last step. This encourages the system to minimize wasted motion. In this task setting we allow the mass of the cart and length of the pole to optimised.

In the **Pendulum** environment the goal is to swing the pendulum so that it remains in an upright position. A reward is given based on the angular position and velocity of the pendulum, and the force applied at each step. A higher reward is given for a stationary upright pendulum achieved using minimal force. In this setting we allow both the mass and length of the pole to be optimised independently, mimicking a change in the material and structure.

The **Mountain Car** environment requires a cart which starts at the bottom of two 'mountains' to drive up the right hand side to reach the goal. The engine is not strong enough to do so, instead the cart must build up momentum by cycling between the mountains. A negative reward is given for all engine output and a sparse reward is given when the goal is reached. The path planning aspect of this environment means we cannot apply our classical control policy. In this setting we allow the mass of the car to be optimised.

Balancer, the final demonstration of our co-optimisation method is in a new environment. We set up a balancing robot whose task is to remain upright by actuating a single wheel-axle, as seen in Figure 5. This is a variation on the cart pole environment with a direct coupling of the upright to the chassis which induces a de-stabilising momentum proportional to the length of the upright. This environment encapsulates the highly coupled nature of design and control since actuation of the upright is inextricably coupled to the movement of the base. A similar reward function to cart pole is used in that the robot should remain upright for as long as possible while trying to minimise wasted movement. In this experiment we allow the length of the upright and the weight of the chassis to be optimised.

B. Baselines

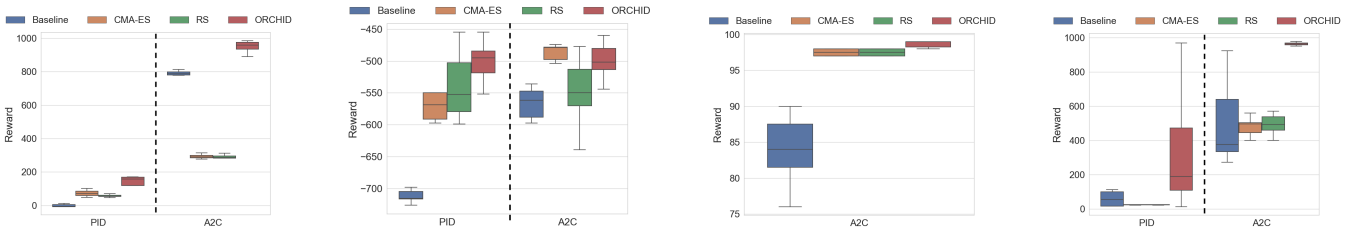
We compare our method against three baselines, the first is a standard PID/A2C baseline used to learn the control policy for a fixed morphology. In the second, the morphology is optimised using Covariance Matrix Adaptation with Evolutionary Selection (CMA-ES) [25], while the control policy is optimised individually using an inner RL loop, either with the PID or A2C implementation. The third uses Random Selection (RS) to optimise the morphology where again, the control policy is optimised using an inner RL loop. An Optuna implementation of both of these methods is used [26]. In all cases, other than mountain car, we evaluate with two different types of control policy.

C. Final Design Performance Analysis

First we show a direct comparison between the final performance of designs chosen by ORCHID and the three baselines, we do this for both the PID and A2C controller. To mitigate the effects of random initialization and ensure that findings are meaningful, every technique was evaluated with 10 different seeds. Figure 3 shows the median, min, max, 25th and 75th percentiles across these runs for each technique. The A2C/PID baseline was trained with the default morphology, while the ORCHID algorithm was initialised with this default morphology. Morphology initialisation for CMA-ES and RS was left to the respective algorithms.

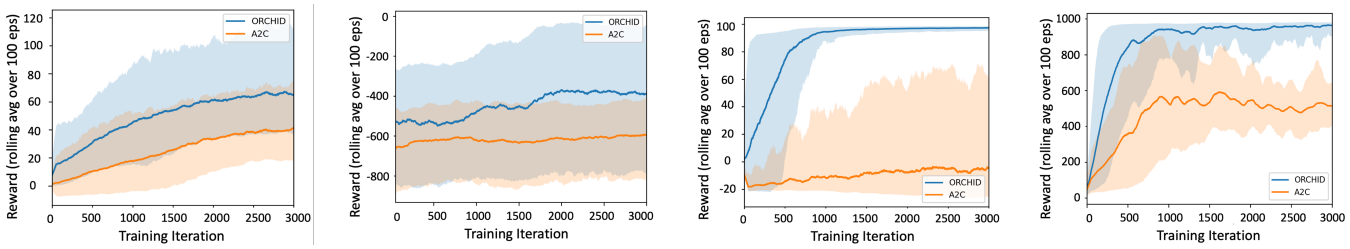
The implementation of ORCHID shows the highest maximum performance across all experiments and control policies. In fact, for all experiments the median with ORCHID is higher than the maximum score achieved with the baseline. This highlights the need for design co-optimisation since the changes in morphology lead to improvements that are unattainable using control policy updates alone.

We note smaller gains between ORCHID and the standard baseline with the PID controller, as well as noticing lower levels of performance compared to the use of a more complex policy. This is due to the limitations of using a simpler architecture to learn a complex policy. However, even with the reduced control capability, the implementation of ORCHID with the PID controller leads to a higher maximum performance than the A2C controller in 2/3 environments. Again highlighting the need to allow for hardware design optimisation.



(a) Cart Pole **(b)** Pendulum **(c)** Mountain Car **(d)** Balancer

Fig. 3: Final performance of our method and baselines in different environments. Each consists of 10 runs at random start points. Note that the start points are the same for each algorithm in the env, with the baseline A2C/PID this is the fixed morphology used throughout.



(a) Cart Pole **(b)** Pendulum **(c)** Mountain Car **(d)** Balancer

Fig. 4: Comparison between ORCHID and baseline in time to solve the task. Results shown are the median, min and max values over 10 runs, each with a random starting seed. Training was done using A2C controller.

D. Design Speed Analysis

Another advantage of our method is the acceleration of the design cycle, and the sample efficiency of training. This speed improvement manifests in two ways. First, as show in Figure 4, the ability to optimize the hardware design actually allows our joint system to converge in a similar time frame, or more rapidly, than refining the control policy alone. This is a very interesting result as our joint optimization is strictly more complex and higher dimensional than the baseline (with our target subsuming the baseline target). The same is seen in the PID experiments but for conciseness the results are omitted. This increase in convergence speed demonstrates that updating the hardware design is often a far more effective way to improve performance than trying to refine a control policy for a sub optimal hardware design.

The second improvement in speed comes when comparing the system to CMA-ES and RS. While our technique only needs to run a single optimization to convergence, these baselines must repeat the entire process multiple times for different samples. In our experiments, this lead to an approximate factor of 3 increase in the runtime for these baselines, as shown in Table I. On top of this, the implementation of both CMA-ES and RS require maintaining the distribution of control policies, leading to high memory overheads.

E. Sim-to-real with ORCHID

Finally we show how the co-optimisation of design and control policy using our approach improves performance in sim-to-real transfer. We show this using an implementation of the balancer robot. The baseline design was trained in simulation using the classical PID control policy, gaining a maximum reward of 102. After joint hardware and control optimization, our method produced a robot with a shorter lighter upright, and a heavier chassis, leading to a maximum reward in simulation of 967. Both variants of the robot were prototyped physically (Figure 5) and tested in real life. Videos of performance are available in the supplementary material. In practice the performance in real life mirrored that in simulation where the optimised design was able to remain upright for a substantially longer amount of time. Even more impressively, the optimized design proved capable of re-stabilising when external perturbations were applied – a scenario which was not included during training. Table II shows a numerical comparison between the two balancer variations.

TABLE I: Comparison of run times for ORCHID and all baselines.

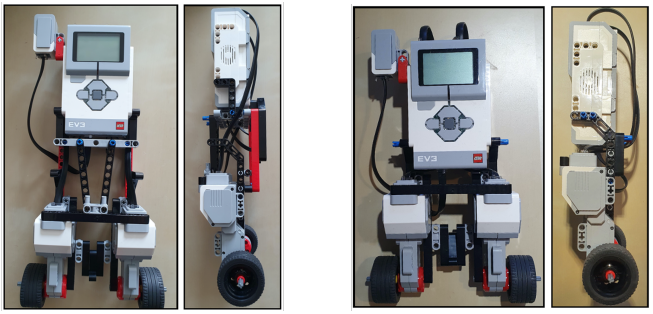
		Time to convergence, average over 10 runs (hrs)			
Env	Control	Baseline	CMA-ES	RS	ORCHID
Cart Pole	PID	**1.2**	8.2	8.1	*4.9*
	A2C	**1.1**	9.4	8.9	*5.1*
Pendulum	PID	**2.7**	15.7	16.3	*6.5*
	A2C	**2.4**	13.8	15.6	*6.8*
Mountain Car	PID	n/a			
	A2C	**2.1**	12.9	13.5	*4.9*
Balancer	PID	**1.8**	12.7	15.8	*7.8*
	A2C	**2.6**	15.4	16.6	*6.7*

TABLE II: Comparison between original prototype and ORCHID design

	Baseline	ORCHID
Chassis Mass	226g	404g
Upright Length	28cm	17cm
Reward in sim (avg over 10 runs)	102	967
% of successful runs of prototype	7	83

V. CONCLUSIONS

In this paper we presented ORCHID – the first RL approach to jointly optimise robotic morphology and control



(a) Original design **(b)** Optimised design

Fig. 5: Comparison between initial design and the optimised design, determined using ORCHID.

policies where changes can be made to the robots kinematics. This method addressed the two main problems with prior research. The first being the need to maintain a distribution of control policies and robotic designs throughout the optimisation process and the second being the iterative way in which state-of-the-art algorithms optimise the control and design separately. Our method overcomes both of these issues by proposing a unified optimization scheme which simultaneously updates both the control parameters and design parameters. The combination of a critic network conditioned on the morphology and a differentiable transition function that allows gradient flow throughout a rollout, led to improvements in the maximum performance for every combination of environment and control policy. In addition to finding the jointly optimal control policy and hardware pairs, ORCHID boasts the added benefit of greatly increasing design speed when compared to naive brute force search methods. We believe that this method of co-optimisation can aid in streamlining the design process of robotic agents, with particular gains in the early prototyping stages. Moving forwards, this method should be advanced to enable changes in fundamental kinematic structure – including degrees of freedom, as a way to further improve the design process.

VI. ACKNOWLEDGEMENTS

This work was partially supported by Surrey Satellite Technology Ltd and the UK Engineering and Physical Sciences Research Council (EPSRC) grant agreement EP/S035761/1 'Reflexive Robotics'.

REFERENCES

[1] Y. Lui, "Deep reinforcement learning: An overiview," *CoRR*, vol. abs/1701.07274, 2017.

[2] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. M. O. Heess, T. Erez, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *Proc. in 4th International Conference on Learning Representations (ICLR)*, San Diego, Ca, USA, 7-9, May San Juan, Puerto Rico, 2015.

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature (London)*, vol. 529, no. 7587, pp. 484–489, 2016.

[4] T. Kivela, J. Mattila, and J. Puura, "A generic method to optimize a redundant serial robotic manipulator's structure," *Automation in Construction*, vol. 81, pp. 172–179, 2017.

[5] A. Hassan and M. Abomoharam, "Modeling and design optimization of a robot gripper mechanism," *Robotics and Computer-Integrated Manufacturing*, vol. 46, pp. 94–103, 2017.

[6] T. W. Dorn, A. G. Schache, and M. G. Pandy, "Muscular strategy shift in human running: dependence of running speed on hip andankle muscle performance," *Journal of Experimental Biology*, pp. 1944–1956, 2012.

[7] J. Park and H. Asada, "Concurrent design optimization of mechanical structure and control for high speed robots," in *1993 American Control Conference*, Jun. 1993.

[8] T. Ravichandran and D. G. Heppler, "Simultaneous plant-controller design optimization of a two-link planar manipulator," *Mechatronics*, vol. 16, no. 3, pp. 233–242, 2006.

[9] S. F. Alyaqout, P. Y. Papalambros, and A. G. Ulsoy, "Combined robust design and robust control of an electric dc motor," *IEEE/ASME Transactions on Mechatronics*, vol. 16, no. 3, pp. 574–582, 2011.

[10] T. Wang, Y. Zhou, S. Fidler, and J. Ba, "Neural graph evolution: Towards efficient automatic robot design," *CoRR*, vol. abs/1906.05370, 2019.

[11] J. E. Auerbach and J. C. Bongard, "Dynamic resolution in the co-evolution of morphology and control," in *In Proc. of the Twelfth International Conference on Artificial Life (ALIFE XII)*, 2010.

[12] T. F. Nygaard, C. P. Martin, E. Samuelsen, J. Torresen, and K. Glette, "Real-world evolution adapts robot morphology and control tohardware limitations," *In Proceedings ofthe Genetic and Evolutionary Computation Conference*, Jul. 2018.

[13] D. Ha, "Reinforcement learning for improving agent design," *Neural Information Processing Systems Reinforcement Learning Workshop*, 2018.

[14] C. Schaff, D. Yunis, A. Chakrabarti, and M. R. Walter, "Jointly learning to construct and control agents using deep reinforcement learning," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019.

[15] J. Lundell, M. Hazara, and V. Kyrki, "Generalizing movement primitives to new situations," in *Proc. 18th Towards Autonomous Robotic Systems*. Guildford, England. 19-21: Springer International Publishing, Jul. 2017, pp. 16–31.

[16] K. S. Luck, H. B. Amor, and R. Calandra, "Data-efficient co-adaptation of morphology and behaviour with deep reinforcement learning," in *Proceedings of the Conference on Robot Learning, PMLR*, vol. 100, Oct. 2020, pp. 854–869.

[17] F. de A. 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 31 (NeurIPS 2018)*, 2018.

[18] J. Liang and M. Lin, "Differentiable physics simulation," in *ICLR 2020 Workshop on Integration of Deep Neural Models and Differenttial Equations*, 2020.

[19] J. Degrave, M. Hermans, J. Dambre, and F. wyffels, "A differentiable physics engine for deep learning in robotics," *Frontiers in Neurorobotics*, vol. 13, 2019.

[20] T. Chen, Z. He, and M. Ciocarlie, "Hardware as policy: Mechanical and computationalco-optimization using deep reinforcement learning," in *Conference on Robotic Learning (CoRL)*, 2020.

[21] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016.

[22] 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)*, 2019, pp. 6265–6271.

[23] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *CoRR*, vol. abs/1602.01783, 2016.

[24] J. T. Barron, "Continuously differentiable exponential linear units," *CoRR*, vol. abs/1704.07483v1, 2017.

[25] I. Loshchilov and F. Hutter, "CMA-ES for hyperparameter optimization of deep neural networks," *CoRR*, vol. abs/1604.07269, 2016.

[26] T. Akiba, S. Sano, T. Tanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.