Generalizing deep reinforcement learning across cable-driven parallel robot configurations with actuator-level policies

Abir Bouaouda, Mohamed Boutayeb, François Charpillet, Dominique Martinez, Rémi Pannequin

Generalizing deep reinforcement learning across cable-driven parallel robot configurations with actuator-level policies

Abir Bouaouda[a,b], Mohamed Boutayeb[a,b,c], François Charpillet[b], Dominique Martinez[d], Rémi Pannequin[a]

[a] *Université de Lorraine, CRAN, Nancy, F-54000, France*
[b] *Université de Lorraine, INRIA, CNRS, LORIA, Nancy, F-54000, France*
[c] *International University of Rabat, Rabat, 11103, Morocco*
[d] *Aix-Marseille Université, CNRS, ISM, Marseille, F-13009, France*

Abstract

Cable-driven parallel robots (CDPRs) present diverse configurations and complex control challenges, which can be addressed by deep reinforcement learning (DRL) by learning their nonlinear dynamics. However, DRL methods often require extensive training time, and the resulting policies do not generalize well to different robot configurations or varying numbers of actuators. In this article, we introduce a novel DRL approach for controlling CDPRs that does not depend on the specific robot configuration. Our method trains an actuator-level policy that controls each motor to achieve its target cable length, in contrast to conventional DRL approaches that learn to control the entire robot to reach a desired end-effector position. To the best of our knowledge, this is the first work to apply DRL to control CDPRs using an actuator-level policy. This approach offers two main advantages: (i) a single shared policy can be applied to any CDPR configuration, regardless of actuator count, and (ii) reliance on inverse kinematics, avoiding the more challenging forward kinematics problem. Training is performed in simulation, and the learned policy is successfully transferred to a real CDPR. Experimental results show that the actuator-level policy (ALP) surpasses traditional reinforcement learning methods in both robustness and precision. We further control a real 8-motor CDPR with 3D motion using a policy trained on a simulated 4-motor planar CDPR operating in 2D. This illustrates that the proposed method is applicable to any CDPR configuration, independent of actuator number or placement.

Keywords: cable-driven parallel robots, deep reinforcement learning, generalization, actuator-level policy, robot control, transfer learning, inverse kinematics

1. Introduction

Cable-driven parallel robots (CDPRs) have garnered considerable interest in recent years due to their wide range of applications, including medical rehabilitation [1], manufacturing assembly tasks [2], construction [3], and logistics for sorting and packaging [4]. This diversity of applications brings a variety of control challenges, for which Deep Reinforcement Learning (DRL) has emerged as a promising solution, thanks to its ability to capture the complex, non-linear dynamics involved in CDPR control. Some researchers have combined DRL with classical control strategies, while others have relied solely on DRL. For example, in [5, 6], the authors employ the DDPG algorithm to learn a control policy for a CDPR, where the resulting policy is represented by a neural network that takes the robot's state as input and outputs the torques to be applied to its actuators.

However, the use of DRL for controlling CDPRs is not without challenges. One major difficulty is the significant training time required to obtain an effective policy. Moreover, with current state-of-the-art methods, the learned policy often cannot be applied to robots with a different number of actuators, which is a major limitation since the number of actuators may vary between applications. Even when the actuator count remains the same, the policy may not generalize well to different robot configurations, such as changes in mass or actuator positions.

Several approaches have been proposed to address this limitation, including domain randomization [7] and transfer learning [8]. However, these methods are less effective when the input space itself changes.

A recent innovative approach to generalization, "Gato," was proposed by [9]. Gato uses a single network with shared weights to perform a wide range of tasks, such as playing Atari, captioning images, chatting, and controlling a real robot arm, by deciding based on context whether to output text, joint torques, button presses, or other tokens. While promising, Gato requires a very large neural network and often needs fine-tuning for specific tasks, making it impractical for many applications. Furthermore, Gato achieves expert-level performance on only 180 out of 604 simulated control tasks.

While solving the generalization problem for large-scale control tasks, as attempted by Gato, may be an ultimate goal for the robotics community, it is also important to develop more efficient and practical solutions for specific classes of robots, such as Cable-Driven Parallel Robots (CDPRs). CDPRs are particularly suitable for studying generalization because their

Email addresses: firstname.lastname@univ-lorraine.fr (Rémi Pannequin), firstname.lastname@inria.fr (Rémi Pannequin), firstname.lastname@univ-amu.fr (Rémi Pannequin)

fixed-frame actuators allow for many possible configurations with varying numbers of actuators, resulting in different input spaces. This makes the generalization challenge both more difficult and more interesting to address. In this paper, we propose

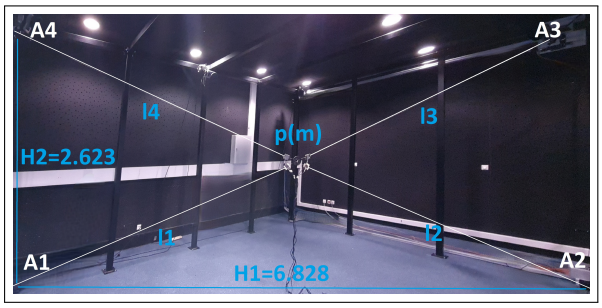

Figure 1: Two-DOF point-mass CDPR with four cables. Here, l_k is the length of cable k, A_k is the position of motor k, and p is the position of the end-effector.

a new approach to address the generalization problem in the control of CDPRs. Our method is based on learning a policy at the actuator level rather than at the robot level. We name this approach "actuator-level policy" (ALP). It could be described as a multi-agent system in which each agent controls a single actuator. The learning is centralized, with a shared policy among all agents, but execution is decentralized, as each agent operates based on its own state and action. The only coordination occurs when computing the target length for each cable, which is determined using the robot's inverse kinematics and the desired end-effector position, ensuring all actuators share the same objective: moving the end-effector to the target.

The main advantage of this approach is that the learned policy can generalize to robots with different numbers and positions of actuators, since the policy is trained to control an actuator in any position, rather than relying on the full robot state. We evaluate our approach on simulated CDPRs with varying actuator counts and on a real robot with eight actuators. Results show that the ALP approach outperforms conventional reinforcement learning methods, as it enables control of a wide range of CDPRs using a single policy trained on a four-actuator robot (see Figure 1). Additionally, it surpasses classical control methods because it does not require direct measurement of the end-effector position—which is often unavailable without a motion capture system—and is more robust to errors in cable length measurements, since it does not depend on forward kinematics for position estimation.

2. Materials and methods

2.1. Cable driven parallel robots control problem formulation

A CDPR is a type of parallel robot in which the end-effector is connected to the base by cables. These cables are actuated by motors, and the end-effector is moved by adjusting the lengths of the cables (see Figure 2). The control problem for a CDPR consists of determining the torques to apply to the motors in order to move the end-effector to a desired 3D position.

Determining the cable lengths required for a given position and orientation of the mobile platform is known as the inverse

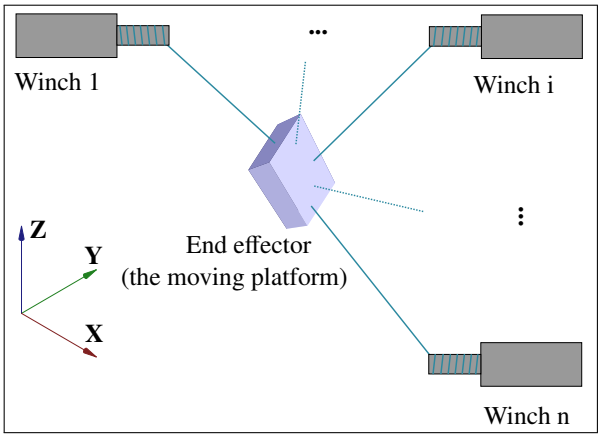

Figure 2: CDPR with n actuators. The end-effector (mobile platform) is connected to the base by n cables, each actuated by a winch motor.

kinematics (IK) problem. Assuming no cable sag, the cable lengths can be computed directly as the norms of the vectors connecting the motors to the end-effector. Conversely, determining the position and orientation of the mobile platform from the cable lengths is called the forward kinematics (FK) problem. While FK can be solved geometrically for translational CDPRs—using the intersection of spheres defined by the cables—it is more complex in the general case and is typically addressed using numerical methods.

In addition to solving the two problems, a model of the robot is needed to simulate its dynamics and to train the control policy. Developing such a model is beyond the scope of this paper; instead, we assume a black-box model that takes the desired motor speeds and the current state of the robot as input and outputs the next state. The model used in this research replicates the behavior of the real robot. For more details on robot modeling, we refer the reader to [10]. The model outputs the robot state, including cable lengths, motor speeds, and motor currents. Using the cable lengths, the position and orientation of the mobile platform can be computed via the robot's forward kinematics.

Having defined the control problem, we now present how it can be addressed using reinforcement learning.

3. Classical reinforcement learning controller for cdprs

Reinforcement learning is a field of machine learning that enables an agent to learn how to control an environment by taking actions based on observed states and receiving rewards. The agent learns a policy that maps states to actions in order to maximize the cumulative reward. In the context of CDPR control, the agent learns a policy that maps the robot's state to control signals applied to the motors to move the end-effector to a desired position.

There are several reinforcement learning algorithms suitable for continuous state and action spaces. In this work, we use three: DDPG [11], PPO [12], and SAC [13], to compare both on-policy and off-policy approaches. DDPG and SAC are off-policy algorithms, while PPO is an on-policy algorithm. The

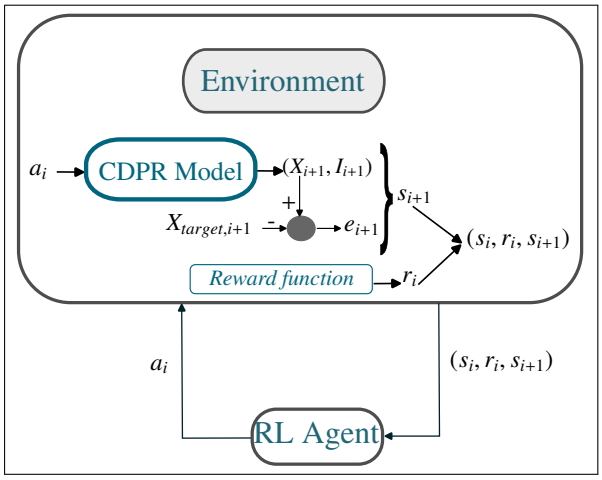

Figure 3: Interaction between the agent and the reinforcement learning environment. The agent receives the tuple (s_t, a_t, r_t, s_{t+1}) and outputs the action a_t to be applied to the motors.

main distinction is that DDPG and SAC use a replay buffer to learn the policy, whereas PPO learns only from the most recent experiences.

In conventional reinforcement learning, the state describes the entire robot (see Figure 3), and is defined as: $s_t = (X_t, X_{t-1}, e_t, e_{t-1}, I_t)$ where X_t is the position of the end-effector, e_t is the error between the current and target positions, and I_t is the motor current. All observations and actions are normalized to the range $[-1, 1]$.

3.1. ALP reinforcement learning controller

The main idea of ALP is to learn a policy that maps the state of each actuator to the torque applied to its motor, rather than mapping the full robot state to all motor torques. The key advantage of this approach is that the learned policy can generalize to robots with different numbers of actuators, since the policy is trained to control an individual actuator rather than the entire robot.

The agent-environment interaction for ALP reinforcement learning is illustrated in Figure 4.

The state of each actuator is defined as:

$$s_t = (L^j_{t-3}, L^j_{t-2}, L^j_{t-1}, L^j_t, L^j_{t,\text{target}}, I^j_t)$$

where L^j_t is the length of cable j at step t, $L^j_{t,\text{target}}$ is the target length of cable j, and I^j_t is the current of motor j. Including the cable length at step $t-3$ provides the agent with information about cable dynamics, tension at the other end, and cable speed, which are important for learning an effective policy (see Appendix A for more details). All observations and actions are normalized to the range $[-1, 1]$. Since there are n actuators, at each step t, the agent performs n interactions with the environment but learns a single shared policy. Thus, only one neural network is used to map the actuator state to the torque, and a single agent learns this policy from samples collected across all actuators.

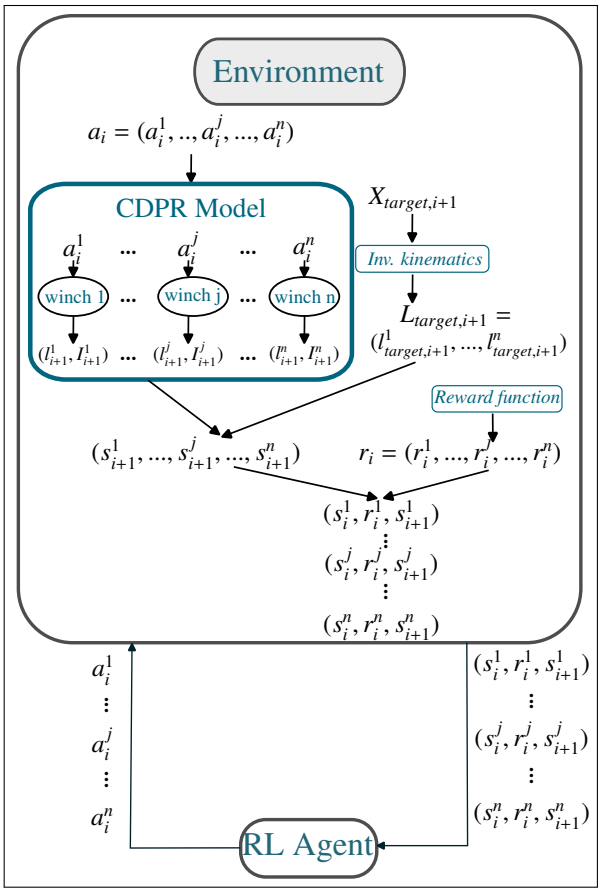

Figure 4: Interaction between the agent and the environment for ALP reinforcement learning. At each step t, the agent interacts with the environment n times, once for each actuator. From each interaction, the agent collects the tuple $(s^j_t, a^j_t, r^j_t, s^j_{t+1})$ and outputs the action a^j_t to be applied to motor j. The agent learns a single policy that maps the state of an actuator to the torque to be applied to its motor.

3.2. Reward-engineering

The reward function is a crucial component of the reinforcement learning algorithm, guiding the agent to learn the optimal policy for a given task. For redundant CDPRs, multiple solutions can achieve the same target position. Therefore, a reward function based solely on position error is insufficient, as it may cause the policy to oscillate between solutions, resulting in noisy control. Furthermore, since our control signal is the motor speed (with a proportional control loop for motor current), the reward function must also account for the control signal to ensure the agent visits only states relevant for effective control.

Error-based reward: The first component of the reward function penalizes the position error: $r_{1,t} = -\|e_t\|$ where e_t is the position error at step t, defined as the distance between the current and target positions of the end-effector. This term encourages the agent to minimize the position error.

Energy optimization reward: The second component penalizes the energy consumed by the motors, following the classical approach to energy optimization. Since energy consumption is related to motor current, we penalize the current as follows: $r_{2,t} = -\|I_t\|^2$ where I_t is the motor current at step t.

Current limits reward: The third component addresses current limits. Since we constrain the current using a saturation function (see Appendix B), we encourage the agent to select actions that keep the current within safe bounds, as actions outside these limits will be saturated and thus ineffective. The reward is: $r_{3,t} = -\left\|a_t - u_{sat,t}\right\|$ where a_t is the action chosen by the agent and $u_{sat,t}$ is the action after applying the saturation function.

So our final reward function is:

$$r_t = \alpha_1 r_{1,t} + \alpha_2 r_{2,t} + \alpha_3 r_{3,t} \tag{1}$$

$$r_t = -\alpha_1 \|e_t\| - \alpha_2 \|I_t\|^2 - \alpha_3 \left\|a_t - u_{sat,t}\right\| \tag{2}$$

where $\alpha_1, \alpha_2, \alpha_3$ are coefficients that weight each term of the reward function. By tuning these coefficients, we adjust the importance of each component to achieve the best performance. After extensive testing, the best results were obtained with $\alpha_1 = 1$, $\alpha_2 = 0.001$, and $\alpha_3 = 0.5$. The energy optimization term (α_2) improves efficiency, while the current limits term (α_3) helps smooth the control signal, though it can slightly degrade tracking performance if set too high. Without this term, the control signal becomes too noisy for real robot use.

3.3. Reward for ALP reinforcement learning

The same reward function can be used for ALP reinforcement learning, where all actuators receive the same reward at each step. In this collaborative setting, an error on one actuator affects the reward for all, as the reward is based on the overall position error. We refer to this as reward 1 (r_1).

Alternatively, an individual reward can be used, based only on each actuator's own error:

$$r_t^j = -\alpha_1 \left|e_t^j\right| - \alpha_2 \left|I_t^j\right|^2 - \alpha_3 \left|a_t^j - u_{sat,t}^j\right| \tag{3}$$

where e_t^j is the error between the current and target lengths of cable j at step t, I_t^j is the current of motor j, a_t^j is the action chosen by agent j, and $u_{sat,t}^j$ is the saturated action. The same coefficients $\alpha_1, \alpha_2, \alpha_3$ are used as in the collaborative reward. We refer to this as reward 2 (r_2). In the training phase, we kept the same values for the coefficients $\alpha_1, \alpha_2, \alpha_3$.

3.4. Target trajectories generation

Visiting all states of the environment is crucial during training, as the agent must learn the optimal action for each possible state. In trajectory tracking tasks, the desired trajectory forms part of the state and can be generated as needed. However, it is important to ensure that generated trajectories are consistent with the robot's dynamic limits and sufficiently explore the entire state space.

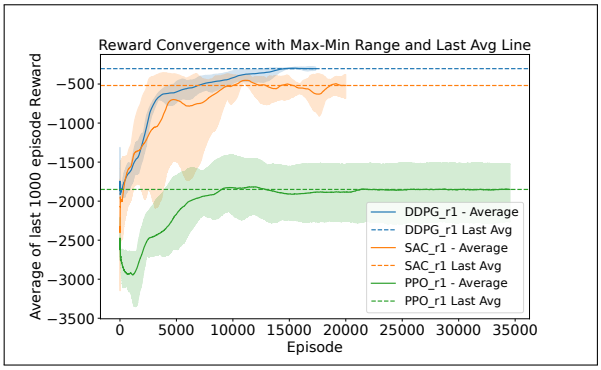

Figure 5: Average reward of the ALP reinforcement learning approach. The solid curve shows the mean reward over four runs with different random seeds, while the shaded region represents the range (min–max) across runs. Each experiment was repeated four times with different random seeds.

To address this, we generate random trajectories with random speeds and accelerations. Since the robot's workspace is bounded and there are constraints on speed and acceleration limits, it is essential to ensure that the generated trajectories comply with these constraints. For this purpose, we use the method described in Appendix C.

4. Experiments and results

4.1. Training setup

The training was performed only on the 4-cable translational robot, assuming the mobile platform is a point mass and all cables are attached to the same point on the platform (see Figure 1). The robot parameters are provided in Appendix D, Table D.3. Training was conducted using three reinforcement learning algorithms: DDPG, PPO, and SAC, with hyperparameters detailed in Appendix D. Table D.5.

4.2. Training results for ALP

To analyze convergence time and maximum average return, we conducted training using the hyperparameters presented in the previous section, repeating the process four times with different random seeds. Each agent was trained using the same hyperparameters as the conventional reinforcement learning approach, with reward function 1 described in Section 3.2. Figure 5 shows that the PPO agent does not converge, while the

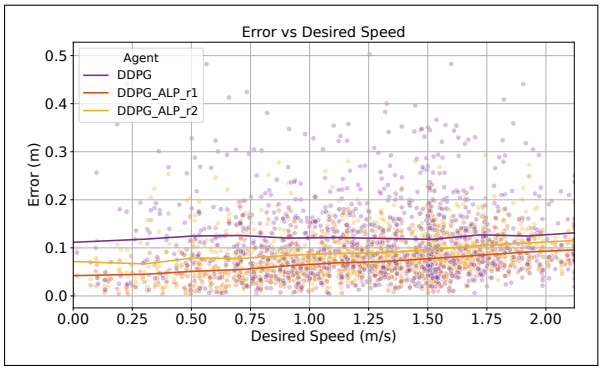

Figure 6: Mean error between the target position and the end-effector position for the 4-cable robot over 10 episodes as a function of speed, using three different policies: DDPG with the conventional RL approach, DDPG with the ALP RL approach with reward 1, and DDPG with the ALP RL approach with reward 2.

DDPG and SAC agents both converge to good policies. The DDPG agent achieves a higher and more stable average reward than the SAC agent, as indicated by the lower variation in average reward. Therefore, we selected the DDPG agent for the remainder of the experiments due to its stability and reliable convergence.

4.3. Testing results in simulation and comparison between reward functions

We trained the ALP DDPG policy using both the collaborative reward function 1 and the individual reward function 2, as described in Section 3.2, on the 4-cable robot (see Figure 1) for four different runs with random . The same procedure was applied to the DDPG with the conventional RL approach, using the same hyperparameters and reward function 1. For each run, we saved the best policy and used it for the testing phase.

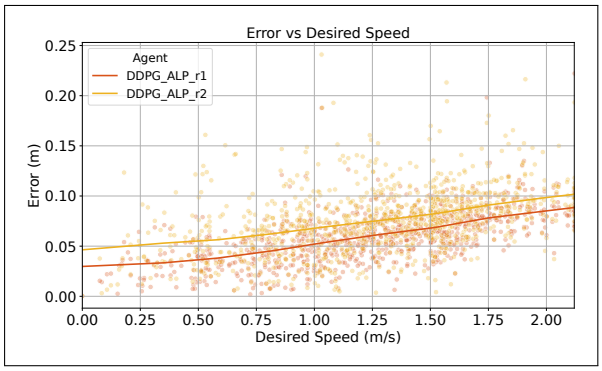

Figure 7: Mean error between the target position and the end-effector position for the 4-cable robot configuration with $H_1 = 5$ m and $H_2 = 5$ m (see Figure 1) over 10 episodes as a function of speed, using the ALP RL approach with reward 1 (cooperative reward) and reward 2 (individual reward).

We tested the three policies on the 4-cable robot with different speeds and target positions (10 episodes of 10 seconds each, as in training), using the same target positions for all policies. In Figure 6, the error ranges from 0 to 0.5 m for the conventional RL approach and from 0 to 0.25 m for the ALP RL approach, indicating that the ALP RL outperforms the conventional RL approach. We also observe that the policy using reward 1 (cooperative reward) outperforms the policy using reward 2 (individual reward). This difference may be attributed to the reward coefficients not being optimally tuned for the individual reward, as the same coefficients were used for both reward functions, and were primarily tuned for the cooperative reward in the conventional RL approach.

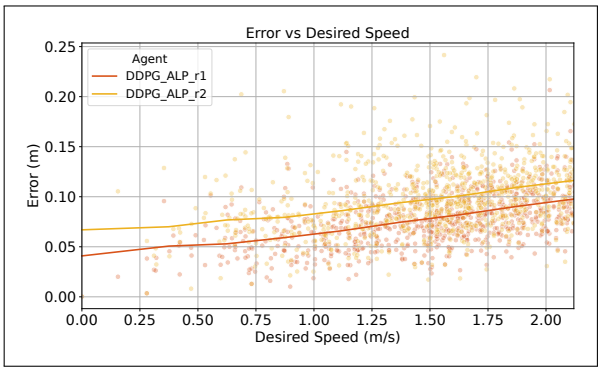

Figure 8: Mean error between the target position and the end-effector position for the 8-cable robot configuration over 10 episodes as a function of speed, using the ALP RL approach with reward 1 (cooperative reward) and reward 2 (individual reward).

However, the real advantage of the individual reward becomes apparent during fine-tuning, when additional training is performed in the implementation phase to achieve optimal performance. The cooperative reward can lead to unstable learning if the robot configuration differs from that used during training, as the scale of the cooperative reward may vary significantly between configurations. In contrast, the individual reward is more stable, since it depends only on the error of each actuator, making it suitable for fine-tuning across different configurations without issue.

Another important observation is that the performance of the ALP reinforcement learning approach surpasses that of the DDPG with the conventional RL approach. Thus, the ALP reinforcement learning approach outperforms the conventional RL approach, as it can be applied to different robot configurations with varying numbers of actuators without requiring additional training.

4.4. Testing results on diffrent configurations

We tested the three policies on the 4-cable robot with a different configuration (workspace with $H_1 = 5$ m and $H_2 = 5$ m, see Figure 1), using the same target positions for all policies. The conventional RL approach fails to track the desired trajectory when trained on a different configuration; it can only track trajectories in new configurations if the difference is small and the number of actuators remains the same.

In Figure 7, we observe that the ALP reinforcement learning approach successfully tracks the desired trajectory in the new configuration, even though it was trained only on a 4-cable robot with a different setup. This is because the policy is learned to control each actuator independently, rather than the entire

robot, allowing it to be applied to any configuration without additional training.

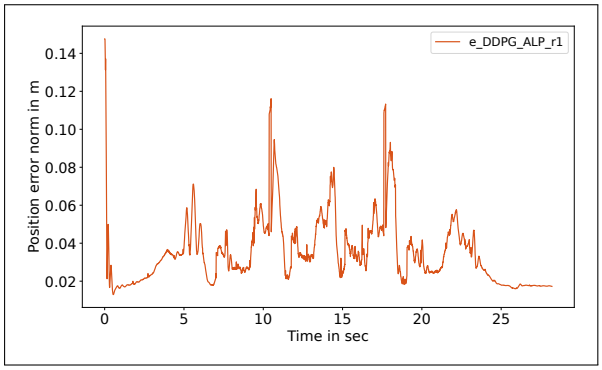

Figure 9: Trajectory tracking of the real robot using the ALP reinforcement learning approach with reward 1. The error between the target position and the end-effector position is less than 0.04 m.

We also tested both policies on the 8-cable translational robot, where the robot has 8 actuators and the end-effector is a point mass, with all cables attached to the same point on the mobile platform. The same target positions were used for both policies. As shown in Figure 8, the performance of both policies on this configuration is similar to their performance on the initial configuration, demonstrating the transferability of the policy learned on the 4-cable robot to the 8-cable robot.

4.5. Testing results on the real robot

We tested the best policy using reward 1 on the real robot in the 8-cable translational robot configuration. The policy's performance on the real robot closely matches its performance in simulation. As shown in Figure 9, the agent is able to track the desired trajectory with high accuracy, and the error between the target position and the end-effector position is less than 0.04 m.

4.6. Robustness to errors in the position estimation

Table 1: A comparison of the performance of the different controllers

Configuration \ Controller	Classical	RL	ALP
4 cables	✓	✓	✓
Different mass than training ($m = 0.5, \ldots, 2$ kg)	✓	✓	✓
Other configurations than training	✓	✓	✓
n cables	✓	×	✓
Error in position estimation	×	×	✓

Most classical controllers use the position of the end-effector to compute the control signal. In the absence of direct position measurement (i.e., no sensor for the end-effector position), the position is estimated using the robot's forward kinematics. Errors in position estimation can significantly degrade controller performance, especially if a cable measurement is incorrect or a cable is sagged, resulting in inaccurate end-effector position and poor trajectory tracking.

In contrast, our ALP reinforcement learning approach does not rely on estimating the end-effector position. The control signal for each actuator is generated independently of the lengths of the other cables, allowing the system to remain robust even if there are errors in the measurement of one cable. As a result, the ALP RL controller can still track the desired trajectory under such conditions.

Table 1 summarizes the major advantages of the ALP reinforcement learning approach compared to classical controllers.

5. Conclusions

In this work, we have proposed a new reinforcement learning-based control approach for cable-driven parallel robots using an actuator-level policy. This method enables systematic generalization, allowing control of any cable robot configuration. Simulation results demonstrate strong performance compared to conventional reinforcement learning approaches, and the proposed method has been validated on a real cable robot, confirming its transferability to different configurations and to real-world hardware.

Limitations: The comparison between different reinforcement learning algorithms remains inconclusive, mainly due to the manual tuning of hyperparameters, which significantly affects performance. Similarly, tuning the reward function coefficients is challenging and does not guarantee optimal results. Another limitation is that the CDPR configurations considered in this paper are restricted to translational CDPRs with a point mass mobile platform. The proposed approach could be extended to rotational CDPRs with a rigid body mobile platform; preliminary results are promising, but further work is needed to fully validate this extension.

Appendix A. Proof of concept

To demonstrate the feasibility of our approach, we use the dynamic model of a single cable.

Assumptions

The following assumptions are made in deriving the dynamic model:

- The cable is inextensible.
- The cable is wound around the winch without slipping.
- There is no dry or viscous friction acting on the system.
- The winch applies a torque $\tau(t)$ which is converted to a tension $T(t)$ at one end of the cable, proportional to the current $I(t)$ flowing through the motor.
- The mobile platform applies a force $F_{\text{ext}}(t)$ at the other end of the cable.
- The length of the cable between the winch and the end-effector is $L(t)$.
- The mass of the cable between the winch and the end-effector is $m(t)$.

Dynamic model of the cable

Applying Newton's second law to the cable:

$$m(t)\ddot{L}(t) = F_{\text{ext}}(t) - T(t) \tag{A.1}$$

$F_{ext}(t)$ and $m(t)$ estimation

Assuming the cable is inextensible and has uniform mass, $m(t)$ can be estimated from the cable length and mass per unit length. $F_{\text{ext}}(t)$ can be estimated from (A.1) using the estimated $m(t)$, the measured tension $T(t)$ (proportional to the motor current $I(t)$), and the measured values of $L(t)$ over recent time steps.

Existence of a control law

Let $\hat{F}_{\text{ext}}$ and $\hat{m}$ be estimates of F_{ext} and m, with:

$$\begin{aligned} \hat{F}_{\text{ext}}(t) &= F_{\text{ext}}(t) - \epsilon_{\text{ext}}(t) \\ \hat{m}(t) &= m(t) - \epsilon_m(t) \end{aligned} \tag{A.2}$$

where $\epsilon_{\text{ext}}(t)$ and $\epsilon_m(t)$ are small bounded errors.

The objective is to track a desired cable length $L_d(t)$, i.e., to choose $T(t)$ such that:

$$\|L_d(t) - L(t)\| = \|e(t)\| \leq \epsilon_d \tag{A.3}$$

where ϵ_d is small (or zero if estimation errors vanish).

Assume $L(t)$ and its derivatives are bounded, which holds for the cable robot.

If we choose:

$$T(t) = \hat{F}_{\text{ext}} - \hat{m}\left(\ddot{L}_d(t) + k_v\dot{e}(t) + k_p e(t)\right) \tag{A.4}$$

then substituting into (A.1) yields:

$$F_{\text{ext}} - \hat{F}_{\text{ext}} + \hat{m}\left(\ddot{L}_d(t) + k_v\dot{e}(t) + k_p e(t)\right) = m\ddot{L}(t) \tag{A.5}$$

$$\epsilon_{\text{ext}}(t) + \hat{m}\left(\ddot{e}(t) + k_v\dot{e}(t) + k_p e(t)\right) = \epsilon_m\ddot{L}(t) \tag{A.6}$$

$$\ddot{e}(t) + k_v\dot{e}(t) + k_p e(t) = \epsilon(t) \tag{A.7}$$

with

$$\epsilon(t) = \frac{\epsilon_m\ddot{L}(t) - \epsilon_{\text{ext}}(t)}{\hat{m}}$$

This is a second-order equation, and by choosing $k_v > 0$ and $k_p > 0$, we ensure convergence as in (A.3). The choice of k_v and k_p can also be formulated as a convex optimization problem.

Thus, this control law can be learned by a neural network using a reinforcement learning agent, provided the actuator state includes the cable length at previous time steps (to estimate cable speed and tension at the other end) and the motor current (to estimate the force applied to the cable).

Appendix B. Current constraints

Current control loop

Cable-driven parallel robots use cables to transmit forces from the motors to the end-effector. The control signal is typically the cable tension, but since tension is not directly measurable, the motor current is used instead. The CDPR in our lab employs a proportional control loop where the motor current is proportional to the difference between the desired speed and the measured speed of the motors:

$$i = K_{motor}(u - v_m)$$

where i is the motor current, K_{motor} is the motor constant, u is the desired motor speed, and v_m is the measured motor speed.

From this control loop, any desired speed less than the measured speed results in a negative desired current, meaning the motor rotates in the opposite direction and cable tension drops to zero, which is unsafe. While we typically control the desired speed, having access to the measured speed allows us to adjust the desired speed to respect current limits.

Current bounds

Cables in a CDPR must always remain in tension; slack cables risk entanglement and loss of robot mobility, requiring intervention to restore safety. To prevent this, we set a minimum current limit i_{min}. Conversely, the current must not exceed a maximum limit i_{max} to avoid damaging the motors, cables, or end-effector. Thus, we enforce:

$$i_{sat} = \begin{cases} i_{max} & \text{if } i > i_{max} \\ i & \text{if } i_{min} < i < i_{max} \\ i_{min} & \text{if } i < i_{min} \end{cases}$$

Based on the current control loop, the agent's action is computed as:

$$u_{sat} = \frac{i_{sat}}{K_{motor}} + v_m$$

By computing the agent's action this way, we ensure the motor current remains within safe limits and cables stay in tension. Mapping unsafe actions to the nearest safe action ensures the agent does not select unsafe commands and learns to control the robot across all states of the environment.

Appendix C. Desired trajectory generation: the case of trajectory tracking in bounded workspace

Table C.2: Target generation parameters

Parameter	Value
Maximum acceleration in x-axis	35 m/s^2
Maximum acceleration in z-axis	35 m/s^2
Maximum velocity in x-axis	4 m/s
Maximum velocity in z-axis	4 m/s

The trajectories are generated as follows: First, we set the maximum acceleration a_{max} and the maximum speed v_{max}. A

random initial position X_0 is chosen at the start of each episode, with speed and acceleration reset to zero. At each time step t, a random acceleration a_t is generated from a Gaussian distribution such that $-a_{max} \leq a_t \leq a_{max}$. The speed v_t is then updated by integrating the acceleration:

$$v_t = v_{t-1} + a_t \cdot dt$$

If the speed exceeds the limits, it is clipped to the boundary value. The trajectory is then generated by updating the position:

$$X_t = X_{t-1} + v_t \cdot dt$$

This approach works very well for low speeds or in an unbounded workspace, but at high speeds, the end-effector often reaches the workspace limits before the episode ends. To address this, we initially tried reducing the episode length, but this led to instability due to fewer reward samples for estimating the average return.

Instead of shortening the episode, we chose to randomly reverse the desired speed v_t when the end-effector reaches the workspace boundaries, keeping it within the workspace. While this method works for high speeds, it is not ideal, as the resulting trajectories may not be smooth or fully consistent with the robot's dynamic limits. However, it is acceptable during training because it ensures the agent explores all states of the environment and learns to control the robot under various conditions.

We use the parameters shown in Table C.2 to generate trajectories for our CDPR. The velocity limits are set to match the robot's capabilities, with the nominal motor speed at 3000 rpm, equivalent to 4 m/s for the cables. The acceleration limits are set higher than the robot's maximum acceleration to allow reaching maximum speed within the limited episode length, and because acceleration is sampled from a random Gaussian distribution, the robot does not reach maximum acceleration in all states.

Appendix D. Hyperparameters and cdprs parameters

The training process is performed in simulation using the OpenAI Gym environment. The CDPR used is a 4-cable robot with parameters listed in Table D.3.

For each agent, a manual hyperparameter tuning process was performed to achieve the best average return and fastest convergence. The training time depends on available computational resources and the experiment setup. During hyperparameter tuning, training is stopped when the agent's average return stabilizes; for the final training, a fixed number of episodes is used to allow the agent to learn the optimal policy for the task. Table D.4 presents the training time per episode and per experiment for each agent.

Table D.3: CDPRs parameters

Parameter	Symbol	Value
Nominal current	I_{nom}	6.8 A
Nominal speed	V_{nom}	3420 rpm
Nominal torque	T_{nom}	0.8 Nm
Control signal frequency	f_e	100 Hz
Motor torque constant	K_c	0.123 Nm/A
Motor current proportional gain	K_{motor}	1 A/rps
Drum radius	r	0.0178 m
End-effector mass	m	1 kg
Time step	dt	0.01 s

Table D.4: Training Time

Agent	Training time per episode	Number of episodes	Training time per experiment
DDPG SAC	$6 - 10(s)$	$1 - 3 \cdot 10^4$	$16 - 85(h)$
PPO	$4.5 - 8(s)$	$4 - 8 \cdot 10^4$	$50 - 100(h)$

Table D.5: Hyperparameters

Hyperparameter	DDPG	PPO	SAC
Max number of episodes	$3 \cdot 10^4$	$8 \cdot 10^4$	$3 \cdot 10^4$
Max number of steps	1000	1000	1000
Q learning rate	$4 \cdot 10^{-5}$	-	$3 \cdot 10^{-4}$
Value learning rate	-	$1 \cdot 10^{-5}$	$3 \cdot 10^{-4}$
Actor learning rate	$2 \cdot 10^{-5}$	$2 \cdot 10^{-5}$	$3 \cdot 10^{-4}$
Update rate	$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$
Discount factor	0.99	0.99	0.99
Buffer size	50000	-	50000
Batch size	128	50	128
Noise deviation	0.033	-	-
Alpha coeff.	-	-	0.01
GAE	-	0.95	-
Log std dev	-	-1.3	-
Epoch episodes	-	10	-
Iterations	-	4	-
Hidden layers	2	2	2
Hidden units	128	128	128

References

[1] D. Surdilovic, R. Bernhardt, STRING-MAN: a new wire robot for gait rehabilitation, in: IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004, IEEE, New Orleans, LA, USA, 2004, pp. 2031–2036 Vol.2. URL: http://ieeexplore.ieee.org/document/1308122/. doi:10.1109/ROBOT.2004.1308122.

[2] A. Pott, C. Meyer, A. Verl, Large-Scale Assembly of Solar Power Plants with Parallel Cable Robots (2010).

[3] R. L. Williams, M. Xin, P. Bosscher, Contour-Crafting-Cartesian-Cable Robot System: Dynamics and Controller Design, in: Volume 2: 32nd Mechanisms and Robotics Conference, Parts A and B, ASMEDC, Brooklyn, New York, USA, 2008, pp. 39–45. URL: https://asmedigitalcollection.asme.org/IDETC-CIE/proceedings/IDETC-CIE2008/43260/39/321910. doi:10.1115/DETC2008-49480.

[4] J. Lamaury, M. Gouttefarde, Control of a large redundantly actuated cable-suspended parallel robot, in: 2013 IEEE International Conference on Robotics and Automation, IEEE, Karlsruhe, Germany, 2013, pp. 4659–4664. URL: http://ieeexplore.ieee.org/document/6631240/. doi:10.1109/ICRA.2013.6631240.

[5] T. Ma, H. Xiong, L. Zhang, X. Diao, Control of a Cable-Driven Parallel Robot via Deep Reinforcement Learning, in: 2019 IEEE International Conference on Advanced Robotics and its Social Impacts (ARSO), IEEE, Beijing, China, 2019, pp. 275–280.

[6] C. Sancak, F. Yamac, M. Itik, Position control of a planar cable-driven parallel robot using reinforcement learning, Robotica (2022) 1–18. Publisher: Cambridge University Press.

[7] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, P. Abbeel, Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World, 2017. URL: http://arxiv.org/abs/1703.06907, arXiv:1703.06907 [cs].

[8] A. A. Rusu, M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, R. Hadsell, Sim-to-Real Robot Learning from Pixels with Progressive Nets, 2018. URL: http://arxiv.org/abs/1610.04286, arXiv:1610.04286 [cs].

[9] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Gimenez, Y. Sulsky, J. Kay, J. T. Springenberg, T. Eccles, J. Bruce, A. Razavi, A. Edwards, N. Heess, Y. Chen, R. Hadsell, O. Vinyals, M. Bordbar, N. de Freitas, A Generalist Agent, 2022. URL: http://arxiv.org/abs/2205.06175, arXiv:2205.06175 [cs].

[10] A. Pott, Cable-Driven Parallel Robots: Theory and Application, volume 120, Springer International Publishing, 2018.

[11] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, 2019. URL: http://arxiv.org/abs/1509.02971, arXiv:1509.02971 [cs, stat].

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal Policy Optimization Algorithms, 2017. ArXiv:1707.06347 [cs].

[13] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, S. Levine, Soft Actor-Critic Algorithms and Applications, 2019. ArXiv:1812.05905 [cs, stat].