Robotics and Computer–Integrated Manufacturing 78 (2022) 102382

Contents lists available at ScienceDirect

Robotics and Computer-Integrated Manufacturing

journal homepage: www.elsevier.com/locate/rcim

Full length article

Synthesizing the optimal gait of a quadruped robot with soft actuators using deep reinforcement learning

Qinglei Ji [a,b], Shuo Fu [b], Kaige Tan [b], Seshagopalan Thorapalli Muralidharan [b], Karin Lagrelius [c], David Danelia [b], Georgios Andrikopoulos [b], Xi Vincent Wang [a], Lihui Wang [a], Lei Feng [b,*]

[a] Department of Production Engineering, KTH Royal Institute of Technology, Stockholm 10044, Sweden
[b] Department of Machine Design, KTH Royal Institute of Technology, Stockholm 10044, Sweden
[c] Department of Computer Science, KTH Royal Institute of Technology, Stockholm 10044, Sweden

ARTICLE INFO

Keywords:
Quadruped robot
Soft actuators
Tendon-driven motion
Reinforcement learning
Robot gait
Motion control

ABSTRACT

Quadruped robots have the advantages of traversing complex terrains that are difficult for wheeled robots. Most of the reported quadruped robots are built by rigid parts. This paper proposes a new design of quadruped robots using soft actuators driven by tendons as the four legs. Compared to the rigid robots, the proposed soft quadruped robot has inherent safety, less weight and simpler mechanism for fabrication and control, but the corresponding challenge is that the accurate mathematical model applicable to model-based control design of the soft robot is difficult to derive by dynamics. To synthesize the optimal gait controller of the soft-legged robot, the paper makes the following contributions. First, the flexible components of the quadruped robot are modeled with different finite element and lumped parameter methods. The model accuracy and computation efficiency are analyzed. Second, soft actor–critic methods and curriculum learning are applied to learn the optimal gaits for different walking tasks. Third, The learned gaits are implemented in an in-house robot to transport hand tools. Preliminary results show that the robot can walk forward and correct the walking directions.

1. Introduction

Legged robots utilize multiple legs for locomotion and have been widely used for terrain exploration. Among all existing categories of legged robots, quadruped robots have the advantage of balanced properties on structural complexity, mobility, and locomotive stability [1,2]. Most of the reported quadruped robots use rigid components such as beams and bearings to transfer the rotational forces from electric motors to the robot leg joints for generating motion. Such robots enjoy fast-response and motion accuracy, but suffer from limitations in their motion ranges caused by the rigid mechanical structures of their joints and electric motors [3].

In recent years, quadruped robots that utilize highly compliant materials to build their bodies have emerged as the main counterpart of the aforementioned rigid approaches. Such robots predominantly use elastomeric polymers for fabricating their actuators, which can be driven by magnetic field [4,5], electric force [6], and most frequently, by pneumatic sources [3,7]. However, these approaches are either described by low energy density or require heavy power sources onboard the robot chassis for extending operating time [8], which greatly limits overall performance. Tendon-driven soft actuators on the other hand, use tendons to transmit forces from, most frequently, electric motors, and have become an ideal solution since they provide a balance between compliance and fast-response. Tendon-driven soft actuators can operate as the legs of quadruped robots [9,10]. Compared to their rigid counterparts, these robots have significantly reduced weight and mechanical complexity caused by the multiple leg joints of the rigid robots. Furthermore, their legs are inherently compliant to the terrain, thus having increased capability of traversing complicated environments.

Compared to rigid legs whose dynamic movement model can be created using multi-body dynamics, soft actuators are hard to model and control due to their high non-linearity, hysteresis in actuation and recovery, drift in repeated actuation, and the involvement of continuum mechanics [11]. Current physical models for soft actuators have to simplify the force distribution on the actuators [12,13] and then develop the models through mechanical analysis [14]. However, theoretical models become hard to acquire in the tendon-driven legged robots where the actuators have to sustain the forces from the driven tendons, the gravitational force of the robot, and the fast-changing

* Corresponding author.
E-mail address: lfeng@kth.se (L. Feng).

https://doi.org/10.1016/j.rcim.2022.102382
Received 6 January 2022; Received in revised form 18 April 2022; Accepted 22 May 2022
Available online 4 June 2022

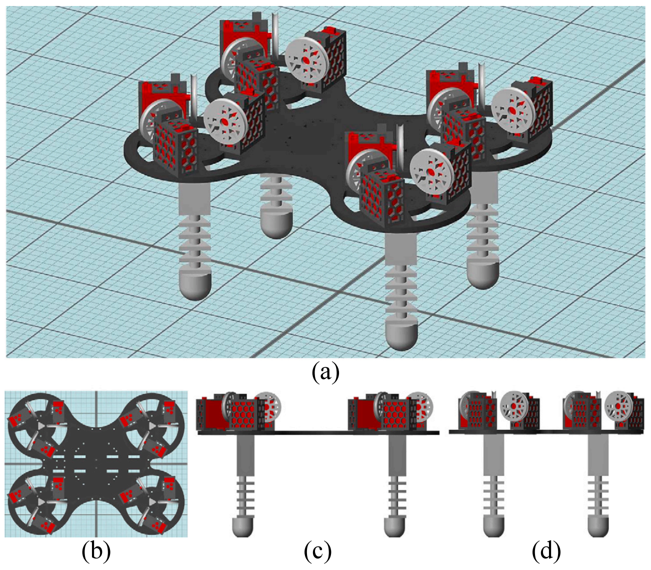

Fig. 1. Rendered view of the modeled quadruped robot. (a) Isometric view. (b) Top view. (c) Left view. (d) Back view.

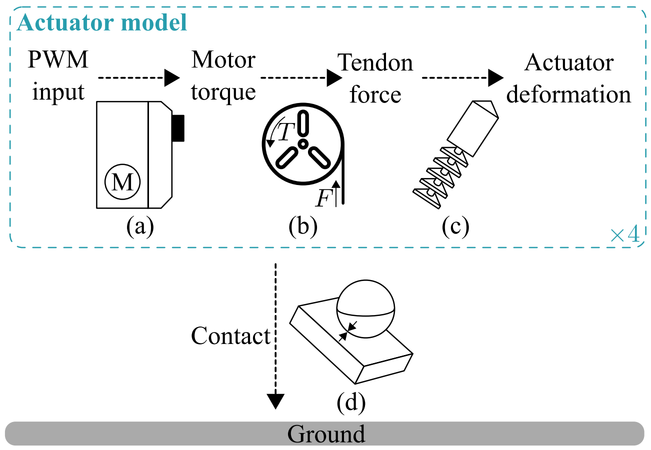

Fig. 2. Overview of the quadruped robot's main subsystems enabled by tendon-driven soft actuators. (a) Servo motor translates PWM signal input to rotational motion control while applying torque on the rotational spool. (b) The spool converts the torque from the servo motor to the traction force in the tendon. (c) The tendon force is applied on the actuator and activates its deformation. (d) The deformation of the actuator causes the relative motion between the feet of the quadruped robot and the different ground conditions, which then cause the robot's motion.

reaction force from the ground. As a result, how to design a suitable motion controller for soft quadruped robots remains unsolved.

The locomotion control of quadruped robots normally follows pipelined approaches that include state estimation, contact scheduling, foot placement planning, etc [15,16]. These processes require expert experiences as well as accurate dynamic models of the quadruped robot. On the other hand, model-free Reinforcement Learning (RL) methods do not require any prior knowledge about the robot system. RL algorithms can interact with the robot motion environment, either physically or via simulation, and improve the motion controller during the interaction to meet the motion target set by human operators.

Numerous reports on the modeling and learning of rigid quadruped robots have been reported [17–19] and many RL algorithms have been developed and proved to be feasible for quadruped robot gait learning [20–22]. Rong et al. [23] synthesized the model of a hydraulic actuated quadruped robot using dynamic equations and simulated the robot using ADAMS and MATLAB to guide the mechanism design and parameter preference for the robot development. Hu et al. [24] used Distributed Proximal Policy Optimization (DPPO) for the learning and optimization of the open source Doggo robot. It is found that providing prior designed gait to the learning process can greatly increase the training efficiency and the learned gait achieves 50% faster walking speed than manually designed trot gaits. Haarnoja et al. [16,21] focused on improving the sample efficiency of robot learning with RL methods and proposed the Soft Actor–Critic (SAC) method that can increase the learning speed of the training process. The proposed method demonstrated faster convergence and better robustness for the walking gait learning of a quadruped robot, and the decreased number of learning episodes enabled the direct data collection using real robots.

As highlighted from reviewing the related literature, very few studies tackle the modeling and control of soft quadruped robots. To this goal, this study reports the modeling process of a quadruped robot enabled by four tendon-driven continuum actuators [10]. Each soft actuator is driven by three servo motors and can hence move in three Degrees of Freedom (DoF), including rotations in two orthogonal directions and the change of length. Section 2 develops the physical model of the quadruped robot by dividing it into several subsystems including the motor driven units, the tendon transmission mechanism, the flexible structure deformation system and the ground contact subsystem. The characterization process for each subsystem is described in detail, while different simplification methods for the subsystems are compared to balance the model accuracy and computation time cost. Section 2.5 builds the entire dynamic model in MATLAB's Simscape environment as demonstrated in Fig. 1. When compared with rigid quadruped robots, the nonlinear properties and continuum mechanics of the utilized flexible legs render the design and control of moving gaits challenging. After comparing different RL algorithms, Section 3 applies the SAC method to learn the robot's walking controller and the training results for different walking speeds are presented. The learned gaits are also compared and analyzed. In the experimental validation presented in Section 4, a robot prototype is developed and the controller learned from the simulated environment is implemented to test walking performance in a real-life environment. Preliminary results show that the robot can walk forward with the learned controller and the forwarding direction can also be automatically corrected. Section 5 concludes the paper.

2. Quadruped robot modeling

As illustrated in Fig. 2, the model of the quadruped robot is divided into the following subsystems: (a) The servo motor that receives Pulse-width modulation (PWM) signals and generates rotational torque via its shaft, which is then transmitted to the following subsystems. It is assumed that the rotational angle of the servo motor follows the input PWM signals without error. (b) The spool that is connected to the shaft of the servo motor converts the motor torque to the force in the tendon. The two ends of the tendon are connected to the spool and the end disk of the following continuum structure. When the spool rotates, the rotational motion of the spool also causes a corresponding length change of the tendon. (c) The tendon force is applied on the actuator and causes deformation. The force balance between the tendon force and reaction force of the actuator determines the deformation dynamics. Three of the motor and spool systems apply forces at equally distributed radial locations of the actuator and can lead to omnidirectional bending motion. Four of the above actuator systems (a)–(c) are assembled together as the four legs of the quadruped robot. (d) The robot is placed on the ground with gravitational forces in the simulation environment. The different motions of the four legs cause relative motion with the ground and generate static or dynamic friction, which then lead to the robot's movement. Different ground conditions have different frictional properties and can result in different motions.

For this study, the MATLAB Simscape toolbox is used to build the above models, while the following subsections will elaborate the aforementioned subsystems for assembling the quadruped robot.

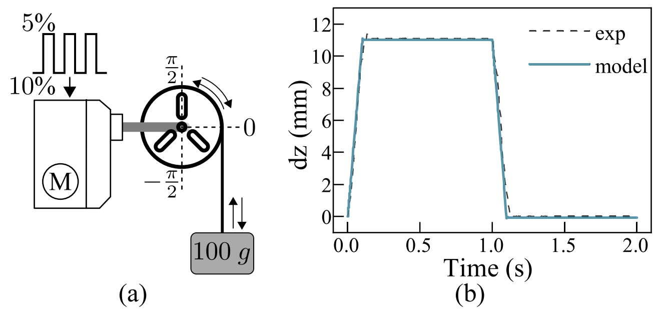

Fig. 3. Motor characterization test. (a) Experimental setup for identifying the motor parameters using a known load. (b) The dynamic responses of the know load actuated by the servo motor rotating to $\pi/6$ and returning to neutral position.

Table 1
Identified servo motor parameters.

Parameters	Value
Stall torque (cm $*$ kg)	11.2
Time traveling 60° (s)	0.2
Nominal voltage (V)	6
Rotational range (rad)	$[-\pi/2, \pi/2]$

2.1. Modeling of the servo motor

The servo motor used in the study is the TS-411MG from TrackStar. The servo motor receives PWM signals with duty cycles from 5% to 10% and then regulates the rotational angle from $\pi/2$ to $-\pi/2$ proportionally using the built-in position controller as illustrated in Fig. 3(a). The middle position 0 of the spool is considered as the neutral position, where the actuator is in its straight shape. When the spool angle rotates to positive values, the tendon is pulled up. For negative values, the tendon turns into slack mode. To characterize the responses of the servo motor, the spool with tendon is installed on the servo motor and different loads are connected to the tendon as shown in Fig. 3(a). Then, different PWM signals are sent to the servo motor and the movement of the load in the vertical direction is recorded.

The same setup is built in the simulation environment and the parameters of the servo motor including the stall torque and time of traveling 60° are tuned until the deviation between the experimental and the simulation data is minimal. Fig. 3(b) shows an example of the comparison for the spool rotating to $\pi/6$ at $t = 0$ s with a load of 100 g and then recover to 0 at $t = 1$ s. The corresponding servo motor parameters for simulation are shown in Table 1. In this study, to enable enough bending of the actuator while still maintaining the balance of the entire robot, the maximum rotational angle of the servo motor is set as $\bar{\alpha}_M = \pi/6$, meaning that the rotational range of the servo motor is as $[-\bar{\alpha}_M, \bar{\alpha}_M]$. The corresponding PWM signal range is [6.67%, 8.33%].

2.2. Continuum actuator design and modeling

As graphically displayed in Fig. 4, the tendon-driven continuum actuator is composed of a sequential combination of rigid disks and flexible cores. Fig. 4(a) shows the typical continuum actuator design using rounded rigid disks. The three driven tendons are equally distributed at a radial formation around the rigid disks of the actuator and passed through respective guiding holes placed on the rigid disks. The rounded rigid disk design makes it time-consuming to fabricate with 3D printing (3DP) [10] rapid prototyping method, as it requires post-processing to manually remove the needed support structures for achieving a good printing result. In this study, we modified the rigid disks to triangle shapes (Fig. 4(b)) so that the actuators can be 3D printed with fewer support material. Furthermore, the length of the actuators with core and disk structure is also reduced and the upper

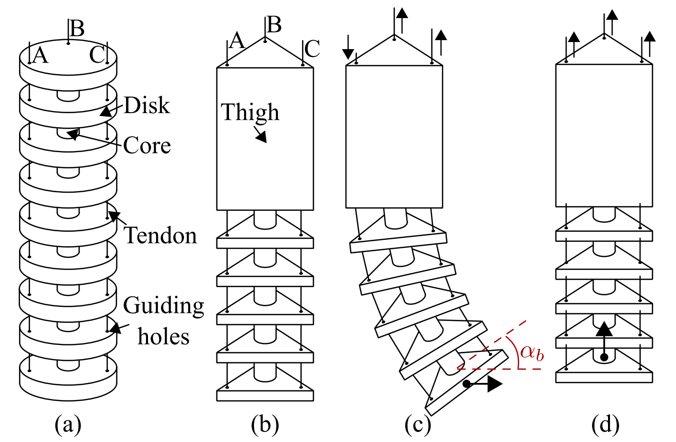

Fig. 4. Graphical representations of tendon-driven continuum actuators. (a) Typical design. (b) Redesigned actuator for easier manufacturing via 3D printing technologies and increased stability. (c) Actuator bending. (d) Actuator compression.

thigh is printed as solid to offer more rigidity with the same length as our previous design [10]. The modification reduces the 3DP time of a single actuator from 11 h to 5 h. For a specific actuator length, the ratio between the length of the actuator with core and disk and the length of the thigh influences the balance of the overall bendability and the longitudinal stiffness of the actuator. Larger ratio offers larger bendable range of the soft actuator while also decrease the longitudinal stiffness, which can decrease the payload of the robot using these soft actuators as the legs. On the other hand, smaller ratio increases the longitudinal stiffness while also reduces the mobility of the robot.

The morphing of the actuators is realized by the traction force exerted via the tendons. As demonstrated in Fig. 4(b) and (c), when the tendons B and C pull up and the other tendon A releases, the actuator bends towards the traction direction. The bending angle α_b of the actuator is defined as the angle between the end disk plane and the horizontal plane as illustrated in Fig. 4(c). When three of the cables pull together with the same amount, the actuator's soft cores are compressed leading to a shorter actuator length as illustrated in Fig. 4(d).

To model the above soft actuator deformations, the following two methods can be applied: 1) Model the entire actuator via flexible segments. For simplicity, the actuator can be assumed to only sustain uniform deformation with material density ρ and Young's module E. FEM analysis can be performed on the actuators to simulate the deformations for different force conditions [14]. 2) Use lumped parameter method [25] where rigid components are applied to build the shape of the cores and the disks while using spring and damper joints for modeling the connections between the cores and disks and enabling the target bending and compression actuator motions. These joints use equivalent stiffness and damping to provide equal response of the entire actuator compared with the flexible segments [26]. Comparing both options, the former one requires large computation efforts for the FEM analysis. The latter option avoids the force analysis on the cores and disks, but only needs to estimate the rigid connecting joint motions with stiffness k and damping d properties in the bending and longitude directions, denoted as k_B, d_B, k_L and d_L respectively. The latter model approximation method increases the computation efficiency but can also introduce modeling error. For the needs of this study, both modeling options were tested and are compared in the sequel with respect to time efficiency and modeling accuracy.

To build the above models, the first step is to acquire the mechanical properties of the flexible material or the equivalent joint characteristics for the actuator. In this study, the NinjaFlex filaments from NinjaTek Inc. are used for 3D printing the actuators. To acquire the static and dynamic properties of the material, static load tests and vibration tests are conducted using an actuator prototype composed of 10 sectional

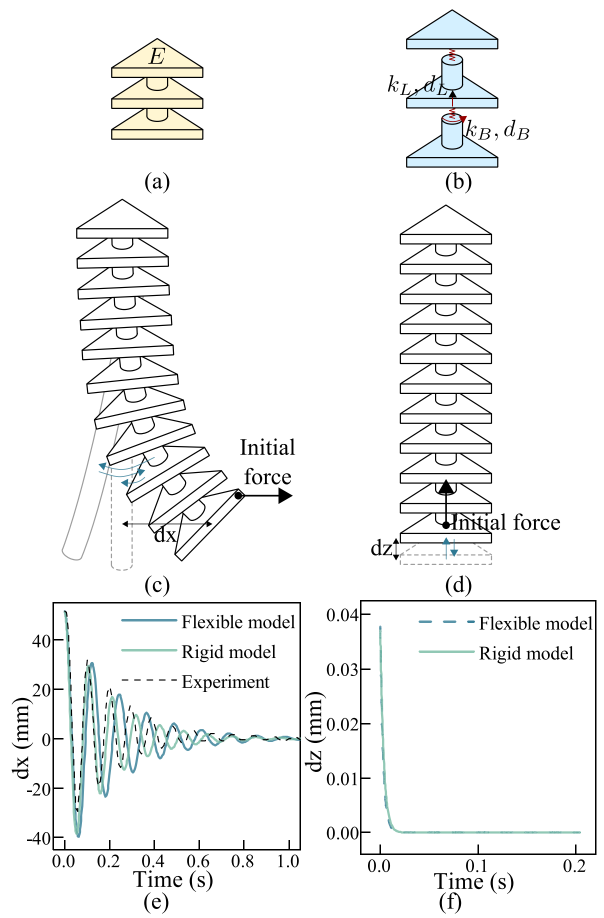

Fig. 5. Continuum actuator modeling approaches (a)–(b) and the model parameters characterization (c)–(f): (a) Model using uniform deformation model via the Young's module E. (b) Model by discretizing the actuator with multiple rigid segments interconnected via spring–damper joints. (c) Static load tests and vibration response tests in the horizontal axis for acquiring the Young's module of model (a) and the bending spring–damper coefficients in model (b). (d) Static load test in vertical axis for acquiring the longitude spring–damper coefficients of model (b). (e) Comparison of the actuator's vibration response in the horizontal axis between the experimental data and the two models. (f) Comparison of the actuator's vibration response in the longitude axis between the two models.

cores of length 8 mm, which connect 11 evenly distributed support disks with thickness of 4 mm as shown in Fig. 5(c). It is noted that the utilized prototype has longer bendable length than the leg for the quadruped robot illustrated in Fig. 4(b)–(d). The longer length enables larger bending distance to be measured during the tests and reducing the measurement errors. The actuator is placed vertically with its upper end fixed. For the static loading test, different initial forces of 1.5 N, 2 N, 2.5 N and 3 N are applied horizontally at the freely suspended end and the static horizontal deformation dx is recorded. For the vibration test, an initial and manually applied force is removed and the displacement response dx of the actuator's end point is recorded with respect to time.

The two model approaches described in Fig. 5(a) and (b) are then created in the MATLAB Simscape Multibody environment, for the same continuum structure utilized in the tests, using (1) the flexible beam block (noted as the flex-model) and (2) the rigid components with telescope joints (noted as the rigid-model), respectively. The same boundary conditions and external forces as in the real tests are applied in the two models and the end tip responses are simulated for each case. In the static loading simulation, the Young's module value E in the flex-model and the bending stiffness k_B in the rigid-model are tuned until the displacements dx are as closely fitted as possible to the experimental results. Then, for the vibration test, the damping

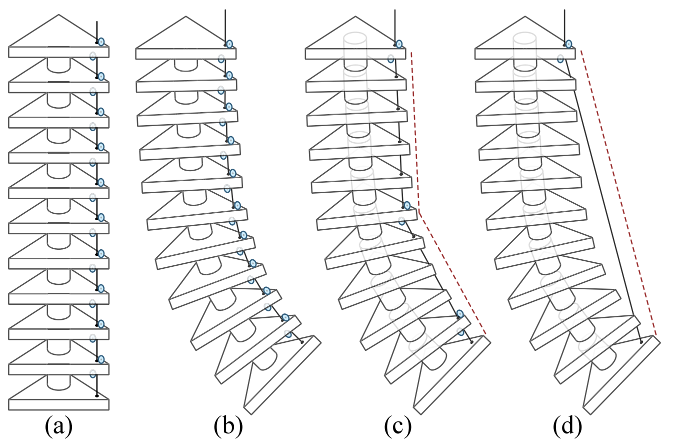

Fig. 6. Different model representations for the actuation tendon management. (a) Inclusion of paired pulleys placed at the guiding holes to regulate the positions of the tendons. (b) Demonstration of bent actuator by pulling the tendon. (c) Schematic of simplified actuator model with reduced numbers of paired pulleys to 3 while having the similar deformation as (b). (d) Further simplified model with 1 pulley pair still creates similar actuator bending but introduces more model inaccuracy.

coefficient ν in the flex-model and the bending damping d_B in the rigid-model are tuned until the best oscillation curve fit of dx with respect to time are observed. The material properties acquired from this test were $E = 12$ MPa and $\nu = 80$, while the equivalent bending properties were estimated to $k_B = 0.0005$ N m/deg and $d_B = 0.00002$ N m/(deg/s). Fig. 5(c) shows the comparison of the recorded vibration responses in both the simulated tests and the experimental trials.

The equivalent longitude properties in the rigid-model are acquired by applying a vertical compression force for the static loading test and for the vibration response tests as shown in Fig. 5(c). As the vertical displacements during the compression of the actuator are small and challenging to measure experimentally, the flex-model with $E = 12$ MPa and $\nu = 80$ is used as the ground truth. The displacement and the vibration plots of the flex-model are used as the reference to calibrate the rigid component model. The similar processes as acquiring the bending properties are then performed to get the longitude stiffness and damping, estimated at $k_L = 15000$ N/m and $d_L = 800$ N/(m/s), where good fitness is achieved in the vertical displacement between the two models as seen in Fig. 5(d).

2.3. Modeling of the force transmission via the spool and tendon

The torque from the servo motor is transmitted to the spool, and then converted to the traction force in the tendon as explained in Fig. 2. The tendons pass through the guide holes in the rigid disks, which allow tendon motion while holding them in place. The tendons can be simulated using the cable functions in most simulations tools. However, limited options are available to simulate the contact and constraints between the tendons and the guide holes. In our study, we propose the method of using two pulleys placed at the two sides of the holes (a paired pulleys) to achieve the same function as illustrated in Fig. 6(a). When the tendon moves, all the pulleys on the disks move accordingly and the actuator bends as seen in Fig. 6(b).

However, the contacts between the tendons, pulleys and disks are complex to analyze in numerical environments. With the increased number of pulleys, the simulation time can also be increased tremendously, which makes the simulation model unsuitable for RL training because it requires fast response from the simulation environment to ensure time efficiency. Thus, simplified models acting as a trade off between computational efficiency and modeling accuracy are also proposed in this study. Fig. 6(c) and (d) show the simplification examples of representing the actuator with fewer pulleys along the longitude

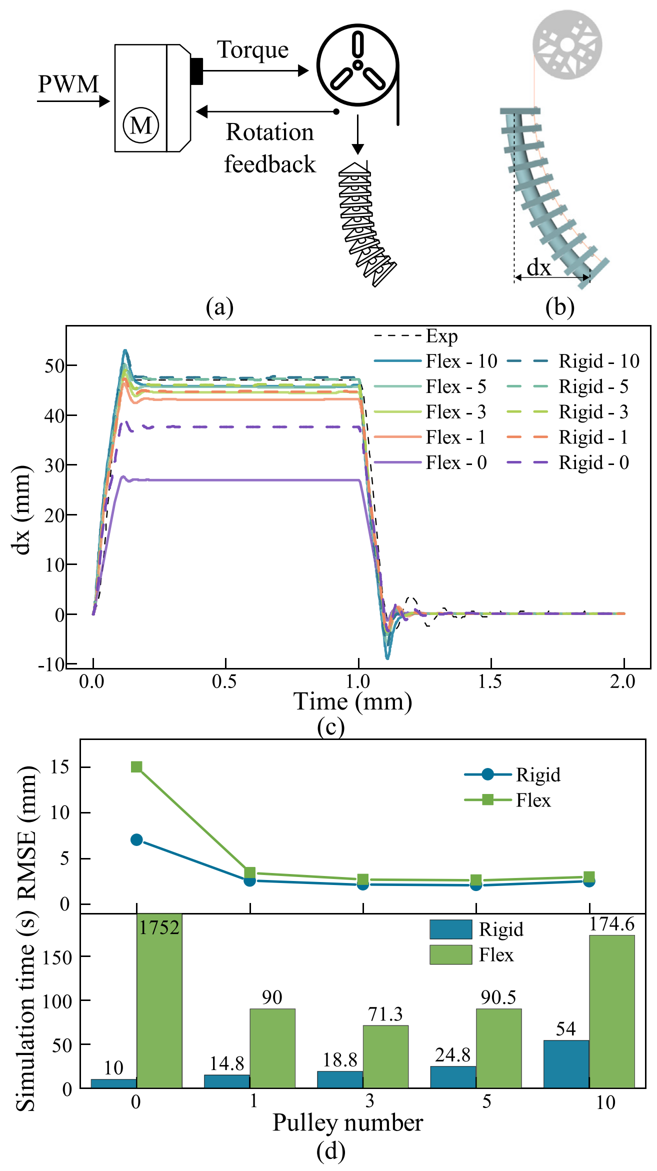

Fig. 7. Actuation system modeling accuracy and computational cost. (a) Actuation system overview. (b) Graphical model designed in the MATLAB Simscape environment. (c) Comparison of different model results and experimental recordings in terms of the horizontal displacement of the actuator versus time. (d) Comparison of the model fitness and the computation time cost for different models.

of the actuator. The absence of pulleys allows the tendon to slide freely out of the disks, thus decreasing the model accuracy, while producing models with much fewer internal forces and thus increased computational efficiency.

The soft actuator can be modeled by either the flex-model or the rigid-model. The number of the pairs of pulleys can also vary. We need to find out the best configuration of the model to balance the computation time and model accuracy. To this end, different combinations of the modeling methods are built in MATLAB Simscape and compared to verify which of the models are most suitable for RL training. Experimental recordings are used as the ground truth for examine the model accuracy. Fig. 7(a) shows the structure of the model built in Simscape, while for the presented test a single tendon was used. The servo motor model built in previous sections is connected with a spool, which then converts the rotational torque to the force in the tendon. The tendon passes along the actuator and the actuator bends when the tendon pulls. The bending of the actuator poses a reaction force to balance the tensile force in the tendon. The rotational angle of the spool is fed back to the servo motor and the servo motor reaches the rotational angle following the PWM input references. The rendered model by Simscape is shown in Fig. 7(b) and the horizontal distance between the tip of the actuator and the neutral position dx is used to express the shape of the actuator.

For driving the motor, a square PWM signal is sent to the servo motor to rotate the spool to $\pi/4$ at $t = 0$ s and then return to its neutral position to 0 at $t = 1$ s. The response of dx are recorded experimentally and in simulations as shown in Fig. 7(c). Different numbers of pulley pairs, including 10, 5, 3, 1, and 0, are used for both the flex and the rigid models. The Root Mean Square Errors (RMSE) between the different simulation results and the experimental data are calculated to represent the fitness of different models as seen in the upper figure in Fig. 7(d). Overall, the flex-models have larger RMSE than the rigid models. When the number of pulleys is decreased, the reached steady angle is also decreasing, which leads to a decrease in model accuracy. For models with pulley pair numbers larger than 1, the RMSEs are around 3 and the variation is not large. On the other hand and as shown in the lower part of Fig. 7(d), decreasing the number of pulley pairs also leads to a decrease in computation time, as recorded on a PC with an Intel Core i7-8700 CPU.

Overall, the simulation time of the flex-models is more than three times longer than that of the rigid models and in most cases larger computation time is required for models with more pulleys. However, this trend is reversed for the flex-models with less than 3 pulley pairs, where the absence of pulleys leads to a large increase in computational load. In this study, the simulation efficiency is considered a priority for better supporting the RL learning process. Thus, the simplest rigid-model with no pulleys is selected for later studies. Nevertheless, to compensate for the model difference inserted by omitting the pulleys, a correction coefficient of α_c is introduced. The correction coefficient is derived using the ratio between the steady displacement values of the experiment and the model during actuator bending as shown in Fig. 7(c). For example, in simulation, when the reference is to rotate the servo motor to θ, the actual reference of $\theta \times \alpha_c$ will be used to bend the actuator closer to the experimental results, and thus decrease the modeling error to acceptable levels.

2.4. Contact model between robot and ground

The contact between the feet of the quadruped robot and the ground is represented using the static and dynamic friction coefficients f_s and f_d. For the needs of this study the two coefficients are experimentally acquired, where the robot was placed on the ground and the horizontal traction forces that enable sliding on the ground and motion at a constant speed were measured respectively using a spring balance. The two forces were then utilized to calculate the two friction coefficients, which were estimated at $f_s = 0.315$ and $f_d = 0.3$.

2.5. Modeling in simulink

With all the above subsystems modeled, the quadruped robot system is assembled and simulated in the MATLAB Simscape environment. Fig. 8(a) shows the overall structure of the robot model. Each soft continuum actuator is driven by three motors and the connected tendons. The total number of motors for driving the soft quadruped robot is twelve. Twelve PWM signal commands are sent as the positional references to the servo motors. Each of the actuators has a spherical end playing the role of the foot, which then contacts the ground via the aforementioned contact model. The correction coefficient of the actuator bending model is tuned to $\alpha_c = 1.16$ to account for any modeling errors. Furthermore, as the continuum actuators utilized in the robot include the thigh segment and without the pulleys fixing the tendon positions, larger model errors will be introduced due to the simulated detachment of tendons from the thighs. To overcome

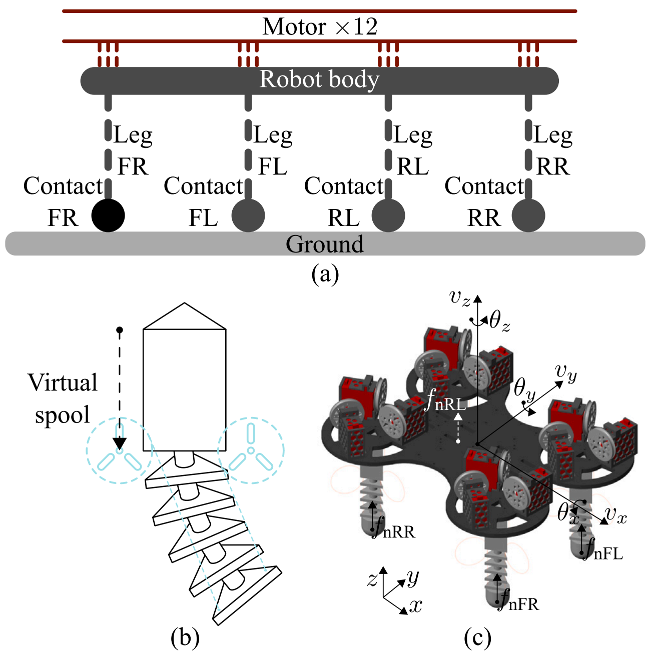

Fig. 8. Quadruped robot model. (a) Overall structure of the simulation model. (b) Shifted spool position in the robot leg to reduce model error. (c) Rendered robot with key state notations.

the issue and as illustrated in Fig. 8(b), the spools at the servo motor shafts are shifted to the bottom of the thigh, such that they prevent the tendons from exiting the thigh structure and decrease the model error introduced due to the absence of pulley pairs.

Fig. 8(c) shows the rendered quadruped robot. The local coordinate system of the robot is defined using the right hand rule, with the front direction as the x direction and the anti-gravity direction as the z direction. Ten key parameters are drawn from the simulation model to represent the robot states. The roll, pitch and yaw movements are represented using the rotational angles of the body center in the three axes x, y and z, noted as θ_x, θ_y and θ_z. The translational moving velocities are also represented by the movement of the body center in the three directions and are noted as v_x, v_y and v_z respectively. The normal reaction forces from the ground to the robot feet are noted as f_{nFR}, f_{nFL}, f_{nRL} and f_{nRR} respectively with subscripts referring to the Front Right (FR), Front Left (FL), Rear Left (RL) and Rear Right (RR) legs. The ten state variables are normalized by dividing the estimated maximal values of each state to limit the output values from the simulation environment between 0 to 1.

The update period of the RL algorithm is $T_s = 0.05$ s and the simulation model of the soft robot is a continuous-time model. To prevent the robot from falling down or entering unstable states in which the simulation results would be unreliable, termination boundaries are set in the simulation environment. When the robot states meet these termination conditions, the simulation ends before the desired simulation time. The roll and pitch angles of the body center θ_x and θ_y, are set to rotate less than 0.1 rad, while the bending angle α_b of all the four legs are set to less than $\pi/3$.

3. Optimal gait control by deep reinforcement learning

3.1. Soft Actor–Critic (SAC) method

To find the optimal gait controller by reinforcement learning, the Simscape soft robot model is used as the environment model for the RL agent. The Simscape model is considered as a Markov Decision Process $(\mathcal{S}, \mathcal{A}, p, r)$, where $\mathcal{S}$ and $\mathcal{A}$ are continuous state and action spaces. The transition function $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, +\infty)$ represents the probability density that an agent at state $\mathbf{s}_t \in \mathcal{S}$ executes an action $\mathbf{a}_t \in \mathcal{A}$ and updates to a new state $\mathbf{s}_{t+1} \in \mathcal{S}$. The agent receives an immediate reward $r_t := r(\mathbf{s}_t, \mathbf{a}_t)$, and $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ by taking an action given the current state. A non-deterministic control policy Π gives the conditional probability density $\Pi(\mathbf{a}_t \mid \mathbf{s}_t)$ of taking action $\mathbf{a}_t$ in the state $\mathbf{s}_t$.

The value of a policy is the expected value of future reward when starting in state $\mathbf{s}_t$ and applying policy Π in the infinite time horizon, which is defined as $\mathbb{E}_\Pi\left[\sum_{i=t}^{+\infty} \gamma^{i-t} r_i \mid \mathbf{s}_t\right]$, where γ is the discount factor, and $\mathbb{E}_\Pi[\cdot] = \mathbb{E}_{(\mathbf{s}_{i\geq t}, \mathbf{a}_{i\geq t})\sim\Pi}[\cdot]$ is the expected accumulated reward given that the robot MDP follows the policy [27]. This study applies the entropy-augmented objective by leveraging the maximum entropy RL for better environment exploration [21,28], where a policy entropy term $\mathcal{H}$ is added in the objective function. The entropy term $\mathcal{H}$ is calculated from the policy distribution according to

$$\mathcal{H}(\Pi(\cdot \mid \mathbf{s}_t)) = \mathbb{E}_\Pi\left[-\ln \Pi(\cdot \mid \mathbf{s}_t)\right], \tag{1}$$

where $\Pi(\cdot \mid \mathbf{s}_t)$ represents the conditional probability density function $\Pr : \mathcal{A} \to [0, \infty)$ at state $\mathbf{s}_t \in \mathcal{S}$.

The policy entropy is a measure to estimate the randomness of the actions that an agent can take at the given state. A higher entropy value indicates a policy with higher uncertainty. By adding the entropy value in the objective function, the exploration during the training gets encouraged, which mitigates the risk of converging on a locally optimal policy. Accordingly, the value of a state by following the policy Π is regarded as the entropy-constrained expectation of the accumulated reward, which is denoted as

$$V_\Pi(\mathbf{s}_t) = \mathbb{E}_\Pi\left[\sum_{i=t}^{+\infty} \gamma^{i-t}\Big(r(\mathbf{s}_i, \mathbf{a}_i) - \alpha \ln \Pi(\cdot \mid \mathbf{s}_t)\Big)\right], \tag{2}$$

where α is the entropy regularization coefficient that explicitly controls the trade-off between exploration and exploitation. We refer to it as the *temperature* [29] in the sequel. It determines the relative importance between the entropy term against the reward, and a higher α corresponds to higher exploration. The SAC learning method uses the Q function for evaluating a control policy. The Q function is defined as

$$Q_\Pi(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_\Pi\left[V_\Pi(\mathbf{s}_{t+1})\right], \tag{3}$$

which calculates the expected reward for taking action $\mathbf{a}_t$ in state $\mathbf{s}_t$ and then following the policy Π. Then, V_Π can be expressed by Q_Π with:

$$V_\Pi(\mathbf{s}_t) = \mathbb{E}_\Pi[Q_\Pi(\mathbf{s}_t, \mathbf{a}_t) - \alpha \ln \Pi(\cdot \mid \mathbf{s}_t)]. \tag{4}$$

Eqs. (3)–(4) build the connection between V_Π and Q_Π. Our target is to find a stochastic policy with maximal expected sum of reward and entropy. From the Bellman optimality principle, the RL problem can be defined to find the optimal policy Π^* with the form of:

$$\Pi^* = \arg\max_\Pi V_\Pi(\mathbf{s}_t), \tag{5}$$

which maximizes the expected future reward with entropy and gives the optimal value at state $\mathbf{s}_t$.

This study realizes the controller through the SAC learning method. The optimal maximum entropy policy Π^* in (5) is learned by soft policy iteration, and its convergence and optimality have been verified in [21,28]. Function approximation is applied to model both the soft Q-function critic and the policy actor with neural networks. The actor network with the parameter $\boldsymbol{\phi}$ characterizes the policy function Π, which is modeled as Gaussian mixtures with mean and variance through the neural network. To reduce overestimations in the critic network, two Q-functions are applied for the state–action evaluation as the Q-value networks [30]. Each of them gives independent estimation of the value function by $Q_1(\mathbf{s}_t, \mathbf{a}_t)$ and $Q_2(\mathbf{s}_t, \mathbf{a}_t)$. In addition, two corresponding target networks, $Q_1^{\text{tar}}(\mathbf{s}_t, \mathbf{a}_t)$ and $Q_2^{\text{tar}}(\mathbf{s}_t, \mathbf{a}_t)$, which have the same structure as the Q-value networks, are generated to improve the stability of the optimization [28]. The minimum of the two target

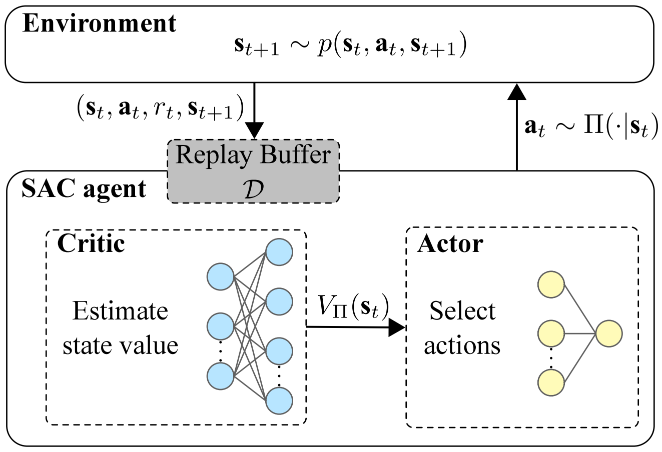
Environment
$\mathbf{s}_{t+1} \sim p(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$
$(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$
$\mathbf{a}_t \sim \Pi(\cdot|\mathbf{s}_t)$
Replay Buffer
$\mathcal{D}$
SAC agent
Critic
Estimate state value
$V_\Pi(\mathbf{s}_t)$
Actor
Select actions

Fig. 9. Schematic of the Soft Actor Critic (SAC) method.

Q functions is used for temporal difference target calculation during the update of both Q value function and policy function. During the training process, the temperature parameter α is updated by evaluating the entropy of the current policy based on past experiences. α is initialized at 1 and gradually decreases to 0 as the training episode grows. Correspondingly, the weight of exploration when evaluating $V_\Pi(\mathbf{s}_t)$ is reduced. Thus, the policy entropy $\mathcal{H}$ also decreases and stabilizes at the target entropy $\mathcal{H}'$ at the end of training.

Fig. 9 presents an overview of our approach to learning a walking gait controller for the quadruped robot. After the initialization, the actor selects an action $\mathbf{a}_t$ in each training step based on the observation $\mathbf{s}_t$ by following the policy $\Pi(\cdot \mid \mathbf{s}_t)$. The action is executed, and the robot interacts with the environment, which results in a reward r_t and the next observation $\mathbf{s}_{t+1}$. Then, the experience $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$ is stored in the replay buffer $\mathcal{D}$. The samples in $\mathcal{D}$ will be used to train and update the parameters in the neural networks. We denote each update time of the neural networks as a training sample.

The agent periodically updates the parameter from the Q-value network to the corresponding target network by a smoothing factor. The parameters in the Q-value networks, the policy actor network and the temperature are updated by minimizing the residual across batch samples in the replay buffer $\mathcal{D}$ through stochastic gradient descent. Note that the actor–critic connection is represented in the update of the actor policy network, where its update utilizes the minimum of soft state value from target Q networks based on (4). The parameters are updated towards the descent directions with the learning rate λ. The agent will constantly repeat the iterations of interacting and learning until the controller converges to an optimal policy or the maximum number of episodes is reached. We leave out the mathematical derivations for simplicity, and the details can be found in [21,28].

3.2. *Definitions of the reward*

Three critical factors that need to be identified in the RL approach, namely state, action, and reward, are elaborated in the following.

State space: We define the state space by the available sensor measurements from the robot. As shown in Fig. 8(b), it includes the moving velocities $\boldsymbol{v}(t) = [v_x(t), v_y(t), v_z(t)] \in \mathbb{R}^3$ along x, y and z axis; rotational angle $\boldsymbol{\theta}(t) = [\theta_x(t), \theta_y(t), \theta_z(t)] \in \mathbb{R}^3$ with regards to roll, pitch, yaw dimensions; and the normalized contact force $\boldsymbol{f}_n(t) = [f_{nFL}(t), f_{nFR}(t), f_{nRL}(t), f_{nRR}(t)] \in \mathbb{R}^4$ between each foot and ground. The above 10 dimensional measurements provide the environment observations that can be used to infer the robot state. Details of the sensor measurement can be found in Section 4.1. In addition, owing to the computation and communication delays in the physical world, the received observation may not be the latest signals in a real-time implementation. This asynchrony between observation and control induced by the hardware latency contradicts the fundamental assumption of the Markov Decision Process, which can lead to significant performance deterioration in robot locomotion tasks [31]. For this reason, we incorporate the action taken from the last time step $\mathbf{a}_{t-1} \in \mathbb{R}^{12}$ into the state space. Thus, the state vector $\mathbf{s}_t \in \mathcal{S}$ can be presented as

$$\mathbf{s}_t = [\boldsymbol{v}(t), \boldsymbol{\theta}(t), \boldsymbol{f}_n(t), \mathbf{a}_{t-1}] \in \mathbb{R}^{22}. \tag{6}$$

Action space: As mentioned in Sections 2.1–2.3, the movement of each robot leg is jointly driven by three servo motors. Therefore, the action is represented as the reference rotational angle of the motors, ranging continuously from $-\bar{\alpha}_M$ to $\bar{\alpha}_M$, where $\bar{\alpha}_M = \pi/6$ as defined in Section 2.1. For faster gradient descent, we normalize the actor output between 0 and 1. Thus, the action space is defined as a continuous 12-dimensional space, and a single action $\mathbf{a}_t$ is a vector containing 12 desired motor positions after standardization, i.e., $\mathbf{a}_t \in [0, 1]^{12}$. Subsequently, the policy network output is converted by the following action–angle mapping

$$\theta_{\text{ref}} = 2\bar{\alpha}_M(\mathbf{a}_t - 0.5). \tag{7}$$

Reward function: Our objective is to encourage the robot to walk straight for a longer period while keeping a steady gait; hence, the reward function is defined as

$$r(\mathbf{s}_t, \mathbf{a}_t) = \epsilon_1 + (1 - \epsilon_2|v_x(t) - v_{\text{ref}}|) - \epsilon_3\theta_z(t)^2 - \epsilon_4 v_y(t)^2 - \epsilon_5 \parallel \ddot{\mathbf{a}}_t \parallel, \tag{8}$$

where $\| \cdot \|$ denotes the Euclidean norm. In each episode, the control steps continue until the robot falls. The criteria of failing are identical to that of simulation termination as previously defined in Section 2.5. At each training step, the robot receives a constant reward ϵ_1 for staying balanced without triggering terminal conditions. Instead of imposing a sufficient punishment at an undesired state when the episode terminates, we give the robot a reward each step for keeping alive, where the accumulated rewards of this term are proportional to the sequence length. By this means, the agent is encouraged to stand for an extended period.

The robot is desired to walk in close proximity to a reference speed v_{ref} along the x-axis, and we regulate its forward speed $v_x(t)$ with a shifted absolute value function with the shape parameter ϵ_2. The essential idea is to encourage the robot to keep moving forward with a positive speed, while a backward movement is not favored. Therefore, the choice of ϵ_2 is subject to the reference speed v_{ref}. When v_{ref} becomes smaller, a larger magnitude of ϵ_2 is desired to give a sharper reward decay near v_{ref}, thus the positive rewards given under a negative velocity can be avoided. For simplicity, we assign $\epsilon_2 = -1/v_{\text{ref}}$, so that a unit reward is the maximum and the negative speed gets punished.

In order to prompt the robot to walk in a stable pose, we penalize the angle deviation on z-axis and the movement along y-axis by defining negative rewards with $-\epsilon_3\theta_z(t)^2$ and $-\epsilon_4 v_y(t)^2$, respectively. In addition, we give punishment to the agent on the large motor acceleration $\ddot{\mathbf{a}}_t$ quantified by ϵ_5, with the intent to deliver a stable actuation signal to the motor. The value of punishment is estimated via the finite differences of the actions in the last three steps with

$$\ddot{\mathbf{a}}_t = \frac{\mathbf{a}_t + \mathbf{a}_{t-2} - 2\mathbf{a}_{t-1}}{T_s^2}, \tag{9}$$

where T_s is the sampling time of the agent interacting with the simulation environment. Note that $\mathbf{a}_{t-1}$ and $\mathbf{a}_{t-2}$ are not included in the state space at the time step t, but is recorded in the robot's memory. The weighting factor $\boldsymbol{\epsilon} = [\epsilon_1, \epsilon_2, \epsilon_3, \epsilon_4, \epsilon_5]$ defines the emphasis on each aspect of the reward function, and they are set to $[0.3, -1/v_{\text{ref}}, 10, 1, 0.01]$.

3.3. Training setup

As mentioned before, two Q-value critic networks, together with their corresponding target critic networks, are generated and trained to estimate the Q function defined by Eq. (4). They have an identical network architecture, where the network input is composed of two parts: observation and action. We process the observation part with two fully-connected layers of 128 units and the action part with one fully-connected layer of 128 units. The outputs from two parts are concatenated and sent to another fully-connected layer with 32 units, while using ReLU (Rectified Linear Unit) as the activation function. The output of the Q network is the estimation of the Q function. The actor network has the environment observation as input, which is followed by two fully-connected layers of 256 units with ReLU as the activation function, and it handles the prediction of the mean and variance for each action by Gaussian distributions. The action generated from the network is bounded to $[0, 1]$ through the squashing function tanh (hyperbolic tangent function).

To avoid over-fitting, we set the learning rate λ of the critic network and actor network to be 0.002 and 0.001, respectively. The magnitude of the learning rate for the temperature α should be consistent with the network learning rate, and it is set to 0.001. We reduce the update frequency of the policy network for the stable update purpose, and the parameters are updated every 3 steps in the simulation. The discount factor γ is set to 0.96 as the number of complete steps in a training episode is 50. We set the target entropy based on the number of actions with $\mathcal{H}' = -12$. The batch size is set to 512, and the size of the replay buffer is 4096. We give the same environment initialization for the robot to explore, and all episodes start with the same initial state $\mathbf{s}_0$.

To accelerate the training process, we pre-determine a working behavior $A^e = [\mathbf{a}^e_{t_0}, \dots, \mathbf{a}^e_T]$, which provides the robot with an initial steady gait. In particular, we let the RL agent imitate the examples from the human knowledge by applying A^e in the first second of each episode. Samples collected from these pairs are stored in the replay buffer and used later for learning. By this means, we resort to the framework of *imitate learning* methods [32]. Even though our designed state–action trajectory cannot guarantee optimality, the agent can still benefit from emulating the guided elementary movements, leading to fast online implementation.

To model the real sensory data, additional noise is injected into all the observation signals. The added noises are independently sampled from zero-mean Gaussian distributions in a random manner. Based on the sensor calibration results, the variances of the noise applied to the velocity, angular and force signal data are set to: $\sigma^2_v = 0.002$, $\sigma^2_\theta = 0.002$ and $\sigma^2_{f_n} = 0.005$, respectively.

In order to examine the operational capacity and investigate the gait patterns of the soft robot, we applied the above RL setups to learn the walking motions with different speed references v_{ref}, where the training settings are identical in each experiment. During the training experiments, we observe that the training performance of the simulated robot is susceptible to the reference speed v_{ref}. With a higher target velocity, the agent is able to walk forward in fewer training episodes with a converged policy. On the other hand, with a smaller v_{ref} as the target, the agent is prone to pitch forward and the episode is terminated. We start the training process with various random seeds, and the training has a higher failure rate with the agent stuck in the local minimal. Hence intuitively, we regard mastering the low-speed locomotion policy as a more difficult task for the agent.

To address this issue, we formulate the problem of learning a smooth low-speed gait as the *curriculum learning problem* [33]. Owing to the good exploration capacity in the SAC algorithm, a converged agent can be retrained with the potential to adapt the policy for different tasks. Therefore, the converged policy that leads the robot to walk at a higher velocity can be reused as the initialized policy in the low-speed task. In the curriculum learning, the agent is firstly trained with a reference speed of $v_{\mathrm{ref}} = 0.35$ m/s. After the training gets converged, we gradually increase the task difficulty by decreasing the reference speed to $v_{\mathrm{ref}} = 0.2$ m/s. The agent is expected to modify the policy and conform to the low-speed objective within a limited amount of episodes. By contrast, we denote the training process with randomly initialized agents as *direct learning*. By this means, we circumvent the tough training problem by motivating the agent to refine the policy under different tasks.

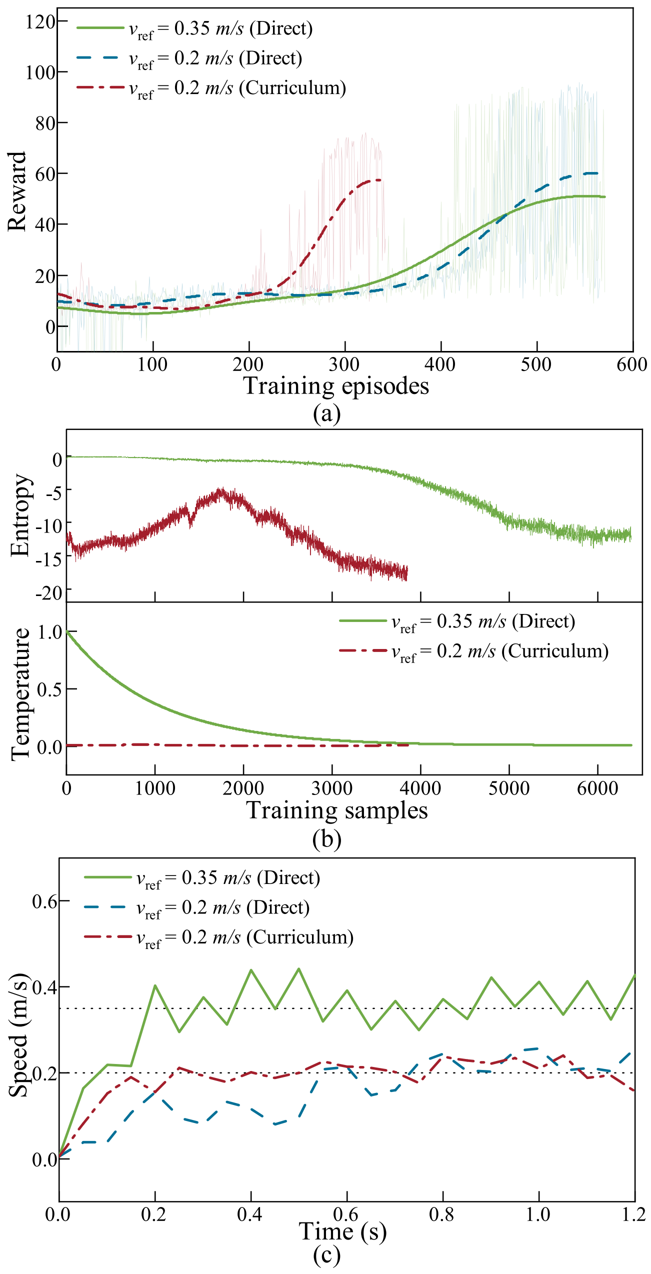

Fig. 10. Comparison of the training results with different setups. (a) Cumulative reward for different training setups versus the training episodes. (b) The variation of entropy and temperature coefficients during the training process. (c) Comparison of the walking speed of different trained control policies.

3.4. Training results

Fig. 10 shows the comparison between different training targets and setups. The bold lines in Fig. 10(a) shows the moving averaged reward with a window size of 75 collected in different episodes during the training process. For the direct training of $v_{\mathrm{ref}} = 0.35$ m/s and 0.2 m/s,

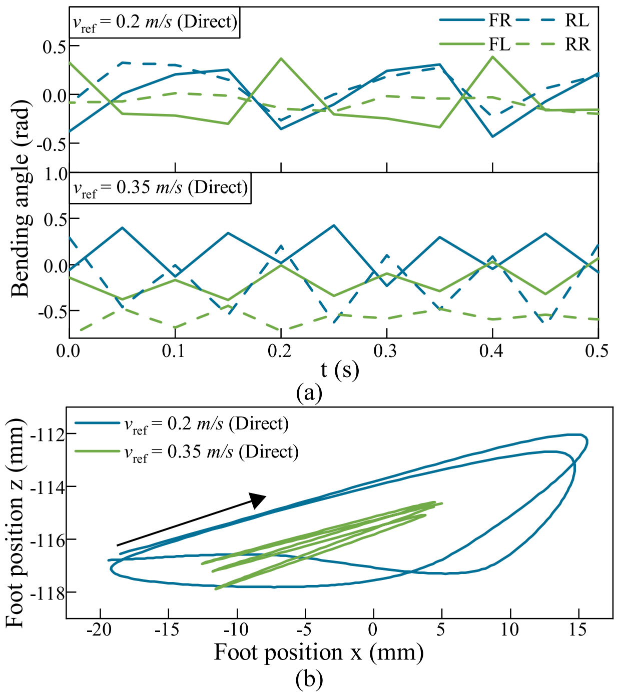

Fig. 11. Gait comparison for directly learned agent for v_{ref} = 0.2 m/s and v_{ref} = 0.35 m/s. (a) Bending angle of the four legs. (b) FR foot moving trajectories.

the averaged reward increases from around 10 and smoothly rises to around 15 before the first 300 episodes. After that, the reward starts to increase fast in the next 200 episodes and stabilizes at around 55 after 500 episodes. In curriculum learning, we use the well-trained agent for the target speed of v_{ref} = 0.35 m/s to train a lower target speed of v_{ref} = 0.2 m/s. The training starts with a slightly higher reward due to the use of a well-trained agent. After around 200 episodes, the average reward starts growing at a much faster rate. Compared with the direct training process, the curriculum learning shows a much shorter warming up phase [34] as compared with the other two direct training recordings.

Fig. 10(b) shows the variation of entropy and the temperature coefficient for the direct learning for v_{ref} = 0.35 m/s and the curriculum learning for v_{ref} = 0.2 m/s. Unlike the cumulative reward that is calculated per training episode, the entropy and temperature coefficient are updated multiple times at each training episode. Thus the training samples, which indicate the updated times of these two parameters, are applied as the horizontal axis as seen in Fig. 10(b). In direct learning, the actor networks with randomized weights have initial entropy of around zero and then decrease gradually to the target entropy while the training proceeds. The temperature coefficient that weighs the importance of entropy tunes automatically in the process and finally converges to around zero. In curriculum learning, the entropy starts at a lower level due to the well-trained agent for v_{ref} = 0.35 m/s. It then increases to around −5 to enable more exploration in the new training environment and cater for the new reward settings of v_{ref} = 0.2 m/s. Finally, the entropy converges to the target entropy again. The temperature coefficient also changes by following the same trend as the entropy but with a very small changing amplitude. This proves that using curriculum learning, the agents trained for different targets can be converted to other similar purposes with fewer training episodes.

To compare the performance in walking speed of different trained agents acquired in Fig. 10(a), they were tested in simulation, and the recorded forward speed is seen in Fig. 10(c). For all the three trained agents, the walking speeds can reach the target speed at around t = 0.2 s. Then the speed oscillates around the target speed. The agent for v_{ref} = 0.2 m/s trained with curriculum learning shows a faster acceleration at the start and achieves more stable speed compared with the directly learned one.

Fig. 11 shows the comparison of the gaits generated with the directly learned agents for v_{ref} = 0.2 m/s and v_{ref} = 0.35 m/s. Fig. 11(a) plots the bending angles of the four legs versus time. A periodic bending motion of 0.2 s is observed for all four legs for the lower speed while the period is halved to 0.1 s for the higher speed to achieve higher motion frequency. Fig. 11(b) shows the examples of the FR foot trajectories with respect to the top of the thigh of the corresponding actuator for the two agents. For a lower speed, when the foot starts to move forward from a backside position, as seen by the arrow marker, the leg is compressed and the foot rises up from the ground to avoid touching the ground. When the leg moves back, the compression is released, and the foot height decreases to touch the ground and generates the forward-moving force. For the directly learned agent for v_{ref} = 0.35 m/s, the foot oscillates within a smaller range with a double frequency than the low-speed agent and most of its trajectory distributes on the backside. The large leg moving frequency poses challenges for the implementation of a real robot. It is also noticed that the curriculum learned agent for v_{ref} = 0.2 m/s also shows similar high leg bending frequency, which is originated from the higher motion frequency of the agent for v_{ref} = 0.35 m/s. Thus the curriculum learned agent is also not suitable for the implementation. Hence for the latter experimental validation, we selected the directly trained agent for v_{ref} = 0.2 m/s which has relatively lower hardware requirements for real-time execution.

4. Experiments on a physical robot

4.1. Robot prototype implementation

To validate the learned agents, a prototype quadruped robot was implemented, following the same core configuration as utilized for the presented simulated studies. To enable the prototype robot's sensing capabilities, multiple sensors are used to detect the robot status as shown in Fig. 12(a). Specifically, four Time of Flight (ToF, Adafruit VL53L1X) sensors are mounted on the four sides of the robot to detect the surrounding distances for the localization of the robot, and one ToF sensor is located on the center bottom of the robot to detect the height of the robot body to the ground. The distances measured by these ToF sensors are also used to derive the robot movement velocities in different directions. The velocities derived by the front and back ToF sensors are averaged as the forwarding speed in the x axis. Similarly, the lateral speed in the y direction is calculated using the ToF sensors on the left and right sides. The speed in the z direction is derived using the sensor at the center bottom.

An Inertial Measurement Unit (IMU, Adafruit BNO055) is equipped at the center of the robot to measure its attitude in the x, y and z directions. To acquire the contact forces between the feet and ground, four Force Sensing Resistors (FSRs, Taiwan Alpha co.) are attached between the legs and the feet. The ToF sensors and the IMU send the sensing signals via I^2C communication, and the forces on the FSRs are estimated using their real-time electric resistances measured via voltage dividers. The voltage dividers output analog voltages that can be measured by analog readers.

The robot is operated via two-level control as seen in Fig. 12(c). The higher level uses a Raspberry Pi Zero 2 W, which communicates wirelessly with a PC. The PC transfers the MATLAB Simulink programs to the Raspberry Pi via external mode and acts as the Graphical User Interface (GUI) for the robot's human operator. The GUI displays the real time status and receives tasks from the operator for the robot to execute. The Raspberry Pi gathers the robot status from the five ToF sensors and the lower level controller.

The robot status is processed as real-time observations for the actor network trained using the simulation environment. The actor network outputs 12 reference actions which are then converted into PWM signal

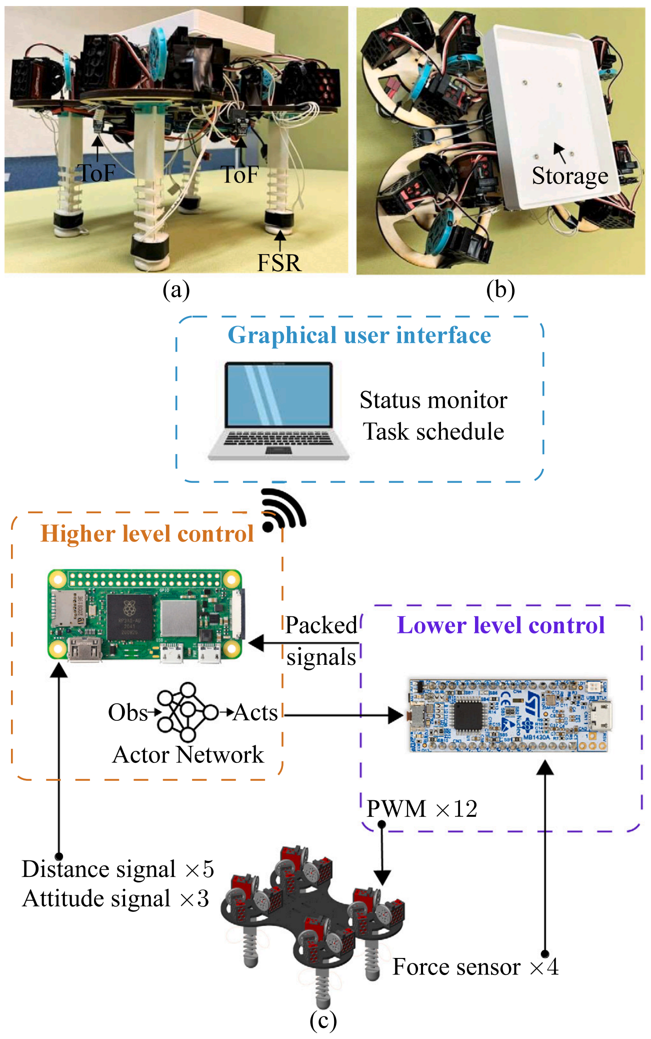

Fig. 12. Robot prototype implementation. (a)(b) Overview of the robot prototype components and integrated storage compartment. (c) Graphical representation of the high and low level control components.

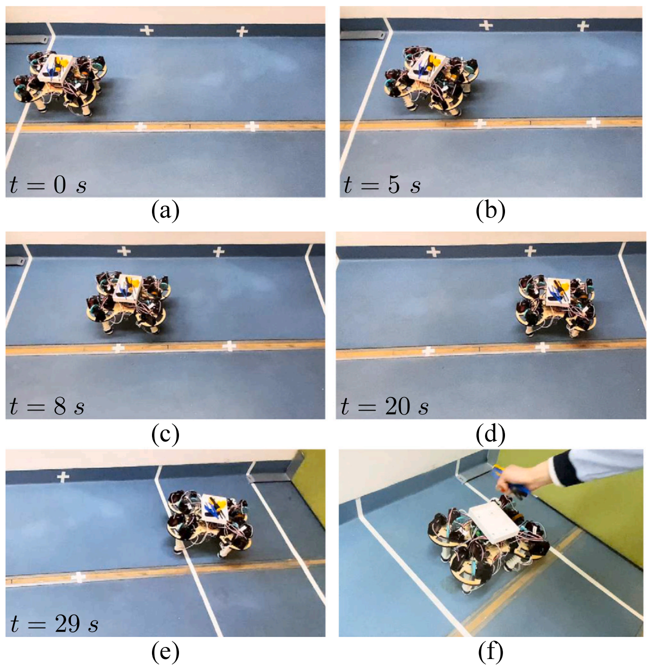

Fig. 13. Snapshots at different time stamps of the robot prototype transporting tools. (a)–(e) The robot walks forward with the trained control policy. (f) The robot stops at the terminal point, and the tools are picked up.

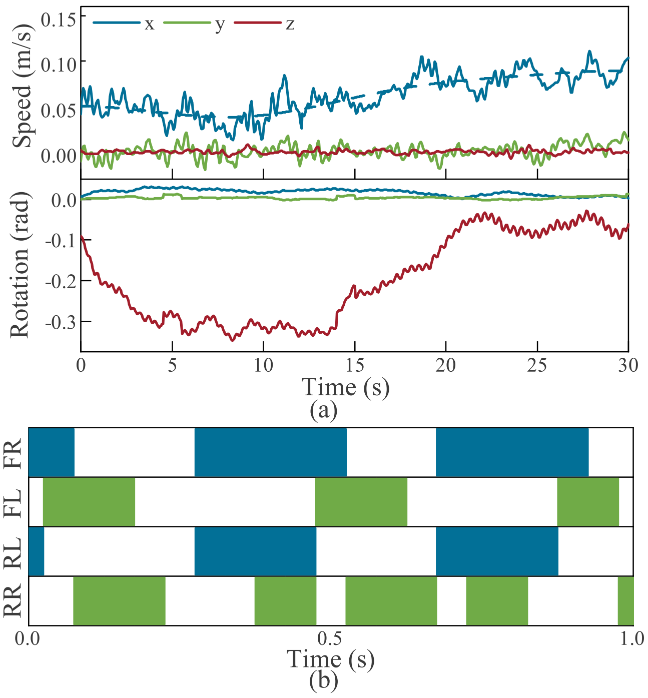

Fig. 14. Robot states during walking. (a) The key status of the robot. (b) Footfall patterns.

values to be sent to the lower level controller via serial communication. The lower level controller runs on a NUCLEO-G431KB micro-controller board (STMicroelectronics) and controls the 12 motors via PWM signals received from the higher level controller. It also reads the four analog data from the FSRs and sends them back to the Raspberry Pi. All the above controllers run at the sampling time of $T_s = 0.05$ s. The robot is fabricated using multiple weight reduction methods, and the total weight of the robot is 2 kg.

To avoid perturbations in the control signals from the trained policy due to sensor measurement noise, which is recognized as a common issue in the field of legged locomotion RL training [31], we filtered the control signals sent to the motors by decreasing $T_s = 0.05$ s to 0.1 s. This modification greatly improved the robot walking performance.

4.2. Preliminary experimental results

The directly learned walking gait for $v_{\text{ref}} = 0.2$ m/s is tested on the robot prototype. To test the robot's ability of performing transportation tasks, we put different hand tools with a total weight of 0.2 kg on the robot as the payload as seen in Fig. 13(a). Two room dividers are placed at the front and back of the robot to form the test field and reflect the light signals from the ToF sensors as well as the localization references for the robot. The task was set to walking from the initial position towards the front divider and then automatically stopping at a distance of 0.3 m to the front divider. Fig. 13 shows photographic stills taken during the walking test. To test the robustness of the learned control policy, a slight initial yaw deviation of around −0.1 rad is manually set as seen in Fig. 13(a). It is noticed that this deviation is quickly corrected automatically as shown in the following time stamps. This proves the robustness of the trained control policy. The tools are collected from the storage box placed on top of the robot when it stops at the target position as shown in Fig. 13(f).

Fig. 14(a) shows the recorded speed change in the three directions of the robot center and the altitude of the robot body during the walking. Small speed perturbations are observed in the y and z directions. The speed in the forward direction starts around 0.05 m/s during the yaw correction phase as shown by the dashed line, which plots the filtered curve of the forwarding speeds. After the yaw deviations are adjusted, the speed gradually goes up to around 0.1 m/s. This achieved speed is still lower than the target speed of 0.2 m/s in simulation due to reality gap caused by the model inaccuracies in the actuators, the sensors, and the foot-ground contacts [31,35]. On the other hand, the rotational movements are small in the roll (x) and pitch (y) directions. Large yaw (z) rotation is seen at the first 20 s and is then corrected during the walking, as explained in Fig. 13.

Due to the hysteresis of the utilized FSRs [36], the contact forces of the four feet with the ground are always larger than 0 during walking. For the visualization of the gait patterns, we define the middle value of the maximum and minimum contact forces for each foot as the threshold of sensing contact. The acquired footfall patterns are shown in Fig. 14(b) with colored areas representing the contact regions. Synchronized motions are seen in the FR and RL legs. A delay of around 0.15 s is observed between the FL and the FR or RL legs. The RR leg shows higher moving frequency compared with the other legs, most probably due to the correction action addressing the yaw rotation error.

In summary, the robot prototype successfully completes the preliminary walking test under real-time execution by utilizing the SAC trained agents. A major deficiency is the notable deviation between the actual speed and the reference speed. This can be improved by minimizing the modeling error and introducing larger environmental disturbances during the reinforcement learning process. Other improving methods include better sensors or sensor data processing algorithms, and larger networks.

5. Conclusions and future work

This work introduced a new quadruped robot design enabled by four tendon-driven soft continuum actuators. The establishment of the environment for simulating the robot movement is introduced. The robot is divided into several subsystems including the servo motor model, the soft actuator model, the tendon force transmission model and the ground contact model. The environment is built in MATLAB Simscape and is validated experimentally. Especially, to ensure the environment suitable for RL in terms of model accuracy and computation efficiency, different model and simplification approaches of the tendon driven soft actuator are used and compared. The model with least computation time consumption is applied for the gait learning using RL.

SAC methods were then applied to learn the walking gait of the quadruped robot. Stable and converged training results are achieved in around 500 training episodes for different target walking speeds. It is found that using curriculum learning can accelerate the learning efficiency but also introduce the high frequency motions originated from high walking speed control policies.

In experimental testing, the gait control policy learned using the simulation environment was implemented on a real robot prototype. The robot was applied for transporting common hand tools in factories, demonstrating its potential of being applied in industry while taking advantage of its inherent safety. However, the achieved walking speed is about half of the target speed in the simulation. This can be improved in the future by increasing the model accuracy of the motors, the actuators and use better sensors with more accurate feedback. The trained control policy shows the ability of self-correcting the walking direction, proving the robustness of the control policy.

The presented methodology and robot implementation are planned to be improved in the following aspects: 1) Parallel computing for the training process of the quadruped robot walking can be implemented to increase the time cost of learning. 2) Better and additional sensors including soft bending sensors integrated to the robot legs and visual sensors for detecting the surrounding environments will be integrated to increase the robot's perception ability, learning efficiency and moving performance. 3) Different subsystem models will be improved and larger perturbations to the simulation environment observations will be added to simulate the real-life noises. 4) Additional experimental tests with different payloads to assess the overall dynamic properties of the robot structure and the performance of the RL-based motion controllers.

CRediT authorship contribution statement

Qinglei Ji: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft, Project administration. **Shuo Fu:** Software, Validation, Formal analysis, Investigation. **Kaige Tan:** Software, Formal analysis, Investigation, Writing – original draft. **Seshagopalan Thorapalli Muralidharan:** Software, Validation, Investigation. **Karin Lagrelius:** Software, Validation, Investigation. **David Danelia:** Software, Validation. **Georgios Andrikopoulos:** Methodology, Software, Supervision, Writing – review & editing. **Xi Vincent Wang:** Supervision, Writing – review & editing. **Lihui Wang:** Supervision, Writing – review & editing. **Lei Feng:** Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The research is financially supported by Swedish Research Council (Vetenskapsrådet) with the projects: Closed-Loop 4D Printing with High Precision, number 2017-04550, Printing of soft untethered devices with re-programmability and self-maintainable shapes, number 2019-05232, KTH XPRES and KTH IRIS Area 2 Integrated mechanics, components and materials design including additive manufacturing. We acknowledge Junjun Yang, Dr. Ding Liu from Xidian University for sharing the computation resources for the RL.

References

[1] P. Biswal, P.K. Mohanty, Development of quadruped walking robots: a review, Ain Shams Eng. J. (2020).

[2] L. Wang, C. Wang, W. Du, G. Xie, K. Song, X. Wang, F. Zhao, Parameter optimization of a four-legged robot to improve motion trajectory accuracy using signal-to-noise ratio theory, Robot. Comput.-Integr. Manuf. 51 (2018) 85–96.

[3] R.F. Shepherd, F. Ilievski, W. Choi, S.A. Morin, A.A. Stokes, A.D. Mazzeo, X. Chen, M. Wang, G.M. Whitesides, Multigait soft robot, Proc. Natl. Acad. Sci. 108 (51) (2011) 20400–20403.

[4] Y. Kim, H. Yuk, R. Zhao, S.A. Chester, X. Zhao, Printing ferromagnetic domains for untethered fast-transforming soft materials, Nature 558 (7709) (2018) 274–279.

[5] Y. Alapan, A.C. Karacakol, S.N. Guzelhan, I. Isik, M. Sitti, Reprogrammable shape morphing of magnetic soft machines, Sci. Adv. 6 (38) (2020) eabc6414.

[6] Q. He, Z. Wang, Y. Wang, A. Minori, M.T. Tolley, S. Cai, Electrically controlled liquid crystal elastomer-based soft tubular actuator with multimodal actuation, Sci. Adv. 5 (10) (2019) eaax5746.

[7] M. Wang, X. Dong, W. Ba, A. Mohammad, D. Axinte, A. Norton, Design, modelling and validation of a novel extra slender continuum robot for in-situ inspection and repair in aeroengine, Robot. Comput.-Integr. Manuf. 67 (2021) 102054.

[8] C. Lee, M. Kim, Y.J. Kim, N. Hong, S. Ryu, H.J. Kim, S. Kim, Soft robot review, Int. J. Control Autom. Syst. 15 (1) (2017) 3–15.

[9] J.M. Bern, P. Banzet, R. Poranne, S. Coros, Trajectory optimization for cable-driven soft robot locomotion., in: Robotics: Science and Systems, vol. 1, 2019.

[10] S.T. Muralidharan, R. Zhu, Q. Ji, L. Feng, X.V. Wang, L. Wang, A soft quadruped robot enabled by continuum actuators, in: 2021 IEEE 17th International Conference on Automation Science and Engineering, CASE, IEEE, 2021, pp. 834–840.

[11] D. Kim, S.-H. Kim, T. Kim, B.B. Kang, M. Lee, W. Park, S. Ku, D. Kim, J. Kwon, H. Lee, et al., Review of machine learning methods in soft robotics, Plos One 16 (2) (2021) e0246102.
[12] T. Kanno, S. Ohkura, O. Azami, T. Miyazaki, T. Kawase, K. Kawashima, Model of a coil-reinforced cylindrical soft actuator, Appl. Sci. 9 (10) (2019) 2109.
[13] A.A.M. Faudzi, M.R.M. Razif, I.N.A.M. Nordin, K. Suzumori, S. Wakimoto, D. Hirooka, Development of bending soft actuator with different braided angles, in: 2012 IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM, IEEE, 2012, pp. 1093–1098.
[14] C. Armanini, C. Messer, A.T. Mathew, F. Boyer, C. Duriez, F. Renda, Soft robots modeling: a literature unwinding, 2021, arXiv preprint arXiv:2112.03645.
[15] G. Bledt, M.J. Powell, B. Katz, J. Di Carlo, P.M. Wensing, S. Kim, MIT Cheetah 3: Design and control of a robust, dynamic quadruped robot, in: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, IEEE, 2018, pp. 2245–2252.
[16] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, S. Levine, Learning to walk via deep reinforcement learning, 2018, arXiv preprint arXiv:1812.11103.
[17] H. Kimura, I. Shimoyama, H. Miura, Dynamics in the dynamic walk of a quadruped robot, Adv. Robot. 4 (3) (1989) 283–301.
[18] M.M. Gor, P.M. Pathak, A.K. Samantaray, J.-M. Yang, S. Kwak, Control oriented model-based simulation and experimental studies on a compliant legged quadruped robot, Robot. Auton. Syst. 72 (2015) 217–234.
[19] O. Cebe, C. Tiseo, G. Xin, H.-c. Lin, J. Smith, M. Mistry, Online dynamic trajectory optimization and control for a quadruped robot, in: 2021 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2021, pp. 12773–12779.
[20] F. Kirchner, Q-learning of complex behaviours on a six-legged walking machine, Robot. Auton. Syst. 25 (3–4) (1998) 253–262.
[21] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al., Soft actor-critic algorithms and applications, 2018, arXiv preprint arXiv:1812.05905.
[22] Z. Zhang, J. Chen, Z. Chen, W. Li, Asynchronous episodic deep deterministic policy gradient: Toward continuous control in computationally complex environments, IEEE Trans. Cybern. (2019).
[23] X. Rong, Y. Li, J. Ruan, B. Li, Design and simulation for a hydraulic actuated quadruped robot, J. Mech. Sci. Technol. 26 (4) (2012) 1171–1177.
[24] B. Hu, S. Shao, Z. Cao, Q. Xiao, Q. Li, C. Ma, Learning a faster locomotion gait for a quadruped robot with model-free deep reinforcement learning, in: 2019 IEEE International Conference on Robotics and Biomimetics, ROBIO, IEEE, 2019, pp. 1097–1102.
[25] Y. Wang, R. Huston, A lumped parameter method in the nonlinear analysis of flexible multibody systems, Comput. Struct. 50 (3) (1994) 421–432.
[26] J. Jung, R.S. Penning, N.J. Ferrier, M.R. Zinn, A modeling approach for continuum robotic manipulators: Effects of nonlinear internal device friction, in: 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2011, pp. 5139–5146.
[27] F.L. Lewis, D. Vrabie, K.G. Vamvoudakis, Reinforcement learning and feedback control: Using natural decision methods to design optimal adaptive controllers, IEEE Control Syst. Mag. 32 (6) (2012) 76–105.
[28] T. Haarnoja, A. Zhou, P. Abbeel, S. Levine, Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, in: International Conference on Machine Learning, PMLR, 2018, pp. 1861–1870.
[29] T. Haarnoja, V. Pong, A. Zhou, M. Dalal, P. Abbeel, S. Levine, Composable deep reinforcement learning for robotic manipulation, in: 2018 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2018, pp. 6244–6251.
[30] S. Fujimoto, H. Hoof, D. Meger, Addressing function approximation error in actor-critic methods, in: International Conference on Machine Learning, PMLR, 2018, pp. 1587–1596.
[31] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, S. Levine, How to train your robot with deep reinforcement learning: lessons we have learned, Int. J. Robot. Res. 40 (4–5) (2021) 698–721.
[32] J. Kober, J. Peters, Imitation and reinforcement learning, IEEE Robot. Autom. Mag. 17 (2) (2010) 55–62.
[33] Y. Bengio, J. Louradour, R. Collobert, J. Weston, Curriculum learning, in: Proceedings of the 26th Annual International Conference on Machine Learning, 2009, pp. 41–48.
[34] X. Zhang, H. Ma, Pretraining deep actor-critic reinforcement learning algorithms with expert demonstrations, 2018, arXiv preprint arXiv:1801.10459.
[35] Z. Ding, H. Dong, Challenges of reinforcement learning, in: Deep Reinforcement Learning, Springer, 2020, pp. 249–272.
[36] Membrane Force Sensor data sheet, http://www.taiwanalpha.com/downloads?target=products&id=298.