Adaptive Actuation of Magnetic Soft Robots Using Deep Reinforcement Learning

Jianpeng Yao, Quanliang Cao,* Yuwei Ju, Yuxuan Sun, Ruiqi Liu, Xiaotao Han,* and Liang Li

Magnetic soft robots (MSRs) have attracted growing interest due to their unique advantages in untethered actuation and excellent controllability. However, actuation strategies of these robots have long been designed out of heuristics. Herein, it is aimed to develop an intelligent method to solve the inverse problem of finding workable magnetic fields for the actuation of strip-like soft robots entirely based on deep reinforcement learning algorithms. Magnetic torques and a dissipation force to the Cosserat rod model are introduced, and the developed model to simulate the dynamics of MSRs is utilized. Meanwhile, under the reinforcement learning framework, soft robots to move forward without human guidance are successfully trained, and the results intelligently adapt to different magnetization patterns and magnetic field restrictions. The learned actuation strategies by directly applying simulated magnetic fields to real MSRs in an open loop way are validated. The experimental results show good accordance with simulations. By presenting the first case of using strategies entirely generated by reinforcement learning to control real MSRs, the potential of using reinforcement learning to achieve autonomous actuation of MSRs is demonstrated, which can be used to establish a route for the creation of highly adaptive design framework.

1. Introduction

Soft robotics has recently become a hot topic for both practice and research due to the distinct advantages over rigid robots, such as high deformability, dexterity, and robustness.[1,2] With the development of responsive soft materials, various kinds of soft robots have been invented. The relevant external stimuli include but are not limited to pressure,[3,4] heat,[5,6] light,[7,8] electric field,[9,10] and magnetic field.[11,12] Among them, magnetic soft robots (MSRs) are considered one of the most promising soft robots in biomedical applications because of their high controllability and safe, noncontact features.[13,14] Over the past 5 years, MSRs have been extensively studied in magnetization methods,[12,15–19] actuation methods,[11,20–22] material preparation,[23–25] and application development.[26–29] The related progress has been well reviewed.[30–32]

Recently, a concern has arisen around the forward-design paradigm of the locomotion of MSRs.[14,22] Researchers in this area have relied on heuristic approaches based on trial and error to design how the robots should move, which depends too much on researchers’ intuitions and inspirations. For example, MSRs with cosine magnetization and multimodal locomotion were reported in 2018.[31] Although being able to iterate verifies the robot’s flexibility, we think the process of enriching MSRs with new features is too uncertain and time consuming if it takes years even for the most talented researchers in this area.

In the area of rigid robotics, deep learning has shaped control systems to be adaptive.[33–35] Especially, deep reinforcement learning enabled robots to autonomously learn adaptive gaits[36] due to its capabilities to solve problems without explicit human guidance.[37,38] Designers abstract control objectives to a reward function, and then agents, the learner, and the decision-maker of RL will learn to maximize reward through interaction with environments. The state-of-the-art RL algorithms are able to solve real-life problems by interacting with environments, and several thrilling autonomous learning applications have been published.[39,40] However, in the area of MSRs, although researchers have used machine learning algorithms to enhance suboptimal gaits,[41] the related work focused on optimization based on manual design strategies. Even in a recent publication that utilized RL to help design parameters of MSRs,[42] the overall field pattern was still fixed (i.e., conical rotating magnetic field). The researchers only used RL to help decide parameters like amplitude and frequency rather than generate whole actuation strategies. Meanwhile, the data for analysis were collected from physical experiments in the existing studies.

J. Yao, Q. Cao, Y. Ju, Y. Sun, R. Liu, X. Han, L. Li
Wuhan National High Magnetic Field Center
Huazhong University of Science and Technology
Wuhan 430074, China
E-mail: quanliangcao@hust.edu.cn; xthan@hust.edu.cn

J. Yao, Q. Cao, Y. Ju, Y. Sun, R. Liu, X. Han, L. Li
State Key Laboratory of Advanced Electromagnetic Engineering and Technology
Huazhong University of Science and Technology
Wuhan 430074, China

The ORCID identification number(s) for the author(s) of this article can be found under https://doi.org/10.1002/aisy.202200339.

© 2022 The Authors. Advanced Intelligent Systems published by Wiley-VCH GmbH. This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

DOI: 10.1002/aisy.202200339
The aim of our paper is to demonstrate reinforcement learning algorithms can be used to generate control strategies for MSRs without any human design. We approach this problem by first training simulated MSRs and then using the trained strategies to actuate real MSRs directly. At this stage, we control real MSRs in an open-loop way, and the feedback is not introduced. In order to achieve our goal, we need to develop a simulation environment suitable for training, structure the control tasks into a framework that RL algorithms can handle, and implement RL algorithms to fulfill the tasks.

Former researchers in MSRs mainly chose an ABAQUS subroutine published by Zhao et al.\(^ {12,43}\) to simulate deformations of the robots. However, finite-element methods (FEMs) are not computationally efficient. It is suitable to conduct mechanical analysis but not a proper choice to serve as a simulation environment for RL because it usually takes up to \(10^5 \approx 10^7\) time steps to train an intelligent agent. In the meanwhile, models from the computer graphics community like discrete differential geometry\(^ {44-46}\) and Cosserat rod\(^ {47-50}\) give rise to new simulation methods for soft robots, which are quicker to compute with fair precision and provide possibilities for the implementation of modern AI techniques like RL.\(^ {51}\) However, previous models seldom consider specific external actuation, limiting their applicability with real soft robots. Our work combines magnetic torques and a novel dissipation model with the original Cosserat rod model, which made the dynamic simulations of MSRs possible, enabling the model to simulate MSRs with 2D deformation and movement.

With the computationally efficient simulation environment, the kinematics of MSRs can be easily extracted, such as position, orientation, and velocity. Using heuristic analysis based on our understanding of MSRs, we structure the control tasks into Markov decision processes (MDPs). We abstract useful information as features and input them to RL agents as state variables. We choose an elementary and straightforward reward function so that RL agents can focus on moving in a predefined direction. RL agents control the external magnetic fields for actuation, which then control the movement of the robots.

As for RL algorithms, we choose TD3, a modern deterministic model-free algorithm.\(^ {52}\) to build our RL agents. TD3 is one of the state-of-the-art off-policy actor-critic algorithms. Compared to DQN, TD3 can output continuous actions, which in our case is to output quantities involving magnetic fields. Compared to DDPG, TD3 is much more stable and has a smaller bias and variance. Compared to other advanced actor-critic algorithms like SAC, TD3 achieves similar performance and, at the same time, is much easier to implement and tune, which is why we choose TD3 over SAC. However, we believe choosing SAC or TD3 will not bring significant differences in results. During training, RL agents interact with environments in simulations and, from experiences, learn to control robots to move forward. After training, we obtain some stable control policies, which are used to generate magnetic fields. We present a few RL cases given the success of training demonstrates that the features have sufficient information for RL agents to learn complete gaits, which includes angular speed, angular position, contact strength, external fields, and so on. This part of our work is useful because there are no explicit guidelines about deciding features. Researchers can take ours as a reference when designing their own features.

We believe in the potential of using RL to control MSRs because researchers can focus on the goals robots are ordered to achieve rather than the complex mechanics of nonlinear motion. Given an objective, soft robots can learn to fulfill that without any human guidance. Besides, with the increasing complexity of application scenarios and functions of MSRs, it will be increasingly challenging to manually design the required magnetic torques acting on these robots under specific actuation requirements, which is a highly nonlinear inverse problem. In our paper, the learning results differ according to different conditions, which indicates that RL agents can adaptively learn to utilize various useful features, which is highly promising in solving the problem above. Of course, in the complex situations, the modeling methods for the dynamics of MSRs still need to be further broken through.

2. Results

2.1. Numerical Model for Predicting Dynamic Behaviors of Magnetic Soft Robots

Cosserat rod model owns the benefits of covering common types of deformation like tension, bending, shear, and torsion.\(^ {47}\) It abstracts slender structures to a centerline curve and the corresponding cross sections. Elements of an elastic rod with an unstressed length of \(l\) can be labeled with arc-length parameter \(s \in [0, l]\) and time \(t\). At each point, a vector \(r(s, t)\) can be used to describe the position under the Eulerian coordinates, and a triad of orthogonal unit vectors \(d_i(s, t)\) \((i = 1, 2, 3)\) can be used to describe the orientation of cross sections. For convenience, \(d_1\) is chosen to be perpendicular to the cross section, and \(d_2\) and \(d_3\) lie in the cross section plane. \(d_1\) points to different directions from the centerline tangent \(\partial r\) due to shear. Under numerical settings, we discretize a rod into nodes and elements. \(r(s, t)\) is of nodal quantities, used to describe the coordinates of nodes; \(d(s, t)\)
belong to elemental quantities, used to describe the orientation of elemental cross sections, as is shown in Figure 1a.

To solve the motion problem for an elastic rod with density $\rho$ and cross-section area $A(s)$, we need to utilize Newton’s dynamical laws on linear and angular momentum$^{[47,53]}$

$$\rho(s)A(s)\frac{\partial}{\partial t}r = \partial_s n(s, t) + f(s, t) \tag{1}$$

$$\frac{\partial}{\partial t}h(s, t) = \partial_s m(s, t) + \partial_s r(s, t) \times n(s, t) + \tau(s, t) \tag{2}$$

In the equations, $n(s, t)$ and $m(s, t)$ are the internal force and torque resultants. $h(s, t)$ is the angular momentum line density. $f(s, t)$ and $\tau(s, t)$ are external body force and torque line densities.

As for MSRs, they are embedded with fine magnetic particles of some specific magnetization profile. Because of the neglectable volumes and uniform distributions of the particles, when we conduct simulations, we can utilize elemental magnetization to calculate elemental magnetic torques and forces without analyzing the ones undertaken by each particle. When exposed to a magnetic field $\mathbf{B}$, an element with magnetization $\mathbf{M}$ is affected by a magnetic torque$^{[54,55]}$

$$\mathbf{r}^m = \mathbf{V}M \times \mathbf{B} \tag{3}$$

where $\mathbf{V}$ is the volume of that element. This equation implies that the element tends to be aligned toward the direction of the uniform magnetic field. This magnetic torque can be integrated into the Cosserat model as an external torque in order to simulate MSRs. In this article, we choose magnetization to be relatively static with the basis $\{\mathbf{d}^1, \mathbf{d}^2, \mathbf{d}^3\}$, as is shown in Figure 1b.

We adopt a novel dissipation model based on relative speed between adjacent nodes to mimic the high dissipation of soft materials. Intuitively, two adjacent nodes in an MSR can be thought as the ends of a damped spring. When considering damping force, the relative speed between two ends should be taken into account rather than the overall translational velocity of the spring, as is shown in Figure 1c. So, for two adjacent nodes $n_j$ and $n_{j+1}$ (corresponding arc-length parameters are $s_j$ and $s_{j+1}$) with speeds observed in Eulerian coordinates of $\nu_j$ and $\nu_{j+1}$, respectively, the damping forces undertaken are

$$f^j_j = -\nu(\nu_j - \nu_{j+1}) \tag{4}$$

$$f^j_{j+1} = -\nu(\nu_{j+1} - \nu_j) \tag{5}$$

where $\nu$ is a damping coefficient. We found that adding this damping force to our system can significantly eliminate vibration when simulated MSRs perform large-scale deformations or contact with environments and, in the meanwhile, brings only a slight influence on overall rigid body motion. In practice, when models are segmented finely, adjacent nodes $n_j$ and $n_{j+1}$ may have too close velocities. In that case, slightly magnifying the interval between the two nodes helps coarse-tune dissipation. In our work, we choose $n_j$ and $n_{j+4}$ as adjacent nodes to calculate relative speed.

We develop our model based on PyElastica$^{[47,48]}$ a numerical package for Python. The original model is for circular-section rods. In order to make the model capable of simulating 2D deformation and movement of band-like MSRs, we make some derivations and equivalent conversions. Please see Section 4: “Conversion from circular-section models to rectangular-section models” section for details. In addition, to reflect the interactions between the MSRs and the ground, we take the approach adopted

---

**Figure 1.** Numerical model of MSRs. a) Cosserat rod model abstracts a slender structure into a centerline curve and cross sections. Positional vector $r$ is used to describe coordinates of nodes under the Eulerian coordinates. A triad of orthogonal vectors $\{\mathbf{d}^1, \mathbf{d}^2, \mathbf{d}^3\}$ is used to describe the orientation of elemental cross sections. $\mathbf{d}^1$ is usually chosen to be perpendicular to the corresponding cross section and is not in the same direction with centerline tangent $\partial_s r$ due to shear. b) Elements in MSRs with magnetization $\mathbf{M}$ are driven by magnetic torques $\mathbf{r}^m$ to be aligned toward the direction of external magnetic field $\mathbf{B}$. $\mathbf{M}$ is set to be relatively static with the triad $\{\mathbf{d}^1, \mathbf{d}^2, \mathbf{d}^3\}$ so the direction of magnetization will move with the motion of elements. c) Elemental damping forces are modeled to be proportional to the difference of speed between nodes in order to eliminate vibration quickly and at the same time make as little influence on rigid body motion.
in the work reported by Gazzola et al.\cite{47} and add a force on the robots. In short, when the robots come into contact with the ground (i.e., vertical coordinates are less than 0), there would be a contact force added to the robots. The contact force can be separated into three components: A component that offsets all the vertical forces that are not from the ground; a component that is proportional to the intersection between the ground and the nodes of the robots (i.e., the absolute value of vertical coordinates); and a component that dissipates the velocity of the robots. This contact force, along with the friction, constrains the motion of the robots.

To validate our model, we conduct simulations of MSRs with sinusoidal magnetization and compare the results with the experiments\cite{11} and corresponding FEM simulations. The FEM model is based on Zhao’s subroutine\cite{12,43} and is executed under Abaqus/Standard, widely used in MSR research.\cite{19,56–58}

For both FEM and Cosserat rod models, we choose magnetization, Young’s modulus, and density of the robots to be 61.3 kA m$^{-3}$, 84.5 kPa, and 1860 kg m$^{-3}$, respectively, and these data are basically consistent with the existing reported data.\cite{11,19}

The distribution of magnetic moment density in simulated soft robots is discretized and simplified. The magnetic moment density is assumed to be the same in a particular part. There are no transition zones between two parts with different magnetic moment densities, making little difference in most conditions. The dimensions of robots in simulation models are 3.7 mm $\times$ 1.5 mm $\times$ 185 μm. From the results shown in Figure S1, Supporting Information, it is evident that FEM simulations are in better agreement with experiments, which is reasonable because FEM simulations are famous for their accuracies and are usually used for mechanical analysis; however, Cosserat rod simulations are practical enough because they correctly capture the patterns of deformation under different magnetic fields despite the usage of a simple rod structure.

### 2.2. Intelligent Design Platform for Autonomous Learning of Magnetic Soft Robots

With the simulation platform described above, information about robot motion can be easily extracted as states and then provided to RL agents for learning. According to Markov property,\cite{59} a present state must include all helpful information about the past agent-environment interaction that will make a difference for future calculations. However, in practice, too many redundant state components may lead to slower learning processes and even make neural networks diverge. Thus, correctly extracting information from simulated environments to state variables is one of the most challenging yet essential tasks when we implement RL to MSRs.

Because uniform magnetic fields interact with MSRs mainly through torques rather than translational forces, we choose quantities describing rotation as part of state components, such as angles and angular velocities. Besides, we include external magnetic fields to state components as quantities in terms of polar coordinates (angles and amplitudes) to be in better accordance with angular state components of robot motion, as is shown in Figure 2c,d. When contacting the ground, MSRs are under gravity, supporting force, and friction. To denote the effect of these forces, we use a simple contact indicator inspired by the ones in OpenAI Gym.\cite{66} For a node in contact with the ground, the corresponding contact indicator in state components has a base value of 1. Additionally, we add an extra part to describe how strongly the node interacts with the ground, which is proportional to the small deformation of the ground in the simulated environment. As for positions of MSRs, we only include the height of nodes under the Cartesian coordinates to help agents locate elements and learn moving gaits. We do not include the horizontal distance from the start point because we think animals trying to walk forward do not have to know where they are precisely.

We set our actions to be the increment of external magnetic fields on each time step to make the transition of fields smoother and degrade the effects of inductance. The field increments in two axes range from −0.3 mT to 0.3 mT. The frequency of field updates is 100 Hz in both simulations and experiments. We set our reward function to be simply proportional to the increase of the horizontal distance of the middle nodes at each time step. Although agents do not know about the exact horizontal positions of the robots, they do know whether the robot is moving forward or not through reward signals.

We use TD3 as our base RL algorithm. TD3 is a modern actor-critic algorithm that has two main parts: an actor for choosing actions according to states and a critic for evaluating state-action pairs. During training, the actor evolves to select actions that are better evaluated by the critic, and the critic evolves to conduct more accurate assessments, as shown in Figure 2a. Both the actor and the critic use networks to generalize input. For the actor network, states are input, and actions are output; for the critic network, states and actions are input, and Q, the corresponding value estimation, is output. The actor is also known as control policy because it maps states to actions. Network structures of the actor and the critic are shown in Figure 2e,f. TD3 uses target networks to prevent overestimation and improve performance.

As shown in Figure 2b, to train our RL agents, we first specify physical parameters and some settings about our training tasks and input them to our simulation platform. Then at each time-step, numerical models calculate and update the motion of MSRs. Necessary information produced by the simulation platform is stored in replay buffers, memories of RL agents, as experience transitions. The stored information of all the past interactions with environments is randomly sampled while learning to prevent overfitting. The TD3 algorithm controls learning processes that are also known as the way neural networks update their parameters. A single parameter update does not guarantee the control policy to be better, but when the times of update grow, control policies become increasingly aware of how they should behave. While training, control policies are used to generate actions, in our case, field increments, even if it is still developing. Field increments change magnetic fields of the numerical models, which then take the change into account in the next time step and continue calculating, updating, storing kinematic information, and so on. Training loops will continue until a predefined maximum time step is reached.

Deterministic RL algorithms, including TD3, have the problem of run-to-run variance.\cite{61} In other words, different
agents trained with different random seeds may have very different results, so not all agents will succeed in learning how to control robots to move. In our tests, if we start with 8 random seeds, 3–5 of them will turn out to be stable control policies. The control policies are then used to control simulated soft robots to generate magnetic fields. (Some settings of simulations for generating fields differ from the ones for training. In Section 4: “Miscellaneous training tips”, we explain why we set the differences.) The generated fields are then used to control the power supplies of Helmholtz coils with an STM32 control board, as is presented in Figure 2g. The Helmholtz coils are used to generate magnetic fields with high uniformity in case of the effects introduced by field gradient, which will add magnetic forces that we do not consider in our numerical models to real MSRs. The field signals decide the corresponding electric currents power supplies provide (See Section 4: “Magnetic actuation system”). For now, we only use two pairs of coils. i) Magnetization method of strip-like soft robots. MSRs presented in our cases are folded and put into a background field in two directions; therefore, robots with two different magnetization patterns are produced.

Figure 2. Components and procedures of the RL control system. a) Interaction between different parts of the TD3 algorithm. b) Training procedures for obtaining stable control policies. c) The simulation platform we use. In the picture, we annotate states and actions. In our work, we choose components of field increment $\Delta B$ as actions, and quantities involving necessary information of robot motion or magnetic field are defined as states. d) State variables consisting of angular orientation, angular velocity, vertical position, contact indicator, field angle, and field amplitude. Notice that state variables involving robot motion are defined element-wise or node-wise, so the fineness of segmentation will affect the total size of state variables. e) The structure of the actor network. The actor is used to map states to actions. f) The structure of the critic network. The critic is used to evaluate state-action pairs. g) Overall procedures of our RL control system. After training, we validate the stable control policies in simulations and, at the same time, obtain a set of magnetic fields. The fields are then used to control the power supply via an STM32 control board. h) Helmholtz coils. For now, we use only two pairs of coils. i) Magnetization method of strip-like soft robots. MSRs presented in our cases are folded and put into a background field in two directions; therefore, robots with two different magnetization patterns are produced.
2.3. Learning Results Under Relatively Small Magnetic Fields

In this and the following sections, we present the learning results of RL agents trained under different conditions. The MSRs we use are of two magnetization patterns, as shown in Figure 2i. We also set two different amplitude limitations on magnetic fields. In the cases of this part, maximum amplitudes are set to 4 mT, while in the following part, maximum amplitudes are set to 10 mT. We fabricate MSRs in the exactly same manner to our prior work (See Section 4: “Fabrication of MSRs”), so we adopted the same set of physical parameters in simulation.\[19\] Magnetization, Young’s modulus, and density of the simulated soft robots are set to $61.3 \, \text{kA m}^{-1}$, $84.5 \, \text{kPa}$, and $1860 \, \text{kg m}^{-3}$, respectively. The dimensions of our robots are $20 \, \text{mm} \times 8 \, \text{mm} \times 0.8 \, \text{mm}$. The static friction coefficient is set to 0.8, and the kinetic friction coefficient is set to 0.6. As for damping coefficients, we choose them through trials and errors due to the difficulties in theoretical computation and experimental material testing.

When field amplitudes are limited to 4 mT, MSRs with both magnetization patterns learn similar gaits (Figure 3a,e, Movie S1 and S2, Supporting Information). The robots move in a cyclic pattern-like crawling, which can be separated roughly into two periods. In the first period, the robots arch up with the distance between two ends shortened. In the second period, the robots stretch with the distance between two ends lengthened, as is shown in Figure 3b,f. In both cases, two ends of the robots keep in touch with the ground, and the strength of touching varies as they move. About the time when robots reach their maximum height, the contact strength of the back ends becomes obviously larger than the front ends. From the position and velocity plots (Figure S4, Supporting Information), the moving gaits of both robots are similar to “anchor push-anchor pull” locomotion strategy of inchworms.\[62\]

As is shown in Movie S1 and S2, Supporting Information, experiments and simulations are consistent, except that the real soft robots respond to fields more rapidly. This is because our damping model is not perfect, as can be explained using an example of a rigid rod. If the rigid rod rotates around its middle point, it will keep rotating in the real world if no other forces are given, but in our numerical model, the rod cannot rotate uniformly due to the damping we add. However, as is observed in experiments, in the processes of crawling, real robots reach their maximum height about 0.3 s faster than simulated robots, which is acceptable in our cases.

Since the magnetization patterns of the two robots are not the same, the learned control policies generate different magnetic fields in the two cases. We plot discrete field points in polar coordinates as is shown in Figure 3d,h. It is evident that contour shapes in the two plots are very much alike, and if we rotate the plots in Figure 3d anticlockwise about 90°, we can get a similar plot as Figure 3h, which makes absolute sense because the orientation difference of fields is the same as the orientation difference of magnetization. By rotating fields, two RL agents generate similar magnetic torques on both robots and thus control both robots to move forward in similar gaits. This is interesting because RL agents do not even know about the magnetization patterns, but they learn from interaction, and produce results meaningful in mechanics.

2.4. Learning Results Under Relatively Large Magnetic Fields

In this part, we present cases with a maximum field amplitude of 10 mT. Because when magnetic fields have larger amplitude, soft robots have much larger deflection and more various deformation patterns than under relatively small fields, the learning results show diverse moving gaits. For robots with magnetization 1, two very different gaits are learned, as is shown in Figure 4. The first gait can be separated into about 3 periods (Figure S5e, Supporting Information). In the first period, the robot arches up and, at the same time, keeps the orientation straight down. In the second period, the robot rotates clockwise, a little less than 180°. In the third period, the robot unfolds, then again, followed by arching. The second gait is simply clockwise rolling (Figure S5f, Supporting Information). The robot with gait 2 moves quicker than the robot with gait 1.

Considering that the defined reward function is simply proportional to the distance the middle points move forward, the second gait is better than the first. However, the velocity of the midpoint in the second gait is much more fluctuant than the velocity in the first gait (See Section 4: “Additional data of the cases” and Figure S5g, Supporting Information). RL agents may fit into different local optimum peaks of value estimations, which results in different control policies and moving gaits.

As for the robot with magnetization pattern 2, we find that when deflection is large, the two ends will easily get attracted to each other in experiments. This is because the magnetization polar of the two ends are opposite, leading to magnetic attraction between the ends, which has not been considered yet. We show this failure case in Figure S2 and S6, Supporting Information for completeness.

2.5. Analysis of RL Agents

Different agents choose different actions under similar states because their evaluations of actions differ. Here we plot the Q-value distribution for different actions in Figure 5.

When small magnetic fields are applied, we choose the timesteps at which robots with both magnetization patterns reach their maximum or minimum heights. When the robot with magnetization pattern 1 reaches its highest point, the best field increment estimated by the critic is that for both axes, field components get incremented by 0.3 mT, which corresponds to the actual field increment shown in field waveforms (see Figure S4e, Supporting Information). Similarly, in most cases, actual chosen actions are the ones with the highest Q values estimated by critics, except when the robot of magnetization 2 reaches its lowest point, as is shown in Figure 5b. The actor chooses the field increment to be $0.3 \, \text{mT}$ in $B_x$ and $-0.3 \, \text{mT}$ in $B_y$, while the action with the best Q-value is about $-0.15 \, \text{mT}$ in $B_x$ and $-0.3 \, \text{mT}$ in $B_y$, which indicates that the actor does not fully satisfy the critic under the particular state. In fact, actor-critic algorithms do not guarantee that actions chosen by the actor always have maximum Q values in every step, which is quite different from algorithms for discrete action spaces like DQN.\[10\] Approximating control policies using neural networks will lose some accuracy in choosing the best actions; however, thoroughly finding the best actions is unrealistic due to the vast
amount of computation for problems with continuous action spaces. And on the other hand, sometimes choosing suboptimal actions still help solve tasks, like the cases presented in our work. When large magnetic fields are applied, we pick the starting condition shared by both gaits because moving gaits are so different. Because the agent of gait 1 tries to arch the robot up while the agent of gait 2 wants to make the robot roll, the Q-value estimations are not alike, even though they are trained under same restrictions.

3. Discussion

To our knowledge, our work presents the first case of using RL to directly learn required actuation fields without any human guidance for MSRs and demonstrate the potential of RL to achieve autonomous manipulation of these robots. Specifically, we develop a computationally efficient simulation method based on Cosserat rod models for predicting dynamic behavior of MSRs. The simulations are in good agreement with experimental observations.
results for both static and dynamic cases, which provides us with an approach to open-loop control of real MSRs with the magnetic fields generated in simulations. Based on that, we introduce deep RL algorithms to solve the problem of designing magnetic field for soft robots inversely without human guidance. We extract key kinematical features of simulated MSRs as state variables input to RL agents that then learn from interaction with environments and successfully generate valid control policies and fields to control MSRs to move. We present several cases involving different magnetization and different field limitations, demonstrating the generality of our method. We believe that our work can help accelerate research in magnetic soft robotics by introducing the brand-new inverse-design approach of magnetic field.

Our work can be further improved from the following aspects. First, the single-rod model still owns limitations on generality. We plan to combine multiple rods together to formulate more complex shapes and deformation patterns or adapt other models, like discrete differential geometry models, to simulate MSRs with complex structures. Besides, the dissipation model still needs improving due to the overdamping it brings. Additionally, we think it necessary to combine more advanced deep learning structures with our system and extend the scenarios to more general tasks. Lastly, we aim to use sensors to introduce feedback by combining sensors\(^6\) or building an image capturing system\(^6\) in our experiments and formulate a closed-loop control system.

4. Experimental Section

**Experimental Design:** In this work, we chose MSRs with two magnetization patterns under two field amplitude conditions to show the generality of using RL to control MSRs. We made different robots fulfill the same task of moving forward to observe how RL agents will behave only when environments change. Relatively large-scale MSRs were chosen due to the restrictions of fabrication. For information about training processes and case selections, please see “Miscellaneous training tips” and “Additional data of the cases” sections.

![Figure 4](https://www.advancedsciencenews.com/)

*Figure 4.* Comparisons between simulations and experiments of the robot with magnetization 1 under relatively large fields. Two different gaits are learned due to run-to-run variances of TD3. The experiments show good agreements with simulations with the exception of highly dynamic conditions at the beginning of gait 2 due to overdamping on rotations. See Movie S3 and S4, Supporting Information for more.
**Figure 5.** Q-value distributions for different actions estimated by different agents. We get these heat maps by inputting uniformly distributed actions to critics under certain states. The axes are in the unit of mT. In the plots, the points annotated by “Action Chosen” are the actual field increment chosen by actors. a) Q-value distributions when robots reach their highest point. Two agents trained under different magnetization patterns both choose different actions that correspond to points with a maximum Q-value. b) Q-value distributions when robots reach their lowest point. The action selected by the agent of magnetization 2 does not have the highest Q-value. c) Q-value distributions of agents corresponding to two gaits are shown in Figure 4. We choose the specific state to be at the starting position because it is shared by both gaits.

**Robot Fabrication:** The magnetic-responsive material used in the experiment was a mixture of Ecoflex 00-10 polymer matrix (Smooth-On Inc.; density: 1.04 g cm\(^{-3}\)) and neodymium (NdFeB) magnetic powders (MQP-15-7, Magnequench, density: 7.61 g cm\(^{-3}\)) with a weight ratio of 1:1. To ensure uniform particle dispersion, the blend was first mixed at 2000 r.p.m for 1 min by the planetary mixer (AR-100, THINKY Corp), followed by defoaming at 2200 r.p.m. for 1 min. Then the thoroughly blending magnetic slurry was scraped into the prepared mold. After thermal polymer curing (room temperature, 4 h), the demolding sample was magnetized by a pulsed magnetic field (about 1.5 T) generated by the pulsed magnet for each task. The field contained x-, y-, and z-axes fields (in our cases, z-axis fields were always 0). An STM32F407 microcontroller was used to read the file, conduct related calculations, and then output continuous voltage signals through a DAC8563 module. The voltage signals were amplified by three power amplifiers (HEA-500 G, HEA-20° C, HEAS-50 for x-, y-, and z-axes fields, respectively), generating currents in 3D Helmholz coils (3HLY10-100). The field amplitudes of three axes were proportional to corresponding currents in coils. To maintain reliable and repeatable experiments, we regularly calibrated the magnetic actuation matrix inside the workspace, i.e., the mapping between the applied electric current and the generated magnetic field.

**Core Concepts of RL:** According to MDPs, the learning problem can be framed into several parts. The learner, as well as the decision-maker, was called the agent. The things the agent interacts with made up the environment. At each time step \(t\), the agent selected an action \(a\) based on the environment’s current state \(s\). The term policy \(\pi\) was used to describe the mapping between \(a\) and \(s\). After that time step, the environment sends a reward \(r\) to the agent, which was usually correlated with a goal the agent was meant to achieve and turned into a new state \(s’\).

For optimal control problems, we aimed to find the best policy \(\pi\), that can most likely get the best results. In order to achieve that, the optimal action-value function \(Q^*(s,a)\) that evaluates actions based on their expected return needed to be calculated. Return means the weighted sum of all future rewards, and in definition, \(Q^*(s,a)\) is expressed as

\[
Q^*(s,a) = \max_{\pi} \sum_{t=0}^{\infty} \gamma^t r_t(a|s) + \delta Q^*(s',a')
\]

However, the definition formula can be hard to compute. The solution to this problem is the framework of generalized policy iteration and temporal difference update. Simply speaking, generalized policy iteration means that as long as we take steps between a better action-value evaluation and a better policy improvement, we can eventually get the true optimal action-value function \(Q^*\) no matter what the start values are. On the other hand, temporal difference updates enable us to use the action-value functions of the next state \(s’\) as a reference to update the action-value functions of the current state \(s\). If the action-value function starts arbitrarily as \(Q(s,a)\), the update rule can be expressed as

\[
Q(s,a) = Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))
\]

This update rule is also known as Q-learning. In the equation, \(\max_{a'} Q(s’, a’)\) is the best possible action-value function of all the next state-action pairs, and it has to be multiplied by discount factor \(\gamma\) to ensure convergence. The iteration can be understood as a step toward the optimal action-value function \(Q^*(s,a)\), with a step size of \(\alpha\). If \(\gamma\max_{a'} Q(s',a')\) is called TD target, which points out the direction and target of iteration steps. The iterating Q-value eventually converged to real \(Q^*(s,a)\).

Temporal difference update introduces a method for updating action-value functions between time steps. This is extremely meaningful to practical RL algorithms because lots of real-life tasks are nonepisodic, without a natural terminal state to mark the end position that is indispensable for calculating a return value.

**Development of Practical Deep RL Algorithms:** The original RL framework fits naturally into problems with discrete states and actions. However, for
real-life issues, states are seldomly discrete; instead, state variables are usually continuous and get restricted by some ranges. Thus, traditional methods that use tables to store action-function values are impractical to solve problems in industries.

Deep Q-learning used a neural network as a function approximator. In other words, the neural network automatically extracted useful features from the input data for solving the control problems, and how to extract useful features (i.e., the parameters of neural networks) is learned by interacting with the environment. Thus, continuous values of input states can be generalized by the nonlinear function presented by the neural network.

In the previous discussion of Q-learning, TD target is the target of iterating action-value function, which also gives out an implication of the loss function for the neural network of deep Q-learning

$$L(\theta) = E \left[ (r + \gamma \max_{a'} Q(s', a'; \theta) - Q(s, a; \theta))^2 \right]$$  
(8)

$r + \gamma \max_{a'} Q(s', a'; \theta)$ is the TD target of deep Q-learning, $\theta$ represents the parameters of the action-value $Q$ network. The neural network uses gradient descent to update its parameters and hopes to eventually make the loss function a local minimum. In other words, in the ideal cases, the action-value function represented by the neural network will converge to be equal to the TD target. However, this naive version of deep Q-learning is unstable and can hardly to applied directly.

One of the most important advancements in modern RL is DQN[18]. The DQN paper presents the first artificial agent capable of learning to excel at a diverse array of challenging tasks. For example, the agent achieved human-level control on the Atari games, which depended on an end-to-end structure that used game images as input.

Two critical innovations of DQN greatly enhanced the stability of using neural networks to approximate action-value functions: replay buffer and separate target network. The replay buffer was used to store separated experience transitions $e_i = \{s_i, a_i, r_i, s_{i+1}\}$ observed when the agent is interacting with the environment. Then, when learning, the agent samples experienced transitions $e_i$ from the replay buffer $D_t = \{e_1, e_2, ..., e_t\}$ randomly and used them to conduct gradient descent. Thus, the observation sequence can be broken apart, and the agent was less prone to overfit the data in a specific time span. The second innovation, the delayed-update target $Q^*\text{ network, was designed to make the temporal difference target in the update rule more consistent. Therefore, the neural networks in RL agents were less prone to diverge.}$

The loss function for $Q$ network is

$$L(\theta) = E_{(s,a,r,s') \sim (s,a,r,s') \sim U(D)} \left[ (r + \gamma \max_{a'} Q(s', a'; \theta) - Q(s, a; \theta))^2 \right]$$  
(9)

where $r + \gamma \max_{a'} Q(s', a'; \theta)$ means that the TD target is no longer calculated based on the $Q$ network, as is in deep Q-learning. Instead, it uses the delayed-update target $Q^*\text{ network, whose parameters are represented as } \theta^*; \theta^*$ is not updated by gradient descent or other learning methods. It simply copies the values of $\theta$ every C steps where $C$ could be a hyperparameter to tune. The expectation was calculated and evaluated based on the experience transitions from the replay buffer, as was shown by $E_{(s,a,r,s') \sim U(D)}$. For simplicity, the following discussions omitted $E_{(s,a,r,s') \sim U(D)}$ in equations. As long as an algorithm utilizes a display buffer, it will learn from samples of experience transitions.

Modern Actor-Critic Algorithms: DDPG and TD3: DQN can solve problems with continuous state space and discrete action space; however, in robotics, the agent we aimed to control usually had a continuous span of action space. In order to solve this problem, the DDPG algorithm brought innovations from DQN to the actor-critic framework.

Actor-critic algorithms differed according to different purposes or under different prerequisites. For example, the policy we wanted to approximate may be stochastic or deterministic, which also changed other parts of the algorithms. Here we focussed on the basic structure that DDPG built upon, which utilized a network to approximate the deterministic policy $\pi(s)$, besides a network to approximate the action-value function $Q(s, a)$. The former network was called the actor network, and the latter was called the critic network. Two networks aimed to achieve different goals, but together they made the algorithm converge to an optimal policy.

Because policy $\pi$ described the mapping between actions and states, the actor network took state values as input and calculates action values as output. The goal for the actor network was to choose the best policy, as was implied by its loss function

$$L(\theta^\pi) = -E[Q(s, \pi(s; \theta^\pi); \theta^\pi)]$$  
(10)

In the equation, the minus sign is added in the beginning because the actor aims to maximize $E[Q(s, \pi(s; \theta^\pi); \theta^\pi)]$ part, and in most modern neural network frameworks, loss function and gradient descent are the default settings rather than the reward function and gradient ascent. $\theta^\pi$ refers to the parameters in the actor network, and $\theta^\xi$ refers to the parameters in the critic network. $\pi(s; \theta^\pi)$ means that the action is an output from the actor network that represents the policy $\pi$. The meaning of this equation can be quite straightforward: we updated the parameters of the actor neural network so that the actor (or policy) can choose better actions that achieved a higher action value estimated by the critic.

On the other hand, the loss function for the critic network is

$$L(\theta^\xi) = E[(r + \gamma Q(s', \pi(s'; \theta^\pi); \theta^\xi) - Q(s, a; \theta^\xi))^2]$$  
(11)

which can also be viewed as two main parts. $r + \gamma Q(s', \pi(s'; \theta^\pi); \theta^\xi)$ is the TD target, and $Q(s, a; \theta^\xi)$ is the action value estimated by the critic. The parameters of the critic network got adjusted so that the estimated action value got closer to the TD target, just as in deep Q-learning.

Besides, DDPG adopted target networks and replayed buffer to enhance the performance of the actor-critic structure. DDPG used a target $Q^*$ network along with a target $\pi$ network to calculate the TD target. Thus, Equation (11) should be changed to

$$L(\theta^\xi) = E[(r + \gamma Q(s', \pi(s'; \theta^\pi); \theta^\xi) - Q(s, a; \theta^\xi))^2]$$  
(12)

where the loss function for the actor network remains the same. In the equation, $\pi(s; \theta^\pi)$ means that the policy for calculating target action value $Q^*$ is derived from the target $\pi$ network, and $Q(s', \pi(s'; \theta^\pi); \theta^\xi)$ means that the target action value $Q^*$ in the TD target is derived from target $Q^*$ network.

The update rule for the target networks was different from the one in DQN. The weights in the target $Q^*$ network and target $\pi$ network were updated by slowly tracking the weights in the $\pi$ network and $Q$ network using exponentially weighted average. The weights should be updated slowly to prevent divergence.

TD3 further improved stability and performance by introducing three main improvements to DDPG in order to partly eliminate variance and bias brought by function approximation error.

The first improvement was target policy smoothing regularization, which aimed to reduce the variance of the TD target. Target actions were updated by exponentially weighted average. The weights should be updated slowly to prevent divergence.

TD3 further improved stability and performance by introducing three main improvements to DDPG in order to partly eliminate variance and bias brought by function approximation error.

The first improvement was target policy smoothing regularization, which aimed to reduce the variance of the TD target. Target actions were updated by exponentially weighted average.

The second improvement was a pair of critics to address overestimation bias, which was built on double Q-learning. In TD3, two separate critic networks $Q_1$ and $Q_2$, along with two target critic networks $Q^*_1$ and $Q^*_2$, were used. When updating parameters, two target values were calculated

$$y_1 = r + \gamma Q^*_1(s', \pi(s'; \theta^\pi) + \epsilon; \theta^\xi_1)$$  
(14)

$$y_2 = r + \gamma Q^*_2(s', \pi(s'; \theta^\pi) + \epsilon; \theta^\xi_2)$$  
(15)
The TD target used in the loss functions for both critic networks is
\[ y = \min(y_i, y_j) \] (16)
and the loss functions for \( Q_1 \) and \( Q_2 \) are
\[ L(\theta_1) = E[|y - Q_1(s, a; \theta_1)|^2] \] (17)
\[ L(\theta_2) = E[|y - Q_2(s, a; \theta_2)|^2] \] (18)
The third improvement is delayed updates on the actor network \( \pi \) along with target networks \( \pi' \). Similar to DDPG, the loss function for \( \pi \) is
\[ L(\theta_\pi) = -E[Q_\pi(s, \pi(s; \theta_\pi); \theta_\pi)] \] (19)
Here we choose \( Q_\pi \) as the value estimator in accordance with the original TD3 paper. As can be seen, the actor network aims to maximize the estimated action value. However, if the estimated value itself is inaccurate with high variance, the calculated policy will be more inclined to diverge. Using delayed policy updates means that we update our actor network and target networks at a lower frequency than the critic networks so that we can stabilize the estimated action value before conducting a policy update. In this way, the performance of the actor improves though with fewer updates.

**Conversion from Circular-Section Models to Rectangular-Section Models:**
Because we develop our simulation platform based on PyElastica,\(^{47,48}\) an open-source Cosserat rod package for Python, and the original model is for rods with circular sections, we have to make some conversions in order to make it valid for simulations of bar-like MSRs.

The main forces that determine deformations of MSRs consist of magnetic torques, gravity, and elastic forces. Magnetic torques and gravity activate deformations, so we call them “active forces/torques” in this section. In contrast, the elastic forces will be generated only when deformation happens, so we call them “passive forces/torques”.

The original rod model in PyElastica supports fully 3D deformations, including 6 deformation modes; however, in our work, we cared only about 2D deformation and movement of MSRs so that the deformation modes can be simplified to 3 deformation modes: bending about tangents, shearing along tangents, and stretching along normals. The modes and corresponding strains, rigidities, and loads are shown in Table S1, Supporting Information.

Imagine a circular-sectional rod with a radius of \( r \) and a rectangular-sectional rod with a width of \( w \) as well as a height of \( h \), as shown in Figure S1, Supporting Information. Take small segments with a length \( \Delta l \) from both rods. Magnetization of both parts is \( M \), and density is \( \rho \). When applied with an external magnetic field \( B \), magnetic torques undertaken by the two segments are
\[ \tau_{cir} = V^{cir}M \times B \] (20)
\[ \tau_{rec} = V^{rec}M \times B \] (21)
where \( V \) stands for volume of the segments, and superscripts indicate shapes of cross sections. The volumes are calculated as
\[ V^{rec} = \frac{wh\Delta l}{12} \] (22)
\[ V^{cir} = \frac{\pi r^2 \Delta l}{4} \] (23)
So, for the two segments, the torques undertaken were proportional to the areas of sections.

Besides, the gravities undertaken by both of the segments were also proportional to the areas of sections, for
\[ \sigma^{rec} = \frac{wh\Delta l}{\rho g} \] (24)
\[ \sigma^{cir} = \frac{\pi r^2 \Delta l}{\rho g} \] (25)
During the conversion, we aimed to keep strains equivalent in both circular and rectangular rods. Say combined active forces and torques are \( F \) and \( \tau \), respectively, the strains generated are then
\[ \Delta \sigma = \frac{F}{S} \] (26)
\[ \Delta \kappa = \frac{\tau}{B} \] (27)
where \( S \) and \( B \) are rigidities. So, when \( F \) and \( \tau \) are proportional to the areas of sections, if rigidities are proportional to the areas of cross sections as well, the strains will be the same.

Strecthing rigidity \( S_n = EA \) and \( E \) is the same for both segments, so it satisfies the requirement. As for bending rigidity \( B_t = E I_t \), we have to solve the equation
\[ \frac{wh^3}{12} / \pi^3 = \frac{wh}{4 \pi} \] (28)
where \( wh^3/12 \) is the second moment of inertia for the rectangular segment and \( wh^4 / 4 \) is the second moment of inertia for the circular segment. \( wh^3 / 4 \pi^3 \) is the ratio of two areas of cross sections. From the equation, we can get \( r = \frac{\sqrt{3}}{2} h \). Therefore, when the radius of the circular segment was equal to \( \sqrt{3}/3 \) times the height of the rectangular segment, bending rigidities of both segments were proportional to the areas of sections.

As for shear rigidity \( S_s = \alpha_G A \), if we set the constant \( \alpha \) to 1 rather than \( 4/3 \) of the original model, we can meet the requirement.

To conclude, we can use a rod model with a circular cross section to simulate the 2D motion of a bar-like MSR if we set the radius equal to \( \sqrt{3}/3 \) times the height of the robot and set the constant \( \alpha_G = 1 \).

**Miscellaneous Training Tips:**
When training, we set a maximum time in order to reset simulations regularly in case agents get stuck in some situations. In our work, we set the maximum time to be 20 s.

But notice that the moving of MSRs should not be modeled as an episodic task. According to definitions, terminal states have a state value of 0, so if the problem is modeled as an episodic task and the terminal states correspond to the states when time hits 20 s, the RL agents may get confused while learning. This is because random states that include information only about motion and fields may be marked as terminal states, and the terminal states may also appear as nonterminal states before time hits 20 s, which leads to conflicting directions when the critic updating its network parameters. We do not include time as a state component because we think specific time is not essential in the period of moving forward.

In order to model the problem as a continuous task and at the same time utilize the benefits of a maximum timestep, we skipped the experience transitions corresponding to the maximum timestep and chose not to add them to replay buffers.

Besides, in order to diversify samples stored in replay buffers and prevent RL agents from overfitting. At the beginning of each episode, we set random initial magnetic fields within the range of maximum amplitude. But noticed that due to inductance, the current in real Helmholz coils started from 0, so when running simulations for generating fields after stable control policies were obtained, we cancelled the settings of random initial fields and made the fields start from 0 in order to generate fields that can be directly loaded.

Lastly, because RL was highly commanding in computational resources, we figured out a way to shorten training periods as is discussed in “Analysis on Computational Speed”. However, this method affected the dynamic responses of our numerical model. In order to solve this problem, after we trained our RL agents for about 1 × 10^7 training steps in the inaccurate simulation environment, we refined the stable control policies in a more accurate simulation environment with regular density and gravity for about 1000 steps. And when we generated magnetic fields, the policies were run in accurate simulation environments. We used a 16-core Intel i7 for our training. A single period consisting of training, refining, and generating waveforms takes less than 24 h, while 6 independent seeds can be trained simultaneously.

For more specific parameters or hyperparameters, please see our code.

**Analysis on Computational Speed:**
In order to approximately measure the computational speed of methods, in Table S2, Supporting Information, we listed the computational time requested of conducting a simulation of the cases presented in Figure S1, Supporting Information. In the first column,
the computational time of using FEM was listed. We conducted FEM simulations in Abaqus/Standard 2019, where analysis type was set to dynamic analysis. The operating system was Windows 10, with a CPU of AMD 3600. When setting FEM models, we separated 64 segments along the length of the robots, 20 segments along their width, and 2 segments along its height. We placed a coarsely separated supporting plane under the robots as is shown in Figure S7, Supporting Information. We set the total simulation time to be $1 \text{s}$ and the maximum step time to be $0.01 \text{s}$. Magnetic fields were loaded from zero, linearly grew with time, and reached the peak at $0.8 \text{s}$.

As for Cosserat rod models, we noticed that the PyElastica numerical package had difficulties dealing with small-scale soft robots. The smaller the soft robots are, the smaller calculation timesteps have to be, which will significantly elongate total calculation time. We thought it might be due to the small moment of inertia of elements in small-scale soft robots, which led to enormous velocity increments in a single timestep. In order words, if timesteps are large, velocity growth in a single timestep may easily exceed the limits of NumPy. We found that if we increased the density of the robots while at the same time inverse-proportionally decreased gravitational acceleration, the gravity as well as deformation of robots remain the same, and timesteps can be set to larger values. In our case, when we set density to be 10.5 times larger and gravity acceleration to be 10.5 times smaller, calculation timesteps can be set from $8 \times 10^{-6} \text{s}$ to about $1 \times 10^{-5} \text{s}$, which brought about 10 times faster calculation. To distinguish between the original set of physical parameters and the approximate set of physical parameters, we referred to the former set as “accurate” and the latter set as “approximate”.

In the second and third columns of Table S2, Supporting Information, we compared the differences between accurate set of parameters and approximate set of parameters. Cosserat rod simulations were implemented in Python and packages imported included PyElastica, Numpy, and Numba. The corresponding operating system was Ubuntu 20.04, with a CPU of Intel i7. We separate 64 elements along its length. For accurate Cosserat rod models, we set the fixed timestep to be $2 \times 10^{-6} \text{s}$, while the timestep of approximate Cosserat rod models was set to be $5 \times 10^{-6} \text{s}$. In accordance with the simulations conducted using FEM, we set the total simulation time to be $1 \text{s}$, and magnetic fields were loaded in similar patterns.

As can be seen from Table S2, Supporting Information, under the accurate set of parameters, the computational speed of Cosserat rod model was slower than FEM models due to the too small timestep. However, as can be seen the computational time remained stable in spite of different conditions, while the computational speed of FEM models was greatly affected by conditions. If physical conditions were complicated, an FEM model was more prone to diverge and then required smaller timesteps and more computational time.

As long as we adopt the approximate set of parameters, the time required by Cosserat rod model can be enormously reduced. The amount of reduction on computational time was because of the larger computational timestep as was discussed earlier in this section.

**Learning Results Under Small Field Amplitude:** We ran 8 random seeds for each task, and 3–5 of them successfully generated stable control policies. For RL agents trained under small field amplitude, stable control policies showed similar gaits, so we picked the ones with the highest average scores as the cases to present. Average learning curve across the 8 seeds and the learning curve of the case we present of the robots with two magnetization patterns are shown in Figure S2a,b,f,g, Supporting Information. As can be seen, the average learning curve of magnetization 2 showed less reward than the one of magnetization 1, which may be due to randomness. The learning curves of the cases we present in our work showed similar average rewards for both robots. We extracted the lateral positions of the front, middle, and back nodes and plot them in Figure S2c, Supporting Information. As can be seen, middle nodes moved much stabler and the others, which was probably due to the fact that we set lateral positions of the middle nodes as reference for reward functions. We used the plots of positions to generate the velocity plots by differentiation, as shown in Figure S2d,i, Supporting Information. Obviously, the velocities of middle nodes were much less fluctuant than other nodes and had the least negative velocities. When robots arched up to the highest point, the back nodes moved forward while the front nodes negligibly moved backward; when robots unfold to the lowest point, the front nodes moved forward while the end nodes negligibly moved backward. The middle nodes almost always had positive velocities. The peak velocities of the front nodes were much larger than the others. The magnetic field waveforms are shown in Figure S2e,j, Supporting Information. We can see that the field amplitudes of the two cases were very similar, but $B_1$ and $B_2$ were in very different shapes.

**Learning Results Under Large Field Amplitude:** For RL agents trained under large field amplitude, stable control policies showed different gaits, so we picked two typical gaits as the cases to present in our work. The corresponding learning curves are shown in Figure S3a–c, Supporting Information. The second gait had larger average rewards than the first gait. We picked the opening orientation angle $\beta$ as a quantity to depict the phases of motion. $\beta$ is defined as the angle between the x-axis and the vector pointing from the middle node to the middle of two ends, as is shown in Figure S3d, Supporting Information. The change of $\beta$ in two gaits is plotted in Figure S3e,f, Supporting Information. It can be seen that in gait 1, the robot first arched up, then rolled for less than 180°, and then unfolded. The motion can be separated into 3 phases in total. But in gait 2, the soft robot rolled all the time. In Figure S3g, Supporting Information, the lateral velocities of the middle nodes in two gaits are plotted. It can be seen that the velocity in gait 2 was much bumpier than the one in gait 1. We plot the trajectories of the middle nodes in the x-y plane in Figure S3h,i, Supporting Information. In Figure S3j–l, Supporting Information, we plot the magnetic waveforms. It can be seen that the field in gait 2 was more like a rotating field and had a larger amplitude than the one in gait 1.

For the MSR with magnetization pattern 2, when applied with large field amplitude, two ends of the robot will stick to each other in experiments. So, we only show the learning curves and field waveforms in Figure S4, Supporting Information and do not conduct any further analysis for this failure case.

**Supporting Information**

Supporting Information is available from the Wiley Online Library or from the author.

**Acknowledgements**

The work was financially supported by National Natural Science Foundation of China (51821005) and Young Elite Scientists Sponsorship Program by CAST (YESS, 2018QNRC001).

**Conflict of Interest**

The authors declare no conflict of interest.

**Author Contributions**

Conceptualization: Q.C., J.Y.; methodology: J.Y., Q.C., and X.H.; investigation: J.Y., Y.J., and Y.S.; R.L.; formal analysis: J.Y. and Y.J.; funding acquisition: Q.C. and L.L.; project administration: Q.C., J.Y., and Y.S., R.L.; formal analysis: J.Y.; visualization: J.Y. and Q.C.; writing—original draft: J.Y., Q.C., Y.J., Y.S., and R.L.; writing—review and editing: Q.C., J.Y., and X.H.

**Data Availability Statement**

The data that support the findings of this study are available from the corresponding author upon reasonable request.
Keywords
Cosserat rod model, deep reinforcement learning, magnetic actuation, magnetic soft robotics, magnetization

Received: October 15, 2022
Revised: December 7, 2022
Published online: January 20, 2023

[1] M. Li, A. Pal, A. Aghakhani, A. Pena-Francesch, M. Sitti, Nat. Rev. Mater. 2022, 7, 235.
[2] M. Ilami, H. Bagheri, R. Ahmed, E. O. Skowronek, H. Marvi, Adv. Mater. 2021, 33, 2003139.
[3] N. S. Usevitch, Z. M. Hammond, M. Schwager, A. M. Okamura, E. W. Hawkes, S. Follmer, Sci. Robot. 2020, 5, eaa0492.
[4] D. Drotman, S. Jadhav, D. Sharp, C. Chan, M. T. Tolley, Sci. Robot. 2021, 6, eay2627.
[5] F. Zhai, Y. Feng, Z. Li, Y. Xie, J. Ge, H. Wang, W. Qiu, W. Feng, Mater. Sci. Eng., C 2021, 4, 3313.
[6] J. Zhang, Y. Guo, W. Hu, M. Sitti, Adv. Mater. 2021, 33, 2100336.
[7] H. Shahsavaran, A. Aghakhani, H. Zeng, Y. Guo, Z. S. Davidson, A. Primagi, M. Sitti, Proc. Natl. Acad. Sci. 2020, 117, 5125.
[8] M. P. Da Cunha, M. G. Debije, A. P. Schenning, Chem. Soc. Rev. 2020, 49, 6568.
[9] X. Ji, X. Liu, V. Cacucciolo, M. Imboden, Y. Civet, A. El Haitami, S. Cantin, Y. Perriard, H. Shea, Sci. Robot. 2019, 4, aaz6451.
[10] G. Li, X. Chen, F. Zhou, Y. Liang, Y. Xiao, X. Cao, Z. Zhang, M. Zhang, B. Wu, S. Yin, Y. Xu, Nature 2021, 591, 66.
[11] W. Hu, G. Z. Lum, M. Mastrangeli, M. Sitti, Nature 2018, 554, 81.
[12] Y. Kim, H. Yuk, R. Zhao, S. A. Chester, X. Zhao, Nature 2018, 558, 274.
[13] Q. Cao, Q. Fan, Q. Chen, C. Liu, X. Han, L. Li, Mater. Horiz. 2020, 7, 638.
[14] Y. Kim, X. Zhao, Chem. Rev. 2022, 122, 5317.
[15] J. Cui, T. Y. Huang, Z. Luo, P. Testa, H. Xu, Z. X. Chen, B. J. Nelson, L. J. Heyderman, Nature 2019, 575, 164.
[16] T. Xu, J. Zhang, M. Salehizadeh, O. Onaizah, E. Diller, Sci. Robot. 2019, 4, aav4494.
[17] Y. Alapan, A. C. Karacakol, S. N. Guzelhan, I. Isik, M. Sitti, Sci. Adv. 2020, 6, eabc6414.
[18] H. Deng, K. Sattari, Y. Xie, P. Liao, Z. Yan, J. Lin, Nat. Commun. 2020, 11, 6125.
[19] Y. Ju, R. Hu, Y. Xie, J. Yao, X. Li, Y. Lv, X. Han, Q. Cao, L. Li, Nano Energy 2021, 87, 106169.
[20] L. Manamanchiaporn, T. Xu, W. Wu, IEEE/ASME Trans. Mechatron. 2020, 25, 2688.
[21] N. Ebrahimi, C. Bi, D. J. Cappelleri, G. Ciuti, A. T. Conn, D. Faivre, N. Habibi, A. Hołosovský, V. Iacovacci, I. S. Khalil, V. Magdanz, Adv. Funct. Mater. 2021, 31, 2005137.
[22] Z. Ren, R. Zhang, R. H. Soon, Z. Liu, W. Wu, H. R. Onck, M. Sitti, Sci. Adv. 2021, 7, abh8202.
[23] L. Cao, D. Yu, Z. Xia, H. Han, C. Liu, T. Yin, Z. He, Adv. Mater. 2020, 32, 2000827.
[24] Q. Ze, X. Kuang, S. Wu, J. Wong, S. M. Montgomery, R. Zhang, J. M. Kovitz, F. Yang, H. J. Qi, R. Zhao, Adv. Mater. 2020, 32, 1906657.
[25] X. Yang, W. Shang, H. Lu, Y. Liu, L. Yang, R. Tan, X. Wu, Y. Shen, Sci. Robot. 2020, 5, eabc8391.
[26] Y. Kim, C. A. Parada, S. Liu, X. Zhao, Sci. Robot. 2019, 4, eaa7329.
[27] Z. Ren, W. Hu, X. Dong, M. Sitti, Nat. Commun. 2019, 10, 2703.
[28] Z. Zheng, H. Wang, L. Dong, Q. Shi, J. Li, T. Sun, Q. Huang, T. Fukuda, Nat. Commun. 2021, 12, 411.
[59] R. S. Sutton, A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge 2018.

[60] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba 2016, Preprint at http://arxiv.org/abs/1606.01540.

[61] T. Haarnoja, A. Zhou, P. Abbeel, S. Levine, in *Proc. of the 35th international conference on machine learning*, Vol. 80, PMLR, Stockholm, Sweden 2018, pp. 1861–1870.

[62] E. B. Joyee, Y. Pan, *Soft Robot*. 2019, 6, 333.

[63] H. Lu, Y. Hong, Y. Yang, Z. Yang, Y. Shen, *Adv. Sci*. 2020, 7, 2000069.

[64] S. O. Demir, U. Culha, A. C. Karacakol, A. Pena-Francesch, S. Trimpe, M. Sitti, *Int. J. Rob. Res.* 2021, 40, 1331.

[65] X. Li, Q. Cao, Z. Ouyang, N. Liu, M. Li, X. Han, L. Li, *J. Mater. Process. Technol.* 2020, 276, 116411.

[66] C. J. C. H. Watkins, P. Dayan, *Mach. Learn.* 1992, 8, 279.

[67] D. P. Kingma, J. L. Ba, in *Proc. of the 3rd Inter. Conf. on Learning Representations*, ICLR, San Diego, USA 2015, pp. 1–15.

[68] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra 2015, Preprint at https://arxiv.org/abs/1509.02971.

[69] J. M. Lucas, M. S. Saccucci, *Technometrics* 1990, 32, 1.

[70] H. van Hasselt, A. Guez, D. Silver, in *Proc. of the Thirtieth AAAI Conf. on Artificial Intelligence*, Vol. 30, AAAI Press, Phoenix, Arizona, USA 2016, pp. 2094–2100.