10.24425/acs.2026.158424

Archives of Control Sciences
Volume 36(LXXII), 2026
No. 1, *pages 133–158*

Reinforcement learning-based obstacle avoidance for continuum robots

Jakub KOŁOTA and Turhan Can KARGIN

This work presents a reinforcement learning framework for controlling a planar three-section continuum robot in environments with static obstacles. Assuming constant curvature for each section, the robot is trained to navigate toward a fixed goal while avoiding collisions with multiple static objects. A custom simulation environment was developed to support three levels of scenario difficulty, easy, medium and hard, each with varying obstacle density and placement. The learning process is driven by the Deep Deterministic Policy Gradient (DDPG) algorithm, which enables smooth and continuous curvature control. Careful attention was paid to the design of the reward function and the network architecture, both of which were critical to achieving stable and reliable policy learning. Performance was evaluated across multiple runs, revealing that the agent successfully generalized its behavior across scenarios of increasing complexity. The proposed framework demonstrates the potential of reinforcement learning as a viable approach to safe and adaptive control in continuum robotic systems, with promising implications for applications such as medical navigation, search and rescue, and inspection in confined environments.

Key words: reinforcement learning; DDPG; continuum robots

1. Introduction

Continuum robots are an innovative class of robotic systems characterized by their flexible, deformable segments that emulate the movements of biological structures such as tentacles and trunks [1]. Unlike traditional rigid-joint manipulators, continuum robots exhibit continuous bending along their length, enabling them to perform complex tasks in constrained and dynamic environments [2].

J. Kołota (corresponding author, e-mail: Jakub.Kolota@put.poznan.pl) is with Institute of Automatic Control and Robotics, Poznan University of Technology, Piotrowo 3A, 60-965 Poznań, Poland.
T.C. Kargin (e-mail: turhancan.kargin@gmail.com) unafiliated, Izmir, Turkey
This work was supported by the statutory grant No. 0211/SBAD/0125.
Received 22.09.2025. Revised 6.02.2026.

This inherent flexibility allows continuum robots to access areas that are challenging for conventional robots, making them particularly valuable in applications requiring precision and adaptability. For example, they are increasingly used in minimally invasive surgery, where their ability to navigate confined spaces with minimal disruption is critical [3–6]. In addition, continuum robots are deployed in industrial pipeline inspections to traverse complex pathways and in search-and-rescue operations to maneuver through debris and confined spaces [7–9]. These unique features, combined with their potential for high degrees of freedom, position continuum robots as a transformative technology in robotics, capable of addressing the demands of challenging operational environments.

Controlling continuum robots presents significant challenges due to their nonlinear behavior and continuously deformable structures. Unlike rigid manipulators, continuum robots lack discrete joints, resulting in complex equations that describe their kinematics and dynamics. These equations often exhibit high degrees of nonlinearity, making analytical modeling and traditional control strategies impractical for many real-world scenarios. Additionally, their infinite degrees of freedom introduce uncertainties that conventional methods, such as PID, optimal control, or predictive control, struggle to address effectively [10]. To overcome these limitations, AI-based learning systems have emerged as promising alternatives. Neural networks and reinforcement learning (RL) approaches, in particular, offer the ability to adaptively learn control policies through interaction with the environment, eliminating the need for precise mathematical models. RL frameworks have shown their potential to manage complex control tasks for continuum robots, providing flexibility and robustness in dynamic and constrained environments [11]. This paradigm shift towards AI-based control represents a significant advancement in addressing the inherent complexities of continuum robot control.

RL has emerged as a transformative approach for controlling continuum robots, offering unparalleled flexibility, adaptability, and the ability to manage complex nonlinear dynamics. Unlike traditional control methods, RL does not require precise mathematical models; instead, it learns optimal control policies by interacting with the environment. Frameworks such as the deep deterministic policy gradient (DDPG) have shown significant promise in this domain. For example, DDPG has been successfully applied to planar and spatial continuum robots, allowing autonomous movement within the task space and demonstrating its potential to achieve position-reaching tasks [12, 13]. Moreover, the design of RL environments, particularly reward functions, has been shown to critically influence algorithm performance [14]. Beyond basic control, RL has also been leveraged for obstacle avoidance tasks in other types of robotic system; for example, combining DDPG with Hindsight Experience Replay (HER) has achieved

successful navigation in both simulated and real 6-DOF robots [15]. Further advancements include the development of the ensemble light-weight model-free reinforcement learning network (ELFNet), which improves sample efficiency and control performance for soft robots with multiple passive degrees of freedom [11]. These studies underscore the potential of RL-based frameworks for addressing the unique challenges of continuum robot control, paving the way for enhanced adaptability and autonomy in complex environments.

Static obstacle avoidance represents a critical enhancement to the control framework for continuum robots, allowing them to navigate environments with increased safety and precision. This feature addresses the challenge of avoiding collisions with stationary objects, a capability essential in many real-world applications. For example, in surgical robotics, the ability to avoid anatomical structures while accessing target sites ensures minimal invasive damage and improved patient outcomes. Similarly, in pipeline inspection, obstacle avoidance allows robots to traverse intricate pathways without causing structural harm, while in search and rescue operations, it ensures safe navigation through debris and confined spaces. Incorporating static obstacle avoidance into RL frameworks not only improves the robot's autonomy but also significantly enhances its adaptability to unstructured environments. This innovation bridges the gap between theoretical advancements and practical applicability, ensuring that continuum robots are better equipped to perform complex tasks in constrained and hazardous settings. Unlike prior continuum-robot RL studies that primarily demonstrate collision-free reaching in a single task configuration or focus on algorithmic variants, this paper emphasizes a static-obstacle avoidance with systematically increasing difficulty and explicitly reports cross-scenario generalization under a fixed DDPG formulation (state/action design, reward terms, and network architecture) [12–14]. This positioning highlights the novelty of our environment and evaluation protocol, which are designed to make obstacle-avoidance performance comparable across difficulty levels.

Incorporating obstacle avoidance into RL frameworks for continuum robots presents significant challenges due to the unique characteristics of these systems. Continuum robots, with their flexible and continuously deformable structures, operate in high-dimensional state-action spaces, making the design of effective control policies inherently complex. Achieving stable and efficient training is further complicated by the need to balance multiple objectives, such as minimizing collision risks while optimizing path efficiency. Formulating a reward function that appropriately captures these trade-offs is particularly challenging, as poorly designed rewards can lead to suboptimal behaviors or prolonged training times. Additionally, non-linear dynamics and infinite degrees of freedom in continuum robots demand neural network architectures capable of capturing intricate

patterns and dependencies, increasing computational requirements. These challenges are compounded when adapting RL algorithms such as DDPG to handle static obstacle avoidance, as they must accommodate the continuous nature of the robot while ensuring real-time decision-making. Addressing these issues requires careful tuning of hyperparameters, robust reward engineering, and innovative architectural designs, highlighting the complexity of integrating obstacle avoidance into RL frameworks for continuum robots.

This paper makes the following contributions to reinforcement learning-based control of continuum robots:

- We formulate planar three-section continuum-robot navigation as a continuous-control obstacle-avoidance task and introduce three static-obstacle scenarios (easy, medium, and hard) with systematically increasing obstacle complexity.
- We implement a DDPG-based controller with continuous curvature-rate actions and a state representation that includes goal information and obstacle distances. Stable learning is enabled by a carefully crafted reward design that combines goal-distance, progress, time, and obstacle-avoidance terms, together with an empirically selected network architecture and hyperparameters to address challenges such as stable training, efficient policy learning, and computational efficiency.
- Using a single learning setup, the policy can be trained across increasing scenario difficulty, achieving success rates of 94%, 89%, and 79% in the easy, medium, and hard cases, respectively, with convergence within 900–1500 episodes.

The remainder of the paper is organized as follows: Section 2 presents the continuum-robot kinematic model, Section 3 details the reinforcement learning formulation and DDPG method, and Section 4 and Section 5 present the experimental setup and results.

2. Kinematics modeling of the continuum robot

Continuum robots differ fundamentally from traditional rigid-link robots in that they do not possess discrete joints. Instead, they are composed of a continuous flexible backbone that allows them to perform smooth, highly dexterous movements, making them well-suited for constrained environments such as surgical robotics, industrial manipulation, and search-and-rescue operations. Modeling the kinematics of continuum robots presents unique challenges due to their flexible and continuously deformable structure. To address these complexities, a common approach is to adopt the constant curvature assumption, which simpli-

fies the representation of the robot shape while maintaining its key characteristics. Under this assumption, each section of the robot is modeled as a circular arc with a constant radius of curvature. This assumption is particularly useful in practical applications, as it allows for an analytical representation of the robot's shape while significantly reducing computational complexity.

We consider a planar continuum robot composed of three serially connected constant-curvature sections. Section $i \in \{1, 2, 3\}$ is parameterized by its arc length l_i, curvature κ_i, and bending angle θ_i, where $\theta_i = \kappa_i l_i$. Kinematic quantities are expressed with respect to a fixed base frame $\{0\}$ attached to the robot base, intermediate section frames $\{1\}$, $\{2\}$, $\{3\}$ attached to the end of each section.

2.1. Constant curvature assumption

The constant curvature assumption simplifies kinematics by assuming that each section of the robot bends in a uniform arc, characterized by a single curvature value. For section i, we use the curvature κ_i and arc length l_i as inputs, and define the bending angle as $\theta_i = \kappa_i l_i$. Using this assumption, the position and orientation of the end-effector of a single-section continuum robot can be determined by integrating the curvature along the arc length. [16] obtained the transformation matrix defining the position of the endpoint in a homogeneous form using the modified Denavit–Hartenberg (DH) convention to express the relative transforms between successive section frames for forward kinematic analysis. This matrix is given as follows:

$$T_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & \frac{1}{\kappa_i}(1-\cos\theta_i) \\ \sin\theta_i & \cos\theta_i & 0 & \frac{1}{\kappa_i}\sin\theta_i \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

where we obtain the position information defined as (x, y), which represents the Cartesian coordinates of the endpoint relative to the base frame.

For continuum robots consisting of multiple sections, the overall transformation matrix is obtained by chaining the transformation matrices of individual sections. For a three-section continuum robot, the final transformation from the base to the end effector is given by:

$$T_{\text{final}} = T_1 T_2 T_3 \tag{2}$$

where each T_i represents the transformation matrix for section i. Here, $T_i \equiv {}^{i-1}T_i(\kappa_i, l_i)$ denotes the homogeneous transformation from frame $\{i-1\}$ to frame $\{i\}$ parameterized by the section curvature κ_i and section length l_i. In the planar

setting considered in this paper, the full configuration is therefore given by $q = [\kappa_1, \kappa_2, \kappa_3]^T$ (with fixed l_i), and the end-effector pose with respect to the base frame follows from ${}^0T_3 = {}^0T_1\,{}^1T_2\,{}^2T_3$. Since $\theta_i = \kappa_i l_i$, the independent kinematic parameters required by the transforms are precisely the pairs (κ_i, l_i), all of which are indicated in Fig. 1.

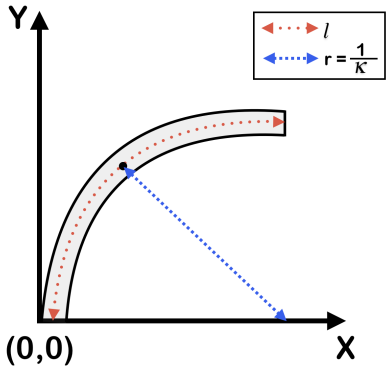


Figure 1: Illustration of a continuum robot modeled using the constant curvature assumption. A single-section continuum robot where the bending is characterized by a constant curvature κ, an arc length l, and a bending angle θ, with $\theta = \kappa l$

By defining a continuum robot as a series of these transformations, its forward kinematics can be derived in a structured manner. In this work, we adopt the modified DH based approach [16] to compute the position of the end-effector while integrating the constraints of obstacle avoidance in subsequent sections.

The choice of constant-curvature modeling combined with the DH formulation provides a computationally efficient means to describe continuum robot motion, forming the foundation for reinforcement learning-based control. The motion of continuum robots is determined not only by their configuration, but also by the velocities of their actuators. Unlike traditional rigid-link robots, continuum robots bend continuously, requiring a different approach to velocity modeling. Velocity kinematics is crucial in motion planning as it allows us to calculate how changes in actuator input affect the movement of the robot end-effector in Cartesian space. Figure 2 illustrates the velocity kinematics of a three-section continuum robot, showing how changes in curvature rates $(\dot{\kappa}_1, \dot{\kappa}_2, \dot{\kappa}_3)$ propagate to the end-effector velocity $(\dot{x}, \dot{y})$. The motion of the end effector, represented by the velocity vector, is determined using the Jacobian-based formulation, which will be discussed in Equation (3).

The relationship between the curvature velocities and the resulting velocity of the end effector is governed by the Jacobian matrix. Given a continuum robot

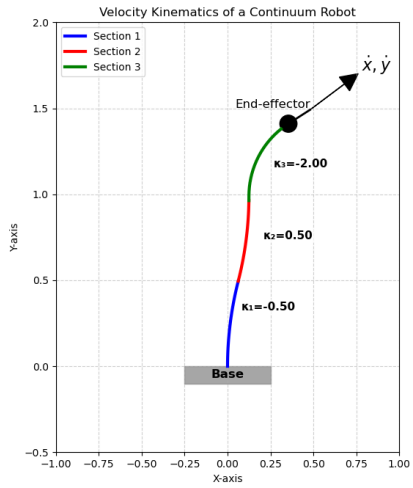


Figure 2: Velocity kinematics of a continuum robot, illustrating the mapping of curvature rates to end-effector velocity

with n independent sections, velocity kinematics can be expressed as:

$$\dot{x} = J\dot{q} \tag{3}$$

where: $\dot{x} \in \mathbb{R}^m$ is the velocity of the end effector in Cartesian space (e.g., linear and angular velocities), $J \in \mathbb{R}^{m\times n}$ is the Jacobian matrix, which maps the joint space velocities to the Cartesian space velocities and $\dot{q} \in \mathbb{R}^n$ represents the velocity of the joint parameters (e.g., curvature rates). For a planar three-section continuum robot, the configuration space consists of the curvature values $\kappa_1, \kappa_2, \kappa_3$. The state space can be defined as:

$$q = [\kappa_1, \kappa_2, \kappa_3]^T \tag{4}$$

The time derivative of the configuration variables, $\dot{q} = [\dot{\kappa}_1, \dot{\kappa}_2, \dot{\kappa}_3]^T$, represents the rate of change in curvature in each section. Using forward kinematics, we can derive the Jacobian matrix elements as:

$$J = \begin{bmatrix} \dfrac{\partial x}{\partial \kappa_1} & \dfrac{\partial x}{\partial \kappa_2} & \dfrac{\partial x}{\partial \kappa_3} \\ \dfrac{\partial y}{\partial \kappa_1} & \dfrac{\partial y}{\partial \kappa_2} & \dfrac{\partial y}{\partial \kappa_3} \end{bmatrix}. \tag{5}$$

As seen in Figure 2, each section of the continuum robot bends with a distinct curvature $(\kappa_1, \kappa_2, \kappa_3)$. The combined effect of these curvatures defines the overall robot posture and influences the velocity of the end-effector. The Jacobian matrix, as defined in Equation (3), captures this mapping by relating curvature velocity

to Cartesian velocity. This transformation is critical in motion planning, where small adjustments in $\dot{\kappa}$ result in desired movements in Cartesian space.

In robotic control, the Jacobian matrix provides essential information for motion planning:

- It allows mapping from actuator velocities to Cartesian velocities, which is critical for trajectory execution.
- The inverse Jacobian can be used for inverse kinematics, computing required curvature changes to achieve a desired end-effector velocity.
- Singularities can be analyzed by examining the Jacobian determinant, providing insights into configurations where motion control becomes problematic.

Numerical computation of the Jacobian. We compute the Jacobian numerically from the forward kinematics mapping $\mathbf{x} = T(\mathbf{q}, l)$, where $\mathbf{x} = [x, y]^T$ denotes the end-effector position and $\mathbf{q} = [\kappa_1, \kappa_2, \kappa_3]^T$ the section curvatures. The Jacobian $\mathbf{J} = \partial T(\mathbf{q}, l)/\partial \mathbf{q} \in \mathbb{R}^{2\times 3}$ is approximated via forward finite differences:

$$\mathbf{J} \approx \frac{T(\mathbf{q} + \epsilon, l) - T(\mathbf{q} - \epsilon, l)}{2\epsilon},$$

where ϵ represents a small change in $\mathbf{q}$, (we use $\epsilon = 0.001$ in all experiments). Using the Jacobian-based velocity formulation, reinforcement learning algorithms can learn effective control policies by optimizing $\dot{q}$ to achieve desired end-effector motions while avoiding obstacles. In the proposed framework, the Jacobian is used inside the RL simulator for velocity/state propagation, rather than as an explicit trajectory-planning method. The next sections build upon this foundation to describe how reinforcement learning is leveraged to control the continuum robot efficiently.

Assumptions and limitations. The kinematic model assumes (i) planar motion (2D workspace), (ii) fixed section lengths l_i, so that the control inputs act through the curvature variables. In addition, the environments considered in this study contain only static obstacles; dynamic interactions and external disturbances are not modeled.

2.2. Modifications for obstacle avoidance

The standard kinematic model of a continuum robot defines its motion based on the constant curvature assumption and velocity kinematics. However, in real-world environments, the robot must avoid collisions with static obstacles while reaching its target position. This requires incorporating obstacle constraints into the kinematic framework. To integrate obstacle avoidance into the kinematics, we define a modified task space representation where obstacles act as constraints.

The task space is formulated as:

$$\mathcal{T} = \{(x, y) \in \mathbb{R}^2 \mid (x, y) \notin O\} \tag{6}$$

where: $\mathcal{T}$ represents the feasible task space for the robot and $O = \{(x_o, y_o)\}$ is the set of obstacle positions in the environment.

A feasible trajectory must ensure that for all time steps t, the robot's end-effector position (x_t, y_t) satisfies:

$$\forall (x_o, y_o) \in O, \quad \|(x_t, y_t) - (x_o, y_o)\| > d_{\text{safe}} \tag{7}$$

where d_{safe} is the minimum safety distance required to avoid collisions.

In reinforcement learning, obstacle constraints are enforced through the reward function. The reward function is modified to include a penalty term R_{obs} that discourages the robot from approaching obstacles:

$$R = R_{\text{goal}} - \sum_{o \in O} \frac{k_{\text{penalty}}}{\|(x_t - x_o, y_t - y_o)\|} \tag{8}$$

where: R_{goal} is the original reward for reaching the target and k_{penalty} is a weight that controls the penalty strength for proximity to obstacles.

This formulation ensures that the reinforcement learning agent learns to navigate around obstacles while minimizing deviations from the optimal path.

By modifying the velocity kinematics and reward function, the robot:

- Maintains a safe distance from obstacles using Equation (7).
- Learns an efficient trajectory through reinforcement learning, balancing goal-reaching with collision avoidance using Equation (8).

These modifications ensure safe and efficient motion planning in environments with static obstacles.

3. Reinforcement learning framework

3.1. Markov Decision Process (MDP) formulation

In reinforcement learning (RL), the control problem is framed as a Markov Decision Process (MDP), where the agent (continuum robot) interacts with the environment to maximize cumulative rewards. The MDP is defined as a tuple:

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma) \tag{9}$$

where: $\mathcal{S}$ is the state space, representing the robot's perception of the environment, $\mathcal{A}$ is the action space, defining the robot's control inputs, $P(s'|s, a)$ is the transition probability function, $R(s, a)$ is the reward function that guides learning and

$\gamma \in [0, 1]$ is the discount factor, determining future reward importance. The state space $\mathcal{S}$ includes essential information for the robot to navigate its environment. The state vector at time step t is defined as:

$$s_t = \left[x_t, y_t, x_{\text{goal}}, y_{\text{goal}}, d_{\text{obs}_1}, d_{\text{obs}_2}, \ldots, d_{\text{obs}_i}\right] \tag{10}$$

where: (x_t, y_t) is current end-effector position, $(x_{\text{goal}}, y_{\text{goal}})$ is goal position and $d_{\text{obs}_i} = \sqrt{(x_{\text{obs}_i} - x_t)^2 + (y_{\text{obs}_i} - y_t)^2}$ defines distance to the i-th obstacle. In other words, s_t collects the current end-effector position, the goal position, and the set of end-effector-to-obstacle distances observed at time t.

The action space $\mathcal{A}$ consists of the time derivatives of the robot's curvatures:

$$a_t = [\dot{\kappa}_1, \dot{\kappa}_2, \dot{\kappa}_3] \tag{11}$$

where: $\dot{\kappa}_i$ represents the rate of change of curvature for section i. Here, a_t denotes the continuous control command (curvature-rate vector) produced by the policy at time t.

The agent selects actions to adjust curvature dynamically while navigating toward the goal.

The reward function $R(s, a)$ is structured to encourage goal-reaching while penalizing unsafe movements. The total reward function consists of:

$$R(s_t, a_t) = R_{\text{goal}}(s_t) + R_{\text{progress}}(s_t) + R_{\text{time}}(s_t) + R_{\text{obstacle}}(s_t) \tag{12}$$

where:

Goal distance reward: A primary objective is reaching the goal efficiently. The robot's Euclidean distance to the goal determines a base cost:

$$R_{\text{goal}}(s_t) = -\frac{\|(x_{\text{goal}}, y_{\text{goal}}) - (x_t, y_t)\|}{\|(x_{\text{goal}}, y_{\text{goal}}) - (x_0, y_0)\|}. \tag{13}$$

Progress-based reward: To promote efficient movement, the change in distance from the previous step is rewarded:

$$R_{\text{progress}}(s_t) = -2 \times \frac{(\text{previous error} - \text{current error})}{\text{initial distance}}. \tag{14}$$

Time penalty: A small penalty discourages excessive time usage:

$$R_{\text{time}}(s_t) = 0.1 \times \Delta t. \tag{15}$$

Obstacle Avoidance Penalty: To avoid collisions, an increasing penalty is applied for proximity to any obstacle:

$$R_{\text{obstacle}}(s_t) = \sum_{i=1}^{N_{\text{obs}}} \begin{cases} -2\left(1 - \frac{d_{\text{obs}_i}}{d_{\text{safe}}}\right), & \text{if } d_{\text{obs}_i} \leqslant d_{\text{safe}} \\ 0, & \text{otherwise} \end{cases} \tag{16}$$

where: N_{obs} is the total number of obstacles and d_{safe} is the predefined minimum safe distance from any obstacle.

Goal proximity bonus: A final positive reward is added when the robot is close to the goal:

$$R_{\text{goal\_proximity}}(s_t) = \begin{cases} 5 \times (0.02 - \|(x_t, y_t) - (x_{\text{goal}}, y_{\text{goal}})\|), \\ \qquad\qquad \text{if } \|(x_t, y_t) - (x_{\text{goal}}, y_{\text{goal}})\| \leqslant 0.02, \\ 0, \qquad\qquad \text{otherwise.} \end{cases} \tag{17}$$

This reward structure ensures:

- **Path efficiency**: The agent is penalized for slow movements and inefficient detours.
- **Obstacle avoidance**: Smooth penalty gradients prevent the agent from being trapped near obstacles.
- **Balanced navigation**: The combination of progress-based rewards and obstacle penalties leads to smooth, adaptive movements.

By extending the reward function to support variable numbers of obstacles, the continuum robot can be tested in environments with different levels of complexity. This allows for a more generalizable reinforcement learning framework, enabling experiments with varying obstacle densities and layouts.

Control loop and state update. At each simulation step t, the policy takes the current state s_t and outputs the continuous action $a_t = [\dot{\kappa}_1, \dot{\kappa}_2, \dot{\kappa}_3]^T$. The simulator integrates this curvature-rate command to update the curvature configuration $q_t = [\kappa_1, \kappa_2, \kappa_3]^T$, computes the resulting end-effector pose (x_t, y_t) through forward and velocity kinematics (T_{final}), and evaluates obstacle proximity/collision constraints using the end-effector-to-obstacle distances. The reward r_t is then computed from the goal-reaching, progress, time, and obstacle terms (Eq. (12)). Finally, the transition (s_t, a_t, r_t, s_{t+1}) is stored in the replay buffer for off-policy DDPG updates (see Figure 3).

3.2. Deep Deterministic Policy Gradient (DDPG) algorithm

Reinforcement learning (RL) for continuum robots requires an algorithm capable of handling high-dimensional, continuous control tasks. The DDPG algorithm [17] is selected due to its ability to operate in continuous action spaces, making it well-suited for controlling the curvature of a continuum robot. Unlike traditional rigid-body robots, continuum robots require smooth and continuous control of their curvature parameters $(\kappa_1, \kappa_2, \kappa_3)$. Standard RL algorithms such as Deep Q-Networks (DQN) [18] are designed for discrete action spaces and are therefore unsuitable for continuum robots.

DDPG is an off-policy, model-free RL algorithm that extends the Deterministic Policy Gradient (DPG) method [19] by leveraging deep neural networks. Its key advantages include:

- **Handling Continuous Action Spaces**: Unlike DQN, which discretizes actions, DDPG directly outputs continuous-valued actions for smooth curvature control.
- **Actor-Critic Framework**: DDPG utilizes separate actor and critic networks, which improve learning stability.
- **Off-Policy Learning**: Experiences are stored in a replay buffer, reducing correlations between consecutive updates and improving sample efficiency.
- **Target Networks for Stability**: DDPG maintains slowly updated target networks to prevent divergence in training.

DDPG operates using an actor-critic architecture, where the **actor network** $\mu_\theta(s)$ maps states to continuous actions and the **critic network** $Q_\phi(s,a)$ evaluates the expected return for state-action pairs. The objective of DDPG is to learn an optimal **policy** $\mu^*(s)$ that maximizes the expected cumulative reward:

$$G(\theta) = \mathbb{E}_{s_t\sim\rho^\mu}\left[Q^\mu(s_t,\mu_\theta(s_t))\right] \tag{18}$$

where: $Q^\mu(s_t,a_t)$ is the action-value function under policy μ and ρ^μ is the state distribution induced by the policy.

Actor update (Policy gradient): The actor is updated using the deterministic policy gradient:

$$\nabla_\theta G(\theta) \approx \mathbb{E}_{s_t\sim\rho^\mu}\left[\nabla_a Q_\phi(s_t,a)\big|_{a=\mu_\theta(s_t)}\nabla_\theta\mu_\theta(s_t)\right] \tag{19}$$

where $\nabla_\theta\mu_\theta(s_t)$ represents how the actor network changes with respect to parameters θ.

Critic update (Bellman equation): The critic network is updated by minimizing the Temporal Difference (TD) error:

$$L(\phi) = \mathbb{E}\left[(y_t - Q_\phi(s_t,a_t))^2\right] \tag{20}$$

where the **target value** y_t is computed using the Bellman equation:

$$y_t = r_t + \gamma Q_{\phi'}(s_{t+1},\mu_{\theta'}(s_{t+1})) \tag{21}$$

where: γ is the discount factor and ϕ' and θ' are the parameters of the slow-moving target networks.

Target network update: To stabilize training, DDPG maintains target networks for both the actor and critic, which are updated using soft updates:

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta' \tag{22}$$

where $\tau \ll 1$ (typically $\tau = 0.001$) ensures gradual updates, preventing instability. As shown in Table 1, DDPG is the most suitable algorithm for continuum robots due to its ability to:

- Directly optimize continuous control signals without discretization.
- Provide deterministic action selection, leading to smooth control.
- Improve stability through experience replay and target networks.

Table 1: Comparison of DDPG with other RL algorithms

Algorithm	Action space	Policy type	Suitability for CR
DQN	Discrete	Value-based	Poor
PPO	Continuous	Stochastic	Moderate
TRPO	Continuous	Stochastic	Moderate
DDPG	Continuous	Deterministic	Excellent

The selection of DDPG for continuum robot control is driven by its capacity to handle high-dimensional continuous action spaces, ensuring smooth trajectory planning and stable learning. Using actor-critic methods and replay buffers, DDPG enables efficient learning of optimal curvature control strategies in dynamic environments.

3.3. Network architecture

The performance of the DDPG algorithm relies heavily on the design of the neural network architecture used for both the **actor** and **critic** networks. Various architectural configurations were tested to determine the optimal balance between training stability, sample efficiency, and generalization capability.

3.3.1. Actor network

The actor network is responsible for mapping states to continuous actions, i.e., the curvature rate changes $\dot{\kappa}_1, \dot{\kappa}_2, \dot{\kappa}_3$. The function approximator is defined as:

$$a_t = \mu_\theta(s_t) = \pi_\theta(s_t|\theta), \tag{23}$$

where: s_t is the state input (robot position, goal position, and obstacle distances), π_θ represents the policy parameterized by neural network weights θ and the output a_t represents the selected actions (curvature adjustments).

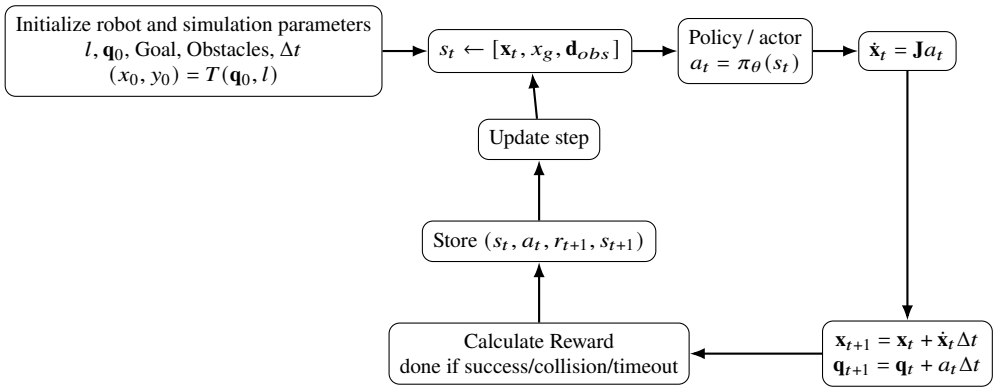


Figure 3: Pipeline overview of the RL-controlled continuum robot simulation loop. The policy outputs curvature-rate commands, which are integrated to update curvature parameters; forward kinematics yields the end-effector pose, which is used to compute distances, reward, and transitions stored for off-policy updates

The final architecture selected for the actor network consists of:

- **Input Layer:** State vector s_t of dimension $(6 + N_{\text{obs}})$, where N_{obs} is the number of obstacles.
- **Hidden Layers:** Three fully connected layers with [400, 300, 200] neurons, respectively. **ReLU activation** for all hidden layers.
- **Output Layer:** Three neurons corresponding to the action vector $\dot{\kappa}_i$. **Tanh activation** to ensure bounded continuous action values.

Activation function choices for the actor network: The actor network was tested with different activation functions. **ReLU** activation provided the best convergence and stability. **Sigmoid** resulted in slower learning due to the vanishing gradients. **Tanh** was selected for the final output layer to bound actions between $[-1, 1]$. The complete function approximation of the actor network is given as:

$$a_t = \tanh(W_3\sigma(W_2\sigma(W_1 s_t + b_1) + b_2) + b_3), \tag{24}$$

where: W_i and b_i represent the weight matrices and biases of the layers and $\sigma(\cdot)$ denotes the ReLU activation function.

3.3.2. Critic network

The critic network evaluates the action-value function $Q(s, a)$, predicting the expected return of a given state-action pair. The function is defined as follows:

$$Q_\phi(s_t, a_t) = f_\phi(s_t, a_t), \tag{25}$$

where: s_t represents the current state, a_t is the selected action and f_ϕ is the function parameterized by network weights ϕ. The final architecture for the critic network consists of:

- **Input Layer:** Concatenated state-action pair (s_t, a_t).
- **Hidden Layers:** Three fully connected layers with [400, 300, 200] neurons, respectively. **ReLU activation** for all hidden layers.
- **Output Layer:** Single neuron producing $Q(s, a)$. **Linear activation** to allow unrestricted output range.

Activation function choices for the critic network: Different activation functions were tested. **ReLU** provided stable training and efficient learning. **Leaky ReLU** was tested but did not significantly improve performance. **Tanh and Sigmoid** were avoided in hidden layers due to saturation effects. The critic network function approximation is given as:

$$Q_\phi(s_t, a_t) = W_3\sigma(W_2\sigma(W_1[s_t, a_t] + b_1) + b_2) + b_3\,, \tag{26}$$

where: $[s_t, a_t]$ represents the input of the concatenated action of the state, W_i and b_i denote the weights and biases of the layers.

3.3.3. Experimentation with different architectures

To optimize performance, several architectural variations were tested:

Table 2: Comparison of different neural network architectures

Architecture	Hidden layers	Activation functions	Performance
A1	[256, 128]	ReLU	Unstable convergence
A2	[400, 300, 200]	ReLU	Best performance
A3	[512, 512]	Leaky ReLU	Slower training
A4	[128, 128, 128]	Tanh	Poor learning

The final selected network architecture consists of three hidden layers with ReLU activations in both the actor and the critic networks. The output layers use Tanh for the actor (ensuring bounded actions) and Linear for the critic (allowing unrestricted Q-values). This architecture demonstrated stable training and efficient learning for continuum robot control.

3.4. Hyperparameter tuning

The performance of the DDPG algorithm is highly sensitive to the choice of hyperparameters. Proper tuning ensures efficient learning, stable convergence, and effective policy generalization. The key hyperparameters investigated include

the learning rate, discount factor, exploration noise, batch size, and replay buffer size.

The learning rate determines how quickly the neural networks (actor and critic) update their parameters. Too high a learning rate can cause instability, while too low a rate leads to slow convergence. The update equations for the actor and critic networks are:

$$\theta' \leftarrow \theta - \alpha \nabla_\theta G(\theta) \quad \phi' \leftarrow \phi - \alpha \nabla_\phi L(\phi) \tag{27}$$

where: α is the learning rate, $\nabla_\theta G(\theta)$ is the update of the policy gradient for the actor and $\nabla_\phi L(\phi)$ is the loss gradient for the critic. Table 3 shows the values of the parameter α that were tested by experimentation. The final selected learning rate was $\alpha = 0.001$.

Table 3: Effect of learning rate on training stability

Learning rate (α)	Performance	Observation
0.001	Stable	Best balance of stability and convergence
0.005	Unstable	High variance in training
0.0001	Slow Convergence	Too small to optimize effectively

The discount factor γ controls how much future rewards influence current decisions. The Bellman equation with discounting is:

$$Q(s_t, a_t) = r_t + \gamma Q(s_{t+1}, a_{t+1}) \tag{28}$$

The final selected discount factor was $\gamma = 0.99$. Exploration noise is added to the action selection to encourage policy exploration. DDPG uses an **Ornstein-Uhlenbeck (OU) process**:

$$\mathcal{N}_{t+1} = \theta(\mu - \mathcal{N}_t) + \sigma W_t \tag{29}$$

where: $\mathcal{N}_t$ is the noise in time step t, θ controls the reversion of the mean noise, σ controls the magnitude of the noise and W_t is Gaussian noise. The final selected noise parameter was $\sigma = 0.3$. The size of the batch affects the accuracy of the gradient estimation. The sampled batch is:

$$B = \{(s_i, a_i, r_i, s_i')\}_{i=1}^{N} \tag{30}$$

where N is the batch size. The final batch size selected was **128**. The replay buffer stores past experiences for off-policy learning. The update process is:

$$D_t = D_{t-1} \cup (s_t, a_t, r_t, s_{t+1}) \tag{31}$$

Table 4: Hyperparameters of the DDPG Algorithm

Hyperparameter	Final value
Learning rate (α)	0.001
Discount factor (γ)	0.99
Exploration noise (σ)	0.3
Batch size	128
Replay buffer size	10^6
Network architecture	
Actor hidden layers	[400, 300, 200]
Actor activation functions	ReLU (hidden), Tanh (output)
Critic hidden layers	[400, 300, 200]
Critic activation functions	ReLU (hidden), Linear (output)

The final replay buffer size was 10^6 transitions. Through extensive experimentation, the following hyperparameters were selected:

These values resulted in stable training, effective exploration, and efficient policy learning.

4. Experimental setup

The reinforcement learning experiments were conducted within a custom Python-based simulation environment. This environment was explicitly developed to simulate the motion of a planar three-section continuum robot and supports the dynamic configuration of task spaces and obstacles.

To evaluate the generalization ability and adaptability of the trained agent, we designed and tested the robot in multiple static obstacle scenarios with increasing levels of complexity:

- **Easy scenario:** Single obstacle placed near the path to the goal.
- **Medium scenario:** Two obstacles requiring moderate path adjustment.
- **Hard scenario:** Three or more obstacles placed close to the goal, requiring precise maneuvering and long-term planning. Each obstacle was modeled as a fixed-radius circular region in the task space, and the robot was trained to avoid these regions while navigating toward randomly placed goal points.

4.1. Training details

Trajectory/episode parameterization. Each episode is defined by the initial random curvature configuration, a goal position $(x_{\text{goal}}, y_{\text{goal}})$, and a set of static obstacles specified by their centers $(x_{\text{obs}}, y_{\text{obs}})$ (with fixed obstacle radius in the

simulator). The simulation proceeds with a fixed time step Δt; at each step, the policy outputs the curvature-rate command $a_t = [\dot{\kappa}_1, \dot{\kappa}_2, \dot{\kappa}_3]^T$, which is integrated to update q_t subject to curvature limits, and the end-effector pose is computed via forward kinematics. Obstacle proximity is evaluated via the end-effector-to-obstacle distances d_{obs_i} and the safety threshold d_{safe} (Eq. (7)); the episode terminates when the goal tolerance is reached (as in the goal-proximity condition) or when a collision/violation occurs.

The training was carried out using the DDPG algorithm with the architecture and hyperparameters specified in the previous sections. The simulation was executed on a workstation with the following configuration: CPU: Intel Core i9, GPU: NVIDIA RTX 4090, RAM: 64 GB, Framework: Python 3.9 with Keras backend [20]. The agent was trained for 5000 episodes, with a variable number of steps per episode, subject to a maximum step cap which is 500. The total number of interaction steps was tracked across all episodes to monitor the progress of the training.

4.2. Evaluation metrics

To assess the performance of the trained policies, the following evaluation metrics were used:

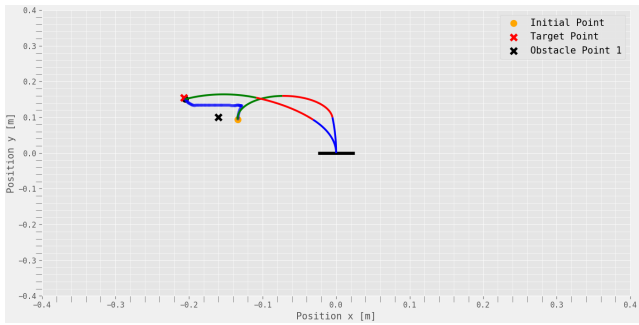
- **Success Rate:** This is defined as the percentage of evaluation episodes in which the robot reaches the goal without colliding with any static obstacles. Success is measured over a set of 100 randomly initialized test episodes across all three scenario types.
- **Effectiveness Score:** The effectiveness score is calculated as the mean value of 10 independent test episodes after training. It incorporates both trajectory efficiency (path length and smoothness) and goal-reaching success. Lower scores indicate more efficient behavior.
- **Training Time:** It was recorded to evaluate computational efficiency, although the exact duration and impact are discussed in the results section.
- **Training Convergence:** The episode reward over time was tracked throughout training. This provided insight into how quickly and consistently the agent learned to navigate within obstacle-rich environments. The convergence trend was later used to compare learning behavior across different reward functions and obstacle configurations.

5. Results and discussion

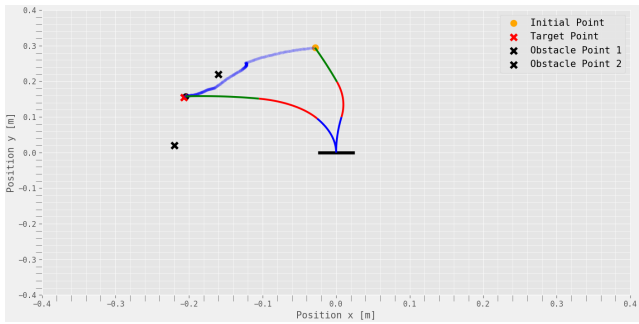
This section presents and analyzes the performance of the trained reinforcement learning agent in various static obstacle environments. The focus is on

understanding how well the robot generalized to different levels of environmental complexity and how architecture, reward design, and hyperparameter choices influenced the policy's behavior.

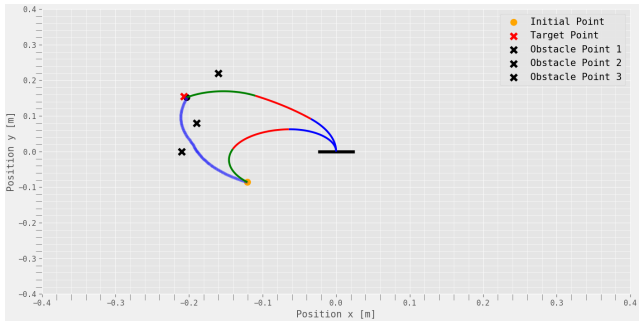
To illustrate the trajectory planning behavior of the continuum robot, we provide a representative result for each of the three difficulty levels: easy, medium, and hard. As shown in Figure 4, the robot navigates toward the goal in each



(a)



(b)



(c)

Figure 4: Completed single evaluation episodes in easy (a), medium (b), and hard (c) obstacle scenarios. Each trajectory ends at the goal while successfully avoiding all obstacles

scenario while adapting its path to avoid static obstacles. Figure 4 can be read as follows: the initial point marks the starting end-effector position, the target point indicates the goal, and the obstacle points denote the centers of static obstacles in the workspace; the plotted backbone shows the robot shape along the executed trajectory. In the easy scenario, the trajectory remains relatively direct with minimal curvature adjustments. In the medium scenario, the robot shows more pronounced bending to avoid dual obstacles. The hard scenario presents closely clustered obstacles, requiring complex trajectory modulation and longer paths, which the agent manages effectively without collisions. We note that obstacle avoidance in the current formulation is driven by a reward term based on the end-effector position and obstacle distances. Consequently, the learned behavior primarily ensures that the distal tip avoids collisions, while intermediate points along the backbone are not explicitly constrained; this can make the hard-scenario visualization appear as if only part of the robot is "avoiding" obstacles. Extending collision checking and reward shaping to incorporate the full robot body (e.g., multiple points along the backbone) is a natural direction for future work.

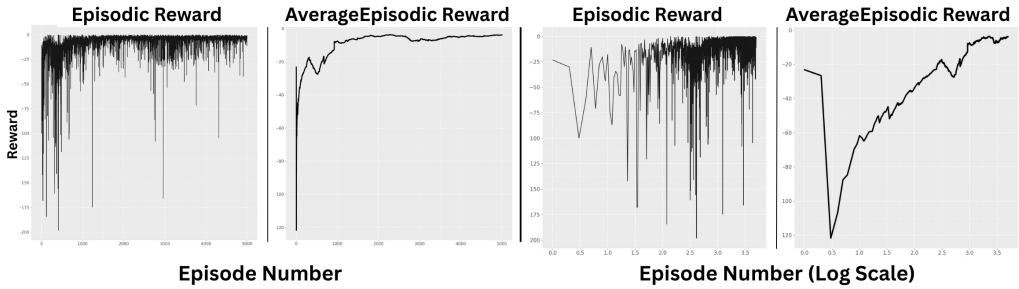
Table 5 summarizes the performance of the trained policy in different obstacle scenarios based on the evaluation metrics defined above. The results are averaged over multiple test runs.

Table 5: Evaluation results across different static obstacle scenarios

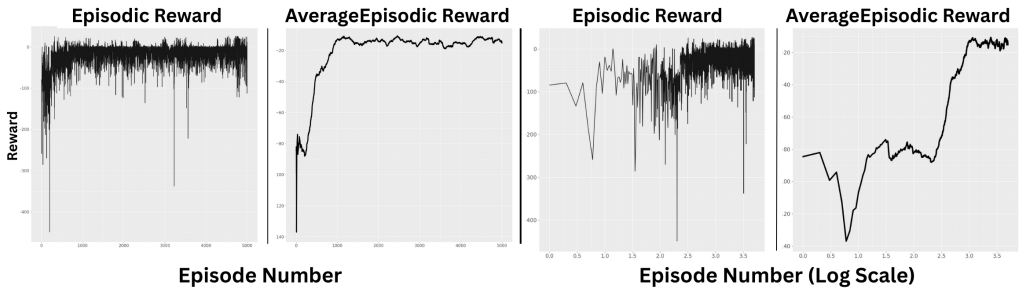
Scenario	Success rate (%)	Effectiveness score	Training time (h)	Converged episode
Easy	94	215.2	2.9	900
Medium	89	283.7	3.5	1000
Hard	79	351.9	4.1	1500

The success rate remains high across all scenarios, with a noticeable drop in more complex environments due to the increased navigation challenge. The effectiveness score increases (i.e. performance worsens) with scenario complexity, reflecting longer paths and tighter maneuvering constraints. Despite the complexity, all policies converge within a reasonable number of episodes, and the easy case stabilizes fastest.

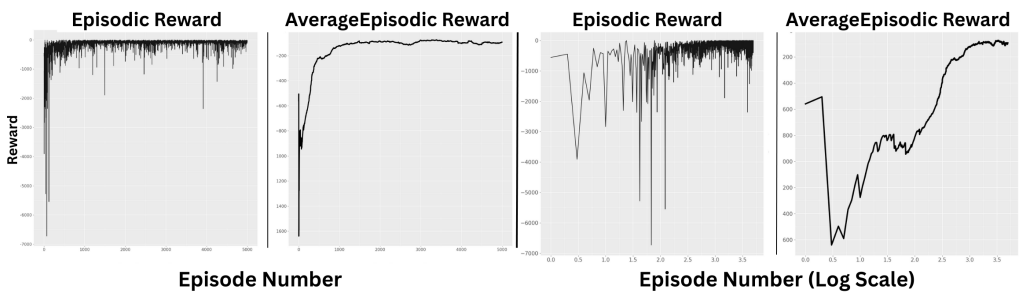
Figure 5 displays twelve graphs showing the progression of the agent's reward throughout training. These plots illustrate the convergence dynamics of the RL agent under varying complexity. In all cases, the reward growth is steady, but convergence is slower in medium and hard scenarios due to higher exploration demands. The log-scaled plots highlight early learning volatility and gradual stabilization in later stages of training.



(a)



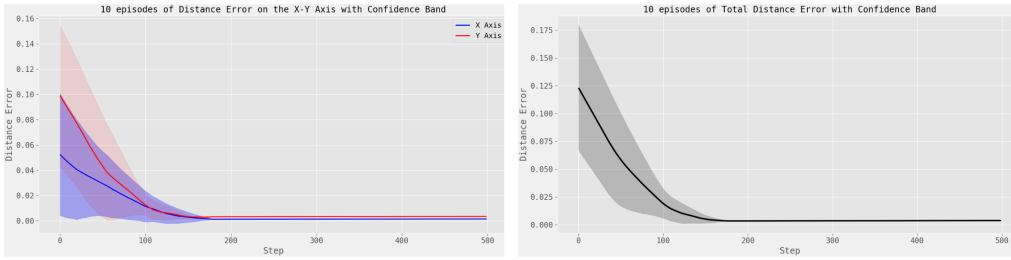
(b)



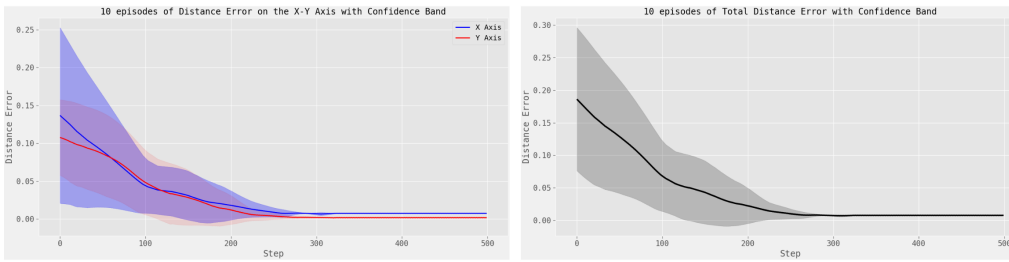
(c)

Figure 5: Episodic and average episodic reward curves (and their logarithmic scales) for easy (a), medium (b), and hard (c) scenarios

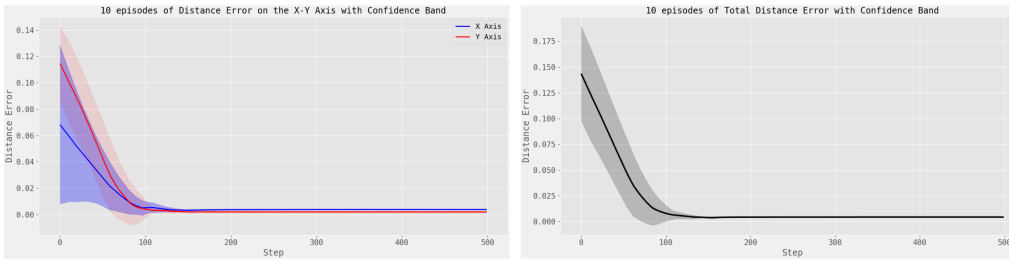
To evaluate trajectory precision, we measured the positional error of the continuum robot over ten test episodes for each scenario. Figure 6 shows the total error magnitude as well as separate error values in the X and Y axes. The results show that the total error remains low across all runs, though slightly higher in obstacle-rich environments. Most of the variation occurs along the X-axis, particularly in the hard scenario, which involves lateral avoidance maneuvers. The Y-axis error is relatively stable, indicating consistent goal progression along the principal trajectory. To better understand the robot's adaptive motion, we analyze



(a)



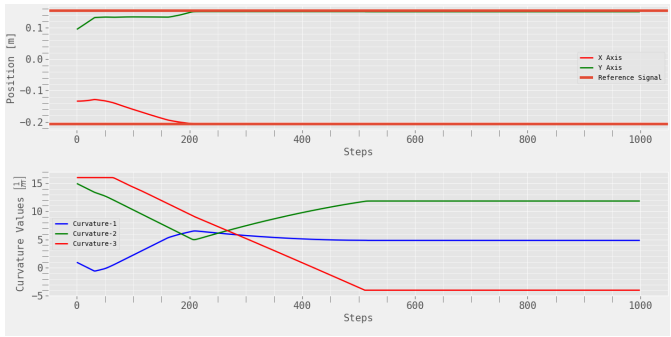
(b)



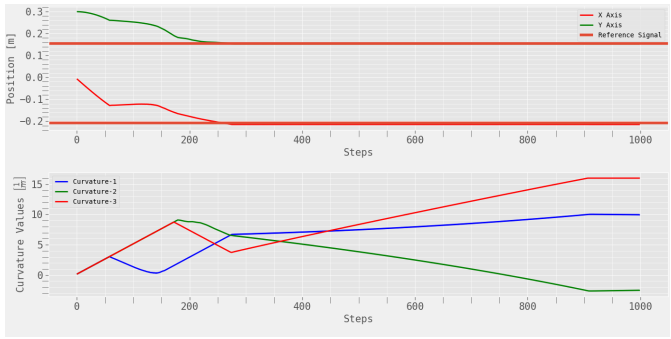
(c)

Figure 6: Total, X-axis, and Y-axis errors of the continuum robot across ten test episodes: easy (a), medium (b) and hard (c)

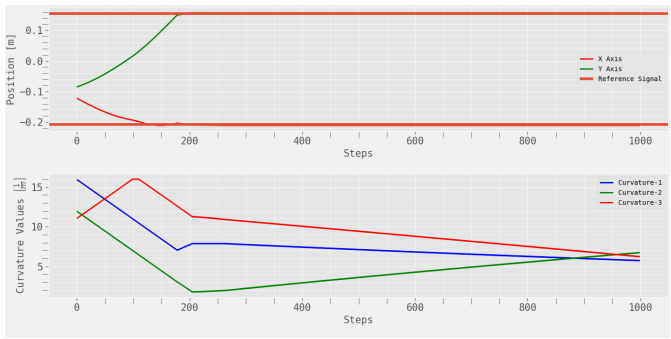
the full trajectory and corresponding curvature values for a single representative episode in each scenario. Figure 7 presents the robot's end-effector position (in X and Y coordinates) and curvature values of each section at every time step. The figure shows how the robot adjusts its curvature in response to obstacle layout. In the easy scenario, changes in curvature are minimal, while in the medium and hard cases, curvature modulation becomes more dynamic. These changes allow the robot to maintain safety while tracking efficient paths. The combined results



(a)



(b)



(c)

Figure 7: Position (X, Y) and curvature values for a single test episode across robot sections 1, 2, and 3 for each scenario: easy (a), medium (b) and hard (c)

demonstrate that the proposed reinforcement learning framework effectively generalizes to increasingly complex obstacle-rich environments. The DDPG algorithm, aided by carefully designed reward functions and network architectures, enables the robot to learn curvature-based control strategies that balance safety and efficiency. While path efficiency slightly declines in more complex settings,

the agent maintains high success rates and convergence consistency, validating the scalability of the approach. Further investigation into online adaptation, dynamic obstacle response, and real-world deployment is encouraged to extend these findings to broader robotics applications.

6. Conclusions

In this study, we presented a reinforcement learning-based control framework for a planar three-section continuum robot operating in environments with static obstacles. Building on previous work in continuum robot kinematics and learning-based control, we extended the system's capabilities by incorporating obstacle avoidance mechanisms directly into the reinforcement learning framework.

The results of our experiments demonstrate that reinforcement learning, specifically using the DDPG algorithm, is capable of learning effective control policies for curvature-based actuation in complex environments. Through extensive training and evaluation in a custom simulation environment with varying levels of obstacle difficulty, the robot consistently learned to avoid static obstacles while navigating toward the goal positions. The key to this success was the careful design of the reward function, which balanced goal-seeking behavior with smooth and safe maneuvering. Furthermore, experimentation with neural network architecture revealed that both the number of layers and the activation functions played a significant role in stabilizing the training and improving generalization between scenarios. The trained models performed reliably even as the complexity of the obstacle layout increased, supporting the robustness of the proposed approach.

While the results are promising, several limitations of the current framework point to potential avenues for future research. First, the current implementation is limited to static environments; the inclusion of dynamic obstacles introduces additional temporal constraints and requires predictive capabilities that are not yet modeled. Future work will focus on extending the system to handle dynamic environments where obstacles may move unpredictably. We will also investigate the use of other reinforcement learning algorithms, like Proximal Policy Optimization (PPO) or Robust Policy Optimization (RPO).

References

[1] G. Robinson and J. Davies: Continuum robots – a state of the art. *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, **4**, Detroit, MI, USA, (1999), 2849–2854. DOI: 10.1109/ROBOT.1999.774029

[2] I.D. Walker: Continuous backbone "continuum" robot manipulators. *International Scholary Research Notices*, **2013**, 1–19. DOI: 10.5402/2013/726506

[3] J. Burgner-Kahrs, D.C. Rucker and H. Choset: Continuum robots for medical applications: A survey. *IEEE Transactions on Robotics*, **31**(6), (2015), 1261–1280. DOI: 10.1109/TRO.2015.2489500

[4] M.T. Chikhaoui and J. Burgner-Kahrs: Control of continuum robots for medical applications: State of the art. *16th International Conference on New Actuators*, Bremen, Germany, (2018), 1–11.

[5] P.E. Dupont, N. Simaan, H. Choset and C. Rucker: Continuum robots for medical interventions. *Proceedings of the IEEE*, **110**(7), (2022), 847–870. DOI: 10.1109/JPROC.2022.3141338

[6] B.S. Arjun, A.A. Krishnan and H.J. Pandya: MRI-compatible patient-specific continuum robots using parametric modelling. *Proceedings of the 19th International Conference on Body Sensor Networks*, Boston, MA, USA, (2023), 1–4, DOI: 10.1109/BSN58485.2023.10331197

[7] G.Z. Meng, G.M. Yuan, Z. Liu and J. Zhang: Forward and inverse kinematic of continuum robot for search and rescue. *Advanced Materials Research*, **712-715**, (2013), 2290–2295. DOI: 10.4028/www.scientific.net/AMR.712-715.2290

[8] I.D. Walker and C. University: Use of continuum robots for remote inspection operations. *Proceedings of the 2017 Computing Conference*, London, UK, (2017), 1382–1385. DOI: 10.1109/SAI.2017.8252274

[9] Y. Sapmaz, S. Dilibal, O. Yilmaz and A.R. Sapmaz: Nickel-titanium SMA springs actuated bioinspired soft robot for pipeline inspections. *Proceedings of the 3rd International Congress on Human-Computer Interaction, Optimization and Robotic Applications*, Ankara, Turkey, (2021), 1–5. DOI: 10.1109/HORA52670.2021.9461361

[10] C. Della Santina, C. Duriez and D. Rus: Model-based control of soft robots: A survey of the state of the art and open challenges. *IEEE Control Systems*, **43**(3), (2023), 30–65. DOI: 10.1109/MCS.2023.3253419

[11] R. Morimoto, S. Nishikawa, R. Niiyama and Y. Kuniyoshi: Model-free reinforcement learning with ensemble for a soft continuum robot arm. *Proceedings of the IEEE 4th International Conference on Soft Robotics*, New Haven, USA, (2021), 141–148. DOI: 10.1109/RoboSoft51838.2021.9479340

[12] T.C. Kargin and J. Kołota: A reinforcement learning approach for continuum robot control. *Journal of Intelligent & Robotic Systems*, **109**(4), (2023). DOI: 10.1007/s10846-023-02003-0

[13] S. Djeffal, M.R. Morakchi, A. Ghoul and T.C. Kargin: DDPG-based reinforcement learning for controlling a spatial three-section continuum robot. *Franklin Open*, **6**, (2024), DOI: 10.1016/j.fraope.2024.100077

[14] J. Kołota and T.C. Kargin: Comparison of various reinforcement learning environments in the context of continuum robot control. *Applied Sciences*, **13**(16), (2023). DOI: 10.3390/app13169153

[15] T. Lindner and A. Milecki: Reinforcement learning-based algorithm to avoid obstacles by the anthropomorphic robotic arm. *Applied Sciences*, **12**(13), (2022). DOI: 10.3390/app12136629

[16] M.W. Hannan and I.D. Walker: Kinematics and the implementation of an elephant's trunk manipulator and other continuum style robots. *Journal of Robotic Systems*, **20**(2), (2003), 45–63. DOI: 10.1002/rob.10070

[17] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver and D. Wierstra: Continuous control with deep reinforcement learning. *arXiv*:1509.02971, (2019). DOI: 10.48550/arXiv.1509.02971

[18] V. Mnih *et al.*: Human-level control through deep reinforcement learning. *Nature*, **518**(7540), (2015), 529–533. DOI: 10.1038/nature14236

[19] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra and M. Riedmiller: Deterministic policy gradient algorithms. *Proceedings of the 31st International Conference on Machine Learning*, E.P. Xing and T. Jebara (Eds.), ser. *Proceedings of Machine Learning Research*, **32**, Bejing, China, (2014), 387–395. [Online]. Available: https://proceedings.mlr.press/v32/silver14.html

[20] F. Chollet *et al.*: Keras. https://keras.io (2015).