A snake-inspired path planning algorithm based on reinforcement learning and self-motion for hyper-redundant manipulators

Journal Title
XX(X):1–9
©The Author(s) 2016
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Abstract

Redundant manipulators are flexible enough to adapt to complex environments, but their controller is also required to be specific for their extra degrees of freedom. Inspired by the morphology of snakes, we propose a path planning algorithm named Swinging Search and Crawling Control, which allows the snake-like redundant manipulators to explore in complex pipeline environments without collision. The proposed algorithm consists of the Swinging Search and the Crawling Control. In Swinging Search, a collision-free manipulator configuration that of the end-effector in the target point is found by applying reinforcement learning to self-motion, instead of designing joint motion. The self-motion narrows the search space to the null space, and the reinforcement learning makes the algorithm use the information of the environment, instead of blindly searching. Then in Crawling Control, the manipulator is controlled to crawl to the target point like a snake along the collision-free configuration. It only needs to search for a collision-free configuration for the manipulator, instead of searching collision-free configurations throughout the process of path planning. Simulation experiments show that the algorithm can complete path planning tasks of hyper-redundant manipulators in complex environments. The 16 DoFs and 24 DoFs manipulators can achieve 83.3% and 96.7% success rates in the pipe, respectively. In the concentric pipe, the 24 DoFs manipulator has a success rate of 96.1%.

Keywords

redundant manipulator, path planning, reinforcement learning, self-motion, crawling control, swinging search

Introduction

In recent decades, more and more researchers have been involved in the study of redundant manipulators to explore complex narrow environments, such as the natural gas pipes, sewer pipes, and tailpipes of airplanes. A manipulator is termed kinematically redundant when it possesses more degrees of freedom (DoFs) than it is needed to execute a given task.[1] The extra DoFs allow the manipulator to perform complex tasks, making path planning a challenge though.

To address the path planning of redundant manipulators, some early researchers proposed geometric methods. Chirikjian et al.[2] proposed the backbone curve approach for hyper-redundant robot kinematics. Later, a shape control geometric analysis method[3] was proposed by Mochiyama H et al. to control the whole shape of a manipulator. But such algorithms are computationally intensive. To reduce computational complexity, Samer Yahya et al.[4] proposed geometric constraints that limit the angles between the adjacent links to be equal. However, the flexibility of the manipulators is also limited due to the introduced constraints.

Other researchers proposed the method based on optimization theory, in which obstacle avoidance can be achieved by optimizing the objective distance function between the manipulators and the obstacles. Matt Zucker et al.[5] proposed the Covariant Hamiltonian Optimization for Motion Planning (CHOMP) method which can quickly converge with generating a smooth collision-free trajectory. Jingdong Zhao et al.[6] applied the ant colony algorithm to plan the collision-free optimal path of the end-effector. Thomas Collins et al.[7] proposed an optimization framework Path Planning with Swarm Optimization (PASO) to efficiently calculate the approximate solution of the collision-free path planning. These methods require calculating the distance from the manipulator to the obstacles, and therefore also require modeling the surface of the obstacles, which is sometimes difficult to complete.

In addition, the random sampling methods are efficient path planning algorithms for redundant manipulators. The Probabilistic Roadmaps (PRM) method[8] proposed by Kavraki et al. randomly samples in the configuration space to search for collision-free configurations.[9] The sampling results are stored in a probabilistic roadmap. For any given start and goal configurations of the manipulator, the motion planning can be transformed into the problem of connecting the two corresponding nodes in the roadmap.[10] Compared with the optimization theory methods, the random sampling methods do not require figuring out the objective distance function, which reduces the computational complexity and improves the algorithm applicability. However, as the DoFs

Corresponding author:

Email:

increases, methods of this kind are also computationally complex.

With the development of artificial intelligence, Reinforcement Learning (RL) has also been applied to the path planning of redundant manipulators. Hua et al.[11] applied the RL method to train the agent to plan the end-effector motion and self-motion[12 13 14] separately. Combining the gradient projection method[15] decouples the motion of redundant manipulators into end-effector motion and self-motion. In this algorithm, the agent has to learn the entire planning process without making full use of the nature of self-motion, which still causes great computational difficulties.

In nature, snakes crawl along the path of their heads during crawling. Their bodies will not collide with the environment, as long as the heads can pass smoothly. Inspired by this biological phenomenon, this paper proposes a path planning algorithm named Swinging Search and Crawling Control (SSCC) for snake-like redundant manipulators. This algorithm consists of the Swinging Search and the Crawling Control, allowing the redundant manipulators with repetitive modules to explore the complex pipeline environments.

By self-motion, the Swinging Search algorithm generates collision-free configurations as the directive configurations, in which the end-effector is in the target point. And then the manipulator can crawl into the pipe along the directive configurations by the Crawling Control. During crawling, each modular link repeats the motion of the foremost module.

The major contributions of this paper are as follows.

(i) Inspired by the biological characteristics of snakes, this paper innovatively proposes the SSCC algorithm for snake-like redundant manipulators to be adapted to narrow environments such as pipes.
(ii) The SSCC algorithm has low computational complexity, since only one directive configuration needs to be searched.
(iii) The SSCC algorithm has good expansibility and adaptability for modular snake-like redundant manipulators with n universal joints.

Snake-like Redundant Manipulators

Mechanical Structure Design

In this paper, to be analogous to snakes in morphology, the modular redundant manipulators with n universal joints are designed. The manipulators are $n \times \mathrm{U}$ open chains[16]. Each module consists of a universal joint and a fixed-length rigid body link, as shown in Figure 1. Modularity allows the DoFs of the manipulators to be easily expanded by increasing the number of modules. The Denavit-Hartenberg (D-H) parameters of the snake-like redundant manipulators are shown in Table 1.

Forward Kinematics

By the D-H parameters, the homogeneous transformation matrices for kinematics are expressed as:

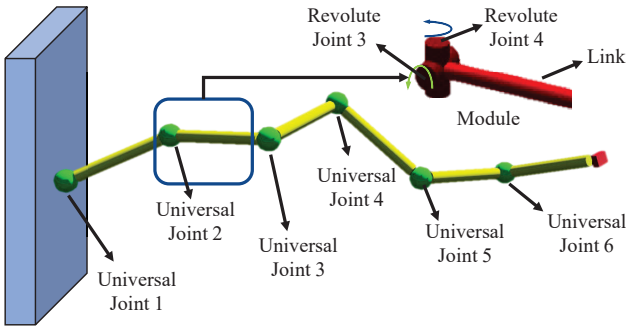


Figure 1. The structure of the general snake-like redundant manipulator with 12 joints.

Table 1. The D-H parameters of the snake-like redundant manipulators.

Joint n	α_i(deg)	a_i(m)	d_i(m)	θ_i(deg)
1	90	0	0	0
2	−90	l	0	0
3	90	0	0	0
4	−90	l	0	0
⋮	⋮	⋮	⋮	⋮
$2n-1$	90	0	0	0
$2n$	−90	l	0	0

$$\boldsymbol{T}_i^{i+1} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad i = 0, 1, 2, \cdots, 2n-1 \tag{1}$$

In forward kinematics, the pose of the end-effector $\boldsymbol{x} = \begin{bmatrix} x & y & z & \alpha & \beta & \gamma \end{bmatrix}^{\mathrm{T}}$, in which the first three and the last three variables represent the position and the orientation of the manipulator respectively. The pose $\boldsymbol{x}$ can be calculated as follows.

$$\boldsymbol{x} = f_{\text{fkine}}(\boldsymbol{q}) = \prod_{i=1}^{2n} \boldsymbol{T}_i^{i+1} \tag{2}$$

Where f_{fkine} is the function of the forward kinematics, $\boldsymbol{q} = \begin{bmatrix} \theta_1 & \theta_2 & \dots & \theta_{2n} \end{bmatrix}^{\mathrm{T}}$ is the joint motion vector of the manipulators in the configuration space.

Self-Motion

The i-th column of the Jacobian Matrix $\boldsymbol{j}_i$ can be calculated by the vector product method as follows.

$$\boldsymbol{j}_i = \begin{bmatrix} \boldsymbol{z}_i \times {}^i\boldsymbol{p}_E^0 \\ \boldsymbol{z}_i \end{bmatrix} \tag{3}$$

Where $\boldsymbol{z}_i$ is the i-th axis vector; ${}^i\boldsymbol{p}_E^0$ is the vector from the end-effector to the i-th joint. Then the Jacobian Matrix can be represented as:

$$\boldsymbol{J} = \begin{bmatrix} \boldsymbol{j}_1 & \boldsymbol{j}_2 & \cdots & \boldsymbol{j}_{2n} \end{bmatrix} \tag{4}$$

The pseudo inverse of Jacobian Matrix is defined as:

$$\boldsymbol{J}^{\dagger} = \boldsymbol{J}^{\mathrm{T}}(\boldsymbol{J}\boldsymbol{J}^{\mathrm{T}})^{-1} \tag{5}$$

With the Jacobian Matrix, the end-effector velocity vector $\dot{\boldsymbol{x}}$ can be expressed as a function of the joint velocity vector $\dot{\boldsymbol{q}}$:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{w} \end{bmatrix} = \boldsymbol{J}\dot{\boldsymbol{q}} \tag{6}$$

Where $\boldsymbol{v}$ and $\boldsymbol{w}$ are the linear velocity vector and the angular velocity vector of the end-effector, respectively.

For control purposes, it is common to solve $\dot{\boldsymbol{q}}$ in Eq.(6) by specifying the desired $\dot{\boldsymbol{x}}$. Since the manipulators are redundant, the dimension of $\dot{\boldsymbol{q}}$ is higher than the dimension of $\dot{\boldsymbol{x}}$. In this way, Eq.(6) can be regarded as a set of underdetermined equations, and the solution of which can be formalized as follows.

$$\begin{aligned} \dot{\boldsymbol{q}} &= \dot{\boldsymbol{q}}_S + \dot{\boldsymbol{q}}_N \\ &= \boldsymbol{J}^{\dagger}\dot{\boldsymbol{x}} + (\boldsymbol{I} - \boldsymbol{J}^{\dagger}\boldsymbol{J})\boldsymbol{m} \end{aligned} \tag{7}$$

Where the term $\boldsymbol{J}^{\dagger}\dot{\boldsymbol{x}}$ is the minimum norm solution $\dot{\boldsymbol{q}}_S$ of Eq.(6); The term $(\boldsymbol{I} - \boldsymbol{J}^{\dagger}\boldsymbol{J})\boldsymbol{m}$ represents the homogeneous solutions $\dot{\boldsymbol{q}}_N$ of $\boldsymbol{J}\dot{\boldsymbol{q}} = 0$:

$$\begin{aligned} \boldsymbol{J}\dot{\boldsymbol{q}}_N &= \boldsymbol{J}(\boldsymbol{I} - \boldsymbol{J}^{\dagger}\boldsymbol{J})\boldsymbol{m} \\ &= (\boldsymbol{J} - \boldsymbol{J}\boldsymbol{J}^{\dagger}\boldsymbol{J})\boldsymbol{m} \\ &= \boldsymbol{0} \end{aligned} \tag{8}$$

All the homogeneous solutions form the null space of $\boldsymbol{J}$. In the null space, redundant manipulators can change their configurations without changing the pose of the end-effector which is named the self-motion. The self-motion can be used in the gradient projection method[15] to realize the obstacle avoidance of redundant manipulators:

$$\begin{aligned} \boldsymbol{m} &= k\nabla \boldsymbol{H}(\boldsymbol{q}) \\ &= k\left[\frac{\partial \boldsymbol{H}(\boldsymbol{q})}{\partial q_1} \quad \frac{\partial \boldsymbol{H}(\boldsymbol{q})}{\partial q_2} \quad \cdots \quad \frac{\partial \boldsymbol{H}(\boldsymbol{q})}{\partial q_{2n}}\right]^{\mathrm{T}} \end{aligned} \tag{9}$$

Where k is a scale factor; $\boldsymbol{H}$ is the objective function which represents the distance between the manipulator and obstacles; $\nabla \boldsymbol{H}$ is the gradient of $\boldsymbol{H}$. Then Eq.(7) can be derived as follows.

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger}\dot{\boldsymbol{x}} + k(\boldsymbol{I} - \boldsymbol{J}^{\dagger}\boldsymbol{J})\nabla \boldsymbol{H} \tag{10}$$

By specifying $\dot{\boldsymbol{x}}$ and calculating $\nabla \boldsymbol{H}$, the control of redundant manipulators can be achieved by the following iterative equation:

$$\boldsymbol{q}(t + \Delta t) = \boldsymbol{q}(t) + \dot{\boldsymbol{q}}\Delta t \tag{11}$$

Where $\boldsymbol{q}(t)$ is the current joint positions, and $\boldsymbol{q}(t + \Delta t)$ is the joint positions after an iteration in the next moment.

Framework of SSCC

In this paper, a path planning algorithm named Swinging Search and Crawling Control is proposed. Through this algorithm, the modular redundant manipulators can reach into the complex pipe environment without collision, allowing the end-effector to achieve the goal pose.

SSCC algorithm consists of two units, the Swinging Search and the Crawling Control. The Swinging Search generates collision-free configurations by applying RL to self-motion. The generated configurations are named directive configurations, in which the end-effector has reached the goal pose. In the Crawling Control, the controlled redundant manipulator crawls into the pipe along the directive configuration like a snake. The flow chart of SSCC is shown in Figure 2.

Since the length of the rigid link in the module cannot be ignored, the manipulator will inevitably deviate from the directive configuration during the Crawling Process, which may also cause the manipulator to collide. The Crawling Process is deterministic, which means that the degree of deviation can be calculated based on the directive configuration. Therefore, we propose the maximum deviation distance to evaluate the directive configuration to train the agent in RL, so that the Swinging Search can generate the directive configuration with a smaller maximum deviation distance to reduce the possibility of collision during crawling.

Both the Swinging Search and the Crawling Control include collision detection. In the Swinging Search, collision detection is used to determine whether to terminate the search. In the Crawling Control, if a collision is detected then the directive configuration is invalid, and the Swinging Search should regenerate a new directive configuration.

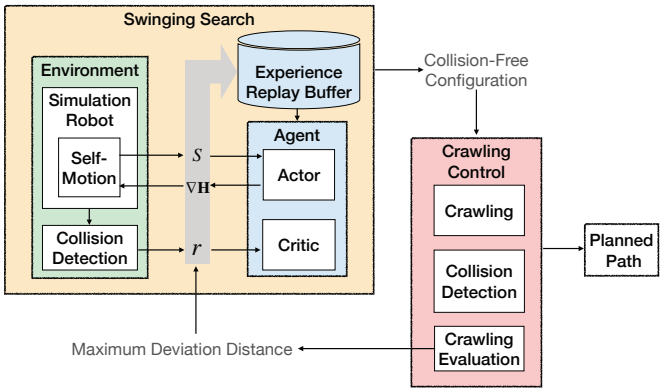


Figure 2. The framework of Swinging Search and Crawling Control.

In Figure 2, $\boldsymbol{s}$ is the state of the environment fed back to the agent; $\nabla \boldsymbol{H}$ is the action the agent decides to perform, which is the gradient in Eq.(9) of the objective function that represents the distance between the obstacle and the manipulator; r is the reward obtained by the agent. The above parameters will be described in detail in the State-Action Design section.

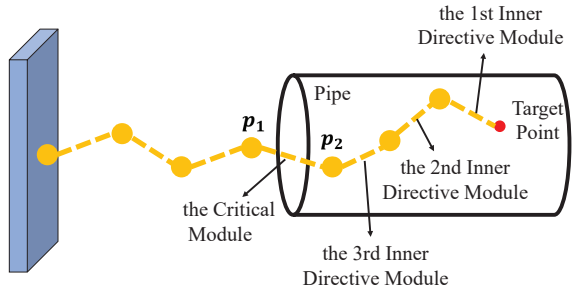
Crawling Control

Crawling Process

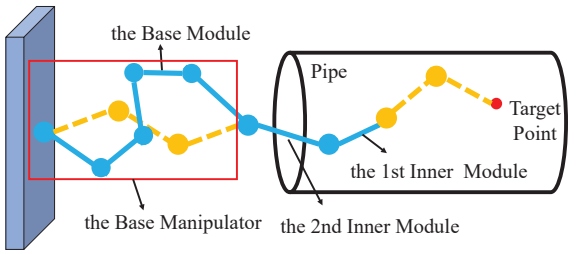
The Crawling Process refers to the process in which the manipulator $\mathbf{M}$ is controlled to move forward along the directive configuration $\mathbf{M}(\boldsymbol{q}_d)$. Through this control, the end-effector of the manipulator can reach the specified pose.

To illustrate the Crawling Process, several definitions are made as follows, and structures are shown in Figure 3.

(i) The modules of $\mathbf{M}(\boldsymbol{q}_d)$ that are completely or partially inside the pipe are called the inner directive modules.
(ii) The inner directive module at the entrance of the pipe is called the critical module. The two ends of the critical module are denoted as $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$.
(iii) Similarly, the modules of the crawling manipulator $\mathbf{M}$ that are completely or partially inside the pipe are called the inner modules.
(iv) The remaining modules of $\mathbf{M}$ are called the base modules. All the base modules form the base manipulator $\mathbf{M}_{\text{base}}$.



(a) the Directive Configuration



(b) the Crawling Manipulator

Figure 3. The structures of the directive configuration and the crawling manipulator. The solid blue polygonal line represents the crawling manipulator. The dashed orange polygonal line represents the directive configuration.

In the Crawling Process, the base modules and the inner modules of $\mathbf{M}$ are controlled separately, as shown in Figure 4. By solving the inverse kinematics, the end of $\mathbf{M}_{\text{base}}$ is controlled to move from $\boldsymbol{p}_1$ to $\boldsymbol{p}_2$ within time T, while keeping the orientation of the foremost module the same as that of the critical module. At the same time, the universal joint angles of the inner modules are concurrently changed to the corresponding joint angles of their next inner directive module. In this way, each inner module will repeat the motion of the foremost inner module, like the movement of a snake. In detail, suppose the number of modules to be inserted is N and the number of modules inserted is i, then the angles of the j-th inner module should be adjusted to the angles of the $(N-i+j-1)$-th inner directive module.

As the modules move forward, the base modules in the front will become the inner modules, and the control over them will be changed, too.

For the next round of insertion, the new $\mathbf{M}_{\text{base}}$ should adjust the orientation of its end, and at the same time, the newly inserted module should keep overlapping the critical module. This can be achieved through pose interpolation of $\mathbf{M}_{\text{base}}$, while planning the first inner module to move to the last inner directive module. The Crawling Process algorithm is summarized in Algorithm 1.

Algorithm 1 Crawling Process

Input: a directive configuration $\mathbf{M}(\boldsymbol{q}_d)$
Output: planned motion $\boldsymbol{q}$(t)

1: Find the critical module from the directive configuration.
2: Count the number N of modules to be inserted.
3: Control the base manipulator $\mathbf{M}_{\text{base}}$ to reach $\boldsymbol{p}_1$.
4: **for** $i = 0$ to N **do**
5: Adjust the orientation of the end of $\mathbf{M}_{\text{base}}$, while keeping the adjacent inner module still.
6: Insert the foremost module of $\mathbf{M}_{\text{base}}$ to reach $\boldsymbol{p}_2$, while adjusting the angles of the inner modules.
7: Redefine $\mathbf{M}_{\text{base}}$ and the inner modules.
8: **end for**

Evaluation of Directive Configurations

The crawling manipulator $\mathbf{M}$ will inevitably deviate from the directive configuration in the Crawling Process. Therefore, we introduce the maximum deviation distance D to evaluate the deviation.

For a specific directive configuration, the Crawling Process will correspondingly generate a planned motion. D is defined as the maximum distance between the inner modules and their corresponding inner directive modules at any moment in the planned motion.

Since each inner module will repeat the motion of the frontmost inner module, it is only necessary to analyze the motion of the foremost inner module to calculate the maximum deviation distance.

Sampling e points uniformly on the foremost inner module and calculating the distance from each point to the $(N-i-1)$-th and the $(N-i)$-th inner directive module, as shown in Figure 5. The distance between the foremost inner module and its corresponding module is defined as the maximum of these distances.

$$d(t,i,j) = d(\boldsymbol{s}_j(\boldsymbol{q}(t)), \mathbf{M}_{N-i}(\boldsymbol{q}_d)) \tag{12}$$

Where the $\boldsymbol{s}_j(\boldsymbol{q}(t))$ indicates the j-th sampling point in the configuration of the $\mathbf{M}$ at time t; the $\mathbf{M}_{N-i}(\boldsymbol{q}_d)$ indicates the $(N-i)$-th inner directive module; and the d indicates the distance function which calculates the distance between the sampling points and the link of the inner directive module. Then the maximum deviation distance can be calculated as follows.

$$D = \max_{t, 0\le i\le N, 1\le j\le e} \{d(t,i,j), d(t,i+1,j)\}. \tag{13}$$

The algorithm of the entire Crawling Control is summarized in Algorithm 2.

Swinging Search

By applying reinforcement learning to self-motion, the Swinging Search generates collision-free directive configurations, in which the end-effector has reached the goal pose.

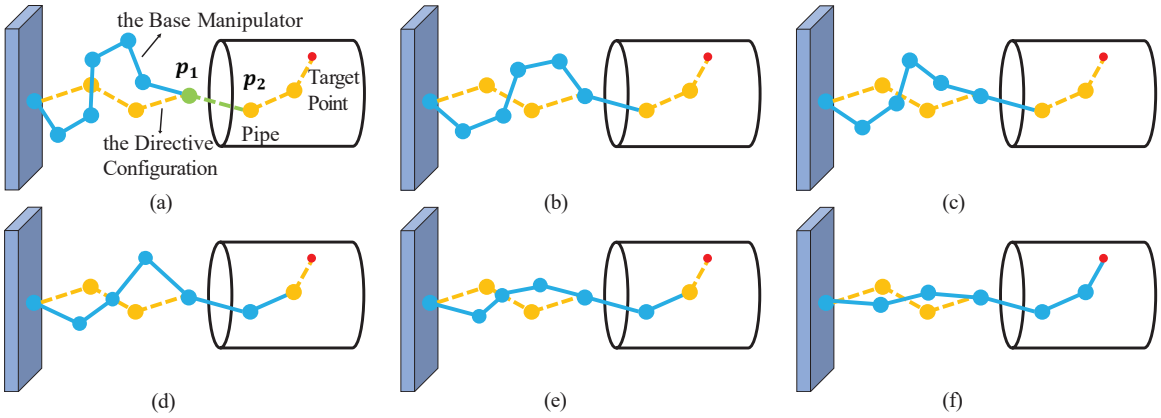


Figure 4. The steps of the Crawling Process.

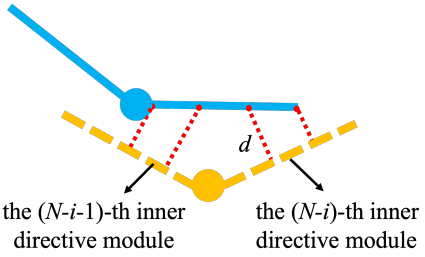


Figure 5. The deviation in the Crawling Process. The solid blue polygonal line represents the crawling manipulator. The dashed orange polygonal line represents the directive configuration. The dashed red line represents the deviation distance from sampling points on the manipulator to the configuration.

Algorithm 2 Crawling Control

Input: a directive configuration $\mathbf{M}(\boldsymbol{q}_d)$

Output: planned motion $\boldsymbol{q}$(t); the maximum deviation distance D; $\text{flag}_{\text{collided}}$

1: $\boldsymbol{q}(t) = \text{CrawlingProcess}(\mathbf{M}(\boldsymbol{q}_d))$
2: $D = \text{CrawlingEvaluation}(\mathbf{M}(\boldsymbol{q}_d), \boldsymbol{q}(t))$
3: $\text{flag}_{\text{collided}} = \text{CollisionDetection}(\boldsymbol{q}(t))$

In this way, the search can make use of the environment information, so as not to explore blindly. Applying the self-motion also reduces the range of configurations that need to be searched.

State-Action Design

The design of state and action has a great influence on the convergence of RL, so the designed state and action should reflect the relationship between the manipulator and the environment.

To achieve self-motion to search for directive configurations, $\dot{\boldsymbol{q}}$ derived from Eq.(10) is used to control the manipulator:

$$\dot{\boldsymbol{q}} = (\boldsymbol{I} - \boldsymbol{J}^{\dagger}\boldsymbol{J})\boldsymbol{m} \tag{14}$$

Intuitively, we design $\boldsymbol{m}$ as the action $\boldsymbol{a}_t$:

$$\boldsymbol{a}_t = \boldsymbol{m} = \begin{bmatrix} m_1 & m_2 & \dots & m_{2n} \end{bmatrix}^{\mathrm{T}} \tag{15}$$

And the state $\boldsymbol{s}_t$ is designed as:

$$\boldsymbol{s}_t = \begin{bmatrix} x & y & z & q_1 & q_2 & \dots & q_{2n} \end{bmatrix}^{\mathrm{T}} \tag{16}$$

Where q_i is the joint angle of the manipulator, and $\begin{bmatrix} x & y & z \end{bmatrix}$ is the coordinate vector of the target position that the end-effector needs to reach.

A well-designed reward function can make the RL algorithm converge quickly. Therefore, a reward function consisting of penalties r'_t and r''_t is proposed. We design the number of modules that collided as a penalty r'_t so that the agent can obtain information about collisions. In this way, the agent tends to make the manipulator avoid obstacles when optimizing, so that it can find directive configurations more quickly.

$$r'_t = \sum_{i=1}^{n} C_i \tag{17}$$

$$C_i = \begin{cases} -1 & \text{if the } i\text{-th module collided} \\ 0 & \text{else} \end{cases} \tag{18}$$

Where reward function r'_t represents the number of modules that collided at time t; C_i is the collision situation of the i-th module. If the i-th module collided with obstacles, the score of the i-th joint is -1, otherwise 0.

In addition, if a directive configuration can be found, the Swinging Search will also receive feedback from the Crawling Control, and the maximum deviation distance will also be used as a penalty r''_t, as follows.

$$r''_t = \begin{cases} -D/l & \text{if a directive configuration is searched} \\ 0 & \text{else} \end{cases} \tag{19}$$

Where D is the maximum deviation distance from the Crawling Control, and l is the length of the manipulator link. Then the r_t is as follows.

$$r_t = \begin{cases} r'_t + w \cdot r''_t & \text{if a directive configuration is searched} \\ r'_t & \text{else} \end{cases} \tag{20}$$

Where the scalar w is a scalar hyperparameter that is set according to the number of steps are in an episode.

Learning by Deep Deterministic Policy Gradient

The snake-like redundant manipulators used in this paper have the characteristics of multiple DoFs and spatial continuity of action, so it is necessary to choose an algorithm that can solve the spatial problem of continuous action. Lillicrap et al. presented a deep deterministic policy gradient (DDPG)[17] algorithm which can operate over continuous action spaces, which is a good choice for this task. The DDPG algorithm uses the Actor-Critic framework, including the Actor network for selecting actions and the Critic network for evaluating the policy. The update of Actor network adopts the strategy gradient descent method, which is specifically expressed as:

$$\nabla_\theta \boldsymbol{J}(\theta) = \frac{1}{m'} \sum_i \nabla_{a_i} Q(s_i, a_i|\omega) \nabla_\theta \mu(s_i|\theta) \quad (21)$$

Where θ is the Actor network weight parameters is the state, a is the action, Q is the evaluation value, and m' is the number of samples of empirical data. The Critic network uses the mean square error loss function to update the parameters:

$$Loss = \frac{1}{m'} \sum_i (r_i + \gamma Q'(s_{i+1}, a_{i+1}|\omega') - Q(s_i, a_i|\omega))^2 \quad (22)$$

Where ω is the Critic network weight parameter, γ is the reward discount factor.

The DDPG algorithm replicates the Actor network and the Critic network as the target network so that the agent can learn the task strategy stably, and its network weight parameters are expressed as θ' and ω' respectively. The target network greatly enhances the stability of the learning process when the agent is training. The specific update method of the Actor target network is:

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta' \quad (23)$$

Where τ is used to control the update speed of the Actor target network weight θ'.

Use the same method to update the Critic target network parameters ω' :

$$\omega' \leftarrow \tau\omega + (1-\tau)\omega' \quad (24)$$

In addition, the DDPG algorithm uses random noise to increase the exploration ability of the Actor network in the continuous action space to form a policy-map μ':

$$\mu'(s_t) = \mu(s_t|\theta) + N' \quad (25)$$

Where N' is the random process of the noise.

Combined with the crawling algorithm, SSCC uses the feasible solution obtained by the RL algorithm to complete the exploring task. And each task can be used as empirical data to train the model. The RL model will be more convergent and the solution speed will be faster when after many tasks.

The Swinging Search algorithm is summarized in Algorithm 3.

Algorithm 3 Swinging Search

```
Initialize critic network, actor network, target critic network and target actor network.
Initialize replay buffer R.
Initialize the manipulator and the environment.
for episode = 1 to M do
    Initialize a random process for action exploration.
    Receive initial observation state s_1.
    for t = 1 to T do
        Select action a_t according to the current policy and exploration noise.
        Execute action a_t for self-motion and observe new state s_{t+1}.
        Detect collision and receive reward r'_t
        if no collision then
            Output directive configuration q_d at t+1.
            Execute Crawling Control by inputting q_d.
            Evaluate q_d and get a maximum deviation distance D.
            Calculate reward r''_t by D.
            Reward r_t=r'_t+r''_t.
            Store transition (s_t, a_t, r_t, s_{t+1}) in R.
            break
        else
            Reward r_t=r'_t.
            Store the transition (s_t, a_t, r_t, s_{t+1}) in R.
        end if
        Sample a random minibatch transition from R.
        Update the critic, the actor policy and the target networks.
    end for
end for
```

Experiments and Analysis

System Description

The performance of the SSCC algorithm is studied through the simulation experiment on the path planning of the snake-like redundant manipulator. The simulation is verified in two environments, and two manipulators are required to move along the inner directive modules respectively in different environments. The hardware equipment used in all experiments in this paper is a MacBook Pro laptop (CPU M1, RAM 16 GB). All experiments are implemented in the software PyCharm CE, and the environment configuration was Python 3.9.7 and Pytorch 1.8.0. We use a robotics toolbox in python called robots-toolbox-python to model and solve the kinematics of a hyper-redundant manipulator.

Testing Case in the Hollow Pipe

In the first experiment, the hollow pipe is set as the obstacle. Parameters of the hollow pipe environment are shown in Table 2. And two snake-like redundant manipulators with different DoFs and equal total lengths are designed to verify the SSCC method. Parameters of manipulators are shown in Table 3.

We generated 30 target positions randomly in this experiment. And the efficiency of the SSCC is analyzed using directive configurations with different DoF manipulators,

Table 2. The hollow pipe environment parameters.

Pipe Radius	Pipe Length	Distance between Base and Pipe
0.2m	1.4m	1.0m

Table 3. The parameters of two snake-like redundant manipulators.

Parameter	Manipulator 1	Manipulator 2
DoFs	16	24
length of links l	0.3m	0.2m

as shown in Table 4. The criterion for the success of the SSCC algorithm is defined as successfully searching for a set of collision-free configurations within 30 seconds and no collision occurs during the crawling process for at least one directive configuration. Experiments show that the success rate of a 24 DoFs manipulator is higher than that of a 16 DoFs manipulator.

Table 4. Experimental results in the hollow pipe environment.

	Manipulator 1	Manipulator 2
Target points	30	30
Success	25	29
Fail	5	1
Success rate	83.3%	96.7%

For a more detailed elaboration, we randomly selected two samples from the two groups of experiments respectively as the explanation examples.

The target position is at $\begin{bmatrix}1.65 & -0.11 & 0.04\end{bmatrix}$. The two directive configurations of the two manipulators are searched by the Swinging Search algorithm, where the inner directive configuration consist of 3 modules and 5 modules, respectively. These configurations are shown in Figure 6, and the angles are not listed in detail due to the high redundancy of manipulators.

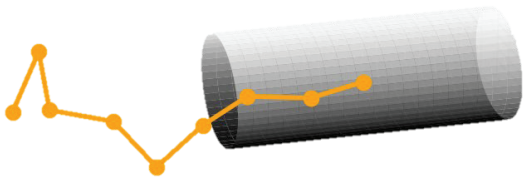
Then, the Crawling Control algorithm is verified in the simulation environment. The manipulators do not collide with the hollow pipe when manipulators enter the hollow pipe.

The joint angles curve of inner modules are shown in Figure 7, and the continuous curves without abrupt change indicate the stability of manipulator motion and verify the effectiveness. The deviation distances are shown in Figure 8. The deviation distances will be close to 0 when the inner manipulators overlap the corresponding inner directive modules. The maximum deviation distance then is calculated as the feedback to the DDPG model.

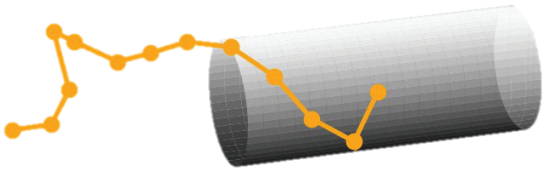
The movement processes of the snake-like redundant manipulators are shown in Figure 9.

Testing Case in the Concentric Pipe

In the second experiment, the path planning using SSCC in the concentric pipe is studied. The parameters of the concentric pipe are shown in Table 5. The controlled 24 DoFs manipulator in this experiment is the same as the second manipulator used in the previous experiment.



(a) The directive configuration of 16 DoFs manipulator in the hollow pipe.



(b) The directive configuration of 24 DoFs manipulator in the hollow pipe.

Figure 6. The directive configurations (the solid orange polygonal line) for 16 DoFs and 24 DoFs manipulators in the hollow pipe.

Table 5. The concentric pipe environment parameters.

Pipe	Pipe Radius	Pipe Length	Distance between Base and Pipe
outer pipe	0.2m	1.4m	1.0m
inner pipe	0.1m	1.4m	1.0m

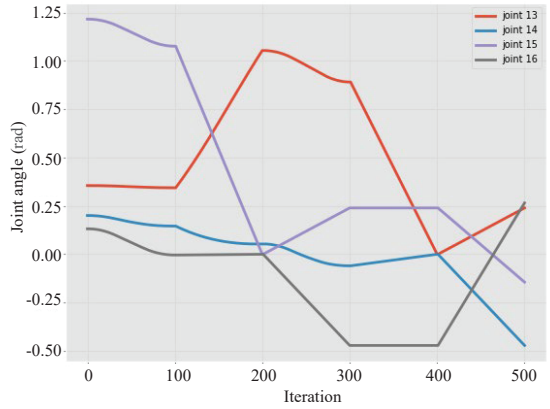
We generated 52 target positions randomly in the concentric pipe. The results are presented as shown in Table 6. The manipulator is requested to reach the target position between the two pipes without collision. The success rate is lower than that of the previous experiment in the pipe, but the SSCC can still complete most of the path planning tasks.

Table 6. Experimental results in the concentric pipe environment.

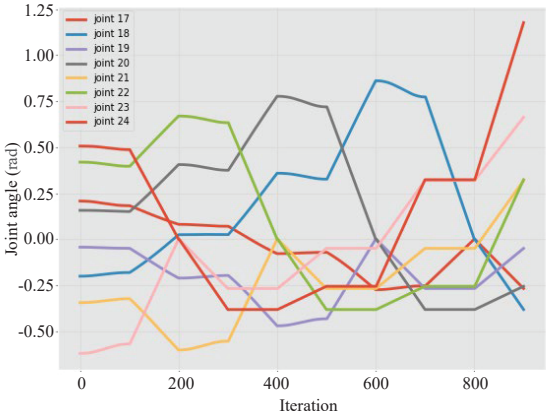
Target points	Success	Fail	Success rate
52	50	2	96.1%

We randomly select a target point $\begin{bmatrix}1.53 & 0.61 & -0.11\end{bmatrix}$ to elaborate the SSCC method in this complex environment. The directive configuration is searched by the Swinging Search, and the inner directive configuration consists of 4 modules, as shown in Figure 10. The Crawling Process of the directive configuration is shown in Figure 11 in the complex environment. Similarly, the continuity of the crawling is verified by the joint angles curve shown in Figure 12. The deviation distance is shown in Figure 13.

In summary, the high efficiency of the SSCC method is proved. In the pipe environment, changing the 16 DoFs



(a) The joint angles curve of the 16 DoFs manipulator.



(b) The joint angles curve of the 24 DoFs manipulator.

Figure 7. The joint angle curves of the inner modules of manipulators in Figure 6. Since the two manipulators have too many DoFs, only the angle curves of joints entering the hollow pipe are shown in the figures.

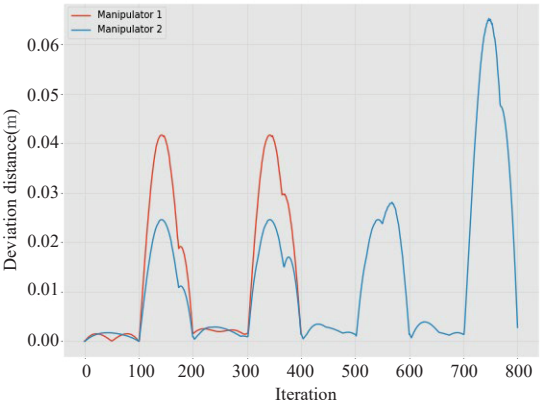


Figure 8. The deviation distances of two manipulators.

manipulator to a 24 DoFs manipulator with a shorter link length can improve search efficiency. In the concentric pipe environment, the hyper-redundant manipulator still has a high success rate for path planning.

Discussion

As the main advantage of the algorithm, we designed the Crawling Control inspired by snake crawling. It makes the Swinging Search only need planning the path of the end-effector instead of planning the body at the same time. What's more, the purpose of the swinging searching algorithm using RL is not to plan the optimal solution, but to speed up the searching of the collision-free configurations as the input of the Crawling Control.

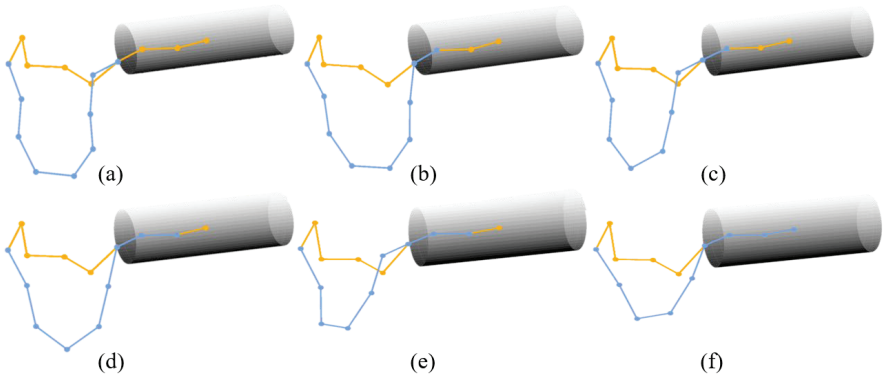
The SSCC still has some limitations that we need to discuss in detail. Because the inner modules will deviate from the directive configuration in the Crawling Process. And the deviation distance is cumulative. This means that the deviation distance may be greater in the more modules extend into the pipe. But this problem will be solved by using the maximum deviation distance as part of the reward function in the RL model. In addition, the Swinging Search only plans a solution and stops, the model is not fully trained. As the future path planning tasks continue to be completed, the RL model will be trained to gradually converge. So that the planner will take less time.

Conclusion

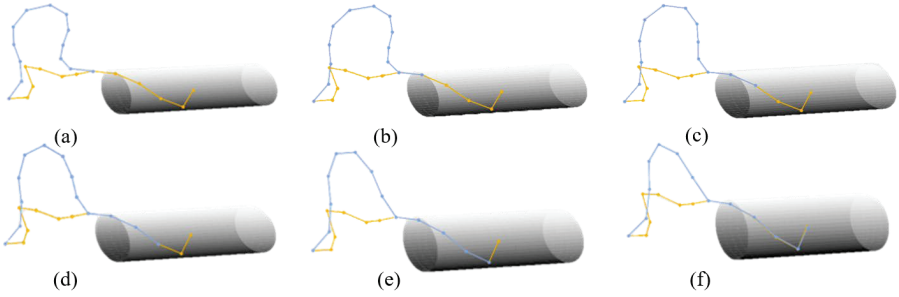
In this paper, inspired by the morphology of snakes, we propose the SSCC algorithm for path planning. The snake-like redundant manipulator can be controlled by using SSCC for exploring complex environments. The snake-like redundant manipulator of the $n \times \mathrm{U}$ open chains is designed to enhance adaptability. The snake's crawling properties are imitated by the manipulator, which makes the manipulator move along the collision-free configuration in a special way. Furthermore, we utilize the maximum deviation distance to optimize the reinforcement learning model, improving the success rate of planning collision-free configurations. Finally, a series of adequate experiments validate that the manipulators can explore complex environments with high success rates by using SSCC.

References

1. Chirikjian GS, Burdick JW. A hyper-redundant manipulator. *IEEE Robotics Automation Magazine* 1994; 1(4): 22-29.
2. Chirikjian GS and Burdick JW. A modal approach to hyper-redundant manipulator kinematics. *IEEE Transactions on Robotics and Automation* 1994; 10(3): 343-354.
3. Mochiyama H. *Shape control of manipulators with hyper degress of freedoms*. PhD Thesis, Japan Advanced Institute of Science and Technology, School of Information Science, Japan, 1998.
4. Yahya S, Moghavvemi M, Yang SS, et al. Motion planning of hyper redundant manipulators based on a new geometrical method. In: *2009 IEEE International Conference on Industrial Technology*, Melbourne, Australia, 13-15 February 2019, pp.1-5. IEEE.
5. Ratliff N, Zucker M, Bagnell JA, et al. CHOMP: Gradient optimization techniques for efficient motion planning. In: *2009 IEEE International Conference on Robotics and Automation*, Kobe, Japan, 1217 May 2009, pp.489-494. IEEE.
6. Zhao J, Zhao L, Liu H. Motion planning of hyper-redundant manipulators based on ant colony optimization. In: *2016*



(a) The Crawling Control of the 16 DoFs manipulator.



(b) The Crawling Control of the 24 DoFs manipulator.

Figure 9. The Crawling Controls for 16 DoFs and 24 DoFs manipulators in the hollow pipe.

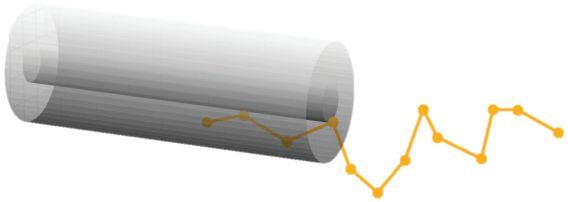


Figure 10. The directive configuration of 24 DoFs manipulator in the concentric environment.

IEEE International Conference on Robotics and Biomimetics, Qingdao, China, 3-7 December 2016, pp.1250-1255. IEEE.

7. Collins T, Shen WM. PASO: an integrated, scalable PSO-based optimization framework for hyper-redundant manipulator path planning and inverse kinematics. *Information Sciences Institute Technical Report* 2016.
8. Kavraki LE, Svestka P, Latombe JC, et al. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation* 1996; 12(4): 566-580.
9. Ghajari M F, Mayorga R V. Specialized PRM trajectory planning for hyper-redundant robot manipulators. *WSEAS Transactions on Systems* 2017; 16: 254-260.
10. Hart, Peter E, Nils J. Nilsson, et al. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics* 1968; 4(2): 100-107.
11. Hua X, Wang G, Xu J, et al. Reinforcement learning-based collision-free path planner for redundant robot in narrow duct. *Journal of Intelligent Manufacturing* 2021; 32: 471-482.
12. Maaroof OW. *Self-motion control of kinematically redundant robot manipulators*. Master Thesis, Izmir Institute of Technology, Turkey, 2012.
13. Sciavicco L, Siciliano B. A solution algorithm to the inverse kinematic problem for redundant manipulators. *IEEE Journal on Robotics and Automation* 1988; 4(4): 403-410.
14. Benzaoui M, Chekireb H. Redundant robot manipulator control with obstacles avoidance using self-motion approach. In: *Proceedings of the 13th IASTED international conference on robotics and applications* Wurzburg, Germany, 29-30 August 2007, pp.21-26.
15. Liegeois, Alain. Automatic supervisory control of the configuration and behavior of multibody mechanisms. *IEEE transactions on systems, man, and cybernetics* 1977; 7(12): 868-871.
16. Lynch KM, Park FC. *Modern robotics*. Cambridge: Cambridge University Press, 2017, p.16.
17. Lillicrap TP, Hunt JJ, Pritzel A, et al. Continuous control with deep reinforcement learning. *arXiv preprint* arXiv:1509.02971, 2015.

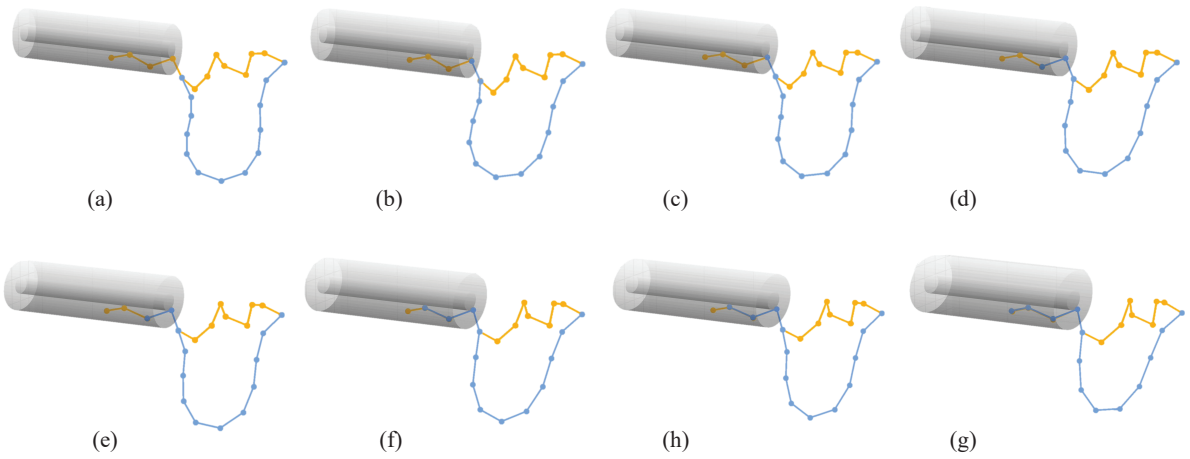


Figure 11. The Crawling Process of the 24 DoFs manipulator in the concentric environment.

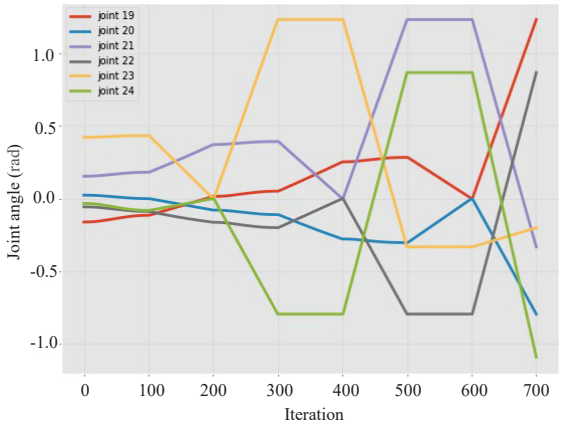


Figure 12. The joint angles curve of the inner modules of the 24 DoFs manipulator in the concentric environment.

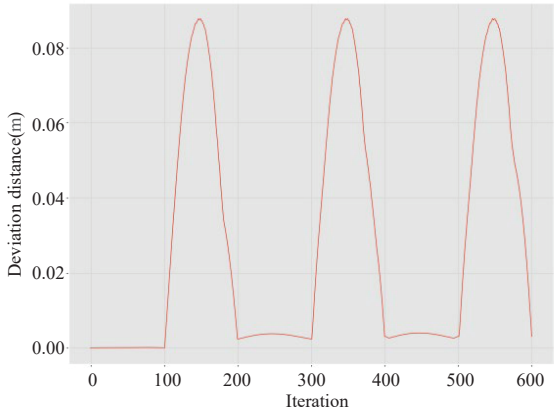


Figure 13. The maximum deviation distance change curve.