2020 IEEE International Conference on Robotics and Automation (ICRA)
31 May - 31 August, 2020. Paris, France

Offline Practising and Runtime Training Framework for Autonomous Motion Control of Snake Robots

Long Cheng*‡, Jianping Huang*‡, Linlin Liu*‡, Zhiyong Jian*‡, Yuhong Huang†‡ and Kai Huang*‡§

***Abstract*— This paper proposes an offline and runtime combined framework for the autonomous motion of snake robots. With the dynamic feedback of its state during runtime, the robot utilizes the linear regression to update its control parameters for better performance and thus adaptively reacts to the environment. To reduce interference from infeasible samples and improve efficiency, the data set for runtime training is chosen from one in several clusters categorized from samples collected in offline practice. Moreover, only the most sensitive control parameter is updated at one iteration for better robustness and efficiency. The effectiveness and efficiency of our approach are evaluated by a set of case studies of pole climbing. Experimental results demonstrate that with the proposed framework, the snake robot can adapt its locomotion gait to poles with different unknown diameters.**

I. Introduction

Snake robots are a class of biomorphic hyper-redundant robots [1], designed to imitate the snake creatures biological limbless locomotion with outstanding rapidity, stability, and diversity in the wild environment. Typically, these snake robots consist of many chain-connected active joint modules, giving them kinematic versatility, like bending, stretching, and crispation. A considerable number of these robots have been implemented in various fields, such as disaster rescuing, factory maintenance, and terrorism surveillance.

In order to better complete predefined tasks, snake robots are required to obtain capabilities of moving autonomously and behaving self-adaptively [2], e.g., making the decision when, where, and how to move based on different situations of itself and the environment. Automatically adapting to the environment is, however, not easy. The reasons are multifolds. First, due to the redundant degrees of freedom, the interaction between the robot and the environment is complex to analyze. Thus, this multi-degree locomotion of the snake robot is difficult to model. Second, when encountering an environment which is not known beforehand, how to decide a suitable control strategy and chose corresponding parameters is not straightforward. Third, the runtime procedure of deciding the control strategy and parameters must introduce little overhead. Otherwise, the desired locomotion of the robot will most probably fail.

There has been several work related to motion controlling of snake robots. Manzoor et.al.[3] designed a unified neural oscillator network to control three different types of locomotion by adjusting different control parameters. Zhelong Wang et.al.[4] presented a double-chain Hopf CPG network to control serpentine and side-winding gaits of a snake-like robot. Smooth transition of robot gait is widely studied using CPG and has obtained plentiful achievements [5], [6]. Such robot gaits optimizing processes rely on the accurate presupposition of the environment. With the increasing complexity of robotic tasks, it is hard for designers to completely model run-time environment in design time. Rollinson et al. proposed a snake robot adaptive control based on state estimation [7]. The approach generates adaptive control parameters based on the current robot state and a pre-given offset. The effectiveness may be greatly hampered if the initial state or the offset is not properly selected by designers. Researchers have proposed neural network models combined with physical environment information to determine the control scheme [8], [9], [10], [11]. These models are only control suggestions and may not provide a good effect on runtime autonomous motion control of snake robots.

In this paper, we address the problem of autonomous motion control for snake robots in environments with uncertainties that cannot be pre-determined offline. An offline and online combined framework is proposed. By letting the robot move in representative environments in offline and using linear regression to update control parameters, the uncertainties can be properly handled by runtime training. Targeting for better performance, the robot adaptively adjusts control parameters during runtime according to its dynamic state and offline experience, thus achieving autonomous motion control. The contributions of this paper are:

- An offline and runtime combined control framework for motion control of snake robots is proposed. In offline, the snake robot practices in a few representative environments with different parameters. During runtime, the robot is periodically trained to adjust its parameters for better performance based on offline experience.
- For more smooth performance improvement and better

Corresponding author: Kai Huang
* School of Data and Computer Science, Sun Yat-Sen University, Guangzhou, China
†College of Computer, National University of Defense Technology, Changsha, China
‡Key Laboratory of Machine Intelligence and Advanced Computing, Sun Yat-Sen University, Guangzhou, China
§Peng Cheng Laboratory, Shenzhen, China
Emails: chenglong3@mail.sysu.edu.cn, {huangjp25, liull28, jianzhy5}@mail2.sysu.edu.cn, huangyuhong17@nudt.edu.cn, huangk36@mail.sysu.edu.cn

robustness, we propose to update only one control parameter at a time and present a method to select the most sensitive parameter by entropy variance.
- A linear-regression based approach is proposed to update the selected parameter and the multiple regression problem is transformed into a unit regression problem.
- A set of case studies of pole climbing is conducted and the results demonstrate that our proposed strategy is effective and efficient in accomplishing autonomous motion control of snake robots.

The rest of this paper is organized as follows. The framework is described in Section II. In Section III and Section IV, the offline and runtime parts are illustrated. The case studies are presented in Section V. Section VI concludes this paper.

II. Robot Gait Model and Framework Overview

A widely used control strategy for snake-liked robots is the serpenoid curve motion model [12] and its parametric 3D extension [13]. Such parametric equations simplify the control strategy of snake-like robots and allow specifying the motion model with a small number of parameters.

We consider snake robots consisting of a chain of single degree of freedom modules, where the joints are alternatively oriented in the lateral and dorsal planes of the robot [14]. Then the robot gaits consist of separate parameterized sine waves that propagate through the lateral and dorsal joints. The angle of the ith joint at time t is:

$$\theta(i,t) = \begin{cases} A \cdot \sin(\omega \cdot t + i \cdot \varepsilon_{lat}) & lateral \\ A \cdot \sin(\omega \cdot t + i \cdot \varepsilon_{dor} + \eta) & dorsal, \end{cases} \quad (1)$$

where A, ω, and η are respectively the amplitude, frequency and the phase shift between lateral and dorsal joints. Parameters ε_{lat} and ε_{dor} describe the phase shift between adjacent joints. In summary, the snake robot gait can be specified by a set of parameters $\mathbf{\Psi} = \{A, \omega, \varepsilon_{lat}, \varepsilon_{dor}, \eta\}$.

Now, we can define the problem: *Given a set of representative environments and the ranges of snake robot parameters in $\mathbf{\Psi}$, our goal is the control policy which can autonomously generate better performance control parameters adaptive to the runtime environment with uncertainties.*

Fig. 1 shows the proposed framework. In offline, we grind the control parameter space and obtain a set of combinations. The robot moves in the representative environments with these combinations. Meanwhile, locomotion samples defined in Tab. I are saved. The samples are then categorized by k-means++ [15], [16] for runtime use. In runtime, training is performed periodically. The locomotion samples are firstly categorized to one of the offline-obtained clusters (Section IV-A.1). Then, the preponderant data in the cluster is chosen for training (Section IV-A.2). Only one parameter is updated at an iteration and it is selected by entropy variance (Section IV-A.3). Finally, the runtime training is performed based on the linear regression to predict the selected parameter's value for better performance (Section IV-B).

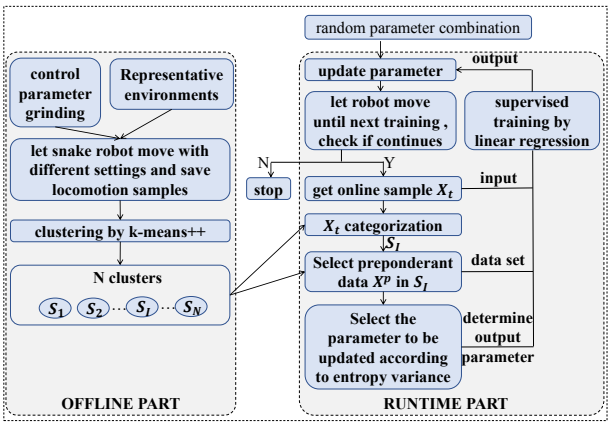


Fig. 1. The overall structure of the proposed framework

TABLE I
Structure of Data Sampled Offline

Field	Symbol	Definition
robot state	$\theta_i(t_k)$	joint angle of the ith joint at t_k
	$M(t_k)$	mean of all joint angles at t_k
performance	$P(t_k)$	performance metric evaluated at t_k
control parameters	$[A,\ \omega\ ]$	gait amplitude and frequency
	ε_{lat}, ε_{dor}	adjacent joints phase shift
	η	lateral and dorsal joints phase shift

III. The Offline Part

In the offline part, we periodically collect the data of robot state and performance for a time interval $[0, t_{max}]$ when the robot moves in representative environments with different combinations of control parameters. In other words, we let the snake robot practice and gain knowledge about its behavior and performance map in different environments. Please note that we don't notify the robot of the environment change. It only stores the performance and behavior data. However, the data itself, that is, the behavior and performance map, has an implicit connection with the environment and thus can be matched with online states to improve performance in an unknown environment. Denote the time instant of sampling as t_k, the structure of data sampled at t_k is listed in Tab. I. Please note that the performance metric function should be defined according to the task target and the environment. $M(t_k)$ is defined as $\sum_{i=1}^{n} \theta_i(t_k)/n$, where n is the number of joints.

All the samples are then categorized into several clusters. The benefits of clustering are two folds. First, the offline samples comprise a significant variety of robot locomotion states. By clustering, we can obtain a set of sub-scenarios in which the samples are similar to each other. During runtime training, we can pick the certain sub-scenario which matches the current state best for parameter updating. In this way, the interferences from the unrelated or even infeasible samples can be avoided and the effectiveness of our approach is

Algorithm 1 Clustering Offline Data by K-means++

Input: Offline Data $\boldsymbol{X}$, The number of clusters N
Output: Clusters set $\boldsymbol{S}$ and centroids set $\boldsymbol{C}$

function INITIALIZE($\boldsymbol{X}, N$)
  Set $\boldsymbol{C}$ as a random point in $\boldsymbol{X}$
  while $|\boldsymbol{C}| < N$ **do**
    $I \leftarrow \arg\max_{1\le i\le|\boldsymbol{X}|} \left(\sum_{j=1}^{|\boldsymbol{C}|} \|X_i - C_j\|^2\right)$
    $\boldsymbol{C} \leftarrow \boldsymbol{C} \cup \{X_I\}$, $\boldsymbol{X} \leftarrow \boldsymbol{X} - X_I$
  end while
  return $\boldsymbol{C}$
end function
function UPDATE($\boldsymbol{X}, \boldsymbol{C}$)
  while $\boldsymbol{C}$ does not converge **do**
    for $i \in [1, |\boldsymbol{X}|]$ **do**
      $I_i \leftarrow \arg\min_{1\le k\le N} \left(\|X_i - C_k\|^2\right)$
    end for
    for $k \in [1, N]$ **do**
      $S_k \leftarrow \{X_i \mid i \in [1, |\boldsymbol{X}|], \text{and } I_i = k\}$
      $C_k \leftarrow \sum_{P\in S} X/|S_k|$
    end for
  end while
  return $\boldsymbol{S}$ and $\boldsymbol{C}$
end function

improved. Second, the time overhead of runtime training can be greatly reduced since we only utilize one part of the offline data set as the online training set.

Denote data obtained in offline as $\boldsymbol{X}$, we use the well-known $kmeans++$ algorithm for clustering, as it has high scalability for large-scale data sets. The pseudo code is listed in Algo. 1. The algorithm returns (1) The cluster centroids $\boldsymbol{C} = \{C_1, \cdots, C_N\}$ and (2) the clusters $\boldsymbol{S} = \{\boldsymbol{S}_1, \cdots, \boldsymbol{S}_N\}$. $\boldsymbol{S}_k$ contains samples that assigned to C_k.

Note that we determine the number of clusters N as the one which leads to the lowest variance in cluster size. Let $\bar{d}_i$ denote the mean distance between members in the ith cluster and their cluster centroid. $E = \sum_{i=1}^{N}(\bar{d}_i)/N$ is the mean value of $\bar{d}_i$ for all clusters. N_{min} and N_{max} are respectively the minimal maximal number of clusters that are pre-given according to the data set scale. We vary $N^\diamond$ in $[N_{min}, N_{max}]$ and cluster data by Algo. 1. Finally, the one with the lowest variance is selected, as Eqn. (2) shows.

$$N = \underset{N_{min}\le N^\diamond\le N_{max}}{\arg\min} \sum_{N^\diamond} (\bar{d}_i - E)^2/N^\diamond \tag{2}$$

IV. The Runtime Part

In this section, we discuss how to optimize the control parameters of a snake robot during running time. The definitions of notations used in this section are listed in Tab. II.

A. Parameter Selection

1) Real-time data categorization: After getting the dynamic data X_t during runtime, we assign it to the closest cluster $\boldsymbol{S}_I$ by Eqn. (3). The similarity between two vectors is evaluated according to the *Euclidian Distance*.

$$I = \underset{k\in[1,N]}{\arg\min} (||X_t - C_k||^2) \tag{3}$$

2) Preponderant Data Selection: Since our goal is to improve performance of snake robots, we only select the samples having better performance than X_t from $\boldsymbol{S}_I$ for training. Denote the set of them as $\boldsymbol{X}^p$, we have:

$$\boldsymbol{X}^p = \{X_i | P(X_i) \ge P(X_t)\ ,\ X_i \in \boldsymbol{S}_I\}. \tag{4}$$

Note that it can also be written in vector format:

$$\boldsymbol{X}^p = [X_1^p, X_2^p, \cdots, X_{|\boldsymbol{X}|}^p]^T. \tag{5}$$

3) The selection of the sensitive parameter: At each optimization iteration, we only update one control parameter instead of all of them. The reason is that control parameters of physical systems usually have coupling relations with others. Varying one parameter at a time can lead to a more smooth performance change, thus improves the robustness of our approach. In this paper, we select the parameter to be updated according to entropy variance.

Firstly, to reduce the time overhead of calculating the entropy variance, we discretize the performance values in $\boldsymbol{X}^p$. Given a discretization step L, performance P in $\boldsymbol{X}^p$ is revised as $\hat{P} = \lfloor \frac{P}{L} \rfloor$ and we obtain a performance sequence:

$$\hat{\boldsymbol{P}}^p = [\hat{P}_1^p, \hat{P}_2^p, \cdots, \hat{P}_{|\hat{\boldsymbol{P}}|}^p]. \tag{6}$$

Then, as aforementioned, a parameter can be varied in its feasible range and has a set of candidates after grinding. So we define an important set $\mathbb{S}_{i,j}$:

$$\mathbb{S}_{i,j} = \{X \mid X \in \boldsymbol{X}^p, \text{and } \Psi(X, i) = \Omega_{i,j}\}, \tag{7}$$

From the definition, we know that $\mathbb{S}_{i,j}$ is the set of samples in $\boldsymbol{X}^p$ whose Ψ_i equals $\Omega_{i,j}$. Then the entropy about performance for $\Omega_{i,j}$ can be computed as below.

$$H_{i,j} = -\sum_{\hat{P}\in\hat{\boldsymbol{P}}} p(\hat{P}, \mathbb{S}_{i,j}) log_2 p(\hat{P}, \mathbb{S}_{i,j}), \tag{8}$$

where $p(\hat{P}, \mathbb{S}_{i,j})$ is the appearance rate of samples whose performance is $\hat{P}$ in set $\mathbb{S}_{i,j}$. With Eqn. (8), we define the entropy variance of the ith gait parameter as V_i.

$$V_i = \frac{\sum_{j=1}^{|\boldsymbol{\Omega}|}(H_{i,j} - E_i)^2}{|\boldsymbol{\Omega_i}|}, \tag{9}$$

TABLE II
NOTATION DEFINITIONS

Symbol	Definition
X	A data sample defined in Tab. I
$\boldsymbol{\Psi}$	the set of all control parameters
Ψ_i	the ith parameter in $\boldsymbol{\Psi}$
$\boldsymbol{\Omega_i}$	the set containing all the candidates of Ψ_i
$\Omega_{i,j}$	the jth member in $\boldsymbol{\Omega_i}$
$\Psi(\boldsymbol{X}, i)$	vector of values of Ψ_i of samples in set $\boldsymbol{X}$
$P(\boldsymbol{X})$	vector of values of P of samples in set $\boldsymbol{X}$
$\boldsymbol{X^p}$	the preponderant data set at runtime training

where $E_i = \sum_{j=1}^{|\mathbf{\Omega}|} H_{i,j}/|\mathbf{\Omega_i}|$ is the mean of entropies of all parameter candidates of Ψ_i .

Finally, we normalize V_i by Eqn. (10) and select the sensitive parameter by Roulette wheel selection algorithm to avoid getting stuck in the high probability selection.

$$R_i = V_i / \sum_{i=1}^{|\mathbf{\Psi}|} V_i \tag{10}$$

B. Update The Selected Parameter

Denote the selected parameter as Ψ_k, now we discuss adopting linear regression to update its value.

At each runtime training instant, we are given the data of current robot state and control parameters as well as the offline obtained samples. Our goal is to predict the next value of Ψ_k such that the robot performs better. Therefore, we can transform this problem as a linear regression problem in which $\boldsymbol{X}^p$ is the training set while current robot states and parameters are input. Then, the value of Ψ_k can be predicted after utilizing the gradient descent method to solve the weighted least squares problem.

The benefits of this method are two folds: First, using the current robot states and parameters as input builds a connection between the dynamic environment and control parameters. Although the runtime environment can be partially matched by offline data set $\boldsymbol{X}^p$, it may still have a certain degree of variability and thus cannot be predetermined in offline. Therefore, considering the runtime data offer the possibility of better performance. Second, by using the linear regression, we can vary the control parameter continuously to make responses to continuous environment changes.

1) The predicting model: The linear regression function for predicting Ψ_k is set as:

$$F_W(X_t) = W^T X_t\,, \quad W = \left[w_1, w_2, \cdots, w_{|X|}\right]^T, \tag{11}$$

where W is a coefficient vector with size $|X_t|$.

2) Runtime Training: We obtain W by supervised learning with data set $\boldsymbol{X}^p$. The well-known gradient descent method is applied to solve the optimization problem.

Step 1: Based on Eqn. (11), we obtain the square error cost function that to be minimized.

$$J(W) = \frac{1}{2|\boldsymbol{X}^p|} \sum_{X \in \boldsymbol{X}} (F_W(X)^T - \Psi(X,k))^2 \tag{12}$$

Step 2: Calculate the gradient of $J(W)$ as below.

$$\nabla_W J = \frac{1}{|\boldsymbol{X}^p|} \boldsymbol{X}^{pT} (F_W(\boldsymbol{X}^p) - \Psi(\boldsymbol{X}^p, k)) \tag{13}$$

For a better fitting result and to control the rate of convergence, we perform the weighted operation to Eqn. (13) as below.

$$\nabla_W J = \frac{1}{|\boldsymbol{X}^p|} \boldsymbol{X}^{pT} M (F_W(\boldsymbol{X}^p) - \Psi(\boldsymbol{X}^p, k)) \tag{14}$$

Fig. 2. The tested snake-like robot with a module design.

where M is a $|\boldsymbol{X}^p|$ by $|\boldsymbol{X}^p|$ diagonal matrix whose ith main diagonal entry is $P(X_i^p)/L_s$ and L_s is the learning step.

Step 3: Update the coefficient vector W by Eqn. (15).

$$W = W - \nabla_W J \tag{15}$$

Step 4: Iterate the steps above until W converges.

After the best-fit coefficient W_{best} is obtained, the selected parameter can be updated as ${W_{best}}^T X_t$.

V. Case Studies

In this section, we evaluate the effectiveness and efficiency of our approach.

A. Setup

We adopt scenarios of climbing poles as the testing environment and the rolling gait as the basic gait for locomotion. The simulations are conducted on the robot simulation platform V-REP (EDU PRO, Version 3.3.2), which runs in a VMware Player virtual machine system. The virtual machine's operating system is Linux 4.4.0-96-generic and is assigned four logical cores and 4 GB memory. The host machine is a desktop with an Intel i5-8400 processor and 16 GB memory. The tested snake robot is a 16-joint robot with 3D space locomotion ability, which is shown in Fig. 2. The diameter and length of a joint are 65 mm and 85 mm, respectively. In all the case studies, the performance metric for the robot is set as its mean velocity in the pole's direction. In the simulation, the velocity data can be directly accessed via the interface provided by VREP. In the real world, it can also be obtained based on the data fusion theory in [17] if accelerometers and gyroscope are embedded. As the robot gait needs time to be stable, the period of runtime parameter optimization is set as 2 seconds in all cases. All data is collected after $t = 10$ second because the program initialization is performed in the beginning several seconds.

B. Practicing Process

In the data acquisition process, we choose two cylinder poles, a 25 cm diameter one and a 35 cm one, as the representative environments. We let the robot climb the poles under different control parameter combinations and collect 25000 volumes of data in total. Note $\varepsilon_{lat} = \varepsilon_{dor}$ and $\eta = \pi/2$ hold for rolling gait, thus we have only three controllable parameters. The amplitude A, phase ε and angular rate ω are varied in ranges $[40, 80]$, $[0, 5]$ and $[1.5, 3]$ with steps 5, 1, and 0.5, respectively. In the clustering process, the number of clusters is set as 25, which is selected from 10 to 32 with step 1. In this case, the size of most clusters is 1000 ± 500.

C. Locomotion Along A Pole With Varying Diameter

To examine how the snake robot adaptively responses to a changing environment, we let the robot climb a pole which is divided into three sections. The diameter of the middle part is 25 cm while the diameters of the lower and higher part are both 35 cm. The climbing process lasts for 80 seconds.

The results are shown as Fig. 3. Please note that the robot itself does not know it should climb a pole or any information of the pole. In the beginning 24 seconds, the robot tries a set of gait parameters to climb and thus its velocity is close to zero. Then, it acquires a locomotion gait fitted to the part with a 35 cm diameter.

From 0 s to 24 s, the robot changes its parameters to adapt to the unknown pipe. Therefore, its velocity is close to zero. In this phase, A, ε and ω are all changing for bigger velocity. At about 24 s, the robot acquires a proper locomotion gait and begins climbing. When it is about 45 s, the robot reaches the interface where the diameter changes. Since the robot can not grasp the pole with a smaller diameter immediately, its velocity fluctuates around zero. Then the robot autonomously adjusts its parameters by our strategy to continue its climbing motion. One can observe that the parameters are also changing in this phase, which indicates the runtime learning. Similarly, when about 63 s, the robot reaches the second interface where the diameter gets bigger. After several times of training, the robot arrives at the highest part and all the parameters are stable at last. In general, the velocity changes toward bigger values during the motion. Note that the degradation of velocity at the second interface is slighter than that at the first interface. The reason is that the robot can still maintain part of the grasp force when the diameter gets bigger.

D. Locomotion Along Poles With Different Diameters

In the offline phase, our approach just needs to collect the locomotion data in some representative environments. During runtime, the approach adopts the linear regression to generate control parameters for different environments, thus achieving autonomous locomotion control. In this section, we evaluate this adaptive ability of our approach by letting the snake robot climb poles with different diameters. Since the diameters of poles used in the offline phase are 25 cm and 35 cm, we conduct several independent simulations on poles whose diameters are 20 cm, 25 cm 30 cm, 35 cm and 40 cm. The results of these simulations are shown in Fig. 4.

From the figures, we can make the following observations. First, after a certain time of adjusting, the snake robot finally can acquire a proper locomotion gait for all the poles. This demonstrates the adaptive ability of the robot for different unknown environments. Second, the snake robot requires more adjusting time when the diameter gets smaller. This is expected since the robot needs more iterations to reach the locomotion shape from its initial shape, which is a straight line. Third, the phase ε keeps unchanged in the scenario of 40 cm diameter. The reason is that the robot is not long enough to form a whole circle to wrap the pole and thus the phase has a negligible influence on the performance.

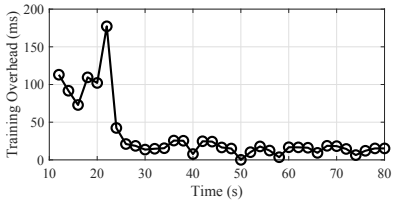


Fig. 5. The training overhead versus time in the scenario of climbing a 30 cm diameter pole.

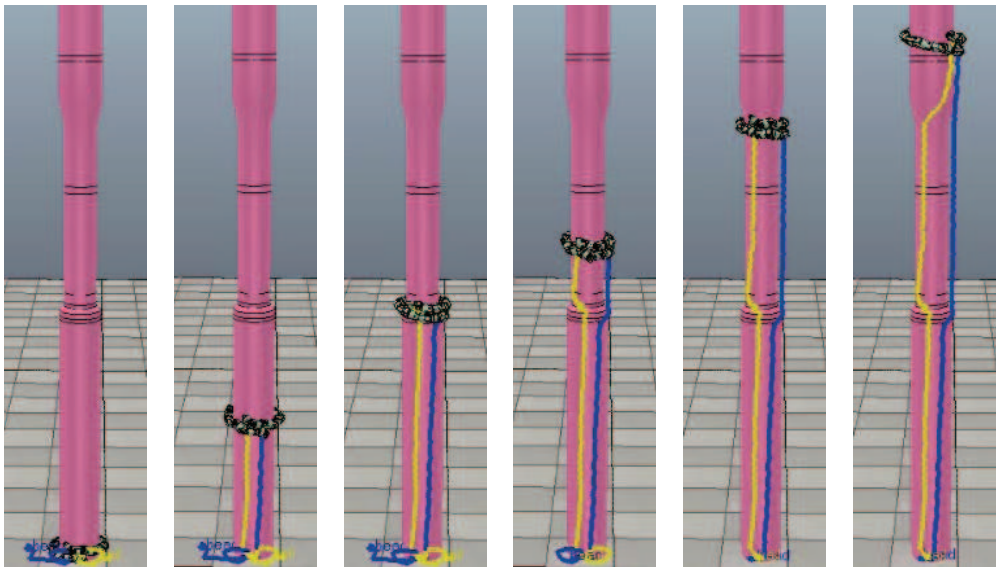
It is worth noting that, for all environments, the movement velocity of the robot ultimately fluctuates within a constant range (Fig. 4(d)). The diameter of the pole only affects the velocity convergence rate. And also the control parameters of the robot will eventually become stable (Fig. 4). This demonstrates that the effectiveness of our approach receives little influence from external environments.

E. Algorithm efficiency

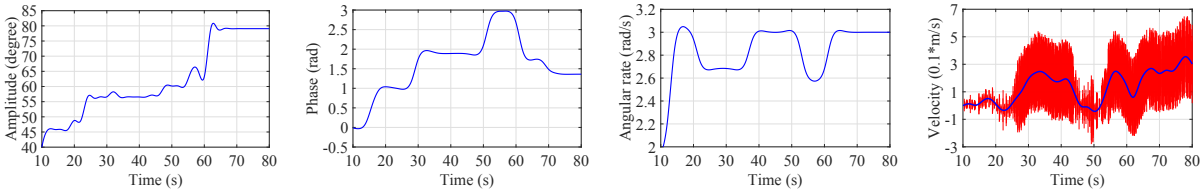
The training process for parameter updating is performed during runtime. Therefore, its time overhead is an important factor for the effectiveness of our approach. Now, we report how the training overhead varies during runtime when climbing the 30 cm diameter pole, which is shown in Fig. 5. One can observe that (1) in general, the training overhead in the beginning is larger than those after 20 seconds. It is expected since the robot adjusts itself in the beginning and thus has a low performance. A lower performance leads to a bigger preponderant data set, which then causes a larger training overhead. (2) After the beginning part, the overhead changes little. The reason is the performance becomes stable. (3) The worst-case training overhead is around 200 ms and the stable overhead is about 20 ms. Consider that the case studies are conducted in a virtual machine and the period of runtime training usually be several seconds, this time overhead is acceptable.

VI. Conclusion

This paper presents an offline and runtime combined framework for the autonomous motion control of snake robots. We use the samples having better performance in the best-matched cluster as training data set during runtime training. The interferences from low-performance or infeasible samples are avoided and the effectiveness of our approach is improved. By clustering the practicing sample offline and selecting only one parameter to update at an iteration during runtime, the overhead of linear regression is significantly reduced, which makes it feasible for runtime training. We conduct a set of case studies and the results demonstrate the effectiveness and efficiency of the proposed framework. This method can be used not only in the case as climbing pole, but also in other robotic applications.

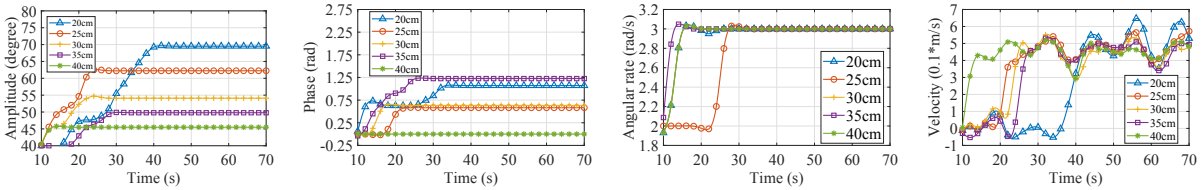


(a) t=24s (b) t=35s (c) t=47s (d) t=56s (e) t=63s (f) t=69s



(g) Amplitude versus Time (h) Phase versus Time (i) Angular rate versus Time (j) velocity versus Time

Fig. 3. Figures (a) to (f) are the climbing process of the snake robot. Figures (g) to (j) show the parameters curve and velocity curve versus time.



(a) Amplitude versus Time (b) Phase versus Time (c) Angular rate versus Time (d) velocity versus Time

Fig. 4. Parameter and performance curves when the snake robot climbs poles with different diameters

VII. Acknowledgments

This work has been partly funded by the Shenzhen Science and Technology Innovation Committee under Grant NO. JCYJ20180507182508857, and the Ministry of Science and Technology of the People's Republic of China under Grant NO. 2018YFB1802405.

References

[1] G. S. Chirikjian & J. W. Burdick, "The kinematics of hyper-redundant robot locomotion," *IEEE Transactions on Robotics and Automation*, vol. 11, no. 6, pp. 781–793, Dec 1995.

[2] Liljeb, P. Ck, K. Y. Pettersen, Stavdahl, yvind, & J. T. Gravdahl, "Snake robots: Modelling, mechatronics, and control," 2013.

[3] S. Manzoor & Y. Choi, "A unified neural oscillator model for various rhythmic locomotions of snake-like robot," *Neurocomputing*, vol. 173, pp. 1112–1123, 2016.

[4] Z. Wang, Q. Gao, & H. Zhao, "Cpg-inspired locomotion control for a snake robot basing on nonlinear oscillators," *Journal of Intelligent & Robotic Systems*, vol. 85, no. 2, pp. 209–227, 2017.

[5] Z. Bing, L. Cheng, K. Huang, M. Zhou, & A. Knoll, "Cpg-based control of smooth transition for body shape and locomotion speed of a snake-like robot," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 4146–4153.

[6] G. Chen, Z. Bing, F. Röhrbein, J. Conradt, K. Huang, L. Cheng, Z. Jiang, & A. Knoll, "Toward brain-inspired learning with the neuromorphic snake-like robot and the neurorobotic platform," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 11, no. 1, pp. 1–12, 2017.

[7] D. Rollinson & H. Choset, "Gait-based compliant control for snake robots," in *2013 IEEE International Conference on Robotics and Automation*, May 2013, pp. 5138–5143.

[8] G. Martius, R. Der, & N. Ay, "Information driven self-organization of complex robotic behaviors," vol. 8, p. e63400, 05 2013.

[9] R. Der & G. Martius, "Novel plasticity rule can explain the development of sensorimotor intelligence," vol. 112, 05 2015.

[10] J. P. Cai, L. Xing, M. Zhang, & L. Shen, "Adaptive neural network

control for missile systems with unknown hysteresis input," *IEEE Access*, vol. 5, pp. 15 839–15 847, 2017.

[11] A. Vitiello, G. Acampora, M. Staffa, B. Siciliano, & S. Rossi, "A neuro-fuzzy-bayesian approach for the adaptive control of robot proxemics behavior," in *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, July 2017, pp. 1–6.

[12] S. Hirose, "Biologically inspired robots," *Snake-Like Locomotors and Manipulators*, 1993.

[13] M. Tesch, K. Lipkin, I. Brown, R. L. Hatton, A. Peck, J. Rembisz, & H. Choset, "Parameterized and scripted gaits for modular snake robots," vol. 23, pp. 1131–1158, 06 2009.

[14] C. Wright, A. Buchan, B. Brown, J. Geist, M. Schwerin, D. Rollinson, M. Tesch, & H. Choset, "Design and architecture of the unified modular snake robot," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 4347–4354.

[15] A. Widodo & I. Budi, "Clustering patent document in the field of ict (information amp; communication technology)," in *2011 International Conference on Semantic Technology and Information Retrieval*, June 2011, pp. 203–208.

[16] M. Dundar, Q. Kou, B. Zhang, Y. He, & B. Rajwa, "Simplicity of kmeans versus deepness of deep learning: A case of unsupervised feature learning with limited data," in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, Dec 2015, pp. 883–888.

[17] Z. Bing, L. Cheng, A. Knoll, A. Zhong, K. Huang, & F. Zhang, "Slope angle estimation based on multi-sensor fusion for a snake-like robot," in *2017 20th International Conference on Information Fusion (Fusion)*. IEEE, 2017, pp. 1–6.