International Journal of Advanced Robotic Systems

OPEN ACCESS
ARTICLE

Learning and Chaining of Motor Primitives for Goal-directed Locomotion of a Snake-like Robot with Screw-drive Units

Regular Paper

Sromona Chatterjee[1]*, Timo Nachstedt[1], Minija Tamosiunaite[1], Florentin Wörgötter[1], Yoshihide Enomoto[2], Ryo Ariizumi[2], Fumitoshi Matsuno[2] and Poramate Manoonpong[3]*

1 Bernstein Center for Computational Neuroscience, Third Institute of Physics, Georg-August University of Göttingen, Göttingen, Germany
2 Department of Mechanical Engineering and Science Graduate School of Engineering, Kyoto University, Kyoto, Japan
3 Embodied AI & Neurorobotics Lab, Center for BioRobotics, The Mærsk Mc-Kinney Møller Institute, University of Southern Denmark, Odense M, Denmark
*Corresponding author(s) E-mail: chatterji.sromona.86@gmail.com; poma@mmmi.sdu.dk

Received 15 December 2014; Accepted 27 August 2015

DOI: 10.5772/61621

Abstract

Motor primitives provide a modular organization to complex behaviours in both vertebrates and invertebrates. Inspired by this, here we generate motor primitives for a complex snake-like robot with screw-drive units, and thence chain and combine them, in order to provide a versatile, goal-directed locomotion for the robot. The behavioural primitives of the robot are generated using a reinforcement learning approach called "Policy Improvement with Path Integrals" (PI^2). PI^2 is numerically simple and has the ability to deal with high-dimensional systems. Here, PI^2 is used to learn the robot's motor controls by finding proper locomotion control parameters, like joint angles and screw-drive unit velocities, in a coordinated manner for different goals. Thus, it is able to generate a large repertoire of motor primitives, which are selectively stored to form a primitive library. The learning process was performed using a simulated robot and the learned parameters were successfully transferred to the real robot. By selecting different primitives and properly chaining or combining them, along with parameter interpolation and sensory feedback techniques, the robot can handle tasks like achieving a single goal or multiple goals while avoiding obstacles, and compensating for a change to its body shape.

Keywords Snake-like Robot Using Screw-drive Mechanism, Goal-directed Locomotion, Motor Primitives, Reinforcement Learning, Policy Improvement With Path Integrals

1. Introduction

Snake-like robots have been an active research topic for several decades [1, 2, 3]. These robots generally have a high flexibility, with several segments connected in a serial manner, giving them a slender shape. This provides them with multifunctionality on the one hand, while making them difficult to control on the other hand, due to the high number of degrees of freedom [2, 7]. They are often used as an experimental platform to study locomotion or motor coordination problems [4]. Due to their structure, their applications include search and rescue operations [5] or

deployment for locomotion in narrow spaces, like pipes [6]. Undulatory movements are the conventional way to generate their locomotion [2], but this form of movement generally requires a greater width than the width of the robot. This can become a problem in narrow spaces. From this perspective, we have developed a new type of snake-like robot using a screw-drive mechanism [8]; this robot does not require undulation for its movement as propulsion is generated by the rotation of the screws. It has four screw-drive units, which are connected serially by three active joints. Furthermore, through the proper combination of these screw angular velocities, omni-directional movement is possible, unlike in most existing snake-like robots. Continuing this development, we have created a framework for generating motor primitives for the robot.

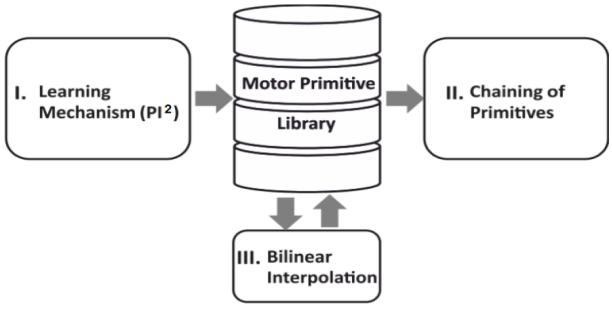

Figure 1. A goal-directed locomotion control framework

We would like to emphasize that the main contribution of this paper is a model-free, goal-directed locomotion control framework of a nonstandard snake-like robot. The framework, as shown in Figure 1, consists of three main mechanisms:

1. A learning mechanism which can learn individual motor controllers (i.e., each controller for each degree of freedom) in parallel, for periodic and nonperiodic motor primitives.
2. A chaining mechanism combining different primitives for a more complex goal-directed locomotion. This can be achieved by manual selection, sensory feedback, and/or a searching process. Here, a symbolic planning approach (acting as a searching process) for automatic action chaining is employed.
3. A bilinear interpolation mechanism for generating new locomotion behaviours based on a library of (learned) motor primitives.

Although a part of the framework for learning nonperiodic motor primitives has already been published in [10], this article presents the new features of the framework (including the mechanism for learning periodic motor primitives, as well as the chaining and interpolation mechanisms and sensing techniques), thereby leading to versatile goal-directed locomotion control for the robot. Furthermore, experimental results are presented, including the results related to goal-directed locomotion with periodic body movements and complex locomotion tasks, which demonstrate the performance of the framework. The framework uses a reinforcement learning approach called "Policy Improvement with Path Integrals" (PI^2) [9]. PI^2 is used to generate different motor primitives, and is here applied to a nonstandard snake-like robot for the first time.

Motor primitives are the "building blocks" [21] of movement generation. They remain operative throughout life in both vertebrates and invertebrates. The biological study in [22] demonstrated the additive properties of primitives, by stimulating two spinal sites in frogs. At a neural level, they are seen as force fields generated by a combination of activations from different muscles at the same time. At the behavioural level, it has been shown how adaptation of sub-movements can generate rapid hand movements [21]. Furthermore, human reaching movements were shown to be formed through a combination of different primitives with hand velocities in [23]. From a robotics point of view, a reasonable repository of motor primitives can provide a robot with self-improvement and evolutionary capabilities. The usage of a basis set of behaviours for the future adaptability of the system thus reduces complex problem of robot control, as it helps with dimensionality reduction [21, 24] and gives the robot the ability to handle new tasks in the future. Traditional principal component analysis (PCA) method have been shown to be used to generate primitives from motion capture data for human arm movements [25]. Movement primitives represented by dynamical systems can be found in [12, 13, 26], and have been shown to handle many complex tasks. The work in [26] shows how primitives are obtained by imitation learning and can be self-improved through reinforcement learning, in order to complete a complex task like the ball-in-a-cup task. The work in [27] shows how motor primitives were learned using different policy gradient-based methods for a baseball hitting task, as well as presenting a study on primitive learning by such various methods. Thus, different approaches exist regarding the ways in which movement primitives can be defined and used in robotic systems.

The motor primitive approach in this work reduces the complexity of the robot control problem for a challenging task like goal-directed locomotion generation of the complex snake-like robot, while also giving the robot the ability to handle unknown situations. Here, the robot learns to locomote toward a goal using PI^2, and thus a proper combination of locomotion control parameters are obtained for different goals and robot shapes (i.e., straight-line, zigzag, arc, etc.). From this, we take certain sets of learned control parameters as motor primitives and then combine them properly using chaining and parameter interpolation to produce new behaviours in an online manner. The approach we present here also overcomes the problems of the classical control mechanisms used for generating locomotion in this snake robot. These classical mechanisms include trajectory tracking based on feedback,

and front-unit control [8]. Both of these are model-based mechanisms which require a kinematic model of the robot and can only deal with simple robot shapes. They fail to provide any closed-form kinematic solutions both for screw velocity and joint periodic movement control, and for the joint-by-joint control of complex body shapes (for instance, zigzag and semi-zigzag shapes). Furthermore, they sometimes have difficulty in finding proper control parameters for goal-directed locomotion, due to the switching of the passive wheels of the robot in contact with the ground. Using sensors to achieve adaptive control typically also requires a kinematic model [8], and it is difficult to find proper sensorimotor connections that can generate effective locomotion for complex robot shapes. Our approach, on the other hand, is able to find proper parameters for controlling both screw velocities and joint periodic movements, and can also generate locomotion with body deformation. It is also robust, meaning that learned motor primitives from a simulation can be directly applied to the real robot without adjusting or tuning the control parameters. Thus, a library of motor primitives can be generated without trouble.

The rest of the paper is organized as follows: in Section 2, we briefly describe the snake-like robot with screw-drive units; Sections 3 and 4 present the learning formulation and experiments using PI^2 to generate motor primitives for both periodic and nonperiodic motions; and Sections 5 and 6 describe how the generated motor primitives are used in real robot experiments to obtain new robot behaviours, and thus to address locomotion in unknown situations using primitive chaining and parameter interpolation.

2. The Snake-Like Robot with Screw-Drive Units

The basic structure of the 10-DOF snake-like robot with screw-drive units is shown in Figure 2. The robot has three active joints and four screw-drive units. Each screw-drive unit has one A-max22 Maxon DC motor, one screw part, and an encoder. The rotation of the screw unit around its rotation axis is driven by the motor. Each screw-drive unit has eight blades attached to it, with each blade having four alternately passive wheels with rubber rings. The rubber rings provide a better grip. Propulsion is generated by the rotation of the screw-units, which facilitates the movement of the robot in any direction. Each screw unit is said to be a "left" or a "right" screw unit, depending on the inclination (α) of its blade. The first screw unit connected to the head is a right screw unit and the other units are alternatively left (if, $\alpha > 0$) or right (if, $\alpha < 0$). Two servo motors (Dynamixel DX-117, Robotis) drive each joint, with each having two degrees of freedom (pitch and yaw angles). Since all the screw units remain in contact with the ground via at least one wheel, and flat ground is here presumed in our present study, the pitch angle of the robot is always zero for all our experiments. The joint angles have a range of $\pm\frac{\pi}{2}$ rad.

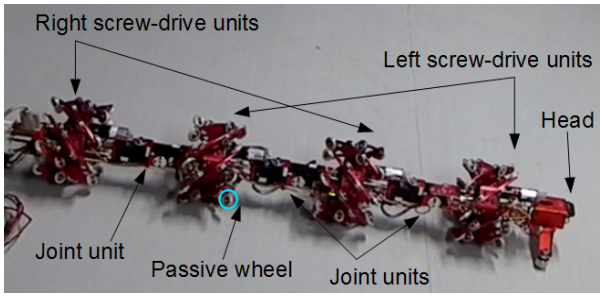

Figure 2. The snake-like robot with screw-drive units: robot structure showing four screw-drive units and three active joints. There are eight blades with passive wheels attached to each screw unit. The head of the robot is provided with a ball bearing, ground contact, and a bumper for stability.

3. Learning Motor Control with PI^2 to Generate Motor Primitives

PI^2 is a probability-based reinforcement learning (RL) approach which follows direct policy search in order to improve the policy. In this study, we focus on providing the framework, rather than comparing different optimization approaches (or learning mechanisms) to the task; we therefore employ the state-of-the-art learning mechanism PI^2. It was selected because it has been successfully used for learning in continuous, high-dimensional action spaces [9, 13, 28], thereby confirming that it is an appropriate choice for the task at hand. It is a robust mechanism, as well as being easy and efficient to implement for the purposes of trajectory rollouts and direct policy searches in parameter space. It is numerically simple without any matrix inversion and can be used as model-free in nature, with easy-to-construct cost function requirements. It has no open parameters to be tuned other than exploration noise [11, 12, 13] and is faster than gradient-based RL approaches by one order of magnitude [9]. Some interesting applications of PI^2 have been seen; for example, in [9], a 12-DOF simulated robotic dog learned to jump a gap. In [13], an 11-DOF arm hand learns both the goal and the shape of the motion required to complete a pick-and-place task. In [28], a robot arm learns to pour water using PI^2 and dynamic movement primitives. In contrast to these previous studies, here we apply PI^2 to the task of learning the locomotion control parameters of the snake robot, in order to generate motor primitives and locomotion.

Typically, RL has been shown to be one of the most suitable learning methodologies to deal with robot control problems [14]. Since the frame is flexible, one can also replace the RL-based PI^2 learning mechanism with other learning mechanisms (e.g., genetic algorithm (GA), particle swarm optimization (PSO) [18, 19], or a combination of RL and PSO [20]), if required. However, GA and PSO, which fall into the category of evolutionary optimization techniques [18, 19], may require searching through a large number of candidate control policies. Thus, they might take more time to learn the best policy [15]. They may also require the complex tuning of their open parameters, like crossover/

mutation rates and population size for GA [16], and inertia factor, self-confidence and swarm confidence for PSO [17, 19]. In GA, certain further components – like chromosome encoding, and selection and replacement strategies – also need to be designed.

In this study, using RL-based PI^2, we generate motor primitives for the robot (i.e., motions with and without periodic body movements). Prior to the start of the learning process, a policy, cost function and exploration noise is defined. After this the learning starts, and the parameter vector to be learned, U (containing locomotion control parameters), is updated using PI^2 at the end of every update t. Each update consists of K noisy trajectories or roll-outs. n updates are performed in order to obtain the final parameter vector, which will make the robot locomote toward a given goal. Table 1 gives the notations used here.

Notation	Description
w	Robot's state vector
u	Robot's control input vector
ϕ_i	Joint angle
$\dot{\phi}_i$	Joint angular velocity
θ_i	Screw angular velocity
(x,y)	Position of robot head
ψ	Orientation of first screw unit with robot head
U_1	Parameter set learned for a prefixed robot shape
U_2	Parameter set with all seven control parameters
U_3	Parameter set learned for periodic movements
φ_i	Joint angle phase
r	Cost function
(x_G,y_G)	Goal position
n	Total number of updates
t	Update index
$\tau(a)$	Trajectory with control parameter set (a)
K	Number of noisy trajectories or roll-outs
k	Roll-out index
$\epsilon_{t,k}$	Noise in k^{th} trajectory of t^{th} update
$\epsilon_{\theta_i(t,k)}$	Noise applied to screw velocity
$\epsilon_{\phi_i(t,k)}$	Noise applied to joint angle
$\epsilon_{\varphi_i(t,k)}$	Noise applied to joint angle phase
$I_{t,k}$	Final cost at the end of noisy roll-out
$P_{t,k}$	Probability weighting

Table 1. Notations

3.1 Policy Formation

The snake-like robot with screw-drive units follows the feature described in Equations (1-3) [8]:

$$A\dot{w} = Bu, \tag{1}$$

$$w = \left[x, y, \psi, \phi_1, \phi_2, \phi_3\right], \tag{2}$$

$$u = \left[\dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3, \dot{\theta}_4, \dot{\phi}_1, \dot{\phi}_2, \dot{\phi}_3\right]. \tag{3}$$

Here, w is the state vector and u is its control input vector. (x,y) gives the head position of the robot. ψ is the absolute orientation of the first screw-drive unit with the robot head. The three yaw joint angles are given by ϕ_1,ϕ_2,ϕ_3 in radian (rad), while $\theta_1,\theta_2,\theta_3,\theta_4$ are the respective angular velocities of the first, second, third and fourth screw-drive units from the head, in radians/sec (rad/s). $\dot{\phi}_1,\dot{\phi}_2,\dot{\phi}_3$ are the angular velocities of the three joints in rad/s. A and B are the system matrices [8] and depend on system configurations. The learning is executed such that, if the initial head position is at (x_0,y_0) and the goal to be reached is G (x_G,y_G), the final state vector w_{goal} on reaching the goal should have the head position as (x_G,y_G). Two parameter vectors representing the control policy are described by Equations (4) and (5):

$$U_1 = \left[\dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3, \dot{\theta}_4\right], \tag{4}$$

$$U_2 = \left[\dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3, \dot{\theta}_4, \phi_1, \phi_2, \phi_3\right]. \tag{5}$$

The parameter vector to be learned is selected according to the learning problem. U_1 is used for experiments when joint angles are fixed, and U_2 is used when all seven control parameters, four screw-drive velocities θ_i and three joint angles ϕ_i are to be learned. Thus the control policy following Equations (1–3) is represented by U_1 and U_2 of Equations (4 – 5).

3.2 Defining Cost Function and Exploration Noise

Here, Euclidean distance is used as cost function r:

$$r(x,y) = \sqrt{(x - x_G)^2 + (y - y_G)^2}. \tag{6}$$

It provides the distance in metres (m) between a reached robot head position (x,y) and a given goal position (x_G,y_G). The final parameter vector is obtained after learning converges and the cost is almost zero (i.e., the goal is reached and the task is completed).

Noise is the only open parameter in PI^2 and is designed as per need. Random values ζ from a normal distribution N(0,1), which has zero mean and a standard deviation of 1, are selected here. Following this, ζ are then dynamically adjusted according to the noise-free cost r_{t-1} obtained at the

end of the previous update. When $r_{t-1}>3$ m, then the noise is drawn as follows: $\epsilon_{t,k}=(\exp\frac{-1}{r_{t-1}})\Big/ L \cdot\zeta,\ \zeta\in N(0,1)$. Here, L=10 metres, and $\epsilon_{t,k}$ is the noise during the k^{th} noisy trajectory or the roll-out of the t^{th} update. When $0.5<r_{t-1}\leq 3$ m, then the noise is adjusted as follows: $\epsilon_{t,k}=0.05\zeta,\zeta\in N(0,1)$. When it is very low – ≤0.5m – then the noise is adjusted as follows: $\epsilon_{t,k}=0.025\zeta,\zeta\in N(0,1).\epsilon_{\theta_i(t,k)}$ (i=1, 2, 3, 4) and $\epsilon_{\phi_i(t,k)}$ (i=1, 2, 3) represent the noise applied to the screw velocities and joint angles, respectively. All seven of these noise distributions follow the above description of $\epsilon_{t,k}$.

3.3 Implementation of PI² for Nonperiodic Motor Primitives

To start with, parameter vector U_1 or U_2 is selected according to the learning task. We then fix the number of roll-outs K per update to 40. In every roll-out k, the robot is simulated to move with noisy parameters for a given time (10s), starting from robot start position (x_0,y_0). The robot is driven by the locomotion control parameters (screw velocities and joint angles). In this way, the corresponding noisy trajectory is obtained as follows:

$$\tau_{t,k}\big(\theta_1+\theta_{1(t,k)},\theta_2+\theta_{2(t,k)},\theta_3+ \\ +\theta_{3(t,k)},\theta_4+\theta_{4(t,k)},\phi_1,\phi_2,\phi_3\big) \tag{7}$$

or

$$\tau_{t,k}\big(\theta_1+\epsilon_{\theta_1(t,k)},\theta_2+\epsilon_{\theta_2(t,k)},\theta_3+\epsilon_{\theta_3(t,k)},\theta_4+ \\ +\epsilon_{\theta_4(t,k)},\phi_1+\epsilon_{\theta_4(t,k)},\phi_2+\epsilon_{\theta_4(t,k)},\phi_3+\epsilon_{\theta_4(t,k)}\big) \tag{8}$$

In the above, $\theta_i+\epsilon_{\theta_i(t,k)}$ (i=1,2,3,4) give the noisy screw velocity parameters and $\phi_i+\epsilon_{\phi_i(t,k)}$ (i=1,2,3) the noisy joint angles. Having obtained the reached position $(x_{t,k},y_{t,k})$ of the robot head at the end of this roll-out, the final cost for the roll-out is calculated by evaluating the cost function as follows: $I_{t,k}=r(x_{t,k},y_{t,k})$. In this way, all K noisy roll-outs from the robot start position within one update process t are completed, and a corresponding $I_{t,k}$ is stored for each roll-out. From here, the PI² update process starts and an exponential value is calculated on $I_{t,k}$ for each roll-out, as follows:

$$S(\tau_{t,k})=\exp^{-\lambda\frac{I_{t,k}-\min\limits_k(I_{t,k})}{\max\limits_k(I_{t,k})-\min\limits_k(I_{t,k})}}. \tag{9}$$

The constant factor λ=30. The probability weighting $P_{t,k}$ for each roll-out is calculated as follows:

$$P_{t,k}=\frac{S(\tau_{t,k})}{\sum\limits_{l=1}^{K}S(\tau_{t,l})}, \tag{10}$$

and the parameter updates are

$$\Delta\theta_i=\sum_{k=1}^{K}P_{t,k}\cdot\epsilon_{\theta_i(t,k)},\qquad \Delta\phi_i=\sum_{k=1}^{K}P_{t,k}\epsilon_{\phi_i(t,k)}. \tag{11}$$

From the above equations, the update vector is constructed as $\Delta U_1=[\Delta\theta_1,\Delta\theta_2,\Delta\theta_3,\Delta\theta_4]$, or $\Delta U_2=[\Delta\theta_1,\Delta\theta_2,\Delta\theta_3,\Delta\theta_4,\Delta\phi_1,\Delta\phi_2,\Delta\phi_3]$. The locomotion control parameter vector at the end of an update t is thus given by $U_1^{(t)}=U_1+\Delta U_1$ or $U_2^{(t)}=U_2+\Delta U_2$. While updating, the joint angles are limited within ±1 rad and the screw-drive velocities within ±1 rad/s to avoid conditions whereby, for example, the robot is made to go into a shape in which, at any instant, $\phi_1=\phi_2=\phi_3=90\ (1.57rad)$. At the end of each update process t, one noise-free trajectory with updated parameters $U_1^{(t)}$ or $U_2^{(t)}$ is simulated, in order to obtain the noise-free cost $r_t=r(x_t,y_t)$ for the reached robot head position (x_t,y_t). If the cost is smaller than a set threshold, no further updates are required; if not, the process is repeated for the next update t+1. This iterative process continues until the robot has learned the required parameters to reach the goal.

3.4 Implementation of PI² for Periodic Motor Primitives

With this mechanism in place, despite having an artificial robot locomotion behaviour involving rotating screw units, the robot can make periodic snake-like movements. In this learning mechanism, to locomote toward a goal while making periodic body movements, the following control parameter vector is learned:

$$U_3=\big[\dot\theta_1,\dot\theta_2,\dot\theta_3,\dot\theta_4,\varphi_1,\varphi_2,\varphi_3\big]. \tag{12}$$

The control policy for the periodic generation task follows Equations (1-3) and is represented by U_3, being a combination of the screw-drive velocities (θ_i) and joint angle phases (φ_i) parameters. At the same time, each joint angle ϕ_i follows a sinusoidal motion shifted in phase as follows:

$$\phi_i=A\sin\big(\omega T+\varphi_i\big). \tag{13}$$

So, the joint angles with amplitude A and frequency ω are represented as above. A=0.2 and ω is taken as 0.6 for the presented data. A restricts the joint angle within ±0.2 rad. The noise values used for the screw-units, $\epsilon_{\theta_i(t,k)}$ (i=1,2,3,4), are drawn as described above. The noise values applied to the phase of each joint angle are $\epsilon_{\varphi_i}(t,k)$ (i=1,2,3) are, and follows the Gaussian distribution $N(0,0.02)$. The learning process is analogous to the one described in Section 3.3. In every roll-out k, the robot moves with the noisy parameters, giving the following trajectory:

$$\tau_{t,k}\big(\theta_1+\epsilon_{\theta_1(t,k)},\theta_2+\epsilon_{\theta_2(t,k)},\theta_3+\epsilon_{\theta_3(t,k)},\theta_4+ \\ +\epsilon_{\theta_4(t,k)},A\sin\big(\omega T+(\varphi_1+\epsilon_{\varphi_1(t,k)})\big), \\ A\sin\big(\omega T+(\varphi_2+\epsilon_{\varphi_2(t,k)})\big), \\ A\sin\big(\omega T+(\varphi_3+\epsilon_{\varphi_3(t,k)})\big)\big) \tag{14}$$

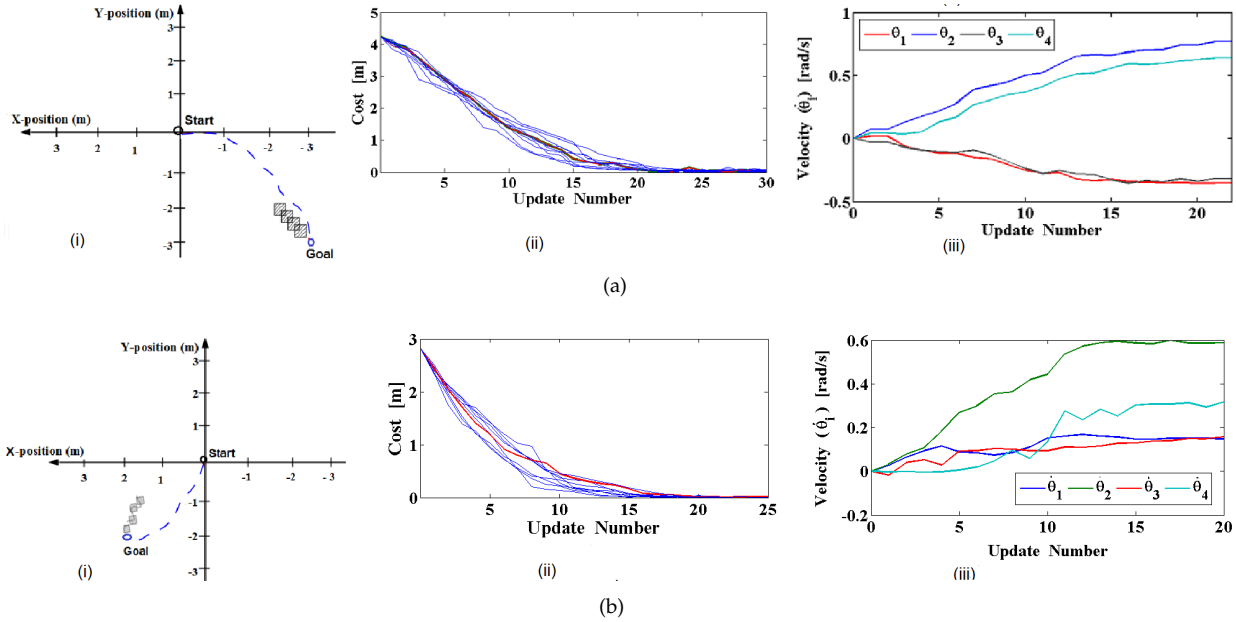

Figure 3. (a) Experiment 1 with a straight-line body shape: (i) Shows the robot reaching the goal position (-3 m, -3 m) (shown by the small blue circle) in a straight-line body shape. The final followed trajectory is indicated by the blue dashed line. (ii) The learning converges to the lowest cost for all 10 runs, taking around 20 updates for the average run (in bold). (iii) Shows that the learning of screw velocities stabilizes after the goal is reached at around 20 updates. The final values are θ_1 = -0.35rad/s, θ_2 = 0.77rad/s, θ_3 = -0.3rad/s, and θ_4 = 0.65rad/s. (b) Experiment 2 with a zigzag body shape: (i) Robot reaches the goal position (2m, -2m; the small blue circle). (ii) Learning converges to the lowest cost at around 16 updates for the average run (in bold red). (iii) The final learned screw velocities in the third picture are θ_1 = 0.16rad/s, θ_2 = 0.61rad/s, θ_3 = 0.14rad/s, and θ_4 = 0.30rad/s.

Once the final cost for this trajectory $I_{t,k}$ is obtained, followed by $S(\tau_{t,k})$ and $P(\tau_{t,k})$ using the equations in (9-10), the updates on the parameters are: $\Delta\theta_i = \sum_{k=1}^{K} P_{t,k} \cdot \epsilon_{\theta_i(t,k)}$ and $\Delta\varphi_i = \sum_{k=1}^{K} P_{t,k} \cdot \epsilon_{\varphi_i(t,k)}$. At the end of this update, the new parameters are $U_3^{(t)} = U_3 + \Delta U_3$, where, $\Delta U_3 = [\Delta\theta_1, \Delta\theta_2, \Delta\theta_3, \Delta\theta_4, \Delta\varphi_1, \Delta\varphi_2, \Delta\varphi_3]$. The update process iteratively continues until the final parameters are obtained. In this way, periodic robot behaviours are generated for different goals, and some are taken as periodic motor primitives. In future, we plan to investigate how these periodic motor primitives can be used to handle locomotion on complex ground conditions, e.g., on a slope.

4. Experiments in Generation of Motor Primitives

These experiments demonstrate the generation of motor primitives using PI^2. For all the following experiments, we select one of three parameter vectors – U_1, U_2 or U_3, from Equations (4), (5) and (12) – depending on our learning task, and initialize it to zero. First, we learn the control parameters with a simulated robot and then we successfully transfer them to the real one. The robot length is around 0.9 m. All goal positions are in metres (m) and the robot starts at (0m, 0m). We encourage readers also to view the supplementary video documenting all the real robot experiments (1-4), available online at http://manoonpong.com/IJARS2015/svideo.mpeg.

4.1 Learning Robot Control for a Straight-Line Shape

In Experiment 1, we restrict the robot shape to a straight line, with ϕ_i (i=1,2,3) = 0rad. Four screw velocities θ_i (i.e., parameter vector U_1) are learned for this body shape and a given goal. Figure 3(a) shows the experiment for goal (-3m, -3m), the learning curves, and the changes in screw velocities during the learning.

4.2 Learning Robot Control for Any Fixed Shape

The experiment in this section demonstrates the learning of θ_i (i=1,2,3,4) when the robot has a different body shape. In Experiment 2, we fix the robot shape into a zigzag – with ϕ_1=0.5rad, ϕ_2= −0.5rad, and ϕ_3=0.5rad – prior to learning. θ_i (i.e., parameter vector U_1) are then learned for this body shape and a given goal. Figure 3(b) shows the experiment for goal (2m, -2m), along with the learning curves.

4.3 Learning All Seven Robot Control Parameters

This experiment demonstrates that the robot learns all of its seven control parameters using U_2, θ_i (i=1,2,3,4) and ϕ_i (i=1,2,3) for locomoting toward a given goal. Figure 4 shows the learning curves for the goal (-1m, -3m).

4.4 Learning Robot Control for Periodic Body Movements

This experiment demonstrates how the robot locomotes toward a given goal with learned screw velocities and joint angle phases, so as to have periodic body movements. The parameter vector learned is U_3. Figure 5 presents the experiment and shows the goal position (-2m, -2m), the learning curves, and the changes to the screw velocities, joint angles phases and joint angles during the learning.

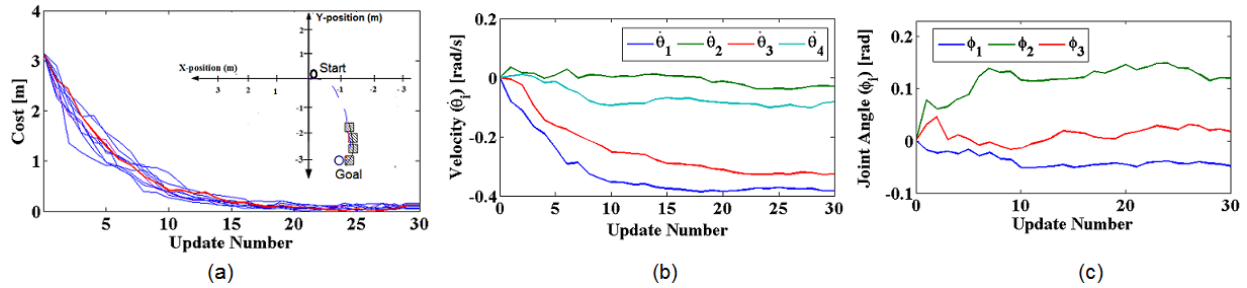

Figure 4. Experiment 3: Learning all seven locomotion control parameters. (a) Learned goal position (-1m, -3m) shown in the inset. Learning converges to lowest cost for all 10 runs, taking around 20 updates for the average run (in red). (b) The learned screw velocities are θ_1 = -0.39rad/s, θ_2 = -0.02rad/s, θ_3 = -0.32rad/s, and θ_4 = -0.08rad/s. (c) The learned joint angles are $\phi_1 = -0.05$rad, $\phi_2 = 0.11$rad, and $\phi_3 = 0.02$rad.

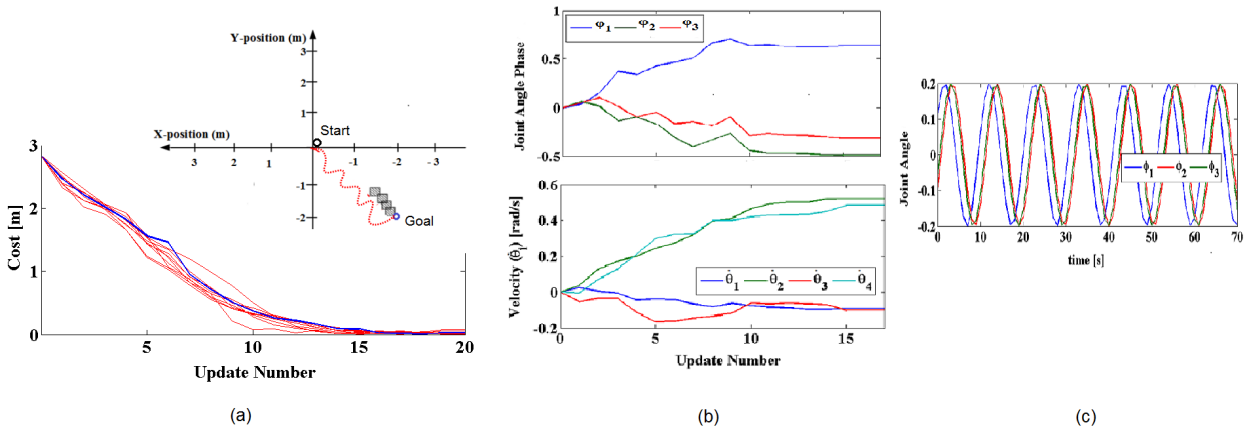

Figure 5. Experiment 4: Learning all seven locomotion control parameters with periodic body movements. (a) The robot reaches the learned goal position (-2 m, -2 m), shown in the inset. The final followed trajectory is indicated by the red dashed line. Learning converges to lowest cost for all 10 runs, taking around 15 updates for the average run (in bold). (b) Learning of the screw velocities (θ_i) and joint angle phases (φ_i) converges around 15 updates. The final learned values are θ_1 = -0.10, θ_2 = 0.53, θ_3 = -0.12, and θ_4 = 0.50; φ_i are φ_1 = 0.6, φ_2 = -0.5, and φ_3 = -0.3. (d) The three learned joint angles ϕ_i following sinusoidal motion shifted in phase. It shows that ϕ_2 and ϕ_3 are almost in phase, while ϕ_1 leads them.

From the results of these experiments, we can see how PI^2 learns control parameters (joint angles and screw velocities) for different goals and body shapes, and thus generates periodic and nonperiodic motor primitives. We can also see that, in most of these cases, convergence in learning was reached and the motor primitives were achieved within 12-20 updates.

5. Generating Complex Behaviour with Motor Primitives

The above demonstrates how motor primitives can be generated. Some of them are shown in Figures 3(a), 3(b), 4(a) and 5(a). A reasonable number of motor primitives are stored to form a library, which can be used to handle a variety of situations. The primitives are described as: "moving straight in a straight configuration", "moving straight in an arc robot shape", "moving diagonally in a zigzag robot shape", etc. Figure 6, taken as an example, shows the main motor primitives. A part of the formed primitive library is given in Table 2. The robot chains some of these primitives, in order to produce a sequence of behaviours in the real environment. Parameter interpolation and sensory feedback are employed to generate new locomotion behaviour.

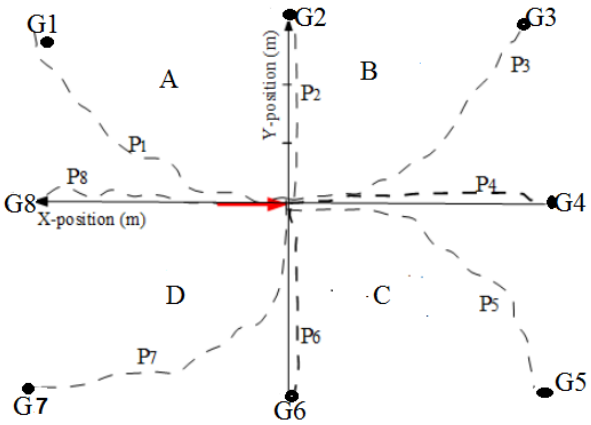

Figure 6. Generated Primitives: P_1 to P_8 give the generated robot behaviours with different body configurations. P_1 gives the robot behaviour to move at 135°; for a similar description of other primitives, refer to Table 2. G1 to G8 give the existing goals, and the red arrowhead indicates the robot's head.

Primitive	Real robot behaviour
P_{1_s}	move at 135° with a straight shape
P_{1_a}	move at 135° with an arc shape
P_{2_s}	move at 90° with a straight shape
P_{2_a}	move at 90° with an arc shape
P_{2_z}	move at 90° with a zigzag shape
P_{3_s}	move 45° diagonally with a straight shape
P_{4_s}	move straight forwards with a straight shape
P_{4_z}	move straight forwards with a zigzag shape
$P_{4_{pe}}$	move at -90° with periodic body movements
P_{5_s}	move -45° diagonally with a straight shape
$P_{5_{pe}}$	move at -30° with periodic body movements
P_{6_s}	move at -90° with a straight shape
P_{7_z}	move at -135° with a zigzag shape
P_{8_s}	move straight backwards with a straight shape
P_{8_a}	move straight backwards with an arc shape
P_9	move at 90° with a semi-arc shape
P_{10}	move straight forwards with a semi-zigzag shape
P_{11}	move at 45° with a semi-zigzag shape
P_{12}	move at 90° with a semi-zigzag shape
P_{13}	move straight backwards with a semi-zigzag shape

Table 2. The Motor Primitive Library

5.1 Chaining of Primitives

Figure 7 shows a graphical representation of how primitives have been chained in this work. The primitives can belong to any robot shape or configuration, whether straight-line, zigzag, arc, etc. As an example, Figure 7 shows that the first primitive to move at 20° (m1) forward is chained with the second primitive to move at 70° (m2) forward, in order to reach position C. These primitives are followed by the third primitive, which moves at -70° (m3) downward, thus finally reaching the goal. Primitives are also chained and driven by sensory feedback as a reactive control mechanism. For example, in an environment with obstacles, multiple primitives are chained by sensory feedback to allow the robot to avoid the obstacles and reach a goal. Similar sensory feedback techniques, such as sensing joint angles, etc., can also be employed. In this way, chaining can be effectively used to obtain new robot behaviours for unknown situations.

5.1.1 Symbolic planning for automatic action chaining

Here, we present a symbolic planning approach (i.e., a STRIPS-like planner [32]) for generating a plan based on learned primitives. This is executed at the highest level of abstraction, as in a multilayer cognitive architecture [29, 30]. The planner searches for plans using a declarative knowledge representation and generates actions to instruct the robot to achieve desired tasks, like moving to a goal or avoiding an obstacle, etc. The list of primitives in the library, shown in Table 2, is used for the action definition. The planning domain definition consists of a list of predicates, actions and planning operators. Predicates are logical formulas which take true or false values. In our example, the predicates are defined as

$$1.\ ongoal(robot) \quad 2.\ obstacle(angle)$$

The predicate $ongoal(robot)$ describes the situation in which the robot is on the goal, whereas $obstacle(angle)$ refers to a situation in which an obstacle is detected in the direction specified by the angle. In our case, $angle \in \{angletoGoal, angletoGoal + 90°, angletoGoal - 90°\}$. An action is defined as $move(angle,shape)$ to instruct the robot to move in the direction specified by the angle, with a given robot body shape. Planning operators (POs) consist of three parts: PO = $\{a,p,e\}$, with a = actions, p = preconditions, and e = effect. If p (a set of predicates) are true, then the corresponding PO can be applied using a, so as to act in order to produce the effect e. The change after execution is also coded as a set of predicates.

For the execution of a specific task, we need to define the planning problem, consisting of an initial state S_{ini} and the goal specification g. In our example, S_{ini} is defined as $\{ongoal(robot),\ obstacle(angletoGoal),\ obstacle(angletoGoal + 90°),\ obstacle(angletoGoal - 90°)\}$, i.e., a set of initial predicates at the start, with each taking true/false values. In turn, g is defined as the specification that $ongoal(robot) = True$, i.e., the final grounded predicate at the end of the task. In our case, we adopt a *replanning* strategy. The high-level planning module makes an evaluation at every action interval. If there is a violation of the preconditions (p) of the POs, while the effects (e) have not been yet obtained, then the system generates a new plan, with a replanning approach. Thus, the planner uses all the above elements to generate a sequence of actions, which produces the sequence of changes necessary to obtain g from S_{ini}. In Section 6, an example is presented of a plan to avoid obstacles.

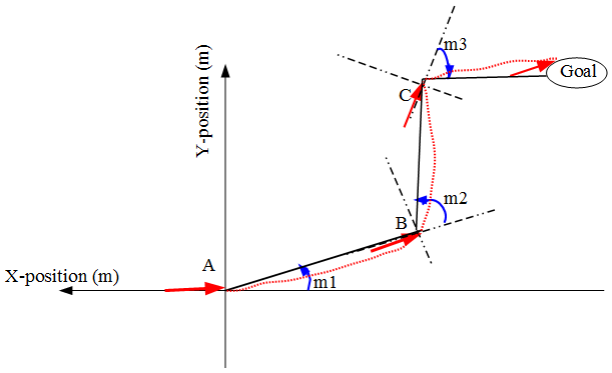

Figure 7. Graphical representation of the chaining of primitives to produce complex robot behaviours, to achieve multiple goals. The red arrowhead indicates the robot's head. m1, m2 and m3 are the three angles relative to instantaneous robot body orientation. Three required primitives are selected to take the robot from start position A to the goal, via B and C points. The red dashed line gives the trajectory.

5.2 Parameter Interpolation

Once this basic primitive set of P_1 to P_8 was selected, as depicted in Figure 6, this work made use of parameter interpolation of nonperiodic primitives, in an attempt to further generalize locomotion generation. The goal was to make the robot generate new motor controls, and actions necessary to reach new goals, from existing primitives.

Here, bilinear interpolation – a basic interpolation method for non-linear systems – is used for interpolating parameters, considering that the snake-like robot follows non-linear kinematics and uses all seven control parameters for interpolation. For interpolation, new goals are present in one of the quadrants marked as A, B, C and D, as shown in Figure 6. For example, by interpolating the learned control parameters for primitives P_1, P_2 and P_8 in the quadrant marked A, the motor control for the robot to move 120°, i.e., diagonally backwards to its start position, is generated. Similarly, to obtain new robot behaviours for new goals in the quadrant marked B, P_2,P_3 and P_4 are used for the interpolation; in the quadrant marked C, P_4,P_5 and P_6 are used; and in quadrant D, P_6,P_7 and P_8 are used for interpolation. Once the new goal has been mapped to see in which quadrant of the coordinate plane it belongs, and the necessary primitives have been selected, the corresponding learned parameters for the primitives are selected to give U_{P_i} for each one, with i being the primitive index:

$$U_{P_i} = \left[\dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3, \dot{\theta}_4, \phi_1, \phi_2, \phi_3\right]. \quad (15)$$

In this work, for convenience, we take each quadrant to be a 2m × 2m area, to help generate motor primitives. Thus, the interpolation is considered within this range. So, the total area covered by all quadrants of the coordinate plane comprises a 4m × 4m grid. For example, let the parameter interpolation set up take place in the quadrant marked as A in Figure 6. The goals $G1,...,G7,G8$ correspond to positions (2, 2), (0, 2), (-2, 2), (-2, 0), (-2, -2), (0, -2), (2, -2) and (2, 0), respectively. All goal positions are in metres (m). So, P_1, P_2 and P_8 are to be interpolated in A, thus giving i=(1, 2, 8) for Equation (15). As a result, the interpolation takes place with $f(G2)=U_{P_2}$, $f(G1)=U_{P_1}$, $f(G8)=U_{P_8}$, $f(O)$=[0,0,0,0,0,0,0], with f giving the mapping between the learned goals and the required primitives. Positions $G2=(x_1,y_2)=(0,2), G1=(x_2,y_2)=(2,2), G8=(x_2,y_1)=(2,0), O=(0,0)$. Thus, the interpolated parameters for a new goal G(x,y) are obtained as follows:

$$f(S_1) = \frac{x_2 - x}{x_2 - x_1} f(O) + \frac{x - x_1}{x_2 - x_1} f(G_8), \text{ where } S_1 = (x, y_1), \quad (16)$$

$$f(S_2) = \frac{x_2 - x}{x_2 - x_1} f(G_2) + \frac{x - x_1}{x_2 - x_1} f(G_1), \text{ where } S_2 = (x, y_2), \quad (17)$$

$$f(G) = \frac{y_2 - y}{y_2 - y_1} f(S_1) + \frac{y - y_1}{y_2 - y_1} f(S_2). \quad (18)$$

In this way, new robot behaviours are obtained through interpolation from the elementary primitive set. An experiment is shown in Figure 8 to demonstrate this. Three primitives from Table 2 (P_{1_s}, P_{12}, P_{13}) are interpolated in order to obtain a new robot behaviour (new control parameters), using Equations (16–18) for the new goal (2m, 1m), as shown in Figure 8(d).

6. Experiments for Complex Tasks

This section presents Experiments (6–10), demonstrating how the robot handles complex tasks – like goal-directed obstacle avoidance, goal-directed locomotion with body deformation, and tasks with multiple goals – using the developed framework. We encourage readers also to view the supplementary video of these experiments, available at http://manoonpong.com/IJARS2015/svideo.mpeg.

6.1 Goal-Directed Locomotion with Obstacle Avoidance

The experiments in Figure 9 show goal-directed locomotion with and without obstacles. Figure 9(a) shows the experiment without any obstacles; here, the robot uses only one motor primitive (taken as Set 1) to reach the goal (-3m, -1 m). Set 1 consists of θ_1 = -0.26rad/s, θ_2 = 0.42rad/s, θ_3 = -0.29rad/s, and θ_4 = 0.34rad/s; and ϕ_1=0.05rad, ϕ_2=0.16rad, and ϕ_3=−0.02rad. Figure 9 (b) shows the experiment with obstacles. The obstacles are avoided here by sensing through an IR sensor attached to the robot's head. To achieve this, three robot behaviours are sequentially chained for avoiding obstacles and moving to the goal. From the primitive library in Table 2, two motor primitives – P_{2_s} (which makes the robot move left, i.e., to 90°) and P_{6_s} (which makes it move right, i.e., to -90°) – are selected and used. The robot starts with a learned parameter set (i.e., Set 1) for reaching the original goal (-3m, -1m). The IR sensor provides feedback on the distance between the obstacle and the robot. Once the robot has detected the obstacle (i.e., the IR signal is higher than the threshold), it automatically selects a predefined motor primitive (i.e., P_{2_s} for moving to the left), so as to avoid the obstacle. Once the obstacle has been avoided (i.e., the IR signal is smaller than a threshold), the robot then returns to its previous motor primitive (i.e., moving straight towards the goal with Set 1). Following a predefined time-out period after moving straight, it then selects another motor primitive (i.e., P_{6_s} moving to the right) and finally reaches the goal. In this way, the primitives are chained and activated using the sensor and a time-out period mechanism.

Another experiment in obstacle avoidance is also shown in Figure 10. Two primitives – P_{4_s} (to move straight) and P_{2_z} (to move left) – are selected from Table 2 and chained, again

Figure 8. Experiment 5: Generating new behaviours with parameter interpolation. Start position is (0m, 0m). (a) Primitive P_{1_s} from the library. It is generated using the goal (2m, 2m) for a straight-line shape. (b) Primitive P_{12} from the library, which is generated using the goal (0m, 2m) for a semi-zigzag shape, having ϕ_1 = 0.5rad, ϕ_2 = -0.5rad, and ϕ_3 = -0.1rad. (c) Primitive P_{13} from the library, which is generated using the goal (2m, 0m) for a semi-zigzag shape. (d) The robot reaches the goal (2m, 1m) using parameters obtained by interpolating the above primitives belonging to different robot shapes.

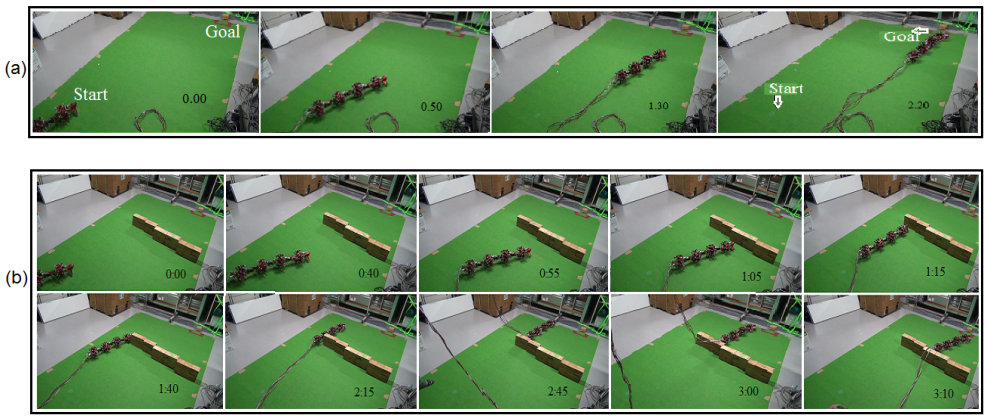

Figure 9. Experiment 6: Goal-directed locomotion of the real robot without and with obstacles in a complex environment. (a) Real robot experiment without obstacles, showing the robot reaching the goal (-3m, -1m) using the learned control parameters. Start position is (0m, 0m). (b) Goal-directed obstacle avoidance behaviour in a real robot experiment. The robot reaches the goal (shown in red) while avoiding obstacles on its path. The robot behaviour is driven by chaining three motor primitives, obtained through PI^2.

using the IR sensor attached to the robot's head. The P_{4_s} primitive is used by the robot when there is no obstacle detected (i.e., the IR signal is lower than a threshold) and P_{2_s} as soon as an obstacle is sensed.

However, further sensors can be added to render the locomotion fully sensor-driven. Additionally, a planner that analyses the scene (as far as is visible) and then automatically selects motor primitives to approach the goal and/or avoid obstacles can be implemented (as an additional module in the framework), using any planning method or reactive/proactive control method [31]. An example planner is presented below.

6.1.1 A symbolic planning example using primitives

Here, we briefly show an example of how a STRIPS-like planner [32] could be used to generate a plan (see Section 5.1.1). The task at hand is obstacle avoidance. The POs for this task (in terms of (a,p,e)) are defined in Table 3. Here, T shows that the predicate takes a "True" value, and F, "False". With the move(), the corresponding primitive (e.g., P_{2_s} and

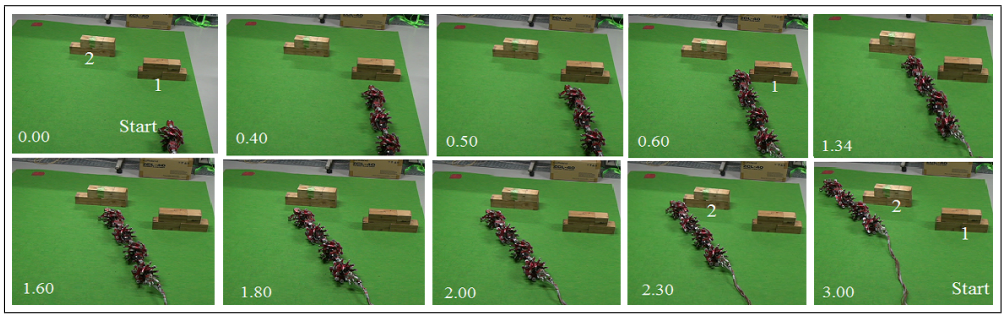

Figure 10. Experiment 7: Real robot experiment demonstrating obstacle avoidance. All primitives are sequenced using IR sensory feedback. The numbers of the obstacles are marked; two obstacles can be seen being avoided at the shown timestamps.

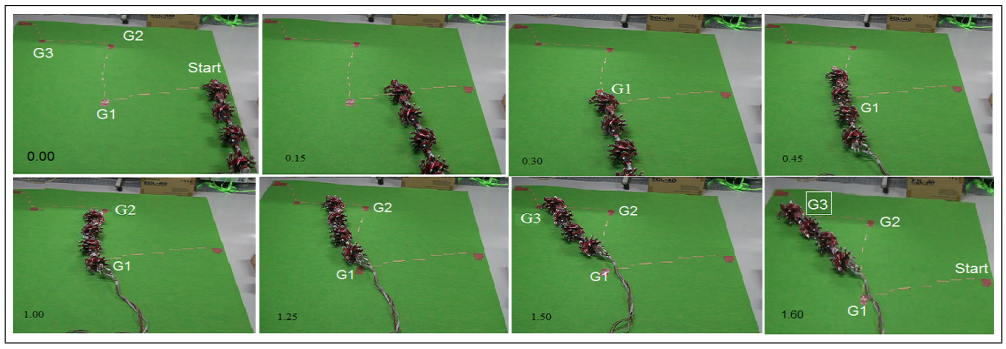

Figure 11. Experiment 8: Real robot movement through multiple goals. The followed trajectory is indicated by the dotted lines and the goals are marked in red. The robot reaches the final goal via G1, G2 and G3. The three motor primitives giving the three control parameter sets are sequentially activated.

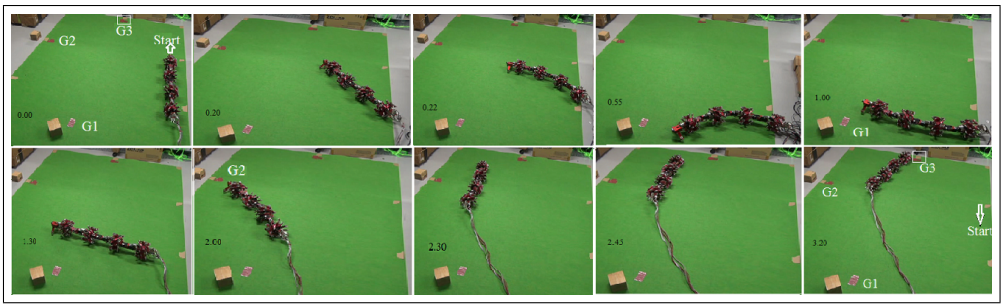

Figure 12. Experiment 9: The robot reaches multiple goals (G1, G2, G3), which are marked. The chaining of the robot behaviours is $IM_1 \rightarrow IM_2 \rightarrow IM_3 \rightarrow P_{5_{pe}} \rightarrow P_{4_{pe}}$. IM_1 (obtained from interpolating P_{8_s}, P_{1_s} and P_{2_s}), which gives the robot behaviour for moving to 120° diagonally in a straight shape, is chained with IM_2 (obtained from interpolating P_{1_a}, P_{2_a} and P_{8_a}). This is followed by IM_3 (obtained from interpolating P_{4_a}, P_{5_a} and P_{6_a}), for moving the robot to −30° with a bent arc-like configuration, with all its joint angles negative. IM_3 is used to change the robot's direction. In this way, G1 is reached. To reach G2, the motor primitive $P_{5_{pe}}$ for moving to −30° diagonally with periodic body movements is used. In order to reach the final goal (G3), $P_{4_{pe}}$, for moving the robot to −90° vertically downward with periodic body movements, is used.

P_{5_s}, in the case of the example shown in Figure 9(b)) is selected from Table 2 for action grounding: PO_1 takes the robot straight, PO_2 takes it left, and PO_3 takes it right.

To show how these operators can be searched and sequenced, the initial scenario shown in Figure 9(b) is taken. For this scenario, the starting state S_{ini} is $\{F,F,F,F\}$, and accordingly, a plan is generated consisting of a single PO: PO_1. At the point where an obstacle is detected, a *replanning* operation is performed with the new initial state $S_{ini_{new}} = \{F,T,F,F\}$. The new plan to reach the goal is then PO_2,PO_1. Alternatively, at the point of the obstacle, PO_3,PO_1 would also be a possible new plan, as can be ascertained from the p in Table 3 and from $S_{ini_{new}}$. Similarly, for the initial setup in Figure 10, a plan PO_1 is generated at the start. When Obstacle 1 is detected, replanning occurs with $S_{ini_{new}} = \{F,T,F,T\}$, and the new plan is PO_2,PO_1. As, $S_{ini_{new}}$ is again $\{F,T,F,T\}$ when Obstacle 2 is detected, the new plan is again PO_2,PO_1, which thus allows the robot to reach the goal while avoiding the obstacles. In this way, a planner that uses the learned primitives obtained by PI^2 can be integrated into the framework for goal-directed locomotion.

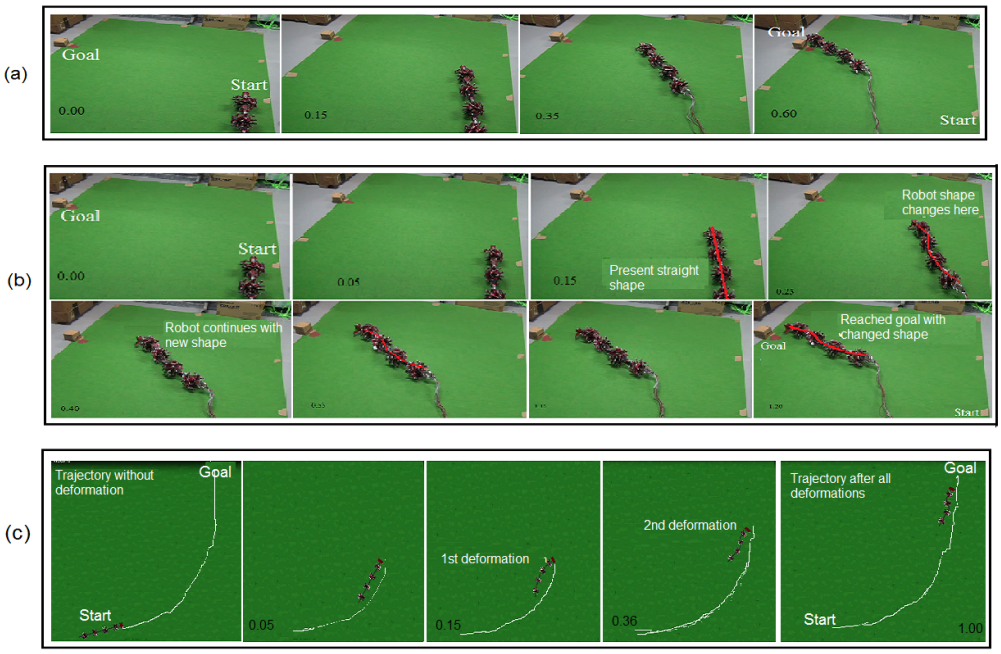

Figure 13. Experiment 10: Goal-directed locomotion while handling changes in body shape. (a) Robot reaches the goal without deformation, maintaining ϕ_i =0rad throughout its locomotion. The goal is (-1m, 2m) and the start position (0m, 0m).(b) Even when its body shape changes on its path at 0.25min, it continues with the new shape to reach the same goal (-1m, 2m). It uses the corresponding motor primitives/robot control for the new body shape to handle this change. (c) Simulation showing that the robot can successfully handle multiple body shape changes on its route, and still reach the goal. The first snapshot shows the robot reaching the goal (-2m, 3m) by moving at 60°, without any deformation and maintaining ϕ_i =0rad throughout. Two deformations – the first from a straight to an arc shape, followed by a change to a semi-zigzag (at both instants, the robot going to 20°) – are handled using two derived motor primitives, as shown in the second snapshot onwards. The pictures also show how the trajectory (the white line) is maintained even when there is deformation. This points to a systematic primitive library formation, which can be used as required.

	Definition
PO_1	a: move(angletoGoal, shape)
	p: obstacle(angletoGoal) = *F*, ongoal(robot) = *F*
	e: ongoal(robot)= *T*
PO_2	a: move(angletoGoal+90°, shape)
	p: obstacle(angletoGoal+90°) = *F*,
	obstacle(angletoGoal)= *T*
	e: obstacle(angletoGoal)= *F*
PO_3	a: move(angletoGoal-90°, shape)
	p: obstacle(angletoGoal-90°)= *F*,
	obstacle(angletoGoal)= *T*
	e: obstacle(angletoGoal)= *F*

Table 3. Planning Operators for Obstacle Avoidance

6.2 Goal-Directed Behaviour with Multiple Goals

Here, the goal-directed behaviour of the robot is demonstrated as it is made to move through multiple goals. Figure 11 shows an experiment in which the robot moves through multiple sub-goals (G1 and G2) in order to reach the final goal (G3). Three primitives (P_{1_s},P_{4_z},P_9) are manually selected from Table 2 for this task, and their final chaining sequence is $P_{1_s} \rightarrow P_{4_z} \rightarrow P_9$. The chaining can be described as follows: the robot uses P_{1_s}, the motor primitive for moving to 135 with a straight-line shape, to reach the first goal (G1) from the start; this is followed by P_{4_z}, for moving straight with a zigzag shape, to reach the second goal (G2); finally, the robot uses P_9, for moving to 90° upward with a semi-arc shape, to reach the final goal (G3). Another multiple goal experiment is shown in Figure 12. Primitives of different shapes, and both nonperiodic and periodic, are selected in these experiments to demonstrate the possibility of chaining between different robot configurations.

6.3 Goal-Directed Locomotion with Body Deformation

In these experiments, the robot successfully moves toward a goal even when it has suffered body deformation, i.e.; changes to its body shape, on its way. A single deformation is shown in Figure 13(b), whereas Figure 13(c) shows multiple deformations in body shape.

Figure 13(a–b) uses three motor primitives – P_{10}, P_{11}, P_{12} – which are manually selected from Table 2 for the task. Figure 13(a) shows the robot behaviour when there is no deformation on its way to the goal (-1m, 2m). The learned parameters for this situation, taken as Set 1, are θ_1 = -0.38rad/s, θ_2 = -0.05rad/s, θ_3 = -0.42rad/s and θ_4 = 0.06rad/s, with ϕ_1=ϕ_2=ϕ_3=0. Figure 13(b) shows the robot behaviour when its body shape changes on the way toward the goal (-1m, 2m), while it is using the above Set 1 parameters. When it detects a change in its body shape, using its joint

angle sensors, it uses another robot behaviour, IM_4, to enable it to handle this morphological change. IM_4 (belonging to the semi-zigzag robot shape of ϕ_1 = 0.5rad, ϕ_2 = -0.5rad, and ϕ_3 = -0.1rad) is generated using parameter interpolation of the primitives P_{10}, P_{11} and P_{12}, in order to move to 30°. The interpolation is performed following Equations (16–18). The primitives to be interpolated are selected based on the new deformed robot shape and the angle to the goal at the instant that deformation takes place. Interpolation is used to generate new behaviours, as no suitable primitive previously existed. Using the derived behaviour IM_4, the robot moves with this new semi-zigzag shape and is able to finally reach the goal.

7. Conclusion

We have successfully developed a framework that provides a model-free goal-directed locomotion controller for a snake-like robot with screw-drive units. The framework handles a large number of behavioural cases (within a defined scope). With the complete framework for generating motor primitives using PI^2, along with parameter interpolation and the chaining of primitives and/or interpolated parameters, the robot can successfully perform goal-directed locomotion in different situations. The framework thus generalizes the locomotion generation for the complex nonstandard snake-like robot.

Real robot experiments show that using the complete framework enables the robot to successfully handle challenging tasks like reaching a single/multiple goal(s) while avoiding obstacles or compensating for a morphological change (such as body damage) during locomotion. Furthermore, it has also been shown that, by learning a proper combination of locomotion control parameters (i.e., screw velocities and joint angles using PI^2), motor primitives can be generated in a numerically simple manner. Proper control parameters were also found for when the robot was configured with different shapes (i.e., straight-line, zigzag, arc, etc.) or with periodic body movements, thereby establishing a rich primitive library which could then be used. Thus, this framework and approach solves the coordination problems relating to such a high degree-of-freedom system as this nonstandard snake-like robot. As a result, the robot is able to reach a given goal in different situations. In addition, this study also shows how PI^2 can be used to learn motor control for this type of nonstandard snake-like robot.

In some of the real robot experiments, a small deviation (i.e., approximately 25cm) from the goal was observed. This deviation was due to real conditions, such as friction, cabling, etc. Major changes, like the variation of the friction coefficient, can be handled by relearning the existing motor primitives. However, sensory signals (e.g., joint angles, goal detection, slip detection, etc.) are required in order to enable the robot to perform online learning autonomously, which can then adjust or optimize the parameters of our motor primitives to deal significantly with body and environmental changes. Implementing such sensors with online learning is one of our future plans.

8. Acknowledgements

This research was supported by Emmy Noether grant MA4464/3-1 of the Deutsche Forschungsgemeinschaft, the Bernstein Center for Computational Neuroscience II Goettingen (BCCN grant 01GQ1005A, project D1), and the HeKKSaGOn network. We would like to thank Dr. Alejandro Agostini for his assistance and help in improving this article. This article is a revised and expanded version of a paper entitled *Reinforcement Learning Approach to Generate Goal-directed Locomotion of a Snake-Like Robot with Screw-Drive Units*, presented at RAAD 2014, 23rd Int. Conference on Robotics in Alpe-Adria-Danube Region, Smolenice Castle, Slovakia, 3–5 September 2014.

9. References

[1] Gray J. (1946) The Mechanism of Locomotion in Snakes. The Journal of Experimental Biology, 23(2): 101–120.

[2] Hirose S. (1993) Biologically Inspired Robots: Snake-Like Locomotors and Manipulators. Oxford University Press, Oxford.

[3] Liljebäck P. et al. (2012) A review on modelling, implementation, and control of snake robots. Robotics and Autonomous Systems, 60(1):29–40.

[4] Lu Z. et al. (2005) Serpentine locomotion of a snake-like robot controlled by cyclic inhibitory CPG model. 2005 IEEE/RSJ Int. Conference on Intelligent Robots and Systems, pp. 96–101.

[5] Miller G. (2002) Snake robots for search and rescue, In Ayers J., David J. L. and Rudolph A. (eds.) Neurotechnology for Biomimetic Robots. MIT Press, Cambridge, MA, pp. 271–284.

[6] Kakogawa A., Nishimura T. and Ma S. (2013) Development of a screw drive in-pipe robot for passing through bent and branch pipes. 44th Int. Symposium on Robotics, pp. 1–6.

[7] Transeth A. A. and Pettersen K. Y. (2006) Developments in Snake Robot Modeling and Locomotion. 9th Int. Conference on Control, Automation, Robotics and Vision, pp. 1–8.

[8] Fukushima H. et al. (2012) Modeling and Control of a Snake-Like Robot Using the Screw-Drive Mechanism. IEEE Transactions on Robotics, 28(3):541–554.

[9] Theodorou E. A., Buchli J. and Schaal S. (2010) A generalized path integral control approach to reinforcement learning. Journal of Machine Learning Research, 11:3137–3181.

[10] Chatterjee S. et al. (2014) Reinforcement Learning Approach to Generate Goal-directed Locomotion of a Snake-Like Robot with Screw-Drive Units. 23rd

Int. Conference on Robotics in Alpe-Adria-Danube Region, IEEE catalog no. 34043.

[11] Theodorou E. A., Buchli J. and Schaal S. (2010) Reinforcement learning of motor skills in high dimensions: A path integral approach. IEEE Int. Conference on Robotics and Automation, pp. 2397–2403.

[12] Pastor P. et al. (2011) Skill learning and task outcome prediction for manipulation. IEEE Int. Conference on Robotics and Automation, pp. 3828–3834.

[13] Stulp F. and Schaal S. (2011) Hierarchical reinforcement learning with movement primitives. 11th IEEE-RAS Int. Conference on Humanoid Robots, pp. 231–238.

[14] Kober J., Bagnell J. A. and Peters J. (2013) Reinforcement learning in robotics: A survey. International Journal of Robotics Research. 32(11):1238–1274.

[15] Whiteson S. (2012) Evolutionary Computation for Reinforcement Learning. In Wiering M. and van Otterlo M. (eds) Reinforcement Learning: State-of-the-Art. Springer, Berlin, pp. 325–355.

[16] Eberhart R. C. and Shi, Y. (2005) Comparison between genetic algorithms and particle swarm optimization. Evolutionary Programming VII, Lecture Notes in Computer Science, 1447:611–616.

[17] Pedersen M. E. H. (2010) Good Parameters for Particle Swarm Optimization. Technical Report no. HL1001, Hvass Laboratories, pp. 1–12.

[18] Das S., Abraham A. and Konar A. (2008) Particle Swarm Optimization and Differential Evolution Algorithms: Technical Analysis, Applications and Hybridization Perspectives. Advances of Computational Intelligence in Industrial Systems, 116:1–38.

[19] Panduro M. A. et al. (2009) A comparison of genetic algorithms, particle swarm optimization and the differential evolution method for the design of scannable circular antenna arrays. Progress In Electromagnetics Research B, 13:171–186.

[20] Grigoris S. et al. (2012) Integrating particle swarm optimization with reinforcement learning in noisy problems. 14th Annual Conference on Genetic and Evolutionary Computation, pp. 65–72.

[21] Flash T. and Hochner B. (2005) Motor primitives in vertebrates and invertebrates. Current Opinion in Neurobiology, 15(6):1–7.

[22] Mussa-Ivaldi F. A. and Bizzi E. (2000) Motor learning through the combination of primitives. Philosophical Transactions of the Royal Society: Biological Sciences, 355:1755–1769.

[23] Thoroughman K. A. and Shadmehr R. (2008) Learning of action through adaptive combination of motor primitives. Nature, 407(6805):742–747.

[24] Fod A., Mataric M. J. and Jenkins O. C. (2002) Automated derivation of primitives for movement classification. Autonomous Robots, 12(1):39–54.

[25] Lim B., Ra S. and Park F. C. (2005) Movement Primitives, Principal Component Analysis, and the Efficient Generation of Natural Motions. Int. Conference on Robotics and Automation, pp. 4630–4635.

[26] Kober J. and Peters J. (2009) Learning motor primitives for robotics. IEEE Int. Conference on Robotics and Automation, pp. 2112–2118.

[27] Peters J. and Schaal S. (2008) Reinforcement learning of motor skills with policy gradients. Neural Networks, 21(4):682–697.

[28] Tamosiunaite M. et al. (2011) Learning to pour with a robot arm combining goal and shape learning for dynamic movement primitives. Robotics and Autonomous Systems, 59(11):910–922.

[29] Ghallab M., Nau D. and Traverso P. (2004) Automated Planning: Theory and Practice. Morgan Kaufmann, Burlington, MA, pp. 1–635.

[30] Quack, B. and Wörgötter, F. and Agostini, A. (2015). Simultaneously Learning at Different Levels of Abstraction. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS 2015), 4600-4607.

[31] Porr B. and Wörgötter F. (2006) Strongly improved stability and faster convergence of temporal sequence learning by utilising input correlations only. Neural Computation, 18(6):1380–1412.

[32] Fikes R. E. and Nilsson N. J. (1971) STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. Artificial Intelligence, 2:3–4.