Neural Comput & Applic (2009) 18:369–375
DOI 10.1007/s00521-008-0187-1

ORIGINAL ARTICLE

A small spiking neural network with LQR control applied to the acrobot

Lukasz Wiklendt · Stephan Chalup · Rick Middleton

Received: 8 July 2007 / Accepted: 8 April 2008 / Published online: 10 May 2008

Abstract This paper presents the results of a computer simulation which, combined a small network of spiking neurons with linear quadratic regulator (LQR) control to solve the acrobot swing-up and balance task. To our knowledge, this task has not been previously solved with spiking neural networks. Input to the network was drawn from the state of the acrobot, and output was torque, either directly applied to the actuated joint, or via the switching of an LQR controller designed for balance. The neural network's weights were tuned using a $(\mu + \lambda)$-evolution strategy without recombination, and neurons' parameters, were chosen to roughly approximate biological neurons.

Keywords Spiking neural networks · Acrobot · LQR · Evolution

1 Introduction

We are studying the applicability of spiking neural networks (SNNs) to the control of robots with complex nonlinear morphologies that lead to unstable states requiring constant active feedback control. This is difficult using conventional control due to the effort required in modeling the robot's equations of motion and deriving a robust control scheme on the basis of that model.

Theoretical results of Maass [14] have shown that SNNs (third generation NNs) are computationally more powerful than standard sigmoid NNs (second generation NNs) or networks of threshold units (perceptrons or first generation NNs) provided certain conditions hold. Such conditions require a delay-coded input which linearly maps an analog input signal to the neuron's fire time. As a first step we chose not to implement these conditions since they are biologically less plausible; but they will be considered for future studies. In the present study we use rate-coded input which maps an analog input signal to the rate at which the neuron fires.

Additional motivation to investigate SNNs is that they are better models to represent the spiking nature of biological neurons, which are used in the mechanical control of biological systems. Recently, networks of 10,000 spiking neurons have been used in the large-scale implementations of the blue brain project [16] to model cortical columns of the brain. However, spiking neurons not only occur in the massive networks of the brain but also in relatively small networks of the peripheral regions of the nervous system such as reflex networks in the limbs [11], which can be modeled with small networks in the order of tens of neurons. It is an open question whether small networks of spiking neurons can successfully be employed to control limb movement and reflexes of unstable robots, such as for example, bi-ped robots. The aim of the present paper is to shed light on these questions by applying small SNNs to the control of an underactuated, simulated robot.

Primary work has been initiated on implementing SNNs as controllers for robots governed by simple dynamics, such as two wheeled robot cars, rather than underactuated unstable robots governed by highly nonlinear dynamics. Joshi and Maass [10] successfully used a SNN as the "liquid" of a liquid state machine to control a fully actuated two-link robot arm in the horizontal plane. The controller was created by learning a filter which transformed the state of the network to a control signal for the robot's motors, while the neurons and synaptic weights

L. Wiklendt (✉) · S. Chalup · R. Middleton
School of Electrical Engineering and Computer Science,
The University of Newcastle, Callaghan, NSW 2308, Australia
e-mail: lukasz.wiklendt@studentmail.newcastle.edu.au

remained constant throughout the learning process. Floreano et al. [6] evolved only the structure of a SNN with ten hidden neurons, while keeping the synaptic weights constant. Even with their greatly simplified model of spiking neurons, customized to fit the microcontroller on a small sugar cube sized two-wheeled robot, the robot was able to successfully navigate an oval track avoiding walls. French and Damper [7] assembled a SNN out of two network components, each evolved to achieve the particular subtask "frequency discrimination" of an overall objective to control a two-wheeled robot to drive towards flashing lights of distinct frequencies. Their evolution strategy allowed for networks of arbitrary cardinality, and neurons and synapses of various models. Federici [5] created a novel algorithm, which evolved the rules of development of a SNN that was used to grow the network from a single cell to its final structure. The technique was applied to a wall-avoidance task for a two-wheel robot with infrared sensors.

Rather than applying SNNs to stable robotic platforms such as the ones mentioned previously, we instead apply them to an unstable robot with highly nonlinear dynamics. The acrobot [19] is a two-link non-linear underactuated robotic platform, commonly used as a benchmark for new artificial intelligence approaches targeted at dynamics control. A drawing of the acrobot is shown in Fig. 1. Its motion resembles that of a gymnast swinging on a horizontal bar, the difference being that unlike the gymnast the acrobot can spin its q_2 joint "hip" through complete revolutions. The object of the task is to swing the acrobot from a hanging-down position to a standing-up position. This is quite difficult as only the q_2 joint is actuated. A more formal description of the acrobot and the swing-up task is given in Sect. 2.

One of the first to approach the acrobot swing-up control problem was Spong [19]. He created two swing-up strategies based on the partial feedback linearization of the acrobot dynamics, one for each joint. He applied each strategy with a linear quadratic regulator (LQR) to balance the acrobot in the standing position, and both strategies produced a successful swing-up and balance. Boone [3] used an N-step lookahead search to select the appropriate torque which first added the required amount of energy to the acrobot which would allow it to reach its standing-up position. Once this was achieved, the search objective was changed to find a trajectory which would get the acrobot close to the standing-up state. Torque output was only one of two possible extreme values, either positive or negative 1 Nm, which is known as *bang-bang* control. Allowing only two possible output values, and limiting the amount of switching between these values reduced the search to a practical size. Yoshimoto et al. [21] successfully applied reinforcement learning to the task, which switched between one of five conventional controllers.

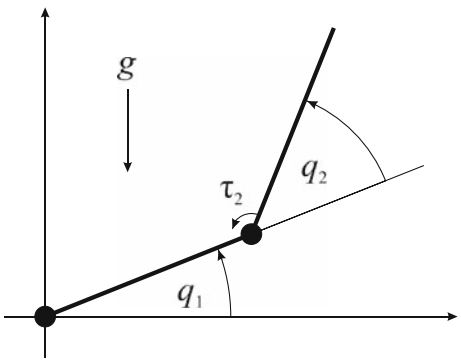

Fig. 1 The acrobot showing directions for gravity, torque and joint angles

The area of acrobot swing-up control is quite advanced with many other existing successful techniques in addition to those mentioned previously, including sigmoid NN function approximation for reinforcement learning [4, 20], evolving a non-feedback vector of torque values [12], a fuzzy controller used to increase the acrobot's energy [13], output zeroing based on angular momentum and rotation angle of center-of-mass [17]. Although the primary focus of this study is the application of SNNs as control models for complex robotic platforms, it has given some specific insights into the techniques which may be used to improve acrobot swing-up solutions.

This paper is organized as follows. Section 2 describes the acrobot and the swing-up task used in this study. Section 3 describes SNNs and the discrete time model that was used for simulations. Section 4 describes the network configuration and the LQR controller used in simulations. Section 5 presents the evolution and simulation setup. A discussion of the results is given in Sect. 6, with conclusions following in Sect. 7.

2 The acrobot

The acrobot is composed of two links, an inner and outer link connected together by an actuated hinge joint, with their relative angle given by q_2. The inner link is anchored with an unactuated hinge joint, and subtends an angle of q_1 to the horizontal. The torque applied at the actuated hinge is given by τ_2, which is limited to $|\tau_2| \leq \tau_{2_{\max}}$. Angular velocities of joints q_1 and q_2 are $\dot{q}_1$ and $\dot{q}_2$. The state of the acrobot at time t is given by the vector $\boldsymbol{x}(t) = (q_1(t)\ q_2(t)\ \dot{q}_1(t)\ \dot{q}_2(t))^{\mathrm{T}}$. There are two notable equilibria, an unstable one at $\boldsymbol{q}_u = (\frac{\pi}{2}\ 0\ 0\ 0)^{\mathrm{T}}$, the other stable at $\boldsymbol{q}_s = (-\frac{\pi}{2}\ 0\ 0\ 0)^{\mathrm{T}}$.

Acrobot Task: Given the initial state $\boldsymbol{x}(0) = \boldsymbol{q}_s$, *find a control strategy which will get the acrobot to the final state* $\boldsymbol{x}(T) = \boldsymbol{q}_u$, *and keep it there as* $t \to \infty$.

In this study $T = 20$ s was chosen, after a series of pilot experiments which showed this value to induce a favorable learning rate without excessive computation time. The acrobot task is considered a difficult control problem because the acrobot is underactuated and has highly nonlinear dynamics.

The equations of motion governing the dynamics of the acrobot are presented in [19]. Masses of the links are given by m_i, lengths l_i, and inertia tensors I_i, where $i = 1,2$ for the inner and outer links, respectively. Acceleration due to gravity is given by g. The parameters used for simulation are listed in Table 1.

The acrobot is simulated using a fourth-order Runge-Kutta integrator [18] with a time step of 1 millisecond.

3 Spiking networks

This section gives an introduction to SNNs, including the models used in our simulations. The word "spiking" will often be omitted here when mentioning spiking neurons or networks.

Neurons and synapses in a SNN are arranged as nodes and edges of a directed graph, respectively. Neurons send and receive data via the timing of spike events which are sent across synapses from a source neuron to a destination neuron. Each neuron i contains its own state u_i representing the exponentially decaying membrane potential of biological neurons. When a spike reaches a neuron then that neuron's membrane potential is momentarily raised (or lowered) in proportion to the postsynaptic potential function ϵ. If the membrane potential rises above the threshold θ then the neuron fires a spike to its destination neurons, and the neuron's membrane potential is decreased by η for a refraction period, thereby preventing more immediate firings.

In biological NNs spikes require a short amount of time to travel from their source neuron to the destination neurons. However, in this study, spikes are not subjected to an explicit synaptic delay. Synapses have a "weight" value w_{ij}, which determines the amount a spike from a source neuron j increases a destination neuron's membrane potential u_i. It is these weights which are tuned during learning. In future studies tunable delays will also be used since they have the potential to increase the computational power of SNNs [14].

The leaky integrate-and-fire model for neurons [8, 9] is used in our simulations since it is concise, simple to implement and fast to simulate. To reduce the number of parameters that need tuning during learning all of the neurons are made homogeneous. The precise neuron model is shown in Eqs. (1)–(4).

$$u_i(t) = \sum_{t_i \in \mathcal{F}_i(t)} \eta(t - t_i) + \sum_j \sum_{t_j \in \mathcal{F}_j(t)} w_{ij}\epsilon(t - t_j) \tag{1}$$

$$\mathcal{F}_i(t) = \{t_i \mid u_i(t_i) > \theta,\ t_i < t\} \tag{2}$$

$$\eta(s) = -v \exp\left(\frac{-s}{\tau_\rho}\right) \tag{3}$$

$$\epsilon(s) = \exp\left(\frac{-s}{\tau_\mu}\right) - \exp\left(\frac{-s}{\tau_\sigma}\right) \tag{4}$$

where $\mathcal{F}_i(t)$ contains all firing times for any neuron i that occur before time t, and v, τ_ρ, τ_μ, τ_σ are constants which specify the size and shape of the refraction function η and the postsynaptic potential function ϵ. This neuron model is approximated in discrete time to simplify implementation and allow for easy integration with the discrete time simulation of the acrobot.

We use a discrete-time model governing the dynamics of a neuron i, given by Eqs. (5)–(8).

$$\boldsymbol{v}_i(t + \Delta t) = \boldsymbol{A}\boldsymbol{v}_i(t) + \boldsymbol{B}\boldsymbol{u}_i(t) \tag{5}$$

$$\boldsymbol{A} = \begin{pmatrix} d_\mu & 0 & 0 \\ 0 & d_\sigma & 0 \\ 0 & 0 & d_\rho \end{pmatrix} \quad \boldsymbol{B} = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & v \end{pmatrix} \quad \boldsymbol{u}_i(t) = \begin{pmatrix} \kappa_i(t) \\ f_i(t) \end{pmatrix} \tag{6}$$

$$f_i(t) = \begin{cases} 1, & \text{if } (1 \ -1 \ -1)\boldsymbol{v}_i(t) > \theta \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

$$\kappa_i(t) = \sum_j w_{ij} f_j(t) \tag{8}$$

where d_μ, d_σ, $d_\rho \in (0,\ 1)$, $v > 0$, and $\theta \geq 0$. The values $d_\mu = 0.9$, $d_\sigma = 0.8$, $d_\rho = 0.9$, $v = 1$, and $\theta = 1$ are used in the simulations presented in this study. Also, a time step of $\Delta t = 1$ ms is used, and t is an integer multiple of Δt. These values were chosen by hand so that the neurons imitate (quite roughly) dynamics such as firing rates and postsynaptic potential durations of biological neurons [11].

The values d_μ, d_σ and d_ρ can be derived from τ_μ, τ_σ and τ_ρ in Eqs. (3) and (4) with the following relation

$$d_\alpha = \exp\left(\frac{-\Delta t}{\tau_\alpha}\right), \quad \text{for } \alpha \in \{\mu, \sigma, \rho\}. \tag{9}$$

The state of neuron i is given by the time-varying vector $\boldsymbol{v}_i = (v_1\ v_2\ v_3)^{\mathrm{T}}$, where $\boldsymbol{v}_i(0) = \boldsymbol{0}$. The value of the inner product $(1\ -1\ -1)\ \boldsymbol{v}_i$ in Eq. (7) represents the membrane potential of neuron i, and θ is the threshold. When the membrane potential rises above the threshold then the neuron fires by setting f_i to 1, and the membrane potential is consequently reduced by the refraction constant v in the

Table 1 Acrobot parameters, where all units are in SI

m_1	m_2	l_1	l_2	I_1	I_2	g	$\tau_{2_{\max}}$
1	1	1	1	1/12	1/12	9.8	10

next step. The f_i term can be thought of as the spiking "output" of neuron i.

The κ_i term in Eq. (8) is the "input" to neuron i. The sum is over all source neurons of neuron i, and w_{ij} is the synaptic weight from neuron j to neuron i. If $w_{ij} > 0$ then a spike arriving at a neuron increases the neuron's membrane potential (with delayed onset since $d_\mu > d_\sigma > 0$). This represents an *excitatory post-synaptic potential* and its practical purpose is to increase the chance of the neuron firing. If $w_{ij} < 0$ then an incoming spike reduces the membrane potential, which represents an *inhibitory post-synaptic potential* and decreases the chance of the neuron firing.

This model is used for the hidden neurons of the network, that is, neurons that are connected only to other neurons. Sensor and motor neurons used in our simulations have a slightly different model as they must also receive input from and send output to the environment.

3.1 Motor and sensor neurons

A hidden neuron can be used to model a motor neuron, where the motor neuron's analog output is given by

$$(1 \ -1 \ 0)\boldsymbol{v}_i(t). \tag{10}$$

However this is a verbose way to model a motor neuron, since the d_ρ and v parameters are no longer used because the neuron is never required to fire spikes.

A sensor neuron can be modeled by replacing $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{u}_i$ in Eqs. (5) and (6) with $\boldsymbol{A}_s$, $\boldsymbol{B}_s$ and $\boldsymbol{u}_{s_i}$ respectively

$$\boldsymbol{A}_s = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & d_\rho \end{pmatrix} \quad \boldsymbol{B}_s = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & v \end{pmatrix} \quad \boldsymbol{u}_{s_i}(t) = \begin{pmatrix} \kappa_{s_i}(t) \\ f_i(t) \end{pmatrix} \tag{11}$$

where the analog input to a sensor neuron is set to κ_{s_i}, eliminating equation (8). For brevity the time-varying analog input to a sensor neuron will be referred to as κ. In simulations the constraint $\kappa \geq 0$ is applied. A value of $\theta = 0$ is used for all sensor neurons, since setting $\theta > 0$ results in a dead-zone for inputs $\leq \theta$. This form of encoding analog input into spikes is commonly known as *rate encoding*, where the firing rate of a sensor neuron is proportional to the input. Another method for encoding is called *delay encoding* where the average firing rate remains constant and the precise time at which the neuron fires is proportional to the input.

A notable property of modeling sensor neurons this way is that their output scales either logarithmicaly or linearly, depending on the magnitude of the input. Assuming a constant input κ to a sensor neuron, the spiking rate of the neuron can be calculated by finding the time between spikes. The state of the sensor neuron is initialized as if it had just spiked, and the time it takes for the v_3 component of the state $\boldsymbol{v}$ to decay to the next spike is calculated using the equation $d_\rho^y(v + \kappa) = \kappa$, where y is the number of milliseconds between spikes. Therefore, the spike rate r (spikes per millisecond) of a sensor neuron for a constant input $\kappa > 0$ is given by the formula

$$r(\kappa) = \frac{1}{\log_{d_\rho}\left(\frac{\kappa}{v+\kappa}\right)} \tag{12}$$

noting that $d_\rho \in (0,1)$ and $v > 0$. For large values of κ the approximation $r(\kappa) \approx k\kappa$ is observed, where the constant $k > 0$, since

$$\lim_{\kappa \to \infty} \frac{r(m\kappa)}{r(\kappa)} = m \tag{13}$$

and for small values of κ the approximation $r(\kappa) \approx \frac{1}{\log_{d_\rho}(\kappa/v)}$ is observed. This allows a sensor neuron to remain sensitive to tiny inputs by having a spiking rate that is inversely proportional to the log of the input, without an exponential increase in sensitivity between large inputs. Although these properties are derived for extreme values of κ, in practice we notice logarithmic proportions for κ as large as 0.01, and linear proportions for κ as small as 1, since $d_\rho = 0.9$ and $v = 1$ in our simulations. However, this property of sensor neurons may be disadvantageous resulting in excessive output given small inputs. Such overly sensitive control is visible in the results, particularly during the acrobot's balance period (after approx. 3,300 ms) in the bottom plot of Fig. 5b.

4 Controller

A combined SNN and LQR [1] controller was used in this study to solve the acrobot task. A picture of the network topology is shown in Fig. 2.

The network contains eight sensor neurons, two for each element $x_i \in \boldsymbol{x}$ of the acrobot state. Each element x_i is split into a positive and negative sensor neuron, whose inputs are $\kappa = x_i$ and $\kappa = -x_i$, respectively. Without this treatment there would be no sensor information to the network about negative state values. Sensor neurons in Fig. 2 are labeled with the acrobot state variable which supplies the input to that neuron, appended with a " + " or "−" symbol to discern between positive and negative input signing.

The network contains two motor neurons. The torque motor neuron sends its output directly to the torque τ_2, and is modeled as mentioned in Sect. 3.1. The LQR controller is activated when it receives positive input ($\kappa > 0$) and deactivated when it receives negative input ($\kappa < 0$). Also, when the LQR neuron is active the torque motor neuron is

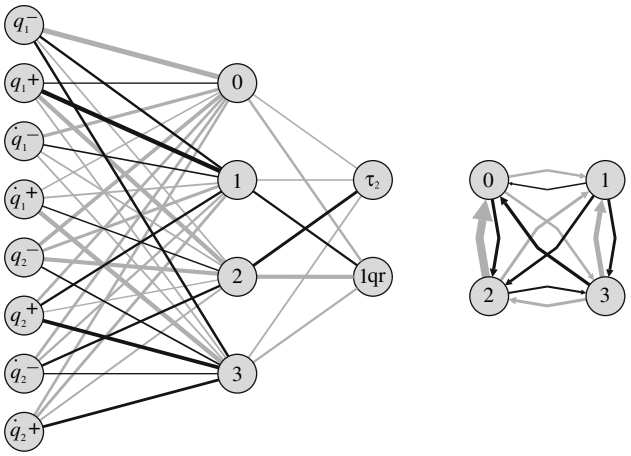

Fig. 2 Network topology and synaptic weights. Positive and negative weights are shown in *black and gray lines*, respectively. A synapse's thickness is shown in proportion to the size of its weight $|w_{ij}|$, where the *thinnest line* represents a weight size of 0.2, and the maximum and minimum weights are 17.4 and −37.7, respectively. The network's synaptic connections have been split into two diagrams for clarity. The diagram on the *left* shows connections from the sensor to the hidden neurons, and from the hidden to the motor neurons. The diagram on the *right* shows connections between hidden neurons only

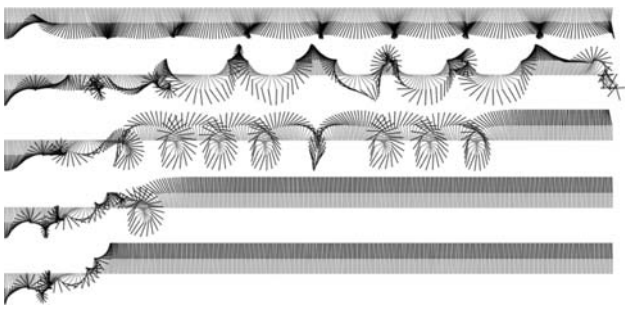

Fig. 3 Stroboscopic sequences of each of the fittest individuals from 5 generations of a single evolution. From top to bottom the generations are 0th, 5th, 11th, 21st and the elite solution at 41st. Each row represents a separate 20-s simulation with 50 ms between frames

disabled, and the LQR neuron takes control by setting its own value for τ_2.

There are also four hidden neurons which are completely connected with each other, without loop-back connections as shown in Fig. 2. They bridge the sensor and motor neurons and their recursive connections allow for complex dynamics.

4.1 LQR controller

When the acrobot is near the unstable equilibrium state $\boldsymbol{q}_u$ it can be kept there using a LQR [1]. To create the LQR the acrobot's equations of motion are linearized about the state $\boldsymbol{q}_u$ and the control law $\tau_2(t) = -\boldsymbol{K}\boldsymbol{x}(t)$ is optimized by minimizing the quadratic cost

$$\boldsymbol{J} = \int_0^\infty \left(||\boldsymbol{x}(t) - \boldsymbol{q}_u||^2 + \tau_2^2 \right) dt. \tag{14}$$

The approximate value of the gain matrix becomes $\boldsymbol{K} = (269.522 \quad 67.522 \quad 98.966 \quad 29.047)$ for the parameters given in Table 1.

The LQR controller is designed for a linearized version of the acrobot at the standing-up state $\boldsymbol{q}_u$, which is an approximation of the acrobot accurate to within only a small margin of $\boldsymbol{q}_u$. The linearization simplifies the acrobot model by making, within the equations of motion, replacements such as $\sin(\varepsilon) \to \varepsilon$, $\cos(\varepsilon) \to 1$ and $\varepsilon^2 \to 0$, for values of ε close to 0. This means the LQR controller does not function as intended for states too far from $\boldsymbol{q}_u$, which empirically corresponds to a few degrees from the standing-up state, and so is incapable of swinging-up and balancing the acrobot on its own from the initial state $\boldsymbol{q}_s$.

5 Evolution

This section covers the evolution strategy (ES) used for evolving the SNN controller. The SNN's weights were evolved using a $(\mu + \lambda)$-ES [2] for 100 generations, with $\mu = 10$ and $\lambda = 70$. To elaborate, there was a population of 10 parents producing 70 offsprings each generation, and the top 10 fittest of the combination of parents and offspring survived to make up the parents for the next generation. No recombination was used, and mutation of the synaptic weights occurred with an evolving standard deviation strategy parameter for each weight. This approach was chosen, after a series of pilot experiments, in order to produce successful swing-up strategies without excessive computation time.

The fitness of an individual was calculated from the cost $\boldsymbol{J}$ given by the equation

$$\boldsymbol{J} = \sum_{t=0}^{2\cdot 10^4} (\boldsymbol{x}(t) - \boldsymbol{q}_u)^{\mathrm{T}} \boldsymbol{Q}\ (\boldsymbol{x}(t) - \boldsymbol{q}_u) \tag{15}$$

where the matrix $\boldsymbol{Q}$ was tuned by hand and given by

$$\boldsymbol{Q} = \begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} \end{pmatrix} \tag{16}$$

to prioritize q_1 over q_2 and angles over angular velocities. Note, the closer the cost is to 0 the fitter the individual chromosome, and only the relative fitness of individuals in the population is important rather than the actual value of their cost.

The fitness for each individual chromosome was determined by transcribing it to the weights of the network and

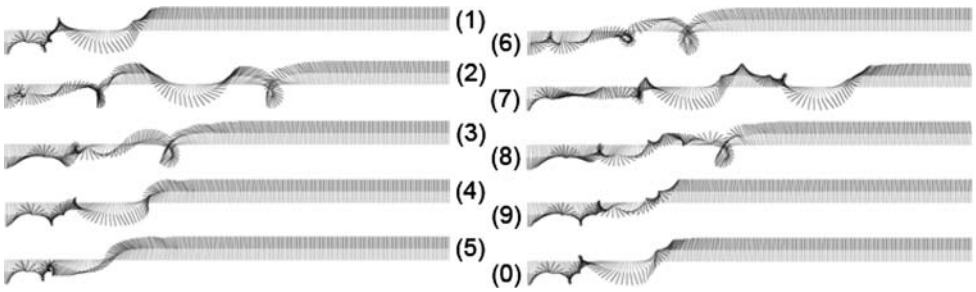

Fig. 4 The elite solution from each of the ten separate evolution runs, shown for the first 10 s of simulation

running a 20-s simulation (20,000 1-ms steps) of the acrobot under the control of the network. Although all elite individuals were able to swing-up and balance the acrobot within 10 s of simulation (Fig. 4), reducing the simulation time to 10 s during evolution resulted in a much reduced success rate. We believe this is due to low-performing early generations having inadequate time to complete the swing up and balance, since it was observed that the swing up and balance would occur much later in the simulation for those generations.

Since evolution strategies are stochastic, ten evolution runs were used to show that a successful strategy was not simply a coincidence of the initial population which was drawn at random from a normal distribution. After 100 generations, the fittest individual from the population was chosen as the solution of the learning strategy, and was given the title *elite*. Detailed results presented in Sect. 6 are for one of the ten elites produced from separate evolutions, where the other nine showed similar performance.

6 Simulation results and discussion

Each of the ten separate evolutions produced a network that was able to swing-up and balance the acrobot within the 20-second time period. One of the elite networks is shown in Fig. 2, and the simulation plots for this network are shown vertically aligned in Fig. 5. Figure 5a and b show the state of the acrobot as it swings up from time 0 to about 3,500 ms, and then remains balanced from 3,500 ms onwards.

Figure 3 presents five of the fittest individuals at different generations from an evolution run, and shows that early generations required more than 10 s to swing up.

The LQR was added since our initial evolutions produced a SNN controller that was only able to swing-up but not satisfactorily balance the acrobot in its standing state. When the LQR was finally introduced, rather than manually initiating it when the acrobot was close to its standing state, the network was given the chance to initiate the LQR at arbitrary times as mentioned in Sect. 4, expecting that it would only initiate around the standing state. However, as shown in Fig. 5b, the network used the LQR at states outside of its region of attraction. In the bottom plot of Fig. 5b we can see that during the swing-up phase only extreme values of torque produced by the LQR are exploited.

Due to the torque artifacts from the motor neuron induced by overly responsive sensor neurons, the network is not successful in keeping a steady balance, although it never loses balance entirely (visible in the bottom plot of Fig. 5b after about 3,300 ms).

Similarities between solutions of uncorrelated evolutions suggest that we have found solutions in the neighborhood of the optimal swing-up trajectory given the parameters governing the acrobot's dynamics used in this study. See Fig. 6 for a typically good swing-up strategy, whose various stages of control were commonly observed among the elite solutions. An observation-based

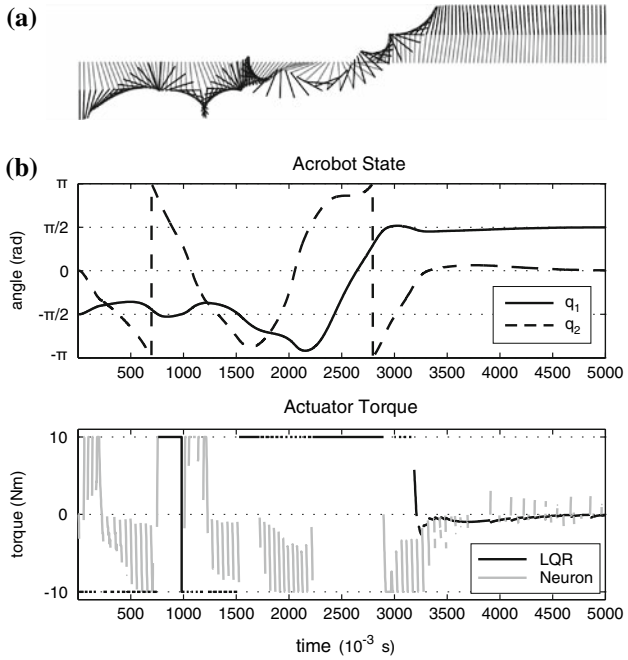

Fig. 5 Vertically aligned plots which depict the first 5 s of a 20 second simulation of an elite NN. **a** Stroboscopic sequence of the acrobot where *gray and black lines* represent the inner and outer links, respectively. **b** The acrobot state and actuation torque, where the *black torque* is based on the LQR controller and the *gray torque* is based on the motor neuron

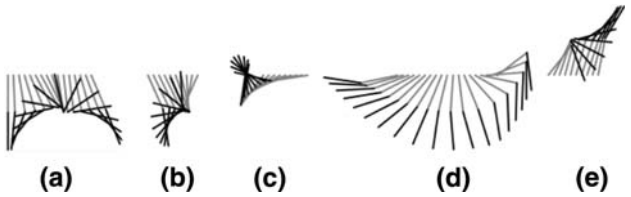

Fig. 6 Common swing-up trajectory: from the start of the simulation at **a** to the moment where balance is initiated at the end of **e**

qualitative analysis of each stage of control follows. (a) One complete revolution of the outer link to inject energy into the acrobot; (b) slow down the outer link consequently swinging up the inner link; (c) extend outer link to apply maximum torque via gravity to q_1 once it is extended; (d) keep the outer link extended to maximize the torque applied to the q_1 joint, and then contract the outer link to reduce slowing of the acrobot as it climbs; (e) extend the outer link, approaching the standing state for balance.

In addition to the experiments with spiking neurons we performed a series of experiments with standard sigmoidal networks. In our experiments sigmoidal networks were successful in solving the same task and seemed to perform at least comparably in terms of success rate. However, such a direct comparison of the two network types is not appropriate because spiking neurons required an input layer which maps analog input to spiking input, and in our simulations this mapping resulted in different network architectures.

7 Conclusion

We have shown in this study that a spiking neural network can be used with an evolutionary strategy to control an unstable and non-linear robot such as the acrobot. However, there is obvious room for improvement, such as eliminating torque artifacts during balance or reducing the rapid switching between torque extremes during swing-up.

We have also conjectured an optimal swing-up trajectory given the simulation parameters used in this study. The trajectory presented can be used to devise a detailed fitness function and perhaps network architecture to bias learning towards the trajectory.

Acknowledgments Funding for this research has been supplied in part by the University of Newcastle Research Scholarship (UNRS) and by The ARC Centre for Complex Dynamic Systems and Control (CDSC). We would also like to thank Maria Seron for helpful discussions.

References

1. Anderson, Moore (1971) Linear optimal control. Prentice Hall, Englewood Cliffs
2. Beyer H-S (2001) The theory of evolution strategies. Springer, Heidelberg
3. Boone G (1997) Minimum-time control of the acrobot. In: Proceedings of IEEE international conference on robotics and automation, vol 4, pp 3281–3287
4. Coulom R (2004) High-accuracy value-function approximation with neural networks applied to the acrobot. European Symposium on Artificial Neural Networks
5. Federici D (2005) Evolving developing spiking neural networks. In: The IEEE congress on evolutionary computation, vol 1, pp 543–550
6. Dario Floreano, Yann Epars, Jean-Christophe Zufferey, Claudio Mattiussi (2006) Evolution of spiking neural circuits in autonomous mobile robots: Research articles. Int J Intell Syst 21(9):1005–1024
7. French RLB, Damper RI (2002) Evolution of a circuit of spiking neurons for phototaxis in a Braitenberg vehicle. In: ICSAB: proceedings of the seventh international conference on simulation of adaptive behavior on From animals to animats. MIT Press, Cambridge, pp 335–344
8. Gerstner W (2001) Pulsed neural networks, Chap. 1: Spiking neurons, pp 3–53. In: Maass and Bishop [15]
9. Gerstner W, Werner M.K (2002) Spiking neuron models: single neurons, populations, plasticity. Formal spiking neuron models, Chap. 4. Cambridge University Press, Cambridge
10. Joshi P, Maass W (2005) Movement generation with circuits of spiking neurons. Neural Comput 17(8):1715–1738
11. Kandel ER, Schwartz JH, Jessell TM (2000) Principles of neural science, 4th edn. McGraw-Hill, New York
12. Kawada K, Fujisawa S, Obika M, Yamamoto T (2005) Creating swing-up patterns of an acrobot using evolutionary computation. Proceedings of IEEE international symposium on computational intelligence in robotics and automation, CIRA 2005, pp 261–266
13. Lai X, She JH, Ohyama Y, Cai Z (1999) Fuzzy control strategy for acrobots combining model-free andmodel-based control. IEE Proc Control Theory Appl 146(6):505–510
14. Maass W (1997) Networks of spiking neurons: the third generation of neural network models. Neural Netw 10(9):1659–1671
15. Maass W, Bishop CM (eds) (2001) Pulsed neural networks. MIT Press, Cambridge
16. Markram H (2006) The blue brain project. Nat Rev Neurosci 7(2):153–160
17. Nam TK, Fukuhara Y, Mita T, Yamakita M (2002) Swing-up control and avoiding singular problem of an acrobot system. In: Proceedings of the 41st SICE annual conference, SICE 2002
18. Press WH, Teukolsky ST, Vetterling WT, Flannery BP (2002) Numerical recipes in C: the art of scientific computing, Chap. 16.1, 2nd edn. Cambridge University Press, Cambridge, pp 710–714
19. Spong MW (1995) The swing up control problem for the acrobot. IEEE Control Syst Magaz 15(1):49–55
20. Xu X, He H (2002) Residual-gradient-based neural reinforcement learning for the optimal control of an acrobot. In: Proceedings of the 2002 IEEE international symposium on intelligent control, pp 758–763
21. Yoshimoto J, Nishimura M, Tokita Y, Ishii S (2005) Acrobot control by learning the switching of multiple controllers. Artif Life Robot 9(2):67–71