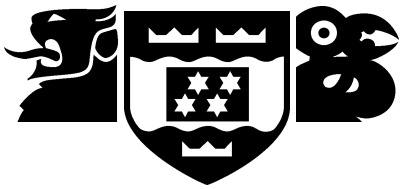
VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui



Computer Science

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@mcs.vuw.ac.nz

Spiking Neurons Models for Control and Evolutionary Robotics

Russell Tod

Supervisor: Dr Marcus Frean

Submitted in partial fulfilment of the requirements for
Bachelor of Science with Honours in Computer Science.

Abstract

Although there is much current interest in biologically plausible spiking neurons, there has been very little work investigating the role the temporal properties of these neurons might play in solving low-level physical control problems of the kind that living systems (and robots) face.

Four computationally inexpensive models of spiking neurons were investigated. Simple controllers with a feed forward architecture of spiking neurons were arrived at through an evolutionary process, and tested on tasks involving the control of unstable physical systems. Two models were readily able to solve a challenging version of the benchmark pole-balancing problem, in which two poles are balanced simultaneously, with agents have no direct access to the cart or pole velocities.

Spike frequency adaptation, a distinctive feature of biological neurons, was found to be a crucial neuro-computational property. Spiking neuron models without this property were unable to solve this task. Neuron models that exhibit adaptation have negative feedback to membrane potential, which dampens pole oscillations and leads to stable control. Moreover, in successful agents the networks of spiking neurons were simpler than those arrived at by evolving conventional recurrent neural networks.

These investigations were then extended to simulated mobile robot tasks involving obstacle avoidance, and pursuit and evasion.

Acknowledgments

I would like to thank my honours project supervisor Marcus Frean for all his help, guidance, and advice throughout the course of the project.

Contents

Figures

Chapter 1

Introduction

Real neurons 'fire', that is they communicate via a short electrical pulse known as an action potential or spike. The vast majority of neural network models used in artificial intelligence transmit information to one another via scalar outputs instead. Applying the same rationale to biological neurons amounts to assuming that only their firing rate contains information. However, this is known not to be the case. For example, visually processing in humans proceeds through 10 stages in less than 100ms. This allows on average only 10ms per stage, which is insufficient for neurons to accurately determine firing rates [37]. There is much variety in the types of neurons in the brain. A common property of many neurons is spike frequency adaptation, where a neuron will fire less frequently after it has recently fired. There are many other neuro-computational properties of biological neurons, such as the 20 recently identified by Izhikevich and reviewed in [20]. Some of these features are: tonic and phasic firing, threshold variability, spike latency, spike frequency adaptation, integrator neurons, resonator neurons, and bursting. It is an interesting question as to whether any of these properties are computationally important or merely artifacts of the physiology of neurons.

In the field of artificial neural networks there are two classes suitable for autonomous robot control, continuous time recurrent neural networks [1] and spiking neural networks [8]. There is growing interest in the use of spiking neurons which model individual neuron firings (spikes) and assume important information is carried in spike timing rather than assuming that only firing rate of the neuron conveys information. Spiking neurons are therefore more biologically realistic than the types of neuron typically used in artificial neural networks. Besides being more biologically realistic than conventional neurons there are two properties of spiking neural networks that may be advantageous for the control of autonomous robots. The first is that the temporal dynamics of neuron activation could detect and exploit time-dependent patterns in sensory-motor environmental interaction with simpler circuits than is possible with recurrent neural networks [8]. And the second is the possibility of very small and efficient hardware implementations of spiking neurons [25].

Many different models of spiking neurons have been proposed. The experiments described here explore and compare several of these spiking neuron models, from the commonly used leaky IF neurons to a recently proposed model by Izhikevich. These models are compared on several well studied problems: obstacle avoidance, pole-balancing, and pursuit-evasion. The experiments are performed in simulation, and solutions would ideally be transfered to real robots [26]. This important step was not performed here however. Controllers can be transfered if the simulation is carefully constructed with appropriate noise [28], although as no simulation can be perfect this is not guaranteed to succeed [21].

Cognitive Science research emphasizes the importance of the close interaction of brain, body and environment in the emergence of intelligence [40]. This means that intelligence

arises from an agent's interaction with their environment. Situatedness and embodiment are the two cornerstones of the bottom-up approach to artificial intelligence. The term situatedness refers to agents being situated in the world. And the related term embodiment means that agents experience the world directly, that is their actions are part of a dynamic with the world with immediate feedback to their own sensations [3]. Bottom-up approaches are based on simple interacting units that use learning or evolution to automatically construct intelligence from the bottom up, for example artificial neural networks. This is in contrast to top-down approaches, where human designers decompose a problem to build rules based on expert knowledge. While situatedness and embodiment typically refer to the physical world the definitions can be loosened slightly to include simulations without losing their meaning [11]. The agent-environment interaction where the agents behaviour effects the stimuli it receives makes design by traditional engineering difficult, beyond the simplest behaviours. Embodied autonomous agents that learn or evolve in interaction with their environments solve the symbol (or representation) grounding problem of traditional top-down artificial intelligence approaches [17]. In these approaches there is nothing to give meaning to the represention they use [2]. Truly intelligent agents can emerge only if embodied and situated in an environment [11].

In recent years, many reseachers have turned to evolution to develop autonomous robot controllers, co-called evolutionary robotics. Artificial evolution is a population based methodology that uses a selection process inspired by Darwinian evolution. Evolutionary robotics uses evolution to automatically design autonomous robot controllers which can out-perform the specifications that are given. With artificial evolution, the specifications are given in the form of a fitness function. Agents are tested on the task and given a fitness based on the fitness function to determine their survival. Artificial evolution is able to find simple, elegant solutions, that can be surprisingly ingenious [23].

Chapter 2

Spiking Neuron Models

2.1 Biological Neurons

Neurons are the fundamental information processing units of biological brains. They are made up of a cell body, a dendritic tree which receives incoming messages, and an axon, see figure 2.1. Neurons communicate by sending short electrical pulses down their axons to axon terminals. Axon terminals form a connection with another neuron's dendrites at a junction called a synapse. Electrical pulses arrive at the axon terminal and are converted to a chemical signal, that is neurotransmitter is released into the synapse. The neurotransmitter diffuses across the synapse to receptors on the dendritic tree of the neuron.

Neurons like all animal cells consist of a lipid membrane containing an aqeuous solution of proteins and chemical ions. An electrical potential exists across the neuron's cell membrane due to differing ion concentrations inside and outside the cell. The most important ions are Na^+ and K^+, but Cl^- and Ca^+ are also involved. Embedded in the cell membrane are proteins that act as ion channels, selectively allowing certain ions to pass through the cell membrane. Ion channels are gated, meaning they may be open under some conditions and closed in others. As well as ion channels there are ion pumps which use energy to move ions against their concentration gradients. The major ion pump in neurons is the sodium-potassium pump, which pumps Na^+ out of the neuron and K^+ in to the neuron.

A neuron is polarised at its resting potential when the neuron has received no input for some time. A neuron's membrane resting potential is typically about -65mV. This polarisation is due to the sodium-potassium pump which pumps out more Na^+ than it pumps in K^+. This results in more Na^+ outside the cell than K^+ inside, so there are fewer positive charges inside the neuron giving a negative voltage across the neuron's membrane. At resting potential Na^+ channels are closed and K^+ channels are open and thus K^+ ions can diffuse out of the neuron whereas Na^+ ions cannot diffuse in.

Ion channels open and close depending on membrane voltage (via voltage-gated channels) and on the presence or absence of chemical messenger molecules in the case of neurotransmitter-gated channels. An action potential from a pre-synaptic neuron releases neurotransmitter into the synapse which binds with receptors of the post-synaptic neuron. This opens (or closes) neurotransmitter-gated ion channels resulting in depolarization or hyperpolarization of the neuron membrane. When sufficiently depolarized (about -50mV) voltage-gated ion channels open, depolarising the neuron further and resulting in an action potential; that is, the neuron "fires". The action potential ($+35$mV) then travels down the axon stimulating other neurons in the same way.

During an action potential Na^+ ions rush into the neuron, which causes K^+ to rush out, see figure 2.2. After firing the neuron takes some time to restore ions to their resting concentrations.

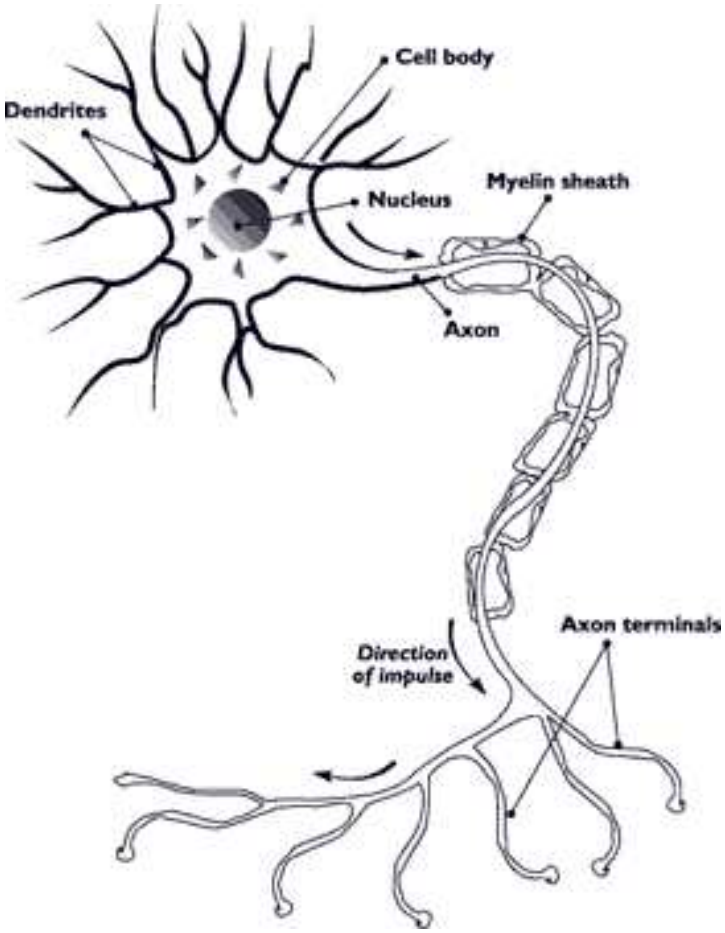


Figure 2.1: A Biological Neuron [35]. This figure shows the important physical components of a biological neuron: the dentrites, cell body, axon, and axon terminals. Information from other neurons is received at the dendrites. When membrane potential of the cell body exceeds the threshold the neuron 'fires' and sends an electrical pulse down its axon to the axon terminals. The electrical pulse at axon terminals releases neurotransmitter, which is received by the dendrites of other neurons.

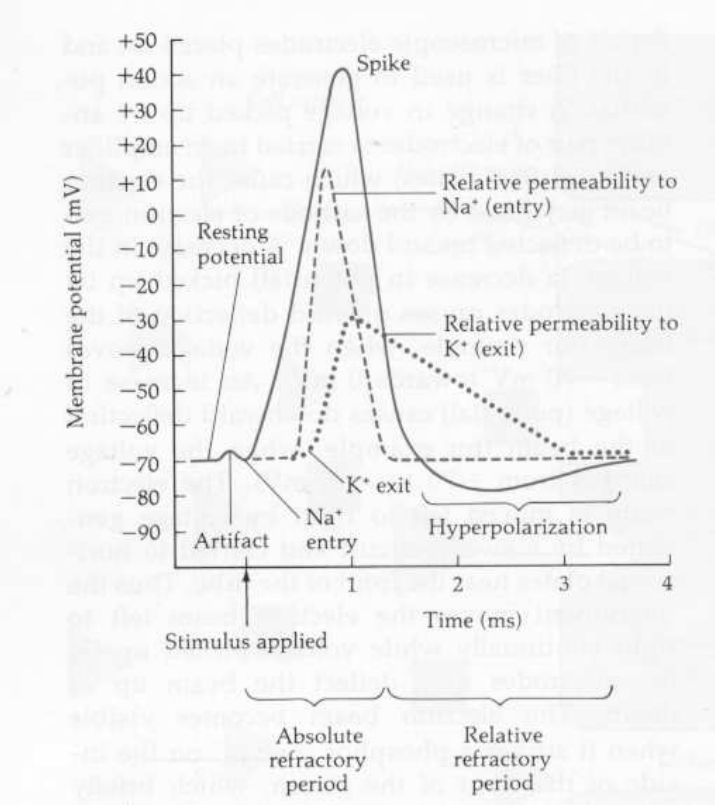


Figure 2.2: An Action Potential [33]. This figure shows the membrane potential rising as sodium ions flow into the neuron and the subsequent falling of the membrane potential as potassium ions flow out of the neuron. Following a spike is a refractory period in which the neuron restores the ion concentrations to their resting levels.

2.2 Detailed Neuron Models

One of the most important neuron models in computational neuroscience is the Hodgkin-Huxley model [18]. Based on their 1952 experiments on the giant axon of the squid, Hodgkin and Huxley developed a mathematical model describing neuron membrane potential and the flow of ions through channels in the membrane. This section reviews that model breifly. Although too costly to be useful in control, it is the origin of the other models to be considered.

In the Hodgkin-Huxley model the cell membrane is considered as a capacitor. An input current $I_{in}(t)$ applied to the membrane may be split into a current which charges the capacitor $I_{cap}(t)$ and a current that passes through the ion channels $I_{chan}(t)$.

$$I_{in}(t) = I_{cap}(t) + I_{chan}(t)$$

Capacitance C is defined by a charge Q and voltage u across the capacitor, $C = Q/u$, and so the capacitive current $I_{cap} = Cdu/dt$. Thus, the membrane potential u is described by the equation,

$$C\frac{du}{dt} = -I_{chan}(t) + I_{in}(t)$$

In the model there are three types of channel; a sodium channel, a potassium channel and a leakage channel. Variables m, n, and h are introduced to describe the gating of the ion channels, where m and h control the opening and closing of the sodium channel and n controls the potassium channel. The equation describing the current of the channels is,

$$I_{chan} = g_{Na}m^3h(u - E_{Na}) + g_K n^4(u - E_K) + g_L(u - E_L)$$

where E_{Na} is 115, E_K is -12 and E_L is 10.6 are the reversal potentials and g_{Na} is 120, g_K is 36 and g_L is 0.3 are the channel conductances. These are the original empirical parameters reported by Hodgkin and Huxley. They assume the resting potential is zero and need to be shifted by -65mV to be biologically accurate. The form of equations for each of the gating variables m, n, and h is,

$$\frac{dx}{dt} = \alpha_x(u)(1 - x) - \beta_x(u)x$$

where x is substituted by each of m, n, and h and the equations for α_x and β_x are,

x	α_x	β_x
n	$(0.1 - 0.01u)/[\exp(1 - 0.1u) - 1]$	$0.125\exp(-u/80)$
m	$(2.5 - 0.1u)/[\exp(2.5 - 0.1u) - 1]$	$4\exp(-u/18)$
h	$0.07\exp(-u/20)$	$1/[\exp(3 - 0.1u) + 1]$

The Hodgkin-Huxley model is biophysically meaningful but very expensive to implement, each neuron requires 1200 floating point computations per ms.

The four-dimensional Hodgkin-Huxley model can be reduced to two dimensions by two approximations [12]. Firstly, variable m is much faster than n and h and can be replaced by a steady state value: this is called the quasi steady-state approximation. And secondly, variables n and $(1 - h)$ are similar and can be replaced by a single effective variable, w.

Several simplified but biologically plausible two-dimensional models exist such as the Morris-Lecar model, the FitzHugh-Nagumo model, and the Izhikevich model. The Morris-Lecar model requires a time step significantly smaller than 1ms and involves hyperbolic tangents and exponents, making it computationally costly. The FitzHugh-Nagumo and the

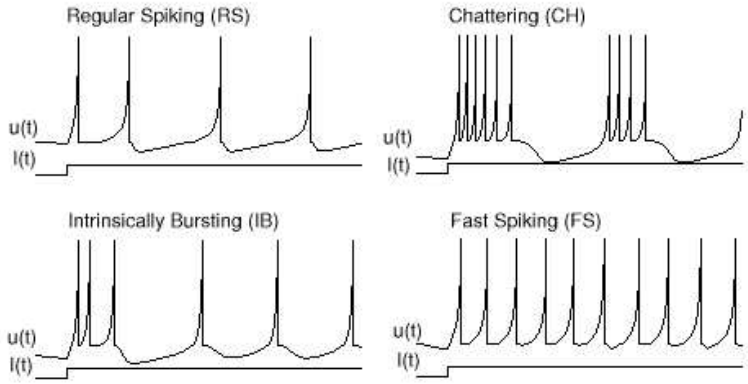


Figure 2.3: Izhikevich neurons with different neuron model parameters were obtained by implementing equations 2.1 and 2.2. Neurons are fed a constant current as input. Izhikevich neurons with four different parameter settings are shown. The different spiking behaviours shown here are also also observed in biological neurons. The Regular spiking (RS) neuron has model parameters a=0.02, b=0.2, c=-65, d=8. The Intrinsically Bursting (IB) neuron has model parameters a=0.02, b=0.2, c=-55, d=4. The Chattering (CH) neuron has model parameters a=0.02, b=0.2, c=-50, d=2. The Fast Spiking (FS) neuron has model parameters a=0.1, b=0.2, c=-65, d=2.

Izhikevich models are similar, the difference being that the FitzHugh-Nagumo model simulates the shape of a spike and so requires a smaller time step and therefore more computations per ms than the Izhikevich model. Thus Izhikevich model offers a good trade-off between biological realism and computational efficency.

2.3 The Izhikevich Neuron Model

A model capable of producing the rich firing patterns of biological neurons was recently proposed by Izhikevich [19]. The model is a reduction of the Hodgkin-Huxley model to a two variable system. Two ordinary differential equations describe the membrane potential u and a "recovery variable" w,

$$\frac{du}{dt} = 0.04u^2 + 5u + 140 - w + I \tag{2.1}$$

$$\frac{dw}{dt} = a(bu - w) \tag{2.2}$$

with an after-spike reset rule: if $u \geq 30$ then $u \leftarrow c$, $w \leftarrow w + d$ [19]. The inactivation of sodium and activation of potassium ion channels are accounted for in the recovery variable w, which also provides negative feedback to u. When a spike reaches its apex of 30mV the membrane potential is reset to c and the recovery variable is incremented by d. Input to the neuron from synaptic currents is via variable I. The typical size of a time step used with this model is 1ms.

This model is capable is producing firing patterns observed for many types of biological neurons such as regular spiking neurons, intrinsically bursting neurons, chattering neurons and fast spiking neurons. The different patterns can be produced by tuning the four parameters of the model as shown in figure 2.3.

The Izhikevich model it is able to produce 20 identifiable neuro-computational features of biological neurons [20]. Some of these features are: tonic and phasic firing, threshold variability, spike latency, spike frequency adaptation, integrator, resonator, and bursting. Each

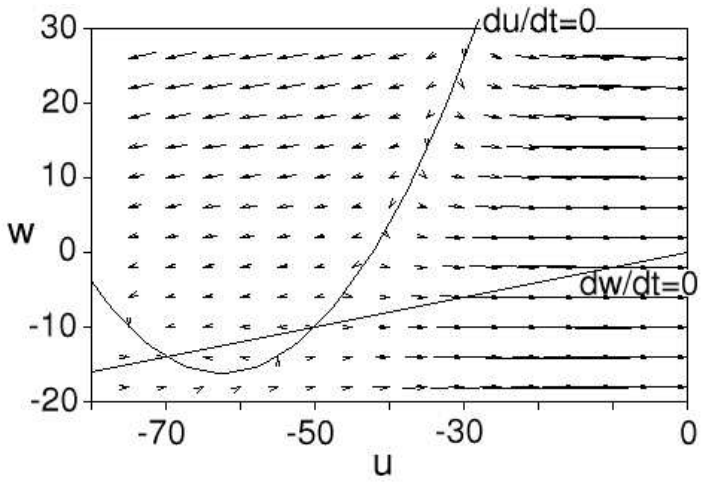


Figure 2.4: Phase portrait of the Izhikevich neuron model. Membrane potential is plotted against the recovery variable. The u-nullcline (curved line) is where $du/dt = 0$ and similarly the w-nullcline (straight line) is where $dw/dt = 0$. Arrows indicate the gradient and were produced by implementing equations 2.1 and 2.2.

of the 20 neuro-computational features can be exhibited by Izhikevich neurons with different parameters, by tuning the 4 parameters of the model. Not all features can be exhibited at once as some of the properties are mutually exclusive.

The model does not have a fixed firing threshold. Instead this depends on previous firing and can be anywhere between -55mV and -40mV, as with biological neurons. When presented with prolonged stimulus a neuron fires first with short inter-spike period after which the period increases, see figure 2.3. This is refered to as spike frequency adaptation, and it is a common property of biological neurons.

The Izhikevich model is as biologically plausible as the Hodgkin-Huxley model, in that both are able to produce the 20 neuro-computational features of biological neurons. However the Izhikevich model is far more computationally efficient, requiring only 13 floating point operations per ms, which is orders of magnitude less than the Hodgkin-Huxley model.

2.3.1 Phase plane analysis

To better understand the dynamics of the Izhikevich model we can visualise the phase portrait of the u and w variables over time. The direction of the flow $(\dot{u}, \dot{w})^T$ where $\dot{u} = du/dt$ and $\dot{w} = dw/dt$ can be plotted as a vector field in the phase plane, see figure 2.4. The set of points where $\dot{u} = 0$ is called the u-nullcline, the direction of flow in the phase portrait are vertical. Similarly, the w-nullcline is defined by $\dot{w} = 0$ and arrows are horizontal. Fixed points, defined by $\dot{u} = \dot{w} = 0$ are given by the intersection of u-nullcline with the w-nullcline.

In the Izhikevich model there are two fixed points. The fixed point at around u is -70 is stable and is the resting potential in the absence of any input current. The other fixed point is unstable and from this point the variables u and w will either converge on the resting potential fixed point or the potential will increase until reset when u reaches 30, that is the neuron fires. The right hand section of the u-nullcline represents the neuron's firing threshold, which increases as w increases.

2.4 Leaky-integrate-and-fire Neuron Model

A widely used model in computational neuroscience is the leaky integrate-and-fire (leaky IF) neuron model, which has just a single variable, the membrane potential u. A single differential equation describes the membrane potential,

$$\tau_m \frac{du}{dt} = u_{rest} - u(t) + RI(t) \tag{2.3}$$

Firing occurs if $u \geq u_{thres}$ in which case $u \leftarrow u_{reset}$, where u is the membrane potential [12]. The typical values of u_{rest} is -65, u_{reset} is -70, and u_{thres} is -50. The typical size of a time step used with this model is 1ms.

In the absence of input, membrane potential tends to the resting potential due to the leak current. Input current I increases the membrane potential. When membrane potential reaches the fixed threshold value the neuron fires, emits a spike.

For constant input the output spike rate can be calculated by integrating equation 2.3 and solving for the inter-spike interval. The reciprocal of the inter-spike interval gives the firing rate υ for a given constant input current,

$$\upsilon = \left[\Delta^{abs} + \tau_m \ln \frac{RI_0}{RI_0 - u_{thres}} \right]^{-1}$$

with the potentials shifted such that u_{rest} and u_{reset} are 0, and where Δ^{abs} is the absolute refractory period [12].

The leaky IF neuron model only takes into account the leak current and does not model the sodium or potassium currents, hence it is not as biologically realistic as the Izhikevich model or as the detailed models. This model is only able to produce 3 of the 20 neuro-computation features of biological neurons [20]. It is not able to produce phasic spiking, bursting, rebound spikes, threshold variability, or spike frequency adaptation. However, the leaky IF neuron model is more computationally efficient than the Izhikevich model requiring only 5 floating point operations per ms.

2.5 Leaky IF with Adaptation Neuron Model

The leaky IF with adaptation model extends the leaky IF model by including an adaptation variable g that models the outward (hyperpolarising) current responsible for making the neuron less likely to fire if it has fired recently. This spike frequency adaptation is an important feature exhibited by biological neurons. The differential equations governing this model are,

$$\tau_m \frac{du}{dt} = u_{rest} - u(t) + RI(t) - g(u(t) - E_k)$$

$$\tau_a \frac{dg}{dt} = -g$$

with the following after-spike reset rule: if $u \geq u_{thres}$ then $u \leftarrow u_{reset}$, $g \leftarrow g + d$. When the potential u reaches the threshold u_{thres} a spike is generated. The typical value for u_{thres} is -50. After a spike the potential is reset to u_{reset} (typically -70) and the adaptation variable g is incremented by d (typically 0.1). In the absence of input current the membrane potential settles to the resting potential u_{rest} (typically -65). The constant E_k has a value of -85.

This model is an improvement in biological realism over the basic leaky IF model in that it exhibits 5 of the neuro-computational features of biological neurons. The leaky IF model with adaptation now has spike frequency adaptation but is still not able to produce many of

the other important features of biological neurons. Adding adaptation to the leaky IF model comes at a cost of making the model more computationally expensive, requiring 10 floating point operations per ms, about the same as the Izhikevich model.

2.6 Quadratic IF Neuron Model

Another variation on the leaky IF model is the quadratic IF neuron model, also known as the theta-neuron or in trigonometric form as the Ermentrout-Kopell canonical model. The model has a single variable, the membrane potential u. It is nonlinear, and given by the differential equation,

$$\tau_m \frac{du}{dt} = a(u(t) - u_{rest})(u(t) - u_{thres}) + RI(t)$$

with the after-spike reset rule: if $u \geq u_{peak}$ then $u \leftarrow u_{reset}$. The typical value of a is 0.05. Membrane potentials u_{rest} and u_{thres} are the resting and threshold potentials respectively with typical values u_{rest} is -65 and u_{thres} is -50. The membrane potential is reset to u_{reset} (typically -70) when it reaches its apex u_{peak} of 30. The membrane potential decays to resting potential in the absence of input current $I = 0$ and $u < u_{thres}$. And when $u > u_{thres}$ the membrane potential increases and an action potential is triggered.

The quadratic IF neuron model exhibits more of the neuro-computational features of biological neurons than the leaky IF model, producing 6 out of the 20 properties, including activity dependent threshold, spike latency, and bistability. It does not show spike frequency adaptation but requires only 7 floating point operations per ms.

2.7 Rate-Based Neuron models

The most commonly used neurons in connectionist neural networks are sigmoid neurons. The output of a sigmoid neuron is given by the equation,

$$\upsilon = \frac{1}{1 + \exp(-\phi)}$$

where ϕ is the input to the neuron. The output υ of a sigmoid neuron can be interpreted as the current firing rate of a neuron [24]. So, an output of $\upsilon = 0.5$ is interpreted as a neuron emitting half its maximum rate of spikes, for example 100Hz if the neuron's maximum is 200Hz. Biological neurons do increase their firing rate as input to them increases, however this increase does not follow the sigmoid function. More importantly, the sigmoid model has no memory of previous input, and therefore cannot exhibit spike frequency adaptation, threshold variability, bursting, or almost any of the other properties of biological neurons.

2.8 Summary

Several spiking neuron models were introduced in this chapter. There are many others not reviewed here, for a survey see [20]. Four neuron models suitable for computational modeling were described here. They can be classified in two ways: whether they are linear or non-linear, and whether they are one dimensional or two dimensional. The Leaky IF neuron model is linear and one dimensional. The Leaky IF with adaptation neuron model is linear and two dimensional. The Quadratic IF neuron model is non-linear and one dimensional. And finally Izhikevich's neuron model is non-linear and two dimensional.

These four neuron models also differ in their biological realism and in the number of neuro-computational properties they exhibit. Izhikevich's model is the most biologically realistic model and is able to exhibit many of the properties of biological neurons, 20 out of 20 of the properties classified in [20]. Whereas the Leaky IF, the Leaky IF with adaptation, and the Quadratic IF neuron models exhibit only 3, 5, and 6 of these properties, respectively.

An interesting question is which of these properties it is important to model and which are just a consequence of the physiology of neurons. To investigate this question the four neuron models were implemented in C++ and tested against several well studied control problems; pole balancing and obstacle avoidance with the Khepera robot as well as pursuit-evasion with the Khepera robot.

Chapter 3

Evolutionary Robotics

Evolutionary robotics draws inspiration from the biological process of evolution by natural selection. To automatically design controllers where the desired behaviour is not prespecified in detail but rather emerges through the evolutionary process. Artificial evolution involves the iteration of two steps; evaluation and reproduction. The evaluation step involves determining a fitness score for a genotype decoded as an agent controller from the agent's behaviour in its environment, using a fitness function. The reproduction step involves probabilistically selecting fit genotypes as parents and creating offspring from them using the genetic operators of mutation or crossover.

3.1 Robot Controllers

Any type of robot controller architecture can be evolved provided it can be replicated with mutation. Artificial neural networks are a natural choice and have been used by many researches [29] [9]. It has even been argued that artificial neural networks are the most appropriate general control architecture for evolutionary robotics [21]. Other types of robot controller have been evolved such as genetic programming of Lisp programs [22], and classifier systems where a genetic algorithm is applied to rules for controlling robots [7]. These approaches seem less appropriate for the evolution of low-level behaviours than neural networks [21]. Neural networks are well suited to artificial evolution due to their graceful degradation (high degree of neutrality) [4]. Evolution is often thought of as search through genotype space, and the relative smoothness of the weight space of neural networks improves the evolutionary search.

Encodings of robot controllers are treated as genotypes in the sense that it is the encoding that is subject to mutation and "evolved". The direct encoding of synaptic weights on the genotype for a fixed neural network architecture [9] is the simplest encoding scheme. Each synaptic weight is coded as floating point number on the chromosome, so for a fixed neural architecture every chromosome in the population has the same length. Direct encoding schemes are not biologically plausible as there are insufficient genes to encode every connection in the brain. Several researchers have developed indirect genotype to phenotype mappings [32] [6] [15] [36] however it is still very much an open area of research.

Vision-based robot controllers composed of spiking neurons have been evolved [8], with recurrent connections and a direct genetic encoding which show promising results.

Apart from evolving fixed weights, learning can be incorporated into the neural network to fine-tune weights. Learning has effects on the evaluation of controllers as the fitness landscape is no longer fixed and static.

3.2 Evolution

In artificial evolution there are two strategies for reproducing a population; a generational algorithm where the entire population is replaced every generation [13], and a steady-state algorithm where agents are replaced by newly created agents one at a time [39]. The generational approach is more commonly used, however it has been claimed that the steady-state approach may be twice as fast or conversely require only half the computations to reach a satisfactory solution due to the higher selection pressure [34] [38].

From an initially diverse population of randomly created agents the evolutionary process causes the population to move towards (converge on) more viable solutions. For evolution to progress selection pressure is applied to the population, by a selection policy and a replacement policy. The selection policy is typically probabilistic and proportionate: more fit agents are more likely to be selected as parents. The replacement policy applies selection pressure by removing less-fit agents, for example by replacing the worst agent in the population.

As artificial evolution is applied to increasingly difficult problems the difficulty is in finding a suitable fitness function. If all initial agents score zero fitness, selection cannot operate, a situation refered to as the bootstrap problem [31]. Evolution by natural selection has no explicit fitness function and one solution to the bootstrap problem is the co-evolution of two populations where agents are scored against one another. Incremental evolution can also be used, to first solve a simple task then to progressively make the task more complex [31]. The downside of this is the requirement for supervision by a designer who may inadvertently introduce unnecessary contraints.

3.2.1 Reproduction

Reproduction is the process of creating a new agent from another using the so-called genetic operators of mutation and crossover. For controllers to be evolvable incremental improvements must be possible with a mutation to the genotype. For evolution to be better than random search, reproduction must be more likely to create a fit individual by mutation of a fit individual than by creating a new individual at random.

In the case of artificial neural networks with connection weights directly encoded as the genotype, reproduction can create a new controller by sampling from a distribution around an existing solution in weight space. As the weight space for artificial neural networks is relatively smooth, weights taken from near a fit solution are also likely to be fit, allowing reproduction to make the incremental changes necessary for selection to work.

3.2.2 Evaluation

The fitness score given to an agent must be an accurate measure of the agent's ability on the task. With autonomous robots an episode involves noisy environmental interaction, so the average of several episodes should be taken to get an more accurate fitness. An episode here refers to a trial on the task for which the robot controller is being evolved to perform.

Typically autonomous robots are evaluated in a simulated environment as many more evaluations can be acheived and then the best controller can be transfered to a real robot and possibly evolved further [26]. In simulation, the fitness function may have access to information from the environment, such as the agent's absolute position, that is difficult to obtain in the real world. For this reason the fitness function should use only information locally available to the robot [31].

Evaluation of agents on simple tasks is relatively straight forward where an explicit fitness function can be given, such as obstacle avoidance or phototaxis. For evolving more

complex behaviours, beyond reactive intelligence, developing a fitness function that captures the desired behaviour and not undesireable behaviours becomes increasingly difficult. The behaviour of the evolved agent is not fully known in advance and so determining the variables and constraints that rate the performance against the expected behaviour is difficult [31].

The fitness function is crucially important for the evolvability of the system. Importantly the fitness function must be gradual, allowing partially successful behaviours to be selected for. If all individuals in the initial population receive zero fitness there is little chance of evolution succeeding. One way of overcoming some of these difficulties is to use an implicit rather than an explicit fitness function, for example by using co-evolution.

3.3 Co-evolution

Competitive co-evolution is attracting growing interest in field of evolutionary computation. The simplest scenario involves two populations, where individuals from one population compete against individuals from the other. The outcome of each competition determines an individual's fitness. The success of one agent is the failure of another. The fitness function does not need to explicitly specify any particular behaviour. Fitness improvements in one population affect the other population, which means that the fitness landscape is constantly changing. As one population adapts and improves it changes the fitness landscape of the other population's agents such that they need to adapt new strategies. This leads to what is termed an evolutionary "arms race". In a co-evolutionary scenario, evolution is open ended and fitness is non-stationary.

A novel method was introduced in these experiments (in section 5.4) to compare different controllers, that of multi-species co-evolution. Typically co-evolution involves two populations in competition with each other, often a predator population and prey population, where the adaptive success of one species implies failure of the other. To determine competitive advantage of multiple types of controller this scenario is extended such that each population contains the multiple types of controller (species) in competition where one type of controller can displace another in the population through the selection process. The type of the controller that comes to dominate a population is assumed to be superior.

Chapter 4

Pole Balancing

The pole-balancing problem has been used as a benchmark task in reinforcement learning for over 30 years [36]. It is useful because it is a very simple example of an inhererently unstable dynamical system, and also because the pole-balancing problem is available in varying degrees of difficulty. The control problem can be made more difficult in several ways: such as by changing the mechanical system, adding a second pole, or by restricting state information given to controller.

In this chapter pole balancing is used to compare the four neuron models described in chapter 2. Two versions of the problem are used, the first is the single pole with no velocities problem, and the second is the most difficult version of the problem that of double pole balancing without velocities.

4.1 The Pole Balancing Problems

The simplest pole balancing task involves a single pole connected by a hinge to a moving cart, where a force must be applied to the cart to keep the pole vertical (within $\pm 12°$) and the cart within the boundaries of the track (± 2.4m). In a more difficult version of the task, double pole balancing, two poles are connected by hinges to the cart, where the two poles are of different length and move independently of each other. As before a force must be applied to the cart to keep both poles vertical and the cart within the boundaries of the track.

The state of the system is defined by cart position x, cart velocity v, pole angle ϑ_i, and pole angular velocity ω_i, where i indexes the pole. The state of the system is governed by ODEs updated 50 times a second.

The pole balancing tasks can be made more interesting by restricting the state information given to the controller to just x and ϑ_i, without the velocities. This makes the problem non-Markovian. That is, controllers have only partial information concerning the state of the system and need memory of previous state to infer velocity.

The most difficult pole balancing problem, the double pole with no velocities problem, has only been solved by a few other systems namely, Cellular Encoding CE [16], Enforced Subpopulations ESP [14], and NeuroEvolution of Augmenting Topologies NEAT [36]. These systems all involve evolving neural networks of sigmoidal neurons.

4.2 A Neurocontroller for Pole Balancing

The neurocontroller architecture used for the single pole balancing task is a feed forward network of four spiking neurons, and for the double pole balancing task a feed forward net-

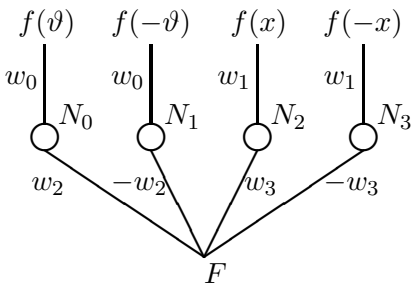


Figure 4.1: Neurocontroller for single pole balancing. The controller is composed of 4 neurons, N_i. Inputs to neurons are state variables, ϑ and x, put though sigmoid function f, and multiplied by weights w_0 and w_1 respectively. Due to the symmetry of the problem controller also has inputs, $f(-\vartheta)$ and $f(-x)$, which are also weighted by w_0 and w_1 respectively. The output force F is the sum of the neuron outputs weighted by w_2, $-w_2$, w_3, and $-w_3$.

work of six spiking neurons is used. Many controller architectures are possible, several of which were experimented with. In each case the chosen controller was the simplest architecture able to solve the problem.

The state of the cart-pole system is updated every 20ms (50 times per second), whereas the spiking neurons use a time step of 1ms. So, each step of the sensori-motor loop consists of taking the cart position and pole angle state variable values, iterating the neurons for 20 cycles, then calculating the network output and applying the force to the cart.

For the single pole balancing task the state variables provided as input to the controller are the pole angle ϑ, and the cart position x, similarly for the double pole balancing task the angles of both poles ϑ_1 and ϑ_2, and the cart position x, are used as input. Velocity state variables v and ω_i are not available to the controller.

The single pole balancing neurocontroller has 4 parameters, $\{w_0, w_1, w_2, w_3\}$, these are the weights of connections in the neural network. Similarly, there are 6 neural network weight parameters, $\{w_0, w_1, w_2, w_3, w_4, w_5\}$, for the double pole balancing neurocontroller.

Spiking neurons receive input in the form of an input current. The state variables are put through a squashing function to get values between 0 and 1. When the input current to a spiking neuron is 0 it will not fire and so the additive inverse of the state variable is also provide to the controller. Input to the 4 neurons of the single pole balancing controller is $f(\vartheta)w_0$, $f(-\vartheta)w_0$, $f(x)w_1$, and $f(-x)w_1$, respectively, where f is the sigmoid function $f(x) = 1/(1+e^{-Kx})$, with scaling constant K. Similarly, input to the 6 neurons of the double pole balancing controller is $f(\vartheta_1)w_0$, $f(-\vartheta_1)w_0$, $f(\vartheta_2)w_1$, $f(-\vartheta_2)w_1$, $f(x)w_2$, and $f(-x)w_2$, respectively.

Spiking neurons output a spike train which needs to be converted to an output value. This is done by counting the number of spikes in a time window of 20ms and putting the count through a squashing function to get a values between 0 and 1. A squashing function was chosen such that more spikes have increasingly less effect on the output value. For example, an increase from no spikes to one spike is considered more significant than an increase from 19 spikes to 20 spikes. Neuron output r_i, is computed using the function $r_i = e^{-1/n_i}$ where n_i is the number of spikes emitted by the i^{th} neuron over the past 20ms. The output of all neurons is summed to get an output force F to apply to the cart. For the single pole neurocontroller the output force is $F = r_0w_2 - r_1w_2 + r_2w_3 - r_3w_3$. And for double pole neurocontroller the output force is $F = r_0w_3 - r_1w_3 + r_2w_4 - r_3w_4 + r_4w_5 - r_5w_5$.

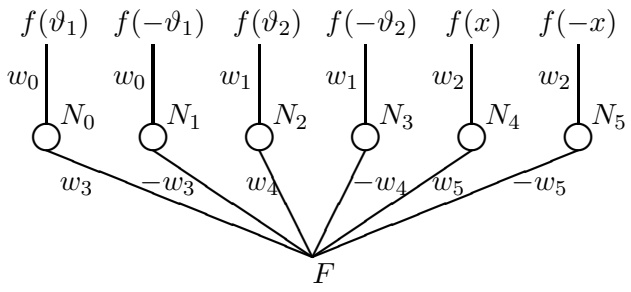


Figure 4.2: Neurocontroller double pole balancing. The controller is composed of 6 neurons, N_i. Inputs to neurons are state variables, ϑ_1 ϑ_2 and x, put though sigmoid function f, and multiplied by weights w_0, w_1 and w_2 respectively. Due to the symmetry of the problem controller also has inputs, $f(-\vartheta_1)$, $f(-\vartheta_2)$ and $f(-x)$, which are also weighted by w_0, w_1 and w_2 respectively. The output force F is the sum of the neuron outputs weighted by w_3, $-w_3$, w_4, $-w_4$, w_5, and $-w_5$.

4.2.1 Experiments

Experiments were performed to compare controllers composed of the four neuron models described in chapter 2 on two versions of the pole balancing problems. The single pole balancing without velocities and double pole balancing without velocities were used in these experiments.

The performance of each type of controller is determined by evolving a population of controllers on the pole balancing problem. A steady state genetic algorithm is used with a population size of 100. In this steady state genetic algorithm the controller with the lowest fitness is replaced by a newly created controller, which is evaluated and the process repeats. New controllers are created by taking a sample from the Gaussian distribution about the parameters of a parent, where the parent is chosen with a probability proportional to its fitness. The parameters changed by mutation are the weight parameters of the neural network and the parameters of the neurons (all neurons have the same parameters). So, a double pole balancing controller composed of Izhikevich neurons has 10 degrees of freedom. The controller with the highest fitness, after evolving for $10,000$ evaluations, is taken as a solution.

During evolution controllers need to be evaluated to determine their fitness. We would like a fitness function that rewards controllers for keeping the pole(s) vertical and the cart centered for as long as possible. The maximum fitness value of 1 is acheived when the pole is balanced for 1000 steps, the pole(s) is kept completely vertical ($\vartheta = 0$) and the cart is kept perfectly centered ($x = 0$) for every step. To reward controllers for these behaviours the following fitness functions ϕ_{sp} and ϕ_{dp} were designed. They are similar to those used in [16]. Each trial is initialised with pole vertical and cart centered and giving the cart a random push in either direction, thus the perfect fitness score of 1 is unachieveable. For the single pole balancing problem, the fitness of a controller is the average of three trials of 1000 steps (20 seconds of simulation time) using the fitness function,

$$\phi_{sp} = \frac{n}{1000}\left(1 - \frac{\mu_{|\vartheta|}}{K_\vartheta}\right)\left(1 - \frac{\mu_{|x|}}{K_x}\right)$$

where n is the number of timesteps the pole was balanced for, $\mu_{|\vartheta|}$ is the average absolute ϑ value over n timesteps. Similarly $\mu_{|x|}$ is the average absolute x value over n timesteps. Scaling constants $K_\vartheta = 0.2$ and $K_x = 2.4$ are used to scale averages to the range $(0, 1)$.

For double pole balancing the fitness of a controller is the average of five trials of 1000

steps (20 seconds of simulation time) using the fitness function,

$$\phi_{dp} = \frac{n}{1000}\left(1 - \frac{\mu_{|\vartheta_1|}}{K_{\vartheta_1}}\right)\left(1 - \frac{\mu_{|\vartheta_2|}}{K_{\vartheta_2}}\right)\left(1 - \frac{\mu_{|x|}}{k_x}\right)$$

where n is the number of timesteps the pole was balanced for, $\mu_{|\vartheta_1|}$, $\mu_{|\vartheta_2|}$ and μ_x are the averages over n timesteps of $|\vartheta_1|$, $|\vartheta_2|$, and $|x|$ respectively. Normalising constants are $K_{\vartheta_1} = K_{\vartheta_2} = 0.2$, and $K_x = 2.4$ to scale averages to the range $(0, 1)$.

The same single cart-pole and double cart-pole parameters are used here as were used in [36] and [16]. For the single cart-pole the parameters are: the cart mass is 1.0kg, the pole mass is 0.1kg, and the pole length is 1.0m. And for the double cart-pole the parameters are same as for the single cart-pole but with the second pole which has a mass of 0.01kg, and a length of 0.1m.

4.2.2 Results

The best controller of the population, after running evolution for $10,000$ evalutions, is taken and re-evaluated on the task 100 times. Three results are calculated from the 100 evaluations of the best controller:

- an average fitness score.
- the average number of steps the pole remains balanced.
- a count of the number of times the pole was successfully balanced for the maximum possible number of steps (1000).

The evolution is repeated 10 times and the results of the evaluation of the best controllers from the 10 runs are averaged. This is done for controllers composed of each type of neuron. The table below shows these results.

Results of Single Pole with No Velocities

Neuron Type	Fitness	stdev	Steps	stdev	Success	stdev
Izhikevich	0.9372	0.0434	1000.0	0.0	100.0	0.0
Leaky IF	0.0446	0.0095	117.3	22.3	0.0	0.0
LIF with Adaptation	0.9466	0.0044	1000.0	0.0	100.0	0.0
Quadratic IF	0.0357	0.0047	76.0	6.4	0.0	0.0

Results of Double Pole with No Velocities

Neuron Type	Fitness	stdev	Steps	stdev	Success	stdev
Izhikevich	0.8183	0.0933	985.3	30.6	94.8	11.1
Leaky IF	0.0269	0.0018	43.6	3.7	0.0	0.0
LIF with Adaptation	0.7325	0.0882	925.5	83.3	88.6	12.3
Quadratic IF	0.0276	0.0043	42.6	7.7	0.0	0.0

Two of the types of controllers perform very well on both tasks, controllers of Izhikevich neurons and controllers of Leaky IF with Adaptation neurons, with near perfect possible scores on the single pole balancing problem and respectable scores on the much more difficult double pole balancing problem. By comparison controllers composed of the Leaky IF neurons and the Quadratic IF neurons score very low and are unable perform the task.

Gruau defined a criteria for solutions to the double pole balancing problem [16]. The criterion defines 625 initial states from which the controller must balance the pole for 1000

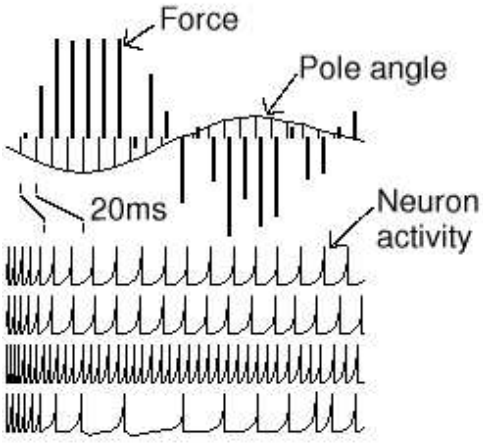


Figure 4.3: This figure is the key for interpreting subsequent pole balancing figures. The figures show the force applied to the cart (thick vertical bars) and the view from above of the pole angle (thin vertical bars) with a trace of pole angle over time (curved line). The figures also show the output spike train of the neurons of the controller. Note that different timescales are used for pole and for neuron activity. Up to the first 2.5 seconds of pole balancing is displayed but only 1 second of neural activity.

steps. The number of successes is refered to as the generalization performance of the solution. To be considered a solution, the controller needs to be successful from 200 of the initial states. The initial states are values $0.05, 0.25, 0.5, 0.75, 0.95$ scaled to the respective ranges for each the state variables, $x, v, \vartheta_1, \omega_1$, which gives gives $5^4 = 625$ initial states.

The best spiking neuron controllers from the experiments would not be considered solutions under Gruau's criterion, as they only manage an average of 52.8 successes with standard deviation of 34.6 out of a possible 625. Gruau's generalisation test is difficult, many of the initial states are very close to the extremes of the double cart-pole system. The generalisation test can be made slightly easier by excluding the more difficult initial states. The initial state values of $0.3, 0.4, 0.5, 0.6, 0.7$ scaled to the respective ranges were used for each the state variables $x, v, \vartheta_1, \omega_1$. On this slightly easier generalisation test we get an average 244.4 successes with standard deviation of 141.2 out of a possible 625.

4.2.3 Discussion

These experiments show that the two neurocontrollers that exhibit spike frequency adaptation (SFA), namely the Izhikevich neuron model and the leaky IF with adaptation neuron model, perform far better on both the single and double pole tasks than neurocontrollers without SFA such as leaky IF neurons or quadratic IF neurons.

The recovery variable w of the Izhikevich model and the adaptation variable g of the leaky IF with adaptation model provide negative feedback to the membrane potential u, and do so with a slower time characteristic than the membrane potential itself. This gives these two neuron models the SFA property. Adaptation appears to be a useful feature to perform the pole balancing tasks. With SFA a neuron may initially fire rapidly when presented with fresh input then become quickly sensitised to this input and slow its firing rate as a result.

There are two neurons which receive input given by the pole angle. One of the neuron's input will increase as the pole falls in one direction and decrease as it falls in the opposite direction. The other neuron's input will do the inverse of this. As the pole falls a force needs

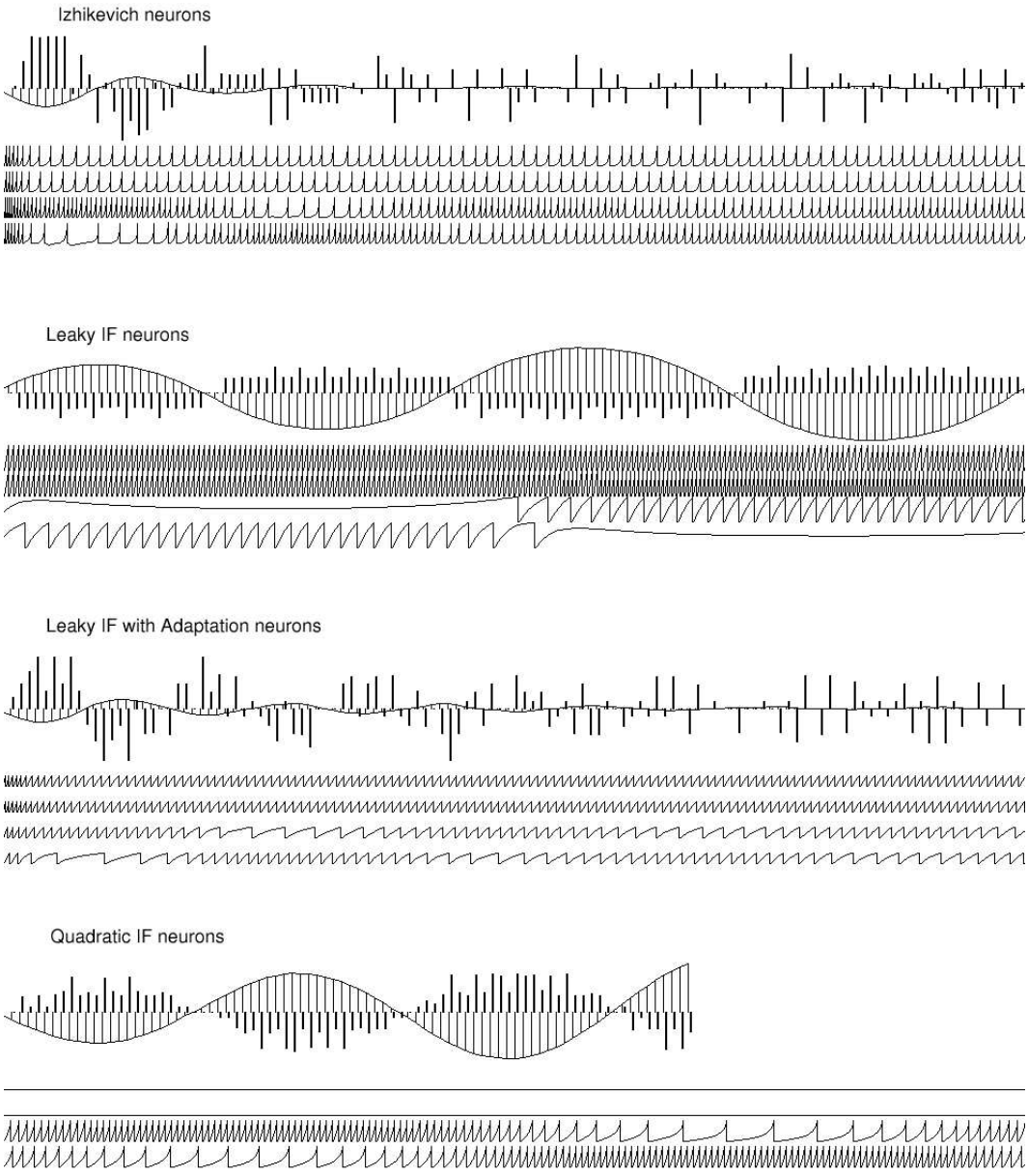


Figure 4.4: Single pole balancing. Controllers composed of four types of spiking neuron models are shown. It can be seen that controllers composed of Izhikevich neurons and leaky IF with adaptation neurons reduce oscillations of the pole over time. Whereas, controllers composed of leaky IF and quadratic IF neurons are unable to balance the pole.

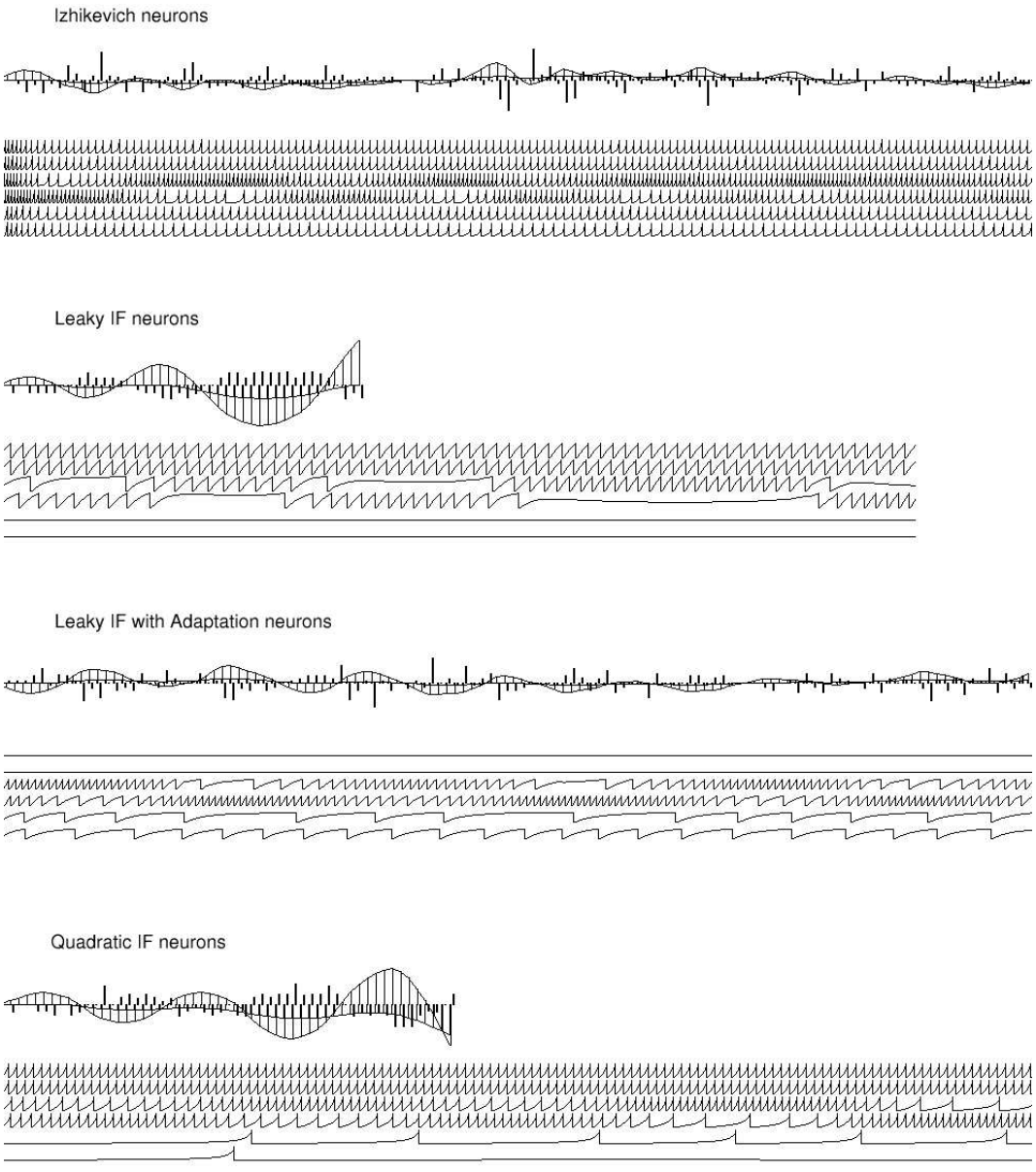


Figure 4.5: Double pole balancing. Controllers of four types of spiking neuron models are shown. Controllers composed of Izhikevich neurons and leaky IF with adaptation neurons keep the two poles near vertical.

to be applied to decelerate the pole and then to accelerate the pole back in the opposite direction towards vertical. Neurons that exhibit SFA apply more force decelerating the pole than accelerating it back again. This is because when decelerating the pole they fire with higher frequency because their input was previously low, but by the time the neuron is accelerating the pole back to vertical again adaptation has slowed their firing rate and the neuron therefore applies less force. If the pole is accelerated back to vertical with less force than was needed to decelerate it then the pole's velocity at vertical is reduced, dampening the oscillation of the pole. Whereas with neuron models without SFA the pole is accelerated with the same force applied to decelerate. Thus the poles velocity is not diminished and the pole oscillates wildly.

For a single pole, as the pole falls left we need to accelerate the cart to the left which has the effect of decelerating to pole. For two poles, things are considerably more complicated. Now, when the long pole falls to the left we actually need to accelerate the cart to the right, in order to get the short pole going the same direction as the long pole, then we can accelerate the cart to the left to bring both poles towards vertical. In the evolved parameters of successful controllers we do indeed see this rather counterintuitive result.

Other solutions to the double pole balancing with no velocities problem are compared in terms of how long it takes to evolve a solution. The current best method is NEAT of Stanley and Miikkulainen [36] which requires $30,000$ evaluations to evolve a solution that meets Gruau solution criteria. The spiking neuron controllers in the experiments described here, also find reasonable solutions after $30,000$ evaluations. However, the spiking neuron controllers are not quite as robust and do not meet Gruau's original solution criteria, but do satisfy a slightly weakened generalization criteria. It should be noted that evaluations on each of these systems is not equivalent. With NEAT a fitness evaluation is a single trial of $180,000$ steps, whereas a fitness evaluation in the evolution used in these experiments is the average of five trials of $3,000$ steps. The number of steps is fairly arbitrary. The number of trials differs because NEAT uses fixed initial state, whereas in these experiments there is a random element to the initial state and so several trials must be averaged. The spiking neuron controllers developed in these experiments almost compete with the current best double pole balancing solutions. In [16] Gruau concluded that a direct encoding of sigmoidal networks could not evolve a solution to double pole balancing with no velocity information problem. The complex encoding used by NEAT is crucial to its success. However as shown here, evolution with a direct encoding scheme was able to evolve spiking neural controllers to solve this difficult problem.

Spiking neuron controllers were able to solve the difficult double pole balancing with no velocities problem, using a very simple feed forward architecture of only a few spiking neurons with SFA. For control problems in general, dampening of oscillations is very desirable. This approach may prove effective for more difficult (higher degrees of freedom) control of unstable systems tasks, such as control of a robotic arm or of a legged robot. From the pole balancing experiments in was observed that spike frequency adaptation was needed for success. At this stage it would be useful to see how spiking neurocontrollers perform on another slightly different task. For this, obstacle avoidance on the Khepera robot, another well studied benchmark problem, was chosen.

Chapter 5

Khepera Robot

The Khepera robot is a simple miniture robot. It was chosen to further investigate spiking neural controllers for several reasons. Firstly, it has only a few degrees of freedom and relatively few sensors. Secondly, reasonably realistic simulators for it exist. Experiments were performed using a Khepera robot simulator to evolve neural controllers composed of spiking neurons of the four models discused in chapter 2. The well studied problem of obstacle avoidance was used in these experiments.

5.1 Khepera Robot Simulator

The Khepera robot is a miniature mobile robot with a diameter of 55mm, and two lateral wheels that can rotate in both directions. It has eight infra-red sensors that respond to nearby objects to about 5cm away, with 6 sensors at the front and 2 sensors at the back of the robot, see figure 5.1. Sensors give two readings: they act as both ambient light detectors and as proximity detectors [31].

A Khepera robot simulator was used [27] that attempts to faithfully reproduce the real world environment and dynamics as sensed by a Khepera robot. Sensor readings are noisy as real world sensors are, and the effect of actuators is also noisy, as wheels slip and motor output may vary.

5.2 A Neurocontroller for the Khepera Robot

The architecture of neurocontroller used for these experiments consists of four spiking neurons. There are two neurons for each wheel, where one neuron provides forward rotation and the other backward rotation. On each side, the difference between the forward and backward neurons' activities is used to set the speed of the wheel.

There are 32 inputs to the controller consisting of 16 sensor readings and their additive inverses. Inputs are scaled to between 0 and 1 using the sigmoid function. All the inputs are connected by weighted connections to each neuron, giving 128 weight parameters in all, see figure 5.2.

One step of the sensori-motor loop for a Khepera robot has a period of 100ms meaning that sensors are read every 100ms and motor commands sent to the neurocontroller. Neurons have a step size of 1ms so are updated 100 times during one sensori-motor step. The output value r for each neuron is calculated by counting the number of spikes n of the previous 100ms time window and using function $r = e^{-1/n}$ as with pole balancing. Each wheel speed is set every 100ms, using a running average of controller output calculated from the difference the forward and reverse neuron output values.

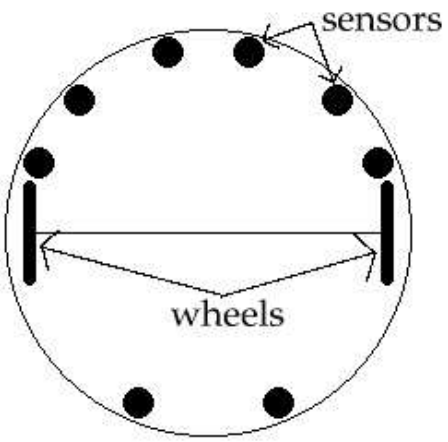


Figure 5.1: Khepera Robot in plan view, showing the position of the sensors and wheels.

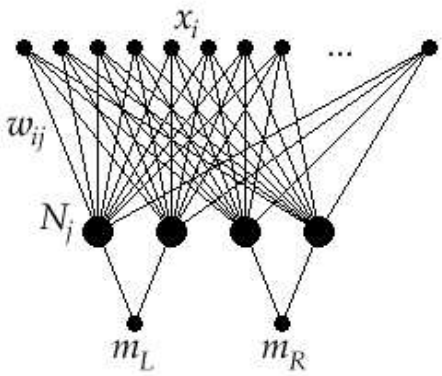


Figure 5.2: Neurocontroller for Khepera Robot. Inputs x_i from sensors are fully connected to neurons N_j. For each motor (M_L and M_R) there are two neurons. The difference between output of the two neurons gives the wheel speed.

5.3 Obstacle Avoidance

Obstacle avoidance is a well studied problem in evolutionary robotics. A suitable fitness function is well known for the evolving obstacle avoidance behaviour.

5.3.1 Experiment

The performance of controllers composed of the four neuron models dicussed in chapter 2 were compared on an obstacle avoidance task. The goal of the task is to navigate an enclosed space as quickly as possible without hitting any objects or walls.

A population of each type of controller was created with a population size of 100. The fitness of each individual in the population is evaluated by averaging their fitness over 5 trials. A trial consists of placing a controller randomly in a 38cm by 22cm enclosure and running them for 100 seconds (1000 steps) or until they collide with a wall or object.

The fitness function ϕ is modified from [29] [9],

$$\phi = V(1 - \sqrt{\Delta v})$$

where V is the average rotation speed of the two wheels, and Δv is the difference between the wheel speeds, with all values scaled to the range $(0, 1)$ so as to give a maximum fitness value of 1.

New individuals are created by taking a sample from a Gaussian distribution (with variance 0.01) centred on the weight vector of a parent selected in proportion to its fitness. The newly created controller is evaluated and replaces the worst individual of the population.

5.3.2 Results

The best controller of the population after $10,000$ evaluations is re-evaluated on the obstacle avoidance problem for 100 trials of 1000 steps (100 seconds) each. Three results are calculated from the 100 trials of the best controller:

- an average fitness score.
- the average number of timesteps it survives without collision.
- the number of successful trials (i.e those without any collision).

This experiment was repeated 100 times for controllers composed of each the four neuron models. The table below show the averages of the best controllers from 100 runs.

Obstacle Avoidance Results

Neuron Type	Fitness	stdev	Steps	stdev	Success	stdev
Izhikevich	0.5952	0.0663	817.4	63.0	80.6	7.6
Leaky IF	0.5614	0.0955	805.5	90.1	78.6	11.4
LIF with Adaptation	0.5973	0.0921	820.7	94.0	80.5	10.9
Quadratic IF	0.5698	0.0806	814.0	82.7	79.8	10.1

Controllers composed of the different types of spiking neurons all achieve very similar performance on the obstacle avoidance problem. The results do not show any type of controller to be significantly better. Although the Izhikevich and leaky IF with adaptation do look marginally better. Controllers show an 80% success rate, that is in 8 out of 10 runs the evolved controllers move around their environment without colliding with walls. Figure 5.3 shows some representative examples.

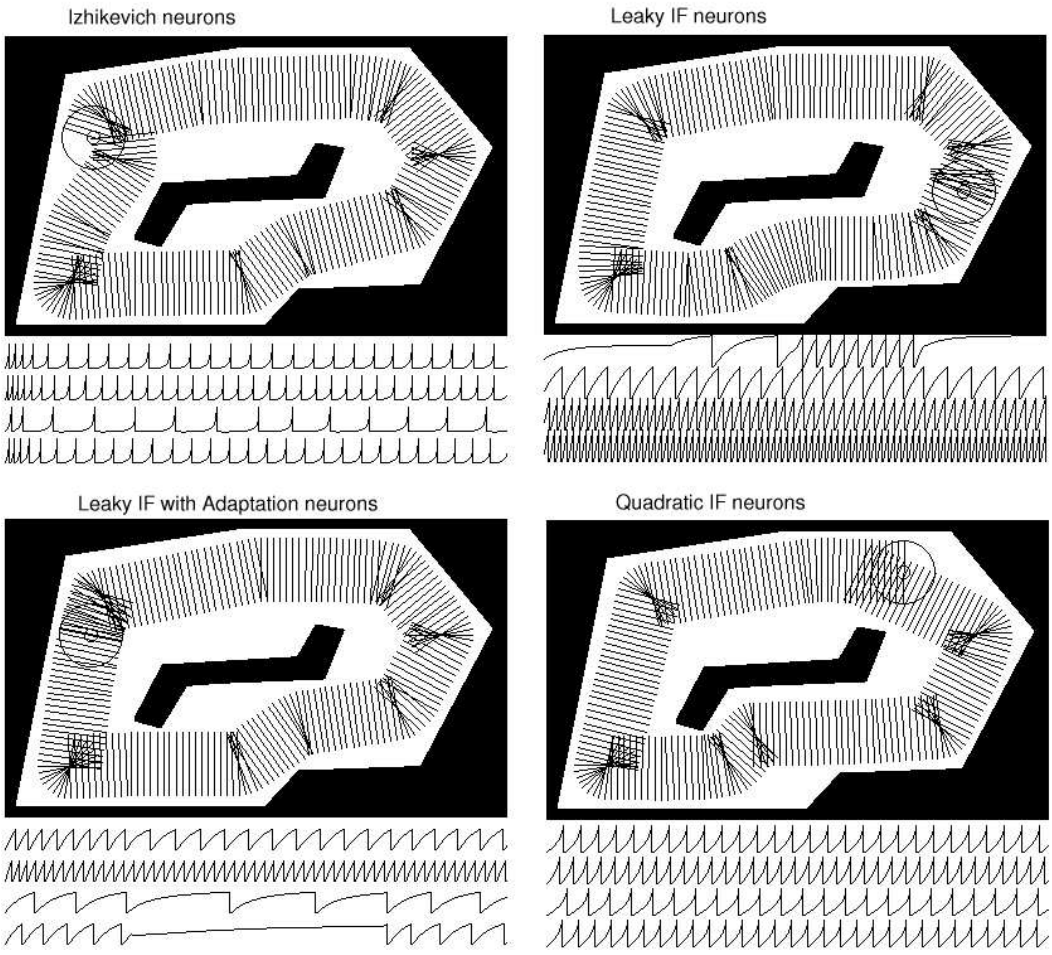


Figure 5.3: Obstacle avoidance by controllers composed of neurons of each of the four neuron models. Examples shown are of the best controller of each type from evolution of $10,000$ evaluations. The figures show a Khepera robot moving in an enclosed environment. Black areas are wall or obstacle and lines are the robot's axle position at a sensori-motor step (100ms). The circle shows the robot in its final position. Below each map the first fraction of a second of neuron activity is shown.

5.3.3 Discusion

Nolfi and Floreano [31] compared neural networks of sigmoidal neurons with and without recurrent connections. They found that recurrent connections were necessary to avoid certain deadlock situations. Without recurrent connections there exist equilibrium points for sensory input for which their effect cancels out and the wheels do not move [31]. Recurrent connections break the symmetry and act as a sort of decaying memory of previous state. It is interesting then that feed forward networks of spiking neurons do not suffer this problem.

The two neuron models that exhibit spike frequency adaptation do show slightly better performance but not significantly. This might suggest that perhaps the SFA property is useful for the obstacle avoidance task, but not nearly as much as it was for pole balancing. Obstacle avoidance with a Khepera robot is not an unstable system like the pole balancing systems. It is sufficient to learn a mapping from the current sensory input to appropriate motor action: for example seeing an object in close proximity to the front left of the robot should cause it to turn to the right. Obstacle avoidance requires only reactive intelligence [31]. Controllers of all neuron models were able to solve this task.

5.4 Pursuit-Evasion

Co-evolution has been used in previous studies of evolutionary robotics [5] [10] [30]. It is an appealing strategy because the fitness function does not need to explicitly specify the desired behaviour, and evolution is potentially open-ended.

5.4.1 Experiments

The performances of different types of neurocontroller were compared in a co-evolutionary scenario that is very familiar to biology - that of predator and prey species in competition with each other.

In the pursuit-evasion experiment two populations are created one of pursuers and one of evaders. There are four types of controller, each composed of neurons of one of the four neuron models. Each type of controller can be considered to be a different species. The pursuer population is made up of four species of pursuer, initially in equal proportion. Similarly the evader population is made up of four species of evader initially in equal proportion. Each population has a total size of 1000 controllers, and so each species of controller has an initial count of 250.

Fitness for pursuers and evaders is determined by competing a pursuer and evader against each other. Each evaluation takes a pursuer and an evader from their respective populations at random. They are placed 150mm apart facing left in an environment, and run for 200 timesteps (20 seconds), or until the pursuer catches the evader, or either pursuer or evader collides with an object or wall. The enclosure is 38cm by 38cm and contains three objects of approximately 55mm in diameter, see figure 5.4. The evader receives a score for the length of time it survives without being caught by the pursuer or colliding with any objects or walls. The pursuer's score is the length of time it took to catch the evader, or zero if the evader was not caught. An agent's fitness is a moving average. The fitness update for the pursuer is the time to catch evader t_{catch} scaled to the range $(0, 1)$, $\phi = t_{catch}/t_{max}$, and for the evader $\phi = 1 - t_{catch}/t_{max}$. The agent's fitness ϕ_e after evaluation e was computed by,

$$\phi_e = (1 - \gamma)\phi_{e-1} + \gamma\phi$$

where a γ value of 0.2 was used.

Randomly chosen pursuers and evaders are evaluated against one other, such that over time each pursuer will be evaluated against many different evaders and each pursuer will be evaluated against many different pursuers thus determining their effective fitness. While pursuers and evaders are being evaluated in this way new controllers are being created and added to the population, replacing the worst controllers. New controllers are created by proportionally selecting a parent from the population and taking a sample from the Gaussian distribution centred on the parent's weight parameters.

5.4.2 Results

Initially evader and pursuer populations contain a mix of four types of controller. Eventually one type of controller comes to dominate each population. Figures 5.4 - 5.6 indicate what typically happens as the co-evolution proceeds. The experiment was repeated 10 times. The table below shows the numbers of each type of controller in each population after 10^6 evaluations.

Evader populations after $1,000,000$ evaluations

Run	Izhikevich	Leaky IF	LIF Adaptation	Quadratic IF
1	505	80	177	238
2	578	128	94	200
3	94	140	715	51
4	346	387	98	169
5	389	180	316	115
6	38	61	763	138
7	680	158	133	29
8	308	195	231	266
9	160	84	642	114
10	452	66	360	122
Totals	3550	1479	3529	1442

Pursuer populations after $1,000,000$ evaluations

Run	Izhikevich	Leaky IF	LIF Adaptation	Quadratic IF
1	325	3	672	0
2	3	23	29	945
3	851	139	7	3
4	194	27	778	1
5	160	132	708	0
6	1	9	990	0
7	273	3	683	41
8	908	0	7	85
9	6	16	978	0
10	24	266	710	0
Totals	2745	618	5562	1075

These results show that for both evaders and pursuers it is the Izhikevich neuron model and the leaky IF with adaptation neuron model that typically come to dominate the population. Remarkably, the totals for the Izhikevich based evader and the leaky IF with adaptation based evader are nearly identical, as are the totals for the leaky IF based evader and the quadratic IF based evader. That is, the two models that exhibit SFA on average come to fill the populations in equal proportions, as do the two models without SFA. This is especially

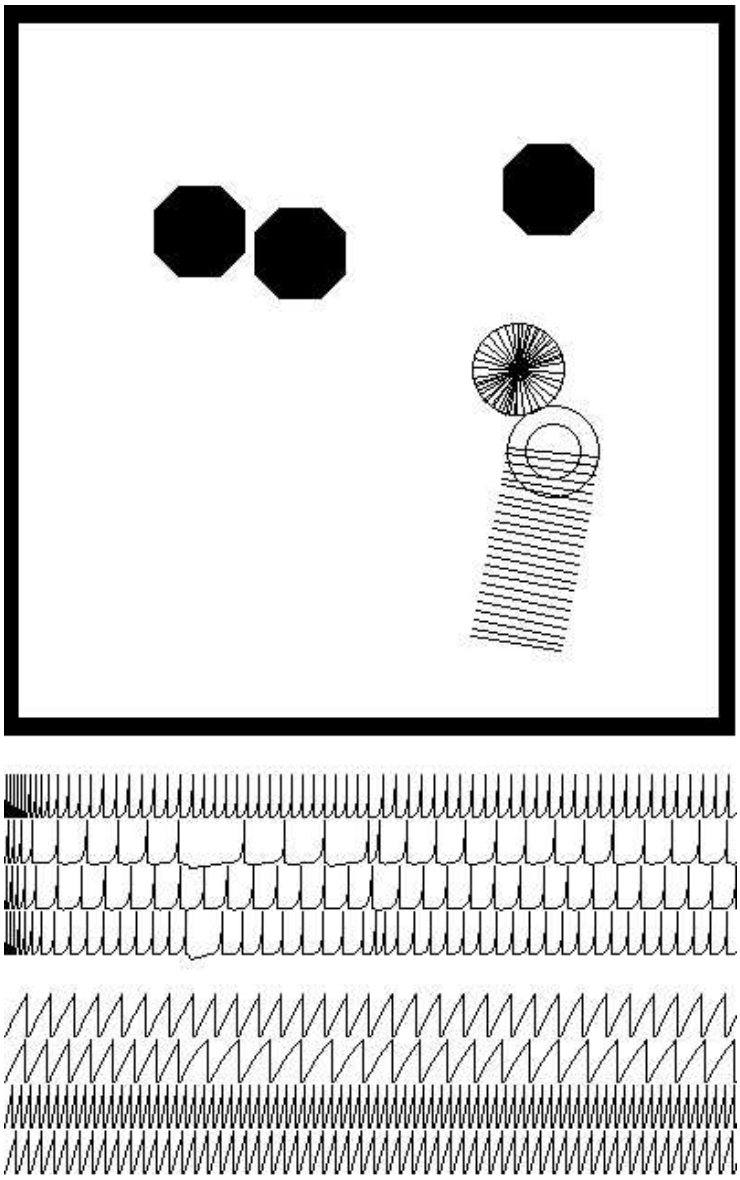


Figure 5.4: This figure shows the best pursuer vs the best evader, after 11000 evaluations. The figure shows the first fraction of a second of neuron activity of the evader (top) and the pursuer (bottom). In this competition the pursuer has moved straight towards the evader who has turned on the spot.

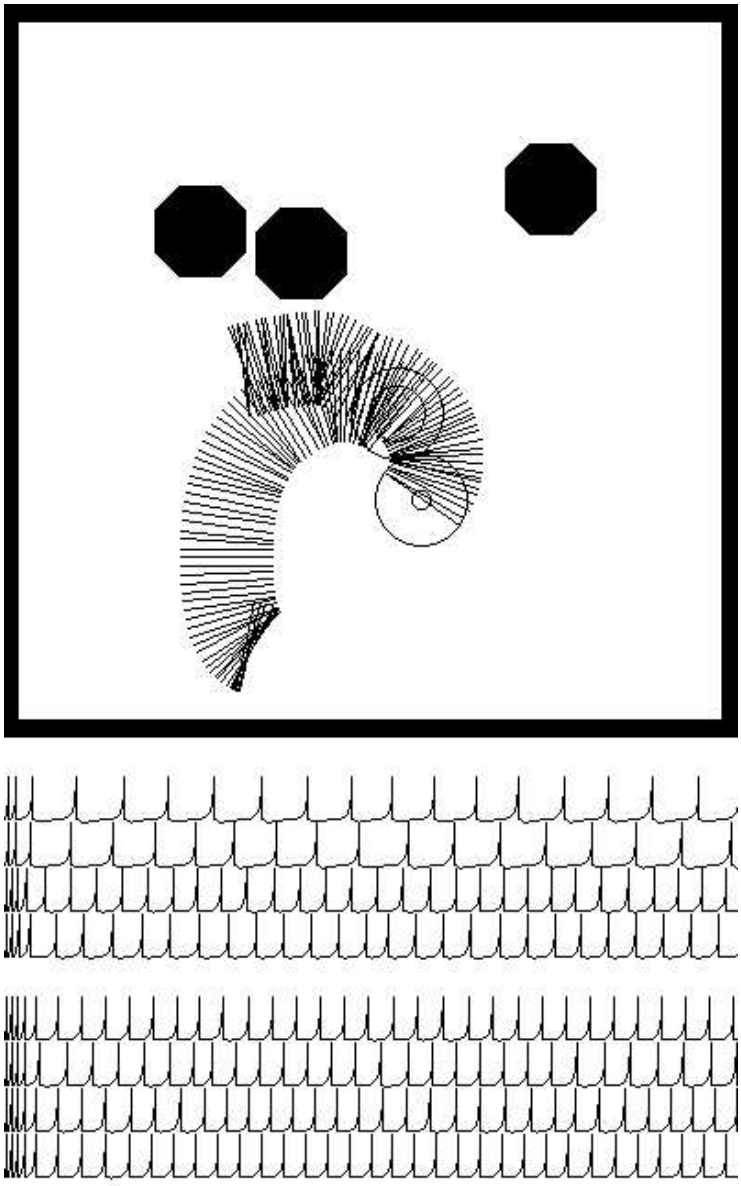


Figure 5.5: This figure shows the best pursuer vs the best evader, after 27000 evaluations. In this competition the evader has attempted to run from the pursuer who has turned towards and caught the evader.

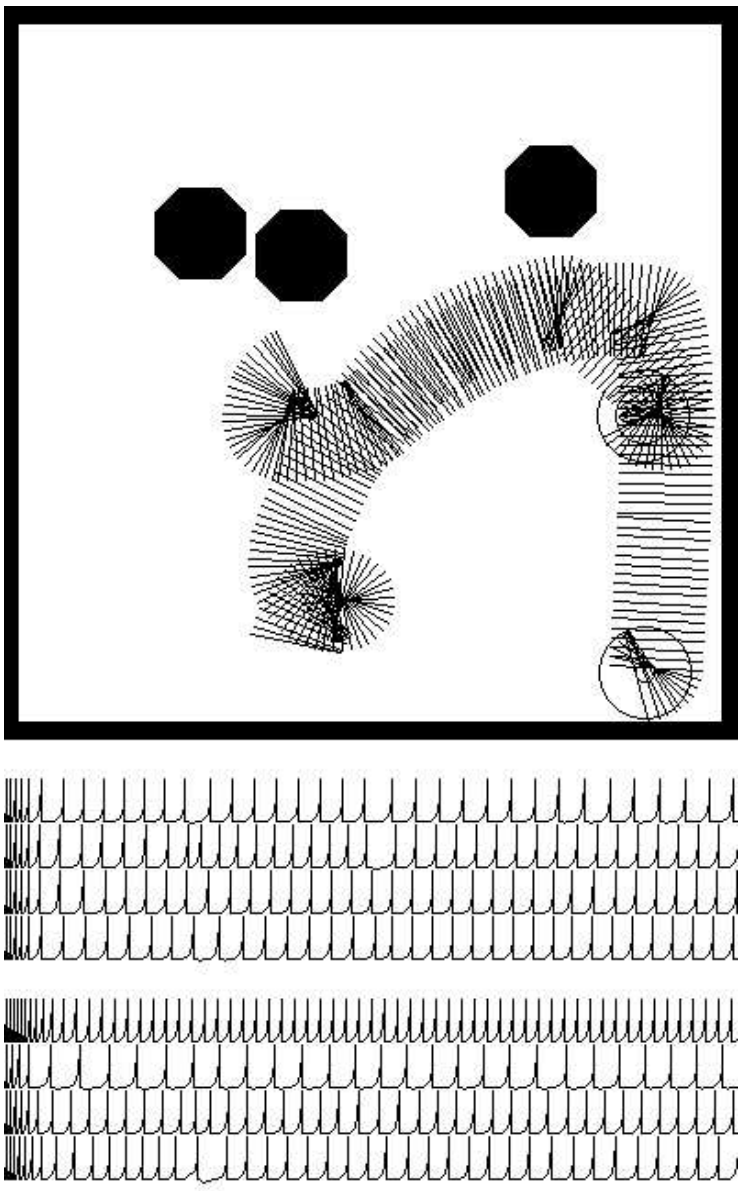


Figure 5.6: This figure shows the best pursuer vs the best evader, after 117000 evaluations. In this competition the evader has managed to evade the pursuer.

surprising given the huge variation between different runs.

5.4.3 Discussion

A novel method of comparing controllers was introduced in this experiment, that of evolving several types of controller simulataneously and observing which controller comes to dominate the population. By this method we again see controllers composed of neurons with SFA out-performing those without SFA. This new method shows differences between several types of controller. Whereas, the artificial evolution used in section 5.3 was unable to show any conclusive differences between the different types of controller. Coevolution does however require quite a large number of evaluations, as each controller needs many evaluations to determine its fitness.

Initially both pursuers and evaders are poorly adapted. The first behaviour of evaders to emerge is spinning in place, as they avoid colliding with anything with this behaviour. Meanwhile pursuers 'learn' to run towards evaders, to easily capture them as the evaders are not moving, as in figure 5.4. The next behaviour that emerges is for evaders to turn and run - early on they often collide with objects or walls but gradually improve, see figure 5.5. As evaders improve pursuers sometimes collide with objects or walls leading to improvement in pursuer obstacle avoidance behaviour, see figure 5.6, in what is termed an evolutionary 'arms race'.

Notice that obstacle avoidance behaviour is evolved without having a fitness function explicitly rewarding for obstacle avoidance behaviour.

Chapter 6

Conclusions

Four computationally inexpensive models of spiking neurons from the literature were investigated. Two models were readily able to solve a challenging version of the pole balancing task, in which two poles are balanced simultaneously, and agents have no direct access to the cart or pole velocities. Spike frequency adaptation, a distinctive feature of biological neurons, was found to be a crucial neuro-computational property. Spiking neurons models without spike frequency adaptation were unable to solve this task. Neuron models that exhibit adaptation have negative feedback to membrane potential, which dampens pole oscillations and leads to stable control. On the obstacle avoidance task, spike frequency adaptation appeared to be beneficial, but it was less conclusive than it was on pole balancing. It is unclear what other neuro-computational features it may be beneficical for controllers to possess on more complicated problems.

We have shown very simple networks of spiking neurons can be evolved to solve control problems and as controllers for simulated autonomous robots. Controllers with a feed forward architecture of spiking neurons were arrived at through an evolutionary process. These controllers were simpler than those arrived at by evolving conventional recurrent neural networks. Sufficient controllers are surprising simple yet are able to display quite rich behaviours. Small controllers with a feed-forward architecture of spiking neurons were able to perform the difficult double pole balancing problem with no velocity information. A task which otherwise requires recurrent neural networks and complicated schemes for evolving network topology.

A novel method of co-evolution was introduced that was able to differentiate between several types (species) of controller. Co-evolution typically involves evolving two populations of agents against each other. Here, co-evolution is extended such that each population consists of several different species of agents. The species of agent that comes to dominate the population does so at the expense of the other species and is assumed to be better than them in some way. Using this method it was possible to differentiate the performance of different types of controller whose fitness was nearly identical using a conventional artificial evolutionary method.

Bibliography

[1] BEER, R. D., AND GALLAGHER, J. C. Evolving dynamic neural networks for adaptive behavior. *Adaptive behavior 1*, 1 (1992), 91–122.

[2] BICKHARD, M. H. Representational content in humans and machines. *Journal of Experimental and Theoretical Artificial Intelligence* (1993), 285–333.

[3] BROOKS, R. Intelligence without reason. In *In proceedings of twelfth international joint conference on artificial intelligence* (San Mateo, CA, 1991), Morgan Kaufmann, pp. 569–595.

[4] CHANNON, A., AND DAMPER, R. Towards the evolutionary emergence of increasingly complex advantageous behaviours. *International Journal of Systems Science 31*, 7 (2000), 843–860.

[5] CLIFF, D., AND MILLER, G. F. Co-evolution of pursuit and evasion II: Simulation methods and results. In *From animals to animats 4* (Cambridge, MA, 1996), P. Maes, M. J. Mataric, J.-A. Meyer, J. B. Pollack, and S. W. Wilson, Eds., MIT Press, pp. 506–515.

[6] DELLAERT, F., AND BEER, R. *Toward an Evolvable Model of Development for Autonomous Agent Synthesis*. MIT Press Cambridge, 1994, pp. 246–257.

[7] DORIGO, M., AND SCHNEPF, U. Genetic-based machine learning and behaviour based robotics: a new synthesis. *IEEE Transactions on Systems, Man and Cybernetics* (1993).

[8] FLOREANO, D., AND MATTUISSI, C. Evolution of spiking neural controllers for autonomous vision-based robots. *Proceedings of the international symposium on evolutionary robotics from intelligent robotics to artificial life* (2001).

[9] FLOREANO, D., AND MONDADA, F. Evolution of homing navigation in a real mobile robot. *IEEE Transactions on Systems, Man and Cybernetics - Part B: Cybernetics* (1996).

[10] FLOREANO, D., AND NOLFI, S. God save the red queen! competition in co-evolutionary robotics. In *Genetic Programming 1997: Proceedings of the Second Annual Conference* (Stanford University, CA, USA, 13-16 1997), J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, Eds., Morgan Kaufmann, pp. 398–406.

[11] FLORIAN, R. V. Autonomous artificial intelligent agents. *Technical Report Coneural-03-01* (2003).

[12] GERSTNER, W., AND KISTLER, W. *Spiking Neuron Models*. Cambridge University Press, Cambridge, United Kingdom, 2002.

[13] GOLDBERG, D. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[14] GOMEZ, F., AND MIIKKULAINEN, R. Solving non-markovian control tasks with neuroevolution. In *International Joint Conference on Artificial Intelligence* (1999).

[15] GRUAU, F. *Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm*. PhD thesis, University Claude Bernard-Lyon, France, 1994.

[16] GRUAU, F., WHITLEY, D., AND PYEATT, L. A comparison between cellular encoding and direct encoding for genetic neural networks. In *Genetic Programming 1996: Proceedings of the First Annual Conference* (Stanford University, CA, USA, 28–31 1996), J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, Eds., MIT Press, pp. 81–89.

[17] HARNAD, S. The symbol grounding problem. *Physica D. 42* (1990), 335–346.

[18] HODGKIN, A. L., AND HUXLEY, A. F. A quantitative description of membrane current and application to conduction and excitation in a nerve. *Journal Physiol.* (1952), 500–544.

[19] IZHIKEVICH, E. Simple model of spiking neurons. *IEEE Transactions on Neural Networks* (2003).

[20] IZHIKEVICH, E. Which model to use for cortical spiking neurons. *IEEE Transactions on Neural Networks* (2004).

[21] JAKOBI, N. *Minimal simulation for evolutionary robotics*. PhD thesis, University of Sussex, 1998.

[22] KOZA, J. R. *Genetic-programming: On the programming of computers by means of natural selection*. MIT Press, Cambridge, MA, 1992.

[23] LUND, H., AND HALLAM, J. Sufficient neurocontrollers can be surprisingly simple. In *Research Paper 824*. Department of Artificial Intelligence, University of Edinburgh, 1996.

[24] MAASS, W. Networks of spiking neurons: The third generation of neural network models. In *Australian Conference on Neural Networks*, P. Bartlett, A. Burkitt, and R. Williamson, Eds. Australian National University, 1996, pp. 1–10.

[25] MAASS, W., AND BISHOP, C. *Pulsed Neural Networks*. MIT Press, Cambridge, MA, 1999.

[26] MEYER, J.-A. Evolutionary approaches to neural control of mobile robots. In *Proc. of the IEEE Int. Conf. on Systems, Man and Cybernetics* (Piscataway, NJ, 1998), IEEE Press.

[27] MICHEL, O. Khepera simulator package (version 2.0). http://diwww.epfl.ch/lami/team/michel/khep-sim/, 1996.

[28] MIGLINO, O., LUND, H. H., AND NOLFI, S. Evolving mobile robots in simulated and real environments. *Artificial Life 2*, 4 (1995), 417–434.

[29] MONDADA, F., AND FLOREANO, D. Evolution of neural control structures: some experiments on mobile robots. *Robotics and Autonomous Systems 16* (1995), 183–195.

[30] NOLFI, S., AND FLOREANO, D. Coevolving predator and prey robots: Do "arms races" arise in artificial evolution? *Artificial Life 4*, 4 (1998), 311–335.

[31] NOLFI, S., AND FLOREANO, D. *Evolutionary Robotics: The biology, intelligence, and technology of self-organizing machines*. MIT Press, Cambridge, MA, 2000.

[32] NOLFI, S., MIGLINO, O., AND PARISI, D. Phenotypic plasticity in evolving neural networks. In *In proceedings of the PerAc'94 Conference. IEEE Computer Society Press.* (1994).

[33] RICHARDSON, G. Eastern kentucky university. http://www.biology.eku.edu/RITCHISO/301notes2.htm

[34] ROGERS, A., AND PRÜGEL-BENNETT, A. Modelling the dynamics of a steady-state genetic algorithm. In *Foundations of Genetic Algorithms 5*, W. Banzhaf and C. Reeves, Eds. Morgan Kaufmann, San Francisco, CA, 1999, pp. 57–68.

[35] SASEK, C. National institute on drug abuse. http://www.drugabuse.gov/MOM/TG/momtg-introbg.html.

[36] STANLEY, K., AND MIIKKULAINEN, R. Efficient evolution of neural network topologies. *Proceedings of the 2002 congress on evolutionary computation* (2002).

[37] THORPE, S., FIZE, D., AND MARLOT, C. Speed of processing in the human visual system. *Nature 381* (1996), 520–522.

[38] VAVAK, F., AND FOGARTY, T. C. Comparison of steady state and generational genetic algorithms for use in nonstationary environments. In *International Conference on Evolutionary Computation* (1996), pp. 192–195.

[39] WHITLEY, L., AND KAUTH, J. Genitor: a different genetic algorithm. In *Proceedings of the Rocky Mountain Conference on Artificial Intelligence* (Denver, CO, 1988), pp. 118–130.

[40] ZIEMKE, T. On the role of robot simulations in embodied cognitive science. *AISB journal* (2003).