DOI: http://dx.doi.org/10.7551/978-0-262-32621-6-ch071

Evolving Spiking Networks for Turbulence-Tolerant Quadrotor Control

David Howard[1] and Alberto Elfes[1]

[1]CSIRO, Autonomous Systems Program, 1 Technology Court Pullenvale, QLD 4069 Australia
david.howard@csiro.au

Abstract

We investigate the automatic development of robust quadrotor neurocontrollers based on spiking neural networks. A self-adaptive evolutionary algorithm is used to generate high-utility topology/weight combinations in the networks, and a simple synaptic plasticity mechanism provides some degree of in-trial adaptation. Incremental evolution gradually increases the severity of environmental conditions that the agent can successfully handle. Results compare the spiking networks to tuned Proportional/Integral/Derivative controllers and feedforward neural networks for waypoint-holding experiments in varied atmospheric conditions. It is shown that the spiking controllers are able to maintain a closer distance to the waypoint than the comparative controllers, and more effectively deal with more challenging environmental conditions.

Introduction

The control of quadrotors in outdoor scenarios is a difficult task as both the agent and environment are highly dynamic. Traditional controllers, e.g. Proportional/Integral/Derivative (PID) (Minorsky, 1922), are still widely used in such scenarios, yet they are restricted to linear control and often require hand-tuning to achieve acceptable performance. Recent research has shown that nonstandard nonlinear controllers such as evolved feedforward neural networks (Rumelhart and McClelland, 1986) can automatically generate controllers that provide performance enhancements (Shepherd and Tumer, 2010).

Due to the dynamical nature of the task, *stateful* neural network representations may make suitable controllers due to their ability to (i) generate highly nonlinear action sequences through internal network dynamics, and (ii) harness latent information that may be found in temporally sequenced states (Maass and Zador, 1999). In this paper, we focus on the evolution of stateful spiking neural controlelrs (Gerstner and Kistler, 2002) for outdoor scenarios.

Spiking control systems offer a plethora of benefits, including temporally-sensitive neuron memory, the ability to perform highly nonlinear state-action mappings which may be demanded by the platform, and relatively straightforward mapping to power-efficient pulsed hardware implementations (important when considering future onboard control). Spiking networks are biologically plausible representations and as such permit numerous extensions based on neuroscientific principles — a simple model of synapse-level Hebbian plasticity (Hebb, 1949) here provides in-trial adaptivity through spike timing coincidences. Action calculation is also based on spike coincidences (at the output neurons).

A Genetic Algorithm (GA) (Holland, 1975) is used to automatically generate high-performance controllers by seeking beneficial network compositions — the weights and topological placement of the synapses, along with neuron numbers and types, are all under GA control. Self-adaptive mutation provides the GA with as much freedom as possible by allowing specific subcomponents of each network to mutate at varying rates.

Despite numerous recorded successes when spiking networks are used to control ground-based robots (e.g., (Floreano et al., 2003a; Rocke et al., 2007)), to date no spiking neuro-evolution has been carried out on highly dynamic flying platforms. We wish to investigate the benefits of pairing a dynamic, stateful controller representation with a problem that displays the same properties. Our hypothesis is that the properties of the spiking networks make them especially suited to the control of dynamic agents in dynamic scenarios. To test this hypothesis we incrementally evolve a population of spiking controllers in increasingly challenging wind conditions and compare performance to a hand-tuned PID controller and evolved Multi-Layer Perceptron (MLP) neural networks (Rumelhart and McClelland, 1986). The controllers must keep the quadrotor as close to a pre-defined waypoint as possible, in spite of prevaling wind conditions.

The main contribution of this paper is the first use of evolutionary computing to generate spiking quadrotor neurocontrollers. We additionally provide evidence that even simple plasticity schemes can be a useful provider of adaptation in dynamic scenarios, and introduce a novel action encoding scheme that permits the spiking networks to operate at a higher frequency than those using traditional e.g., rate-based encoding schemes.

Background

Spiking Neuro-Controllers

Spiking neural networks are a bioinspired information processing paradigm based on studies of neural structure and activity in the brain. A number of neurons are linked via unidirectional, weighted connections. Each neuron has a measure of excitation (membrane potential) and communicates to other neurons via voltage spikes. A neuron spikes when its membrane potential exceeds some threshold, which typically requires a cluster of spikes arriving at the neuron within a short time period. Produced spikes are received by all forward-connected neurons. As membrane potential provides implicit memory, spiking networks are able to produce temporally dynamic activation patterns (Maass, 1997). Saggie-Wexler et al. (2006) highlight that this property makes them particularly suited for temporal problems such as robot control as steady-state errors and unmodelled dynamics can be accounted for. Recent studies in cognitive science highlight the importance of dynamic processes in memory and learning (Buzsaki, 2006).

Non-evolutionary approaches to spiking robot control include dynamic learning of navigation behaviour via conditioning (Helgadottir et al., 2013) and real-time training of neuro-controllers based on resovoir computing (Burgsteiner, 2005). Wiles et al. (2010) creates biologically plausible models of a rat's brain via self-organised learning for phototaxis. Commonalities between the studies include reports of adaptive learning, high performance in dynamic environments, and fast-changing nonlinear behaviour generation.

Evolutionary Robotics (Nolfi and Floreano, 2001) involves the use of evolutionary algorithms to design robot controllers. Neural networks are frequently used in such a setting as they are especially amenable to evolutionary search for beneficial compositions of connection weights and network topology — generally termed *neuro-evolution* — and have been shown to generate high-performance controllers. Hagras and Sobh (2002) evolve spiking networks for online robot control. Full topology-plus-weight evolution is used for both monolithic (Clune et al., 2009) and ensemble (Howard et al., 2010) controllers. Of particular interest is the work of Dario Floreano's group, who produce compact controllers for ground-based robots (e.g., Floreano et al. (2003a)), and evolve a spiking controller for an autonomous blimp (Floreano et al., 2003b). They demonstrate that even simple spiking representations can effectively process temporal state information.

Hebbian plasticity (Hebb, 1949) is thought to provide the brain with the properties of adaptation and self-organisation. Mechanistically, if one neuron can be said to have caused the subsequent firing of another neuron within a short time window, the connection strength between the two neurons increases and more strongly correlates their future activity. The reverse rule causes weight decay if the postsynaptic neuron fires first. Plasticity can provide unsupervised in-trial learning, and as such is a valuable inclusion in robotic control systems that can be directly implemented in most spiking network models. Previous research by Wiles et al. (2010) demonstrates high performance of plastic networks in a frequency discrimination task in dynamic environments. Florian (2005) use a combination of STDP and global reward signals to effectively handle environmental uncertainty. Howard et al. (2014) show expedient adaptive re-organisation to recover from movement of the reward signal in the T-maze (Blynel and Floreano, 2003).

Evolving Rotorcraft Controllers

Evolution of rotorcraft controllers is difficult as finding an initial fitness gradient can be tricky — much of the network search space leads to controllers that crash the platform. Population seeding is often used to remedy this: three main examples include (i) incremental approaches (Hoffmann et al. (1998) generate fuzzy PID controllers for helicopter control), (ii) seeding with known successful controller parameters (De Nardi et al. (2006) evolve simple fixed-topology neurocontrollers that initially mimic PID control, and eventually evolve to outperform them; similar results are shown by Shepherd and Tumer (2010) for hierarchical waypoint-following tasks), and (iii) ensuring non-divergent networks in the initial population (Phillips et al. (1996) control a full-sized helicopter by evolving fuzzy modular controllers in a custom architecture based on how a human pilot flies a helicopter). All report higher performance when using seeding.Koppejan and Whiteson (2009) combine evolutionary algorithms and reinforcement learning for noisy hovering tasks similar to those we are interested in. Hovering tasks have been solved using Genetic Programming (Koza, 1992), e.g. by Dracopoulos and Effraimidis (2012).

Justification of our research methodology and approach is based on the knowledge that (i) spiking networks have previously been shown to be effective controllers for dynamic robotics tasks, (ii) evolved stateless neural networks (and other representations) have made promising candidate controllers for rotorcraft, and (iii) incremental approaches have shown to be beneficial given the difficulty of locating an initial fitness gradient. There is therefore some precedent to believe that incrementally evolved spiking quadrotor controllers will provide performance benefits when compared to stateless or traditional control schema in dynamic scenarios.

The System

To clarify the nomenclature that will be used herein: An *experiment* lasts for 500 evolutionary generations and focuses on a single controller in a single task. A *task* is a specific wind configuration in which the platform is tested. Each *generation*, 25 new controllers are created, trialled on the task, and assigned a fitness value. Each *trial* consists of a maximum of 300 timesteps and ends with a fitness value

being assigned. Each *timestep* begins with the current state being processed by the controller and ends with the resulting action being carried out and the simulator state updated. The state of the system is captured every 50 generations and used to generate the results. Fifteen experimental repeats for each controller type are used to create statistics and averages.

An incremental approach is used to focus the evolutionary search in areas that proved fruitful in similar, earlier tasks. This is designed to offset the poor initial fitness gradient caused by the catastrophic nature of failures, coupled with the fact that large portions of the network space were found to produce invalid controllers. For the first task, the population of networks is randomly generated with five hidden layer neurons. For subsequent tasks, the best 100 networks from the previous task are used as the initial population — each network is mutated at their current rates before being trialled on the problem.

Experimental Setup

We compare the ability of each controller type (spiking network, MLP, PID) to handle a series of four dynamic tasks. The QRSim (De Nardi, 2013) simulator was used as it models the physcial platform we intend to eventually deploy the controllers on, and provides a high-fidelity wind model.

The quadrotor is an Asctec Pelican, whose flight performance is captured through system identification. Sensors and motors are subject to realistic normally-distributed noise; pertinently the GPS is modelled in detail accounting for factors such as varying numbers of available satellites and errors associated with transmission through various atmospheric layers. The quadrotor's software layer — which transforms desired control inputs into rotor RPM — is qualititively modelled to reflect the real platform.

All tasks take place in a three-dimensional arena with boundaries for each axis at $x_{min}/y_{min}/z_{min}$=-10, $x_{max}/y_{max}/z_{max}$=10 (z_{min} is ground level). At the start of a trial, the quadrotor in initialised at target waypoint W with position $x = 0, y = 0, z = 0$ and the three control inputs are set to neutral (pitch θ=0^o, roll ϕ=0^o, thrust T=0.59) — see figure 1(a). Due to the complexity of the task, yaw is decoupled and stabilised to face North by a simple proportional controller. An image captured partway through a trial is shown in figure 1(b).

The quadrotor must stay as close as possible to W for 30 seconds of flight time — a control rate of 10Hz gives a maximum of t_{max}=300 timesteps per trial. At each timestep, the controller receives the noisy GPS-estimated difference between the quadrotor's estimated position and W in three dimensions, given as Δx_t, Δy_t, and Δz_t (units in metres, transformed to quadrotor body coordinates). Control inputs θ, ϕ, and T are calculated and passed to the simulator, which fuses them with the current yaw rate and wind state, and updates the platform state. Fitness f follows equation 1. Lower fitness indicates better hovering behaviour.

$$f = \sum_{t=0}^{t<t_{max}} (abs(\Delta x_t) + abs(\Delta y_t) + abs(\Delta z_t)) \quad (1)$$

Violating a boundary causes trial termination and penalises the controller with a final fitness of 30$\times$ the remaining number of timesteps in the trial, which is added to it's current fitness and is used as a pressure to generate controllers that can keep the quadrotor airborne.

We additionally compare the adaptation of the controllers to the task by recording the first generation that each experiment creates a *valid* controller G_{val} (that never violates a boundary), as well as the first generation with a controller that keeps the quadrotor within (i) 1.5m ($G_{1.5}$), (ii) 1m ($G_{1.0}$), and (iii) 0.5m ($G_{0.5}$) of the waypoint in every axis throughout the trial. If some of these criterea are not met by the end of the experiment, they are set to the maximum number of generations.

The tasks are as follows: (i) 1m/s steady wind, (ii) 1m/s gusting wind, (iii) 3m/s gusting wind, and (iv) 5m/s gusting wind, all with a general northerly direction. Wind is modelled as a noisy time-varying vector which is applied to the quadrotor centre of gravity. For steady wind, the magnitude of the vector is normally distributed around the mean values. Gusting wind follows a more in-depth model based on the Dryden spectrum, providing a plausible and challenging dynamic turbulence model — Liepmann (1952) provides full implementation details.

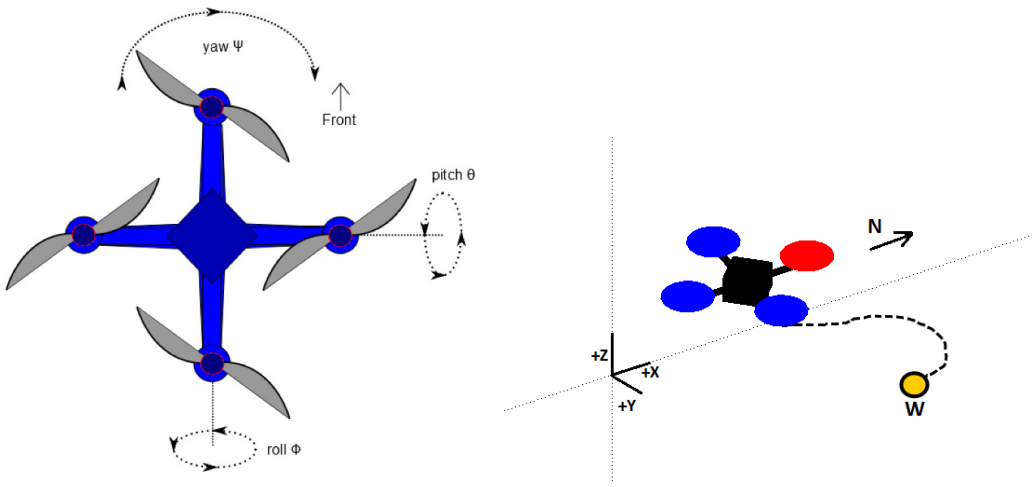
Spiking Controller

Leaky Integrate and Fire (LIF) (Gerstner and Kistler, 2002) networks are used as spiking controllers — three layers of neurons are connected by numerous weighted connections (weights constrained [0-1]), which can be recurrent within the hidden layer only. Neurons are either excitatory (transmit voltages $V \geq 0$) or inhibitory (V<0).

On network creation, the hidden layer is populated with 5 hidden layer neurons, whose types are intitially excitatory with P=0.5, otherwise they are inhibitory. Each possible connection site is initially likely to have a connection with P=0.5, and all connections are weighted uniform-randomly in the range [0-1].

During activation, stimulation by incoming voltage alters the membrane potential y, $y > 0$, which by default decreases over time. Surpassing a threshold y_θ causes a spike to be transmitted to all forward-connected neurons. The amount of voltage sent is equal to the weight of the connection, multiplied by -1 if sent from an inhibitory neuron. Immediately after spiking, the neuron resets its membrane potential to c. The membrane potential of a neuron at timestep t+1 is given in equation 2; equation 3 shows the reset formula.

$$y(t+1) = y(t) + (I + a - by(t)) \quad (2)$$



(a) (b)

Figure 1: (a)Top-down view of the quadrotor showing pitch, roll and yaw. Thrust controls the homoegenous RPM of the rotor blades. Arrows show the direction of negative change.(b)The test environment, zoomed to show the quadrotor. The quadrotor (red rotor front) must stay as close as possible to the waypoint W (yellow circle). A sample path is shown (dashed line).

$$\text{If } (y(t) > y_\theta)\ y(t) = c \tag{3}$$

Here, $y(t)$ is the neuron membrane potential at timestep t, I is the input voltage, a is a positive constant and b is the leak constant. A spike sent between two hidden layer neurons is received n ($n > 1$) steps after it is sent, where n is the number of neurons between the sending neuron and receiving neuron. A sample network is shown in figure 2. Parameters are $a = 0.3$, $b = 0.05$, $c = 0.0$, $y_\theta = 0.6$

A simple discrete-time linear model of plasticity is used to dynamically reconfigure the connection weights during the lifetime of the agent. The instantaneous weight w of a given connection is altered if it's two connected neurons spike in subsequent timesteps; no change occurs if both spike concurrently as causality cannot be implied. The sign of the weight change depends on which neuron — presynaptic (weight increase) or postsynaptic (weight decrease) — fires first, Δw=0.05 (experimentally determined). After a trial, each connection is reset to it's starting weight.

At a given timestep, three input neurons receive Δx_t, Δy_t, and Δz_t, which are scaled to the range [0,1]. The network sequentially updates it's input, hidden, and output layer neurons. Spike coincidences are checked and the plasticity rule applied to any affected synapses. Three pairs of output neurons control the commanded pitch θ (θ_{min}=-45^o, θ_{max}=45^o), roll ϕ,(ϕ_{min}=-45^o, ϕ_{max}=45^o) and thrust T (T_{min}=0, T_{max}=1). If the first neuron spikes and the second does not, the relevant control variable is increased by 0.05 of it's total range (experimentally determined). If only the second neuron spikes, the control variable is decreased by the same amount. If neither or both neurons spike, the control variable is unchanged. Note that with this action encoding, (i) the total time to traverse the entire range of each variable is 4.4s, and (ii) the controller does not need to wait for multiple network processing steps before generating an action as with rate-based schemes. This will become more important in future experimentation with onboard control.

Comparitive Controllers We compare the spiking controllers (stateful, nonlinear) to PID controllers (stateful, linear) and MLP networks (stateless, nonlinear).

PIDs are chosen as they are a common provider of closed-loop control. The PID is hand-tuned experimentally from ground-truth readouts from the simulator. Full PIDs are used for attitude and altitude control. The provided PIDs have more stringent pre-defined operating limits of $\pm 20^o$ for θ and ϕ and, due to their linearity, do not operate well outside of these bounds. More of the flight envelope is therefore available to the spiking/MLP controllers ($\pm 45^o$).

MLPs are a challenging benchmark that have previously shown promising results for quadrotor control. Sigmoidal activation functions are used and each hidden layer neuron has a random bias between 0 and 1. Rather than mapping continuous outputs to continuous angles/thrust, a similar pairwise encoding to the spiking networks is used, i.e., increase in a given variable occurs when the output of the first neuron is >0.2 more than that of the second neuron. Initial experimentation (not shown) compared the chosen encoding to a standard continuous-output interpretation and indicated no significant differences between the two encodings, possibly due to the compression effect of the sigmoid function around midrange outputs. Plasticity is not used as no spike-time information is available.

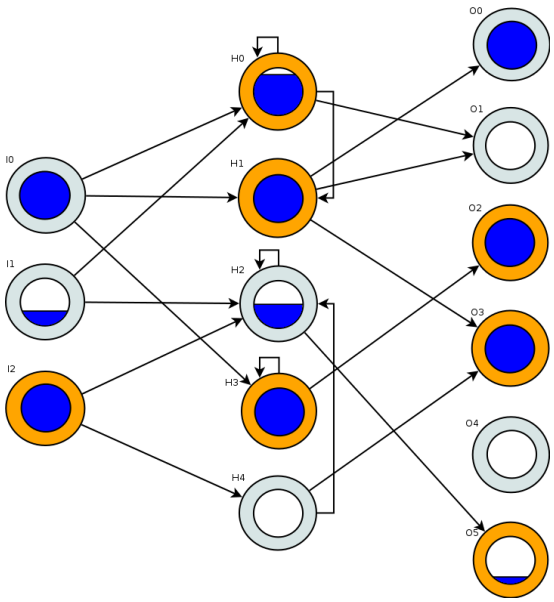


Figure 2: Showing a sample spiking network during activation. A neuron may be excitatory (orange) or inhibitory (grey), and has a membrane potential (centre circle). Membrane potential may build through a number of timesteps, eventually surpassing a threshold and emitting a spike (filled inner circle), before resetting (empty inner circle). Three input neurons take the differences in x, y, and z in the body frame between current position and target position. The six output neurons are paired; each pair controls either pitch, roll, or thrust. Actions are based on the presence or absense of instantaneous spikes at that timestep.

Genetic Algorithm

Per generation, following controller evaluation, the population is ranked based on fitness. The worst 25 networks are deleted. Children are then created up to the population limit of 100 by inverse-fitness-proportionate selection of a candidate parent, which is copied and probabilistically mutated before being inserted into the population. Crossover is omitted as — for these experiments — sufficient solution space exploration is obtained via a combination of weight and topology mutations; a view that is reinforced in the literature (e.g., by Rocha et al. (2003)).

Each network has it's own self-adaptive mutation rates, which are initially seeded uniform-randomly in the range [0,0.3] and mutated as with an Evolution Strategy (Rechenberg, 1973) as they are passed from parent to child following equation 4. This approach is adopted as it is envisaged that efficient search of weights and neurons will require different rates, e.g., adding a neuron is likely to impact a network more than changing a connection weight, so less neuron addition events than connection weight change events are likely to be desirable.

$$\mu \rightarrow \mu \; exp^{N(0,1)} \tag{4}$$

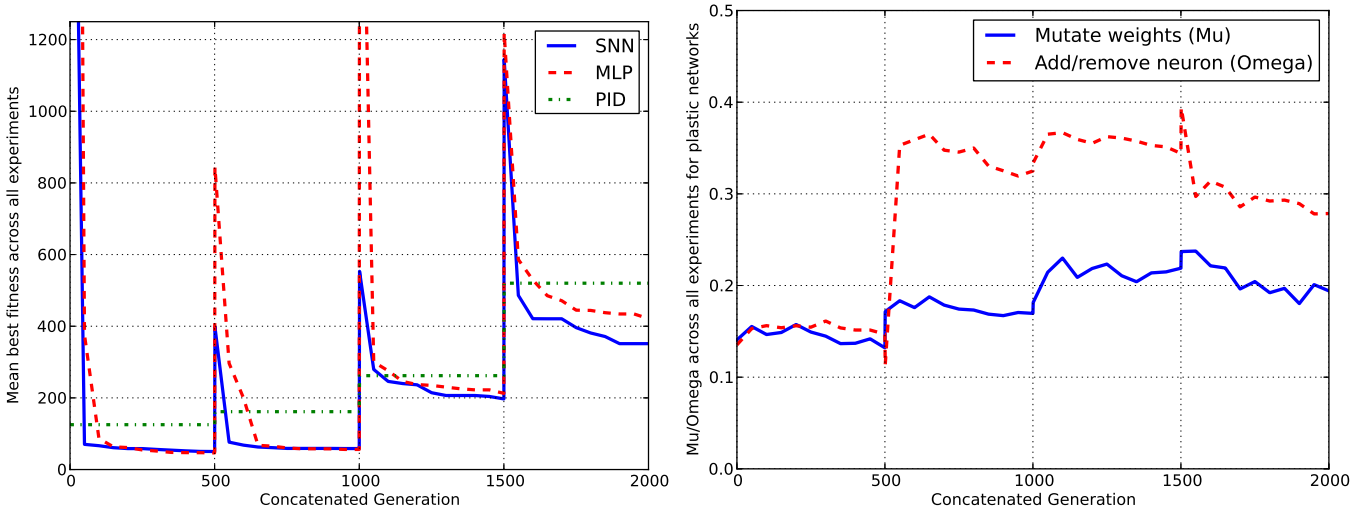
The genome of each network comprises a variable-length vector of connections and a variable-length vector of neurons, plus the mutation rates themselves. Different parameters govern the mutation rates of connection weights (μ), connection addition/removal (τ), neuron addition/removal (ω), and neuron type change (ζ). For each comparison to one of these rates a uniform-random number is generated; if it is lower than the rate, the variable is said to be *satisfied* at that allelle. During GA application, for each connection, satisfaction of μ alters the weight by $\pm$0-0.1. Each possible connection site in the network is traversed and, on satisfaction of τ, either a new connection is added if the site is vacant, or the pre-existing connection at that site is removed. ω is checked once, and equiprobably adds or removes a neuron from the hidden layer if satisfied. ζ is checked once per neuron, and changes a neuron's type from excitatory to inhibitory (or *vice versa*) if satisfied. New neurons are randomly assigned a type, and each connection site on a new neuron has a 50% chance of having a connection. New connections are randomly weighted between 0 and 1.

Results

In the following comparisons, statistical significance is assessed using twin-tailed t-tests. Results are significantly different when $p<0.05$. Initial experimentation on the first task justifies the use of plasticity. Compared to identically-evolved spiking networks with static connection weights, plastic networks have significantly better average best fitness (50.32 for plastic networks vs 55.21 for non-plastic), as well as faster creation of controllers that attain $G_{1.5}$ and $G_{1.0}$ (average 6.7 vs 18.3 generations), and $G_{0.5}$ (average 9.3 vs. 24.5 generations).

Performance

Figure 3(a) shows the performance of each controller through all experiments. We note that MLP networks find the transition between environments to be more challenging than spiking networks (larger initial increases in fitness before recovery). In the final task, spiking networks start to pull away from the MLP and PID controllers. A gradual upwards trend in final fitness values reflects the increasing challenge of the tasks. Table 1 reveals that, for the first three tasks, both MLP and spiking controllers have significantly better final fitness than PIDs. For the final task, spiking controllers have significantly better final fitness than MLPs and PIDs. Average fitness values are strongly influenced by the number of valid controllers produced per generation, and reflects the ability of the GA to create child networks that inherit useful features from highly-fit parent networks. This ability may be useful for future on-line/on-board evolution. Spiking controllers are found to be more "evolvable" in this



(a) (b)

Figure 3: (a) Fitness recovery between tasks. MLPs suffer from greater initial fitness spikes and take longer to recover. (b) Mutation rates for connection weights and frequency of node addition/removal events for spiking networks across tasks. Note (i) the significant variance of final used mutation rates for each task, (ii) the gradual increase in weight mutation and neuron addition/removal events in response to more challenging environments. MLPs display similar trends.

sense — for the final two tasks spiking networks have significantly better average fitness than MLPs. Related to this, spiking networks were found to be more quickly able to evolve controllers that displayed the ability to hover. For all tasks, spiking networks were significantly faster than MLPs at attaining $G_{1.5}$, and for the first two tasks at attaining $G_{1.0}$ and $G_{0.5}$. Only spiking networks achieve $G_{0.5}$ in task 3, and no controller type achieves $G_{0.5}$ in task 4. Performance differences in all cases are attributed to the ability of the spiking networks to handle temporal phenomena (all other experimental parameters were identical, network action encodings were made as similar as possible).

We additionally note that in tasks 3 and 4, spiking and MLP networks utilised their extended flight envelopes to quickly recover from being gusted; the highest angle used was for spiking networks in task 4 (34^o), compared to the PID maximum of 20^o. This behaviour was seen more frequently in the later tasks where stronger responses were required. Due to the amount of noise present and the variability of wind gusts, general controllers — unable to specialise to the conditions — were evolved.

Network Composition

Both neural networks benefit from the compressive effect of connection disabling; the best network of each type per task was always <50% connected. In the first task, MLP networks required significantly fewer connections and neurons than spiking networks (average 44.2% / 12.4 vs 48.3% / 14.8 respectively). The opposite is true in the final task (MLP average 50.9% / 15.2, spiking average 46.1% / 10.1). Smaller networks reduce computational requirements, increasing the chances that prospective onboard implementation will be viable. Plasticity increases with task difficulty, with significantly higher mean numbers of weight change events per network between tasks 1 and 3, 1 and 4, and 2 and 4, indicating that the extra flexibility permitted by the plasticity scheme is harnessed by the network.

Mutation Rates

Mutation rates generally increase throughout to allow sufficient exploration in more challenging tasks. Larger jumps are evidenced at the start of a new task (figure 3(b)) as the controllers require additional search of the network space to tailor the current population to the more complex task. After these peaks, small downward trends are evidenced as stability is brought into the search process. Peak size indicates how difficult the transition is for the controller.

Final rate values vary across tasks for the same controller. This highlights the context-sensitivity of the process and leads us to believe that a single mutation rate per parameter would be suboptimal overall. μ (rate of connection weight change), ω (rate of neuron addition/removal), and τ (rate of connection enabling/disabling) are often significantly higher in spiking networks than MLP networks — see table 2. A possible explanation is that the network types fundamentally differ in how they perform input-output mappings, and the spiking network space requires more variable search to locate fit solutions as a result.

Task	Controller	Best fit	Avg fit	G_{val}	$G_{0.5}$	$G_{1.0}$	$G_{1.5}$
0.5m/s	Spiking	50.3 (4.3)*	497.6 (127)	3.8 (3.9)†	9.3(5.2)†	6.7 (4.8)†	6.7 (4.8)†
	MLP	46.4 (6.3)*	465.9 (141)	11.3 (5)	65.9(17.5)	48.5 (17.1)	42.7 (16.3)
	PID	125.2 (55.5)	NA	NA	NA	NA	NA
0.5m/s Gust	Spiking	58.6 (7.5)*	352.3 (120)	0 (0)	5.6 (4.0)†	0.5 (0.98)†	0 (0)†
	MLP	56.1 (8.6)*	438.4 (82)	0 (0)	75.8 (34.8)	56.2 (34.2)	10.1 (8.9)
	PID	161.5 (75.1)	NA	NA	NA	NA	NA
3m/s Gust	Spiking	197.1 (24.3)*	811.4 (82) †	0 (0)	499.3 (2)	62.2 (4.8)	5 (6.7)†
	MLP	212.6 (34.4)*	1258.5 (108)	0.3 (0.7)	500 (0)	48.5 (17.1)	18.9 (16.1)
	PID	262 (88.5)	NA	NA	NA	NA	NA
5m/s Gust	Spiking	346.7 (68.4) † *	1555 (78) †	0 (0)	500 (0)	451.6 (102)	185 (174) †
	MLP	421.26 (54.8)	1746 (80)	0 (0)	500 (0)	500 (0)	391.2 (148)
	PID	520.1 (179)	NA	NA	NA	NA	NA

Table 1: Averages and standard deviations for performance parameters of Spiking, MLP, and PID controllers across all four environments. Symbols indicate the controller type is statistically ($p<0.05$) better than † = MLP, * = PID.

Task	Rate	Spiking avg (s.d)	MLP avg (s.d)
1	μ	0.132 (0.08)	0.024 (0.008)
	τ	0.017 (0.003)	0.012 (0.001)
2	μ	0.17 (0.04)	0.052 (0.03)
	ω	0.325 (0.12)	0.173 (0.137)
	τ	0.103 (0.07)	0.012 (0.002)
3	ω	0.344 (0.12)	0.185 (0.07)
	τ	0.161 (0.05)	0.024 (0.009)
4	τ	0.3 (0.09)	0.054 (0.01)

Table 2: Detailing averages and standard deviations for significantly different final self-adaptive mutation rates between the network types.

Effect of Incremental Evolution

In separate tests (not shown) we evolved spiking and MLP controllers for task 4 with random initial populations — neither produced a valid controller. This confirms that incremental evolution is seeding subsequent populations in useful areas of the search space. We also note that values of G_{val} for both networks generally decreases between experiments, i.e. valid controllers are evolved quickly despite the increasing difficulty of the task.

Discussion

We have demonstrated the benefits of spiking controllers for the control of dynamic platforms for dynamic tasks. The spiking controllers provided performance benefits on all tasks considered, with the fitness gap between spiking and MLP/PID control increasing with increasing task difficulty. Incremental evolution was found to be beneficial — seeded populations in harder tasks were significantly faster at creating valid controllers than random populations in the original, easiest task. Self-adaptive genetic search was found to be beneficial in tailoring search rates to the task considered by allowing the various constituent parts of the network to vary at different rates, and by allowing increased network space exploration at the start of each new task to allow the controller to adapt to the environment.

Referring back to our hypothesis, we have shown that spiking networks were more able to cope with the unpredictabilities inherent in the tasks than the comparative controllers, possibly through harnessing latent temporal information in the wind which was effecting the quadrotor. Spiking controllers were significantly more rapid than MLPs in keeping the quadrotor within at least one of the predefined bounds ($G_{0.5}$/$G_{1.0}$/$G_{1.5}$) on all tasks considered.

Results encourage further experimentation — we aim to transfer our controllers to a physical quadrotor and expect that plasticity will play a more prominent role in providing online adaptation to discrepances between simulation and reality. We also wish to investigate the possibility of harnessing additional bio-inspired spiking mechanisms and integrating them into the neuro-controllers.

Acknowledgements

This work was funded as part of the OCE Evolutionary Aerial Robotics project.

References

Blynel, J. and Floreano, D. (2003). Exploring the t-maze: Evolving learning-like robot behaviors using ctrnns. In Cagnoni, S., Johnson, C., Cardalda, J., Marchiori, E., Corne, D., Meyer, J.-A., Gottlieb, J., Middendorf, M., Guillot, A., Raidl, G., and Hart, E., editors, *Applications of Evolutionary Computing*, volume 2611 of *Lecture Notes in Computer Science*, pages 173–176. Springer Berlin / Heidelberg.

Burgsteiner, H. (2005). Training networks of biological realistic spiking neurons for real-time robot control. In *Proceedings of the 9th international conference on engineering applications of neural networks, Lille, France*, pages 129–136.

Buzsaki, G. (2006). *Rhythms of the Brain*. Oxford University Press.

Clune, J., Beckmann, B. E., Ofria, C., and Pennock, R. T. (2009). Evolving coordinated quadruped gaits with the hyperneat generative encoding. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 2764–2771. IEEE.

De Nardi, R. (2013). The qrsim quadrotors simulator. *RN*, 13(08):08.

De Nardi, R., Togelius, J., Holland, O., and Lucas, S. (2006). Evolution of neural networks for helicopter control: Why modularity matters. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 1799–1806.

Dracopoulos, D. C. and Effraimidis, D. (2012). Genetic programming for generalised helicopter hovering control. In *Proceedings of the 15th European conference on Genetic Programming*, EuroGP'12, pages 25–36, Berlin, Heidelberg. Springer-Verlag.

Floreano, D., Schoeni, N., Caprari, G., and Blynel, J. (2003a). Evolutionary bits'n'spikes. In *Proceedings of the eighth international conference on Artificial life*, pages 335–344, Cambridge, MA, USA. MIT Press.

Floreano, D., Zufferey, J.-C., and Mattiussi, C. (2003b). Evolving Spiking Neurons from Wheels to Wings. In *Dynamic Systems Approach for Embodiment and Sociality*, volume 6 of *Advanced Knowledge International, International Series on Advanced Intelligence*, pages 65–70. K. Murase and T. Asakura (eds.).

Florian, R. V. (2005). A reinforcement learning algorithm for spiking neural networks. In *Symbolic and Numeric Algorithms for Scientific Computing, 2005. SYNASC 2005. Seventh International Symposium on*, pages 8–16. IEEE.

Gerstner, W. and Kistler, W. M. (2002). *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press.

Hagras, H. and Sobh, T. (2002). Intelligent learning and control of autonomous robotic agents operating in unstructured environments. *Information Sciences*, 145(1):1–12.

Hebb, D. O. (1949). *The organisation of behavior*. Wiley, New York.

Helgadottir, L., Haenicke, J., Landgraf, T., Rojas, R., and Nawrot, M. (2013). Conditioned behavior in a robot controlled by a spiking neural network. In *Neural Engineering (NER), 2013 6th International IEEE/EMBS Conference on*, pages 891–894.

Hoffmann, F., Koo, T. J., and Shakernia, O. (1998). Evolutionary design of a helicopter autopilot. In *in 3rd On-line World Conf. on Soft Computing (WSC3*, pages 201–214.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, Michigan.

Howard, G., Bull, L., de Lacy Costello, B., Gale, E., and Adamatzky, A. (2014). Evolving spiking networks with variable resistive memories. *Evol. Comput.*, 22(1):79–103.

Howard, G., Bull, L., and Lanzi, P.-L. (2010). A spiking neural representation for xcsf. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8.

Koppejan, R. and Whiteson, S. (2009). Neuroevolutionary reinforcement learning for generalized helicopter control. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09, pages 145–152, New York, NY, USA. ACM.

Koza, J. R. (1992). *Genetic Programming: vol. 1, On the programming of computers by means of natural selection*, volume 1. MIT press.

Liepmann, H. (1952). On the application of statistical concepts to the buffeting problem. *Journal of the Aeronautical Sciences (Institute of the Aeronautical Sciences)*, 19(12):793–800.

Maass, W. (1997). Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659 – 1671.

Maass, W. and Zador, A. M. (1999). Dynamic stochastic synapses as computational units. *Neural Computation*, 11(4):903–917.

Minorsky, N. (1922). Directional stability of automatically steered bodies. *Journal of the American Society for Naval Engineers*, 34(2):280–309.

Nolfi, S. and Floreano, D. (2001). Evolutionary robotics. the biology, intelligence, an technology of self-organizing machines.

Phillips, C., Karr, C. L., and Walker, G. (1996). Helicopter flight control with fuzzy logic and genetic algorithms. *Engineering Applications of Artificial Intelligence*, 9(2):175–184.

Rechenberg, I. (1973). *Evolutionsstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution*. Frommann-Holzboog, Stuttgart, Germany.

Rocha, M., Cortez, P., and Neves, J. (2003). Evolutionary neural network learning. In *Progress in Artificial Intelligence*, volume 2902 of *Lecture Notes in Computer Science*, pages 24–28. Springer Berlin / Heidelberg.

Rocke, P., McGinley, B., Morgan, F., and Maher, J. (2007). Reconfigurable hardware evolution platform for a spiking neural network robotics controller. In Diniz, P., Marques, E., Bertels, K., Fernandes, M., and Cardoso, J., editors, *Reconfigurable Computing: Architectures, Tools and Applications*, volume 4419 of *Lecture Notes in Computer Science*, pages 373–378. Springer Berlin / Heidelberg.

Rumelhart, D. and McClelland, J. (1986). *Parallel Distributed Processing*, volume 1 & 2. MIT Press, Cambridge, MA, USA.

Saggie-Wexler, K., Keinan, A., and Ruppin, E. (2006). Neural processing of counting in evolved spiking and mcculloch-pitts agents. *Artificial Life*, 12(1):1–16.

Shepherd, III, J. F. and Tumer, K. (2010). Robust neuro-control for a micro quadrotor. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, GECCO '10, pages 1131–1138, New York, NY, USA. ACM.

Wiles, J., Ball, D., Heath, S., Nolan, C., and Stratton, P. (2010). Spike-time robotics: a rapid response circuit for a robot that seeks temporally varying stimuli. In *17th International Conference on Neural Information Processing (ICONIP)*.