International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 5 (2013) pp. 365-372
© MIR Labs, www.mirlabs.net/ijcisim/index.html

Evolutionary Spiking Neural Networks as Racing Car Controllers

Elias E. Yee[1] and Jason Teo[2]

School of Engineering and Information Technology, Universiti Malaysia Sabah,
Jalan UMS, 88400 Kota Kinabalu, Sabah, Malaysia
[1]*eliasyee0@gmail.com*
[2]*jtwteo@ums.edu.my*

***Abstract*—The Izhikevich spiking neural network model is investigated as a method to develop controllers for a simple, but not trivial, car racing game, called TORCS. The controllers are evolved using Evolutionary Programming, and the performance of the best individuals is compared with the hand-coded controller included with the Simulated Car Racing Championship API. A set of experiments using the sigmoid neural network was also conducted, to act as a benchmark for the network of Izhikevich neurons. The results are promising, indicating that this spiking neural network model can be applied to other games or control problems.**

***Keywords*—Car racing, evolutionary programming, games, Izhikevich neuron model, Spiking neural networks, TORCS.**

I. Introduction

Networks of spiking neurons have been increasingly gaining popularity over the years as a computationally powerful and biologically more plausible model of distributed computation [1]–[3]. It is the third generation of artificial neural networks, which was modeled to resemble the biological brain as close as possible, as the biological brain transmits information through electric pulses (action potentials), which is fired at certain points in time by the neurons. In an artificial spiking neural network, incoming pulses (spikes) stimulates a postsynaptic potential according to a response function, and when the voltage potential exceeds a threshold, it triggers a pulse. After the emission of the pulse, the neuron's membrane potential resets to its resting state. The input to the neuron does not affect the size and shape of the spike, but it affects the time when the neuron fires. Therefore, information is capable of being transmitting by individual spike timings, which in turn makes spiking neural networks capable of exploiting time as a resource for coding and computation in more sophisticated way than other conventional models [4]. Furthermore, spiking neural networks are able to simulate sigmoidal feedforward neural networks and approximate any continuous function.

Computer games have received much attention as computational intelligence research tools for many years, because it adds value and functionality to the games, and it allows researcher to use these games as test beds for research. Car racing is a challenging problem that could generate considerable excitement, which is evident from the multitude of resource invested in it by racers and observers alike, to practice and watch the races. Hence, the problem in racing is not trivial because many parameters influence it. To drive a car, the speed and steer has to be adjusted at the right amount and time, but many situations can happen, given that there are so many parameters influencing the behavior of the car, including the characteristics of the track, road curvature, inclination, surface friction, and banks. Others include the state of the car such as the current speed, acceleration, direction, slipping and skidding of wheels. Furthermore, cars have different characteristics including horsepower, traction, air resistance, and center of gravity [5] All these parameters manipulate how the car needs to be driven to achieve desirable results.

Spiking neural networks (SNN) are investigated in many areas and problems. Pavlidis et. al. evolved spiking neural networks using parallel differential evolution for classification problems [6]. There are studies in the area of robotics as well, including the evolution of spiking neural controller for a vision-based mobile robot [7], and indoor flight of a vision-based micro-robot composed of adaptive spiking neurons [8]. There are other application areas as well, such as temporal pattern classification, speech recognition, computer vision, XOR problems, associative memory, and function approximations

Artificial evolution of neural networks has been investigated by Togelius and Lucas for car racing [9,10]. Another study investigated the imitation of human behaviors in driving using the TORCS racing game [11]. However, spiking neural networks have not been investigated before as a computational intelligence technique in evolving racing car controllers. Hence, motivated by the encouraging results of SNN applications in other domains, this forms the main objective of our study. A successful outcome will not only demonstrate the usefulness of SNNs as a potential car racing AI agent but also in other computer game genres or even real-world problems that exhibit similar real-time control requirements.

This is an extended paper of an earlier published work in which we considered the application of the Izhikevich spiking neuron model for the neural network, trained using Evolutionary Programming, for the control of a simulated racecar [12]. In this paper, we included a set of experiments using the sigmoid neural network as an additional benchmark, alongside other controllers, including the rule-based and TORCS included controllers, as well as the human

player, for the spiking neural network. Section 2 briefly introduces the Izhikevich spiking neuron model, and section 3 describes the methods applied, including the simulator, fitness function, and optimization algorithm. Then, the report of the experimental results are presented and discussed. The paper concludes with a summary of the current work and ideas for future works.

II. Spiking Neural Network

The nature of biological neurons led to explorations of modeling the neuron into computational models. A widely known mathematical model, which earned its authors the Nobel Prize, is the Hodgkin-Huxley model. This model reproduces the behavior of the giant axon of the squid and includes terms that represent the specific ionic currents through the neuron membrane. This neuron model accurately represents many of the behaviors of biological neurons, but it places a significant burden on digital computers because it comprise of four coupled differential equations. On the other hand, the Integrate-and-Fire model is more computationally efficient, but it is unable to produce the many spiking behaviors exhibited by biological neurons. Dr. Izhikevich presented a paper in 2003 that describes a new simple spiking neuron model that is capable of reproducing many neuron behaviors while also maintaining computational efficiency. The section below describes a brief introduction to the Izhikevich neuron model.

A. Neuron Model

Izhikevich (2003) introduced a neuron model that is capable of producing many patterns of biological neurons, which is as biologically plausible as the Hodgkin-Huxley model, yet as computationally efficient as the integrate-and-fire model. This model is a simplification of the Hodgkin-Huxley model to a system of two ordinary differential equations. These two equations describe the membrane potential, v, and the recovery variable, u, which is roughly considered to represent the activation of K+ and the inactivation of Na+ ionic currents, and provide negative feedback to the membrane potential, v, with an auxiliary after-spike reset rule. When the membrane potential, v, exceeds its peak of 30mV, an action potential (spike) occurs. The membrane potential is reset to its initial value, c, and the recovery variable is incremented by d. When $v \geq 30$, then $v \leftarrow c$, $u \leftarrow u + d$. Synaptic currents are conducted to the neuron through the variable I. The typical time-step used with this model is 1ms.

The variables a, b, c, and d are dimensionless model parameters that have constant values. The variable a is the time scale of the recovery variable, u, where smaller values mean slower recovery. The variable b is the sensitivity of the recovery variable, u, to the sub-threshold fluctuations of the membrane potential, v, where bigger values couples the variables v and u more strongly, which would result in low-threshold spiking dynamics. The variable c is the membrane potential after spike reset value that is caused by the high-threshold K+ conductance. The variable d is the recovery variable after-spike reset value that is caused by the slow high-threshold Na^+ and K^+ conductance [13], [14].

The Izhikevich model is capable of producing the firing patterns that biological neurons could produce, and are classified as excitatory, which includes regular spiking, intrinsically bursting, chattering, and thalamo-cortical neurons; and inhibitory, which includes fast spiking, resonator and low-threshold spiking neurons. Furthermore, this model is capable of modeling all twenty identified firing behaviors that neurons could exhibit by tuning the four constant parameters of the model. Although, not all spiking behaviors could be modeled simultaneously as some behaviors are mutually exclusive.

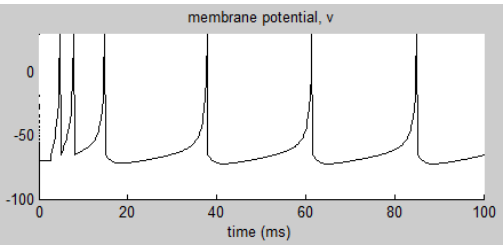

Figure 1. Voltage response of a neuron model exhibiting a regular spiking firing pattern, with input current, I = 20.

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I$$

$$\frac{du}{dt} = a(bv - u)$$

This model does not have a fixed firing threshold but is dependent on previous firings and can be anywhere between -55 mill volts (mV) and -40mV [13].

III. Methods

This section describes the controller, its environment, fitness function, optimization technique, and the experiment setup.

A. Car Simulator

The racing simulator that is employed for performing experiments in this paper is *The Open Racing Car Simulator* (TORCS). TORCS is a very realistic simulator with a sophisticated physic engine and many game contents like different cars, tracks and controllers. TORCS is not only an open source racing game but was also designed so that anyone could create their own car controller. This simulator takes into account many aspects including damage due to collision, fuel consumption, aerodynamics, wheel slippage and so on. This section describes only the relevant aspect for the controller used.

The Computational Intelligence in Games (CIG) Simulated Car Racing Championship provides an API to TORCS for developments. It is a client-server architecture where the controllers run as external processes and connected to the server through UDP connections. Furthermore, races run in real-time where in every game-tick, which roughly corresponds to 20 milliseconds (ms) of simulated time, the server sends sensory information to the client and waits 10 milliseconds (ms) of real-time for an action respond. If no action signals arrive, the server will use the last performed signal.

There are many sensor information available to the controller, but only those that we think are most essential are used. These include:

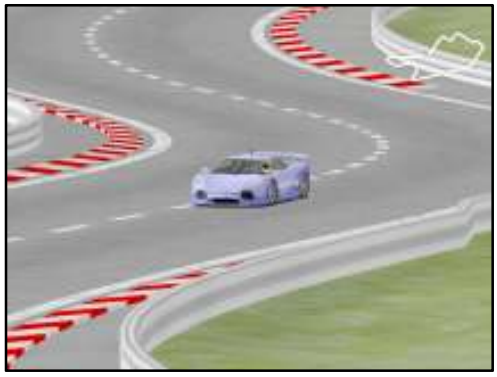

Figure 2. Track 3 of The Open Racing Car Simulator (TORCS)

- The angle between car direction and direction of track axis.
- Distance between the car and track axis, and the distance between the car and track edge within 200 meters.
- Car speed along the longitudinal axis of the car.
- Current gear and revolutions per minute (R.P.M.).
- Rotation speed of the wheels.
- Distance raced, current and last lap time.

The actuators to control the car include the steering wheel, gas, brake, and clutch pedals and the gearbox. However, the car is assumed to have automatic transmission, hence active gear changing is not necessary, but may be included in future works. The car is also assumed to have an anti-lock braking system (ABS). Both the automatic transmission and ABS were adopted from the controller that came with the CIG competition's API.

B. Controller

The car is controlled by a feedforward network of Izhikevich model neurons. In the experiments reported in this paper, the network is composed of eight input neurons and four output neurons. The inputs are directly mapped to the outputs, so it has no hidden layer, which means this is a single layered network. The neural network weights are real numbers with values in the range of [-1, 1]. We employed a spike rate encoding method, which is similar to the method used by Floreano [7]. This means that the strength of the stimulation is represented by the probability of spike emissions within a given time interval. We have taken the firing rate of the neurons measured over 20 milliseconds (ms) as commands for the decision of the controller to steer, accelerate and brake. The Izhikevich neuron model parameter values used correspond to cortical pyramidal neurons exhibiting regular spiking firing patterns [15].

The inputs include the angle between the car direction and track direction, distance between the car and track axis, speed of the car, and five range finder sensors to measure the distance between the car and the track edge, and all inputs are normalized to have the value [0, 10]. The outputs, with values in the range of [0, 1], correspond to the steering wheel, gas and brake pedals. The command to steer right or left is determined by the difference of two outputs, where a negative value means to steer right, and a positive value means to steer left. On the other hand, the acceleration and brake values are denoted by the difference between the other two outputs. A positive value indicates to accelerate, while a negative value indicates to brake.

Figure 3. Human player racing against SNN and TORCS controllers (left). SNN controller racing against TORCS controller (right).

In addition, we also ran tests using a sigmoid neural network with most parameters being similar to the spiking neuron network, with the exception that the sigmoid neural network has a hidden layer of six hidden units, with a continuous firing rate, instead of a firing rate measured over 20 milliseconds (ms) as used for the spiking neuron network. Another difference is the values of the inputs, which are normalized to have values in the range of [0, 1]. Apart from that, the number of inputs, outputs and the correspondences of the inputs and outputs are the same.

C. Fitness Function

The fitness function used is somewhat similar to the fitness function used by Simmerson, winner of the WCCI 2008 simulated car-racing competition [16] with some differences.

The controller's fitness is determined by how far the car was driven, average speed and amount of damage it took throughout the whole race, and the ability to stay inside the track, measured using the number of time-steps, also known as game ticks. In this paper, we employed 10000 time-steps for the evaluation of each controller, which is roughly about 3 minutes of simulated time.

The fitness of a controller is given by the equation:

$$Fc = d_{raced} + 100\left(\frac{\sum_0^{T_{max}} v}{T_{max}}\right) + (T_{max} - T_{out}) - D$$

where d_{raced} is the total distance raced, v is the velocity of the car, and T_{max} is the maximum number of game ticks in each race. T_{out} is the number of game ticks the car had been outside the track, and D is the amount of damage the car had sustained.

While a car is being evaluated, the damage the car took is also monitored, so much so that if the damage exceeds 1000, the controller is immediately disqualified, and the evaluation of the next controller will begin.

D. Optimization

Evolutionary Programming is employed to optimize the controller for all experiments described in this paper. In each generation, the fitness of all controllers were evaluated and compared with their respective parents. The controller with the higher fitness score will act as parents for the next generation. Their genes are perturbed with a Gaussian distribution random generator based on a constant mutation rate, to produce the offspring for the next generation. Experiments in this paper are conducted with a population size of 10 controllers and 10 runs, with 1000 generations

each run, unless stated otherwise.

E. Experimental Setup

The clients for the spiking and sigmoid neural networks were developed by extending the sample client included with the TORCS CIG competition API. Loops were added into the main controller for the generation and number of drivers in each generation. The driver was redeveloped to include a neural network object and a function call to process the outputs from the inputs. However, the automatic transmission and anti-lock braking system functions included with the sample driver were retained.

Preliminary experiments were carried out to test if the controller could be evolved to respond to turns or curvatures on a simple track and then on a more difficult track. Then, experiments were carried out to race without going off the track, and drive as fast and far as possible, in the hope of completing a lap within the shortest time.

The three tracks used in experiments are depicted in Fig.4, starting from the simplest track to the hardest track of the three. Track 1 is 1908.32 meters in length, while track 2 is 2057.56 meters, and track 3 is 3823.05 meters in length.

As mentioned in the methods section, the experiment was carried out with three tracks, each track consists of 10 runs. The population goes through 1000 generations each run, with 10 drivers in each generation, and each generation has 10000 game ticks. All these remained constant throughout the experiment, and all racecars begin the race from the starting line. The mutation rate used is 0.7, and the Gaussian distribution has a mean 0 and a standard deviation 1, N (0, 1). A general overview of the implementation is given in the flowchart.

IV. Results And Discussions

A. Spiking Neural Network

Results showed that the evolved spiking neural network (SNN) controllers managed to race through all three defined tracks, with minimal or no damage. The controllers even showed sophisticated driving techniques when turning corners. Figures below present the collected results of the best controllers of all runs for each track.

Fig. 6 presents the average fitness growth of the best SNN controllers evolved on track 1. It shows a sharp improvement of the controllers during early generations, slowed down after the first hundred generations, and converges around generation 500. The population was probably lucky enough to produce well performing offspring in the beginning. The spiking neural network controllers evolved on track 1 have the highest fitness scores among the three tracks, reaching values as high as 50,000, but these scores are dependent on the environment as well. As track 1 is much simpler than the other tracks, the controllers do not need to slow down too much while turning, so it gathered scores from larger distance raced, and higher average speeds. Having said that, some explanations for the early discovery of good performing solutions is most likely because track 1 is easier to drive in. Some interesting points to note, in addition to the track being easy, are the sophisticated driving behaviors the controller has developed on the track, where it made some distance to the outer side of a curve to turn, instead of making a sharp turn, thus it does not need to slow down too much for each turn.

The charts in Fig. 7 and Fig. 8 show a similar trend as Fig. 6, but have lower fitness scores and are more curved. The experiment on track 2, as presented in Fig. 7, had a steep but steady climb in early generations, until around generation 150 when the steep climb ended, and the population started to progress slowly. It was not until around generation 800 when the controllers start to converge. However, considering the pattern of the chart, the controllers might not yet converge, since the graph showed that the controllers made some improvements at almost the end of the runs, signifying further improvements are possible.

The experimental results from the SNN controllers on track 3 on the other hand, as presented in Fig. 8, show that the controllers raced in it gained lower scores than those from track 1 and 2, barely reaching 30,000. As track 3 has U-turns and more curves, it explains why controllers in track 3 gained lower fitness scores, because they needed to slowdown for turns more. Nevertheless, the population also experienced a quick improvement in their performance in early generations, and only started to slowdown after approximately a hundred generations. Yet the population still maintained its' advance albeit slowly until the end. Like the results in Fig. 7, the populations evolved on this track most likely have not converged, as can be seen from the graph in Fig. 8, where the population's fitness have the potential to increase if given more generations to evolve in. Furthermore, if drawn a trend line, we can see that the graph have not leveled out.

After having obtained the results, we re-simulated the solutions to observe their behavior on the track. We found out that the controller evolved on track 1 was driving at top speed and not slowing down much during turns, which is good. Conversely, the controllers evolved on tracks 2 and 3 did not drive at top speed, but managed to avoid colliding into or gliding along the rail of the tracks. We also noticed that the controllers did not accelerate at maximum (flooring the gas pedal) often, even on straight roads, which give reasons why the cars seldom or did not reach full speed. Although it is quite reasonable not to accelerate at maximum in places that has many turns, probably the genes to floor the gas pedal in straight roads had not emerge yet. We also observed that many controllers are still going out of the track in track 3, particularly during the hard turn after the straight road, as it took speed but failed to slowdown enough to avoid skidding off the track. Yet some controllers managed to have evolved the behavior to overcome skidding off the track.

Early 100 generations consists of the controllers crashing into the sides of the track and gliding against the rail, which deducted fitness scores, driving slowly or not being able to race far. Hence, the logarithmic-like graph, but after that early generations, the controllers started to improve very slowly. By this point, they were already able to drive fairly far or complete a full lap, avoid crashing into the sides or going out of the track often. Therefore, the only thing remaining is for them to learn strategies for increasing speeds at certain segments of the track and how much they needed to slow down for each turn, in addition to strategies for turning, such as avoiding sharp turns to avoid losing too much speed, so larger distance could be covered.

Table 1. Best Lap Times of Spiking Neural Network

Run	Best Lap (seconds)					
	Track 1		*Track 2*		*Track 3*	
1	25.98		48.65		128.47	
2	25.77		48.27		126.58	
3	25.74		44.87		127.46	
4	25.79		46.73		129.47	
5	25.49		47.21		128.55	
6	25.82		44.71		125.78	
7	25.89		46.67		124.99	
8	25.82		45.89		124.42	
9	25.92		46.73		125.74	
10	25.72		51.48		125.76	
μ / σ	25.79	0.1340	47.12	1.9879	126.72	1.6871

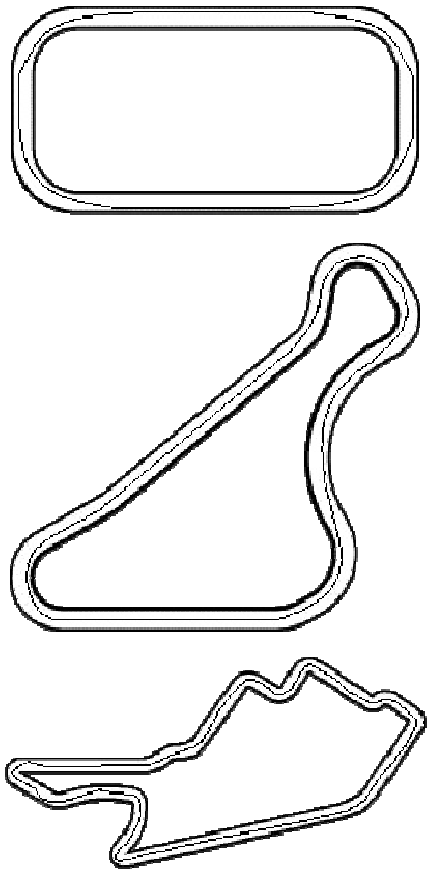

Figure 9. The path driven by the respective best controllers on the tracks used in this paper.

As the selection process we used is similar to the selection method used in the differential evolution (DE) algorithm, where a better performing child directly replaces its parent, we see a very slow but gradual fitness improvement. One very interesting question we consider is if another selection technique, like tournament selection, round robin, or hall-of-fame, would increase the speed of the fitness improvement, and perhaps help the controllers escape from local optima. We also like to find out how much coevolution would do to help the controllers improve.

Table 1 shows the shortest times achieved by the controllers for each track of all 10 runs. The SNN controllers evolved on track 1 achieved an average time of 25.8 seconds with a standard deviation of 0.134 seconds, and the fastest controller took 25.49 seconds to complete a lap. The controllers evolved on track 2 needed an average time of 47.12 seconds with a standard deviation of about 2 seconds. Its fastest controller took 44.71 seconds to complete a lap. The SNN controllers evolved on track 3, on the other hand, needed a staggering average time of 126.72 seconds, which is about two minutes, and a standard deviation of 1.69 seconds, and its fastest controller took 124.42 seconds to complete a lap. Furthermore, unlike the controllers evolved on tracks 1 and 2, the controllers evolved on track 3 managed to complete only one lap throughout the whole duration of 10000 game ticks. The fact, also, that the controllers on track 2 had a standard deviation of about 2 seconds might give a clear sign that the controllers may have not fully converge yet.

Though it is not presented in the table, but by examining the raw data of the lap times and fitness score, we found the controller that achieved the shortest time to complete a lap does not mean it obtained the highest fitness score. However, from the last generation of all runs, we found that the time for completing a lap is proportional to the fitness scores obtained by the controllers, and this is true for the fittest controller, as they achieved the shortest time and has the highest fitness score. Yet there are some cases where a controller achieved a shorter time than the time another controller achieved, but obtained a lower fitness score than the other controller, and vice versa.

Some explanation we could come up for this occurrence is in the deduction of the fitness scores. A controller might have achieved a shorter time to complete a lap, but it went out of the track's boundary and had its fitness score deducted. As a car comes closer to the inner edge of the curved path on the road, its distance to the finish line becomes shorter. The aforementioned controller could have drove outside the boundary of the track, but was nearer to the inner edge of the curved path on the road. Hence, its distance to the finish line was made shorter than the controller that kept its course within the boundary of the track.

We tried playing our SNN controllers against some TORCS included controllers and the hand-coded/rule-based controller, included with the TORCS CIG competition API. We also tried playing against the SNN controllers ourselves and witnessed some amusing discovery. Firstly, the evolved controllers were able to beat the rule-based controllers without much effort on all three tracks. Secondly, the controllers evolved on track 1 were able to beat both the TORCS included controllers and us. The controllers evolved on track 2, on the other hand, were able to beat us, but was not able to beat the TORCS included controllers. The controllers evolved on track 3 did not manage to overtake both the TORCS included controllers and us, but drove so much better than us, as we kept crashing onto the sides of the track.

Observations we made and noted from our play against the evolved controllers and the competition we set between them and the TORCS included controllers, is that the evolved controllers accelerates slower. The major cause of this, from

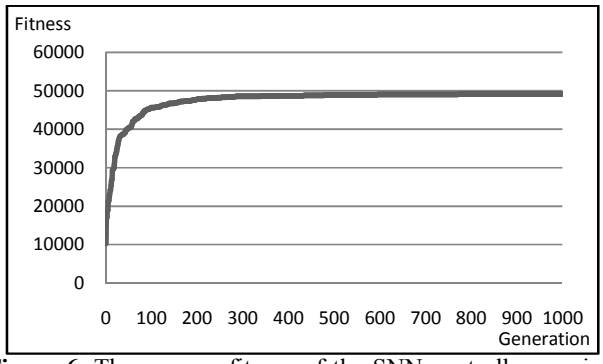

Figure 6. The average fitness of the SNN controllers against generation chart for track 1

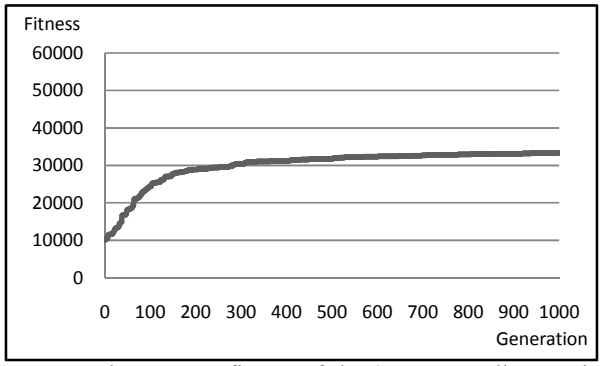

Figure 7. The average fitness of the SNN controllers against generations chart for track 2

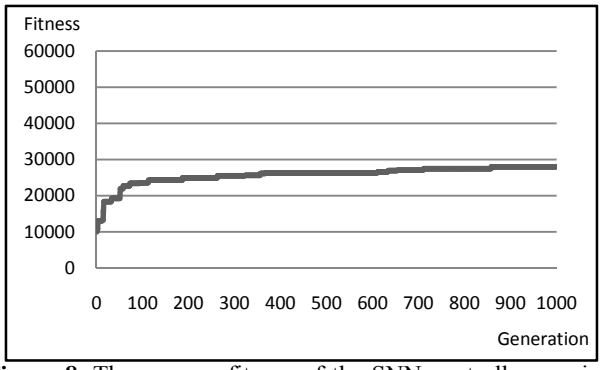

Figure 8. The average fitness of the SNN controllers against generations chart for track 3

our observation, is in the automated gear shifting strategy used in the controllers' car. Apparently, the car shifted its gear earlier than the TORCS controllers and the human controlled car, hence the reason for the slow buildup of speed. Furthermore, the SNN controller accelerates with values in the range of [0.0, 1.0], but the human controlled car and the TORCS controllers accelerates at maximum value (flooring the gas pedal). The SNN controllers from track 1 learned to floor the gas pedal throughout the race, but the SNN controllers from track 2 and 3 did not manage to learn it yet. Hence, the TORCS controllers have the upper hand to win because it picks up speed faster. This matter provides us with further interesting improvements that we could conduct on our part.

B. Sigmoid Neural Network

The experiment with the sigmoid neuron network on the other hand is used as an additional benchmark for the spiking neuron controllers, alongside other benchmarks as mentioned in Section 1. Based on the result charts, the sigmoid neural network exhibits an almost similar pattern to the charts of the SNN controllers. The sigmoid NN controllers evolved on track 1 has the highest fitness scores, where as the fitness scores for controllers evolved on track 3 are lowest.

Fig. 10 presents the average fitness growth of all the best sigmoid NN controllers evolved on track 1 for all 10 runs. It shows a sharp improvement in early generations and slowed down after about 100 generations to start converging. It finished the evolutionary run at the fitness of about 50,000. Much as the controllers evolved using the SNN, the sigmoid NN controllers show quick improvement and convergence. The observations made are quite similar to the SNN controllers, whereby they are able to display sophisticated driving behaviors, such as by increasing it distance between the car and the inner boundary of the track before turns to avoid skidding.

Fig. 11 shows the average results of all the best sigmoid NN controllers on track 2. The chart shows a slower improvement compared to the results obtained from the spiking NN controllers. Although both neural network controllers finished the evolution process between the fitness score of 30,000 and 35,000, the sigmoid NN controllers required more generations to gain fitness scores above 30,000. Still, similar to the results obtained from the SNN, the controllers may have not converged yet, since the slope (tangent line) of the graph at $900 < \text{generation } (x) < 1000$ is still positive.

The average results obtained from the sigmoid neural network controllers on track 3, as presented in Fig. 12, also gained the lowest scores among the three tracks. The sigmoid NN controllers experienced a slow but gradual improvement in performance throughout its evolutionary run, with earlier periods of about 200 generations being faster than after it has reached a fitness score of over 20,000. However, one significant difference between the two neural networks, which is most notably seen in the results obtained on track 3, is that the sigmoid NN only managed to obtain scores above 20,000, but below 25,000, where as the SNN managed to obtain scores above 25,000. Yet, based on its slope, the sigmoid NN controllers should still be able to improve if given more generations to evolve.

Table 2 details the fastest sigmoid NN controllers on all three tracks. The sigmoid NN controllers evolved on track 1 achieved an average time of 25.80 seconds with a standard deviation of 0.1701 seconds, while the fastest controller took 25.48 seconds to finish a lap. The sigmoid NN controllers evolved on track 2 took an average time of 49.43 with a standard deviation of 2.6417, which indicated to us that it could still improve. The fastest controller evolved on track 2 took 45.08 seconds to complete a lap. Sigmoid NN controllers evolved on track 3 needed an average time of 155.94 seconds with a standard deviation of 12.6369 seconds, whereas the fastest controller took 126.18 seconds to complete a lap.

Weighting the differences, based on Table 1 and 2 between the Spiking Neural Network and the Sigmoid Neural Network controllers, both neural network strategies performed quite similar to each other for track 1, except some small details, which might make the difference in a

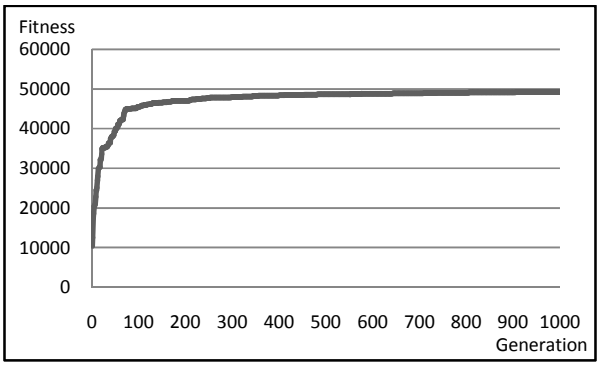

Figure 10. The Sigmoid NN controllers' average fitness against generation chart for track 1.

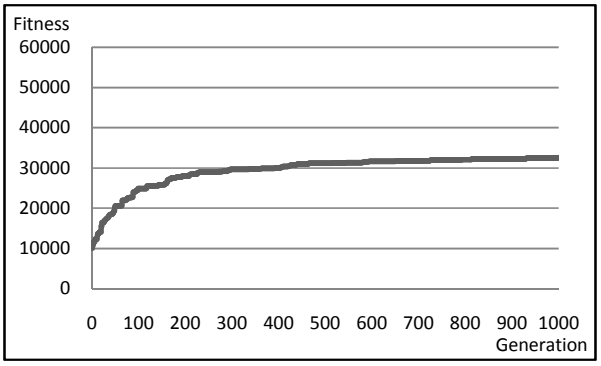

Figure 11. The Sigmoid NN controllers' average fitness against generation chart for track 2.

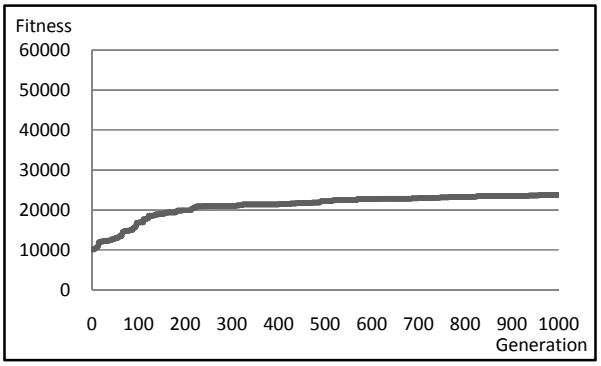

Figure 12. The Sigmoid NN controllers' average fitness against generation chart for track 3.

race. The fastest controller between both NN strategies comes from the Sigmoid NN strategy, with a time of 25.48 seconds. A bigger difference started to appear from track 2, most notably from the average time and standard deviation. Only one SNN controller took more than 50 seconds to complete a lap, but five, nearly six, sigmoid NN controllers took more than 50 seconds. The winner for track 2 comes from the SNN strategy, with 44.71 seconds. Track 3 has the largest difference, as seen from the standard deviation of both neural networks and the performance of the controllers. Only one sigmoid NN controller managed to achieve a time of 126.18 seconds, which intriguingly could beat five best SNN controllers from its respective runs, while other sigmoid NN controllers could not even achieve a time under 130 seconds.

One most possible and likely reason for this occurrence is the increased complexity of the sigmoid neural network, as it has more number of neurons compared to the spiking neural network. Hence, the likelihood of the sigmoid NN to need a longer period of evolution to achieve comparable lap times to the SNN. Some observations we noted from the simulation of the sigmoid NN is that it did not learn how to fully floor the gas pedal on tracks 2 and 3 yet. In track 1, both SNN and sigmoid NN controllers raced neck to neck on the track, and at some points knocking out the other or each other off the track. On track 2 on the other hand, we begin to see the controllers start to outplay each other more as the fastest sigmoid NN was able to win against five SNN controllers. Although based on Table 1 and 2, the fastest sigmoid NN controller should be able to defeat seven SNN controllers, but in the simulation, we observed that two slower SNN controllers knocked the fastest sigmoid NN controller off course rendering it to waste time recovering its path or impossible to continue the race.

The sigmoid NN that was used as a benchmark has a hidden layer of six hidden units, which may not be fair to the SNN that has no hidden layer. Hence, it may be more appropriate if the experiment with the sigmoid NN was conducted with no hidden layer as well. However, despite the differences between the two strategies, the SNN controllers were able to outrun the sigmoid NN controllers, as if without much effort. Although necessary experiments should be conducted for the sigmoid NN to find the best number of hidden units and/or layer, it should be conducted for the SNN too. Even though the SNN controllers may be underfitted, since having too few neurons in a hidden layer for a feedforward network will result in underfitting, which will result in the inadequacy to detect the signals in a complicated dataset [17], it still performed well. It drove with sophisticated behaviors and win against the rule-based controller on all tracks. It also outran the TORCS included controller on track 1 and human player on tracks 1 and 2. Furthermore, it performed better than the sigmoid neural network controllers did. It will be interesting to see how it will do with additional neurons and network recurrences.

Spiking neural networks present many challenges but also opens up many new opportunities for pioneering innovations in artificial intelligence research [18].

Table 2. Best Lap Times of Sigmoid Neural Network

Run	Best Lap (seconds)					
	Track 1		*Track 2*		*Track 3*	
1	25.74		45.85		163.78	
2	25.68		51.56		157.24	
3	25.89		50.27		171.68	
4	25.88		48.29		145.52	
5	26.03		47.59		164.35	
6	25.81		52.78		152.51	
7	25.65		45.08		157.65	
8	26.03		50.75		126.18	
9	25.79		52.24		162.47	
10	25.48		49.91		158.02	
μ / σ	25.80	0.1701	49.43	2.6417	155.94	12.6369

V. Conclusion and Future Works

This study has shown that cars or games for that matter, controlled by evolved spiking neuron models could perform well. The generated controllers were not only capable of driving through a complete racetrack without inflicting much or any damage on itself but could also demonstrate sophisticated driving behaviors. We have presented and noted many interesting reasons behind many occurrences during the experiment. Interestingly in some generations, the controllers decided to break rules and drove off-track so it could reduce its distance to the finish line. The results obtained from this experiment showed that the potential of using spiking neural networks in games, similar areas and more are immense. Our focus in this paper was to discover and show that spiking neuron models are capable of acting as well performing controllers in games, and we chose a racing game platform called TORCS, and interfaced our controllers through the TORCS CIG competition API.

There are many areas we can include and/or optimize for our future works, as the complexity of the game, evolution process and network structure are not limited to the methods we used for this paper. Future works may include a competitive coevolution optimization strategy, a self-adaptation method of the constant parameter values of a, b, c, and d, using an adaptive fitness function, and so forth. Perhaps employing incremental learning could introduce more generalization, and including multi-objective techniques could generate better solutions.

References

[1] S.J. Thorpe, A. Delorme, and R. Van Rullen, "Spike-based strategies for rapid processing," *Neural Networks*, vol. 14, pp. 715–726, 2001.

[2] Sander M. Bohte, "The Evidence for Neural Information Processing with Precise Spike-times: A Survey," *Natural Computing*, vol. 3, pp. 195–206, 2005.

[3] Wolfgang Maass, "Networks of Spiking Neurons: The Third Generation of Neural Network Models," *Neural Networks*, vol. 10, pp. 1659-1671, 1997.

[4] Wolfgang Maass, "Computation with spiking neurons," in *The Handbook of Brain Theory and Neural Networks*, 2nd ed.: MIT Press (Cambridge), 2003, pp. 1080-1083.

[5] Morgan Jakobsen, "Learning To Race In A Simulated Environment," Department of Information Technology, Østfold University College, Master's Thesis 2007.

[6] N.G. Pavlidis, O.K. Tasoulis, V.P. Plagianakos, G. Nikiforidis, and M.N. Vrahatis, "Spiking Neural Network Traning Using Evolutionary Algorithms," in *IEEE International Joint Conference on Neural Networks (IJCNN)*, Montreal, Que., 2005, pp. 2190-2194.

[7] Dario Floreano and Claudio Mattiussi, "Evolution of Spiking Neural Controllers for Autonomous Vision-Based Robots," in *LNCS 2217*. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, 2001, pp. 38–61.

[8] Dario Floreano, Jean-Christophe Zufferey, and Jean-Daniel Nicoud, "From Wheels to Wings with Evolutionary Spiking Circuits," *Artificial Life*, vol. 11, no. 1-2, pp. 121-138, January 2005.

[9] J. Togelius and S.M. Lucas, "Evolving controllers for simulated car racing," in *The 2005 IEEE Congress on Evolutionary Computation*, 2005, pp. 1906-1913.

[10] J. Togelius, P. Burrow, and S.M. Lucas, "Multi-population competitive co-evolution of car racing controllers," in *Evolutionary Computation, 2007. CEC 2007. IEEE Congress*, Singapore , 2007, pp. 4043-4050.

[11] Jorge Muñoz, German Gutierrez, and Araceli Sanchis, "A human-like TORCS controller for the Simulated Car Racing Championship," in *Proceedings 2010 IEEE Conference on Computational Intelligence and Games*, Copenhagen, Denmark, 2010, pp. 473-480.

[12] Elias Yee and Jason Teo, "Evolutionary Spiking Neural Networks As Racing Car Controllers," in *Hybrid Intelligent Systems (HIS), 2011 11th International Conference on*, Melacca, 2011, pp. 411-416.

[13] Eugene M. Izhikevich, "Simple Model of Spiking Neurons," *IEEE Trans. Neural Networks*, vol. 14, no. 6, pp. 1569-1572, 2003.

[14] Eugene M. Izhikevich, "Polychronization: Computation with Spikes," *Neural Computation*, vol. 18, no. 2, pp. 245-282, 2006.

[15] E. M. Izhikevich, *Dynamical systems in neuroscience: The geometry of excitability*. Cambridge, MA: The MIT Press, 2006.

[16] D. Loiacono et al., "The WCCI 2008 Simulated Car Racing Competition," in *Proc. IEEE Symp. Comput. Intell. Games*, 2008, pp. 119-126.

[17] Amit Ganatra, Y P Kosta and Devyani Panchal Gaurang Panchal, "Behaviour Analysis of Multilayer Perceptrons with Multiple Hidden Neurons and Hidden Layers," *International Journal of Computer Theory and Engineering*, vol. 3, no. 2, pp. 332-337, April 2011.

[18] Peter Stratton and Janet Wiles, "Why Spiking Neurons," University of Queensland, Brisbane, Technical Report TS-2007001, 2007.

Author Biographies

Elias Yee is a current member of the Evolutionary Computing Laboratory at Universiti Malaysia Sabah. He received his Bachelor of Computer Science degree from Universiti Malaysia Sabah in 2010 majoring in software engineering.

Jason Teo is currently an associate professor in computer science and deputy dean of the School of Engineering and Information Technology at Universiti Malaysia Sabah. He is also currently heading the Evolutionary Computing Laboratory there. Jason received his Bachelor of Computer and Mathematical Sciences degree from the University of Western Australia in 1993 with majors in information technology and biochemistry and a minor in mathematics. He completed the Master of Information Technology degree with distinction from Charles Sturt University in Australia externally between 1995 and 2000, and completed his Doctor of Information Technology degree with the University of New South Wales at the Australian Defence Force Academy in 2003.