Theta Neuron Networks: Robustness to Noise in Embedded Applications

Sam McKennoch, Preethi Sundaradevan, and Linda G. Bushnell

Abstract - **In this paper, we train a one-layer Theta Neuron Network (TNN) to perform a Braitenberg obstacle avoidance algorithm on a Khepera robot. The Theta neuron model is more biologically plausible than the leaky integrate and fire model typically used in Spiking Neural Networks. Our motivation is to determine if the dynamical properties of the theta neuron model can be leveraged to increase the noise robustness in an embedded application. We compare Khepera obstacle avoidance results with traditional Artificial Neural Network and TNN implementations under different levels of sensor noise. As the noise increases, the performance of the TNN is the least affected. At high noise levels, the ANN and Braitenberg implementations calculate the incorrect turn direction 42% more often than the TNN and deviate from a straight path trajectory over 10 times as far. The results demonstrate that TNNs warrants further development for engineering applications.**

I. Introduction

As the field of neuroscience continues to develop in leaps and bounds, there is an increasingly large opportunity for the adaptation of neuroscience concepts into the engineering community. Spiking neural network (SNN) research is one area that has greatly benefited from this cross-pollination in the past. The initial development of Artificial Neural Networks (ANNs) was inspired by the way biological nervous systems, such as the brain, process information. Likewise SNNs, such as those used in the SpikeProp training algorithm [3], add greater biological realism by encoding the exact timings of spikes rather than just the rate at which they are received. By using individual spikes as the signals sent between neurons, SNNs are able to include spatio-temporal information in communication similar to that of real neurons. SNNs have been shown to be computationally superior to ANNs even in the presence of noise and imprecision [7]. The implementation of phenomena such as spike timing dependent plasticity (STDP) and temporal coding lend SNNs greater biological plausibility over ANNs.

The use of SNNs in controllers for embodied agents began only recently, but they have already been shown as superior to continuous-time recurrent neural networks (CTRNNs) [7]. These SNNs have been implemented using a variety of different approaches. Work on SNNs has used gradient-descent based methods to train SNNs under a variety of different conditions, including multiple output spikes [4] and faster training times [13], [14], [17]. Genetic Algorithms (GAs) have also been used to evolve the weights of the SNNs online for wall-following tasks in the presence of noise [9]. Another recent SNN learning algorithm is based on an STDP rule to modify the connection weights for obstacle avoidance in a recurrent SNN [15]. A recurrent SNN was also used to train a robot to imitate and generalize the behavior of a Khepera robot which was initially controlled by a Braitenberg-type algorithm [5]. A further approach to training SNNs is based on self-organization where the presence or absence of neural connections was changed without adjusting the value or weight of the connections [1].

The type of neuron model used with SpikeProp and most machine learning applications is the leaky integrate and fire (IF) model. This model is very powerful for its simplicity, but misses a lot of the interesting dynamics that occur with biological neurons [6], [10]. An incremental step into higher complexity is the Theta neuron model, which is canonical for neuron models with a Type I frequency response. Theta neurons are able to model observed neuron behaviors such as activity-dependent thresholding [11], a property which is exploited in this paper to improve noise robustness in a trained Theta Neuron Network (TNN).

The purpose of this paper is as follows: to provide confirmation of the results published in [16]; to demonstrate a widely-used embedded application with TNNs; and to demonstrate some of the properties of TNNs that differentiate them from ANNs or other types of SNNs. More specifically, this paper applies well-known noise properties of sensors to an embedded obstacle avoidance application of TNNs. Noise robustness has long been one metric by which machine learning methods are judged [2], [12]. In our case, noise is used to demonstrate the different input-output sensitivities of ANNs vs. TNNs. While the sensitivity for the trained ANN and the Braitenberg algorithm is typically constant, the sensitivity for the TNN varies with the sizes of the sensor inputs.

This paper is organized as follows. Section II provides a background on theta neurons. Section III details the experimental setup of the algorithm implementations. Section IV includes the results and analysis of the three implementations of the Braitenberg obstacle avoidance algorithm each with and without noise on the Khepera robot and in simulation. The properties of TNNs are examined relative to their ability to effectively filter out noise under the right conditions. Summary and future work follow in Section V.

II. Spiking Neuron Background

A. Leaky Integrate and Fire Model

The leaky IF model is a widely-used neuron model con-

Manuscript received January 31, 2007. This work was supported in part by the US National Science Foundation Grant No. ECS-0322618.

All authors are affiliated with the Electrical Engineering Dept., University of Washington, Seattle, WA 98195-2500 USA (Corresponding author: skennoch@u.washington.edu).

sisting of a leaky capacitor whose potential simulates the membrane potential of a spiking neuron [7]. The leaky capacitor is modeled by a capacitor, *C*, in parallel with a resistor, *R*, and is driven by a current, *I(t):*

$$RC\frac{du}{dt} = -u(t) + RI(t) \tag{1}$$

where *u(t)* describes the membrane potential. As soon as the membrane potential crosses an activation threshold value, ϑ, it is reset to a baseline value of u_r for an interval called the refractory period during which the ability of the neuron to fire again is restrained. Spikes are able to propagate from layer to layer and eventually result in a time series of spikes at the output layer. One generalization of the leaky IF neuron is the Spike Response Model (SRM), used in SpikeProp (a gradient-descent based SNN training method [3]).

B. Theta Neuron Model

Although the leaky IF model is the most popular model in engineering applications of SNNs, it misses many interesting neuron dynamics such as spike latencies and activity-dependent thresholding. A plethora of models exist, which with increasing complexity model observed neuron dynamics [6], [11]. These models are largely unexplored in the engineering machine learning domain.

Neuron models can be categorized by their frequency response. As input current into Type I neuron is increased, at some point the frequency of output spikes smoothly begins to increase upward from 0. For Type II neurons, due to a Hopf bifurcation, the frequency of output spikes jumps from 0 to some non-zero value. The leaky IF model is Type I.

A modification of the leaky IF model which contains more realistic neuron dynamics is the quadratic neuron model:

$$\tau\frac{du}{dt} = a_o(u - u_{rest})(u - u_c) + RI \tag{2}$$

where u_{rest} is the resting potential and u_c is the critical voltage, which if passed means that the neuron will soon fire. A canonical model to which any Type I neuron model described by smooth ODEs, including the quadratic neuron [8], can be mapped to is:

$$\frac{\partial\theta}{\partial t} = (1 - \cos\theta) + \alpha I(t)(1 + \cos\theta) \tag{3}$$

where α is a scaling constant; θ is the neuron phase; and *I(t)* is the input current that drives the dynamics. The neuron phase is directly mappable to the membrane potential. This equation describes the trajectory of neuron phase. Neurons described by this model are called theta neurons. Canonical models do not generally exist for Type II neurons due in part to the wider range of behaviors exhibited by Type II neurons.

The input current is defined as the sum of a baseline current plus a series of current impulses at times t_j with size modulated by weights w_j:

$$I(t) = I_o + \sum_{j=1}^{J} w_j\delta(t - t_j) \tag{4}$$

When the baseline current, I_o, is less than zero, two fixed points can be found by setting (3) equal to zero. One fixed point is a saddle point, and the other is a stable fixed point. The neuron phase moves around its phase circle toward the stable fixed point and away from the saddle point. When the phase passes through π, the neuron is said to have fired. The phase at neuron *j* right before an input spike from neuron *n* is received is defined as θ^-_{nj}. Because input spikes are modelled as impulses, the phase at neuron *j* right after an input spike from neuron *n* is

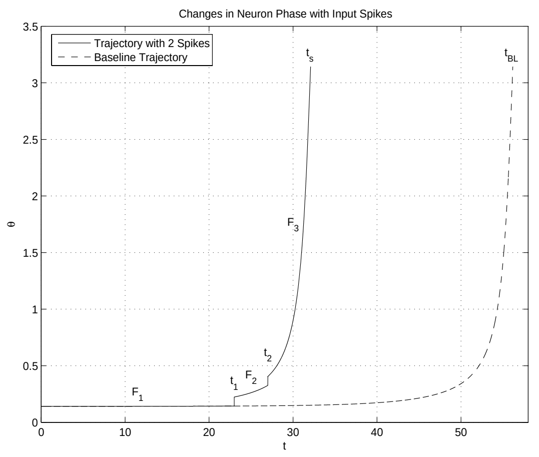


Fig. 1. The Remaining Time Function
The neuron phase, θ, is shown over time given an initial phase just greater than the saddle point. Two input spikes occur at times 23 and 27. Given that the input spikes at t_1 and t_2 have equal magnitudes, the effect of the spike at t_2 in changing the output firing time is less than that of the input spike at t_1.

$$\theta^+_{nj} = \theta^-_{nj} + \alpha w_{nj}(1 + \cos\theta^-_{nj}) \tag{5}$$

which follows from equations (3) and (4). As in [16], the remaining time until the next output spike as a function of the neuron phase is:

$$F(t) = \int_{\theta(t)}^{\pi} \frac{d\theta}{(1 - \cos\theta) + \alpha I_o(1 + \cos\theta)} = F\Big|_{\theta(t)}^{\pi} \tag{6}$$

If there are input spikes, *F* is discontinuous and is broken up into continuous pieces, which is demonstrated graphically in Figure 1:

$$F(t_i) = \sum_{\substack{j \ge i \\ j < N}} \int_{\theta^+_j}^{\theta^-_{j+1}} \frac{d\theta}{(1 - \cos\theta) + \alpha I_o(1 + \cos\theta)} = \sum_{j \ge i} F\Big|_{\theta^+_j}^{\theta^-_{j+1}} \tag{7}$$

where *N* is the number of inputs to the neuron and $\theta^-_N = \pi$.

Activity-dependent thresholding is one property that distinguishes the theta neuron model from the leaky IF model. In the leaky IF model, an input spike of a given magnitude will change the output firing time by an equal amount regardless of the input spike timing. However, in the theta neuron model, a spike's effect on the phase is determined by the magnitude and timings of input spikes already received.

C. Output Layer Weight Training

A gradient based method for training the output firing times of a single layer of theta neurons has recently been developed [16] and is summarized here. Error is defined as a Sum of Squared Errors (SSE) in output spike times for all output neurons, O.

$$E = \frac{1}{2}\sum_{o_j \in O}(t_{o_j} - t_{o_j}^d)^2 \tag{8}$$

The basic weight learning rule for the weight between input n and neuron p is:

$$\Delta w_{np} = -\eta\frac{\partial E}{\partial w_{np}} = -\eta\frac{\partial E}{\partial t_{o_p}}\frac{\partial t_{o_p}}{\partial F}\frac{\partial F}{\partial w_{np}} \tag{9}$$

where η is the learning rate. The first gradient term follows from (8) and the second term is unity. For the third term:

$$\frac{\partial F}{\partial w_{np}} = \frac{\partial F}{\partial \theta_{np}^+}\frac{\partial \theta_{np}^+}{\partial w_{np}} + \sum_{j>n}\left(\frac{\partial F}{\partial \theta_{jp}^+}\frac{\partial \theta_{jp}^+}{\partial w_{jp}} + \frac{\partial F}{\partial \theta_{jp}^-}\frac{\partial \theta_{jp}^-}{\partial w_{np}}\right) \tag{10}$$

In [16], the author ignores the summation part of this term since there is no a-priori information about spikes that will occur after the current spike being analyzed. Thus:

$$\frac{\partial F}{\partial w_{np}} \approx \frac{\partial F}{\partial \theta_{np}^+}\frac{\partial \theta_{np}^+}{\partial w_{np}} \tag{11}$$

The first term in (11) follows from (6) and the second follows from (5). Putting it all together:

$$\Delta w_{np} = -\eta\delta_p y_{np} \tag{12}$$

where

$$\delta_p = (t_{o_p} - t_{o_p}^d) \tag{13}$$

and

$$y_{np} = \frac{-\alpha(1+\cos\theta_{np}^-)}{(1-\cos\theta_{np}^+) + \alpha I_o(1+\cos\theta_{np}^+)} \tag{14}$$

As is the case with the basic back-prop algorithm, y is calculated by moving forward through the network, δ is calculated by propagating the errors back through the network. δ is specific to a neuron, regardless of which input is being examined. For our system the interaction between the transfer function and input timings do not allow y to be separated out, and so like the weights, y is specifically a measure of the effect of an output spike from neuron n to neuron p.

III. Experimental Setup

A. Implementation of Obstacle Avoidance Algorithm

In this paper, three different implementations of the Braitenberg obstacle avoidance algorithm on a Khepera II robot are investigated. The Khepera II is a popular mobile robot which contains eight infrared sensors for distance measurements whose values fall in the range 0 to 1023. In our experiments, six of the robot's sensors (ignoring the two on the back side) are used as inputs to control the robot's two wheel motors.

The first implementation is a direct programming of the Braitenberg algorithm, which is simply a scaled linear com-

TABLE 1: Neural Network Training Regions

Region	Sensor Values	Obstacle Location
0	<100 <100 <100 <100 <100 <100	None Within Sight
1	<100 <100 <100 >900 >900 >900	Right-Side
2	<900 <900 <900 >100 >100 >100	Left-Side
3	<100 <100 >900 >900 <100 <100	Directly Ahead

bination of the input sensor values. Specifically:

$$\vec{m} = \frac{\vec{c}\cdot\vec{s}}{KSCALING} + KSPEED \tag{15}$$

where $\vec{m}$ is an output motor vector; $\vec{s}$ is an input sensor vector; *KSCALING* (50 in our case) linearly controls the sensitivity of motor values to sensor values; *KSPEED* (5 in our case) determines the baseline speed; and $\vec{c}$ is a coefficient matrix that determines the bias for each input sensor. In this paper, a simple matrix was used to help clarify the results:

$$\vec{c} = \begin{bmatrix} -1 & -1 & -1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 & -1 & -1 \end{bmatrix} \tag{16}$$

The second and third implementation are types of neural networks, and therefore require a training set for supervised learning. The networks were trained on data that fell into four different regions as shown in Table 1. These cases are the primary situations that the robot will encounter when avoiding obstacles.

In the second implementation a standard rate-encoded ANN was used. The network has six inputs for the six sensors, two outputs for the two motors, and eight hidden neurons. The squashing functions are sigmoidal and linear for the hidden and output layers respectively. The training error tolerance is 1 (0.81% of the motor value range), trained using QuickProp.

The third implementation is a single layer composed of two theta neurons corresponding to the two motor outputs. Each neuron has seven input weights for the six sensors and a reference input. Sensor readings must be encoded into input spike times and output spike times must be decoded into motor values. The sensor readings are encoded into input times so that high sensor values correspond to smaller input times:

$$t_i = 5 - \frac{4s_i}{1023} \tag{17}$$

Thus input sensor values of 0 to 1023 correspond to input times of 5 to 1. In the theta neuron, the sensitivity of the output spike time decreases dramatically as the phase increases above the saddle point. The simple linear encoding allows the theta neuron to be pre-programmed to have a lower sensitivity to low sensor values in order to deal with the expected noise conditions. The reference time, t_r is equal to 1.

For the decoding of the theta neuron output spike times, the range of expected motor values can be calculated from the Braitenberg algorithm. The output spike times are trained to fall within the middle range of possible output spike times as determined by network constants like I_o. Table 2 contains a full network constants list. Initial theta offset is the offset of

TABLE 2: Theta Neuron Network Parameters

Parameter	Value
Alpha	1
Baseline Current	-0.005
Initial Theta Offset	0.0001
Simulation Timestep	0.001
Learning Rate	0.000015

the initial neuron phase above the saddle point. Thus the formula for decoding output spike times into motor values is:

$$m_i = \left(\frac{t_{s_i} - t_{s_{min}}}{t_{s_{max}} - t_{s_{min}}}\right)(m_{max} - m_{min}) + m_{min} = \left(\frac{t_{s_i} - 25}{17}\right)(122.76) - 56.38 \quad (18)$$

The TNN was trained with the method described in Section II.C. Because only a single layer of theta neurons is being used, there are limitations on what types of functions are trainable. For example, consider input vectors that fall in regions 0 and 3 from Table 1. In both cases, the Braitenberg algorithm dictates that the Khepera continue to move forward at *KSPEED*. In reality, the readings of one side of sensors is usually slightly different than the other side, causing the robot to have a slight direction bias. Regardless, a single theta neuron cannot produce the same output for both of these cases. In region 0, a series of late input spikes will have a reduced effect on the output spike time, thus the output spike time will mostly be determined by the reference time spike. In region 3, the two high sensors readings will result in earlier input spikes times, and thus earlier output spike times and higher motor speeds. The TNN, lacking a hidden layer, is therefore overconstrained. However, it is still possible to train the TNN so that it turns in the correct direction, even if the speed of movement is not always accurate. The overall goal of avoiding obstacles is much more dependent on turning in the right direction than it is on speed. So training to a very low SSE was not possible. Instead a stopping criteria based on the testing and training SSE was used as it often used in ANNs to prevent memorization over generalization.

B. Sensitivity to Noise

In this obstacle avoidance experiment, low sensor readings are injected with more noise than higher readings. There are many cases of sensors which have increased sensitivities to low inputs, and therefore more noise sensitivity at these low inputs (ChemFETs for example). Alternatively, the environment can also create more noise in low sensor readings, such as when there is a large amount of air particulates.

Sensitivity to sensor noise has a large impact on obstacle avoidance. For the Braitenberg implementation the differentiation is straightforward and results in a constant sensitivity:

$$\frac{\partial m_i}{\partial s_j} = \frac{c_{ij}}{KSCALING} = 0.02 \quad (19)$$

For the case of the ANN, it is assumed that each of the hidden nodes is not in saturation, but rather is in the approximately linear region of the sigmoid since the Braitenberg algorithm used for training is linear. Also the slopes of the linear activation functions are denoted by k_i.

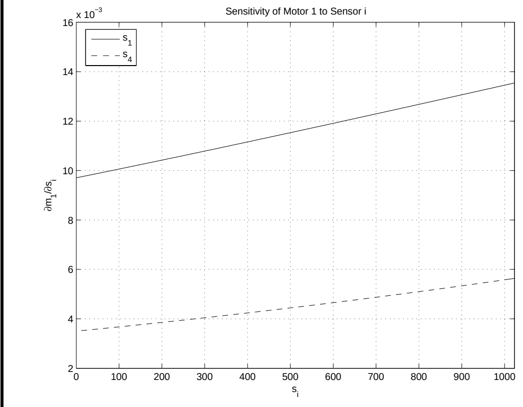


Fig. 2. Sensitivity in Theta Neurons
The TNN is about 30% less sensitive to low input sensor values than high input sensor values as shown in this figure. For the case of s_4, it is assumed that s_1 to s_3 were high sensor value readings. Due to the activity-dependent threshold property of the model, the overall sensitivity is decreased further in the s_4 case.

$$\frac{\partial m_i}{\partial s_j} = \sum_{n=1}^{N} \frac{\partial m_i}{\partial h_n}\frac{\partial h_n}{\partial s_j} \approx k_i \sum_{n=1}^{N} w_{s_{jn}} w_{h_{ni}} \quad (20)$$

where N is the number of hidden layer neurons; w_s are weights between input and hidden layer nodes; and w_h are weights between hidden and output layer nodes. Sensitivity of an unsaturated ANN node is approximately constant.

For the TNN case, encoding and decoding must be taken into account. From the results in Section II:

$$\frac{\partial m_i}{\partial s_j} = \frac{\partial m_i}{\partial t_{s_i}}\frac{\partial t_{s_i}}{\partial F}\left(\frac{\partial F}{\partial \theta_j^+}\frac{\partial \theta_j^+}{\partial \theta_j^-} + \frac{\partial F}{\partial \theta_j^-}\right)\frac{\partial \theta_j^-}{\partial t_j}\frac{\partial t_j}{\partial s_j} \quad (21)$$

Expanding this equation:

$$\frac{\partial m_i}{\partial s_j} = 0.0282\left(1 - \frac{(1-\cos\theta_j^-) + \alpha I_o(1+\cos\theta_j^-)}{(1-\cos\theta_j^+) + \alpha I_o(1+\cos\theta_j^+)}(1 - \alpha w_{ji}\sin\theta_j^-)\right) \quad (22)$$

Figure 2 shows how this sensitivity changes as a function of input sensor readings. This dynamic sensitivity range is a direct result of activity-dependent thresholding. Spikes received when the neuron phase is close to π, corresponding to low sensors readings, have a reduced effect on output spike times and therefore the motor values as well.

IV. Results and Analysis of Khepera Experiment

In combining an obstacle avoidance experiment with noise sensitivity investigations, it is helpful to separate sensitivity to noise from the effect of noise on performance in avoiding obstacles. Three experiments were performed in addition to simulations.

Figure 3 shows the simulation results of how often noise causes the robot to turn in the incorrect direction for each of three implementations across two of the regions in Table 1 and for four different noise levels. Noise level 0 indicates that no noise is added to the sensor readings. For noise greater than zero:

$$s = s + \left(\frac{k^c}{s^c + 1}\right) N(0, 1) \qquad (23)$$

where c is the noise level; k is a constant equal to about 92; and s is the sensor reading to which noise is added. This method of adding noise creates more noise for smaller sensor values. Figure 4 shows directly how adding sensor noise has changed the motor value.

For the first Khepera experiment, the robot is set to follow a straight path of length 15 cm and its deviation from the path is measured. Each implementation, at different noise levels, was run 5 to 10 times and outlying data was removed. The average distance from the straight path as well as its standard deviation increases with more noise. This increase is due to the sudden turns made by the robot in response to the false alarms raised by the noisy readings. While the ANN and Braitenberg implementations deviated from the path considerably, the TNN was able to suppress the effect of noise and stay nearly straight, deviating less than a tenth of the distance of that of the other two implementation; see Figure 5.

In the second experiment, the robot is set 5 to 10 cm away from obstacles to approach them at specific angles and the closest distance is measured. Table 3 shows the complete results. At 90 degrees, the TNN is not able to avoid the obstacles, regardless of noise, because as discussed in Section III, the TNN training is overconstrained. As the robot approaches an obstacle that is directly ahead, its speed decreases to nearly a dead stop, preventing it from turning quickly. Another interesting feature is that the Braitenberg implementation was able to avoid obstacles at 90 degrees better in the presence of noise. As can be seen in (15), if the sensor values are perfectly balanced, the robot will move straight into an obstacle. Noise disrupts this balance. But in general, the robot gets closer to the obstacle in the presence of noise for the ANN and Braitenberg than for the TNN. For example, at 30 degrees, the ANN's performance decreased 38% due to noise while the change was only 11% for the TNN, demonstrating the TNN's more consistent, steady behavior.

The third experiment involved letting the robot wander around in an area containing many obstacles and then counting the number of obstacles touched. The results were surprising in that even though the ANN and Braitenberg implementations were affected by noise, their performance in avoiding obstacles stayed the same, except in the case described in the previous paragraph, where the performance improved. The TNN behavior in the presence of high noise did not change. It continued to get stuck when the angle of approach was close to 90 degrees, and avoided all other obstacles.

V. Conclusion and Future Work

This paper has provided confirmation of the TNN training method published in [16], demonstrated obstacle avoidance on a Khepera robot using TNNs, and demonstrated how the activity dependent threshold property of TNNs, which is not present in ANNs and leaky IF SNNs, can be useful for noise robustness. The effect of noise on the behavior of the TNN has been shown to be small. At high noise levels, the ANN and Braitenberg implementations deviate from a straight path trajectory over 10 times as far as the TNN implementation. However, we have also shown that insensitivity to noise does not necessarily improve obstacle avoidance performance. Sometimes the effect of noise, which was felt greater by the ANN and Braitenberg implementations, is beneficial in breaking symmetry. Other types of experiments, such as line

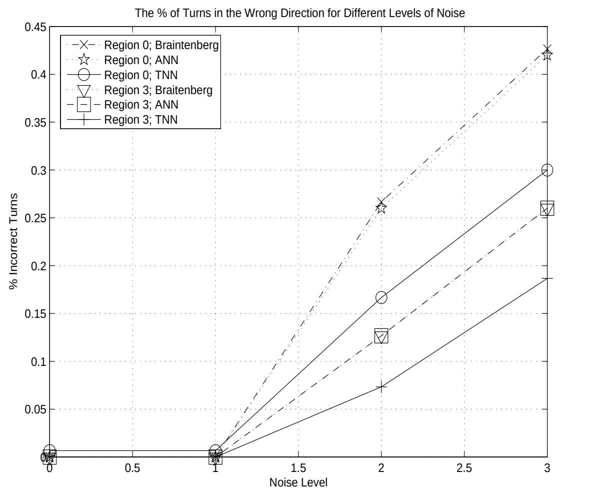


Fig. 3. Noise vs % Correct Turning Direction in Simulation
In both region 0 and 3, as noise increase the performance of the TNN implementation improves relative to the other two implementations. For regions 1 and 2 there were no cases of wrong turns.

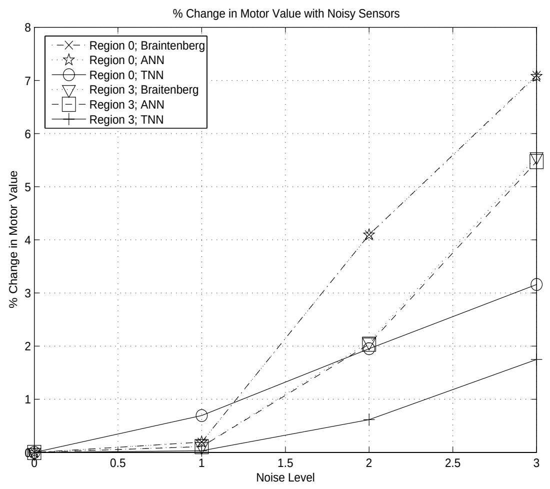


Fig. 4. Noise vs % Change in Motor Values in Simulation
As the sensor noise level increases, the Braitenberg and ANN implementations experience a greater % change in their motor values relative to the TNN.

TABLE 3: Experimental Minimum Distance from Obstacle for Different Angles of Arrival

	Without Noise (c=0)			With High Noise (c=3)		
Angle	Braitenberg	ANN	TNN	Braitenberg	ANN	TNN
30	3.13	4.57	1.88	2.90	2.83	1.67
60	2.13	3.21	1.00	2.20	1.88	0.94
90	0.50	1.71	0.00	1.10	0.55	0.00

following might better demonstrate increased performance through reduced noise sensitivity.

Future work includes investigating multi-layer TNNs in order to make TNNs which are universal function approximators and to be able to investigate more complicated embedded applications, which further take advantage of TNN properties.

ACKNOWLEDGEMENTS

The authors thank Thomas Voegtlin for many helpful discussions.

REFERENCES

[1] Alnajjar, F. and Murase, K., "Self-Organization of Spiking Neural Network Generating Autonomous Behavior in a Real Mobile Robot," *International Conference on Computational Intelligence for Modelling, Control and Automation*, 2005.

[2] Beck, S. and Ghosh, J., "Noise sensitivity of static neural network classifiers," *Proceedings of SPIE Science of Artificial Neural Networks*, Vol. 1709, April 1992.

[3] Bohte, S., "Spiking Neural Networks," Ph.D. dissertation, Centre for Mathematics and Computer Science (CWI), Amsterdam, 2003.

[4] Booij, O., and Nguyen, H., "A gradient descent rule for spiking neurons emitting multiple spikes," *Information Processing Letters*, Vol. 95, No. 6, 30 September 2005, pps. 552-558.

[5] Burgsteiner, H., "Training networks of biological realistic spiking neurons for real-time robot control," *Proceedings of the 9th International Conference EANN*, 2005.

[6] Feng, J. F., "Is the integrate-and-fire model good enough? - a review," *Neural Networks*, Vol 14., pps. 955-975.

[7] Florian, R. V., "Biologically inspired neural networks for the control of embodied agents," Technical Report Coneural-03-03, 2003.

[8] Gerstner, W., and Kistler, W., *Spiking Neuron Models*. Cambridge, UK: Cambridge University Press, 2002.

[9] Hagras, H.A.K., Pounds-Cornish, A., Colley, M.J., Callaghan, V., and Clarke, G., "Evolving Spiking Neural Network Controllers for Autonomous Robots," *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, New Orleans, 2004.

[10] Izhikevich, E. M., "Neural Excitability, Spiking, and Bursting," *International Journal of Bifurcation and Chaos*, Vol. 10, 2000, pps. 1171 - 1266.

[11] Izhikevich, E. M., "Which Model to Use for Cortical Spiking Neurons?," *IEEE Transactions on Neural Networks*, Vol. 15, 2004, pps. 1063-1070.

[12] Kalapanidas, E., Avouris, N., Craciun M., and Neagu, D., "Machine Learning Algorithms: A study on noise sensitivity," *Proceedings of the 1st Balcan Conference in Informatics*, Thessaloniki, November 2003, pp. 356-365.

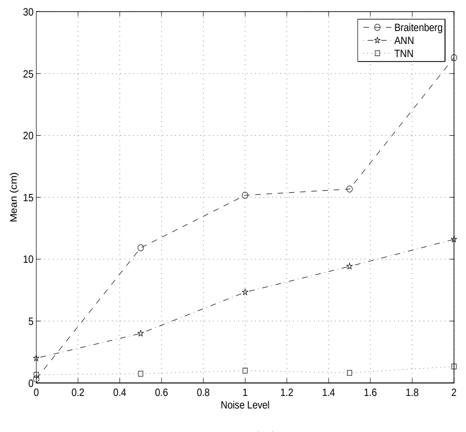
[13] McKennoch, S., Liu, D., and Bushnell, L.G., "Fast Modifications of the SpikeProp Algorithm," *IEEE World Congress on Computational Intelligence (WCCI)*, Vancouver, BC, July 2006.

[14] Schrauwen, B. and Van Campenhout, J., "Extending SpikeProp," *Proceedings of the International Joint Conference on Neural Networks*, 2004. pp. 471-476.

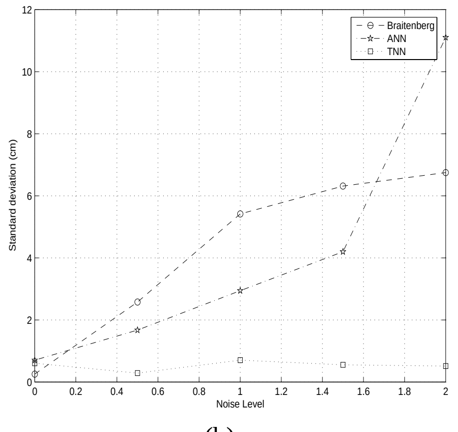
[15] Soula, H., Alwan, A. and Beslon, G., "Learning at the edge of chaos: Temporal coupling of spiking neuron controller of autonomous robotic," *Proceedings of AAAI Spring Symposia on Developmental Robotic*, 2005.

[16] Voegtlin, T., "Temporal Coding using the Response Properties of Spiking Neurons," *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, Vol. 19, 2006.

[17] Xin, J., & Embrechts, M., "Supervised learning with spiking neural networks," *Proceedings of International Joint Conference on Neural Networks (IJCNN)*, 2001, pp. 1772-1777.



(a)



(b)

Fig. 5. Experimental Straight Line Path Deviation for Different Noise Levels
Both the mean (a) and the standard deviation (b) of the ANN and Braitenberg implementations increase dramatically as more noise is introduced. However, the TNN behavior is nearly unchanged.