Path Planning using a Spiking Neuron Algorithm with Axonal Delays

Jeffrey L. Krichmar
Department of Cognitive Sciences
Department of Computer Science
University of California, Irvine
Irvine, CA, USA, 92697-5100
jkrichma@uci.edu

***Abstract*—A path planning algorithm is introduced that uses the timing of spiking neurons to create efficient routes. The algorithm is inspired by recent evidence showing activity-dependent plasticity of axon myelination after learning. Using this finding as inspiration, the algorithm's learning rule varies the simulated axon conductance velocity between neurons based on the relative cost of traversing the environment. In terms of path length and path cost, the spiking algorithm is as good or better than other path planners. However, the present spiking algorithm has the added advantage of adapting to change and context by altering axon delays in response to environmental experience. Because the spiking algorithm is suitable for implementation on neuromorphic hardware, it has the potential of realizing orders of magnitude gains in power efficiency and computational gains through parallelization, and thus should offer advantages for small, embedded systems.**

Keywords—neuromorphic chips; path planning; robotics; spiking neurons

I. Introduction

Path planning involves calculating an efficient route from a starting location to a goal, while avoiding obstacles and other impediments. Despite much advancement over several decades of robotic research, there are still many open issues for path planners [1, 2]. For example, most path planners are not designed to handle a dynamic environment and changing context and are computationally expensive.

Classic path planning algorithms include Dijkstra's algorithm, A*, and D*. Dijkstra's algorithm uses a cost function from the starting point to the desired goal. A* additionally considers the distance from the start to the goal "as the crow flies" [3]. D* extends the A* algorithm by starting from the goal and working towards the start positions. It has the ability to re-adjust costs, allowing it to replan paths in the face of obstacles [4]. These algorithms can be computationally expensive when the search space is large. Rapidly-Exploring Random Trees (RRT) are a less expensive approach because they can quickly explore a search space with an iterative function [5]. Still, these algorithms are not flexible in dynamic environments or in situations where the context changes. Algorithms have been introduced, which use a cost function and optimization techniques to adjust to changing environmental conditions [6]. Still these path planners are computationally expensive and may not be appropriate for small, embedded systems, such as micro-aerial vehicles.

Because of their event driven design and parallel architecture, neuromorphic hardware hold the promise of lowering size, weight and power, and may be ideal for embedded applications [7]. These systems are modeled after the brain's architecture and typically use spiking neural elements for computation [8]. Spiking neurons are event driven and use an Address Event Representation (AER), which holds the neuron ID and the spike time, for communicating between neurons. Since spiking neurons do not fire often and post-synaptic neurons do not need to calculate information between receiving spikes, AER allows for efficient computation and communication.

One class of path planning that may be a good fit for neuromorphic applications is the wave front planner [6, 9] or diffusion algorithms [10]. In a standard wave front planner, the algorithm starts by assigning a small number value to the goal location. In the next step, the adjacent vertices (in a topological map) or the adjacent cells (in a grid map) are assigned the goal value plus one. The "wave" propagates by incrementing the values of subsequent adjacent map locations until the starting point is reached. Typically, the wave cannot propagate through obstacles. A near optimal path, in terms of distance and cost of traversal, can be read out by following the lowest values from starting location to the goal location. This method has been used in neurorobot applications [11, 12], and in neuromorphic hardware [13].

We present a spiking neuron wave front algorithm for path planning that adjusts to changes in the environment. Prior work showed that a spike-based wave planner could run on neuromorphic hardware [13]. The present algorithm builds on this prior work by adding an adaptive element that considers the relative cost of traversing through different parts of the environment. The adaptive element is inspired by recent empirical findings supporting experience dependent plasticity of axonal conduction velocities [14]. Specifically, the cost of traversing through space is represented in the axon delay between neurons. In this way, the lowest cost path is reflected in the spike timing, that is, the spike propagation will travel faster over lower cost paths.

Supported by the National Science Foundation Award #1302125 and Northrop Grumman Aerospace Systems.

978-1-5090-0623-6/16/$31.00 ©2016 IEEE

II. Methods

A. Neuron Model and Connectivity

To show how a spiking neuronal wave could be used in a path planning algorithm, we constructed a simple spiking neuron. The present neuron model contained a membrane potential (v), a recovery variable (u), and received current input (I) from synaptically connected neurons:

$$v_i(t+1) = u_i(t) + I_i(t) \quad (1)$$

$$u_i(t+1) = min(u_i(t) + 1, 0) \quad (2)$$

$$I_i(t+1) = \sum_j (1 \text{ if } d_{ij}(t) = 1; 0 \text{ otherwise}) \quad (3)$$

$$d_{ij}(t+1) = max(d_{ij}(t) - 1, 0) \quad (4)$$

$d_{ij}(t)$ is the axonal delay between when neuron $v_j(t)$ fires an action potential and $v_i(t)$ receives the action potential. When $v_j(t)$ fires an action potential, $d_{ij}(t)$ is set to the delay value of $D_{i,j}(t)$, which is described in section II.B and Eq. (5). Note from Eq. (4) that d_{ij} has a null value of zero unless the pre-synaptic neuron fires an action potential.

Equations (1-4) calculate the membrane potential, recovery variable, synaptic input, and axonal delay for neuron i at time step t, which is connected to j pre-synaptic neurons. The neuron spiked when v in Eq. (1) was greater than zero, in which case v was set to 1 to simulate a spike, u was set to minus 5 to simulate a refractory period, and the axonal delay buffer, d, was set to D. The recovery variable, u, changed each time step per Eq. (2). The delay buffer, d, changed each time step per Eq. (4). I in Eq. (3) was the summation of the j pre-synaptic neurons that delivered a spike to post-synaptic neuron i at time t.

The neural network consisted of a 64x64 grid of spiking neurons as described in Eqs. (1-4). Each neuron corresponded to a location in the environment and was connected to its eight neighbors (i.e., N, NE, E, SE, S, SW, W, NW). At initialization ($t = 0$), v and u were set to 0. All delays, D, were initially set to 5, but could vary depending on experience in the environment. D represented the time it takes to propagate a pre-synaptic spike to its post-synaptic target.

B. Axonal Delays and Plasticity

A spike wave front proceeds by triggering a single spike at a neuron that corresponds to the start location. This neuron then sends a spike to its synaptically connected neighbors. The delivery of the spike to its post-synaptic targets depends on its current axonal delay. Each synapse has a delay buffer, which governs the speed of the spike wave.

$$D_{i,j}(t+1)=D_{i,j}(t) + \delta(map_{x,y} - D_{i,j}(t)) \quad (5)$$

Where the delay $D_{i,j}(t)$ represents the axonal delay at time t between neurons i and j, $map_{x,y}$ is the value of the environment at location (x,y), and δ is the learning rate. For the present experiments, δ was set to 1.0, which allows the system to instantaneously learn the values of locations. The learning is expressed through axonal delays. For example, if the spike wave agent encountered a major obstacle, with a high traversal cost (e.g., 25), the neuron at that location would schedule its spike to be delivered to it's connected neurons 25 time steps later, whereas, if the value of a location was 1, the spike would be delivered on the next time step. It should be noted that in the present study the delay buffers were reset before each route traversal. The effect of different learning rates and building maps with experience is discussed in Section IV.

C. Map of Environment

The environment consisted of a 64x64 grid (see Fig. 1). Regions of the environment had cost values that included roads (value = 1), minor obstacles (value = 5), major obstacles (value = 25), and open spaces (value = 3). The agent could move in eight directions from any position on the map.

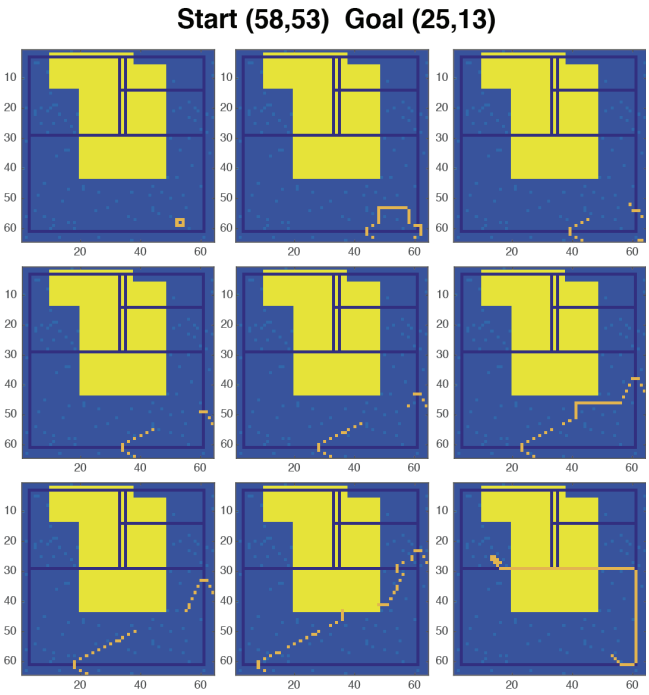

Fig. 1. Spike wave front algorithm in a typical environment. The environment is a 64x64 grid. The colors represent different objects in the environment. Dark blue lines represent "roads" and have a cost of 1. Medium blue dots represent minor obstacles and have a cost of 5. Large yellow regions represent major obstacles and have a cost of 25. All other regions (light blue regions) represent open space and have a cost of 3. Different stages of the spike wave front are shown with the early stage in the top left image, and the later stages in the images proceeding from left to right columns and from the top to bottom rows. The final path is shown in the bottom right. The state of the spike wave and final route are depicted in gold.

D. Spike Wave Propagation and Path Readout

Fig. 1 shows the progression of a spike wave in a typical environment. Per the AER, the neuron ID and time step are recorded for each spike, which is shown in gold. The gold square in the top left image of Fig. 1 shows the wave emanating from location (58,53). The ensuing 8 figure panels going from left to right and top to bottom show the wave's progress. In this instance the wave has broken up depending on the cost of the environment. Note that spikes along the road, which have a value of 1, have progressed further because of a shorter axonal delay. The spike wave algorithm ends when there is a spike at the goal location (25,13).

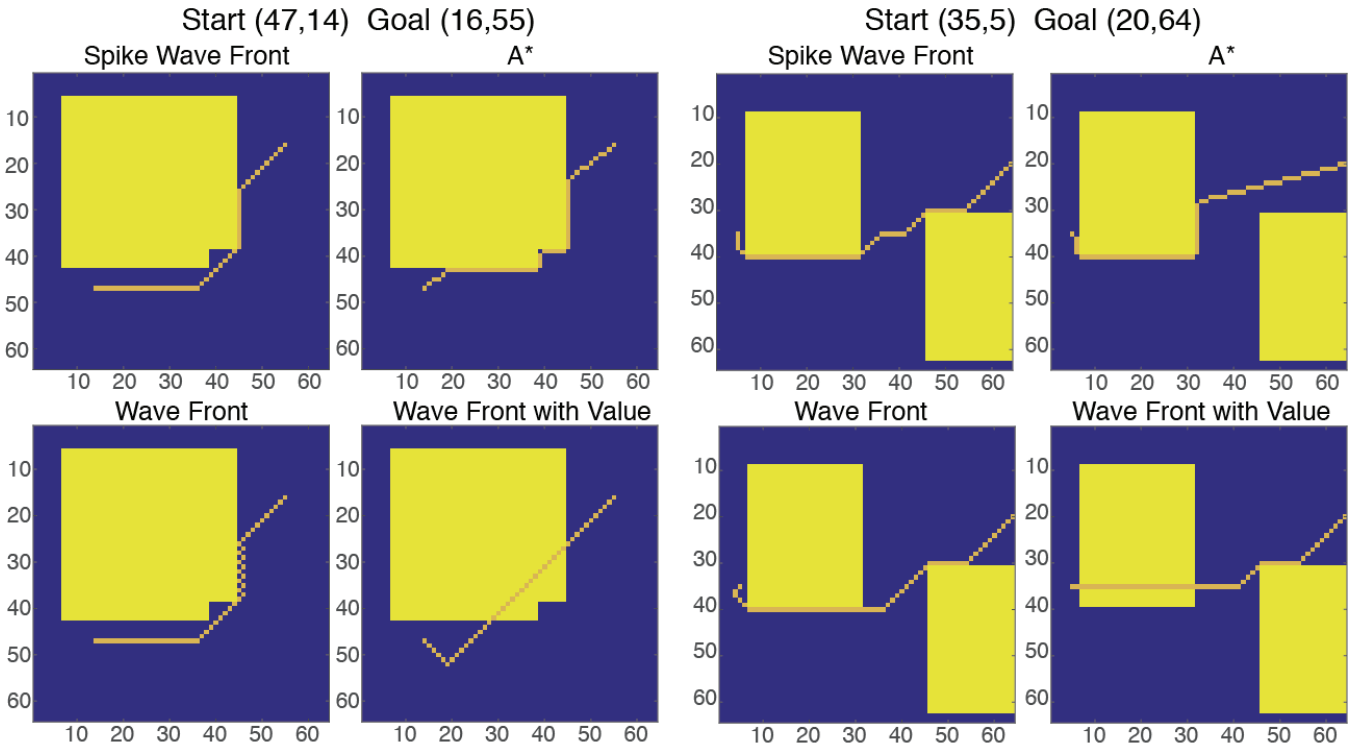

Fig. 2. Comparison of the four planning algorithms described in Section II on two representative maps. On both maps, the obstacles are shown in yellow, open space is shown in blue, and the paths generated by the algorithms are shown in gold. The map on the left with a start location of (47,14) and a goal location of (16,55) had one obstacle. The map on the right with a start location of (35,5) and a goal location of (20,64) had two obstacles. See text for description.

To find the best path between the start and goal locations, we used the list of spikes held in AER format. From the goal, the list was searched for the most recent spike from a neuron whose location was adjacent to the goal location. If more than one spike met this criterion, the neuron whose location corresponded to the lowest cost and was closest to the start location was chosen. This iteratively proceeded from the goal through other neuron locations until a spike at the start location was found. The bottom right image in Fig. 1 shows the found path. Note how the path avoided obstacles and found the best road to get from the start (58,53) to the goal (25,13).

E. Standard Wave Front Algorithms and A Path Planner*

The standard wave front planner, a wave front planner with dynamic values, and the A* path planner were implemented to compare the spike wave front planner with other existing algorithms. The standard wave front planner was described above and implemented according to Section 4.5 of [15]. A value based wave front planner was also implemented. In this algorithm, instead of just incrementing the grid location of the wave's leading edge, the cost of the location was also added.

In addition, we implemented the A* algorithm [3], which is commonly used in path planning. A* uses a best-first search and attempts to find a least-cost path from the start location to the goal location. The cost includes the Euclidean distance from the start, the Euclidean distance from the goal, and the cost of traversing the location. From the start location, adjacent locations are placed in a node list. Then the node list is searched for the node with the lowest cost. The location corresponding to this low cost node is expanded by placing adjacent, unevaluated locations on the node list. The process is repeated until the goal location is reached. The A* algorithm can find the shortest path based on its cost function.

F. Statistics for Comparing Algorithms

To compare the path lengths, traversal costs and computation metrics, a paired-sample t-test was used (MATLAB, MathWorks, Inc.). The statistic tested for the null hypothesis that the experimental metrics from two algorithms, each with the same start, goal, and map, were equal. The p-values were Bonferroni corrected for multiple comparisons. The metric data was checked for normality using Q-Q plots.

III. Results

Fig. 1 shows a representative trial from an experiment. Several maps were generated to test the planner algorithms. For each map, multiple start and goal locations were tested. In all trials, the start and goal locations were not allowed to be at a major obstacle and the Euclidean distance between the start and goal locations had to be greater than 50.

A. Comparing Wave Front Planners

Fig. 2 shows representative trials comparing the spike wave front planner (top left for each map), to the A* algorithm (top right for each map), to the standard wave front planner (bottom left for each map), and to the wave front planner with values (bottom right for each map). The spike wave front and the

standard wave front are essentially equivalent and generate similar paths. The A* algorithm uses the Euclidean distance to the start and to the goal in its cost function and tends to be drawn towards those points (note how close it hugs the obstacles in Fig. 2).

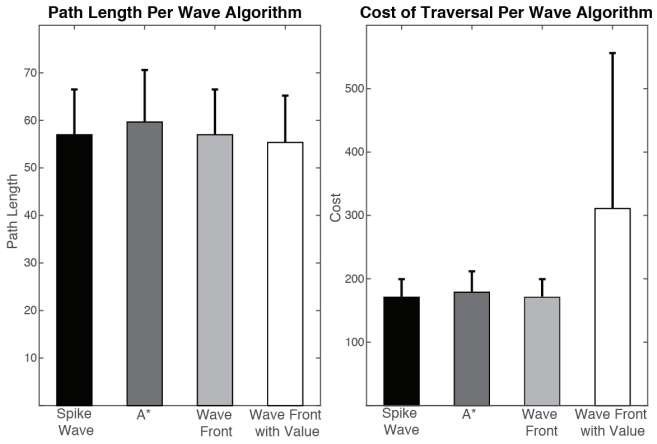

Fig. 3.Path lengths and traversal costs of the four path planning algorithms. The thick bars are the mean values of 300 trials (3 maps, 100 trials per map) and the error bars denote the standard deviations.

The standard wave front planner was not allowed to propagate across a major obstacle, but the other algorithms did not have this restriction. Depending on the start or goal location, the wave front with values algorithm occasionally propagated through major obstacles (see both maps in Fig. 2). Similarly, the spike wave front and the A* algorithms had the potential of passing through major obstacles. In the case of the spike wave front, if the wave propagated through the obstacle fast enough it might generate a path through an obstacle. In the case of A*, a path could be generated through an obstacle if the cost was lower. Despite this potential, paths through major obstacles never occurred in all the maps and locations tested with the spike wave and the A* algorithms.

To test the four path planning algorithms, three different maps were generated and 100 trials were run on each map. Fig. 3 shows the average path lengths (i.e., the number of locations visited between the start and goal) and traversal costs for each of these algorithms. The left bar graph in Fig. 3 shows that the spike wave front planner had significantly shorter paths than A* ($p < 0.0001$; t-test). The spike wave front paths did not differ significantly from the wave front planner; in fact they had almost identical distances. That the wave front planners generated shorter paths than A* is somewhat surprising, but looking at Fig. 2, it can be seen that A*'s cost function could pull the path close to an obstacle resulting in a longer path around obstacles. Because the wave front planner with values occasionally generated paths through obstacles, its paths were significantly shorter than the other algorithms ($p < 0.0001$; t-test).

The cost of traversal was calculated by summing the costs of locations along the generated paths (Fig. 3, right). Similar to the path length analysis, the cost of traversing the paths was significantly less for the spike wave front and wave front algorithms than the A* algorithm ($p < 0.0001$; t-test). Because the wave front with values sometimes generated paths through obstacles its cost was significantly higher ($p < 0.0001$; t-test).

Because the spike wave front planner can be implemented on neuromorphic hardware or on parallel spiking neural network simulators [16], it offers speedups over standard algorithms, such as A* or the wave front planner. When the environment contained open space and obstacles. The spike wave front planner was essentially equivalent to the standard wave front planner. However, whereas cycles for the wave front planners require mathematical operations executed on a conventional computer, the spike wave algorithm is compatible with neuromorphic hardware, which has been shown to have orders of magnitude savings in computation and bandwidth when using spike computations along with the AER format [17].

B. Spike Wave Front Planner's Adaptation to Change

A key feature of the spike wave front planner and A* is their ability to adapt to environmental change. The spike wave front planner's learning rule in Eq. (5) resembles the well-known delta rule and results in the planner being able to predict the cost of future locations through the agent's experience. The heuristic cost function of the A* algorithm can also take traversal costs into account when planning routes.

To show how these algorithms respond to contextual changes, we tested one case where the roads had a cost value of 1, and another case where the roads had the same cost value as open space (i.e., value = 3). Ten different maps (5 with one obstacle and 5 with two obstacles) were generated with 10 trials per map.

Table I shows the mean ± the standard deviation of the path length, the traversal cost, and the number of iterations for both algorithms in the road and no road cases. The number of iterations corresponds to how many loops each algorithm took to find a path. The Spike Wave Front and A* were significantly different ($p < 0.05$) for Path Length and Iterations in the "No Road" trials. All comparisons between the Spike Wave Front and A* for the "Road" trials were significantly different ($p < 0.0001$; t-test).

TABLE I. COMPARISON BETWEEN SPIKE WAVE FRONT PLANNER AND A* PLANNER WITH AND WITHOUT ROADS

	Spike Wave Front		**A***	
	Road	*No Road*	*Road*	*No Road*
Path Length	81.9±15.0	58.4±10.0	61.4±14.6	59.9±13.6
Traversal Cost	109.09±33.7	180.8±36.3	140.3±36.7	179.8±40.8
Iterations	88.8±16.6	168.7±29.2	245.7±242.0	227.3±258.1

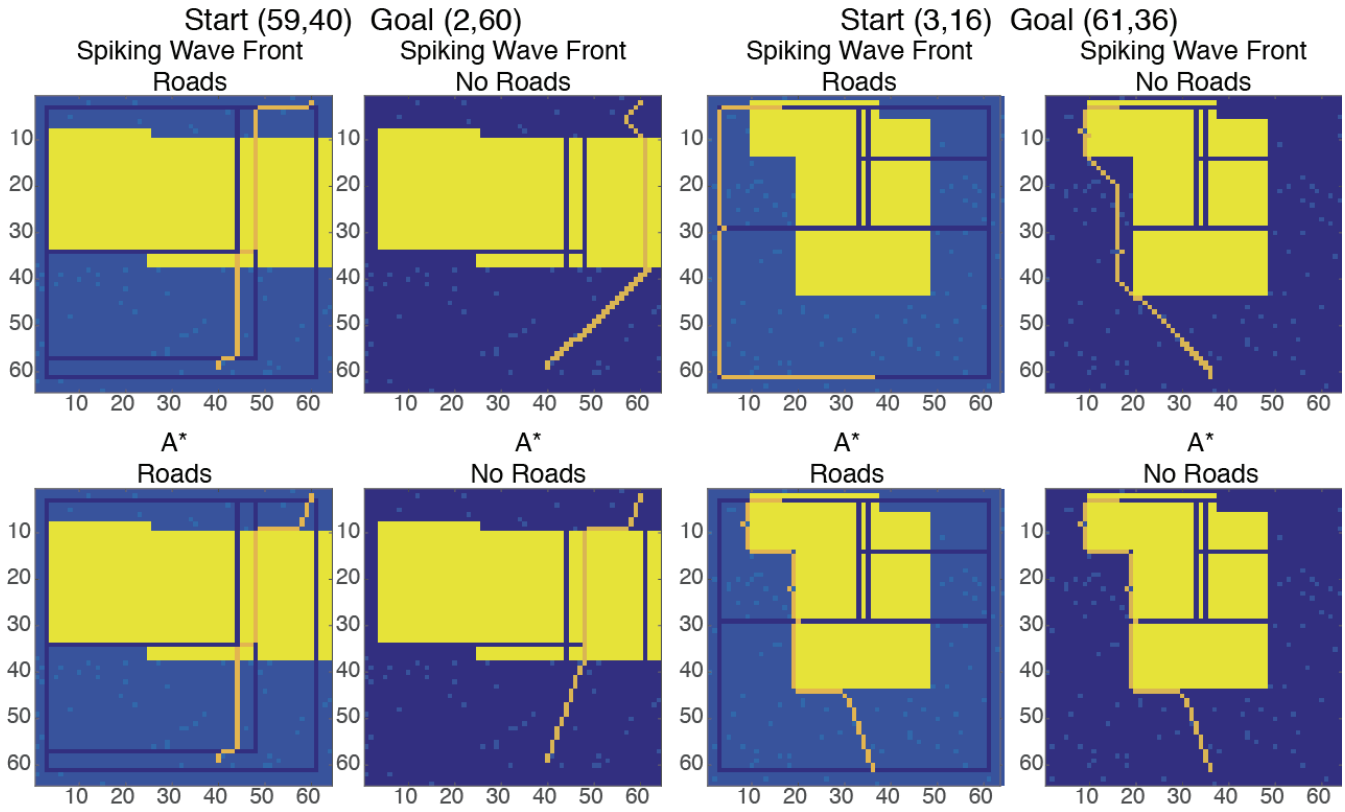

Fig. 4. Paths generated by the spike wave front planner and A* algorithm in the same environment, but with different costs. Two representative environments are shown. The left half of the figure shows a trial on the first environment with a start location at (59,40) and a goal location at (2,60). The right side of the figure shows a trial in the other environment with a start location at (3,16) and a goal location at (61,36). Images are pseudo colored to reflect the relative costs of objects in the environment with lowest cost being dark blue and highest cost being bright yellow. In the "Roads" condition, the cost of roads equals 1, cost of open space equals 3, cost of minor obstacles equals 5, and cost of major obstacles equals 25. In the "No Roads" condition, the roads were set equal to open space (i.e., value = 3). The calculated routes are shown in gold.

When roads were available, the path lengths were longer, but the traversal cost was much less. Because the A* algorithm did not use the roads as much as the spike wave, A*'s paths were shorter, but the cost was higher. For the A* algorithm longer paths meant more iterations. But, for the spike wave the number of iterations was related to the traversal cost because low cost paths meant spikes arrived quicker and the algorithm ended sooner. Fig. 4 shows two representative examples of these cases. When roads were present both algorithms tended to choose paths that used the roads. However, because the A* algorithm's cost function depended on the start and goal locations it oftentimes chose shorter paths that did not use roads (see maps on right of Fig. 4). When the roads were removed, both the spike wave front and the A* algorithm planned paths that utilized the open space while avoiding major and minor obstacles.

IV. Discussion

In the present paper, we introduced a path planning algorithm that used spiking neurons and axonal delays to compute efficient routes. The spike wave front path planner has several interesting properties: (i) Paths are near optimal and comparable to conventional path planning algorithms, such as the A* algorithm or a wave front planner. (ii) A learning rule based on axonal delays allows the algorithm to consider the cost of traveling through an environment and adapts paths accordingly. (iii) The algorithm uses spikes and the popular AER format making it compatible with neuromorphic hardware. (iv) The spike wave front takes significantly less iterations than the A* algorithm. (v) Because the spike wave front is a local algorithm (i.e., computations are independent and based on neighboring neurons), it is suitable parallel implementation.

A. Axonal Delay Plasticity

Recent evidence suggests that the myelin sheath, which wraps around and insulates axons, may undergo a form of activity-dependent plasticity [14, 18]. These studies have shown that the myelin sheath becomes thicker with learning motor skills and cognitive tasks. A thicker myelin sheath implies faster conduction velocities and improved synchrony between neurons.

The present algorithm introduced a learning rule inspired by activity-dependent myelinization, in which a path that traverses through an easy portion of the environment (e.g., via a road) would have shorter axonal delays than a path that travels through rough terrain. Assuming a robot or other autonomous vehicle had the ability to sense the cost of moving through a portion of the environment, by vision, accelerometers or other means, the robot could add such information to its map. In this way, the robot's path planner

could calculate a route based on difficulty. For example, Fig. 4 and Table I simulated a robot choosing between "off-road" and "on-road" routes just by changing the cost of the road in the robot's map. Moreover, paths could be planned based on the robot's present state or context. For example, if the robot was low on energy, it might risk traveling through obstacles to get to a charging station quickly. To achieve such a path, the relative cost of obstacles could be made lower and the calculated route would reflect this preference. Taken together, the axonal delay learning rule presented here could be a flexible method for selecting context and experience based path planning.

B. Hardware Implementation of Spike Wave Front Planner

Neuromorphic hardware differs from the conventional Von Neumann computer architecture in that it is asynchronous and event-driven, with parallel computation [7, 8]. The artificial neurons do not take up computation cycles unless they receive a spike event from a connected neuron. Typical neuromorphic designs have the memory, in the form of synapses, co-located with the processing units, that is, the neurons. This allows computations to be local, independent, and parallel. These features allow neuromorphic chips to have low size, weight and power [7, 19]. Nearly all these chips use spiking neuron elements and some form of AER.

The present algorithm is compatible with neuromorphic hardware. It builds upon a hardware implementation of a spike-based wave front planner [6]. It adds a learning rule, which depends on axonal delays, to make the planner more flexible and to consider the relative costs of traversing an environment. Axonal delays have been introduced in large-scale spiking neural network simulations [20, 21], but are not typical for neuromorphic hardware. However some neuromorphic designs include axonal delays [22]. To implement the present algorithm in neuromorphic hardware, all that would be needed is a delay buffer or a means to change the conductance properties of a connection. Because a synaptic based learning rule, such as Spike Timing Dependent Plasticity (STDP), is not needed for the present algorithm, the circuitry to support the spike wave front planner could be simplified.

It should be noted that the present algorithm used a simplified spiking neuron to demonstrate the algorithm's path planning capabilities. Many neuromorphic hardware designs support more complex neuron models, such as leaky integrate and fire neurons or the Izhikevich neuron [17, 22-25]. In previous work, we have shown that similar wave dynamics to that shown here can be demonstrated in more complex neuron models [26]. Therefore, it would be feasible to convert the present algorithm to a more complex neuron model given the right settings.

In the present algorithm, the AER representation is used to read out the path, which may be a limitation since it requires saving the AER list for each planned route. It also requires a planning calculation and readout for every route. A more natural implementation might use the rank order of the spike wave in a similar way to that proposed by Thorpe and colleagues [27, 28]. Such alternative readout implementations will be explored in the future.

C. Learning Unknown and Uncertain Environments

The present simulations assumed a known, static environment with instantaneous learning ($\delta = 1.0$ in Eq. 5). In addition, the system started out naïve before every route traversal. That is, the delay buffers were reset to their initial value before being given a start and goal location. Future extensions of the spike wave front planning algorithm could readily work in unknown or uncertain environments by setting the learning rate to be between 0.0 and 1.0, and not resetting the delay buffer values (D in Eq. 5) after each traversal. Instead, the system could retain its learned values between trials and the learning rate could be set to reflect the dynamics of the environment.

If an environment were uncertain, using the spike wave front planner with a learning rate between 0 and 1 could build a dynamic value map of the environment. For example, if a location were traversable 25% of the time, the algorithm would learn this uncertainty and plan accordingly. Similar to [29], this would result in the spike wave front planner predicting the cost of traversing locations in an environment. Moreover, the planner's prediction would take the uncertainty into consideration. Such a planner would have advantages in learning the landscape of an environment, responding flexibly and fluidly to change, and not falling into local minima.

In an unknown environment, a robot or autonomous agent can sense the cost of locations in the environment and update its map accordingly. When the agent is tasked with planning a route between locations it can use the cost values for which it has experience and can specify a cost value for unknown locations in the map. The cost of unknown locations could be set depending on the agent's context (e.g., low for exploration vs. high for exploitation). Moreover, as the agent gained more experience traversing routes, the map could become more complete with this cost metadata.

D. Spike Wave Front Comparison to Other Planning Algorithms

In terms of path lengths, traversal cost, and iterations through the algorithm, the spike wave front planner was as good or better than other implemented path planning algorithms (Fig. 3 and Table I).

Since the underlying algorithm is the same, the performance of the spike wave front planner was roughly equivalent to the wave front planner. However, the axonal delay learning rule allowed the spike wave front to take traversal cost into consideration and could adapt to environmental change (Fig. 4).

The spike wave front planner had interesting differences from the A* algorithm. A* uses the start and goal location to calculate paths and this leads to short paths that may not have the lowest traversal cost. The spike wave front only considers the local computations at the front edge of the wave. Not only does this make the algorithm more sensitive to traversal cost, but it also means that the spike wave front computations are local and independent. Efficient routes are computed without using the global start and goal locations.

V. Conclusions

In summary, the spike based wave front planner introduced here has interesting features that make it attractive for autonomous and embedded systems. A form of activity-dependent plasticity (i.e., changes in axon myelinization), which is rarely used in neural simulations, inspired the algorithm. The algorithm shows improvements over other path planning algorithms because it computes with spiking neurons and it can adapt to environmental change. Spiking algorithms have been shown to run efficiently and with low power on neuromorphic hardware, making this algorithm suitable for micro aerial vehicles and other embedded devices. Straightforward extensions to the algorithm would further improve its flexibility and adaptive behavior.

Acknowledgment

The author would like to thank the members of the Cognitive Anteater Robotics Laboratory and Philippe Gaussier for many useful discussions.

References

[1] S. M. LaValle, "Motion Planning," *Robotics & Automation Magazine, IEEE,* vol. 18, pp. 79-89, 2011.

[2] S. M. LaValle, "Motion Planning," *Robotics & Automation Magazine, IEEE,* vol. 18, pp. 108-118, 2011.

[3] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *Systems Science and Cybernetics, IEEE Transactions on,* vol. 4, pp. 100-107, 1968.

[4] A. Stentz and I. Carnegle, "Optimal and Efficient Path Planning for Unknown and Dynamic Environments," *International Journal of Robotics and Automation,* vol. 10, pp. 89-100, 1993.

[5] S. M. LaValle and J. J. Kuffner, "Randomized Kinodynamic Planning," *The International Journal of Robotics Research,* vol. 20, pp. 378-400, May 1, 2001 2001.

[6] M. Soulignac, "Feasible and Optimal Path Planning in Strong Current Fields," *Robotics, IEEE Transactions on,* vol. 27, pp. 89-98, 2011.

[7] G. Indiveri, B. Linares-Barranco, T. J. Hamilton, A. van Schaik, R. Etienne-Cummings, T. Delbruck*, et al.*, "Neuromorphic silicon neuron circuits," *Front Neurosci,* vol. 5, p. 73, 2011.

[8] J. L. Krichmar, P. Coussy, and N. Dutt, "Large-Scale Spiking Neural Networks using Neuromorphic Hardware Compatible Models," *J. Emerg. Technol. Comput. Syst.,* vol. 11, pp. 1-18, 2015.

[9] J. Barraquand, B. Langlois, and J. C. Latombe, "Numerical potential field techniques for robot path planning," *Systems, Man and Cybernetics, IEEE Transactions on,* vol. 22, pp. 224-241, 1992.

[10] R. E. Bellman, "On a routing problem," *Quarterly of Applied Mathematics,* vol. 16, pp. 87-90, 1958.

[11] P. Gaussier, A. Revel, J. P. Banquet, and V. Babeau, "From view cells and place cells to cognitive map learning: processing stages of the hippocampal system," *Biological Cybernetics,* vol. 86, pp. 15-28, Jan 2002.

[12] M. Quoy, P. Laroque, and P. Gaussier, "Learning and motivational couplings promote smarter behaviors of an animat in an unknown world," *Robotics and Autonomous Systems,* vol. 38, pp. 149-156, Mar 31 2002.

[13] S. Koziol, S. Brink, and J. Hasler, "Path planning using a neuron array integrated circuit," in *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*, 2013, pp. 663-666.

[14] R. D. Fields, "A new mechanism of nervous system plasticity: activity-dependent myelination," *Nat Rev Neurosci,* vol. 16, pp. 756-767, 12//print 2015.

[15] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki*, et al.*, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Cambridge, MA: The MIT Press, 2005.

[16] M. Beyeler, K. D. Carlson, T.-s. Chou, N. Dutt, and J. L. Krichmar, "CARLsim 3: A User-Friendly and Highly Optimized Library for the Creation of Neurobiologically Detailed Spiking Neural Networks," in *International Joint Conference on Neural Networks*, Killarney, Ireland, 2015.

[17] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan*, et al.*, "Artificial brains. A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science,* vol. 345, pp. 668-73, Aug 8 2014.

[18] R. D. Fields, "White matter in learning, cognition and psychiatric disorders," *Trends Neurosci,* vol. 31, pp. 361-70, Jul 2008.

[19] N. Srinivasa and J. Cruz-Albrecht, "Neuromorphic adaptive plastic scalable electronics: analog learning systems," *IEEE Pulse,* vol. 3, pp. 51-6, Jan 2012.

[20] E. M. Izhikevich and G. M. Edelman, "Large-scale model of mammalian thalamocortical systems," *Proc Natl Acad Sci U S A,* vol. 105, pp. 3593-8, Mar 4 2008.

[21] E. M. Izhikevich, J. A. Gally, and G. M. Edelman, "Spike-timing dynamics of neuronal groups," *Cereb Cortex,* vol. 14, pp. 933-44, Aug 2004.

[22] J. M. Cruz-Albrecht, M. W. Yung, and N. Srinivasa, "Energy-Efficient Neuron, Synapse and STDP Integrated Circuits," *Biomedical Circuits and Systems, IEEE Transactions on,* vol. 6, pp. 246-256, 2012.

[23] T. Sharp, F. Galluppi, A. Rast, and S. Furber, "Power-efficient simulation of detailed cortical

microcircuits on SpiNNaker," *J Neurosci Methods,* vol. 210, pp. 110-8, Sep 15 2012.
[24] E. M. Izhikevich, "Which model to use for cortical spiking neurons?," *IEEE Trans Neural Netw,* vol. 15, pp. 1063-70, Sep 2004.
[25] T. Pfeil, A. Grubl, S. Jeltsch, E. Muller, P. Muller, M. A. Petrovici*, et al.*, "Six networks on a universal neuromorphic computing substrate," *Front Neurosci,* vol. 7, p. 11, 2013.
[26] T.-S. Chou, L. D. Bucci, and J. L. Krichmar, "Learning Touch Preferences with a Tactile Robot Using Dopamine Modulated STDP in a Model of Insular Cortex," *Frontiers in Neurorobotics,* vol. 9, 2015.
[27] S. Thorpe, A. Delorme, and R. Van Rullen, "Spike-based strategies for rapid processing," *Neural Netw,* vol. 14, pp. 715-25, Jul-Sep 2001.
[28] R. VanRullen, R. Guyonneau, and S. J. Thorpe, "Spike times make sense," *Trends Neurosci,* vol. 28, pp. 1-4, Jan 2005.
[29] W. Schultz, P. Dayan, and P. R. Montague, "A neural substrate of prediction and reward," *Science,* vol. 275, pp. 1593-9, Mar 14 1997.