A Rapid Adapting and Continual Learning Spiking Neural Network Path Planning Algorithm for Mobile Robots

Harrison Espino[1] and Robert Bain[2] and Jeffrey L. Krichmar, *Senior Member IEEE*[1,3]

Abstract*—*MAPPING traversal costs in an environment and planning paths based on this map are important for autonomous navigation. We present a neurorobotic navigation system that utilizes a Spiking Neural Network (SNN) Wavefront Planner and E-prop learning to concurrently map and plan paths in a large and complex environment. We incorporate a novel method for mapping which, when combined with the Spiking Wavefront Planner (SWP), allows for adaptive planning by selectively considering any combination of costs. The system is tested on a mobile robot platform in an outdoor environment with obstacles and varying terrain. Results indicate that the system is capable of discerning features in the environment using three measures of cost, (1) energy expenditure by the wheels, (2) time spent in the presence of obstacles, and (3) terrain slope. In just twelve hours of online training, E-prop learns and incorporates traversal costs into the path planning maps by updating the delays in the SWP. On simulated paths, the SWP plans significantly shorter and lower cost paths than A* and RRT*. The SWP is compatible with neuromorphic hardware and could be used for applications requiring low size, weight, and power.**

***Index Terms*—Autonomous Vehicle Navigation, Motion and Path Planning, Neurorobotics**

I. INTRODUCTION

Finding one's way around in an ever-changing world is an important part of everyday life. Similarly, robots and other autonomous systems require this capability. Researchers and industry have made considerable progress developing navigation systems. However, critical open issues have been identified such as: 1) Generating maps using data from multiple sensors, 2) Continual learning without offline retraining, and 3) Flexibility in the face of a changing environment and different navigational objectives, such as conserving battery usage or avoiding foot traffic [1]–[3]. In the proposed work, we address these issues by developing a novel approach that takes different traversal costs into account when navigating.

Manuscript received: May 10, 2024; Revised: August 6, 2024; Accepted: August 27, 2024.

This paper was recommended for publication by Editor Tetsuya Ogata upon evaluation of the Associate Editor and Reviewers' comments.

This work was supported by the Air Force Office of Scientific Research (AFOSR) Contract No. FA9550-19-1- 0306, and by the National Science Foundation (IIS-RI award 1813785 and NSF-FO award IIS-2024633).

[1]Espino is with the Department of Computer Science, University of California, Irvine, Irvine, CA, USA `espinoh@uci.edu`

[2]Bain is with the Department of Neurobiology and Behavior, University of California, Irvine, Irvine, CA, USA `rkbain@uci.edu`

[1,3]Krichmar is with the Department of Cognitive Sciences and Department of Computer Science, University of California, Irvine, Irvine, CA, USA `jkrichma@uci.edu`

Digital Object Identifier (DOI): see top of this page.

We present a spiking neural network navigation system that simultaneously constructs environmental cost maps and uses those maps to plan efficient paths. The system is tested on a ground robot in rugged, varied outdoor terrains with cost maps for obstacles, slope, and for the robot's effort based on the motor's current draw. We show that the robot rapidly learns to plan paths that avoid impassable trees and benches with an obstacle cost map, and plans smoother or flatter paths with the current or slope cost map.

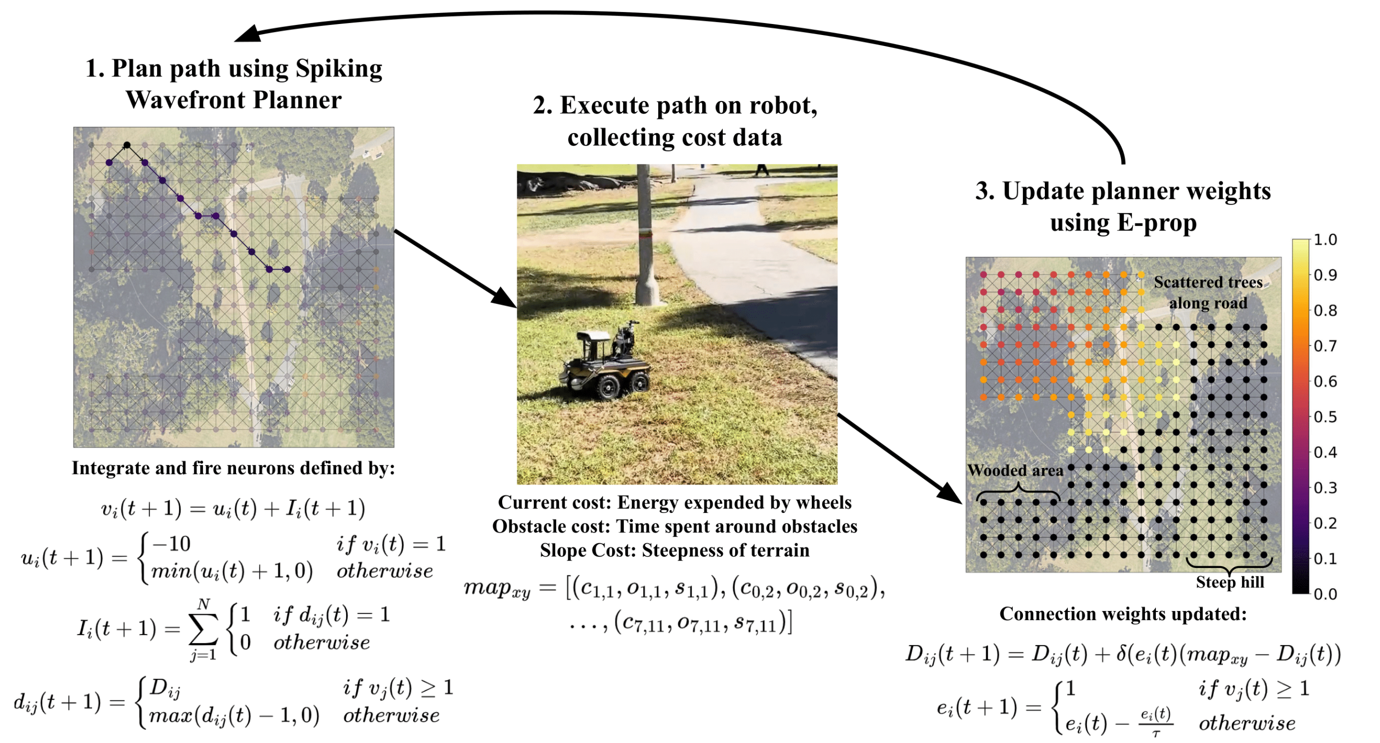
Our utilization of spiking neurons makes implementation on highly parallel hardware, like neuromorphic hardware, possible in the future. Compared to conventional architectures, neuromorphic networks provide greater energy efficiency and hardware size advantages [4], [5]. The SWP [6] and E-prop [7] have been implemented on neuromorphic hardware. The present work demonstrates the applicability of these elements for robotic navigation.

SWP was introduced in [8], and compared to A* in simplified pre-mapped environments, where the costs were uniform values for sidewalks, grass, and obstacles. E-Prop was added to SWP to update connection delays in simple, grid world simulations with uniform, noise-free costs [9]. An open question is whether the SWP has advantages where costs are discovered by a physical robot's sensors.

The main contributions of this work are as follows:

1) We show that our navigation system can simultaneously map complex, real-world environments in real time and plan paths over multiple measures of cost, which are measured with noisy sensor readings onboard a physical robot. This map is continuously learned online through experience. The robot uses this map to plan trajectories depending on what costs are considered.
2) In trials with a ground robot, we show that the robot can learn a cost map in a few hundred training steps and several hours of runtime. We also show that the robot can adapt to changes in the environment in just a few trials without taking the system offline.
3) By exhaustively simulating all paths through our learned costmap, we find that the SWP is the best candidate for minimizing cost and path length between itself, A*, RRT*, and a shortest Euclidean distance planner.
4) The present spiking neuron implementation could be implemented on neuromorphic hardware to reduce the size, weight, and power of a navigation system.

Figure 1 provides an overview of the path planning system. The equations in the figure are described in Sections III-A and


1. Plan path using Spiking Wavefront Planner
2. Execute path on robot, collecting cost data
3. Update planner weights using E-prop
Scattered trees along road
Wooded area
Steep hill
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Integrate and fire neurons defined by:
Current cost: Energy expended by wheels
Obstacle cost: Time spent around obstacles
Slope Cost: Steepness of terrain
Connection weights updated:

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

$$u_i(t+1) = \begin{cases} -10 & if\ v_i(t) = 1 \\ min(u_i(t)+1, 0) & otherwise \end{cases}$$

$$I_i(t+1) = \sum_{j=1}^{N} \begin{cases} 1 & if\ d_{ij}(t) = 1 \\ 0 & otherwise \end{cases}$$

$$d_{ij}(t+1) = \begin{cases} D_{ij} & if\ v_j(t) \geq 1 \\ max(d_{ij}(t)-1, 0) & otherwise \end{cases}$$

$$map_{xy} = [(c_{1,1}, o_{1,1}, s_{1,1}), (c_{0,2}, o_{0,2}, s_{0,2}), \ldots, (c_{7,11}, o_{7,11}, s_{7,11})]$$

$$D_{ij}(t+1) = D_{ij}(t) + \delta(e_i(t)(map_{xy} - D_{ij}(t))$$

$$e_i(t+1) = \begin{cases} 1 & if\ v_j(t) \geq 1 \\ e_i(t) - \frac{e_i(t)}{\tau} & otherwise \end{cases}$$

Fig. 1: Overview of our navigation system. **1.** Neurons in the SWP represent locations in space. To plan a path, a spike is induced at the robot's start location and propagates until it reaches the goal location. **2.** The robot navigates to the goal using the planned path. Internal measurements and environmental sensors track costs when traversing between neuron waypoints. **3.** Using the learned costs, the planner is updated using E-prop. The figure shows each neuron's eligibility trace, which determines the update magnitude based on how recently the neuron spiked.

III-B.

II. RELATED WORKS

A. Path Planning

Traditional solutions to path planning often approach the problem by traversing through a graph of costs, or through direct sampling of the state space. Here, we briefly review popular algorithms from both.

A* traverses a graph by considering nodes according to a priority queue sorted by an immediate cost value and a heuristic. Given an admissible heuristic, A* has been proven to provide cost optimal paths [10]. A* has been utilized for robot navigation as well as applications such as puzzle solving and resource allocation. In unknown environments, the D* algorithm (shortened from Dynamic A*) aims to reach a goal while continually re-planning when new information about the environment is discovered [11].

Sampling-based algorithms form a viable path by exploring the space of permissible states. Rapidly exploring random tree (RRT) is one such algorithm designed to efficiently construct a space filling tree until a sequence of valid states between a start and goal state is found [12]. It is often employed in high-dimensional trajectory planning problems in which quickly generating a feasible path is more important than generating an optimal one [13].

Many variants of RRT have been created to address problems such as optimality or environmental changes. For the former, RRT* is a variant of RRT which aims to optimize for some cost using a tree rewiring step [14]. RRTX further builds on this concept for dynamic environments [15].

B. SLAM

Mapping of an environment is often accomplished through the process of simultaneous localization and mapping (SLAM). This is a thoroughly explored problem in robotics in which the environmental features and the robot's position in the environment are unknown, and both must be estimated through sensory and self-motion data. Classical solutions to this problem such as extended Kalman filter SLAM (EKF-SLAM) iteratively estimate a posterior probability distribution for the robot pose and landmark positions [16]. Other methods use the data as constraints to a graphical network representing the posterior [17], or rely on tracking changes in visual input [18]. In a more biologically-inspired approach, RAT-SLAM integrates visual place recognition and robot position to represent the robot and environment through a grid of "pose cells" [19].

In these cases, the map consists of geometric environmental features or salient visual features that serve the purpose of aiding localization. Not present are the aspects of steepness, unevenness, or other measures of traversal difficulty meant to aid in navigation. Methods such as [20] or [21] use deep learning approaches such as inverse reinforcement learning or semantic segmentation to generate a costmap of such features for the surrounding environment. Semantic segmentation uses offline training with a dataset of semantically labeled images. Inverse reinforcement learning requires human demonstration with which to infer a reward function.

Solutions to SLAM also assume that the robot's trajectory is decided externally through manual controls or a separate path planning policy. The emerging field of active SLAM combines path planning with SLAM, with the objective of choosing a trajectory which accurately and quickly maps an environment [22]. The goal of path planning in this paradigm is to formulate a path which minimizes the uncertainty of the SLAM algorithm. However, for some applications, it may be necessary to dynamically consider other goals depending on the context. For instance, an autonomous robot may need to manage battery usage at certain times by planning paths which minimize power consumption. Thus, a navigation system which affords such versatility is needed.

C. Deep Learning Methods

Deep learning solutions to trajectory planning include imitation learning and reinforcement learning for end to end navigation. Imitation learning methods such as [23]–[25] use human annotated data to mimic expert demonstrations. Due to their reliance on pre-collected data, they do not allow for continuous or autonomous data. Reinforcement learning (RL) methods closely mimic humans learning from interaction with their environments, and have found great success navigating through complex environments. However, they require extensive offline training and expensive onboard computation to run in real-time [26]–[29]. As such, there is a need for an online and continuously learning sample-efficient navigation system.

III. BACKGROUND

A. Spiking Wavefront Planner

Here we briefly describe the SWP model. For more details, see [8], [9].

The spiking wavefront propagation algorithm assumes a grid representation of space, where connections between units represent the ability to travel from one grid location to a neighboring location. Each unit in the grid is represented by simplified integrate and fire neurons. Rather than weights between neurons, the connections between neurons represent a propagation delay, such that a spike signal takes D time steps before being received by a downstream neuron. The activity of neuron i at time $t+1$ is represented by (1):

$$v_i(t+1) = u_i(t) + I_i(t+1), \tag{1}$$

in which $u_i(t)$ is the recovery variable, $I_i(t)$ is the input current, and t refers to time when simulating neuron dynamics.

The recovery variable $u_i(t)$ is described by (2):

$$u_i(t) = \begin{cases} \beta & \textit{if } v_i(t) = 1 \\ \min(u_i(t-1)+1, 0) & \textit{otherwise} \end{cases}, \tag{2}$$

such that immediately after a membrane potential spike, the recovery variable starts as a negative value β and linearly increases toward a baseline value of 0. For our experiments, β is set to -10. We found this to be sufficiently large enough to prevent a spike from reactivating previously visited nodes.

The input current I at time $t+1$ is given by (3):

$$I_i(t+1) = \sum_{j=1}^{N} \begin{cases} 1 & if\ d_{ij}(t) = 1 \\ 0 & otherwise \end{cases}, \tag{3}$$

such that $d_{ij}(t)$ postpones the integration of input, I, from neighboring neuron j to neuron i. This delay is given by (4):

$$d_{ij}(t+1) = \begin{cases} D_{ij} & \textit{if } v_j(t) \geq 1 \\ \max(d_{ij}(t)-1, 0) & \textit{otherwise} \end{cases}. \tag{4}$$

The value of $D_{ij}(t)$ is the propagation delay between neurons i and j, and denotes the expected cost of traveling from location i to j. This is initialized to 1 for all values. Cost is an open parameter, which could depend on a number of variables. In the present paper, multiple measures of cost are measured simultaneously, which is explained in more detail in section V-A.

B. E-Prop

The E-Prop learning rule was developed to learn sequences in recurrent spiking neural networks by using an eligibility trace to implement backpropagation through time to minimize a loss function [30]. The present work used E-Prop to learn a map of the environment, which is represented by a recurrent spiking neural network, based on the sensed cost of traversal. For path planning purposes, the active neurons after a wave propagation are eligible for updates. An eligibility trace based on time elapsed since the wave reaches the goal destination dictates the eligibility. E-Prop is applied to the delay D_{ij} between neuron i and j along the traversed path.

$$D_{ij}(T+1) = D_{ij}(T) + \delta(e_i(t)(m_{xy} - D_{ij}(T)), \tag{5}$$

where δ is the learning rate, set to 0.5, $e_i(t)$ is the eligibility trace for neuron i, and m_{xy} represents the cost observed from the robot's sensors at location (x, y), which corresponds to neuron i. In this case, T represents time incremented each time a path is completed. This rule is applied for each of the neighboring neurons, j, of neuron i. The loss in Eqn. 5 is m_{xy} - D_{ij}.

The eligibility trace for neuron i is given by (6):

$$e_i(t+1) = \begin{cases} 1 & if\ v_j(t) \geq 1 \\ e_i(t) - \frac{e_i(t)}{\tau} & otherwise \end{cases}, \tag{6}$$

where τ is the rate of decay for the eligibility trace, set to 25. An example of the eligibility trace from a planned path can be seen in the right panel of Fig. 1.

To determine a path from the robot's current location to a destination, a signal is sent originating from the neuron corresponding to the robot's current location. This signal propagates based on the delays D to the origin neuron's neighbors. This is repeated for each of these neurons and their

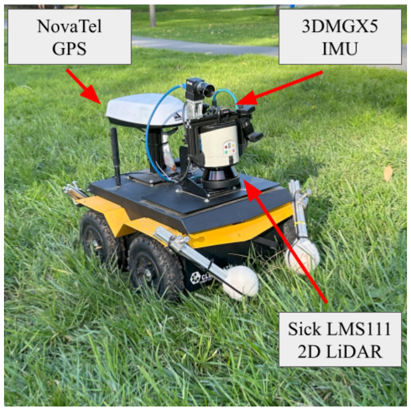


Fig. 2: The Clearpath Jackal robot.

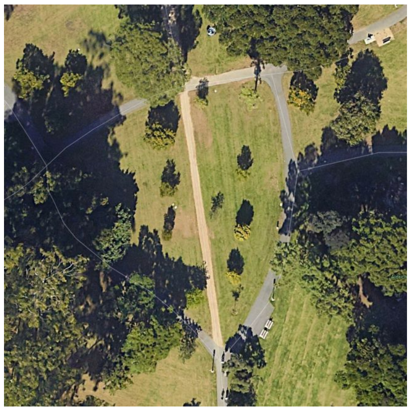


Fig. 3: Top down view of the Aldrich Park environment. Imagery ©Google

neighbors until a signal reaches the neuron corresponding to the destination. The origin of this signal is recursively traced backwards until the neuron representing the current location is reached. This sequence of neurons represents the path of least cost given the robot's information about the environment.

IV. Experimental Setup

A. Robot Platform and Environment

The robot platform used in the experiments was the Jackal Unmanned Ground Vehicle from Clearpath Robotics (Fig. 2). The Jackal is capable of navigating through difficult, uneven terrain. To localize the robot in its environment and determine the distance to waypoints we used a NovaTel GPS unit. To determine the robot's heading and bearing, we used the Lord Microstrain 3DMGX5 inertial measurement unit.

We tested our navigation system in Aldrich park: A hilly park located at the center of of the University of California, Irvine. A top down view of the environment can be seen in Fig. 3. A 17x17 grid, 5.1 meters apart were used as waypoints (see Fig. 4). This distance was chosen based on the precision of the GPS unit of 1.2 meters. Terrain in the environment was hilly and varied between thick grass, paved road, and dirt road. The environment contained a number of trees which served as obstacles for the robot in addition to foot traffic along the road. The robot explored this environment for 350 trials over 4 days and approximately 12 hours of total runtime. Some sections of the grid were removed from consideration, as they were completely intraversable due to large root structures that could not be detected by the LiDAR. The system was initially tested in a field near the UCI campus with similar results, but we found Aldrich Park to have more challenging features.

B. Evaluation and Comparisons

Using the learned costmap, we compared the SWP to the RRT* [14], A* [10], and D* Lite [11] path planning algorithms. For RRT*, the extension of the tree was constrained to be only in the direction of waypoints. This was necessary, as the costmap only contains delays for movement in the cardinal and ordinal directions. For A* and D* Lite, we used the common heuristic of shortest Euclidean distance.

We tested paths generated by these algorithms as well as the SWP on 25 randomly selected paths whose start and end waypoints were at least 3 waypoints apart to ensure meaningful choices in path planning. During traversal, we collected measures corresponding to the learned costs.

To evaluate the effectiveness of the SWP in navigating our cost maps, we also exhaustively simulated all potential start and end locations with a minimum separation of 3 waypoints. This was similarly done for A*, RRT*, D* Lite, and a naive planner which generated a path with the shortest Euclidean distance.

V. Methods

A. Cost Measures

We considered three measures of cost:

1) **Current Cost:** The amount of current from the battery to the left and right wheels. This was determined internally using the status messages automatically published by the Jackal. For each incoming current reading, we considered the minimum value between the left and right side. This was to prevent spikes in current caused by turning from influencing the cost, which may happen due to trajectory recalculation and not from features in the environment.
2) **Obstacle Cost:** The presence of obstacles during traversal. To recognize obstacles we used the SICK LMS111 2D LiDAR, which has an aperture angle of 270 degrees and an angular resolution of 0.5 degrees. Objects were considered obstacles if the LiDAR detected a number of consecutive data points below a threshold of 2 meters. The cost before normalization was calculated as the fraction of time spent with obstacles in the field of view en route to the waypoint. The Jackal avoided an obstacle by turning away from it until the obstacle was out of view. In the case that a collision was still eminent (such as if part of the obstacle was too high or low for the LiDAR), manual controls were used to direct the Jackal past the obstacle. This was a rare occurrence, and was only necessary due to benches in the environment lying above the field of view.

3) **Slope Cost:** The slope of the ground was based on the pitch and roll readings from the IMU. Because flat ground is measured as 0 radians of rotation about both axes, this cost (before normalization) was calculated as the sum of the rotation about the pitch and roll axes.

Additionally, a cost representing completely intraversable locations was included, which incurred cost only when the robot was unable to reach a waypoint in the allotted time. In this case, a maximum cost of 10 was assigned to the delays into this waypoint to discourage its use in future paths.

In order to convert costs to values suitable for training the SWP, sensor data needed to be converted into integers representing learnable delay values. We chose 10 as the maximum delay value to maintain a fast network response during wave propagation. To obtain normalization constants for each measure, the Jackal was driven between waypoints prior to training. Minimum and maximum values were calculated as two standard deviations below and above the mean. Outliers during training were clamped prior to normalization.

Each of these costs are maintained individually for a given neuron's delay values. To combine costs into a single map, the delay values for each chosen cost are added together. This value is then normalized between 1 and 10 again across all neurons to maintain a fast network response.

B. Environment Mapping

A single trial with the robot proceeded as follows. First, an end point was randomly determined using a Levy Flight distribution. This distribution is commonly used to model foraging patterns in animals [31]. The Levy Flight distribution will tend to focus search in a local area while occasionally jumping to a distant area. This was a better exploration strategy than a more random search pattern, such as Brownian motion.

Next, the SWP planned a path from the robot's current location to the end point by setting the activity v_i of the starting neuron to 1 and simulating the neuron behavior as in Eqns. (1) through (4). The delay values d_{ij} for each neuron used a costmap combining all measures of cost. The eligibility trace generated by the SWP was used to update the delays with E-prop as in Eqn. (5). The robot then navigated between waypoints determined by the generated path.

To reach a waypoint, the robot oriented to the direction of the waypoint by rotating until the robot's heading (orientation with respect to a global reference frame) matched the desired bearing (orientation with respect to the waypoint). When the heading was suitably close, the robot would proceed towards the waypoint. If at any point during traversal the difference between the bearing and the heading exceeded $\frac{\pi}{12}$, the robot would stop to rotate to the correct orientation before proceeding. We found this value to be a suitable level of precision for our IMU.

The trial was complete once all waypoints were reached, or the robot was unable to reach a waypoint after 45 seconds. In the latter case, the robot would return to the previous waypoint. After each trial, the delays of the costmap were saved. The saved delays were used in simulated experiments to analyze possible paths taken by the robot under different path planning algorithms.

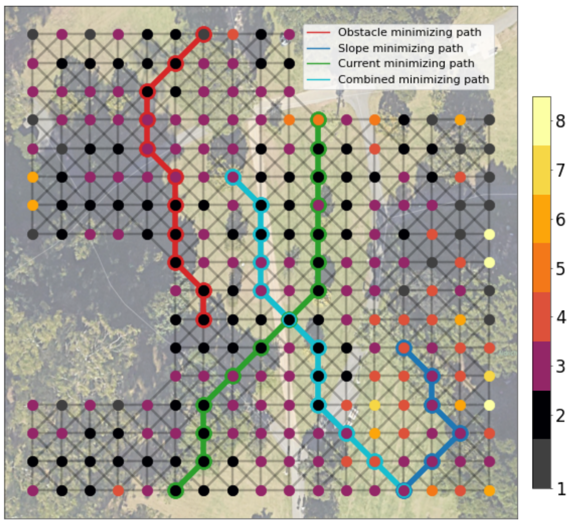


Fig. 4: The costmap for all costs added and normalized after learning. Nodes are colored according to the mean of the delays D from other nodes. Example paths minimizing current drawn, obstacles encountered, and steepness are colored green, red, and blue, respectively.

VI. RESULTS

A. Environment Mapping

The costs determined from the robot's exploration of Aldrich park are shown in Fig. 4. Nodes are colored according to the mean of delays D from other nodes using all costs combined. Candidate paths minimizing current drawn, obstacles encountered, and steepness, as well as the path taken when these costs are combined, are shown in the figure.

Using the current cost criteria, we found similar values between the grass, dirt road, and pavement, indicating they were similarly traversable in terms of energy consumption. Areas with slightly elevated current cost included waypoints around trees and at intersections between terrain. We speculate this is because the transition from grass to pavement or to dirt road caused increased unevenness, and consequently, stress on the motors. The current minimizing path took a relatively straight path visiting a minimal amount of waypoints (green line in Fig. 4).

The obstacle cost criteria produced sparser costs, with higher costs at tree and bench locations. Foot traffic along the sidewalk also resulted in higher obstacle costs. Occasionally, due to the unevenness from the road or from sloped areas, the robot was also at an angle steep enough to briefly detect the ground as an obstacle. In the obstacle minimizing path (red line in Fig. 4), the robot avoided a row of trees by taking a slightly longer route.

Because of a steep hill, there was a higher slope cost in the bottom right quadrant of the cost map. When planning a path through this hill, the robot traversed up the hill, moved along a flat ridge for most of its route, and then down the hill to reach the goal (blue line in Fig. 4). Incorporating all three measures

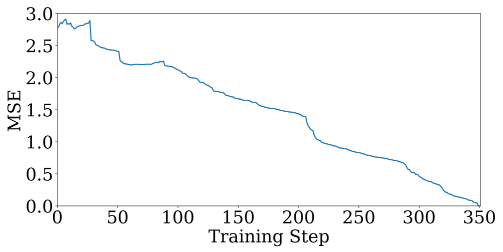


Fig. 5: Mean squared error between the delays D of the model at each training step, during a single 12-hour online learning run, and the final learned costs of the model.

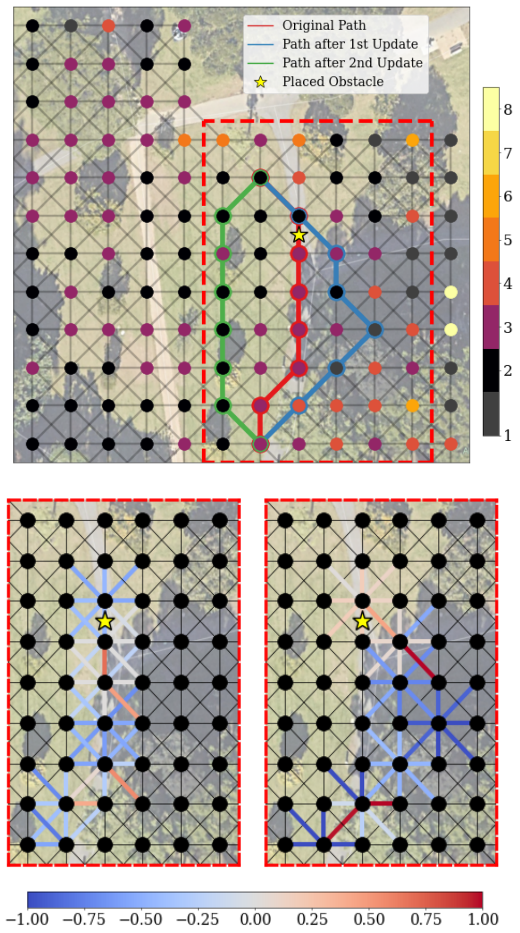


Fig. 6: Top image shows example paths demonstrating adaptation after multiple experiences in a changing environment. Red, green, and blue lines outline the planned path after zero, one, and two updates to the model, respectively. The location of placed obstacles are marked with a yellow star. Bottom images show changes to the model delays after the first (left) and second update (right). Colored edges indicate the extent of the delay change between the two neurons.

of cost into path planning resulted in a path that attempted to avoid the sloped area before traversing along grass that is lower cost than the dirt road where there tends to be obstacles (cyan line in Fig. 4).

TABLE I: Comparison between RRT*, A*, and SWP on paths taken by robot.

Planner	RRT*	A*	SWP (ours)
Path Length	60.23*	46.98	46.6
Current drawn	468.74	382.82	415.89
Obstacles encountered	4.45*	3.95	2.06
Slope	7.52	7.72	7.57
Normalized Cost	24.63	21.83	21.67

** denotes p<0.05; t-test with Bonferonni correction*

B. Continual Learning and Adaptation

Continual learning is realized through E-prop's ability to generate an increasingly accurate map of the environmental costs as the robot explores its environment. Fig. 5 compares the delays of the SWP after each executed path with that of the final learned model during a single 12-hour run with online learning. Mean squared error of each comparison steadily declined, which suggests that each training step is improving upon the last.

Continual learning facilitated rapid adaptation in the face of environmental changes. To demonstrate adaptation, we placed an obstacle in the robot's path. After one or two experiences in this new situation, the robot updated its cost map with this information. Fig. 6 shows the new paths taken by the robot after the first and second model update.

The initial path (shown in red) travels along waypoints on the road, however the presence of the obstacle incurs high obstacle cost, as evidenced by high weight changes at the obstacle location. After a single update, the second path (shown in blue) avoids the obstacle but travels through a sloped hill area to do so, incurring a higher slope cost. This can be seen from the high weight changes when traversing up or down the hill. With one final update, the final path (shown in green) then travels away from the road along flat grass to reach the goal, avoiding both the placed obstacle and the hill.

C. Comparisons with Existing Path Planning Algorithms

In tests with the physical robot in paths in the environment (Table I), the SWP planned significantly shorter paths and minimized obstacle costs better than RRT*. Both planners performed similarly on other costs. A* was most comparable to the SWP, as neither path length nor measures of cost were significantly different between the two algorithms. The lack of significance in many of these metrics could be due to not enough sample routes, which we address by simulating more paths (see Table II), and showing that our SWP has significant advantages as routes get longer (Fig. 7).

To overcome the small sample size, we simulated all possible paths of length greater than 3 (n = 57086) on the learned costmap itself. The results of this are shown in Tab. II. The SWP significantly outperformed RRT*, A*, and D* Lite on minimizing cost. Additionally, paths generated by the Spiking Wavefront Planner were significantly shorter than its comparisons. D* Lite and A* performed similarly, likely due to the similarity in planning using a heuristic and priority

TABLE II: Comparison between RRT*, A*, a Naive planner, D* Lite, and SWP on simulated paths.

Planner	RRT*	A*	Naive	D* Lite	SWP
Path Length	59.01*	51.73*	47.73*	53.60*	50.91
Current Cost	22.94*	20.17	22.13*	21.08*	20.08
Obstacle Cost	14.21*	13.21*	15.44*	13.00*	12.80
Slope Cost	25.63*	22.19	23.79*	22.78*	22.17
Normalized Cost	22.07*	19.55*	21.85*	20.01*	19.36

** denotes p<0.05; t-test with Bonferonni correction*

TABLE III: Comparison between the SWP and D* Lite on simulated paths with a changed environment.

Planner	D* Lite	SWP (Ours)
Path Length	43.15*	44.80
Current Cost	17.28*	17.01
Obstacle Cost	11.74*	12.06
Slope Cost	19.34*	18.98
Normalized Cost	16.94	16.78

** denotes p<0.05; t-test with Bonferonni correction*

queue. The paths were longer than the naive planner, however this was expected as the naive planner travels the shortest euclidean distance regardless of cost.

Fig. 7 illustrates how the comparison between algorithms changes over increasingly longer path lengths. The SWP performed the best, and RRT* performed the worst in terms of cost. As the paths got longer, the disparity between algorithm performance became more pronounced.

To evaluate our model against dynamic planners such as D* Lite, we simulate paths through our costmap where the robot can observe costs one step ahead. We randomly selected start and end points where both the SWP and D* Lite initially planned the same path (n = 10000). Environmental change is simulated midway by elevating a random cost measure between two waypoints by 3.0. When observed by our model, E-prop is run to determine a new path from the current location to the goal. Results in Tab. III show that D* Lite paths were shorter and minimized obstacle costs, whereas the SWP minimized slope and current. The D* Lite comparison was tested in simulation because cost measures, such as current measured by power consumption and slope measured by IMU, can only be determined by physically traversing between waypoints. This means that sensing the costs of waypoints ahead of time, a necessary component of replanning for D* and RRTX methods, was not possible.

A time-complexity analysis of our model and its comparisons are shown in Fig. 8. The analysis was conducted on a Dell XPS 15 9510 laptop, with an Intel Core i7-11800H processor and 16GB of DDR4 RAM. A* and D* Lite outperform both the SWP and RRT* across all lengths, and grows at a much slower rate. The SWP performs better than RRT*, which suffers from high variance due to its random nature. As discussed earlier, the SWP has the potential for highly parallel implementations on conventional hardware and on neuromorphic computers, which would dramatically reduce computation time and power consumption.

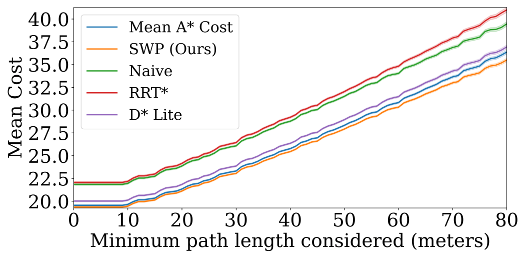


Fig. 7: Difference in performance for each method as the length of simulated paths is increased. Y-axis is the mean cost of the paths. X-axis is the minimum path length considered. Shaded areas represent 99% confidence interval.

VII. Conclusions and Future Work

The SWP with E-Prop learning can rapidly learn traversal costs for navigation and can adapt to change without lengthy offline retraining. It demonstrated shorter and more cost effective paths than other path planning algorithms. The cost maps learned through E-prop may be utilized with a number of existing path planning algorithms, including A*, D*, and RRT*. Although A* may perform similarly with a tailored heuristic, this requires careful consideration of how to estimate the future measure of cost at a given location. Moreover, a different heuristic may be necessary for each cost or combination of costs. As evidenced by the growing disparity of performance in Fig. 7, an improper heuristic may result in worse performance as longer paths are required. By contrast, the SWP can be universally applied to a costmap regardless of what combination of costs are considered.

Our navigation system is not without its limitations. Currently, costs are obtained through experience only and cannot generalize between waypoints or environments, or observed ahead of time. Depending on the learning rate, it may take multiple passes between the same two waypoints to properly learn an accurate cost. It has also been shown here and in previous work that the SWP is computationally slower than A* [8].

These limitations serve as avenues for future research. Our current implementation does not use vision, however computer vision techniques for self-labeling such as those found in [26], [29] could be used to estimate the cost of current and nearby trajectories during traversal. It has also been shown that incorporating biologically-inspired memory replay can improve exploration speed and adaptation to changes in the environment [32].

While our model was tested on traditional hardware, the spiking nature of our model enables the possibility of implementation on neuromorphic hardware. Architectures, such as Intel's Loihi [33] and the DYNAP-SE neuromorphic processor [34], support synaptic delays and the updating of synaptic weights given local learning rules, which are key elements for our model. This demonstrates a clear path towards neuromorphic implementation in the future.

In summary, this work demonstrates our efficient navigation system for off-road navigation that learns continuously from interaction with its environment in real-time, without the need

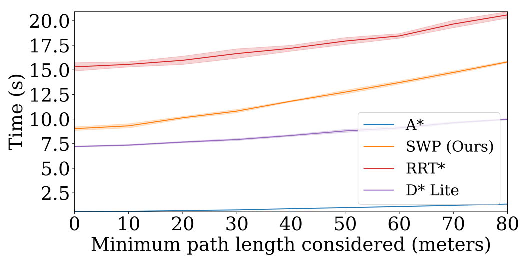


Fig. 8: Time elapsed for path planning algorithms as a function of simulated path length. Y-axis is the mean cost of the paths. X-axis is the path length considered. Line and shaded area represent the mean and standard deviation from 5 runs of 1000 randomly selected paths.

for multiple rounds of training and deployment or expensive hardware. The system learns multiple measures of cost in parallel, and can plan paths that minimize such costs or any combination of them when traversing the environment. Through our real world and simulation results, we determined that these paths are shorter and more cost effective than A* and RRT*. The simplicity of the software stack supports future development, perhaps using context from a camera, memory replay [32], more robust obstacle detection and avoidance, and neuromodulation [35] to control combining costmaps.

References

[1] H. Qin, S. Shao, T. Wang, X. Yu, Y. Jiang, and Z. Cao, "Review of autonomous path planning algorithms for mobile robots," *Drones*, vol. 7, no. 3, 2023.

[2] X. Xiao, B. Liu, G. Warnell, and P. Stone, "Motion planning and control for mobile robot navigation using machine learning: a survey," *Autonomous Robots*, vol. 46, p. 569–597, Mar. 2022.

[3] C. S. Tan, R. Mohd-Mokhtar, and M. R. Arshad, "A comprehensive review of coverage path planning in robotics using classical and heuristic algorithms," *IEEE Access*, vol. 9, pp. 119310–119342, 2021.

[4] M. Davies, A. Wild, G. Orchard, Y. Sandamirskaya, G. Fonseca Guerra, P. Joshi, P. Plank, and S. Risbud, "Advancing neuromorphic computing with loihi: A survey of results and outlook," *Proceedings of the IEEE*, vol. PP, pp. 1–24, 04 2021.

[5] S. Furber, F. Galluppi, S. Temple, and L. Plana, "The spinnaker project," *Proceedings of the IEEE*, vol. 102, pp. 652–665, 05 2014.

[6] K. D. Fischl, K. Fair, W.-Y. Tsai, J. Sampson, and A. Andreou, "Path planning on the truenorth neurosynaptic system," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–4, 2017.

[7] A. Rostami, B. Vogginger, Y. Yan, and C. G. Mayr, "E-prop on spinnaker 2: Exploring online learning in spiking rnns on neuromorphic hardware," *Frontiers in Neuroscience*, vol. 16, 2022.

[8] T. Hwu, A. Y. Wang, N. Oros, and J. L. Krichmar, "Adaptive robot path planning using a spiking neuron algorithm with axonal delays," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 10, no. 2, pp. 126–137, 2018.

[9] J. L. Krichmar, N. A. Ketz, P. K. Pilly, and A. Soltoggio, "Flexible path planning through vicarious trial and error," *bioRxiv*, 2021.

[10] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[11] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pp. 3310–3317 vol.4, 1994.

[12] S. LAVALLE, "Rapidly-exploring random trees : a new tool for path planning," *Research Report 9811*, 1998.

[13] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How, "Real-time motion planning with applications to autonomous urban driving," *IEEE Transactions on Control Systems Technology*, vol. 17, no. 5, pp. 1105–1118, 2009.

[14] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[15] M. W. Otte and E. Frazzoli, "Rrtx: Asymptotically optimal single-query sampling-based motion planning with quick replanning," *The International Journal of Robotics Research*, vol. 35, pp. 797 – 822, 2016.

[16] R. C. Smith and P. Cheeseman, "On the representation and estimation of spatial uncertainty," *The International Journal of Robotics Research*, vol. 5, no. 4, pp. 56–68, 1986.

[17] S. Thrun and M. Montemerlo, "The graph slam algorithm with applications to large-scale mapping of urban structures," *The International Journal of Robotics Research*, vol. 25, no. 5-6, pp. 403–429, 2006.

[18] R. Mur-Artal, J. Montiel, and J. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, pp. 1147 – 1163, 10 2015.

[19] M. Milford, G. Wyeth, and D. Prasser, "Ratslam: a hippocampal model for simultaneous localization and mapping," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 1, pp. 403–408 Vol.1, 2004.

[20] S. Triest, M. G. Castro, P. Maheshwari, M. Sivaprakasam, W. Wang, and S. Scherer, "Learning risk-aware costmaps via inverse reinforcement learning for off-road navigation," 2023.

[21] A. Shaban, X. Meng, J. Lee, B. Boots, and D. Fox, "Semantic terrain classification for off-road autonomous driving," in *Proceedings of the 5th Conference on Robot Learning* (A. Faust, D. Hsu, and G. Neumann, eds.), vol. 164 of *Proceedings of Machine Learning Research*, pp. 619–629, PMLR, 08–11 Nov 2022.

[22] J. A. Placed, J. Strader, H. Carrillo, N. A. Atanasov, V. Indelman, L. Carlone, and J. A. Castellanos, "A survey on active simultaneous localization and mapping: State of the art and new frontiers," *IEEE Transactions on Robotics*, vol. 39, pp. 1686–1705, 2022.

[23] F. Codevilla, M. Müller, A. Dosovitskiy, A. M. López, and V. Koltun, "End-to-end driving via conditional imitation learning," *CoRR*, vol. abs/1710.02410, 2017.

[24] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," *CoRR*, vol. abs/1604.07316, 2016.

[25] S. Ross, N. Melik-Barkhudarov, K. Shankar, A. Wendel, D. Dey, J. Bagnell, and M. Hebert, "Learning monocular reactive uav control in cluttered natural environments," *Proceedings - IEEE International Conference on Robotics and Automation*, 11 2012.

[26] G. Kahn, P. Abbeel, and S. Levine, "BADGR: an autonomous self-supervised learning-based navigation system," *CoRR*, vol. abs/2002.05700, 2020.

[27] T. Manderson, S. Wapnick, D. Meger, and G. Dudek, "Learning to drive off road on smooth terrain in unstructured environments using an on-board camera and sparse aerial images," 2020.

[28] K. Zhang, F. Niroui, M. Ficocelli, and G. Nejat, "Robot navigation of environments with unknown rough terrain using deep reinforcement learning," *2018 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 1–7, 2018.

[29] D. Shah and S. Levine, "Viking: Vision-based kilometer-scale navigation with geographic hints," in *Robotics: Science and Systems Foundation*, Jun 2022.

[30] G. Bellec, F. Scherr, A. Subramoney, E. Hajek, D. Salaj, R. Legenstein, and W. Maass, "A solution to the learning dilemma for recurrent networks of spiking neurons," *Nature Communications*, July 2020.

[31] X.-S. Yang, "Random walks and optimization," in *Nature-Inspired Optimization Algorithms*, ch. 3, pp. 45–65, Academic Press, 2014.

[32] H. Espino, R. Bain, and J. L. Krichmar, "Selective memory replay improves exploration in a spiking wavefront planner," in *2023 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2023.

[33] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[34] F. Sandin and M. Nilsson, "Synaptic delays for insect-inspired temporal feature detection in dynamic neuromorphic processors," *Frontiers in Neuroscience*, vol. 14, 2020.

[35] J. Xing, X. Zou, and J. L. Krichmar, "Neuromodulated patience for robot and self-driving vehicle navigation," in *2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2020.