

Biologically Inspired SLAM

Brent Komer¹

Abstract—This report demonstrates an extension to the RatSLAM algorithm that uses a spiking neural network. This is a replacement of the pose cell network in RatSLAM with a more biologically plausible one that performs path integration and receives visual input to correct for errors. The network is constructed using Nengo and the neuron model used facilitates comparisons between the artificial system and real neural recordings from animals. Three different network architectures are explored and simulations are performed using the *iRat 2011* dataset. The final system is successfully run on the SpiNNaker neuromorphic hardware platform and is able to produce a map of the environment that is similar quality to one produced by the original RatSLAM algorithm.

I. INTRODUCTION

This report presents a SLAM system that runs using a spiking neural network. This system is built upon the RatSLAM algorithm [1] and only uses sensors that are available to animals (vision and proprioception).

A. Motivation

The primary motivation for this work is an interest in how humans and other animals perform SLAM. Humans are much better than robots at understanding their environment in most situations, so learning more about how the brain works could lead to advances in robotics. A first step is to build a system that mimics the ways a brain could perform the SLAM task. From there it is possible to refine the system when new neural data is discovered as well as track the progress on benchmark robotics tasks as more neural detail is added. A spiking neural network is an ideal architecture because it facilitates comparisons of the artificial system to real neural systems. Properties of the artificial neurons can be modeled after real neurons in the brain area of interest (firing rate, connectivity, synaptic time constants, etc).

The brain is also very energy efficient and only uses about 20W of power. Employing the kinds of computations and heuristics that the brain uses could produce algorithms for robotic control that are computationally efficient and can be carried out in real-time.

The two goals that this project strives towards simultaneously are improvements to models of the brain and improvements to robotics.

B. Applications

One of the main applications for this work is a tool for understanding the mechanisms involved in spatial representation and navigation in animals. Since the system is designed

using a spiking neural network, spike recordings obtained in the artificial system can be compared to what neuroscientists record in a real animal, creating a model that researchers can use to make predictions about various cognitive functions or behavioural experiments. The model can be tuned to match experimental animal recordings and then subsequently used in traditional SLAM tasks.

Another advantage of using a spiking neuron architecture is that the system can be integrated into other existing spiking neural architectures. For example, the Spaun [2] model could be enhanced to include spatial tasks in addition to the cognitive tasks it already performs.

The spiking neural architecture also allows the system to be run on neuromorphic hardware. This is specialized hardware designed to run these kinds of neural networks very efficiently and quickly. There are many different architectures currently being developed, with some of the more popular ones being SpiNNaker [3], TrueNorth [4], and Neurogrid [5]. The properties that all of these hardware platforms have in common is low power consumption, highly parallel processing, and guaranteed real-time performance (if the model is able to be loaded onto the system). In particular the low power consumption and real-time performance are very useful for robotic applications.

C. Related Work

This project is a direct extension of RatSLAM [1]. The original RatSLAM algorithm is inspired by the brain but is not implemented with a neural network or any constraints based on the properties of neurons. The authors more recently released an open source and modular version of the algorithm with the hopes that other researchers will be able to build upon it and incorporate more biologically realistic modules [6].

A controlled attractor model of path integration using spiking neurons is described in [7]. Due to the level of detail provided by the neuron model used, this work produced predictions of the neural firing patterns of grid cells [8] and the relationship between head direction and place cells which was later confirmed experimentally [9].

The first use of a spiking neural network with neuromorphic hardware for a spatial task is [10]. This is a proof-of-concept for using biologically plausible hippocampal models on a robot. A set of ring oscillators of different frequencies is used to encode position in space. This robot operates in a simple environment with only two predetermined places that it knows about and does not use a visual system or produce a map.

¹Brent Komer is with the Computational Neuroscience Research Group, Department of Systems Design Engineering, University of Waterloo, N2L 3G1, Canada bjkomer@uwaterloo.ca

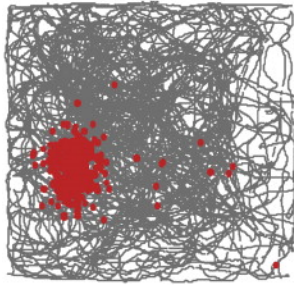


Fig. 1: Place Cell

The gray lines indicate the trajectory of a rat throughout a square environment. Red dots indicate locations where a particular place cell was active. Figure from [13]

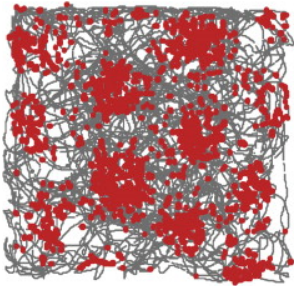


Fig. 2: Grid Cell

The gray lines indicate the trajectory of a rat throughout a square environment. Red dots indicate locations where a particular grid cell was active. Figure from [13]

II. BACKGROUND

A. The Brain

The area of the brain that is widely considered to play a major role in spatial navigation is the hippocampus [11]. Neurons have been observed in this area that are tuned to specific areas of space or landmarks. A ‘place cell’ is a type of neuron that fires strongly when the animal is located within a particular region of space, called the ‘place field’ of the cell, and does not fire when the animal is outside this region [12]. The place fields of every cell is different and can differ in size, with boundaries between the place fields typically occurring with visual changes to the environment (such as entering a new room or turning a corner in a passageway). Often each cell has only one particular place in the environment for which it responds, but for some larger environments one cell could respond to more than one distinct place. An example of the firing pattern of this type of cell is shown in Figure 1.

Another important type of cells that are found in a nearby region are ‘grid cells’ [14]. These cells fire at regular spatial intervals as an animal moves throughout its environment. When plotting the firing of a particular cell as a function of location, a clear grid pattern emerges, as can be seen in Figure 2. This grid exists at different spatial scales and phases for different grid cells and is thought to play an important role in path integration. From using just the recorded spike data it is possible to reconstruct the trajectory of the animal.

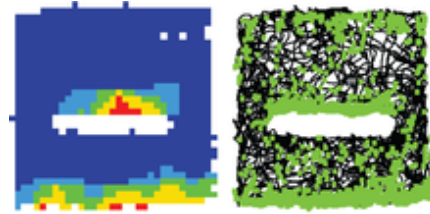


Fig. 3: Boundary Vector Cell

Response of a boundary vector cell (also called a border cell) in a rat while traversing a square environment with a wall in the middle. The figure on the right shows the trajectory in black, and locations where the particular cell was active as green squares. The figure on the left is a heat map of that activity. Figure from [16]

A third type of cell related to spatial processing are ‘boundary vector cells’ [15]. These cells fire when the animal is at a particular distance and direction from a wall or other obstacle. An example of the firing pattern for this type of cell is shown in Figure 3.

Most neural recordings of the hippocampus are gathered from rodents, but it is hypothesized that similar functionality exists in humans and other mammals.

B. RatSLAM

RatSLAM is a SLAM algorithm inspired by the rat hippocampus [1]. The key idea behind this algorithm is using a competitive attractor network to combine odometry information (analogous to grid cells) and landmark sensing (analogous to place cells) to form a representation of the environment. Keeping in line with the senses that are available to animals, this algorithm only requires vision as input and can run with a low quality monocular camera. Odometry inputs can optionally be provided to the algorithm (analogous to proprioception in animals) but an odometry signal is estimated from vision if not present. There are four main modules in this algorithm, and they are each implemented as their own ROS node in the OpenRatSLAM project [6]. A diagram of the interaction between these modules is depicted in Figure 4.

The Experience Map contains the representation for the map created by the RatSLAM system. It contains nodes covering a discretized space that the robot has been to, and edges connecting the nodes forming paths.

The Posecell Network is the attractor network that represents the belief about the current pose. It is a three dimensional network of cells with indices representing the x , y , and θ (heading) state variables. The connectivity of the network is such that it wraps around at the ends, forming a 3D toriod structure. Each cell in the network is connected to itself and its neighbours with a strength depending on the distance. The profile of the connection strength is defined by a difference of Gaussians (Mexican hat) function. Cells close by have a large positive connection strength, cells a little further away have a negative connection strength, and cells even further away have a connection strength decaying to zero. A velocity signal shifts these connection weightings to produce an off-center difference of Gaussians, causing the

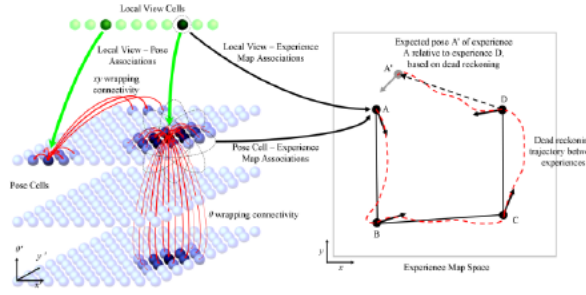


Fig. 4: RatSLAM Modules

Graphical depiction of the RatSLAM modules and connections between them. Figure taken from [6]

location of the primary bump of activity to move. Each node in the Experience Map is associated to the cell in the Posecell Network that had the highest activity when that node was created.

Computation in the Posecell Network is carried out in six sequential stages. The first is local excitation, where energy is added around each pose cell using a Gaussian kernel. This is followed by local inhibition, where energy is subtracted from around each pose cell using a different Gaussian kernel. The next stage is global inhibition, where an equal amount of energy is removed from all pose cells and any cell that goes below zero is set to zero. The system is then normalized so that the sum of all energy in the system is one (to ensure stability). Next the odometry information is used for path integration, shifting the energy in the system. Finally the centroid of the dominant activity packet is found and is used in conjunction with the Local View Cell information to determine how to update the Experience Map.

The Local View Cells contain representations of landmarks and are associated to both locations in the Posecell Network and nodes on the Experience Map. Whenever a visual input that is sufficiently different from any other view cell occurs, a new cell is created. The cell itself stores a downsampled grayscale version of the image which is used for comparing the similarity. The matching threshold and amount to downsample are tunable parameters. When a visual input that matches a previous view cell occurs, energy is injected into the Posecell Network at the locations associated to that view cell. Connection strength between view cells and pose cells is strengthened by Hebbian learning. Details on how this mechanism works is described in [1].

Visual Odometry is the optional node that estimates translational and rotational velocity from a series of images.

C. Neural Simulation

The Neural Engineering Framework (NEF)[17] provides a means of representing arbitrary vectors using the properties of neurons as a basis. This is done through a nonlinear encoding mechanism carried out by the tuning curves of the neurons, and a weighted linear decoding of the responses of the neurons to retrieve an approximation of the vector being encoded. Tuning curves are a characterization of the response

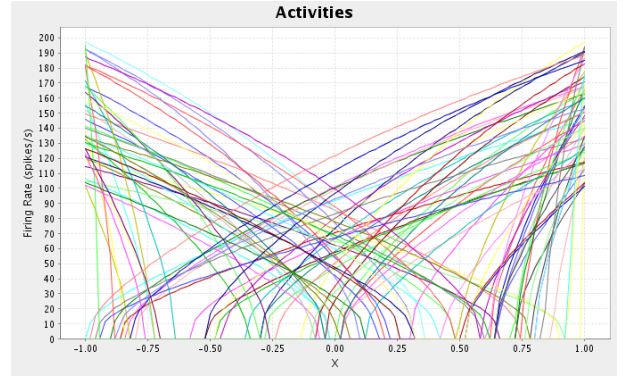


Fig. 5: Tuning Curves

Example set of tuning curves for an ensemble of neurons. Each line represents the response properties of a particular neuron (firing rate) to the state variable X . Each neuron has a preferred direction in which it fires maximally. In the 1D case this is either +1 or -1, but in general the preferred direction can be any vector along a unit hypersphere. The maximum firing rates of the neurons exist within some distribution, which in this case is set to be uniform between 100 and 200 spikes per second. Figure produced using Nengo. [21]

of a group of neurons to some stimuli. A visualization of tuning curves for a group of neurons for a one dimensional input is shown in Figure 5. A transformation can be applied to the underlying representation by specifying different weights on the linear decoding. Any computable function can be approximated through a transform, and the degree of accuracy of the decoding is dependent on the number of neurons used and the complexity of the function. The neurons themselves are a part of a dynamical system where timing effects and filters across connections play a role in the behaviour of the system. For more detail on the NEF, see [18], [19], [20].

An implementation of the NEF is carried out by the software package Nengo [21]. Nengo is written in Python and contains many tools for building biologically plausible neural networks. The type of neuron model used is flexible and different neuron types can be used in different parts of the same network, but the most common type in Nengo models is the Leaky Integrate and Fire (LIF) neuron [22]. This is a spiking neuron model that is fast to compute while still providing a reasonable functional approximation of a real neuron.

In Nengo, groups of neurons that represent a particular state variable are called an ensemble. The main parameters to an ensemble are the number of neurons and the dimensionality of the state being represented. Connections can be defined from one ensemble to another or recurrently from an ensemble back to itself. A transformation matrix or a function to approximate can be passed to the connection. This matrix or function is typically defined in the state space of the vector being represented by the neurons, rather than the neuron state space. Since the decoding is linear, a least squares optimization is carried out by Nengo to determine the optimal decoding weights for the connection to approximate the desired function. These weights can also be learned over

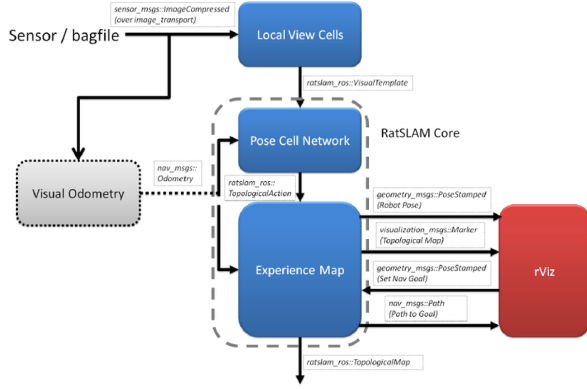


Fig. 6: RatSLAM Node and Message Structure

Organization of ROS nodes and messages in the OpenRatSLAM system. Figure from [6]

time by setting up a learning rule from an error signal in the network. Connections to and from individual neurons within an ensemble can also be specified.

Nodes can be defined that allow the neural network to interface with external inputs or outputs. A ROS node can be defined within a Nengo node to allow the network to communicate with other ROS components.

The script developed using Nengo provides a high level description of the network, which can then be compiled to run on a particular backend, depending on the application. The three most common supported backends are CPU (NumPy), GPU (OpenCL), and SpiNNaker.

III. SPIKING NEURAL NETWORK

The component of the RatSLAM algorithm that is most analogous to function in the brain is the Posecell Network. Since OpenRatSLAM uses ROS it is possible to replace the Posecell Network with a spiking neural network model and still interface with the rest of the system. A diagram of the node and message structure of RatSLAM is shown in Figure 6. The various network configurations that were tried are described in the following sections.

A. Gaussian Weighted Connections

The first approach is the closest to a direct translation of the RatSLAM code to a spiking neural network. Here a small 1D ensemble of neurons is defined for each cell in the 3D pose cell network. Each of these cells is connected to its neighbours with the connection strength defined by a difference of Gaussians function. The velocity signal influences these weightings to produce an offset difference of Gaussians, with the offset proportional to the velocity vector. The evolution of the state of this network can be thought of as convolving the network activity with a difference of Gaussians kernel at every time step, with the mean of the Gaussians determined by the current velocity. Normalization of the activity of the network is also required to prevent the system from becoming unstable. A diagram of the connectivity of the network is shown in 7.

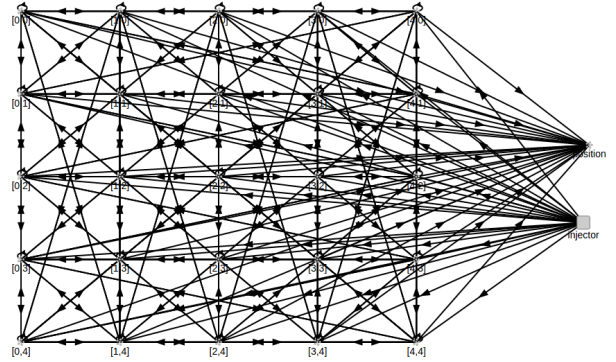


Fig. 7: Gaussian Weighted Connections Network Diagram

Visualization of the Nengo network structure for a 5x5x1 pose cell grid with Gaussian weighted connections. The size of the pose cell network in OpenRatSLAM for this task is 11x11x36 but a network of that size is difficult to display. The position ensemble computes the indices of the maximally active pose cell ensemble, and the injector node is used to inject energy into the system based on the view cells. Not shown here, but an additional ensemble is needed to compute the normalization of the network as well as a method for updating the connection weights based on the velocity signals.

B. Function Space Representation

The second approach defines the state as a Gaussian function centered on the location of the pose cell with maximum activity. The variance of the Gaussian along each axis represents the uncertainty of that pose estimate for each state variable. Instead of using neurons to represent the activation of individual pose cells, the neurons collectively represent the function space itself. This is done by defining a set of Gaussian bases to cover the space and the neural activity will represent the coefficients on those bases. In order to prevent discontinuities at points in the space where the representation wraps around, the basis functions used are periodic Gaussians, with the period in a particular dimension equal to the length of the pose cell network in that dimension.

One of the difficulties of this approach is choosing an appropriate number of bases to use. A large number of bases increases the computation time required to build and simulate the network as well as increases the number of neurons required to approximate the coefficients on those bases effectively, which in turn also slows down the simulation. Too few bases means that parts of the space are not adequately represented, leading to degradation of performance when the state enters those regions. As the dimensionality of the state increases, an exponentially many more bases are required to cover the space with the same degree of accuracy.

There is evidence in the brain that the signals that represent head direction come from a different area than the signals representing position in space [23]. Keeping in line with this distinction and in contrast to the original RatSLAM system, one ensemble of neurons is used to represent θ and a separate ensemble represents both x and y . Splitting the state representation in this manner allows less basis functions to be required and reduces the computation time. The bases themselves are randomly chosen from a uniform distribution

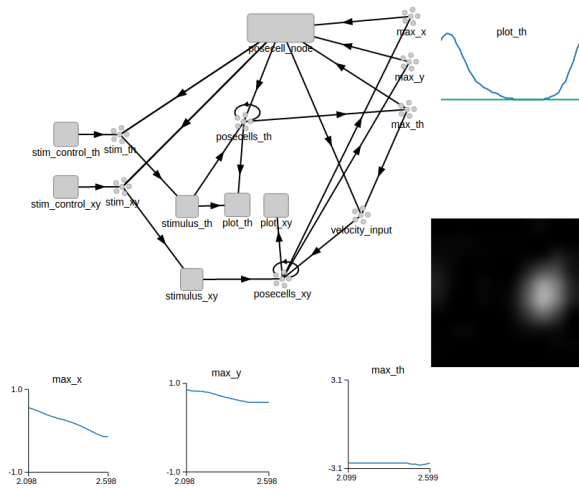


Fig. 8: Function Space Network Diagram

Diagram of the function space network in Nengo. Nodes are depicted as boxes, and ensembles as groups of circles. The *posecell_node* handles communication to and from the other ROS nodes. The *posecells_xy* ensemble contains the neurons representing the 2D Gaussian bases for the x - y location, with the decoded representation displayed on the image on the right. The *posecells_th* ensemble contains the neurons representing the 1D Gaussian bases for θ , with the decoded representation displayed on the plot in the top right. Each of the *max* ensembles represents the maximally active position for each state variable (mean of the Gaussian). The *stim* ensembles are used to inject energy into the system based on the view cells, and the *velocity_input* ensemble contains the current translational and rotational velocity. The other nodes are utilities for managing and plotting the function space representation.

covering the space.

The effect of the velocity input on the basis coefficients is nonlinear so in order for the neurons to approximate this relationship the velocity information must be present in the same ensemble as the bases. For example, if 50 bases are used to represent the x and y position then the resulting neural ensemble will be 52 dimensional. The first 50 dimensions correspond to the coefficients on each 2D periodic Gaussian basis function, and the last two dimensions correspond to the x and y velocity. The recurrent connection from the ensemble back onto itself is defined in such a way that the mean of the Gaussian reconstructed from the bases will move over time proportional to the velocity. A visualization of the Nengo network along with some plots during a simulation is shown in Figure 8.

C. Cyclic Integrator

The third approach uses neural integrators to store the state variables. A neural integrator is an ensemble of neurons that computes the integral of its input signal. This is achieved by creating a recurrent connection from the ensemble back onto itself and weighting the input connection to the ensemble with a value equivalent to the synaptic time constant of the neurons. This allows the ensemble to maintain a stable representation of a value over time (memory) and any external input from other neurons is integrated into this memory. Since individual spiking neurons have a maximum

firing rate, the magnitude of the value being represented by the neurons is limited (based on the number of neurons and the desired accuracy of the represented value). There will be a point where in order to represent a value on one of the extremes, each neuron is either firing maximally or not firing at all and driving the system harder will not cause a change in the neural activity (cannot fire less than zero and cannot fire more than the maximum), meaning the represented value will not become larger.

Luckily this saturation effect is not a limitation for the RatSLAM system since the state being represented by the pose cells wraps around at particular values, keeping it bounded within a fixed region at all times. In order to get this wrapping functionality, the recurrent connection of the integrator can be modified to approximate the use of a modulus operator. The problem with this approach is that the sharp discontinuity at the point where the value wraps around is difficult for neurons to approximate, leading to a degradation of the signal in that region. One solution to this is to redefine the value to be represented as a point on a unit circle rather than a scalar. Integration can be achieved by moving around the circle in a particular direction and defining the state variable as the angle to that point which can be decoded from this representation by using the arctangent of the current x - y position. Using this underlying representation, wrapping is smooth and spiking neurons can do a good job of approximating this functionality with no discontinuities.

To keep things simple, a separate integrator is defined for each of the x , y , and θ state variables. The rotational velocity is used as the input to the θ integrator. Another ensemble of neurons takes as input the translational velocity as well as the current heading angle decoded from the θ integrator to produce signals for the x and y components of the velocity to be fed into their respective integrators. There are also additional connections to the integrators to account for energy being injected into the system from the view cells. These connections push the state of the integrators towards the corresponding point on the circle with a magnitude proportional to the energy being injected. A visualization of the Nengo network structure along with some plots during a simulation is shown in Figure 9.

IV. SIMULATIONS

A. Dataset

Simulations were completed using the *iRat 2011* dataset, which is a ROS bagfile recorded from a small rat-like robot moving around in a constructed indoor environment [24]. Any dataset that provides images that can be sent as a ROS message could be used, but this dataset in particular was chosen because it was used with the OpenRatSLAM system, allowing for direct comparison of performance to the spiking neuron implementation. The scale of the motions recorded in this dataset are also similar in magnitude to those a rat would experience, meaning that any tuning of the parameters of the algorithm to work on this dataset would also be helpful in simulating rat data.

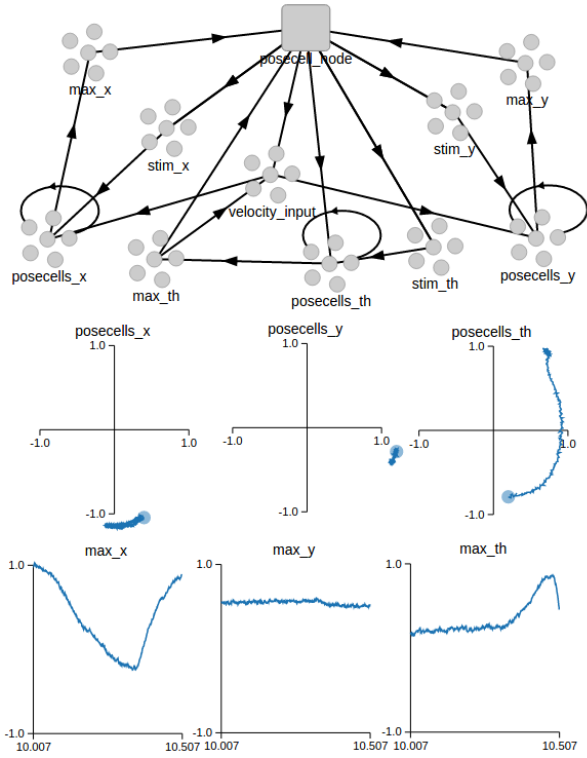


Fig. 9: Cyclic Integrator Network Diagram

Diagram of the cyclic integrator network in Nengo. The *posecell* node handles communication to and from the other ROS nodes. Each of the *posecells* ensembles is a cyclic integrator representing a particular state variable. The decoded representations are shown in the middle set of plots. The *max* ensembles represent the scalar state variable which is decoded directly from the corresponding *posecells* ensemble. It is possible to remove these ensembles and just connect the *posecells* ensembles to the *posecell* node directly with the appropriate transform, but they are left in the model for clarity. Note that the plots in the middle and bottom are produced by the same neural activity, only the linear decoding weights are different. The *stim* ensembles are used to inject energy into the system based on the view cells, and the *velocity* input ensemble contains the current translational and rotational velocity.

The dataset contains approximately 16 minutes of video of the robot moving through the environment depicted in Figure 10. Each image comes from a front facing camera on the robot; an example image is shown in Figure 11.

B. implementation

Before making the pose cell network in Nengo, an implementation in pure Python is created and compared to the C++ version from OpenRatSLAM. This is used to ensure that no changes to the results occur from using a Python interface to ROS (for example, how the message queue is handled or the speed of processing could potentially have an effect). There is no significant difference in the maps produced, meaning results obtained from the spiking network can be directly compared to benchmarks done on the C++ system.

From here the Python implementation can be updated to include a spiking neural network using Nengo. The interface to ROS is abstracted away from the rest of the network and contained within a single Nengo node. This node accepts a 3 dimensional input (location of maximally



Fig. 10: iRat 2011 Environment Overhead View

The iRat robot is the white triangular object with the red nose. It traverses the area along the dark blue paths.



Fig. 11: iRat 2011 Camera View

active pose cell in x , y , and θ) and produces a 6 dimensional output (translational velocity, rotational velocity, location to inject energy in x , y , and θ and the amount of energy to be injected). The ROS publishers and subscribers for the pose cell network are wrapped within this Nengo node and include the publisher for PoseCell/TopologicalAction (effect on edges and nodes on the experience map), the subscriber for LocalView/Template (information on the currently active view cell), and the subscriber for /odom (linear and rotational velocity). The rest of the Nengo network does not need to know the implementation details of this node, it just needs to connect its inputs and outputs to the node and all processing on those signals to convert them to and from ROS messages is handled within the node. The spikes coming into this node are filtered to produce an estimate of the instantaneous spike rate, which gives a less noisy approximation of the represented value which is used in constructing the ROS messages.

The Gaussian weighted connection method was extremely slow to simulate and contained many biologically unrealistic properties (global normalization, instantaneous weight changes) so it was not tested with the full RatSLAM system.

The function space approach was also quite slow, but a still lot faster than the previous method worked well enough that simulations could be performed. Initial simulations of the network performed quite poorly, the final map is shown in Figure 12a. One potential reason for this is the network could not be run in real-time on a CPU (about 0.15x real-time on average). This is true for most large spiking neural network models as the computations to keep track of all of the neural activity throughout the network at every time step can be intensive for a sequential processor. The timing being

out of line with the rest of the RatSLAM system can cause the performance to degrade, due to visual input and odometry being associated to the wrong parts of the pose cell network. One way to account for this is to play the ROS bag file at a lower rate, to match the speed of the neural network. A map produced using a playback rate of 0.15 is shown in Figure 12b. The map seems slightly better, but still far from perfect. One reason why changing the playback rate won't fix the problem is that the slowdown of the neural network is not consistent. At some points it may be running at 0.2x real-time, while at others it would be 0.1x real-time. The speed of the network is dependent on the intensity of the calculations it needs to do at a particular time step. Instances where neurons are spiking more frequently cause the network to run slower than periods of less spiking. Setting the playback rate to be the slowest time will not work because the network running faster than that rate will also cause issues because the internal dynamics will be accelerated in comparison to the rest of the system. Modifying the instantaneous playback rate of the the bag files to match the current lag in the network is not easy to do. The better alternative is to run the network on the SpiNNaker system to guarantee real-time performance. Unfortunately the network was not able to run on SpiNNaker at this time. The code in Nengo that allows the construction of networks to represent function spaces is very new, still under development and not officially supported. This code caused errors when being compiled to SpiNNaker. One of the reasons why this code is slow in the first place could be due to inefficiencies in the function space code. 8000 neurons were used in this network when producing the plots, and while using a lower number of neurons improved the speed, the representation of the Gaussian activity became noticeably worse.

The cyclic integrator method is able to run much faster than the previous approach (about 0.5x real-time on average). The map created with standard rosbag playback is shown in Figure 12d. This map looks okay despite the time not matching up. One possible reason is that 0.5x speed is still fast enough to process the ROS messages coming in without a backlog building up. The difference in speeds would mostly affect the effective scaling of the velocity signal, and since units in the pose cell network are arbitrary and cyclic, this may not have a negative effect. To test this further, another network is created with the same functionality but using a larger number of neurons (7500 total). Increasing the number of neurons will increase the accuracy of the simulation, but more importantly for this test, decrease the speed. This network ran at about the same speed as the function space method and produced a bad map, as shown in Figure 12c. The output of the 1500 neuron model with a playback rate of 0.5 is shown in Figure 12e, and on SpiNNaker with full playback speed in Figure 12f.

For comparison, the map created with the pure C++ OpenRatSLAM system is shown in Figure 12g and the map created using only odometry information and no pose cell network in Figure 12i. The use of a pose cell network shows a clear improvement from odometry-only mapping as loop

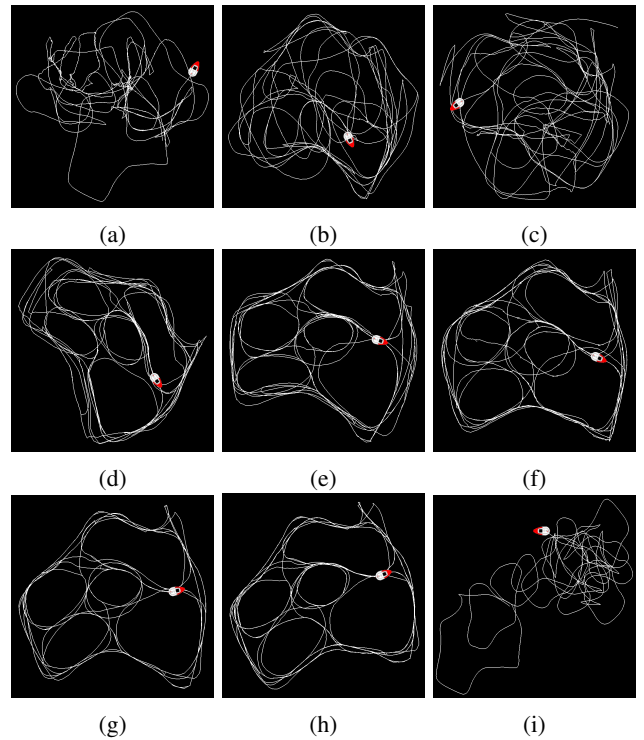


Fig. 12: Experience Map Comparisons

Experience map created after one iteration through the dataset with different posecell network implementations. **a**: function space network. **b**: function space network with 0.15x playback rate. **c**: cyclic integrator network with a large number of neurons **d**: cyclic integrator network. **e**: cyclic integrator network with 0.5x playback rate. **f**: cyclic integrator network run on SpiNNaker. **g**: unmodified OpenRatSLAM implementation. **h**: Python posecell network without neurons. **i**: no posecell network, only odometry.

closures can be detected and path integration errors corrected.

V. CONCLUSIONS

A spiking neural network is able to be used as the pose cell network in a RatSLAM system and produce reliable mapping results. There is no noticeable advantage in terms of accuracy in using spiking neurons over a traditional approach (at least for the dataset chosen) but the advantage comes from the ability to make connections to biological data as well as integrate this algorithm into more advanced cognitive systems that use spiking neurons.

The cyclic integrator spiking neural network architecture performs the best out of the ones tested, in terms of both speed and accuracy. The size of the model is small enough that it can be loaded onto a single SpiNNaker board and run in real-time to produce accurate maps.

The main representational limitation of this approach is that it cannot represent and propagate beliefs of different states simultaneously. The two previous methods had some way of maintaining more than one bump of activity (analogous to a particle filter) at least for a small amount of time before one takes over. This ability does not seem to be important for this particular mapping task and the computational savings of the neural integrator approach make it more desirable. For mapping in environments where there

are many rooms that look identical this ability may be more useful.

All of the code for this project is available on GitHub (<https://github.com/bjkomer/spiking-ratslam>).

VI. FUTURE WORK

One interesting area of future work is adding a biologically inspired visual system to the network. A new type of camera has been developed modeled after the retina that can asynchronously detect changes in pixel intensity with high temporal resolution [25]. The asynchronous event driven nature of this camera makes it a good fit to use with the parallel computation that spiking neural networks provide.

A change that would be interesting to explore is combining the cyclic integrators for the x and y position into one neural ensemble that encodes x and y position together. This will likely involve a representation that is a point on the unit hypersphere with two possible axes of rotation for the integration. Since the definition of x and y are arbitrary, this combined representation may be able to better model the kinds of computations that are going on in real brains. In animal studies, grid cells are found to tile the horizontal plane in a hexagonal structure, meaning an orthogonal prior may not be ideal for a brain model.

Oscillations in neural activity have been observed in the hippocampus and are thought to play a role in navigation [26]. Incorporating oscillators into the model in a similar manner as was used in [10] would be a worthwhile extension.

This RatSLAM system could be used with a robot that is made to move around in an environment similar to the ones used with rats in the grid cell experiments. The spikes from the neurons in the artificial system can be recorded and compared to those observed in a real rat to see what are the similarities and differences as well as see if grid cell firing patterns emerge in the artificial system.

Another possible extension is to add a component to the algorithm that is analogous to Boundary Vector Cells. This would allow the system to have a representation of obstacles rather than just paths that have been previously explored. This information could guide decisions for new areas to explore.

Since this spiking neural network is developed using Nengo it can be integrated with other spiking neural networks using the same software. This opens up interesting opportunities to add spatial representation capabilities to a larger cognitive neural system and run the entire system in real-time on neuromorphic hardware.

REFERENCES

- [1] M. J. Milford, G. F. Wyeth, and D. Rasser, "Ratslam: a hippocampal model for simultaneous localization and mapping," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 1. IEEE, 2004, pp. 403–408.
- [2] C. Eliasmith, T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, and D. Rasmussen, "A large-scale model of the functioning brain," *science*, vol. 338, no. 6111, pp. 1202–1205, 2012.
- [3] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown, "Overview of the spinnaker system architecture," *Computers, IEEE Transactions on*, vol. 62, no. 12, pp. 2454–2467, 2013.
- [4] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.
- [5] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, 2014.
- [6] D. Ball, S. Heath, J. Wiles, G. Wyeth, P. Corke, and M. Milford, "Openratslam: an open source brain-based slam system," *Autonomous Robots*, vol. 34, no. 3, pp. 149–176, 2013.
- [7] J. Conklin and C. Eliasmith, "A controlled attractor network model of path integration in the rat," *Journal of computational neuroscience*, vol. 18, no. 2, pp. 183–203, 2005.
- [8] E. A. Zilli, "Models of grid cell spatial firing published 2005–2011," *Front. Neural Circuits*, vol. 6, no. 16, pp. 10–3389, 2012.
- [9] A. P. Maurer, A. W. Lester, S. N. Burke, J. J. Ferng, and C. A. Barnes, "Back to the future: Preserved hippocampal network activity during reverse ambulation," *The Journal of Neuroscience*, vol. 34, no. 45, p. 15022, 2014.
- [10] F. Galluppi, J. Conrad, T. Stewart, C. Eliasmith, T. Horiuchi, J. Tappson, B. Tripp, S. Furber, and R. Etienne-Cummings, "Live demo: Spiking ratslam: Rat hippocampus cells in spiking neural hardware," in *Biomedical Circuits and Systems Conference (BioCAS), 2012 IEEE*. IEEE, 2012, pp. 91–91.
- [11] A. D. Ekstrom, M. J. Kahana, J. B. Caplan, T. A. Fields, E. A. Isham, E. L. Newman, and I. Fried, "Cellular networks underlying human spatial navigation," *Nature*, vol. 425, no. 6954, pp. 184–188, 2003.
- [12] J. O'Keefe and J. Dostrovsky, "The hippocampus as a spatial map. preliminary evidence from unit activity in the freely-moving rat," *Brain research*, vol. 34, no. 1, pp. 171–175, 1971.
- [13] D. Derdikman and E. I. Moser, "A manifold of spatial maps in the brain," *Trends in cognitive sciences*, vol. 14, no. 12, pp. 561–569, 2010.
- [14] T. Hafting, M. Fyhn, S. Molden, M.-B. Moser, and E. I. Moser, "Microstructure of a spatial map in the entorhinal cortex," *Nature*, vol. 436, no. 7052, pp. 801–806, 2005.
- [15] C. Lever, S. Burton, A. Jeewajee, J. O'Keefe, and N. Burgess, "Boundary vector cells in the subiculum of the hippocampal formation," *The journal of neuroscience*, vol. 29, no. 31, pp. 9771–9777, 2009.
- [16] T. Hartley, C. Lever, N. Burgess, and J. O'Keefe, "Space in the brain: how the hippocampal formation supports spatial cognition," *Phil. Trans. R. Soc. B*, vol. 369, no. 1635, p. 20120510, 2014.
- [17] C. Eliasmith and C. H. Anderson, *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. MIT press, 2004.
- [18] C. Eliasmith, "How to build a brain: From function to implementation," *Synthese*, vol. 159, no. 3, pp. 373–388, 2007.
- [19] T. C. Stewart, T. Bekolay, and C. Eliasmith, "Neural representations of compositional structures: Representing and manipulating vector spaces with spiking neurons," *Connection Science*, vol. 23, no. 2, pp. 145–153, 2011.
- [20] C. Eliasmith, *How to build a brain: A neural architecture for biological cognition*. Oxford University Press, 2013.
- [21] T. Bekolay, J. Bergstra, E. Hunsberger, T. DeWolf, T. C. Stewart, D. Rasmussen, X. Choo, A. R. Voelker, and C. Eliasmith, "Nengo: a python tool for building large-scale functional brain models," *Frontiers in neuroinformatics*, vol. 7, 2013.
- [22] A. N. Burkitt, "A review of the integrate-and-fire neuron model: I. homogeneous synaptic input," *Biological cybernetics*, vol. 95, no. 1, pp. 1–19, 2006.
- [23] J. Ranck Jr, "Head-direction cells in the deep cell layers of dorsal presubiculum in freely moving rats," in *Soc Neurosci Abstr*, vol. 10, no. 176.12, 1984.
- [24] D. Ball, S. Heath, G. Wyeth, and J. Wiles, "irat: Intelligent rat animat technology," in *Proceedings of the 2010 Australasian Conference on Robotics and Automation*, 2010, pp. 1–3.
- [25] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128 × 128 120 db 15 μ s latency asynchronous temporal contrast vision sensor," *Solid-State Circuits, IEEE Journal of*, vol. 43, no. 2, pp. 566–576, 2008.
- [26] M. E. Hasselmo, J. Hay, M. Ilyn, and A. Gorchetnikov, "Neuro-modulation, theta rhythm and rat spatial navigation," *Neural Networks*, vol. 15, no. 4, pp. 689–707, 2002.