GRANT: Ground-Roaming Autonomous Neuromorphic Targeter

Jonathan D. Ambrose, Adam Z. Foshie,
Mark E. Dean, James S. Plank, Garrett S. Rose
Department of Electrical Engineering and Computer Science
The University of Tennessee
Knoxville, TN 37996
Email: [jambros2, afoshie]@vols.utk.edu
[markdean, jplank, garose]@utk.edu

J. Parker Mitchell, Catherine D. Schuman
Oak Ridge National Laboratory
Oak Ridge, TN 37830
Email: [mitchelljp1, schumancd]@ornl.gov

Grant Bruer
School of Computational Science and Engineering
Georgia Institute of Technology
Atlanta, GA 30332
Email: gbruer@gatech.edu

***Abstract*—In this work we describe the design, implementation, and testing of the first neuromorphic robot capable of obstacle avoidance, grid coverage, and targeting controlled by the second generation Dynamic Adaptive Neural Network Array (DANNA2) digital spiking neuromorphic processor. The simplicity of the DANNA2 processor along with the TENNLab hardware/software co-design framework allows for compact spiking networks that can run efficiently on a small, resource-constrained, platform such as a Xilinx Artix-7 field-programmable gate array. Additionally, we present the dynamic reconfigurability of DANNA2 arrays as a method of realizing complex, multi-objective tasks on hardware that is restricted to relatively small networks.**

***Index Terms*—Autonomous Robots, Field-programmable Gate Array, Neural Network Hardware, Neuromorphic, Recurrent Spiking Neural Networks, Robot Control**

I. Introduction

In the last three decades, research into autonomous robotics and vehicles controlled by neural networks has been primarily rooted in traditional Artificial Neural Networks (ANNs), with more recent works primarily utilizing Deep Neural Networks (DNNs) [1], [2]. While relatively successful, these types of neuromorphic models often require large networks with thousands of neurons and synaptic connections numbering in the millions to achieve favorable results in real-time applications [3], [4]. When working with small, resource-constrained, embedded robotics platforms, it is difficult to host these networks natively. To overcome this limitation, many of these types of implementations utilize wireless communication to support running the large networks on a separate, computationally powerful host and merely use the robot to provide inputs and execute outputs. In recent times however, research focus has began to shift towards recurrent spiking neural networks (RSNNs) due to efforts in bringing artificial intelligence to small, low power devices [5]. Furthermore, RSNNs have gained popularity for their ability to encode spatio-temporal information as well as their capabilities of being Super-Turing [6], [7]. While many architectures have been designed to support these RSNNs in both hardware and software, little work has been done to see these architectures outfitted onto a real-time, embedded platform.

We show that not only are RSNNs capable of running real-time robotics applications on a resource-constrained platform, but that it is also possible to dynamically reconfigure the hardware to accomplish a larger task space using the same number of neuromorphic elements with negligible delay caused by this reconfiguration in operation. This paper covers the design, implementation, and testing of our robot, GRANT (Ground-Roaming Autonomous Neuromorphic Targeter), the first neuromorphic robot capable of autonomous navigation and targeting using only onboard processing. We briefly describe the Dynamic Adaptive Neural Network Array Two (DANNA2) neuromorphic processor behind the operation of our robot and the TENNLab co-design framework used to train a neural network with evolutionary optimization for it [8]. Additionally, we discuss NeoN, GRANT's neuromorphic predecessor designed for autonomous navigation and grid coverage, and how we expand upon NeoN's capabilities with GRANT. Our physical, digital, and software design choices are explained and the results we observed are analyzed. Finally, we suggest future work that could improve, expand, or build upon the results gained by GRANT.

Notice: This material is based in-part on research sponsored by the Air Force Research Laboratory under agreement number FA8750-19-1-0025 as well as the U.S. Department of Energy, Office of Science, and Office of Advanced Scientific Computing Research, under contract number DE-AC05-00OR22725. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory or the U.S. Government. This manuscript has been authored in part by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

II. Related Work

In this section, we explore work that performs robot control, obstacle avoidance, and target acquisition using related neuromorphic systems. These embedded platforms use a variety of neuromorphic processors and network topologies to attempt tasks such as obstacle avoidance, targeting, path following, and grid coverage.

A. Obstacle Avoidance and Target Acquisition with the ROLLS Processor

In a collaborative effort, groups at the University of Zurich and the Technical University of Munich created a robot that is capable of utilizing wireless communication to a ROLLS neuromorphic processor in order to perform obstacle avoidance and object targeting. While the robot must wirelessly communicate with the neuromorphic processor, the authors theorized that this processor could, in fact, be affixed to a robot, albeit a large one. This robot utilized a dynamic vision sensor (DVS) camera in order to both detect obstacles and the target. Neuronal architectures were manually created to accomplish each of the desired tasks, where some neuron populations were used to represent detected obstacles while others were used to control the motors and their speed. These neuron populations comprised a total neuron count of 96. Furthermore, additional neuron populations were used to add in information about perceived targets, and these populations required an additional 128 neurons. These neuronal architectures were designed independently, so to combine the two behaviors, more weighting was given to obstacle avoidance. This configuration decision meant that the resulting architecture would prioritize avoiding obstacles over pursuing a target [9].

In testing, it was found that the resulting neuronal architecture could avoid obstacles at speeds as high as 0.5 meters per second despite operating in a room with variable obstacles and inconsistent behavior due to the analog components of the ROLLS processor. Object targeting proved more difficult for the robot however, as the neuronal architecture could not adequately discern the difference between an obstacle and the target. In 80% of the test cases, the robot recognized the target as being an obstacle when approaching. In the remaining 20% of test cases, the target was not recognized as being an obstacle. This was an expected result, however, as the vision preprocessing that was performed was not expected to differentiate between a target and an obstacle [9].

B. Trail following using the IBM NS1e

A research endeavor by groups at the University of Maryland and the University of California, Irvine created a robot that could follow a trail utilizing a deep convolutional neural network running on IBM's TrueNorth neuromorphic processor [10]. The groups retrofitted the IBM NS1e, a board containing 4096 TrueNorth cores, to the CARLorado robot, an Android-based robotic platform powered by a Samsung Galaxy S5 smart phone. They created datasets by manually driving the robot along a mountainous trail in Telluride, Colorado, and recording the decisions of the human driver. This was done in four independent runs, and the results were used as a classification dataset for the deep convolutional neural network, where the network classified the given scene as a decision for the robot to turn right, turn left, or continue straight. The neural network was trained to be an energy-efficient deep neuromorphic network, as these networks have been shown to run effectively on TrueNorth. To communicate between the Android phone and the NS1e, a wireless hotspot was used to transmit data. A remote host, however, was not required, as both the Android phone and the NS1e were powered by the robot, and the wireless hotspot was local to these two devices [11].

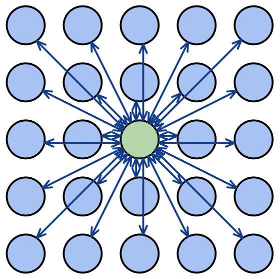
Fig. 1. DANNA2 connectivity pattern

The network that resulted from the training was able to successfully navigate the trail with minimal intervention by the researchers that accompanied it. When the robot veered off of the trail, it had to be manually placed back at the center. The robot successfully traversed the 0.5 kilometer stretch of trail from the dataset, up and down, with this minimal human intervention [11].

C. Previous Works

This work is the latest iteration of autonomous neuromorphic robots created by TENNLab. As such, it directly draws influence from the Neuromorphic Control System for Autonomous Robotic Navigation (NeoN) [12]. NeoN was the first robot created by TENNLab to successfully demonstrate autonomous operation controlled by spiking neuromorphic hardware. NeoN was controlled with the first generation Dynamic Adaptive Neural Network Array (DANNA) neuromorphic processor programmed onto a Xilinx Kintex-7 FPGA. NeoN also used a MicroBlaze core that was implemented on the FPGA to facilitate easy programming and monitoring of the system's inputs and outputs. Obstacle avoidance and grid coverage were both successfully accomplished by NeoN with the use of a 120 degree sweeping LIDAR [12]. As the natural successor to NeoN, GRANT takes advantage of the lessons learned in the development of NeoN and the newest iteration of DANNA neuromorphic processors, DANNA2.

III. DANNA2 Neuromorphic Processor & TENNLab Co-design Framework

The second generation Dynamic Adaptive Neural Network Array, DANNA2, digital spiking neuromorphic processor developed by TENNLab is central to the operation of this

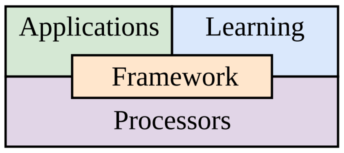

Fig. 2. TENNLab hardware/software co-design framework

robot [13]. The DANNA2 neuromorphic processor is written as generically as possible in VHDL with implementations on a Field-Programmable Gate Array (FPGA) or with Very Large Scale Integration (VLSI) in mind. DANNA2 is made up of a two-dimensional grid of core elements each acting as a leaky-integrate-and-fire neuron with 24 possible synaptic connections. Each synapse connects to one of that core element's 24 nearest neighboring core elements. The connectivity pattern of DANNA2 is shown in Figure 1. Each element outputs a single binary spike from the neuron which propagates to elements with connected synapses. Inputs are provided to the leftmost edge elements of the array and outputs are read from the rightmost edge elements of the array, where input spikes can have four bits of resolution in their weights to represent the input's synaptic strength. The DANNA2 array is given a 100 MHz global clock with element calculations completed every 10 global cycles. This results in a network cycle frequency of 10 MHz which is quite fast compared to other neuromorphic systems that have network cycles that are measured in the kilohertz range [14]. A particularly important property of DANNA2 elements are their reconfigurability. This feature allows for different network configurations to be loaded dynamically. More information about the DANNA2 architecture and its implementation may be found in [15].

DANNA2 networks are trained using a software simulator of the hardware, an application created to model the physical environment that the robot experiences, and evolutionary optimization learning algorithm. The TENNLab hardware/software co-design framework makes this transition from software-trained networks to hardware-realized networks possible [16]. The basic components of the co-design framework are illustrated in Figure 2. The framework, or core, acts as a glue consisting of interfaces and common functions that allow each of the other components to work with one another. The processor component consists of the different neuromorphic processor models, such as DANNA2, that have been developed to run RSNNs. Each processor shares similar functions, but how they are implemented may differ. The applications act as software simulations of the problems that need to be solved. The learning component performs the training that results in neural networks to be loaded onto the processor to implement the application. The main learning methodology is a genetic algorithm called Evolutionary Optimization of Neuromorphic Systems (EONS), but other methodologies, such as backpropagation [17], [18] and reservoir computing [19] are also being incorporated. More detail on the framework may be found in [20].

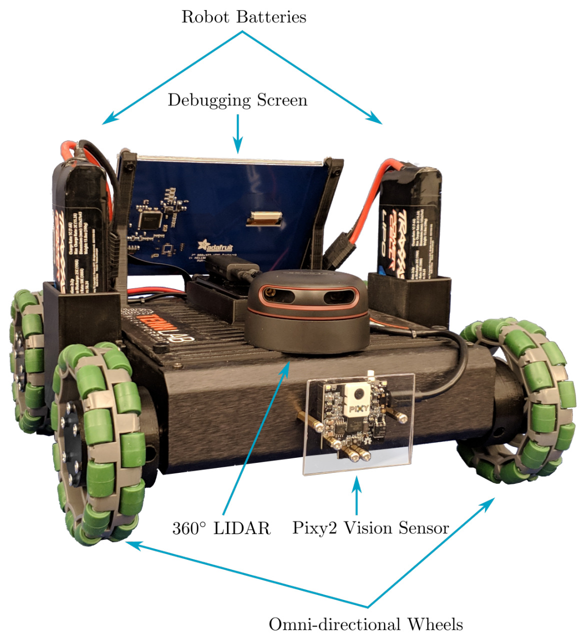

Fig. 3. Front view of an assembled GRANT robot

IV. Design

A fully assembled GRANT robot can be viewed in Figure 3. The robot is built on a lightweight aluminum chassis with omni-directional wheels to allow for effortless, in-place turning. This design choice makes it possible for the network to make left and right turning decisions without a more complex steering geometry.

Four 12 V brushed DC motors with integrated quadrature encoders are used to drive the robot. Each encoder has a resolution of 1920 counts per revolution of the gearbox's output shaft.

A 360 degree rotating RPLIDAR A2 LIDAR is used to scan for and detect obstacles at a maximum frequency of 15 Hz [21]. This LIDAR sensor operates by pulsing a laser up to 8000 times per rotation measuring the time it takes for the laser to reflect back to it after each pulse. Since the LIDAR rotates a full 360 degrees, measurements can be taken at any discrete angle around the robot. In our application, however, we are only concerned with objects within the front 180 degrees of the robot, and thus, we allow components of the robot to block the back 180 degrees.

To perform object targeting, a Pixy2 robot vision sensor is used [22]. The Pixy2 sensor operates at 60 frames per second using a hue-based color filtering algorithm to detect and differentiate between different objects. By using an object with a drastically different color than any of the other objects in the robot's environment, a target object (or objects) can be reliably detected.

To control the robot, a PYNQ-Z1 board with a Xilinx ZYNQ-7000 System on Chip (SoC) is used [23]. This SoC offers an embedded ARM Cortex-A9 processor, such that one can develop standard software utilizing existing libraries for the various sensors while still being able to operate with a digitally designed neuromorphic processor within its FPGA component. This particular SoC was chosen as it offered the smallest FPGA capable of fitting the necessary network size and allowed the demonstration of the performance of the DANNA2 neuromorphic processor on a resource-constrained platform.

A small LCD screen is also included as a debugging and diagnostic tool but is not required for any of the essential operations of the robot.

A. Digital Design

When designing the robot, it was decided that in order to utilize existing USB libraries for each of the sensors, Petalinux would be run on the FPGA [24]. Petalinux is a Linux release built for embedded platforms designed to be run on SoCs such as the PYNQ-Z1 that offer a CPU alongside an FPGA. To begin with this endeavor, a base design was used for both the Petalinux configuration and the corresponding FPGA configuration that was capable of connecting the ARM core to all of the board's offered peripherals properly [25], [26]. The FPGA design was then modified to add the additional inputs and outputs necessary for the robot. PWM IP blocks were added to the project to allow the robot's motors to be controlled by speed controllers. These IP blocks were connected across GPIO blocks to the main AXI-4 interconnect that communicated with the processor. Furthermore, the necessary direction pin for these speed controllers was added to another GPIO block. A hardware quadrature decoding block was also used for each motor's encoder. VHDL code for quadrature decoding provided by Digi-Key was modified to output a 32-bit signed integer and a bit to represent the last direction of turning [27]. These decoding blocks were then attached to GPIO blocks such that the number of counted pulses could be directly accessed by the processor.

The last component of the digital design involved adding in a DANNA2 grid array as the neuromorphic processor. Since grid arrays are reprogrammable, this allows various networks to be used for different desired behaviors without having to reconfigure the hardware project itself. To communicate with this component, an AXI-DMA engine was used. Since the maximum supported bus width of the DMA is 32 bits and the packet width for DANNA2 is 512 bits, AXI width converters were used on the input and output of the array. A FIFO was also used on the output of the array to ensure that the network would not stall while waiting for the consumption of output packets. Due to the way the DMA sends large packets and the way the width converters function, intermediate components also had to be added to reorient the bits into the intended order. A high-level block diagram describing the structure of the robot's digital design can be seen in Figure 4.

B. Software Design

Since the use of the Petalinux operating system allows for typical C/C++ applications to be compiled and ran, existing libraries were used to control many of the sensors and interfaces required to operate the robot. Libraries for interfacing with the GPIO IP blocks and the PWM IP blocks come included in the

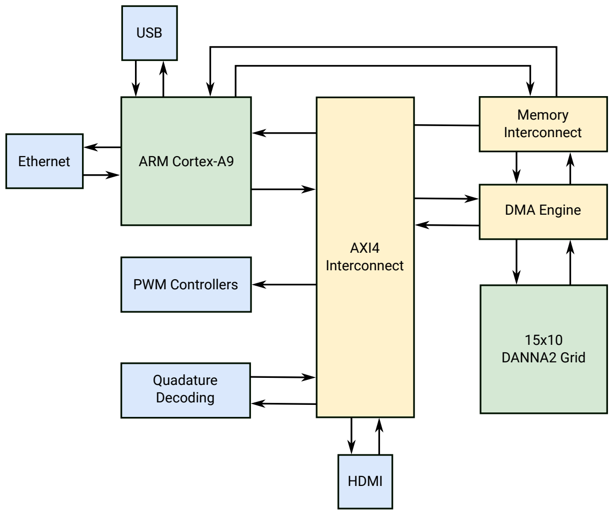

Fig. 4. GRANT digital design block diagram

Petalinux distribution, and these were used for managing the speed controllers for the motors and reading the counts of the encoder. A software development kit was also provided for use with the RPLIDAR A2, and this was used for accessing scan results [28]. Similarly, a software library for the Pixy2 vision sensor also exists, and this was used to receive information about recognized targets [29]. A library for using a JSON object for configuration was also integrated, as this simplified adding adjustable parameters to the configuration of the robot's software [30]. The last external library used was a library that allowed direct communication with the Xilinx AXI-DMA IP block from userspace, which allowed us to communicate directly with the DANNA2 array from our program [31].

In the physical system's implementation, the robot begins in a state of roaming. It can only switch out of this state when it sees a target. The robot updates its sensors every 150 milliseconds, but it makes a decision every 10 milliseconds. This means that the network utilizes old LIDAR information for approximately 14 cycles. During these cycles, the position of the target relative to the robot is maintained by using the displacement measured by the encoders. This method is continually used for tracking the object until the object is either seen again, in which it is reset based on the information gathered from the camera, or, if the target goes without being seen for long enough, the robot reverts to roaming until it sees the target again. If the robot gets close enough to the target (within 24 inches), it will be considered to have "won" by reaching the target, and thus, will stop.

To provide a basis of comparison against the performance of the robot when controlled by neuromorphic hardware, traditional algorithms were implemented that used a basic decision tree to determine whether to move forwards, turn right, or turn left. These algorithms simply utilize the LIDAR and the position of the target to make these decisions, turning to avoid obstacles and trying to maintain sight of the target to increase the accuracy of the assumed position of the target.

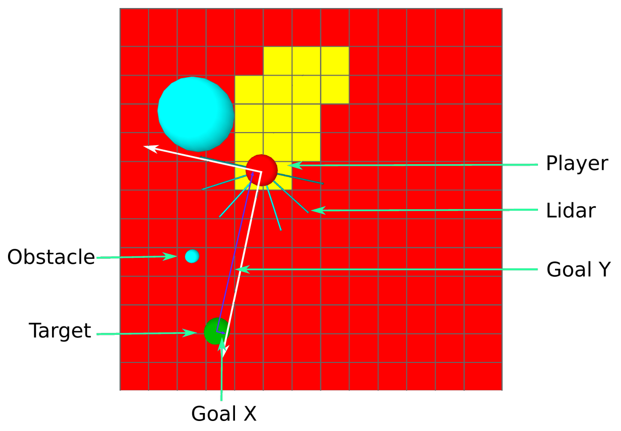

Fig. 5. Simulator explanation and network input information

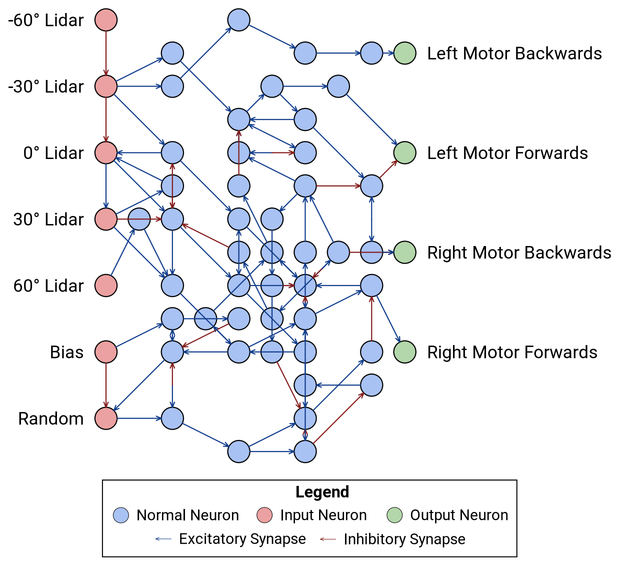

Fig. 6. Best-performing grid coverage network

For the neuromorphic counterparts of these algorithms, the information was simply encoded into input spikes for the network and they were sent through the DMA to the network. All of the corresponding output spikes were accounted for, and after the last output spike was received, motor speeds were set according to the network's decision.

In order to allow the robot to search the room for the target, grid coverage was performed until the target was acquired. At this point, the robot reprograms the DANNA2 array to use a targeting network and begins pursuing the target. Should it lose sight of the target for long enough, it again reverts to the grid coverage network. In this way, networks are able to be trained for independent tasks (exploration and targeting), and the robot can adapt to situationally determine which network should be used. By doing this, a seemingly complex set of tasks can be reduced to less complex subtasks that networks can be trained to perform.

C. DANNA2 Inputs and Outputs

In consideration of how to best give information to a neural network for control and how to best allow it to execute control over the robot, many configurations were tried. With the sensors used on the robot, Figure 5 shows the information available for use as potential input.

For the purpose of grid coverage, it was found that information could be gleaned from the previous work in NeoN to find a good input and output structure [12]. Just like NeoN, five lines of the LIDAR were used, evenly spaced from -60 degrees to 60 degrees with a maximum sight distance of one meter and a minimum distance of 0.333 meters. At the maximum distance, the input neuron associated with a given LIDAR fires with minimum weight and at the minimum distance (or anything less than the minimum distance), the input neuron fires with maximum weight with values scaled linearly within this range. Additionally, a bias input is added that will always provide fires of maximum weight such that the network has input even when the LIDAR is not active. The final input is a random input in which a random value is passed to the network with a new random value being generated every 100 cycles. This input is in place to try to encourage diversity in turning direction. For outputs, the left and right side motor pairs each get a forward and backward neuron. The robot is biased to go forward at 50% speed on each side. For each vote to a forward or backward neuron for a given side, the motor speed is increased or decreased, respectively, by 10% [12]. This allows the neural network to directly control not only how it moves, but the speed at which in moves in that manner. We refer to this output scheme as the speed output scheme. The best-performing grid coverage network, trained with EONS, can be seen in Figure 6.

For targeting, many other input and output methodologies were experimented with. It was found that the trained networks performed best with an additional two LIDAR lines used at -90 degrees and 90 degrees encoded in the same manner as those for grid coverage. Furthermore, targeting information was provided to the network using neurons to represent the left, right, forwards, and backwards components of the target's location relative to the front of the robot with a maximum distance of one meter and a minimum distance of zero meters. The input neuron for each direction is fired with a weight on a linear scale with any distance greater than or equal to the maximum distance being represented by a maximum weight fire and the minimum distance being represented by no fire. It was found that with the targeting application's higher degree of complexity, trained networks performed better with a simplified output scheme that had the network vote to turn left, turn right, or move forwards. If votes were tied, the action with higher precedence was performed with precedence ordered as turning left, turning right, and moving forwards. We call this the decision output scheme. The best-performing targeting network, trained with EONS, can be seen in Figure 7. All other network topologies that were tried either had worse performance in training or the behavior between simulation and the physical system did not match.

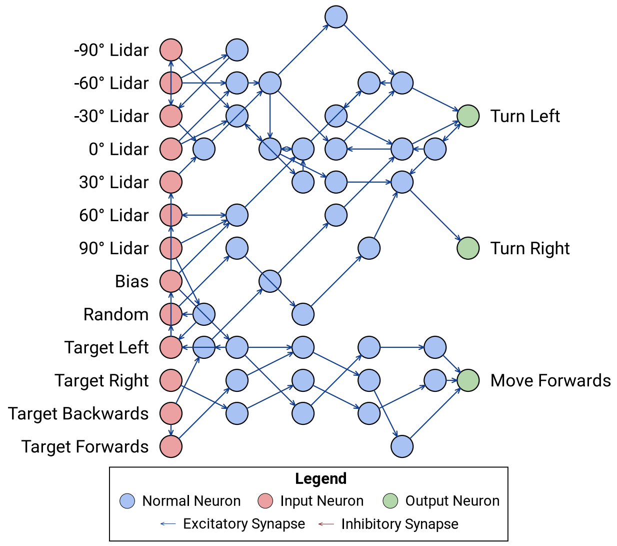

Fig. 7. Best-performing targeting network

As can be seen in both Figures 6 and 7, networks capable of performing the desired functions are incredibly small in comparison to their typical counterparts used in similar robotics applications.

V. Results & Discussion

A. Simulation

In training networks using EONS, it was found that the same input and output structure used by NeoN provided good results in grid coverage, with average room coverage being around 60% for the best-performing network. This number is representative of total room coverage and does not account for areas of the room that are unreachable because of either being occupied by an obstacle or access to it being blocked by obstacles. This particular network attempts to turn left to sweep around obstacles and turns right to avoid anything it sees straight ahead of it. The network is also incredibly small in size compared to those typically used for this type of application, with the network only containing 81 neurons and 110 synapses. An example of the simulated network's performance can be seen in Figure 8.

In the training of targeting networks using EONS, many parameters were found to have considerable implications to the performance of the system; however, many of these implications were entirely manifested in the physical implementation of the robot and not in the simulator itself. Of the parameters that actually affect the performance of the network within the simulator, it was found that the parameters that mattered the most were a penalty for not facing towards the target, a penalty to greatly discourage crashing into an obstacle, the maximum and minimum values for targeting information provided to the network, and the priority for actions when using the decision output scheme. The penalty for not facing toward the target allowed the training to develop desirable behavior at early stages, as facing towards the target when moving yielded a

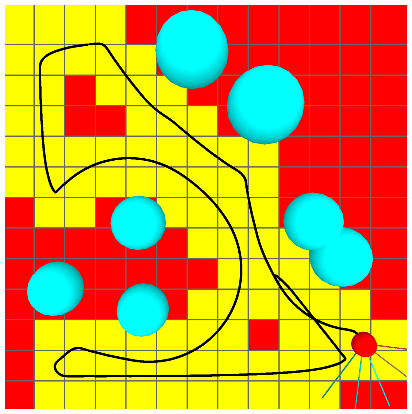

Fig. 8. Grid coverage performance

greater fitness score. The penalty that discouraged crashing into obstacles aided in developing networks that would go towards the target while still prioritizing avoiding things that got in its way. It also forced the network to recognize the correlation between information about the target position and LIDAR information, such that the network could ignore the LIDAR if it thought the detected object was the target and, as a result, could get close enough to the object to "win." By modifying the maximum and minimum values for the targeting information, the resolution of the network's "vision" was effectively changed. This means that by decreasing the range between the maximum and minimum values, the margin of error for the robot to be facing the target was decreased, ultimately resulting in better performance. This is greatly impactful, as the four-bit resolution in DANNA2 input spikes only allows for sixteen distinct values to be encoded within the range. The priority of the decisions in the decision output scheme was found to need to emphasize turning first, as doing otherwise would result in networks that were more prone to crashing into obstacles. All parameters that only altered behavior on the robot and their implications will be discussed in the next section.

In a large room, the best-performing network is able to reach the target, regardless of the number of obstacles, 87% of the time. These losses are usually attributed to the network being unable to reach the target with the path it has chosen, but on rare occasion, the network will hit an obstacle because it believes that it could be the target due to proximity. This network does exhibit the behavior of turning both directions and generally trying to keep a straight path towards the target. The network's size, like the grid coverage network, was also small, as the network only utilized 72 neurons and 89 synapses. Examples of this network's behavior can be seen in Figures 9 and 10.

Traditional algorithms were also ported to the simulator, such that these algorithms could be compared to the solutions provided by the neural networks. These traditional algorithms succeed in reaching the target 99.2% of the time; however, the traditional algorithms receive more LIDAR information (33 distinct lines) and perform tasks to filter out the target hitting

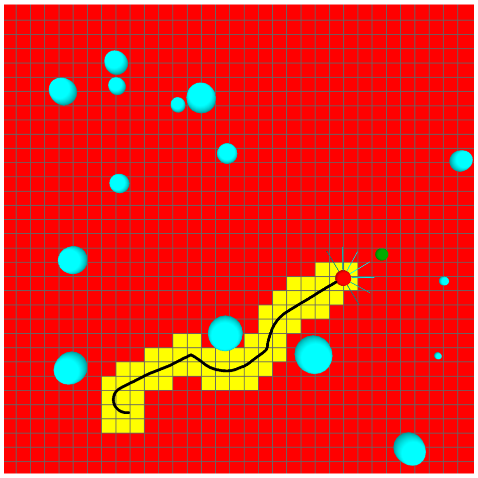

Fig. 9. Targeting performance example 1

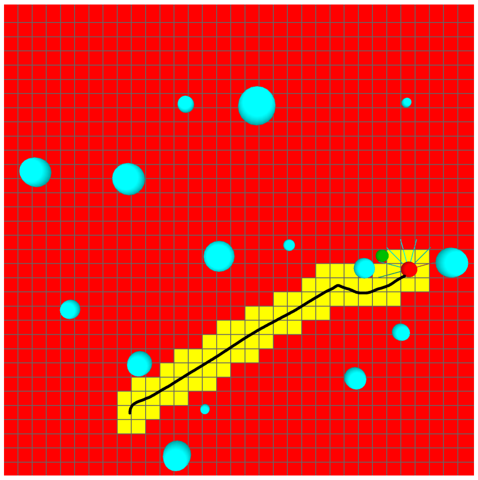

Fig. 10. Targeting performance example 2

the LIDAR. In this way, the traditional algorithms have an advantage over the neural networks. To combat this, the number of LIDAR used by the traditional algorithms was reduced for fair comparison to seven distinct lines, and with this reduced information, the traditional algorithms only reach the target 41% of the time. While these traditional algorithms are not the most optimal solution one could achieve, these results show the advantage that the RSNNs can provide with limited information in a resource-constrained environment.

B. Physical Implementation

When training networks to be evaluated on the physical robot, it was found that discrepancies existed between the simulator and the robot. Namely, motor speed was not adequately modeled, and thus, the speed output scheme would not work correctly. Even trying to mitigate low speed motions that were not possible on the physical system proved insufficient in creating a set of actions that matched between the simulator and the physical system. This led to the development of the decision output scheme. It was also found that the addition of a moving target to the simulator aided the ability of the network to accurately move toward the target on the physical system. This improved behavior can be attributed to the fact that the camera used to discern the location of the target often produced slightly inconsistent results. The addition of a moving target also yielded the ability for the physical system to not only find a target, but follow it around a room even as it moves. To further avoid the issues of crashing into obstacles, LIDAR lines were "binned," such that at a given degree measurement, the distance used was actually that of the closest object within an angular range. Optimal performance was found to be achieved when these angular ranges were nine degrees wide.

In practice, the simulator and the traditional algorithms on the physical system appear to produce similar results. The robot is able to move towards the target while avoiding obstacles, rarely having any issue discerning the difference between an obstacle and the target. For grid coverage using the neural network, behavior also seems to match between the simulator and physical system. It appears as though any inconsistency caused by using the speed output scheme is not evident, as seemingly random turn amounts actually helps to increase the grid coverage. For targeting, the physical system's behavior again seems to mostly match that of the simulator.

VI. Future Work

It is theorized that one could potentially resolve conflicts between the physical system and the simulator through a couple methods. One such method would be to use a PID loop to ensure that the response of the motors corresponds to a physical speed that can then be matched identically in the simulator. Additionally, adding the drift caused by the omni-directional wheels to the simulation would allow the simulation to better match the physical system. Further work would need to be done to model the transition between consecutive decisions made by the network as the current implementation simply treats each time interval as discrete, such that the current motion of the robot is not taken into account for the next timestep.

To allow the neuromorphic implementation of object targeting to better compete with the traditional implementation, work could be done to allow larger networks to be run on the robot itself. Such work could involve the use of a single board computer to run the DANNA2 network simulator with larger networks. Using the simulator would also allow the removal of connectivity restrictions enforced by the digitally-implemented design. One could also utilize a larger FPGA on the robot or wireless communication to a host with a larger DANNA2 array to accomplish this goal of removing the network size restrictions enforced by the robot's resource constraints. The implementation of a DANNA2 array in an application specific integrated circuit (ASIC) would also allow for greater density while simultaneously increasing the network speed and decreasing the power consumption.

Furthermore, while the robot is constrained to a two-dimensional plane, this work could be built upon to allow a drone to navigate a three-dimensional environment with a similar input and output structure.

VII. Conclusions

GRANT is the first self-contained, embedded, neuromorphic system to utilize the hardware implementation of the DANNA2 neuromorphic processor for a control application. The robot also shows the ability of the trained networks to maintain obstacle avoidance while pursuing a target given only estimated positional data. The trained targeting network also shows capability of performing with significantly less information than that given to the traditional algorithms, as well as proving its ability to grossly outperform these algorithms when they were given this reduced information. Lastly, the robot shows the ability of DANNA2 grid arrays to support real-time switching of neural networks to facilitate situational transitions between subtasks of a multi-objective function. This greatly expands the achievable functionality of a resource-constrained, embedded platform as multiple small networks can be used to achieve a larger, more complex objective.

References

[1] G. A. Bekey *et al.*, *Neural networks in robotics*. Springer Science & Business Media, 2012, vol. 202.

[2] W. Liu *et al.*, “A survey of deep neural network architectures and their applications,” *Neurocomputing*, vol. 234, pp. 11–26, 2017.

[3] K. D. Fischl *et al.*, “Neuromorphic self-driving robot with retinomorphic vision and spike-based processing/closed-loop control,” in *2017 51st Annual Conference on Information Sciences and Systems (CISS)*, Mar. 2017, pp. 1–6. DOI: 10.1109/CISS.2017.7926179.

[4] M. Bojarski *et al.*, “End to end learning for self-driving cars,” *CoRR*, vol. abs/1604.07316, 2016. arXiv: 1604.07316. [Online]. Available: http://arxiv.org/abs/1604.07316.

[5] C. D. Schuman *et al.*, “A survey of neuromorphic computing and neural networks in hardware,” *CoRR*, vol. abs/1705.06963, 2017. arXiv: 1705.06963. [Online]. Available: http://arxiv.org/abs/1705.06963.

[6] J. Vreeken, *Spiking neural networks, an introduction*, 2003.

[7] J. Cabessa *et al.*, “The super-turing computational power of plastic recurrent neural networks,” *International journal of neural systems*, vol. 24, no. 08, p. 1 450 029, 2014.

[8] C. D. Schuman *et al.*, “An evolutionary optimization framework for neural networks and neuromorphic architectures,” in *International Joint Conference on Neural Networks*, Vancouver, Jul. 2016.

[9] M. B. Milde *et al.*, “Obstacle avoidance and target acquisition for robot navigation using a mixed signal analog/digital neuromorphic processing system,” *Frontiers in neurorobotics*, vol. 11, p. 28, 2017.

[10] F. Akopyan *et al.*, “Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, Oct. 2015, ISSN: 0278-0070. DOI: 10.1109/TCAD.2015.2474396.

[11] T. Hwu *et al.*, “A self-driving robot using deep convolutional neural networks on neuromorphic hardware,” in *2017 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2017, pp. 635–641.

[12] J. P. Mitchell *et al.*, “NeoN: Neuromorphic control for autonomous robotic navigation,” in *IEEE 5th International Symposium on Robotics and Intelligent Sensors*, Ottawa, Canada, Oct. 2017, pp. 136–142.

[13] J. P. Mitchell *et al.*, “DANNA 2: Dynamic adaptive neural network arrays,” in *International Conference on Neuromorphic Computing Systems*, Knoxville, TN: ACM, Jul. 2018. DOI: 10.1145/3229884.3229894.

[14] P. Merolla *et al.*, “A digital neurosynaptic core using embedded crossbar memory with 45pj per spike in 45nm,” in *2011 IEEE Custom Integrated Circuits Conference (CICC)*, Sep. 2011, pp. 1–4. DOI: 10.1109/CICC.2011.6055294.

[15] J. P. Mitchell, “Danna2: Dynamic adaptive neural network arrays,” Master’s thesis, The University of Tennessee, Aug. 2018.

[16] J. S. Plank *et al.*, “The TENNLab exploratory neuromorphic computing framework,” *IEEE Letters of the Computer Society*, vol. 1, no. 2, pp. 17–20, Jul. 2018. DOI: 10.1109/LOCS.2018.2885976. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/LOCS.2018.2885976.

[17] S. B. Shrestha *et al.*, “SLAYER: Spike layer error reassignment in time,” in *Advances in Neural Information Processing Systems 31*, S. Bengio *et al.*, Eds., Curran Associates, Inc., 2018, pp. 1412–1421. [Online]. Available: http://papers.nips.cc/paper/7415-slayer-spike-layer-error-reassignment-in-time.pdf.

[18] W. Severa *et al.*, “Training deep neural networks for binary communication with the Whetstone method,” *Nature Machine Intelligence*, vol. 1, pp. 86–94, Jan. 2019. [Online]. Available: https://doi.org/10.1038/s42256-018-0015-y.

[19] J. J. M. Reynolds *et al.*, “Intelligent reservoir generation for liquid state machines using evolutionary optimization,” in *IJCNN: The International Joint Conference on Neural Networks*, Budapest, 2019, pp. 1–8. DOI: 10.1109/IJCNN.2019.8852472.

[20] J. S. Plank *et al.*, “A unified hardware/software co-design framework for neuromorphic computing devices and applications,” in *IEEE International Conference on Rebooting Computing (ICRC 2017)*, Washington, DC, Nov. 2017.

[21] *Rplidar-a2 laser range scanner_ solid laser range scanner|slamtec*. [Online]. Available: https://www.slamtec.com/en/Lidar/A2.

[22] *Pixy2 – pixycam*. [Online]. Available: https://pixycam.com/pixy2/.

[23] *Pynq-z1 [reference.digilentinc]*. [Online]. Available: https://reference.digilentinc.com/reference/programmable-logic/pynq-z1/start.

[24] *Petalinux tools*. [Online]. Available: https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html.

[25] M. Orsucci *et al.*, *Petalinux-arty-z7-20*, 2018. [Online]. Available: https://github.com/Digilent/Petalinux-Arty-Z7-20.

[26] S. Bobrowicz *et al.*, *Arty-z7-20-base-linux*, 2018. [Online]. Available: https://github.com/Digilent/Arty-Z7-20-base-linux.

[27] S. Larson, *Quadrature decoder (vhdl)*, 2017. [Online]. Available: https://www.digikey.com/eewiki/pages/viewpage.action?pageId=62259228.

[28] T. Huang *et al.*, *Rplidar_sdk*, 2019. [Online]. Available: https://github.com/slamtec/rplidar_sdk.

[29] R. LeGrand, *Pixy2*, 2019. [Online]. Available: https://github.com/charmedlabs/pixy2.

[30] N. Lohmann, *Json for modern c++*, 2019. [Online]. Available: https://github.com/nlohmann/json.

[31] B. Perez *et al.*, *Xilinx_axidma*, 2018. [Online]. Available: https://github.com/bperez77/xilinx_axidma.