A Self-Driving Robot Using Deep Convolutional Neural Networks on Neuromorphic Hardware

Tiffany Hwu*†, Jacob Isbell‡, Nicolas Oros§, and Jeffrey Krichmar*¶
*Department of Cognitive Sciences
University of California, Irvine
Irvine, California, USA, 92697

†Northrop Grumman
Redondo Beach, California, USA, 90278

‡Department of Electrical and Computer Engineering
University of Maryland
College Park, Maryland, USA, 20742

§BrainChip Inc.
Aliso Viejo, California, USA, 92656

¶Department of Computer Sciences
University of California, Irvine
Irvine, California, USA, 92697
Email: jkrichma@uci.edu

***Abstract*—Neuromorphic computing is a promising solution for reducing the size, weight and power of mobile embedded systems. In this paper, we introduce a realization of such a system by creating the first closed-loop battery-powered communication system between an IBM Neurosynaptic System (IBM TrueNorth chip) and an autonomous Android-Based Robotics platform. Using this system, we constructed a dataset of path following behavior by manually driving the Android-Based robot along steep mountain trails and recording video frames from the camera mounted on the robot along with the corresponding motor commands. We used this dataset to train a deep convolutional neural network implemented on the IBM NS1e board containing a TrueNorth chip of 4096 cores. The NS1e, which was mounted on the robot and powered by the robot's battery, resulted in a self-driving robot that could successfully traverse a steep mountain path in real time. To our knowledge, this represents the first time the IBM TrueNorth has been embedded on a mobile platform under closed-loop control.**

I. Introduction

As the need for faster, more efficient computing continues to grow, the observed rate of improvement of computing speed shows signs of leveling off [1], [2]. In response, researchers have been looking for new strategies to increase computing power. Neuromorphic hardware is a promising direction for computing, taking a brain-inspired approach to achieve magnitudes lower power than traditional Von Neumann architectures [3], [4]. Mimicking the computational strategy of the brain, the hardware uses event-driven, massively parallel and distributed processing of information. As a result, the hardware has low size, weight, and power, making it ideal for mobile embedded systems. While the traditional solution to performing computation with a limited power supply is often to offload computation through cloud computing, this is unreliable in areas of limited connectivity and would be ineffective for tasks requiring immediate results. With neuromorphic hardware, computationally intensive algorithms could be run at low power on the device itself.

One such application is autonomous driving [5]. In order for an autonomous mobile platform to perform effectively, it must be able to process large amounts of information simultaneously, extracting salient features from a stream of sensory data and making decisions about which motor actions to take [6]. Particularly, the platform must be able to segment visual scenes into objects such as roads and pedestrians [5]. Deep convolutional networks (CNNs) have proven very effective for many of these tasks [7]. A CNN is a multi-layer feedforward neural network, most often used to classify input data with spatial information such as images [8]. From one layer to the next, a filter of weights is convolved along the spatial dimensions. The weights are trained using the standard backpropagation rule, comparing the desired and actual output of the final layer of the network [9]. Training is performed iteratively, by running a batch of the training data through a forward pass of the network, calculating the difference between expected and actual output, and incrementally adjusting network weights by gradient descent.

CNNs have previously been applied in autonomous driving.

For instance, Huval et al. [10] used deep learning on a large dataset of highway driving to perform a variety of functions such as object and lane detection . Recently, Bojarski et al. [11] showed that tasks such as lane detection do not need to be explicitly trained. In their DAVE-2 network, an end-to-end learning scheme was presented in which the network is simply trained to classify images from the car's cameras into steering commands learned from real human driving data. Intermediate tasks such as lane detection were automatically learned within the intermediate layers, saving the work of selecting these tasks by hand.

Such networks are suitable for running on neuromorphic hardware due to the large amount of parallel processing involved. In fact, many computer vision tasks have already been successfully transferred to the neuromorphic domain, such as handwritten digit recognition [12] and scene segmentation [13]. However, less work has been done in embedding the neuromorphic hardware on mobile platforms. An example includes NENGO simulations embedded on SpiNNaker boards controlling mobile robots [14], [15]. Addressing the challenges of physically connecting these components, as well as creating a data pipeline for communication between the platforms is an open issue, but worth pursuing given the small size, weight and power of neuromorphic hardware.

At the Telluride Neuromorphic Cognition Workshop 2016, we embedded the the IBM NS1e board containing the IBM Neurosynaptic System (IBM TrueNorth chip) [16] on the Android-Based Robotics platform [17] to create a self-driving robot that uses a deep CNN to travel autonomously along an outdoor mountain path. The result of our experiment is a robot that is able to use video frame data to steer along a road in real time with low-powered processing.

II. Platforms

A. IBM TrueNorth

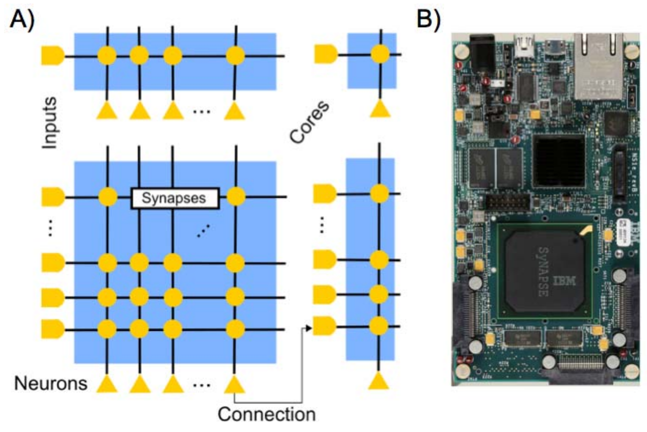

Fig. 1. A) Core connectivity on the TrueNorth. Each neuron on a core can be configured to connect to all, none, or an arbitrary set of input axons on the core. Neuron outputs connect to input axons on any other core in the system (including the same core) through a Network-on-Chip. B) The IBM NS1e board. Adapted from [18].

The IBM TrueNorth (Figure 1) is a neuromorphic chip with a multicore array of programmable neurons. Within each core, there are 256 input axon lines connected to 256 neurons through a 256x256 synaptic crossbar array. Each neuron on a core is configurably connected with every other neuron on the same core through the crossbar, and can communicate with neurons on other cores through their input axon lines. In our experiment, we used the IBM NS1e board, which contains 4096 cores, 1 million neurons, and 256 million synapses. The integrate-and-fire neuron model has 23 parameters and may be configured to use trinary synaptic weights of -1, 0, and 1. As the TrueNorth has been used to run many types of deep convolutional networks, and is able to be powered by an external battery, it served as ideal hardware for this task [19] [18] .

B. Android Based Robotics

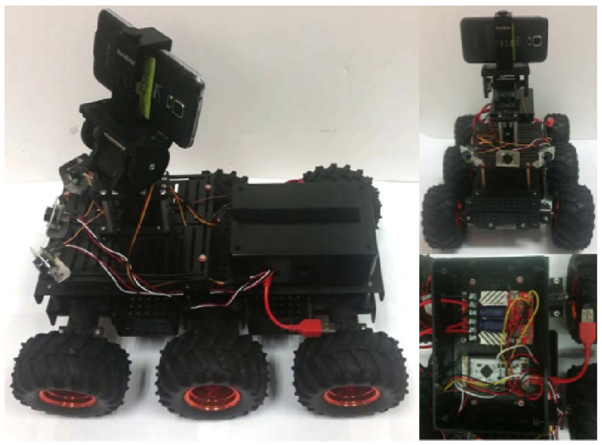

Fig. 2. Left: Side view of CARLorado. A pan and tilt unit supports the Samsung Galaxy S5 smartphone, which is mounted on a Dagu Wild Thumper chassis. A plastic enclosure holds the IOIO-OTG microcontroller and RoboClaw motor controller. A velcro strip on top of the housing can attach any other small components. Top Right: Front view of CARLorado. Three front-facing sonars can detect obstacles. Bottom Right: Close-up of IOIO-OTG and motor controller.

The Android-Based Robotics platform (Figure 2) was created at the University of California, Irvine, using entirely off-the-shelf commodity parts and controlled by an Android phone [17]. The robot used in the present experiment, the CARLorado, was constructed from a Dagu Wild-Thumper All-Terrain chassis that could easily travel through difficult outdoor terrain. A IOIO-OTG microcontroller (SparkFun Electronics) communicated through a Bluetooth connection with the Android phone (Samsung Galaxy S5). The phone provided extra sensors such as a built-in accelerometer, gyroscope, compass, and global positioning system (GPS). The IOIO-OTG controlled a pan and tilt unit for changing the camera view and a motor controller for the robot wheels, using pulse width modulation to change the intensity and direction of movement. The IOIO-OTG also communicated with ultrasonic sensors for detecting obstacles. A differential steering technique was used, moving the left and right sides of the robot at different speeds for turning. The modularity of the platform made it easy to add extra units such as the IBM TrueNorth.

Software for controlling the robot was written in Java using Android Studio. With various support libraries for the IOIO-OTG, open-source libraries for computer vision such as OpenCV, and sample Android-Based Robotics code, it was straightfoward to develop intelligent controls.

III. Methods and Results

A. Data Collection

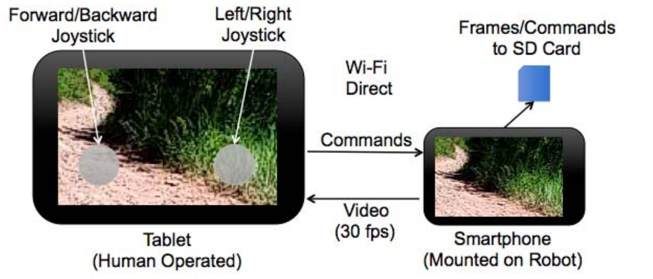
Forward/Backward Joystick
Left/Right Joystick
Wi-Fi Direct
Frames/Commands to SD Card
Commands
Video (30 fps)
Tablet (Human Operated)
Smartphone (Mounted on Robot)

Fig. 3. Data collection setup. Video from the smartphone mounted on the robot was sent to the tablet through a Wi-Fi direct connection. A human operator used two joysticks on the touchscreen of the tablet to issue motor commands, which were sent to the phone through the same connection. With the joysticks, the operator was able to change the speed of moving and turning by changing the pulse width modulation signal sent to the motor controller. Video frames and their corresponding motor commands in the form of pulse width values were saved to the SD card on the smartphone.

First, we created datasets of first-person video footage of the robot and motor commands issued to the robot as it was manually driven along a mountain trail in Telluride, Colorado (Figures 5 and 9 top). This was done by creating an app in Android Studio that was run on both a Samsung Galaxy S5 smartphone and a Samsung Nexus 7 tablet (Figure 3). The smartphone was mounted on the pan and tilt unit of the robot with the camera facing ahead. JPEG images captured by the camera of the smartphone were saved to an SD card at 30 frames per second. The JPEGs had a resolution of 176 by 144 pixels. Through a Wi-Fi direct connection, the video frame data was streamed from the phone to a handheld tablet that controlled the robot. The tablet displayed a control for moving the robot forward and backward at an adjustable speed and a control for steering the robot left and right at an adjustable speed. These commands from the tablet were streamed continuously in the form of pulse width values to the smartphone via the Wi-Fi direct connection. The smartphone then relayed these values to the IOIO-OTG, which generated the pulse width modulation signals for controlling the steering power of the robot. These values were also saved on the smartphone as a text file for training purposes. A total of 4 datasets was recorded on the same mountain trail, with each dataset recording a round trip of .5 km up and down a single trail segment. To account for different lighting conditions, we spread the recordings across two separate days, and on each day we performed one recording in the morning and one in the afternoon. In total we collected approximately 30 minutes of driving data. By matching the time stamps of motor commands to video images, we were able to determine which commands corresponded to which images. Images that were not associated with a left, right, or forward movement such as stopping were excluded. Due to lack of time, only the first day of data collection was used in actual training.

B. Eedn Framework

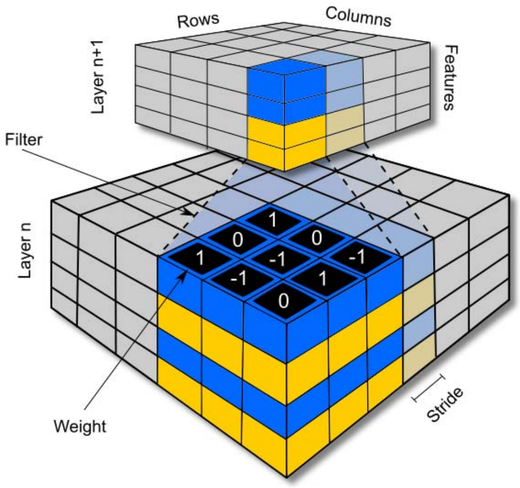

Fig. 4. Convolution of layers in a CNN on TrueNorth. Neurons in each layer are arranged in three dimensions, which can be convolved using a filter of weights. Convolution occurs across all three dimensions. The third dimension represents different features. The convolution can be divided along the feature dimension into groups (indicated by blue and yellow colors) that can be computed separately on different cores. Adapted from [18].

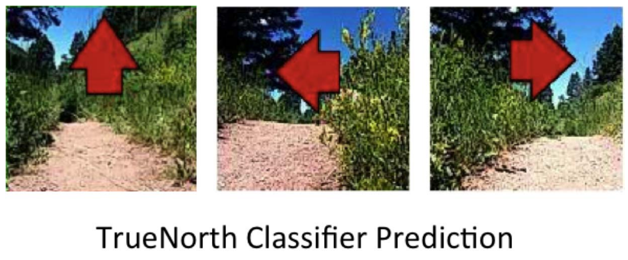

Fig. 5. The CNN classified images into three classes of motor output: turning left, moving forward, and turning right. Accuracy of training was above 90 percent.

We used the dataset to train a deep convolutional neural network using an energy-efficient deep neuromorphic network (Eedn), a network that is structured to run efficiently on the TrueNorth [18]. As the TrueNorth currently does not support on-chip training, the framework provides a method for training network weights off-line and assigning them to the chip. In summary, a CNN is transferred to the neuromorphic domain by first structuring the network according to the core, axon, and neuron structure of TrueNorth. By dividing each layer into groups along the feature dimension (Figure 4), the convolutional operation may be distributed among cores of the TrueNorth to take advantage of the parallel processing.

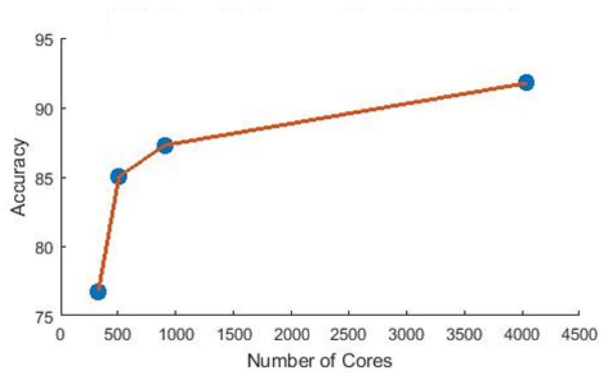

Fig. 6. Effect of the number of cores used on the accuracy of the CNN. One NS1e board contains 4096 cores. An accuracy above 85 percent was still maintained even with the network reduced to one quarter of a full chip.

When a neuron targets multiple core inputs, exact duplicates of the neuron and synaptic weights are created, either on the same core or a different core. The response of each neuron is the binary thresholded sum of synaptic input, in which the trinary weight values are determined by different combinations of two input lines. Then the weights of the restructured CNN are learned using a backpropagation algorithm that trinarizes the network weights, consistent with the low dimensional representation of synaptic weights in TrueNorth. Information from one forward pass of the externally trained CNN is represented in Eedn according to the binary spike patterns of each neuron's response to its input, at every timestep. A more complete explanation of the Eedn flow and structure of the convolutional network used (1 chip version) can be found in [18].

The video frames were preprocessed by down-sampling them to a resolution of 44 by 36 pixels and separating them into red, green, and blue channels. The output is a single layer of three neuron populations, corresponding to three steering movements of ”left”, ”straight”, or ”right”, as seen in Figure 5. Since the steering commands in the training data allowed for different intensities of steering, the commands were simplified such that any amount of turning left was classified as ”left” and any amount of turning right was classified as ”right”.

Using the Eedn MatConvNet package, a Matlab toolbox for implementing convolutional neural networks, the network was trained to classify images using the motor command class labels. Every image in the dataset was labeled with the corresponding human-trained command of steering to the left, right, or straight. In this way, the CNN learned the types of images and image characteristics associated with each command, completely from the dataset provided. No hand labeling of images or a priori human knowledge of scene characteristics was required. To test accuracy, the dataset was split into train and test sets by using every fifth frame as a test frame (in total 20 percent of the dataset). We achieved an accuracy of over 90 percent, which took 10K iterations and a few hours to train. Training was performed separately from the TrueNorth chip, producing trinary synaptic weights (-1,0,1) that could be used interchangeably in the traditional CNN or Eedn.

For more extensive analysis on the training accuracy achieved by the TrueNorth, multiple CNNs were designed that used different amounts of processor capability. Although one NS1e board contains 4096 cores, not all of them need to be used. Using fewer cores reduces the amount of power consumed by the chip [18], at the cost reduced accuracy. To explore this tradeoff, we trained a range of CNNs that filled from one tenth of a chip to the full chip. The sizes of the CNNs were changed by controlling the stride, filter size, or number of layers [20]. Using only one quarter of the chip resulted in 85 percent accuracy (Figure 6). Using the complete chip increased the accuracy to over 90 percent (Figure 6).

C. Data Pipeline

With the methods used in [18], the weights of the network were transferred to the TrueNorth chip. The CNN was able to run on the chip by feeding input from the camera on the Android Galaxy S5 to the TrueNorth using a TCP/IP connection. In order to achieve this, the phone had to replicate the preprocessing used when training the network. The pre-processing on the phone was achieved by using the Android OpenCV scaling function to downsample the images. Then, the images were separated into red, green, and blue channels. Next, the filter kernels from the first layer of the CNN were pulled from the Eedn training output and applied to the image using a 2D convolution function from the Android OpenCV library. The result of the convolution was thresholded into binary spiking format, such that any neuron with an activity greater than zero was set to spike. The spiking input to the TrueNorth was sent in XYF format, where X, Y, and F are the three dimensions to describe the identity of a spiking neuron within a layer. At each tick of the TrueNorth, a frame was fed into the input layer by sending the XYF coordinates of all neurons that spiked for that frame. A detailed diagram of the pipeline is found in Figure 7. Output from the TrueNorth was sent back to the smartphone through the TCP/IP connection in the form of a class histogram, which indicated the firing activity of the output neurons. The smartphone could then calculate which output neuron was the most active and issue the corresponding motor command to the robot.

D. Physical Connection of Platforms

The TrueNorth was powered by connecting the robot's battery terminals from the motor controller to a two-pin battery connection on the NS1e board. It was then secured with velcro to the top of the housing for the IOIO and motor controller. A picture of the setup is seen in Figure 8. The robot, microcontroller, motor controller, servos, and NS1e were powered by a single Duratrax NiMH Onyx 7.2V 5000mAh battery.

E. Testing

With this wireless, battery-powered setup, the trained CNN was able to successfully drive the robot on the mountain

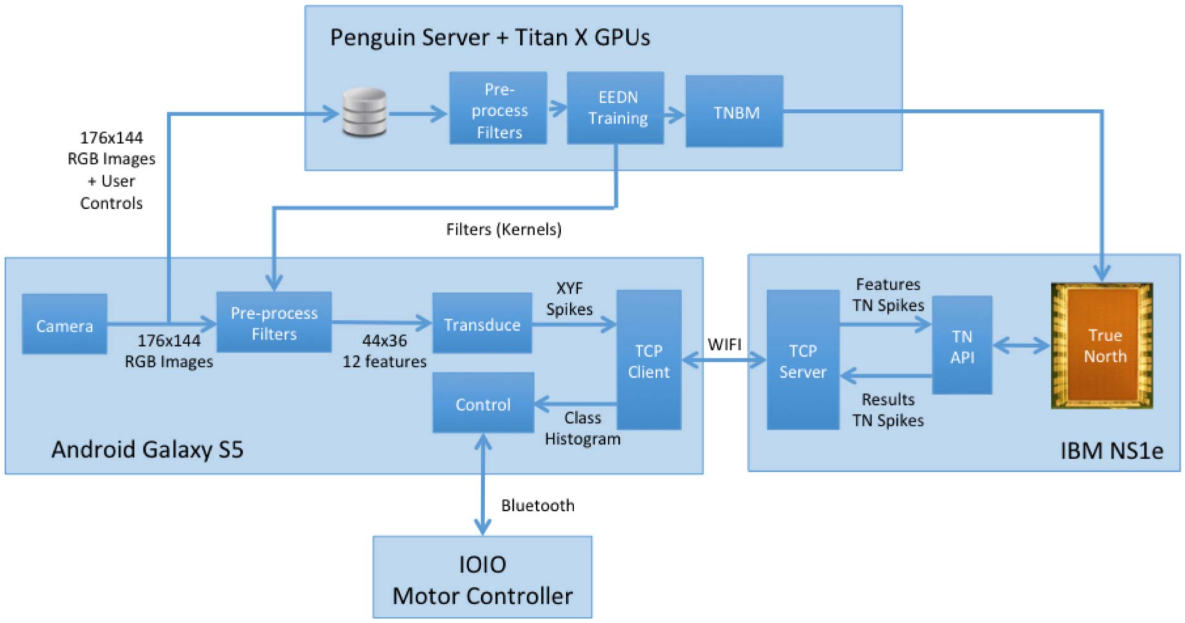

Fig. 7. Data pipeline for running CNN. Training is done separately using the Eedn MatConvNet package using Titan X GPUs. A Wi-Fi connection between the Android Galaxy S5 and IBM NS1e transmit spiking data back and forth, using the TrueNorth (TN) Runtime API.

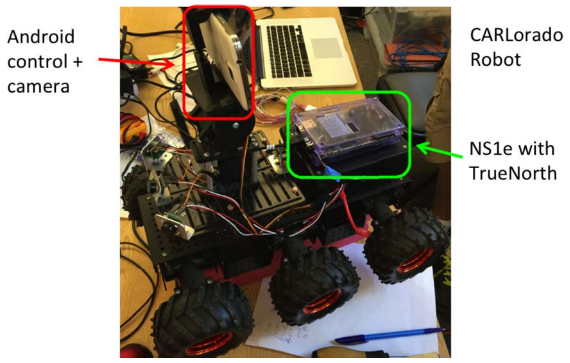

Fig. 8. Physical connection of TrueNorth NS1e and CARLorado. The NS1e is attached to the top of the housing of the electronics housing using velcro. The NS1e is powered by running connections from the motor controller within the housing. The motor controller itself is powered by a Ni-MH battery attached to the bottom of the robot chassis.

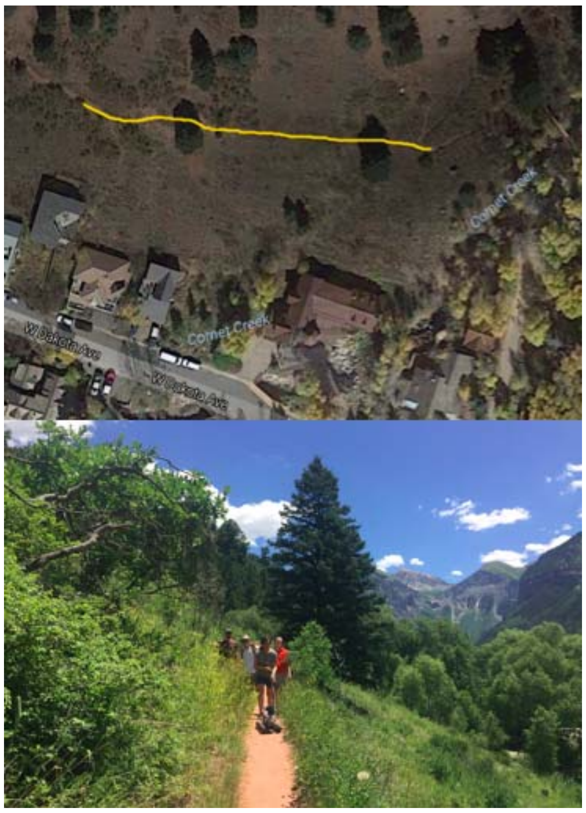

Fig. 9. Mountain trail in Telluride, Colorado. Top: Google Satellite image of trail (highlighted) Imagery ©2016 Google. Bottom: Testing CNN performance.

trail (Figure 9). A wireless hotspot was necessary to create a TCP connection between the NS1e and the Android phone. We placed the robot on the same section of the trail used for training. For testing, the robot was programmed to drive forward constantly while steering left, right, or straight according to the class histograms received from the TrueNorth output, which provided total firing counts for each of the three output neuron populations. Steering was done by using the histogram to determine which output population fired the most, and steering in that direction. As a result, the robot stayed near the center of the trail, steering away from green brush on both sides of the trail. At some points, the robot did travel off the trail and needed to be manually redirected back towards the center of the trail. The robot drove approximately 0.5 km uphill, and returned 0.5 km downhill with minimal intervention. It should be noted that there was a steep dropoff on the south side of the trail. Therefore, extra care was taken to make sure the robot did not tumble down the mountainside. A video of the path following performance can be seen at https://www.youtube.com/watch?v=CsZah2hydeY.

IV. Discussion

To the best of our knowledge, the present setup represents the first time the IBM NS1e has been embedded on a mobile platform under closed loop control. It demonstrated that a low power neuromorphic chip could communicate with a smartphone in an autonomous system. Furthermore, it showed that a CNN using the Eedn framework was sufficient to achieve a self-driving application. Additionally, this complete system ran in real-time and was powered by a single off-the-shelf hobby grade battery, demonstrating the power efficiency of the TrueNorth chip. This was possible due to the power savings of running the CNN computations on neuromorphic hardware instead of directly on the smartphone. In comparison with current methods of general-purpose computing on graphics processing unit (GPGPU) approaches to running CNNs which require up to 235 W to operate [21], the Eedn framework is able to train on classic benchmark datasets at 1,200 and 2,600 frames/s and uses between 25 and 275 mW [18]. Similar power savings should also hold for our application, as our processing rate of 30 frames/s was sufficient for the road following task.

An expansion of this work would require better quantification of the robot's performance. This could be achieved by tracking the number of times the robot had to be manually redirected, or comparing the CNN classifier accuracy on the training set of images versus the classifier accuracy on the actual images captured in real time. Increasing the amount of training data would likely increase the classifier accuracy, since only 15 minutes of data were used for the training as compared to other self-driving CNNs, which have used several days or even weeks of training [10], [11]. Our success was due in part to the simplicity of the landscape, with an obvious red hue to the dirt road and bold green hue for the bordering areas. It would therefore be useful to test the network in more complex settings.

With further development, path following on Eedn can integrate with a larger system of autonomous driving on a neuromorphic system. For instance, a variety of other sensors such as sonars may be used to train the CNN for better obstacle detection. Additionally, the system can be combined with longer-term strategies, such as path planning and decision-making. A neurally inspired path planning algorithm using spiking wavefront propagation has recently been implemented on the IBM TrueNorth [22], along with a simulated version of the algorithm that runs on the Android-based robot [23] and adapts to changes in the environment. With a CNN structure that leaves a few cores open on the TrueNorth chip, the path following and path planning algorithm can run on the same board simultaneously, sharing a communication line with the Android phone. The result would be an entirely self-contained neuromorphic navigating robot.

V. Conclusion

In the present study, we demonstrated a novel closed-loop system between a robotic platform and a neuromorphic chip, operating in a rugged outdoor environment. We have shown the advantages of integrating neuromorphic hardware with popular machine learning methods such as deep convolutional neural networks. We have shown that neuromorphic hardware can be integrated with smartphone technology and off-the-shelf components resulting in a complete autonomous system. The present setup is one of the first demonstrations of using neuromorphic hardware in an autonomous, embedded system.

Acknowledgment

The authors would like to thank Andrew Cassidy and Rodrigo Alvarez-Icaza of IBM for their support. This work was enabled by a pre-release of the TrueNorth Runtime API and was supported by the National Science Foundation Award number 1302125 and Northrop Grumman Aerospace Systems. We also would like to thank the Telluride Neuromorphic Cognition Engineering Workshop, The Institute of Neuromorphic Engineering, and their National Science Foundation, DoD and Industrial Sponsors.

References

[1] J. Backus, "Can programming be liberated from the von neumann style?: a functional style and its algebra of programs," *Communications of the ACM*, vol. 21, no. 8, pp. 613–641, 1978.

[2] A. Danowitz, K. Kelley, J. Mao, J. P. Stevenson, and M. Horowitz, "Cpu db: recording microprocessor history," *Communications of the ACM*, vol. 55, no. 4, pp. 55–63, 2012.

[3] C. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1629–1636, 1990.

[4] G. Indiveri, B. Linares-Barranco, T. J. Hamilton, A. Van Schaik, R. Etienne-Cummings, T. Delbruck, S.-C. Liu, P. Dudek, P. Häfliger, S. Renaud *et al.*, "Neuromorphic silicon neuron circuits," *Frontiers in neuroscience*, vol. 5, p. 73, 2011.

[5] S. Thrun, "Toward robotic cars," *Communications of the ACM*, vol. 53, no. 4, pp. 99–106, 2010.

[6] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt *et al.*, "Towards fully autonomous driving: Systems and algorithms," in *Intelligent Vehicles Symposium (IV), 2011 IEEE*. IEEE, 2011, pp. 163–168.

[7] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[8] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[9] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[10] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue *et al.*, "An empirical evaluation of deep learning on highway driving," *arXiv preprint arXiv:1504.01716*, 2015.

[11] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[12] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *arXiv preprint arXiv:1608.08782*, 2016.

[13] Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," *International Journal of Computer Vision*, vol. 113, no. 1, pp. 54–66, 2015.

[14] J. Conradt, F. Galluppi, and T. C. Stewart, “Trainable sensorimotor mapping in a neuromorphic robot,” *Robotics and Autonomous Systems*, vol. 71, pp. 60–68, 2015.

[15] F. Galluppi, C. Denk, M. C. Meiner, T. C. Stewart, L. A. Plana, C. Eliasmith, S. Furber, and J. Conradt, “Event-based neural computing on an autonomous mobile platform,” in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 2862–2867.

[16] 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.

[17] N. Oros and J. L. Krichmar, “Smartphone based robotics: Powerful, flexible and inexpensive robots for hobbyists, educators, students and researchers,” Center for Embedded Computer Systems, University of California, Irvine, Irvine, California, Tech. Rep. 13-16, 2013.

[18] S. K. Esser, P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch, C. di Nolfo, P. Datta, A. Amir, B. Taba, M. D. Flickner, and D. S. Modha, “Convolutional networks for fast, energy-efficient neuromorphic computing,” *Proceedings of the National Academy of Sciences*, vol. 113, no. 41, pp. 11 441–11 446, 2016.

[19] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, B. Taba, M. Beakes, B. Brezzo, J. Kuang, R. Manohar, W. Risk, B. Jackson, and D. Modha, “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, 2015.

[20] M. D. Zeiler and R. Fergus, *Visualizing and Understanding Convolutional Networks*. Cham: Springer International Publishing, 2014, pp. 818–833. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-10590-1_53

[21] K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss, and E. S. Chung, “Accelerating deep convolutional neural networks using specialized hardware,” *Microsoft Research Whitepaper*, vol. 2, no. 11, 2015.

[22] 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*. IEEE, 2017.

[23] 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*, 2017.