Controlling Articulated Robots in Task-Space with Spiking Silicon Neurons

Samir Menon[1], Sam Fok[2], Alex Neckar[2], Oussama Khatib[1], and Kwabena Boahen[3]

***Abstract*— Emulating how humans coordinate articulated limbs within the brain's power budget promises to accelerate progress in building autonomous biomimetic robots. Here, we used a neuromorphic approach—low-power analog silicon spiking neurons—to control an articulated robot in real-time. We obtained a closed-form control function that computes robot motor torques given the robot's joint angles (state) and desired end-effector forces; factorized the function into a set of sub-functions over five unique three-dimensional domains; and regressed each sub-function on to the steady-state spiking responses of one out of five silicon spiking-neuron pools. The spiking pools controlled a three degree-of-freedom robot's motor torques in real-time and performed reaches to arbitrary locations in space with less than 2 cm root-mean-square trajectory tracking error (of an analytical controller). The controller is compliant and can draw shapes with a pen on a dynamically perturbed surface while remaining stable. Using force control resulted in linear responses to perturbations in end-effector coordinates (task-space), which effectively filtered noise due to neuron spikes. Factorizing the controller reduced the neural regression's complexity to cubic in the dynamic range of the robot's state and desired forces. Doing so made acquiring spiking responses for regression tractable in time (~2–3 min), and enabled reliable trajectory tracking with only 1280 neurons. This is the first time a neuromorphic system has achieved real-time manipulation for an articulated robot with three or more degrees-of-freedom.**

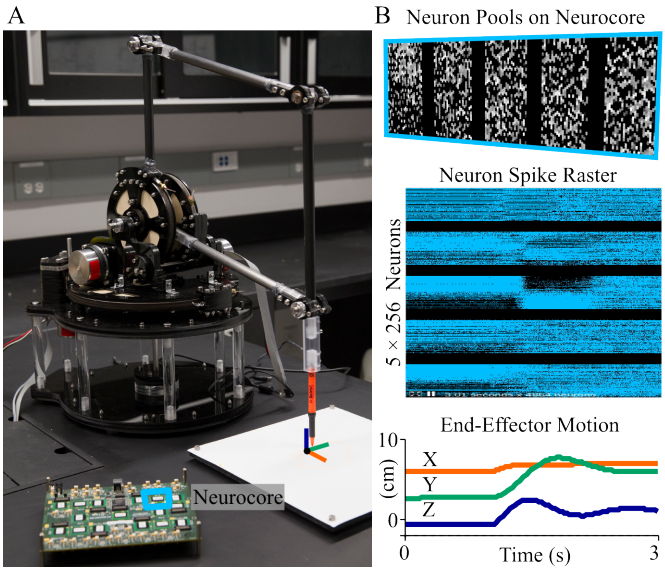

Fig. 1. Robot Control with Neurogrid. A. Neurogrid, a million-neuron neuromorphic system made up of sixteen Neurocores, controlled Neurobot, a lightweight force-controlled desktop robot. B. Five 256 spiking-neuron pools on a Neurocore implemented Neurobot's end-effector force-control function. A spike raster displays neural activity while Neurogrid controls the end-effector in task-space.

I. Neuromorphic Robot Control

Neuromorphic systems present a novel avenue for autonomous robot control by providing programmable mega-scale networks of biologically inspired spiking neurons that operate in real-time while consuming a few watts [1]. Neuromorphics minimize power and operate in real-time by using analog neurons, and can be programmed to compute arbitrary mathematical functions by engineering neuron connection weights [2]. Using analog neurons, however, requires accommodating stochastic noise, heterogeneity and millisecond-scale time constants, which have stymied efforts to engineer neuromorphic controllers for robotic manipulation. The present state-of-the-art is to control planar mobile robots [3], [4] and arm models in simulation [5], [6], or to correct linear torque errors for a two degree-of-freedom robot [7].

Previous attempts to use spiking neural networks for control relied on model-free learning to determine connections weights [3], [4], [5]. The nonlinear multidimensional nature of robot kinematics and dynamics, however, leads to an exponential increase in search space and training time with increasing degrees-of-freedom. Moreover, past research simulated spiking neural networks on digital computers, which imposes a prohibitive computational cost and limits the number of neurons that can operate in real-time. The primary challenge facing neuromorphic motor control today is thus to demonstrate real-time manipulation by articulated robots with three or more degrees-of-freedom.

In this paper, we present a framework for real-time robot control using spiking silicon neurons (Fig. 1). Our approach is to implement kinetic energy optimal task-space controllers [8], [9] as dynamical systems; factorize these dynamical systems into low-dimensional functions; and map these functions on to pools of neurons using the Neural Engineering Framework [10], [2]. We demonstrate our framework by using a neuromorphic system, Neurogrid [11], to control

*This work was supported by a Stanford Bio-X fellowship (S. Menon), a National Science Foundation Graduate Research Fellowship (S. Fok), a National Institute of Health (NIH) Directors Pioneer Award (DPI-OD000965, K. Boahen), a NIH/National Institute of Neurological Disorders and Stroke Transformative Research Award (R01NS076460, K. Boahen), and an Office of Naval Research Long Range Science and Technology Grant (N00014-13-1-0419, K. Boahen).

[1]S. Menon and O. Khatib are with the Artificial Intelligence Laboratory, Department of Computer Science, Stanford University, Stanford, CA 94305, USA smenon@stanford.edu, ok@cs.stanford.edu

[2]S. Fok and A. Neckar are with the Department of Electrical Engineering, Stanford University, Stanford, CA 94305, USA samfok@stanford.edu, aneckar@stanford.edu

[3]K. Boahen is with the Department of Bioengineering, Stanford University, Stanford, CA 94305, USA boahen@stanford.edu

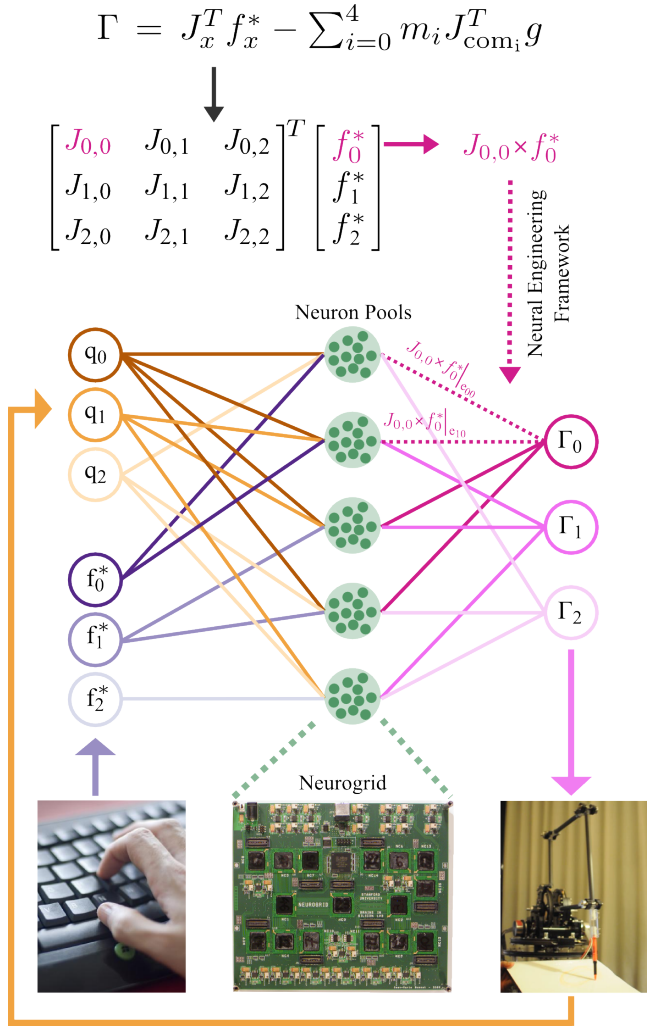

Fig. 2. Spiking Neural Control Framework. The control function was mapped on to Neurogrid by first decomposing it into a set of three-dimensional sub-functions computed using five spiking neuron pools. Each pool takes as its input a unique combination of three joint angles or desired end-effector force components. The sub-functions—$J_{0,0} \times f_0^*$, for instance—are computed as a linearly weighted sum of the neurons' spike rates in each pool. These weighted spike rates are summed for the set of sub-functions required to compute each component of Neurobot's controlled joint torques.

a three degree-of-freedom robot. The robot successfully executes arbitrary motions in free space and manipulates a pen in contact with a surface. Neurogrid is thus the first neuromorphic hardware platform to successfully manipulate an articulated robot while being exposed to arbitrary environmental perturbations.

II. Task-Space Control

We chose to implement task-space force control using the Operational Space Framework [8]. A user (or trajectory generator) provides desired forces (f_x^*) in task-space. Task forces are projected into joint torques (Γ) using a task Jacobian (J_x) and a gravity-compensation torque is added (Γ_g),

$$\Gamma = J_x^T f_x^* + \Gamma_g. \tag{1}$$

The gravitational compensation, Γ_g, is given by

$$\Gamma_g = -\sum_{i=0}^{4} m_i J_{\text{com}_i}^T g, \tag{2}$$

where m_i are the five articulated link masses, $J_{\text{com}_i}^T$ are the link center-of-mass Jacobians, and g is the acceleration due to gravity. Such force control is intrinsically compliant in task-space, which helps it compensate for perturbations and noise.

The operational space framework is attractive for neuromorphic controllers because it provides a closed-form, time-invariant function that locally linearizes complex robot dynamics and unifies motion and force control [8]. These functions can be factorized into sub-functions on lower dimensional sub-domains, simplifying the problem of finding connection weights for the spiking neural network (Fig. 2). In our case, the six-dimensional domain of joint angles and commanded task-space forces was factorized into five three-dimensional sub-domains. We used open-loop force control, but force feedback may be easily accommodated in the future since it involves additions and subtractions, which impose no additional cost on computation.

III. Task-Space Control with Spiking Neurons

We mapped our task-space force controller on to Neurogrid using the Neural Engineering Framework [10], which computes functions with pools of spiking neurons by regressing function outputs on to steady-state neural responses over the function's domain. Each neuron in a pool samples the domain by taking its inner product with an assigned encoding vector within the domain (i.e., the neuron spikes at a higher rate when the pool's inputs align with its encoding vector). Finally, a regularized linear regression over each pool's responses yields a weighted summation of steady-state neuron responses that fits the desired control function—the vector of summation coefficients is called a decoder.

Our mapping procedure involves three stages, factorization, regression, and programming.

A. Factorization

While a single pool of neurons can compute arbitrary nonlinear functions, the number of samples required to fit the function grows exponentially with the dimensionality of the function's domain. We limited the sample space by factorizing our control function into nonlinear sub-functions over three of the domain's six dimensions, which achieved a cubic sample size.

To understand our strategy, consider that $f = (x + y) \cdot z$, whose domain is $\mathbb{R}^3$, can be factorized into $f_0 = x \cdot z$ and $f_1 = y \cdot z$, whose domains are $\mathbb{R}^2$. Doing so reduces the samples required from cubic to quadratic in the domain's dynamic range. Since addition is implicit while decoding [2], the factorized sub-functions are readily combined to reconstruct the original function.

We avoided factorizations that involve function composition, such as $f_0 = x + y$ and $f_1 = f_0 \cdot z$, which require

multiple neuron layers. Doing so introduces a delay because neurons in the additional (f_1) layers take additional time to respond to new inputs. Note, however, that addition and subtraction need not incur additional delay because these operations can be realized by convergent inputs.

Using our factorization strategy, we found the simplest set of nonlinear sub-functions that could be summed to yield our desired control function. These sub-functions spanned the smallest possible domains. The resulting network contains five pools that each represent three variables drawn from the robot joint angles and end-effector force components. Two sub-functions were decoded from the spike rates of neurons in each pool (see Fig. 2). The decoded sub-functions were summed to obtain the three commanded torques (see Appendix for more details).

B. Regression

We sampled the neural responses of Neurogrid's five three-dimensional pools over their respective input state spaces at points on a $10 \times 10 \times 10$ mesh (Fig. 3). We restricted joint angles to $\pm 30^\circ$ and the commanded forces to ± 5.5N, which fully captures the robot's operating range. The encoding vectors were chosen using $\mathcal{L}_1$ basis-pursuit. Given a pool's response matrix, A, which contains the firing rates for each neuron at each mesh point, we then performed regularized linear regression to solve for the decoding weights, d, according to

$$d = \text{argmin}_d \; ||Ad - y||_2 + \lambda ||d||_2, \tag{3}$$

where y is the desired function outputs (y exemplar estimated in Fig. 3.B) and λ is a regularization parameter.

C. Programming

We programmed the decoding weights into Neurogrid's FPGA (Field Programmable Gate Array) daughter board, which weights spikes from the five three-dimensional pools to compute the commanded torques. To decode M sub-functions from a pool of N neurons, the FPGA stores the $M \times N$ decoding matrix, W, and multiplies this matrix with the vector of instantaneous neural reponses using a probabilistic approach [1], [12].

When neuron i spikes, the FPGA looks up the associated row of the decoding matrix. For each w_{ij} in the row, the FPGA generates a random number, r_j (both numbers are 8-bit fixed-point). If $r_j < w_{ij}$, the FPGA produces an output spike addressed to j. Thus, the rate of the spikes being sent addressed to j should, in expectation, be equal to $\sum_i a_i w_{ij}$, where a_i is neuron i's firing rate and w_{ij} is its decoding weight for the j^{th} sub-function.

The FPGA thus produces one stream of output spikes per decoded sub-function, with rates proportional to each sub-function's values. The FPGA sends these streams back to the computer. The computer exponentially filters the spike streams to obtain estimates of the rates, which are sent to the robot as commanded torques.

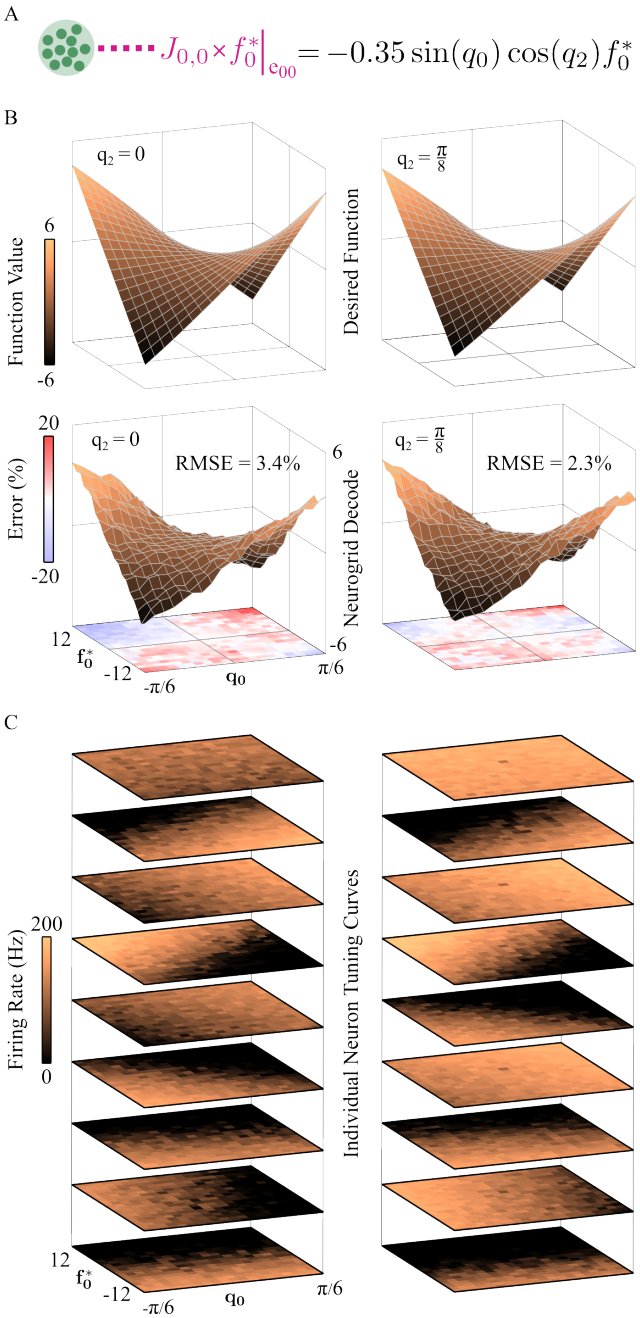

Fig. 3. Mapping Control Functions on to a Spiking Neuron Pool. A. Sub-function to be mapped on to the pool of neurons. B. The target function to be decoded from the pool's activity (top). Neurogrid's decoded function approximates the target function (bottom). Decodes were obtained while varying q_0 and f_0^* with $q_2 = 0$ (left), or $q_2 = \pi/8$ (right). C. Responses of neurons with the nine largest decode weights. Columns match B's columns.

IV. Characterizing Controller Performance

We characterized Neurogrid's control performance using a motor task that involved consecutive reaches to random locations in three-dimensional space—the robot did not return to the zero position between reaches. A single reach lasted three seconds and involved a linearly interpolated end-effector trajectory towards a goal position. The goal was randomly drawn from a uniform distribution. The reaching

task tested Neurogrid's ability to smoothly recompute the Jacobian—a complex nonlinear function—across arbitrary trajectories that span the robot's different configurations.

A. Reaching and Drawing with Neurogrid

Since our control framework accepts force commands, we generated force trajectories for the reaching task by implementing a proportional-derivative control law on the computer:

$$f_x^* = k_p^T \left(x_{\text{des}}(t) - x_{\text{curr}}\right) - k_v^T \, \dot{x}_{\text{curr}}, \tag{4}$$

where k_p and k_v are the proportional and derivative gains, x_{des} is the desired end-effector position, and x_{curr} and $\dot{x}_{\text{curr}}$ are the current end-effector position and velocity.

We required each reach to be followed by a three-second hold at the goal position to test Neurogrid's control stability. Being able to maintain a stable position at arbitrary points in space indicates a lack of spatial bias along with a baseline torque noise level below the friction threshold.

We also used a planar drawing task to test Neurogrid's ability to operate in a plane subject to random user-induced perturbations (see video demonstration). Remaining stable while the user applied large (~5 cm) perturbations to the drawing surface demonstrated the time-delays (millisecond-scale) due to Neurogrid's slow neurons do not destabilize force control. Since we did not use direct force feedback in our control loop, we reduced the vertical axis gains during this task to increase compliance.

B. Increasing Performance with more Neurons

We tested how the Neurogrid controller's performance varied with the number of neurons. Our goal was to find the least number of neurons required to achieve trajectory tracking within a small percent error of an analytical task-space controller implemented on the computer. To do so, we exponentially increased the number of neurons in each neuron pool (64, 128, and 256 neurons per pool) and computed the root-mean-square trajectory errors (Fig. 4). With 256 neurons per pool, the Neurogrid controller tracked end-effector trajectories with 5.6, 8.5, and 5.2% position error (x, y, z), which were similar to the analytical controller's 5.0, 3.4, and 4.8% errors.

The Neurogrid and the analytical controller's performance were limited by the lack of an inertial model. Without an inertial model, the maximum control gains that do not destabilize the robot are a nonlinear function of the robot's configuration. This forced us to use conservative values for the controller's proportional gains while keeping the system marginally overdamped. The low conservative gains resulted in residual tracking errors during reaches. While adding an integral control term could compensate for such errors, we avoided doing so in order to get a deterministic estimate of Neurogrid's performance compared to a time-invariant controller.

The limits on control gains due to the lack of an inertial model also diminished the Neurogrid controller's ability to overcome the Neurobot's anisotropic friction. As such, the

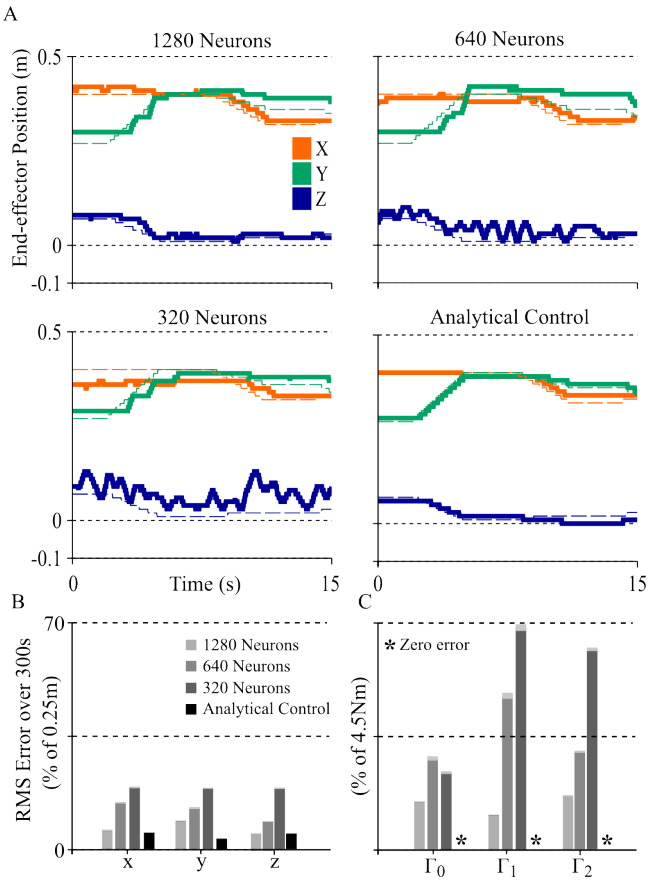

Fig. 4. Controller Performance. A. End-effector trajectory tracking accuracy increased with neuron count. The analytical controller's performance is shown for comparison. Solid lines are executed end-effector trajectories; dashed lines are desired trajectories. B. With 256 neurons/pool (1280 neurons total), Neurogrid's trajectory tracking root-mean-square error (RMSE) approached the analytical controller's. Medians and bootstrapped 95% confidence intervals are shown (grey highlights). C. Neurogrid's torque error, computed using the analytical controller's torques as a reference, also decreased with neuron count. The relatively larger torque errors were due to high frequency spiking noise, which is low-pass filtered by the robot's physical dynamics.

controller produced constant offsets during the hold period that followed each reach. Moreover, the effect of gravity at Neurobot's joints changes abruptly while moving the end-effector across a gravity-balanced point on the x-axis. This abruptly changes friction and leads to control errors during the trajectory-tracking task.

V. Conclusion

We presented the first demonstration of a neuromorphic hardware system controlling an articulated robot. Our system, Neurogrid, controlled a three degree-of-freedom robot (Neurobot) in free-space, and in contact with a surface that was actively perturbed by a human. These results demonstrate the feasibility of implementing nonlinear feedback control on analog hardware. Our force controller only used 1280 out of Neurogrid's million neurons, which suggests that Neurogrid can potentially control eight hundred Neurobots simultaneously in real-time—achieving 3.75 mW/robot. Neurogrid is suitable for robot control in interactive environments

TABLE I
MAPPING Γ TO NEURON POOLS

Pool	Γ_0	Γ_1	Γ_2
0	$-0.35\sin(q_0)\cos(q_2)f_0^*$	0	$-0.35\cos(q_0)\sin(q_2)f_0^*$
1	$0.225\sin(q_0)\sin(q_1)f_0^*$	$-0.225\cos(q_0)\cos(q_1)f_0^*$	0
2	$-0.225\cos(q_0)\sin(q_1)f_1^*$	$-0.225\sin(q_0)\cos(q_1)f_1^*$	0
3	$0.35\cos(q_0)\cos(q_2)f_1^*$	0	$-0.35\sin(q_0)\sin(q_2)f_1^*$
4	0	$\sin(q_1)(0.225f_2^* - 1.026)$	$\cos(q_1)(-0.35f_2^* - 1.612)$

where trajectory-tracking position errors of ~1 cm are acceptable.

Three fundamental challenges remain in order to generalize our spiking neural control framework to humanoid robots. First, mapping algorithms that compute task-space inertias in order to decouple task-control from the robot's nonlinear dynamics [9], which will improve control performance. This involves computing a quadratic form over each robot-link's center-of-mass Jacobian (i.e., $J_{\text{com}}^T M_{\text{com}} J_{\text{com}}$), which will double the complexity of individual control terms (i.e., six-dimensional neuron pools instead of three). Second, simultaneously controlling the force *and* torque that the end-effector applies, which is necessary to maintain contact with arbitrarily shaped surfaces. This involves computing functions over the group of rotations because orientations are not commutative (i.e., not a vector space like positions), which further increases the controller's complexity. And third, simultaneously controlling multiple tasks, which is necessary for humanoids (e.g., walking while holding a cup of coffee). This involves projecting one task into the acceleration null-space of the other [13], which involves computing the Jacobian's inertia-regularized generalized inverse. These challenges set immediate goals for future neural robot control research.

APPENDIX

Factorizing the Control Equation

The control equations given in (Eq. 1) are restated here

$$\Gamma = J_x^T f_x^* + \Gamma_g,$$

where

$$J_x = \begin{bmatrix} J_{x,0} & J_{x,1} & J_{x,2} \end{bmatrix}$$

with

$$J_{x,0} = \begin{bmatrix} \sin(q_0)(-0.35\cos(q_2) + 0.225\sin(q_1)) \\ \cos(q_0)(0.35\cos(q_2) - 0.225\sin(q_1)) \\ 0 \end{bmatrix}$$

$$J_{x,1} = \begin{bmatrix} -0.225\cos(q_0)\cos(q_1) \\ -0.225\cos(q_1)\sin(q_0) \\ 0.225\sin(q_1) \end{bmatrix}$$

$$J_{x,2} = \begin{bmatrix} -0.35\cos(q_0)\sin(q_2) \\ -0.35\sin(q_0)\sin(q_2) \\ -0.35\cos(q_2) \end{bmatrix}.$$

The gravity compensation term is given by

$$\Gamma_g = -\sum_{i=0}^{4} m_i J_{\text{com}_\text{i}}^T g,$$

where

$$\sum_{i=0}^{4} m_i J_{\text{com}_\text{i}}^T g = \begin{bmatrix} 0 \\ 1.026\sin(q_1) \\ 1.612\cos(q_2) \end{bmatrix}.$$

We see that the Γ is a function of 6 inputs—3 joint angles q_0, q_1, and q_2 and 3 forces f_0^*, f_1^*, and f_2^*. However, after multiplying through, expanding, and breaking Γ into terms being added or subtracted, each term is only a function of 3 or fewer inputs. There are 5 combinations of 3 inputs that comprise the terms in Γ being added or subtracted together, and we map these combinations to the groups of neurons on Neurogrid (Table I).

Neurogrid

Neurogrid [1] contains 16 Neurocores, each with 65,536 neurons, for a total of over 1 million neurons. The Neurocores are arranged in a tree. An asynchronous digital router communicates spikes between Neurocores and to/from the computer [14]. The computer connects to one of the Neurogrid tree's leaves through a programmable logic device via USB. An encoder FPGA is situated at a different leaf. A decoder FPGA facilitates weighted connections between neurons, and is situated at the root of the tree.

Neurogrid's neurons are implemented using subthreshold analog circuits [15]. Several modes of operation allow for different neuron models to be implemented. As neuron implementation is as an analog circuit, there is no update schedule for the neuron dynamics equations: the state variables are updated continuously according to the laws of physics. We programmed Neurogrid to implement quadratic-integrate-fire neurons with dynamics equation $\dot{x} = x^2/2 - x + x_0 + g_e(e_{\text{rev,e}} - x) + g_i(e_{\text{rev,i}} - x)$, where x is the soma voltage, x_0 is a bias current, and $e_{\text{rev,e}}$ and $e_{\text{rev,i}}$ are the excitatory and inhibitory reversal potentials for our two shared conductance-based synapses, which control g_e and g_i [16].

Each pool outputs spike streams that are then decoded by the decoding FPGA. To recover the rate of each stream, which is associated with a commanded torque, these spike streams must still be passed through a low-pass filter that approximates the rate (the computer applies a decaying

exponential filter). Since the spikes arrive as part of a USB bulk transfer, the filtering software must maintain a buffer of spikes deep enough so that it is never starved in the event of a long interval between bulk transfer receipt. This would cause the commanded torques to fall to zero for a short time, making control difficult. In maintaining this buffer, we must thus inject a worst-case USB latency into our control loop.

In addition to the decoding FPGA, a second encoding FPGA generates a set of spike streams whose rates are proportional to the robot's state and commanded forces; the computer sends packets that update these rates. In this case, the FPGA stores a $N \times M$ encoding matrix for each pool, where M is now the dimensionality of the domain of the function computed by the pool. We populate the encoding matrix so that each row is a neuron's encoding vector. Thus, if x is the 3-vector of angle/force input values to a pool, a neuron with encoding vector e receives a spike rate proportional to $e^T x$, which implements the preferred directions of the neurons.

A future improvement will be to do the low-pass filtering on the FPGA itself, so that recovering the commanded torques from Neurogrid only injects the average-case USB latency. A more complete set of improvements removes the computer from the loop entirely, having the FPGA communicate directly with Neurobot over EtherCAT. These improvements will eliminate latencies due to Neurogrid-computer communication.

Neurobot

Our experiments used Neurobot, a three degree-of-freedom lightweight robot designed to draw shapes on a plane using force control. The robot uses brushless EC60 flat motors and 4096 counts-per-revolution quadrature encoders, and is driven by a set of three Synapticon Somanet nodes. The nodes receive torque commands and return joint angles over Ethercat at a control rate of > 500Hz. Motor responses are linear beyond control torques of 0.2Nm.

ACKNOWLEDGMENTS

We would like to thank Chris Aholt for helping build Neurobot, Scott Bertics for developing the driver interface, and Kevin Chavez for testing our control decompositions using the Nengo ($www.nengo.ca$) simulator.

REFERENCES

[1] S. Choudhary *et al.*, "Silicon neurons that compute," in *Artificial Neural Networks and Machine Learning ICANN 2012*, ser. Lecture Notes in Computer Science, A. Villa, W. Duch, P. rdi, F. Masulli, and G. Palm, Eds. Springer Berlin Heidelberg, 2012, vol. 7552, pp. 121–128.

[2] C. Eliasmith and C. Anderson, *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems*. MIT Press, 2004.

[3] P. Arena, L. Fortuna, M. Frasca, and L. Patane, "Learning anticipation via spiking networks: Application to navigation control," *Neural Networks, IEEE Transactions on*, vol. 20, no. 2, pp. 202–216, Feb 2009.

[4] X. Wang, Z.-G. Hou, A. Zou, M. Tan, and L. Cheng, "A behavior controller based on spiking neural networks for mobile robots," *Neurocomputing*, vol. 71, no. 46, pp. 655 – 666, 2008, neural Networks: Algorithms and Applications 4th International Symposium on Neural Networks 50 Years of Artificial Intelligence: a Neuronal Approach Campus Multidisciplinary in Perception and Intelligence.

[5] A. Bouganis and M. Shanahan, "Training a spiking neural network to control a 4-dof robotic arm based on spike timing-dependent plasticity," in *Neural Networks (IJCNN), The 2010 International Joint Conference on*, July 2010, pp. 1–8.

[6] N. R. Luque, J. A. Garrido., R. R. Carillo, S. Tolu., and E. Ros, "Adaptive cerebellar spiking model embedded in the control loop: Context switching and robustness against noise," *International Journal of Neural Systems*, vol. 21, no. 05, pp. 385–401, 2011.

[7] R. Carrillo, E. Ros, C. Boucheny, and O. J.-M. Coenen, "A real-time spiking cerebellum model for learning robot control," *Biosystems*, vol. 94, no. 12, pp. 18 – 27, 2008, seventh International Workshop on Information Processing in Cells and Tissues IPCAT 2007.

[8] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *Robotics and Automation, IEEE Journal of*, vol. 3, no. 1, pp. 43–53, February 1987.

[9] ——, "Inertial properties in robotic manipulation: An object-level framework," *International Journal of Robotics Research*, vol. 14, no. 1, pp. 19–36, 1995.

[10] C. Eliasmith, "A unified approach to building and controlling spiking attractor networks," *Neural Comput.*, vol. 17, no. 6, pp. 1276–1314, June 2005.

[11] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. Arthur, P. 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, May 2014.

[12] A. Goldberg, Cauwenberghs, "Probabilistic synaptic weighting in a reconfigurable network of vlsi integrate-and-fire neurons," *Neural Networks*, vol. 14, pp. 781–793, 2001.

[13] L. Sentis and O. Khatib, "Synthesis of whole-body behaviors through hierarchical control of behavioral primitives," *IJHR*, vol. 2, no. 4, pp. 505–518, dec 2005.

[14] P. Merolla, J. Arthur, R. Alvarez, J.-M. Bussat, and K. Boahen, "A multicast tree router for multichip neuromorphic systems," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 61, no. 3, pp. 820–833, March 2014.

[15] P. Gao, B. V. Benjamin, and K. Boahen, "Dynamical system guided mapping of quantitative neuronal models onto neuromorphic hardware." *IEEE Trans. on Circuits and Systems*, vol. 59-I, no. 10, pp. 2383–2394, 2012.

[16] B. Benjamin, J. Arthur, P. Gao, P. Merolla, and K. Boahen, "A superposable silicon synapse with programmable reversal potential," in *Engineering in Medicine and Biology Society (EMBC), 2012 Annual International Conference of the IEEE*, Aug 2012, pp. 771–774.