Neuro-musculoskeletal modeling reveals muscle-level neural dynamics of adaptive learning in sensorimotor cortex

Travis DeWolf[1*], **Steffen Schneider**[1*], **Paul Soubiran**[2], **Adrian Roggenbach**[3], **& Mackenzie Weygandt Mathis** ✉[1,3]

[1] Ecole Polytechnique Fédérale de Lausanne (EPFL). Geneva, Switzerland

[2] MPI for Intelligent Systems. Tübingen, Germany

[3] The Rowland Institute at Harvard, Harvard University. Cambridge, MA USA

Abstract

The neural activity of the brain is intimately coupled to the dynamics of the body. Yet how our hierarchical sensorimotor system dynamically orchestrates the generation of bodily movement while adapting to incoming sensory information remains unclear (1–4). In mice, the extent of encoding from posture to muscle-level features across the motor (M1) and primary sensory forelimb (S1) cortex and how these are shaped during learning are unknown. To address this, we built a novel 50-muscle model of the adult forelimb amenable to studying motor control and learning in a physics simulation environment. We show that we can imitate 3D limb kinematics collected during a joystick task by solving inverse kinematics and deriving a sensorimotor control model that drives the same actions. Using the internal computations from our model, we find that populations of layer 2/3 M1 and S1 neurons encode high-level position, and lower-level muscle space and proprioceptive dynamics. During adaptive learning, these functionally distinct neurons map onto specific computational motifs. Strikingly, S1 neurons more prominently encode sensorimotor prediction errors. Moreover, we find that neural latent dynamics differentially change in S1 vs. M1 during this within-session learning. Together, our results provide a new model of how neural dynamics in cortex enables adaptive learning.

Introduction

Seamless execution of skilled actions requires animals to adapt in dynamic environments to maintain performance. While many circuits from the cortex to the spinal cord are active and required for generating effective movements, it remains unclear how neural circuits across the brain contribute to motor learning and adaptation – the ability to learn from perturbations to restore performance. Several studies have shown that primary somatosensory cortex (S1) (4–7) is critical for learning to adapt, while others have argued that motor cortex (M1) is critical for learning a new skill (8, 9). Both insights may be mapped to mechanisms that support rapid plasticity (8, 10, 11) and updating of internal models (4, 12). But in *what* coordinate framework – kinematics, joint torques, or musculoskeletal features –, and *how* the neural dynamics are changing in S1 and M1 during motor adaptation remains an open question (2).

In the 1920's N. Bernstein used motion analysis to demonstrate that while a person can be variable in their trajectory, such as when swinging a hammer to hit a nail, they are quite good at hitting the nail (end goal) (13, 14). He argued against a simple 'one-to-one' mapping of sequences of defined M1 neurons that could consistently drive a trajectory, and (in hindsight) hinted at a population level computations. In the ensuing years, a myriad of studies have shown that single neurons in M1 can encode high-order kinematic features (15–17), although models questioned this conclusion and argue for muscle-level control (2). Dynamical systems models can both explain cortical dynamics in reaching tasks and provide an attractive framework for understanding neural coding (3, 18, 19), but remain disconnected from neuron-level computations. Understanding the latent dynamics may give rise to clues on the sensorimotor transformation (internal models) that ultimately enable learning. Yet marrying this population-level analysis with mechanistic, cellular-level insights during motor learning tasks remains an open challenge (20).

In human motor learning studies ample evidence supports the existence of internal models that guide rapid adaptation. This has been demonstrated in various tasks such as a physical perturbation to the limb or a task that decouples visual feedback to limb or eye movements (8, 21–27). In 2D planar reaching forelimb tasks in humans, upon introduction of a force field perturbation,

* co-first authors. ✉correspondence: mackenzie.mathis@epfl.ch

hand movements are initially deviated from the baseline trajectory. Yet with repeated exposure to the same perturbation, one can compensate for the force field and restore performance similar to the baseline level by, theoretically, computing a sensory prediction error (SPE) that is used to update an internal model (22, 27–32). Note that SPEs could occur in different mechanics reference frameworks (kinematics, dynamics) and update separate internal models and possibly be used for direct policy optimization (33–36).

Several works in non-human primates have begun to explore the single-cell resolution neural basis of motor adaptation, focusing on premotor areas or M1 (8, 37), but none have explored the role of S1, despite evidence of its essential role in driving motor adaption in mice (4) and humans (5). In both primates and rodents, recent evidence points to whole arm encoding during movement (38–40) and S1 has been shown to be immediately active after limb perturbations (41, 42). Collectively, these results suggest S1 could be important for driving motor adaptation. Moreover, the question of cell-type and layer-specific recordings in M1 and S1 during motor adaptation remain to be explored. Given the evidence, in different tasks, for mismatch or prediction error coding in layer 2/3 (43, 44), M1 and S1 are hypothesized candidates for not only driving learning but computing SPEs (4).

Computational models that have sufficient detail of both neural and behavioral data promise to aid in our understanding of sensorimotor control (45). Effectively constraining computational models based on experimental data is a basic premise of scientific research as it yields testable hypotheses and gives us mechanistic insight into complex systems. Models can be constrained at multiple different levels, including the behavioral level (kinematics), the circuit level (neural computations), and single-cell level (encoding properties). An ideal model for studying sensorimotor control thus links not only neural circuits to neural computations, but to the body in the form of musculoskeletal biomechanical models such that they have realistic constraints (46, 47). While recent works have started to develop mouse biomechanical models (48), no adult mouse forelimb musculoskeletal model has been developed, which would be required to accurately study complex skilled reaching and learning in mice.

Thus, we developed a novel *in silico* musculoskeletal-biomechanical model of adult mouse forelimbs that can be controlled under theory-guided computational constraints to match the real kinematic behavior of mice performing a skilled reach, grasp and pull task. In brief, we can control this system by solving the inverse kinematics, developing an analytical solution for generating muscle activations that execute desired joint torques that recapitulate the same behaviors as mice in a complex joystick-based task. Then, we recorded simultaneously from M1 and S1 in layer 2/3 of mice to test the hypothesis that neural dynamics encode the reference frameworks required for updating internal models, and probe these representations during a motor control and within-session, adaptive learning task.

Results

A realistic adult mouse musculoskeletal biomechanical arm model.

Understanding the effect of movement has been important to disambiguate motor signals from sensory or decision-making variables (50–53). However, in sensorimotor control, “movement” or “action” is likely too high-level of a catch-all analysis and is not well aligned to the goal of understanding how motor commands and proprioceptive feedback are generated and adapted during learning. Having a richer understanding of the components of “movement” – 3D position, velocity, force (torques) and even down to the level of muscles – is critical. Indeed, recording from some muscles (with EMGs) has been highly impactful at providing important signals to better understand reaching (3, 19, 54, 55), but ultimately having measurements of all muscles would greatly enhance our ability to better understand what underlying features of “movement” are being encoded in neural dynamics across the sensorimotor hierarchy. Given this is not experimentally possible, realistic musculoskeletal models of the subject in study (i.e., a mouse arm during reaching tasks) would be hugely impactful for better explaining neural data (45).

While it is clear that sensorimotor cortex encode movement, a large limitation for a deeper understanding of which features change during motor actions has been access to physics simulators that allow for more detailed 3D modeling of many muscles and therefore better estimation of feedback and and motor control polices. Notably, musculoskeletal models in humans have been critical for sports science, medical applications (such as modeling for prosthetics), and in robotics, but are less used in neuroscience (46, 47, 56, 57). Exceptions have been to model the role of proprioception (40, 58) or motor control in humans and non-human primates (59–62). Recent progress in developing rodent biomechanical models have already shown promise (48),

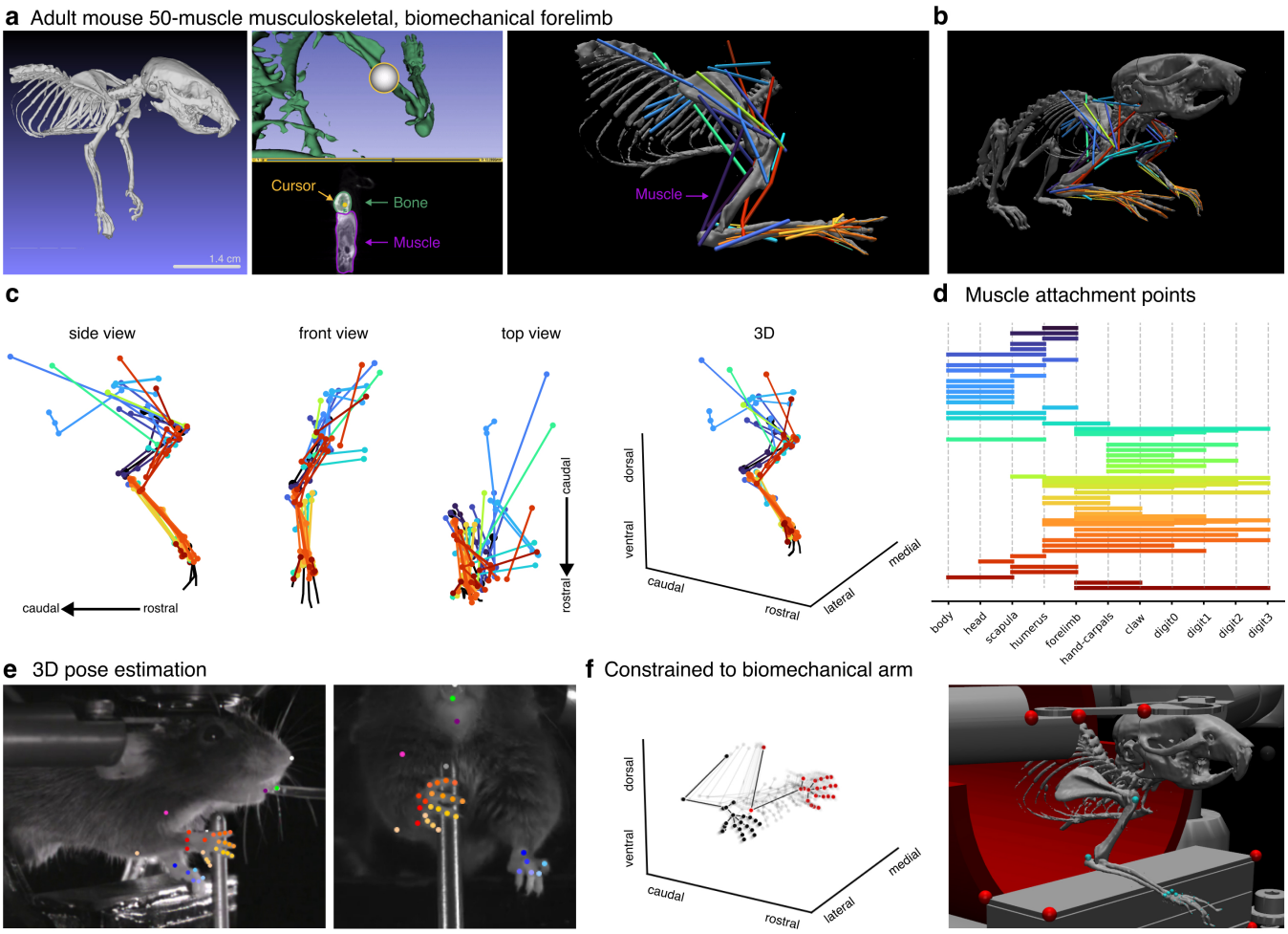

Figure 1. Musculoskeletal biomechanical model of the adult forelimb. **a**, Development of the model: micro-CT scans were acquired in reaching, standing, and t-pose positions (see Methods). Muscle insertion points were manually extracted from CT (right panel) and MRI scans then optimized with a neural network. In total, there are 50 muscles in the MuJoCo model. **b**, Example image of full model of the mouse musculoskeletal biomechanical arms and shoulders. **c**, The full 50-muscle arm, viewed from the side, front, and top view. **d**, Clustering of functional groups of muscles, delineating the scapula, shoulder, elbow, wrist, all digits. **e**, We collected 2D keypoints with DeepLabCut (49) and a refined with a transformer model (see Methods). **f**, Movements of the 3D inferred data across resting position (black) and reach (red), noting the wrist rotation; Right: Fitting the 3D keypoints to the musculoskeletal model and aligned in MuJoCo coordinates.

but a remaining gap has been the lack of development of a full adult musculoskeletal forelimb. The mouse forelimb is extremely complex (63, 64), having 50 muscles that control the shoulder, arm, and hand.

Here, we developed a novel model of adult mouse musculoskeletal biomechanical forelimbs by performing micro-CT scans in various postures (including reaching and grasping as in the task we use), and MRI guided analysis for determining muscle-insertion points in 50 muscles (Figure 1a-d, Suppl. Figure S1a, b; see Methods). Moreover, we developed a method using machine learning to optimize the final placement of the muscles by maximizing the amount of force that can be effected on the arm while recapitulating movements from experimental recordings (Suppl. Figure S1c; see Methods).

Physics-simulator based sensorimotor controller.

Our musculoskeletal model was developed in MuJoCo (47), a physics engine designed for efficient simulation of articulated structures interacting with the environment plus the ability to accurately model tendon dynamics. We aimed to fully model a reach, grasp, and joystick pull task for mice that was shown to elicit motor adaptation within a session (4). Thus, we generated 3D pose estimation data (49, 65) (Figure 1e, f, and Video 1; see Methods) and used inverse kinematics optimizations to identify joint trajectories that best reproduced the 3D pose with our model in MuJoCo (Figure 1f, Video 2).

We developed a novel analytical approach for generating torques and muscle activations that drive movements matching the

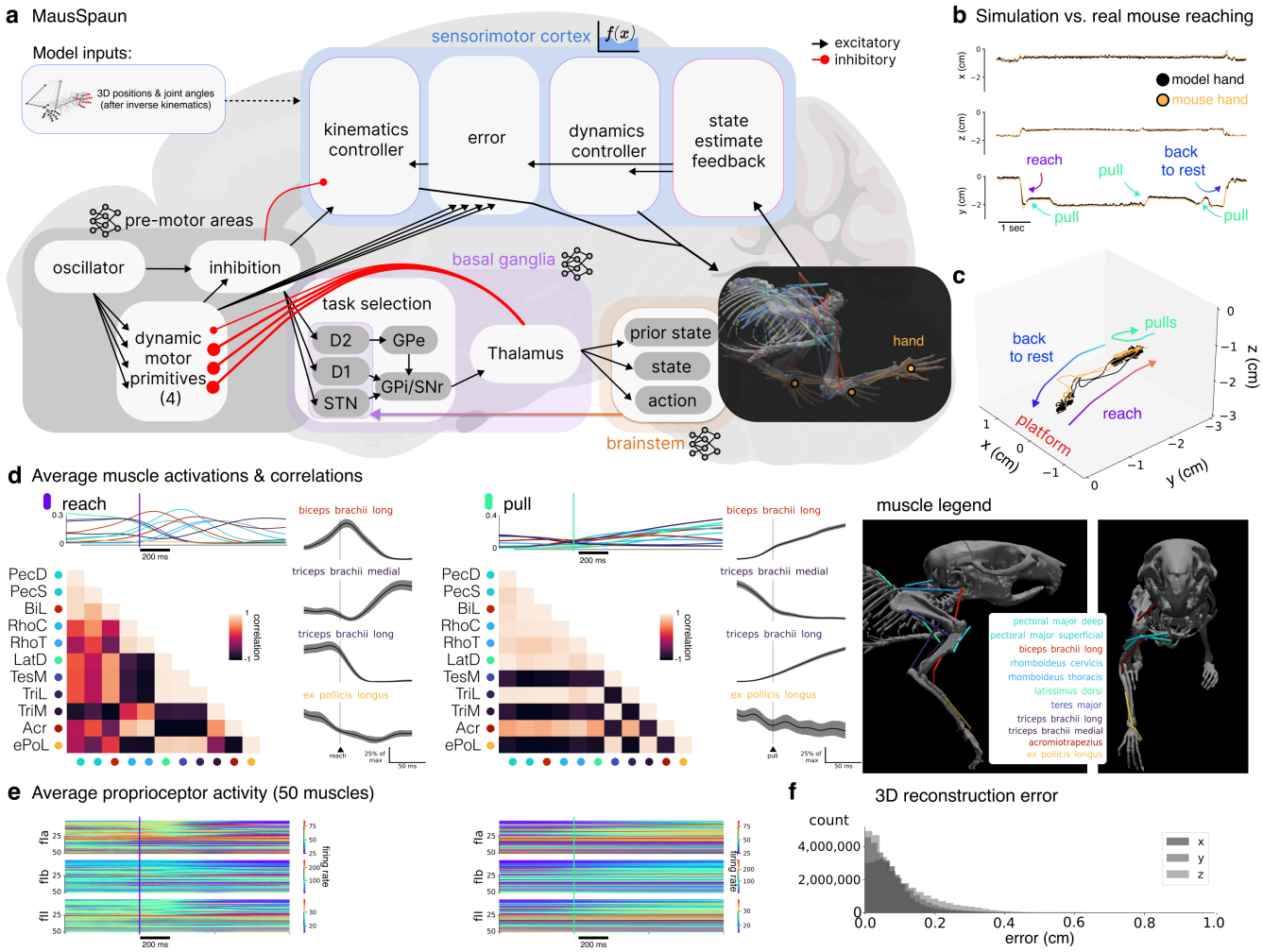

Figure 2. MausSpaun: a large-scale model of motor control in mice. **a**, A diagram of the MausSpaun model, which is a hybrid computational model containing modules divided by associated brain area and implemented with varying levels of biological realism. The kinematics and dynamics controllers, error calculation, and state estimate feedback modules directly execute the mathematical equations of their associated functions (see Suppl. Figure S3a), developed to match behavioral level phenomena (see Methods). The task selection and action selection (brainstem) module is a set of spiking neural networks (SNNs) inspired by basal ganglia and brainstem functionality. The model can be set to control joint torques and generate muscle-activations to drive the 50-muscle arm model for full musculoskeletal control. **b**, We simulate the same reach, grasp, pull task that real mice perform; here is an example reproducing a 6 second sequence from experimental data, showing the x, y, z trajectories of the hand, as well as in **c**, a 3D plot of the action. **d**, We compute the average muscle activations in a set of anatomically-grouped muscles (key on the right) during stereotyped reach (n=60) and pull actions (n=365, see Methods). Extended muscle group results can be found in Suppl. Figure S4. **e**, Muscle spindle (fIIa, fII) and GTO (fIb) outputs (from all 50 muscles) for the same time epoch as in **d** is shown. **f**, Quantification of the 3D simulated movements vs. real mice movements across all actions (n=26 sessions; 139,624,600 time steps in total).

recorded behavioral data in our joystick task setting (Figure 2a, see Methods, n=26 sessions). We did not want to solely model the sensorimotor system of our digital mouse as an unstructured series of neural networks, but rather model a control system using the theoretical principles we believe underlie the biological functionality (66, 67), which affords us interpretability. We implemented a control theoretic approach to control the arm model (Figure 2a, Suppl. Figure S3a), extending a generalized coordinates controller (68) to generate muscle activation signals in order to reproduce observed limb and hand kinematics.

In brief, we control our muscle-based arm model by generating joint torques using a proportional-derivative (PD) controller to follow a pre-planned joint angle trajectory (while accounting for the forces of inertia and gravity), and transform these forces into muscle activations. This transformation is performed through a dot product calculation with the Jacobian between muscle space and joint space. The resulting forces exerted on the arm are the best approximation of the torques specified by the joint space PD controller, given the arm's muscle configurations and current states, and the precise attachment points we found to be critical for reliable force control (see Methods, and Suppl. Figure S1d-f).

We developed a hybrid computational model based on specific algorithmic hypotheses about the underlying function of brain

regions (see Methods). We model the series of kinematics and dynamic controllers based on features of sensorimotor control (15, 69) optimized to approximate the ideal mathematical formulations. In order to animate the model, we used the Neural Engineering Framework (NEF; 70) to design a series of inputs generated by spiking neural networks to the analytical controllers based on known anatomy and function. This approach can be considered a 'white-box', mechanistic approach, where the underlying computations performed by the system are well-understood, without the standard trappings of machine learning methods (Figure 2a).

Specifically, we built "pre-motor" areas that provide both inhibition and motor primitives to our analytical controller ("cortex", based on experimental works; 19). This receives input from task-selection and action-state areas, modeled based on findings showing clear action-tuning in the brainstem (71, 72) (see Methods). These components are implemented as spiking neural networks (SNNs, see Methods). Our novel approach also allows us to *in silico* record internal control signals – from kinematics to muscle-level control and feedback – and record from our algorithmically-guided SNNs during imitated or simulated movements (Figure 2b). Collectively, we refer to our musculoskeletal biomechanical mouse model and the control-theoretic plus neural-analytical controller as "MausSpaun" (Figure 2a, Video 3).

We validated our model by computing the model vs. real-mouse trajectories and showing that they are highly similar (mean hand error in x is 0.08 cm ± 0.07 (STD); y = 0.08 cm ± 0.08, z =0.10 cm ± 0.11; for reference, the forearm length is 1.3 cm (Table 1; Figure 2b-f)). We computed the correlation of antagonist and agonist muscle groups based on example electromyographic (EMG) recordings in mice, rats, cats, and primates (55, 63, 64, 73, 74). Namely, we grouped the muscles (63, 64) (see Methods) and computed the correlation of co-activation during limb movements (Figure 2d, Suppl. Figure S4). We also extracted the outputs of the proprioceptor models for use in downstream analyses (Figure 2e). Next, we compared mouse forelimb EMG data (55) and confirmed expected results (i.e., muscles such as biceps should contract, triceps relax during pulling; Suppl. Figure S5), but given not all muscles have been experimentally recorded in mice, others stand as predictions (to note, many are very small and current EMG technology would prevent their recording).

Encoding of kinematics and muscle-level control during forelimb reaching.

Mice have emerged as an excellent model system for neural circuit studies during reaching and handling (4, 55, 71, 75–83). It has been shown that in mice M1 approximately 40-50% of the L2/3 excitatory neurons are tuned to events surrounding motor tasks, such as licking or lever pressing (77, 84, 85) and in S1 (and even in M1) they encode peripersonal proprioceptive space during passive movements (39). But it remains to be clarified how these neurons are related to kinematic or dynamic features during skilled volitional motor tasks such as reaching and pulling, or during motor adaptation.

We examined the activity of 9,268 excitatory L2/3 neurons using two-photon single cell resolution mesoscope imaging (86) simultaneously in the contralateral M1 and S1 forelimb areas of mice performing a joystick task (4) (Figure 3a-e). S1 was previously shown to be essential for learning to adapt, and M1 was required for dexterous grasping in this task. Briefly, thirsty mice were placed in the dark and trained to reach out and grab a joystick that had 2-degrees of freedom. They learned to pull the joystick towards their body into a virtual target box positioned approximately 8 mm from the joystick start location in order to receive a liquid reward. They became experts as evidenced by consistent pulls that had bell-shaped velocity profiles (Figure 3b). During the behavior we tracked the mouse using 3D DeepLabCut (49, 65) to extract posture and movement (Figures 1e, 3b, see Methods).

We computed behavior-triggered trial averages across all neurons (Figure 3f), which showed that M1 and S1 had clear movement-related activity. Given the clarifications needed regarding encoding of high-order "movement" tuning and lower-level muscle features in M1 (2, 19), and our lack of knowing specifically in the mouse which is most prevalent, we next aimed to measure the tuning of individual neurons across sensorimotor forelimb cortex.

We designed Generalized Linear Models (GLMs, Figure 4a-c, Suppl. Figure S6a; see Methods) that included the task-relevant and kinematic features. Given that kinematic features can be correlated, we thematically clustered regressors to measure their relative effects. Specifically, we built a global design matrix of groups that captured the 3D position & velocity (group 1), joint position & velocity (group 2), joint torques (group 3), muscle-space activations (group 4), muscle length & velocity (group 5), and proprioceptive signals (muscle spindles & Golgi tendon organs, group 6). This allowed us to directly test the hypotheses that mice sensorimotor cortex represents high-level and lower-level features (2, 19).

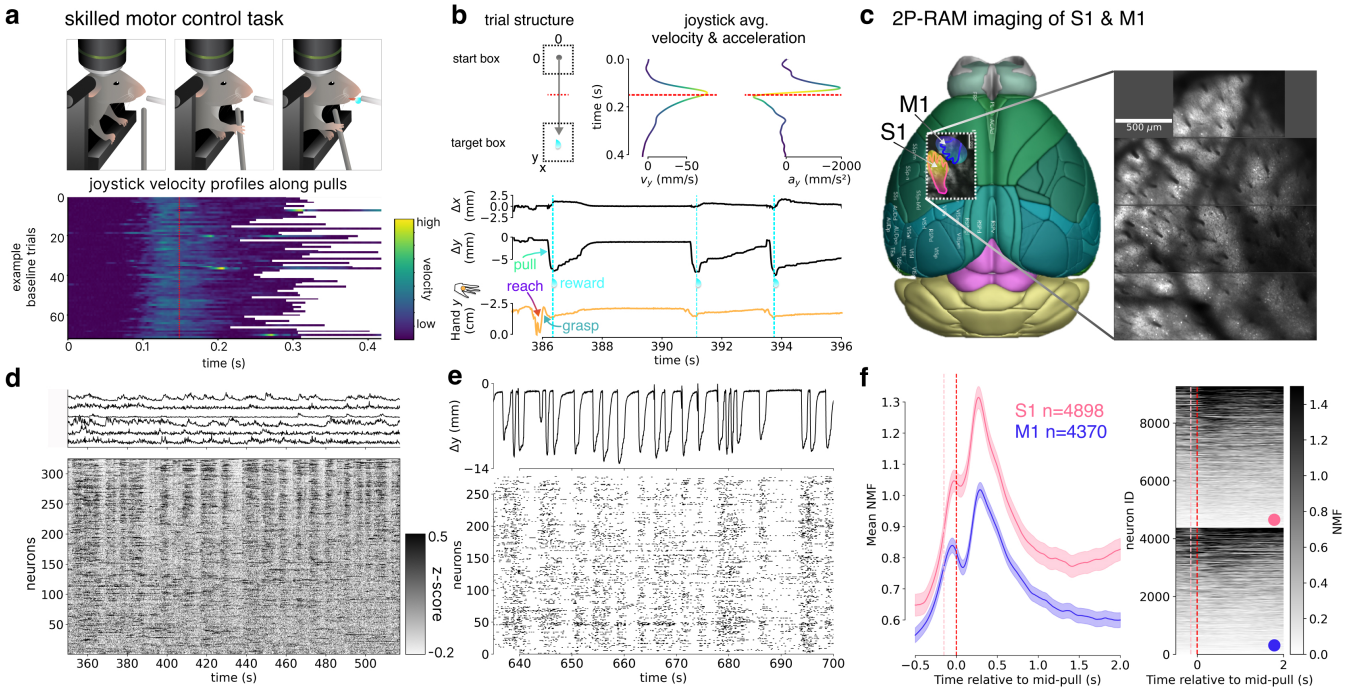

Figure 3. Activity of neurons in M1 and S1 during forelimb reach, grasp, and pull task. a, Schematic of the experimental setting showing a head-fixed mouse engaged in a reach, grab, and pull task, the session structure, with example velocity plots from a single session, aligned to a positional bin (highlighted in red). Cartoon by Julia Kuhl. **b**, Trial structure, average velocity (mean ± SEM), and acceleration (mean ± SEM) profiles across trials (n = 6 mice, 22 baseline sessions, n = 4,667 trials), and example joystick and hand trajectories from a individual mouse below. **c**, Example two-photon imaging area with the 2P-RAM system, covering forelimb S1 and M1 with example scan fields in a behaving mouse. Average intensity projection of four imaging fields recorded simultaneously at 5.8 Hz. **d**, A z-scored recording of ~400 neurons within a single baseline session, with example $\Delta F/F$ traces for approx. 1 min. **e**, Example deconvolved and binarized traces for neurons shown as a rastermap, plus joystick trajectory, for approx. 1 min. **f**, Total count of M1 and S1 neurons co-recorded, and peri-stimulus time histogram aligned to trial-based movement (red line, as in **a**, which is nearly peak-velocity) for all M1 and S1 neurons recorded. Right: heatmap of all neurons recorded in S1 and M1, order ranked by activity. The pink line denotes the estimated pull-onset.

We simulated each real-mouse session with MausSpaun in order to read-out 3D kinematics, internal muscle-control dynamical features, and proprioceptive feedback in the same *in silico* framework. For the GLMs we computed a cross-validated pseudo-R^2 and only further considered neurons with performance over 0.025 (Suppl. Figure S6a-c), which accounted for 49.5% of the population (Figure 4d). S1 had slightly more tuned neurons than M1 (S1: 50.7% vs. M1: 48.2%). Of the significantly tuned neurons (in both areas), high-level kinematics (3D position, joints), lower-level muscle-space (force, length, velocity) and proprioception were the highest ranked (groups: 1 (28.8%), 2 (28.3%), 4 (9.4%), 5 (8.1%), 6 (18.6%); Figure 4d), and joint torque tuning was the least represented (group 3, 6.9%). The unique contribution GLMs had the same top groups – meaning if we post-hoc grouped the regressors into groups we find similar results (Suppl. Figure 4d). Thus, we consider our group-based GLMs further.

Considering 3D kinematics we found most tuned neurons encode position (82.0%), and dominantly so for distal body parts (100% of the position was for hand and wrist), while 13.5% encode velocity, and 4.5% showed mixed selectively (Figure 4e). Notably, we did not find significantly tuned neurons for more proximal body parts (such as elbow or shoulder), and across M1/S1 there were minimal differences in tuning (Suppl. Figure S6e). Given the high fraction of distal (hand and wrist) tuned neurons we next asked, do neurons encode joint angles, angular velocity, or joint torques? Interestingly, joint torques had the fewest tuned neurons (group 3, Figure 4d). Yet, many neurons (28.3%) encoded joint angles (group 2, Figure 4d, e).

Next we examined groups that include muscle dynamics and proprioceptive feedback. We use models for two spindle types (fIa, fII) and one for Golgi tendon organs (fIb, see Methods). A fraction of neurons (8.1%) were tuned for forelimb muscle length and velocity, or activations (force; 9.4%). The most represented muscles were synergistically related to moving the arm in the major reach and pull axis (Figure 3d, e, f), namely, muscles for rotating and grasping with the forelimb and the hand (Figure 3e, f). However, many more neurons were tuned to proprioceptive signals (18.6%) for forelimb muscles (Figure 3e), with minimal differences across the areas (Suppl. Figure S6e). Taken together this suggests that perhaps the CCF forelimb area

(fS1 and part of M1 (87)) in mice is dominated by muscle and related proprioceptive, distal features, akin to Brodmann's area 3a in primates (17, 88).

Overall, our data suggests that the high-level egocentric 3D hand position and lower-level proprioception and the muscles that support forearm and hand movements were well represented across sensorimotor cortex. Collectively, these signals would be essential for effective state estimation – knowing where in 3D space one's body is, plus knowing the muscle-space that controls dynamically moving through these spaces. This additionally motivated us to consider neural population dynamics in relation to these key features, as they are most likely the substrates for control and learning.

Neural population dynamics encode muscle-level features.

Thus, we leveraged a new self-supervised learnable latent dynamics approach called CEBRA (89) (Figure 4f) to train encoder models across all neurons jointly with specific group features (Figure 4g). All groups showed better than chance goodness of fit,

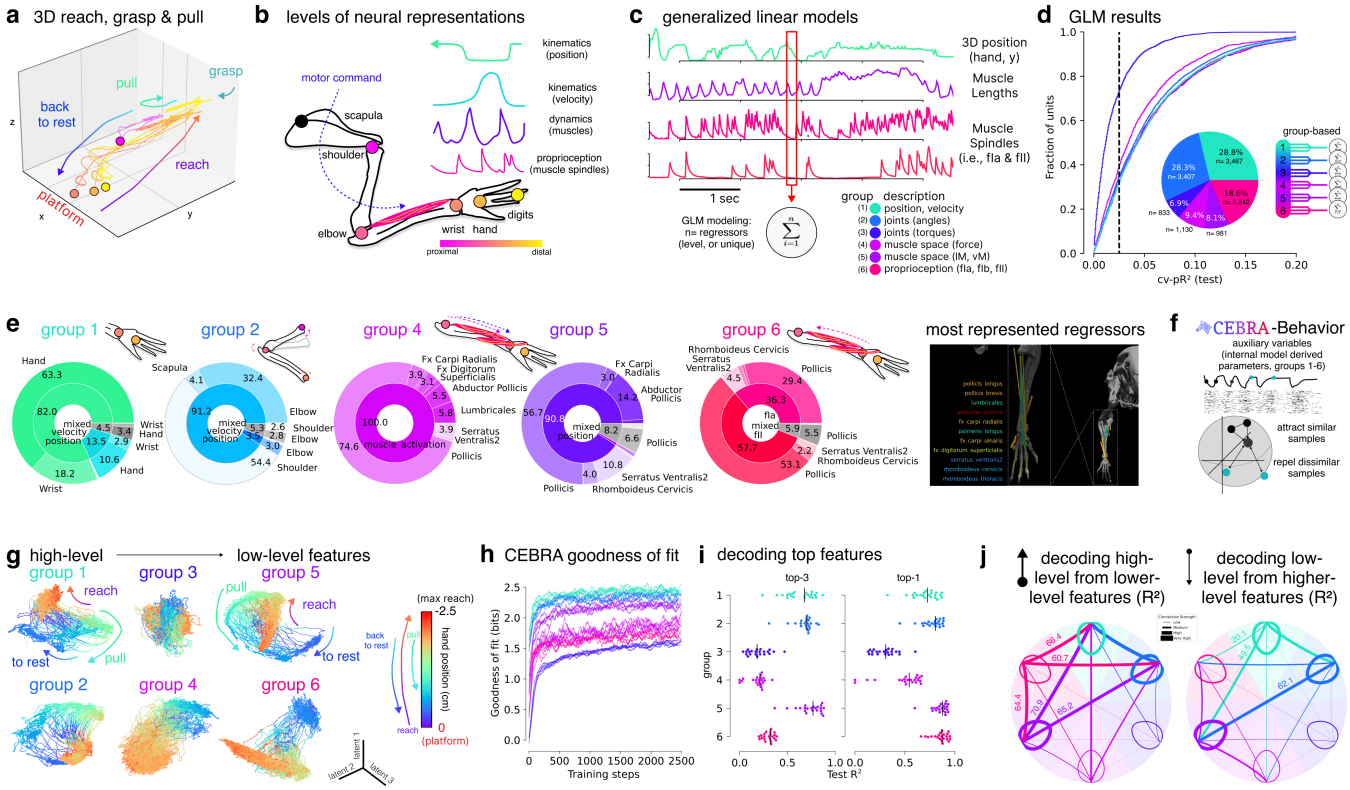

Figure 4. Sensorimotor cortex signals 3D position, muscle-level control, and proprioception. a, 3D plot of an example reach, grasp, and pull (and retraction to platform) by a mouse, highlighting the wrist, hand, and finger (colors as in **b**). **b**, Our 3D data allows us to extract kinematics and dynamics across joints (and with our model mouse, muscles). **c**, Diagram of the GLMs design matrix (with only example regressors such as hand, muscle length, and spindles) for measuring unique contributions and grouped by major levels (see Table 6 for full description). **d**, Cumulative variance explained vs. pseudo-R^2 for all neurons, showing our cutoff of 0.025 for downstream analysis. Inset pie chart shows the percentage of neurons tuned for a specific level (where any score over 0.025 is counted for that level and then we normalize to equal 100%). **e**, Pie chart demonstrating the tuning of neurons, considering level 1, 2, 5 and 6: in the inner ring is the level factors (position and velocity, joint angles, muscle length and velocity, and proprioception, respectively) and the outer ring is the specific body parts that are significantly tuned. Each ring includes only significantly tuned neurons. Far right: images of the muscles most represented in sensorimotor cortex of our mice during this task. **f**, Diagram of CEBRA, a self-supervised nonlinear method for joint behavioral and neural embeddings. **g**, 6D CEBRA embeddings trained across all neurons with auxiliary variables selected based on GLM results from each level (see Methods), with the top-3 latents shown. Embeddings are posthoc color coded based on the hand position, where 0 is closest to the platform they stand on. Major actions (reach, pull, back-to-rest) are annotated in group 1 and on the color bar to aid in visual understanding and interpretability. **h**, We computed the goodness of fit for each level for our CEBRA models (n=5 runs each), which showed that at the population level 3D kinematics and muscle-features could be much better explained than joint torques. **i**, We use the CEBRA encoder embeddings to perform decoding of the the most significantly single-neuron tuned features, such as 3D hand position (level 1), across all levels (see Methods, test R^2 is shown). Nicely, we could decode kinematic, muscle, and proprioceptive features. **j**, We also perform cross-level decoding to explore the relationship between the high-level and low-level features. The left plot shows when we take a CEBRA-model trained on a given group, i.e., group 5 or 6, how well can we decode another group, i.e. 1. The plot on the right shows the inverse relationships, i.e. if we decode from group 1 into group 6, there is very low decodability. We also show the within group decoding performance as the within group loops. The color of the line denotes the source, the thickness denotes the performance (thicker is higher, and key R^2 values are denoted).

with high-level kinematics and low-level muscle features driving the best fits (Figure 4h). This suggests that while many single neurons could show pure or mixed selectivity for specific regressors in position and muscle-space (but less so in joint space), at the population level (with a nonlinear encoder), joint-control (group 3) could be represented (albeit not as well). Theoretically, joint-control could be derived from these other groups in a downstream manner (such as in layer 5), which could serve to explain differing results across studies which often did not report which layer or cell type is being recorded (2, 3, 15, 69).

To test this hypothesis we performed decoding of the top three features within a group per session (examples in Figure 4e), which showed high decodability from the population dynamics for groups 1, 2, and 5. There was less decodability for 6 if considering the top-3, but for top-1-regressor it was as high as groups 1 and 2. For either top-3 or top-1, the lowest decoding performance was for groups 3 and 4 (Figure 4i), which confirms that the dominant information in M1 and S1 is 3D position, muscle-level control signals, and muscle-level feedback (proprioception). To test how information is potentially shared across neural latent dynamics, we asked if we could decode these features across embeddings that were trained within a group. Namely, could we decode 3D position of the hand from the modeled proprioceptive signals or joint torques? Strikingly, we could decode the 3D hand position from the muscle and proprioception signals, but not the inverse (Figure 4j). Nor could we decode the joint torques from any feature, including position or muscle-space, even within the nonlinear encoder-decoder setting.

Sensorimotor cortex encodes prediction errors.

Having established that sensorimotor cortex in mice richly encode 3D kinematic and muscle-dynamic features of limb control, we next leveraged a within-session learning task to study how it might support adaptation. Within-session learning is a powerful way to understand the neural substrate that supports rapid learning, with the confidence one is recording from the same neurons. Moreover we previously established a joystick task which we know requires S1 to learn and M1 to execute (4, 90).

After mice became experts at the baseline task we challenged them in a single session to adapt to an external perturbation to their hand-path trajectory (Figure 5a). After 75 baseline trials a force-field was unexpectedly introduced that perturbed their hand path orthogonal to the pull axis. Over trials mice could adapt their motor outputs, as measured by a reduction in perpendicular displacement (Figure 5a, b). Note, here we use a delayed-onset force-field meaning that mice (and humans (91)) cannot learn to pull straight, but an ideal reduction in displacement still allows them to end up in the reward zone (Figure 5a). They also displayed an aftereffect, showing evidence of having an updated internal model, as previously reported (4), and the data was well fit by a state estimator model that relied on sensory prediction errors (see Methods, Figure 5b, c).

Our optimal state estimator model makes concrete predictions about latent computations that could drive learning (Figure 5d). Specifically, the model makes predictions about prediction errors, a state estimation signal, and the command signal that would be required to adapt across trials (Figure 5d). Upon close inspection of our data, we find the neural population during this task significantly increase their firing during early perturbations and in early washout compared to baseline firing rates, although in washout their movement trajectory returns to baseline trajectories (Figure 5e). A change only in firing during perturbation could reflect a surprise signal, but here we see the signal change across learning, and again change in washout. This is akin to prediction error neurons reported in primary visual cortex (43) and in the vestibular system (92), but to our knowledge, never reported in M1 or S1 during motor learning. This suggests M1/S1 encodes *sensorimotor* prediction errors and is involved in updating of an internal model (t-test, baseline (b) vs. early perturbations $p < 0.0001$; b vs. mid-perturbations 0.0008, b. late perturbations 0.0001, b vs. early washout 0.005. b vs. late washout 0.665 Figure 5e, Suppl. Figure S8b-f). Notably, both M1 and S1 neurons showed this effect (Suppl. Figure S8a-f).

The population showed signatures of sensorimotor prediction errors, but we aimed to discover if other functional groups were present in the data. Using the PSTH per neuron across binned trials (as in Figure 5e; see Methods), we performed PCA and hierarchical clustering (Figure 5f). We found we could map these functional clusters to specific model computations as hypothesized in Figure 5c. Sensorimotor prediction error dynamics – which showed an increase in firing during early perturbation, a decrease over learning, and again a change in firing during washout – were in three clusters. Each cluster (0, 3, 4) had slight variants on this overall signature, showing both predicted sign changes at washout (increase and/or decrease; Figure 5g). Notably, these clusters accounted for 24.8% of the neurons we recorded (n=421), and most were in S1 vs. M1 (n=254 S1; Figure 5f, g).

We also found a cluster that could be mapped to a command-like (u) signal – which presents as an increase over learning and a decrease in washout (cluster 1, Figure 5d). These were in only 4% of the neurons (n=65), and also mostly in S1 (n=50). Lastly,

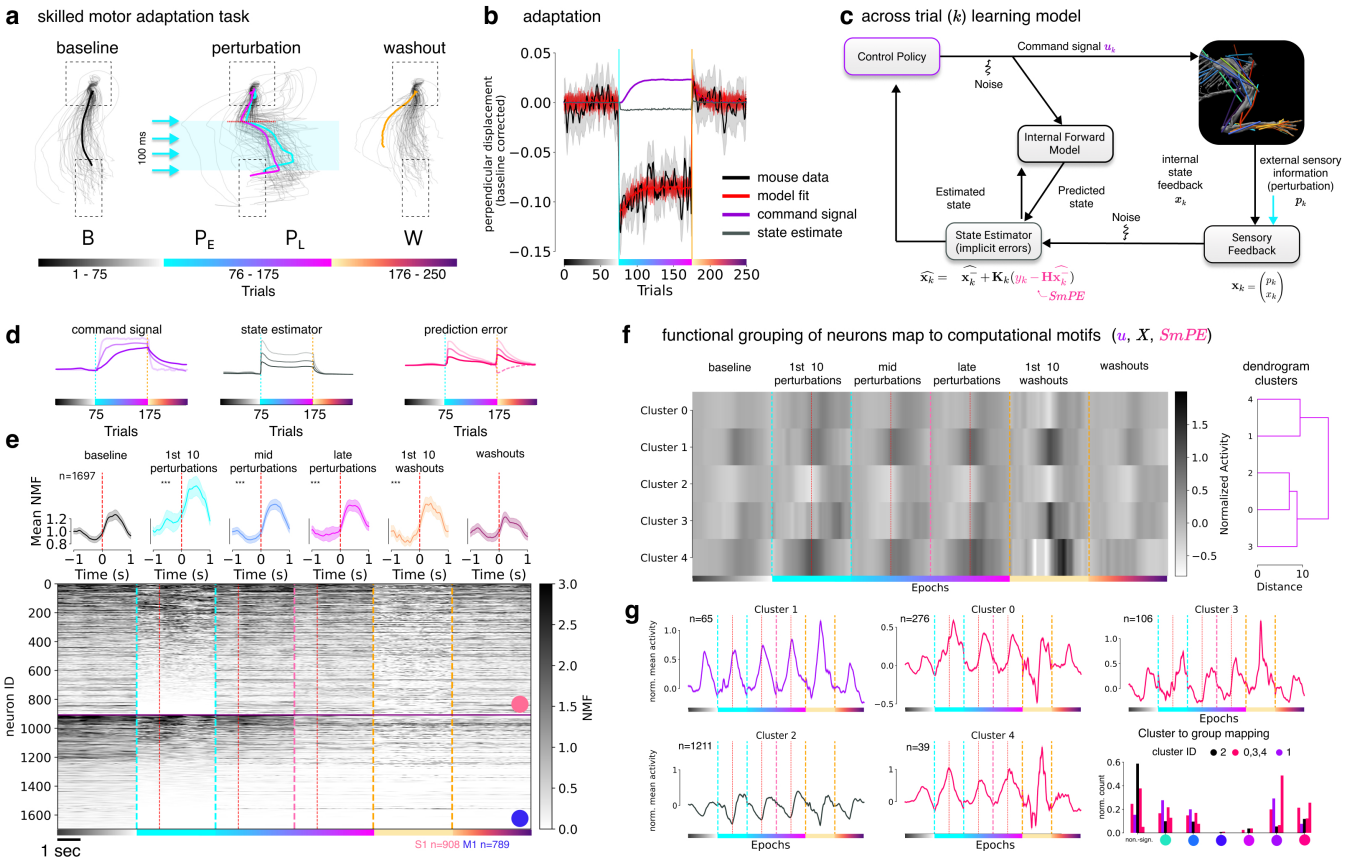

Figure 5. Sensorimotor cortex encodes sensorimotor prediction errors. **a**, An example adaptation session showing pull trajectories; baseline average trajectory in black, avg. first 10 perturbations in cyan, last 10 in pink, avg. first 3 washout trials, and otherwise all trajectories per block are shown. **b**, Perpendicular displacement across trials (n=4 mice, avg. $\pm$ SEM), showing reduction in error during learning and an after-affect in washout. Data is smoothed (within block) with an Savitzky-Golay filter (polynomial order 3, window length 11). Median model fit is shown in red (with 10 runs in alpha), model-predicted u_k in purple and x_k in green. **c**, Across trial learning model for state estimation computing internal signals across trials during adaptation (see Methods). On the right we show absolute value predictions for the three major nodes: the motor command to counter the perturbation, the state estimator, and the prediction error. Each subplot has 3 lines to demonstrate that, depending on internal estimates, each is an tunable 'hyperparameter' for the mouse (lighter equates to a more idealized variant). **d**, Heatmap of all recorded neurons during the adaptation task. Ordering is ranked by activity during the early perturbation trials (first 10). The red dashed line denotes force-field onset time. The color bar denotes the block; for baseline we used trials 25-75, then the first ten perturbation trials, then the next 45 early perturbation, then 45 late perturbation, first ten washout, followed by the next 45 washout trials. **e**, Normalized NMF PSTH plots over all neurons for a listed trial-bin. $***$ denotes a significant difference ($p < 0.001$, paired t-test) between a given plot and baseline. **f**, Functional clustering of neurons with PCA (n=20) followed by hierarchical clustering (n=5). The Dendogram is shown on the right. **g**, PSTH of the clusters across epochs. Neuron numbers discovered per cluster are shown, and plots are colored based on their best-match to our model computations (see panel **c**). Red line is force-field onset time, with 1 sec shown before and after. Bottom right shows the normalized counts per cluster, distributed across their GLM-group ID. Group colors as in Figure 4.

the largest cluster of neurons (cluster 2) appeared to track the state estimation-signal, namely an increase in firing across the adaptation epoch that tailored off in washout and returned to baseline levels (Figure 5f, g). They included nearly equally M1 and S1 neurons (S1 n=604, M1 n=607). To note, if we clustered baseline-only sessions with this method we did not find these computational motifs (Suppl. Figure S9a).

We then asked if these specific functional clusters, beyond mapping to model-based optimal control, could map to specific groups of kinematic or muscle features we uncovered in baseline sessions. We computed GLMs for the adaptation sessions (Suppl. Figure S9b) then assigned a group ID to each neuron to see if the population that clustered together fell into a specific grouping. Within the sensorimotor prediction error clusters (clusters 0, 3, and 4) the most represented group was for muscle and proprioception (groups 5 and 6), but not exclusively so (Figure 5g). Next, we checked if the S1 neurons in group 6 (proprioception) alone showed the signature of sensorimotor prediction errors. Strikingly, if we clustered all group 6 S1 neurons we find the signature, but not in M1 neurons (Suppl. Figure S9c). Thus, while sensorimotor prediction errors could be computed in multiple reference frameworks, the data suggests it is enriched for muscle-features (Figure 5g).

While they constitute a smaller fraction of neurons, the command-like neurons in cluster 1 were more represented in higher-

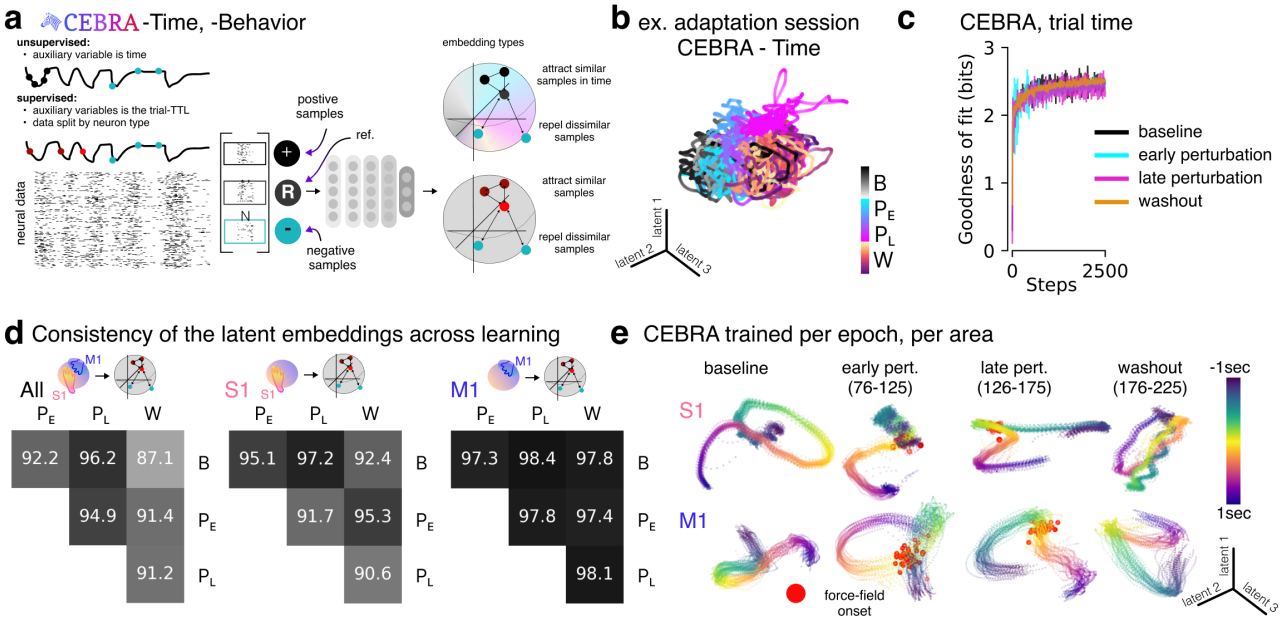

Figure 6. Sensorimotor cortex reflects the neural dynamics of learning to adapt. **a**, Diagram of how we used CEBRA in an self-supervised and supervised manner to quantify changes in the latent dynamics across learning. We use CEBRA-Time on each individual session (see b, c) then build single encoder models from across all mice in a neuron and trial dependent manner (see d, e). **b**, An example CEBRA-Time (offset-10-model-mse, 8D) colored by baseline perturbation, and washout. The first three latent variables are shown (no ranking). **c**, We trained a single encoder model per block: 50 baseline trials, 50 early perturbation, 50 late perturbation, and 50 washout, and measure the change in the latent space between blocks with a consistency metric (shown in matrix, see Methods). Goodness of fit for all models showed there was no difference in their ability to train. **d**, Quantification of CEBRA models trained per epoch and area. S1 n=908 neurons, M1 n=789 neurons; each showing the output embeddings and consistency matrices. **e**, Visualization of CEBRA embeddings from **d**.

level kinematic space (groups 1 and 2), but also included neurons from the muscle-feature space (group 5). State estimation neurons in cluster 2 were mostly comprised of neurons that had GLM-based non-significant tuning to the explicit motor features we considered. This could mean that state estimation is represented in a more abstract space across sensorimotor cortex. Future work should aim to understand what the functional role this fraction of neurons plays (nearly 50% of all recorded neurons are not significantly tuned to the features we explored).

Collectively, this suggests that the neural dynamics in S1 encode sensorimotor prediction errors most strongly, while both S1 and M1 encodes the dynamically changing state estimate of the limb. Given that we showed that 3D kinematic information can be integrated from muscle-level features (Figure 4j), this could mean that neurons within S1 layer 2/3 are using the muscle-level efference copy (i.e., the motor command, u_k) to build an internal model of the posture of the limb (i.e., by leveraging sensorimotor prediction errors, primarily in muscle-feature space, to build an optimal state estimator).

Neural latent dynamics during motor adaptation.

Sensory prediction errors are key in state estimation, which is implemented as a dynamical system (4, 27, 28, and see Methods). Therefore, we leveraged CEBRA to model nonlinear changes in the neural dynamics. Using self-supervised CEBRA models, we observed changes in the latent space across learning (Figure 6a, b, Suppl. Figure S10a). To quantify the changes in the latent spaces, we trained individual encoder models across each epoch (Figure 6c, d, Suppl. Figure S10b, Video 4). Importantly, we could equally fit each epoch with the same number of latent dimensions, model parameters, and training time (Figure 6c), which is an important control before quantifying changes across models (i.e., with the consistency metric, Figure 6d). Next, we measure the change in the latent dynamics between epochs. We found that the latent space changed compared to baseline in the early perturbation phase, and returned toward baseline during late perturbation trials (despite the trajectories remaining different from baseline). Then upon unexpected removal of the force-field in washout, the latent space remained separated from baseline (Figure 6d). This suggests that when their internal models are updated, even though baseline movements are restored, cortical dynamics remain changed.

To attribute which neural populations were driving this change, we built M1-only and S1-only CEBRA models (see Methods, Figure 6d). Single encoder models trained across mice with only M1 data showed a remarkable consistency of the latent neural

dynamics across blocks, while S1 neurons shifted during perturbation (Figure 6d, e). This finding supports our prior finding that S1 is computing prediction errors during learning (Figure 5f, g, and Suppl. Figure S9c). Note, we also had seen that S1 neurons that showed more command-like signals (although it was a much smaller fraction compared to the prediction errors), which presents as a slower shift in the motor command signal across learning (Figure 5f, Suppl. Figure S10c), but this also likely contributes to the change in consistency we report. Collectively, here we show that the sensorimotor cortex neural dynamics richly encode low-level muscle and high-level kinematic features that are the critical signals related to an internal model of the world.

Discussion

Understanding how neural dynamics enable adaptive, dynamic control of the body is a fundamental tenant of neuroscience and it has implications for improving applications in robotics, artificial intelligence, and neuroprosthetics. Here we aimed to synergize conflicting reports and theoretical predictions of high- and low-level encoding across sensorimotor cortex. We report how sensorimotor cortex supports skilled motor control and learning by a multi-dimensional representation of kinematics (position) and dynamics (muscular and proprioceptive) information. We focused on recording from excitatory pyramidal neurons in layer 2/3, which is postulated to be critical integrator of top-down (pre-motor) and bottom-up (thalamic, brainstem) inputs, namely motor efference and proprioceptive feedback, and due to S1 having a known essential role for the ability to adapt (in both mice and humans (4, 5)). Collectively, these circuits are core for action selection, motor-decision-making, and generating volitional movement. Therefore we built the first large-scale mouse brain model – inspired by known anatomy, physiology, and theoretical roles – and leveraged this to develop a systematic analysis of neural dynamics during the execution of motor skills and motor adaptation.

Moreover, we aimed to synergize theoretical predictions and past experimental observations. In the last 50 years several key frameworks for motor cortical control have been put forth: from joint and force control in the 1960's (69), population vectors for hand-direction tuning in the 1980's (15), to optimal feedback control for task-relevant control in the 2000's (27, 30), to modern-day population dynamical systems (3, 19). A challenge has been to synthesize this data, as early recordings were sparse and often recorded in different tasks. Yet, notably, in 2000 Todorov (2) argued that models that prioritized muscle-control could capture the "spurious" correlations to other high-order regressors. Here, we took an approach to systematically, and in a biologically-inspired model, create groups of features reported in the literature. In support of Todorov (2) we find predominately position and muscle-space encoding (not velocity or torque), and in support of Churchland et al. (3) the neural latent factors in this coordinate framework showed neural dynamics during control and learning.

Critically, we find that low-level features can be used to decode high-level positional kinematics, but not inversely, which likely explains why high-level features like directional tuning or "movement" are often reported across cortex (51). Future work should aim to delineate the boundaries of movement encoding vs. specific movement features as reported here. It is likely that muscle-features are more specific to cortical sensorimotor areas that directly receive proprioceptive input and provide output to spinal cord.

We report how specific functional groups of neurons can be mapped to computational motifs. Namely, prediction errors and motor command signals can be neurally delineated and specific. Collectively, these signals are critical for optimal state estimation and feedback control theory (27, 93). Coupled with the latent dynamics analysis that shows changes across learning, we argue that these are the essential ingredients for adaptive optimal state estimation, i.e., an internal model. This synergistically explains the underlying neural basis of prior work that found S1 to be essential for learning to adapt (4, 5).

In future work it will be important to deeply consider both upstream and downstream circuits from sensorimotor cortex, such as premotor areas, basal ganglia, brainstem, cerebellum, and spinal cord. Our MausSpaun model has a premotor area, basal ganglia, and brainstem implemented as spiking neural networks, which can serve as an *in silico* model to make concrete predictions about what these areas might encode during this task. Here, we consider the error computation to be in sensorimotor cortex (which maps to our finding of prediction errors in cortex), but it is likely that errors with other reference frameworks are executed in cerebellar circuits (94). Implementing real-time feedback controllable central pattern generators as a spinal cord would also be interesting to make predictions about perturbations that can be handled by low-level circuits vs. high-level perturbations. It is likely that each "level" in the sensorimotor hierarchy contributes to reducing prediction errors, and contributes to optimal state estimation (42, 54).

In summary, our work provides a normative computational framework for integrating high-level and low-level motor control into a theoretically-guided model. We leveraged this model to tackle a long-standing question about the nature of neural representations across sensorimotor cortex during control and learning. We were able to harmonize muscle-level control with neural dynamical systems approaches and show muscle-level features can be used as a substrate during learning and discover computational functional motifs – i.e., sensorimotor prediction errors – in S1 that can directly support motor learning.

References

1. Andrew J. Peters, Haixin Liu, and Takaki Komiyama. Learning in the rodent motor cortex. *Annual review of neuroscience*, 40:77–97, 2017.
2. Emanuel Todorov. Direct cortical control of muscle activation in voluntary arm movements: a model. *Nat Neurosci*, 3(4):391–398, April 2000. ISSN 1097-6256. doi: 10.1038/73964.
3. Mark Churchland et al. Neural population dynamics during reaching. *Nature*, 2012.
4. Mackenzie Weygandt Mathis, Alexander Mathis, and Naoshige Uchida. Somatosensory Cortex Plays an Essential Role in Forelimb Motor Adaptation in Mice. *Neuron*, 93(6):1493–1503.e6, March 2017. ISSN 0896-6273. doi: 10.1016/j.neuron.2017.02.049.
5. Neeraj Kumar, Timothy F Manning, and David J. Ostry. Somatosensory cortex participates in the consolidation of human motor memory. *PLoS Biology*, 17, 2019.
6. Shahryar Ebrahimi and David J. Ostry. The human somatosensory cortex contributes to the encoding of newly learned movements. *Proceedings of the National Academy of Sciences of the United States of America*, 121, 2024.
7. Constantine Pavlides, Eizo Miyashita, and Hiroshi Asanuma. Projection from the sensory to the motor cortex is important in learning motor skills in the monkey. *Journal of neurophysiology*, 70 2:733–41, 1993.
8. Chiang-Shan Ray Li, Camillo Padoa-Schioppa, and Emilio Bizzi. Neuronal Correlates of Motor Performance and Motor Learning in the Primary Motor Cortex of Monkeys Adapting to an External Force Field. *Neuron*, 30(2):593–607, May 2001. ISSN 0896-6273. doi: 10.1016/S0896-6273(01)00301-4.
9. Risa Kawai, Timothy Markman, Rajesh Poddar, Raymond Ko, Antoniu L. Fantana, Ashesh K. Dhawale, Adam R. Kampff, and Bence P. Ölveczky. Motor Cortex Is Required for Learning but Not for Executing a Motor Skill. *Neuron*, 86(3):800–812, May 2015. ISSN 0896-6273. doi: 10.1016/j.neuron.2015.03.024.
10. RJ Nudo, GW Milliken, W Jenkins, and M. M. Merzenich. Use-dependent alterations of movement representations in primary motor cortex of adult squirrel monkeys. In *Journal of Neuroscience*, 1996.
11. Andrew E. Papale and Bryan M. Hooks. Circuit changes in motor cortex during motor skill learning. *Neuroscience*, 368:283–297, 2018.
12. Stephen H. Scott. The computational and neural basis of voluntary motor control and planning. *Trends in Cognitive Sciences*, 16(11):541–549, November 2012. ISSN 1364-6613. doi: 10.1016/j.tics.2012.09.008.
13. N. A. Bernstein. Issledovania po biomekhanike udara s pomoshiu svetovoi zapisi (studies of biomechanics of the strike with the camera recording) [in russian]. *Issledovanija Centr. Instituta Truda*, 1:19–79, 1923.
14. N. A. Bernstein. *The Coordination And Regulation Of Movements*. Pergamon Press, 1967.
15. Apostolos P. Georgopoulos, Andrew B. Schwartz, and RE Kettner. Neuronal population coding of movement direction. *Science*, 233 4771: 1416–9, 1986.
16. John F. Kalaska, Dan A D Cohen, M. L. Hyde, Michel Prud 'hommel, Richard Bouchoux, Marc Bourdeau, Robert R. Cartier, Gilbert Duhau, and Jean-Philippe Jodoin. A comparison of movement direction-related versus load direction- related activity in primate motor cortex, using a two-dimensional reaching task. In *Journal of Neuroscience*, 1989.
17. James M. Goodman, Gregg Tabot, Alex S. Lee, Aneesha K. Suresh, Alexander T. Rajan, Nicholas G. Hatsopoulos, and Sliman J. Bensmaia. Postural representations of the hand in the primate sensorimotor cortex. *Neuron*, 104:1000–1009.e7, 2019.
18. Apostolos Georgopoulos et al. On the relations between the direction of two-dimensional arm movements and cell discharge in primate motor cortex. *J. Neurosci*, 2(11):1527–1537, 1982.
19. Mark M. Churchland and Krishna V Shenoy. Preparatory activity and the expansive null-space. *Nature reviews. Neuroscience*, 2024.
20. Mackenzie W. Mathis, Adriana Perez Rotondo, Edward F. Chang, Andreas S. Tolias, and Alexander Mathis. Decoding the brain: from neural1 representations to mathematical models. *aXiv*, 2024.
21. N. Hogan. An organizing principle for a class of voluntary movements. *J. Neurosci.*, 4(11):2745–2754, November 1984. ISSN 0270-6474.
22. R. Shadmehr and F. A. Mussa-Ivaldi. Adaptive representation of dynamics during learning of a motor task. *J. Neurosci.*, 14(5):3208–3224, May 1994. ISSN 0270-6474, 1529-2401.
23. J. R. Lackner and P. Dizio. Rapid adaptation to Coriolis force perturbations of arm trajectory. *Journal of Neurophysiology*, 72(1):299–313, July 1994.
24. John W Krakauer and Pietro Mazzoni. Human sensorimotor learning: adaptation, skill, and beyond. *Current Opinion in Neurobiology*, 21(4): 636–644, August 2011. ISSN 0959-4388. doi: 10.1016/j.conb.2011.06.012.
25. Jonathan S. Tsay, Hrach Asmerian, Laura T. Germine, Jeremy B Wilmer, Richard B. Ivry, and Ken Nakayama. Large-scale citizen science reveals predictors of sensorimotor adaptation. *Nature human behaviour*, 2024.
26. Stephen H. Scott. A functional taxonomy of bottom-up sensory feedback processing for motor actions. *Trends in Neurosciences*, 39:512–526, 2016.
27. Emanuel Todorov and Michael I. Jordan. Optimal feedback control as a theory of motor coordination. *Nat Neurosci*, 5(11):1226–1235, November 2002. ISSN 1097-6256. doi: 10.1038/nn963.
28. D. M. Wolpert, R. C. Miall, and M. Kawato. Internal models in the cerebellum. *Trends Cogn. Sci. (Regul. Ed.)*, 2(9):338–347, September 1998. ISSN 1364-6613.
29. Mitsuo Kawato. Internal models for motor control and trajectory planning. *Current Opinion in Neurobio*, 9(6):718–727, 1999.
30. Stephen H. Scott. Optimal feedback control and the neural basis of volitional motor control. *Nat Rev Neurosci*, 5(7):532–546, July 2004. ISSN 1471-003X. doi: 10.1038/nrn1427.

31. Reza Shadmehr and John W Krakauer. A computational neuroanatomy for motor control. *Exp Brain Res*, 185(3):359–381, March 2008. ISSN 1432-1106. doi: 10.1007/s00221-008-1280-5.
32. Daniel M. Wolpert, Jörn Diedrichsen, and J. Randall Flanagan. Principles of sensorimotor learning. *Nat Rev Neurosci*, 12(12):739–751, December 2011. ISSN 1471-003X. doi: 10.1038/nrn3112.
33. Pietro Mazzoni and John W. Krakauer. An Implicit Plan Overrides an Explicit Strategy during Visuomotor Adaptation. *J. Neurosci.*, 26(14): 3642–3645, April 2006. ISSN 0270-6474, 1529-2401. doi: 10.1523/JNEUROSCI.5317-05.2006.
34. Michael I. Jordan and David E. Rumelhart. Forward models: Supervised learning with a distal teacher. *Cogn. Sci.*, 16:307–354, 1992.
35. John W. Krakauer, Maria Felice Ghilardi, and Claude Ghez. Independent learning of internal models for kinematic and dynamic control of reaching. *Nature Neuroscience*, 2:1026–1031, 1999.
36. Alkis M. Hadjiosif, John W. Krakauer, and Adrian M. Haith. Did we get sensorimotor adaptation wrong? implicit adaptation as direct policy updating rather than forward-model-based learning. *The Journal of Neuroscience*, 41:2747 – 2761, 2021.
37. Saurabh Vyas, Daniel J. O'Shea, Stephen I. Ryu, and Krishna V. Shenoy. Causal role of motor preparation during error-driven learning. *Neuron*, 2020.
38. Raeed H. Chowdhury, Joshua I. Glaser, and Lee E. Miller. Area 2 of primary somatosensory cortex encodes kinematics of the whole arm. *eLife*, 9, 2019.
39. Ignacio Alonso, Irina Scheer, Melanie Palacio-Manzano, Noémie Frézel-Jacob, Antoine Philippides, and Mario Prsa. Peripersonal encoding of forelimb proprioception in the mouse somatosensory cortex. *Nature Communications*, 14, 2023.
40. Alessandro Marin Vargas, Axel Bisi, Alberto S Chiappa, Chris Versteeg, Lee E Miller, and Alexander Mathis. Task-driven neural network models predict neural dynamics of proprioception. *Cell*, 187(7):1745–1761, 2024.
41. Hermano Igo Krebs, Neville Hogan, Mindy L. Aisen, and Bruce T. Volpe. Robot-Aided Neurorehabilitation. *IEEE Trans Rehabil Eng*, 6(1):75–87, March 1998. ISSN 1063-6528.
42. Mohsen Omrani, Chantelle D. Murnaghan, J. Andrew Pruszynski, and Stephen H. Scott. Distributed task-specific processing of somatosensory feedback for voluntary motor control. *eLife*, 5:e13141, April 2016. ISSN 2050-084X. doi: 10.7554/eLife.13141.
43. Georg B. Keller, Tobias Bonhoeffer, and Mark Hübener. Sensorimotor Mismatch Signals in Primary Visual Cortex of the Behaving Mouse. *Neuron*, 74(5):809–815, June 2012. ISSN 0896-6273. doi: 10.1016/j.neuron.2012.03.040.
44. Pawel Zmarz and Georg B. Keller. Mismatch receptive fields in mouse visual cortex. *Neuron*, 92:766–772, 2016.
45. Sébastien B Hausmann, Alessandro Marin Vargas, Alexander Mathis, and Mackenzie W. Mathis. Measuring and modeling the motor system with machine learning. *Current Opinion in Neurobiology*, 70:11–23, 2021.
46. Scott L. Delp, Frank C. Anderson, Allison S. Arnold, Peter Loan, Ayman Habib, Chand T. John, Eran Guendelman, and Darryl G. Thelen. Opensim: Open-source software to create and analyze dynamic simulations of movement. *IEEE Transactions on Biomedical Engineering*, 54:1940–1950, 2007.
47. Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.
48. Shravan Tata Ramalingasetty, Simon M. Danner, Jonathan Arreguit, Sergey N. Markin, Dimitri Rodarie, Claudia Kathe, Grégoire Courtine, Ilya A. Rybak, and Auke Jan Ijspeert. A whole-body musculoskeletal model of the mouse. *IEEE access : practical innovations, open solutions*, 9:163861 – 163881, 2021.
49. Alexander Mathis, Pranav Mamidanna, Kevin M Cury, Taiga Abe, Venkatesh N Murthy, Mackenzie Weygandt Mathis, and Matthias Bethge. Deeplabcut: markerless pose estimation of user-defined body parts with deep learning. *Nature neuroscience*, 21(9):1281–1289, 2018.
50. Cristopher M. Niell and Michael P. Stryker. Modulation of visual responses by behavioral state in mouse visual cortex. *Neuron*, 65:472–479, 2010.
51. Simon Musall, Matthew T. Kaufman, Ashley L. Juavinett, Steven Gluf, and Anne K. Churchland. Single-trial neural dynamics are dominated by richly varied movements. *Nature neuroscience*, 22:1677 – 1686, 2019.
52. Carsen Stringer, Marius Pachitariu, Nicholas A. Steinmetz, Charu Bai Reddy, Matteo Carandini, and Kenneth D. Harris. Spontaneous behaviors drive multidimensional, brainwide activity. *Science*, 364, 2019.
53. Mackenzie W. Mathis. A new spin on fidgets. *Nature Neuroscience*, 22:1614 – 1616, 2019.
54. J. Andrew Pruszynski, Isaac Kurtzer, Joseph Y. Nashed, Mohsen Omrani, Brenda Brouwer, and Stephen H. Scott. Primary motor cortex underlies multi-joint integration for fast feedback control. *Nature*, 478(7369):387–390, October 2011. ISSN 0028-0836. doi: 10.1038/nature10436.
55. Andrew Miri, Claire L. Warriner, Jeffrey S. Seely, Gamaleldin F. Elsayed, John P. Cunningham, Mark M. Churchland, and Thomas M. Jessell. Behaviorally selective engagement of short-latency effector pathways by motor cortex. *Neuron*, 95:683–696.e11, 2017.
56. Albert King. A review of biomechanical models. *Journal of biomechanical engineering*, 106 2:97–104, 1984.
57. Emilie Mathieu, Sylvain Crémoux, David Duvivier, David Amarantini, and Philippe Pudlo. Biomechanical modeling for the estimation of muscle forces: toward a common language in biomechanics, medical engineering, and neurosciences. *Journal of NeuroEngineering and Rehabilitation*, 20, 2023.
58. Kai J. Sandbrink, Pranav Mamidanna, Claudio Michaelis, Matthias Bethge, Mackenzie W. Mathis, and Alexander Mathis. Contrasting action and posture coding with hierarchical deep neural network models of proprioception. *eLife*, 12, 2023.
59. Olivier Codol, Jonathan A. Michaels, Mehrdad Kashefi, J. Andrew Pruszynski, and Paul L. Gribble. Motornet: a python toolbox for controlling differentiable biomechanical effectors with artificial neural networks. *eLife*, 2024.
60. Muhammad Noman Almani, John Lazzari, Andrea Chacon, and Shreya Saxena. µsim: A goal-driven framework for elucidating the neural control of movement through musculoskeletal modeling. *bioRxiv*, 2024.
61. Timothy P Lillicrap and Stephen H Scott. Preference distributions of primary motor cortex neurons reflect control solutions optimized for limb biomechanics. *Neuron*, 77(1):168–179, 2013.
62. Alberto Silvio Chiappa, Pablo Tano, Nisheet Patel, Abigaïl Ingster, Alexandre Pouget, and Alexander Mathis. Acquiring musculoskeletal skills with curriculum-based reinforcement learning. *bioRxiv*, 2024. doi: 10.1101/2024.01.24.577123.
63. Eunice Chace Greene. Anatomy of the rat. *Transactions of the American Philosophical Society*, 27:iii–370, 1935.
64. Takamasa Iwaki, Hiroshi Yamashita, Toshiyuki Hayakawa, et al. *Color atlas of sectional anatomy of the mouse*. Braintree Scientific Inc, 2005.
65. Tanmay Nath, Alexander Mathis, An Chi Chen, Amir Patel, Matthias Bethge, and Mackenzie W. Mathis. Using deeplabcut for 3d markerless pose estimation across species and behaviors. *Nature Protocols*, 14:2152 – 2176, 2019.

66. Chris Eliasmith et al. A large-scale model of the functioning brain. *Science*, 338(6111), 2012.
67. Travis DeWolf, Kinjal Patel, Pawel Jaworski, Roxana Leontie, Joe Hays, and Chris Eliasmith. Neuromorphic control of a simulated 7-dof arm using loihi. *Neuromorphic Computing and Engineering*, 3(1):014007, 2023.
68. Jan M Skowronski. *Control Theory of Robotic Systems*, volume 11. World Scientific, 1989.
69. Edward V Evarts. Relation of pyramidal tract activity to force exerted during voluntary movement. *J Neurophys*, 31(1):14–27, 1968.
70. Chris Eliasmith and Charles H Anderson. *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. MIT Press, 2004.
71. Wuzhou Yang, Harsh Kanodia, and Silvia Arber. Structural and functional map for forelimb movement phases between cortex and medulla. *Cell*, 186:162 – 177.e18, 2023.
72. Ludwig F Ruder, Riccardo Schina, Harsh Kanodia, Sara Valencia-Garcia, Chiara Pivetta, and Silvia Arber. A functional map for diverse forelimb actions within brainstem circuitry. *Nature*, 590:445 – 450, 2021.
73. Bryce Chung, Muneeb Zia, Kyle A. Thomas, Jonathan A. Michaels, Amanda L. Jacob, Andrea R. Pack, Matt Williams, Kailash Nagapudi, Lay Heng Teng, Eduardo Arrambide, Logan Ouellette, Nicole F. Oey, Rhuna Gibbs, Philip Anschutz, Jiaao Lu, Yu Ping Wu, Mehrdad Kashefi, Tomomichi Oya, Rhonda Kersten, Alice C. Mosberger, Sean O'Connell, Runming Wang, Hugo Marques, Ana Rita Mendes, Constanze Lenschow, Gayathri Kondakath, Jeong Jun Kim, William P Olson, Kiara N. Quinn, Pierce Perkins, Graziana Gatto, Ayesha Thanawalla, Susan K. Coltman, Taegyo Kim, Trevor Smith, Benjamin I. Binder-Markey, Martin Zaback, Christopher K. Thompson, Simon F. Giszter, Abigail L. Person, Martyn D. Goulding, Eiman Azim, Nitish V. Thakor, Daniel H. O'Connor, Barry Andrew Trimmer, Susana Q. Lima, Megan R. Carey, Chethan Pandarinath, Rui M. Costa, J. Andrew Pruszynski, Muhannad S. Bakir, and Samuel J. Sober. Myomatrix arrays for high-definition muscle recording. *eLife*, 12, 2023.
74. Alain Frigon, Yann Thibaudier, and Marie-France Hurteau. Modulation of forelimb and hindlimb muscle activity during quadrupedal tied-belt and split-belt locomotion in intact cats. *Neuroscience*, 290:266–278, 2015.
75. Matthew I. Becker, Dylan J. Calame, Julia Wrobel, and Abigail L. Person. Online control of reach accuracy in mice. *Journal of neurophysiology*, 2020.
76. Kelly A. Tennant, DeAnna L. Adkins, Nicole A. Donlan, Aaron L. Asay, Nagheme J. Thomas, Jeffrey A. Kleim, and Theresa A. Jones. The organization of the forelimb representation of the c57bl/6 mouse motor cortex as defined by intracortical microstimulation and cytoarchitecture. *Cerebral cortex*, 21 4:865–76, 2011.
77. Takaki Komiyama, Takashi R Sato, Daniel H. O'Connor, Ying-Xin Zhang, Daniel Huber, Bryan M. Hooks, Mariano Gabitto, and Karel Svoboda. Learning-related fine-scale specificity imaged in motor cortex circuits of behaving mice. *Nature*, 464:1182–1186, 2010.
78. Karin Morandell and Daniel Huber. The role of forelimb motor cortex areas in goal directed action in mice. *Scientific Reports*, 7, 2017.
79. D. Huber, D. A. Gutnisky, S. Peron, D. H. O'Connor, J. S. Wiegert, L. Tian, T. G. Oertner, L. L. Looger, and K. Svoboda. Multiple dynamic representations in the motor cortex during sensorimotor learning. *Nature*, 484(7395):473–478, April 2012. ISSN 0028-0836, 1476-4687. doi: 10.1038/nature11039.
80. Mark J. Wagner, Joan Savall, Tony Hyun Kim, Mark J. Schnitzer, and Liqun Luo. Skilled reaching tasks for head-fixed mice using a robotic manipulandum. *Nature Protocols*, 15:1237 – 1254, 2020.
81. Spencer Bowles, Jordan Hickman, Xiaoyu Peng, William R. Williamson, Rongcheng Huang, Kayden A. Washington, Dane C Donegan, and Cristin G. Welle. Vagus nerve stimulation drives selective circuit modulation through cholinergic reinforcement. *Neuron*, 110:2867–2885.e7, 2022.
82. John Martin Barrett, Martinna G. Raineri Tapies, and Gordon M. G. Shepherd. Manual dexterity of mice during food-handling involves the thumb and a set of fast basic movements. *PLoS ONE*, 15, 2019.
83. Riichiro Hira, Fuki Ohkubo, Katsuya Ozawa, Yoshikazu Isomura, Kazuo Kitamura, Masanobu Kano, Haruo Kasai, and Masanori Matsuzaki. Spatiotemporal dynamics of functional clusters of neurons in the mouse motor cortex during a voluntary movement. *The Journal of Neuroscience*, 33:1377 – 1390, 2013.
84. Andrew J. Peters, Simon X. Chen, and Takaki Komiyama. Emergence of reproducible spatiotemporal activity during motor learning. *Nature*, 510: 263–267, 2014.
85. Eun Jung Hwang, Jeffrey E. Dahlen, Yvonne Yuling Hu, Karina Aguilar, Bin Yu, Madan Mukundan, Akinori Mitani, and Takaki Komiyama. Disengagement of motor cortex from movement control during long-term learning. *Science Advances*, 5, 2019.
86. Nicholas James Sofroniew, Daniel Flickinger, Jonathan King, and Karel Svoboda. A large field of view two-photon mesoscope with subcellular resolution for in vivo imaging. *eLife*, 5, 2016.
87. Quanxin Wang, Song-Lin Ding, Yang Li, Joshua J. Royall, Hongkui Zeng, and Lydia Ng. The allen mouse brain common coordinate framework: A 3d reference atlas. *Cell*, 181:936–953.e20, 2020.
88. Eric R Kandel, James H Schwartz, and Thomas M Jessell. *Principles of neural science*. McGraw-Hill, Health Professions Division, New York, 2000. ISBN 0-8385-7701-6 978-0-8385-7701-1.
89. Steffen Schneider, Jin Hwa Lee, and Mackenzie W. Mathis. Learnable latent embeddings for joint behavioural and neural analysis. *Nature*, 617: 360 – 368, 2023.
90. Mackenzie Weygandt Mathis. Prediction errors drive learning in a mouse model of motor adaptation. *Harvard University, PhD Dissertation*, 2017. doi: https://dash.harvard.edu/handle/1/41142031.
91. K A Thoroughman and R Shadmehr. Learning of action through adaptive combination of motor primitives. *Nature*, 407(6805):742–747, October 2000. ISSN 0028-0836. doi: 10.1038/35037588.
92. Kathleen E. Cullen. Vestibular processing during natural self-motion: implications for perception and action. *Nature Reviews Neuroscience*, 20: 346 – 363, 2019.
93. Tomohiko Takei, Stephen G. Lomber, Douglas J. Cook, and Stephen H. Scott. Transient deactivation of dorsal premotor cortex or parietal area 5 impairs feedback control of the limb in macaques. *Current Biology*, 31:1476–1487.e5, 2021.
94. Maurice A. Smith and Reza Shadmehr. Intact Ability to Learn Internal Models of Arm Dynamics in Huntington's Disease But Not Cerebellar Degeneration. *Journal of Neurophysiology*, 93(5):2809–2821, May 2005. doi: 10.1152/jn.00943.2004.
95. Tony Hyun Kim, Yanping Zhang, Jérôme A. Lecoq, Juergen C. Jung, Jane Li, Hongkui Zeng, Cristopher M. Niell, and Mark J. Schnitzer. Long-term optical access to an estimated one million neurons in the live mouse cortex. *Cell reports*, 17 12:3385–3394, 2016.
96. Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein,

Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *ArXiv*, abs/1912.01703, 2019.
97. Markus Frey, Wesley Monteith-Finas, Travis DeWolf, Steffen Schneider, and Mackenzie W. Mathis. MouseArmTransformer: a transformer-based 3D lifting module for an adult mouse arm. *Zenodo*, July 2024. doi: 10.5281/zenodo.12673173.
98. Dimitri Yatsenko, Jacob Reimer, Alexander S. Ecker, Edgar Y. Walker, Fabian H Sinz, Philipp Berens, Andreas Hoenselaar, R. James Cotton, Athanassios S. Siapas, and Andreas Savas Tolias. Datajoint: managing big scientific data using matlab or python. *bioRxiv*, 2015.
99. B. Srinivasa Reddy and Biswanath N. Chatterji. An fft-based technique for translation, rotation, and scale-invariant image registration. *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, 5 8:1266–71, 1996.
100. Andrea Giovannucci, Johannes Friedrich, Pat Gunn, Jérémie Kalfon, Brandon L Brown, Sue Ann Koay, Jiannis Taxidis, Farzaneh Najafi, Jeffrey L. Gauthier, Pengcheng Zhou, Baljit S. Khakh, David W. Tank, Dmitri B. Chklovskii, and Eftychios A. Pnevmatikakis. Caiman an open source tool for scalable calcium imaging data analysis. *eLife*, 8, 2019.
101. Eftychios A. Pnevmatikakis, Daniel Soudry, Yuanjun Gao, Timothy A. Machado, Josh Merel, David Pfau, Thomas R. Reardon, Yu Mu, Clay Lacefield, Weijian Yang, Misha B. Ahrens, Randy M. Bruno, Thomas M. Jessell, Darcy S. Peterka, Rafael Yuste, and Liam Paninski. Simultaneous denoising, deconvolution, and demixing of calcium imaging data. *Neuron*, 89:285–299, 2016.
102. James P Charles, Ornella Cappellari, Andrew J Spence, Dominic J Wells, and John R Hutchinson. Muscle moment arms and sensitivity analysis of a mouse hindlimb musculoskeletal model. *Journal of anatomy*, 229(4):514–535, 2016.
103. Pierre Kibleur, Shravan R Tata, Nathan Greiner, Sara Conti, Beatrice Barra, Katie Zhuang, Melanie Kaeser, Auke Ijspeert, and Marco Capogrosso. Spatiotemporal maps of proprioceptive inputs to the cervical spinal cord during three-dimensional reaching and grasping. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 28(7):1668–1677, 2020.
104. H.M. Hudson and R.S. Larkin. Accelerated image reconstruction using ordered subsets of projection data. *IEEE Transactions on Medical Imaging*, 13(4):601–609, 1994. doi: 10.1109/42.363108.
105. Andriy Fedorov, Reinhard Beichel, Jayashree Kalpathy-Cramer, Julien Finet, Jean-Christophe Fillion-Robin, Sonia Pujol, Christian Bauer, Dominique Jennings, Fiona Fennessy, Milan Sonka, et al. 3d slicer as an image computing platform for the quantitative imaging network. *Magnetic resonance imaging*, 30(9):1323–1341, 2012.
106. Arthur Prochazka and Monica Gorassini. Ensemble firing of muscle afferents recorded during normal locomotion in cats. *The Journal of physiology*, 507(1):293–304, 1998.
107. Arthur Prochazka and Monica Gorassini. Models of ensemble firing of muscle spindle afferents recorded during normal locomotion in cats. *The Journal of physiology*, 507(1):277–291, 1998.
108. Eduardo Martin Moraud, Marco Capogro:wsso, Emanuele Formento, Nikolaus Wenger, Jack DiGiovanna, Gregoire Courtine, and Silvestro Micera. Mechanisms underlying the neuromodulation of spinal circuits for correcting gait and balance deficits after spinal cord injury. *Neuron*, 89(4):814–828, 2016.
109. Arthur Prochazka. Quantifying proprioception. *Progress in brain research*, 123:133–142, 1999.
110. Fanwang Meng, Michael Richer, Alireza Tehrani, Jonathan La, Taewon David Kim, Paul W. Ayers, and Farnaz Heidar-Zadeh. Procrustes: A python library to find transformations that maximize the similarity between matrices. *Computer Physics Communications*, 276(108334):1–37, 2022. ISSN 0010-4655. doi: https://doi.org/10.1016/j.cpc.2022.108334.
111. Tsuneo Yoshikawa. Manipulability of robotic mechanisms. *The international journal of Robotics Research*, 4(2):3–9, 1985.
112. Microsoft. Neural Network Intelligence. *GitHub*, 1 2021.
113. Feng-Xuan Choo. Spaun 2.0: Extending the world's largest functional brain model. *University of Waterloo*, 2018.
114. Travis DeWolf, Terrence C. Stewart, Jean-Jacques Slotine, and Chris Eliasmith. A spiking neural model of adaptive arm control. *Proceedings of the Royal Society of London B: Biological Sciences*, 283(1843), 2016. ISSN 0962-8452. doi: 10.1098/rspb.2016.2134.
115. T DeWolf and C Eliasmith. Trajectory generation using a spiking neuron implementation of dynamic movement primitives. In *27th Annual Meeting for the Society for the Neural Control of Movement*, 2017.
116. Trevor Bekolay, James Bergstra, Eric Hunsberger, Travis DeWolf, Terrence C. Stewart, Daniel Rasmussen, Feng-Xuan Choo, Aaron R. Voelker, and Chris Eliasmith. Nengo: a python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics*, 7, 2014.
117. Aaron R Voelker and Chris Eliasmith. Improving spiking dynamical networks: Accurate delays, higher-order synapses, and time cells. *Neural computation*, 30(3):569–609, 2018.
118. Stefan Schaal et al. Computational approaches to motor learning by imitation. *Philosophical Transactions of the Royal Society of London. Series B: Bio Sciences*, 358(1431):537–547, 2003.
119. Peter Pastor, Mrinal Kalakrishnan, Franziska Meier, Freek Stulp, Jonas Buchli, Evangelos Theodorou, and Stefan Schaal. From dynamic movement primitives to associative skill memories. *Robotics and Autonomous Systems*, 61(4):351–361, 2013.
120. Kevin Gurney et al. A computational model of action selection in the basal ganglia. i. a new functional anatomy. *Bio Cyber*, 84(6):401–410, 2001.
121. *Dynamic Behaviour of a Spiking Model of Action Selection in the Basal Ganglia*, 2010.
122. John R Anderson, Bonnie E John, Marcel Adam Just, Patricia A Carpenter, David E Kieras, and DE Meyer. Production system models of complex cognition. In *Proceedings of the seventeenth annual conference of the cognitive science society*, pages 9–12, 1995.
123. Jan Gosmann and Chris Eliasmith. Optimizing semantic pointer representations for symbol-like processing in spiking neural networks. *PloS One*, 11(2):e0149928, 2016.
124. Ivana Kajić, Jan Gosmann, Terrence C Stewart, Thomas Wennekers, and Chris Eliasmith. A spiking neuron model of word associations for the remote associates test. *Frontiers in psychology*, 8:225355, 2017.
125. Jan Gosmann. An integrated model of context, short-term, and long-term memory. *University of Waterloo*, 2018.
126. Jun Izawa and Reza Shadmehr. Learning from Sensory and Reward Prediction Errors during Motor Adaptation. *PLoS Comput Biol*, 7(3): e1002012, March 2011. doi: 10.1371/journal.pcbi.1002012.
127. R. S Sutton and A. G Barto. *Reinforcement learning: An introduction*, volume 1. Cambridge Univ Press, 1998.
128. Max Berniker and Konrad Kording. Estimating the sources of motor errors for adaptation and generalization. *Nat Neurosci*, 11(12):1454–1461, December 2008. ISSN 1097-6256. doi: 10.1038/nn.2229.
129. R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *J. Basic Eng.*, 82(1):35–45, March 1960. ISSN 0098-2202. doi: 10.1115/1.3662552.

130. Andrew J. Peters, Simon X. Chen, and Takaki Komiyama. Emergence of reproducible spatiotemporal activity during motor learning. *Nature*, 510 (7504):263–267, June 2014. ISSN 0028-0836. doi: 10.1038/nature13235.
131. Charles R. Harris, K. Jarrod Millman, Stéfan van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fern'andez del R'io, Marcy Wiebe, Pearu Peterson, Pierre G'erard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with numpy. *Nature*, 585:357 – 362, 2020.
132. Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods*, 17(3):261–272, March 2020. doi: 10.1038/s41592-019-0686-2.
133. TensorFlow Developers. Tensorflow. *Zenodo*, 2017. doi: 10.5281/zenodo.12726004.
134. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Contributions. T.D. developed the MuJoCo musculoskeletal model and optimization pipeline, built the MausSpaun controller and neural models, analyzed the biomechanics data, and contributed to editing the manuscript. St.S. Built additional neural data analysis tools and analyzed the data (GLMs and CEBRA), and contributed to editing the manuscript. P.S. Prepared and collected all the CT images, parsed the biomechanical skeleton, and analyzed muscle attachment data. A.H. Built the initial DataJoint data analysis pipeline and performed preliminary neural data analysis. M.W.M. Conceived the project, performed surgeries, mouse training, conducted 2P imaging, developed the DeepLabCut models, assisted in all data analysis, wrote the manuscript, co-prepared visualizations, acquired funding, and supervised the project. All authors contributed to visualizations, writing the methods, and approved the final manuscript.

Acknowledgments. This was a project that spanned over 7 years in the Mathis Lab (since 2017), thus we are deeply grateful to all former and current members from Harvard and EPFL for their inputs. In particular, we greatly thank A. Mathis for ample discussions on motor control, bio-inspired modeling, and biomechanics. We thank M. Black, S. Zuffi & S. Pujades for resources and consultation on mouse CT scans. We also thank T. Nath for helping developing the original DataJoint pipeline, M. Popova and N. Poulsen with numerical optimizations, M. Frey and W. Monteith for 3D GT labeling and assistance with 3D pose estimation, K. Sandbrink for early GLM analysis, S. Ye for reviewing and improving the code base, S. Hausmann for rig CAD files, E. Madsen at the Rowland Institute Machine Shop for customized joystick handle, and X. Choo for his assistance with the basal ganglia modeling. We thank A. Mathis, A. Perez Rotondo, S. Bowles, S. Scott, and A. Tolias for feedback on the manuscript.

Funding: We greatly thank the Rowland Institute at Harvard, SNSF Grant 310030_20105, The Harvard Brain Science Initiative, an NVIDIA Hardware Grant, and EPFL for funding. MWM is the Bertarelli Foundation Chair of Integrative Neuroscience.

Conflict of Interest. The authors declare no conflicts of interest.

Materials & Correspondence. For all requests please contact the corresponding author.

Data Availability. Two-photon and related behavioral data will be released with publication of the paper.

Code Availability. The MuJoCo-based biomechanical model, MausSpaun model code, and analysis tools will be available upon publication at: `https://github.com/AdaptiveMotorControlLab/MausSpaun`. Code to reproduce the figures: `https://github.com/AdaptiveMotorControlLab/DeWolf_Schneider_etal_2024`. All other requests should be made to the corresponding author.

Methods

Animals & Surgery

Data was collected from 6 adult (P90-P365) male and female from layer 2/3 neurons that were in mice that were either slc17a7 crossed to ai148 or Thy1-GCaMP6s-4.3 mice (Jackson Laboratory). Surgeries were performed under aseptic conditions with animals under isoflurane (1-2% at 1.0 L/min oxygen) anesthesia, and pre-operatively mice were given an anti-inflammatory (dexamethasone, 5 mg/kg, S.Q.). Buprenorphine (0.1 mg/kg, I.P. every 6-8 hours) was administered postoperatively for 48 hours. Mice were surgically implanted with a custom head plate (4) and a "crystal skull" glass cranial window (95). All surgical and experimental procedures were in accordance with the National Institutes of Health Guide for the Care and Use of Laboratory Animals and approved by the Harvard Institutional Animal Care Use Committee and/or Caton Geneva, under license number GE68.

All mice were housed on an inverted 12h dark/12h light cycle (dark from 09:00 AM - 09:00 PM) and each performed the behavioral task at approximately the same time of day. After surgery mice were individually housed and allowed to recover for 7 days. All mice were maintained above 90% of their starting weight once on water restriction. No mouse was given less than 1 mL of water per day. Only right-handed mice were used in this study and where never used in other studies or published elsewhere.

Joystick task for mice

Mice were water restricted and taught to manipulate a 2D of freedom joystick as described in Mathis et al (4). To note, the mice presented in this work are not the same mice previously used. In brief, we utilized a joystick system (https://github.com/AdaptiveMotorControlLab/JoystickControlSystem) and rewarded mice with a approx. 3 uL drop of water for pulling the joystick towards themselves 6-9 mm into a virtual target box (of 2 mm in width). Note, they grasped the joystick with their hand in a vertical fashion, not as a pulled lever (see Figure 1a). The training progressed in three stages, where at first if the mice moved the joystick towards them any distance they were rewarded (this persisted for two to four days until they could do this consistently). Then, in stage two, and in a fully automated fashion, if they pulled 6 mm into a virtual target box and waited 100 ms they were rewarded. Once the mice could do this for greater than 100 trials at 80% accuracy for at least two days in a row the task shifted to phase three where they needed to pull 10 mm into the target box (width 2 mm). This was considered as the full baseline task. The joystick system records the position of the joystick at 1kHz within LabView and is synchronized with the video and mesoscope imaging for post-hoc analysis. We collected twenty-two baseline sessions and four adaptation sessions, where the mice where perturbed with a systemically applied force-field (see below) during two-photon imaging (see below). In six baseline sessions we included randomly applied force-field on 10% of the trials to increase the coverage of the x-y joystick space, i.e., in only 150 trials of the 4,667 total baseline trials in the data set).

The adaptation session paradigm was adopted from Mathis et al. (4). In brief, after the mice are experts at the baseline task above, we challenge them in a single session with a delayed-onset force-field perturbation. The force-field is an approximately 0.2N force orthogonal (x) to the main pull axis (y, see Figures 3b, and 5a). This was only applied in trials 76-175. Other trials (1-75 and 176-250) had no force-field.

Pose estimation

To measure the pose and kinematics of the mice we used two Imaging Source cameras 640x480 pixels recorded at 75-150Hz placed in stereo-view around the mouse's right arm and head. We then trained pose estimation models for either the front and side view (or a single combined view model) using DeepLabCut version 2.0.7 or 2.0.9 (49, 65) (https://github.com/DeepLabCut/DeepLabCut). In brief, we labeled 32 keypoints across approx. 500 2D frames (then 95% were used for training). We used a ResNet50-based neural network with default augmentation and training parameters for 1.2M training iterations. We then used a p-cutoff of 0.9 to condition the x,y coordinates for 3D triangulation. This network was then used to analyze videos from similar experimental settings. To filter any noisy 3D triangulated data, we additionally used a manual labeling step to obtain ground truth 3D information for a subset of approx. 500 frames.

Lifting model for transforming 2D to 3D space. We trained a simple transformer model that took as input the 2D DeepLabCut extracted coordinates from *one* camera and extracts the 3D coordinates through a linear output layer (96, 97). The input to the transformer consists of sequences (of length T=2 frames) of 2D coordinates representing the keypoint positions. Each keypoint is represented by an x and y position. We then encode the flattened joint coordinates using a transformer layer and then project the encoder output to a three-dimensional space using a fully connected layer. We trained this with a triangulation loss, continuity loss, and ground truth loss. For more details see Frey et al. (97) (https://github.com/AdaptiveMotorControlLab/mousearmtransformer). The transformer model is trained to output 3D coordinates in the MuJoCo reference frame, using the linear transformation parameters identified by aligning keypoints on the experimental platform (see below). Data is then normalized such that 0,0,0 is egocentric space in relation to the base of the head of the mouse. We trained the model across 10 sessions where two cameras are available, and used the resulting weights to generate 3D predictions for sessions with only one camera.

Two-Photon imaging and analysis

To record neural activity across several brain regions a commercially available two-photon mesoscope was used (2P-RAM system, Thorlabs). This microscope is designed to scan multiple fields at arbitrary locations within a 5 × 5 mm field of view and depths up to 1 mm (86). In brief, a resonant scanner provides fast line scanning (24 kHz line rate) over a range of 0.6 mm in the fast scanning direction, while the position of the scan field and movement along the slow scan axis is controlled by 3 galvo scanners. Fast scanning of different depths is achieved by a remote focusing unit, which shifts the focal point in z-direction by moving a lightweight mirror on a voice coil. High instantaneous photon densities to drive two-photon absorption are provided by a pulsed Ti:Sapphire laser with a tunable wavelength between 700 and 1100 nm (Tiberius laser, Thorlabs). All recordings were acquired using wavelengths between 960 and 970 nm and with laser powers below 60 mW after the objective.

The locations of the scan fields in the sensorimotor cortex were selected in the microscope control software (ScanImage 2018-2021, Vidrio, Alexandria, VA, USA) based on relative coordinates from Bregma (centered over +0 mm Bregma, -2 mm lateral, as defined in Mathis et al. (4), Tennant et al. (76), and the Allen Brain Atlas). Bregma was located in each recording by matching the vessel pattern visible in the full 5 mm field of view to the bregma location relative to the vessel pattern during the cranial window surgery. Depending on the size of the scanned fields, scanning rates varied between 5.7 to 16.2 Hz.

All post-recording analysis code was managed within a relational database using DataJoint (98). To extract neural signals from the acquired two-photon scans, a publicly available template for a two-photon processing pipeline from the Tolias Lab (https://github.com/cajal/pipeline) was adapted. The processing steps from the recorded image stacks to neural activity traces include raster correction, motion correction, segmentation, mask classification, and deconvolution. To correct for possible shifts of pixels scanned in the forward and backward direction of the resonant scanner, raster correction was performed. The shift was computed by shifting even and odd lines of the scan with respect to each other and calculating the position of maximal correlation between neighboring lines. All detected shifts were below 1 pixel due to the manual raster correction performed in the microscope control software (ScanImage 2018-2021, Vidrio, Alexandria, VA, USA).

All recorded frames were motion corrected for each scan field separately using a Discrete Fourier Transform (DFT) algorithm to register each frame to a template by x-y shifting (99). The template for each field was created by Anscombe transforming, averaging, and smoothing (Gaussian filter, σ = 0.7 pixel) a section of 2000 frames from the middle of the scan.

The x- and y-shifts were determined by finding the maximum of the convolution between each frame and the template. Shifts larger than 20 μm were considered outliers and were linearly interpolated. Then, the x- and y-shifts were median filtered (kernel width: 3) to remove spurious jumps of x- or y-shifts on single frames. Datasets with excessive z movement were excluded from further analysis. To detect the localized neuronal signal sources from the whole recording, the image stack was segmented using the CaImAn package (100) (https://github.com/flatironinstitute/CaImAn). The algorithm computes a constrained non-negative matrix factorization (CNMF) to extract the time course of localized sources (101). The recorded data $\mathbf{Y} \in \mathbb{R}^{d \times T}$ (d: number of pixels, T: time) was decomposed into three parts:

$$\mathbf{Y} = \mathbf{AC} + \mathbf{B} + \mathbf{E}, \tag{1}$$

with the matrix $\mathbf{A} \in \mathbb{R}^{d \times n}$ containing n extracted localized masks and the matrix $\mathbf{C} \in \mathbb{R}^{n \times T}$ representing the corresponding temporal components. $\mathbf{B} \in \mathbb{R}^{d \times T}$ is the extracted background activity and $\mathbf{E} \in \mathbb{R}^{d \times T}$ is the residual activity that cannot be decomposed (100).

After segmentation, the mask shapes were evaluated with a convolutional neural network. Only shapes that were classified as a soma (p > 0.75) were further considered for analysis. Next, all masks underwent a manual review and rare edge artifacts that were classified as somas were removed. In total, we collected 9,268 neurons: 4,370 from M1 and 4,898 S1 across 6 mice. Nueron ID was assigned based on the visible imaging window in relation to bregma using the CCF atlas.

To normalize the recorded fluorescent signal, the relative fluorescence change $\Delta F/F$ was calculated. The background fluorescence F for each neuron was estimated by median filtering (kernel size 120 seconds, padding with median values at the edges) the time course of the background signal extracted by the CNMF segmentation at the location of the neuron mask. The fluorescent increase ΔF was the extracted signal time course from the CNMF algorithm.
To extract the underlying neural activity, all signals used for analysis were deconvolved by fitting an autoregressive process to the recorded fluorescent signal (101). This deconvolution algorithm estimates the underlying neural activity that best explains the recorded fluorescent signal based on the calcium indicator dynamics. The assumption of this model is that the number of currently fluorescent molecules $c(t)$ is generated by a number of spikes $s(t)$ at times t paired with a decay of the signal $c(t) = \gamma_1 c(t-1) + \gamma_2 c(t-2) + s(t)$ with the decay constants γ_1 and γ_2. The recorded fluorescent signal $y(t)$ results from $c(t)$ according to $y(t) = \alpha(c(t)+b) + \epsilon(t)$ with a non-negative scalar α to scale the signal, the baseline fluorescence b and zero mean Gaussian noise $\epsilon(t)$ (101).

Building a functional, biologically realistic muscle-based mouse arm model

Building an anatomically accurate forelimb model. Following the modeling process of (102, 103), contrast-enhanced micro-computed tomography (CT) scanning was used to create a 3D model of the mouse's skeletal structure, which was then separated digitally into component parts. Eight mice were imaged using an Inveon Multimodality CT scanner using the following parameters: 360 projects in 1° steps, 80 kV peak tube voltage, 500 µA per projection, exposure time of 250 ms, and a resultant effective pixel size of 103.80 µm. The reconstruction of the CT scans was performed using the ordered subset expectation maximization (104) (Cobra version 6.3.39; Exxim Computing Corporation) algorithm with 10 iterations, slight noise reduction, and no filter or down-sampling. The reconstructed voxel size was 103.80 μm^3. Segmentation of the mouse skeleton was performed in 3D Slicer (105) using a threshold function applied to the reconstructed DICOM files. For all mice, a minimum value of 450 Hounsfield Unit (HU) was set for segmenting the skeleton and then manually fine-tuned.

The model was divided into 23 bone segments to allow forelimb articulation around 10 major joints (Tables 1 and 2). Specifically, the model was segmented into the main body, scapula, humerus, radius-ulna (treated as a single segment), carpals, and separate digit components. These constituent parts were then reassembled to form the arm model in MuJoCo, using the xml-based MJCF API. A slide joint and a hinge joint connect the scapula to the main body. The slide joint is aligned to allow movement along the anterior-posterior axis, and the hinge joint is connected perpendicular to the scapula to allow rotation around the lateral-medial axis. A ball joint is used for the shoulder joint connecting the scapula to the humerus, and a ball joint is used for the elbow joint connecting the humerus to the radius-ulna. The wrist joint and each of the finger joints are modeled using hinge joints.

In total there are 50 muscles defined in the model (Table 3), representing nearly all of the muscles in the mouse shoulder and arm and including all of the principle muscles affecting movement of the scapula (63, 64). The muscle origin and insertion points, which were defined as single points, were determined on the resulting 3D skeletons based on descriptions and diagrams from (63, 64).

Table 3. Mouse Arm 50 Muscle List - Grouped by Anatomical Location

No.	Muscle	No.	Muscle
Shoulder and Upper Back			
1	Acromiotrapezius	2	Clavotrapezius
3	Spinodeltoid	4	Spinotrapezius
5	Supraspinatus	6	Infraspinatus

Continued on next page

Table 3 – continued from previous page

No.	Muscle	No.	Muscle
7	Serratus Ventralis1	8	Serratus Ventralis2
9	Serratus Ventralis3	10	Subscapularis
11	Rhomboideus Cervicis	12	Rhomboideus Thoracis
13	Latissimus Dorsi	14	Teres Major
15	Teres Minor		
Upper Arm			
16	Triceps Brachii Lateral	17	Triceps Brachii Long
18	Triceps Brachii Medial	19	Biceps Brachii Long
20	Brachialis	21	Coracobrachialis
22	Pectoral Major Deep	23	Pectoral Major Superficial
Forearm			
24	Pronator Teres	25	Supinator
26	Ex Carpi Radialis Brevis	27	Ex Carpi Radialis Longus
28	Ex Carpi Ulnaris		
Wrist and Hand Extensors			
29	Ex Pollicis Longus	30	Ex Pollicis Brevis
31	Ex Indicis Proprius	32	Ex Digiti Quinti
33	Ex Digiti Quart	34	Ex Digitorum Communis
Wrist and Hand Flexors			
35	Fx Carpi Radialis	36	Fx Carpi Ulnaris
37	Fx Digiti Quinti Brevis	38	Fx Digitorum Profundis
39	Fx Digitorum Superficialis	40	Palmaris Longus
Intrinsic Hand Muscles			
41	Abductor Digiti Quinti	42	Abductor Pollicis
43	Opponens Digiti Quinti	44	Lumbricales
45	Interossei Dorsal 1	46	Interossei Dorsal 2
47	Interossei Dorsal 3	48	Interossei Volar 1
49	Interossei Volar 2	50	Interossei Volar 3

The fIa and fII muscle spindles and fIb Golgi tendon organ functions from Kibleur et al. (103), Prochazka and Gorassini (106, 107) and Moraud et al. (108) were used to generate the proprioceptive firing rate feedback. The mean parameter values were chosen to match those reported in Kibleur et al. (103), and different proprioceptors (i.e., we simplify to one for each type per muscle) were created by setting parameters using a Gaussian distribution with a standard deviation of 0.1 times the mean. Specifically, we implemented these models as the following:

$$f_{\text{Ia}}(t) = \max\{0, k_v v^M p_v + k_{dI}(l^M - l^M_{opt}) + k_{nI} a + c_I\}, \tag{2}$$

$$f_{\text{Ib}}(t) = \max\{0, k_f \frac{F^M}{F^M_{max}}\}, \tag{3}$$

$$f_{\text{II}}(t) = \max\{0, k_{dII}(l^M - l^M_{opt}) + k_{nII} a + c_{II}\}, \tag{4}$$

where l^M is the fiber length, l^M_{opt} is the optimal fiber length, v^M is the muscle fiber contraction velocity, a is the normalized muscle activity, F^M is the force exerted by the muscle fiber, F^M_{max} is the maximal isometric force, and c_*, k_*, and p_* are numerical coefficients determined in Prochazka (109). The l^M, v^M, F^M, and a parameters are read from MuJoCo at each time step. The l^M_{opt} parameter is calculated on model instantiation, and is a function of the length ranges of each muscle, as described see below. The F^M_{opt} parameter for each muscle is determined through an optimization process, also described below.

Building an analytical muscle-based controller. We will initially sketch the idea and then detail the mathematics below. First, we calculate a proportional-derivative (PD) control signal in joint-space, given a target trajectory for each of the joint

Table 1. Bones of the Mouse Arm

Bone Name	Length (cm)	Bone Name	Length (cm)
finger0_0	0.190855	finger2_0	0.208992
finger0_1	0.213156	finger2_1	0.192786
finger0_2	0.13053	finger2_2	0.163729
finger0_3	0.0993316	finger2_3	0.073004
finger1_0	0.222189	finger3_0	0.140258
finger1_1	0.251804	finger3_1	0.149808
finger1_2	0.137059	finger3_2	0.113512
finger1_3	0.0794417	finger3_3	0.0701662
metacarpals	0.185991	thumb_0	0.0777345
forearm (ulna + radius)	1.3834	thumb_1	0.0984265
humerus	1.24144	scapula	1.19748

Table 2. Mouse Arm Joint Parameters List

Joint	Joint Degree-of-Freedom	Joint Range-of-Motion
Scapula1	1 (slide)	-.4 to 1.0 cm
Scapula2	1 (hinge)	-28° to 90°
Shoulder	3 (ball)	–
Elbow	3 (ball)	–
Wrist	1 (hinge)	-115° to 115°
Fingers 1 - 4	1 each (hinge)	-6° to 42°

angles of the arm. We apply the inertia of the arm and gravitational forces at each joint to linearize the control of each arm segment, compensating for gravity and preventing the movement of arm segments from affecting each other. Second, the joint torques applied directly to the arm move the arm along the target trajectory. Lastly, to convert this joint torque control signal to a set of muscle activations, it is multiplied by the negative of the tendon Jacobian (transforming between joint space and tendon space), provided by the MuJoCo simulation engine (47). The resulting muscle activations apply the forces specified by the joint torque control signal to each joint as best as possible given the tendon configuration and muscle parameters.

We define a vector $\mathbf{q}$ containing the joint angles of the scapula, shoulder, elbow, wrist, and all digits (see Figure 1c), and $\dot{\mathbf{q}}$ containing their respective derivatives:

$$\mathbf{q} = \begin{bmatrix} q_{scapula_1} \\ q_{scapula_2} \\ \mathbf{q}_{shoulder} \\ \mathbf{q}_{elbow} \\ q_{wrist} \\ \mathbf{q}_{fingers} \end{bmatrix}, \quad \dot{\mathbf{q}} = \begin{bmatrix} \dot{q}_{scapula_1} \\ \dot{q}_{scapula_2} \\ \dot{\mathbf{q}}_{shoulder} \\ \dot{\mathbf{q}}_{elbow} \\ \dot{q}_{wrist} \\ \dot{\mathbf{q}}_{fingers} \end{bmatrix} \tag{5}$$

The units and ranges for the scalar elements in this vector are outlined in Table 2. Note that $\mathbf{q}_{shoulder} \in \mathbb{H}$ and $\mathbf{q}_{elbow} \in \mathbb{H}$ are quaternions, where $\mathbb{H}$ is the 4-dimensional unit hypersphere, and that the derivatives of quaternions are 3-dimensional.

The error signal, $\tilde{\mathbf{q}}$, is defined as a function of the target joint angles $\mathbf{q}^*$ and current joint angles $\mathbf{q}$. The error signal is calculated per joint, based on the type of joint. For each of the slide or hinge joints, the error calculation is simply the difference between the target and current positions. For the two ball joints in the arm model (shoulder and elbow), whose position in $\mathbf{q}$ is represented by 4D quaternions, the error is calculated as

$$\tilde{\mathbf{q}}_i = \frac{(\mathbf{q}_i^* \cdot \bar{\mathbf{q}}_i)_{[1,2,3]}}{\text{sign}((\mathbf{q}_i^* \cdot \bar{\mathbf{q}}_i)_{[0]})} \tag{6}$$

where $\mathbf{q}_i$ represents the 4D quaternions (corresponding to the shoulder or elbow joints), $\cdot$ represents quaternion multiplication, and $\bar{\mathbf{q}}$ denotes the conjugate of the quaternion $\mathbf{q}$. Note that the output of the error calculation for ball joints is a 3-dimensional

vector, even though the input are 4-dimensional quaternions. The full error signal is constructed by concatenating the 1D position errors for the two scapula joints, the 3D position error for the shoulder, the 3D position error for the elbow, and then the 1D position errors for the wrist and fingers:

$$\tilde{\mathbf{q}} = \begin{bmatrix} q^*_{scapula_1} - q_{scapula_1} \\ q^*_{scapula_2} - q_{scapula_2} \\ \tilde{\mathbf{q}}_{shoulder} \\ \tilde{\mathbf{q}}_{elbow} \\ q^*_{wrist} - q_{wrist} \\ \mathbf{q}^*_{fingers} - \mathbf{q}_{fingers} \end{bmatrix}. \tag{7}$$

We can now define the control signals generated for driving the model. Given the PD gain terms k_p and k_v, the joint space control signal $\mathbf{u}_{\text{joint}}$ is calculated as

$$\mathbf{u}_{\text{joint}} = \mathbf{M}(\mathbf{q})(k_p\tilde{\mathbf{q}} - k_v\dot{\mathbf{q}}) - \mathbf{g}(\mathbf{q}), \tag{8}$$

where $\mathbf{M}(\mathbf{q})$ denotes the inertia matrix and $\mathbf{g}(\mathbf{q})$ is the vector of gravitational torques. The joint space control signal $\mathbf{u}_{\text{joint}}$ is then transformed into muscle space using the Jacobian between joints and tendons, $\mathbf{J}(\mathbf{q})$, which captures how the movement of each tendon affects the movement of each joint:

$$\mathbf{u} = -\mathbf{J}(\mathbf{q})^T\mathbf{u}_{\text{joint}}. \tag{9}$$

The Jacobian $\mathbf{J}(\mathbf{q})$ is computed analytically based on the configuration of joints and the muscle state and attachment points in the MuJoCo model. Note, as a variation of muscle-based control, it is also possible to leverage $\mathbf{u}_{\text{joint}}$ as a control signal for joint-based control, using the same logic outlined above.

To note, while in this work we used generalized coordinates control with behavioral data to directly imitate real movements, MausSpaun supports other methods, which could be highly useful for generating synthetic data. For example, our approach extends to operational space control implementations, where the goal is to control the position of a point of interest on the arm (e.g., the tip of the index finger, or center of the back of the hand). In this case, the positional control term is calculated in end-effector space, given a target 6D trajectory (allowing for control of both end-effector position and orientation) for a specific point on the mouse arm, such as the palm of the hand or tip of a finger, to follow. We then transform this signal from end-effector space into joint space, using the Jacobian relating joint and end-effector space. From here, the process is the same as above, where compensation for inertia, gravity, and current velocity are added in, and the signal is transformed to muscle activations. Formally, the full control signal for muscle-based end-effector control is specified:

$$\mathbf{u} = -\mathbf{J}(\mathbf{q})^T(\mathbf{M}(\mathbf{q})(k_p\mathbf{J}_{\text{ee}}(\mathbf{x}^* - \mathbf{x}) - k_v\dot{\mathbf{q}}) - \mathbf{g}(\mathbf{q})), \tag{10}$$

where $\mathbf{J}_{\text{ee}}$ is the end-effector Jacobian, and $\mathbf{x}^*$ and $\mathbf{x}$ are the target state and current state of the end-effector. Note that only the calculation of the positional control term changes in this formulation, and the compensation for gravity and velocities remains the same. The interface with MuJoCo and the control algorithms are implemented using the 'abr_control' package (https://github.com/abr/abr_control).

Constraining kinematics with experimental behavior. There are three steps to our process of generating joint angle trajectories for the model mouse arm to follow that recapitulate the mouse movements from experimental sessions. The first step is using DeepLabCut (49, 65) to retrieve 3D estimates of keypoints on the mouse arm from experimental videos (see above), and identify corresponding sites on the mouse arm model in MuJoCo. The chosen keypoints were the shoulder, elbow, three points on the wrist, back of the hand, and each of the digits of the mouse.

Second, the frames of reference for the DeepLabCut keypoints need to be aligned with the MuJoCo frame of reference. To do this requires reference points that are consistent between the experimental videos and the MuJoCo model that can be used to determine the linear transformations required to move between these frames of reference. We chose to use points on the platform the mice stood on and were head-mounted to during the experiment as reference points for alignment. This required implementing a replica of the experimental platform in MuJoCo, which was done using the platform's 3D design CAD files. After identifying the 3D positions of the chosen platform keypoints from the experimental videos and in the model, the Procrustes linear transformation method (110) is used to transform the 3D pose data into the MuJoCo model reference frame.

Lastly, once the experimental data is aligned to the model reference frame, the inverse kinematics for each experimental trajectory are calculated using a minimization optimization to determine the joint angles that best align the experimental and model 3D keypoints throughout each trajectory. Specifically, we use the 'scipy.optimize.minimize' function with the limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm with an $\epsilon = 0.001$ step size. The mouse keypoints define the configuration of the arm fully enough that the optimization process will converge to the same minima every time it is run. To ensure this, however, we set the initial configuration of the arm for each step of the optimization process to be the resting state angles of the arm, and the cost function includes a term that penalizes distance from this resting configuration. Formally, our cost function is defined:

$$\mathbf{q} = \arg\min_{\mathbf{q}\in\mathbb{H}} \left(w_1 \|\mathbf{m}^* - \mathbf{m}(\mathbf{q})\| + (1 - w_1)\|\mathbf{q}_0 - \mathbf{q}\|\right), \tag{11}$$

where $\mathbf{m}^*$ denotes the 3D keypoints from the aligned experimental data, $\mathbf{m}(\mathbf{q})$ is the set of 3D keypoints from the model given the joint angles $\mathbf{q}$, $\mathbf{q}_0$ represents the resting configuration of the arm, and w_1 is a scalar weighting the relative contribution of each term, set to 0.75.

Tuning the forelimb model for functionality. There are a total of 507 parameters to tune in the arm model, including 5 stiffness and damping terms for each joint (digit joints are excluded and use default values), 50 tendon length range upper and lower bounds, and 50 muscle strength terms for each muscle. Additionally, the exact position of the attachment points for each tendon (muscle) can have an effect on the ability of the corresponding muscle to manipulate the arm. In total, 115 attachment points are tuned, each with x, y, and z values. In the controller there are only 2 parameters to tune, the control gains k_p and k_v.

The muscle length ranges determine the minimum and maximum length of the model muscles during simulation. To set the muscle length ranges, we took the inverse kinematics across the 26 sessions of experimental recordings (approximately 1,000 minutes of data), set the arm configuration to match the specified inverse kinematic joint angles at each time step, and recording the length of each muscles at every time step. The range of muscle lengths spanned throughout all movements was calculated for each muscle, and minimum and maximum length ranges were set to the minimum and maximum values throughout the movement $\pm$ half the range.

The remaining 407 parameters of the arm model and controller were optimized for functionality in 3 stages: 1) controller gains, 2) tendon attachment points, and 3) joint stiffness and damping and muscle strengths. The first two stages are optimized independently, and the third stage optimization is dependent on the results of the first two stages. To tune the model and controller for biologically realistic movements, a set of joint angle trajectories were identified to be reenacted by the arm and controller. Sixteen target trajectories were chosen by analyzing a set of labelled experimental videos and identifying sequences between 1 and 10 seconds where the mouse starts with its hand on the platform, reaches to the joystick, pulls it, releases the joystick, and returns its hand to the platform. After each stage of optimization ran until convergence, we tested the ability of the tuned model to recapitulate full experimental trajectories. Sections in these reenacted full experimental trajectories where the model performed poorly were identified, and ten second segments from these times were added to the trajectories used during optimization. The optimization process for the joint stiffness and damping and muscle strengths was then performed again with the larger set of target trajectories. We performed three such iterations, and a total of 27 trajectories were used to optimize the joint stiffness and damping and muscle strengths in the end. Our train error across all markers (excluding the lower-wrist marker, which had a high 3D error) was 0.266 cm $\pm$ 0.159, and test error was 0.24 cm $\pm$ 0.142. For reference, the hand marker train error was 0.193 cm $\pm$ 0.129, and the test error was 0.178 cm $\pm$ 0.116. To note, using these 27 snippets it takes roughly 7 minutes (on a workstation, Ubuntu 22.04.4 64-bit with Ryzen 9 5950x 16-core processor x 32 and 64GB memory) to evaluate a hyperparameter set. We evaluated 43,845 parameter sets to set the final best parameters (thus takes approx. 13 days).

In the first stage, we tune the PD controller gains using joint torque actuators to control the arm instead of muscle-based actuators. We do this because joint torque control is more precise than muscle-based control, and represents the best possible performance achievable by the muscle based controller. The cost function used during this stage is based on the difference between the experimental and simulated joint angles and 3D keypoints, and the simulate joint velocities. We include the 3D keypoints on the arm in the cost function to ensure that deviations at joints earlier in the kinematic tree (e.g., the shoulder or elbow) are weighted with a heavier penalty. The result of a discrepancy at the shoulder has a strong downstream effect, and can result in a trajectory that does not overlap with the experimental trajectory at all, even if all downstream joints follow their target trajectory perfectly. Additionally we include the weighted sum of velocity at each joint throughout the movement in the

cost term, to prioritize parameter sets that more "smoothly" while following the experimental trajectory. Formally, the cost function is:

$$cost = \sum_t \left(w_2 \tilde{\mathbf{q}}(t) + w_3 \|\mathbf{m}^*(t) - \mathbf{m}(t)| + w_4 * \dot{\mathbf{q}}(t) \right), \tag{12}$$

where $\mathbf{q}^*$ and $\mathbf{q}$ are the target and actual joint angle trajectories, $\mathbf{m}^*$ and $\mathbf{m}$ are the target and actual 3D key-point positions on the arm, $\dot{\mathbf{q}}$ is the joint angles velocities, and w_2, w_3, w_4 are gain terms. The gain terms were set to normalize the angle and 3D key-point costs relative to each other and the velocity term to 1/5th, based on values collected during a joint torque controlled movement: $w_2 = 0.01212$, $w_3 = 0.0144$, $w_4 = 2.94e-6$.

In the second stage, the x, y, and z locations of the 115 attachment points are tuned. The initial placements were made by matching the positions identified by the CT scan analysis as precisely as possible. To account for noise and small errors in the joint modeling and attachment point placement process, a hyperparameter search was conducted across each tendon's attachment points inside a 3 cubic millimeter space around the initial locations. This fine-tuning is an important step, because the effect each tendon is able to exert over the arm is tightly coupled with their exact attachment point locations relative to positions of joints they cross over.

To evaluate each set of attachment points we moved the arm through the test set of joint angle trajectories, calculated the manipulability term (111) for the tendon Jacobian at each point in time, and summed the manipulability across all of the time steps and each of the movements. The manipulability term is a scalar measure of the efficacy of actuators on the target task, calculated as the square root of the Gram determinant of the Jacobian. Formally,

$$m = \sqrt{\det(\mathbf{J}(\mathbf{q})^T \mathbf{J}(\mathbf{q}))}. \tag{13}$$

Calculating the manipulability term of the tendon Jacobian in our hyperparameter search provides a metric to evaluate how small variations in a tendon's positioning enable or degrade the amount of force a muscle is able to exert over the joints it crosses. The units of this metric are referred to as 'manipulability units', and we denote them as 'm' when reporting numbers in Suppl. Figure S1e.

Lastly, in the third stage of parameter tuning of the arm, the joint damping and stiffness and the muscle strengths were tuned. These results are dependent on the controller gains and attachment points from the first two stages of optimization. Technically, there is a stiffness and damping term for each of the tendons, in addition to the joint damping and stiffness and the muscle strengths terms. Our initial optimization attempts performed a full scale parameter sweep across the full 160 parameter space, which proved to be too large a problem for an effective search given the viable range for each of the parameters and the available software and computational resources. To overcome this obstacle, the space was heuristically reduced by optimizing for stiffness and damping only at the joints (and setting the stiffness and damping of each tendon to 0), and grouping muscles with similar positioning and effect. This reduction and grouping lead to a much more tractable search problem over 48 parameters instead of 231. The muscle groups used during our hyperparameter sweeps are listed in Table 4.

Table 4. Muscle groupings used during optimization of muscle strength parameters.

Muscle group	Muscles
pectoral	pectoral major deep, pectoral major superficial
serratus ventralis	serratus ventralis 1, serratus ventralis 2, serratus ventralis 3
rhomboideus	rhomboideus cervicis rhomboideus thoracis
trapezius	acromiotrapezius, spinatrapezius, clavotrapezius
spinitis	supraspinitis, infraspinitis
pollicis	ex pollicis brevis, ex pollicis longus
interossei dorsal	interossei dorsal 1, interossei dorsal 2, interossei dorsal 3
interossei volar	interossei volar 1, interossei volar 2, interossei volar 3

We used the same cost function defined in Eq. (12) for this stage of optimization. The search over these parameters was conducted iteratively, by analyzing the hyperparameter sets from a given search that achieved the best results and adjusting the parameter ranges when there was clustering at a limit.

For all of the parameter sweeps, the Neural Network Intelligence 2.5 (NNI; 112) software was used to conduct the search, which provides access to optimized hyperparameter-space search methods. The controller gain terms used in the final model were $k_p = 11769$ and $k_v = 193$. The final model's joint damping, stiffness, and muscle strength terms are listed in Table 5.

Table 5. Muscle Stiffness and Damping Parameters List. Scale is in relation to the normalized muscle activation profile in MuJoCo.

Name	Stiffness range (N/m)	Damping range	Length range (cm)	Scale range (N)
joint scapula1	2.5 - 12.5	325 - 1720	0	0
joint scapula2	2.5 - 7.5	11455 - 34315	0	0
joint shoulder	2.5 - 20	727.5 - 1542.5	0	0
joint elbow	0 - 15	1852.5 - 2892.5	0	0
joint wrist	0 - 15	60 - 222.5	0	0
latissimus dorsi	0	0	0.880 - 3.309	500 - 2675
teres major	0	0	0.501 - 1.182	125 - 3775
teres minor	0	0	0.640 - 1.238	5875 - 7075
infraspinitis	0	0	0.995 - 1.648	25 - 150
corachobrachialis	0	0	0.578 - 0.836	30075 - 54600
brachialis	0	0	0.846 - 1.398	4640 - 5050
triceps brachii long	0	0	0.753 - 1.526	3050 - 8175
triceps brachii lateral	0	0	0.897 - 1.213	775 - 2475
triceps brachii medial	0	0	0.633 - 0.839	400 - 2300
biceps brachii long	0	0	0.784 - 1.419	11350 - 18275
spinodeltoid	0	0	0.388 - 1.217	550 - 3800
supinator	0	0	0.602 - 0.876	375 - 750
abductor digiti quinti	0	0	0.247 - 0.376	4350 - 6250
ex indicis proprio	0	0	1.931 - 2.100	475 - 24200
fx digiti quinti brevis	0	0	0.254 - 0.374	1900 - 5200
opponens digiti quinti	0	0	0.270 - 0.346	4400 5425
lumbricales	0	0	0.622 - 0.881	175 - 6025
ex carpi ulnaris	0	0	1.134 - 1.397	1900 - 7175
fx carpi ulnaris	0	0	1.035 - 1.514	11150 - 18950
ex digiti quarti	0	0	1.809 - 1.974	125 - 1450
palmeris longus	0	0	0.929 - 1.497	3575 - 16350
ex digiti quinti	0	0	1.300 - 1.570	475 - 4450
fx digitorum superficialis	0	0	1.878 - 2.463	25 - 1375
ex digitorum communis	0	0	3.733 - 4.032	16300 - 24025
fx digitorum profundis	0	0	2.107 - 2.515	100 - 200
ex carpi radialis brevis	0	0	1.302 - 1.513	8025 - 11150
fx carpi radialis	0	0	0.997 - 1.582	700 - 2300
ex carpi radialis longus	0	0	1.277 - 1.488	100 - 1850
pronator teres	0	0	0.397 - 0.944	100 - 3950
pectoral major deep	0	0	0.298 - 1.705	15900 - 19375
pectoral major superficial	0	0	0.119 - 1.705	15900 - 19375
serratus ventralis1	0	0	-0.519 - 2.248	1550 - 5125
serratus ventralis2	0	0	-0.340 - 2.135	1550 - 5125
serratus ventralis3	0	0	-0.161 - 2.082	1550 - 5125
rhomboideus cervicis	0	0	-0.136 - 2.471	19975 - 45000
rhomboideus thoracis	0	0	0.009 - 2.503	19975 - 45000

Continued on next page

Table 5 continued from previous page

Name	Stiffness range	Damping range	Length range	Scale range
acromiotrapezius	0	0	0.557 - 1.581	11900 - 39650
spinotrapezius	0	0	0.212 - 3.421	11900 - 39650
clavotrapezius	0	0	0.455 - 1.926	11900 - 39650
supraspinatus	0	0	1.006 - 1.327	25 - 150
abductor pollicis	0	0	0.109 - 0.290	88425 - 96525
adductor pollicis	0	0	0.069 - 0.109	88425 - 96525
fx pollicis brevis	0	0	0.089 - 0.129	88425 - 96525
ex pollicis brevis	0	0	0.632 - 0.789	2975 - 9125
ex pollicis longus	0	0	0.746 - 0.903	2975 - 9125
interossei dorsal 1	0	0	0.382 - 0.451	300 - 5025
interossei dorsal 2	0	0	0.452 - 0.481	300 - 5025
interossei dorsal 3	0	0	0.412 - 0.446	300 - 5025
interossei volar 1	0	0	0.226 - 0.423	425 - 1200
interossei volar 2	0	0	0.286 - 0.447	425 - 1200
interossei volar 3	0	0	0.276 - 0.430	425 - 1200

Building MausSpaun: a modular large-scale neural model

SPAUN (66, 113) and REACH (114) models form the foundation of the MausSpaun brain model. In this work, we integrate the framework and methods of these models and focus on the problem of movement control. The MausSpaun brain represents several different brain areas, inspired by known brain functions, modelled as performing functionally distinct operations (see also Figure 2a):

1. A "**Dynamic Motor Primitives** (aka pre-motor cortical areas (PMC))", which encodes joint-space trajectories from experimental recordings, broken down into movement primitives, whose timing and order is controlled by inhibition from the the basal ganglia module (see below). It also includes an "oscillator", which serves as a 'clock' determining when transitions are made between actions, and an "inhibition" area that directly interfaces with the Kinematics Controller (see below).

2. A "**Kinematics Controller** (akin to primary motor cortex (M1))", which generates a position control signal to move the arm along the target trajectory.

3. A "**Dynamics Controller** (also akin to M1)", which generates a velocity control signal that compensates for the effects of inertia and the force of gravity on the arm.

4. A "**State estimator/sensory feedback** (akin to somatosensory cortex (S1))", which takes in feedback from the arm and projects the current joint angle and velocity to M1 and the state to the error region.

5. An "**Error**" cortical region that calculates how to move each joint to minimize the difference between the target position provided by the PMC and the system state provided by S1.

6. A "**Basal Ganglia (BG)**", which acts as a neural state machine that controls the flow of information in the network and carries out a set of pre-programmed automations.

7. An "**Action State Module** (akin to brainstem)" that encode the prior state, current state and desired action.

We take advantage of the modularity of the network architecture to implement a subset of the components in spiking neurons and simulate the rest without using neurons. The full model architecture is shown in Suppl. Figure S2a,b. All spiking neural networks are implemented using standard Leaky-Integrate and Fire (LIF) neuron models, with firing rates between 200-400Hz, and post-synaptic filter values based on neurotransmitters in the corresponding part of the brain. The use of neurons with high firing rates is a computational convenience; the same population level performance can be achieved using more neurons

with lower, more biologically realistic, firing rates. We use a 2D oscillator (implemented in spiking neurons) as the 'clock', to determine when transitions are made between actions. The oscillator is implemented using the methods of the Neural Engineering Framework (described in brief below) to implement standard 2D oscillator dynamics:

$$\dot{\mathbf{x}} = k_o \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \mathbf{x}, \tag{14}$$

where k_o is a scalar term that controls the speed of the oscillator, and $\mathbf{x} \in \mathbb{R}^2$ is the oscillator state vector. The oscillator is initialized to represent [1, 0] at the beginning of a simulation, and when the oscillator returns to [1, 0] this is the signal to the action selection system to move to the next action in the sequence. At the same time, during this period of changing current actions, an inhibitor signal is sent to the Kinematics Controller (shown with the red connections ending in circles) model which prevents any output during the transition. Without any input from the Kinematics Controller, the arm will simply hold its current position, as the Dynamics Controller model works to minimize any velocity or gravity forces moving the arm. The Kinematics and Dynamics controllers are not implemented in spiking neurons, but analytically (see compiler, below).

The action selection system (fully, and necessarily, implemented in spiking neurons) consists of a basal ganglia and thalamus model, and several working memories representing what we call the brainstem of the mouse. This system, described in detail below, implements a fast and robust soft winner-take-all circuit, and is based upon detailed physiology analysis. The only input to the action selection system in the MausSpaun model is a repeating linear ramp based on the state of the 2D oscillator.

The output from the action selection system projects inhibitory signals to the PMC, where spiking neuron populations represent the available actions, and activity in the unselected action areas is inhibited. There are four PMC populations, each of which implement neural Dynamic Movement Primitives (nDMPS (115)) to generate target trajectories, and are described in detail below. The uninhibited PMC population then projects a set of target joint angles for the arm into to an 'error signal' area which calculates the difference between the current state of the arm and the target state from PMC.

The error between the current and target positions, $\tilde{\mathbf{q}}$, is calculated as shown in Eq. (6), and is projected to the Kinematics Controller (M1). M1 also receives the current joint angles of the arm to calculate the tendon Jacobian, which allows it to transform the joint space control signal into a set of muscle activations. The equation calculated by the Kinematics Controller is

$$\mathbf{u}_{\text{kinematics}} = -\mathbf{J}(\mathbf{q})^T \mathbf{M}(\mathbf{q}) k_p \tilde{\mathbf{q}} \tag{15}$$

The Dynamics Controller (M1) also receives the current joint angles (from the state estimator) to transform from joint torques to muscle activations, as well as the joint velocities (so that it can generate a compensatory force to cancel them out). The equation calculated by the Dynamics Controller is

$$\mathbf{u}_{\text{dynamics}} = \mathbf{J}(\mathbf{q})^T (\mathbf{M}(\mathbf{q}) k_v \dot{\mathbf{q}} + \mathbf{g}(\mathbf{q})) \tag{16}$$

The sum of the Kinematics and Dynamics Controllers is the muscle activations control signal presented in Eq. (9). S1 receives the output from the arm, and relays the joint angles and joint velocities to the error and M1 populations (see also Figure 2a). The S1 module is implemented analytically in this MausSpaun model.

To implement the MausSpaun model, we use the Nengo neural modeling and simulation package (https://github.com/nengo/nengo). Nengo is Python based software that provides a frontend API for scripting complex neural models quickly, and multiple backends that allow efficient simulation on many different kinds of hardware, including CPUs, GPUs, and specialized neuromorphic hardware (116). Nengo also has a GUI which allows users to visualize their network architectures, the neural activity, and the flow of information through the network while it simulates (https://github.com/nengo/nengo-gui). Neural models built in Nengo can be developed using machine learning methods or the Neural Engineering Framework (NEF; 70).

The NEF can be thought of as a 'neural compiler', which provides an approach to developing spiking neural network models using population decoding to represent and transform input signals in a vector space. Briefly, for a neuron i, the neuron's preferred direction is represented by an encoder, $\mathbf{e}_i$, and the neuron's activity, a_i, is determined by

$$a_i = G(\alpha_i \mathbf{e}_i \cdot \mathbf{x} + b_i) \tag{17}$$

where G is a neural non-linearity, α_i is a gain parameter, $\mathbf{x}$ is the input signal, and b_i is the constant background bias for the neuron. The neuron's activity is sampled over the range of input values expected. Based on this sampling and a target function to perform on the vector representation, a set of 'decoder' values, $\mathbf{d}$, are calculated which guarantee a globally optimal approximation of the computation. Formally,

$$f(\mathbf{x}) = \Sigma a_i \mathbf{d}_i, \tag{18}$$

where $f(\mathbf{x})$ is the target function of $\mathbf{x}$ to be calculated. The connection weight, ω, between two neurons i and j is then defined as

$$\omega_i = \mathbf{d}_i \cdot \mathbf{e}_j. \tag{19}$$

Solving for the decoders $\mathbf{d}$, we can build neural networks with rate or spiking neurons that can represent and perform transformations on vector-based signals. For spiking neuron implementations, the post-synaptic filter is used to smooth out the noise from a spiking approximation of rate neurons. To be able to implement ordinary differential equations in spiking neural networks (as necessary for our PMC and working memory models), properly accounting for the time constants on the post-synaptic filters is necessary. Assume we have a value $\mathbf{x}$ being represented, and want to implement the following dynamics:

$$\dot{\mathbf{x}} = \mathbf{A}(\mathbf{x}) + \mathbf{B}(\mathbf{u}), \tag{20}$$

where $\mathbf{x}$ is the value being represented, $\mathbf{u}$ is some input signal, and $\mathbf{A}$ and $\mathbf{B}$ are arbitrary functions. To accurately implement this function, a recurrent connection will be used to implement the first term on the right-hand side. It is necessary to know the time constant, τ, on this recurrent connection and adjust the function calculated to $\tau\mathbf{A}(\mathbf{x}) + \mathbf{x}$. By then setting the input function to $\tau\mathbf{B}(\mathbf{u})$, we are able to accurately implement dynamics in spiking neural networks (There has been an extension to these methods that provides greater accuracy by accounting for a discrete implementation of continuous time dynamical systems. For simplicity, we present the original formulation and refer the reader to Voelker and Eliasmith (117) for further details).

Dynamic Motor Primitives: the pre-motor cortex for trajectory generation. We encode joint space trajectories determined by the DeepLabCut tracking and inverse kinematics process using neural Dynamic Movement Primitives (nDMPS; preliminary presented (115)). First we review the equations for DMPS, and the present the implementation of nDMPS.

DMPs are stable dynamical systems used to generate spatially and temporally flexible trajectories, and can be extended to include desirable movement features such as obstacle avoidance and online compensation for unexpected deviations based on sensory feedback (118, 119). There are two primary parts to a DMP: an attractor system and a forcing function. The attractor system has standard spring dynamics that pull the system towards a specified target:

$$\ddot{y} = k_a(k_b(y^* - y) - \dot{y}), \tag{21}$$

where y, $\dot{y}$, $\ddot{y}$ are the position, velocity, and acceleration of the attractor, y^* is the final target position, and k_a and k_b are gain terms. For our implementation, we use $k_a = 1000$ and $k_b = 250$. The forcing function is a supplementary term added to the spring dynamics that push the system along a path of interest as it converges to the specified goal. The forces of the forcing function are calculated through the difference of the target trajectory acceleration and the spring attractor dynamics:

$$f(t) = \ddot{y}^*(t) - k_a(k_b(y^* - y^*(t)) - \dot{y}(t)), \tag{22}$$

where y^*, $\dot{y}^*$, and $\ddot{y}^*$ are the trajectory position, velocity, and acceleration vectors at time t. The full formulation of the DMP dynamics include additional terms that allow temporal scaling and remove dependence on t, but this is unused in the current implementation, and so we leave it out for simplicity. The full DMP attractor dynamics for our implementation are then:

$$\ddot{y}(t) = k_a(k_b(y^* - y) - \dot{y}) + f(t) \tag{23}$$

Importantly, the structure of DMPs is simple and can be accurately implemented in spiking neural networks. Neural DMPS are an implementation of DMPs using spiking neural networks, built with the methods of the NEF. For each dimension of the trajectory, a population of neurons is instantiated to follow point attractor dynamics, and the last position in the trajectory is projected into the population as a convergence point. The forcing function is decoded out of a population of neurons (the 'ramp') that represents a signal that linearly moves from 0 at the beginning of the trajectory to 1 at the end of the trajectory.

The input to the ramp population is provided by the 2D oscillator in the network. The connection from the 2D oscillator to the ramp is 0 when the 2D oscillator representing an angle between $\frac{3\pi}{2}$ and $\frac{\pi}{2}$, and the output signal to the ramp increases from 0 to one as the oscillator angle moves between $\frac{\pi}{2}$ and $\frac{3\pi}{2}$. Specifically, we calculate the ramping signal r as

$$r = \frac{(\arctan(y[1], y[0]) - \frac{\pi}{2})}{\pi}. \tag{24}$$

For this implementation of MausSpaun, we have chosen to have a 'trajectory library' of nDMPs all driven by a single oscillator that outputs the ramping function to each nDMP, and all project directly to the kinematics control system. The role of the action selection system is to determine which of these nDMPs in the trajectory library should be active at any given point in time, and inhibit the rest. Each nDMP has 27 recurrent populations implementing the dynamics presented in Eq. (23) for each of MausSpaun's 27 positional dimensions.

Each neural population in the nDMPs model has 500 Leaky-Integrate and Fire (LIF) neurons, with firing rates between 200 and 400 Hz. We note that the high firing rates here are for simulation convenience. Lower firing rates are more realistic, but require more neurons and result in no functional difference. The target functions for each connection are sampled evenly throughout the unit hypersphere, and the decoders are computed through a least-squares calculation to solve Eq. (18).

A neural state machine: the basal ganglia. In Gurney et al. (120) a computational model of the basal ganglia, implemented using rate neurons, is presented showing how a highly scalable and efficient winner-take-all operation can be performed in the cortico-basal-ganglia-thalamic loop. As part of this model, the authors carefully account for the known biological constraints on connectivity and types of neurons in the basal ganglia. Briefly, the model is composed of five areas in the basal ganglia: Striatum D1 cells, striatum D2 cells, the subthalamic nucleous (STN), the globul pallidus external (GPe), and the globus pallidus internal (GPi)/substantia nigra (SNr).

Input 'utility' signals for each action are projected into the striatum D1 and D2 cells as well as the STN, and the output from GPi/SNr corresponds to the number of actions to choose between. The output from the GPi/SNr is inhibitor, so activation of output cells in this region means inhibition of the corresponding action. The striatum D1 cells project inhibitory activity to each of the corresponding action output areas in the GPi/SNr, the higher the utility of an action, the larger the corresponding inhibition of the GPi/SNr cells. At the same time, the STN projects a weighted excitatory signal representing the sum of all of the action utilities to the GPi/SNr. These two connections work together to excite GPi/SNr areas representing lower utility actions, and inhibit GPi/SNr areas representing higher utility actions. This circuit is effective, but not robust to wide ranges of utility values (e.g., all utility values very large or very small). The GPe is used as a modulator in this model, with connections in from striatum D2 cells and the STN that form the same type of circuit the striatum D1 cells and STN form with the GPi/SNr. However, the GPe projects back to the STN as well as forward to the GPi/SNr. This regulates the action selection system, allowing it to function robustly against a wide range of values.

The circuit diagram from this work was then implemented in spiking neurons with a population decoding representation of input actions in ste (121) using the NEF methods. The spiking neuron model includes the use of realistic neurotransmitter time constants in the synapse models, modeling inhibitory GABA connections with a longer time constant of 10 milliseconds and excitatory AMPA-type connections with a faster 5 millisecond time constant. The authors of ste (121) show how the spiking neuron model with realistic neurotransmitter time constants results in a mechanistic explanation for the commonly used 50 millisecond cognitive cycle time required for decisions (e.g., Anderson et al. (122)). The spiking neuron basal ganglia model that we use here is from these works, and implemented using the NengoSPA `https://www.nengo.ai/nengo-spa` package, which provides a Python API for implementing instruction sets using the basal ganglia. Eliasmith et al. (66) and Choo (113) showed how this model could then be extended and used to control the flow of information throughout large scale functional brain models implement and general instruction following.

For MausSpaun, we take advantage of this instruction following functionality to implement a neural state machine, that controls the timing and sequence of four movements in a reaching-and-pulling behavior. Using the NengoSPA package, we define a set of actions available for the model to choose between, a set of working memories for storing relevant information while running, and the rules for determining which action should be chosen given the current input and working memory states. For MausSpaun, we create three working memories: 'state', 'previous_state', and 'action', which store information about the current state of the arm, the previous state of the arm, and the last action taken. States can be 'AT_REST', 'AT_START', or

‘AT_TARGET‘, which represent the mouse with its hand on the podium, the mouse with its hand on the joystick, and the mouse mid-pull when the joystick is in the region where a reward is delivered. Available actions include ‘REST_TO_START‘, ‘START_TO_TARGET‘, ‘TARGET_TO_START‘, and ‘START_TO_REST‘. The transition between states is dependent on the context in the “action state” module and the state of the ramping input from the oscillator.

- if state == ‘AT_REST’ and ramp < 1 then action = ‘REST_TO_START’
- if state == ‘AT_REST’ and ramp == 1 then state = ‘AT_START’ and previous_state = ‘AT_REST’
- if state == ‘AT_START’ and previous_state == ‘AT_REST’ and ramp < 1 then action = ‘START_TO_TARGET’
- if state == ‘AT_START‘ and previous_state == ‘AT_REST‘ and ramp == 1 then state = ‘AT_TARGET‘ and previous_state = ‘AT_START‘
- if state == ‘AT_TARGET‘ and ramp < 1 then action = ‘TARGET_TO_START‘
- if state == ‘AT_TARGET‘ and ramp == 1 then state = ‘AT_START‘ and previous_state = ‘AT_TARGET‘
- if state == ‘AT_START‘ and previous_state == ‘AT_TARGET‘ and ramp < 1 then action = ‘START_TO_REST‘
- if state == ‘AT_START‘ and previous_state == ‘AT_TARGET‘ and ramp == 1 then state = ‘AT_REST‘ and previous_state = ‘AT_START‘

The combination of this instruction set implemented in the basal ganglia model, the working memories (described in the next section), and the ramping input signal from the network’s oscillator result in an action selection system that progresses reliably through the four sub-trajectories of an overall reach-and-pull motion.

Action State Modules: the brainstem for action-state memory. MausSpaun has three working memories, storing the current state, action, and previous state. The input to the working memories are 16-dimensional semantic pointer vectors encoding one of three possible states (‘AT_REST’, ‘AT_START’, or ‘AT_TARGET’) or one of four possible actions (‘REST_TO_START’, ‘START_TO_TARGET’, ‘TARGET_TO_START’, ‘START_TO_REST’). Each working memory network is composed of two ensembles of neurons of spiking neurons with a recurrent connection implementing integrator dynamics. Using the NEF methods to account for the post-synaptic filter time constants, the full dynamical equation for each integrator is:

$$\dot{\mathbf{x}} = (\tau + \mathbf{I})\mathbf{x} + \tau\mathbf{u}, \tag{25}$$

where $\mathbf{x}$ is the 8-dimensional vector represented by each population, $\mathbf{u}$ is the input semantic pointer, and τ is the time constant of the recurrent connection (set to 100 milliseconds).

These working memory implementations take advantage of the saturation points (maximum firing rates) of spiking neurons. All semantic pointer vectors are of unit length, so it is possible to accurately represent any semantic pointer input vector by representing the correct direction. For example, in 1-dimensional space it is possible to represent two semantic pointers: +1 and -1. By setting the representational limits of the population (i.e., the value represented by max firing rate of neurons tuned in that direction) to be of unit distance, we can rely on the correct value being represented by the integrator given sufficient exposure to the input.

This intuition extends to multiple dimensions representing any vector on the unit hypersphere: as long as the input is presented long enough to override the current state (on the order of 1 second in 8-dimensional space) the integrator will be pushed to store the input signal and maintain this representation until a new input is presented. As mentioned, we are using two 8-dimensional populations instead of one 16-dimensional population, and so the full output will not be exactly the direction of the input semantic pointer, but this approach has proven sufficiently accurate for implementing complex cognitive behaviors (123–125).

Run Time for generating synthetic data. On a workstation (Ubuntu 22.04.4 64-bit with Ryzen 9 5950x 16-core processor x 32 and 64GB memory) it takes 22.13 seconds to simulate 8 seconds of data. Here, we computed 139,624,609 time steps from 26 sessions (recorded at 75 Hz).

State Estimator Modeling

The force field operates orthogonal to the pull axis. Motor adaptation can thus be analyzed by modeling the changes of the perpendicular deviation from the pull axis. We utilized the state estimator model previously deployed in Mathis et al. (4), which was modified from Izawa and Shadmehr (126). We built a discrete dynamical system that models the evolution of the scalar perpendicular deviation of the mouse hand and how sensory prediction error driven signals act upon it. We model the perpendicular displacement of the joystick x_k at a particular position and in trial k is the consequence of a motor command u_k, the applied perturbation p_k, and normally distributed motor noise $\eta_k^m \sim \mathcal{N}(0, \sigma_m^2)$, such that:

$$x_k = u_k + p_k + \eta_k^m. \tag{26}$$

We assume that the mouse internally estimates its hand position by $y_k = x_k + \eta_k^s$ and also modified by sensory noise $\eta_k^s \sim \mathcal{N}(0, \sigma_s^2)$. We assume that their internal estimate of the perturbation is given by a discrete dynamical system with retention factor r (less or equal to 1; fitted based on data) and memory noise $\eta_k^p \sim \mathcal{N}(0, \sigma_p^2)$:

$$\widehat{p_{k+1}} = r\widehat{p_k} + \eta_k^p. \tag{27}$$

The goal of this modeling is to estimate internal motor control policy the mice use to adapt. These are then used as proxy variables to measure their effect on neural encoding (both at the single cell and population level). Thus, to model the motor command u_k, we assume the mice estimate their hand position $\widehat{x_k}$ based on the efference copy of their motor signal and their internal estimate of the perturbation $\widehat{p_k}$ by:

$$\widehat{x_k} = u_k + \widehat{p_k}. \tag{28}$$

Thus, the optimal behavioral control policy can be derived from the optimal state estimator and the optimal policy (27, 126, 127).

Sensory prediction error driven learning. As used in Mathis et al. (4), the state estimator is given by the Kalman filter (28, 126, 128, 129). The generative model – internal model – can be written as a state evolution model:

$$\mathbf{x}_k = \mathbf{F}\mathbf{x}_{k-1} + \mathbf{u}_k + \eta_k, \tag{29}$$

with state transition model $\mathbf{F} = \begin{pmatrix} r & 0 \\ 1 & 0 \end{pmatrix}$, control input model $\mathbf{u}_k = \begin{pmatrix} 0 \\ u_k \end{pmatrix}$, process noise $\eta_k \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$ with $\mathbf{Q} = \begin{pmatrix} \sigma_p^2 & 0 \\ 0 & \sigma_m^2 \end{pmatrix}$ and states $\mathbf{x}_k = \begin{pmatrix} p_k \\ x_k \end{pmatrix}$. The observer model is given by:

$$y_k = \mathbf{H}\mathbf{x}_k + \eta_k^s, \tag{30}$$

with measurement matrix $\mathbf{H} = \begin{pmatrix} 0 & 1 \end{pmatrix}$ and sensory noise $\eta_k^s \in \mathcal{N}(0, R)$ with $R = \sigma_s^2$.
In this situation the classical results by Kalman imply that the optimal state estimation is given by recursively iterating (128, 129):

$$\widehat{\mathbf{x}_k^-} = \mathbf{F}\widehat{\mathbf{x}_{k-1}} + \mathbf{u}_k \tag{31}$$

$$\widehat{\mathbf{x}_k} = \widehat{\mathbf{x}_k^-} + \mathbf{K}_k(y_k - \mathbf{H}\widehat{\mathbf{x}_k^-}). \tag{32}$$

The first equation describes the *a priori* prediction of the hand position and perturbation. The subsequent state estimate is a combination of this a-priori estimate and the sensory prediction error (SPE), $y_k - \mathbf{H}\widehat{\mathbf{x}_k^-}$, gated by the Kalman gain matrix, $\mathbf{K}_k$ (i.e., $\mathbf{K}_k = \begin{pmatrix} k_1 \\ k_2 \end{pmatrix}$). The Kalman gain matrix is updated according to:

$$\mathbf{P}_0^- = \mathbf{Q} \tag{33}$$

$$\mathbf{P}_k^- = \mathbf{F}\mathbf{P}_{k-1}\mathbf{F}^T + \mathbf{Q} \tag{34}$$

$$\mathbf{K}_k = \mathbf{P}_k^-\mathbf{H}^T \cdot (\mathbf{H}\mathbf{P}_k^-\mathbf{H}^T + R)^{-1} \tag{35}$$

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H})\mathbf{P}_k^-. \tag{36}$$

Given this state estimate, the optimal command to compensate for the perturbation is: $u_k = -\widehat{p_k}$. $\mathbf{F}$ is the model dynamics. R is the measurement noise covariance, representing the uncertainty in the measurements. $\mathbf{P}_k^-$ is the predicted (*a priori*) error covariance matrix at time step k. It estimates the uncertainty before the measurement update. $\mathbf{F}$ is the state transition matrix, which models how the state evolves from time step $k-1$ to k. I: The identity matrix.

To fit the model, we supply the perpendicular deviation during the force field per trial, averaged across mice. Then we fit during only the perturbation epoch (trials 76-175) with the differential evolution algorithm (from scipy). We estimated the p_k to be the mean of the first three adaptation trials across sessions.

Generalized Linear Modeling (GLM)

One of the primary goals of our study was to establish which features – posture, dynamics (velocity, torque), or muscles – are encoded in individual sensorimotor neurons in the mouse M1 and S1 forelimb areas, as this had not been previously described at this resolution with matched behavioral kinematics. To this end, we used Generalized Linear Models (GLMs) as adapted in prior works (17, 51) to predict the neuronal responses across the sessions.

To examine how much variance of the neural activity could be directly explained by behavioral variables, a GLM with an exponential nonlinearity was used. The deconvolved neural activity was binarized (with a threshold of 0.01). We constructed six design matrices containing different features derived from the different groups (Figure 4a-c and see Table below). Features were individually min-max-scaled to a range (0,1) prior to inputting them to the GLM. Binarized activity was predicted from a single time lag to limit the number of regressors in the design matrix. To note, this is therefore a stringent criteria that will decrease the fraction of neurons considered tuned. The GLM model is then parametrized as

$$r_i = \exp\left(\mathbf{x}^\top \mathbf{w}^{(i)} + w_0\right) \tag{37}$$

where r_i is the rate of neuron i predicted from regressors $\mathbf{x}$ (including a bias term w_0), and $\mathbf{w}^{(i)}$ are the regression weights.

To compensate for behavioral variability within sessions, all models were trained and tested on five splits of the data (each with 20% test set in three equally spaced blocks). Predictions on the test portion of the dataset are then combined, and used to compute McFadden's Pseudo-R^2,

$$\text{cv-pR}^2 = 1 - \frac{\log L(\mathbf{w}, w_0)}{\log L(0, w_0)}, \tag{38}$$

where $\log L(\mathbf{w}, w_0)$ is the log-likelihood of the fitted model, and $\log L(0, w_0)$ is the log-likelihood of the null model (a model with only the intercept). We then used a pseudo-R^2 threshold of 0.025 and only further considered neurons above this threshold. This was determined based on capturing approx. 40-50% of the variability, as used in primate sensorimotor studies (17), and matches prior reports of fraction of tuned neurons in mouse M1 (83, 130).

We fit these GLM models on all six regression groups defined in Table 6. Within each group, we analyse feature importance by further splitting the regressors into subgroups corresponding to body parts, and the type of signal (positions, velocities, joint angles, joint velocities, etc.). We then analysed the feature importance within each subgroup. For analyzing feature importance, we used the regression weights $\mathbf{w}^{(n)} \in \mathbb{R}^d$. For a subgroup G (here, a subset of indices from $1...d$) of regressors, we computed the length of the regression weight vector,

$$\|\mathbf{w}_G\| = \sqrt{\sum_{i \in G} w_i^2}. \tag{39}$$

We obtain this quantity for every neuron in the dataset, and denote the corresponding regression weight lengths $\|\mathbf{w}_G^{(n)}\|$ for each neuron n in the dataset. For groups with at least two subgroups, we test the regressor weights on mixed selectivity to both subgroups. The distribution of $\|\mathbf{w}_G^{(n)}\|$ across neurons n approximately followed a log-Normal distribution. We first determined the winning subgroup for each neuron by

$$G_n^* = \arg\max_G \|\mathbf{w}_G^{(n)}\| \tag{40}$$

and then computed the mean and standard deviation $\mu = \frac{1}{N}\sum_{n=1}^{N} \log\|\mathbf{w}^{(n)}\|$ and $\sigma = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(\log\|\mathbf{w}^{(n)}\| - \mu)^2}$ across the regression weights on the five data splits during cross-validation. To consider a neuron to be tuned to the group G^*, we required that

$$\log\|\mathbf{w}_{G^*}\| - \mu > \sigma, \tag{41}$$

and neurons that cross the significance threshold for multiple groups without fulfilling this criterion are called mixed selective neurons. At the end of this process, we obtained assignment of each neurons to none, one, or multiple subgroups within each of the six high-level groups.

We also assigned neurons to one of the six high-level groups for part of the analysis by a winner-takes-all approach within the neurons exceeding a pseudo-R^2=0.025 on at least one level.

Finally, we ran a variant of GLMs trained on each individual regressor within each group. We performed a sub-selection based on the highest tuned regressors from the multi-regressor analysis by level. By this, we obtained pseudo-R^2 values for every neuron relative to every regressor. For these unique contribution GLMs we additionally considered water rewards. We filtered the binarized valve opening time (recorded in hardware TTL) with a Gaussian filter ($\sigma = 500$) to consider the period when the mice consume the rewards.

Table 6. Overview of six different groups of regressors in our GLM analysis.

Group	Components
Group 1 - 3D Kinematics (position and velocity)	scapula, scapula_velocity, scapula2, scapula2_velocity, shoulder, shoulder_velocity, elbow, elbow_velocity, wrist, wrist_velocity, back of hand, back of hand_velocity, fingers (0-2, averaged), fingers_speed (0-2, averaged), (x, y, z for each)
Group 2 - Joint Space Kinematics (joint angles & angular velocity)	ja_hinge_joint_scapula, jv_hinge_joint_scapula, ja_slide_joint_scapula2, jv_slide_joint_scapula2 ja_ball_joint_shoulder_a, ja_ball_joint_shoulder_b, ja_ball_joint_shoulder_c, ja_ball_joint_shoulder_d, jv_ball_joint_shoulder_a, jv_ball_joint_shoulder_b, jv_ball_joint_shoulder_c, jv_ball_joint_shoulder_d, ja_ball_joint_elbow_a, ja_ball_joint_elbow_b, ja_ball_joint_elbow_c, ja_ball_joint_elbow_d, jv_ball_joint_elbow_a, jv_ball_joint_elbow_b, jv_ball_joint_elbow_c, jv_ball_joint_elbow_d, ja_hinge_joint_wrist, jv_hinge_joint_wrist, ja_hinge_joint_fingers (0-2, averaged), jv_hinge_joint_fingers (0-2, averaged)
Group 3 - Joint Space Torques (Forces)	jt_hinge_joint_scapula, jt_slide_joint_scapula2 jt_ball_joint_shoulder_pitch, jt_ball_joint_shoulder_roll, jt_ball_joint_shoulder_yaw, jt_ball_joint_elbow_pitch, jt_ball_joint_elbow_roll, jt_ball_joint_elbow_yaw, jt_hinge_joint_wrist, jt_hinge_joint_fingers (0-2, averaged)

Continued on next page

Group	Components
Group 4 - Muscle Space Forces	ma_abductor_digiti_quinti, ma_abductor_pollicis, ma_acromiotrapezius, ma_biceps_brachii_long, ma_brachialis, ma_clavotrapezius, ma_coracobrachialis, ma_ex_carpi_radialis_brevis, ma_ex_carpi_radialis_longus, ma_ex_carpi_ulnaris, ma_ex_digiti_quarti, ma_ex_digiti_quinti, ma_ex_digitorum_communis, ma_ex_indicis_proprius, ma_ex_pollicis_brevis, ma_ex_pollicis_longus, ma_fx_carpi_radialis, ma_fx_carpi_ulnaris, ma_fx_digiti_quinti_brevis, ma_fx_digitorum_profundis, ma_fx_digitorum_superficialis, ma_infraspinitis, ma_interossei_dorsal_1, ma_interossei_dorsal_2, ma_interossei_dorsal_3, ma_interossei_volar_1, ma_interossei_volar_2, ma_interossei_volar_3, ma_latissimus_dorsi, ma_lumbricales, ma_opponens_digiti_quinti, ma_palmeris_longus, ma_pectoral_major_deep, ma_pectoral_major_superficial, ma_pronator_teres, ma_rhomboideus_cervicis, ma_rhomboideus_thoracis, ma_serratus_ventralis1, ma_serratus_ventralis2, ma_serratus_ventralis3, ma_spinodeltoid, ma_spinotrapezius, ma_subscapularis, ma_supinator, ma_supraspinatus, ma_teres_major, ma_teres_minor, ma_triceps_brachii_lateral, ma_triceps_brachii_long, ma_triceps_brachii_medial
Group 5 - Muscle Space (length and velocity)	lM (50 muscles, as listed above, per muscle length), vM (50 muscles, as listed above, per fibre contraction velocity),
Group 6 - Proprioceptive Muscle Spindles & Golgi Tendon Organs (fIb)	fII (50 muscles, as listed above, per muscle), fIa (50 muscles, as listed above, per muscle), fIb (50 muscles, as listed above, per muscle)

Neural population analysis with CEBRA

To measure neural representations we used CEBRA, a new self-supervised learning algorithm for obtaining interpretable, **C**onsistent **E**m**B**eddings of high-dimensional **R**ecordings using **A**uxiliary variables to test the relationship of behavioral, learning, and task variables to the neural dynamics. In brief, CEBRA contrasts a reference example $\mathbf{x}$ to a given positive example ($\mathbf{y}_+$), and multiple negative examples ($\mathbf{y}_i$) and minimizes the loss function:

$$\mathcal{L}_B(\psi) = \underset{\substack{\mathbf{x}\sim p(\mathbf{x}),\ \mathbf{y}_+\sim p(\mathbf{y}|\mathbf{x}) \\ \mathbf{y}_1,\ldots,\mathbf{y}_n\sim q(\mathbf{y}|\mathbf{x})}}{\mathbb{E}} \left[-\psi(\mathbf{x},\mathbf{y}_+) + \log \sum_{i=1}^{B} e^{\psi(\mathbf{x},\mathbf{y}_i)} \right]. \tag{42}$$

for a positive sample distribution $p(\cdot|\cdot)$ and a negative sample distribution $q(\cdot|\cdot)$. To measure how well the CEBRA model is fit compared to a null model, we define the goodness of fit (GoF; in bits) as

$$\mathrm{GoF}(\psi) = \frac{1}{\log 2}(\log B - \mathcal{L}_B(\psi)), \tag{43}$$

where B is the batch size and $\mathcal{L}$ the InfoNCE loss of a converged CEBRA model ψ. A goodness of fit of 0 bits denotes chance level, higher goodness of fit values are better.

For baseline sessions, we fit five 6D CEBRA-Behavior embedding per group using the same cross-validation scheme described for the GLMs. We select all auxiliary variables from the GLM analysis (Figure 4) that showed more than 3% tuning. Specially, we used the 'offset10-model' model, batchsize = 1024, learning rate = 1e-4, temperature = 0.5, output_dim = 6, hidden_dim = 128 and time_offsets = 10 and trained for 2,500 iterations. Neural data across all n=22 baseline sessions was aligned based on the chosen auxiliary variable to fit one embedding per group and dataset split. From the estimated embeddings, we fit linear regression models and estimate R^2 of regressing the different auxiliary variables. Note that for decoding analysis, the test set embedding is computed by first aligning neural data based on the labels (across sessions). The linear regression model is fitted on the train embedding, and evaluated on the test embedding. Hence, decoding analysis has the main purpose of comparing different sensory spaces/groups on a relative scale, and is not an accurate estimate of purely held-out test accuracy.

For each adaptation session we fit an individual embedding using CEBRA-Time. Specially, we used the 'offset10-model-mse' model, batchsize = 512, learning rate = 5e-5, temperature = 0.01, output_dimension = 3, distance = euclidean, and time_offsets = 10. We set the maximum iterations based on the convergence of the loss function (range: 200-3000).

We trained separate embeddings on 50-trial sections of the data, from baseline (trials 25-75), to early perturbation (trials 76-125), late perturbation (126-175), and washout (trials 176-225) using a continuous auxiliary variable around the trial TTL (red line in Figure 3a), which is also the same position bin as the force-field onset time (also shown as a red line in Figure 5a) to compare the consistency across learning. This continuous variable is computed by assigning values from -100 to 500 samples around trial TTL to the sequence. Neural data from multiple sessions is aligned on the fly based on the trial TTL to learn a single embedding across all n=4 adaptation sessions. We used the 'offset10-model' model, batchsize = 512, temperature = 0.5, learning rate = 1e-4, time_offset = 10, hidden_dim = 128, output_dim = 10 and trained for 2,500 iterations. We used either all neurons or split the training into M1 and S1 models. To measure changes in the latent space we used the consistency metric described in Schneider et al. (89). To estimate consistency, all embedding points were binned by the trial TTL auxiliary variable, now only considering -1 and 1 sec around the trial time TTL (i.e., force-field onset). We compute the mean embedding per bin, and estimate the consistency metric by fitting a linear regression model and averaging R^2 scores for forward and backward prediction between the embeddings across the four phases.

Other data analysis methods

Identifying pulls and reaches (for muscle correlations)

Identifying stereotyped reaching and pulling movements in the experimental data was performed in two stages, first, determining candidate reach and pull times in the experimental data, and secondly, filtering the candidate reach and pull movements for stereotyped kinematics. To determine candidate reach and pull times, we first isolated the y-axis movements of the mouse hand and joystick, which is the principal axis for reaching and pulling movements. For reaching movements, the mouse right hand DeepLabCut marker was used to identify reach times, and for pulling movements, the joystick marker was used to identify pull times. Next we defined threshold-lines along the y-axis, which, when crossed, suggest a reach from the platform to the joystick or a pull of the joystick has occurred. For reaches, this was done by performing binary classification of the y-axis data as above or below threshold, calculating the derivative to identify points where the threshold was crossed, filtering for movements from the platform to the joystick, and then a secondary filtering removing any partial reaches where the mouse returns to the platform within 300 milliseconds. For reaches, we use -0.75 centimeters along the y-axis as the threshold boundary point to determine candidate moments for reach times. We then require that in the 150 milliseconds before the reach, the mouse and model hand are under the -0.5 centimeters line on the y-axis, and that within 150 milliseconds after the reach time they cross beyond -1.8 centimeters.

The pull times we defined by a TTL pulse that was defined at a positional bin (as shown in Figure 3a, b). With the reach and pull times, we analyzed a window from 150 milliseconds before and after the event time. We checked to make sure that for the full 150 milliseconds before the reach or pull time that both the mouse and model hand position was inside the threshold area, and that at some point between the reach or pull time and 150 milliseconds after the mouse and model hand are beyond a secondary threshold line representing a successful pull or reach. When analyzing the pull movements, we also require that the joystick marker is moving with the hand, to ensure that we are isolating actual pulls of the joystick and not including any movements where the hand is simply returning to the platform. For reaches, we also require the height of the hand during the

reach to be below a threshold level, to better match with the data from Miri et al. (55), where the reach target is set up lower than the joystick used in our experiments.

For pulls, we require that all joystick and mouse and model hand positions are beyond -1.85 centimeters in the 150 milliseconds before the pull time, and that within 150 milliseconds after the pull time they cross the -1.5 centimeters line along the y-axis.

EMG analysis and cross-correlation calculations

To generate muscle EMG signals, we post-processed the muscle activation signals generated by our model following the steps described in (55). Specifically, we downsampled the signals to 1 kHz, high-pass filtered at 40 Hz, rectified, and convolved with a Gaussian having a 10 ms standard deviation. Using the reach and pull times identified above, we then generated the cross-correlation matrix using data from 50 milliseconds before the trial time to 100 milliseconds after using the NumPy corrcoef function.

Peri-stimulus time histograms

To compute PSTH plots we used the extracted NMF signal then extract a window before and after a defined TTL that is defined by a positional bin during the pull. As shown in Figure 3a, b, this position is approximately peak velocity of pulls, and as shown in Figure 5a the time the force-field comes on during perturbation trials only. To compare across epochs, as in Figure 5, we normalized the activity by 10 timesteps within the pre-TTL period.

PCA and hierarchical clustering

For functional clustering across trials in Figure 5f, g, we constructed a vector of neuron x trials across the epochs. Namely, we take the data as shown in Figure 5e, which is ± 1 sec before the trial TTL (red line as shown in Figure 3a), which is the point of force-field onset during the adaptation task. We normalized activity within the epoch to account for any systematic changes across the session. Then we take the n=1697 M1 an S1 neurons and perform ran PCA with $n = 20$ components using scikit-learn. Then, we used Agglomerative Clustering algorithm to perform hierarchical clustering. The number of clusters was optimized for non-overlapping functional groupings, and set to $n = 5$. To compute the dendrogram we use the linkage function from sci-kit learn with the 'ward' method.

Statistical testing and software

All statistical tests were computed in Python 3.8 (or above) with scipy and tests were two-sided. Alpha was preset to 0.05. We used Python, MuJoCo (47), NumPy (131), SciPy (132), Pytorch (96), TensorFlow (133), Scikit-learn (134), DataJoint (98), DeepLabCut (49, 65), and CEBRA (89).

Supplemental Materials

Video Captions

VIDEO 1: Video showing the 3D lifted DeepLabCut outputs and input video.

VIDEO 2: Video demonstrating MausSpaun reenacting the main mouse behavior: reach, pull, and return hand to base.

VIDEO 3: Real-time recording of the MausSpaun model in the Nengo interface, showing the spiking neural networks.

VIDEO 4: Overview of data from an adaptation session with CEBRA embeddings from S1 and M1. Top left: we show a 2D video with DeepLabCut overlay. Bottom left: the joystick movements (top view). Middle: co-recorded M1 and S1 neurons for an example session, and below are joystick and extracted hand coordinates. Right top: S1 embedding, Right bottom: M1 embedding. Epoch as shown in video title.

Supplemental Figures

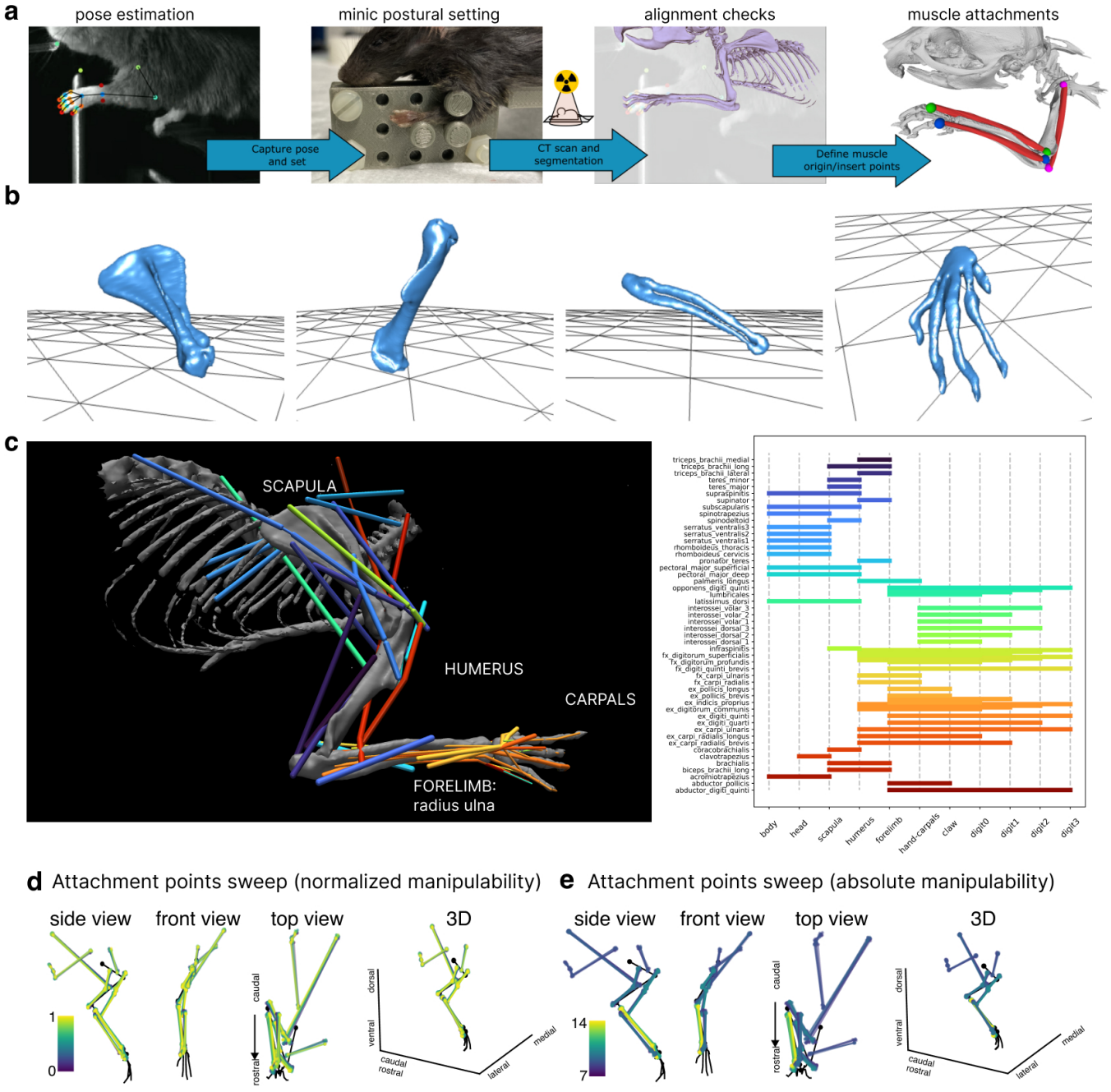

Figure S1. Development and optimization of the adult mouse forelimb model. **a**, overview of posing mice in experimentally realistic postures for CT scanning, with post-hoc alignment and muscle attachment configuration. **b**, Example segmented forelimb showing the scapula, humerus, forelimb (ulna and radius) and hand (here as a "single" image). **c**, Overview of the major groupings (left) attachment points of the 50 muscles modeled (right). **d**, Sample attachment point configurations tested during in the hyperparameter sweep for several muscles. Each muscle's color represents the resulting manipulability score calculated using Eq. (13), normalized per muscle. Within each set of configurations, small variations can lead to large changes in manipulability. **e**, Sample attachment point configurations tested during in the hyperparameter sweep for several muscles. Each muscle's color represents the resulting absolute manipulability score. This plot highlights the relative contributions of each of the muscles to the arm's overall manipulability, calculated using Eq. (13). More proximal muscles, such as the latissimus dorsi or serratus ventralis have relatively low manipulability scores (see Methods for a discussion on the metric) for all configurations, 7.94 and 8.96, respectively. The extensor carpi ulnaris, which spans the elbow and the wrist joints, has several attachment point configurations with relatively high manipulability scores, 14.66. The high manipulability scores for some attachment point sets for extensor carpi ulnaris indicates that its exact placement is the most important of this set of muscles to be able to manipulate the arm as effectively as possibly. The initial attachment point set (placed by hand) achieved a manipulability score of 8.09. Applying the best attachment point sets for each muscle found with a hyperparameter search inside a 0.01^3 mm box around each initial location achieved a manipulability score of 61.52. Applying the best attachment point sets for each muscle found with a hyperparameter search inside a 0.03^3 mm box around each initial location achieved a manipulability score of 4113.06, three orders of magnitude larger than the initial placements.

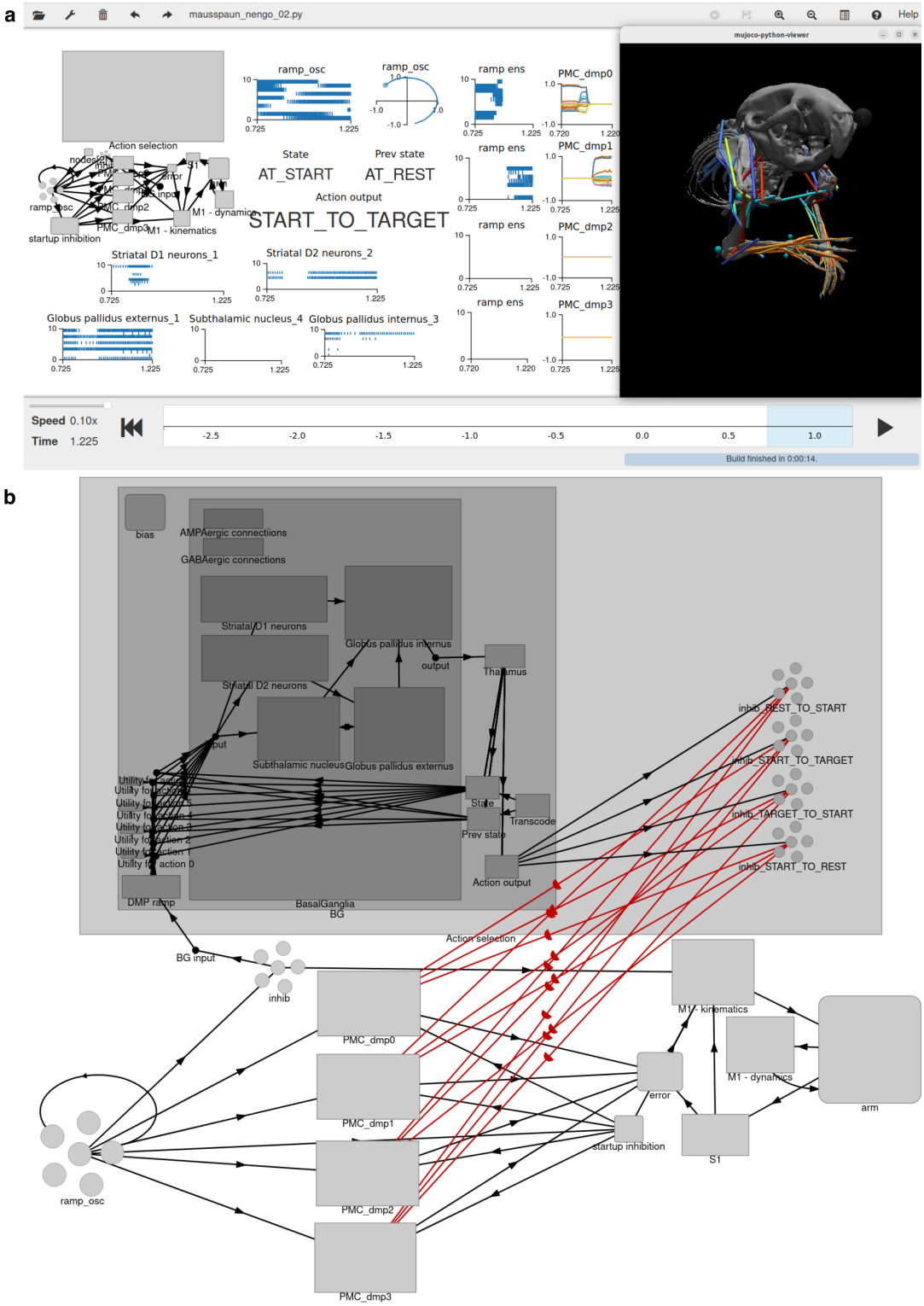

Figure S2. MausSpaun software. **a**, A visualization of the MausSpaun Nengo implementation in the Nengo Graphical User Interface (GUI). The network layout (simplified) is shown in the top right. Raster plots of the spiking neuron activities and line plots of the population level vector representations show network information as the simulation runs. The three word plots in the center show the semantic pointers of the three working memories, storing current and previous state, and the last selected action. The MuJoCo mouse arm model is shown on the right (in a separate window). based control of the mouse arm, demonstrating the spiking neural network sub-components (see Video 3). **b**, A more detailed network diagram of the Nengo implementation of MausSpaun shown in the top-left inset in panel **a**, highlighting the brain-inspired, theory-guided design of the system.

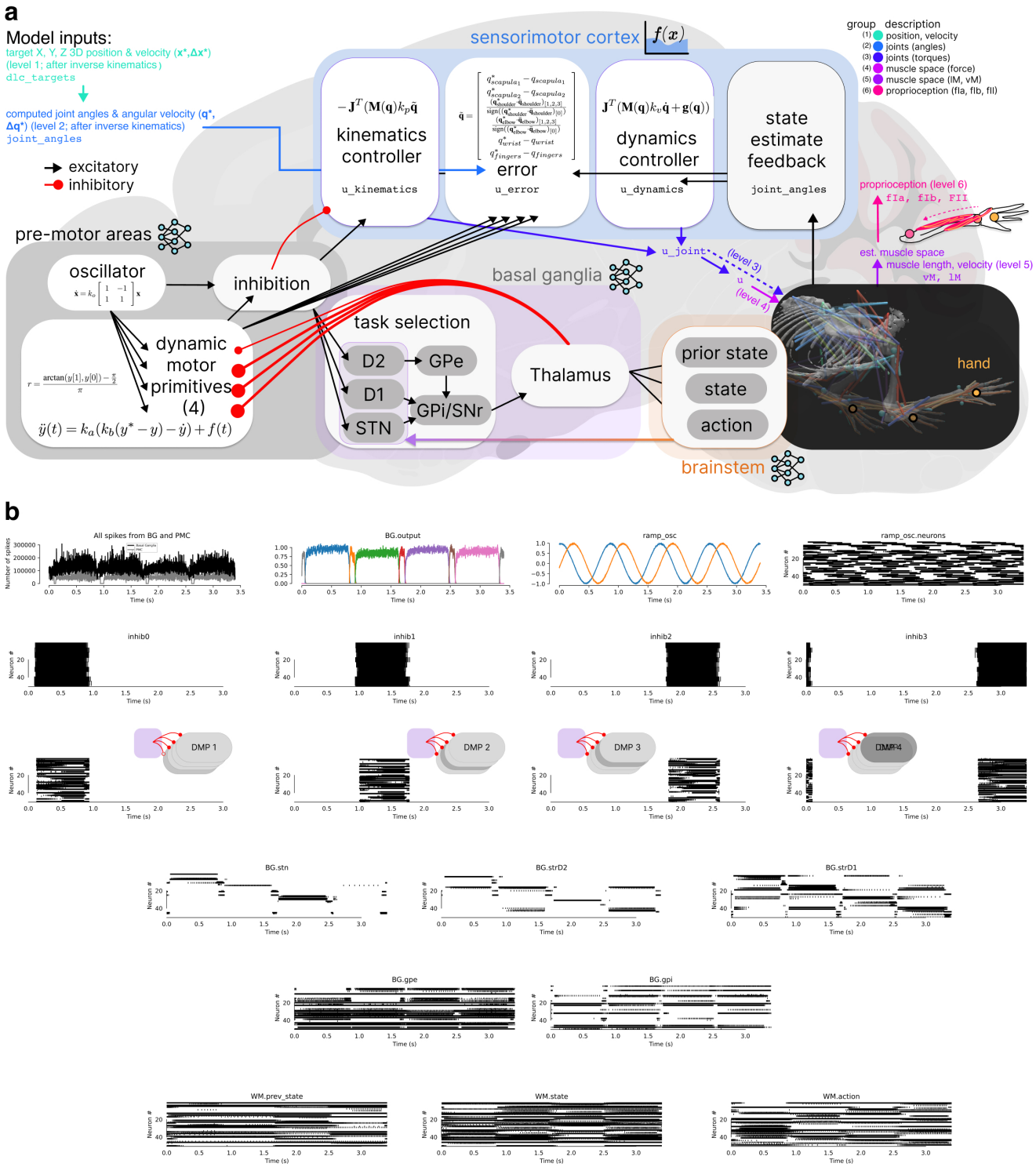

Figure S3. MausSpaun Model & Outputs. **a**, Overview of the modules within MausSpaun, as shown in Figure 2a, with module-specific equations that it implements. Additionally, we provide the code variable names and groups as described in the inset (top right) and in relation to Figure 4. **b**, Example outputs during a reach, grasp and pulling movement from each spiking node of MausSpaun, as labeled. Note, the number of neurons and the spiking rate is a changeable parameter, such that more neurons require less spikes-per-neuron (thus can be within realistic bounds) The example SNN outputs from the "pre-motor areas", "basal ganglia", and "brainstem" are shown from the same time period shown in Figure 2**b**. The activity of the pre-motor areas shows the clear delineation points between the execution of different trajectory segments, as only one dynamic motor primitive network is active at a time, while the others are actively suppressed by the thalamus. The implementation shown here contains a total of 14,444 neurons.

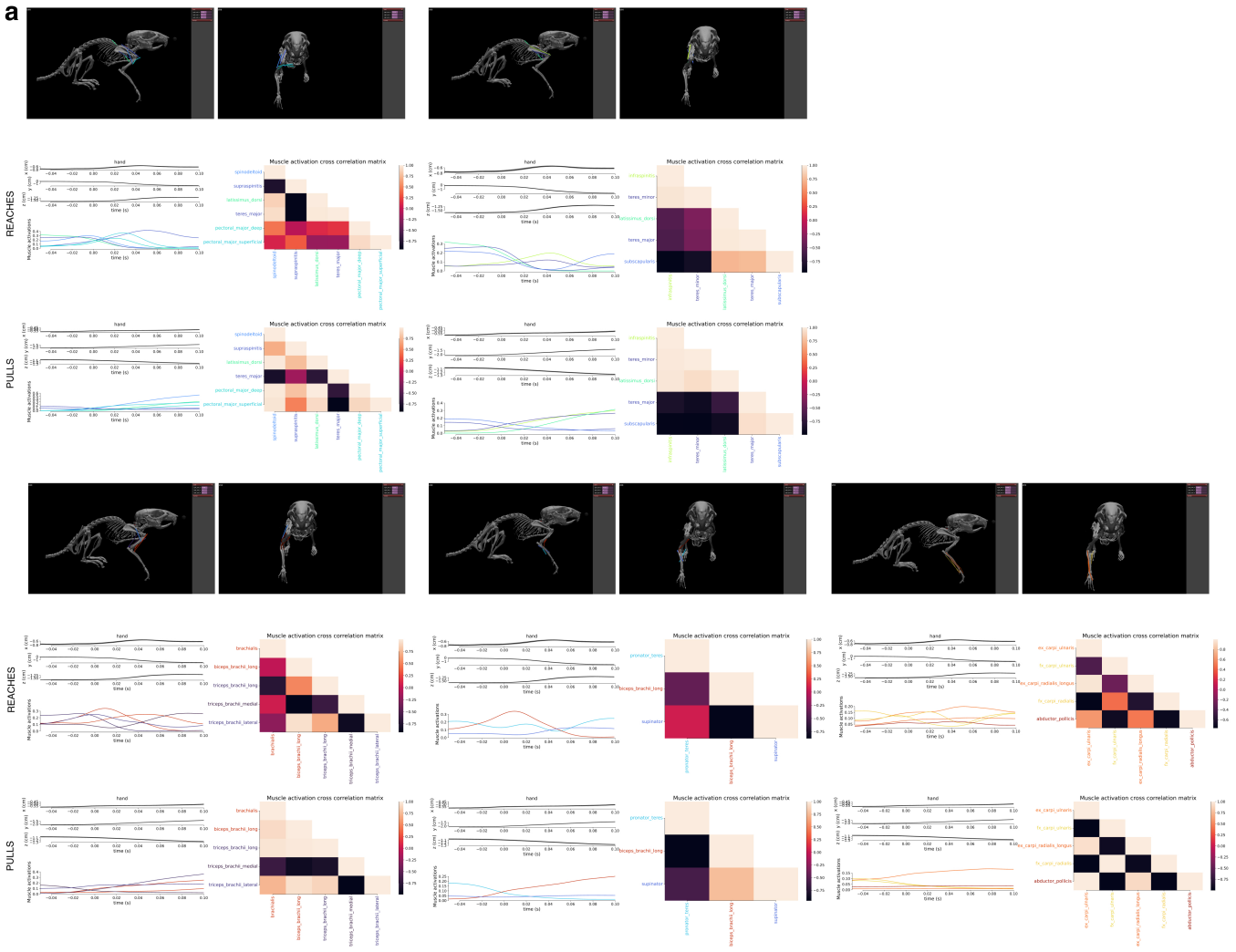

Figure S4. **MausSpaun muscle outputs**. **a**, We measured the correlation of muscles during pull (n=365) and reach (n=60) epochs and then computed the average correlation over the time-window shown. Within each panel the hand x, y, and z mean position and 95% confidence intervals are shown on the top left, bottom left is the mean muscle activations (see Methods), and right is the correlation matrix.

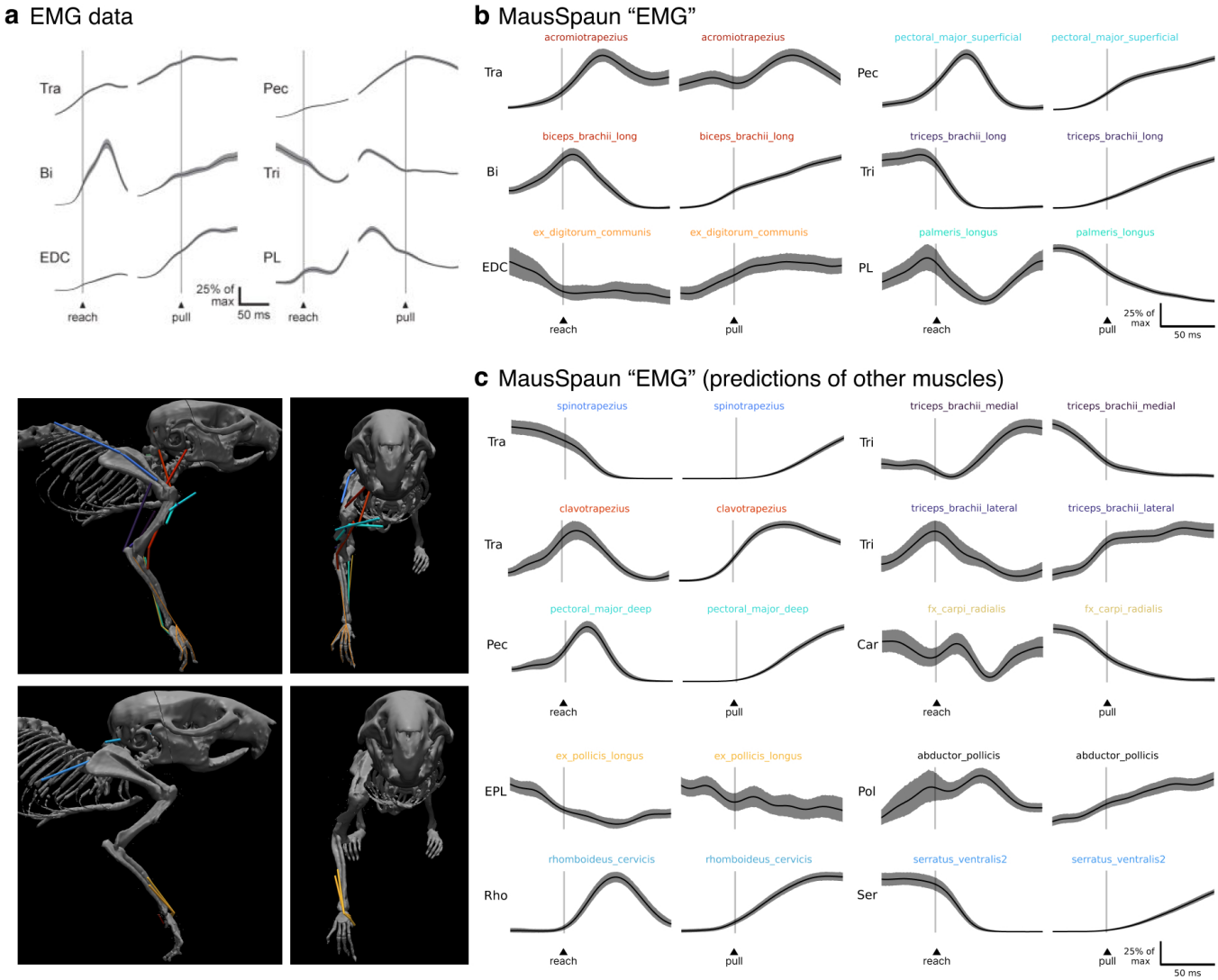

Figure S5. EMG comparisons. **a**, Adapted from Miri et al. (55), where head-fixed mice are reaching and then pulling on a lever. They recorded directly from six muscles: Tra = Trapezius, Tri = triceps, Bi = Biceps, EDC = extensor digitorum communis, and PL = palmeris longus. Note, the kinematics are not identical to our synthetic mouse, but we show in **b** that our general extracted EMG-like signals during "reaching" and "pulling" are well conserved in the 6 muscles they recorded from. We calculated the mean and 95% confidence intervals of these EMG-like signals using the muscle activations during pull (n=365) and reach (n=60) epochs. Note, in **c** we highlight we can make other predictions (up to 50 in total), and some are also shown in Figure 2d. **c** shows an additional subset of muscles related to the upper arm, i.e., additional muscles of the Triceps (Tri), Pectoral muscles (Pec) and Trapezius (Tra)); the forearm, i.e., ex. pollicis longus (Pol), abductor pollicis (Pol); and shoulder, i.e., rhomboideus cervicis (Cer) and serratus ventralis(2, Ven).

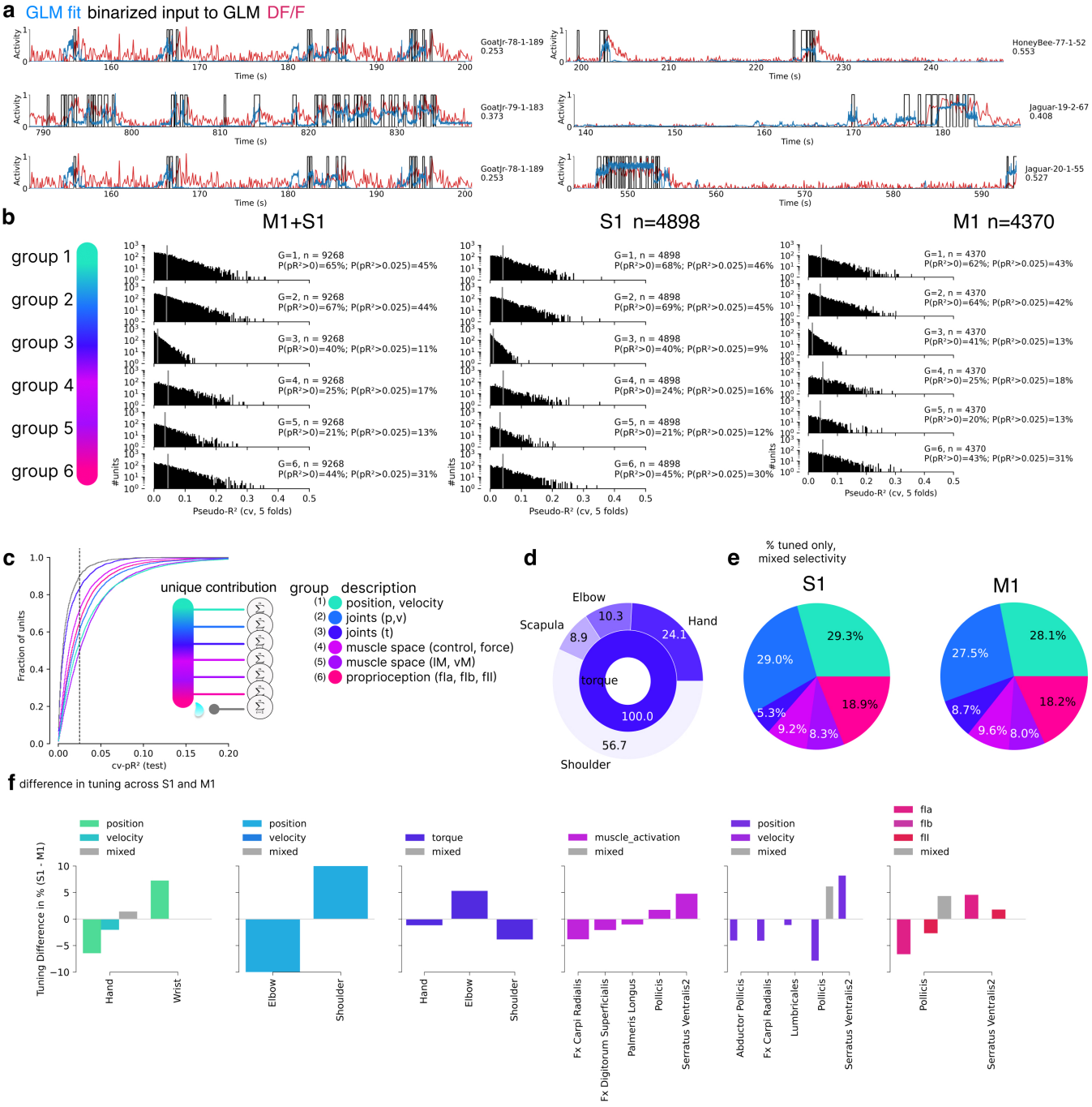

Figure S6. GLM models. **a**, Example GLM model fits of individual neurons on the held-out test dataset. These are selected from the best-fit neurons. pseudo-R^2 is reported on the far right, and each signal is min-max normalized. **b**, Full GLM run results, showing the distribution of all neurons (n) per group (L). The percentage of neurons above a given pseudo-R^2 threshold are inset. Middle and right plots show results for M1 and S1 independently. **c**, GLM results where each neuron is regressed independently against all regressors (see Table 6) then post-hoc placed into a given group. This controls for group-effects. **d**, Pie charts showing the tuning of neurons at each group; group 1, 2, 5, and 6 are also shown in Figure 3, but here again for reference across groups. Only those regressors with greater than 3% are shown in the pie chart for clarity. **e**, Pie chart of the tuned neurons (only), for the percentage of neurons with a $cv - pR^2$ above 0.025 for each group (thus, a neuron can be counted in more than one category). **f**, Differences in encoding across M1 and S1 during baseline sessions. Note, changes less than 1% are not shown in bar plots.

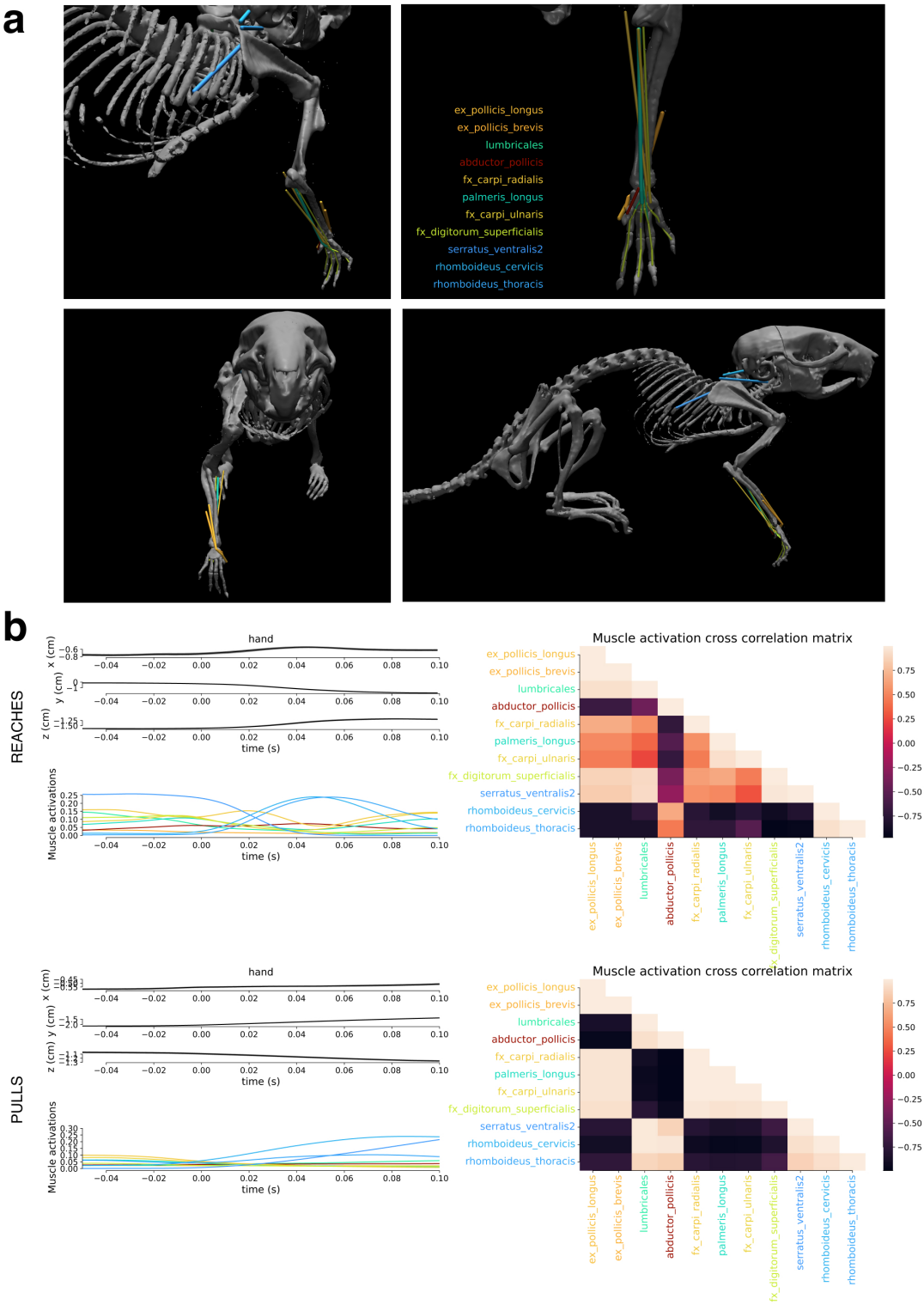

Figure S7. Muscles most tuned to neurons in M1/S1 based on GLMs. **a**, Images of the muscles that were most tuned from the GLM analysis. **b**, We measured the correlation of muscles during pull (n=365) and reach (n=60) epochs and then computed the average correlation over the time-window shown. Within each panel the hand x, y, and z mean position and 95% confidence intervals are shown on the top left, bottom left is the mean muscle activations (see Methods), and right is the correlation matrix.

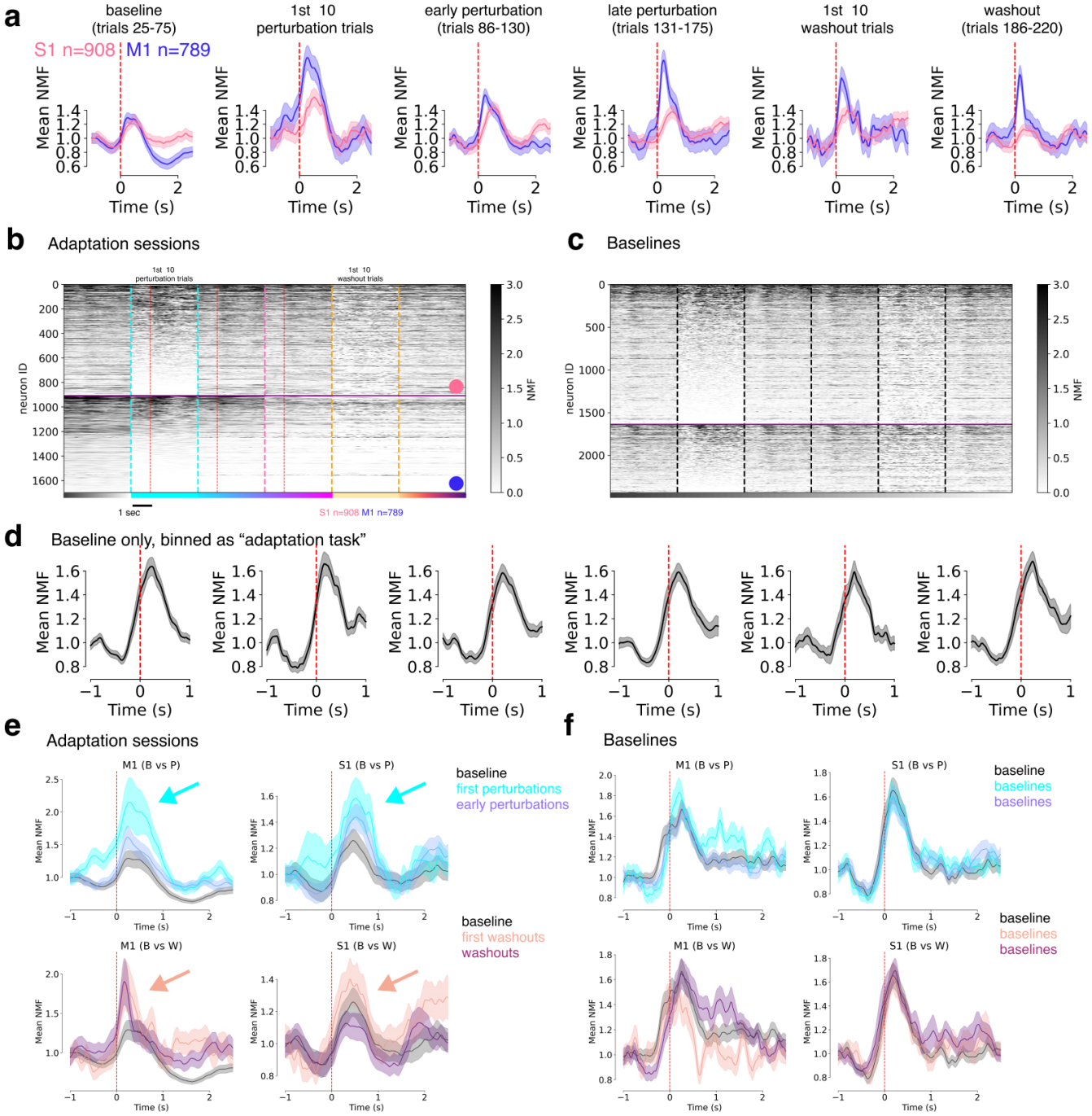

Figure S8. Neural properties during adaptation. **a**, PSTH's computed based on force-field onset time (delineated by the red line): -1 sec to 2 sec. Note, the force-field was only applied in perturbation trials. All recorded M1 and S1 neurons are used, and trials included are denoted in the plot title. **b**, Heatmap of all neurons recorded during the adaptation task, rank order by activity of neurons during early perturbation (as shown in Figure 5d). **c**, Same as in **b**, yet for baseline only sessions (n=11 sessions with >250 trials for comparison) as a control. Neurons did not show a change across these artificial blocks. **d**, PSTH plots of baseline only sessions (n=11 with >250 trials), yet binned as in panel **b** & **c** as a control. There was no significant change from baseline in any binned block (t-test p-values, baseline vs. other blocks left to right: $0.55, 0.51, 0.63, 0.26, 0.34$). **e**, As plot in Figure 5f, showing the difference in activity in M1 or S1 neurons during baseline, early perturbations, mid-perturbations, early washout and washout (as colored in the legend). **f**, Same as in f yet for only baseline sessions as a control, again showing no significant difference.

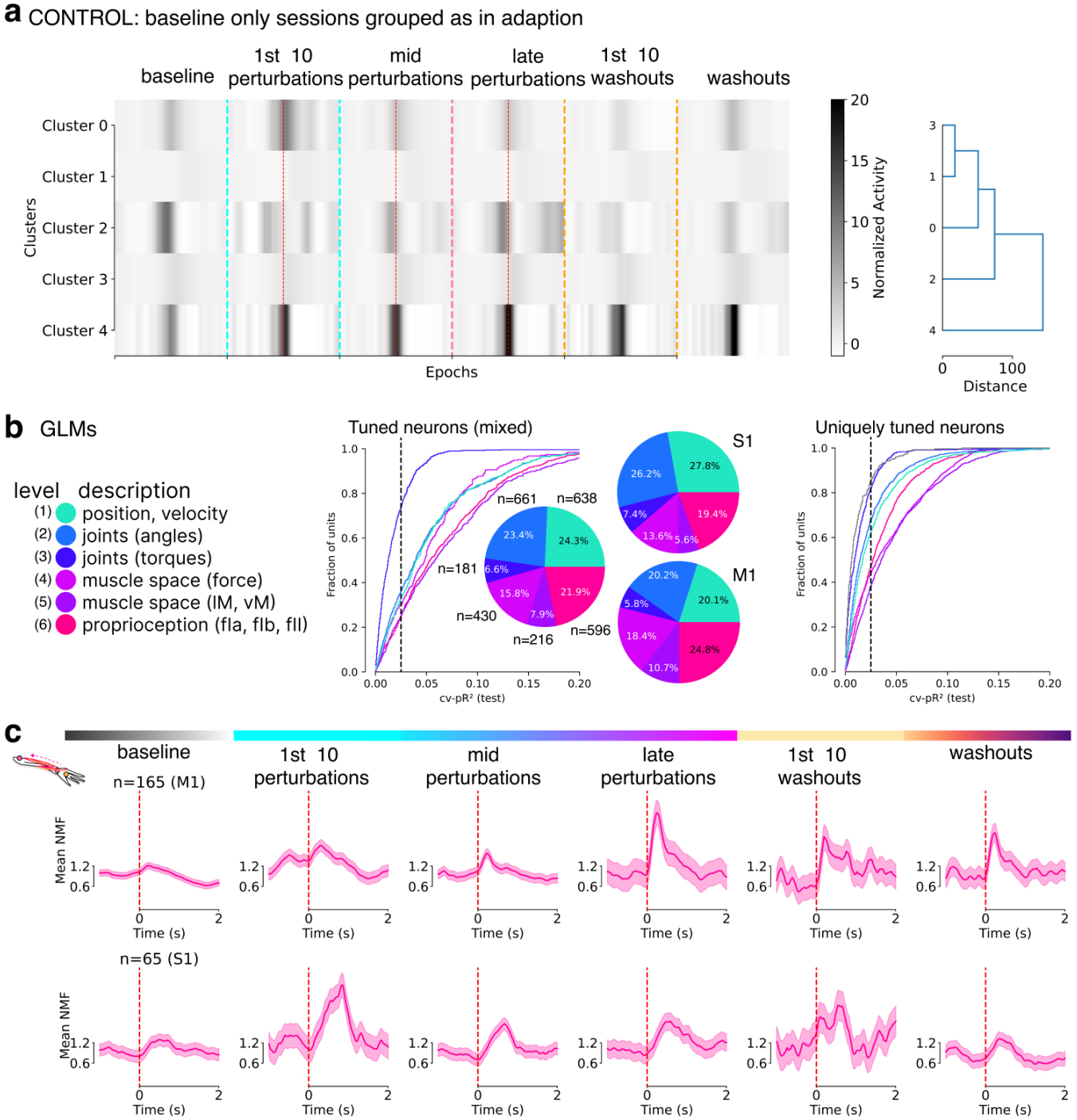

Figure S9. Neural tuning properties. **a**, Baseline-only sessions were used as a control for the clustering. Namely, there was no force-field perturbation, but we used the same PCA and hierarchical clustering as in Figure 5g. Here, unlike in the adaptation task, we do not see specific clusters that map to computations like prediction errors. **b**, GLM fits of the neurons recorded during the adaptation task sessions. Pie charts show the distribution of group-tuning (only considering the significantly tuned ($cv-pR^2 > 0.025$) neurons). Far right shows the unique contribution – i.e., winner takes all for each group, as computed in Suppl. Figure S6c for baseline sessions. **c**, All neurons significantly tuned for group 6 are used to compute PSTHs in each epoch. M1 n= 165, S1 n=65 neurons; only S1 shows the sensorimotor prediction error.

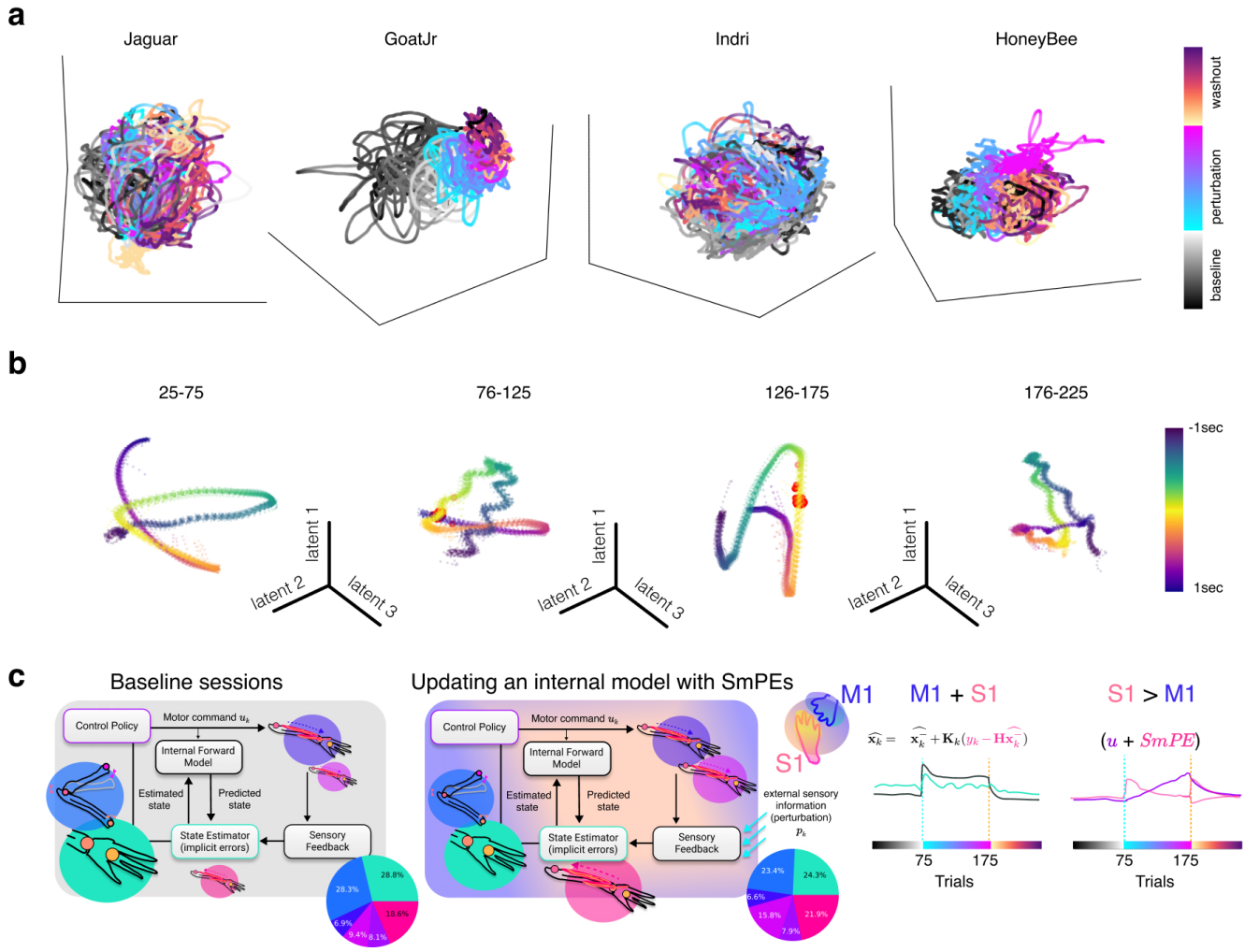

Figure S10. Adaptation CEBRA embeddings. **a**, Individual sessions were fit with CEBRA-Time. Specially, we used the 'offset10-model-mse' model, batchsize = 512, learning rate = 5e-5, temperature = 0.01, output_dimension = 3, distance = euclidean, and time_offsets = 10. It is colored by the time within each block. **b**, Using trial-time, we fit CEBRA-Behavior models from 50-trial blocks from baseline, early perturbation, late perturbation, and washout. Shown are the embeddings from 6D model (3 randomly selected are shown), which were used to make quantitative comparisons in Figure 5i. **c**, Summary diagram highlighting that in baseline we see minimal changes in tuning, while in learning specific features are expanded and related computations map to specific areas and functional neuron types. SmPE is sensorimotor prediction error. The size of the icon is reflective of the change in representation across baseline to adaptation. Right of the internal model updating scheme, we provide the intuition about which signals are in M1 and S1. Namely, groups 1, 2 and non-significantly tuned neurons, across M1 and S1 collectively show state estimation. S1 most strongly encode sensorimotor prediction errors, and a smaller fraction of command-like neurons were also primarily in S1. To note, this means that the perturbation trials are going to most-different to baseline in the S1 neurons vs. M1, which is what we observe with our latent dynamics analysis (see Figure 6d, e).