A spiking neural model of adaptive arm control - Supplementary material

S1 Past work

There are several previous models of the motor control system that overlap with the work presented here. For instance, [30] presents a model based on an attractor network implementation that captures the stereotyped sudden transitions (STs) in the dorsal pre-motor cortex (PMd) predicting forthcoming actions on a single-trial basis. These researchers suggest that there is a cascade of STs throughout the areas of the brain involved in motor control starting with those areas that plan the movement (PMd) and down to areas executing the movement, i.e. the primary motor cortex. While this model captures some of the neurophysiological phenomena of the motor system, it offers little in the way of mechanistic explanation of the function of these areas. It also does not demonstrate complex paths, or adaptation to online changes in dynamics or kinematics of the arm.

In [27], the authors present a model able to achieve repeatable activation patterns by training networks of rate neurons to converge to a specific path through the state space. The authors suggest that a similar training mechanism and network could serve as a basis for trajectory generation in the brain, by decoding the neural activity at different parts of the path into a sequence of target positions. While this model is able to perform some of the functionality required by trajectory planning areas, it was implemented in rate neurons as opposed to spiking neurons (the authors noted explicitly that how to implement their network in spiking neurons was unclear), and lacks the generalizability of a trajectory generation system based on dynamic movement primitives. As well, it does not control a nonlinear 3-link arm, or demonstrate online dynamic or kinematic adaptation.

Recently, in [31], the authors implemented a spiking neuron system that performs the transformation between task-space and low-level control signals, similar to the primary motor cortex portion of our model. The focus of that work was the implementation on neuro-morphic hardware and the controlling of a physical robotic arm, so no comparisons were made to the motor control system of the brain. We have gone beyond this work by implementing a secondary controller that operates in the null space of the first [26, 45], applying nonlinear adaptive control methods to account for unmodelled dynamics and changes to the kinematics online, and providing a comparison to cortical activity by analyzing spiking data.

Similarly, in [11] the authors explore the control of a 2 DOF muskuloskeletal model using spiking neurons, that is linked to a robotic arm. The model focuses on employing reinforcement learning in the network, which consists several hundred neurons divided into three layers. They demonstrate an improvement in the smoothness of the movement after learning, compared to a simpler arm model. Unfortunately, the neural activity and behavioural profile of this model is not compared to empirical studies. In contrast, the REACH model is mapped in greater anatomical detail, incorporates rapid dynamics learning, includes complex path following, and demonstrates a good match to behavioural and physiological data collected from a variety of empirical experiments.

Motivated by work presented in [38], the authors of [37] present a behavioral model of human adaptation in the context of reaching under force fields. The authors show that the nonlinear adaptation methods of [36] give rise to the same trajectory profiles post-learning as seen in humans, as well as similar after-effects when the force field is removed. The scope of this model is narrow, however, focusing on a behavioral comparison of adaptation only. No other behavioral measures, such as velocity profiles or total squared jerk, are used, and no mapping to biological substrates or discussion of neural implementation are provided.

The model presented in [40] is a partially-neural model that encompasses the motor cortex and cerebellum. The authors employ internal models of the dynamics of the system, learned in an anatomically grounded cerebellar circuit during fast movements to allow the arm to move more precisely during ballistic movements. The authors acknowledge several limitations of their model, however, including the lack of a biologically plausible learning rule, lack of ability to model non-linear

functions effectively, and the use of a simple two-link planar arm with no redundant configuration space. Additionally, only the adaptive portion of the cerebellum is modeled with neurons, without any discussion on how the rest of the system functionality could be implemented in a neural network. These limitations do not apply to REACH.

There are also a number of models that focus less on behavioral data, and instead attempt to capture specific neural observations. Many of the computational models of the cerebellum fall into this category. They address the cerebellum's role in activities ranging from supervised learning [24,32] to ballistic movement control [8,20] to locomotive and balance control [34,48] suggesting a variety of neural mechanisms that would allow these structures to perform different functions. In the REACH model the scope of the cerebellum model is limited to supervised learning and ballistic movement control.

The primary motor cortex model presented in [51] demonstrated how output from the motor cortex in the terms of activation of muscles can account for the high-level movement parameters that have been previously correlated with neural activity during movement recorded in experiments. This model, however, only shows how these correlations could be explained given a system that outputs muscle activation signals, and does not explain how a behaviorally effective controller could be implemented in neural networks.

There is also experimental research [6,7], which has examined the activity of neurons in motor cortex and argued for the existence of an underlying dynamical system responsible for the activity and correlations seen. The methods in this and related work are largely developed and used for the purposes of data analysis rather than hypothesizing mechanisms that give rise to the data and resulting behaviors.

The MOSAIC model and its extensions focus primarily on modelling the cerebellum [57]. The basic idea of a MOSAIC system is to exploit a mixture-of-experts structure to predict the forward dynamics of the system. Since its original presentation, the MOSAIC model has spawned a number of extensions, including hierarchical MOSAIC [21], MOSAIC-MR [50] for multiple reward scenarios, MACOP [55] which looks at having multiple controllers for each part of state space where each controller outputs control signals for different primitive actions and learns how to weight these different actions such that the overall desired trajectory is carried out. As well, MOSAIC was extended to produce the model MODEM [2], which reframed the idea of coupled forward and inverse models in the context of a modularized hierarchical system structure.

Neuroanatomically, the original MOSAIC model proposed that the forward and inverse models match well to the structure and assumed functionality of the cerebellum. Further models have extend this mapping. In MODEM, a sensorimotor hierarchy is divided into a low and high level, with a 'gate selector' component that chooses among the high level features to carry out a given task. The high level is proposed to be a basic model of the motor cortex, and the function of the gate selector is proposed as an appropriate role for the basal ganglia. All of these models are strictly behavioral, however, with no consideration of a neural implementation.

In [42], a framing of the motor control system in terms of optimal feedback control theory is presented. There are two main ideas exploited in optimal feedback control theory. The first is the joining of trajectory planning and execution, in that the trajectory is updated as the system moves in order to take into account perturbations and obstacles. The second is the 'minimum intervention principle' which states that movement through the redundant solution space is not of concern, and only movement that affects task-space trajectories is corrected. The basic assignment of functions to anatomy in this proposed structure is that the primary motor cortex generates the optimal control signal, and the cerebellum is involved in online, feedback-based correction.

The main obstacle in using optimal feedback control theory as a model of the motor control system is that it is based on solving the Bellman equation, which is defined as the cost of moving from the current state to the target optimally. Solving the Bellman equation is a very difficult and computationally expensive task, and it is unclear how this problem could be solved by biologically plausible algorithms and processes analogous to those used in optimal feedback control [42].

Analytic solutions to the Bellman equation for nonlinear systems involves performing singular

value decomposition across a matrix representing all possible states weighted by the desirability of being in those states given the current task [52]. Other, possibly more biologically plausible approaches to solving the Bellman equation, such as the iLQG method [53], involve an iterative process using forward models, where an initial control policy is proposed and the predicted effects are simulated. The results of this simulation are then used to modify the control signal such that a more optimal outcome is achieved. This process is then repeated until an optimal control policy is converged upon given the current model of the system, and this process is repeated at the next time step, using the previously optimal control signal as a seed for the next round of iterations. This process is extremely computationally intensive, requiring many matrix inverses, for which it is not clear how to perform in a neural implementation.

In short, past work largely focuses on either behavioral or neural phenomena. None of this work has provided an in-depth, neural account of performing complex behavioral tasks such as controlling an arm through reaches and handwriting, scaling movements online, and adapting to external forces or correcting inaccurate internal system models using plausible neuron-level mechanisms.

S2 Predictions of different models

In this section we elaborate of the predictions made by the REACH model as compared to those discussed in the previous section. Specifically, we compare to two "compositional control" models: hierarchical MOSAIC (HMOSAIC) model [21], and an implementation of the optimal control feedback framework known as KL-control [10, 52]. We will examine the expected roles and activity of the different neural areas involved during the task of controlling the arm through a learned, complex trajectory.

In the compositional control model based on KL-control, a large set of learned movement patterns, or 'synergies', are stored and weighted as a set of basis functions for generating the desired movement. In the neural mapping of this control system described in [10], the motor cortex stores these synergies, with low-level muscle-space synergies stored in M1 and higher-level end-effector-space synergies stored in PMC. The basal ganglia compares the desired trajectory to the available synergies, and outputs a weighting over the learned movements. For the PMC, the desired trajectory input is a high-level abstract representation of the movement, and the weighted output from PMC is used as the desired trajectory input to M1.

In the HMOSAIC model, M1 contains a set of controllers that are trained for guiding the system through different movements under different conditions. At the higher levels of the motor cortex, weightings of the M1 controllers which act as prior probabilities for different tasks are learned and output to M1. In M1, the controller weightings appropriate to the current behavioral and dynamics context is learned, and the output is a control signal that moves the body.

In Figure S1 we show the different control structures of the REACH, HMOSAIC, and KL-control models, which forms the basis of our predictions for distinguishing between the different models. The distinctive features of each of these models that we focus on for our predictions include: 1) The use of generalizable DMPs in the REACH model for trajectory generation, 2) temporal motor synergies in the compositional control models, 3) the form of the high-level control signal, and 4) the means of dynamics compensation. We will now look at each of these in greater detail, along with the predictions they generate.

As discussed in the main paper, the REACH model makes use of DMPs for trajectory generation because they are highly flexible, robust, and generalizable. As a result, with a single DMP, a large number of kinematically distinct trajectories can be created. In the compositional control models, paths are created based explicitly on the desired kinematic trajectory; so writing the number '2' right-side up versus upside-down would require different components. In the REACH model, all that is required to draw an upside-down 2 is to change the end-target location of the trajectory that is passed in to the DMP; so while we expect a different trajectory to be represented by the neural activity in the PMC, we also expect that largely the same set of neurons will be active in both cases.

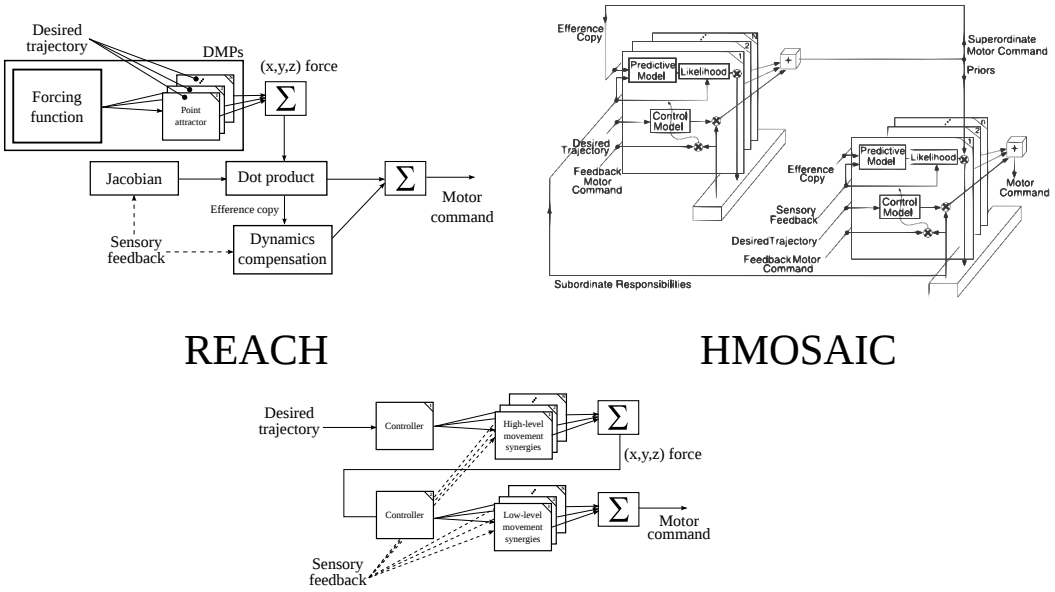

Fig S1. A diagram of the REACH, HMOSAIC, and KL-control models system flow. HMOSAIC diagram from [21].

In compositional models, however, we expect largely distinct sections of the PMC to be active for the generation of these two trajectories, because there will be no single underlying system capable of generating both of them.

In compositional models the output is generated by a weighting over a set of basis components, where the weights are provided by the basal ganglia. In the REACH model, however, the high-level signal is generated internal to the pre-motor cortex, not dependent on an external weighting. This means that when learning a new complex trajectory, we expect to see the effects of learning to be primarily in the basal ganglia in compositional models, and primarily in the pre-motor cortex in the REACH model.

In both the REACH model and the KL control model, the signal projected from the higher levels of the control system down to M1 represent the forces to apply to the degrees of freedom under control for the task. As a result, we predict that performing PCA analysis on the neural activity of the PMC during the generation of learned complex trajectories that involve different numbers of degrees of freedom (e.g. tracing a 2D path out along a tabletop and a 3D path around a sphere or a 1D path along an edge) will result in a different amount of variance explained by the primary principal component. We expect the amount of variance explained to increase as the DOFs of the movement decrease. Figure S2 shows a graph of the amount of variance that we expect to be explained for movements with different numbers of DOFs. The high-level signal in the HMOSAIC model, however, is not related to the number of DOFs of the movement, but is instead based on the number of low-level controllers that are involved in the movement. We expect instead that for well-learned movements the variance explained by the primary principal component will exhibit a much stronger correlation with the complexity of the movement, rather than the number of DOFs being controlled.

Finally, the component controllers in M1 in the HMOSAIC model are based on different system contexts; meaning that depending on the current dynamics of the system, different controllers will be

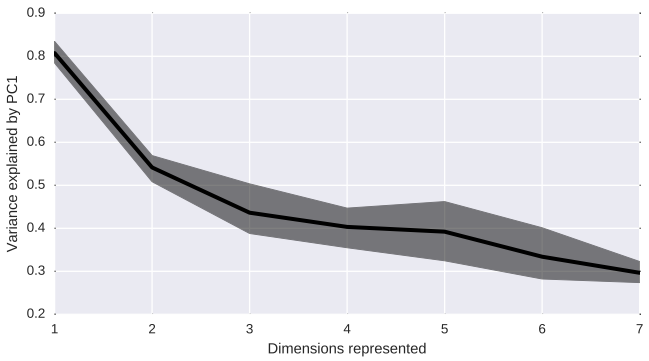

Fig S2. The variance explained by the primary principal component after performing PCA analysis on a neural populations representing vectors of different dimensions as they perform 50 trials moving to randomly chosen targets. The mean is the black line and the surrounding grey area shows the 95% confidence interval.

used. Because the higher levels in the HMOSAIC provide priors over the probability of the different controllers that will be selected at the low level, we also then expect a change in the neural activity of the higher levels when the dynamics of the system change. This is in sharp contrast to the REACH model, where the output of the PMC is related only to the kinematics of the movement, and the underlying system dynamics will have no effect on the neural activity patterns. Therefore, analyzing the neural activity during the generation of complex learned movements under different system dynamics should provide clear support for or against the described implementation of the REACH model. We expect that there should be relatively little change in the neural activity patterns of the PMC during trajectory generation under a change to the system dynamics, such as a weight added to the wrist, as compared to the HMOSAIC model, which predicts that changes to the system dynamics changes will directly affect the neural activity of the higher levels.

Each of these predictions is intended to distinguish between specific implementations of the REACH, HMOSAIC, and KL control models. It needs to be kept in mind, however, that ruling out a specific implementation by no means provides conclusive evidence for or against a particular control algorithm or system. For both the HMOSAIC and KL control models, numerous assumptions about a mapping onto the underlying neuroanatomy were made; both because they have not been explicitly specified previously and because they have not been implemented in spiking neurons. Additionally, there are a multitude of ways to implement a given control algorithm in neurons. Instead, the predictions and tests described above allow us to distinguish between the underlying mechanisms used in each of the implementations discussed, identifying features such as single generalizable systems for trajectory generation or the form of output from higher-levels of the control system. While this is less conclusive than identifying an overarching control algorithm of the motor system, it still provides valuable information about the types of mechanisms that could be used to implement the control of movement in the brain.

S3 Methods and materials

S3.1 Neural modeling

The spiking neuron implementation of this model is based on the Neural Engineering Framework (NEF; [12]). The NEF is a general method for implementing high-level, dynamic systems into spiking neuron models, by producing a connection weight matrix to perform a desired function. The first two (of three) principles of the NEF capture how a) groups of neurons form distributed representations of

vectors, and b) connections between groups of neurons can be used to specify a computation to be performed on those vectors. In essence, the NEF provides a method for analytically solving for the synaptic connection weights that will compute a given function in a spiking network. The details of the NEF methods are provided in several forms [12, 13, 49].

Using the NEF allows us to convert high-level computations written in terms of vectors and transformations of those vectors into spiking neuron models. However, these methods do not determine the topology of the network uniquely, and they do not determine the appropriate transformations for characterizing a specific behavior. In this model, we employ recent developments in control theory to suggest such transformations and provide a mapping to known anatomical structures as described in Section S3.4.

S3.2 Neural simulation

To implement and simulate the REACH model we used the Nengo [4] neural simulation package, which is freely available for academic use from https://nengo.ca. Nengo provides a Python scripting interface for creating neural networks using the methods of the NEF. Once the structure of the neural network is specified in a Python script the user is able to perform the simulation of the neural network on different hardware.

In all the models presented here, neurons in the cortical and cerebellar populations are generated with firing rates between 0 and 100Hz [41], and 10 and 200Hz [28], respectively, synapses are assumed to use the AMPA or GABA neurotransmitter with a synaptic decay time constant of 5ms and 10ms, respectively, and connections are determined as described in Section S3.4. The simulations reported in the main paper were carried out on a Dell Precision T7600 workstation with dual Intel Xeon E5-2630 (6-core, 2.3Ghz) processors.

S3.3 Nonlinear adaptive control in neurons

In nonlinear adaptive control, it is assumed that system performance errors are a result of sub-optimal control signals, caused by inaccuracies in the controller's model of the system. The goal is to use learning to improve the controller's estimate of the system's dynamics and kinematics, thereby improving the generated control signals and reducing system performance errors. To achieve this, two adaptive components are added; one for learning dynamics, and one for learning kinematics. Each component is a set of nonlinearities, which act as basis functions, driven by system information. The output of each component is a summation of the basis function activity weighted by a set of learned parameters. This output is then either added to or multiplied by the control signal. Although referred to throughout the main text as cerebellar and cortical adaptation, throughout this section these two types of adaptation will be referred to as dynamics and kinematics adaptation, respectively, reflecting their functional role.

Using the NEF, it is possible to implement the nonlinear adaptive control methods of [5, 46] directly, by calculating specific functions of the system state to use as bases, and learning an appropriate weighting across these functions. However, this original formulation of the adaptive control methods requires carefully choosing the basis functions such that they perfectly match the kinematics and dynamics of the system and its environment. It is more desirable to, instead, simply use some multitude of simple basis functions that cover a state space sufficiently, such that they are able to approximate a wide variety of functions.

In [36], the authors use sets of Gaussian functions with multi-dimensional output to tile the state space, and learn a weighting over these functions that approximates the unknown dynamics of a system. To be able to use neurons as the basis functions, we need to rework this approach such that we have scalar basis function output and multi-dimensional weights (NEF decoders). An interesting benefit here is that the curse-of-dimensionality is avoided when tiling multi-dimensional spaces. This is because in the NEF implementation different dimensions are tiled independently, which avoids the coupling between dimensions that leads to sparse coverage of state space. Additionally, we note

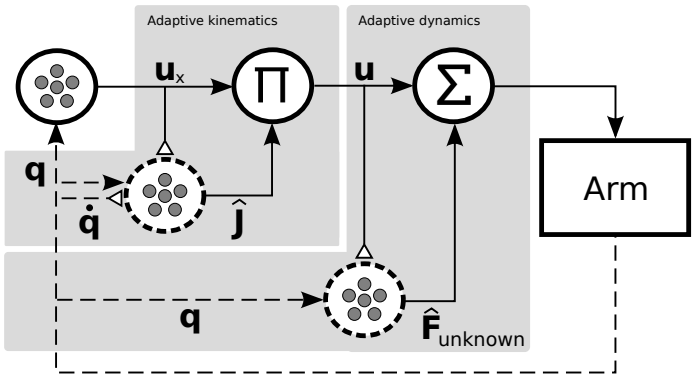

Fig S3. A diagram of the basic setup of a control system using dynamics and kinematics adaptation components. Solid arrows represent normal connections, hollow triangles are modulatory connections used in learning, and dotted circles denote adaptive neural populations. The multiplicative adaptation component takes the system's joint angles $\mathbf{q}$ as input and the torques $\mathbf{u}$ and joint angle velocity $\dot{\mathbf{q}}$ to generate the learning signal, and outputs an approximation of the system Jacobian, $\hat{\mathbf{J}}$. This signal is then used in a dot-product with the desired end-effector forces, $\mathbf{u_x}$, to generate the corresponding set of joint torques, $\mathbf{u}$. The adaptive dynamics component takes the system's joint angles $\mathbf{q}$ as input and the torques $\mathbf{u}$ as a learning signal, and outputs an approximation of the unknown forces, $\hat{\mathbf{F}}_{unknown}$. This signal is then summed with $\mathbf{u}$ and sent out to the arm.

that we can then use the NEF to initialize the neural network with an approximation of the ideal system (when prior knowledge is available), which speeds up convergence time and gives reasonable performance at initialization.

Restated, the critical difference between the work of Sanner and Slotine [36] and that presented here is, in their work, each of the basis functions output a multi-dimensional signal which is weighted by a scalar, whereas in the system here each basis function component outputs a scalar term, and the set of learned parameters are multi-dimensional. This novel reformulation is presented formally below, and proofs of global asymptotic stability of the system using Lyapunov's direct method can be found in [9].

S3.3.1 Adaptive dynamics

In the dynamics adaptation component, there are assumed to be some forces acting on the system that are unaccounted for by the control signal. Shown in the 'Adaptive dynamics' block in Figure S3, the system's joint angles, $\mathbf{q}$, are projected into a population of neurons which act as basis functions, shown in the dotted circle. The outgoing joint torque control signal, $\mathbf{u}$, is used as a modulatory signal to train the adaptive dynamics component to generate a force that compensates for some unknown perturbance, $\hat{\mathbf{F}}_{unknown}$.

In the original formulation presented in [46], the unknown forces affecting the system are approximated

$$\hat{\mathbf{F}}_{unknown} = \mathbf{Y}_d(\mathbf{q})\hat{\boldsymbol{\theta}}_d \tag{9}$$

where $\mathbf{Y}_d \in \mathbb{R}^{n\times d}$ is the set of basis functions and $\boldsymbol{\theta}_d \in \mathbb{R}^{d\times 1}$ are the learned parameters, where n is the number of basis functions and d is the dimensionality of the control signal, $\mathbf{u}$.

Our reformulation is such that the combined output of the basis functions is now a vector (rather than a matrix) and the learned parameters are now a matrix (rather than a vector). Let the basis functions in vector form be denoted $\mathbf{X}_d \in \mathbb{R}^{1\times n}$, and the corresponding set of decoders is of the form $\hat{\boldsymbol{\theta}}_d \in \mathbb{R}^{n\times d}$. The unknown forces approximation is unchanged,

$$\hat{\mathbf{F}}_{unknown} = \mathbf{X}_d(\mathbf{q})\hat{\boldsymbol{\theta}}_d, \tag{10}$$

but now the learning rule is done with an outer product rather than an inner product:

$$\dot{\hat{\boldsymbol{\theta}}}_d = \mathbf{L}_d \mathbf{X}_d \otimes ((\dot{\mathbf{q}} - \dot{\mathbf{q}}_{\text{des}}) + \Lambda(\mathbf{q} - \mathbf{q}_{\text{des}})), \tag{11}$$

where $\mathbf{L}_d$ is the learning rate, $\mathbf{q}$ and $\mathbf{q}_{\text{des}}$ are the current and target joint angles, $\dot{\mathbf{q}}$ and $\dot{\mathbf{q}}_{\text{des}}$ are the current and target joint velocities, $\otimes$ denotes the outer product operation, and Λ is some symmetric positive definite matrix.

S3.3.2 Adaptive kinematics

The kinematics adaptation component is shown in the 'Adaptive kinematics' block in Figure S3, Given the current system joint angles, $\mathbf{q}$, and target (not shown in Figure S3), the end-effector forces which will move the hand to the target are generated. To transform the end-effector forces into joint torques that can be sent to the arm, the adaptive population, shown in the dotted circle, learns to approximate the Jacobian of the system. This Jacobian approximation, $\hat{\mathbf{J}}(\mathbf{q}, \hat{\boldsymbol{\theta}}_k)$, is projected to a dot-product population along with $\mathbf{u}_{\mathbf{x}}$, where an approximation to the joint torques that will move the hand as desired are calculated.

When implementing kinematics adaptation in a neural network, several obstacles appear. The first obstacle results from using a distributed system for implementation. In the original formulation of multiplicative adaptation [5], the adaptation component is trained to approximate the end-effector velocity

$$\dot{\mathbf{x}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} = \mathbf{Y}_k(\mathbf{q}, \dot{\mathbf{q}})\hat{\boldsymbol{\theta}}_k, \tag{12}$$

where the matrix $\mathbf{Y}_k(\mathbf{q}, \dot{\mathbf{q}})$ is chosen carefully to be the set of system kinematics functions, such that the learned components will converge to the arm segment lengths using the corresponding learning rule

$$\dot{\hat{\boldsymbol{\theta}}}_k = \mathbf{L}_k \mathbf{Y}_k(\mathbf{q}, \dot{\mathbf{q}})^T \mathbf{K}_p(\mathbf{x} - \mathbf{x}_{\text{des}}), \tag{13}$$

where $\mathbf{L}_k$ is the learning rate, $\mathbf{K}_p$ is the proportional gain term, and $\mathbf{x}$ and $\mathbf{x}_{\text{des}}$ are the current and target end-effector positions. These learned parameters are then be extracted artificially from $\mathbf{Y}_k(\mathbf{q}, \dot{\mathbf{q}})\hat{\boldsymbol{\theta}}_k$ to build the approximation of the Jacobian $\hat{\mathbf{J}}(\mathbf{q}, \hat{\boldsymbol{\theta}}_k)$. Unfortunately, this can't be done in a distributed system, because the approximation of each of the elements aren't cleanly separated into distinct system parameters. Rather, a weighted summation across all of the nonlinear components of the distributed system must be taken to get a meaningful signal. The result of this weighted summation then is the signal $\hat{\mathbf{J}}(\mathbf{q}, \hat{\boldsymbol{\theta}}_k)\dot{\mathbf{q}}$, from which $\hat{\mathbf{J}}(\mathbf{q}, \hat{\boldsymbol{\theta}}_k)$ cannot be cleanly extracted. To avoid this, the adaptive system we use here instead approximates the Jacobian directly. This, however, means that the learning rule used in the original formulation is no longer appropriate; we must now incorporate joint velocity as well into the training signal.

To address this, we first reformulate the problem from using multi-dimensional basis function output and scalar learned parameters to using scalar basis functions and multi-dimensional learned parameters, as in the dynamics adaptation case. We define a set of basis functions $\mathbf{X} \in \mathbb{R}^{1 \times i}$ to be a vector representing the activity of a set population of neurons. The learned parameters, $\hat{\boldsymbol{\theta}} \in \mathbb{R}^{i \times j \times k}$, will be a tensor, and the goal will be to train them such that

$$\mathbf{X}\hat{\boldsymbol{\theta}} = \mathbf{J}(\mathbf{q}), \tag{14}$$

where the Jacobian $\mathbf{J}(\mathbf{q}) \in \mathbb{R}^{j \times k}$, and the subscripts have been removed so we may use Einstein summation notation below.

Incorporating joint velocity into our new learning rule, it can be written (using Einstein summation notation):

$$\dot{\hat{\boldsymbol{\theta}}}_{ijk} = L\mathbf{X}_i(\mathbf{q})\dot{\mathbf{q}}_k \mathbf{K}_{jl}(\mathbf{x} - \mathbf{x}_{\text{des}})_l, \tag{15}$$

where matching subscripts between terms indicate a dot product along those dimensions, L is a learning rate, and $\mathbf{K}$ is the proportional gain term.

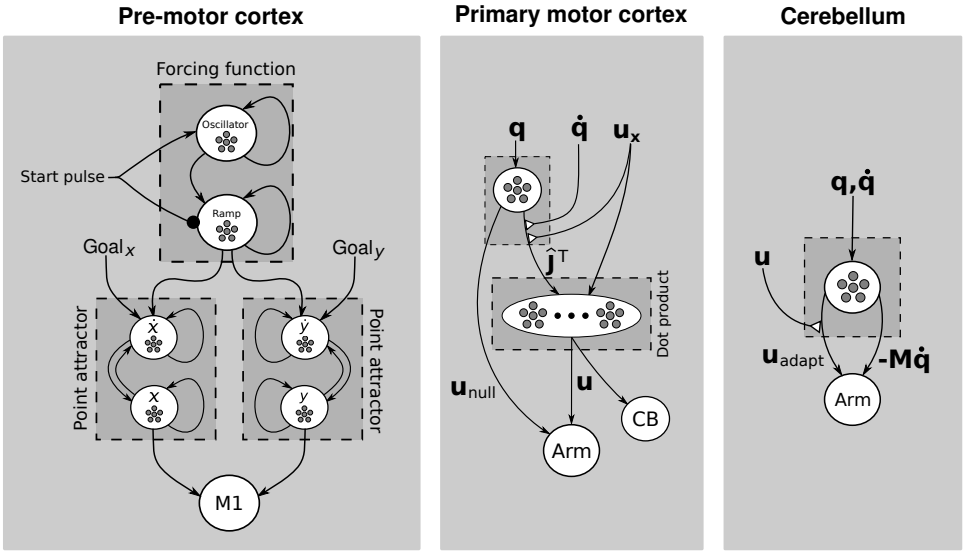

Fig S4. The pre-motor cortex, primary motor cortex, and cerebellum models used in REACH. Solid arrow denote normal connections, solid circles denote inhibitory connections, and hollow triangles denote modulatory connection used for learning. See text for details.

S3.4 System architecture

In this section we detail the relationship between the neural populations used to implement the control model described previously and the various brain areas encompassed by the model.

S3.4.1 The pre-motor cortex

The pre-motor cortex is modeled using a neural implementation of dynamical movement primitives (DMPs) [22,39], which specify desired trajectories in operational space. DMPs are simple controllers that can be used to rapidly learn and generate complex trajectories. Trajectories are typically specified in an abstracted space, such as the Cartesian coordinates of the end-effector, which allows multiple limbs to be driven by the same DMP set (support for such abstraction in humans is found in handwriting studies, such as [56]). DMPs are dynamical systems that have several nice properties, including guaranteed convergence, and are easily generalized both spatially (e.g. scaling, rotation, etc.) and temporally (e.g. rapid approximate, or slow accurate movements).

There are two basic parts to DMPs; the point attractor and the canonical system. The point attractor is a simple system with spring dynamics that moves from a starting location towards a target. This system provides a stable starting point for our trajectory generation system, with easily characterized, and convergent, behavior. The role of the canonical system is to supply a force signal to the attractor that causes it to move along some desired path along its way to the target. In the REACH model, the canonical system is a simple dynamical system that is initialized to 0 and linearly increases to 1. The neurons of the canonical system have predictable activity profiles as the value represented in the population goes from 0 to 1. We take advantage of this fact by connecting these neurons to the attractor population and using the NEF to assign weights to the connection such that a desired forcing signal is decoded out of their activity. This causes the attractor to be pushed along the desired trajectory.

Figure S4 shows a diagram of the pre-motor cortex model on the left-hand side. The PMC is provided a 'start pulse' signal, which is a step function that transitions an oscillator from a default resting state into its limit cycle activity pattern. While the start pulse function is active the goal states are the starting positions for the arm, driving the x and y populations to the initial end-effector

position. The start pulse also inhibits the forcing function, resetting it to 0 and allowing the x and y point attractors to quickly converge to their targets. These desired positions x and y are then sent to M1, where the rest of the motor system is responsible for physically moving the arm to that starting location for the trajectory. When the start pulse finishes, the goal input to x and y changes to the end position of the trajectory, and the point attractors begin moving towards the new targets.

At the same time, the oscillator projects a high-frequency signal with a non-zero mean into the forcing function, which causes it to move linearly from 0 to 1. The reason for using a high-frequency oscillator as opposed to a constant signal is that the small fluctuations of the oscillating signal help to prevent the forcing function population from getting stuck in local minima while integrating. As the forcing function value moves from 0 to 1, the neural activity is decoded into the learned function that moves the $\dot{x}$ and $\dot{y}$ point attractors along the desired trajectory.

Figure S5 demonstrates the REACH model using these circuits to guide the arm through an example set of complex trajectories. The digits from 0-9 and the words "hello world" were learned from a single presentation of one of the authors' handwriting. In addition, Figure S5C demonstrates the spatial generalizability of the learned path to arbitrary sizes by adjusting a single internal parameter in the DMP online, during movement. This kind of path learning is rapid because we directly optimize neural connection weights offline (using the NEF) generate the desired paths.

There has been much work extending the DMP architecture [22], including allowing the system to generalize, incorporating system position feedback, and to coupling multiple DMPs or signals from the environment. Here, we have implemented a basic model of a DMP architecture in spiking neurons here, capable of spatial generalization (see Figure S5) which lays the groundwork for the development of neural models that include such extensions. It should be noted, however, that DMPs do not solve two critical problems faced by the motor system. The first is to allow for adaptation to unknown dynamics and kinematics. The second is to determine how to control a redundant higher degree-of-freedom system in such a way that it realizes the path planned by the DMP. For these purposes, we turn to other aspects of the REACH system.

S3.4.2 The primary motor cortex

The primary motor cortex (M1) is used in the REACH model to map high-level control signals defined in an abstract space to a low-level control signal that can be sent to the body; in this implementation end-effector forces are mapped to joint torques. This mapping accounts for some of the effects of inertia on the system and helps to cancel them out, as well as generating a secondary control signal in the null space of the primary control signal which serves to keep the system near a comfortable 'resting state', or default position. In addition to these functions it also learns to correct for inaccuracies in the Jacobian, minimizing performance errors using only the high-level control signal $\mathbf{u_x}$ and system velocity $\dot{\mathbf{q}}$ as training signals.

The neural network implementation of these computations can be broken down into several parts. The first is a population representing the system state, from which the adaptive Jacobian, $\hat{\mathbf{J}}$ is decoded. Note that the adaptive Jacobian also includes the inertia matrix $\mathbf{M_x}$. The connection along which the Jacobian is generated is subject to adaptation, using the high-level control signal $\mathbf{u_x}$ and system velocity $\dot{\mathbf{q}}$ as training signals, implementing the learning algorithm described in Section S3.3.2, using the learning rule detailed in [29]. This ensures that the internal representation of the Jacobian $\hat{\mathbf{J}}$ (and the inertia matrix $\mathbf{M_x}$) is improved and kept up to date if the properties of the system and its environment change.

The second is an array of ensembles which the Jacobian approximation is projected to along with the high-level control signal $\mathbf{u_x}$, where a dot-product operation is performed to calculate the low-level control signal, as described in Eq. 8. As shown in the middle diagram in Figure S4, the resulting low-level control signal, $\mathbf{u}$, is then sent to the cerebellum as a training signal and to the arm.

Finally, a secondary control signal is generated in the null space of the primary control signal. The role of this secondary control signal is to keep the arm near its resting position without interfering

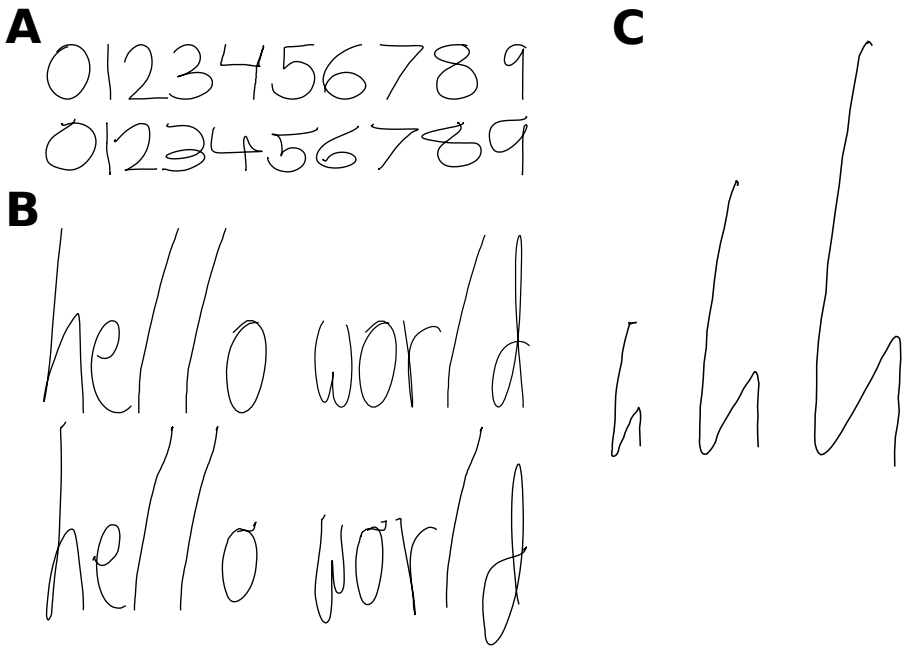

Fig S5. A sample of an authors' handwriting (which was used to train the model) compared with the model's handwriting. These trajectories were learned from single example presentations. A) Writing out the numbers 0-9; an authors' handwriting used for training above, the model's path below. B) Writing out the phrase 'hello world'; an authors' handwriting above, the model's below. C) Different scaling of the same path by adjusting a single parameter during movement.

with the movement specified by the primary control signal, in effect solving the configuration redundancy problem that arises when there is more than one way to move along a trajectory. This secondary control signal, $\mathbf{u}_{null}$, is calculated using Eq. 8, and is sent to the arm directly.

S3.4.3 The cerebellum

In the REACH model, the cerebellum provides a corrective signal to the arm that cancels out the effects of inertia and learns to account for and minimize errors due to unmodeled dynamics that arise during movement.

Motor control researchers generally identify two kinds of internal models: forward models, which map potential actions to their outcomes; and inverse models, which map desired outcomes to actions. In the REACH model, the cerebellum receives an efferent copy of the M1 generated control signal as well as sensory information regarding the current state of the body. The cerebellum uses this information to learn the result of an action given the resulting error. This error correction incorporates both forward and inverse models, where the cerebellum takes in the control signal and current system state to predict the resulting movement error, and then generates a corrective control signal that is added to the M1 output. Support for the cerebellum providing error correction based on predicted error is found in numerous studies [15, 47, 54].

To do this, the cerebellum calculates the forces due to the current motions of the system, $\mathbf{M}\dot{\mathbf{q}}$ and projects a compensatory signal to the arm. At the same time, an adaptive signal is also projected to the arm, $\mathbf{u}_{adapt}$, which learns to correct for unmodeled forces using the low-level control signal as a training signal, through the algorithms described in Section S3.3.1.

Although the model presented here is in spiking neurons and functionally detailed, there is a wealth of neuroanatomical details of the cerebellar structure that are not included in the REACH model [40]. Important future work will be to determine how the known anatomical structure of the

cerebellum can be exploited to perform the proposed computations.

S3.5 Neural activity analysis

S3.5.1 Neuron selection

In the implementation of REACH presented in this paper, there are 31,500 spiking neurons: 4,500 in the pre-motor cortex, 24,000 in the primary motor cortex, and 3,000 in the cerebellum.

In the REACH's cerebellar network, neurons are randomly assigned preferred directions throughout a 6-dimensional state space which encodes the arm joint angles and joint velocities. The high-level movement parameter correlation analysis (shown in Figure 4 of the main text) was performed on neurons in the cerebellum (a different neuron for each high-level movement parameter) that displayed strong correlations throughout multiple separate reaching trials.

In Figure 4 A of the main text we perform an analysis of a single neuron in REACH's primary motor cortex model, from the neural population which generates the Jacobian approximation, $\hat{\mathbf{J}}$. This population consists of 3000 neurons. The neurons in this population are randomly assigned preferred directions in a 6-dimensional state space sensitive to the joint angles. We display the activity of the same neuron throughout 5 trials of 8 different reaches, chosen for its similarity to the neuron in the Georgopoulos experiment [19].

In Figure 4 B of the main text we perform jPCA analysis on a group of randomly selected neurons from the primary motor cortex area responsible for transforming the high-level control signal from hand forces into low-level joint torques (3000 neurons). We selected 218 neurons, the same size as the group selected in [6]. These neurons were chosen from the part of the M1 model that performs the dot product between the Jacobian transpose approximation, $\hat{\mathbf{J}}^T$, and high-level control signal, $\mathbf{u_x}$. The neurons in this part of the network have preferred directions randomly generated in a 2-dimensional state space sensitive to one dimension of $\hat{\mathbf{J}}$ and one dimension of $\mathbf{u_x}$.

S3.5.2 Spike filtering

We recorded the spike trains from the neurons at a millisecond time step, and binned the spikes with a 10ms window. This neural activity was smoothed out using a 20ms Gaussian filter. The code used to perform this processing is available at https://github.com/studywolf/REACH-paper/blob/master/analysis/. We employed the same code used in the original experiments for generating the figures in the main paper.

S3.5.3 jPCA analysis

Briefly, in jPCA the neural data is projected into the space of the top principal components (PCs) of the population activity. This provides a low-dimensional representation of the neural activity during each trial. Next, a matrix that best captures the dynamics of the low-dimensional trajectory over time is calculated, with the constraint that the matrix must be skew-symmetric. The skew-symmetry forces the dynamics to be captured in terms of a set of oscillators, and ensures that the PCs of the matrix come in complex conjugate pairs. These PC pairs are used to create 'jPC' pairs, where the first is the normalized sum of the real components of the PC pairs, and the second is the normalized sum of the imaginary components of the PC pairs. Finally, the low-dimensional representation is projected into this rotation-centric space. We performed the same analysis on our model data as was performed on the monkey data to generate Figure 4 B in the main text.

In the remainder of this section, we describe additional simulations that provide insight into jPCA, and the reasoning behind our final prediction in the paper. That prediction suggests jPCA analysis will provide rotational responses from a wide variety of neural areas processing information over time. One of the benefits of being able to implement dynamical systems in spiking neurons is that we are able to perform neural analysis on spiking data where we explicitly know the underlying representations and dynamics being computed. To gain insight into the method, we performed the

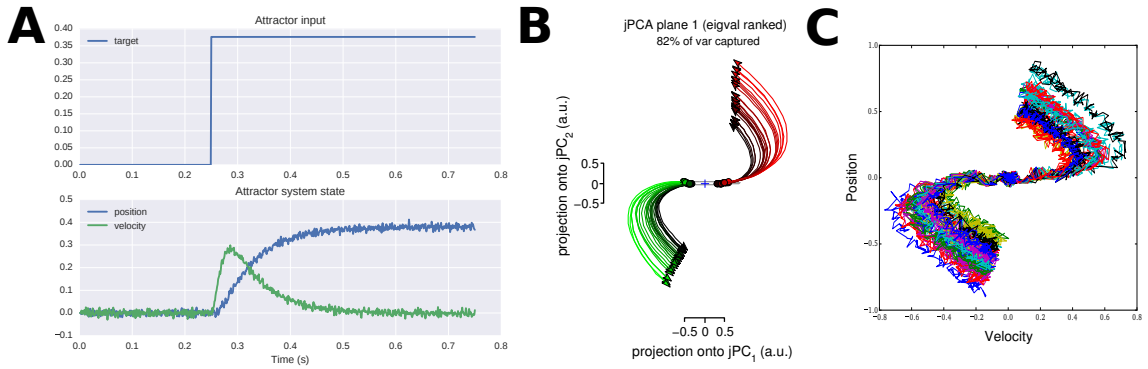

Fig S6. jPCA analysis of an attractor network implemented in spiking neurons. A) Top: The input is a signal representing the target state. Bottom: The dynamics of the network drive it towards the target input. B) Resulting trajectories in low-dimensional state space performing jPCA analysis on data collected from 50 trials to random targets, where randomly generated white noise is added to each trial. C) Plotting the attractor network's position and velocity signals against one another, without jPCA analysis. Data from 50 trials moving directly to the target are shown here.

jPCA analysis on a neural network implementing a 1D point attractor, representing the system position x and velocity $\dot{x}$. To drive the system to the target, the velocity is calculated as the difference between the target and current state of the system. The code for generating this model can be fonud at https://github.com/studywolf/REACH-paper/blob/master/analysis/.

The network was generated and then run 50 times with randomly seeded white noise, moving to randomly chosen targets between .5 and 1 or -.5 and -1. The spiking neural data was recorded and then processed using the methods described in Section S3.5.2. Once binned and smoothed, we imported the data into Matlab and jPCA analysis was performed using the code supplied by the authors in [6], which can be found at http://churchlandlab.neuroscience.columbia.edu/code/jPCA_ForDistribution.zip.

Results of the jPCA analysis are shown in Figure S6B, where very clean rotations through state space can be seen. These rotations are expected, because plotting the represented position and velocity signals against each other gives rise to a rotating movement, even before analysis, as shown in Figure S6C. The system position moves towards a target value while the system velocity increases and then decreases, giving a rotational movement through state space.

Specific predictions result from this analysis. We can predict that anywhere both the position and velocity of the system during movement are represented these rotations will be present, whether the underlying neural architecture implements an attractor network with recurrent connections or is simply representing signals projected in through system feedback. In addition to explaining the source of the rotations in the pre-motor cortex neural activity (which modeled a 2-dimensional attractor networks as part of the DMP implementation) and the primary motor cortex (which represents both the system position and velocity), we predict that these rotations will be present both in the motor areas of the cerebellum and in the sensory cortex during movements, as they are expected to represent position and velocity information as well.

S3.5.4 Correlations of neural responses in motor areas

The following table summarizes correlations reported across a series of 14 studies between movement parameters and neural activity, and describes which model representation accounts for the correlation in the REACH model. Figure 4 provides data identifying these correlations in the model.

Movement parameter	**Model representation**	**References**
Acceleration	system state	M1 [3, 14]
Arm position	system state	CB [35] M1 [25, 43]
Distance to target	control signal	CB [18], {M1} [17]
Hand position	system state	M1 [3]
Movement direction	control signal	CB [16, 18, 35], M1 [3, 19]
Movement force	control signal	M1 [23, 44]
Movement magnitude	control signal	CB [16], M1 [33]
Movement velocity	system state	CB [35] M1 [1]

Table S1. A summary of the correlations with movement parameters found in both experimental neural recordings and the REACH model from different areas of the motor system. The model representation column identifies the signal that carries information about the given movement parameter in the model (see Section S3). CB: cerebellum; M1: primary motor cortex.

References

1. G. Alexander and M. Crutcher. Neural representations of the target (goal) of visually guided arm movements in three motor areas of the monkey. *J Neurophys*, 64(1):164–178, 1990.

2. M. Andani et al. Modem: a multi-agent hierarchical structure to model the human motor control system. *Bio Cyber*, 101(5-6):361–377, 2009.

3. J. Ashe and A. Georgopoulos. Movement parameters and neural activity in motor cortex and area 5. *Cerebral Cortex*, 4(6):590–600, 1994.

4. T. Bekolay et al. Nengo: a python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics*, 7, 2013.

5. C.-C. Cheah et al. Adaptive tracking control for robots with unknown kinematic and dynamic properties. *The Int'l J. Robotics Research*, 25(3):283–296, 2006.

6. M. Churchland et al. Neural population dynamics during reaching. *Nature*, 2012.

7. M. M. Churchland et al. Cortical preparatory activity: representation of movement or first cog in a dynamical machine? *Neuron*, 68(3):387–400, 2010.

8. C. Darlot et al. Computation of inverse dynamics for the control of movements. *Bio Cyber*, 75(2):173–186, 1996.

9. T. DeWolf. *A neural model of the motor control system*. PhD thesis, University of Waterloo, 2015.

10. T. DeWolf and C. Eliasmith. The neural optimal control hierarchy for motor control. *J. Neural Eng*, 8(6):065009, 2011.

11. S. Dura-Bernal, X. Zhou, S. A. Neymotin, A. Przekwas, J. T. Francis, and W. W. Lytton. Cortical spiking network interfaced with virtual musculoskeletal arm and robotic arm. *Frontiers in Neurorobotics*, 9:13, 2015.

12. C. Eliasmith and C. H. Anderson. *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. MIT Press, 2004.

13. C. Eliasmith et al. A large-scale model of the functioning brain. *Science*, 338(6111), 2012.

14. D. Flament and J. Hore. Relations of motor cortex neural discharge to kinematics of passive and active elbow movements in the monkey. *J. Neurophys*, 60(4):1268–1284, 1988.

15. J. R. Flanagan et al. Prediction precedes control in motor learning. *Current Bio*, 13(2), 2003.

16. P. Fortier et al. Cerebellar neuronal activity related to whole-arm reaching movements in the monkey. *J Neurophysiol*, 62(1):198–211, 1989.

17. Q. Fu et al. Temporal encoding of movement kinematics in the discharge. *J Neurophys*, 1995.

18. Q.-G. Fu et al. Relationship of cerebellar purkinje cell simple spike discharge to movement kinematics in the monkey. *J. Neurophys*, 78(1):478–491, 1997.

19. A. 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.

20. S. Grossberg and M. Kuperstein. *Neural dynamics of adaptive sensory-motor control: Ballistic eye movements*, volume 30. Elsevier, 2011.

21. M. Haruno et al. Hierarchical mosaic for movement generation. In *Int'l congress series*, volume 1250, pages 575–590. Elsevier, 2003.

22. A. J. Ijspeert et al. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural Comp*, 25(2):328–373, 2013.

23. J. Kalaska et al. Parietal area 5 neuronal activity encodes movement kinematics, not movement dynamics. *Exp Brain Research*, 80(2):351–364, 1990.

24. M. Kawato and H. Gomi. A computational model of four regions of the cerebellum based on feedback-error learning. *Bio Cyber*, 68(2):95–103, 1992.

25. R. Kettner et al. Primate motor cortex and free arm movements to visual targets in three-dimensional space. iii. positional gradients and population coding of movement direction from various movement origins. *J. Neurosci*, 8(8):2938–2947, 1988.

26. O. Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *Robotics and Automation*, 3(1):43–53, 1987.

27. R. Laje and D. V. Buonomano. Robust timing and motor patterns by taming chaos in recurrent neural networks. *Nature Neurosci*, 16(7):925–933, 2013.

28. R. Llinas and M. Sugimori. Electrophysiological properties of in vitro purkinje cell somata in mammalian cerebellar slices. *J. Phys*, 305:171, 1980.

29. D. MacNeil and C. Eliasmith. Fine-tuning and the stability of recurrent neural networks. *PLoS ONE*, 6, 2011.

30. M. Mattia et al. Heterogeneous attractor cell assemblies for motor planning in premotor cortex. *J. Neurosci*, 33(27):11155–11168, 2013.

31. S. Menon et al. Controlling articulated robots in task-space with spiking silicon neurons. In *IEEE Int'l Conf on Biomedical Robitics and Biomechatronics*. IEEE Press, 2014.

32. R. Miall and D. Wolpert. Forward models for physiological motor control. *Neural Networks*, 9(8):1265–1279, 1996.

33. D. W. Moran and A. Schwartz. Motor cortical representation of speed and direction during reaching. *J. Neurophys*, 82(5):2676–2692, 1999.

34. S. Morton and A. Bastian. Cerebellar control of balance and locomotion. *The Neuroscientist*, 10(3):247–259, 2004.

35. A. V. Roitman et al. Position, direction of movement, and speed tuning of cerebellar purkinje cells during circular manual tracking in monkey. *J. Neurosci*, 25(40):9244–9257, 2005.

36. R. Sanner and J. Slotine. Gaussian networks for direct adaptive control. *Neural Networks, IEEE Transactions on*, 3(6):837–863, 1992.

37. R. M. Sanner and M. Kosha. A mathematical model of the adaptive control of human arm motions. *Bio Cyber*, 80(5):369–382, 1999.

38. R. M. Sanner and J. Slotine. Stable adaptive control of robot manipulators using "neural" networks. *Neural Comp*, 7(4):753–790, 1995.

39. S. Schaal. Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. In *Adaptive Motion of Animals and Machines*. Springer, 2006.

40. N. Schweighofer et al. Role of the cerebellum in reaching movements in humans. ii. a neural model of the intermediate cerebellum. *European J. Neurosci*, 10(1):95–105, 1998.

41. P. Schwindt et al. Quantitative analysis of firing properties of pyramidal neurons from layer 5 of rat sensorimotor cortex. *J. Neurophys*, 77(5):2484–2498, 1997.

42. S. H. Scott. Optimal feedback control and the neural basis of volitional motor control. *Nature Reviews Neurosci*, 5(7):532–546, 2004.

43. S. H. Scott and J. Kalaska. Changes in motor cortex activity during reaching movements with similar hand paths but different arm postures. *Changes*, 73(6), 1995.

44. L. E. Sergio et al. Motor cortex neural correlates of output kinematics and kinetics during isometric-force and arm-reaching tasks. *J. Neurophys*, 94(4):2353–2378, 2005.

45. B. Siciliano and J. Slotine. A general framework for managing multiple tasks in highly redundant robotic systems. In *Advanced Robotics, 1991.'Robots in Unstructured Environments', 91 ICAR., Fifth Int'l Conf on*, pages 1211–1216. IEEE, 1991.

46. J.-J. Slotine and W. Li. On the adaptive control of robot manipulators. *The Int'l J. Robotics Research*, 6(3):49–59, 1987.

47. M. Smith and R. Shadmehr. Intact ability to learn internal models of arm dynamics in huntington's disease but not cerebellar degeneration. *J. Neurophys*, 93(5):2809–2821, 2005.

48. C. Stefanini et al. A compliant bioinspired swimming robot with neuro-inspired control and autonomous behavior. In *Robotics and Automation (ICRA), 2012 IEEE Int'l Conf on*, pages 5094–5098. IEEE, 2012.

49. T. Stewart and C. Eliasmith. Large-scale synthesis of functional spiking neural circuits. *Proc. of the IEEE*, 102(5):881–898, May 2014.

50. N. Sugimoto et al. Mosaic for multiple-reward environments. *Neural Comp*, 24(3), 2012.

51. E. Todorov. Direct cortical control of muscle activation in voluntary arm movements: a model. *Nature Neurosci*, 3(4):391–398, 2000.

52. E. Todorov. Efficient computation of optimal actions. *Proc. of the National Academy of Sciences*, 106(28):11478–11483, 2009.

53. E. Todorov and W. Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *American Control Conf.* IEEE, 2005.

54. Y.-w. Tseng et al. Sensory prediction errors drive cerebellum-dependent adaptation of reaching. *J. Neurophys*, 98(1):54–62, 2007.

55. T. Waegeman et al. Macop modular architecture with control primitives. *Frontiers in Comp Neurosci*, 7, 2013.

56. A. M. Wing. Motor control: Mechanisms of motor equivalence in handwriting. *Current Biology*, 10(6):R245–R248, 2000.

57. D. M. Wolpert and M. Kawato. Multiple paired forward and inverse models for motor control. *Neural Networks*, 11(7):1317–1329, 1998.