LETTER **Communicated by Dongsung Huh**

Hierarchical Control Using Networks Trained with Higher-Level Forward Models

Greg Wayne
gregwayne@google.com
L. F. Abbott
lfabbott@columbia.edu
Department of Neuroscience and Department of Physiology and Cellular Biophysics, Columbia University College of Physicians and Surgeons, New York, NY 10032-2695, U.S.A.

We propose and develop a hierarchical approach to network control of complex tasks. In this approach, a low-level controller directs the activity of a "plant," the system that performs the task. However, the low-level controller may be able to solve only fairly simple problems involving the plant. To accomplish more complex tasks, we introduce a higher-level controller that controls the lower-level controller. We use this system to direct an articulated truck to a specified location through an environment filled with static or moving obstacles. The final system consists of networks that have memorized associations between the sensory data they receive and the commands they issue. These networks are trained on a set of optimal associations generated by minimizing cost functions. Cost function minimization requires predicting the consequences of sequences of commands, which is achieved by constructing forward models, including a model of the lower-level controller. The forward models and cost minimization are used only during training, allowing the trained networks to respond rapidly. In general, the hierarchical approach can be extended to larger numbers of levels, dividing complex tasks into more manageable subtasks. The optimization procedure and the construction of the forward models and controllers can be performed in similar ways at each level of the hierarchy, which allows the system to be modified to perform other tasks or to be extended for more complex tasks without retraining lower-levels.

1 Introduction

A common strategy used by humans and machines for performing complex, temporally extended tasks is to divide them into subtasks that are more easily and rapidly accomplished. In some cases, the subtasks themselves may be quite difficult and time-consuming, making it necessary to further

Neural Computation **26, 2163–2193** (2014)
doi:10.1162/NECO_a_00639

divide them into sub-subtasks. We are interested in mimicking this strategy to create hierarchical control systems. The top-level controller in such a hierarchy receives an external command that specifies the overarching task objective, whereas the bottom-level controller issues commands that actually generate actions. At each level, a controller receives a command from the level immediately above it describing the goal it is to achieve and issues a command to the controller immediately below it describing what that controller is supposed to do. In this approach, optimization of a global cost function is abandoned in favor of a novel, more practical, but approximately equivalent approach of optimizing cost functions at each level of the hierarchy, with each level propagating cost-related information to the level below it.

As an example of this hierarchical approach, we solve a problem that requires two levels of control, using what we call lower-level and higher-level controllers. The basic problem is to drive a simulated articulated semitruck backward to a specified location that we call the final target location (the truck is driven backward because this is harder than driving forward). The backward velocity of the truck is held constant, so the single variable that has to be controlled is the angle of the truck's wheels. This problem was first posed and solved by Nguyen and Widrow (1989), and their work is an early example of the successful solution of a nonlinear control problem by a neural network. We make this problem considerably harder by moving the final target location quite far away from the truck and, inspired by the swimmer of Tassa, Erez, and Todorov (2011), distributing a number of obstacles across the environment. Although the lower-level controller can drive to a nearby location when no obstacles are in the way, it cannot solve this more difficult task. Thus, we introduce a higher-level controller that feeds a series of unobstructed, closer locations that we call subtargets to the lower-level controller that generates the wheel-angle commands. The job of the higher-level controller is to generate a sequence of subtargets that lead the truck to its ultimate goal, the final target location, without hitting any obstacles. Thus, we divide the problem into lower-level control of the truck and higher-level navigation.

The controllers at both levels of the hierarchy we construct are neural networks. The specific form of these networks is not unique and is unlikely to generalize to other tasks, so we discuss their details primarily in the appendix. We focus instead, within the text, on general principles of their operation and construction. The tasks we consider are dynamic and ongoing, so commands must be computed by the network controllers at each simulation time step. To realize the speed for this computationally intensive requirement, the network controllers are constructed to implement complex look-up tables. Each controller receives input describing the goal it is to achieve and "sensory" input providing information about the environment relevant to achieving this goal. Its output is the command specifying the goal for the controller one level down in the hierarchy.

The network controllers are trained to implement the appropriate look-up table by backpropagation on the basis of optimal training data. The training data consist of input-output combinations computed to optimize a cost function defined for each hierarchical level. At each level, optimization is achieved with the aid of an additional neural network that implements a forward model of the controller being trained. The forward model is used only for optimization during learning; the fully trained model consists of only the controller networks. For the higher-level controller, the training procedure involves what is effectively a control-theory optimization in which the "plant" being controlled is actually the lower-level controller. This approach thus extends ideas about forward models and optimization from the problem of controlling a plant to that of controlling a controller. Once the optimal output commands are determined for a large set of input commands and sensory inputs, these are used as training data for the controller, which effectively "memorizes" them.

We begin by describing how the hierarchical approach, consisting of lower- and higher-level controllers, operates after both networks have been fully trained. We do this sequentially, first showing the lower-level controller operating the truck when its subtarget data are generated externally (by us) rather than by the higher-level controller. We then discuss how the higher-level controller generates a sequence of subtarget locations to navigate through the environment. To allow the higher-level controller to detect and locate obstacles, we introduce a sensory grid system. We present and analyze the complete hierarchical system with the two controllers working together and compare its operation with that of a more conventional optimal controller. After we have shown the system in operation and analyzed its performance, we present the procedures used for training, including the cost functions and forward models used for this purpose at each level.

2 Results

All of the networks we consider run in discrete time steps, and we use this step as our unit of time, making all times integers. Distance is measured in units such that the length of the truck cab is 6, the trailer is 14, and both have a width of 6. In these units, the backward speed of the truck is 0.2. The final target for the truck and the obstacles it must avoid have a radius of 20. Distances from the initial position of the truck to the final target are typically in the range of 100 to 600.

2.1 Driving the Truck. Our hierarchical model for driving the truck (see Figure 1) starts with a lower-level controller that sends out a sequence of commands $u(t)$ that determines the angle of the wheels of the truck. This controller is provided with "proprioceptive" sensory information, namely, the cosine and sine of the angle between the cab and trailer of the truck, $[\cos(\theta_{\text{rel}}), \sin(\theta_{\text{rel}})]$, and a subtarget location toward which it is supposed to

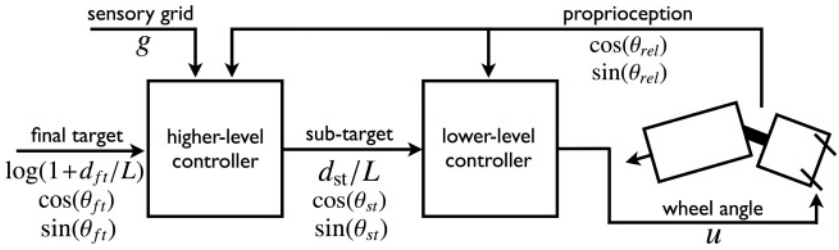

Figure 1: Flow diagram of the hierarchical control system. Commands that control the wheel angle of the truck are issued by the lower-level controller, which receives information about a subtarget direction toward which the truck should be driven from the higher-level controller. Both controllers receive proprioceptive information about the angle between the cab and trailer of the truck, and the higher-level controller also receives information about obstacles in the environment from a grid of sensors. In addition, the higher-level controller receives input about the final target that the truck is supposed to reach.

direct the truck (to ensure continuity and promote smoothness, we process all angles by taking their cosines and sines). The target information is provided as a distance from the truck to the subtarget, d_{st}/L ($L = 100$ is a scale factor) and the cosine and sine of the angle from the truck to the subtarget, $[\cos(\theta_{\mathrm{st}}), \sin(\theta_{\mathrm{st}})]$. This subtarget information is provided by a higher-level controller that receives the same proprioceptive input from the truck as the lower-level controller but also receives sensory information about obstacles in the environment (described later). In addition, the higher-level controller is provided with external information about the distance from the truck to the final target location and also the cosine and sine of the angle from the truck to this location, $[\log(1 + d_{\mathrm{ft}}/L), \cos(\theta_{\mathrm{ft}}), \sin(\theta_{\mathrm{ft}})]$. The task of the higher-level controller is to provide a sequence of subtargets to the lower-level controller that lead it safely past a set of obstacles to the final target location. Note that the higher-level controller receives the logarithm of the distance to the final target, $\log(1 + d_{\mathrm{ft}}/L)$, rather than d_{ft}/L itself. This allows for operation over a larger range of distances without saturating the network activities. The logarithm is not needed for d_{st}/L because the distance to the subtarget is maintained within a constrained range by the higher-level controller.

2.1.1 Lower-Level Controller. The job of the lower-level controller is to generate a sequence of wheel angles, $u(t)$, given the proprioceptive data, $[\cos(\theta_{\mathrm{rel}}(t)), \sin(\theta_{\mathrm{rel}}(t))]$, and a subtarget location specified by $[d_{\mathrm{st}}(t), \cos(\theta_{\mathrm{st}}(t), \sin(\theta_{\mathrm{st}}(t))]$ (see Figure 1). The proprioceptive information is needed by the controller not only to move the truck in the right direction but also to avoid jackknifing. The lower-level controller is a three-layer

Figure 2: (A) The lower-level controller directs the truck to a subtarget (white square). The black trace shows the path of the back of the truck. (B) The lower-level controller directs the truck to trace the constellation Ursa Major (the white line is the path of the truck) by approaching subtargets at the locations of the stars. The subtargets appear one at a time; when the back of the truck arrives close to the current subtarget, it is replaced by the next sub-target. (Photo by Akira Fujii.)

basis function network with the 5 inputs specified above, 100 gaussian-tuned units in a hidden layer, and 1 output unit that reports u as a linear function of its input from the hidden layer (see the appendix). Figure 2A shows an example in which the subtarget location is held fixed and the lower-level controller directs the truck along the backward path indicated by the curved line.

At this point, we are showing the lower-level controller working autonomously with the subtargets we specified, but when the truck is directed by the higher-level controller, it will be given a time-dependent sequence of subtargets. To test whether it can deal with sequential subtargets, we switched the subtarget we provide every time the truck got close to it using a rather fanciful sequence of subtargets (see Figure 2B). This indicates that the lower-level control is up to the job of following the directions that will be provided by the higher-level controller.

2.1.2 Higher-Level Controller. The higher-level controller is a five-layer feedforward network with a bottleneck architecture. It has 205 inputs (3 specifying the final target location, 2 the angle between the cab and trailer of the truck, 199 describing the state of the sensory grid described below, and a bias input; see Figure 1); hidden layers consisting of 30, 20, and 30 units; and 3 command outputs providing the subtarget information for the lower-level network (see the appendix). The bottleneck layer with 20 units ensures that the network responds only to gross features in the input that reliably predict the desired higher-level command. When we initially trained the higher-level controller without any form of bottleneck, it did not generalize well to novel situations.

In the absence of any obstacles, the job of the higher-level controller is to provide a sequence of subtargets to the lower-level controller that lead it to the location of the final target, which is specified by the variables

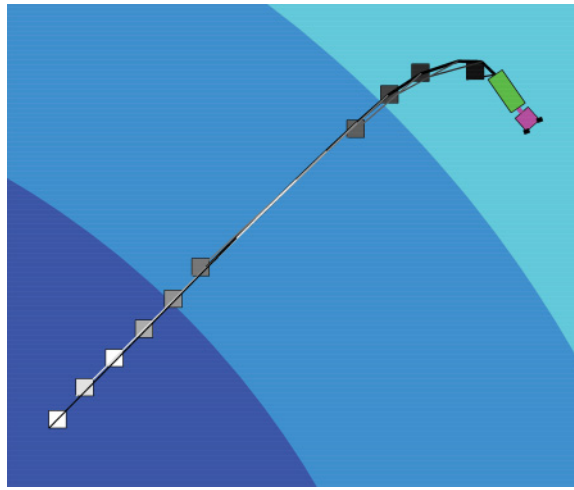

Figure 3: The truck following a sequence of subtargets provided by the higher-level controller. The subtargets are indicated by black, gray, and white squares, with darker colors representing earlier times in the sequence. The trajectory that the truck follows in pursuing the subtargets is shown in black. A connecting line indicates the subtarget that is active when the truck reaches particular trajectory points. The background shading indicates the distance to the final target, located off the lower-left corner.

$[d_{\text{ft}}, \cos(\theta_{\text{ft}}), \sin(\theta_{\text{ft}})]$ that the higher-level controller receives as external input. The higher-level controller also receives a copy of the proprioceptive input provided to the lower-level controller (see Figure 1). The subtargets that the higher-level controller propagates to the lower-level controller are natural parameterizations of the lower-level goals because they define a target state in terms of the sensory information available at the lower level. Such parameterizations have previously appeared in the motor control literature where they are known as via points (Jordan, Flash, & Arnon, 1994) and are also used frequently in navigational planning (Lazanas & Latombe, 1995). Similar ideas have also been applied to robot walking where a target state is defined at a time slice of the walker's dynamical orbit or Poincaré return map (Tedrake, Zhang, & Seung, 2004). Figure 3 shows a trajectory generated by the higher-level controller and the motion of the truck as directed by the lower-level controller, leading to a target just beyond the bottom-left corner of the plot. Note the sequence of target locations that lead the truck along the desired path. Although this example shows that the higher-level controller is operating as it should and that the lower-level controller can follow its lead, this task is quite simple and could be handled

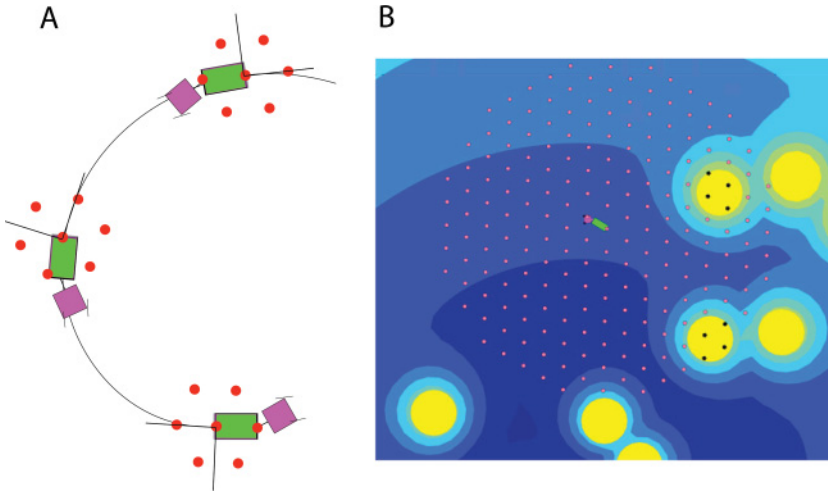

Figure 4: (A) An egocentric coordinate system surrounds the truck, composed of grid points. During movement, the grid shifts with the truck. Only a small fraction of the grid points is shown here. (B) The full set of grid points in an environment with obstacles (yellow circles). The points that lie within an obstacle are blackened, indicating that the grid element is activated.

by the lower-level controller alone. To make the task more complex so that it requires hierarchical control, we introduced obstacles into the environment.

The obstacles are discs with the same radius as the final target scattered randomly across the arena (see Figure 4B). These are soft obstacles that do not limit the movement of the truck, but during training, we penalize commands of the higher-level controller that cause the truck to pass too close to them (see below). Making this environmental change requires us to introduce a sensory system that provides the higher-level controller with information about the locations of the obstacles. Just as the final and subtarget locations are provided in "truck-centric" (egocentric) coordinates (distances and angles relative to the truck), we construct this sensory system in a truck-centric manner (see Figure 4A).

Specifically, we construct a hexagonal grid of points around the truck (see Figure 4). The grid is a lattice of equilateral triangles with sides of length 20 units. One grid point lies at the back of the trailer, and the most distant grid points are 150 units away from this point. In total, there are 199 grid points. These points move with the truck and align with the longitudinal axis of the trailer (see Figure 4A). If a grid point lies inside an obstacle, we consider it to be activated; otherwise, it is inactive. The state of the full grid is specified by a 199-component binary vector $\mathbf{g}$ with component i specifying whether grid point i is active ($g_i = 1$) or inactive ($g_i = 0$). Neither

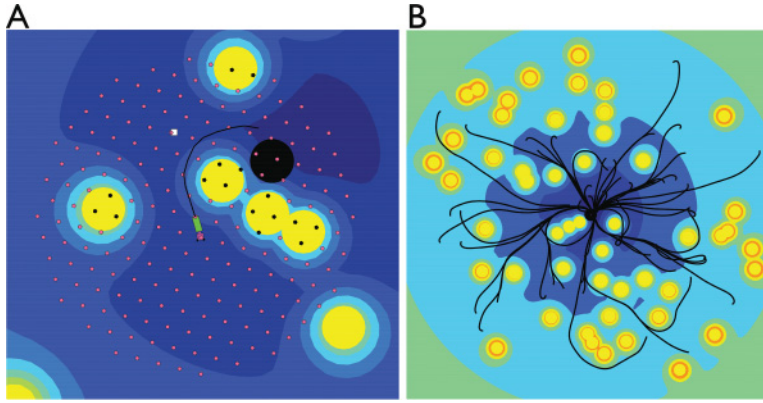

Figure 5: (A) The truck is directed to avoid the obstacles and reach the final target. The white square indicates the first subtarget; note that it is not at a position the truck actually reaches. The higher-level controller merely uses this to indicate the desired heading to the lower-level controller. (B) With 50 static obstacles, more than the 20 that were present during training, the higher-level controller steers the truck around all of the obstacles to the final target on each of 50 consecutive trials. The black lines show the paths taken by the back of the trailer.

topological closeness nor Euclidean distance information is explicit in this vector representation. The grid vector is provided as additional input to the higher-level controller (see Figure 1).

2.1.3 Operation of the Full System. We now show how the full system operates when the higher-level controller provides the lower-level controller with subtargets as they drive the truck together through a field of obstacles to the final target (see Figure 5A). Figure 5B shows a number of guided trajectories through an obstacle-filled arena.

To quantify the performance of the system, we executed 100 trials in 100 different environments with 20 obstacles. The hierarchical controller avoids the obstacles on each trial; the minimum distance to an obstacle never decreases below one obstacle radius (20 units; see Figure 6A). As the number of environmental obstacles is increased (see Figure 6B), the probability of obstacle collision grows slowly. This occurs even though the controllers were trained with only 20 obstacles in the environment. The control system directs the truck to the final target along short paths (see Figure 6C) that are comparable in length to the straight-line distance between the initial position and the nearest edge of the final target, with deviations when the truck must execute turning maneuvers or circumnavigate obstacles.

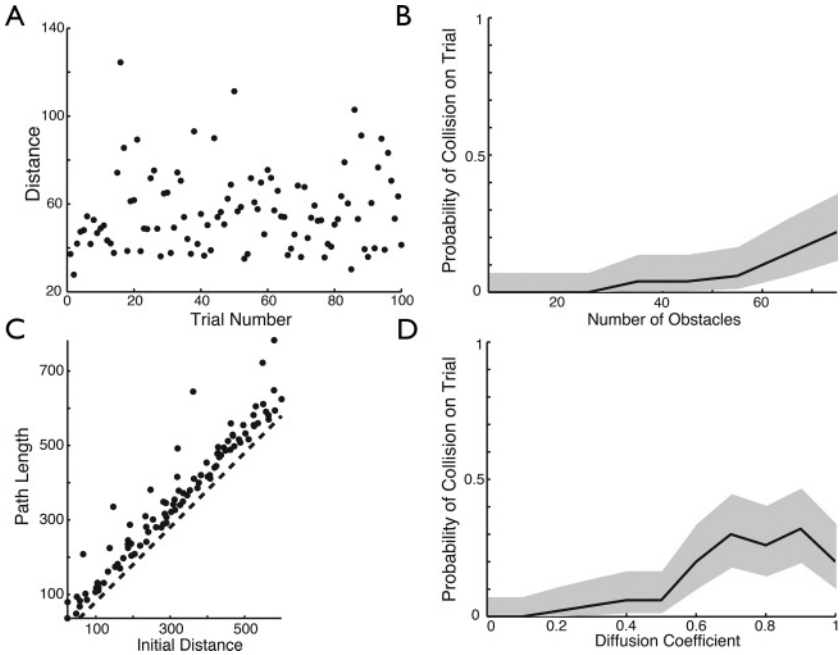

Figure 6: Performance measures. (A) Obstacle avoidance: Black dots show the minimum distance between the truck and any target averaged over 50 runs to the final goal with 20 obstacles in the environment. (B) Collisions versus obstacles: The black line shows the probability of a collision with an obstacle per trip to the final target as a function of the number of obstacles. The shaded regions are 95% confidence intervals. As the number of obstacles increases from 5 to 75, the probability of a collision grows slowly, despite high obstacle densities. (C) Target directedness: The black dots show the lengths of paths taken to the final target, averaged over 50 trials, in an environment with 20 obstacles. The dashed line shows the straight-line distance from the initial location of the truck to the nearest edge of the final target. (D) Brownian obstacle motion: The black line shows the probability of a collision with an obstacle per trip as a function of the diffusion constant of the obstacle motion. The gray region is as in panel B. Although we did not explicitly train the controllers to handle obstacle movement, the controller can frequently navigate to the goal without collision in an environment of 20 obstacles undergoing Brownian motion.

The trained higher-level controller continuously generates subtargets based on the sensory information it receives (see video 1 in the online supplement). Because all the contingencies are memorized, it needs very little time to compute these plans. Thus, the higher-level controller should be able to respond quickly to changes in the environment. To illustrate this,

we tested the system with obstacles that moved around, even though it was trained with stationary obstacles. The obstacle motions were generated as random walks (see video 2 in the online supplement). The probability of collision grows slowly with increasing diffusion constant of the random walk (see Figure 6D). At high rates of diffusion, the obstacles move significantly farther than the truck for small numbers of time steps. For example, when the diffusion coefficient is 1 (units $[L^2/T]$), the obstacles typically diffuse (but can diffuse farther than) the width of the trailer within 9 time steps. It takes the truck 30 time steps to travel the same distance.

2.1.4 Comparison to Optimal Control. The hierarchical system cannot be described as optimal with respect to a single cost criterion for several reasons. First, the responses of the networks are memorized and therefore only approximate the responses of optimization computations. Second, the networks do not observe all state variables in the environment exactly; they observe mappings of those state variables through a sensory system. Third, the temporal horizon, or the amount of planning foresight granted during training, is less than the total duration of typical trials. Fourth, and most important, the controller networks are trained on separate cost functions and then coupled. We therefore compare the results obtained from the hierarchical network with those from an optimal control calculation. It should be stressed that the optimal control approach is not a practical way to solve the truck problem we considered because it is far too slow without speed tweaks and, as we will see, it fails to find reasonable paths a fair fraction of the time. Nevertheless, comparing paths produced by the hierarchical controller, at an expense of tens of milliseconds per path, with paths constructed by an optimization program over many seconds per path provides a way to judge the success of the hierarchical approach.

To compare our results with those of an optimal control calculation, we generated 150 random environments and computed solution trajectories from identical initial conditions using either the network hierarchy or an optimal control calculation computed by differential dynamic programming, a commonly used algorithm for hard optimal control calculations. We compared the two solutions using the nonconstraint portions of the cost function used in the optimal control calculation (see the appendix). Note that this means that we are judging the hierarchical model using a cost function that was not used in its construction.

Figure 7A shows a comparison of paths generated by the hierarchical network and the optimal control algorithm. Many of the paths are quite similar. Other paths are clearly equivalent, although they differ by going around opposite sides of an obstacle. In some cases, the optimal control algorithm has clearly failed to find a good solution, and it produces paths with loops in them. These cases appear as a long negative tail on the distribution of cost-function differences shown in Figure 7B. Importantly, there are very few cases in which the cost of the solution provided by the

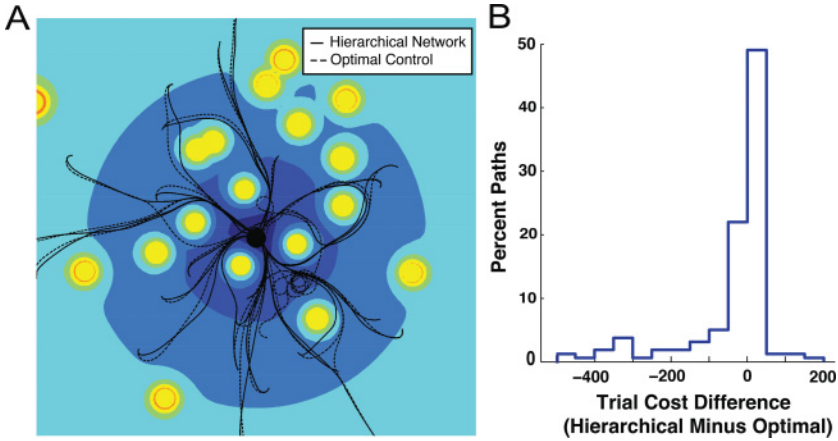

Figure 7: Comparison with optimal control. (A) In 25 trials in a single environment, we can see that the hierarchical controller's (continuous line) and optimal control solver's (dashed line) trajectories often overlap. On some trials, the trajectories diverge symmetrically around an obstacle. On other trials, the optimal control solver finds poor local minima in which the trajectories develop loops. This behavior is extremely rare for the hierarchical controller. (B) The per-trial cost differences over 150 trials in random environments show that the hierarchical controller infrequently performs worse than the optimal control solver (positive tail in the distribution), often performs equivalently (peak near 0), and sometimes performs much better when the optimal control calculation fails (negative tail). The average cost per path for either scheme is approximately 300.

hierarchical network is significantly worse than that of the optimal control path (the small positive tail in Figure 7B). Our conclusions are that the majority of paths constructed by the hierarchical network come close to being optimal with respect to the cost function used for this comparison and that the hierarchical model produces fewer and less disastrously bad paths when it fails to approximate optimality.

2.2 The Training Procedure. The controllers at each level of the hierarchy work because they have been trained to generate commands (subtargets or wheel angles for the truck) that are approximately optimal for the input they are receiving at a given time. Recall that this input consists of the subtarget received from the upstream controller and whatever sensory information is provided. The coupling between the two levels of the hierarchy requires an extension of the methodology of model-based optimal control. The key modification has two connected parts. Input from the higher level not only acts as a command but also serves to specify the cost

function that the lower-level controller is trained to minimize. As a result, the lower-level controller is trained to minimize a family of cost functions parameterized by the value of the higher-level command. The higher-level control problem is to choose a sequence of higher-level commands to send to the lower-level controller that will minimize a higher-level cost function. To find the appropriate higher-level command, a higher-level forward model is trained to predict the feedback that will result from propagating commands. Because the higher-level optimization acts on the higher-level forward model, which is trained only after the lower-level controller has been trained, the optimization problems at both levels are decoupled. This is a virtue because we do not need to execute nested optimizations to train the higher-level controller.

Tasks like the one we consider are difficult because a significant amount of time may elapse before the cost associated with a particular command strategy can be determined. In our lower-level example, it takes a while for the truck to move far enough to reveal that the wheels are not at a good angle. Typically, as tasks get more complex, this delay gets longer. For example, it takes longer to evaluate whether a subtarget issued by the higher-level controller is going to get the truck closer to the final target without leading it into an obstacle. One consequence of this delay is that we cannot assess the cost associated with a single command; we must evaluate the cost of a sequence of commands. This requires that we predict the consequences of issuing a command, which we do at each level using a forward model. Once we have minimized the cost function by choosing an optimal sequence of commands, we train a controller network at each level to memorize the optimal commands given particular inputs.

To deal with the hierarchy of timescales associated with a hierarchy of control levels, we introduce two timescales per level. The first is associated with the temporal scale over which a process needs to be controlled. There is no point in issuing commands that change more rapidly than the dynamics of the object being controlled. In the truck example, wiggling the wheels back and forth rapidly is not an intelligent way to drive the truck, and swinging the subtarget around wildly is not a good way to guide the lower-level controller. At level l of the hierarchy, we call this dynamic timescale T_l. In general, the choice of T_l is governed by the dynamics of the system being controlled by level l. For the truck problem, we take $T_1 = 6$ and $T_2 = 72$ time steps.

In optimal control problems, the cost of a command is considered only in the context of an entire sequence of commands. What is good to do now depends on what will be done later. The second timescale is therefore the length of the sequence of commands used to compute the cost of a trajectory. At level l, we denote this number by K_l. In other words, it requires K_l commands, spaced apart by T_l time steps, to determine the cost of a particular command strategy. In the case of the truck, we set $K_1 = 15$ and $K_2 = 10$. An explanation of these particular settings is provided in the appendix.

2.2.1 The Cost Functions. We denote a command given at time t by the level l controller by the vector $\mathbf{m}_l(t)$. For the lower-level controller in the truck example, $\mathbf{m}_1(t) = u(t)$, and for the upper-level controller $\mathbf{m}_2(t) = [d_{\text{st}}(t), \cos(\theta_{\text{st}}(t)), \sin(\theta_{\text{st}}(t))]$. We also define a vector $\mathbf{s}_l(t)$ that represents the sensory input to layer l at time t on which the decision to issue the command sequence is based. For the case of the truck, $\mathbf{s}_1(t) = [\cos(\theta_{\text{rel}}), \sin(\theta_{\text{rel}}), d_{\text{st}}, \cos(\theta_{\text{st}}), \sin(\theta_{\text{st}})]$ and $\mathbf{s}_2 = [\cos(\theta_{\text{rel}}), \sin(\theta_{\text{rel}}), \mathbf{g}, \log(1 + d_{\text{ft}}/L), \cos(\theta_{\text{ft}}), \sin(\theta_{\text{ft}})]$ (see Figure 1).

The cost of a sequence of commands $[\mathbf{m}_l(t), \mathbf{m}_l(t + T_l), \mathbf{m}_l(t + 2T_l), \ldots, \mathbf{m}_l(t + K_l T_l)]$ takes the general form

$$\mathcal{S}_l = \sum_{k=0}^{K_l} \mathcal{L}_l\big(\mathbf{s}_l(t + (k+1)T_l), \mathbf{m}_l(t + kT_l); \mathbf{m}_{l+1}(t)\big). \tag{2.1}$$

The functions $\mathcal{L}_l$ for the lower- ($l = 1$) and higher- ($l = 2$) level controllers are specified below. It is important to observe that the cost function at level l depends parametrically on the command from the level above, $\mathbf{m}_{l+1}(t)$. This command is provided at time t and, during training, is fixed until time $t + K_1 T_1$.

We would like the truck to drive toward the target along a straight angle of attack, without articulating the link between the cab and trailer too much and using minimal control effort. A cost function satisfying these requirements can be constructed from

$$\mathcal{L}_1 = \alpha_1 d_{\text{st}} + \beta_1 \theta_{\text{st}}^2 + \gamma_1 (|\theta_{\text{rel}}| - \theta_{\max})^2 \Theta\left(|\theta_{\text{rel}}| - \theta_{\max}\right) + \zeta_1 u^2. \tag{2.2}$$

The third term makes use of the Heaviside step function, which is 1 if $x \geq 0$ and 0 otherwise. We use the convention that all angles are in radians, and we center all angles around 0 so they fall into the range between $\pm\pi$. The parameters α_1, β_1, γ_1, ζ_1, and $\theta_{\max}$ are given in Table 1 in the appendix.

The higher-level cost function is divided into three parts: $\mathcal{L}_2 = \mathcal{L}_2^{\text{sensory}} + \mathcal{L}_2^{\text{command}} + \mathcal{L}_2^{\text{obstacle}}$. All the parameters in these cost functions are given in Table 1. The sensory cost contains a distance-dependent term, but we no longer need to penalize large cab-trailer angles because the lower-level controller takes care of this on its own, so

$$\mathcal{L}_2^{\text{sensory}} = \alpha_2 \log\left(1 + d_{\text{ft}}/L\right). \tag{2.3}$$

The higher-level motor command portion of the cost is given by

$$\mathcal{L}_2^{\text{command}} = \beta_2 \left(\cos(\theta_{\text{st}})^2 + \sin(\theta_{\text{st}})^2 - 1\right)^2 + \gamma_2 \Big(\left(d_{\text{st}} - d_{\min}\right)^2 \Theta\left(d_{\min} - d_{\text{st}}\right) + \left(d_{\text{st}} - d_{\max}\right)^2 \Theta\left(d_{\text{st}} - d_{\max}\right)\Big). \tag{2.4}$$

The first term in this equation may look strange because the sum of the squares of a cosine and a sine is always 1. However, the optimization procedure does not generate an angle θ_{st} and take its cosine and sine. Instead, it generates values for the cosine and sine directly without any constraint requiring that these obey the laws of trigonometry. As a result, this constraint needs to be included in the cost function. The distance-dependent terms in equation 2.4 penalize subtarget distances that are either too short or too long.

The final term in the higher-level cost function, $\mathcal{L}_2^{\text{obstacle}}$ penalizes truck positions that are too close to an obstacle. We are not concerned with the distance from the truck to every single obstacle; rather, we care primarily if the truck is too near a single obstacle, the closest one. Thus, we choose the smallest distance to an obstacle, $d_{\min}^{\text{obstacle}}(t)$ and impose a gaussian penalty for proximity to this closest obstacle with a standard deviation equal to the disc's radius, σ_{disc}. We also add a smaller, flatter penalty with a larger standard deviation, σ_{areola}, as a warning signal to prevent the truck from wandering near the obstacle. The resulting cost function is

$$\mathcal{L}_2^{\text{obstacle}} = \rho_2 \exp\left(-\frac{\left(d_{\min}^{\text{obstacle}}\right)^2}{2\sigma_{\text{disc}}^2}\right) + \nu_2 \exp\left(-\frac{\left(d_{\min}^{\text{obstacle}}\right)^2}{2\sigma_{\text{areola}}^2}\right). \tag{2.5}$$

Recall that the obstacles are detected by the sensory grid system shown in Figure 4, and thus the distances to obstacles are not directly available. We solve this problem by introducing a network that evaluates the cost function 2.5 directly from the sensory grid information $\mathbf{g}(t)$. We call this network the obstacle critic because it serves the same role as critic networks in reinforcement learning (Widrow, Gupta, & Maitra, 1973; Sutton & Barto, 1998): predicting the cost of sensory data, ultimately to train another network. It has 199 grid inputs and one bias and a single output, representing the estimated cost of the sensor reading (see the appendix). We train this network to predict the obstacle cost from grid data by creating a large number of measurement scenarios and computing the true cost function.

The complete higher-level cost function is the sum of the costs given in equations 2.3 to 2.5. The trajectories that minimize the complete higher-level cost function are goal seeking and obstacle avoiding.

2.2.2 The Forward Models. In equation 2.1, the cost function depends not only on the sequence of commands but also on the entire sequence of sensory consequences of those commands. In other words, $\mathcal{S}_l$ depends on $\mathbf{s}_l(t) \ldots \mathbf{s}_l(t + K_l T_l)$, but only $\mathbf{s}_l(t)$ is provided. To predict the future sensory data resulting from the command sequence, we build a forward model at each level of the hierarchy. To do this, we sample the space $(\mathbf{s}_l(t), \mathbf{m}_l(t))$ and record the resultant states $\mathbf{s}_l(t + T_l)$. We then build a network that predicts the resultant sensory data.

The forward model for level l predicts the future sensory data $\mathbf{s}_l(t + T_l)$ that result from passing the command $\mathbf{m}_l(t)$ to the level below it. We use the convention that the forward model is named after the level that issues the commands and receives the sensory data, not the level that follows those commands and causes the sensory data to change. Thus, the higher-level forward model is actually modeling the sensory consequences of sending a command to the lower-level controller, and the lower-level forward model is modeling the consequences of sending a command to the truck. Because we have a full kinematic description of the truck, we could use the truck itself, rather than a forward model of it, to optimize the lower-level cost function, though this solution would be limited to simulations with perfect knowledge of the truck dynamics. Nevertheless, we use a forward model so that we can treat and discuss the lower and higher levels in a similar manner. The higher-level forward model could also be eliminated in three ways. First, we could resort to a reinforcement learning method based on noise injection to detect correlations between propagated subgoals and resultant higher-level trajectory costs. This is likely to scale poorly for long sequences of subgoals. Second, we could optimize sequences of subgoals using an expensive finite difference calculation. Third, we could optimize directly through the lower-level controller and the environment by backpropagating at the lower-level timescale. However, by using the higher-level forward model, we can optimize at a much coarser timescale and also decouple the optimization entirely from execution of the lower-level controller. This interesting property implies that long-term planning can be done without operation or minutely detailed simulation of lower-level motor systems and is a recognizable feature of human planning.

The lower-level forward model, a single network, is trained by randomly choosing a set of sensory data, $\mathbf{s}_1(t)$, consisting of the distance and cosine and sine of the angle to a target, and a command $\mathbf{m}_1(t)$, defining a wheel angle, within their allowed ranges. We then simulate the motion of the truck for a time T_1 and determine the sensory data $\mathbf{s}_1(t + T_1)$, indicating the articulation of the cab with respect to the trailer and where the truck lies in relation to the subgoal. We continue to gather data for the lower-level forward model by applying another command to the truck and taking a new sensory measurement after the delay. In this way, we generate several command sequences along a single trajectory to gather more data. To create a diversity of training cases for the forward model, we periodically terminate a trajectory and start a new trial from a random initial condition. On the basis of several thousand such measurements, we train the lower-level forward model network to predict the motion and articulation of the truck over the full range of initial conditions and motor commands.

The higher-level forward model is composed of three networks: proprioceptive, goal related, and obstacle related (see the appendix). Each of these networks receives the command $\mathbf{m}_2(t)$. The proprioceptive network also receives the proprioceptive information, $[\cos(\theta_{\text{rel}}(t)), \sin(\theta_{\text{rel}}(t))]$,

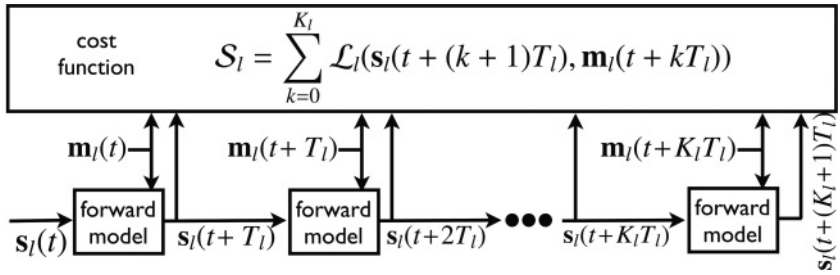

Figure 8: Schematic showing the iterated use of the forward model for computing an optimal command sequence. The system is provided with sensory input at time t through the vector $\mathbf{s}_l(t)$. The forward model is used repeatedly to generate predictions of the sensory vector at times $t + T_l, \ldots, t + (K_l + 1)T_l$. The command sequence $\mathbf{m}_l(t), \mathbf{m}_l(t + T_l), \ldots, \mathbf{m}_1(t + K_lT_l)$ is an input to both the cost-function computation and the forward model, and it is optimized.

and predicts $[\cos(\theta_{\text{rel}}(t + T_2)), \sin(\theta_{\text{rel}}(t + T_2))]$. The goal-related forward model receives these proprioceptive data and the goal-related information, $[\log(1 + d_{\text{ft}}/L), \cos(\theta_{\text{ft}}), \sin(\theta_{\text{ft}})]$, and predicts the goal-related variables at time $t + T_2$. The obstacle-related forward model is the most complicated. It receives the proprioceptive information and the obstacle grid data $\mathbf{g}(t)$. Unlike the other forward model networks we have described, which make deterministic predictions, the predictions of the goal-related forward model are probabilistic. This is necessary because the grid predictions are underdetermined. For example, at time t, the grid input cannot provide any information about obstacles outside its range, but one of these may suddenly appear inside the grid at time $t + T_2$. The goal-related forward model cannot predict such an event with certainty. Therefore, we ask the obstacle-related network to predict the probability that each grid point will be occupied at time $t + T_2$, given a particular grid state and command issued at time t. This is obviously a number between 0 and 1, in contrast with the true value of $g_i(t + T_2)$, which would be either 0 or 1. We explain how this is done in the appendix.

When we use forward model networks to predict sensory information along a trajectory, $\mathbf{s}_l(t), \mathbf{s}_l(t + T_l), \mathbf{s}_l(t + 2T_l), \ldots$, we simply iterate using the predicted sensory data time $t + kT_l$ to generate a new prediction at $t + (k + 1)T_l$. This allows us to compute the summed cost functions of equation 2.1 for both controller levels (see Figure 8).

2.2.3 Computing Optimal Commands. Our procedure for generating optimal command sequences is schematized in Figure 8. We begin by initializing states of the environment and truck randomly (within allowed ranges) and measure $\mathbf{s}_l(t)$. We then choose an initial (or nominal) sequence of

commands $[\mathbf{m}_l(t), \mathbf{m}_l(t + T_l), \ldots, \mathbf{m}_1(t + K_l T_l)]$. For the lower-level controller, we choose $\mathbf{m}_1(t + kT_1) = u(t + kT_1) = 0$ for all k, indicating that pointing the wheels straight is our first guess for the optimal command. For the higher-level controller, we choose the initial command sequence to consist of identical commands placing a subtarget 50 length units directly behind the truck. We apply these commands sequentially to calculate an entire truck trajectory.

The command sequences are then optimized by using the dynamic optimization algorithm described in the appendix—also known as Pontryagin's minimum principle (Stengel, 1994) or backpropagation through time (LeCun, 1988). Briefly, we calculate the effect that a small change to each command will have on the total trajectory cost and determine the gradient of the cost function with respect to the parameters defining the commands. This gradient information is used to change the commands according to a variable-metric update accomplished with the limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) method in "minFunc" (Schmidt, 2013). L-BFGS uses successive gradient computations to compute an approximation of the inverse Hessian of the cost function, which speeds up optimization considerably compared to steepest descent. We iterate the optimization until performance does not improve (more details are provided in the appendix).

The above procedure generates a single command sequence that is optimal for the initial sensory data $\mathbf{s}_l(t)$. However, we can use the sensory data after the first command has generated its sensory consequences (at time $t + T_l$) to optimize a new trajectory starting from the new sensory data. This process can obviously be repeated at subsequent times. We periodically terminate the trajectory and begin a new trial to obtain sufficient variety in the initial sensory data. When this process is completed, the pairings of initial sensory data and optimal commands constitute a training set for the controller being trained, which learns to associate each sensory measurement with the appropriate command.

Computing an optimal trajectory of K_1 steps at the lower level takes on average 1.3 seconds on our computer (averaged over 100 such optimizations). Computing an optimal trajectory of K_2 steps at the higher level takes on average 12.47 seconds (averaged over 10 such optimizations). Once the lower- and higher-level controllers have been trained, only 0.001 seconds is needed to run the two controllers in series. Clearly, memorization provides a tremendous advantage in speed over online optimization.

2.2.4 Training the Networks. Having drawn random initial sensory data $\mathbf{s}_l(0)$ and solved the optimal control problem defined by equation 2.1, subject to the dynamics of the forward model, we then train the controller network to predict the first motor command in the sequence from the sensory data: $\mathbf{m}_l(0)$ from $\mathbf{s}_1(0)$. We do not use the full sequence of optimal commands for training the controller; we use the full sequence only to

evaluate the cost function. The rationale is that all the future commands beyond the first one in the sequence are based on sensory inputs predicted by the forward model. Training the controller to associate these predicted sensory states with their paired commands in the sequence introduces errors into the controller because the sensory predictions of the forward model are not entirely accurate.

The training procedure for each controller is straightforward. We apply a particular input to the controller network and use backpropagation to modify its parameters so as to minimize the squared difference between the output command given by the controller and the optimal command for that particular set of inputs. After a sufficient number of such trials (see the appendix), the controller network learns to produce the desired command in response to a particular input. Furthermore, if the network is properly designed, it will generalize to novel inputs by smoothly interpolating among the trained examples.

3 Discussion

3.1 Biological Evidence for Hierarchical Control and Forward Models. Although our networks are certainly not models of specific biological neural systems, we cannot resist drawing various connections to physiology. Biological evidence for hierarchical control is diverse, although we lack a clear understanding of its architectural logic. Lashley (1930) and Bernstein (1967) independently advocated hierarchical theories of motor control after reasoning from psychophysical experiments and intuition. Lashley's principle of motor equivalence paralleled Bernstein's degrees of freedom problem: namely, there is more than one movement that will accomplish a task goal. It is, for example, possible to write with one's left or right hand. Raibert (1977) studied the problem of cursive writing while varying properties of the utilized effector. He wrote, "Able was I ere I saw Elba," using both hands, with an immobilized wrist, and even with his teeth, demonstrating a relative invariance of the basic form of the orthography, suggesting that musculoskeletal control is only loosely coupled to the problem of goal-directed planning.

Anatomically, of course, projections from motor cortex to the spinal cord are hierarchical in that control of muscle contraction is indirect (Loeb, Brown, & Cheng, 1999). This was perhaps first grasped by Hughlings Jackson in 1889 (Jackson, 1889). Therefore, from the perspective of cortical control centers, the problem of movement corresponds to the problem of sending commands to the spinal cord in such a way that will satisfy higher-level intentions. In the octopus, large networks of ganglia, interposed between the central nervous system and the arm muscles, are themselves necessary and sufficient for the production of complicated movements; when the axial nerve cords of the peripheral nervous system are stimulated electrically, severed arms propagate a bend that results in qualitatively normal

extension (Sumbre, Gutfreund, Fiorito, Flash, & Hochner, 2001). Analogously, in zebra finch singing behavior, the nucleus HVc controls song timing through its projections to the motor region RA, which is ultimately responsible for the control of syllable vocalization by syringeal muscles (Albert & Margoliash, 1996).

A long-standing hypothesis is that the cerebellum implements a forward model that predicts the delayed sensory consequence of a motor command (Miall, Weir, Wolpert, & Stein, 1993; Shadmehr, Smith, & Krakauer, 2010). Intriguingly, a recent study suggests that neurons in Clarke's column in the spinal cord simultaneously receive input from cortical command centers and from proprioceptive sensory neurons, and the authors speculate that this joining of sensory and motor inputs could function as a predictor, akin to a forward model (Hantman & Jessell, 2010). Experimentalists have therefore already proposed multiple loci for forward models, functioning at different levels of the motor system. This connection suggests that motor behavior may be produced by hierarchies of controllers that are trained by forward models at each level of the hierarchy.

3.2 Relationship to Other Approaches. Research on hierarchical control is dispersed among several different fields with varying agendas and, in fact, conceptions of the word *hierarchy*. Robotics has embraced hierarchical mechanisms wholeheartedly, and most current systems for complicated tasks are now hierarchical. A typical mobile robot will segregate low-level control from planning and navigation, a functional division that we have borrowed. This is the case in Stanley, the self-driving car that won the DARPA Grand Challenge (Thrün et al., 2006), which was modeled on the three-layer architectures of Gat (1998). In these architectures, however, planning is entirely separated from lower-level motor control; in particular, higher levels do not model the result of commanding lower levels, so no notion of feasibility is available to the higher-level system. As a workaround, the navigational models in these systems are heuristically instructed to generate paths that are smooth and free from obstruction. We feel that endowing higher-level systems with explicit models of the results of propagating commands to lower levels can make robots more flexible and better able to cope with or exploit the particularities of their lower-level controller and plant dynamics.

Hierarchical reinforcement learning (Dietterich, 1998; Barto & Mahadevan, 2003; Sutton, Precup, & Singh, 1999) considers a different notion of hierarchy, which we might term a *recursive hierarchy*. Whereas our notion of hierarchy is structural and embedded in a hierarchy of circuits, hierarchical reinforcement learning systems are representatives of the same form of hierarchy as procedural programs—procedures that call subprocedures recursively. The procedures in hierarchical reinforcement learning correspond to tasks, which can be recursively decomposed into subtasks. Termination conditions within each subtask return the system back to the caller task. Our

major borrowing from this literature is the idea of a subtask or subgoal, but our implementation of this idea is only superficially related to the models in reinforcement learning.

Mathematically, our design procedure is closest to work in inverse optimal control. This field concerns itself with the problem of inferring the cost function that an observed agent is obeying (Ratliff et al., 2009; Abbeel & Ng, 2004; Kalakrishnan, Pastor, Righetti, & Schaal, 2013). Within inverse optimal control, work connected to bilevel programming (Albrecht, Sobotka, & Ulbrich, 2012; Mombaur, Truong, & Laumond, 2010) is the closest to ours, as it explicitly considers the question of how to choose parameters for a lower-level cost function to minimize a higher-level cost function. In most studies using inverse optimal control, the higher-level cost function is a measure of deviation between the experimentally measured movements of a subject and the outputs of an optimal control model optimizing the lower-level cost function, defined by the higher-level parameters. The aim of this work is different from ours in that the goal is to understand the behavior of an experimental subject. Additionally, in implementation, these inverse optimal control studies have not used higher-level forward models to decouple the optimization across levels or memorized the results of optimization to construct feedback controllers.

Other work on hierarchical control in the context of theoretical modeling of human motor control has sought to reduce the dimensionality of the state space, sending the higher level a compressed description of the state of the plant to simplify the computation of optimal commands (Liu & Todorov, 2009). While this may often prove useful, it is orthogonal to our approach. In our model, the state dimensionality at the higher level is in fact almost two orders of magnitude larger than at the lower level because it includes new sensory data unavailable to the lower level. This massive dimensionality expansion is, for example, consistent with the observation that the visual and motor information for visuomotor control is first merged cortically, even though reflexive movements, for example, triggered by noxious stimuli, are processed entirely within the spinal cord. Thus, higher-level control can serve roles beyond dimensionality reduction and can take into account altogether new channels of information.

Work on dynamical movement primitives (Ijspeert, Nakanishi, Hoffmann, Pastor, & Schaal, 2013; Schaal, Peters, Nakanishi, & Ijspeert, 2005) aims to construct lower-level control systems obeying attractor or limit cycle dynamics that simplify the production and planning of smooth, feedback-sensitive movements. In this research program, a canonical low-dimensional dynamical system with intrinsic dynamics generates control outputs, and the goal is specified by sending a set of parameters to the movement primitive that specify the end goal, for example. This work is quite closely connected to our own in its purpose. The primary difference is that work in this field studies a somewhat limited set of dynamical systems of a specific kind, and the higher-level control of these systems has not

used higher-level forward models but instead sampling-based reinforcement learning and supervised learning from demonstration.

The idea of controlling a physical system by building a model of it is quite old, entering the connectionist literature with Nguyen and Widrow (1989) and Jordan and Rumelhart (1992) but existing in a prior incarnation as model-reference adaptive control. More recently, model-based control has been identified as a very data-efficient means to train controllers (Deisenroth & Rasmussen, 2011). Neural networks have also received renewed attention in recent years as generic substrates for feedback controllers (Huh & Todorov, 2009; Sutskever, 2013). To the best of our knowledge, the idea of using a network to model the feedback from another network and then to use that model to control the modeled network is novel. We expect it to have ramifications that exceed the traditional framework of motor control. Forward models could be constructed by either motor babbling or a more intelligent experimentation procedure. Once the controllers have been trained, they exhibit automaticity, an ability to generate answers without extensive computation. Automating complex computations by caching has been proposed before (Kavukcuoglu, Ranzato, & LeCun, 2008; Dayan, 2009) in different contexts, but using cached circuits as lower-level substrates for higher-level control circuits has not.

4 Conclusion

Our work consists of three innovations: (1) dividing the task hierarchically by designing a higher level that propagates cost-related information to a lower level, (2) introducing higher-level forward models to train higher-level controllers, and (3) caching computed optimal commands in network controllers.

Apart from navigation problems, the hierarchical scheme and training procedure using forward models should be useful for tackling any problem requiring simultaneous application of sensorimotor and cognitive skills. Consider, for example, a robotic gripper moving blocks. It would be quite a challenge to construct a unitary network that directs the gripper to pick up, move, and drop blocks and that decides how to arrange them into a prescribed pattern at the same time. It is easier to separate the problem into a manual coordination task and a puzzle-solving task, and a natural vehicle for this separation is the network architecture itself. More generally, the approach we have described is suited to problems that can be formulated and solved by division into easier subproblems. Many interesting tasks have this structure, so this strategy should be quite widely applicable. Surprisingly, despite the intuitiveness of this idea, it has received little attention.

The construction of the hierarchy is recursive or self-similar. This makes training considerably easier because each controller in the hierarchy is basically doing the same thing: receiving and issuing commands describing goals. Thus, we can apply the same training procedure at each level.

Another advantage of this approach is that lower-level controllers do not need to be retrained if the overall task changes. In addition, the hierarchy can be extended by adding more levels if the task gets more difficult, again without requiring retraining of the lower levels.

Although we have trained the forward model and controller networks sequentially, we do not preclude the attractive possibility of training them at the same time, maybe even at multiple levels of the hierarchy simultaneously. To do so, we expect it would be crucial to account for errors in the forward models; we could generate cautious commands by penalizing those commands that a forward model is unlikely to predict accurately, or we could also generate commands that attempt to probe the response properties of the level below to improve the model.

This study presents a hard result, a hierarchical neural network, and a method for training it based on higher-level forward models; perhaps equally important, it bolsters a soft view. To create intelligent behaviors from neuron-like components, we need to embed those neurons in modules that perform specific functions and operate to a large extent independently. These modules should have protocols for interfacing one to another, protocols that effectively hide the complexity of the full computation from the constituents. In experimental neuroscience, studying how such modules interact may require us to move beyond single-area recordings to understand the causal interactions between connected regions and identify the goals of the computations performed by each region.

Appendix: Methods

A.1 Parameters. In Table 1, the lower-level dynamic timescale T_1 was chosen to be as many time steps as possible without causing the emergence of jackknifing of the truck during optimization. The length of the lower-level control sequence K_1 was chosen to be relatively short, subject to the requirement that the lower-level controller needed to be able to turn around and approach a nearby target. The performance of the lower-level controller was not strongly sensitive to these particular values for K_1 and T_1. The upper-level dynamic timescale T_2 was chosen to be shorter than the amount of time it takes for the truck to cross directly through an obstacle. Otherwise the optimization could command the truck through an obstacle, but because the forward model is showing only a before-and-after snapshot of the location of the truck, the whole system would be blind to its error. The length of a higher-level command sequence K_2 was chosen so that a truck that drives straight for K_2T_2 time steps would not reach beyond the edge of the obstacle grid observed at time $t = 0$. That is, the sensory system contains most of the information needed for planning K_2T_2 steps. Again, small variations in these numbers do not change the behavior of the system significantly, but the numbers are intelligently chosen to make the system work as well as possible.

Table 1: Parameters for the Truck and Obstacles, the Lower- and Higher-Level Cost Function, and the Lower- and Higher-Level Training.

Truck and Obstacles		Lower-Level Cost Function		Higher-Level Cost Function		Lower-Level Training		Higher-Level Training	
Parameter	Value	Parameter	Value	Parameter	Value	Parameter	Value	Parameter	Value
r (speed)	0.2	α_1	1	α_2	1	T_1	6	T_2	$12 \cdot T_1$
l_{cab} (length)	6	β_1	0.5	β_2	100	K_1^{model}	15	K_2^{model}	1
w_{cab} (width)	6	γ_1	0.5	γ_2	100	$K_1^{\text{optimization}}$	15	$K_2^{\text{optimization}}$	10
l_{trailer}	14	ζ_1	0.5	$d_{\min}$	0.1	K_1^{total}	30	K_2^{total}	10
w_{trailer}	6	$\theta_{\max}$	$\frac{\pi}{2} - \frac{\pi}{6}$	$d_{\max}$	0.9	$N_{c,1}$	2×10^4	$N_{c,2}$	5×10^5
σ_{disc} (radius)	20			σ_{disc}	20	$N_{\text{fm},1}$	5×10^4	$N_{\text{fm},2}$	2×10^6
				σ_{areola}	30				
				ρ_2	2				
				ν_2	0.5				

Note: $N_{c,l}$ denotes the number of example patterns used to train the controller and $N_{\text{fm},l}$ the number for the forward model. K_l^{total} refers to the number of steps taken when optimizing trajectories before restarting the truck in a new environment.

Table 2: Architectures of the Networks.

Network	Dimensions and Activation Functions
Lower-level forward	$6 \times$ [G]150 $\times$ [L]5
Lower-level controller	$5 \times$ [G]100 $\times$ [L]1
Higher-level forward (proprio.)	$5 \times$ [T]40 $\times$ [T]20 $\times$ [T]20 $\times$ [L]2
Higher-level forward (goal)	$8 \times$ [T]40 $\times$ [T]20 $\times$ [T]20 $\times$ [L]3
Higher-level forward (obstacle)	$205 \times$ [T]300 $\times$ [T]200 $\times$ [T]200 $\times$ [T]300 $\times$ [Si]199
Higher-level critic (obstacle)	$199 \times$ [T]300 $\times$ [T]100 $\times$ [T]100 $\times$ [So]1
Higher-level controller	$205 \times$ [So]30 $\times$ [So]20 $\times$ [So]30 $\times$ [L]3

A.2 Network Structure and Training. The lower-level makes use of radial basis function (RBF) neural networks. RBF networks possess a strong bias toward generating smoothly-varying outputs as a function of their inputs and can train very quickly on low-dimensional input data, two qualities that are useful at the lowest level of the hierarchy. The functional targets for the higher-level networks were more complicated and higher-dimensional, so we had to develop "deep" (or many-layered) networks to approximate them accurately.

The architectures of the networks are shown in Table 2. The bracketed letters indicate the activation functions used for the units. [G] is a normalized gaussian RBF for input x,

$$\frac{\exp\left(-||\mathbf{x}-\mu_i||^2/(2\sigma_i^2)\right)}{\sum_j \exp\left(-||\mathbf{x}-\mu_j||^2/(2\sigma_j^2)\right)},$$

with basis function centers μ_i and standard deviations σ_i. The RBF centers were chosen by randomly selecting exemplars from the input data. They were not further adapted in training. The RBF standard deviations were initialized to scale linearly with the number of input dimensions. To avoid division-by-zero, we computed using the inverse standard deviations $1/\sigma_i$.

All the other multiplication signs in Table 2 imply matrix multiplication followed by an activation function. [L] is a linear activation function, [T] is $\tanh(x)$, [Si] is a logistic sigmoid $1/(1+\exp(-x))$, and [So] is a "soft-rectification" function $\log(1+\exp(x))$. We chose these activation functions using a mixture of prior knowledge and experimentation. The soft rectification in the obstacle critic imposed the constraint that costs are positive. The logistic function in the obstacle grid forward model bounds the outputs between 0 and 1 and allows us to interpret them as the probabilities that the grid points will be occupied. The soft-rectification in the higher-level controller alleviated overfitting.

A weight matrix W from a layer of size M to another layer was initialized to have independent gaussian entries of mean 0 and standard deviation

$1/\sqrt{M}$. This makes the networks balanced in that the mean input to every unit is 0 and the response variance of all units is self-consistently $\mathcal{O}(1)$. This ensures a constant response variance from one layer to the next. All multilayer networks included a single additional bias unit in their inputs.

We chose to interpret an obstacle grid model output as the probability that the corresponding grid element would be occupied after movement. We consequently used the cross-entropy cost function to train this network. We can derive this cost function from a maximum-likelihood argument. Suppose we have a data set of patterns $(\mathbf{x}_k, \mathbf{y}_k)_{k=1}^{N_{\text{patterns}}}$ and a neural network with M outputs whose ith output unit as a function of the kth input vector given by $z_i(\mathbf{x}_k)$ represents the binomial probability that $y_i(\mathbf{x}_k) = 1$. The likelihood of the data is binomial, so the log likelihood is

$$\sum_{k=1}^{N_{\text{patterns}}} \sum_{i=1}^{M} y_i(\mathbf{x}_k) \log z_i(\mathbf{x}_k) + (1 - y_i(\mathbf{x}_k)) \log(1 - z_i(\mathbf{x}_k)).$$

When negated, this gives the cross-entropy cost function.

Optimization of the network parameters was accomplished using batch training with the quasi-Newton optimization method L-BFGS in minFunc (Schmidt, 2013). To train the lower-level forward model or the lower-level controller, at the beginning of each trial a random angle θ_{st} was drawn along with a random distance d_{st} in the range between 0 and 500. The trailer angle θ_{trailer} was similarly drawn uniformly. The cab angle was initialized to be within $\pm(\pi/2 - \pi/64)$ radians of the trailer angle.

The higher-level proprioceptive and goal-related models were trained to predict not the values of their targets but the difference between the values of their targets before and after movement. This reduced training time because the interesting predictions of many forward models are the deviations from the identity.

A.3 Equations for the Truck. The truck is a kinematic model of a cab and trailer (see Figure 9), first defined by Nguyen and Widrow (1989). The cab is connected to the trailer by a rigid linkage. The wheels are connected to the front of the cab and translate backward by distance r in one time step. Note that the wheels drive the cab the same way that the linkage drives the trailer, so we can solve for the motion of the cab and trailer in a similar way.

We begin by decomposing the motion of the front of the cab caused by the wheels into a component orthogonal to the front of the cab, defined as $A = r\cos(u(t))$, and a component parallel to the front of the cab, $C = r\sin(u(t))$. Only the orthogonal component, A, gets transferred through the linkage to the trailer. Performing a similar decomposition of the motion of the front of the trailer, we find an orthogonal component $B = A\cos(\theta_{\text{cab}}(t) - \theta_{\text{trailer}}(t))$ and a parallel component $D = A\sin(\theta_{\text{cab}}(t) - \theta_{\text{trailer}}(t))$. In one time step,

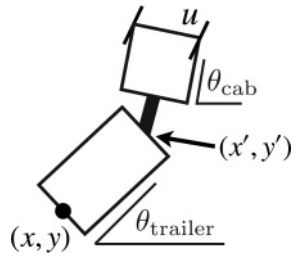

Figure 9: Variables describing the truck. The position of the center of the back of the trailer is given by (x, y), and the center of the front of the trailer is at (x', y'). The angle of the cab with respect to the x-axis is $\theta_{\text{cab}}(t)$ and for the trailer, $\theta_{\text{trailer}}(t)$. The angle by which the front wheels deviate from straight ahead is u, and the relative angle between the cab and trailer is $\theta_{\text{rel}} = \theta_{\text{cab}} - \theta_{\text{trailer}}$.

the center of the front of the trailer (see Figure 8) therefore moves by

$$x'(t+1) = x'(t) - B\cos(\theta_{\text{trailer}}(t)) + D\sin(\theta_{\text{trailer}}(t)),$$
$$y'(t+1) = y'(t) - B\sin(\theta_{\text{trailer}}(t)) - D\cos(\theta_{\text{trailer}}(t)).$$

The back of the trailer is constrained to move straight backward, so

$$x(t+1) = x(t) - B\cos(\theta_{\text{trailer}}(t)),$$
$$y(t+1) = y(t) - B\sin(\theta_{\text{trailer}}(t)).$$

If the length of the trailer is L_{trailer}, $x'(t) = x(t) + L_{\text{trailer}}\cos(\theta_{\text{trailer}}(t))$ and $y'(t) = y(t) + L_{\text{trailer}}\sin(\theta_{\text{trailer}}(t))$. The tangent of the angle of the trailer is equal to $(y' - y)/(x' - x)$, so at time $t+1$, we have

$$\tan(\theta_{\text{trailer}}(t+1)) = \frac{L_{\text{trailer}}\sin(\theta_{\text{trailer}}(t)) - D\cos(\theta_{\text{trailer}}(t))}{L_{\text{trailer}}\cos(\theta_{\text{trailer}}(t)) + D\sin(\theta_{\text{trailer}}(t))}.$$

An identical argument applied to the cab yields

$$\tan(\theta_{\text{cab}}(t+1)) = \frac{L_{\text{cab}}\sin(\theta_{\text{cab}}(t)) - C\cos(\theta_{\text{cab}}(t))}{L_{\text{cab}}\cos(\theta_{\text{cab}}(t)) + C\sin(\theta_{\text{cab}}(t))}.$$

Consolidating all of the equations, we have

$$A = r\cos(u(t)),$$
$$B = A\cos(\theta_{\text{cab}}(t) - \theta_{\text{trailer}}(t)),$$
$$C = r\sin(u(t)),$$

$$
\begin{aligned}
D &= A\sin(\theta_{\text{cab}}(t) - \theta_{\text{trailer}}(t)),\\
x(t+1) &= x(t) - B\cos(\theta_{\text{trailer}}(t)),\\
y(t+1) &= y(t) - B\sin(\theta_{\text{trailer}}(t)),\\
\theta_{\text{cab}}(t+1) &= \tan^{-1}\left(\frac{L_{\text{cab}}\sin(\theta_{\text{cab}}(t)) - C\cos(\theta_{\text{cab}}(t))}{L_{\text{cab}}\cos(\theta_{\text{cab}}(t)) + C\sin(\theta_{\text{cab}}(t))}\right),\\
\theta_{\text{trailer}}(t+1) &= \tan^{-1}\left(\frac{L_{\text{trailer}}\sin(\theta_{\text{trailer}}(t)) - D\cos(\theta_{\text{trailer}}(t))}{L_{\text{trailer}}\cos(\theta_{\text{trailer}}(t)) + D\sin(\theta_{\text{trailer}}(t))}\right).
\end{aligned}
$$

It is worth mentioning that $(x, y, \theta_{\text{cab}}, \theta_{\text{trailer}})$ is a four-dimensional state vector with a one-dimensional control variable, u. Control theorists call problems in which the control vector is lower dimensional than the state vector "underactuated"; such problems are typically more difficult than fully actuated problems because it may take more than one step to modify a given state variable, and it may be impossible to drive the system to an arbitrary point in the state-space (a concept known as *controllability*). The truck is also a nonlinear system and an unstable one because setting $u = 0$ amplifies any angular deviation of the cab from the trailer.

A.4 Minimizing the Cost Functions. In this section we describe how to minimize the cost functionals with respect to the command parameters. We are minimizing a cost functional of the form

$$\mathcal{S} = \sum_{k=0}^{K} \mathcal{L}(\mathbf{s}(k+1), \mathbf{m}(k)) \tag{A.1}$$

as in equation 2.1, but to streamline the notation, we have dropped the time t and the temporal scale factor T that appear in equation 2.1. This means that we have shifted the time variable to starting at time t and are measuring time in units of T. The original equations can be recovered by shifting and scaling back. We have also dropped the subscripts l because the same procedure is applied at each level.

The sensory vector $\mathbf{s}$ is estimated by a forward model, and we denote the output of the forward model by $\mathbf{F}(\mathbf{s}, \mathbf{m})$, so that $\mathbf{s}(k+1)$ is estimated as $\mathbf{F}(\mathbf{s}(k), \mathbf{m}(k))$. To implement this constraint, we introduce Lagrange multipliers for every component of $\mathbf{m}$ and at every moment in time and minimize

$$\mathcal{S}_{\text{constrained}} = \sum_{k=0}^{K} \mathcal{L}(\mathbf{s}(k+1), \mathbf{m}(k)) + \lambda(k)^{\top}(\mathbf{F}(\mathbf{s}(k), \mathbf{m}(k)) - \mathbf{s}(k+1)). \tag{A.2}$$

For convenience, we define $\lambda(K+1) = 0$.

Algorithm 1. Dynamic Optimization.

Input: Initial state $\mathbf{s}(0)$ and "nominal" control tape $\{\mathbf{m}(k)\}_{k=0}^{K}$
  repeat
    for k := 0 to K **do**
      $\mathbf{s}(k+1) := \mathbf{F}(\mathbf{s}(k), \mathbf{m}(k))$;
    end for
    $\lambda(K+1) := 0$;
    for k := K+1 down to 1 **do**
      $\lambda(k-1) := \mathcal{L}_{\mathbf{s}}(\mathbf{s}(k), \mathbf{m}(k-1)) + \mathbf{F}_{\mathbf{s}}(\mathbf{s}(k), \mathbf{m}(k))^{\top}\lambda(k)$;
      $\frac{\delta S_{\text{constrained}}}{\delta \mathbf{m}(k-1)} := \mathcal{L}_{\mathbf{m}}(\mathbf{s}(k), \mathbf{m}(k-1)) + \mathbf{F}_{\mathbf{m}}(\mathbf{s}(k-1), \mathbf{m}(k-1))^{\top}\lambda(k)$;
      Update $\mathbf{m}(k-1)$ using a gradient-based method (steepest descent, L-BFGS, etc.) provided with $\frac{\delta S_{\text{constrained}}}{\delta \mathbf{m}(k-1)}$.
    end for
  until convergence

The gradient of this function with respect to the sensory vector is

$$\frac{\delta \mathcal{S}_{\text{constrained}}}{\delta \mathbf{s}(k)} = \mathcal{L}_{\mathbf{s}}(\mathbf{s}(k), \mathbf{m}(k-1)) + \mathbf{F}_{\mathbf{s}}(\mathbf{s}(k), \mathbf{m}(k))^{\top}\lambda(k) - \lambda(k-1),$$

where the subscript $\mathbf{s}$ indicates a derivative with respect to that variable. At a minimum of the cost functional, we find the backward equation for the Lagrange multipliers:

$$\lambda(k-1) = \mathcal{L}_{\mathbf{s}}(\mathbf{s}(k), \mathbf{m}(k-1)) + \mathbf{F}_{\mathbf{s}}(\mathbf{s}(k), \mathbf{m}(k))^{\top}\lambda(k). \tag{A.3}$$

It is easy to find an extremum of the cost functional with respect to the sensory variables and the Lagrange multipliers. It is more difficult to minimize with respect to the command variables, where the relevant gradient is

$$\frac{\delta \mathcal{S}_{\text{constrained}}}{\delta \mathbf{m}(k)} = \mathcal{L}_{\mathbf{m}}(\mathbf{s}(k+1), \mathbf{m}(k)) + \mathbf{F}_{\mathbf{m}}(\mathbf{s}(k), \mathbf{m}(k))^{\top}\lambda(k).$$

This is done using Algorithm 1.

A.5 Differential Dynamic Programming Comparison. We used an implementation of differential dynamic programming (DDP) based on the description in Erez (2011). The DDP cost function was closely related to the two cost functions used in the hierarchical optimization, $\mathcal{L}_1$ and $\mathcal{L}_2$, but we had to modify a few of the terms because DDP appeared more sensitive to discontinuities of the derivatives of the original costs. Our overall cost function could be decomposed as $\mathcal{L}_{\text{DDP}} = \mathcal{L}_{\text{DDP}}^{\text{obstacle}} + \mathcal{L}_{\text{DDP}}^{\text{goal}} + \mathcal{L}_{\text{DDP}}^{\text{constraints}}$.

Here, $\mathcal{L}_{\text{DDP}}^{\text{constraints}}(\mathbf{s}_2) = u^2/2 + 20(1 - \cos(\theta_{\text{rel}}))^4$. These two terms roughly accomplish the same purpose as the constraint terms in the lower-level cost function but are $\mathcal{C}^\infty$ smooth. $\mathcal{L}_{\text{DDP}}^{\text{goal}}(\mathbf{s}_2) = \log(1 + d/L)$, as in equation 2.3. The obstacle cost function was identical to equation 2.5, except that the minimum operation used to compute $d_{\text{min}}^{\text{obstacle}}$ was replaced by a smooth approximation: $\min(x_1, x_2, \ldots, x_n) \approx (\sum_{i=1}^{n} x_i^p)^{1/p}$, with $p = 10$.

We executed DDP as a receding-horizon controller in which trajectories of length $K_2 T_2$ steps were planned at once. The first step of the plan was executed, and the entire sequence of optimized motor commands was shifted back by one step. This was used to "warm-start" DDP on the next iteration of planning. The motor command sequence at the initialization of each trial was chosen to be the same as the command sequence that the hierarchical controller selected.

When comparing the costs of trajectories, we computed $\mathcal{L}_{\text{DDP}}^{\text{obstacle}} + \mathcal{L}_{\text{DDP}}^{\text{goal}}$ for both trajectories. Therefore, the hierarchical controller was evaluated on a cost function it was not exactly trained to minimize.

Acknowledgments

We are grateful to Yann LeCun, Sara Solla, John Krakauer, and Peter Dayan for ideas, support, and criticism. We thank Yashar Ahmadian, Loïc Matthey, Mattia Rigotti, Ann Kennedy, Darcy Wayne, and Saul Kato for helpful discussions. This research was supported by the Gatsby, Swartz, Kavli, and National Science Foundations and by NIH grant MH093338.

References

Abbeel, P., & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. *Proceedings of the Twenty-First International Conference on Machine Learning*. New York: ACM.

Albert, C. Y., & Margoliash, D. (1996). Temporal hierarchical control of singing in birds. *Science*, *273*(5283), 1871–1875.

Albrecht, S., Sobotka, M., & Ulbrich, M. (2012). A bilevel optimization approach to obtain optimal cost functions for human arm-movements. *Numerical Algebra, Control and Optimization*, *2*, 105–127.

Barto, A. G., & Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, *13*(4) 341–379.

Bernstein, N. A. (1967). *The co-ordination and regulation of movements*. Cambridge: Pergamon Press.

Dayan, P. (2009). Goal-directed control and its antipodes. *Neural Networks*, *22*, 213–219.

Deisenroth, M., & Rasmussen, C. E. (2011). Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on Machine Learning* (pp. 465–472). New York: ACM.

Dietterich, T. G. (1998). The maxq method for hierarchical reinforcement learning. In *Proceedings of the 15th International Conference on Machine Learning* (pp. 118–126). San Francisco: Morgan Kaufmann.

Erez, T. (2011). *Optimal control for autonomous motor behavior*. Doctoral Dissertation. Washington University, St. Louis.

Gat, E. (1998). On three-layer architectures. In D. Kortencamp, R. P. Bonasso, & R. R. Murphy (Eds.), *Artificial intelligence and mobile robots*. Cambridge, MA: MIT Press.

Hantman, A. W., & Jessell, T. M. (2010). Clarke's column neurons as the focus of a corticospinal corollary circuit. *Nature Neuroscience*, *13*, 1233–1239.

Huh, D., & Todorov, E. (2009). Real-time motor control using recurrent neural networks. In *Adaptive dynamic programming and reinforcement learning*. Piscataway, NJ: IEEE.

Ijspeert, A. J., Nakanishi, J., Hoffmann, H., Pastor, P., & Schaal, S. (2013). Dynamical movement primitives: Learning attractor models for motor behaviors. *Neural Computation*, *25*(2), 328–373.

Jackson, J. H. (1889). On the comparative study of diseases of the nervous system. *British Medical Journal*, *2*, 355–362.

Jordan, M. I., Flash, T., & Arnon, Y. (1994). A model of the learning of arm trajectories from spatial deviations. *Journal of Cognitive Neuroscience*, *6*(4), 359–376.

Jordan, M. I., & Rumelhart, D. E. (1992). Forward models: Supervised learning with a distal teacher. *Cognitive Science*, *6*, 359–376.

Kalakrishnan, M., Pastor, P., Righetti, L., & Schaal, S. (2013). Learning objective functions for manipulation. In *Proceedings of the 2013 IEEE International Conference on Robotics and Automation* (pp. 1331–1336). Piscataway, NJ: IEEE.

Kavukcuoglu, K., Ranzato, M., & LeCun, Y. (2008). *Fast inference in sparse coding algorithms with applications to object recognition* (Tech. Rep. CBLL-TR-2008-12-01). New York: Computational and Biological Learning Lab, Courant Institute, NYU.

Lashley, K. S. (1930). Basic neural mechanisms in behavior. *Psychological Review*, *37*, 1–24.

Lazanas, A., & Latombe, J.-C. (1995). Landmark-based robot navigation. *Algorithmica*, *13*(5), 472–501.

LeCun, Y. (1988). A theoretical framework for backpropagation. In *Proceedings of the 1988 Connectionist Models Summer School*. San Francisco: Morgan Kaufmann.

Liu, D., & Todorov, E. (2009). Hierarchical optimal control of a 7-dof arm model. In *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, 2009 (pp. 50–57). Piscataway, NJ: IEEE.

Loeb, G., Brown, I., & Cheng, E. (1999). A hierarchical foundation for models of sensorimotor control. *Experimental Brain Research*, *126*(1), 1–18.

Miall, R. C., Weir, D. J., Wolpert, D. M., & Stein, J. F. (1993). Is the cerebellum a Smith predictor? *Journal of Motor Behavior*, *26*, 203–216.

Mombaur, K., Truong, A., & Laumond, J.-P. (2010). From human to humanoid locomotionan inverse optimal control approach. *Autonomous Robots*, *28*(3), 369–383.

Nguyen, D., & Widrow, B. (1989). The truck backer-upper: An example of self-learning in neural networks. In *Proceedings of the International Joint Conference on Neural Networks*. Piscataway, NJ: IEEE.

Raibert, M. H. (1977). *Motor control and learning by the state space model*. Doctoral dissertation, MIT.

Ratliff, N., Ziebart, B., Peterson, K., Bagnell, J. A., Hebert, M., Dey, A. K., & Srinivasa, S. (2009). Inverse optimal heuristic control for imitation learning. In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics.* New York: Springer.

Schaal, S., Peters, J., Nakanishi, J., & Ijspeert, A. (2005). Learning movement primitives. In *Proceedings of the International Symposium on Robotics Research* (pp. 561–572). New York: Springer.

Schmidt, M. (2013). minFunc. http://www.di.ens.fr/∼mschmidt/Software/minFunc.html

Shadmehr, R., Smith, M. A., & Krakauer, J. W. (2010). Error correction, sensory prediction, and adaptation in motor control. *Annual Review of Neuroscience*, *33*, 89–108.

Stengel, R. F. (1994). *Optimal control and estimation*. New York: Dover.

Sumbre, G., Gutfreund, Y., Fiorito, G., Flash, T., & Hochner, B. (2001). Control of octopus arm extension by a peripheral motor program. *Science*, *293*(5536), 1845–1848.

Sutskever, I. (2013). *Training recurrent neural networks*. Doctoral dissertation, University of Toronto.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Sutton, R. S., Precup, D., & Singh, S. (1999). Between MDPS and semi-MDPS: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*(1), 181–211.

Tassa, Y., Erez, T., & Todorov, E. (2011). *Fast model predictive control for reactive robot swimming*. http://www.cs.washington.edu/homes/todorov/papers/MPCswimmer.pdf

Tedrake, R., Zhang, T. W., & Seung, H. S. (2004). Stochastic policy gradient reinforcement learning on a simple 3d biped. In *Proceedings on Intelligent Robots and Systems* (Vol. 3, pp. 2849–2854). Piscataway, NJ: IEEE.

Thrün, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., . . . Mahoney, P. (2006). Stanley: The robot that won the DARPA grand challenge. *Journal of Field Robotics*, *23*, 661–692.

Widrow, B., Gupta, N. K., & Maitra, S. (1973). Punish/reward: Learning with a critic in adaptive threshold systems. *IEEE Transactions on Systems, Man, and Cybernetics*, *3*, 455–465.

Received May 29, 2013; accepted April 13, 2014.