

Milestone 4.1.3 -- Benchmarking analysis for neural controllers

Travis DeWolf and Chris Eliasmith

Milestone 4.1.3: Perform extensive testing of the controller on the established benchmarks, ensuring fair comparisons, and adjusting benchmarks as needed.

1.0 Introduction

The goal of this part of the project is to provide the results of benchmarking our neural controller, the Recurrent Error-driven Adaptive Control Hierarchy (REACH) model, presented in the first part of the project, using the benchmarking software built for the second deliverable. We provide a full benchmarking comparison of REACH to three different control methods: Linear Quadratic Regulation (LQR; [Simon and Stubberud, 1970](#); [Anderson and Moore, 2007](#)), Operational Space Control (OSC; [Khatib, 1987](#); [Nakanishi et al., 2008](#)), and iterative Linear Quadratic Gaussian (iLQG; [Todorov and Li, 2005](#)), which is a standard set of comparisons for state-of-the-art algorithms ([Sutskever, 2013](#); [Levine and Koltun, 2013](#)). Additionally, we have basic results from implementing the recurrent neural network trained using Augmented Hessian-Free (AHF; [Sutskever, 2013](#)) deep learning for controlling an arm. The control results described in ([Sutskever, 2013](#)) were not matched (see Section 4 for discussion). However, we favorably compare against the iLQG algorithm, which was used as the golden metric for comparison of the AHF controller.

What aspects of your algorithm can be considered neuro-inspired?

The REACH model is based on operational space control (Khatib, 1987), Dynamic Movement Primitives (DMPs; Schaal et al, 2006), and nonlinear adaptive control (Slotine and Li, 1987; Cheah and Slotine, 2005). These control algorithms were designed with the intention of generating movement as robust and adaptable as that of biological systems. We have taken these engineered algorithms and designed a spiking neural network architecture that approximates their performance with biologically plausible constraints. Both DMPs and nonlinear adaptive control have very natural neural implementations.

What is the nature of the task, is it object or action recognition? Tracking? Control? What objects or actions are being recognized?

In this project we implemented several controllers with the goal of effectively controlling a robotic arm through a benchmarking suite, also developed as part of the project. The tasks chosen from the benchmarking suite come from both control and neuroscience literature. The benchmarking suite contains 4 tasks.

1. The centre-out reaching task, where the controller must guide the hand to 8 targets from a center position.

2. The force field task, where the controller performs the centre-out reaching task while under the influence of an unknown force, which perturbs the arm based on the joint velocities.
3. The arm change task, where the controller performs the centre-out reaching task controlling an arm with unexpected kinematic parameters, i.e. the length of the arm segments is longer than expected.
4. The postural task, where the controller moves the hand to 8 different positions and attempts to hold it there while one of 8 unknown forces is applied.

2.0 Methods

What is the size of the dataset? Number of images or video frames, resolution, length of time, etc.? How many categories or classes?

For the controller trained with deep learning, the data is self-generated. The input to the neural network is the current state of the arm (joint positions and joint velocities) as well as the target joint positions and velocities. The neural network attempts to guide the arm to 64 different target positions around the centre-point. To aid training, one run is performed, the input is recorded, and after the input is held constant while 30 iterations of batch learning are performed. This stabilizes the network input/output relationship enough for the network to begin to converge to a solution. We have found this to greatly aid training / convergence.

For all of the other controllers, the testing dataset consists of reaching to 8 points evenly spread around the circle. For REACH, the adaptation is online during these reaches. All tests are performed 8 times to gather statistics to estimate confidence intervals, shown in the results (section 3).

What metric is being used for evaluation? Length of learning time? Accuracy? Computational complexity, Others?

Several metrics are being used to compare controllers across the different benchmark tasks:

1. The amount of time taken to reach the target.
2. The total length of the trajectory for reaching the target.
3. The cost function used in the deep learning training:

$$L_{x^*, x_0}(\theta) = \sum_{t=0}^T l_{x^*}(x_t, u_t, t)$$

$$l_{x^*}(x_t, u_t, t) = \|x_t - x^*\|^2 / 2 * \delta_{t,T} + \alpha \|u_t\|^2 / 2$$

where L is the cost function, x^* is the target, x_0 is the initial state, T is the final time, l_x is the immediate cost function, x_t is the state of the system at time t , u_t is the control signal at time t , and $\delta_{t,T}$ and α are gain terms. This was proposed by Sutskever, and is one of the benchmark costs used for evaluation.

4. The computational/energy efficiency cost is also computed and compared across algorithms.

What conventional algorithm is typically used? How does the best conventional algorithm perform?

Standard state-of-the-art controllers used for engineering applications have also been implemented in this comparison, including a linear quadratic regulator (LQR), iterative linear quadratic Gaussian (iLQG), and operational space controller (OSC). Their performance has been used to benchmark our model.

What is the relevant information of the algorithm, e.g., number of nodes/layers, arithmetic precision, and sparsity?

In the REACH model there are three separate neural areas, corresponding to anatomically and functionally distinct areas in the brain. These areas are the pre-motor cortex (PMC), the primary motor cortex (M1), and the cerebellum (CB). The network architecture for each of these areas is shown below:

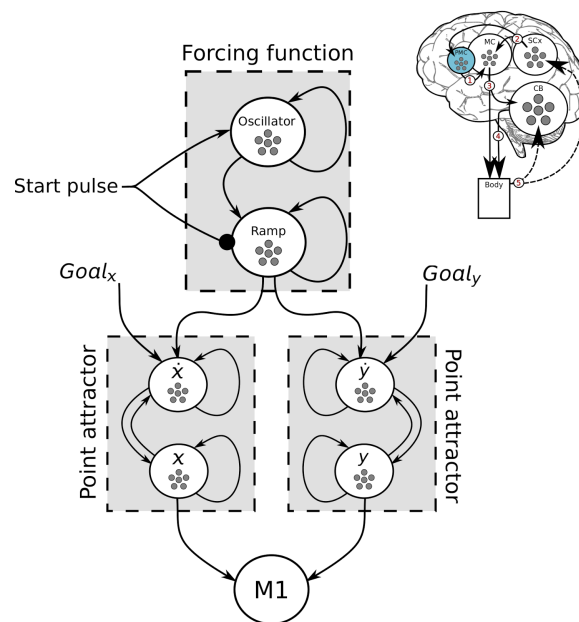


Figure 1: The pre-motor cortex model network architecture.

The pre-motor cortex is responsible for the generation of (x,y) trajectories for the hand to follow, and is employed in the writing tasks performed by the REACH model. The PMC is a biologically plausible implementation of Dynamic Movement Primitives (DMPs; Schaal et al, 2006). For both control of the 2-link arm and the 3-link arm it employs 4,500 LIF neurons.

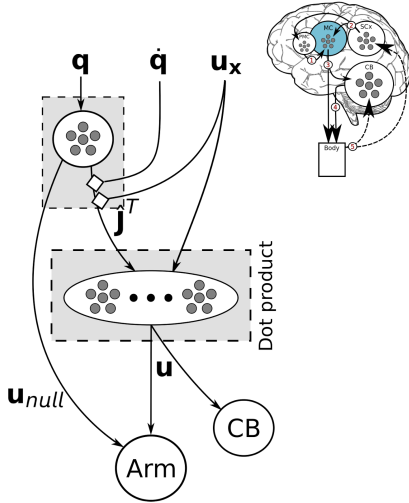


Figure 2: The primary motor cortex model network architecture.

The primary motor cortex transforms (x, y) forces into joint torques (u) to be sent to the arm simulation. Additionally, it has a secondary controller which generates a control signal that works to keep the arm near its resting joint angles (u_{null}). Finally, it also has the ability to adapt the Jacobian online to account for unexpected changes in the system kinematics. The M1 model uses 12,500 LIF neurons for control of the 2-link arm, and 21,000 LIF neurons for control of the 3-link arm.

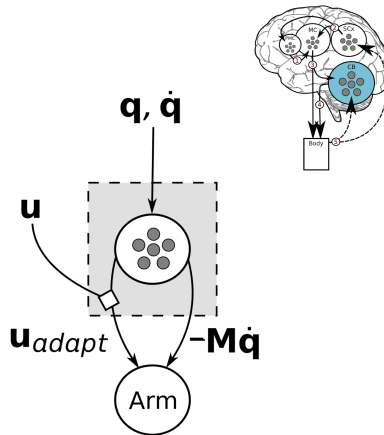


Figure 3: The cerebellum model network architecture.

The cerebellum calculates the effects of the internal dynamics of the system and sends out compensatory torques ($-M\dot{q}$). Additionally, it also adapts online to unexpected dynamics either from unexpected external perturbation or inaccurately modeled internal dynamics. The cerebellum uses 2,000 LIF neurons for control of the 2-link arm, and 3,000 LIF neurons for control of the 3-link arm.

3.0 Results

3.1 Trajectory comparisons

Accurate quantitative measure of obvious qualitative differences is a difficult problem. We begin with a figure of the trajectories generated by each controller, so that the qualitative discrepancy between the effectiveness of the different controllers can be appreciated. These results are shown in Figure 1A and 1B. The REACH 8-bit controller (demonstrated for reaching) was implemented to test the precision required by the connection weight matrix, using 8 bits instead of the standard 32 bits to store weights. Showing that the 8 bit version maintains comparable accuracy is an important result for implementation on neuromorphic hardware, where memory resources are very costly. Our efficiency calculations for the neuromorphic implementation assume 8 bit weights.

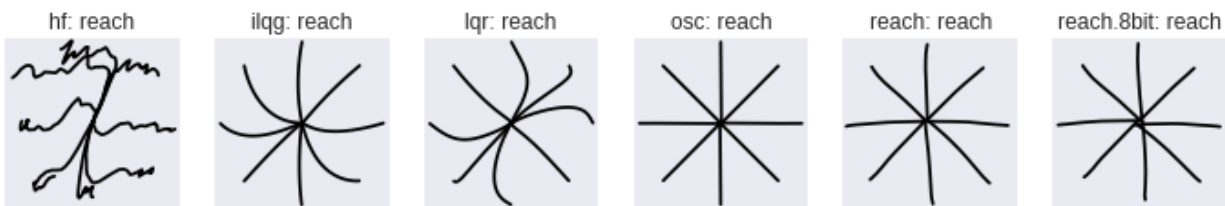


Figure 1A: The trajectories generated by the AHF (far left), iLQG (second from the left), LQR (third from left), OSC (third from the right), REACH (second from right), and REACH 8-bit (far right) control algorithms as they perform the reaching task.

As can be seen in Figure 1A, all of the controllers except AHF perform reasonably well at the reaching task. The AHF performance was after 62,432 training epochs, and 3 days of training. In general, a straight line is an optimal trajectory. However, if control costs are taken into account paths are often slightly curved as seen both in human and animal movements. We report the cost of each controller with respect to various cost functions in sections 3.3-3.5.

Figure 1B shows the performance on the remaining benchmarks, which highlight the importance of adaptation. As can be seen from the straight and stable paths exhibited by the REACH controller, it performs significantly better than the other controllers. Unsurprisingly from these graphs, REACH's performance on the perturbation benchmark is consistently the best of the tested controllers. However, the superior paths of REACH on the force field task are difficult to measure using standard cost functions as described below.

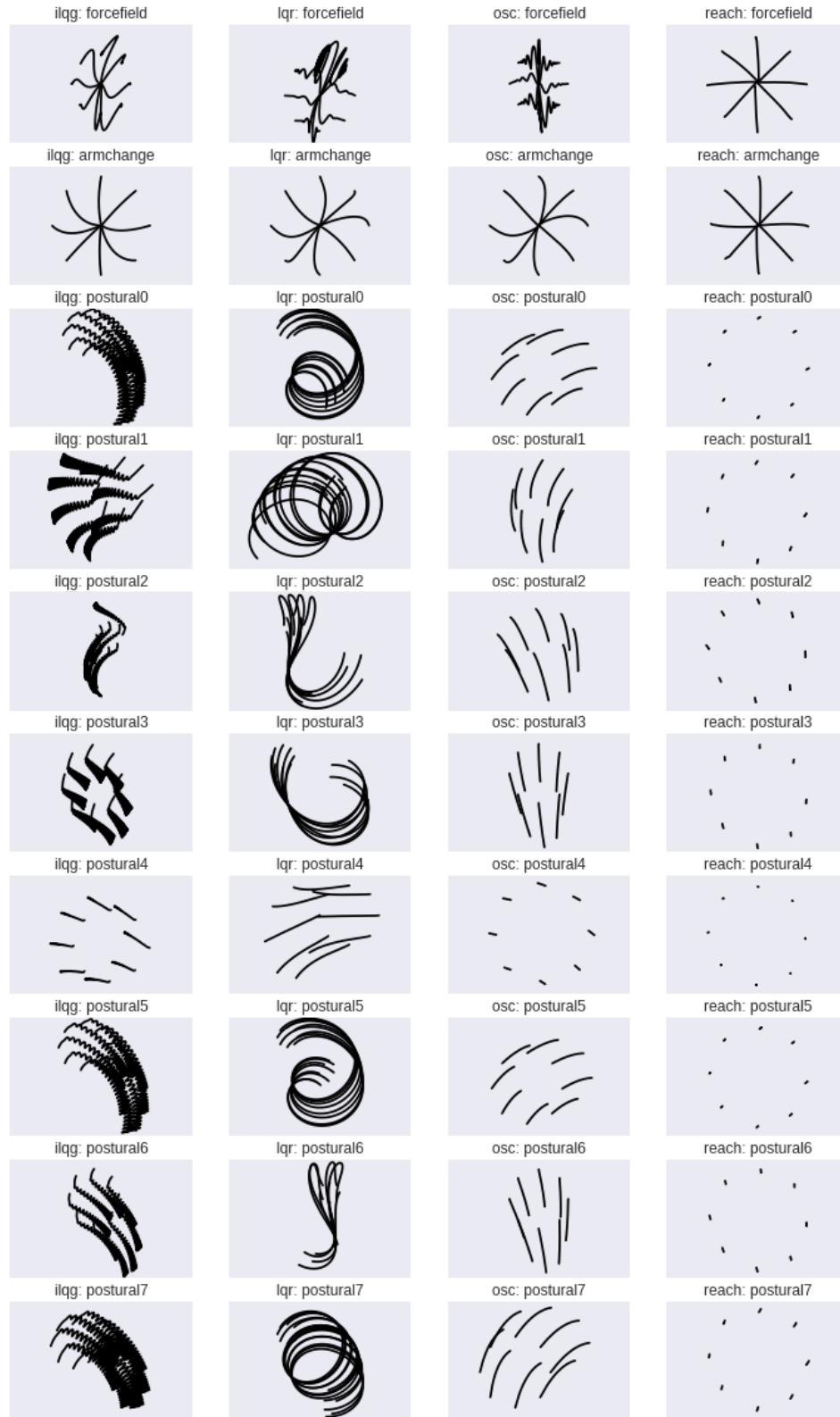


Figure 1B: The trajectories generated by the iLQG (far left), LQR (second from left), OSC (second from right), and REACH (far right) control algorithms as they perform the arm change, force field, and postural tasks.

3.2 Computational efficiency

We have calculated the energy efficiency of each of these control algorithms by determining the number of floating point operations (FLOPs) used for each of the standard controllers. The energy for the AHF network is calculated as the sum of the FLOPs required for the matrix-vector dot products needed to pass information through the artificial neural network at each timestep, which is a function of the number of nodes in each layer. To compare to REACH, we have calculated the equivalent energy demands as implemented on neuromorphic hardware. The method used for this calculation is the same as that used in deliverable 4.2.3 (section 4), and explained there in more detail. These results are shown in Figure 2. In Figure 3 we show how the increase in complexity of the system under control affects the energy cost of each algorithm.

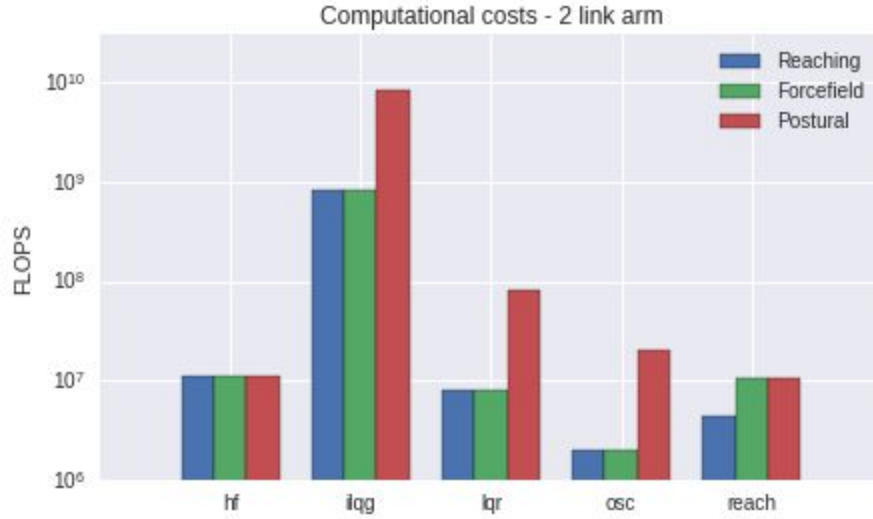


Figure 2: The computational cost of each of the control algorithms for the 2-link arm in the reaching, force field, and postural tasks. The y-axis is a logarithmic scale.

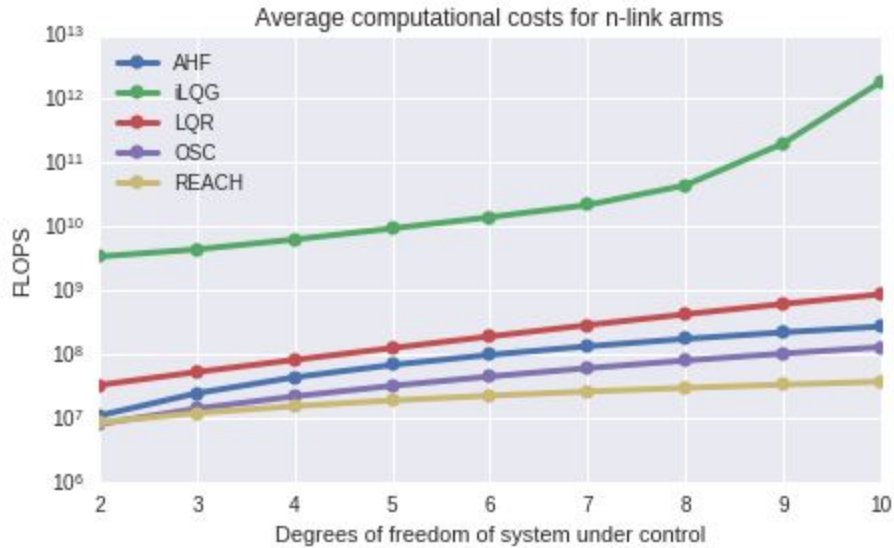


Figure 3: The computational cost of each control algorithm as the complexity of the system under control increases. The y-axis is a logarithmic scale.

Figure 3 highlights that as the complexity of the controlled system increases, the REACH algorithm energy cost grows much slower than the others. As a means of incorporating the energy cost into our results, we provide both raw and energy ‘normalized’ results in sections 3.3-3.5. The normalized results provide a measure of accuracy per FLOP. The normalization is performed using the 2-link values.

We have also examined the efficiency of the REACH controller running on neuromorphic hardware compared to running the equivalent model on a GPU or CPU. In particular, we determined the overhead (idle) power of the SpiNNaker 48-chip board, an Intel Xeon E5-1650v3 @ 3.5Ghz CPU and an ASUS GeForce GTX Titan X GPU. We then ran the same REACH controller on each using the Nengo 2.0 simulation environment and recorded the difference in power usage. We normalized the power usage to account for real-time performance (the GPU ran faster than real time). Consequently, to run at real time, the CPU required 33W the GPU required 25W and the SpiNNaker board required <1W (our measuring device provided accuracy within 1W). In conclusion, the neuromorphic board was approximately 30x more efficient running the same algorithm.

Although the AHF controller is also a neural network, it is not implemented in spiking neurons, and so cannot be implemented on our neuromorphic hardware. The energy cost for the AHF network comes to 10,836,000 FLOPs for control of the 2-link arm, which is approximately 2.5 times more expensive than the REACH controller. As shown in Figure 3, control of the 2-link arm is when the AHF controller is the closest to being as efficient as the REACH controller, as the complexity of the system being controlled increases the AHF energy cost grows much faster than the REACH energy cost.

3.3 Speed and accuracy costs

The first benchmark is a standard combined measure of speed and accuracy for a controller. Specifically this is the sum over the path of the distance between the current position of the end effector, x , and the target position, x^* . As can be seen in Figure 3, REACH performs similarly to the other controllers on most tasks, though it performs significantly better on the postural task.

However, it should be noted that this benchmark provides some surprising results. For instance, the iLQG controller takes a very winding path during the force field task and does not get close to the correct target (see Figure 1B). Nevertheless it has a low cost by this metric because it is extremely fast (i.e. it quickly stops moving even though it is not close to the target).

Figure 4, the energy normalized cost function, demonstrates the high cost of the iLQG controller from an energy efficiency standpoint.

3.4 Total path length cost

A second metric that we employed is the total path length cost. This measurement sidesteps the difficulties of fast but highly inaccurate controllers. Because the optimal path for reaching is a

straight line, and the optimal path for the postural task is no movement, path length correlates well with performance. The results of computing this cost are shown in Figure 5.

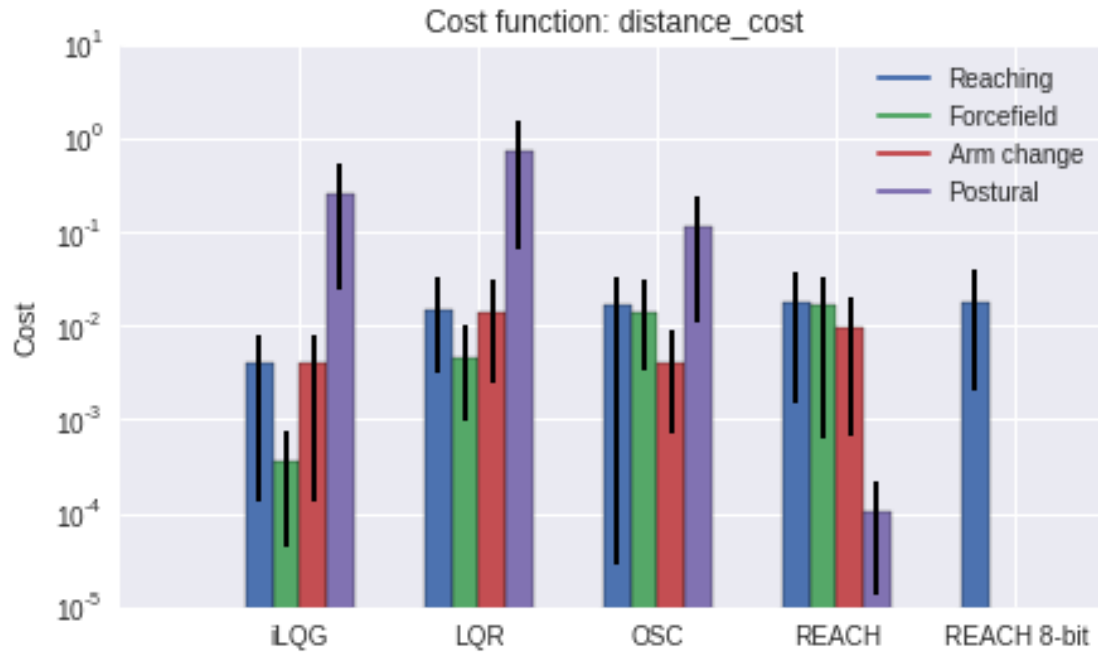


Figure 4: Data analysis using the sum of $(x-x^*)$ during the reach, where x is the system position and x^* is the target position. The y-axis is a logarithmic scale.

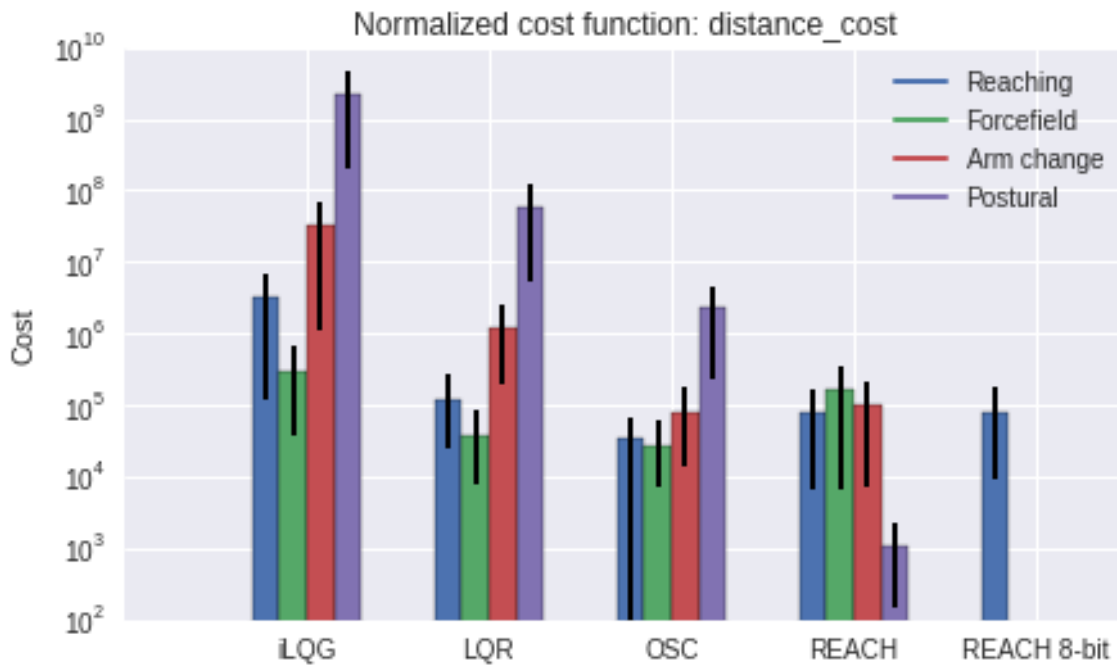


Figure 5: The same data analysis presented in Figure 6, normalized by the computational cost of each algorithm. The y-axis is a logarithmic scale.

Again, some very poor paths (e.g. iLQG on the force field task; see Figure 1B) score surprisingly well on this metric. This is because the path ends so far from the correct position that, despite it being both winding and indirect, it remains a short path.

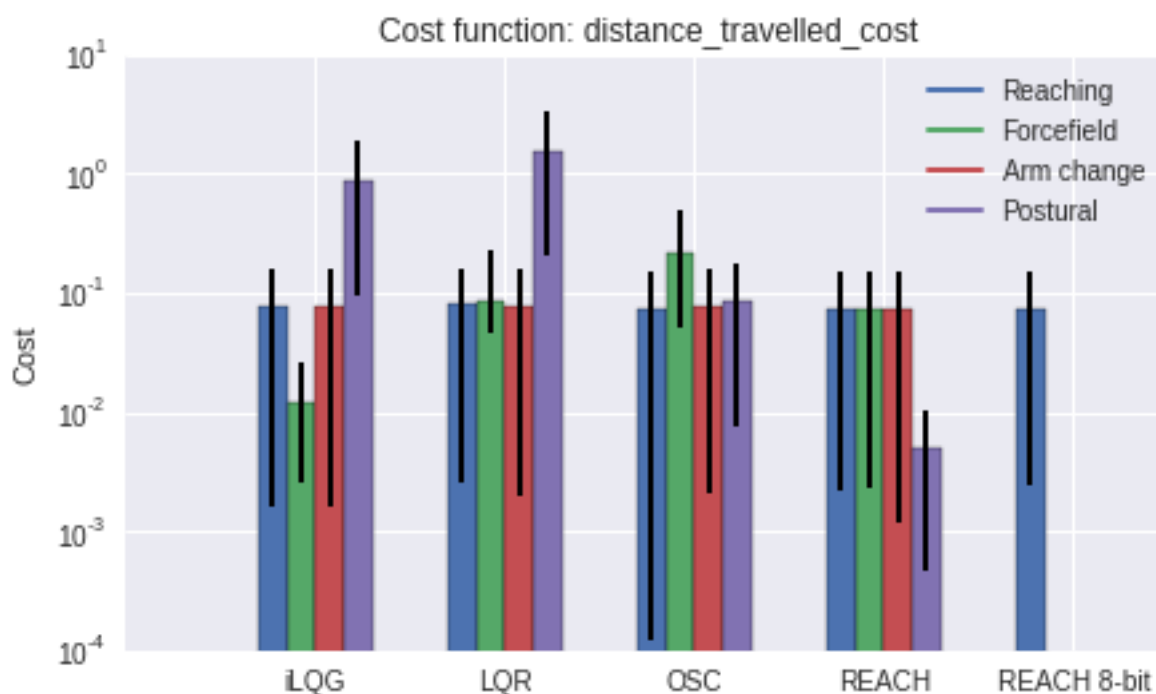


Figure 6: Data analysis using the total distance travelled during the reach as the cost function.. The y-axis is a logarithmic scale.

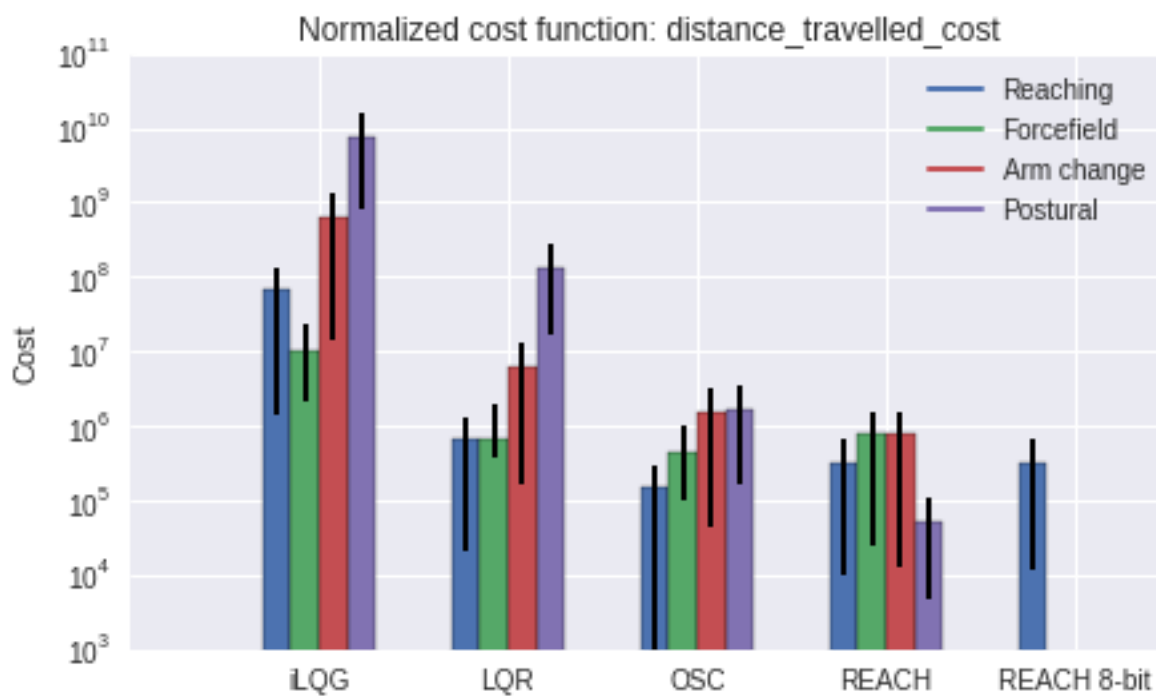


Figure 7: The same data analysis presented in Figure 8, normalized by the computational cost of each algorithm. The y-axis is a logarithmic scale.

In general, REACH performs comparably to the other controllers using this metric. This is somewhat surprising given the very different paths that are taken by the controllers (Figure 1B). Figure 6 highlights the good combination of accuracy and energy efficiency displayed by the REACH controller.

3.5 Speed, accuracy and control costs

The cost function proposed in Sutskever's work and used AHF during training is:

$$L_{x^*, x_0}(\theta) = \sum_{t=0}^T l_{x^*}(x_t, u_t, t)$$

$$l_{x^*}(x_t, u_t, t) = \|x_t - x^*\|^2/2 * \delta_{t,T} + \alpha \|u_t\|^2/2$$

where L is the cost function, x^* is the target, x_0 is the initial state, T is the final time, l_x is the immediate cost function, x_t is the state of the system at time t , u_t is the control signal at time t , and $\delta_{t,T}$ and α are gain terms. This cost function is intended to weight accuracy and control cost to determine the overall effectiveness of the controller. The gain terms used are the same as those employed by Sutskever. The results of computing this cost are shown in Figure 7.

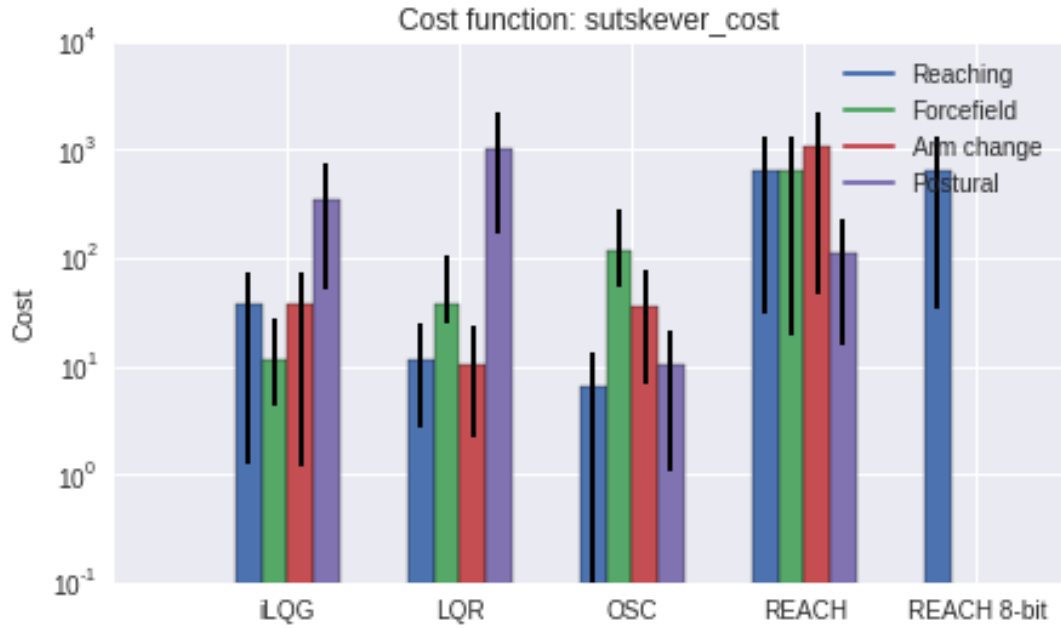


Figure 8: Data analysis using the cost function defined in Sutskever's thesis. The REACH model compares best on the postural task, when unexpected perturbations are applied to the system. The y-axis is a logarithmic scale.

As is evident in Figure 7, very poor trajectories (e.g. OSC on the postural task, iLQG on the force field task) can score surprisingly well. Because of the high weight on control cost, the REACH model does not perform as well as the other controllers. The higher control cost for the REACH controller is a result of it adapting during the task. However, that adaptation also results in much better paths than the other controllers. Consequently, we believe this is a very poor metric for performance and would not recommend using it in the future, unless principled reasons can

be given for specific choices of the gain parameters. Normalizing these results demonstrate the high computational cost of the iLQG controller as shown in Figure 8.

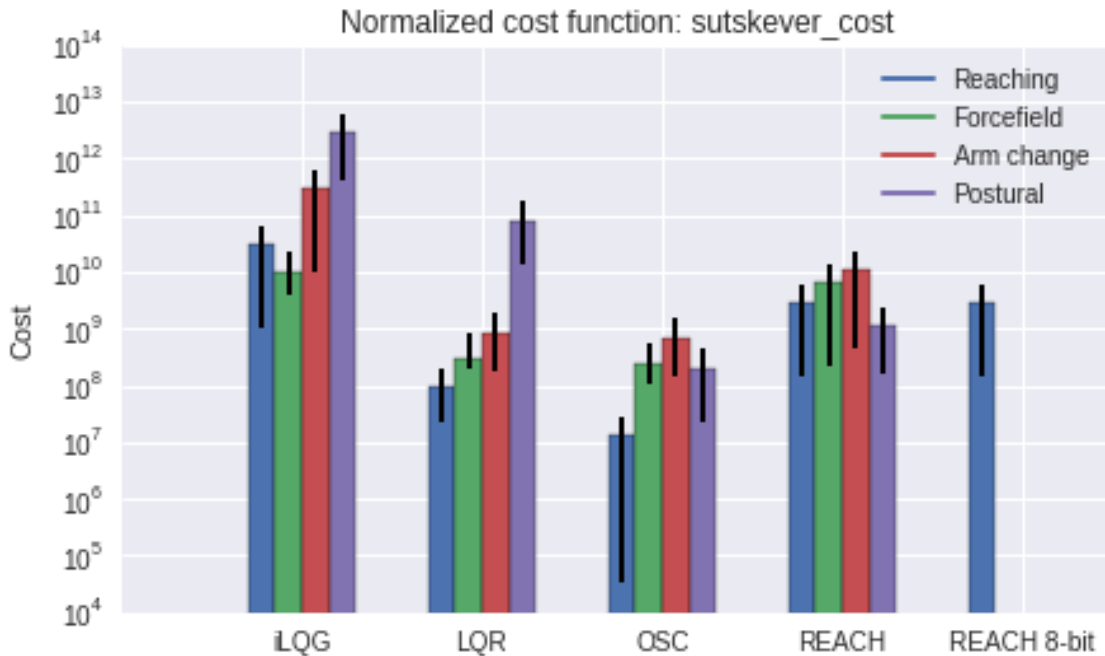


Figure 9: The same data analysis presented in Figure 5, normalized by the computational cost of each algorithm. The y-axis is a logarithmic scale.

4.0 Discussion

In general, the REACH algorithm is performing comparably to the state-of-the-art algorithms on most metrics (except the AHF training metric, which we believe is flawed as discussed). As well, accounting for the computational costs of each algorithm works to the advantage of the REACH algorithm, although not in all cases (e.g. OSC is slightly more efficient for 2-link reaching). It is worth noting again that this energy efficiency normalization was performed using the computational cost estimates for the 2-link arm, and that as the complexity of the system increases the REACH model will become significantly more efficient than the other benchmarked algorithms (Figure 3).

There are several other issues worth noting with these particular metrics that are not conveyed by the above figures. The first is that the iLQG and LQR algorithms are using finite differences to calculate the derivatives of the system, which are required for their computations. The benefit of this approach is that it easily generalizes to more complex models, allowing the controllers to be applied to any resettable system, simulated or physical. However, this is a more computationally intensive method than analytically computing the derivatives, if a model of the system is known. However, if there is uncertainty in such a model, or it is inaccurate, analytic computation is unlikely to provide an accurate enough model for effective control.

Relatedly, the OSC algorithm requires an explicit dynamical model of the system under control, and cannot operate using the generalizable finite differences method. As a result it has a much

lower computational cost, but also the ease-of-use is significantly impaired, as intimate knowledge of the system under control is required. Consequently, the OSC algorithm is likely to perform poorly under circumstances of uncertainty regarding the system being controlled or when the system becomes significantly more complex.

Finally, we have had significant difficulty in replicating the AHF methods to sufficient accuracy (see Figure 1A). We believe that this difficulty in implementing the Augmented Hessian-Free (AHF) controller described in ([Sutskever, 2013](#)) has several root causes. The first is that the training regime used for training the network on the arm simulation was not sufficiently described. We found great variability in performance with minor variations of how many iterations were run, how often network input was regenerated, and the random seed used in initialization. We also found that the use of a ‘squashing’ function was necessary during training to keep the state of the arm within a reasonable range and discourage instability, which was not mentioned in Dr. Sutskever’s thesis.

The second cause of difficulty in reproducing the AHF is that we used a slightly more complex arm model, which is the standard formulation of 2-link arm dynamics, rather than the specially derived 2-link arm model used by Dr. Sutskever in his thesis, which was reworked to be particularly amenable to deep learning. The fact that this additional complexity caused difficulty suggests that these methods are unlikely to scale effectively.

The third cause of difficulty is that we used finite differences to approximate the derivative of the arm, rather than explicitly providing the derivative of the dynamics. This allowed us to generalize the training and test several arm models, but potentially could also reduce the accuracy of the calculated derivative. Again, if finite differences cannot be used, it is unlikely that the method will be able to handle uncertainty in the system being controlled or more complex systems.

Fourth, a major impediment was the sheer amount of training time required to train the AHF network. Our training regime involved 1) running the network forward to test the performance, 2) running 32 epochs (where each epoch collects data from 64 runs of the plant over 40 timesteps at $dt=0.01$) with a potential 96 rounds of conjugate gradient descent, 3) repeating. To achieve the control shown in Figure 1 required 1,951 iterations of steps 1 and 2, or 62,432 total epochs. Running on the GPU of a modern supercomputer this took over 72 hours. Considering the number of free parameters in a training regime and neural network this quickly becomes an unmanageable number for parameter searching. The algorithm is on average monotonic, as shown in Figure 9, but it moves slowly (note the logarithmic scale).

Finally, a fifth difficulty was that in Dr. Sutskever’s thesis his network starts out with an error of .25. After 62,432 epochs we are now at an error of .285, which is approaching the error level that his networks began their training with. How the parameters for these initial networks were determined was not specified, but they began with an already approximate solution of the problem, whereas the networks we trained here were initialized randomly as is typically performed.

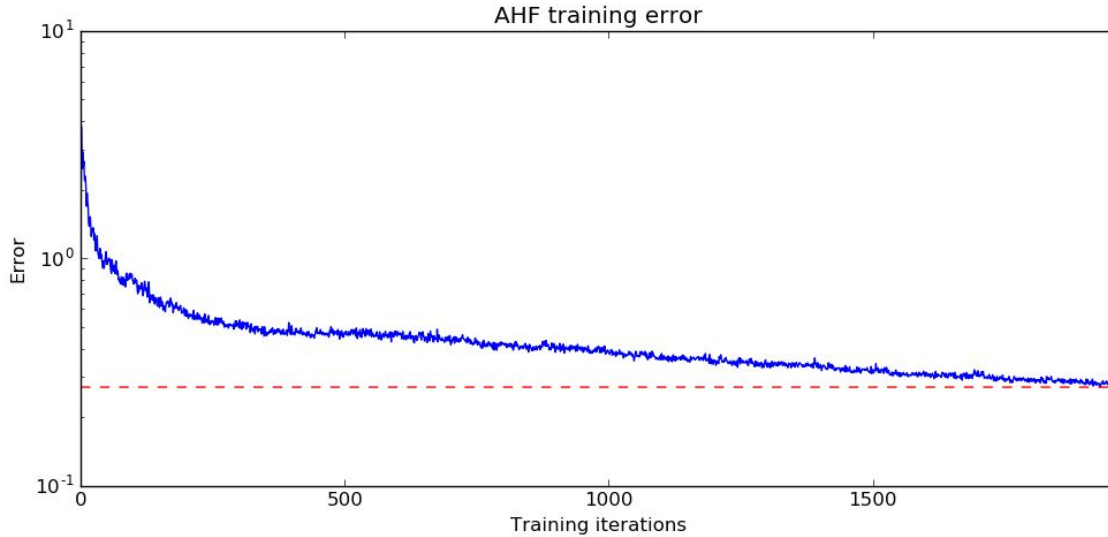


Figure 10: The error of the AHF network during training. Each training iteration consists of 32 epochs. Note the logarithmic y-axis. After 1,951 trials we are approaching the error that the Sutskever AHF network started at.

Advantage and disadvantage of Neuro-Inspired vs conventional algorithms?

There are two main advantages of using the neuro-inspired algorithm over conventional algorithms. The first is the adaptive dynamics compensation, which allows very robust control in the face of unexpected external or internal dynamics. The second main advantage to using an algorithm implemented in a spiking neural network is that they can be implemented on highly efficient neuromorphic hardware. This allows these algorithms to compute highly complex operations in real-time, using several orders magnitude less power than standard CPU or GPU implementations.

5.0 Recommendations

Recent developments in training methods for controllers, such as guided policy optimization ([Levine and Abbeel, 2014](#)), provide a means of successfully training deep networks for control. We believe that these are much more likely to provide good controllers than AHF. Furthermore, the iLQG algorithm was used as the gold standard for comparison in Dr. Sutskever's thesis on AHF control. We have shown here that the REACH controller performs better, in general, than the iLQG controller and much better when efficiency is taken into account. As well, we have demonstrated that the rapid adaptation of the REACH controller allows it to perform tasks that other controllers are not able to perform well (e.g. the force field, arm change, and postural tasks). While AHF is trained, adaptation is extremely slow and so it will also not be able to perform these tasks rapidly. Consequently, we recommend moving on to the final stage of the project (i.e., application of REACH on robotic hardware) in lieu of further efforts to train and benchmark the AHF controller.

6.0 References

Slotine, Jean-Jacques E., and Weiping Li. "On the adaptive control of robot manipulators." *The international journal of robotics research* 6.3 (1987): 49-59.

Cheah, Chien-Chern, Chao Liu, and J. Slotine. "Adaptive Jacobian tracking control of robots based on visual task-space information." *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE, 2005.

Khatib, Oussama. "A unified approach for motion and force control of robot manipulators: The operational space formulation." *Robotics and Automation, IEEE Journal of* 3.1 (1987): 43-53.

Levine, Sergey, and Pieter Abbeel. "Learning neural network policies with guided policy search under unknown dynamics." *Advances in Neural Information Processing Systems*. 2014.

Schaal, Stefan. "Dynamic movement primitives-a framework for motor control in humans and humanoid robotics." *Adaptive Motion of Animals and Machines*. Springer Tokyo, 2006. 261-280.

Sutskever, Ilya. *Training recurrent neural networks*. PhD Thesis. University of Toronto, 2013.