Proceedings of International Joint Conference on Neural Networks, Orlando, Florida, USA, August 12-17, 2007

Toyota Prius HEV neurocontrol

Danil Prokhorov

***Abstract*— I propose a neural network controller for improved fuel efficiency of the Toyota Prius hybrid electric vehicle. The approach is based on recurrent neural networks and an effective combination of off-line and on-line training methods including the extended Kalman filter and the simultaneous perturbation stochastic approximation (SPSA). The proposed approach is quite general and applicable to other control systems.**

I. Introduction

Hybrid powertrains have been gaining popularity due to their potential to improve fuel economy significantly and reduce undesirable emissions. Control strategies of the hybrid electric vehicle (HEV) are more complex than those of the internal combustion engine-only vehicle because they have to deal with multiple power sources in sophisticated configurations. The main function of any control strategy is power management. It typically implements a high-level control algorithm which determines the appropriate power split between the electric motor and the engine to minimize fuel consumption and emissions, while staying within specified constraints on drivability, reliability, battery charge sustenance, etc.

Computational intelligence techniques have previously been applied to HEV power management by various authors. A rule-based control was employed in [1]. Fuel economy improvement with a fuzzy controller was demonstrated in [2], relative to other strategies which maximized only the engine efficiency. An intelligent controller combining neural networks and fuzzy logic which could adapt to different drivers and drive cycles (profiles of the required vehicle speed over time) was studied in [3]. Recently a neurocontroller was employed in a hybrid electric propulsion system of a small unmanned aerial vehicle which resulted in significant energy savings [4].

The references cited above indicate a significant potential for improving HEV performance through more efficient power management based on application of computational intelligence (CI) techniques. Though the Toyota HEV Prius efficiency is quite high already, there is a potential for further improvement, as illustrated in this paper.

Unlike traditional hybrid powertrain schemes, series or parallel, the Prius hybrid implements what is called the power split scheme. This scheme is quite innovative and has not been studied extensively yet from the standpoint of application of CI techniques. The Prius powertrain uses a planetary gear mechanism to connect an internal combustion engine, an electric motor and a generator. A highly efficient engine can simultaneously charge the battery through the generator and propel the vehicle (Figure 1). It is important to be able to set the engine operating point to the highest efficiency possible and at sufficiently low emission levels of undesirable exhaust gases such as hydrocarbons, nitrogen oxides and carbon monoxide. The motor is physically attached to the ring gear. It can move the vehicle through the fixed gear ratio and either assist the engine or propel the vehicle on its own for low speeds. The motor can also return some energy to the battery by working as another generator in the regenerative braking mode.

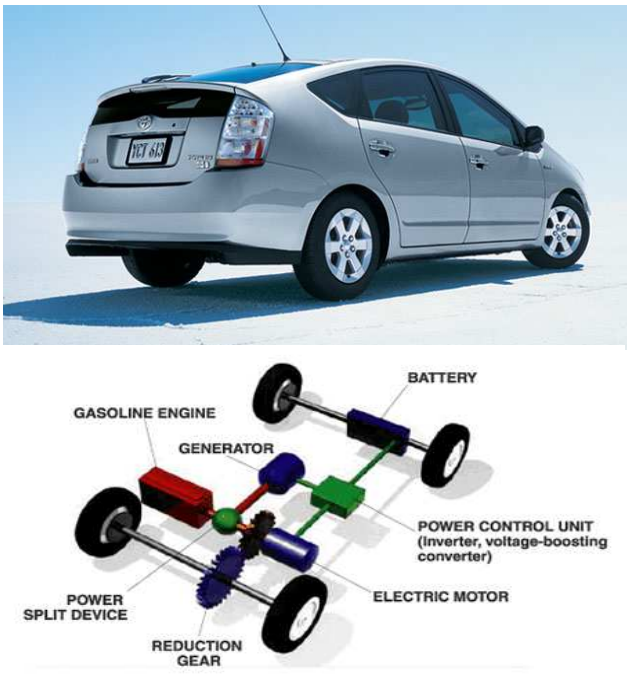

Fig. 1. The Prius car and the main components of the Toyota hybrid system.

As in the previous work [5], [6], I employ recurrent neural networks (RNN) as controllers and train them for robustness to parametric and signal uncertainties (known bounded variations of physical parameters, reference trajectories, measurement noise, etc.). I intend to deploy the trained neurocontroller with fixed weights. It is still desirable to have a possibility to influence the closed-loop performance in case some degree of adaptivity is needed, e.g., when an intermittent fault in the system occurs which temporarily makes significant changes in its performance (until repairs are made). It may be not helpful to adapt weights of the controller because 1) it would compromise its already trained weights, i.e., its long-term memory, which is undesirable

Danil Prokhorov is with Toyota Technical Center, a division of Toyota Motor Engineering and Manufacturing North America (TEMA), Ann Arbor, MI 48105, e-mail: dvprokhorov@gmail.com

in the intermittent fault case, and 2) adaptation in strongly nonlinear systems can cause bifurcations. I prefer to augment the fixed-weight RNN controller with a conceptually simple direct adaptive control scheme.

This paper is structured as follows. In the next section I describe main elements of the off-line training. The approach permits me to create an RNN controller which is ready for deployment with fixed weights. In Section III I discuss the proposed on-line training which features a combination of the fixed-weight controller and an adaptive controller trained on-line to supplement the first controller outputs. I then describe my experiments in Section IV, followed by conclusions and directions for future work.

II. Off-line training

I adopt the approach of indirect or model based control development for off-line training. The Prius simulator is highly complex, distributed software which makes training a neurocontroller directly in the simulator difficult. I implemented an approach in which the most essential elements of the simulator are approximated sufficiently accurately by a neural network model. The NN model is used to train a neurocontroller by effectively replacing the simulator. The trained neurocontroller performance is then verified in the simulator.

The use of differentiable NN for both model and controller makes possible application of the industrially proven training method. I describe here only the main elements of the method, referring the reader to other references for its more comprehensive account [5], [7].

Truncated backpropagation through time (BPTT(h), where h stands for the truncation depth) offers potential advantages relative to forward methods for obtaining sensitivity signals in NN training problems. First, the computational complexity scales as the product of h with the square of the number of nodes (for a fully connected NN). The required storage is proportional to the product of the number of nodes and the truncation depth h. Second, BPTT(h) often leads to a more stable computation of dynamic derivatives than do forward methods because its history is strictly finite. Third, the use of BPTT(h) permits training to be carried out asynchronously with the RNN execution. This feature enabled testing a BPTT based approach on a real automotive hardware as described in [7].

After the derivatives are computed via BPTT(h), I can update the NN weights. Unlike weight update methods that originate from the field of differentiable function optimization, the extended Kalman filter (EKF) method treats supervised learning of a NN as an optimal filtering problem. The NN weights $\mathbf{w}$ are interpreted as states of the trivially evolving dynamic system, with the measurement equation described by the NN function $\mathbf{h}$:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \boldsymbol{\nu}(t) \quad (1)$$

$$\mathbf{y}^d(t) = \mathbf{h}(\mathbf{w}(t), \mathbf{i}(t), \mathbf{v}(t-1)) + \boldsymbol{\omega}(t) \quad (2)$$

The weights $\mathbf{w}$ may be organized into g mutually exclusive weight groups. This trades off performance of the training method with its efficiency; a sufficiently effective and computationally efficient choice, termed node decoupling, has been to group together those weights that feed each node. Whatever the chosen grouping, the weights of group i are denoted by $\mathbf{w}_i$. The corresponding derivatives of network outputs with respect to weights $\mathbf{w}_i$ are placed in N_{out} columns of $\mathbf{H}_i$.

To minimize at time step t a cost function $E = \sum_t \frac{1}{2}\boldsymbol{\xi}(t)^T\mathbf{S}(t)\boldsymbol{\xi}(t)$, where $\mathbf{S}(t)$ is a nonnegative definite weighting matrix and $\boldsymbol{\xi}(t)$ is the vector of errors, $\boldsymbol{\xi}(t) = \mathbf{y}^d(t) - \mathbf{y}(t)$, where $\mathbf{y}(t) = \mathbf{h}(\cdot)$ from (2), the decoupled EKF equations are as follows [7]:

$$\mathbf{A}^*(t) = \left[\frac{1}{\eta(t)}\mathbf{I} + \sum_{j=1}^{g}\mathbf{H}_j^*(t)^T\mathbf{P}_j(t)\mathbf{H}_j^*(t)\right]^{-1} \quad (3)$$

$$\mathbf{K}_i^*(t) = \mathbf{P}_i(t)\mathbf{H}_i^*(t)\mathbf{A}^*(t) \quad (4)$$

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \mathbf{K}_i^*(t)\boldsymbol{\xi}^*(t) \quad (5)$$

$$\mathbf{P}_i(t+1) = \mathbf{P}_i(t) - \mathbf{K}_i^*(t)\mathbf{H}_i^*(t)^T\mathbf{P}_i(t) + \mathbf{Q}_i(t) \quad (6)$$

In these equations, the weighting matrix $\mathbf{S}(t)$ is distributed into both the derivative matrices and the error vector: $\mathbf{H}_i^*(t) = \mathbf{H}_i(t)\mathbf{S}(t)^{\frac{1}{2}}$ and $\boldsymbol{\xi}^*(t) = \mathbf{S}(t)^{\frac{1}{2}}\boldsymbol{\xi}(t)$. The matrices $\mathbf{H}_i^*(t)$ thus contain scaled derivatives of network outputs with respect to the ith group of weights; the concatenation of these matrices forms a global scaled derivative matrix $\mathbf{H}^*(t)$. A common global scaling matrix $\mathbf{A}^*(t)$ is computed with contributions from all g weight groups through the scaled derivative matrices $\mathbf{H}_j^*(t)$, and from all of the decoupled approximate error covariance matrices $\mathbf{P}_j(t)$. A user-specified learning rate $\eta(t)$ appears in this common matrix. (Components of the measurement noise matrix $\mathbf{R}$ are inversely proportional to $\eta(t)$.) For each weight group i, a Kalman gain matrix $\mathbf{K}_i^*(t)$ is computed and is then used in updating the values of the group's weight vector $\mathbf{w}_i(t)$ and in updating the group's approximate error covariance matrix $\mathbf{P}_i(t)$. Each approximate error covariance update is augmented by the addition of a scaled identity matrix $\mathbf{Q}_i(t)$ that represents additive data deweighting.

I employ a multi-stream version of the algorithm above. A concept of multi-stream was proposed in [8] for improved training of RNN via EKF. It amounts to training N_s copies (N_s streams) of the same RNN with N_{out} outputs. Each copy has the same weights but different, separately maintained states. With each stream contributing its own set of outputs, every EKF weight update is based on information from all streams, with the total effective number of outputs increasing to $M = N_sN_{out}$. The multi-stream training may be especially effective for heterogeneous data sequences because it resists the tendency to improve local performance at the expense of performance in other regions.

Figure 2 illustrates how the neurocontroller is trained in our off-line training. After the NN model of the plant is trained, I run the closed-loop system forward for h time steps. I then backpropagate through the temporal chain of the closed-loop system copies from step $k+h$ to step k while computing derivatives of a performance measure with

respect to neurocontroller weights. The EKF update is initiated once the derivatives are computed. The entire process (going forward, collecting derivatives, updating weights) is repeated many times by stepping through a trajectory until an acceptable performance is achieved. More details can be found in [5].

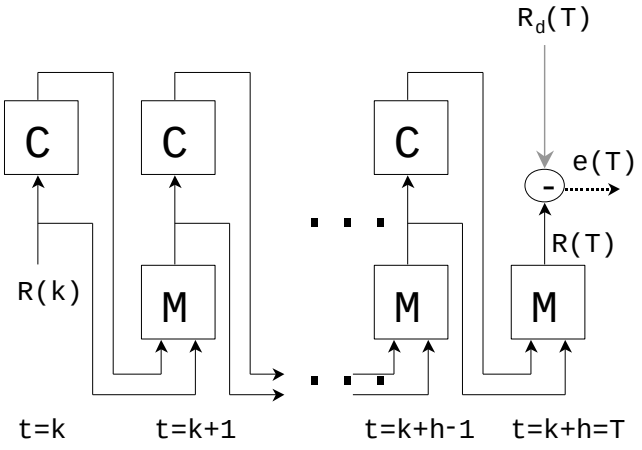

Fig. 2. Pictorial representation of unfolding the closed-loop system consisting of neurocontroller C and plant model M. The variables predicted by the model R are compared with their desired values R_d, generating the error e. While this is done only for the last step of the unfolding, the unfolding process itself is repeated many times, starting from $t = k+1, k+2, ...$

III. On-line training

In [9] I discussed two options for augmenting the fixed-weight RNN controller trained as described in the previous section with another NN for on-line adaptation. One of the options is to use a J critic in a scheme shown in Figure 3 (see, e.g., [10] for more details on adaptive critics).

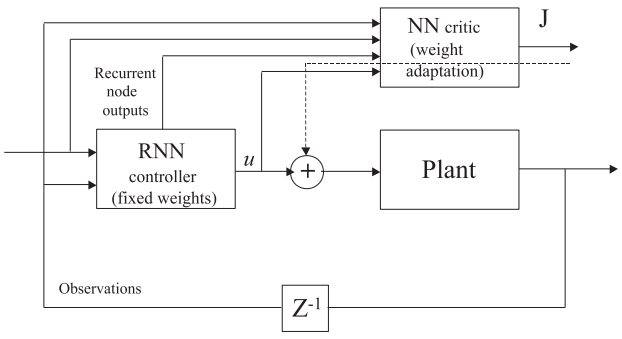

Fig. 3. An augmentation of the robust recurrent neurocontroller with fixed weights by the J critic for improved adaptive control of the plant. The critic network uses the same inputs as the robust neurocontroller, its output u, plus all outputs of the state nodes of the robust neurocontroller. The adaptive correction Δu of control u (dashed line) is computed by backpropagating through the critic, i.e., $\Delta u = -\mu \partial J / \partial u$.

Figure 4 shows another option for on-line adaptation with a special NN. This special NN is an adaptive portion of the combined neurocontroller, which is to be trained by a suitable method. The advantage of this option over Figure 3 is that

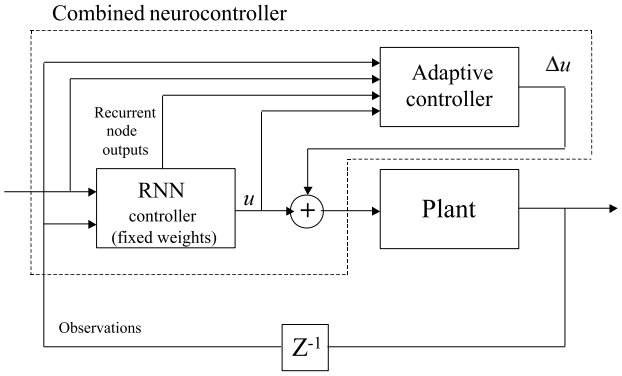

Fig. 4. Proposed combination of the robust recurrent neurocontroller with fixed weights and an adaptive controller for improved control of the plant. The adaptive controller can be another NN which uses the same inputs as the robust neurocontroller, its output u, plus all outputs of the state nodes of the robust neurocontroller. Unlike Figure 3, the adaptive correction Δu of control u is computed directly, thereby avoiding possible arbitrariness in choosing the learning rate μ (see the caption of Figure 3).

bounded control corrections Δu are computed directly, rather than via backpropagation through the critic.

For training the adaptive NN, I resort to the simultaneous perturbation stochastic approximation (SPSA) method [11]. A popular form of the gradient descent-like SPSA uses two cost evaluations *independent* of parameter vector dimensionality to carry out one update of each adaptive parameter. Each SPSA update is

$$W_i^{next} = W_i - aG_i(\mathbf{W}) \quad (7)$$

$$G_i(\mathbf{W}) = \frac{Cost^+ - Cost^-}{2c\mathbf{\Delta}_i} \quad (8)$$

where $\mathbf{W}$ is a weight vector of the adaptive controller, $Cost^{\pm}$ is a cost function to be minimized, $\mathbf{\Delta}$ is a vector of symmetrically distributed Bernoulli random variables generated anew for every update step (e.g., the i-th component of $\mathbf{\Delta}$ denoted as $\mathbf{\Delta}_i$ is either $+1$ or -1), c is size of a small perturbation step, and a is a learning rate. The notation $Cost^{\pm}$ corresponds to the $Cost$ values obtained for the weight perturbations $\mathbf{W} \pm c\mathbf{\Delta}$.

Each SPSA update requires that two consecutive values of the $Cost$ function be computed. This means that one SPSA update occurs no more often than once every other time step. It may also be helpful to let the $Cost$ function represent changes of the cost over a short window of some number of time steps τ, in which case each SPSA update would be even less frequent. In the example below $Cost$ is the windowed sum of $cost(t)$, or differences between desired and actual quantities of interest, e.g., plant outputs. This allows the plant additional time to react to changing $\mathbf{W}$.

My pivotal and realistic assumption is that all signals in the closed-loop system are bounded, which assures the bounded input bounded output (BIBO) stability. This is critical, especially in the case of on-line training. Depending on the application, the range of possible changes in Δu may need to be restricted for reasons of safety.

All-weight training in a nonlinear NN may result in output bifurcations, i.e., small changes of the weights may cause dramatic changes of the NN output. During continuous on-line training of the adaptive controller its bifurcations may be undesirable. In such a case only an output subset of the controller weights may have to be trained, as it is done in the so-called echo state network (ESN) [12].

IV. Experiments

I first train a NN model to enable off-line training the neurocontroller as discussed in Section II. To do supervised training of the NN model in Figure 5, I gather the input-output pairs from 20 diverse drive cycles generated in the Prius simulator. I trained a 25-node structured RNN for 3000 epochs using the multi-stream EKF with $g = 1$ in (3) [5] and attained the training RMSE of $5 \cdot 10^{-4}$ (the largest generalization RMSE was within 20% of the training RMSE).

The closed-loop control system for training the NN controller is shown in Figure 5. The converter determines the required values of the speed ω_r^d and the torque T_r^d at the ring gear of the planetary mechanism to achieve the desired vehicle speed specified in the drive cycle. This is done on the basis of the Prius model of motion. The constraint verifier assures satisfaction of various constraints which must hold for the engine, the motor and the generator speeds and torques in the planetary gear mechanism, i.e., ω_e and T_e, ω_m and T_m, ω_g and T_g, respectively.

The first control goal is to minimize the average fuel consumed by the engine. However, fuel minimization only is not feasible. The Prius nickel-metal hydride battery is the most delicate nonlinear component of the system with long-term dynamics due to discharging, charging and temperature variations. It is important to avoid rapid and deep discharges of the battery which can drastically reduce its life, requiring costly repairs or even battery replacement. Thus, the second goal of the HEV control is to maintain the battery State Of Charge (SOC) throughout the drive cycle in the safe zone. The SOC can vary between 0.0 (fully discharged) and 1.0 (fully charged), but the safe zone is typically above 0.4.

I combine the two control goals to obtain $cost(t) = \lambda_1 sf^2(t) + \lambda_2(t)(SOC^d(t) - SOC(t))^2$, where $sf(t)$ stands for specific fuel or fuel rate consumed by the engine at time t, $\lambda_1 = 1$, and $\lambda_2(t) \in [10, 50]$ due to about one order of magnitude difference between values of sf and those of SOC. The desired $SOC^d(t)$ is constant in our experiments for simplicity. I encourage the controller to pay approximately the same level of attention to both sf and SOC, although the optimal balance between λ_1 and λ_2 is yet to be determined. I also penalize reductions of the SOC below SOC^d five times heavier than its increases to discourage the controller from staying in the low-SOC region for long. Thus, $\lambda_2(t) = 10$ if $SOC(t) \geq SOC^d$, and $\lambda_2(t) = 50$ if $SOC(t) < SOC^d$.

Ultimately, I would also like to minimize emissions of the harmful gases. In this paper emission reduction is influenced indirectly through reducing the fuel consumption because they are often strongly correlated.

The RNN controller has 5-10R-2 architecture, i.e., five inputs, ten recurrent nodes in the fully recurrent hidden layer, and two bipolar sigmoids as output nodes. The RNN receives as inputs the required output drive speed ω_r^d and torque T_r^d, the current engine fuel rate sf, the current SOC and the desired SOC SOC^d (see Figure 5; the desired fuel rate is implicit, and it is set to zero). (Additional inputs could be the coolant or the battery temperatures, but I have not experimented with adding these to the inputs yet.) The RNN produces two control signals normalized in the range of ± 1. The first output is the engine torque τ_e, and the second output is the engine speed w_e which become T_e and ω_e, respectively, after passing through the constraint verifier.

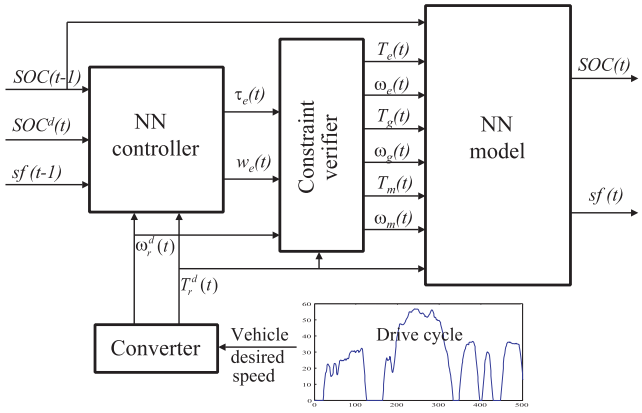

Fig. 5. Block diagram of the closed-loop system for training the NN controller. The converter determines the required values of speed ω_r^d and torque T_r^d at the ring gear of the planetary mechanism to achieve the desired vehicle speed profile. The constraint verifier makes sure not only that the torques and speeds are within their specified physical limits but also that they are consistent with constraints of the planetary gear mechanism. The trained NN model takes care of the remaining complicated dynamics of the plant. The feedback loop is closed via SOC and the fuel rate sf, but the required ω_r^d and T_r^d are guaranteed to be achieved through the appropriate design of the constraint verifier.

The RNN controller is trained off-line using the multi-stream EKF algorithm described in Section II. I train according to the 5-stream EKF algorithm and BPTT(h) with $h = 20$ in which each stream is assigned to a particular instantiation of the NN model (each stream has a slightly different copy of the NN model to imitate parametric and signal uncertainties in the Prius system). Every stream is assigned to its own 50-point segment of the reference trajectory (drive cycle), with the starting point chosen at random. I also choose $SOC^d(0)$ randomly from the range $[0.5, 0.8]$ and keep it constant for the entire drive cycle. All parameters of every NN model and drive cycles are redrawn every 20 training epochs, each epoch consisting of processing all 250 points for all streams. I train for 1200 epochs total (about 60000 weight updates) with $\eta = 0.01$ and $diag(\mathbf{Q}) = 10^{-4}\mathbf{I}$ (these are reasonably

effective values for the training control parameters; I have not optimized them for the best performance yet).

When training of the NN controller from Figure 5 is finished, I can deploy it inside the high-fidelity simulator which approximates well behavior of the real Prius and all its powertrain components. The same simulator is also used to perform the simulated on-line training, as discussed in Section III. As expected, I observed some differences between the neurocontroller performance in the closed loop with the NN model and its performance in the high-fidelity simulator because the NN model and the verifier only approximate the simulator's behavior. My results below pertain to the simulator, rather than its NN approximation.

The basic idea of the current Prius HEV control logic is discussed in [13]. When the power demand is low and the battery SOC is sufficiently high, the motor powers the vehicle. As the power demand and vehicle speed increase, or the SOC reduces below a threshold, the engine is started. The engine power is split between propelling the vehicle and charging the battery through the generator. As the power demand continues to grow, the engine might not be able to stay within its efficiency limits. In those cases the motor can provide power assist by driving the wheels to keep the engine efficiency reasonably high, as long as the battery can supply the required power. During decelerations the motor is commanded to operate as a generator to recharge the battery, thereby implementing regenerative braking.

It is hard to make this *baseline* strategy optimal for such a complex powertrain. A strategy based on a data-driven learning system presents an opportunity to improve over the baseline strategy because of its ability to discern differences in driving patterns and take advantage of them for improved performance.

I compare our RNN controller trained for robustness with the baseline control strategy of the Prius on many drive cycles including both standard cycles (required by government agencies) and non-standard cycles (e.g., random driving patterns). The RNN controller is better by 17% on average than the baseline controller in terms of fuel efficiency. It also reduces variance of the SOC over the drive cycle by at least 25% on average.

Figure 6 shows an example of the neurocontroller results, which should be compared with Figure 7. The latter depicts the baseline controller results on the same drive cycle. The NN controller advantage appears to be in more efficient usage of the engine, e.g., longer idling at higher torque values. The engine efficiency is 37% vs. 31% for the baseline controller. An even bigger improvement is achieved in the generator efficiency: 72% vs. 49%; other component efficiencies remain basically unchanged.

I also test the effectiveness of the proposed on-line training. I employ the combination of Figure 4 with a feedforward NN adaptive controller trained by the SPSA method. The SPSA training parameters a, c and τ should be chosen such that they reduce RMS errors for many drive cycles, a broad set of disturbances, etc. For various values of the

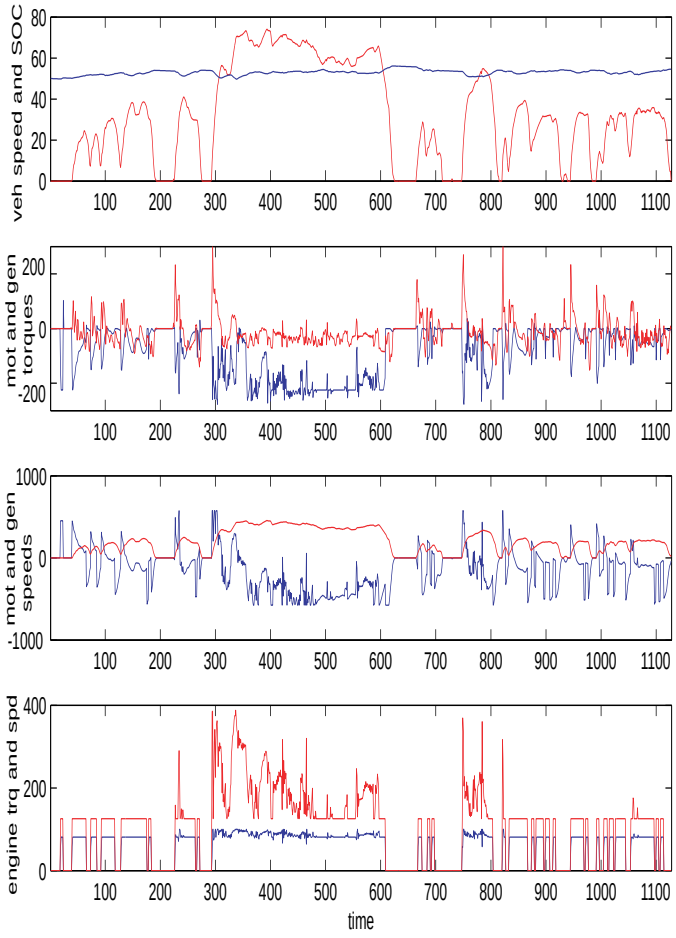

Fig. 6. Illustrative performance of the neurocontroller on a typical drive cycle. The top panel shows the vehicle speed and SOC, the second panel shows the motor (red) and the generator (blue) torques, T_m and $10 \times T_g$, respectively; the third panel shows the motor ω_m (red) and the generator ω_g (blue) speeds, and the bottom panel shows the engine torque T_e (blue) and speed ω_e (red). Note that $\omega_m = \omega_r^d$ due to the system design constraint.

SPSA training parameters obtained by trial and error I often observe improvements between 1% and 10% with respect to fuel efficiency values of the fixed-weight neurocontroller. The observed improvements appear to be accompanied by increased variance of the SOC over drive cycles, sometimes by as much as 8%.

V. Conclusions

The contribution of this paper is two fold. First, I propose an effective combination of off-line and on-line NN training methods which may be applicable to various real-world control problems. I strike a reasonable engineering balance between off-line and on-line methods to achieve improved results. Second, I illustrate the approach on the Toyota Prius HEV through high-fidelity simulation, with results improved over those of the baseline controller.

In this paper I used a neural network model of the plant. This enabled the use of multi-stream EKF as the off-line training method. It is also possible to employ an existing

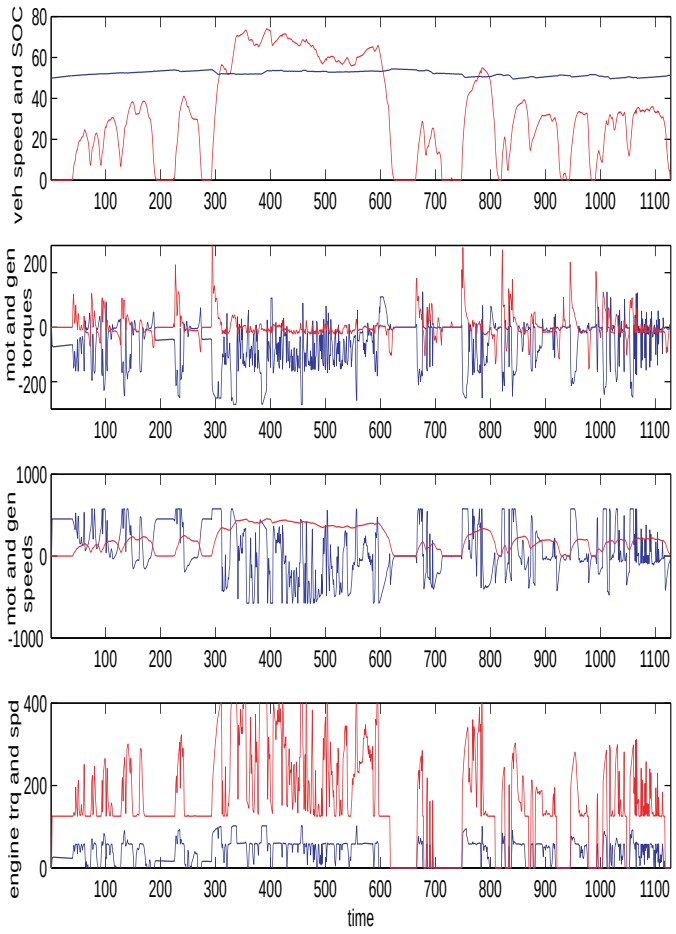

Fig. 7. Illustration of the baseline controller performance. The drive cycle and notation are the same as in Figure 6.

non-neural model of a plant and a *nonlinear* Kalman filter method for neurocontrol as described in [6].

Various issues remain to be clarified with the SPSA method. My choice of the SPSA training parameters is by no means optimal. It may be necessary to resort to adaptation of the SPSA training parameters themselves, depending on the application and the current level of performance. And the adaptive controller architecture may need to be selected in a principled way.

REFERENCES

[1] Baumann, B. M., Washington, G N., Glenn, B. C., and Rizzoni, G, "Mechatronic Design and Control of Hybrid Electric Vehicles," *IEEE/ASME Transactions on Mechatronics*, v.5 n.1, pp. 58–72, 2000.

[2] N. Schouten, M. Salman, N. Kheir, "Fuzzy logic control for parallel hybrid vehicles," *IEEE Transactions on Control Systems Technology*, v.10, n.3, May 2002, pp. 460–468.

[3] Baumann, B., G. Rizzoni, and G. N. Washington,"Intelligent Control of Hybrid Vehicles Using Neural Networks and Fuzzy Logic, SAE Technical Paper 981061, SAE Int. Cong. and Exposition, 1998.

[4] F. Harmon, A. Frank, and S. Joshi, "The Control of a Parallel Hybrid-Electric Propulsion System for a Small Unmanned Arial Vehicle Using a CMAC Neural Network," *Neural Networks*, v. 18, June/July 2005, pp. 772–780.

[5] D. V. Prokhorov, G. V. Puskorius, and L. A. Feldkamp, "Dynamical neural networks for control," in *A Field Guide to Dynamical Recurrent Networks*, Edited by J. Kolen and S. Kremer, IEEE Press, 2001, pp. 257–289.

[6] D. Prokhorov, "Training Recurrent Neurocontrollers for Robustness with Derivative-Free Kalman Filter," *IEEE Trans. Neural Networks*, November 2006, pp. 1606–1616.

[7] G. V. Puskorius, L. A. Feldkamp, and L. I. Davis, Jr., "Dynamic neural network methods applied to on-vehicle idle speed control," *Proceedings of the IEEE*, vol. 84, no. 10, pp. 1407–1420, 1996.

[8] L. A. Feldkamp and G. V. Puskorius, "Training controllers for robustness: Multi-stream DEKF," in *Proceedings of the IEEE International Conference on Neural Networks,* Orlando, 1994, pp. 2377–2382.

[9] D. Prokhorov, "Toward effective combination of off-line and on-line training in ADP framework," in *Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL), Symposium Series on Computational Intelligence (SSCI),* April 1-5, 2007, Honolulu, HI, pp. 268–271.

[10] Publications of Paul J. Werbos at *www.werbos.com*.

[11] J. C. Spall and J. A. Cristion, "Model-free control of nonlinear stochastic systems with discrete-time measurements," *IEEE Trans. Automatic Control*, vol. 43, no. 9, September 1998, pp. 1198–1210.

[12] H. Jaeger and H. Haas, "Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless telecommunications," *Science*, April 2, 2004, pp. 78–80.

[13] D. Hermance, "Toyota Hybrid System," *SAE TOPTEC Conference Proc.*, Albany, NY, May 1999.