Real-Time Implementation and Hardware Testing of a Hybrid Vehicle Energy Management Controller Based on Stochastic Dynamic Programming

Daniel F. Opila
Graduate Research Assistant
Department of Mechanical Engineering
University of Michigan
Ann Arbor, Michigan 48109
Email: dopila@umich.edu [*]

Xiaoyong Wang
Ryan McGee
Ford Motor Company
Research and Advanced Engineering
Dearborn, Michigan 48120
Email: {xwang67,rmcgee3}@ford.com

J.W. Grizzle
Professor
Department of Electrical Engineering
University of Michigan
Ann Arbor, Michigan 48109
Email: grizzle@umich.edu [†]

An energy management controller based on shortest path stochastic dynamic programming (SP-SDP) is implemented and tested in a prototype vehicle. The controller simultaneously optimizes fuel economy and powertrain activity, namely gear shifts and engine on-off events. Previous work reported on the controller's design and its extensive simulation-based evaluation. This paper focuses on implementation of the controller algorithm in hardware. Practical issues concerning real-time computability, driver perception, and command timing are highlighted and addressed. The SP-SDP controllers are shown to run in real-time, gracefully handle variations in engine-start and gear-shift-completion times, and operate in a manner that is transparent to the driver. A hardware problem with the test vehicle restricted its maximum engine torque, which prevented a reliable fuel economy assessment of the SP-SDP controller. The data that were collected indicated that SP-SDP controllers could be straightforwardly designed to operate at different points of the fuel economy tradeoff curve and that their fuel economy may equal or exceed that of a baseline industrial controller designed for the vehicle.

[*]This material is based upon work supported under a National Science Foundation Graduate Research Fellowship. D.F. Opila was supported by NDSEG and NSF-GRFP fellowships.

[†]D.F. Opila and J.W. Grizzle were supported by a grant from Ford Motor Company. Portions of this work have appeared in [1–3].

1 Introduction

Hybrid vehicles are coming on the market at an increasing rate. At the heart of a hybrid vehicle is an energy management controller which determines the amount of power to be delivered by each energy source in the vehicle [4]. In order to improve drivability, power commands may be coordinated with transmission shifts. Many different energy management algorithms have been proposed for an array of vehicle configurations. There are relatively few results in the literature that test such controllers in hardware [5–11], or that address the many practical considerations during the implementation process. There is a significant gap between the number of published results based on simulations and results that report hardware testing.

This paper describes the implementation and hardware testing of an energy management controller based on shortest path stochastic dynamic programming (SP-SDP) [12–14], a specific variant of stochastic dynamic programming [15, 16]. The controller is designed to address both fuel economy and constraints on powertrain activity. The controller design and its simulation-based evaluation using a detailed vehicle model were reported in [17, 18]. Based on the simulation results, it was decided to further evaluate the controller on a prototype Volvo S-80 provided through a University of Michigan and Ford Motor Company alliance; the vehicle is shown in Figure 1.

Three main issues were addressed to obtain a functional controller in the vehicle. The first is the development of a real-time implementation that operates within current computation and memory requirements. One of the oft-perceived

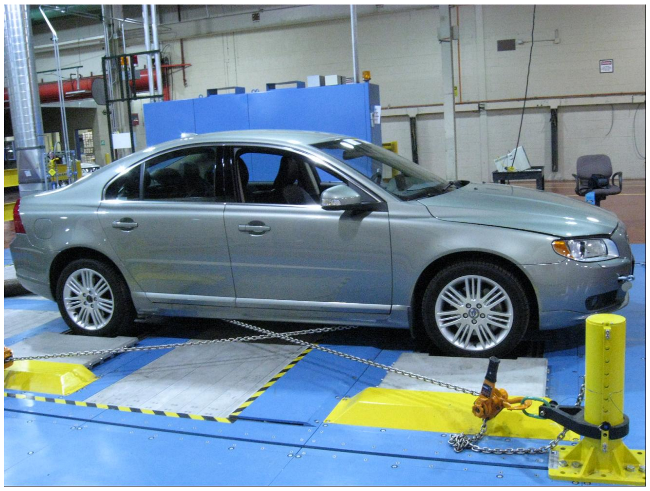


Fig. 1: The Prototype Hybrid: A Modified Volvo S-80.

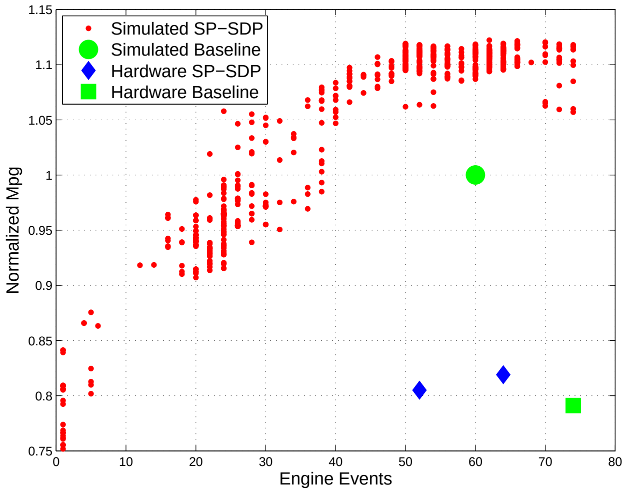


Fig. 2: Fuel economy and engine activity for simulation and hardware testing on the Federal Test Procedure (FTP72). A component failure limited engine torque for all hardware testing, resulting in decreased fuel economy. All results are normalized to the simulated baseline controller.

drawbacks of dynamic programming algorithms is the computational burden. The SP-SDP algorithm used here does require extensive off-line computation, but the on-line computations are shown to be feasible with current technology. The second major issue was to provide rapid pedal response. Most optimization-based energy management algorithms are designed for a 1s update period, but a typical driver will not be satisfied with a 1s delay in pedal response. A multi-rate implementation of the controller is proposed which updates electric machine and engine torque commands rapidly in response to pedal variations, but updates the gear and engine on-off commands more slowly. The third topic is a technique to reliably operate the controller when the execution of an actuator command, such as a gear shift or engine start, takes longer than expected in the model used for controller design.

The controller was implemented in a progressive manner, first in a hardware-in-the-loop (HIL) system, and then in the vehicle. The test environments used in each step are described in detail. Most of the topics addressed in this paper are applicable to any optimization-based energy management controller and are not specific to SP-SDP.

Partway through testing, the engine controller detected a fault and limited engine torque to 150 Nm, whereas 300 Nm is full scale. This issue was not repaired and the fuel economy data reported here reflects this limitation. Figure 2 shows the tradeoff between fuel economy and engine activity for both simulations and hardware tests. The results that were obtained do not contradict the general trends shown in simulation, but certainly there is insufficient data to confirm those trends. The hardware tests do confirm that an SP-SDP controller can be executed in real-time on an an embedded microprocessor, drive cycles, deal with non-ideal real hardware, and generate acceptable vehicle behavior.

The remainder of the article is organized as follows. Section 2 describes the vehicle configuration and the 5 modeling and testing environments used in controller development. Section 3 summarizes the controller design process developed in [18]. Section 4 demonstrates real-time computability of the SP-SDP controller. Section 5 discusses a technique for issuing actuator commands at multiple rates, while Section 6 describes how to handle unpredictable actuator response timing. Section 7 details further refinement to the controller implementation that occurred once it was running in the vehicle. Section 8 provides test data.

2 Vehicle

2.1 Description

The vehicle studied in this paper is a prototype Volvo S-80 series-parallel electric hybrid and is shown schematically in Figure 3. A 2.4 L diesel engine is coupled to the front axle through a dual clutch 6-speed transmission. An electric machine, $EM1$, is directly coupled to the engine crankshaft and can generate power regardless of clutch state. A second electric machine, $EM2$, is directly coupled to the rear axle through a fixed gear ratio without a clutch and always rotates at a speed proportional to vehicle speed. Energy is stored in a 1.5 kWh battery pack. The system parameters are listed in Table 1.

The vehicle hardware allows three main operating conditions:

1. **Parallel Mode**-The engine is on and the clutch is engaged.
2. **Series Mode**-The engine is on and the clutch is disengaged. The only torque to the wheels is through $EM2$.
3. **Electric Mode**-The engine is off and the clutch is disengaged; again the only torque to the wheels is through $EM2$.

These mode definitions do not restrict the direction of power flow. The electric machines can be either motors or generators in all modes.

Table 1: Vehicle Parameters

Engine Displacement	2.4 L
Max Engine Power	120 kW
Electric Machine Power $EM1$ (Front)	15 kW
Electric Machine Power $EM2$ (Rear)	35 kW
Battery Capacity	1.5 kWh
Battery Power Limit	34 kW
Battery SOC Range	0.35-0.65
Vehicle Mass	1895 kg

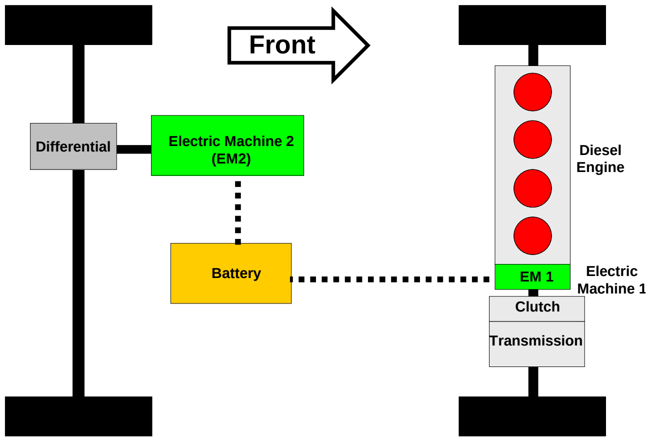


Fig. 3: Vehicle Configuration

2.2 Operational Assumptions

Several operational assumptions were imposed based on the prototype vehicle used. Specifically, the clutch cannot slip to start the vehicle. Starting torque from a full stop is provided by $EM2$. The clutch allows the diesel engine to be decoupled from the wheels. There are no traction control restrictions on the amount of torque that can be applied to the wheels. In terms of the controller, regenerative braking is used as much as possible up to the actuator limits, with the friction brakes providing any remaining torque.

3 Controller Design and Development

The controller design process is briefly summarized here. The interested reader should consult [17, 18] for further information.

3.1 SP-SDP Controller

The controller is designed using Shortest Path Stochastic Dynamic Programming (SP-SDP), which, as explained in [12–14, 19], is a specific formulation of Stochastic Dynamic Programming (SDP) that allows infinite horizon optimization problems to be addressed without the use of dis-

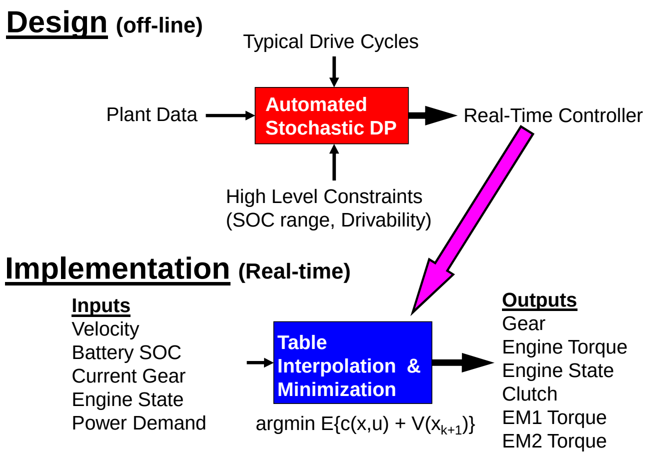


Fig. 4: The overall development process

counting. In the energy management problem, the acceleration requested by the driver, which is the equivalent of a drive cycle, is modeled as a stationary, finite-state Markov chain [20]. The controller minimizes the expected value of a cost function, which was chosen to reflect a tradeoff between fuel consumption and powertrain activity, with the latter measured by accumulated number of engine starts and gear shifts over a drive cycle.

The controllers generated through SP-SDP are causal state feedbacks and hence are directly implementable in a real-time control architecture. The controllers are provably optimal if the driving behavior matches the assumed Markov chain model and the vehicle model is accurate. In this paper, the Markov chains representing driver behavior are modeled on standard government test cycles, as in [12, 20]. It is also possible to build the Markov chains on the basis of real-world driving data, as reported in [17].

The controller design process consists of two steps, one off-line and the other on-line, as shown in Figure 4. The off-line solution of the optimal control problem yields the value function $V^*(x)$ and the optimal control $u^*(x)$, both as a function of the state x . The optimal control is a minimizer of the sum of the current cost $c(x,u,w)$ and the expected future cost $V^*(f(x,u,w))$,

$$u^*(x) = \underset{u \in U}{\operatorname{argmin}} E_w[c(x,u,w) + V^*(f(x,u,w))] \qquad (1)$$

where w is a random variable representing the drive cycle, E_w is the expectation, $f(x,u,w)$ is the system dynamics, U is the set of admissible controls, and V satisfies the Bellman equation,

$$V^*(x) = \min_{u \in U} E_w[c(x,u,w) + V^*(f(x,u,w))]. \qquad (2)$$

A standard iterative method of solving (1) and (2) is given in [13, 14]. The state and control values are first quantized into finite grids. At each step of the iteration, the optimal control and value function are evaluated only at the grid points of the

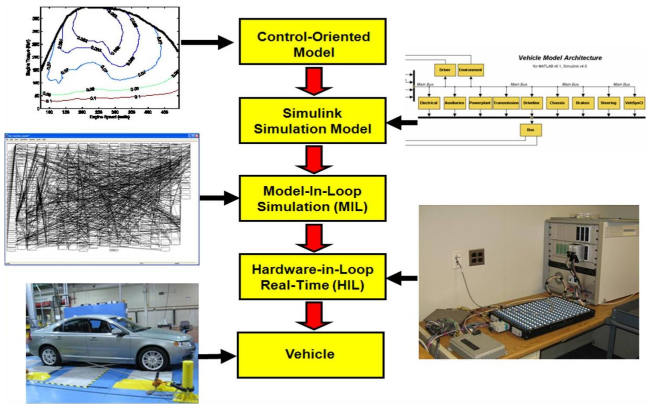

Control-Oriented Model
Simulink Simulation Model
Model-In-Loop Simulation (MIL)
Hardware-in-Loop Real-Time (HIL)
Vehicle
Vehicle Model Architecture

Fig. 5: The increasing complexity of controller testing in this work.

state variables, while the value function at the next time step of the dynamics, $V^*(f(x,u,w))$, is determined off the grid points through interpolation.

3.2 Development Environments

The path from the controller representation (1) to industrial hardware requires dealing with many challenges beyond those faced in most theoretical analyses. The process followed here involved five different models and testing environments, as illustrated in Figure 5. The first two steps have been reported in [17, 18]. Controllers were initially designed and tested on a control-oriented model. They were then extensively evaluated using a High-Fidelity Vehicle Simulation Model.

Model and Testing Environments:

1. **Control-Oriented Model**-Simple, table-based model used for controller design.
2. **High-Fidelity Vehicle Simulation Model**-Ford's in-house model used to simulate fuel economy. Complex, MATLAB/Simulink based model with a large number of parameters and states [21].
3. **Model in the Loop (MIL)**-Simulink-based vehicle model combined with simulated implementation of Ford's real-time vehicle controller, which is a combination of C and autocoded Simulink.
4. **Hardware in the Loop (HIL)**-Vehicle model simulated in real-time on dedicated hardware. Real-time controller runs on actual vehicle processor and interacts with simulated vehicle in real time over the same interface used in the vehicle.
5. **Vehicle**- Full-up testing with real-time controller and vehicle hardware.

After the simulation-based testing showed promising results, the algorithm was implemented in the prototype's real-time Vehicle System Controller, which is a combination of C and autocoded Simulink. The vast majority of the vehicle controller was reused, only the high-level energy management function was replaced. This step is challenging because the SP-SDP algorithm had to interact with all the existing vehicle control modules, such as the engine start sequence, braking logic, and mode switching coordination. The real-time controller was subsequently implemented in a model-in-the-loop (MIL) testbed with a simulated vehicle [22]. The controller was then compiled and run on the actual real-time embedded processor, which was connected to a simulated vehicle in a hardware-in-the-loop (HIL) testbed. The final step was to place the real-time embedded processor in the actual vehicle.

This systematic process allowed progressive development of the algorithm and its real-time implementation. Each step of the process was roughly equivalent in terms of difficulty and time, with attendant opportunities to identify errors and validate results. Section 4 describes the high-level decisions about the algorithm structure, while Section 5 describes the multi-rate implementation. Section 6 addresses unpredictable actuation delays.

4 Real-Time Implementation

The real-time implementation of the optimal control (1) can be done in at least two ways. As mentioned in Section 3, the off-line calculation of the optimal control policy yields both the policy itself u^* and the value function V^* at a set of grid points, say $\{x_i \mid 1 \leq i \leq N\}$, used in the numerical solution of the Bellman equation from dynamic programming. Hence, the optimal policy can be stored as a state-feedback lookup table. To keep the off-line problem computationally feasible, however, the continuous control inputs (engine torque and motor torques) are discretized into a relatively coarse grid of about 20 possible values. The stored optimal feedback policy would carry this coarse discretization, namely, $u^*(x_i) \in \{u_j \mid 1 \leq j \leq 20\}$, with the nearest neighbor interpolation being used, for example, to define the controls at state values not in the grid used for computing the optimal policy.

It was observed in [12, 23] that a better approximation of the optimal policy can be obtained as follows. The value function $V^*(x_i)$ is stored at grid points and $V^*(x)$ is approximated by linear interpolation. The optimal policy is determined by on-line minimization of (1), in which the engine torque control input is discretized into 100 possible values, yielding increments of 3 Nm. Because the minimization involves selecting a value from a discrete set of fixed size, its execution is fast and deterministic. Simulations have shown that this on-line refinement of control inputs is important, yielding 2-3% better fuel economy than simply implementing the coarse policy $u^*(x_i) \in \{u_j \mid 1 \leq j \leq 20\}$. Minimizing with a continuous control input does not yield significant improvement over a control space with 100 values. Conducting the minimization in (1) on-line allows the flexibility to incorporate additional features, as will be discussed later.

4.1 Code structure

The calculations represented by (1) were coded in Simulink for the MIL and subsequent models to allow easy integration with the existing Vehicle System Controller, automatic code generation, and interaction with MATLAB. The HIL was used to set table sizes, memory allocation, and assess precision. Three functions must be stored as tables: the cost function $c(x,u)$, the system dynamics $f(x,u,w)$, and the value function $V^*(x)$. The largest table was actually the system dynamics function $f(x,u,w)$, rather than the value function $V^*(x)$. The size of these stored tables scales with the desired numerical accuracy of the solution. The expected values in (1) can be pre-computed to reduce the on-line computation.

For each update, the algorithm is passed a 2D array of 700 possible control choices along with the current state. This array represents 100 possible engine torques and 7 possible transmission states including the six gears and series mode. The instantaneous expected cost of each possible control along with the expected future cost $E_w[V(f(x,u,w))]$ are similarly stored as a 700 element array. Selecting a cost-minimizing command is then a matter of determining the minimum total cost.

4.2 Results

The vehicle control system runs on a DSPACE MicroAutoBox DS1401 which contains an IBM PPC603 processor at 300 MHz with 8 MB of local RAM. The baseline Vehicle System Controller uses this hardware, and the SP-SDP energy management software is added to the existing control system. Both controllers continuously run in parallel to enable easy transitions between the two.

The compiled version of the SP-SDP controller requires 900 kB of memory, including all code and data tables. All calculations are implemented in a manner identical to simulation work reported in [17, 18]. The dynamics $f(x,u,w)$ are implemented analytically, with the exception of the next SOC, which is stored as an array of 54,978 single-precision values. The array size was limited by 16-bit memory addressing (65,536 points). Simulations typically used an array 5 times larger, but little numerical precision is lost with the smaller table. The value function is stored as an array of 21,384 double precision values, the same size as used in simulation.

The SP-SDP controller calculations, running in parallel with the baseline controller, could be completed in less than 16ms on the HIL. Because the controller easily ran with the available resources, little effort was expended to reduce computation requirements. The controller would likely run on a significantly less powerful processor.

5 Multi-Rate Updates to Enhance Driver Perception

Hardware implementation of any energy management controller requires dealing with issues that are commonly ignored in simulation studies. Update rate is the focus of this section. The SP-SDP controllers used here are designed to update with a $\sim$1s period, as are many energy management controllers in the literature. One reason for using a relatively slow update rate in the controller design process is that computing the value function for shorter time steps requires greater numerical accuracy and yields slower off-line solution convergence. Decreasing the sample interval below $\sim$1s is difficult because discrete control actions, such as engine start and gear shift, take roughly one second to complete. Shorter update periods would invalidate the simple gear and engine on-off state representation used in our controller design model, where intermediate states, such as a partially started engine, were not used. The $\sim$1s update captures the relevant dynamics of the system while ignoring fast transients. While controllers with this relatively slow update rate can follow drive cycles, deliver good fuel economy, and in general look good in simulation, a real driver is bothered by a pedal with a 1s lag.

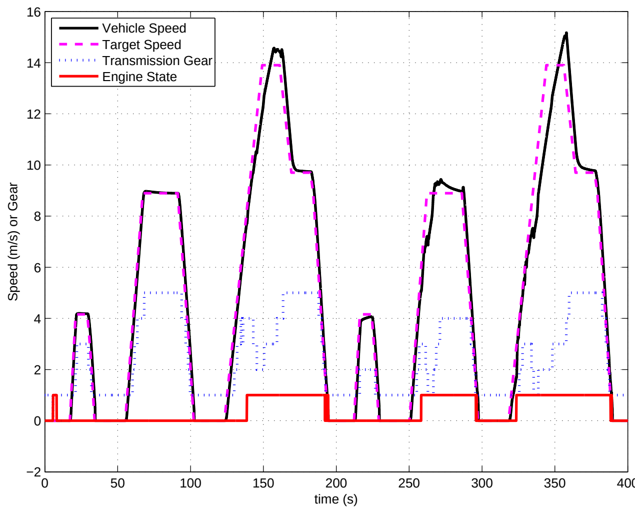


Fig. 6: One of the early HIL simulations on NEDC. The vehicle speed, gear, and engine state show reasonable behavior. The automated driver model was not well-tuned at this point, so the velocity tracking shows some lag and overshoot.

5.1 Multi-rate updates

Our solution to perceived pedal lag was a multi-rate implementation of the controller in which actuators are updated at different rates based on their capability, as illustrated in Figure 7. The engine on-off and transmission gear transitions are relatively slow and thus are updated with a period of 1.2 s, or 0.83 Hz. Both engine torque and sound are very noticeable to a driver, so commanded engine torque is updated at 2.5 Hz to provide improved pedal responsiveness. Finally, the two electric machines are updated at 60 Hz to yield fast pedal response.

The real-time controller continued to execute without problems in the embedded hardware at these faster rates. Figure 6 shows the result of early testing of the multi-rate updates on the HIL simulation. The simulated vehicle shows reasonable behavior, although with poor velocity tracking due to a poorly tuned automatic driver.

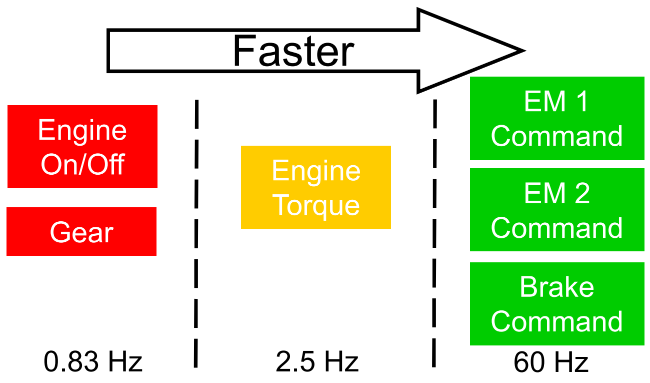


Fig. 7: Multi-rate actuator commands

5.2 Implementation details

The process of selecting control commands is shown pictorially in Figure 8a. Possible control choices are organized as a matrix, where the columns correspond to transmission gear commands plus series mode, and the rows correspond to possible engine torque values. The series column on the far right in Figure 8a represents the clutch disengaged, and one entry in this column is used to represent engine off. The value in each entry of the matrix is the estimated total cost for the corresponding control values. The min operation selects the entry of minimum value, with the columns and row indexes of the minimum providing the control commands. For a given vehicle speed, certain gear choices will be infeasible because they violate a constraint, such as an engine speed limit, and are disallowed when the minimum is computed. As described in [17, 18], the required electric machine torques are uniquely determined from engine torque, transmission gear, and driver power request.

Figure 8b represents a time series of controller updates. At each 1.2s update, a matrix of controls is evaluated. At the initial time step t=0, four possible gears are valid, and the algorithm selects one. This fixes the gear and engine state commands over the next 1.2s interval. At the intermediate updates, t=0.4 and t=0.8 in the figure, engine torque is recomputed with a constrained minimization of (1) over the limited control space

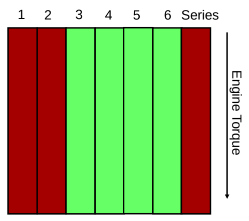
$$u(x) = \underset{u \in U_{eng,gear}}{\operatorname{argmin}} \; E_w[c(x,u,w) + V^*(f(x,u,w))], \qquad (3)$$

where $U_{eng,gear}$ restricts the engine state and gear to the values at the last full update. When the next 1.2s interval occurs, the engine and gear commands are once again updated.

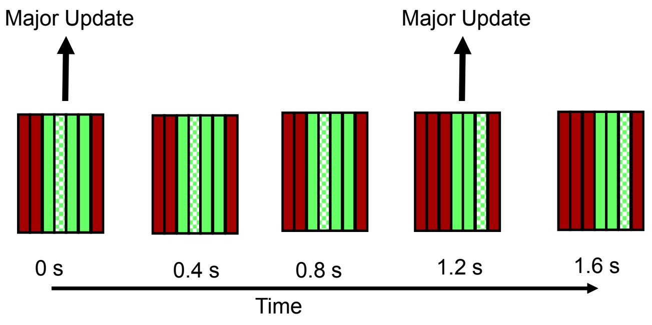
The 1.2s engine and gear update period for the hardware test is longer than the 1s interval used in the simulations reported in [17, 18] to increase the likelihood that the engine and transmission will execute their commands before the next time step. Actuator response time is discussed next.

6 Variable Actuator Response Times

The controller design and simulation models assign discrete values to states that are actually somewhat continuous.



(a) For a given vehicle state, the expected costs for possible control choices are strategically organized as a matrix. The columns represent the 6 possible transmission gears along with series mode. Possible engine torques are rows in the matrix. Electric mode (engine off) is represented by a zero torque point in the series mode. For a given vehicle state, some gear choices will be infeasible, as shown by the dark (red) columns, and others will be feasible, as shown by the light (green) columns.



(b) Transmission gear and engine state commands are updated with a 1.2s sample interval, while engine torque commands are updated at a 0.4s sample interval. The selected column for gear and engine state is hashed, and is fixed for the intermediate updates at 0.4, 0.8, and 1.6s.

Fig. 8: Illustration of the command update scheme. Controls are organized to easily permit updates at multiple rates while respecting appropriate constraints.

For example, the controller design model assumes the engine is either on or off. The more detailed simulation model assumes the engine will spin up to its nominal idle speed within 1s, and then be available to provide torque, supporting the controller design model. In the prototype vehicle, however, it was observed that starting the engine and engaging the clutch sometimes takes 1.5s or longer. Consequently, a controller update may occur at a time when the engine has not completely started, and is thus *neither on nor off.* In a similar manner, the transmission gear command may not complete within 1s. A related but different issue is that the transmission controller may override the gear requested by the energy management controller.

The basic SP-SDP controller used here is not equipped to handle these problems as it assumes the engine can only be on or off, the clutch is fully engaged or not, and the transmission is in a specific gear.

A ready solution is available when commanding torque at the 0.4s sample intervals: the engine is considered off until ready to deliver torque, which is a discrete yes/no signal available in the engine control module. Furthermore, engine torque is computed on the basis of current gear, as reported by the transmission controller module, independent of what gear was commanded at the previous 1.2s update.

Issuing correct engine on-off and gear change commands at the 1.2s update intervals is more subtle. As an example, if the engine is off, at some point in time the algorithm will issue an engine start command. At the next major update (1.2s later) the engine may be in the process of starting but not fully started. If the engine is considered off, the optimal decision may be to leave the engine off, that is, issue an engine-off command. The resulting off-on-off change in the engine state is very undesirable. A similar scenario is clear for the transmission hunting between gears.

The solution is for the algorithm to issue new engine state and transmission commands assuming that the commands issued at the previous update have in fact completed, even if they are still in the process of being completed. This yields much more consistent behavior.

The ability to deal with delayed or uncertain actuation completion was quite useful. In the final implementation, the transmission manufacturer was unable to modify the transmission controller to accept external gear commands over the vehicle CAN bus. Consequently, gear selection was made by the existing transmission controller, while the SP-SDP controller handled engine and clutch state. The controller implementation described above could be used with no further modifications. The inability to independently command gear turned out to be a only a minor setback to the controller evaluation process for two reasons: comparison of the SP-SDP gear commands to the gear selection made by the transmission controller revealed almost no differences; and the previous simulation work had showed that fuel economy of the prototype vehicle is more sensitive to engine on-off activity than gear number.

7 Refining the Controller in Hardware

After standard testing and debugging in the MIL/HIL setup, the SP-SDP controller was tested in the vehicle on a two-axle dynamometer as shown in Figure 1. The vehicle is chained in place, and electric motors on the dynamometer rolls simulate the vehicle drag and rolling losses. The algorithm is implemented in the on-board vehicle system controller and is transparent to the driver. The driver uses standard controls and pedals, while a laptop provides real-time vehicle monitoring and data capture. Desired vehicle behavior is set off-line by changing the penalties used in the cost function, solving the optimal control problem, and building the associated look-up tables. The look-up tables for several different controllers are stored simultaneously in the real-time processor and can be selected without recompiling.

7.1 Initial Test

The SP-SDP algorithm worked correctly the first time thanks to extensive validation in the MIL/HIL setup. Figure 9 shows data from one of the initial tests with a human driver following the New European Drive Cycle (NEDC). At this point, the controller deliberately had limited functionality: it used neither the front electric machine $EM1$ nor series mode. The controller provided reasonable behavior and performance on the first set of hardware tests with no debugging or tuning. The poor velocity tracking is primarily due to the human driver; tracking cycles is quite difficult and takes practice.

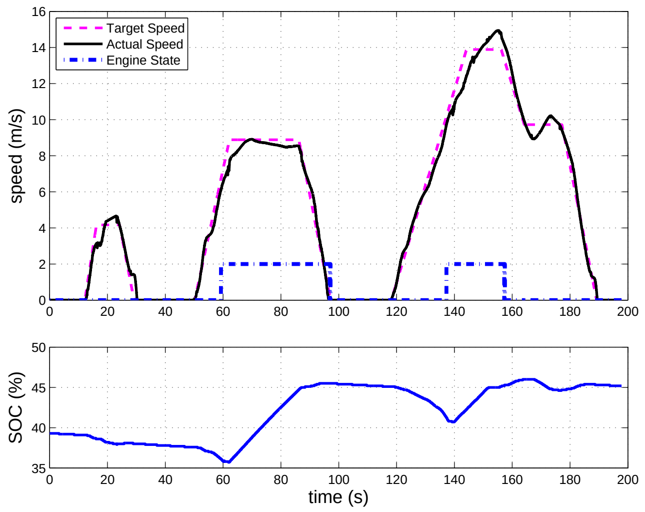


Fig. 9: The first driving attempt in the vehicle, which corresponds to the first three hills of NEDC. The difference between targeted and actual speed is due to driver inexperience. Accurately following speed traces on a chassis roll dynamometer is an acquired skill.

Full controller functionality, including series mode and the front electric machine $EM1$, was implemented in a second step. The interaction with the existing vehicle controller became more complex as these additional vehicle modes were used. The MIL, HIL and vehicle itself were all used in this part of the development process.

7.2 Model Improvement

Once the controller was fully functional and had successfully driven a number of cycles, the test data were analyzed to check the accuracy of the controller design model. In a vehicle prototyping process such as the one reported here, it is common for hardware changes to be made without models being fully updated. Another source of inaccuracy is the model reduction process used to obtain the controller design model, where dynamics are neglected, lumped, and simplified.

Some of these simplified dynamics depend partly on the *controller*, rather than the vehicle model. One would like to imagine the model as not depending on the controller, but this is not always the case with reduced-order models. One example is engine start. The control-oriented model lumps the process of starting the engine and engaging the clutch into three parameters: the time to execute a start, the fuel burned during the start, and the battery charge used to spin up the engine. In hardware, the way the baseline and SP-SDP controllers execute this process is similar but not identical. Therefore, the parameters for the reduced-order model are different depending on which controller is used.

Using the test data, the basic vehicle parameters were identified and adjusted in the control-oriented model. The

SP-SDP controllers were recomputed on the basis of the updated model.

7.3 Smoothing torque delivery

The next phase of the development addressed issues relating to driver perception that only became apparent when we were driving the vehicle on the chassis rolls. One such issue was the smoothness of torque delivery. The SP-SDP controller updates commands in a discrete fashion, including continuous variables such as torque. In the absence of discrete events, such as shifts or engine starts, drivers expect the torque delivery to be smooth. The discrete SP-SDP updates can occasionally yield jumps in engine torque that feel jarring. A set of low-pass filters and initialization values was developed to yield a smooth, yet responsive, torque command.

7.4 Infeasible Conditions

At the intermediate update times, depicted by 0.4s and 0.8s in Fig. 8b, it can happen that it is impossible to meet the driver's power demand with the current engine state and gear number. When no valid torque commands are available at an intermediate update, a full update of all controls occurs regardless of the normal waiting time. Such updates are termed "feasibility" updates.

Feasibility updates are especially important in two cases. The first is for a so-called "gorilla stomp" in which the driver suddenly demands large torques that are unavailable in electric mode or higher gears. The full update occurs immediately, forcing an engine start or a downshift. The transition out of electric mode is especially important for driver perception because otherwise the driver could wait more than a second before hearing an engine start. A second case is during rapid deceleration with the engine on. The clutch cannot remain engaged below a certain vehicle speed or it will pull the engine speed below its minimum allowed value. It can happen that first gear will be valid at one full update, but become invalid before the next full update due to vehicle speed change. Executing a feasibility update allows the clutch to be disengaged at one of the intermediate updates if the current gear becomes infeasible.

7.5 Engine torque oscillation

During testing, it was observed that the engine torque would oscillate while the vehicle was seemingly at steady state. One example of this is shown in Figure 10, where commanded engine torque is varying by 100 Nm while the pedal position is nearly constant. Similar events were never observed in simulation. In the vehicle, they occurred at low pedal and nearly constant vehicle speed, usually around 25 kph. Because the pedal input is almost constant, there is no obvious reason for these torque oscillations. Such behavior is clearly unacceptable to a driver.

The underlying reason becomes clear when studying the SP-SDP total cost estimate. The engine torque command is selected by minimizing the total cost per (1) at a major up-

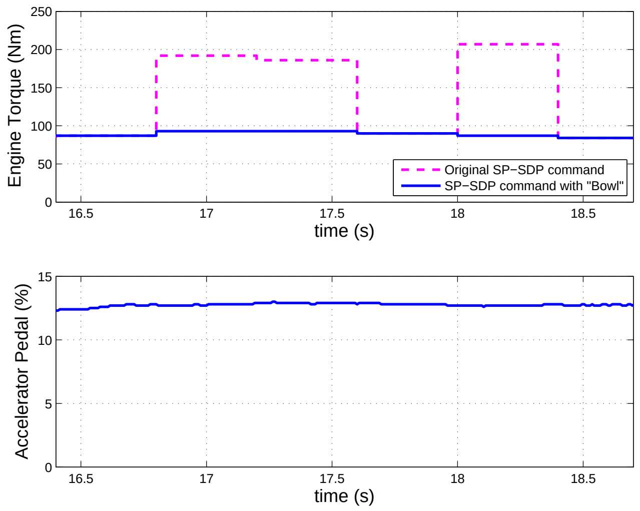


Fig. 10: Engine torque and driver pedal commands. The upper plot shows the original SP-SDP engine torque command as a dotted line (red) oscillating at relatively constant pedal. The solid (blue) line shows the command after this problem was fixed with a "bowl" penalty. Both commands are the raw output of the SP-SDP algorithm before low-pass filtering. The bottom plot shows accelerator pedal command in percentage of full range.

date, or (3) at an intermediate update. The left column of Figure 11 shows the total cost versus engine torque at successive intermediate update times, that is, multiples of 0.4 s. It is seen that the total cost function has two local minima that are very close in value, indicated by the dotted vertical lines in Figure 11. Small variations in vehicle state are causing the torque command to oscillate between the two values.

In the design of the SP-SDP controller, the rate of change of engine torque was not considered. The algorithm is free to use a jump in torque so as to minimize cost. Although perceptible jumps in torque rarely occurred, they were disconcerting and we sought to eliminate this behavior.

One possible solution is to augment the controller design model with a state that stores the last commanded engine torque, per $x_e = (x, u_{eng}^{last})$, and modify the cost function to penalize rapid torque changes,

$$c_e(x_e, u) = c(x, u) + c_{aug}(u_{eng} - u_{eng}^{last}). \tag{4}$$

A new value function would be computed for the modified cost function and the controller implemented as in (1) using the augmented state,

$$u^*(x_e) = \underset{u \in U}{\operatorname{argmin}} E_w[c_e(x_e, u) + V_e^*(f_e(x_e, u, w))]. \tag{5}$$

We estimated that this approach would increase off-line computation of the value function by roughly a factor of ten.

Instead we choose to implement an idea from [23] and [1], which incorporates additional control objectives into the

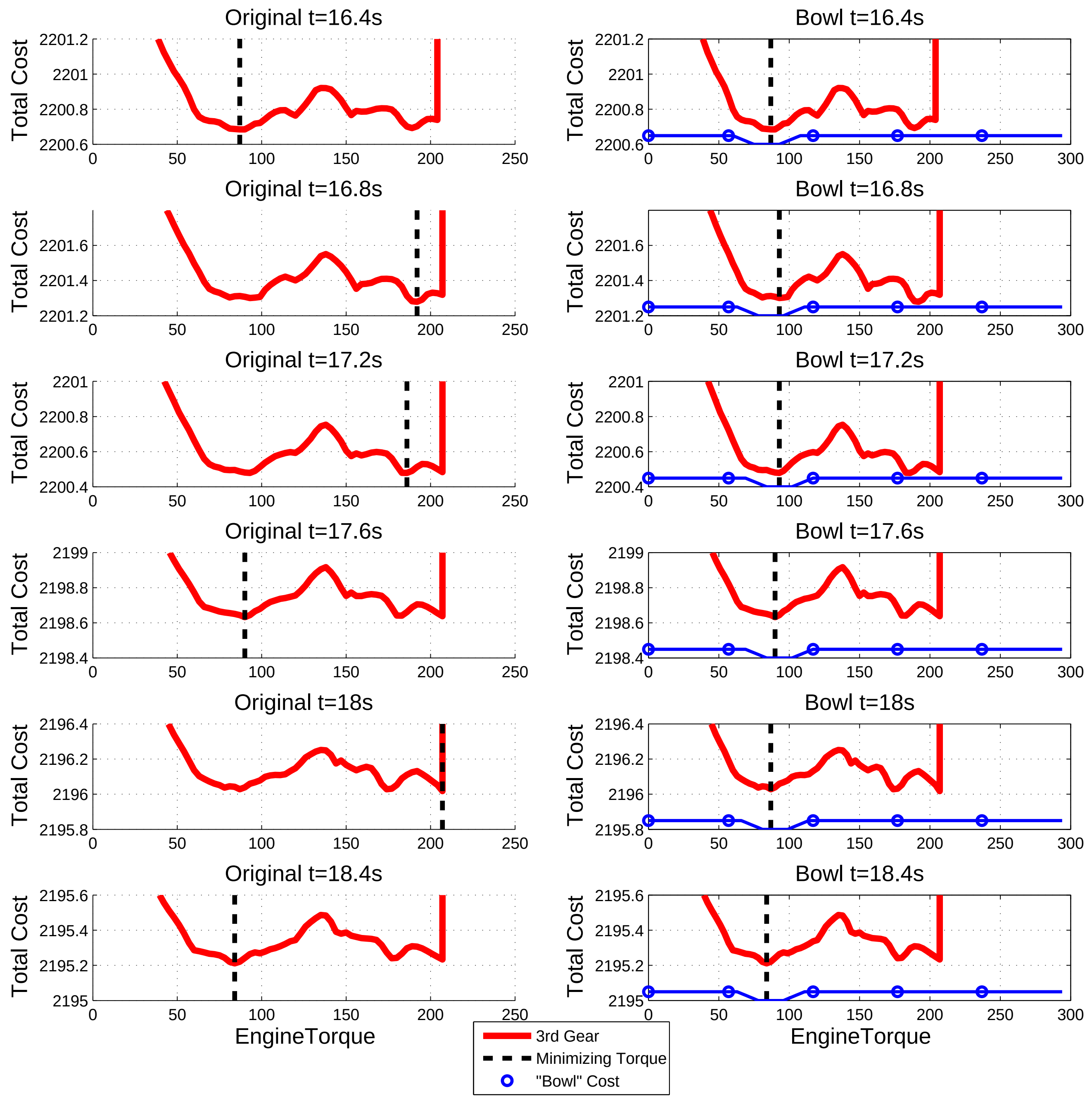


Fig. 11: The value function during an unexpected pedal oscillation. The recorded vehicle data is shown on the left for controller updates at 6 consecutive time steps. The total expected cost for each possible engine torque command is shown as a heavy solid line (red). The minimizing torque selected by the controller is indicated by the vertical dashed line (black), and demonstrates the cause of the oscillations shown in Figure 10. This oscillation is removed by adding a "bowl" penalty on engine torque which adds a cost for torque changes. The column on the right represents this improved control response applying (6) to the same vehicle data. The bowl penalty is marked with a solid line and circles (blue) at the bottom of each plot. The bowl penalty is centered at the last commanded torque and visibly changes position from t=16.8s to t=17.2s due to the change in torque command. The minimizing torque selection no longer oscillates.

running cost $c(x,u)$ without recomputing the value function. The real-time controller is then

$$u^*(x_e) = \operatorname*{argmin}_{u \in U} E_w[c_e(x_e,u) + V^*(f(x,u,w))]. \qquad (6)$$

The penalty term $c_{aug}(u_{eng} - \tau_{eng}^{last})$ in (4) was selected to have a "bowl" shape as shown in Figure 12. The penalty is zero for small torque changes and saturates to allow large jumps in torque if they are sufficiently less costly. The most important parameter is the saturation value, which is set just high

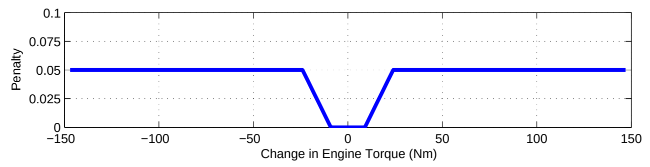


Fig. 12: Additional penalty added to the value function based on the change in engine torque. This is termed a "bowl" penalty due to its shape.

enough to eliminate "frivolous" oscillations.

This method is computationally very advantageous, but suboptimal. The modified controller (6) was evaluated on the detailed simulation model and no reduction in fuel economy was observed. The modified cost term was then implemented on the vehicle, with results given in the right column of Figure 11. The actual bowl penalty is shown as a line in the bottom of each plot. The bottom of the bowl moves with the last commanded torque, and the total penalty is very small compared to other variations in the cost. This small penalty is sufficient to eliminate the torque oscillations, as shown by the minimizing values and the torque command in Figure 10.

8 Hardware testing results

The test vehicle experienced a hardware failure that was not repaired; more on this given in Sec. 8.3. The results in this section reflect the malfunctioning vehicle. The controllers still function largely as designed, though the fuel economy numbers are unreliable.

8.1 Overall performance

Figure 13 shows the baseline controller and two SP-SDP controllers driving the Federal Test Procedure, FTP72 cycle. The two SP-SDP controllers use a different penalty for engine start/stop, yielding different behavior observed in the data. Changing the penalties does effectively modify vehicle behavior, as predicted by the simulation studies in [17, 18].

Figure 14 presents three controllers run on NEDC. As predicted by the analysis in [17, 18], on NEDC, the two SP-SDP controllers yield similar numbers of engine events. This is due to the contrived nature of NEDC; because it is composed of repeated ramps with constant acceleration, engine starts will naturally occur at the same places unless large penalties are used to change behavior.

The raw fuel economy results are shown in Table 2 along with the final SOC deviation. Both raw and corrected fuel consumption (i.e., adjusted for difference final and initial SOC) are normalized to the baseline controller running in hardware on FTP72. .

The vehicle fuel economy and engine activity of the baseline and SP-SDP controllers are plotted along side the values from a simulation study for FTP72 cycle in Figure 2. In this case, all fuel economy values are normalized to the simulated baseline controller. This type of plot is used to characterize the optimal tradeoff curve between fuel economy and engine activity as discussed in [17, 18]. The vehicle test data suggest a trend similar to the simulated tradeoff curve, but the limited number of cycles executed on the vehicle, due to the hardware failure, makes any meaningful comparison impossible.

8.2 Hardware fault

The torque-speed engine operating points are shown in Figure 15 for the baseline controller and the SP-SDP controller running FTP72. The plots show both the commanded torque and delivered torque. The nature of the hardware failure is clear: the engine control unit is clipping the commanded engine torque. The source of the error was not identified before the termination of the project.

The dark black line in Figure 15 is the operational limit for noise and vibration specified during the design phase. The SP-SDP controller generally respects this constraint, although the baseline controller calculates the limit differently. The SP-SDP controller slightly overshoots the limit when operating on the boundary if the engine speed drops before the next engine torque update.

8.3 Detailed Vehicle Response

For a more detailed view of the system dynamics, a zoomed view of the third NEDC "hill" is shown in Figure 16. The vehicle accelerates from rest in electric mode, the engine starts, the transmission engages, and the engine begins delivering torque. Transmission gear shifts are clearly visible as sawtooth profiles in engine speed. The bottom two plots show the electrical dynamics, namely SOC as well as the $EM1$ and $EM2$ commands. Before the engine starts, the vehicle is propelled by $EM2$ only and the SOC drops. $EM1$ is then used to start the engine. The engine then provides the motive power and charges the battery through $EM1$, while $EM2$ is idle. After the engine shuts off, the vehicle is again in electric mode with $EM2$ providing propulsion and braking.

As mentioned in Section 6, the engine start dynamics are more complex than originally modeled. The engine start event of Figure 16 is shown in greater detail in Figure 17. The SP-SDP controller selects parallel mode, so the low-level controllers start the engine and engage the clutch. This command is issued as "Parallel Mode Request" at 124.5s, and the "Parallel Mode Actual" responds at 126s once the engine is on and the clutch engaged. The engine start is executed by a low-level controller. During the start process, $EM1$ applies positive torque to spin the engine. The torque delivered to the wheels is zero until the clutch engages. Once the clutch is engaged, the SP-SDP controller starts issuing torque commands. The SP-SDP torque command is initialized at the engine torque estimate from the Engine Control Unit to avoid rapid transients.

9 Conclusions

An energy management controller based on Stochastic Dynamic Programming has been successfully implemented in a prototype HEV. Theoretical and practical issues affecting

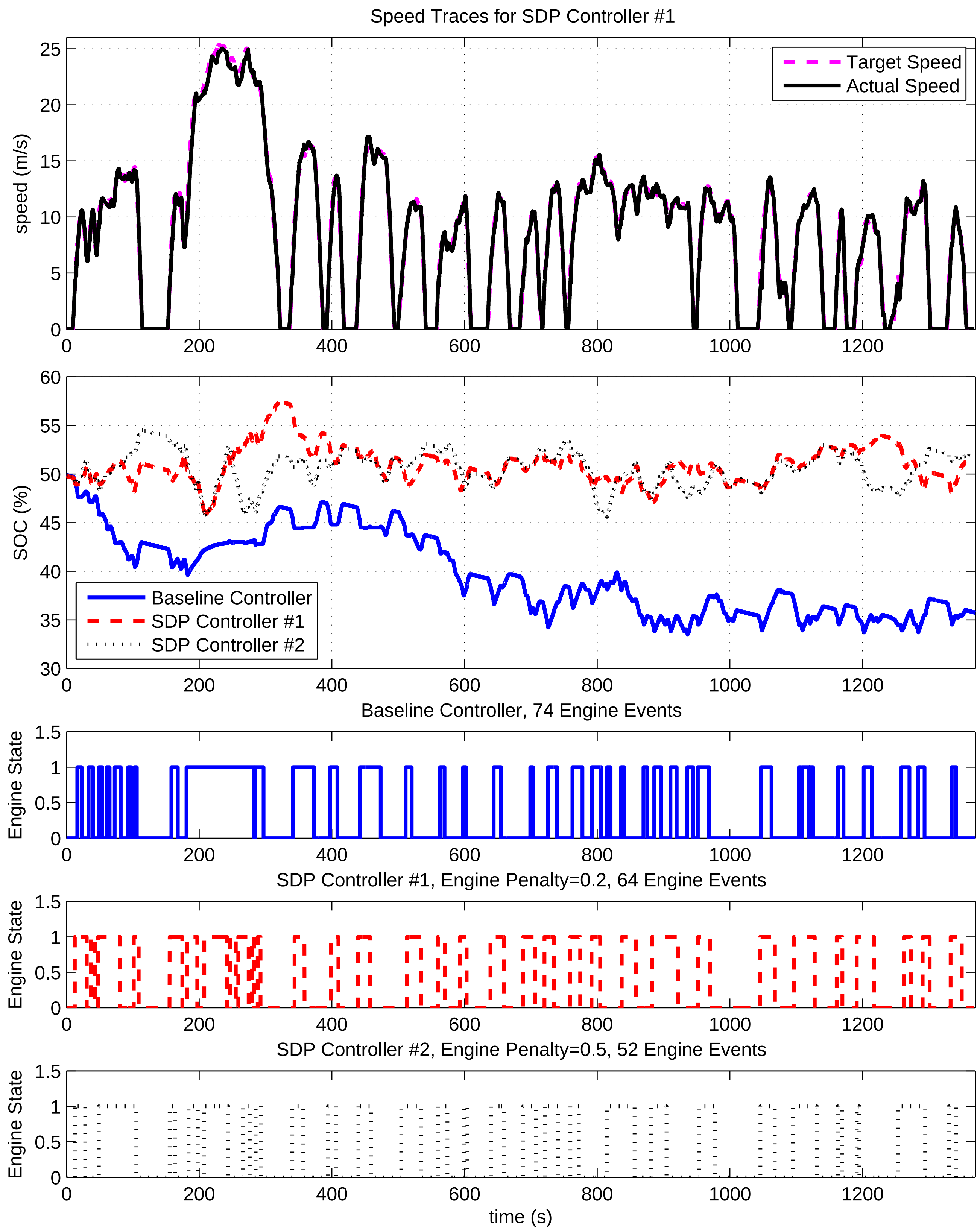


Fig. 13: Driving the FTP72 cycle with the baseline controller and two different SP-SDP controllers.

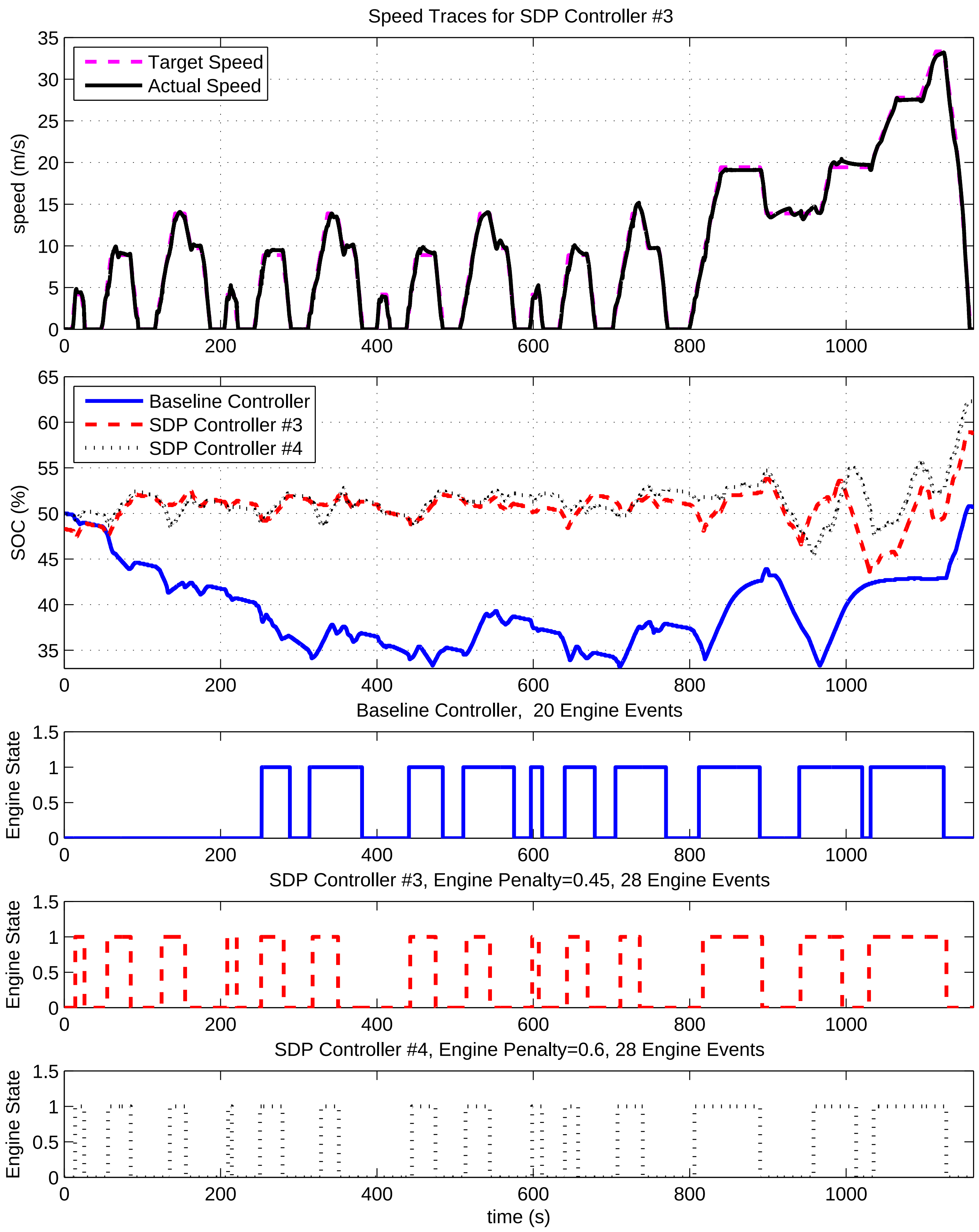
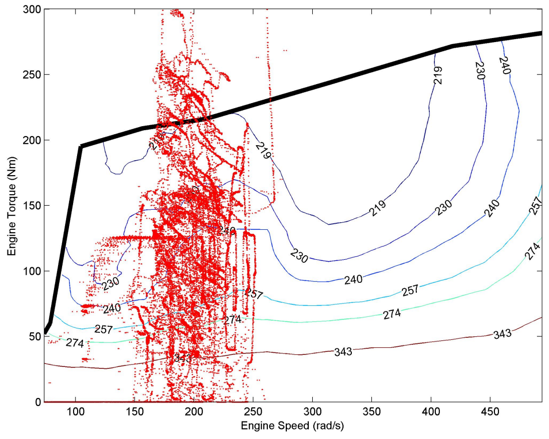


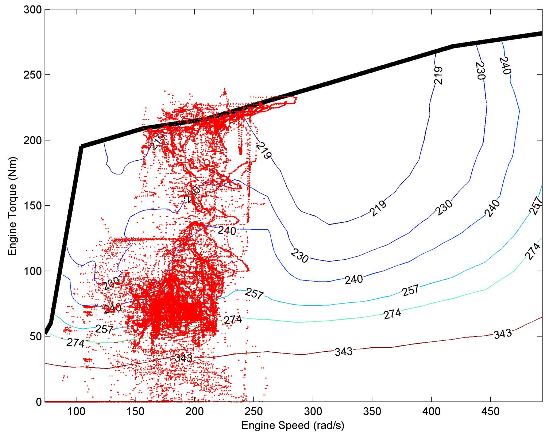
Fig. 14: Driving the NEDC with the baseline controller and two different SP-SDP controllers.

Table 2: Fuel economy summary for malfunctioning hardware with SOC correction

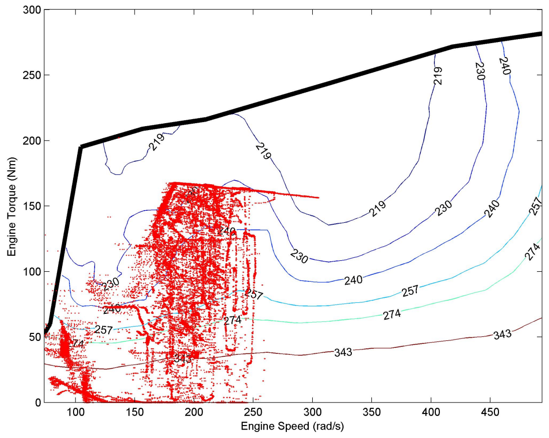
Controller	Cycle	Normalized Uncorrected Fuel Economy (MPG)	ΔSOC	Normalized Corrected Fuel Economy (MPG)	Improvement	Engine Events
Baseline	FTP72	1.078	-14.0%	1.000		74
SP-SDP	FTP72	1.028	1.4%	1.035	3.528%	64
SP-SDP	FTP72	1.006	2.2%	1.018	1.764%	52
Baseline	NEDC	0.966	0.5%	0.969		20
SP-SDP	NEDC	0.921	12.2%	0.983	1.439%	28
SP-SDP	NEDC	0.933	16.7%	1.024	5.662%	26



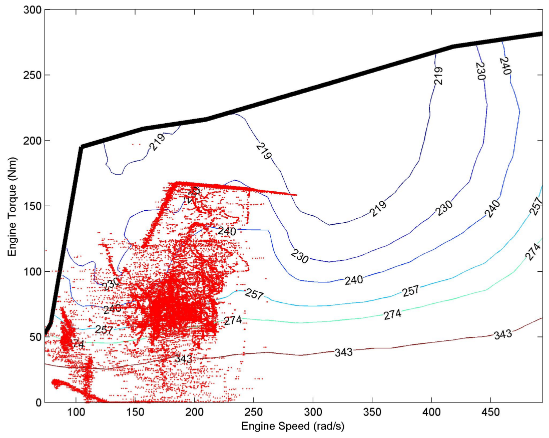
(a) Baseline controller: commanded torque-speed operating points on FTP72



(b) SP-SDP controller: commanded torque-speed operating points on FTP72



(c) Baseline controller: achieved torque-speed operating points on FTP72



(d) SP-SDP controller: achieved torque-speed operating points on FTP72

Fig. 15: Engine torque-speed operating points demonstrating the effects of a hardware failure. The plots on the left (15a and 15c) show the baseline controller, while the plots on the right (15b and 15d) show the SP-SDP controller. The top plots show the commanded torques, while the bottom plots show the delivered torque. The engine control computer clips the delivered torque at about 170 Nm.

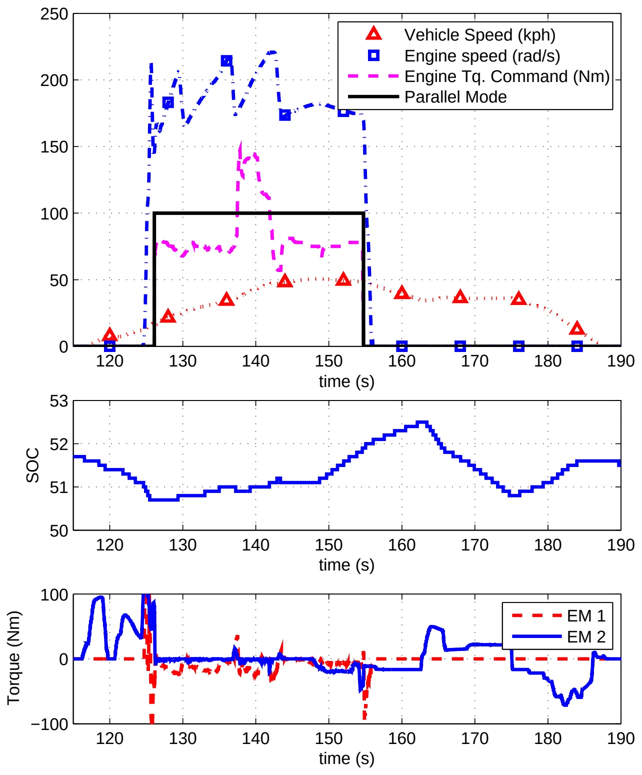


Fig. 16: Detailed actuation traces for one “hill” of NEDC.

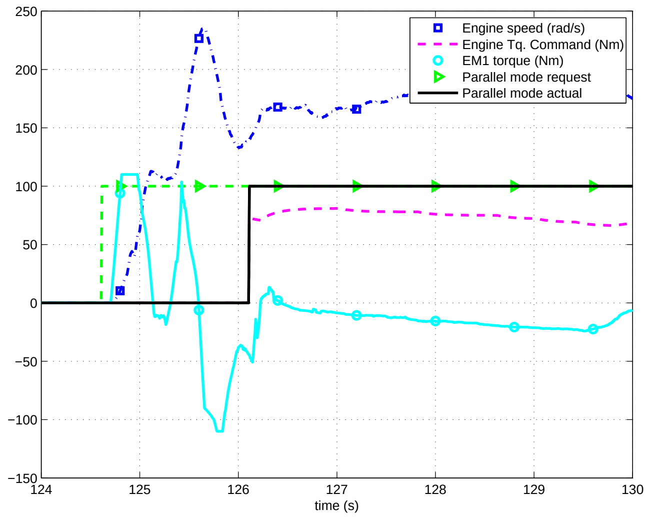


Fig. 17: Zoom of the engine start dynamics. Note the 1.5s delay between the parallel mode request and actual clutch engagement. $EM1$ is attached to the crankshaft and provides engine starting torque.

real-time implementation were addressed. The controllers run in real-time on embedded hardware with typical automotive computing capacity.

Optimization-based energy management algorithms are typically designed based on relatively slow ($\sim 1s$) update rates, but directly implementing such an algorithm would yield poor driving characteristics. A method was developed and tested that allows different update rates for various actuators to improve driver perception of pedal response. The proposed implementation specifically deals with actuator delays and infeasible operating points.

These results demonstrate the practical feasibility of using advanced optimal control techniques for energy management controller design. There is a large gap between the simple models used for optimization in the literature and the tremendous complexity of production vehicle controllers. Although additional issues must be addressed in a real vehicle, the fundamental optimization based on relatively simple models is effective. The controllers can be directly implemented in hardware and yield good performance with minimal manual tuning.

Acknowledgements: The authors would like to thank Ameet Deshpande for his assistance in the real-time implementation of this work.

References

[1] Opila, D., Aswani, D., McGee, R., Cook, J., and Grizzle, J., 2008. “Incorporating drivability metrics into optimal energy management strategies for hybrid vehicles”. In Proc. IEEE Conference on Decision and Control, pp. 4382–4389.

[2] Opila, D., Wang, X., McGee, R., Cook, J., and Grizzle, J., 2009. “Performance comparison of hybrid vehicle energy management controllers on real-world drive cycle data”. In Proc. American Control Conference, pp. 4618–4625.

[3] Opila, D., Wang, X., McGee, R., Cook, J., and Grizzle, J., 2009. “Fundamental structural limitations of an industrial energy management controller architecture for hybrid vehicles”. In Proc. ASME Dynamic Systems and Control Conference, pp. 213 –221.

[4] Sciarretta, A., and Guzzella, L., 2007. “Control of hybrid electric vehicles”. *IEEE Control Systems Magazine,* **27**(2), pp. 60–70.

[5] Paganelli, G., Tateno, M., Brahma, A., Rizzoni, G., and Guezennec, Y., 2001. “Control development for a hybrid-electric sport-utility vehicle: strategy, implementation and field test results”. In Proc. American Control Conference, pp. 5064–5069.

[6] Paganelli, G., Ercole, G., Brahma, A., Guezennec, Y., and Rizzoni, G., 2001. “General supervisory control policy for the energy optimization of charge-sustaining hybrid electric vehicles”. *JSAE Review,* **22**(4), Oct., pp. 511–518.

[7] Lin, C.-C., Peng, H., Grizzle, J., Liu, J., and Busdiecker, M., 2003. “Control system development for an advanced-technology medium-duty hybrid electric truck”. In Proceedings of the International Truck & Bus Meeting & Exhibition, Ft. Worth, TX, USA.

[8] Boyali, A., Demirci, M., Acarman, T., Guvenc, L., Tur, O., Ucarol, H., Kiray, B., and Ozatay, E., 2006. “Mod-

eling and control of a four wheel drive parallel hybrid electric vehicle". In Proc. IEEE International Conference on Control Applications, pp. 155–162.

[9] Graham, L., Christenson, M., and Karman, D., 2006. "Light duty hybrid vehicles - influence of driving cycle and operating temperature on fuel economy and ghg emissions". In Proceedings of the IEEE EIC Climate Change Technology, M. Christenson, ed., pp. 1–6.

[10] Kermani, S., Delprat, S., Trigui, R., and Guerra, T. M., 2008. "Predictive energy management of hybrid vehicle". In Proc. IEEE Vehicle Power and Propulsion Conference, pp. 1–6.

[11] Kermani, S., Delprat, S., Guerra, T. M., and Trigui, R., 2009. "Predictive control for HEV energy management: experimental results". In Proc. IEEE Vehicle Power and Propulsion Conference, pp. 364–369.

[12] Tate, E., Grizzle, J., and Peng, H., 2008. "Shortest path stochastic control for hybrid electric vehicles". *International Journal of Robust and Nonlinear Control,* **18**, pp. 1409–1429.

[13] Bertsekas, D., 2005. *Dynamic Programming and Optimal Control*, Vol. 1. Athena Scientific.

[14] Bertsekas, D., 2005. *Dynamic Programming and Optimal Control*, Vol. 2. Athena Scientific.

[15] Johannesson, L., Asbogard, M., and Egardt, B., 2007. "Assessing the potential of predictive control for hybrid vehicle powertrains using stochastic dynamic programming". *IEEE Trans. Intell. Transp. Syst.,* **8**(1), pp. 71–83.

[16] Lin, C.-C., Peng, H., and Grizzle, J., 2004. "A stochastic control strategy for hybrid electric vehicles". In Proc. of the American Control Conference, pp. 4710 – 4715.

[17] Opila, D., 2010. "Incorporating drivability metrics into optimal energy management strategies for hybrid vehicles". PhD thesis, University of Michigan.

[18] Opila, D., Wang, X., McGee, R., Gillespie, R., Cook, J., and Grizzle, J., 2010. "An energy management controller to optimally tradeoff fuel economy and drivability for hybrid vehicles". *Submitted to IEEE Transactions on Control Systems Technology*.

[19] Bertsekas, D. P., and Tsitsiklis, J. N., 1996. *Neuro-Dynamic Programming*. Athena Scientific.

[20] Lin, C.-C., Peng, H., Grizzle, J., and Kang, J.-M., 2003. "Power management strategy for a parallel hybrid electric truck". *IEEE Transactions on Control Systems Technology,* **11**(6), pp. 839–849.

[21] Belton, C., Bennett, P., Burchill, P., Copp, D., Darnton, N., Butts, K., Che, J., Hieb, B., Jennings, M., and Mortimer, T., 2003. "A vehicle model architecture for vehicle system control design". In Proc. SAE World Congress & Exhibition. Paper no. 2003-01-0092.

[22] McGee, R. A., 2003. "Model-based control system design and verification for a hybrid electric vehicle". In SAE Tech. Paper Series, Future Transportation Technology Conf., Costa Mesa, CA. June 2003. SAE Paper 2003-01-2308.

[23] Tate, Edward Dean, J., 2007. "Techniques for hybrid electric vehicle controller synthesis". PhD thesis, University of Michigan.