

Incorporating Drivability Metrics into Optimal Energy Management Strategies for Hybrid Vehicles

by

Daniel F. Opila

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Mechanical Engineering)
in The University of Michigan
2010

Doctoral Committee:

Professor Jessy W. Grizzle, Co-Chair
Associate Professor Brent Gillespie, Co-Chair
Professor Huei Peng
Professor Anna G. Stefanopoulou
Professor Jing Sun

© Daniel Opila 2010
All Rights Reserved

To my wife Lora and my parents, Mary Anne and Paul, for their endless support.

ACKNOWLEDGEMENTS

I would like to thank a number of individuals without whom the work of this thesis could not have been completed. My co-advisors, Jessy Grizzle and Brent Gillespie, have been extremely helpful. Jeff Cook has been a great inspiration and was invaluable in my interactions with industry. My collaborators at Ford enabled a unique collaboration between the industrial and academic realms: Ryan McGee, Xiaoyong Wang, Deepak Aswani, and Ameet Deshpande. A large portion of the novel results in this work are based on large-scale computation, roughly 3 orders of magnitude more than previous work. This would not have been possible without the dedicated staff at Michigan's Center for Advanced Computing: Brock Palen, Matthew Britt, and Andy Caird.

Daniel Opila
Ann Arbor, March 2010

TABLE OF CONTENTS

DEDICATION	ii
ACKNOWLEDGEMENTS	iii
LIST OF FIGURES	vii
LIST OF TABLES	xi
LIST OF APPENDICES	xii
ABSTRACT	xiii
CHAPTER	
I. Introduction	1
1.1 Background	1
1.2 The topics addressed in this dissertation	2
1.3 Summary of Contributions	6
1.4 Literature review	8
1.4.1 Rule-Based Methods	10
1.4.2 Equivalent Consumption Minimization Strategy	10
1.4.3 Dynamic Programming Solutions	11
1.4.4 Other Methods	12
II. Dynamic Programming Theoretical Developments	14
2.1 Dynamic Programming	14
2.2 Computation Reduction	15
2.2.1 Theory of Minimization Decomposition	16
2.2.2 Related Comments on Minimization Decomposition	17
2.2.3 Examples	18
2.3 Controlling Additional Behavior using SPSDP	21
2.3.1 Motivation	21
2.3.2 Formulation Methods	21
2.3.3 Penalty Implementation	24
2.3.4 Computation	25

III. Problem Formulation and Approach for Energy Management of a Prototype Hybrid Vehicle	27
3.1 Vehicle	27
3.1.1 Vehicle Architecture	27
3.1.2 Vehicle Models	28
3.1.3 Control-Oriented Model	29
3.1.4 Vehicle Simulation Model	32
3.2 Drivability Constraints	33
3.2.1 Motivation	33
3.2.2 Simplified Drivability Metrics	34
3.3 Shortest Path Stochastic Dynamic Programming	34
3.3.1 Cost Function	34
3.3.2 Problem Formulation	36
3.3.3 Terminal State	37
3.3.4 Implementable Constraints	39
3.4 Comparison of SPSDP to the Equivalent Consumption Minimization Strategy (ECMS)	39
IV. Simulations on Government Test Cycles	41
4.1 Introduction	41
4.2 Simulation Procedure	41
4.3 Results: Performance Trends	43
4.3.1 Fuel Economy Results	43
4.3.2 Discussion	45
4.4 Detailed Performance	45
4.4.1 Results	45
4.4.2 Discussion	47
V. Real World Driving	51
5.1 Introduction	51
5.2 Robustness to Real-World Driving	52
5.2.1 Motivation	52
5.2.2 Drive Cycle Data	52
5.2.3 Simulation Procedure	53
5.2.4 Results	54
5.2.5 Discussion	56
5.3 Robustness to Variable Driving Styles and Sensitivity to Design Cycle Statistics	58
5.3.1 Results	59
5.3.2 Discussion	60
VI. Fundamental Limitations of an Industrial Controller	61
6.1 Introduction	61

6.2	Baseline Industrial Controller	63
6.2.1	Architecture	63
6.2.2	Performance Capability	64
6.2.3	Parameter Sweep Procedure	65
6.3	Academic Benchmark	65
6.4	Simulation Procedure	66
6.5	Results	67
6.5.1	FTP Cycle	67
6.5.2	Real-World Drive Cycles	68
6.5.3	Discussion	70
6.6	Conclusions	71
VII. Development and Validation of Drivability Metrics		72
7.1	Formulation	72
7.2	Simplified Drivability Metrics	74
7.3	Drivability on Government Test Cycles	74
7.4	Drivability on Ensemble Cycles	76
7.5	Discussion	78
VIII. Hardware Testing		80
8.1	Introduction	80
8.2	Controller Design	82
8.3	Implementation	83
8.3.1	Overall Process	83
8.3.2	Vehicle Models	84
8.3.3	Structural Setup	85
8.3.4	Multi-Rate Updates	87
8.4	Refining the controller in hardware	93
8.4.1	Hardware Debugging	93
8.4.2	Additional Penalty Implementation	95
8.5	Hardware testing results	99
IX. Conclusions		106
9.1	Overall Summary	106
9.2	General Comments	108
9.3	Future Directions	109
APPENDICES		110
BIBLIOGRAPHY		128

LIST OF FIGURES

Figure

1.1	The Prototype Hybrid: A Modified Volvo S-80.	1
2.1	Schematic diagram of a conceptual “Super Electric Machine” that optimizes the mix between the two electric machines. This allows one degree of freedom of the control optimization to be carried out off-line while maintaining the optimality of the solution.	20
3.1	The Prototype Hybrid: A Modified Volvo S-80.	27
3.2	Vehicle Configuration	28
3.3	Vehicle SOC Dynamics model. The three inputs are Engine Torque, Electric Machine 1 Torque, and Transmission Gear. The vehicle velocity and required torque are provided by the stochastic driver model. In Series Mode, the Electric Machine 1 command is speed rather than torque.	32
3.4	Value Function $V(x)$ for several velocities and fixed deceleration at $-0.57 m/s^2$. The quadratic penalty on SOC strongly affects the value function at low speeds when the driver is more likely to turn the key off and end the trip.	38
4.1	FTP and NEDC cycles.	42
4.2	Performance of SPSDP controllers on FTP and NEDC. A separate <i>family</i> of controllers is designed for FTP and NEDC using each cycle’s statistics. The family is designed by sweeping the parameters α and β in the cost function (3.7). Figures 4.2a and 4.2b show the data as a 3-D scatterplot of fuel economy vs. drivability events. SPSDP controllers are shown as red dots. The baseline controller is shown as a large green circle. Figures 4.2c and 4.2d show the view along the <i>Gear Events</i> axes of Figures 4.2a and 4.2b respectively. The raw data points, isoclines, and baseline controller are still visible. Figures 4.2e and 4.2f show the final SOC for these controllers. All controllers start with SOC=0.5.	44
4.3	Time traces of selected simulation parameters. The baseline controller is shown as a solid (blue) line, and one particular SPSDP controller that yields the best overall fuel economy is shown as a dashed (red) line; this is SPSDP #1 in Table 4.1. The transmission gear is shown only when the engine is on and the clutch is engaged.	47

4.4	Engine Torque-Speed operating points on the FTP cycle. The solid black line represents an operational restriction, which both the SPSDP and baseline controllers respect. Baseline controller operating points are shown as circles (blue) and SPSDP controller operating points are shown as plus signs (red). The SPSDP controller is SPSDP #1 in Table 4.1 and also shown in Figure 4.3. The isoclines show constant brake specific fuel consumption in g/kWh, an inverse measure of efficiency.	49
5.1	Statistics of Real-World Driving. Figure 5.1a is the Cumulative Distribution Function of trip distance for the source data and two subsets. Mean distances for the sets are: Full Data Set - 11.7 mi., <i>Ensemble 1</i> - 12.7 mi., <i>Ensemble 2</i> - 9.9 mi. Figure 5.1b is the Cumulative Distribution Function of vehicle velocity for the source data, two subsets, and two government test cycles.	53
5.2	Fuel Economy and Drivability Metrics on Ensemble 1 when simulated as Concatenated and Individual Cycles for the baseline controller and SPSDP controller <i>families</i> designed with statistics from FTP, NEDC, <i>Ensemble 1</i> , and <i>Ensemble 2</i> . Fuel Economy, Gear Events, and Engine Events are cumulative for the whole cycle set, representing approximately 1000 miles. Results are normalized to the baseline controller on FTP, as in Section IV.	55
5.3	Statistical fuel economy. The SPSDP controller <i>family</i> designed on <i>Ensemble 1</i> is simulated on <i>Ensemble 1</i> and <i>Ensemble 2</i> , and each cycle is corrected for SOC. The mean, standard deviation, and 10 th and 90 th percentile are calculated. The mean, standard deviation (error bars), and 10 th and 90 th percentile are also calculated for the baseline controller.	57
5.4	Fuel Economy and Drivability Metrics on the FTP and NEDC Cycle for 5 controller options. Controller <i>families</i> are designed with statistics from FTP, NEDC, <i>Ensemble 1</i> , and <i>Ensemble 2</i> . All fuel economy figures are normalized to the baseline controller performance on FTP, shown as a large green dot in Figure 5.4a. The controllers are the same as those shown in Figure 5.2.	59
6.1	High Level Baseline Controller Architecture.	63
6.2	Two possible design processes. The SPSDP process conducts the optimization in one step, but may be more complex. The “Common” two stage optimization is often used in industry. Its simpler structure may be easier to tune, but may sacrifice performance.	64
6.3	Best Case performance of the baseline controller running FTP compared to SPSDP controllers. The gray dots are all possible baseline controllers, the black triangles are the “best” available baseline controllers, and the blue squares are selected randomly from the other reasonable controllers. SPSDP controllers are shown for comparison as red diamonds. Fuel economy is normalized to the default baseline controller running the FTP cycle, shown as a large green circle. All markers represent the same controllers in Figures 6.3-6.5.	66
6.4	Final Battery SOC of the baseline and SPSDP controllers running FTP. All cycles start at SOC=0.5. All markers represent the same controllers in Figures 6.3-6.5.	68

6.5	Best Case performance of the baseline controller and SPSDP running the ensemble of 100 cycles. Fuel economy is normalized to the default baseline controller running the FTP cycle. All markers represent the same controllers in Figures 6.3-6.5.	69
7.1	Drivability Metric Reduction. The seven complex engine and transmission metrics are divided into two categories, mean dwell times and short-duration dwell times. These metrics are then reduced to the two simplified metrics.	73
7.2	Comparison of simple and complex drivability metrics on the FTP cycle. Complex engine activity metrics are compared to the simplified <i>Engine Events</i> metric in subfigures 7.2a-7.2c. Three gear activity metrics are compared to the simplified <i>Gear Events</i> metric in subfigures 7.2d-7.2f. Subfigures 7.2b and 7.2e have straight-line fits to the data, and the parameters are shown in Tables 7.2g and 7.2h.	75
7.3	Comparison of simple and detailed drivability metrics for concatenated and individual Ensemble 1 cycles. Detailed engine activity metrics are compared to the simplified <i>Engine Events</i> metric in Figures 7.3a-7.3c for the concatenated Ensemble, where the 100 cycles are treated as a single trip. Each marker represents the same drive cycle, the concatenated Ensemble 1. The same metrics are studied for the individual Ensemble 1 in Figures 7.3d-7.3f, where each cycle is simulated individually. Each marker represents one controller running one of 100 cycles, so the markers no longer represent the same cycle.	77
8.1	The increasing complexity of controller testing in this work.	81
8.2	The overall development process	82
8.3	Multi-rate actuator commands	86
8.4	Real-time algorithm implementation in Simulink	87
8.5	Illustration of the command update scheme.	91
8.6	Illustration of the command update scheme.	92
8.7	The first driving attempt. The SPSDP controller worked on the first try. .	94
8.8	Engine Torque and Pedal Commands during an unexpected oscillation. . .	96
8.9	The value function during an unexpected pedal oscillation. The recorded real data is shown on the left. The proposed solution to this oscillation is an additional “bowl” penalty on engine torque. It is quite difficult to exactly replicate hardware test conditions, so the column on the right represents what the commands would have been with the improved controller measuring the same vehicle state data.	98
8.10	Additional penalty added to the value function based on the change in Engine Torque. This is termed a “bowl” penalty due to its shape.	98
8.11	Driving Traces on FTP.	100
8.12	Driving Traces on NEDC.	101
8.13	Engine torque-speed operating points during hardware test on FTP. The top plots show commanded operation while the bottom show the actual torque reflecting an engine controller torque limit.	102
8.14	Representative vehicle behavior on a typical NEDC “hill” demonstrating electric mode, engine start, parallel mode, engine stop, and a return to electric mode.	104
8.15	Detailed view of the engine start dynamics shown in Figure 8.14.	105

A.1	Selection of stochastic variables. Using a plaid grid requires a “square” of grid points. Using velocity and acceleration as driver states tends to underutilize the high speed/high acceleration points (Figure A.1a). Using speed and power has the same problem, but at low speeds. A transformation is used (Figure A.1b) that basically uses normalized acceleration at low speeds at normalized power at speeds above roughly 10 mph.	116
B.1	Comparison of “Local” and “SPDP-Drivability” Controllers on FTP.	125
B.2	Comparison of “Local” and “SPDP-Drivability” Controllers on FTP	126

LIST OF TABLES

Table

2.1	Comparison of Event Implementations	24
2.2	Additional Event implementations	25
2.3	Comparison of Computation Requirements	26
3.1	Vehicle Parameters	28
3.2	Vehicle Mode Definitions.	31
3.3	Vehicle Model States	37
4.1	Selected SPSDP controller performance	46
4.2	Selected SPSDP controller performance	48
8.1	Fuel Economy Summary for malfunctioning hardware	81
B.1	Vehicle Parameters	118
B.2	Vehicle Dynamic Model	120

LIST OF APPENDICES

Appendix

A.	Computation Techniques	111
B.	Implementing Penalties using the Instantaneous vs. Extended Model Method	117

ABSTRACT

Incorporating Drivability Metrics into Optimal Energy Management Strategies for Hybrid Vehicles

by

Daniel F. Opila

Co-Chairs: Jessy W. Grizzle and Brent Gillespie

Hybrid Vehicle fuel economy performance is highly sensitive to the “Energy Management” strategy used to regulate power flow among the various energy sources and sinks. Optimal solutions are easy to specify if the drive cycle is known a priori. It is very challenging to compute controllers that yield good fuel economy for a class of drive cycles representative of typical driver behavior. Additional challenges come in the form of constraints on powertrain activity, like shifting and starting the engine, which are commonly called “drivability” metrics. These constraints can adversely affect fuel economy. In this dissertation, drivability restrictions are included in a Shortest Path Stochastic Dynamic Programming (SPSDP) formulation of the energy management problem to directly address this tradeoff and generate optimal, causal controllers. The controllers are evaluated on Ford Motor Company’s highly accurate proprietary vehicle model and compared to a controller developed by Ford for a prototype vehicle. The SPSDP-based controllers improve fuel economy more than 15% compared to the industrial controller on government test cycles. In addition, the SPSDP-based controllers can directly quantify tradeoffs between fuel economy and drivability. Hundreds of thousands of simulations are conducted using real-world drive cycles to evaluate performance and robustness in the real world, demonstrating 10% improvement compared to the baseline. Finally, the controllers are tested in a real vehicle.

CHAPTER I

Introduction

1.1 Background

Hybrid vehicles have become increasingly popular in the automotive marketplace in the past decade. The most common type is the electric hybrid, which consists of an internal combustion engine (ICE), a battery, and at least one electric machine (EM). Hybrids are built in several configurations including series, parallel, and the series-parallel configuration considered here. Hybrid vehicles are characterized by multiple energy sources; the strategy to control the energy flow among these multiple sources is termed “Energy Management” and is crucial for good fuel economy. An excellent overview of this area is available in [90].

Designing an effective Energy Management controller for hybrid vehicles is a challenging, rich problem and has been studied extensively. Varying control design methods are used, including rule-based [4, 5, 26, 98], neural networks [85], game theory [21], and fuzzy logic



Figure 1.1: The Prototype Hybrid: A Modified Volvo S-80.

[48]. There are many proposed methods available for both the non-causal (cycle known in advance) and causal (cycle unknown in advance) cases [42, 80, 81], and those with partial future information [32, 40].

This research is a collaborative effort between the University of Michigan and Ford Motor Company. This work uses Ford’s high-fidelity vehicle simulation model [6], which is used to develop HEV control algorithms and evaluate fuel economy for production vehicles. The vehicle studied here is a modified Volvo S-80 prototype (Figure 1.1) that does not match any vehicle currently on the market. As a benchmark, Ford provided a controller developed for this prototype vehicle. This industrial controller was introduced in [72] and is termed the “baseline” controller.

This dissertation is based primarily on a series of publications by the author on the topic of energy management [70–74]. Preliminary hardware testing results presented here are new and will be the subject of a future publication.

1.2 The topics addressed in this dissertation

The most commonly used optimization strategies are the Equivalent Consumption Minimization Strategy (ECMS) [18, 20, 67, 68, 75, 89] and Stochastic Dynamic Programming (SDP) or Deterministic Dynamic Programming methods [59, 60, 94, 100]. The majority of existing work focuses on optimal controllers that seek to minimize fuel consumption. Such controllers, which ignore other important drivetrain behavior, can be undesirable in practice because they will use high levels of powertrain activity like shifting and starting the engine in order to save miniscule amounts of fuel [17, 19, 43, 56, 60, 76]. These powertrain behaviors are known as “drivability.” Previous researchers have recognized this problem and incorporated penalties on engine starts in an ECMS formulation [89] and gear shifting in a dynamic programming-based rule extraction method [56, 60].

The method used here is Shortest Path Stochastic Dynamic Programming (SPSDP), which generates optimal, causal controllers that can control complex vehicle behavior and respect constraints while minimizing fuel consumption. In addition to generating a class of optimal controllers, the SPSDP method allows direct study of the tradeoffs between different

performance goals, specifically drivability and fuel economy. The ability to easily generate Pareto tradeoff curves is perhaps just as interesting as a specific fuel economy benefit. The designer can generate both the maximum attainable performance curve and causal controllers that generate that performance. Drivability is studied here, but one could also study the fuel economy tradeoff with other attributes like emissions, battery wear, or engine noise characteristics. In this dissertation, drivability restrictions are included in a Shortest Path Stochastic Dynamic Programming (SPSDP) formulation of the energy management problem to directly address this tradeoff and generate optimal, causal controllers.

When applying SPSPD methods, computational tractability is a key issue. The SPSPD controller calculations done here were conducted using a two-step optimization strategy that preserves optimality while decreasing computation by at least a factor of 10, thereby allowing the controller design to be conducted on a desktop PC. The two-step optimization strategy is a general result for Dynamic Programming (either Stochastic or Deterministic). The strategy is applicable to other vehicle configurations that have multiple actuator degrees of freedom.

The main decision in the design of an SPSPD controller is the selection of a cost function which reflects desired vehicle behavior while maintaining feasible off-line computation. This dissertation focuses on two specific drivability attributes: the engine and transmission behavior. If the SPSPD method is to be used in practice, engineers will likely seek to incorporate additional criteria for vehicle behavior. Many other performance attributes can be studied using this theoretical framework, but care must be taken to avoid excessive computational burden. Several options for incorporating vehicle behaviors in the problem formulation are discussed, again in a general framework applicable to various Dynamic Programming problems. These general theoretical techniques for Dynamic Programming are in Chapter II.

Although the methods are different, SPSPD is somewhat similar to the more straightforward (and popular) ECMS method. Despite vast differences in problem setup, both methods address the fundamental question in the energy management problem: the future value of battery charge in terms of fuel consumption. SPSPD has advantages in its ability to optimize for other behaviors along with fuel economy. ECMS handles additional tradeoffs

suboptimally. The theoretical formulation, algorithm, and comparison to ECMS are found in Chapter III.

Most of the existing literature focuses on theoretical methods with a few test cases, typically simulations of a representative vehicle on the certification test cycles. There are relatively few results showing how these algorithms perform in practice [13, 27, 35, 42, 43, 61, 77, 78] and how they compare to the existing industrial state of the art. It is unclear how much of the existing literature is used by industry in actual production vehicles.

There are numerous practical considerations required to actually use SPSDP in industrial development or production. Important instances of these requirements are addressed via comparison to a baseline industrial controller on government (Chapter IV) and real-world (Chapter V) test cycles to evaluate performance, robustness, sensitivity, fuel economy vs. drivability tradeoffs, and real-world vs. test cycle performance. About 500,000 simulations are conducted using large numbers of real-world drive cycles. With these data, controller performance can be evaluated and optimized with respect to various classes of driver behavior [22, 23, 36, 41, 50] in addition to government certification cycles [71–73]. The simulations are analyzed to determine not just average performance, but standard deviations and worst-case performance. The real-world robustness testing addresses a common customer complaint that the fuel economy shown on the “window sticker” does not match the vehicle performance obtained in practice [24, 37, 69, 86, 93]. To provide a realistic benchmark, SPSDP energy management controllers are compared to an industrially-designed baseline controller. These large-scale simulation studies and robustness tests are discussed in Chapter V.

It is often difficult to compare energy management controllers. Most algorithms, even those from academia based on formal optimization methods, have at least some parameters that must be selected by the designer. This is even more true of industrial controllers, which tend to use extensive hand-tuning and in-vehicle calibration in order to trade off what are often very subjective driving quality attributes.

The SPSDP controllers developed here will be shown to demonstrate excellent performance compared to the baseline controller. Because the baseline controller requires manual tuning, one is left to wonder if the performance of the baseline controller is limited by its

architecture or by the particular tuning used. To this end, the baseline industrial controller is first evaluated to determine its Pareto tradeoff curve: the upper limit of possible performance in terms of fuel economy versus engine activity. A comparison is then made to the Pareto tradeoff curves of the SPSDP controllers.

The Pareto frontier for the SPSDP controllers is shown to lie above the Pareto frontier of the baseline controller, meaning that the SPSDP controllers achieve superior fuel economy performance for a given level of engine on-off activity, for *any* possible tuning of the baseline controller. This limitation is fundamental to the structure of the baseline algorithm: no amount of parameter tuning or calibration can generate performance that equals that of the SPSDP algorithm. These fundamental limitations of the baseline industrial controller are studied in Chapter VI.

A major enabling result for the work in this dissertation was the development of the drivability metrics for powertrain activity. The SPSDP algorithm is very sensitive to the number of states required to track the metrics, so simple metrics must be used to keep the problem tractable. Simplified drivability metrics are shown to be well correlated with more detailed metrics on both government test and real-world driving. This validates the simplified metrics as a useful approximation for controller design. The development and validation of these metrics is discussed in Chapter VII.

There are relatively few results that test energy management controllers in real hardware [13, 35, 42, 43, 61, 77, 78], or that address the many practical considerations required to do so. One major goal was to sufficiently develop the algorithms so that they could be tested in a real vehicle. This is a surprisingly difficult process and required significant algorithm refinement and several conceptual innovations that are discussed in Chapter VIII.

Partway through testing, the engine controller detected a fault and limited engine torque to 150 Nm; 300 Nm is full scale. This issue was not repaired, so most of the hardware fuel economy data reported here reflects this limitation. Although the fuel economy results are unreliable, the main conclusion to draw from these results is not that one controller is better or worse than the other, but that these SPSDP controllers do work in actual hardware and generate performance comparable to a baseline controller. The hardware testing process and results are described in Chapter VIII.

The main drawback of SPSDP and Dynamic Programming in general is significant off-line computation. A important portion of the effort in this dissertation is the development of efficient, reliable software code to compute numerical solutions. This software is really what enabled all of these results; the SDP problem with drivability only becomes tractable on reasonable computing resources with extensive code development. Most publications only consider a few controllers on a few cycles, typically only looking at fuel economy. All told, this work developed about 15,000 controllers and simulated roughly 3 million cycles. Many researchers using these techniques are often initially handicapped by inefficient code rather than a lack of theoretical understanding. Simple tips, tricks, and best practices can drastically improve results but are not usually included in publications. The combination of theoretical considerations (Chap. II) with refined software enabled the massive scale in this dissertation. The lessons learned and best practices are included in Appendix A.

1.3 Summary of Contributions

In essence, this thesis can be viewed as a fairly comprehensive study of the energy management problem when incorporating constraints. Some of the details are necessarily specific to this vehicle, but most of the simulation results show trends that are applicable to hybrid vehicles in general and can be extended to plug-in hybrids. Several theoretical results were developed that are applicable to Dynamic Programming problems in general and are illustrated with vehicle examples. These controllers were tested in an actual vehicle, and techniques developed for the real-time controller are not vehicle specific and can even be used for other non-vehicle applications.

This dissertation differs from most projects in that the proposed techniques are fully developed through a process of theoretical development, extensive real-world simulation, and hardware testing. It is relatively easy to propose a new technique, but subjecting that technique to rigorous evaluation and hardware testing requires great effort and often reveals weaknesses. Many academic results are viewed skeptically by industry because they are not rigorously evaluated or demonstrated.

The main contributions of this dissertation may be roughly organized into three broad

categories. Each component was briefly described in this introduction and more fully in appropriate chapters.

1. **Incorporating Drivability into Energy Management** (Chap. III)

- (a) Developed and validated simplified drivability metrics (Chaps. III and VII)
- (b) Theoretical techniques to drastically reduce computation (Chap. II)
- (c) Pareto Tradeoff curves (Chaps. IV and V)
- (d) Efficient code to generate controllers (Chap. II and Appendix A)
- (e) Highly automated grid computing (Chaps. IV and V)

2. **Realistic Evaluation of Controllers**

- (a) Controllers evaluated using Ford's detailed simulation model (Chaps. III, IV, and V)
- (b) Controllers compared to an industrial baseline (Chaps. IV, V, and VI)
- (c) Millions of simulations of real-world driving (Chap. V)
 - i. Robustness to driver/trip variations
 - ii. Variability with different drivers
 - iii. Sensitivity to the statistics used to design the controller
- (d) Limitations of industrial controller evaluated (Chap. VI)

3. **Hardware Testing** (Chap. VIII)

- (a) Real-time implementation in vehicle
- (b) Handle timing issues to have good pedal response
- (c) Deal with unreliable hardware and unexpected system response
- (d) Deliver a pleasant driving experience

Many of these results are interconnected, making it somewhat difficult to organize this document. Chapters are organized roughly sequentially, such that each chapter may be understood based on the previous chapters. Appendix A is deeply tied to chapters III and

II and enables the results in the remaining chapters. Chapter VI is a rigorous evaluation of the previous results and is easier to understand with the context of the previous results. The chapter sequence is shown below for easy comparison.

- Chapter II - Dynamic Programming Theoretical Developments
- Chapter III - Problem Formulation and Approach
- Chapter IV - Simulations on Government Test Cycles
- Chapter V - Real World Driving
- Chapter VI - Fundamental Limitations of an Industrial Controller
- Chapter VII - Development and Validation of Drivability Metrics
- Chapter VIII - Hardware Testing
- Appendix A - Computation Techniques
- Appendix B - Implementing Penalties using the Instantaneous vs. Extended Model Method

1.4 Literature review

Hybrid vehicle energy management controllers have been carefully studied for more than a decade. The central problem is to minimize a cost (typically fuel) over a drive cycle subject to charge-sustaining operation where the final SOC matches the starting SOC. The control solutions in the literature can be roughly grouped into four categories: rule-based control, Equivalent Consumption Minimization Strategy (ECMS) and related methods, Dynamic Programming methods, and a broad category of “other” approaches.

As the field has expanded, numerous papers are available that attempt to minimize fuel consumption with varying degrees of sophistication. Later attempts introduce more complex goals including emissions reduction [58, 68]. The field is still evolving. While several methods have become *de facto* standards, there is no single control solution that

has emerged as the best in all cases. Much of the work is problem and vehicle specific, making comparisons difficult and results less portable.

In addition, this field is expanding from the standard problem with the addition of next-generation concepts like plug-in vehicles and the use of future information [28]. Other concepts include large capacitors as an additional energy storage element [14]. Many of the same basic techniques are used, but are being extended into this new application domain. Plug-in hybrids are no longer charge sustaining, which adds an additional twist to the basic problem. The idea of future information opens up many questions about the value of this information, how it can best be used, and how much of it can realistically be incorporated in real-time.

Overall, the controllers used in production vehicles seem quite distinct from the majority of those in the literature. There are several possible reasons for this. One is that companies are hesitant to disclose proprietary controller data in a public forum, which prevents information flow from industry to academia and makes comparisons difficult. The second reason is the simplicity of most academic simulations compared with the massive complexity of actual production controllers. Companies would always like to use the best methods to obtain fuel economy, but a large part of real-time embedded control software design has very little to do with energy management function. Many different subsystems, operating modes, and conflicting goals must all interact. Automakers have a difficult time juggling those requirements and making the system work reliably. Redesigning the basic energy management strategy is a major undertaking that is not often considered, even on a timescale of years. Even if a company would like to redesign its energy management, academic publications cannot be used as an “off the shelf” solution because they do not provide adequate evidence that the proposed methods are sufficiently reliable and can be engineered to handle all the conflicting requirements.

All energy management algorithms that seek to minimize fuel basically boil down to one question: “How valuable is battery charge in terms of fuel?” If such a number is available, the fuel-minimizing solution is exactly determined. All the algorithms discussed here are attempting to estimate this number in some form or another.

There are many proposed methods available for both the non-causal (cycle known in

advance) and causal (cycle unknown in advance) cases, and those with partial future information [32,40]. The energy management problem is usually addressed at slower timescales and transients are ignored, although some results suggest this may sacrifice some performance [82].

A closely related problem is the investigation of different component or vehicle configurations, including plug-ins [10,29]. These papers may not include sophisticated control algorithms. Other related research attempts to collect real data [47] or estimate current driving conditions [34,49,52–55,97,99].

1.4.1 Rule-Based Methods

The first controllers developed for hybrid vehicles used a heuristic or rule-based approach [26,98]. Engineers would directly specify feedback control laws that determined vehicle behavior. The underlying idea was to use engineering judgement to extract performance from the vehicle [13]. Three main ideas were typically employed: use regenerative braking as much as possible, use the battery for “load leveling” to operate the engine at relatively constant powers [4,5], and operate the engine near its maximum efficiency as much as possible [12,38,66]. This approach was reasonably successful at first. It generated functional controllers that were easy to understand. Controllers generated excellent performance, primarily because the benchmark for comparison was usually a non-hybrid vehicle.

The drawbacks of this method eventually became clear: it requires extensive hand-tuning, and even with long development times there are no guarantees that a given solution is optimal or any way to check the optimality of a given solution. These drawbacks motivated the search for more advanced methods.

1.4.2 Equivalent Consumption Minimization Strategy

One of the most commonly used optimization strategies is the Equivalent Consumption Minimization Strategy or ECMS [18,20,67,68,75,89]. This idea initially became popular because of its intuitive nature. The main idea is to choose a control that minimizes a weighted sum of fuel consumption and battery usage. The “equivalence factor” determines the relative weighting of the two and can be conceptualized as the value of battery charge

in terms of fuel. Later results explicitly proved the optimality of the ECMS solution [91], subject to certain assumptions [16]. The base ECMS method assumes the battery efficiency is independent of charge and that the battery state of charge never hits a limit. More complex analysis can relax these assumptions [91]. The proof is based on the Pontryagin minimum principle.

ECMS was the first optimization-based method to be used after the initial rule-based controllers, although initially it was not usually treated explicitly as an optimization method. The simplest variant of ECMS uses a constant equivalence factor and guarantees optimal performance over a given cycle, again subject to assumptions. The equivalence factor is cycle-dependent and therefore the cycle must be exactly known in advance. The simplest procedure to accomplish this is simply to guess the equivalence factor, do a forward simulation, and repeat with another guess using a branch and bound method until the final SOC matches the starting SOC. On one hand, this technique is much faster than the dynamic programming approaches, which is part of its attractiveness. On the other hand, the actual value of the equivalence factor must be determined exactly; deviations less than 0.1% can change the final SOC by 20% on a 20 minute cycle.

ECMS methods can generally be classified as causal or non-causal. A drive cycle is generally not known in advance and real vehicles must have causal controllers. For this reason, various schemes have been proposed to estimate the equivalence factor in a causal way while maintaining SOC within some reasonable bounds; one typical example is Adaptive ECMS (A-ECMS) [84]. ECMS performs quite well with the on-line adaptation of the equivalence factor, but the optimality guarantees are lost.

ECMS controllers have also been proposed for plug-in hybrids using future information to guarantee the battery drains appropriately. The ECMS framework includes the option to include other vehicle attributes like powertrain behavior [3].

1.4.3 Dynamic Programming Solutions

Dynamic Programming is a general class of algorithms that consider all possible systems states and appropriate control actions. The details of the various algorithms are outside the scope of this work, but these algorithms may be divided into causal and non-causal variants,

much like ECMS to some extent. The evolution of Dynamic Programming algorithms is somewhat similar to the ECMS story. Initial attempts used deterministic dynamic programming (DDP) to determine the optimal control for a known cycle, and many papers address this topic [79–81]. Using those results, state-feedback laws (or rules) were extracted to imitate the behavior of the DDP algorithm, but in a causal controller [57, 58, 60, 62, 100, 101]. This process was quite effective, but extracting the feedback laws from the DDP solution is time-consuming and there is no explicit guarantee that such laws may be found or that they retain optimality.

The next step was the use of stochastic dynamic programming (SDP) [8, 9], which uses a statistical representation of driver behavior rather than exact future knowledge [46, 59, 94, 95]. The method used in this dissertation, Shortest Path Stochastic Dynamic Programming (SPSDP) is a specific variant of the more general SDP. SDP directly generates causal controllers without any rule extraction. The SDP controllers are optimal with respect to the driving statistics and minimize the expected cycle cost. SDP controllers are not optimized for a particular cycle, but typically perform well on standard test cycles.

One advantage of dynamic programming (deterministic or stochastic) is the ability to optimally handle additional model states. These model states can then be used to optimally implement desired vehicle behavior, as they are in this dissertation. DDP is guaranteed to give an optimal (non-causal) solution and is often used as a benchmark for evaluating causal methods [63, 64, 84].

DDP is an obvious choice to incorporate future information, but can be difficult to compute in real-time and approximations are often used [10, 30–33]. A few results incorporate route information using SDP [39, 40].

Both DDP and SDP algorithms have been proposed for plug-ins in various contexts. Numerous methods have been applied to this class of problems [8, 9, 15]

1.4.4 Other Methods

A variety of other techniques have been proposed for the energy management problem, although none has received as much attention as the other methods. These include controllers based on neural networks [85], game theory [21], and fuzzy logic [25, 48, 65, 88, 92, 102],

and numerical optimization [44, 87, 103]. Perhaps the most realistic of these “other” methods is model predictive control (MPC) [11, 31, 42, 43]. Essentially, MPC must deal with the same issues as the other methods, namely estimating the value of battery charge in terms of fuel. For MPC, this shows up as a requirement to estimate the future cycle during the prediction horizon as well as a final cost at the end of the horizon. Essentially, MPC and DDP are very similar, if one were to only compute the DDP solution for a finite horizon.

CHAPTER II

Dynamic Programming Theoretical Developments

2.1 Dynamic Programming

There are several variants of the dynamic programming algorithm, including both deterministic and stochastic versions. The general optimization goal is the minimization of a performance metric J which is an additive cost function

$$J = \sum_k c_k(x, u), \quad (2.1)$$

where $c_k(x, u)$ is the cost incurred at time step k for a state x and control u . The particular formulation used to determine the optimal control strategy for this vehicle is the Shortest Path Stochastic Dynamic Programming (SPSDP) algorithm [7–9, 59, 94], which attempts to minimize the expected cost over an infinite horizon

$$J = E \left[\sum_0^{\infty} c_k(x, u) \right]. \quad (2.2)$$

This method directly generates a causal controller; characteristics of future driving behavior are specified via a finite-state Markov chain rather than exact future knowledge. The system model is formulated as

$$x_{k+1} = f(x_k, u_k, w_k), \quad (2.3)$$

where u_k is a particular control choice in the set of allowable controls U , x_k is the state,

and w_k is a random variable arising from the unknown drive cycle. Given this formulation, the optimal cost $V^*(x)$ over an infinite horizon is a function of the state x and satisfies

$$V^*(x) = \min_{u \in U} E_w[c(x, u) + V^*(f(x, u, w))], \quad (2.4)$$

where $c(x, u)$ is the instantaneous cost as a function of state and control. This equation represents a compromise between minimizing the current cost $c(x, u)$ and the expected future cost $V(f(x, u, w))$. The control u is selected based on the expectation over the random variable w , rather than a deterministic cost, because the future can only be estimated based on the probability distribution of w . Note that the cost $V(x)$ is a function of the state only. This cost is finite for all x if every point in the state space has a positive probability of eventually transitioning to an absorbing state that incurs zero cost from that time onward.

An optimal control u^* is any control that achieves the minimum cost $V^*(x)$

$$u^*(x) = \operatorname{argmin}_{u \in U} E_w[c(x, u) + V^*(f(x, u, w))]. \quad (2.5)$$

Note that the disturbance w in (2.4) and (2.5) may be conditioned on the state and control input,

$$P(w|x, u). \quad (2.6)$$

These results are well established and quite general. The challenge lies in applying the algorithm to a practical problem. The methods used in this dissertation are discussed in Chapter III. In addition, the actual code used to discretize the state space and numerically solve these problems is very important and is discussed in Appendix A.

2.2 Computation Reduction

The largest drawback of Dynamic Programming is its computational cost. Simple problems can quickly become intractable for even the fastest computers. Problems must be formulated very carefully to keep the problem feasible. This section discusses a theoretical result that can significantly reduce computation by exploiting structure in the problem.

2.2.1 Theory of Minimization Decomposition

Proposition: (Minimization Decomposition)

Consider a Bellman equation of the form

$$V^*(x) = \min_{\hat{u} \in \hat{U}(x), \bar{u} \in \bar{U}(x, \hat{u})} E_w[c(x, \hat{u}, \bar{u}) + V^*(f(x, \hat{u}, w))], \quad (2.7)$$

and define

$$\hat{c}(x, \hat{u}) = \min_{\bar{u} \in \bar{U}(x, \hat{u})} c(x, \hat{u}, \bar{u}). \quad (2.8)$$

Then $V^*(x)$ satisfies (2.7) if and only if it satisfies

$$V^*(x) = \min_{\hat{u} \in \hat{U}(x)} E_w[\hat{c}(x, \hat{u}) + V^*(f(x, \hat{u}, w))]. \blacksquare \quad (2.9)$$

Remark: This result allows a significant reduction in computation complexity for problems that have the specific structure (2.7). The reduced Bellman equation (2.9) may be solved using only the reduced control space $\hat{U}(x)$. Minimization Decomposition may also be used when solving for non-stationary value or deterministic functions by appropriately replacing $V(x)$ with a time-dependent $V_k(x)$, for either deterministic or stochastic cases [40]. This means it may be applied for general dynamic programming formulations, including the deterministic, stochastic finite-horizon, infinite horizon, shortest path variants.

Proof:

Equation (2.7) may be written as

$$V^*(x) = \min_{\hat{u} \in \hat{U}(x)} \min_{\bar{u} \in \bar{U}(x, \hat{u})} E_w[c(x, \hat{u}, \bar{u}) + V^*(f(x, \hat{u}, w))], \quad (2.10)$$

and by the linearity of expectation

$$V^*(x) = \min_{\hat{u} \in \hat{U}(x)} \min_{\bar{u} \in \bar{U}(x, \hat{u})} (E_w[c(x, \hat{u}, \bar{u})] + E_w[V^*(f(x, \hat{u}, w))]). \quad (2.11)$$

The cost function $c(x, \hat{u}, \bar{u})$ is independent of w and the expectation may be dropped. The

expectation of the value function is independent of \bar{u} yielding

$$V^*(x) = \min_{\hat{u} \in \hat{U}(x)} \left(\min_{\bar{u} \in \bar{U}(x, \hat{u})} c(x, \hat{u}, \bar{u}) + E_w[V^*(f(x, \hat{u}, w))] \right). \quad (2.12)$$

Using the definition (2.8), (2.12) becomes (2.7). ■

2.2.2 Related Comments on Minimization Decomposition

To implement the controller developed using Minimization Decomposition, \bar{u} must still be determined. It may be precomputed and stored when calculating (2.8),

$$\bar{u}^*(x, \hat{u}) = \operatorname{argmin}_{\bar{u} \in \bar{U}(x, \hat{u})} c(x, \hat{u}, \bar{u}), \quad (2.13)$$

and

$$\hat{c}(x, \hat{u}) = c(x, \hat{u}, \bar{u}^*(x, \hat{u})) = \min_{\bar{u} \in \bar{U}(x, \hat{u})} c(x, \hat{u}, \bar{u}). \quad (2.14)$$

This process reduces the space of control actions by \bar{U} . The computation required to solve (2.4) or (2.5) scales linearly with the number of possible control actions, and can be significantly reduced depending on the problem structure and the size of \bar{u} .

Remark: (Functional Form to use Minimization Decomposition) Suppose a system has dynamics $f(x, \hat{u}, \bar{u}, w)$ that are independent of some control component \bar{u} and can be reformulated into a function \hat{f} , such that

$$\hat{f}(x, \hat{u}, w) = f(x, \hat{u}, \bar{u}, w) \quad (2.15)$$

with probability 1 (w.p. 1). Then the Bellman equation satisfies (2.7) and the Minimization Decomposition may be used. ■

While the property (2.15) seems quite restrictive, it occurs surprisingly often in the energy management problem. It is likely to occur if the number of control inputs M exceeds the dimension of the state space N , leaving a null control direction as used in [83].

Remark: (State Decomposition) In this energy management problem (as in most for-

mulations) the dynamics may clearly be broken down into two parts

$$f(x, u, w) = \begin{bmatrix} f_u(x, u) \\ f_w(x, w) \end{bmatrix} \quad (2.16)$$

where the deterministic states are the known vehicle dynamics and the stochastic driver dynamics are independent of the control input. ■

This allows the control inputs to be studied without the effect of w , simplifying the condition (2.15). Define P as the dimension of f_u , the state space that depends on u in (2.16). For problems where $M > P$, (2.15) likely holds.

The main point of this section is this: if the number of control inputs exceeds the number of states, the required computation can often be drastically reduced. Even with discrete states (i.e. gear number) the same techniques may often be used.

2.2.3 Examples

This structure (2.7) appears quite often in energy management problems. The decomposition has been exploited previously without explicit theoretical justification [39, 40, 64]. A typical example is the power-split configuration which uses engine power and speed as inputs without an engine speed state [64]. The fuel-minimizing engine speed (\bar{u}) for each engine power (\hat{u}) is precomputed and stored as a table. The vehicle considered in this dissertation allows a different usage of the decomposition; a physical explanation of the structure (2.7) and its implementation are also discussed.

2.2.3.1 Power-Split Hybrid

Consider for example the “Power-Split” architecture of the Toyota Prius and Ford Escape, with a cost function that penalizes fuel use and SOC deviations from nominal to attain charge sustenance. If one assumes that the dynamics of engine speed changes are negligible at the timescales for energy management, the only vehicle state is SOC, as velocity and acceleration are assigned by the driver (stochastically when using SPSDP). Assuming the vehicle matches driver demand torque, the system is defined by two inputs. By using

specific definitions of the system variables, the optimization reduces to two one degree of freedom problems. A common method is to treat the two control inputs as engine speed and engine power. Here we choose engine speed ω_{ICE} and electrical power P_{elec} , a slightly different definition. This allows a major decoupling of the system dynamics. The evolution of the the SOC state is now only dependent on $P_{elec} = \hat{u}$ and completely independent of $\omega_{ICE} = \bar{u}$. The engine speed that results in minimum fuel use for a given P_{elec} can be calculated off-line because it is independent of SOC. This results in engine fuel consumption as a 1-D function of power $\hat{c}(x, \hat{u}) = \hat{c}(x, P_{elec})$, rather than the standard 2-D functions of power and speed $c(x, \hat{u}, \bar{u}) = \hat{c}(x, P_{elec}, \omega_{ICE})$.

2.2.3.2 Super Electric Machine

The vehicle considered in this dissertation uses two electric machines, yielding an additional degree of freedom (see Chapter III). In comparison to previous work in [70] (also in Appendix B), the addition of a second electric machine makes the computation of a SPSDP solution potentially more complex. Exploiting structure (2.7) and using Minimization Decomposition reduced the computational cost to that of a vehicle with a single electric machine, a 90% reduction. The addition of the second electric machine is approximately “free” in terms of computation.

The system inputs require a tradeoff between the two electric machine torques. Define $\bar{T}_{(\cdot)}$ as the wheel torque delivered by a particular actuator. The system state is only affected by the total amount of torque delivered to the wheels \bar{T}_{SEM}

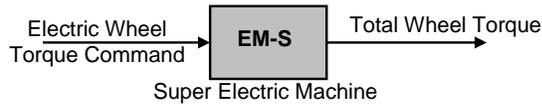
$$\hat{u} : \bar{T}_{SEM} = \bar{T}_{EM1} + \bar{T}_{EM2} \quad (2.17)$$

and not by the split between the two machines T_{DEM}

$$\bar{u} : \bar{T}_{DEM} = \bar{T}_{EM1} - \bar{T}_{EM2}. \quad (2.18)$$

This torque splitting may be considered \bar{u} . Since this one degree of freedom optimization is static (i.e., independent of the dynamic states of the model including SOC), it takes the

Mechanical Analog:



Internal Function:

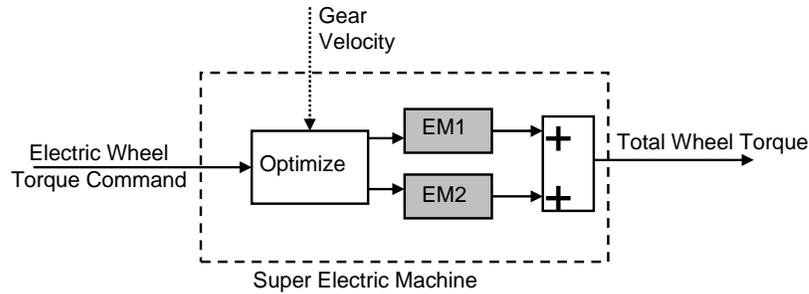


Figure 2.1: Schematic diagram of a conceptual “Super Electric Machine” that optimizes the mix between the two electric machines. This allows one degree of freedom of the control optimization to be carried out off-line while maintaining the optimality of the solution.

form (2.7) and can be computed *a priori* using (2.8) without loss of optimality. This reduces the dimension of the control space by one. The fundamental assumption that allows this to work is that the electric machine behaviors are dependent only on the current gear, engine state, and velocity, and not the past values of those states.

The physical control inputs to the system are engine torque, transmission gear, EM1 torque and EM2 torque. By replacing the two electric machine commands with a single Electric wheel torque command, the SPSDP algorithm has only 3 control inputs.

A more intuitive explanation is to treat the system as a single “Super Electric Machine.” This device is a black box that takes a desired wheel torque command as an input and uses the vehicle velocity and transmission gear to match the command torque with minimal electric power as shown in Figure 2.1. Once the optimization is complete, this device acts just like a normal electric machine for the SPSDP optimization. Internally, the device optimizes between the two (or possibly more) electric machines and issues appropriate commands.

2.3 Controlling Additional Behavior using SPSDP

2.3.1 Motivation

The design of effective energy-management controllers for production vehicles is a challenging problem because designers must consider many different vehicle behaviors. This dissertation studies drivability, but engineers must also consider other attributes like emissions, battery wear, durability, noise, etc. Even if an attribute is already considered, designers may want more detailed control. Regardless of the complexity of the current algorithm, designers are often seeking to control additional vehicle behaviors.

This section discusses methods for incorporating vehicle behavior into the SPSDP algorithm. The descriptions largely refer to a design process: a basic SPSDP controller exists, and a designer would like to adjust additional vehicle behavior. This approach covers several levels of complexity. If the basic SPSDP controller minimizes fuel, the designer may want to add drivability (as in this dissertation). A more advanced SPSDP controller may handle fuel and drivability but also require emissions control, for example.

These additional vehicle behaviors could also be addressed with a rule-based controller that operates downstream of the SPSDP controller, but it is more satisfying to incorporate these behaviors in the problem formulation. If these vehicle behaviors require additional model states, solving the Bellman equation for SPSDP can become intractable, so here we discuss alternative methods with their benefits and drawbacks.

2.3.2 Formulation Methods

Fundamentally, controlling vehicle behavior requires the designer to specify a cost function for a model. Suppose we have a basic cost function $c(x, u)$ and model $f(x, u, w)$ that incorporates some level of functionality. The standard on-line SPSDP controller is

$$u^*(x) = \underset{u \in U}{\operatorname{argmin}} E_w [c(x, u) + V^*(f(x, u, w))]. \quad (2.19)$$

We discuss three possible methods to incorporate additional behaviors which offer a tradeoff between computation and optimality:

2.3.2.1 Additional Penalties

It may be possible to control additional behavior by adding penalties to the cost function using only the states in the base model $f(x, u, w)$. If so, this yields a new cost function,

$$c_{pen}(x, u) = c(x, u) + \alpha \mathbf{I}_{Event1}(x, u) + \beta \mathbf{I}_{Event2}(x, u) \dots \quad (2.20)$$

where $\mathbf{I}(x, u)$ is the indicator function of when a behavior occurs.

The solution complexity for the value function size remains the same since the base model is still used,

$$V_{pen}^*(x) = \min_{u \in U} E_w [c_{pen}(x, u) + V_{pen}^*(f(x, u, w))]. \quad (2.21)$$

This is termed the ‘‘Additional Penalties’’ method and yields an optimal control law. Even though the value function computation remains the same, $V_{pen}^*(x)$ must be solved for each set of penalty values $(\alpha_1, \beta_1, \dots)$ in (2.20), so number of times $V_{pen}^*(x)$ must be evaluated to tune the controller is exponential in the number of penalty parameters (α, β, \dots) .

2.3.2.2 Extended Model

If the new behavior of interest requires additional states \tilde{x} or control inputs \tilde{u} , the existing model is extended to include them

$$x_e = [x, \tilde{x}] \quad (2.22)$$

$$u_e = [u, \tilde{u}] \quad (2.23)$$

$$f_e(x_e, u_e, w). \quad (2.24)$$

The SPSDP controller design uses the extended cost $c_e(x_e, u_e)$ and dynamic model $f_e(x_e, u_e, w)$, and the new value function $V_e^*(x)$ is solved using the Bellman equation,

$$V_e^*(x_e) = \min_{u_e \in U_e} E_w [c_e(x_e, u_e) + V_e^*(f_e(x_e, u_e, w))]. \quad (2.25)$$

This is the standard, provably optimal method and is implemented using the on-line minimization (2.19). This is termed the “Extended Model” method. The complexity of computing the value function grows exponentially with system states, requiring more off-line computation. The number of value function solutions to tune the controller is still exponential in the number of penalties (α, β, \dots) in the cost function. This is the process used in this dissertation to include the simplified drivability metrics.

2.3.2.3 Instantaneous Cost

Additional states and penalties may also be incorporated solely in the on-line cost function [70, 94, 95] (see Appendix B). A basic SPSDP controller is designed first, and other behaviors are added later only in the on-line cost function (2.19). The value function $V(x)$ is first calculated via the Bellman equation using a simple cost $c(x, u)$ and model $f(x, u, w)$ that only includes minimal detail,

$$V^*(x) = \min_{u \in U} E_w[c(x, u) + V^*(f(x, u, w))]. \quad (2.26)$$

The real-time controller is then implemented using the simple value function and an extended cost function $c_e(x_e, u_e)$ that includes the additional vehicle behavior. The real-time controller must still track the extended set of states, but the value function is only solved for a reduced state space, requiring much less computation. The real-time controller is then

$$u^*(x_e) = \operatorname{argmin}_{u_e \in U_e} E_w[c_e(x_e, u_e) + V^*(f(x, u, w))]. \quad (2.27)$$

The value function is approximated using the simple model $f(x, u, w)$, while the cost function is calculated using the extended model. In this case, the new behavioral restrictions are still implemented via optimization, but they are only instantaneous in the sense that there is no estimate of the future cost [95] because they are not included in the value function. This method is termed the “Instantaneous Cost” method. It can be implemented on-line with no additional off-line computation because it uses the original value function; it can modify vehicle behavior but the optimality guarantees are lost. The performance loss is

highly dependent on the specific problem; drivability is studied in [70] and Appendix B.

2.3.3 Penalty Implementation

Designers must carefully consider how events and penalties are defined; small changes in definition can save large amounts of computation. The selection of the drivability definitions and cost function for this work was not easy, despite the simplicity of the final metrics.

Generating usable controllers requires three steps: selecting the cost function, calculating the value function off-line, and running a simulation. The mapping from penalty to behavior is not known a priori, so the simulation must be conducted to determine the behavior; tuning penalties accounts for most of the total computation. The relative benefits and drawbacks of the three methods are shown in Table 2.1.

These three options are illustrated with examples from this dissertation: Simplified drivability metrics were added to a fuel-only SPSDP controller using the Extended Model method by adding states to the value and cost functions. Once the current transmission gear state was included to track gear events, we added a penalty for non-sequential shifts (ie. $1^{st} - 3^{rd}$), without additional states, using the Additional Penalties method. An Instantaneous Cost implementation can be used to limit the slew rate of engine power, which would otherwise require an additional state representing current engine power. This slew rate restriction was not added in the simulations (Chapters IV and V), but was used in the hardware implementation (Chapter VIII). These remarks may become more clear after reading the remainder of this dissertation.

Table 2.1: Comparison of Event Implementations

Option	Exponential growth in $V(x)$ solution complexity	Recompute $V(x)$ for each tuning	Optimal	Rerun simulation for each tuning	Usable for any behavior
Additional Penalties		X	X	X	
Extended Model	X	X	X	X	X
Instantaneous Cost				X	X

Metrics and penalties for behaviors other than simplified drivability were also included or are planned, but have not been extensively studied. Their implementations are listed in Table 2.2 as examples. The Additional Penalties method is generally preferred because it maintains optimality, but not all penalties can be implemented this way; some behaviors required additional states. Several methods are often possible; Table 2.2 refers to the im-

plementation we selected. Although in some cases additional penalties are computationally “free,” from a practical perspective the number of possible parameter tunings still grows exponentially with additional behaviors.

Table 2.2: Additional Event implementations

Behavior	Method
Series Mode Entry & Exit	Additional Penalties
Pedal Correlation with Engine Noise	Additional Penalties
Pedal Correlation with Engine Speed	Additional Penalties
Pedal Correlation with Engine Torque	Additional Penalties
Only single-step gear shifts	Additional Penalties
Only even-odd gear shifts	Additional Penalties
Emissions	Extended Model
Time delay between drivability events	Extended Model
Climate Control Loading	Extended Model
Slew rate of Engine Power	Instantaneous Cost
Slew rate of Engine Speed	Instantaneous Cost

2.3.4 Computation

As an example, this section quantifies the computational burden of adding the simplified drivability metrics using the Extended Model and Instantaneous Cost methods described in Section 2.3.2.

As mentioned previously, after defining the cost function there are two basic steps: calculating the value function $V(x)$ off-line, and then implementing it in a controller to drive a cycle. The on-line implementation requires calculating the current cost $c(x, u)$, calculating the set of next states x_{k+1} , and interpolating into the precomputed $V(x)$.

Both methods require similar operations to calculate the current cost function, which is somewhat trivial. The main difference is in calculating the value function, which uses table interpolation both on-line and off-line. The number of points used for each table is shown in Table 2.3 for comparison.

Each point in the on-line table represents a state, and each point in the off-line table represents a state, control, and disturbance combination. The scaling between off- and on-line table sizes is roughly the number of control and disturbance combinations available at each state, a factor of 126 for the Instantaneous Cost method and 882 for the Full

Table 2.3: Comparison of Computation Requirements

Method	Off-Line Table Size	On-Line Table Size
Extended Model	$1.2 \cdot 10^7$	$1.4 \cdot 10^4$
Instantaneous Cost	$1.3 \cdot 10^5$	$1 \cdot 10^3$

Model method. Off-line computations can be conducted on a desktop PC. The main point here is that the Instantaneous Cost method requires 2 orders of magnitude less off-line computation, but both methods have relatively simple on-line implementations that run in real-time.

CHAPTER III

Problem Formulation and Approach for Energy Management of a Prototype Hybrid Vehicle

3.1 Vehicle

3.1.1 Vehicle Architecture

The vehicle studied in this dissertation is a prototype Volvo S-80 series-parallel electric hybrid and is shown schematically in Figure 3.2. 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 3.1.



Figure 3.1: The Prototype Hybrid: A Modified Volvo S-80.

Table 3.1: 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
Vehicle Mass	1895 kg

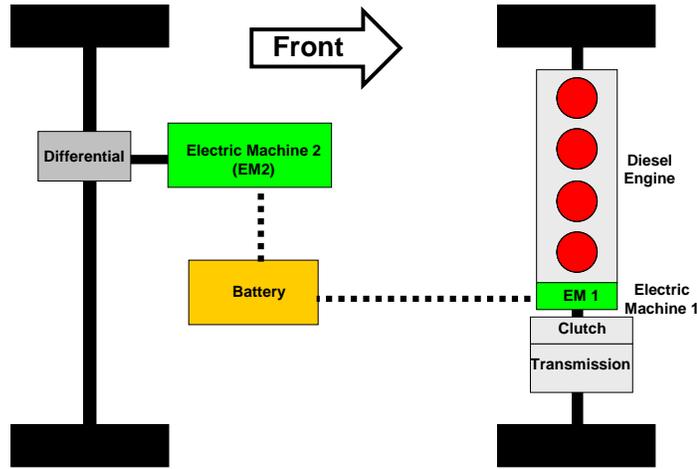


Figure 3.2: Vehicle Configuration

3.1.2 Vehicle Models

The work presented in this dissertation uses two different dynamic models to represent the same prototype hybrid vehicle. The first model is quite simple; it has a sample time of 1s, uses lookup tables, and has very few states. It is used primarily to design the controller and do the optimization, and is called the “control-oriented” model.

The second model comes from Ford Motor Company and uses its in-house modeling architecture. This sophisticated model is used to evaluate fuel economy and controller behavior by simulating controllers on drive cycles. This model is referred to as the “vehicle simulation” model in this dissertation [6].

This combination of models allows the controller to be designed on a simple model that keeps the problem feasible, while providing accurate fuel economy results on a highly representative model that is different from the design model.

Note that Chapter VIII introduces additional models used for the real-time implementation

3.1.3 Control-Oriented Model

When using Shortest-Path Stochastic Dynamic Programming, the off-line computation cost is very sensitive to the number of system states. For this reason, the model used to develop the controller must be as simple as possible. The vehicle model used here contains the minimum functionality required to model the vehicle behavior of interest on a second-by-second basis. Dynamics much faster than the sample time of 1s are ignored. Long-term transients that only weakly affect performance are also ignored; coolant temperature is one example.

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*.

The model does not restrict the direction of power flow. The electric machines can be either motors or generators in all modes.

The dynamics of the internal combustion engine are ignored; it is assumed that the engine torque exactly matches valid commands and the fuel consumption is a function only of speed, ω_{ICE} , and torque, T_{ICE} . The fuel consumption F is derived from a lookup table based on dynamometer testing,

$$Fuel\ flow = F(\omega_{ICE}, T_{ICE}).$$

The dual clutch transmission has discrete gears and no torque converter. The transmission is modeled with a constant mechanical efficiency of 0.95. Transmission gear shifts are allowed every time step (1s) and transmission dynamics are assumed negligible. When

the clutch is engaged, the vehicle is in **Parallel Mode** and the engine speed is assumed directly proportional to wheel speed based on the current transmission gear ratio R_g ,

$$\omega_{ICE} = R_g \omega_{wheel}.$$

The electric machine $EM1$ is directly coupled to the crankshaft, and thus rotates at the engine speed ω_{ICE} ,

$$\omega_{EM1} = \omega_{ICE}.$$

In **Parallel Mode**, the engine torque T_{ICE} and $EM1$ torque T_{EM1} transmitted to the wheel are assumed proportional to wheel torque based on the current gear ratio R_g and the transmission efficiency η_{trans} . The rear electric machine torque T_{EM2} transmitted to the wheel is proportional to the machine's gear ratio R_{EM2} and rear differential efficiency η_{diff} . The total wheel torque T_{wheel} from both wheels is thus the sum of the ICE and $EM1$ torques to the wheel $\eta_{trans} R_g (T_{ICE} + T_{EM1})$ and the rear electric machine $EM2$ torque to the wheel $\eta_{diff} R_{EM2} T_{EM2}$,

$$\eta_{trans} R_g (T_{ICE} + T_{EM1}) + \eta_{diff} R_{EM2} T_{EM2} = T_{wheel}. \quad (3.1)$$

The clutch can be disengaged at any time, and power is delivered to the road through the rear electric machine $EM2$. This condition is treated as the “neutral” gear 0, which combines with the 6 standard gears for a total of 7 gear states. If the engine is on with the clutch disengaged, the vehicle is in **Series Mode**. The engine- $EM1$ combination acts as a generator and can operate at an arbitrary torque and speed. If the engine is off while the clutch is disengaged, the vehicle is in **Electric Mode**. The clutch is never engaged with the engine off, so this mode is undefined and not used.

The battery system is similarly reduced to a table lookup form. The electrical dynamics due to the motor, battery, and power electronics are assumed sufficiently fast to be ignored. The energy losses in these components can be grouped together such that the change in battery State of Charge (SOC) is a function $\bar{\kappa}$ of electric machine speeds ω_{EM1} and ω_{EM2} ,

Table 3.2: Vehicle Mode Definitions.

Mode	Clutch State	Engine State	Gear State
Electric	Disengaged	Off	0
Series	Disengaged	On	0
Parallel	Engaged	On	1-6
Undefined/not used	Engaged	Off	1-6

torques T_{EM1} and T_{EM2} , and battery SOC at the present time step,

$$SOC_{k+1} = \bar{\kappa}(SOC_k, \omega_{EM1}, \omega_{EM2}, T_{EM1}, T_{EM2}). \quad (3.2)$$

Assuming a known vehicle speed, the only state variable required for this vehicle model is the battery SOC. Changes in battery performance due to temperature, age, and wear are ignored. During operation, the desired wheel torque is defined by the driver. If we assume the vehicle must meet the torque demand perfectly, then the sum of the ICE and EM contributions to wheel torque (3.1) must equal the demanded torque T_{demand} ,

$$T_{wheel} = T_{demand}. \quad (3.3)$$

This adds a constraint to the control optimization, reducing the 4 control inputs to a 3 degree of freedom problem. In **Parallel Mode** the control inputs are *Engine Torque*, *EM1 Torque*, and *Transmission Gear*. In **Series Mode**, the electric machine command becomes *EM1 Speed*.

Optimization using the control-oriented model will assume a “perfect” driver. The desired road power is calculated as the exact power required to drive the cycle at that time. Now, given vehicle speed, demanded road power and this choice of control inputs, the dynamics become an explicit function κ of the state *Battery SOC* and the three control choices as shown in Figure 3.3,

$$SOC_{k+1} = \kappa(SOC_k, T_{ICE}, T_{EM1}, Gear). \quad (3.4)$$

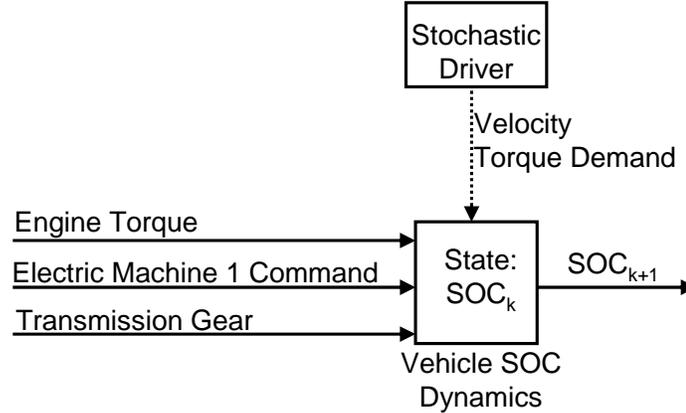


Figure 3.3: Vehicle SOC Dynamics model. The three inputs are Engine Torque, Electric Machine 1 Torque, and Transmission Gear. The vehicle velocity and required torque are provided by the stochastic driver model. In Series Mode, the Electric Machine 1 command is speed rather than torque.

In **Series Mode**, T_{EM1} is replaced with ω_{EM1} . The engine fuel consumption can be calculated from the control inputs.

Operational Assumptions:

This control-oriented model uses several assumptions about the allowed vehicle behavior.

1. The clutch in the transmission allows the diesel engine to be decoupled from the wheels.
2. There is no ability to slip the clutch for starts.
3. There are no traction control restrictions on the amount of torque that can be applied to the wheels.

3.1.4 Vehicle Simulation Model

As part of this project, Ford provided an in-house model used to simulate fuel economy. It is a complex, MATLAB/Simulink based model with a large number of parameters and states [6]. Every individual subsystem in the vehicle is represented by an appropriate block. For each new vehicle, subsystems are combined appropriately to yield a complete system.

This vehicle simulation model contains the baseline controller algorithm. To generate simulation results using this controller, an automated driver follows the target cycle using the baseline controller.

To use the vehicle simulation model with the control algorithm developed here, the SPSDP controller is implemented in Simulink by interfacing appropriate feedback and command signals: Battery State of Charge, Vehicle Speed, Engine State, Gear Command, etc. The vehicle simulation model can then be “driven” by the SPSDP controller along a given drive cycle.

3.2 Drivability Constraints

3.2.1 Motivation

Customer perception is a crucial component in vehicle purchasing decisions. The driver’s perception of overall vehicle response and behavior is termed “drivability.” Manufacturers are very aware of this and exert significant development effort to satisfy drivability requirements. Generally speaking, drivability concerns affect designs as much as fuel economy goals.

Drivability is a rather vague term that covers many aspects of vehicle performance including acceleration, engine noise, braking, automated shifting activity, shift quality [83], and other behaviors. Improving drivability often comes at the expense of fuel economy. For example, optimal fuel economy for gasoline engines typically dictates upshifting at the lowest speed possible, but this leaves the driver little acceleration ability after the upshift. Thus upshifts are scheduled to occur at higher speeds than what is optimal for fuel economy. This dissertation addresses the “basic” drivability issues of gear selection and when to start or stop the internal combustion engine.

Current academic work in hybrid vehicle optimization primarily focuses on fuel economy. Such results are somewhat less useful to industry because of drivability restrictions in production vehicles. If these fuel-optimal controllers are used, drivability restrictions are typically imposed as a separate step (see Appendix B) [70].

This dissertation investigates the usefulness of optimizing for fuel economy and drivability simultaneously. By including these real-world concerns, one can generate controllers that improve performance and are one step closer to being directly implementable in production.

3.2.2 Simplified Drivability Metrics

In the context of the overall system, two significant characteristics that are noticeable to the driver are the basic behaviors of the transmission and engine. These are included in the simulation model presented in Section 3.1. To effectively design controllers, qualitative drivability requirements must be transformed into quantitative restrictions or metrics. Drivability experts at Ford Motor Company were consulted to assist in developing numerical drivability criteria. Developing simple, quantitative drivability metrics was a major enabling contribution of this dissertation. This process is discussed in detail in Chapter VII.

Two baseline metrics are used to quantify behavior for a particular trip. The first is *Gear Events*, the total number of shift events on a given trip. The second metric is *Engine Events*, the total number of engine start and stop events on a trip.

By definition, engine starts and stops are each counted as an event. Each shift is counted as a gear event, regardless of the change in gear number. A 1st – 2nd shift is the same as a 1st – 3rd shift. Engaging or disengaging the clutch is not counted as a gear event, regardless of the gear before or after the event.

These simplified metrics are useful because they can be implemented in the optimization process with few additional states, generate acceptable vehicle behavior, and are well correlated with more sophisticated metrics as shown in Chapter VII.

3.3 Shortest Path Stochastic Dynamic Programming

3.3.1 Cost Function

In order to design a controller with acceptable drivability characteristics, the optimization goal over a given trip of length T would ideally be defined as

$$\begin{aligned} \min \sum_0^T Fuel\ flow \\ \text{such that} \\ \sum_0^T GE \leq GE_{max}, \sum_0^T EE \leq EE_{max} \end{aligned} \tag{3.5}$$

where GE and EE are the number of Gear and Engine Events respectively, and GE_{max} and EE_{max} are the maximum allowable number of events on a cycle. T is the time duration from “key-on,” the start of the trip, to “key-off,” the end of the trip.

This constrained optimization incorporates the two major areas of concern: fuel economy and drivability. Constraints of this type cannot be incorporated in the Stochastic Dynamic Programming (Chapter II) algorithm used here because the stochastic nature of the optimization cannot directly predict performance on a given cycle. Instead, the drivability events are included as penalties, and those penalty weights are adjusted until the outcome is acceptable and meets the hard constraints.

Controllers based only on fuel economy and drivability completely drain the battery as they seek to minimize fuel. An additional cost is added to ensure that the vehicle is charge sustaining over the cycle. This SOC-based cost only occurs during the transition to key-off, so it is represented as a function $\phi_{SOC}(x)$ of the state x , which includes SOC. The performance index for a given drive cycle is then

$$J = \sum_0^T Fuel\ flow + \alpha \sum_0^T GE + \beta \sum_0^T EE + \phi_{SOC}(x_T). \quad (3.6)$$

The search for the weighting factors α and β involves some trial and error, as the mapping from penalty to outcome is not known a priori. Note that setting α and β to zero means solving for optimal fuel economy only.

Now, to implement the optimization goal of minimizing (3.6), a running cost function is prescribed as a function only of the state x and control input u at the current time

$$c_{full}(x, u) = F(x, u) + \alpha \mathbf{I}_{GE}(x, u) + \beta \mathbf{I}_{EE}(x, u) + \phi_{SOC}(x) \quad (3.7)$$

where the functions $\mathbf{I}(x, u)$ are the indicator function and show when a state and control combination produces a Gear Event or Engine Event. Fuel use is calculated by $F(x, u)$. The SOC-based cost $\phi_{SOC}(x)$ applies only at the end of the trip, when the system transitions to the key-off absorbing state. Many other vehicle behaviors can be optimally controlled by adding appropriate functions of the form $\phi(x, u)$; a typical example is limiting SOC

deviations during operation to reduce battery wear; see Section 2.3.

3.3.2 Problem Formulation

To determine the optimal control strategy for this vehicle, the Shortest Path Stochastic Dynamic Programming (SPSDP) algorithm is used as discussed in Chapter II. This method directly generates a causal controller; characteristics of future driving behavior are specified via a finite-state Markov chain rather than exact future knowledge. The system model is formulated as

$$x_{k+1} = f(x_k, u_k, w_k), \quad (3.8)$$

where u_k is a particular control choice in the set of allowable controls U , x_k is the state, and w_k is a random variable arising from the unknown drive cycle. Recall that the disturbance w may be conditioned on the state and control input,

$$P(w|x_k, u_k). \quad (3.9)$$

In order to use SPSPDP, the driver demand is modeled as a Markov chain. This “driver” is assigned two states: current velocity v_k and current acceleration a_k , which are included in the full system state x . The unknown disturbance w in (2.4) and (3.8) is the acceleration at the next time step, which is assigned a probability distribution. This means estimating the function

$$P(a_{k+1}|v_k, a_k) \quad (3.10)$$

for all states v_k, a_k . The transition probabilities (3.10) are estimated from known drive cycles that represent typical behavior, dubbed the “design cycles.” The variables v_k , a_k , and a_{k+1} are discretized to form a grid. For each discrete state $[v_k, a_k]$ there are a variety of outcomes a_{k+1} . The probability of each outcome a_{k+1} is estimated based on its frequency of occurrence during the design cycle, and is clearly a function of state as in (3.9). See [59, 94] for more detail.

To track the gear event and engine event metrics in Section 3.2, two additional states are required: the *Current Gear* (1-6) and *Engine State* (on or off).

Bringing this all together, the full system state vector x contains five states: one state for the vehicle (*Battery SOC*), two states for the stochastic driver (v_k, a_k), and two states to study drivability (*Current Gear* and *Engine State*). This formulation is termed the “SPDP-Drivability” controller. A summary of system states is shown in Table 3.3. The control u contains the two inputs *Engine Torque* and *Transmission Gear*, as described in Section 3.1.3 and Table 3.3.

Table 3.3: Vehicle Model States

State	Units
Battery Charge (SOC)	[0-1]
Vehicle Speed	m/s
Current Vehicle Acceleration	m/s^2
Current Transmission Gear	Integer 1-6
Current Engine State	On or Off

Remark: As demands on controller functionality grow, so also must the complexity of the design model. For example, to study fuel economy using deterministic dynamic programming, the only state required is the battery state of charge; the control inputs are *Engine Torque* and *Transmission Gear*. Two more states are required to study the stochastic version, and the drivability model requires two additional states.

3.3.3 Terminal State

As mentioned in Chapter II, the dynamics of the system must contain an absorbing state. For this case, the absorbing state represents “key-off,” when the driver has finished the trip, shut down the vehicle, and removed the key. Once the key-off event occurs, there are no further costs incurred, the trip is over, and the vehicle cannot be restarted. The probability of transitioning to this state is zero unless the vehicle is completely stopped ($v_k, a_k = 0$). The probability of a trip ending once the vehicle is stopped is calculated based on the design cycles. This probability is less than one because a stopped vehicle could represent a traffic light or other typical driving event that does not correspond to the end of a trip.

For fuel economy certification, the battery final SOC must be close to the initial SOC

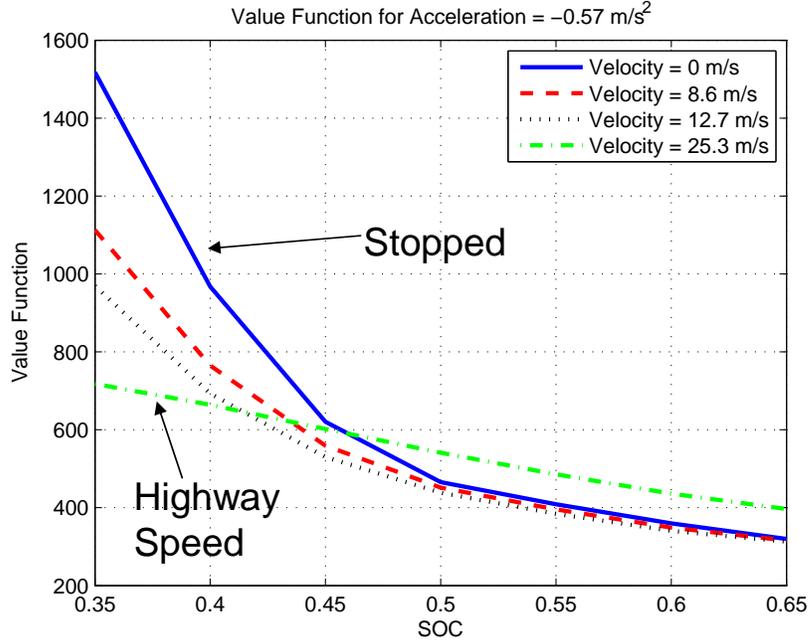


Figure 3.4: Value Function $V(x)$ for several velocities and fixed deceleration at -0.57 m/s^2 . The quadratic penalty on SOC strongly affects the value function at low speeds when the driver is more likely to turn the key off and end the trip.

or else the test is invalid. To include this in the SPDP formulation, a cost is imposed when the vehicle transitions into the key-off state and the SOC is less than the initial SOC. This penalty accrues only once, so the absorbing state has zero cost from then onwards. Here we add a quadratic penalty in SOC if the final SOC is less than the initial SOC. No penalty is assigned if the final SOC is higher than the initial SOC.

The effects of this key-off penalty are clearly visible in the value function $V(x)$. For the fuel-only case, the value function depends on the current acceleration, velocity, and SOC. Figure 3.4 shows $V(x)$ as a function of SOC for one particular acceleration and several velocities, with target final SOC equal to 0.5. Notice that at low velocities, the value function has a pronounced quadratic shape for SOC under 0.5, but it flattens out at higher speeds. The SOC penalty only occurs at key-off, which can only occur at zero speed. Thus the SOC key-off penalty strongly affects the value function at low speeds, when there is a higher probability of key-off in the near future. At higher speeds, there is little chance of key-off anytime soon, so the SOC penalty only weakly affects the value function.

3.3.4 Implementable Constraints

Stochastic Dynamic Programming is inherently computationally intensive and can quickly become intractable. The computation burden is exponential in the number of system states; thus the cost function (3.7) should depend on a minimal number of states. The selection and implementation of the cost function is discussed extensively in Section 2.3.

For optimization, at each time step a penalty is assigned if either a shift or engine event occurs. The only two states required to implement this cost function are the current gear and the engine state. Even so, including drivability in the optimization imposes roughly a factor of ten increase in computation over the fuel-only case.

In contrast, suppose the metric of interest were based on a moving window in time. The number of grid points required scales with the number of time steps used to specify the metric. For the 1 s update time studied here, penalizing engine events 5 seconds or less (rather than the simple on/off) would require 5 grid points for the time history, increasing the size of the state-space by the same factor of 5 over the on/off case.

3.4 Comparison of SPSDP to the Equivalent Consumption Minimization Strategy (ECMS)

One of the most well known optimization methods for energy management in HEVs is known as the “Equivalent Consumption Minimization Strategy” (ECMS) [67, 77]. This method optimizes for fuel economy only; it requires little computation and is easy to implement. At each time step, the controller minimizes a function that trades off battery usage vs. fuel,

$$u_k^*(x) = \operatorname{argmin}_{u \in U} [Fuel(x, u) + \lambda_k \Delta SOC(x, u)]. \quad (3.11)$$

The design parameter is the weighting factor λ_k , which represents the relative value of battery charge in terms of fuel. The difficulty arises in calculating this weighting factor as it is highly cycle dependent.

Consider now the SPSDP algorithm for the fuel only case. The cost function $c(x, u)$ in (2.4) is not a function of the random variable w and can be removed from the expectation.

The value function $V(x)$ can be linearized about the operating point, transforming (2.5) into (3.12). This is a valid approximation because SOC only changes slightly at each time step,

$$u^*(x) = \underset{u \in U}{\operatorname{argmin}} [c(x, u) + \frac{\partial Q(x, u)}{\partial \text{SOC}} \Delta \text{SOC}(x, u)] \quad (3.12)$$

where

$$Q(x, u) = E_w[V(f(x, u, w))]. \quad (3.13)$$

Notice that the local slope of the value function $\frac{\partial Q}{\partial \text{SOC}}$ in (3.12) is equivalent to the weighting factor λ in (3.11). The SPDP algorithm has the same structure as the ECMS method, but the weighting factor is a function of several variables. There is a variant of ECMS method called Adaptive ECMS (A-ECMS) in which the weighting factor is allowed to change over time based on the current driving conditions [67]. A-ECMS is even more similar to the SPDP algorithm in that both methods have a non-constant weighting factor.

This relationship is clearly illustrated by again studying the value function $V(x)$ as a function of SOC for fixed acceleration and velocity as shown in Figure 3.4. The local slope of $V(x)$ in the figure is exactly the weighting factor $\frac{\partial Q}{\partial \text{SOC}}$ in (3.12) and analogous to λ in (3.11).

Fundamentally, all fuel-minimizing control algorithms must estimate the value of battery charge in terms of fuel and thus have some equivalent to the weighting factor λ . It may appear explicitly as in ECMS, or implicitly as in SPSDP. Once this weighting factor is determined, the control problem is a simple static optimization. All known information is incorporated in this weighting factor, including plant dynamics, states, and expected future driving patterns.

A basic difference between algorithms lies in how they estimate this weighting factor: ECMS uses a value assigned by the designer; A-ECMS estimates the value based on battery charge and recent history; Deterministic Dynamic Programming uses exact future knowledge; and SPSDP uses estimates of cycle statistics. In addition, the dynamic programming methods like SPSDP can also optimally incorporate constraints on behavior, like the drivability metrics studied here.

CHAPTER IV

Simulations on Government Test Cycles

4.1 Introduction

This chapter evaluates the controllers developed in Chapter III on government test cycles. This serves to validate the basic usefulness of the SPSDP method. Controllers are designed for a wide range of drivability penalties, which yields tradeoff curves between different attributes. The SPSDP controllers are also compared to a baseline controller developed at Ford for this vehicle.

4.2 Simulation Procedure

SPSDP-based controllers are compared to a baseline industrial controller. SPSDP controllers are designed using the control-oriented model and evaluated using the Simulink-based vehicle simulation model. This demonstrates some robustness by using two models of the same vehicle, differing in the level of detail in their dynamics.

Both SPSDP and the baseline controllers are simulated on two government test cycles, the US Federal Test Procedure (FTP) and the New European Drive Cycle (NEDC), which are shown in Figure 4.1. Procedurally, this is conducted as follows:

1. A “family” of SPSDP controllers is designed according to the methods of Section 3.3. A *family* is generated by fixing the model driving statistics and sweeping the 2 drivability penalties α and β in (3.7).
2. Each controller in the *family* is simulated on the vehicle simulation model.

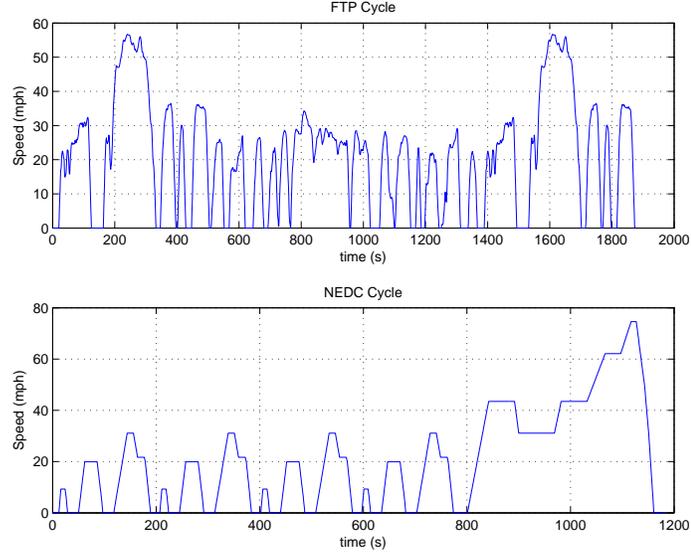


Figure 4.1: FTP and NEDC cycles.

3. The fuel consumption and drivability metrics are recorded.

In the end, each *family* contains a few hundred individual controllers which have each been simulated on the cycle in question. Each simulation yields a data point with associated fuel economy and drivability metrics. Each controller in the *family* has different drivability and fuel consumption characteristics because of the varying drivability penalties.

Each controller is simulated on the vehicle simulation model discussed in Section 3.1.4. The simulations are all causal, so the final SOC is not guaranteed to exactly match the starting SOC. This could yield false fuel economy results, so all fuel economy results are corrected based on the final SOC of the drive cycle. This is done by estimating the additional fuel required to charge the battery to its initial SOC, or the potential fuel savings shown by a final SOC that is higher than the starting level. This correction is applied according to

$$\Delta F_{fuel} = C_{Batt} \Delta SOC \frac{BSFC_{min}}{\eta_{max}^{Regen}} \quad (4.1)$$

where ΔF_{fuel} is the adjustment to the fuel used, C_{Batt} is the battery capacity, ΔSOC is the difference between the starting and ending SOC, $BSFC_{min}$ is the best Brake Specific Fuel Consumption for the engine, and η_{max}^{Regen} is the best charging efficiency of the electric system.

Fuel economy numbers in this dissertation include the SOC correction unless stated otherwise, and are normalized so that the baseline controller running the FTP cycle has a fuel economy of one.

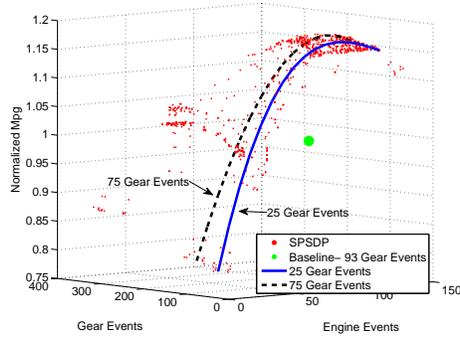
4.3 Results: Performance Trends

4.3.1 Fuel Economy Results

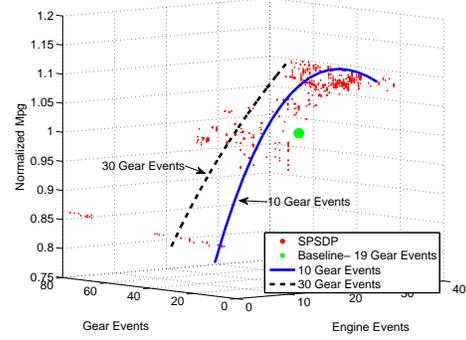
The main goal of this section is to trade off fuel economy and drivability requirements by using a class of optimal controllers, and validate the result against industrial design methods. The three metrics of interest during vehicle driving are the number of *Gear Events*, *Engine Events*, and the total fuel consumption corrected for SOC. These metrics yield conflicting goals and there is a distinct tradeoff among them. To study this tradeoff, several hundred controllers are designed with varying penalties assigned to each *Gear Event* and *Engine Event*. This creates one *family* of controllers as described in Section 4.2. The results are shown for FTP and NEDC cycles in Figure 4.2.

After simulation, the resulting data show the tradeoff between fuel economy and drivability. The typical result is a 3-D scatterplot of one *family* of controllers as shown in Figure 4.2a. Each point represents a single controller driven on the cycle in question. The controllers are all driven on the same test cycle. The total number of *Gear Events*, and *Engine Events* are shown on the horizontal axes, while fuel economy is shown on the vertical axis as normalized MPG. The combination of these points form a surface in 3-D space depicting the tradeoff surface of various operating conditions. This particular figure shows a *family* of controllers designed using FTP statistics running the FTP cycle. Fuel economy for all nondimensional results is normalized to the baseline controller performance on FTP, shown as a large green circle. A response surface is fit to the raw data and used to generate isoclines of constant gear, shown as solid (blue) and black (dashed) lines.

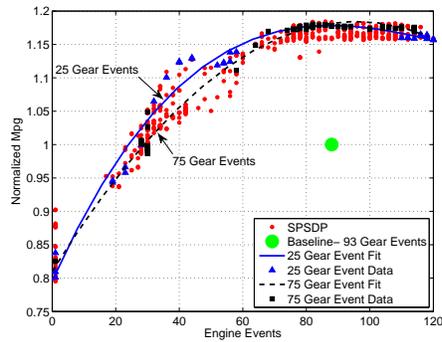
Figure 4.2c is a 2-D view of Figure 4.2a looking along the *Gear Events* axis. Each line in the plot represents a constant number of Gear Events, while the horizontal and vertical axes show the number of *Engine Events* and normalized fuel economy respectively. This particular vehicle is relatively insensitive to the number of Gear Events, so most of the



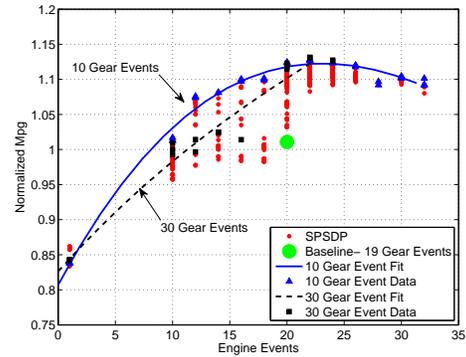
(a) FTP: Fuel Economy 3-D Scatterplot. Isoclines of constant *Gear Events* (GE) are shown as solid blue (25 GE) and dashed black (75 GE) lines.



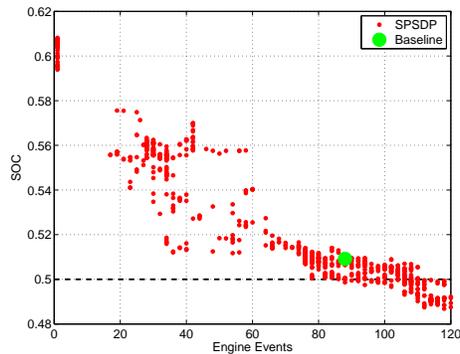
(b) NEDC: Fuel Economy 3-D Scatterplot. Isoclines of constant *Gear Events* (GE) are shown as solid blue (10 GE) and dashed black (30 GE) lines.



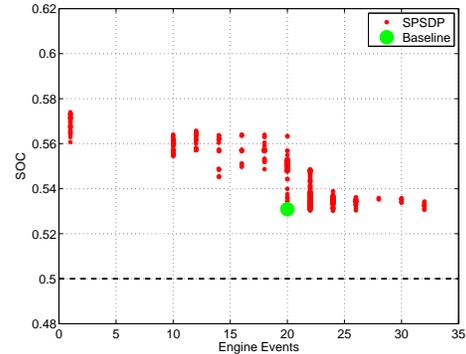
(c) FTP: 2-D view along the *Gear Events* axis of subfigure 4.2a. The data used to fit the constant *Gear Events* isoclines are shown as blue triangles (25 GE) black squares (75 GE).



(d) NEDC: 2-D view along the *Gear Events* axis of subfigure 4.2b. The data used to fit the constant *Gear Events* isoclines are shown as blue triangles (10 GE) black squares (30 GE).



(e) FTP: Final SOC for the cycle.



(f) NEDC: Final SOC for the cycle.

Figure 4.2: Performance of SPSPD controllers on FTP and NEDC. A separate *family* of controllers is designed for FTP and NEDC using each cycle's statistics. The family is designed by sweeping the parameters α and β in the cost function (3.7). Figures 4.2a and 4.2b show the data as a 3-D scatterplot of fuel economy vs. drivability events. SPSPD controllers are shown as red dots. The baseline controller is shown as a large green circle. Figures 4.2c and 4.2d show the view along the *Gear Events* axes of Figures 4.2a and 4.2b respectively. The raw data points, isoclines, and baseline controller are still visible. Figures 4.2e and 4.2f show the final SOC for these controllers. All controllers start with SOC=0.5.

results concentrate on the tradeoff between engine activity and fuel economy. The final SOC for these simulations is shown in Figure 4.2e. All simulations start at 0.5 SOC.

Similarly, a *family* of controllers is designed and simulated on the NEDC cycle. Fuel economy results are again shown in 3-D and 2-D in Figures 4.2b and 4.2d, while the final SOC is shown in Figure 4.2f. Again, fuel economy is normalized to the baseline controller performance on FTP, so the baseline controller performance is slightly better on NEDC (1.01) than FTP (1.00).

4.3.2 Discussion

The frontiers of the 2-D and 3-D point clouds in Figure 4.2 clearly demonstrate the tradeoff between fuel economy and drivability. Assuming the same *a priori* information (causal controllers) and a given Markov chain model, no controller’s average performance can exceed the SPSDP frontier. The optimality guarantee for SPSDP is based on an expectation of driver behavior. A different controller could do better on a *particular* realization of the statistics (a particular cycle).

The plot of final SOC for the FTP cycle (Figure 4.2e) shows a distinct downward trend for large number of engine events. The target final SOC is 0.5, which the controllers come very close to achieving when engine events are unrestricted (low penalties). The final SOC penalty $\phi_{SOC}(x)$ in (3.6) is only applied if the final SOC is below this target. For final SOC above the target, the only cost is the fuel spent charging the battery. With smaller numbers of engine events, the controller has less freedom to turn the engine on and charge the battery. In effect, the controllers become more conservative and maintain higher SOC in general to avoid either additional engine starts or a final SOC that is too low. This characteristic behavior appears in other cycles as shown in Chapter V.

4.4 Detailed Performance

4.4.1 Results

Several controllers are studied in greater detail on the FTP cycle, which generally yields more interesting behavior than the NEDC. The performance of the baseline controller is

Table 4.1: Selected SPSDP controller performance

Controller Description	Corr. F.E.	EE	GE	SOC_{final}	Raw F.E.
Baseline Controller	1.000	88	86	0.509	0.995
SPSDP #1-Best FE	1.184	86	96	0.500	1.184
SPSDP #2-Similar EE/GE	1.183	88	97	0.500	1.183
SPSDP #3-Min. EE/GE.	1.007	28	0	0.560	0.976

compared to 3 SPSDP controllers in Table 4.1, all running the FTP cycle. The SPSDP controllers are designed using FTP statistics and are selected from those shown in Figures 4.2a-4.2e. SPSDP #1 is the controller with the best corrected fuel economy without regard to drivability. The peak of the fuel economy surface (Figure 4.2a) is very close to the baseline controller operating point in terms of drivability. SPSDP #2 has the closest drivability metrics to the baseline controller, and is closely related to SPSDP #1. SPSDP #3 is selected by finding a controller with similar fuel economy to the baseline controller and a minimal number of drivability events.

Time histories of the baseline and SPSDP #1 controllers are presented for the first 500 seconds of the FTP cycle, which is also known as “Bag 1,” in Figure 4.3. The transmission gear is not shown if the engine is off or the clutch disengaged. The engine torque/speed operating points for these two controllers on the full FTP75 cycle are shown in Figure 4.4.

While it is not easy to pinpoint the performance differences between the controllers, some summary metrics are shown for the baseline and SPSDP #1 controllers in Table 4.2.

The Forward Wheel Energy is the integral of all motoring (output) wheel power, the Total Engine Mechanical Output Energy is the total energy delivered at the engine output shaft, Engine Brake Specific Fuel Consumption (BSFC) (g/kWh) is the total fuel consumed divided by the total engine output energy, and Braking Energy is the energy dissipated by the friction brakes.

For the electrical propulsion system, the Electro-Mechanical Charge Energy is the total mechanical energy absorbed by the electric machines, and the Electro-Mechanical Discharge Energy is the forward mechanical energy provided by the electric machines. The losses are the difference between the two, and represent all losses in the electrical system including

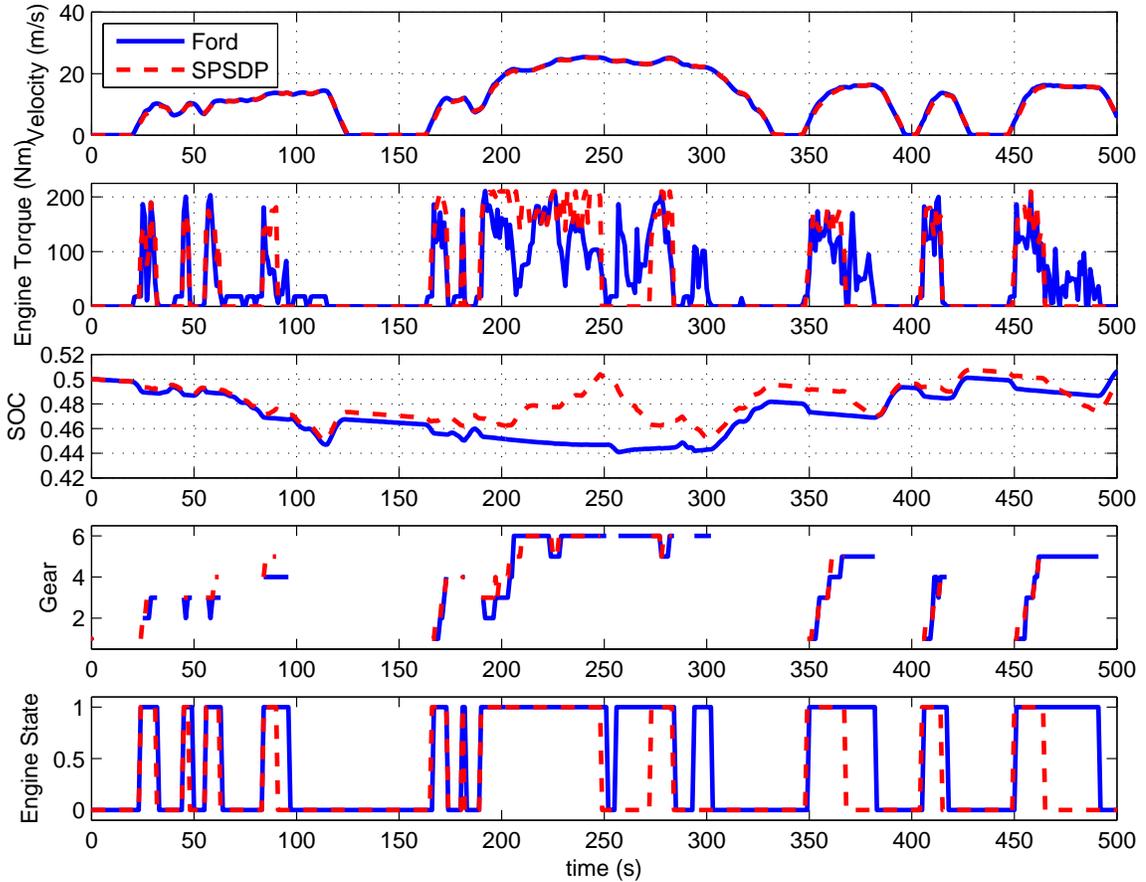


Figure 4.3: Time traces of selected simulation parameters. The baseline controller is shown as a solid (blue) line, and one particular SPSDP controller that yields the best overall fuel economy is shown as a dashed (red) line; this is SPSDP #1 in Table 4.1. The transmission gear is shown only when the engine is on and the clutch is engaged.

accessory loads. The Round-Trip Electrical Efficiency is the Discharge Energy scaled by the Charge Energy.

For each metric, the Net Change is shown as the difference between the baseline value and the SPSDP value. The Percent Change is the Net Change scaled by the baseline value. The final column only applies for energy metrics and is the Net Change scaled by the Forward Wheel Energy, a measure of the total energy required for the cycle.

4.4.2 Discussion

In Table 4.1, SPSDP #3 has zero gear events, which arises because of the way gear events are defined. A gear event is counted only when the transmission shifts with the

Table 4.2: Selected SPSDP controller performance

	Baseline	SPSDP #1	Net Change	Percent Change	$\frac{NetChange}{TotalForwardWheelEnergy}$
Forward (Motoring) Wheel Energy (kJ)	8736	8295	-441	-5.05%	-5.05%
Total Engine Mechanical Output Energy (kJ)	12298	-894	11404	-7.27%	-10.2%
Braking Energy	673	0	-673	-100%	-5.47
Electro-Mechanical Charge Energy (kJ)	5275	6873	1598	30.3%	18.3%
Electro-Mechanical Discharge Energy (kJ)	2310	3565	1255	54.3%	14.4%
Electro-Mechanical Losses (kJ)	2965	3308	343	11.5%	3.93%
Round-Trip Electrical Efficiency (%)	45.0	51.1	6.1	-	-
Engine BSFC (g/kwH)	266.9	238.6	-28.3	-10.6%	-

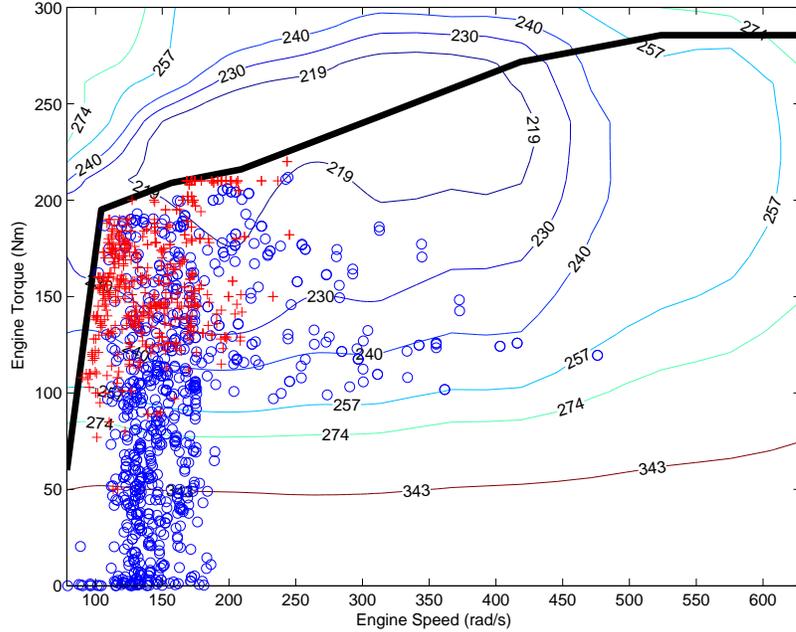


Figure 4.4: Engine Torque-Speed operating points on the FTP cycle. The solid black line represents an operational restriction, which both the SPSDP and baseline controllers respect. Baseline controller operating points are shown as circles (blue) and SPSDP controller operating points are shown as plus signs (red). The SPSDP controller is SPSDP #1 in Table 4.1 and also shown in Figure 4.3. The isoclines show constant brake specific fuel consumption in g/kWh, an inverse measure of efficiency.

clutch engaged. No penalty is incurred if the clutch disengages, the transmission shifts, and then reengages. With sufficiently high penalties, the algorithm will always disengage the clutch or shut down the engine when a shift is required. The chosen definition works well for most reasonable controllers, but can generate unexpected behavior in the extreme cases. This is an excellent example of a practical consideration: the algorithm will automatically minimize cost based on the *defined* cost function. If one wanted to eliminate this behavior, the designer has several options. Gear Events could be redefined to count a gear event when the clutch disengages, which would generate different behavior. Alternatively, one could add an additional event definition that counts clutch engage/disengage events.

A similar situation occurs when the Engine Events penalty is very high. The clutch cannot slip at low speeds, so the only option is to shut down the engine, or disengage the clutch to enter series mode. Series mode is generally not used, but with sufficiently high penalties on engine activity, the controller will always enter series mode at low speeds to

avoid engine shutdown. This yields a cycle with zero engine events, other than the initial start and final stop.

Figures 4.3 and 4.4 and Table 4.2 lend some insight into the performance differences between the SPSDP and baseline controller. Table 4.2 shows the SPSDP controller is more efficient in its use of the diesel engine. The engine operates largely in a “bang-bang” fashion, either at a high efficiency operating point or completely off, and yields a lower average BSFC as shown in Figure 4.4. The high-torque operating points are also visible in Figure 4.3. This allows the engine to remain off for slightly longer periods of time (Figure 4.3). Note that in general, the engine start/stop activity is almost identical with the exception of a few cases. The electric machines are used more extensively by the SPSDP controller, which allows more efficient ICE utilization and more efficient overall electrical propulsion.

Some of the fuel economy difference between the SPSDP and baseline controllers can be attributed to subtle differences in the driver algorithm that generates commands. The baseline controller uses a PID driver with feedforward that updates quickly at 10ms. The SPSDP controller as formulated here has a discrete 1s update time and the driver torque command is only updated at that rate. Using a proportional controller with feedforward allows the automated driver to accurately drive the cycle at the slow update rate. However, this slower update effectively decreases the variance in the driver torque demand when compared to the baseline controller. This explains the difference in the total positive wheel energy. The update rate explains about 3-5% of the fuel economy difference between the SPSDP and baseline, as confirmed by other simulations with both algorithms using the same update rate. For hardware testing, a slightly different implementation is used which allows a faster update based on driver demand (see Section VIII).

CHAPTER V

Real World Driving

5.1 Introduction

SPSDP has the potential to be used directly in production vehicles with minimal manual tuning. To test controller performance and robustness under realistic conditions, about 500,000 simulations are conducted using recorded data from several thousand real-world drive cycles. With these data, controller performance can be evaluated and optimized with respect to various classes of driver behavior [22, 23, 36, 41, 50] in addition to government certification cycles (Chapter IV and [71–73]). These simulations are analyzed to determine not just mean performance, but standard deviations and worst-case performance. This real-world robustness testing addresses a common customer complaint that the fuel economy shown on the “window sticker” does not match the vehicle performance obtained in practice [24, 37, 69, 86, 93]. To provide a realistic benchmark, SPSPDP energy management controllers are compared to an industrially-designed “baseline” controller.

As part of this real-world evaluation, controllers are constructed using statistics from multiple sets of design cycles, including both real-world and government test cycles. They are then evaluated on multiple sets of simulation cycles of various types. This allows evaluation of controller robustness to different drive cycles and the effects of the statistics assumed in the controller design.

The chapter is organized as follows: After a statistical comparison of real-world and government test drive cycles, Section 5.2 examines how various controllers (each designed for a given drive cycle) perform on real-world drive cycles. Section 5.3 then establishes the

robustness of controller performance to different drive cycles, including the dependence of engine events (a drivability metric) and fuel economy. Note that Chapter VII develops the reduced (simplified) set of drivability metrics used here and validates their use by comparison to a complex set of drivability metrics.

5.2 Robustness to Real-World Driving

5.2.1 Motivation

Controller performance is often reported on standard test cycles (FTP, NEDC, US06) for comparison and relevance to government certification. If real-world fuel economy is lower than on government test cycles, either the controllers are tuned primarily for the test cycles, or real-world driving is fundamentally less fuel-efficient than the test cycles. The real-world data used here is aimed at evaluating controller robustness and performance in the “off-cycle” real world.

In this chapter, we address practical considerations required to actually use SPSDP [73, 74] in industrial development or production. Controllers are compared to a baseline industrial controller on real-world and government test cycles to evaluate performance, robustness, sensitivity, fuel economy vs. drivability tradeoffs, and real-world vs. test cycle performance.

5.2.2 Drive Cycle Data

The drive cycle data used here was collected by the University of Michigan Transportation Research Institute (UMTRI) [51]. The “source” data set supplied to us contains 2500 trips made by 87 drivers. Very short trips (less than 3 minutes or 0.5 km) are ignored. We randomly selected two sets of 100 drive cycles from the UMTRI data. They are called “Ensemble 1” and “Ensemble 2.”

To gain some insight into the nature of the drive cycles, we briefly study the characteristics of their distributions. The cumulative distribution functions (CDF) of trip distance for the source data and both subsets are shown in Figure 5.1a. The statistics for the two Ensembles are a reasonable match for the source data set. Each Ensemble represents about

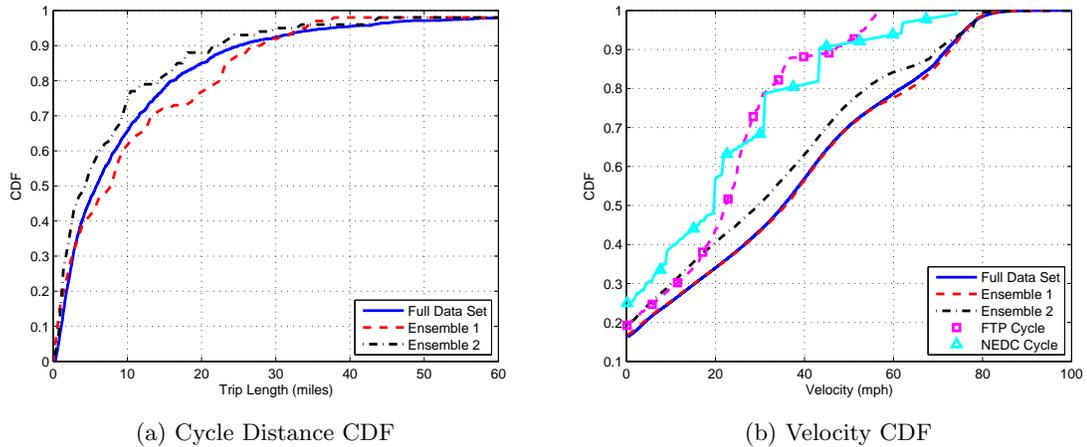


Figure 5.1: Statistics of Real-World Driving. Figure 5.1a is the Cumulative Distribution Function of trip distance for the source data and two subsets. Mean distances for the sets are: Full Data Set - 11.7 mi., *Ensemble 1* - 12.7 mi., *Ensemble 2* - 9.9 mi. Figure 5.1b is the Cumulative Distribution Function of vehicle velocity for the source data, two subsets, and two government test cycles.

1000 miles of driving, or 3 tanks of gas.

The CDFs of vehicle speed for Ensembles 1 and 2 are depicted in Figure 5.1b, using vehicle velocity on a second-by-second basis. Two standard government test cycles are also shown, the Federal Test Procedure 75 (FTP) and the New European Drive Cycle (NEDC). This yields a total of five total in the figure: the Source Data, *Ensemble 1*, *Ensemble 2*, FTP, and NEDC.

There are three interesting things to notice in Figure 5.1b. The first is that the government test cycles are fundamentally different from the real-world data. The real-world cycles contain substantially higher velocities in general. The second detail is the step-like nature of the NEDC cycle, which arises because it is contrived. The cycle is composed of perfect ramps to constant speeds and is specified by hand. Lastly, *Ensemble 2* has lower velocities than *Ensemble 1*, a difference that will be reflected in the fuel economy results presented in Section 5.2.4.

5.2.3 Simulation Procedure

To study the effectiveness of the SPSDP controller design methodology, controllers are first designed based on statistics from particular drive cycles (Chapter III). Large numbers

of controllers are then simulated on a set of real-world driving data using two different methods:

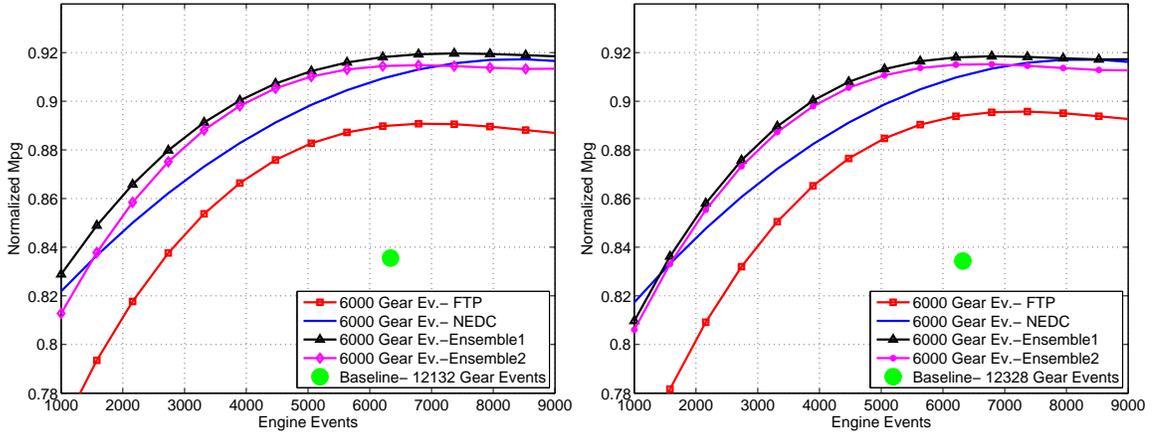
- **Concatenated Cycle** - The Ensemble of 100 cycles is assumed to represent one vehicle's driving history, about 1000 miles. The starting SOC of the first trip is 0.5, and the starting SOC for each subsequent trip is the ending SOC of the previous trip.
- **Individual Cycles** - Each cycle is studied individually, and the starting SOC for each trip is 0.5.

Each method has its advantages, so both are used here. For the **Concatenated Cycle**, fuel economy for the Ensemble is simply the total fuel consumption divided by the total distance. Drivability events are summed over all trips. The engine is off at the start and end of each trip. The total fuel consumption is corrected based on final SOC, but the battery energy is negligible compared to the fuel consumption on such a long trip. The SOC correction (4.1) impacts fuel economy less than 0.1%. This is one major advantage of this method: there is no concern about SOC correction yielding false fuel economy results. This method is arguably more representative of typical driving as the SOC varies at the start of each trip. However, the Ensemble trips are selected from a group of drivers, so they may better represent a vehicle shared by a household rather than a single driver.

For the **Individual Cycles**, total fuel use is individually corrected for each cycle based on SOC. The fuel economy for the Ensemble is computed as the total of corrected fuel consumption divided by total driving distance. This yields a weighted average over all the trips. The drivability events are the sum of all the trips. This method is very useful for generating statistics. Each controller now has 100 sample points which can be used to calculate the mean, standard deviation, and $10^{th}/90^{th}$ percentile bands for fuel economy and final SOC. This allows deeper understanding than simply studying the weighted average of fuel consumption.

5.2.4 Results

The SPSDP controllers generate significantly better performance than the baseline controller on real-world cycles and are reasonably robust to variations in driving patterns and



(a) Fuel Economy for Ensemble 1-Concatenated Cycle (b) Fuel Economy for Ensemble 1-Individual Cycles

Figure 5.2: Fuel Economy and Drivability Metrics on Ensemble 1 when simulated as Concatenated and Individual Cycles for the baseline controller and SPSDP controller *families* designed with statistics from FTP, NEDC, *Ensemble 1*, and *Ensemble 2*. Fuel Economy, Gear Events, and Engine Events are cumulative for the whole cycle set, representing approximately 1000 miles. Results are normalized to the baseline controller on FTP, as in Section IV.

the statistics used to design the controller.

The Ensemble is first treated as a Concatenated Cycle representing a single vehicle (Sec. 5.2.3). Fuel economy and drivability results are shown in Figure 5.2a. The figure shows five controller sets evaluated on Ensemble 1: the baseline controller and 4 SPSDP controller families designed on statistics from FTP, NEDC, Ensemble 1 and Ensemble 2. Recall that a controller family is developed by fixing the driver statistics and sweeping the drivability penalties to generate a tradeoff curve (see Section IV). All fuel economy numbers are normalized to the baseline controller running FTP, which has fuel economy 1 just as in Section IV (Figures 4.2 and 5.4a).

The cycles in each Ensemble are also treated as Individual Cycles, where the SOC always starts at 0.5 (Section 5.2.3). The weighted average of fuel economy compared to Engine Events is shown in Figure 5.2b for Ensemble 1. The superiority of the SPSDP controllers with respect to the baseline controller is maintained under real-world driving conditions as shown in Figure 5.2. For the same drive quality index, the fuel economy is approximately 10% higher.

Using the individual simulation method, each controller now has 100 data points rep-

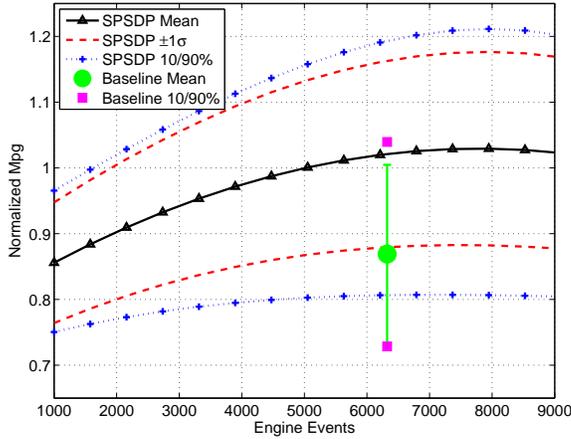
representing performance on individual drive cycles. The family of controllers designed with Ensemble 1, shown as a black curve in Figure 5.2b, is selected for more detailed study. The (non-weighted) mean, standard deviation, and 10th/90th percentile bands are calculated for corrected fuel economy and final SOC for each controller running both Ensemble 1 and Ensemble 2. A response surface is fitted to these data as shown in Figure 5.3. The same statistics are calculated for the baseline controller. The distributions are not Gaussian and differ with the number of engine events as well as the simulated Ensemble.

5.2.5 Discussion

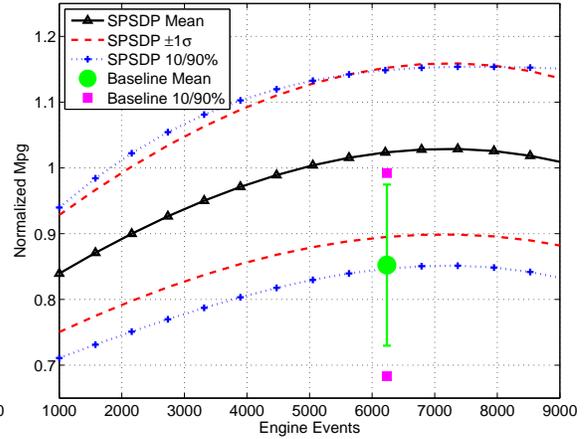
The performance of the SPSDP controllers on the concatenated Ensembles confirms that their superiority is legitimate and not an artifact of SOC correction; the energy in the SOC errors is minimal on such a long cycle. Figure 5.2 demonstrates that simulating the cycles individually and correcting for SOC yields results similar to the concatenated version. Specifically, the SOC correction method is valid and the exact starting SOC for each cycle is not crucial for overall performance. This is quite useful because the simulation of individual cycles is easily parallelized and avoids the difficulties of simulating such a long (16 hour) cycle.

Figure 5.2 shows that the SPSDP controllers and the baseline controller yield lower fuel economy in the real-world than on government test cycles, implying the difference is fundamental to the cycle itself and not a result of controller tuning. On FTP the baseline controller has fuel economy 1 and the SPSDP controllers attain fuel economies of 1.18. This confirms a known weakness with the government test cycles: they are not representative of real-world driving. Real-world driving requires 15-20% more fuel consumption than the test cycles. Recall the differences among cycles illustrated in Figure 5.1b. This causes a significant mismatch between the “window sticker” fuel economy from certification testing and the fuel economy consumers would measure in practice. Recent changes to the government testing procedure in the US have attempted to address this problem and bring the certification fuel economy predictions closer to real-world practice [1,2]. Additional results on the government test cycles are presented in Section 5.3.1 (Figure 5.4).

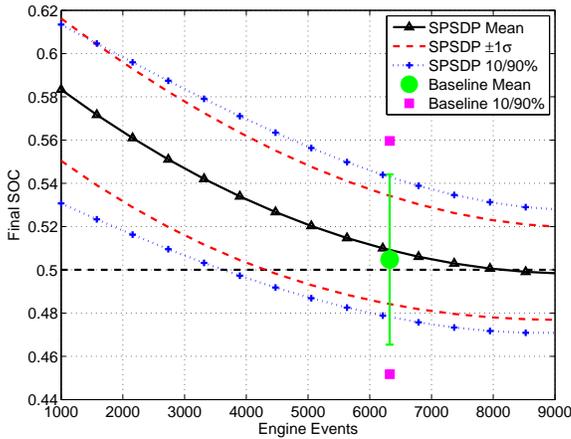
The fuel economy curves in Figures 5.2 and 5.3 show a distinct “knee”, where the



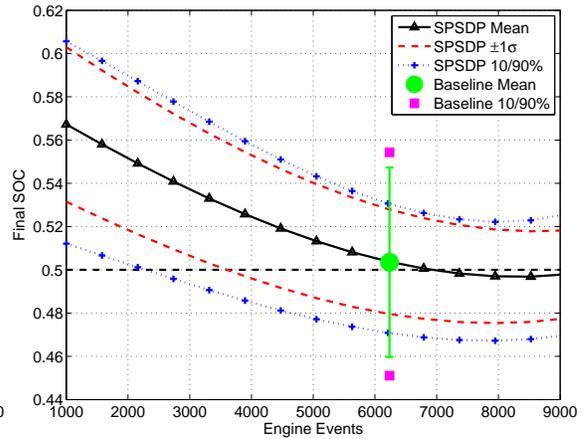
(a) Fuel economy statistics for Ensemble 1



(b) Fuel economy statistics for Ensemble 2



(c) Final SOC statistics for Ensemble 1



(d) Final SOC statistics for Ensemble 2

Figure 5.3: Statistical fuel economy. The SPSDP controller *family* designed on *Ensemble 1* is simulated on *Ensemble 1* and *Ensemble 2*, and each cycle is corrected for SOC. The mean, standard deviation, and 10th and 90th percentile are calculated. The mean, standard deviation (error bars), and 10th and 90th percentile are also calculated for the baseline controller.

tradeoff between fuel economy and engine activity becomes acute. In both figures, the number of engine events may be reduced from 9,000 to 5,000 with no significant loss of fuel economy. This illustrates the power of having this optimal Pareto curve available. Not only is the tradeoff quantified, but in some cases it may be possible to reduce engine activity without any sacrifice in fuel economy. Without such a curve, a fuel-optimal controller may be designed with 9,000 engine events without the designers knowing the same fuel economy can be attained with roughly half the engine activity.

The statistical analysis of the individual cycles in Figure 5.3 shows very consistent performance. For the SPSDP controllers, performance one standard deviation below the mean still exceeds the mean of the baseline controller.

Battery charge maintenance for real-world driving is very consistent as shown in Figures 5.3c and 5.3d; there is little variation across the different drive cycles. The nominal target SOC is 0.5, which is nearly achieved in the high fuel economy operating region (above 5000 engine events). Both figures show a trend towards higher SOC with fewer engine events. As engine events become more costly, the controller tends to operate at a higher SOC, presumably to stay in electric mode and avoid engine starts whenever possible. The SPSDP controllers also maintain tighter control of final SOC compared to the baseline controller, as evidenced by the standard deviation and percentile bands.

The results in Chapters IV and V represent more than 4000 controllers and 500,000 simulated cycles, requiring roughly 10.5 CPU-years of computing time on a cluster of desktop-class machines at the University of Michigan Center for Advanced Computing. Most problems can be solved with fewer controllers and cycles: the sensitivity, robustness, and real-world driving studies are not required in every application.

5.3 Robustness to Variable Driving Styles and Sensitivity to Design Cycle Statistics

Drive cycles are used here for two purposes: to design a controller, and then to simulate its performance. As described in Chapter III, SPSDP uses a Markov chain to model driver behavior, with the states and transition probabilities extracted from one or more design

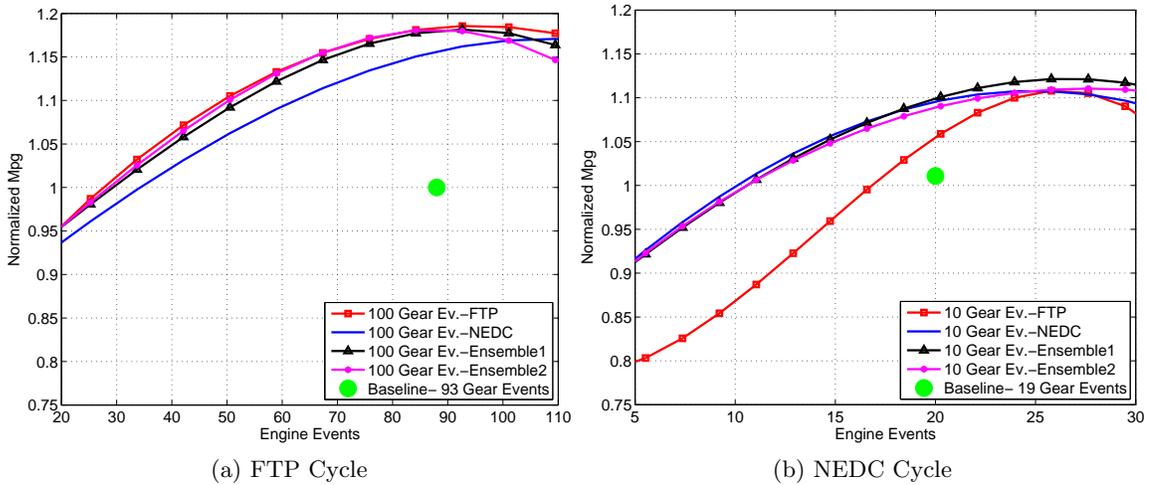


Figure 5.4: Fuel Economy and Drivability Metrics on the FTP and NEDC Cycle for 5 controller options. Controller *families* are designed with statistics from FTP, NEDC, *Ensemble 1*, and *Ensemble 2*. All fuel economy figures are normalized to the baseline controller performance on FTP, shown as a large green dot in Figure 5.4a. The controllers are the same as those shown in Figure 5.2.

cycles provided during the controller development process. The statistical driver model clearly affects the final controller and the obtained closed-loop behavior, which opens a question about the relationship between the (possibly different) cycles used to design and simulate the controller. Here we investigate the sensitivity of closed-loop behavior to the assumed driver statistics.

In Section 5.2.4, four different *families* of controllers were designed using statistics from FTP, NEDC, Ensemble 1 and Ensemble 2. These controllers were then simulated on the two Ensemble sets. In this section, these four families of controllers are also simulated on the FTP and NEDC cycles.

5.3.1 Results

In general, the SPSDP method is relatively insensitive to the choice of cycles used to design the controllers, although the controller that performs best on a given cycle is generally the controller designed for that cycle, as would be expected. The controllers from Section 5.2.4 are shown running the FTP and NEDC cycles in Figure 5.4, using the same markers as in Figure 5.2.

5.3.2 Discussion

This cross-combination of design and simulation cycles allows study of a controller's performance on both the design and arbitrary cycles. Excluding the NEDC, design cycle statistics cause less than 3.5% difference between controllers across all other cycles. The fuel economy difference when using Ensemble 1 or 2 statistics is small, which indicates that the sample sizes are large enough to reasonably represent typical driving.

This idea is reinforced in the surprising robustness of controllers when dealing with the FTP cycle. The enormous 20% fuel economy difference between FTP (Figure 5.4a) and Ensemble 1 (Figure 5.2) seems to indicate that the statistics of the two cycles are not representative of each other. Nevertheless, controllers designed on the Ensemble cycles do very well on the FTP cycle, and the FTP-based controllers sacrifice 3% performance on the Ensemble cycles. While the FTP and Ensemble cycles are very different, the statistics from either cycle are adequate to generate controllers that perform reasonably well on both cycles.

The NEDC cycle is somewhat of an outlier in this study. NEDC is used to address the European market in addition to US regulations. Although the fuel economy of the NEDC-based controllers is reasonable on the FTP and Ensemble cycles, the NEDC is generally avoided as a test cycle. As seen in Figure 5.1b, the NEDC cycle is vastly different from both real-world driving and the FTP. It is periodic and clearly specified by hand, which is not well suited to a Markov chain based expectation.

Figures 5.2 and 5.4 allow a comparison of performance on real-world and government test cycles, as mentioned in Section 5.2.5. The controllers are the same in each figure and use consistent markers, they are just running different cycles. The fuel economy normalization is the same in all figures. The baseline controller drops from a fuel economy of 1 Mpg on FTP (Figure 5.4a) to 0.835 Mpg on Ensemble 1 (Figure 5.2a). The best SPSDP controllers achieve 1.18 Mpg on FTP, but only 0.92 Mpg on Ensemble 1.

CHAPTER VI

Fundamental Limitations of an Industrial Controller

6.1 Introduction

While many design methods have been proposed to develop energy management controllers for HEVs, it is difficult to compare them. Most algorithms, even those from academia based on formal optimization methods, have at least some parameters that must be selected by the designer. This is even more true of industrial controllers, which tend to use extensive hand-tuning and in-vehicle calibration in order to trade off what are often very subjective driving quality attributes.

Any performance comparison of controller design methods is only as good as the engineers that tune the various algorithms, and thus the comparison always suffers from the refrain that “algorithm X could have been tuned better.” Comparisons are even more difficult when the designer is forced to compromise among competing performance attributes, such as the tradeoff between fuel economy and engine start-stop activity, which is investigated here. The relative value of one characteristic compared to another is highly subjective, meaning comparisons among different operating points necessitate a qualitative value judgement.

The astute reader may have already posed this question. Extensive results are shown comparing SPSDP controllers to the industrial baseline, but how can one know if the baseline controller is tuned correctly?

The goal of this chapter is to study the performance of the industrial controller previously introduced as a baseline, which was developed by Ford Motor Company for this prototype

vehicle. The baseline industrial controller is first evaluated to determine its Pareto tradeoff curve: the upper limit of possible performance in terms of fuel economy versus engine activity. This is accomplished by sweeping the parameters of the controller over a wide range of values, thereby generating a point cloud of possible fuel economy and engine start-stop operating points of the prototype HEV under this controller. The frontier of this point cloud is the Pareto tradeoff curve of maximum attainable performance; the HEV with this controller can be operated anywhere on that line, but not above it. A comparison is then made to the Pareto tradeoff curves of the SPSDP controllers.

The method of SPSDP generates causal controllers that are directly implementable in a real-time setting [94, 95]. In particular, the resulting controllers do not use *future* drive cycle information. This is in contrast to Deterministic Dynamic Programming [60], which is cycle dependent (it relies on *a priori* knowledge of the entire drive cycle). The causal nature of a SPSDP controller allows a fair comparison to the baseline controller.

The Pareto frontier for the SPSDP controllers is shown to lie above the Pareto frontier of the baseline controller, meaning that the SPSDP controllers achieve superior fuel economy performance for a given level of engine on-off activity, for *any* possible tuning of the baseline controller. This limitation is fundamental to the structure of the baseline algorithm: no amount of parameter tuning or calibration can generate performance that equals that of the SPSDP algorithm. An advantage of the SPSDP algorithm is that it directly generates controllers that lie on the tradeoff curve, and does so without requiring hand calibration. The role of expert judgement is then to decide where on the Pareto tradeoff curve to operate the vehicle for a given market.

Traditional vehicle software is produced through a process of continuous improvement. While each year's model has better vehicle control software than the last, in practice, control design engineers are hesitant to change the basic structure of the energy management algorithms, both because of their inherent complexity as well as their complex relationship to other vehicle systems. Instead, if a better controller is developed, its actions are analyzed in detail, and the existing software is tuned to mimic the actions of the new controller. This chapter emphasizes that such an approach will not always work. While it is possible that a given controller architecture may be tuned for a particular vehicle to achieve the maximum

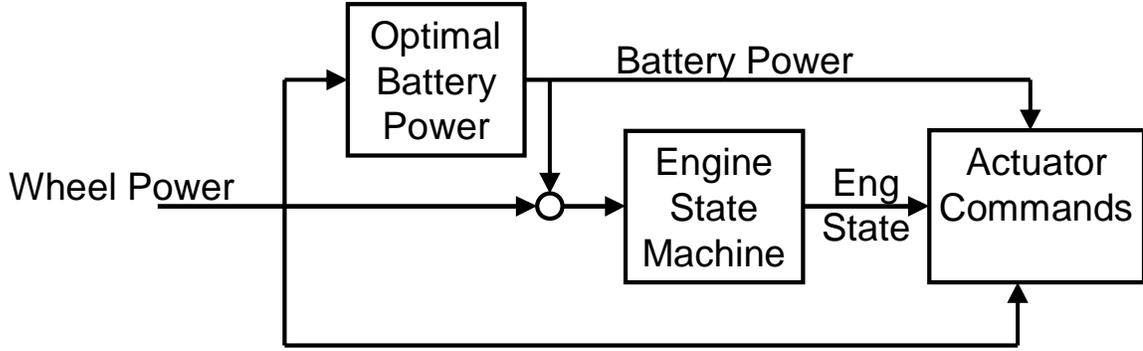


Figure 6.1: High Level Baseline Controller Architecture.

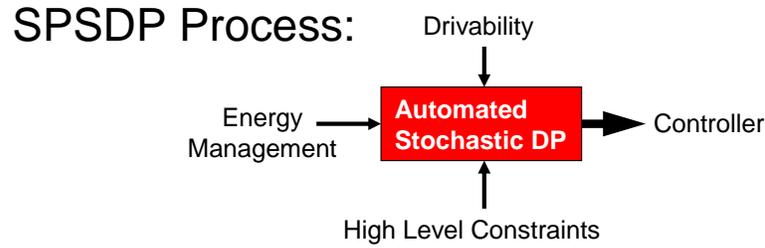
performance, there are no guarantees. When manually tuning an algorithm, engineers may be unaware they are finding the maximum attainable performance for a *particular controller architecture* rather than an optimal causal controller. A more general benchmark that avoids specifying a controller architecture is required to correctly gauge performance. SPSDP is one such method for generating causal controllers.

6.2 Baseline Industrial Controller

6.2.1 Architecture

The “baseline” prototype energy management controller studied here is very complex. Its key features are contained in three modules, as depicted in Figure 6.1. Driver power demand is determined from pedal position. One module determines the optimal battery power flow and adds it to the driver demand to determine the *Total Power*. A second module determines the optimal engine state based on the *Total Power* using a state machine with hysteresis. A third rule-based module then determines individual actuator commands (e.g., power from the engine and the two electric machines) based on the *Total Power* and the desired engine state. The transmission gear is selected independently by the transmission.

This architecture is fundamentally different from the SPSDP algorithm discussed in this dissertation, as illustrated in Figure 6.2. The SPSDP controller is a single step optimization, while the baseline algorithm follows the common two step (or sequential) design procedure. Structurally, the two-stage algorithm is similar to the “local” optimization discussed in Appendix B and [70, 94, 95].



Common Development Process:

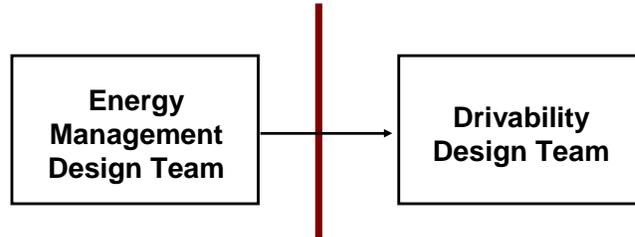


Figure 6.2: Two possible design processes. The SPSPD process conducts the optimization in one step, but may be more complex. The “Common” two stage optimization is often used in industry. Its simpler structure may be easier to tune, but may sacrifice performance.

6.2.2 Performance Capability

The flexibility of rule-based controllers with many calibration parameters is tempered by the fact that there is no *a priori* guarantee of optimality. The goal here is to determine the Pareto tradeoff curve of the baseline architecture and compare it to a controller based upon stochastic optimal control. Performance is evaluated in terms of fuel economy and engine activity, but other important tradeoffs could also be considered.

The Pareto tradeoff curve of the baseline controller is estimated numerically by sweeping a set of tuning parameters over a wide range and evaluating performance on the vehicle simulation model. The primary tuning “parameters” are actually five scalar functions, two in the “Optimal Battery Power” module and three functions of vehicle speed in the “Engine State Machine” module. These are the same functions that a calibrator would adjust in the vehicle. This is obviously a very large space to search, especially for an engineer tuning the algorithm by hand. One advantage of the baseline architecture is that engine behavior and battery charge maintenance features are largely confined to their respective blocks with minimal interaction, simplifying the tuning process considerably: parameters can be tuned to adjust one behavior without affecting the other.

6.2.3 Parameter Sweep Procedure

First, the three functions in the Engine State Machine were varied, using both small perturbations from the nominal tuning and a brute force sweep of a larger function space, while the functions in the Optimal Battery Power module were held fixed. This process generated approximately 100,000 possible controllers, which were each simulated on the FTP cycle. The fuel economy numbers were recorded and corrected¹ based on final battery state of charge (SOC), and the number of *Engine Events* was recorded as discussed in Section 3.2.

200 of the best tunings were selected for further study². For each fixed tuning of the Engine State Machine, the two functions in the Optimal Battery Power module were then varied. This yielded a primary set of 180,000 controllers, each of which was evaluated on the FTP cycle.

To evaluate robustness to real-world driving, 210 of these controllers were selected and simulated on a set of real-world drive cycles obtained from the University of Michigan Transportation Research Institute (UMTRI) [51]; see Chapter V and [72–74]. Fuel economy and number of *Engine Events* were recorded.

6.3 Academic Benchmark

In order to evaluate the degree of optimality of the baseline industrial algorithm, it is compared to controllers designed using Shortes Path Stochastic Dynamic Programming (SPSDP). The SPSDP controllers operate under the same information conditions as the baseline controller (in particular, no future drive cycle information is used).

With an optimization based approach such as SPSDP, the designer specifies the *cost* to be minimized and the algorithm produces the required closed-loop dynamics. This is different from manual design and tuning where the designer typically varies parameters with the goal of achieving a closed-loop behavior that minimizes cost, with no optimality guarantees.

¹See Section 6.4.

²Controllers were selected from both the frontier and the interior of the point cloud using the same method discussed in Section 6.5.1

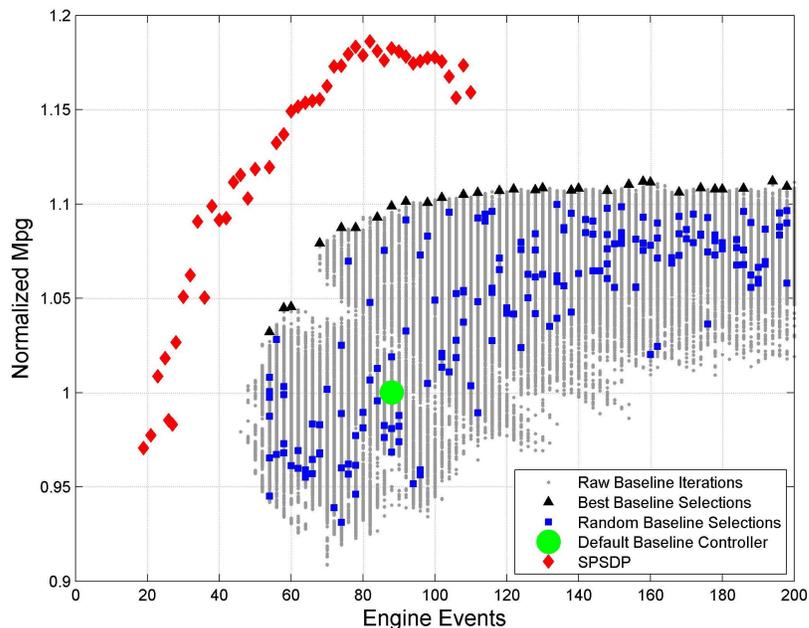


Figure 6.3: Best Case performance of the baseline controller running FTP compared to SPSDP controllers. The gray dots are all possible baseline controllers, the black triangles are the “best” available baseline controllers, and the blue squares are selected randomly from the other reasonable controllers. SPSDP controllers are shown for comparison as red diamonds. Fuel economy is normalized to the default baseline controller running the FTP cycle, shown as a large green circle. All markers represent the same controllers in Figures 6.3-6.5.

On one hand, the SPSDP algorithm is systematic, optimal, and yields implementable controllers. On the other hand, it suffers from off-line computational complexity. In effect, a control designer is trading off the need to decide on an *a priori* controller architecture and tuning values against the burden of setting up the algorithm and doing the off-line computations.

6.4 Simulation Procedure

The baseline and SPSDP controllers are evaluated on the vehicle simulation model discussed in Section 3.1.4. These simulations are all causal, so the final battery SOC is not guaranteed to exactly match the starting SOC. This could yield false fuel economy results, so all fuel economy results are corrected based on the final SOC of the drive cycle. This is done by estimating the additional fuel required to charge the battery to its initial SOC, or the potential fuel savings shown by a final SOC that is higher than the starting level. This

correction is applied according to

$$\Delta Fuel = C_{Batt} \Delta SOC \frac{BSFC_{min}}{\eta_{max}^{Regen}} \quad (6.1)$$

where $\Delta Fuel$ is the adjustment to the fuel used, C_{Batt} is the battery capacity, ΔSOC is the difference between the starting and ending SOC, $BSFC_{min}$ is the best Brake Specific Fuel Consumption for the engine, and η_{max}^{Regen} is the best charging efficiency of the electric system.

Controllers are initially evaluated on the US government’s FTP test cycle, in which case there is only one simulation per controller. To study robustness to drive cycle variations, controllers are also evaluated on a set of real-world driving data collected by the University of Michigan Transportation Research Institute (UMTRI) [51]. 100 cycles are randomly selected from these data to generate an *ensemble* of cycles. Procedurally, this is conducted as follows:

1. Each controller is simulated on each of the 100 cycles in the ensemble using the vehicle simulation model.
2. The results for the ensemble of 100 cycles are compiled to generate average or cumulative performance for that particular controller.

In the end result, each controller has average performance metrics (fuel economy and drivability) representing cumulative performance on the ensemble of cycles. Note that studying 100 controllers on 100 cycles each means 10,000 simulations.

6.5 Results

6.5.1 FTP Cycle

As discussed in Section 6.2.3, a primary set of 180,000 tunings of the industrial energy management controller were first simulated on the FTP cycle. The fuel economy numbers were corrected based on final SOC per (6.1) and the number of *Engine Events* was recorded

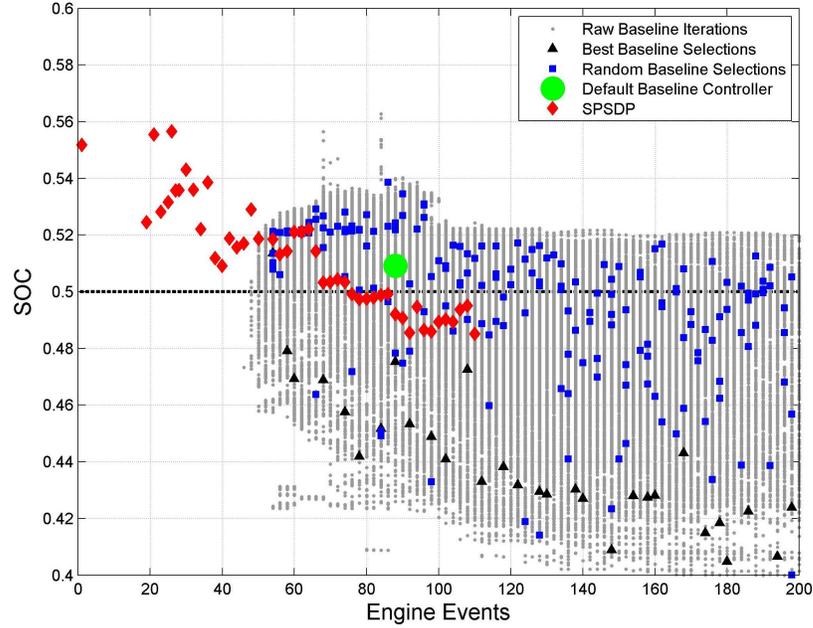


Figure 6.4: Final Battery SOC of the baseline and SPSDP controllers running FTP. All cycles start at SOC=0.5. All markers represent the same controllers in Figures 6.3-6.5.

as discussed in Section 3.2. These data pairs³ are presented in Figure 6.3 as small gray dots. The default tuning of the baseline controller (provided by Ford) is shown as a large green circle. The controllers designed using SPSDP are shown in these figures as red diamonds. The fuel economies are normalized to the nominal baseline controller running the FTP cycle (i.e., green circle has fuel economy of 1.0).

Varying the engine state machine parameters does change the battery SOC behavior, but the controllers are still reasonably charge-sustaining, as shown in Figure 6.4. Final battery SOC is used to correct the cycle fuel economy for all results shown. This correction generally only changes the results 1-2% and does not alter the relative comparison. The SPSDP controllers generally achieve better performance than the baseline, both in uncorrected fuel economy and in final SOC.

6.5.2 Real-World Drive Cycles

Fuel economy on government test cycles differs from that of real-world driving. Real-world performance is studied by evaluating controllers on an ensemble of 100 drive cycles

³Many of the parameter values yielded unreasonable controllers with poor fuel economy and large numbers of engine events. The corresponding data pairs are outside the bounds of the figure.

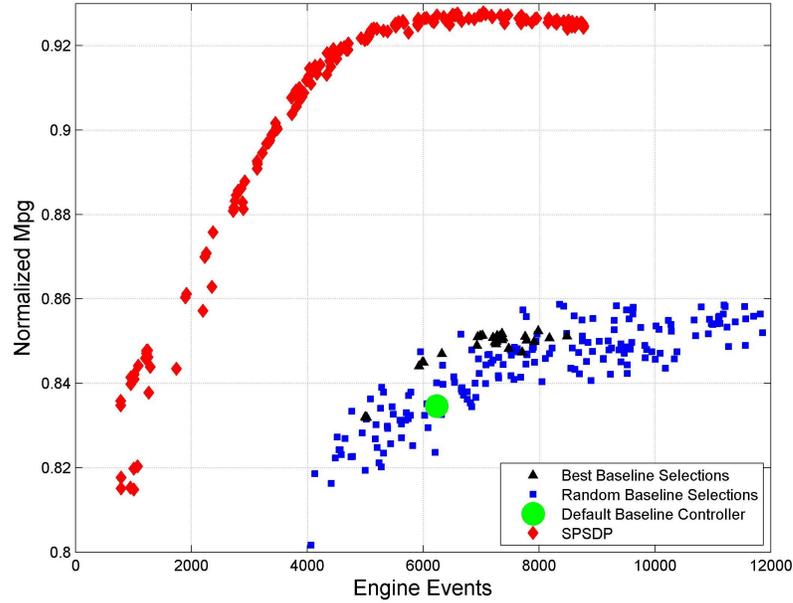


Figure 6.5: Best Case performance of the baseline controller and SPSDP running the ensemble of 100 cycles. Fuel economy is normalized to the default baseline controller running the FTP cycle. All markers represent the same controllers in Figures 6.3-6.5.

as discussed in Section 6.4. It is impractical to evaluate 180,000 controllers on 100 cycles each, so the majority of the brute-force search was conducted on the FTP cycle, and a subset of 210 controllers were selected for further evaluation. 30 of those were selected from the Pareto frontier of Figure 6.3 to represent the “best” available, and 180 were selected randomly from the cloud of reasonable controllers⁴. These controllers are termed the “best” and “random” controllers and are shown as black triangles and blue squares respectively in Figures 6.3-6.5.

The cumulative performance of each controller on the 100 ensemble cycles is shown in Figure 6.5. The results are normalized to the baseline controller running the FTP cycle⁵, so both the SPSDP and baseline controller yield lower fuel economy in the real-world (0.86 & 0.93) than on government test cycles (1.1 & 1.18). The SPSDP controllers in Figures 6.3-6.5 are the same for all three figures, and are designed using statistics from the ensemble of 100 real-world drive cycles; see Chapter V and [72–74].

⁴Note that the baseline tuning provided by Ford is not on the Pareto frontier of the FTP cycle in Figure 6.3.

⁵The green circle in Figure 6.3.

6.5.3 Discussion

The fundamental tradeoff between vehicle fuel economy and the amount of engine activity is clearly visible as a Pareto tradeoff curve in the results. The SPSDP controllers achieve equal or better performance than the baseline under all conditions, as would be expected with the method’s optimality guarantees. One major benefit of SPSDP is clearly visible: controllers are always on the frontier of attainable performance without iterative searching. Varying the cost function merely moves the operating point along the Pareto curve of maximum performance.

The fundamental limitations of the baseline controller likely arise from three sources. As illustrated in Figure 6.1, the optimal battery charging power and engine state are determined sequentially and not simultaneously. Other major automakers use similar two-stage architectures that likely exhibit these limitations. A second possible source is that the engine state machine is inherently rule-based as a function of total power demand. While the total power demand is strongly correlated with optimal trajectories, a rule-based strategy is likely suboptimal in comparison to a more general function of the other state variables in addition to total power demand. A third possible source of limitations is the “actuator commands” block, which is rule-based and not a pure optimization.

While it is not easy to pinpoint why the SPSDP controllers perform better, in general, they are more aggressive and efficient in their use of the diesel engine and the electric machines. The engine operates largely in a “bang-bang” fashion, either at a high efficiency operating point or completely off. The electric machines are generally used closer to their maximum efficiency, or near maximum power to enable high engine power outputs when little road load power is required.

These general operating principles may seem intuitive, but they underline one of the major benefits of SPSDP: it automatically generates the optimal controller without a designer specifying control actions. Even given these principles, a designer would be hard-pressed to formulate control laws that generate optimal performance. In addition, these principles do not necessarily hold in general and may change with different vehicles. Guessing the wrong “rules of thumb” in the design phase can impose performance limits, as demonstrated here.

6.6 Conclusions

In this chapter, the Pareto tradeoff curve of fuel economy versus engine on-off activity was estimated for an industrial energy management algorithm. This was accomplished by numerically sweeping a large set of functions that a calibrator would use to tune the industrial algorithm. The Pareto curve was then computed for a causal controller designed using Shortest Path Stochastic Dynamic Programming (SPSDP), and it was found to lie strictly above the Pareto curve of the industrial controller. There is no possible tuning or calibration of the industrial algorithm that can match the performance of the SPSDP controller. This implies fundamental structural limitations of the baseline algorithm. These limitations likely arise for three reasons: the battery power flows and engine start-stops are determined sequentially and not simultaneously; the engine on-off control is constrained to be a function of total power demand; and some actuator selection is rule-based.

The SPSDP-based controllers do not exhibit similar limitations. In particular, a SPSDP-based controller uses full-state feedback, and thus power flows, engine on-off events and gear number can be general functions of vehicle speed, battery SOC, gear number, engine state and total power demand. While it is very possible that a simpler feedback structure may exist, that is, one that depends on fewer variables and hence is more easily calibrated in the field, the search for such a feedback is a separate problem. As part of that search, the control designer has to decide how much degradation in performance is acceptable for ease of tunability, maintenance, or other considerations.

The work presented here underlines the point that making an *a priori* choice of feedback structure or vehicle behavior can induce significant structural barriers to obtaining optimal vehicle performance, barriers that cannot be overcome at later stages in the design process, no matter how well the nominal controller is tuned. One way to avoid making these choices at an early stage is to adopt a more sophisticated controller design procedure in the prototyping phase, one that automatically searches over all possible state feedback controllers. One such method is SPSDP.

CHAPTER VII

Development and Validation of Drivability Metrics

7.1 Formulation

As mentioned in Chapter III, two significant characteristics that are noticeable to the driver are the basic behaviors of the transmission and engine. There are many qualitative characteristics that describe powertrain behavior [96]. While metrics exist to quantify a large variety of behaviors, overall value judgements are largely qualitative. There are no high-level rules or metrics that exactly quantify the overall drivability performance or describe the relative importance of various behaviors. An important contribution of this dissertation is the translation of these value judgements into quantitative metrics that can be used in the optimization formulation. The first step is to describe and quantify engine and transmission behaviors using performance metrics, and the second step is to reduce the complexity of these metrics so that they may be easily used in optimization.

A primary concern in drivetrain activity is the the frequency and timing of events, like gear shifts and engine start/stop. Two categories of metrics are used, the mean time between events and the number of short-duration events; the latter are especially bothersome to drivers. A short duration event occurs when the dwell time in a particular state is less than some specified value; the metric is the number of these occurrences. This type of metric is denoted “Dwell time less than X seconds,” where X is the cutoff criteria. These “mean” and “short duration” categories of metrics applied to the engine and transmission generate 7 distinct metrics, termed the “complex” metrics. These 7 metrics represent a detailed description of vehicle behavior and are shown in the top table in Figure 7.1. Many other

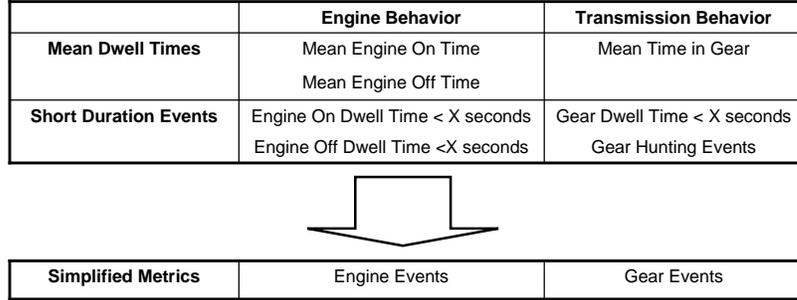


Figure 7.1: Drivability Metric Reduction. The seven complex engine and transmission metrics are divided into two categories, mean dwell times and short-duration dwell times. These metrics are then reduced to the two simplified metrics.

metrics could obviously be used, but these are an important subset of the possibilities.

For the transmission, a particularly annoying short-duration event is “hunting,” rapid shifting back and forth between the same two gears. We define a gear “hunting” event as a sequential upshift-downshift or downshift-upshift that occurs faster than some cutoff time X . The metric is the number of occurrences of a hunting event. This type of shifting often occurs in normal driving, but only becomes bothersome when the shifts are closely spaced. Shifting that is frequent or perceived to be unnecessary is often termed shift “busyness,” and is reflected in both mean dwell time and short duration metric categories.

The most bothersome engine events are those of very short duration, and to some extent drivers ignore the long-horizon (10-20s) engine behavior as long as there are no short-duration events.

Although it is theoretically possible to incorporate these metrics in the optimization formulation, the computation burden of the required additional states makes the problem intractable. Another disadvantage is the large parameter space of penalty weights for the various metrics. Even if all these metrics were directly implemented and a controller computed, the control designer is left with a very complex design process.

Therefore, we choose to simplify these complex metrics into something that can be easily used. Ideally, the information contained in the seven metrics listed above could be distilled into a smaller number of simple metrics. Indeed, the behaviors measured by these metrics are well correlated with the two simple metrics proposed below, which allows effective control of complex behavior with a simple implementation.

7.2 Simplified Drivability Metrics

The seven complex metrics are reduced to two baseline metrics to quantify behavior for a particular trip. These are the two simplified metrics previously described. The first is *Gear Events*, the total number of shift events on a given trip. The second metric is *Engine Events*, the total number of engine start and stop events on a trip. This reduction is depicted in Figure 7.1.

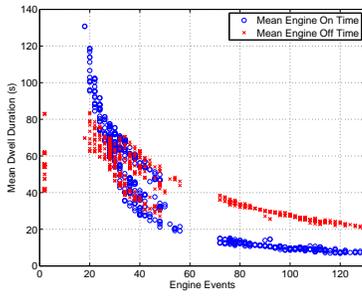
By definition, engine starts and stops are each counted as an event. Each shift is counted as a gear event, regardless of the change in gear number. A $1^{st} - 2^{nd}$ shift is the same as a $1^{st} - 3^{rd}$ shift. Engaging or disengaging the clutch is not counted as a gear event, regardless of the gear before or after the event.

The utility of these simplified drivability metrics is validated by studying how they relate to the more detailed metrics. By finding simple metrics that are well correlated with the complex metrics, one can incorporate the simple metrics into the full SPSDP algorithm to maintain some control over complex behavior while keeping the problem feasible.

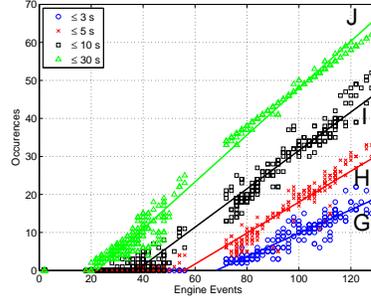
7.3 Drivability on Government Test Cycles

Simulations on the FTP cycle show strong correlations between the simple and complex metrics. For a *family* of controllers running the FTP cycle, drivability metrics are recorded for both the simple and complex metrics. The engine activity is studied in Figures 7.2a-7.2c. The metric *Engine Events* is shown on the horizontal axis for all 3 figures. The mean engine on and off times are shown in Figure 7.2a. Short duration engine events are studied next, the engine on and off Dwell Time less than X seconds are shown in Figures 7.2b and 7.2c respectively for various cutoff criteria X.

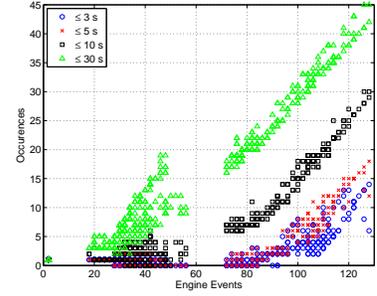
The gear shifting activity of the vehicle is studied in Figures 7.2d-7.2f. The metric *Gear Events* is shown on the horizontal axis for all 3 figures. Figure 7.2d shows the mean dwell time in gear. This metric studies the time spent in one particular gear without shifting or disengaging the clutch. Short durations between shifts or clutch disengagements are indicated by gear dwell times less than some variable cutoff criteria as shown in Figure 7.2e. Transmission gear “hunting” is shown in Figure 7.2f for varying time length criteria.



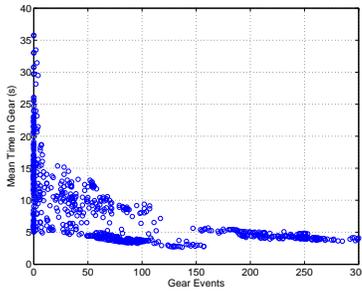
(a) Mean engine on and off durations compared to *Engine Events*.



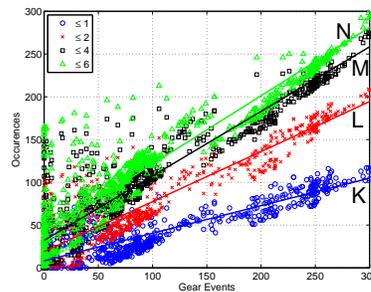
(b) Engine on durations less than some number of seconds compared to *Engine Events*. Data are shown for cutoffs of 3, 5, 10, and 30 seconds.



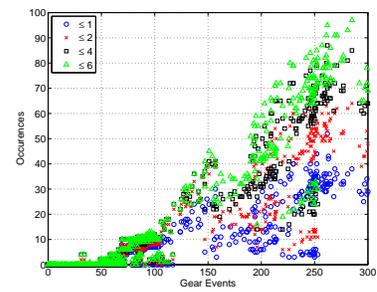
(c) Engine off durations less than some number of seconds. Data are shown for cutoffs of 3, 5, 10, and 30 seconds compared to *Engine Events*.



(d) Mean dwell time in gear between shifts or clutch disengagements compared to *Gear Events*.



(e) Gear dwell times less than some number of seconds between shifts or clutch disengagements compared to *Gear Events*. Data are shown for cutoffs of 1, 2, 4, and 6 seconds.



(f) Gear hunting events that occur within some number of seconds. Data are shown for cutoffs of 1, 2, 4, and 6 seconds in comparison to *Gear Events*.

Label	Metric	Straight-line fit	σ	Range
G	On ≤ 3	$\hat{y} = 0.31x - 20.9$	0.90	0-25
H	On ≤ 5	$\hat{y} = 0.41x - 22.7$	1.37	0-35
I	On ≤ 10	$\hat{y} = 0.50x - 18.9$	1.63	0-55
J	On ≤ 30	$\hat{y} = 0.62x - 13.7$	2.01	0-65

(g) Curve fitting for engine on durations from subfigure 7.2b. σ is the standard deviation between the data and the fit.

Label	Metric	Straight-line fit	σ	Range
K	In Gear ≤ 1	$\hat{y} = 0.32x + 8.25$	13.8	0-120
L	In Gear ≤ 2	$\hat{y} = 0.58x + 19.0$	20.5	0-210
M	In Gear ≤ 4	$\hat{y} = 0.75x + 32.1$	24.1	0-280
N	In Gear ≤ 6	$\hat{y} = 0.80x + 41.5$	26.4	0-300

(h) Curve fitting for gear dwell times from subfigure 7.2e. σ is the standard deviation between the data and the fit.

Figure 7.2: Comparison of simple and complex drivability metrics on the FTP cycle. Complex engine activity metrics are compared to the simplified *Engine Events* metric in subfigures 7.2a-7.2c. Three gear activity metrics are compared to the simplified *Gear Events* metric in subfigures 7.2d-7.2f. Subfigures 7.2b and 7.2e have straight-line fits to the data, and the parameters are shown in Tables 7.2g and 7.2h.

These figures (Figures 7.2a-7.2f) show that the complex metrics are approximately monotone functions of the simple metrics. To highlight this property, the data in Figures 7.2c and 7.2e are shown with a straight line least squares fit to the data. Each cutoff time criteria is treated separately and each line matches the color shown for the underlying data.

The relationships between the metrics in these figures are very nearly piecewise linear. They generally are zero up to a certain value on the horizontal axis, then follow a straight line. These data are fit with functions of the form

$$\hat{y} = \max(0, mx + b)$$

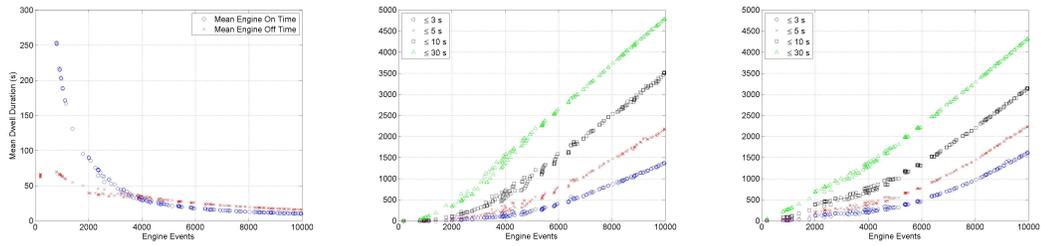
where m and b are the slope and intercept of the best straight-line fit to the nonzero data..

7.4 Drivability on Ensemble Cycles

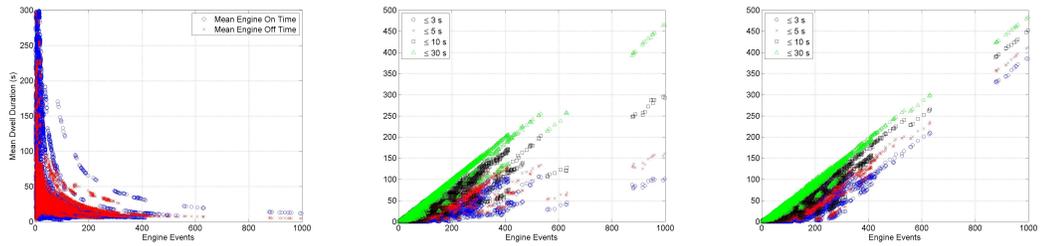
The FTP results demonstrate that simplified drivability metrics of total engine events and total gear events can yield good vehicle behavior when evaluated in terms of more detailed metrics such as engine on-off dwell times and gear hunting. This is also true for real-world driving on the Ensemble cycles. This particular vehicle is relatively insensitive to gear activity, so here we focus on engine activity.

As discussed in Section 5.2.3, there are two ways to study the Ensemble cycles. The first is to treat each Ensemble as a single trip, about 1000 mi. The second method is to treat each cycle individually as a unique data point. The drivability metrics are studied using these two methods. The three detailed engine activity metrics (Figures 7.2a-7.2c) are first studied for the Concatenated Ensemble 1 in the top row of Figure 7.3 (Figures 7.3a-7.3c). Each data point represents one controller running a single simulation, the Concatenated Ensemble 1. Each data point represents the same cycle, and variation is tied to controller tuning (i.e., choice of penalties in the cost function).

The second method, treating each cycle individually, is shown in the bottom row of Figures 7.3d-7.3f. Each data point represents a controller running one of the 100 cycles in Ensemble 1. In this case, variation between data points arises from different controller tunings *and* different cycles.



(a) Engine On-Off seconds-Concatenated Ens. 1 (b) Engine On TBD seconds-Concatenated Ens. 1 (c) Engine Off TBD seconds-Concatenated Ens. 1



(d) Engine On-Off seconds-Individual Ens. 1 (e) Engine On TBD seconds-Individual Ens. 1 (f) Engine Off TBD seconds-Individual Ens. 1

Figure 7.3: Comparison of simple and detailed drivability metrics for concatenated and individual Ensemble 1 cycles. Detailed engine activity metrics are compared to the simplified *Engine Events* metric in Figures 7.3a-7.3c for the concatenated Ensemble, where the 100 cycles are treated as a single trip. Each marker represents the same drive cycle, the concatenated Ensemble 1. The same metrics are studied for the individual Ensemble 1 in Figures 7.3d-7.3f, where each cycle is simulated individually. Each marker represents one controller running one of 100 cycles, so the markers no longer represent the same cycle.

7.5 Discussion

The strong correlations between the simple and complex metrics allow the drivability attributes to be easily quantified. A designer can be confident that the simple metrics are directly related to the complex versions, and that prescribing behavior with respect to the simple metrics will yield the desired results. The problem can be greatly simplified in that the designer is not required to specifically track and control each complex behavior of interest. The main algorithm will generate tunable performance that meets the criteria in a general sense.

For example, optimizing for fuel economy often leads to gear hunting behavior near a shift point. As the total number of shifts is penalized and reduced, hunting behavior is usually eliminated first as these frequent shifts do not significantly improve fuel economy.

Similarly, for fuel-optimal operation, the engine on/off decision can become very sensitive to driver demand, causing many short-duration engine events even when the driver applies a nearly constant pedal input. Reducing the total number of engine events tends to eliminate these short-duration events (Figure 7.2b) and make the engine state less sensitive to the driver demand.

The detailed drivability metrics are still related to the simplified metrics in an approximately monotone fashion on real-world driving (Figure 7.3). For the concatenated Ensembles (Figures 7.3a-7.3c, the correlation is even more clear than on FTP. The nearly straight-line fits for short duration engine on/off events demonstrate that the simple and detailed metrics are related by a nearly constant ratio, which was unexpected.

Perhaps most surprising are the results for the individual Ensemble cycles (Figures 7.3d-7.3f). Each plot now depicts data points representing 100 different cycles with different controllers. The drivability behavior is no longer directly related to the controller but confounded by different cycles, yet the straight-line trends are still clearly visible. Even for the real-world drive cycles, it seems that the short duration engine event metrics are still related to the simple metrics by a nearly constant ratio.

These methods are quite effective and produce controllers that are very well behaved. This method is very useful for adjusting vehicle behavior in practice. Suppose a controller is

selected based on simulation. Once implemented, the engine state is very sensitive to pedal, as in the previous example. The designer simply selects a new (still optimal) controller with higher penalties and thus engine events, and the engine state becomes less sensitive to the pedal. The designer has an easy way to control behavior that is simple to tune and still optimal. Contrast this to a rule-based controller, where the designer changes the rules that determine when the engine turns on. It is very difficult to directly tune those rules and maintain the best fuel economy.

CHAPTER VIII

Hardware Testing

8.1 Introduction

Hybrid vehicle energy management controllers have received extensive attention in the past decade. Many different algorithms have been proposed using multiple vehicle configurations and models. There are relatively few results in the literature that test such controllers in real hardware [13, 35, 42, 43, 61, 77, 78], or that address the many practical considerations required to do so.

This chapter describes the implementation and hardware testing of energy management controllers based on stochastic dynamic programming. These controllers are designed to address both fuel economy and constraints on powertrain activity. Previous Chapters demonstrated good performance on real-world driving using a detailed, realistic vehicle model. The controllers were then implemented in a real vehicle provided through a partnership with Ford Motor Company. Implementing the controllers in practice required a major effort equal in scope to all previous development work.

The overall process used to implement the real-time controller was based on a progressive increase in model and controller complexity. The three main issues were the real-time controller implementation, integration with the existing vehicle controller, and dealing with real actuators which may be unpredictable. The various test environments used in each step are described in Section 8.3.2 and illustrated in Figure 8.1. Several of the issues were unexpected and would never have been considered if only simulation models were used.

It is difficult to guess how many of the results in this dissertation may make their way

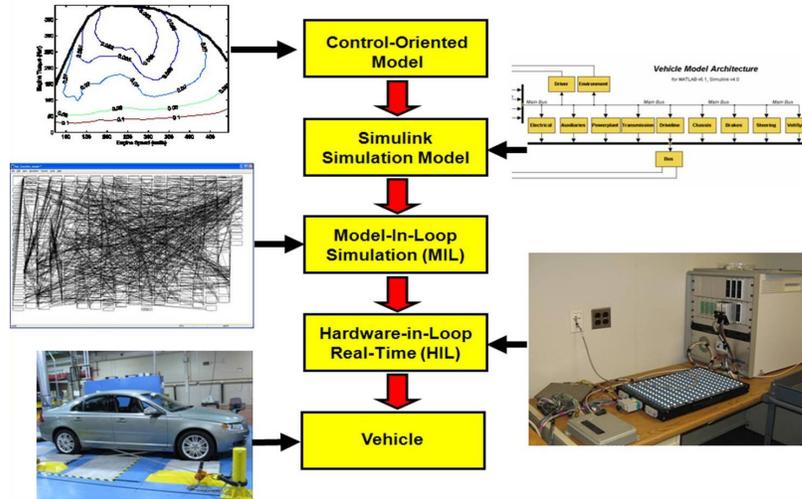


Figure 8.1: The increasing complexity of controller testing in this work.

Table 8.1: Fuel Economy Summary for malfunctioning hardware

Controller	Cycle	Normalized Corrected Fuel Economy (MPG)	Improvement
Baseline	FTP72	1.000	
SDP	FTP72	1.035	3.5%
Baseline	NEDC	0.969	
SDP	NEDC	1.024	5.7%

into production vehicles, but industry will not adopt advanced techniques without sufficient justification.

The vehicle in question is a prototype and comes with the associated benefits and drawbacks. Unfortunately, the vehicle was plagued with hardware problems, including a transmission replacement. This left the vehicle drivable, but made it difficult to drive complete test cycles.

Partway through testing, the engine controller detected a fault and limited engine torque to 150 Nm; 300 Nm is full scale. This issue was not repaired, so most of the fuel economy data reflects this limitation. A summary of the data that was collected is shown in Table 8.1. These data may be unreliable for several other reasons as well. More detail can be found in Section 8.5.

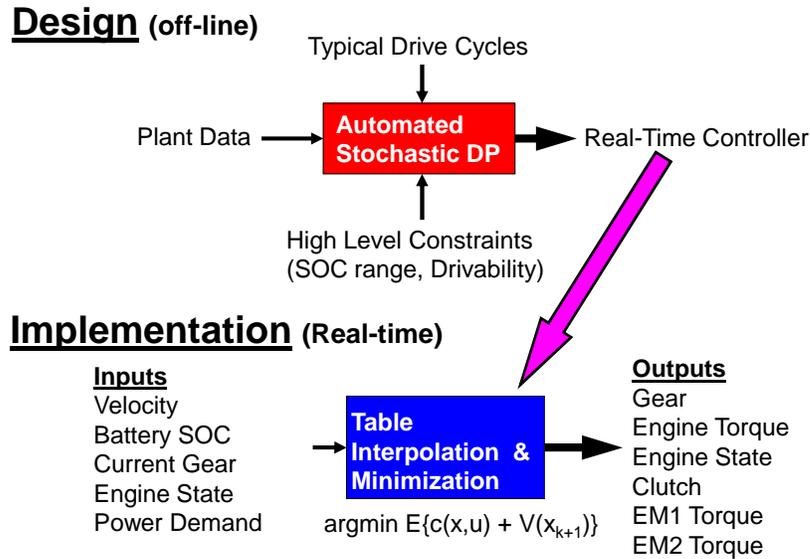


Figure 8.2: The overall development process

8.2 Controller Design

So far, an energy management controller design method was developed based on Stochastic Dynamic Programming. These controllers are causal and were extensively tested in simulation using realistic models and real world driving data. This chapter focuses on the hardware implementation and testing of those controllers. The basic details of the controller design process are summarized here for convenience.

One major benefit of this method is that the control designer does not actually specify control actions. The SPSDP algorithm uses a vehicle model, desired vehicle behavior, or constraints, and a statistical driver model to automatically generate a causal controller.

The overall controller development process is divided into two steps. The optimal control policy and its value function are calculated off-line in a computationally intensive process. The controller is then implemented in real-time, which requires much less computation. This process is shown in Figure 8.2.

The off-line calculation of the optimal control policy (Figure 8.2) yields the policy itself $u^*(x)$ and the value function $V^*(x)$ and is the most difficult part of the process. The optimal control $u^*(x)$ is determined by minimizing the sum of the current cost $c(x, u)$ and

the expected future cost $V^*(f(x, u, w))$,

$$u^*(x) = \underset{u \in U}{\operatorname{argmin}} E_w[c(x, u) + V^*(f(x, u, w))]. \quad (8.1)$$

Once the approximated value function is found, finding the control (8.1) is relatively simple. The SPSDP controllers in this Chapter were previously implemented on-line in simulation (Chapters IV and V), but not in real-time and not in hardware. This implementation is the major topic of this Chapter.

The SPSDP controller design makes three major assumptions about vehicle behavior.

Operational Assumptions:

1. The clutch in the transmission allows the diesel engine to be decoupled from the wheels.
2. There is no ability to slip the clutch for starts.
3. There are no traction control restrictions on the amount of torque that can be applied to the wheels.

8.3 Implementation

8.3.1 Overall Process

The implementation was developed to allow an easy transition between off-line controller development and its on-line implementation.

The implementation descriptions in this section roughly follow the actual development process. This reflects the increasing complexity and detail of both the controller and the test models (Figure 8.1) as the controller approaches a hardware test.

Many of the challenges and solutions discussed here are applicable to a broad array of energy management control techniques, not just SPSDP. Although the underlying algorithms differ, many of these optimal control techniques share common characteristics including a relatively slow update rate (1s), requirements to interact with other vehicle control systems, and the need to handle unreliable actuation.

8.3.2 Vehicle Models

The road from an academic algorithm to industrial hardware requires dealing with many challenges that are not typically addressed in a dissertation. This process involved five major steps using different testing conditions and models, as illustrated in Figure 8.1. The different test environments are also listed below.

Testing Environments:

1. **Control-Oriented Model**-Simple, table-based model used for controller design.
2. **Simulink-based model**-Ford's in-house model used to simulate fuel economy. Complex, MATLAB/Simulink based model with a large number of parameters and states [6].
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 (MIL)**-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.

Controllers were initially designed and tested on a simple control-oriented model. They were then extensively evaluated using a complex Simulink-based model provided by Ford. These two steps are described in detail in Chapters III-V. After this simulation-based testing showed promising results, we started down the road to hardware by implementing the controller in Ford's real-time Vehicle Controller, which is a combination of C and autocoded Simulink. This was rather difficult because the SPSDP algorithm had to interact with all the existing vehicle controls. The real-time controller was tested in simulation using a "model in the loop" (MIL) method with a simulated vehicle. The controller was then compiled and run on the actual real-time hardware, which connected to a simulated vehicle in a "hardware in the loop" (HIL) testbed. The final step was to put the real-time hardware

in the vehicle. Each of these steps was roughly equivalent in terms of difficulty and time. This overall process allowed systematic development of the algorithm and implementation. Each step in the process provided complementary opportunities to identify failures, debug, and validate results.

8.3.3 Structural Setup

Implementing the SPSDP controller required dealing with several issues that can be safely ignored in simulation. One major problem is timing. During the design process, computing the value function for SPSDP can be challenging for fast time steps. The controllers are designed to update at roughly 1s, as are many energy management controllers in the literature. This captures the relevant dynamics of the system while ignoring fast transients. Some control actions like engine start and gear shift take roughly one second. While these controllers can follow drive cycles, generate acceptable performance, and look good in simulation, a real driver is bothered by a pedal with a 1s lag. This issue was first addressed once we decided to test in hardware and began work on the MIL.

The solution was a multi-rate implementation that commands actuators at different rates based on their capability, as shown in Figure 8.3. The engine on-off state and transmission gear are relatively slow and thus are updated at 1 Hz. The engine torque command is faster and very noticeable to the driver, so it is updated at 3 Hz to give some pedal responsiveness. Finally, the two electric machines are updated at 20 Hz to yield very fast pedal response. The reasons for this multi-rate scheme are clear, but the challenge was to implement such a scheme in a nearly optimal fashion, as discussed later.

The hierarchical structure of the on-line implementation presents some interesting choices as well. As mentioned previously, the off-line calculation of the optimal control policy (Figure 8.2) yields the policy itself $u^*(x)$ and the value function $V^*(x)$. Either may be used in the implementation. In principle, the policy itself can be stored as a pure state-feedback lookup table. However, better numerical accuracy can be obtained by storing the value function and doing additional computation on-line.

To keep the off-line problem feasible, the continuous control inputs (like engine torque) are discretized into a relatively coarse grid of about 20 possible values. The stored optimal

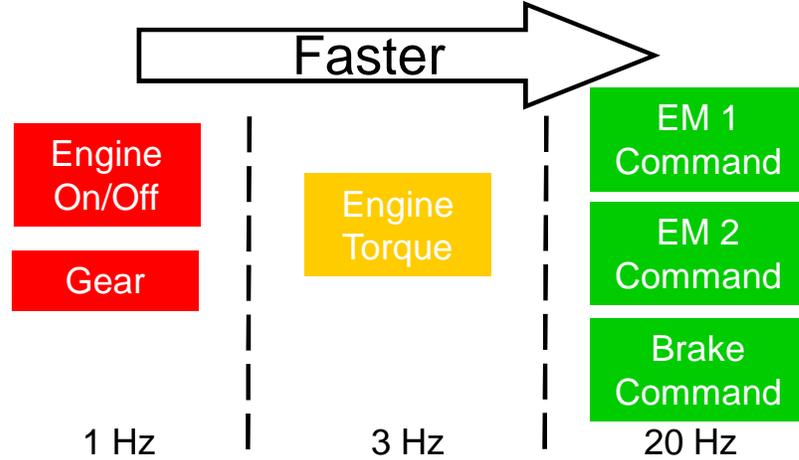


Figure 8.3: Multi-rate actuator commands

policy $u^*(x)$ will carry this coarse discretization. A better option is to store the value function $V^*(x)$ with an on-line execution of the standard dynamic programming equation (8.1).

This is a much easier problem in the on-line case because we must only find the optimal control $u(x)$ for the current state x rather than for all states. As our approximation of V^* is already known, this is a rather simple problem. Specifically, u^* can be solved much more accurately, either with a very finely discretized grid or a continuous search. In the on-line implementation, the engine torque control input is discretized into 100 possible values, yielding increments of 3 Nm. Minimizing over a finite set of values is generally faster and more reliable than minimizing a function which may not have nice properties. Simulations have shown that this on-line refinement of control inputs is important; this technique yields 2-3% better performance than simply implementing the stored coarse policy $u^*(x)$. This technique was studied early on with the Control-Oriented model and was initially proposed by [94, 95].

Given this choice of implementation, the system is set up to clearly represent the SPSDP equation (8.1) and allow for easy understanding and debugging as shown in Figure 8.4. The ability to conduct the minimization 2.5 on-line allows some flexibility to include additional features as will be discussed later. The basic structure of (Figure 8.4) is used for all controllers in the overall process (Figure 8.2), but the control-oriented model and the Simulink simulation model use controllers implemented as .m files. The algorithm is coded

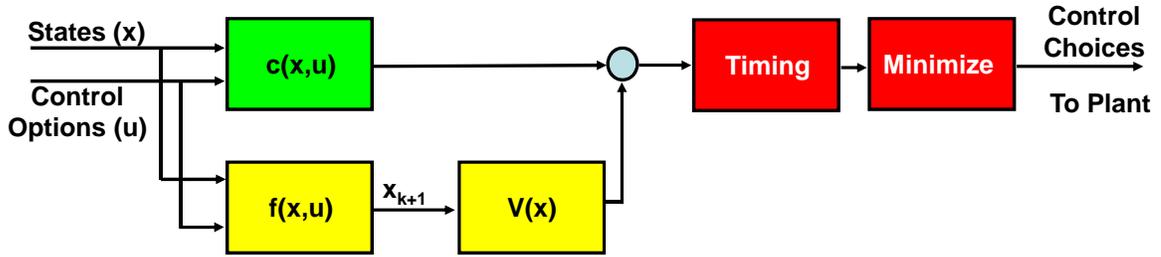


Figure 8.4: Real-time algorithm implementation in Simulink

in Simulink for the MIL and subsequent models to allow easy integration with the Ford real-time controller, automatic code generation, and simple interaction with Matlab. The HIL was used to debug the real-time aspects including table sizes, memory, and precision. The largest table was actually the system dynamics function $f(x, u)$ rather than the value function $V(x)$ itself.

For each update, the algorithm in Figure 8.4 is passed a matrix of about 700 possible control choices along with the current state. The instantaneous cost of each possible control along with the future cost $V(f(x, u))$ are similarly stored as a matrix. Selecting a cost-minimizing command is then simply a matter of picking the minimum total cost. The possible control choices are strategically stored in the matrix to make it easy to restrict the available control options based on timing, as discussed in the next section.

8.3.4 Multi-Rate Updates

Three major features drove the controller update scheme. The first is the ability to respond quickly to driver demand using the faster actuators, as previously mentioned. The second is handling unpredictable hardware, and the third is gracefully dealing with infeasible operating conditions.

8.3.4.1 Multi-Rate Commands

The first goal is to issue nearly-optimal commands at varying rates as in Figure 8.3. The engine state and gear command are updated slowly (1.2s) and the engine torque command is updated more quickly (0.4s). The control choices are evaluated by simultaneously considering a “frame” of possible controls as shown in Figure 8.5a that are stored in columns for

different gears and the rows representing possible engine torque. Gear choices that violate some constraint (like engine speed limits) are disallowed and shown in dark red. The light green columns represent valid gear choices for this speed. The series column represents the clutch disengaged, one entry in this column represents engine off. The values in the frame represent the estimated total cost of each control, so the algorithm picks the minimum. As described in Section 3.1.3, the required electric machine torques are exactly determined based on engine torque and transmission gear.

We desire to update the engine and gear commands slowly, while updating the engine torque more quickly. Figure 8.5b represents a time series of controller updates. At each update, a frame of controls is evaluated. At the initial timestep $t=0$, four possible gears are valid, and the algorithm selects one. The gear and engine state commands are only updated every 1.2s. At the intermediate updates ($t=0.4$ and $t=0.8$), engine state and gear are fixed and only the torque may vary. Other states are declared invalid (red), leaving only one column of valid torques. When the next full update occurs, the engine and gear commands are updated. It was surprisingly difficult to synchronize the multiple rates in real-time. Implementations that worked well in simulink broke down once they were autocoded; the HIL setup was quite valuable in sorting this out.

8.3.4.2 Dealing with unreliable actuation

In simulation, the system dynamics are simple, especially with a 1s update. An actuation command is issued and is assumed to occur before the next time step. Starting the engine is a typical example: engine starts reliably occur within less than one second in simulation. Engine torque production is unpredictable for that 1s, but the start does not affect the following time step. Real hardware is a different story.

The transmission gear is an important example. During the controller design and simulation process, the gear command was assumed to execute almost immediately. In reality, the shifting process can take more than a second. In addition, it was initially unclear if it would even be possible to command the transmission gear without major changes to the vehicle software. For these reasons, the SPSDP algorithm must be ready to deal with commands that are not followed. Specifically, the algorithm picks an engine state and torque

assuming that the gear can be selected. If that gear is not immediately engaged, it is unclear what the torque command should be.

The overall solution to this problem was to select the discrete actuation (engine state and transmission gear) assuming that the commands will be followed, but selecting the engine torque commands based on the *current* engine and transmission state. Discrete actuation commands are issued normally at each full update and held during the intermediate updates. The engine torque command is selected based on the *current* engine and transmission state, regardless of which engine and gear commands have been issued. The only valid engine torques are those for the current gear. The current gear is shown in green in the interim time steps $t=0.4$ and $t=0.8$ in Figure 8.5b. The full updates at 0s and 1.2s show the valid gears that may be selected, but the actual torque command is still issued based on the current gear. Figure 8.6a attempts to illustrate this concept by showing that the actual vehicle mode is used to determine valid torque commands, but the mode commands are selected assuming they will be followed. The fundamental organizing principle can be expressed as: “Select modes based on what you *want* to happen, but issue commands based on what *is* happening.”

A major problem arises when determining system state because the control model assigns discrete states to processes that are actually somewhat continuous. For example, the model assumes the engine changes discrete states (on or off) between time steps, while in reality, a controller update may occur when the engine has not completely started and is thus neither on nor off. Starting the engine and engaging the clutch takes about 1.5s on average and can require up to 3 s. The basic SPSDP formulation used here is not equipped to handle this problem and assumes the engine can only be on or off. A ready solution is available when commanding torque; the engine is considered off until ready to deliver torque, which is a binary signal. A more challenging problem occurs within the SPSDP optimization itself.

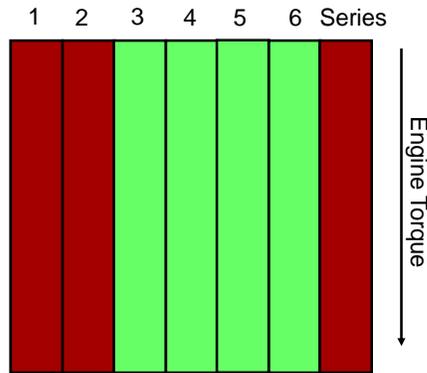
If the engine is off, at some point the algorithm will issue an engine start command. At the next update (1s later) the engine will be in the process of starting but not fully started. If the engine is considered off, the optimal decision may be to leave the engine off due to the engine start penalty and accompanying hysteresis. This would yield undesirable behavior, with frequent chatter in the engine state command. The solution is that the

SPSDP algorithm reads an engine state that matches the expected dynamics: if a command is issued to change engine state, that change is assumed to have occurred at the next time step. This yields much more consistent behavior; the algorithm tends to hold the engine on command throughout the start process.

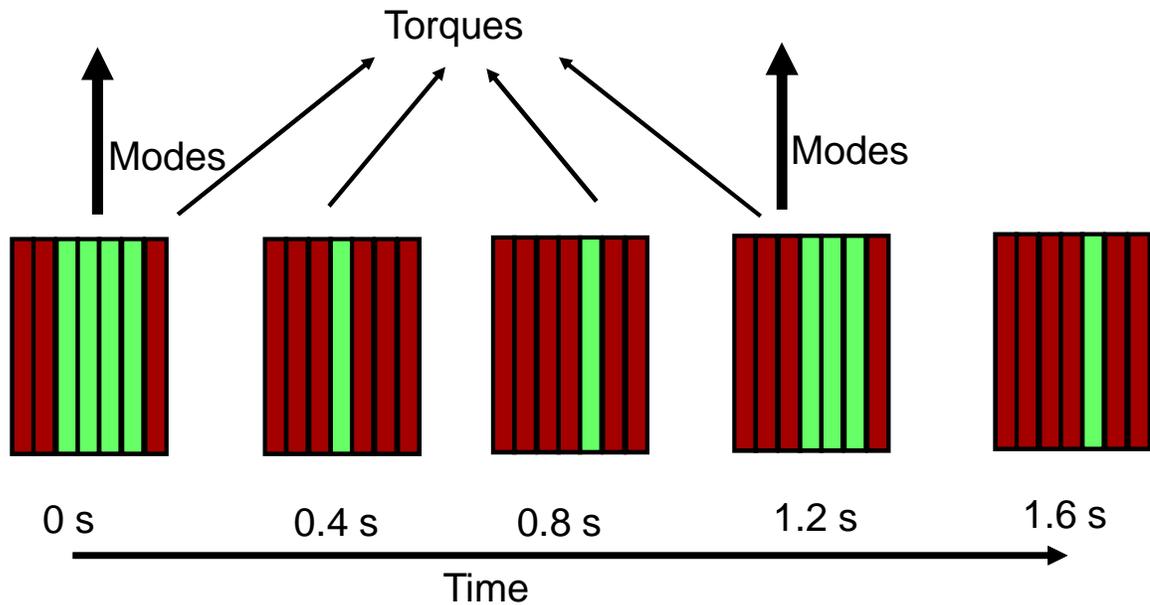
8.3.4.3 Dealing with infeasible conditions

An additional feature of the implementation is the “feasibility update” which allows controller commands to be updated immediately if required to meet driver demand or protect hardware. One benefit of the structure in Figure 8.5a is that it is very easy to determine which controls are feasible. In most cases, there is some combination of commands which meet driver demand and satisfy all constraints, but under some conditions, it may not be possible to meet driver demand with the current gear and engine state. During the intermediate updates (0.4s and 0.8s in Figure 8.5b) the engine and gear commands are fixed and no valid torque is available, although valid commands exist given the choice of any engine and gear state. When no valid commands are available at an immediate update, a full update occurs instead, regardless of the normal waiting time. This process is illustrated in Figure 8.6b. At the intermediate update $t=0.8s$, no valid torque command exists for the current gear and engine state, shown by the fully red frame. The controller instead immediately allows a full update of all commands, shown by the arrow from the frame at 1.2 s.

This “feasibility update” is rarely used but is important in two cases. The first is for a “gorilla stomp” in which the driver suddenly demands large torques which are unavailable in electric mode or higher gears. The full update occurs immediately, forcing an engine start or a downshift. The electric mode transition is especially important for driver perception as drivers would otherwise wait up to a full second before hearing an engine start. The second case is during rapid deceleration with the engine on. The clutch cannot be engaged below a certain vehicle speed or it will pull the engine speed below its minimum. Under some conditions, first gear will valid at one full update, but become invalid before the next full update due to vehicle speed changes. This feature allows the clutch to be disengaged at one of the intermediate updates once that gear becomes infeasible.

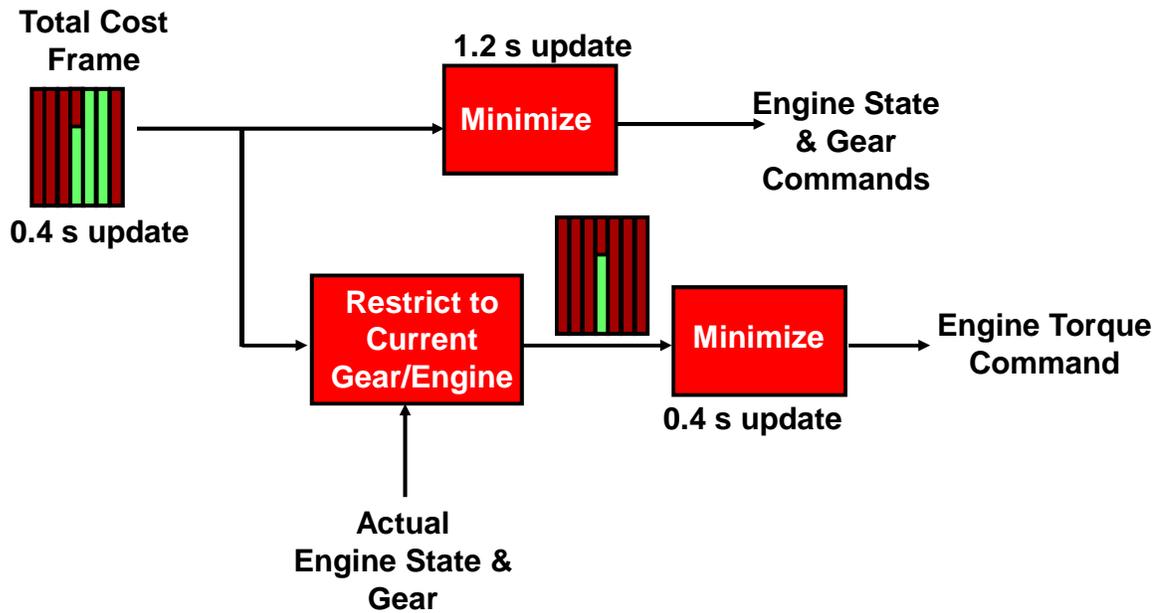


(a) A “frame” of possible control choices are represented strategically as an array. The columns represent the 6 possible transmission gears along with series mode. Possible engine torques are rows in the array. Electric mode (engine off) is represented by a zero torque point in the series mode column.

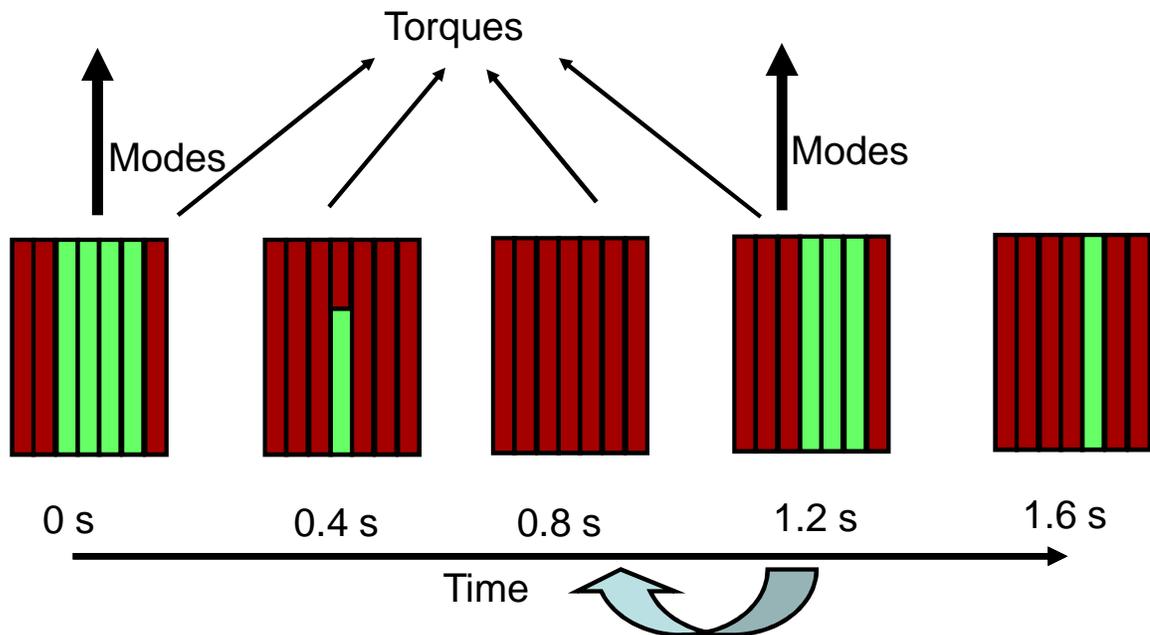


(b) Mode updates to the discrete actuators (engine state and transmission gear) occur at a slower rate (1.2 s) than the faster updates to the continuous torque commands. At these inter-sample updates, the torque selection is restricted to a particular mode. In the ideal case, the mode selection acts as a zero-order hold and the inter-sample torque is selected based on the last mode command.

Figure 8.5: Illustration of the command update scheme.



(a) To deal with unreliable actuators and unpredictable actuation delays, the inter-sample update scheme of Figure 8.5b is modified. The discrete mode commands are issued assuming the actuators will respond as requested. The continuous torque commands are selected based on the current vehicle mode, regardless of the mode command. The torques are selected by minimizing the cost among all possible torques for the current mode.



(b) If the only feasible control choices require a mode change, no valid commands are available during an inter-sample update. In this case, the mode update occurs immediately rather than waiting for the standard update time. One example is a sudden increase in torque demand while in electric mode, which requires an engine start to meet demand.

Figure 8.6: Illustration of the command update scheme.

A more general question is how to respond when there are no feasible operating points. The vehicle components are generally sized to meet driver demand, so this situation is temporary and typically occurs because driver demand exceeds the vehicle capability, or a delayed engine start or gear shift restricts available torque. The requirement to meet driver demand is considered a hard constraint, so a given engine torque command is declared invalid if it requires the electric machines to exceed limits in order to meet driver demand. If these invalid commands are simply eliminated, decisions become ambiguous when there are no valid commands. A better choice is to treat driver demand as a soft constraint. If a command does not match driver demand, a large additional penalty is added that reflects the shortfall of available torque. In this way, commands that do meet demand are selected if at all possible. If no valid commands are available, the lowest-cost command will be the one that delivers the maximum available torque.

8.4 Refining the controller in hardware

8.4.1 Hardware Debugging

After significant testing and debugging in the MIL/HIL setup, the SPSDP controllers were tested in the vehicle and worked correctly the first time. Figure 8.7 shows data from one of the initial tests. These early controllers deliberately had limited functionality: they could not use the front electric machine *EM1* or use series mode. This setup still reflects the major issues in energy management, as the vehicle rarely uses series mode and the addition of *EM1* does not change overall vehicle function significantly.

Despite the initial success, it took several months to develop the controllers to be reliable, smooth, and pleasant to drive. The first step was to implement full controller functionality including series mode and the front electric machine *EM1*. 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. Unreliable hardware slowed the process significantly, sometime making it difficult to discern if failures were related to the SPSDP controller or the hardware.

The majority of this time was devoted to issues that only became apparent in hardware,

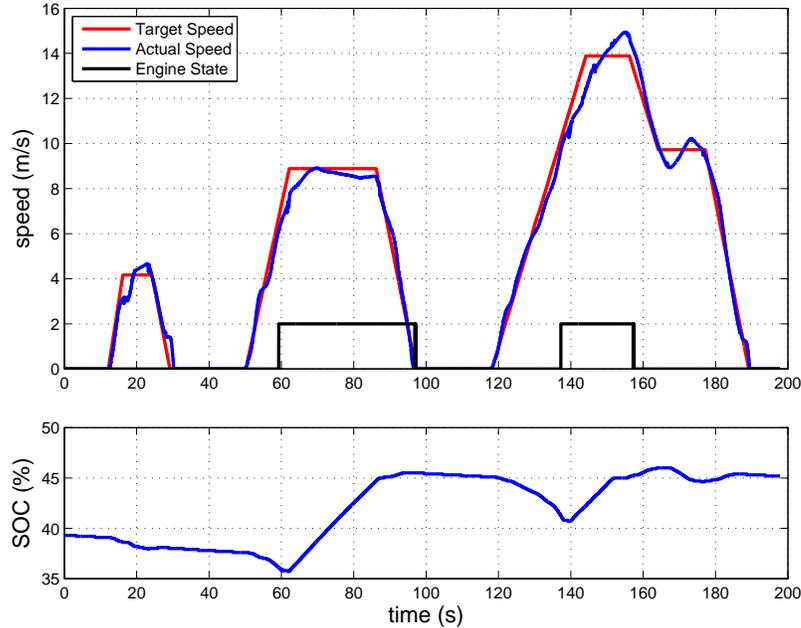


Figure 8.7: The first driving attempt. The SPSDP controller worked on the first try.

especially issues relating to driver perception. One such issue was the smoothness of torque delivery. The SDP controller updates commands in a discrete fashion, which tends to yield jumps in engine torque that feel jarring. Torque commands are also very important during mode switching, including clutch engagement. A set of filters and initialization values was developed to yield a smooth, continuous, yet responsive torque command.

Another tricky problem was how to actually coordinate torque delivery to the wheels to match the driver command. The main idea is that the sum of the engine torque and the two electric machine torques must match the driver demand. In the SDP algorithm, the engine torque is assigned, and the remaining torque is optimally partitioned between the two electric machines. There are two engine torque signals available in the baseline infrastructure: the torque command and a torque estimate from the engine controller. The estimate matches the command in general, but they diverge near gear shifts and clutch movements. Ideally, the estimated engine torque should be used in the wheel torque calculation, which allows the electric machines to quickly add supplemental torque if the engine does not immediately follow the command.

An unexpected problem arose with this method, highlighting the challenges in a real vehicle. A feedback loop between the *EM1* command and the engine controller created

undesirable oscillations in engine torque. The $EM1$ torque is determined as a function of estimated engine torque, and $EM1$ is directly connected to the crankshaft, which led to an interaction with the engine controller. Filtering the update signal for $EM1$ helped, but the final solution was simply to use the engine torque command in the wheel torque calculation.

Once the controller was fully functional and driving a few cycles, the test data were analyzed for model matching. This is a very important part of the process because SDP is inherently a model-based method and makes decisions based on the model specified. Two typical sources of error include hardware changes from the initial model and the usual problem of matching hardware to simulation. A third type of error is much more subtle and arises from the model reduction process. The real-time controller and vehicle contain many dynamics that are neglected, lumped, or simplified in the control-oriented model. Some of these simplified dynamics depend partly on the *controller* rather than the vehicle model. One would like to imagine that the model and controller can be separated, but this is not always the case with these reduced-order models. A reduced-order model calibrated from data using a different controller may be incorrect.

One clear example for this vehicle is the engine start. The control-oriented model lumps the process of starting the engine and engaging the clutch into 3 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 SDP controllers execute this process is similar but not identical. Therefore, the parameters for the reduced-order model change slightly depending on which type of controller is used.

Using test data, the basic vehicle parameters were identified and adjusted in the control-oriented model. The simulation models were not particularly well-matched to the hardware before commencing this process. This step improved performance about 8%.

8.4.2 Additional Penalty Implementation

A very interesting phenomenon occurred in the vehicle that was not seen in simulation. Under some conditions, the engine torque would oscillate while the vehicle was seemingly at steady state. These events typically occur at low pedal and a nearly constant speed, usually around 25 kph. One such event is shown in Figure 8.8. The SPSDP controller updates are

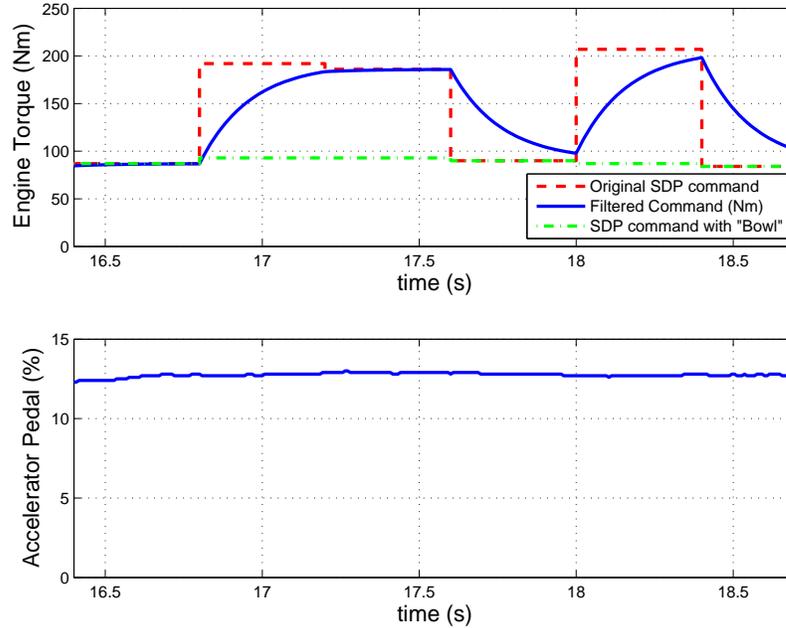


Figure 8.8: Engine Torque and Pedal Commands during an unexpected oscillation.

naturally discrete as discussed previously, so the output is filtered for a smooth ride. The original SPSDP command is shown as a dotted red line along with its filtered version. The green line shows the command after this problem was fixed and will be discussed shortly. The bottom plot in Figure 8.8 shows the accelerator pedal command in percentage of full range. The pedal input is basically constant, so there is no obvious reason for these torque oscillations. Such behavior is annoying and confusing to a driver when applying constant pedal.

The underlying reason becomes clear by studying the total cost estimate of the SPSDP process. Recall that the total cost is calculated for a range of engine torques for each gear. The actual engine torque command is selected based on the current gear by minimizing the total cost. The plots in Figure 8.9 show 7 lines, each representing a single gear or series mode. The horizontal axis is the engine torque command, and the vertical axis is the total cost. To determine the engine torque command, the algorithm finds the current gear and selects the minimum cost. In these plots, the current gear is represented by a thick line, and the minimizing torque is shown as a vertical dashed black line.

The left column of plots in Figure 8.9 shows the cost estimates used when selecting the engine torque shown in Figure 8.8 and are marked with time step. The underlying reason

for the torque oscillations is that the total cost function has two local minima that are very close in value. With a simple minimization, very small changes in vehicle states can cause the torque command to jump back and forth.

In the SDP controller design, the rate of change of engine torque was not considered and thus algorithm is free to use a jump in torque so as to minimize cost. Although perceptible jumps in torque are relatively rare, they can be disconcerting and we wish to add additional controller functionality in order to eliminate this behavior. Several methods to control this behavior were discussed in Chapter 2.3.3. One obvious approach is to add a state to the SDP model that tracks the last commanded engine torque and to penalize rapid changes; this approach is termed the “extended model” method in Chapter 2.3.3 and would increase computation by roughly a factor of 10. Instead we choose the “Instantaneous Cost” method of Chapter 2.3.3 and add a penalty only in the on-line cost function, retaining the original value function.

A state in the on-line model stores the last commanded engine torque and a penalty is assigned based on the change in torque. This penalty function should be chosen with care as it directly affects the system dynamics and can significantly alter behavior. We wish to eliminate the torque oscillations with a minimal effect on other aspects of controller performance. The penalty function selected is shown in Figure 8.10 and has a “bowl” shape. It has three main features: the penalty is zero for small torque changes, increases linearly beyond the dead zone, and saturates. This does not affect small changes to engine torque and still allows large jumps in torque if they are sufficiently less costly. These parameters were calibrated to get good behavior. The most important parameter is the saturation value, which is set just high enough to eliminate frivolous oscillations.

The right column of Figure 8.9 shows the outcome after adding the bowl penalty. The actual bowl penalty is shown as a magenta 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. The torque command with this additional bowl penalty is shown in green in Figure 8.8.

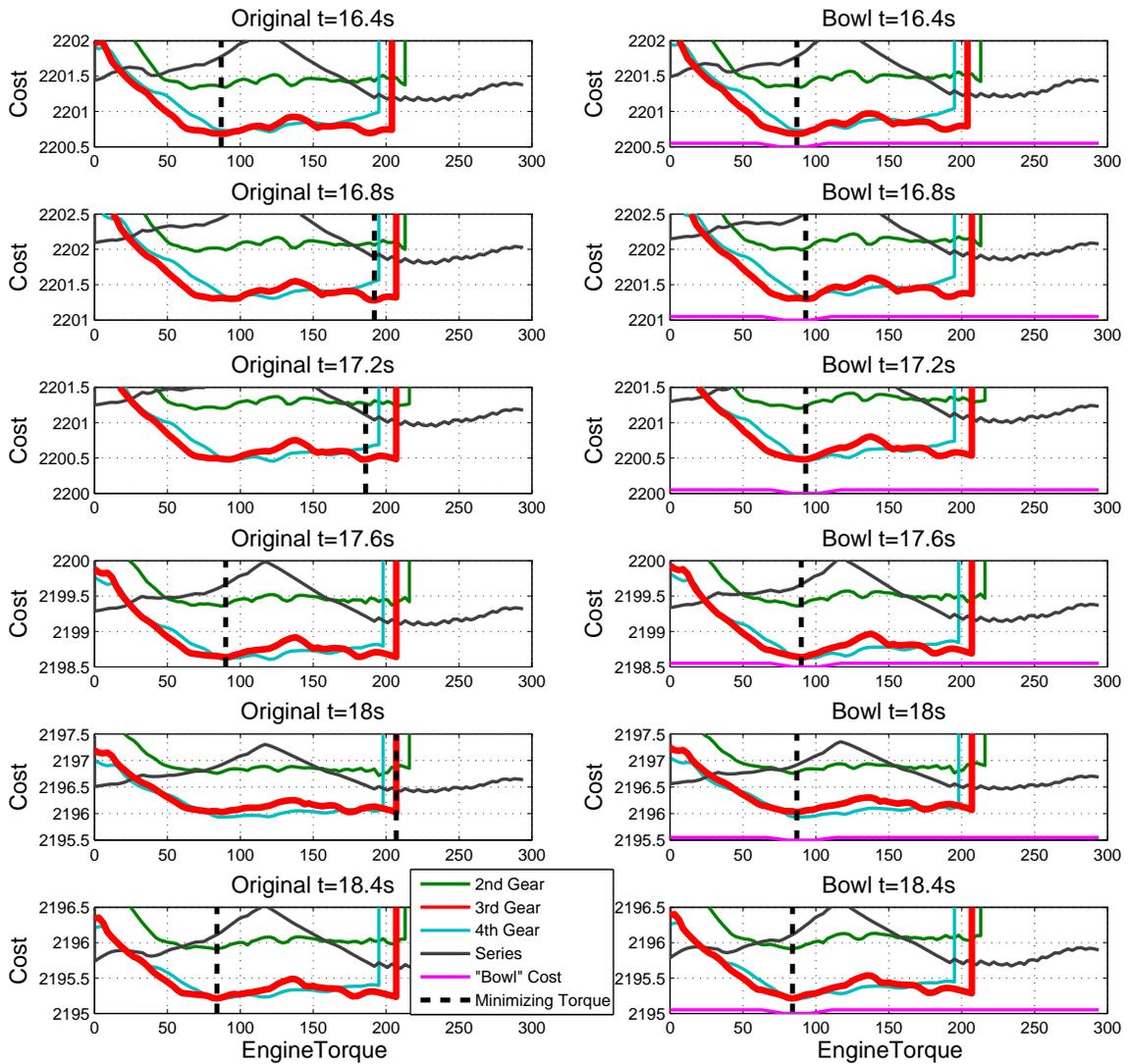


Figure 8.9: The value function during an unexpected pedal oscillation. The recorded real data is shown on the left. The proposed solution to this oscillation is an additional “bowl” penalty on engine torque. It is quite difficult to exactly replicate hardware test conditions, so the column on the right represents what the commands would have been with the improved controller measuring the same vehicle state data.

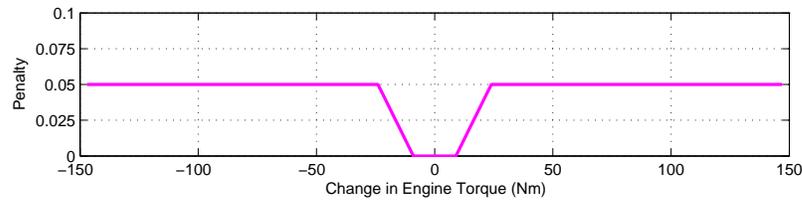


Figure 8.10: Additional penalty added to the value function based on the change in Engine Torque. This is termed a “bowl” penalty due to its shape.

8.5 Hardware testing results

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. The structural implementation of the feedback controller does not change with different parameter values, only the values stored in a lookup table change. Several different controllers are stored simultaneously in the real-time controller and can be selected without recompiling. The baseline controller and two different SPSDP controllers are shown driving the FTP72 cycle in Figure 8.11. The two SPSDP controllers each have a different penalty for engine start/stop, yielding different behavior. Changing the penalties does effectively modify vehicle behavior, as predicted by the simulation studies in Chapters IV and V. The test vehicle experienced a hardware failure that was not repaired and the results in this section reflect the malfunctioning vehicle. The controllers and vehicle still function largely as designed, but the fuel economy numbers may be unreliable.

Three controllers were also run on the NEDC cycle, as shown in Figure 8.12. These controllers also have different penalties, but yield similar numbers of engine events. This is due to the contrived nature of the NEDC, which is composed of repeated constant ramps to constant speeds. It is fairly well-defined when engine starts should occur. As shown in Chapters IV and V, varying the penalties in SDP generally allows changes in the total number of engine events.

For further comparison, the torque-speed engine operating points are shown in Figure 8.13 for the baseline controller and the SPSDP controller running FTP. The top two figures (Figures 8.13a and 8.13b) show the commanded torque. A hardware failure prevented the engine from reaching its rated torque, so Figures 8.13c and 8.13d show the actual torque. The dark black line is an operational limit for noise and vibration specified during the design phase. The SPSDP controller (Figure 8.13b) generally respects this constraint, although the baseline controller calculates the limit differently. The SPSDP controller slightly overshoots the limit when operating on the boundary and the engine speed drops before the next engine

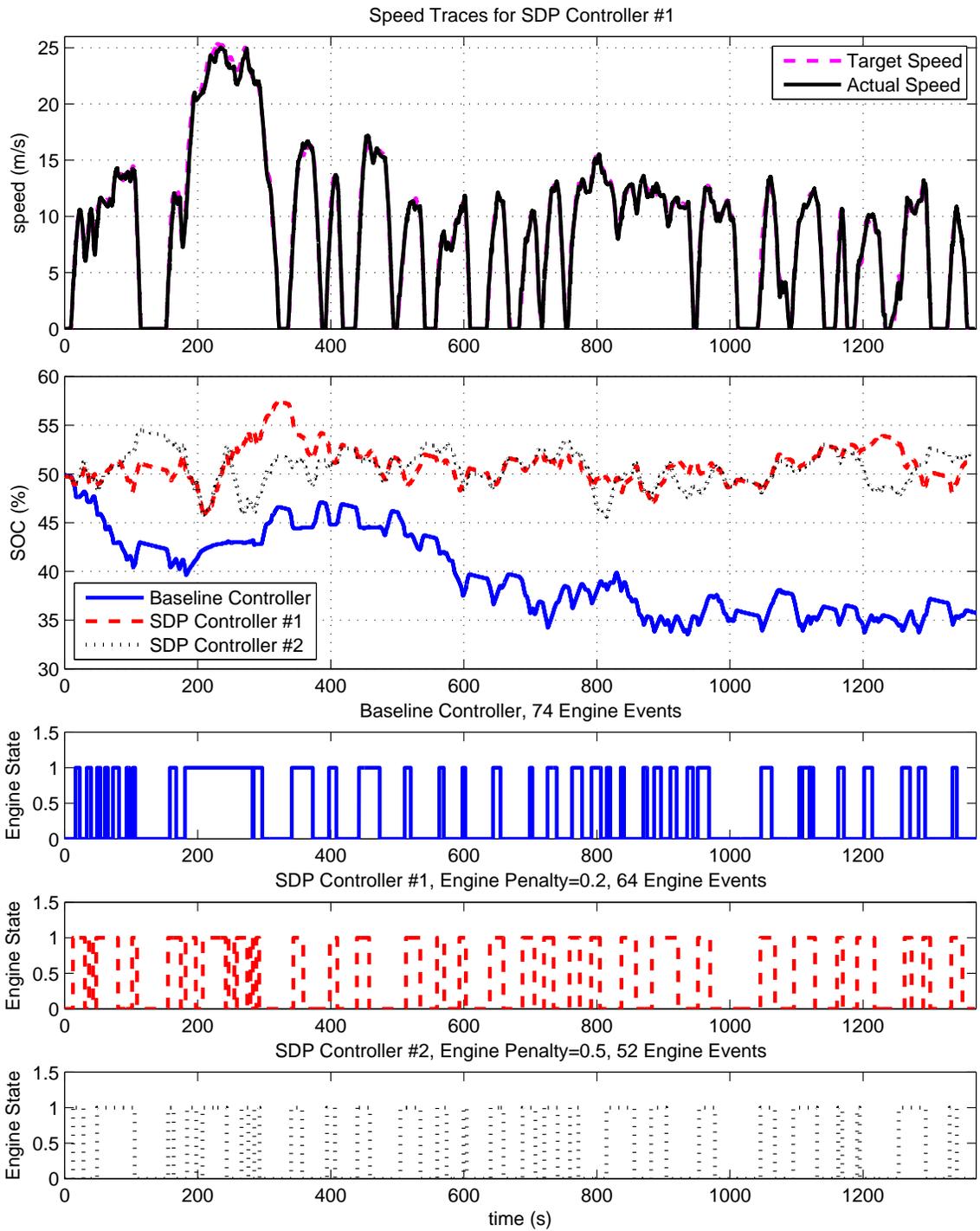


Figure 8.11: Driving Traces on FTP.

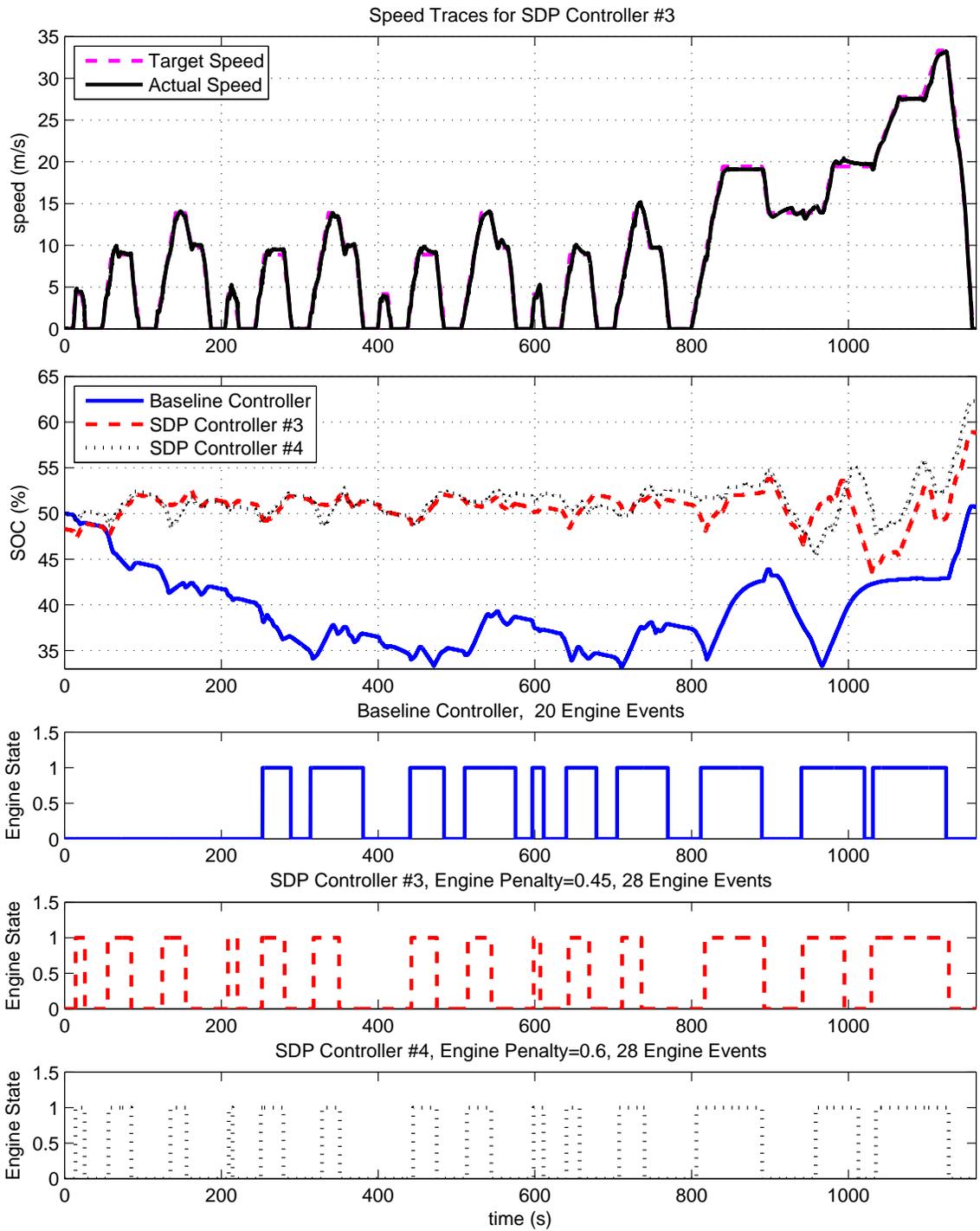
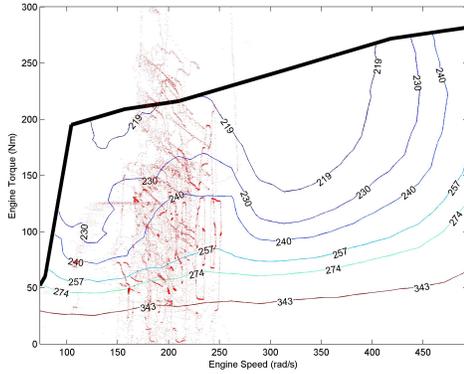
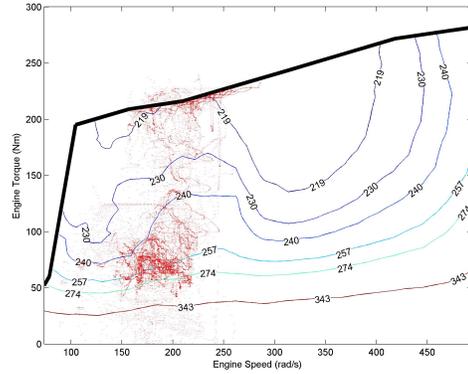


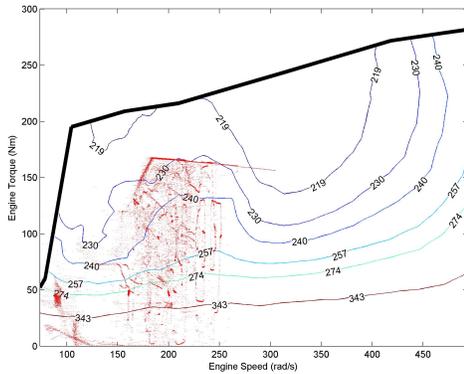
Figure 8.12: Driving Traces on NEDC.



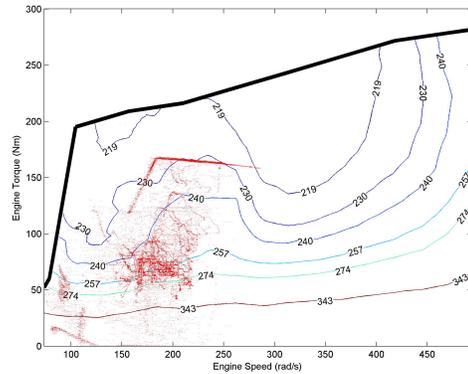
(a) Ford Torque-speed commanded operating points on FTP



(b) SPSDP Torque-speed commanded operating points on FTP



(c) Ford Torque-speed actual operating points on FTP



(d) SPSDP Torque-speed actual operating points on FTP

Figure 8.13: Engine torque-speed operating points during hardware test on FTP. The top plots show commanded operation while the bottom show the actual torque reflecting an engine controller torque limit.

torque update as shown in Figure 8.13b

For a more detailed view of the system dynamics, a zoomed view of the third NEDC “hill” is shown in Figure 8.14. The vehicle accelerates from rest in electric mode, the engine starts, the transmission engages, and the engine begins delivering torque. Transmission gear shifting is clearly visible as sawtooth steps in engine speed. The bottom two plots show the electrical dynamics: SOC and the ERAD and CISG commands. Before the engine starts, the vehicle is propelled by the ERAD only and the SOC drops. The CISG is then used to start the engine. The engine then provides the motive power and charges the battery through the CISG while the ERAD is idle. After the engine shuts off, the vehicle is again

in electric mode with the ERAD providing propulsion and braking.

As mentioned in Section 8.3.4.2, the engine start dynamics are more complex than originally modeled. The engine start event of Figure 8.14 is shown in greater detail in Figure 8.15. The SPSDP 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. During the start process, the CISG applies positive torque to spin the engine. The estimated engine torque due to combustion is shown along with the engine torque command. The torque delivered to the wheels is zero until the clutch engages. The SPSDP engine torque command is also shown. The engine start is executed by a low-level controller. Once the clutch is engaged, the SPSDP controller starts issuing torque commands. Note that the SPSDP torque command is initialized at the engine torque estimate to avoid rapid transients. The estimated engine torque begins following the actual torque once the start dynamics have smoothed out.

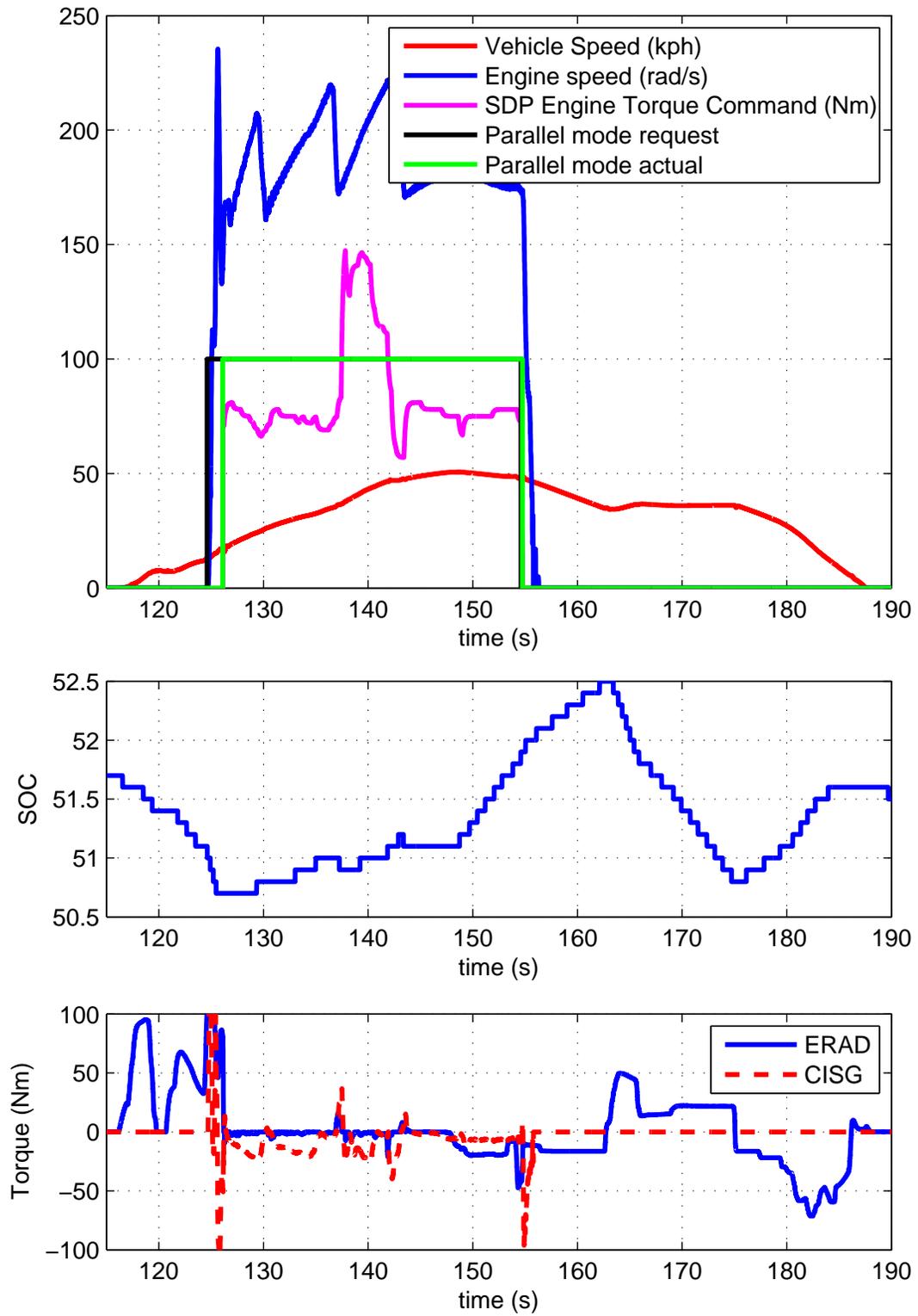


Figure 8.14: Representative vehicle behavior on a typical NEDC “hill” demonstrating electric mode, engine start, parallel mode, engine stop, and a return to electric mode.

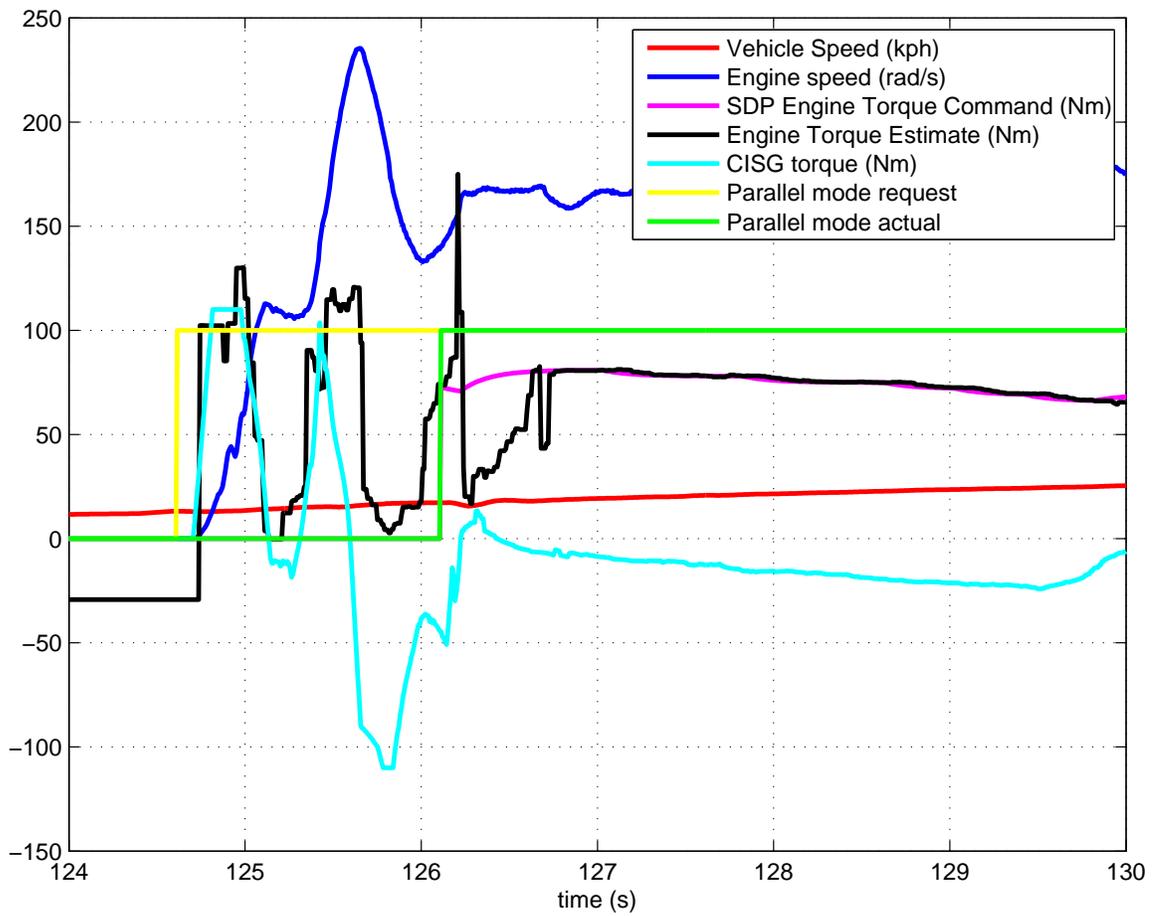


Figure 8.15: Detailed view of the engine start dynamics shown in Figure 8.14.

CHAPTER IX

Conclusions

9.1 Overall Summary

The energy management controller for a hybrid vehicle is a major factor in the vehicle's overall performance. This dissertation has studied controllers generated using Shortest Path Stochastic Dynamic Programming (SPSDP). The SPSPD-based controllers use a statistical description of expected driving behavior to minimize a cost function that is a weighted sum of consumed fuel and drivability penalties, such as shift events and engine on-off events. By varying the weights, a control designer can systematically trade off fuel economy and drivability. These tradeoffs are optimal for given driving statistics, yielding the Pareto tradeoff curve of fuel economy versus engine on-off activity. The performance of the SPSPD-based controllers has been compared against an industrial controller provided by Ford Motor Company.

In addition to the individual contributions, the work presented in this dissertation is somewhat rare in that it spans the full range of a development process: theoretical analysis, initial testing, rigorous simulation, real-time implementation, and hardware test.

Chapter II discussed general theoretical contributions to the dynamic programming algorithm: computation reduction using the minimization decomposition technique, and a general classification of how various behaviors may be incorporated into the algorithm. Specific examples of hybrid vehicle applications were provided. Chapter III described the vehicle model and problem formulation, along with an introduction to the drivability metrics. Chapter IV demonstrated the SPSPD-based controllers deliver greater than 15% perfor-

mance improvement over the industrial controller on two government test cycles.

Chapter V focused on practical issues. Controller performance and robustness on real-world drive cycles were evaluated using a highly accurate simulation model and compared to a baseline industrial controller. The fuel economy, drivability, and battery SOC maintenance were studied on sets of 100 real-world cycles using both cumulative and statistical methods. Results show the SPSDP-based controllers yield 10% better performance than the baseline controller on real-world driving data. The SPSDP-based controllers are robust to variations in drive cycle and the statistics used to design the controllers. This was shown by simulating about 1000 controllers designed using four different sets of drive cycle statistics on large numbers of real-world drive cycles.

One major contribution of this dissertation was the comparison to an industrial benchmark. Given that the industrial controller is manually tuned, one may wonder if the comparison is valid and if the benchmark controller could be tuned better. Chapter VI addressed this issue by exploring the maximum attainable performance of the baseline controller and demonstrated that no possible tuning of the baseline controller matched the performance attained by SPSDP. This implied fundamental structural limitations of the baseline algorithm that could not be overcome by tuning.

Simplified metrics were developed and used throughout this dissertation to study the generally qualitative concept of drivability. Chapter VII described the development of these metrics and demonstrated the validity of the simplified metrics even when considering more complex metrics.

Finally, Chapter VIII described the hardware implementation and testing of SPSDP controllers in a real vehicle, addressed issues relating to unreliable actuators and uncertain timing, and presented test data.

The combined simulation and hardware testing have shown that Shortest Path Stochastic Dynamic Programming is a viable method for designing real-world controllers. The controllers can be implemented directly with little manual adjustment, generate performance comparable to the current industrial state of the art, and function in a real vehicle.

9.2 General Comments

Throughout the presentation of the results, we have mostly emphasized the excellent fuel economy performance of the SPSDP controllers in comparison to the baseline controller. Nevertheless, we feel the main contribution of this dissertation is the demonstration of stochastic optimization as a viable *design method* for practical energy management controllers. This dissertation has highlighted the fuel economy vs. drivability tradeoff. The controllers generated through SPSDP are directly implementable in real-time and are provably optimal. The primary engineering tasks are identifying the vehicle models (simplified and detailed), and performing the off-line computation of the stochastic dynamic programming algorithm. The off-line computations can be arduous¹, but they are straightforward and much more rapid than the traditional manual design process, where engineers spend most of their time writing rule-based controllers that require significant hand tuning with no optimality guarantees.

It is straightforward with SPSDP to generate Pareto tradeoff surfaces. SPSDP not only can identify the optimal tradeoff surface, but it directly generates controllers that operate on it. This greatly simplifies the design process by precisely quantifying tradeoffs among various attributes. Once the Pareto tradeoff surface is known, a decision can be made about where to operate the vehicle for a particular market.

The SPSDP controllers developed in this dissertation have been implemented and tested in the prototype vehicle. The controllers run in real time on a rapid-prototyping system, and the vehicle can be driven normally. To change vehicle behavior, it is straightforward to simply select an operating point from the Pareto surface, download the appropriate controller, and drive the vehicle again. The on-line computations are modest, consisting primarily of interpolation and minimization, and it is possible to trade storage memory for on-line computation. Changing controller designs simply involves replacing tables that are computed off-line. In addition, a designer can develop the full SPSDP controller and then reduce it to a simpler functional form. At its heart, SPSDP is a pure state feedback controller, so the feedback function can often be reduced to a simpler form with some

¹A single instance of the controllers used here can be computed on a 2008-era laptop in a few hours.

performance loss [60, 61]. Manufacturers may use this technique not just to save on real-time computation, but to make controller behavior more transparent, directly tunable, or easier to maintain.

9.3 Future Directions

One place where SPSDP can have a major impact is in controller design for new vehicles. Significant effort is required to develop a controller for a new drivetrain, especially with a completely new architecture (e.g., Series-Parallel vs. Power Split). The SPSDP method can automatically generate a provably optimal controller for a given vehicle architecture and component sizing much faster than a person could do it manually. This is especially valuable early in a program during the hardware design phase. When comparing architectures and components, the vehicle performance is highly dependent on the controller design, and it is difficult and time-consuming to manually tune a controller for each possible vehicle design. Effectively, the design must be conducted with limited ability to estimate the closed-loop performance of each candidate design. A method like Deterministic Dynamic Programming can automatically generate the best possible performance for a given vehicle, assuming known future drive-cycle information. Stochastic Dynamic Programming generates a causal controller that is a more reliable indicator of what is possible in practice.

While it is unlikely that existing vehicle controllers will be redesigned using SPSDP, hopefully the next generation of production hybrids can take advantage of the results presented in this dissertation, especially in the early design phases. There is still significant work to be done before a full production controller can be designed using a method like SPSDP.

Stochastic dynamic programming in general is a very powerful tool and is particularly applicable to new types of vehicles and new operating modes that involve future information. Vehicle controllers can theoretically generate superior performance by using future information like GPS, traffic data, or driver inputs. Electric and plug-in vehicles are projected to become more popular; these vehicles in particular will benefit from these advances because they have finite range.

APPENDICES

APPENDIX A

Computation Techniques

A.1 Background

Although not particularly glamorous, computer code makes or breaks SDP. Extensive computation is required to solve even simple problems, and inefficient code can easily require five orders of magnitude more memory and time than it should. Efficient code is the hidden enabling factor in this dissertation and required several years of development. Solving a five state SDP is only possible with well-developed techniques; most research papers use three states. Controllers in this dissertation were designed and simulated rapidly with minimal computation and memory requirements. Combined with grid computing, this allowed the testing of tens thousands of controllers over hundreds of drive cycles each.

When research groups start using SDP, they often lose several years trying to efficiently implement their code. Everyone seems to make the same mistakes along the way. The goal of this section is to convey to practical aspects of solving these problems that don't make it into journal papers.

There are two main themes to come away with here. One is that careful coding can easily reduce memory and time requirements by a factor of 2-10 *for each change*. The second is that there is almost always a tradeoff between memory and computation time.

This chapter is divided into sections that each express a main point. Any code is necessarily system-specific, but these best practices are rather general.

A.2 Choose an Appropriate Model

This is the single most important step, and arguably the most difficult. For most problems, accurate simulations require a complex, detailed model. Unfortunately, it is difficult to build controllers for these complicated models, so a reduced-order model is often used. This is especially true for Dynamic Programming. Most projects will end up using two models: a simple control design model and a complex validation model. The process of selecting states and variables to be included in the reduced model may not be easy. It can also be difficult to validate the two models and judge when the simple model is “good enough.” A closely related issue is the speed or update rate of the reduced-order model, which can allow very slow or fast dynamics to be neglected while capturing the relevant dynamics of interest for a particular problem.

There are two important considerations in this process: the number of states and control inputs to the reduced model, and the speed of its implementation. Depending on certain choices, the reduced model may be called millions of times during the solution process, so every millisecond counts.

Minimize calls to the Control Model

When solving a dynamic programming problem, the algorithm basically maps out all possible combinations of state and control input. This means a large number of calls to the control model. Anything to speed this process up is valuable. For example, Simulink (a MathWorks product) can be called with a vector of initial conditions and will return the results all at once.

There are at least three main ways to reduce time spent in evaluating the model. The first is simply to speed up the control model somehow. A second option is to store the reduced model as an interpolated lookup table, which makes model evaluation faster but requires more memory and sacrifices some accuracy. The third option is perhaps the most elegant. In most dynamic programming problems, the model is called many times for the same parameter values. By exploiting this structure, one can simply pre-compute once for each case and store the result, which does not entail any loss of accuracy.

The worst thing to do is to use a slow, complex model and call it repeatedly for each and every state and control combination. This can increase solution times from minutes to days. This is a very common problem when people start writing code.

A.3 Carefully define limits, restrictions, and cost functions

A dynamic programming algorithm will literally seek out all possible ways to minimize a cost function. When designing controllers by hand, there are often unarticulated assumptions or behavioral restrictions. These conditions must somehow be incorporated into the plant model or the cost function. Otherwise, the algorithm will take advantage of the situation. These situations are typically obvious in hindsight, but difficult to enumerate a priori.

In this vehicle, the selection of the penalty on gear shifting demonstrates this phenomenon. A shift is simply defined as a change in gear when the clutch is engaged. Engaging or disengaging the clutch is usually smoother than a shift, so no penalty is assigned. Similarly, when the clutch is disengaged, shifts in the transmission are not perceptible to the driver and thus are also not penalized. At first pass, all this seems logical. However, when the penalty on gear shifting is sufficiently high and a shift is required, the algorithm will then disengage the clutch, shift gears, and reengage. This action incurs no cost. In hindsight, it is easy to see that the cost function actually encourages this behavior. A human designer would be unlikely to implement such a scheme. A qualitative goal like “smooth shifting” entails vast amounts of detail and subtlety. Attempts to rigorously define qualitative behavior through cost functions typically lose some of the details because that function becomes the only goal.

As a project moves through the development process, various unexpected behaviors are likely to appear and require control. These SDP algorithms are not widely used in industry yet, and usually start out with simple goals on simple models. This is especially true of academic projects. As the controllers show improved performance and begin to compete with other design methods, they are viewed as legitimate controllers and no longer just an academic curiosity. This is both good and bad; it is quite satisfying to benchmark against a

“real” industrial controller. However, the standards used to evaluate production controllers are quite complicated, and very different from the simple goals used in the beginning. Suddenly, the SDP controllers must satisfy a large number of constraints that were not articulated at the beginning.

This project faced a moving set of objectives. Initial goals were simple, and initial fuel economy performance was less than the baseline industrial controller. Once the SDP performance started exceeding the benchmark by 10-15%, we learned that the industrial controller was respecting various limits and conditions that were not articulated in the beginning and were therefore not included in the SDP algorithm. This made it difficult to do a true “apples to apples” comparison. For example, the baseline controller used an additional engine torque limit based on noise considerations, while the SDP algorithm only respected hardware limits. The baseline controller also had some traction control restrictions on how wheel torque was distributed but the SDP controller did not. These limits have since been incorporated and SDP still outperforms the baseline controller.

A.4 Grid point utilization

Dynamic programming problems are numerically solved by discretizing the state space into grid points [56]. The numerical accuracy of the solution depends on how well the grid points represent the underlying continuous state space, while the computational burden depends on the number of grid points. The selection of these points is critical to obtaining a good solution in a reasonable amount of time. This section presents several different ideas about how to structure the discretization. Relatively minor structural changes can reduce complexity by a factor of 2-10. The repeated redesign of the code structure used in this dissertation allowed continuous improvement, for a total improvement of about 3 orders of magnitude with respect to the initial work. The examples presented here are specific, but the ideas are general.

A key idea when assigning grid points is to ensure that grid points are used “equally”. This tends to ensure that frequently used regions of the state space are more finely discretized than infrequently visited portions of the state space. The most obvious benefit

comes from eliminating grid points that are never used. These unused points are often created when using square or “plaid” grids, which are convenient for interpolation.

One option is to use discrete states in the system model, such that the system always moves directly from one grid point to another. This requires no interpolation. Another option is to use a more complex interpolation method that does not require plaid grids. These techniques are used occasionally, but tend to be much slower and are avoided in the main iterations for dynamic programming. Interpolation into non-plaid data is used when building the lookup tables for system dynamics because the hardware specification data are not always conveniently spaced.

The approach used here was to carefully define the system variables to get the best of both worlds: fast interpolation with good grid utilization. From a hardware perspective, the designer can specify the system dynamics in any number of coordinate systems which are all equally accurate, and the final choice can be somewhat arbitrary. This choice becomes very important for grid utilization and requires careful thought. The main goal is to select system variables that are relatively independent of each other. For example, the engine torque delivery could be represented as either the wheel torque or the engine output torque, which are clearly related by the transmission ratio. Either way will work, but the wheel torque delivery varies widely with vehicle speed, while the engine torque remains mostly constant regardless of vehicle speed.

More opportunity for improvement comes when one considers exactly what information must be encoded in the system state. Grid points can be eliminated not only when they are underutilized, but also when they encode excessive information. A clear example in this dissertation comes from the transmission. Nominally, 6 gears and a clutch would require 2 independent states and a total of 12 grid points (6×2). Consider what each state represents. The gear number is clearly important knowledge when the clutch is engaged. What about when the clutch is disengaged? The transmission does not affect any other dynamics in this case. The designer must decide if this is an important detail; it was neglected in this dissertation. Instead, the clutch was assumed to be engaged when a gear is selected, and the disengaged state was represented as a seventh gear. This leaves 7 grid points instead of 12, avoiding a 70% increase in memory and computation. With this trick, the addition

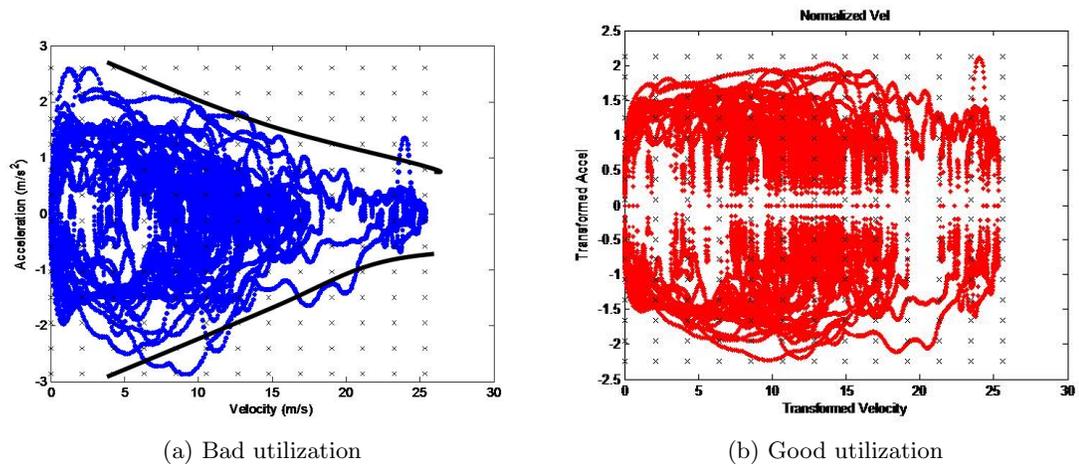


Figure A.1: Selection of stochastic variables. Using a plaid grid requires a “square” of grid points. Using velocity and acceleration as driver states tends to underutilize the high speed/high acceleration points (Figure A.1a). Using speed and power has the same problem, but at low speeds. A transformation is used (Figure A.1b) that basically uses normalized acceleration at low speeds at normalized power at speeds above roughly 10 mph.

of an engine on/off state uses 14 grid points. A similar trick can be used after observing that the engine is never really off with the clutch engaged because by definition the engine is on if it is spinning. This means the engine on and clutch engaged state (parallel mode) can be represented with 6 grid points for the gears, the engine on and clutch disengaged state (series mode) requires one grid point, and the engine off and clutch disengaged state (electric mode) requires one state. This means the grid points can be reduced from 14 to 8. This final trick was not implemented in the code used in this dissertation, primarily for time reasons. In summary, the engine state, transmission gear, and clutch information would normally require 24 grid points ($6 \times 2 \times 2$) but was reduced to 14 grid points in this dissertation and could easily be reduced to 8 grid points.

APPENDIX B

Implementing Penalties using the Instantaneous vs. Extended Model Method

B.1 Introduction

As discussed in Section 2.3, there are usually several options for incorporating additional behaviors into the cost function, each with their own benefits and drawbacks. The work presented in Chapters III through VIII addressed drivability using the extended model method, which introduces additional states to the model. Initially, the instantaneous cost method was also considered, especially given the computational cost of the extended model method. This appendix studies the performance implications of this decision.

This portion of the work was carried out early in the project and published in [70]. As such, the vehicle model and some of the assumptions do not exactly match the vehicle model used in the rest of this dissertation. This appendix uses an earlier version of the control-oriented model (Section 3.1.3) for both controller design and simulation. The model uses different parameters than in the rest of this dissertation, the vehicle has only one electric machine on the rear axle (*EM2*), and the engine is assumed to turn off if the clutch is disengaged. The vehicle model is described in its entirety here to clarify these differences.

It is shown that the extended model method yields up to 10% fuel economy improvement when compared to the simpler instantaneous cost method. This result is obtained by comparing a SPSDP controller with drivability added using the extended model method to

a second SPSDP controller, designed for fuel economy only, that incorporates drivability through a supplemental cost term. This result justifies the additional complexity required to use the extended model method.

Note: The vocabulary used in the majority of this dissertation differs from that originally used in the publication of these results [70]. The vocabulary in this Appendix is from [70]. In this appendix, Shortest Path Stochastic Dynamic Programming is abbreviated SPDP rather than SPSDP. A controller designed using the extended model method of Section 2.3 is termed the “SPDP-Drivability” controller, and a controller using the instantaneous cost method is termed the “SPDP-Local” controller.

B.2 Vehicle Model

B.2.1 Vehicle Architecture

The vehicle model studied in this appendix is a parallel electric hybrid. A 2.4 L diesel engine is coupled to the front axle through a clutched 6-speed automated manual transmission. An electric machine is directly coupled to the rear axle through a fixed gear ratio without a clutch, therefore the electric machine is always rotating at a speed proportional to vehicle speed. Energy is stored in a 1.5 kWh battery pack. The system parameters are listed in Table B.1.

Table B.1: Vehicle Parameters

Engine Displacement	2.4 L
Max Engine Power	120 kW
Electric Machine Power	35 kW
Battery Capacity	1.5 kWh
Battery Power Limit	34 kW
Vehicle Mass	1895 kg

B.2.2 Modeling Assumptions

For computational reasons, the vehicle model must be as simple as possible. The vehicle model used here contains the minimum functionality required to model the vehicle behavior

of interest on a second-by-second basis. Dynamics much faster than the sample time of 1s are ignored. Long-term transients that only weakly affect performance are also ignored; coolant temperature is one example.

The dynamics of the internal combustion engine are ignored; it is assumed that the engine torque exactly matches valid commands and the fuel consumption is a function only of speed, ω_{ICE} , and torque, T_{ICE} . The fuel consumption F is derived from a lookup table based on dynamometer testing,

$$Fuel\ flow = F(\omega_{ICE}, T_{ICE}).$$

The automated manual transmission has discrete gears and no torque converter. Losses in this highly efficient transmission [45] are ignored. The engine speed is assumed directly proportional to wheel speed based on the current transmission gear ratio R_g ,

$$\omega_{ICE} = R_g \omega_{wheel}.$$

The engine torque T_{ICE} transmitted to the wheel is similarly assumed proportional to wheel torque based on the current gear ratio R_g . The electric machine torque T_{EM} transmitted to the wheel is proportional to the constant EM gear ratio R_{EM} . The total wheel torque T_{wheel} is thus the sum of the ICE torque to the wheel $R_g T_{ICE}$ and the electric machine torque to the wheel $R_{EM} T_{EM}$,

$$R_g T_{ICE} + R_{EM} T_{EM} = T_{wheel}.$$

The engine can be turned off at any time, in which case the clutch is disengaged and engine speed is zero independent of wheel speed. Transmission gear shifts are allowed every time step (1s) and transmission dynamics are assumed negligible.

The battery system is similarly reduced to a table lookup form. The electrical dynamics due to the motor, battery, and power electronics are assumed sufficiently fast to be ignored. The energy losses in these components can be grouped together such that the change in battery State of Charge (SOC) is a function $\bar{\kappa}$ of Electric Machine speed ω_{EM} , torque T_{EM} ,

and battery SOC at the present time step,

$$SOC_{k+1} = \bar{\kappa}(SOC_k, \omega_{EM}, T_{EM}). \quad (\text{B.1})$$

Assuming a known vehicle speed, the only state variable required for this vehicle model is the state of charge (SOC). Changes in battery performance due to temperature, age, and wear are ignored.

The control inputs for this vehicle are the IC engine torque, electric motor torque, and the gear. Given the control choices, ICE speed and EM speed are fixed given vehicle velocity. During operation, the desired wheel torque is defined by the driver. If we assume the vehicle must meet the torque demand perfectly, then the sum of the ICE and EM contributions to wheel torque must equal the demanded torque T_{demand} ,

$$R_g T_{ICE} + R_{EM} T_{EM} = T_{demand}.$$

With this constraint, the choice of ICE and EM torque are no longer independent. Their relationship can be expressed in several ways, including as a *Power Split Ratio* defined as the ratio of ICE power to the road power demand [60]. For computational convenience, the ICE crankshaft torque is chosen as the control input. This leaves the system control inputs as *Engine Torque* and *Transmission Gear*.

Simulation is conducted assuming a “perfect” driver. At each time step, the vehicle velocity is the desired cycle velocity. The desired road power is calculated as the exact power required to drive the cycle at that time. Now, given vehicle speed, demanded road power, and this choice of control inputs, the dynamics become an explicit function κ of the state *Battery SOC* and the two control choices as shown in Table B.2,

Table B.2: Vehicle Dynamic Model

State	Control Inputs
Battery Charge (SOC)	Engine Torque
	Transmission Gear

$$SOC_{k+1} = \kappa(SOC_k, T_{ICE}, Gear). \quad (\text{B.2})$$

The engine fuel consumption can be calculated from the control inputs.

B.2.3 Operational Assumptions

This model uses several assumptions about the allowed vehicle behavior.

1. The clutch in the automated manual transmission allows the diesel engine to be decoupled from the wheels. This allows the engine to shut off during forward motion.
2. There is no option to have the engine idle with the clutch disengaged; the engine shuts off.
3. There is no ability to slip the clutch for starts.
4. There are no traction control restrictions on the amount of torque that can be applied to the wheels.

B.3 Shortest Path Stochastic Dynamic Programming

If drivability is an issue in controller design, there are two options. The first choice is to include the drivability issues in the original cost function using the extended model method. A second choice is to use the instantaneous cost method. Controllers are designed using both methods to evaluate the relative performance gains.

B.3.1 Cost Function

Just as in Chapter III, we wish to minimize a performance index

$$J = \sum_0^T Fuel\ flow + \alpha \sum_0^T GE + \beta \sum_0^T EE + \phi_{SOC}(x_T), \quad (\text{B.3})$$

where GE and EE are the number of Gear and Engine Events respectively, α and β are weighting factors, and $\phi_{SOC}(x)$ is an additional cost added to ensure that the vehicle is charge sustaining over the cycle.

To implement the optimization goal of minimizing (B.3), a running cost function is prescribed as a function only of the state x and control input u at the current time

$$c_{full}(x, u) = F(x, u) + \alpha \mathbf{I}_{GE}(x, u) + \beta \mathbf{I}_{EE}(x, u) + \phi_{SOC}(x) \quad (\text{B.4})$$

where the function $\mathbf{I}(x, u)$ is the indicator function and shows when a state and control combination produces a Gear Event or Engine Event. Fuel use is calculated by $F(x, u)$. The SOC-based cost $\phi_{SOC}(x)$ still applies only at key-off, when the systems transitions to the key-off absorbing state. Note that setting α and β to zero means solving for optimal fuel economy only.

B.3.2 Extended Model Formulation

The first approach is to use the extended model method, just as in Section 2.3. To track the drivability, two additional states are required: the *Current Gear* (1-6) and *Engine State* (on or off).

Bringing this all together, the full system state vector x contains five states: one state for the vehicle (*Battery SOC*), two states for the stochastic driver (v_k, a_k), and two states to study drivability (*Current Gear* and *Engine State*). This formulation is termed the “SPDP-Drivability” controller. The control u contains the two inputs *Engine Torque* and *Transmission Gear*, as described in Section B.2 and Table B.2.

B.4 Instantaneous Cost Formulation

The second approach is to use the instantaneous cost method by designing an SPDP controller for the fuel-only case and addressing drivability after the fact. In this case, the drivability restrictions are still implemented via optimization, but they are only “local” in the sense that there is no estimate of the future cost [94, 95].

This local design method is implemented as follows. The value function $V_{local}(x)$ is calculated by optimizing for fuel only using a cost $c_{local}(x, u)$ that only includes fuel $F(x, u)$

and key-off SOC $\phi_{SOC}(x)$,

$$V_{local}^*(x) = \min_{u \in U} E_w [c_{local}(x, u) + V_{local}^*(f(x, u, w))],$$

$$c_{local}(x, u) = F(x, u) + \phi_{SOC}(x).$$

Recall that the only states required are velocity, acceleration, and SOC; this makes the computation much easier. The real-time controller is then implemented using the reduced dimension value function $V_{local}(x)$, but the full dimensional cost function (B.4) that includes drivability. The real time-controller must still track the full set of states, but it is much easier in real-time than when calculating $V(x)$. The real-time controller is then

$$u^*(x) = \operatorname{argmin}_{u \in U} E_w [c_{full}(x, u) + V_{local}(f(x, u, w))].$$

This method is termed a “local” controller.

The main difference between the two controller design options is computation. Using a local controller saves significant computation, but the result is sub-optimal and likely sacrifices some amount of performance. The main idea of this appendix is to determine if the performance benefit of using SPDP-Drivability controllers justifies the increased off-line computation. See Section 2.3.4 for a comparison of the computation requirements.

B.5 Main Results

The main purpose of this chapter is to quantify the benefits of including drivability in the full optimization. To that end, two sets of controllers are designed: a set of SPDP-Drivability controllers as in Section B.3.2, and a set of local controllers as in Section B.4. Many different controllers are designed, each with different drivability penalties. In the end result, one can compare the effectiveness of optimization for drivability using the SPDP-Drivability vs. the Local method.

Both sets of controllers are designed and simulated on the Federal Test Procedure (FTP) cycle. These controllers are causal; the real-time implementation only has knowledge of the drive cycle statistics. Each individual controller is simulated and the metrics of interest are

recorded.

There are two major ways to compare the results. The first method is to simply tabulate the total cost of a cycle based on the cost function (B.4). Then for each set of drivability penalties α and β , compare the total accrued cost of the local to the SPDP-Drivability controllers. This method answers the question: “Given the cost of drivability events, which method provides a better controller?”

The second method takes a different approach. Both local and SPDP-Drivability controllers are found that produce a given number of gear and engine drivability events. The fuel economy of the two is then compared. This method answers a different question: “Given a desired drivability performance on a certain cycle, which controller yields better fuel economy?” This second method is used here as it is more realistic. This method allows controllers to be selected by making informed judgements about drivability events. The number of events can be benchmarked against existing vehicles, and engineers can easily judge “too many” or “too few.”

The results of this comparison are shown in Figure B.1. This figure shows the fuel economy obtainable for a given number of gear and engine events using the two different control methods. The SPDP-Drivability controller yields performance improvement of up to 10%. This performance improvement occurs exactly in the region of interest for production vehicles: typical vehicles have 50-80 engine events and 60-80 gear events on FTP.

Figure B.1 also clearly illustrates the tradeoffs between drivability and fuel economy. This allows an intelligent selection of the desired operating point. The simplest purpose of this figure is to select the best controller for given drivability criteria. However, the figure also shows the sensitivity to changes about that operating point. For example, the figure shows a high sensitivity to engine events. The designer may choose to increase fuel economy by allowing more engine activity. Similarly, suppose the initial operating point has 100 shifts. The designer knows that fuel economy is insensitive to gear activity in this vehicle and can choose to decrease shift activity with little loss of fuel economy. The results here depend on the hardware in question; other vehicle configurations may show different characteristics.

To generate Figure B.1, a large number of controllers were generated and simulated. The

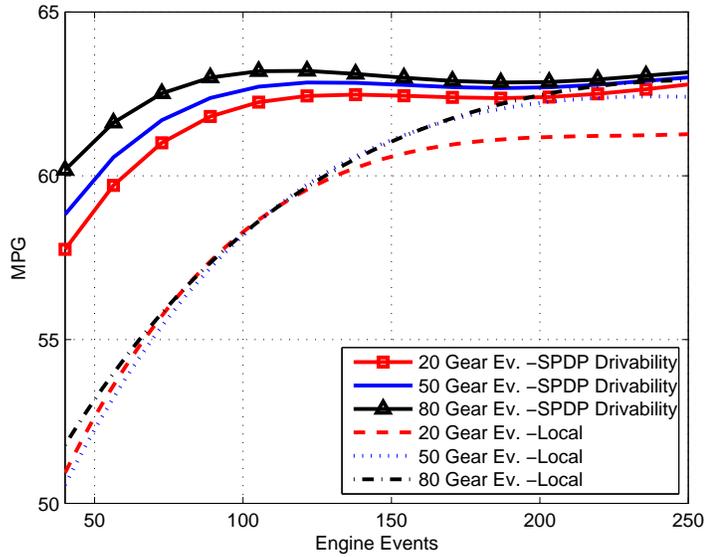
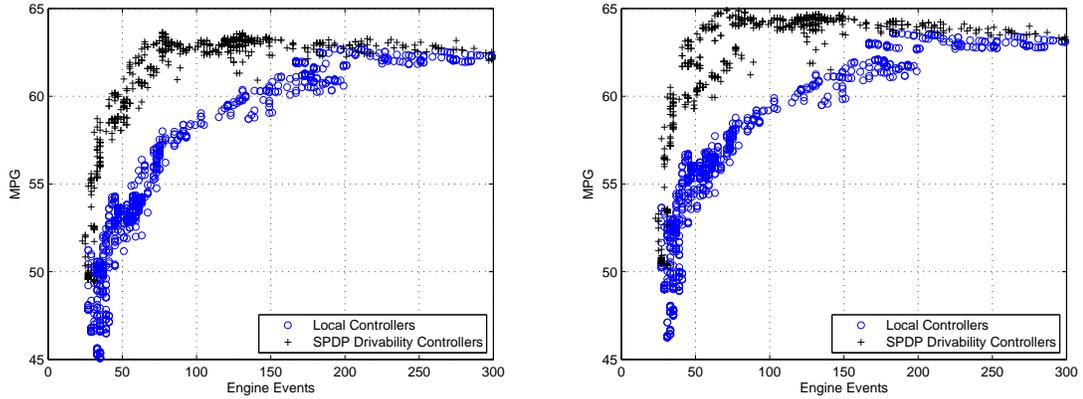


Figure B.1: Comparison of “Local” and “SPDP-Drivability” Controllers on FTP.

results were then fit with a response surface to produce the curves shown. The raw results of these simulations are shown in Figure B.2a. Note that the fuel economy is a function of both gear and engine events, so this is naturally a 3-D table. In this case, fuel economy is relatively insensitive to gear events, so the data are shown as a function of engine events only.

Due to the stochastic nature of the optimal control problem, the final SOC is not guaranteed to end at any particular value. The final SOC for these simulations is always close to the initial SOC, but there is variation. Ignoring this variation could cause false fuel economy estimates. In this case, the SPDP-Drivability controllers not only got better fuel economy, but tended to have a higher final SOC. Thus the conclusions comparing the two controller types are valid. In order to make a more reliable comparison, one must estimate the relative value of SOC deviations at the end of a trip. For this vehicle and this particular cycle, a final SOC that is 1% higher than the initial SOC roughly corresponds to a 0.3 MPG decrease in fuel economy. The final fuel economy values are corrected based on this estimate and the final SOC to yield the data shown in Figure B.2b.



(a) Fuel Economy uncorrected for final SOC.

(b) Fuel Economy adjusted based on final SOC.

Figure B.2: Comparison of “Local” and “SPDP-Drivability” Controllers on FTP

B.6 Discussion

The optimal energy management strategy for a hybrid vehicle depends on the future drive cycle. This knowledge is not available in practice, leaving a challenging control design problem. One practical and successful option is an on-line optimization to minimize a cost function that depends on the current state and control. A second, more computationally intensive method is Stochastic Dynamic Programming (SPDP), which also includes a stochastic estimate of future costs.

Drivability is an important consideration in designing a deployable controller. When using SPDP, these restrictions could be implemented in two ways. The first and easiest option is to design a SPDP controller for fuel economy only, and add an additional on-line instantaneous optimization to include drivability. This method is very similar to how one would include drivability in an Equivalent Consumption Minimization Strategy (ECMS) type controller. The second option is to design a SPDP-Drivability controller with a full set of states that include drivability, but this method requires significant additional computation. The results presented here show that using this SPDP-Drivability controller with a representative vehicle simulation yields significant (up to 10%) fuel economy improvements that can justify the increased off-line computational complexity compared to the local on-line optimization.

When designing an energy management controller, the designer has a range of choices

that trade off controller complexity for performance and functionality. Local optimization methods like ECMS are simple to design and implement, but other methods are more robust. Optimizing for fuel use with Shortest Path Dynamic Programming is useful to add robustness, but requires significantly more computation than ECMS. Including drivability in the SPDP controller imposes a factor of 10 increase in off-line complexity, but yields performance improvements on the order of 10% over the fuel-only SPDP local case under drivability constraints.

BIBLIOGRAPHY

BIBLIOGRAPHY

- [1] E. P. Agency. (2006, December) Epa regulatory announcement epa420-f-06-069. [Online]. Available: <http://www.epa.gov/fueleconomy/420f06069.pdf>
- [2] ———, “Fuel economy labeling of motor vehicles: Revisions to improve calculation of fuel economy estimates,” *Federal Register*, vol. 71, no. 278, pp. 77 872–77 969, December 2006.
- [3] O. Barbarisi, O. Barbarisi, E. Westervelt, F. Vasca, and G. Rizzoni, “Power management decoupling control for a hybrid electric vehicle,” in *Proc. and 2005 European Control Conference Decision and Control CDC-ECC '05. 44th IEEE Conference on*, E. Westervelt, Ed., 2005, pp. 2012–2017.
- [4] S. Barsali, M. Ceraolo, and A. Possenti, “Techniques to control the electricity generation in a series hybrid electrical vehicle,” *IEEE Trans. Energy Convers.*, vol. 17, no. 2, pp. 260–266, 2002.
- [5] S. Barsali, C. Miulli, and A. Possenti, “A control strategy to minimize fuel consumption of series hybrid electric vehicles,” *IEEE Trans. Energy Convers.*, vol. 19, no. 1, pp. 187–195, 2004.
- [6] C. Belton, P. Bennett, P. Burchill, D. Copp, N. Darnton, K. Butts, J. Che, B. Hieb, M. Jennings, and T. Mortimer, “A vehicle model architecture for vehicle system control design,” in *Proceedings of SAE 2003 World Congress & Exhibition.*, 2003.
- [7] D. Bertsekas, *Dynamic Programming and Stochastic Control*. Academic Press, 1976.
- [8] ———, *Dynamic Programming and Optimal Control Vol 1*. Athena Scientific, 2005.
- [9] ———, *Dynamic Programming and Optimal Control Vol 2*. Athena Scientific, 2005.
- [10] Y. Bin, Y. Li, Q. Gong, and Z.-R. Peng, “Multi-information integrated trip specific optimal power management for plug-in hybrid electric vehicles,” in *Proc. ACC '09. American Control Conference*, 2009, pp. 4607–4612.
- [11] H. Borhan, “Predictive energy management of a power-split hybrid electric vehicle,” *2009 AMERICAN CONTROL CONFERENCE, VOLS 1-9*, pp. 3970–3976, 2009.
- [12] P. Bowles, H. Peng, and X. Zhang, “Energy management in a parallel hybrid electric vehicle with a continuously variable transmission,” in *Proceedings of the American Control Conference*, 2000.

- [13] A. Boyali, M. Demirci, T. Acarman, L. Guvenc, O. Tur, H. Ucarol, B. Kiray, and E. Ozatay, "Modeling and control of a four wheel drive parallel hybrid electric vehicle," in *Proc. IEEE International Conference on Control Applications*, Oct. 2006, pp. 155–162.
- [14] M. Camara, F. Gustin, H. Gualous, and A. Berthon, "Energy management strategy for coupling supercapacitors and batteries with dc-dc converters for hybrid vehicle applications," *Power Electronics and Motion Control Conference, 2008. EPE-PEMC 2008. 13th*, pp. 1542–1547, 2008. [Online]. Available: <http://ieeexplore.ieee.org/stamp/stamp.jsp?isnumber=4635237&arnumber=4635486&punumber=4629335>
- [15] H. S. Chang, M. C. Fu, J. Hu, and S. I. Marcus, *Simulation-based Algorithms for Markov Decision Processes*. Springer-Verlag London, 2007.
- [16] S. Delprat, T. M. Guerra, G. Paganelli, J. Lauber, and M. Delhom, "Control strategy optimization for an hybrid parallel powertrain," in *Proc. American Control Conference the 2001*, vol. 2, 2001, pp. 1315–1320 vol.2.
- [17] S. Delprat, T. M. Guerra, and J. Rimaux, "Control strategies for hybrid vehicles: optimal control," in *Proc. VTC 2002-Fall Vehicular Technology Conference 2002 IEEE 56th*, vol. 3, 2002, pp. 1681–1685 vol.3.
- [18] —, "Optimal control of a parallel powertrain: from global optimization to real time control strategy," in *Proc. IEEE 55th Vehicular Technology Conference VTC Spring 2002*, vol. 4, 2002, pp. 2082–2088 vol.4.
- [19] —, "Control strategies for hybrid vehicles: synthesis and evaluation," in *Proc. VTC 2003-Fall Vehicular Technology Conference 2003 IEEE 58th*, vol. 5, 2003, pp. 3246–3250 Vol.5.
- [20] S. Delprat, J. Lauber, T. M. Guerra, and J. Rimaux, "Control of a parallel hybrid powertrain: optimal control," *IEEE Trans. Veh. Technol.*, vol. 53, no. 3, pp. 872–881, 2004.
- [21] C. Dextreit, F. Assadian, I. Kolmanovsky, J. Mahtani, and K. Burnham, "Hybrid vehicle energy management using game theory," in *Proceedings, SAE World Conference April 2008*, 2008.
- [22] Y. Ding, "Trip-based explanatory variables for estimating vehicle fuel consumption and emission rates," *URBAN AIR QUALITY - RECENT ADVANCES, PROCEEDINGS*, pp. 61–77, 2002.
- [23] E. Ericsson, "Independent driving pattern factors and their influence on fuel-use and exhaust emission factors," *Transportation Research Part D: Transport and Environment*, vol. 6, no. 5, pp. 325–345, 2001.
- [24] A. Esteves-Booth, T. Muneer, J. Kubie, and H. Kirby, "A review of vehicular emission models and driving cycles," *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, vol. 216, no. 8, pp. 777–797, 2002.
- [25] G. Fargione, G. Fargione, A. Risitano, and D. Tringali, "A soft-computing robotic shift for a hybrid vehicle," in *Proc. 10th IEEE Conference on Emerging Technologies and Factory Automation ETFA 2005*, A. Risitano, Ed., vol. 2, 2005, pp. 8 pp.–.

- [26] M. Gokasan, S. Bogosyan, and D. J. Goering, “Sliding mode based powertrain control for efficiency improvement in series hybrid-electric vehicles,” *IEEE Trans. Power Electron.*, vol. 21, no. 3, pp. 779–790, 2006.
- [27] J. Gonder, T. Markel, A. Simpson, and M. Thornton, “Using gps travel data to assess the real world driving energy use of plug-in hybrid electric vehicles (phevs),” in *Transportation Research Board (TRB) 86th Annual Meeting*, 2007.
- [28] J. Gonder, “Route-based control of hybrid electric vehicles,” in *SAE World Congress and Exhibition*, 2008, sAE Doc number: 2008-01-1315.
- [29] J. Gonder and T. Markel, “Energy management strategies for plug-in hybrid electric vehicles,” in *SAE World Congress, April 16-19, Detroit, Michigan*, 2007.
- [30] Q. Gong, Y. Li, and Z.-R. Peng, “Optimal power management of plug-in hev with intelligent transportation system,” in *Proc. ieee/asme international conference on Advanced intelligent mechatronics*, 2007, pp. 1–6.
- [31] —, “Trip based power management of plug-in hybrid electric vehicle with two-scale dynamic programming,” in *Proc. IEEE Vehicle Power and Propulsion Conference VPPC 2007*, 2007, pp. 12–19.
- [32] —, “Trip-based optimal power management of plug-in hybrid electric vehicles,” *IEEE Trans. Veh. Technol.*, vol. 57, no. 6, pp. 3393–3401, 2008.
- [33] —, “Trip based optimal power management of plug-in hybrid electric vehicles using gas-kinetic traffic flow model,” in *Proc. American Control Conference*, 2008, pp. 3225–3230.
- [34] Q. Gong, Y. Li, and Z. Peng, “Power management of plug-in hybrid electric vehicles using neural network based trip modeling,” in *Proc. ACC '09. American Control Conference*, 2009, pp. 4601–4606.
- [35] L. Graham, M. Christenson, and D. Karman, “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., 2006, pp. 1–6.
- [36] J. Hansen, M. Winther, and S. Sorenson, “The influence of driving patterns on petrol passenger car emissions,” *The Science of the Total Environment*, vol. 169, pp. 129–139, 1995.
- [37] B. Holmen and D. Niemeier, “Characterizing the effects of driver variability on real-world vehicle emissions,” *Transportation Research Part D: Transport and Environment*, vol. 3, no. 2, pp. 117–128, 1998.
- [38] N. Jalil, N. Jalil, N. Kheir, and M. Salman, “A rule-based energy management strategy for a series hybrid vehicle,” in *Proc. American Control Conference the 1997*, N. Kheir, Ed., vol. 1, 1997, pp. 689–693 vol.1.
- [39] L. Johannesson, M. Asbogard, and B. Egardt, “Assessing the potential of predictive control for hybrid vehicle powertrains using stochastic dynamic programming,” in *Proc. IEEE Intelligent Transportation Systems*, 2005, pp. 366–371.

- [40] —, “Assessing the potential of predictive control for hybrid vehicle powertrains using stochastic dynamic programming,” *IEEE Trans. Intell. Transp. Syst.*, vol. 8, no. 1, pp. 71–83, 2007.
- [41] S. Kamble, T. Mathew, and G. Sharma, “Development of real-world driving cycle: Case study of pune, india,” *Transportation Research, Part D: Transport and Environment*, no. 2, pp. 132–, 2009.
- [42] S. Kermani, S. Delprat, T. M. Guerra, and R. Trigui, “Predictive control for hev energy management: experimental results,” in *Proc. IEEE Vehicle Power and Propulsion Conference VPPC '09*, 2009, pp. 364–369.
- [43] S. Kermani, S. Delprat, R. Trigui, and T. M. Guerra, “Predictive energy management of hybrid vehicle,” in *Proc. IEEE Vehicle Power and Propulsion Conference VPPC '08*, 2008, pp. 1–6.
- [44] A. Kleimaier, A. Kleimaier, and D. Schroder, “Optimization strategy for design and control of a hybrid vehicle,” in *Proc. 6th International Workshop on Advanced Motion Control*, D. Schroder, Ed., 2000, pp. 459–464.
- [45] M. A. Kluger and D. M. Long, “An overview of current automatic, manual, and continuously variable transmission efficiencies and their projected future improvements,” in *Proceedings, SAE International Congress and Exposition*, 1999.
- [46] I. Kolmanovsky, I. Siverguina, and B. Lygoe, “Optimization of powertrain operating policy for feasibility assessment and calibration: stochastic dynamic programming approach,” in *Proceedings of the American Control Conference.*, vol. 2, 2002.
- [47] A. V. Kulkarni and R. R. Sapre, *Gps-Based Methodology for Drive Cycle Determination*. Society of Automotive Engineers, 400 Commonwealth Dr , Warrendale, PA, 15096, USA, 2005.
- [48] R. Langari and J.-S. Won, “Integrated drive cycle analysis for fuzzy logic based energy management in hybrid vehicles,” in *Proc. 12th IEEE International Conference on Fuzzy Systems FUZZ '03*, vol. 1, 2003, pp. 290–295 vol.1.
- [49] —, “Intelligent energy management agent for a parallel hybrid vehicle-part i: system architecture and design of the driving situation identification process,” *IEEE Trans. Veh. Technol.*, vol. 54, no. 3, pp. 925–934, 2005.
- [50] H. Larsson and E. Ericsson, “The effects of an acceleration advisory tool in vehicles for reduced fuel consumption and emissions,” *Transportation Research, Part D: Transport and Environment*, no. 2, pp. 141–, 2009.
- [51] D. LeBlanc, J. Sayer, C. Winkler, R. Ervin, S. Bogard, J. Devonshire, M. Mefford, M. Hagan, Z. Bareket, R. Goodsell, and T. Gordon, “Road departure crash warning system field operational test: Methodology and results.” University of Michigan Transportation Research Institute, Tech. Rep. UMTRI-2006-9-1, June 2006, http://www-nrd.nhtsa.dot.gov/pdf/nrd-12/RDCW-Final-Report-Vol.1_JUNE.pdf.
- [52] W. Li, “Dynamic energy management for hybrid electric vehicle based on adaptive dynamic programming,” *2008 IEEE INTERNATIONAL CONFERENCE ON INDUSTRIAL TECHNOLOGY, VOLS 1-5*, pp. 699–704, 2008.

- [53] —, “Dynamic energy management for hybrid electric vehicle based on approximate dynamic programming,” *2008 7TH WORLD CONGRESS ON INTELLIGENT CONTROL AND AUTOMATION, VOLS 1-23*, pp. 7864–7869, 2008.
- [54] C.-C. Lin, S. Jeon, H. Peng, and J. Lee, “Driving pattern recognition for control of hybrid electric trucks,” *International Journal of Vehicle Mechanics and Mobility*, vol. 42, no. 1-2, pp. 41–58, 2004.
- [55] C.-C. Lin, H. Peng, S. Jeon, and J. Lee, “Control of a hybrid electric truck based on driving pattern recognition,” in *Proceedings of the 2002 Advanced Vehicle Control Conference*, 2002.
- [56] C.-C. Lin, “Modeling and control strategy development for hybrid vehicles,” Ph.D. dissertation, University of Michigan, 2004.
- [57] C.-C. Lin, J.-M. Kang, J. Grizzle, and H. Peng, “Energy management strategy for a parallel hybrid electric truck,” in *Proc. American Control Conference*, vol. 4, 25–27 June 2001, pp. 2878–2883.
- [58] C.-C. Lin, H. Peng, and J. Grizzle, “Power management strategy for a parallel hybrid electric truck,” in *Proceedings of the 2002 Mediterranean Control Conference*, 2002.
- [59] —, “A stochastic control strategy for hybrid electric vehicles,” in *Proceedings of the American Control Conference*, 2004.
- [60] C.-C. Lin, H. Peng, J. Grizzle, and J.-M. Kang, “Power management strategy for a parallel hybrid electric truck,” *IEEE Transactions on Control Systems Technology*, vol. 11, no. 6, pp. 839–849, 2003.
- [61] C.-C. Lin, H. Peng, J. Grizzle, J. Liu, and M. Busdiecker, “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*, 2003.
- [62] C.-C. Lin, Y. Wang, Z. S. Filipi, L. Louca, H. Peng, and D. N. Assanis, *Integrated, Feed-Forward Hybrid Electric Vehicle Simulation in Simulink and Its Use for Power Management Studies*. Society of Automotive Engineers, 400 Commonwealth Dr , Warrendale, PA, 15096, USA, 2001.
- [63] J. Liu, J. Liu, and H. Peng, “Control optimization for a power-split hybrid vehicle,” in *Proc. American Control Conference*, H. Peng, Ed., 2006, pp. 6 pp.–.
- [64] J. Liu and H. Peng, “Modeling and control of a power-split hybrid vehicle,” *IEEE Trans. Control Syst. Technol.*, vol. 16, no. 6, pp. 1242–1251, 2008.
- [65] A. Melero-Perez, W. Gao, and J. Fernandez-Lozano, “Fuzzy logic energy management strategy for fuel cell/ultracapacitor/battery hybrid vehicle with multiple-input dc/dc converter,” *Vehicle Power and Propulsion Conference, 2009. VPPC '09. IEEE*, pp. 199–206, 2009.
- [66] M. Mohebbi, M. Mohebbi, M. Charkhgard, and M. Farrokhi, “Optimal neuro-fuzzy control of parallel hybrid electric vehicles,” in *Proc. IEEE Conference Vehicle Power and Propulsion*, M. Charkhgard, Ed., 2005, pp. 26–30.

- [67] C. Musardo, G. Rizzoni, and B. Staccia, "A-ECMS: An adaptive algorithm for hybrid electric vehicle energy management," in *Proceedings of the European Control Conference Decision and Control CDC-ECC.*, 2005.
- [68] C. Musardo, B. Staccia, S. Midlam-Mohler, Y. Guezennec, and G. Rizzoni, "Supervisory control for no/sub x/ reduction of an hev with a mixed-mode hcci/cidi engine," in *Proc. American Control Conference the 2005*, B. Staccia, Ed., 2005, pp. 3877–3881 vol. 6.
- [69] K. S. Nesamani and K. P. Subramanian, "Impact of real-world driving characteristics on vehicular emissions," *JSME International Journal, Series B: Fluids & Thermal Engineering*, no. 1, pp. 19–, 2006.
- [70] D. Opila, D. Aswani, R. McGee, J. Cook, and J. Grizzle, "Incorporating drivability metrics into optimal energy management strategies for hybrid vehicles," in *Proceedings of 2008 IEEE Conference on Decision and Control*, 2008.
- [71] D. Opila, X. Wang, R. McGee, J. Cook, and J. Grizzle, "Fundamental structural limitations of an industrial energy management controller architecture for hybrid vehicles," in *ASME Dynamic Systems and Control Conference*, 2009.
- [72] —, "Performance comparison of hybrid vehicle energy management controllers on real-world drive cycle data," in *Proceedings of the American Control Conference*, 2009.
- [73] D. Opila, X. Wang, R. McGee, R. Gillespie, J. Cook, and J. Grizzle, "Incorporating drivability metrics into optimal energy management strategies for hybrid vehicles Part 1: Model, methods, and government drive cycles," *Submitted to IEEE Transactions on Control Systems Technology*, 2010.
- [74] —, "Incorporating drivability metrics into optimal energy management strategies for hybrid vehicles Part 2: Validation and real-world robustness," *Submitted to IEEE Transactions on Control Systems Technology*, 2010.
- [75] G. Paganelli, S. Delprat, T. Guerra, J. Rimaux, and J. Santin, "Equivalent consumption minimization strategy for parallel hybrid powertrains," in *Proc. IEEE 55th Vehicular Technology Conference VTC Spring 2002*, S. Delprat, Ed., vol. 4, 2002, pp. 2076–2081 vol.4.
- [76] G. Paganelli, T. M. Guerra, S. Delprat, J.-J. Santin, M. Delhom, and E. Combes, "Simulation and assessment of power control strategies for a parallel hybrid car," *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, vol. 214, pp. 705–717, 2000.
- [77] G. Paganelli, M. Tateno, A. Brahma, G. Rizzoni, and Y. Guezennec, "Control development for a hybrid-electric sport-utility vehicle: strategy, implementation and field test results," in *Proc. American Control Conference*, vol. 6, 2001.
- [78] G. Paganelli, G. Ercole, A. Brahma, Y. Guezennec, and G. Rizzoni, "General supervisory control policy for the energy optimization of charge-sustaining hybrid electric vehicles," *JSAE Review*, vol. 22, no. 4, pp. 511–518, Oct. 2001.

- [79] L. V. Perez, "Optimization of power management in an hybrid electric vehicle using dynamic programming," *Mathematics and Computers in Simulation*, vol. 73, no. 1-4 SPEC. ISS., pp. 244–254, 2006.
- [80] L. Perez, G. Bossio, D. Moitre, and G. Garcia, "Supervisory control of an hev using an inventory control approach," *Latin American Applied Research*, vol. 36, no. 2, pp. 93–100, 2006.
- [81] L. Perez and E. Pilotta, "Optimal power split in a hybrid electric vehicle using direct transcription of an optimal control problem," *Mathematics and Computers in Simulation*, vol. 79, no. 6, pp. 1959–1970, 2009.
- [82] R. Piffner, "Fuel-optimal control of cvt powertrains," *Control Engineering Practice*, vol. 11, no. 3, pp. 329–336, 2003, iD: EVII; ID: Engineering Index (Compendex).
- [83] P. Pisu, K. Koprubasi, and G. Rizzoni, "Energy management and drivability control problems for hybrid electric vehicles," in *Proceedings of the European Control Conference Decision and Control CDC-ECC.*, 2005.
- [84] P. Pisu and G. Rizzoni, "A comparative study of supervisory control strategies for hybrid electric vehicles," *IEEE Trans. Control Syst. Technol.*, vol. 15, no. 3, pp. 506–518, 2007.
- [85] D. Prokhorov, "Training recurrent neurocontrollers for real-time applications," *IEEE Trans. Neural Netw.*, vol. 18, no. 4, pp. 1003–1015, July 2007.
- [86] M. Ross, "Automobile fuel consumption and emissions. effects of vehicle and driving characteristics," *Annual Review of Energy and the Environment*, vol. 19, pp. –, 1994.
- [87] S. Sager, C. Kirches, and H. G. Bock, "Fast solution of periodic optimal control problems in automobile test-driving with gear shifts," in *Proc. 47th IEEE Conference on Decision and Control CDC 2008*, 2008, pp. 1563–1568.
- [88] N. Schouten, N. Schouten, M. Salman, and N. Kheir, "Fuzzy logic control for parallel hybrid vehicles," *IEEE Trans. Control Syst. Technol.*, vol. 10, no. 3, pp. 460–468, 2002.
- [89] A. Sciarretta, M. Back, and L. Guzzella, "Optimal control of parallel hybrid electric vehicles," *IEEE Trans. Control Syst. Technol.*, vol. 12, no. 3, pp. 352–363, 2004.
- [90] A. Sciarretta and L. Guzzella, "Control of hybrid electric vehicles," *IEEE Control Systems Magazine*, vol. 27, no. 2, pp. 60–70, 2007.
- [91] L. Serrao, S. Onori, and G. Rizzoni, "Ecms as a realization of pontryagin's minimum principle for hev control," St. Louis, MO, United states, 2009, pp. 3964 – 3969, hybrid electric vehicle;Pontryagin's minimum principles;Simulation result;. [Online]. Available: <http://dx.doi.org/10.1109/ACC.2009.5160628>
- [92] G. Shi, Y. Jing, A. Xu, and J. Ma, "Study and simulation of based-fuzzy-logic parallel hybrid electric vehicles control strategy," in *Proc. Sixth International Conference on Intelligent Systems Design and Applications ISDA '06*, vol. 1, Oct. 2006, pp. 280–284.

- [93] A. Sjodin and M. Lenner, "On-road measurements of single vehicle pollutant emissions, speed and acceleration for large fleets of vehicles in different traffic environments," *The Science of the Total Environment. Vol. 169*, vol. 169, pp. –, 1995.
- [94] E. Tate, J. Grizzle, and H. Peng, "Shortest path stochastic control for hybrid electric vehicles," *International Journal of Robust and Nonlinear Control*, vol. 18, pp. 1409–1429, 2008.
- [95] J. Tate, Edward Dean, "Techniques for hybrid electric vehicle controller synthesis," Ph.D. dissertation, University of Michigan, 2006.
- [96] X. Wei, "Dynamic modeling of a hybrid electric drivetrain for fuel economy, performance and driveability evaluations," *American Society of Mechanical Engineers, Dynamic Systems and Control Division (Publication) DSC*, vol. 72, no. 1, pp. 443–450, 2003.
- [97] J.-S. Won and R. Langari, "Intelligent energy management agent for a parallel hybrid vehicle-part ii: torque distribution, charge sustenance strategies, and performance results," *IEEE Trans. Veh. Technol.*, vol. 54, no. 3, pp. 935–953, 2005.
- [98] J.-S. Won, R. Langari, and M. Ehsani, "An energy management and charge sustaining strategy for a parallel hybrid vehicle with cvt," *IEEE Trans. Control Syst. Technol.*, vol. 13, no. 2, pp. 313–320, 2005.
- [99] J.-S. Won and R. Langari, "Intelligent energy management agent for a parallel hybrid vehicle," in *Proc. American Control Conference the 2003*, vol. 3, 2003, pp. 2560–2565 vol.3.
- [100] B. Wu, C.-C. Lin, Z. Filipi, H. Peng, and D. Assanis, "Optimal power management for a hydraulic hybrid delivery truck," *Vehicle System Dynamics*, vol. 42, no. 1-2, pp. 23–40, 2004.
- [101] ———, "Optimization of power management strategies for a hydraulic hybrid medium truck," in *Proceedings of the 2002 Advanced Vehicle Control Conference*, 2002.
- [102] D. Zhang, "A study on fuzzy control of energy management system in hybrid electric vehicle," *2009 ASIA-PACIFIC POWER AND ENERGY ENGINEERING CONFERENCE (APPEEC), VOLS 1-7*, pp. 3033–3036, 2009.
- [103] Y. Zhu, Y. Chen, G. Tian, H. Wu, and Q. Chen, "A four-step method to design an energy management strategy for hybrid vehicles," *American Control Conference, 2004. Proceedings of the 2004*, vol. 1, pp. 156–161 vol.1, 2004.