AN INTEGRATED SOFTWARE ENVIRONMENT FOR POWERTRAIN FEASIBILITY ASSESSMENT USING OPTIMIZATION AND OPTIMAL CONTROL

Ilya V. Kolmanovsky, Jing Sun, and Shiva N. Sivashankar

Asian Journal of Control
Vol. 8, No. 3, pp. 199-209
September 2006

AN INTEGRATED SOFTWARE ENVIRONMENT FOR POWERTRAIN FEASIBILITY ASSESSMENT USING OPTIMIZATION AND OPTIMAL CONTROL

Ilya V. Kolmanovsky, Jing Sun, and Shiva N. Sivashankar

ABSTRACT

With the increase in automotive powertrain complexity, an upfront assessment of powertrain capability in meeting its design targets is important early on in the development programs. The optimization of control policy based on powertrain simulation models can facilitate this assessment and establish limits of achievable performance for a given powertrain configuration and parameters. The paper discusses several computational optimization and user interface solutions for deploying a numerical optimal control approach in a user-friendly software environment.

KeyWords: Powertrain control, powertrain assessment, optimal control, optimization, dynamic programming.

I. INTRODUCTION

An assessment of powertrain capability to meet its design targets and constraints (referred afterwards as feasibility assessment) is important early on in powertrain development programs. The targets and constraints may be prescribed for different powertrain attributes such as fuel economy, emissions, driveability, NVH, performance, cost, *etc*. They may also be prescribed for individual subsystems and components of the powertrain, depending on the scope of the development.

Both powertrain design parameters and its control policy influence the capability of powertrain to meet the prescribed targets and constraints. New powertrain systems, in particular, are characterized by an ever increasing number of control inputs as well as by strong static and dynamic interactions. Thus a synergistic treatment of design and control issues is especially important for the analysis of feasibility of such powertrains and in order to fully identify relevant design and control requirements.

Manuscript received May 3, 2005; accepted December 1, 2005.

I.V. Kolmanovsky is with the Ford Motor Company, Dearborn, MI 48124 (e-mail: ikolmano@ford.com).

J. Sun is with the Department of Naval Architecture and Marine Engineering, the University of Michigan, Ann Arbor, MI 48109 (e-mail: jingsun@umich.edu).

S.N. Sivashankar is with the Emmeskay, Inc., Plymouth, MI 48170 (e-mail: shiva@emmeskay.com).

The feasibility of given values of powertrain design parameters can be assessed on the basis of the best achievable powertrain performance, *i.e.*, powertrain performance while operating under an optimal control policy. Due to multitude of objectives that powertrain systems must satisfy, a multi-objective optimization problem must be addressed. Thus the best achievable performance is characterized by a Pareto front, and the feasibility of the powertrain means that a subset of the Pareto front is located within the target set. See Fig. 1 for an example. Different powertrain configurations or parameter values can be compared with each other based on selected points on the Pareto fronts (*e.g.*, on the basis of best emission constrained fuel economy). By examining the behavior of trajectories corresponding to these selected points, subsystem level targets can be set and an insight into required control system architecture, actuator coordination and calibration can be gained.

In the early phases of the development programs, no hardware prototypes may yet exist or the technology may still be evolving. In this situation, the feasibility assessment can only be based on dynamic simulation models.[1] The Pareto front can be computed via the numerical optimization applied to these models. Specifically, a weighted sum of the objectives can be minimized with respect to the control input trajectory for different values of the weights.

[1] The availability of suitable simulation models which can adequately predict the new powertrain performance has been assumed throughout.

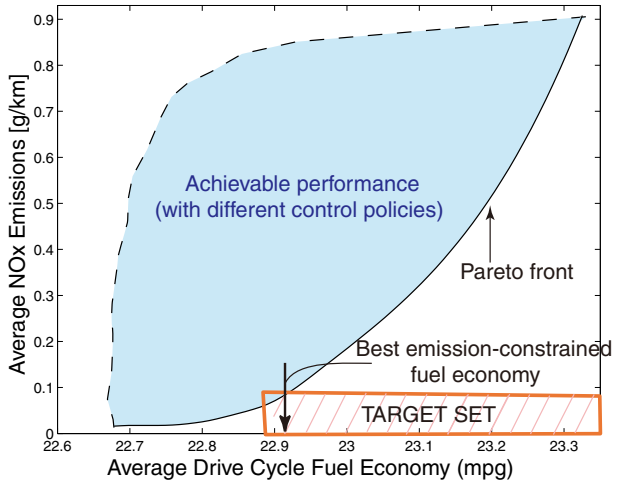

Fig. 1. Feasibility assessment: Achievable performance, best achievable performance and target performance.

A less computationally demanding approach is to only optimize certain parameters in an *a priori* defined parameter-dependent control policy. In the latter case, engineering judgement is necessary to define upfront a parameter-dependent control policy that is most likely to meet the targets. Also at times practical constraints make it not possible or desirable to change the control policy completely except for the values of a few parameters. In this case, the values of these parameters can be determined using parameter optimization while a complete optimal control policy can be used to understand the potential for further improvements.

To carry out the feasibility analysis, the powertrain system community needs effective tools that can integrate the modeling, optimization, and design sensitivity analysis in a seamless fashion. For conventional powertrain systems, such assessment can be successfully carried out with quasi-static optimization [1-6], which ignored the dynamics and focused on the static relations of the performance to control variables and design parameters. When the dynamics are dominant, such as for the engine cold start problem, dynamic optimization is still manageable when the time horizon is short [7,8]. The introduction of energy and storage elements, such as the battery for hybrid vehicles and lean NOx trap for lean burn engines, has however necessitated a new optimization paradigm, where the dynamics optimization is essential to address the performance trade-offs over various driving cycles. Several recent publications have demonstrated the successful applications of deterministic and stochastic dynamic programming [9-13], linear programming [14], and computational game theory [15]. These efforts have laid the foundation for establishing a model-based and optimization-enabled software environment for carrying out powertrain feasibility assessment.

In this paper we discuss the background and some of the features of a recently developed software environment for performing powertrain feasibility analysis. The software implements both the optimization based on dynamic programming [16] and the optimization of parameters in parameter-dependent control policies. It provides automotive engineers, who may not be experts in optimization, with the capability to:

- interface with powertrain models in MATLAB/Simulink;[2]
- define and solve state and control constrained optimal control problem via dynamic programming while taking advantage of fast/slow dynamics decomposition to deal with the "curse of dimensionality";
- define parameter-dependent operating policies and perform parameter optimization;
- visualize solutions in different ways and analyze their sensitivity;
- automatically generate analysis reports;
- speed-up the optimization and parameter sweeps by taking advantage of parallel computations.

II. IMPLEMENTATION OF DYNAMIC PROGRAMMING

The optimal powertrain control policies are computed using the dynamic programming. It is applied to discrete-time system models decomposed into a cascade of a static nonlinear subsystem and a dynamic nonlinear subsystem,

$$x(t+1) = f_d(x(t), u_d(t), w(t), v(t), p),$$
$$v(t) = f_s(w(t), u_s(t), x(t), p), \quad (1)$$

where x is the state of the dynamic subsystem, u_d is the control input of the dynamic subsystem, $w(t)$ is the vector of known operating variables of the powertrain (such as engine speed and engine torque or wheel speed and wheel torque), p is the vector of powertrain design parameters, $v(t)$ is the output of the static subsystem, and $u_s(t)$ is the control input of the static subsystem. The decomposition (1) can be viewed as an approximation of a system with slow and fast dynamics. Such a time scale separation is characteristic of and occurs quite frequently in powertrain applications. The fast dynamics can be approximated by the static subsystem for v while the slow dynamics by the dynamic subsystem for x. Since dynamic programming based optimization becomes computationally demanding as the state dimension increases, the main advantage of the approximation (1) is keeping the effective state dimension low so that the attention can be focused on the behavior of key states of the powertrain.

For example, in the case study in [9], a Direct Injection Spark Ignition (DISI) engine behavior was approximated as a static subsystem so that $w(t)$ is the vector of

[2] MATLAB and Simulink are registered trademarks of the MathWorks, Inc. of Natick, MA.

engine speed and engine torque at time t; $v(t)$ is the vector of exhaust air-to-fuel ratio, exhaust temperature, oxides of nitrogen feedgas flow rate, hydrocarbon feedgas flow rate, carbon monoxide feedgas flow rate, exhaust mass flow rate and fueling rate at time t; and $u_s(t)$ is the vector of in-cylinder air-to-fuel ratio, exhaust gas recirculation (EGR) rate, spark timing, injection timing, fuel rail pressure, swirl control valve setting, and fueling rate at time t. The vector u_d is empty in this case. The dynamic subsystem in [9] represented the behavior of the aftertreatment system and the state x was a vector of stored oxides of nitrogen and oxygen in aftertreatment system, and aftertreatment system temperature. See Fig. 2. The parameters p define the properties of the aftertreatment system such as the NOx and oxygen storage capacities. In that case parametric models describing the sensitivity of performance variables existed and had been incorporated.

Another computational simplification is to consider only a finite set of possible control policies for the static subsystem so that

$$u_s = g(w, x, m), \quad m = 1, \cdots, M, \tag{2}$$

where m is an integer which identifies the control policy out of M total control policies. Then the dynamic programming can be applied to the optimization of the control input

$$u = \begin{bmatrix} u_d \\ m \end{bmatrix}. \tag{3}$$

This procedure results in significant computational savings in those cases when the dimensionality of u_s is large. A finite set of representative control policies can be generated in a number of different ways, depending on the application. In [9], following the approach of [17], a weighted sum of engine fuel consumption and emissions was optimized at each engine speed and engine torque operating point, w, for a finite set of weight combinations and for two different combustion regimes (stratified and homogeneous); each combination of a weight and a combustion regime would then define a separate policy for the static subsystem (engine). In effect, in that application m may be viewed as an identifier for one of a finite number of engine maps optimized for different trade-off between fuel consumption and emissions. See Fig. 3.

With (2) and (3), the system (1) can be written in the form

$$x(t+1) = f(x(t), u(t), w(t), p). \tag{4}$$

The optimization objective is to minimize a stage-additive cost functional,

$$\sum_{t=0}^{t=T} J(x(t), u(t), w(t), p) \to \min,$$

where the incremental cost, $J(x, u, w, p) = \lambda^T y$, is a weighted sum of trade-off variables,

$$y(t) = h_y(x(t), u(t), w(t), p), \tag{5}$$

and where λ is the vector of the weights. The trade-off variables may include fuel consumption and tailpipe emissions. The software provides flexibility to specify whether the weighting factors are applied at all time instants, at a final time instant or only at certain specified time instants. It can also select the weights automatically. The optimization is repeated for a specified sweep of the powertrain parameters, p.

The dynamic programming relies on backward-in-time value function iterations. Specifically, let $V(t, x, p)$ be the cost-to-go from the time instant t and state x to the end of the optimization horizon, $t = T$. Suppose $V(t+1, \cdot, p)$ has been already determined and let

$$Q(t, x, u, p) = J(x, u, w(t), p) + V(t+1, f(x, u, w(t), p), p). \tag{6}$$

Then the state and time-dependent optimal control at time t satisfies

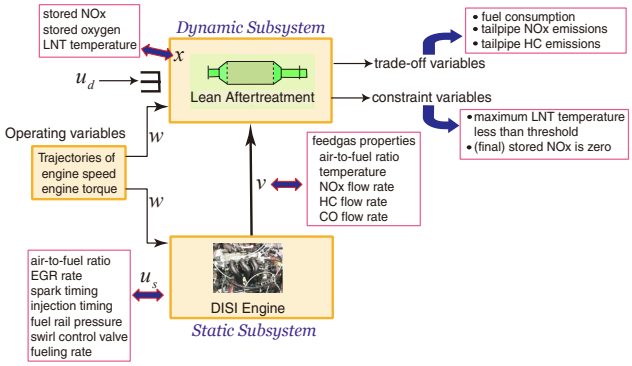

Fig. 2. Approximation of DISI engine and aftertreatment system behavior as a cascade of a static subsystem and a dynamic subsystem.

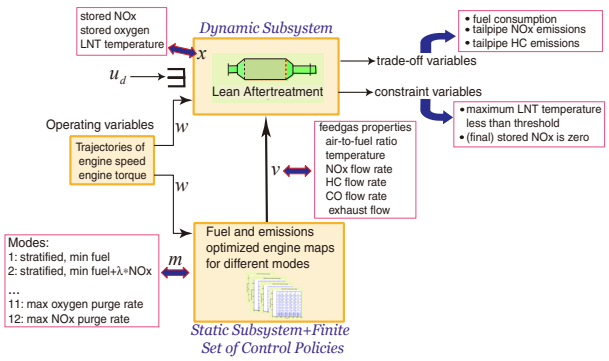

Fig. 3. Restricting static subsystem to a finite set of control policies.

$$u^*(t, x, p) \in \min_u Q(t, x, u, p), \tag{7}$$

and $V(t, x, p)$ is determined as

$$V(t, x, p) = Q(x, t, u^*(t, x, p), p). \tag{8}$$

To perform iterations (6)-(8) numerically, a grid of values of $x(t)$, $u(t)$ and $w(t)$ are chosen. The grids for $x(t)$ are defined via the State Grid Set, X_{ij}, and State Grid Sequence Function, S_X. Specifically, X_{ij}, $i = 1, \cdots, I_x$, is the ith element of the *State Grid Set* for the jth component of the vector $x(t)$; $S_X(t) = i$ implies that the jth component of the vector $x(t)$ assumes values from the grid X_{ij}. The grids for $u(t)$ are defined analogously with the help of *Control Sets*, U_i and a Control Set Sequence function, S_U. Specifically, U_{ij}, $i = 1, \cdots, I_U$, $j = 1, \cdots, K(i)$, is the jth vector in the ith Control Set, U_i; $S_U(t) = i$ implies that $u(t) \in \{U_{i1}, U_{i2}, \cdots, U_{iK(i)}\}$ (the number of vectors, $K(i)$, may be different for different i). Finally, the Operating Variable Set, W, and Operating Variable Sequence function, S_W, are introduced so that W_i, $i = 1, \cdots, I_W$, is the ith vector in the Operating Variable Set, W, and $S_W(t) = i$ implies that at time t, $w(t) = W_i$.

If the user desires to use the same state grids for all t, then $I_X = 1$ and $S_X(t) = 1$; in this case the user can specify the grid for each state variable as a vector using the Regular Grid option, see Fig. 4. The software can also pick the regular grids automatically. The Advanced Grid option is used if $I_X > 1$, and in this case the State Grid Set and State Grid Sequence function are loaded from MATLAB workspace or from a file. The latter option can be used to implement iterative dynamic programming whereby the state grid at each time instant is centered around the optimal state trajectory from the previous iteration and reduced in size with each new iteration (the total number of the grid points remains the same for each iteration). The iterative dynamic programming may be used when the state dimension is high and the use of conventional dynamic programming is computationally prohibitive.[3]

The Control Sets, U_i, and Control Set Sequence function, SU, may be specified by the user in MATLAB workspace or in a file. They can also be generated as a finite set of control policies (2), (3) with the help of the special static optimization environment applied to the static subsystem in (1). The static optimization environment is supported by its own set of integrated graphic user interfaces, see Fig. 5. The user can define multiple operating modes, where each mode may have its own objective function, equality and inequality constraints, and ranges for input variables. The objective function for each mode is specified as a weighted sum of the trade-off variables for this mode and is minimized with respect to u_s for different values of the weights at different operating points (defined by w). Each m in (2) then corresponds to a particular combination of an operating mode and a weight in the static optimization environment. The static optimization environment can be applied either to MATLAB/Simulink models of the static subsystem or to a static subsystem response data set (*e.g.*, engine mapping data). A special interpolation procedure was implemented to enable the latter option. See the Appendix. The Control Set reduction functionality performs a pairwise comparison of the vectors in each of U_i, and eliminates one in the pair of two if they deviate from each other by less than the specified tolerance.

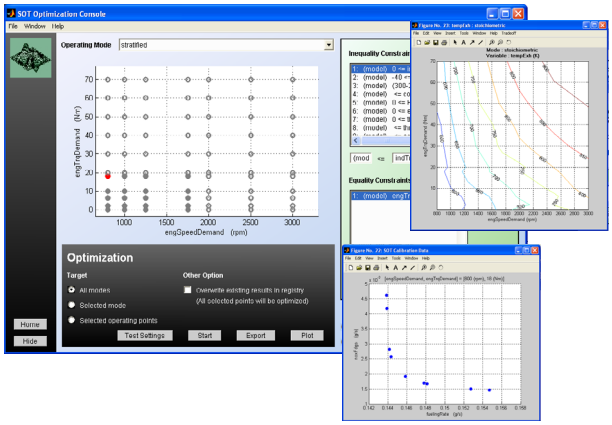

Fig. 4. Optimization console of the static optimization environment shows the status of static optimization at different operating points (w) after its completion. If circle is 'filled' then a feasible solution has been found, otherwise it is 'hollow'. The user can analyze the optimal responses using trade-off, contour and surface plots which can be open by clicking on operating points (examples shown).

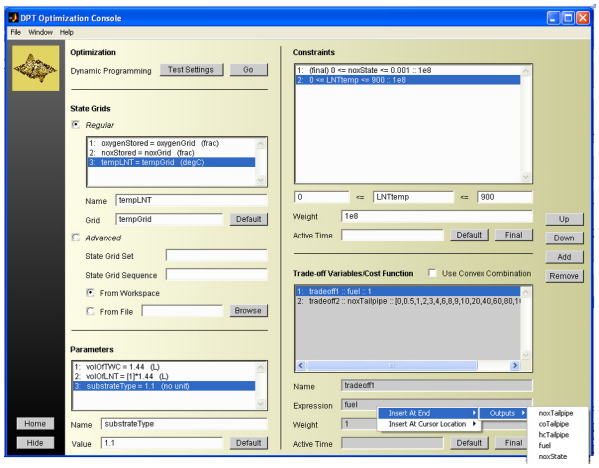

Fig. 5. The appearance of Graphic User Inteface for running dynamic programming. The insertion menu at the bottom is opened with a right mouse click and enables the user to enter a previously defined variable name into a trade-off variable expression.

[3] The resulting optimal control will, however, be specific to the selected initial condition.

The Operating Variable Set, W, and Operating Variable Sequence function, S_W, can be either directly specified by the user (in MATLAB workspace or in a file) or they may be generated automatically based on the built-in powertrain and drive cycle libraries. In the latter case, the user selects a vehicle type and a drive cycle type. Similarly to Control Set reduction, the Operating Variable Set reduction functionality can be applied to reduce the number of elements in W based on the user-defined tolerances.

To improve the computational efficiency, the output map, $h_y(x, u, w, p)$ and the state update map $f(x, u, w, p)$ are first pre-computed for all values of x, u, and w in the specified State Grid Set, Control Sets and Operating Variable Sets. Then the calculation of $J = \lambda^T y$ and the updates (6)-(8) are made in the vectorized form which can be efficiently and rapidly[4] handled by MATLAB [9]. The user is provided with several options to trade-off available RAM versus the speed of computations. If memory is limited, the pre-computed output map and state update map can be stored in multiple files and are loaded into the memory on "as needed" basis.

The pointwise-in-time constraints, $z_{\min} \le z(t) = h_z(x(t), u(t), w(t), p) \le z_{\max}$, are handled by augmenting a penalty term,

$$\sum_{t=1}^{t=T} \lambda_z^T \, \phi(z(t)) \,,$$

to the cost function. Here ϕ is a built-in penalty function applied to each component of z, which is non-zero only if $z(t)$ violates the constraints while λ_z is the vector of weights. The software provides flexibility to handle constraints that are either persistent (*i.e.*, applied at all time instants), final (applied only at the final time instant) or intermittent (applied only at specified time instants). It prebuilds the tables for z in the same way as for y.

The software environment leads the user through a series of integrated Graphical User Interfaces (GUIs) that load the model and model information file; define State Grid Set, Control Sets and Operating Variable Set as described above; define objective function, constraints and desired parameter sweeps; and, finally, provide means to visualize and analyze the results in numerous ways. One of the user interfaces for running the dynamic programming is illustrated in Fig. 5. Prior to running the dynamic programming (which is a computationally intensive and time-consuming task) the user can quickly test settings to check for errors and warnings; in particular, the software checks to ensure that it can calculate all expressions that the user has defined. The user-specified initialization and termination scripts are executed before the start of the optimization and after its completion; the initialization script may also be specified to run after each parameter or weight run.

[4] This property helps reduce computing times in a situation when new weights are added iteratively based on the analysis of past dynamic programming results.

Once the dynamic programming phase has been completed, the user can analyze and visualize optimal trajectories in numerous ways. Since the dynamic programming provides an optimal control function, $u^*(t, x)$, the optimal trajectory corresponding to any initial condition can be generated rapidly via simulations of the model (either of the same model as was used for optimization or, if necessary, of a higher fidelity model). Figure 6 shows a trade-off plot where by clicking on any point the user can plot the trajectories of optimal inputs, outputs, and states corresponding to that point. A smoothing utility permits the user to graphically change the control trajectory (*e.g.*, smooth it out if it shows excessive chattering) and compare the resulting cost and constraint violation with the original optimal solution. The original solution is referred to as Parent while the solutions derived via this smoothing are referred to as Children; the software keeps track of Children and graphically shows them on the trade-off plot connected to Parent via dashed lines. The smoothing can be particular effective when it is applied to the mode m in (3). Since sometimes optimal solutions may appear non-intuitive, smoothing provides a mechanism for checking if certain features of optimal trajectories are really required, or if

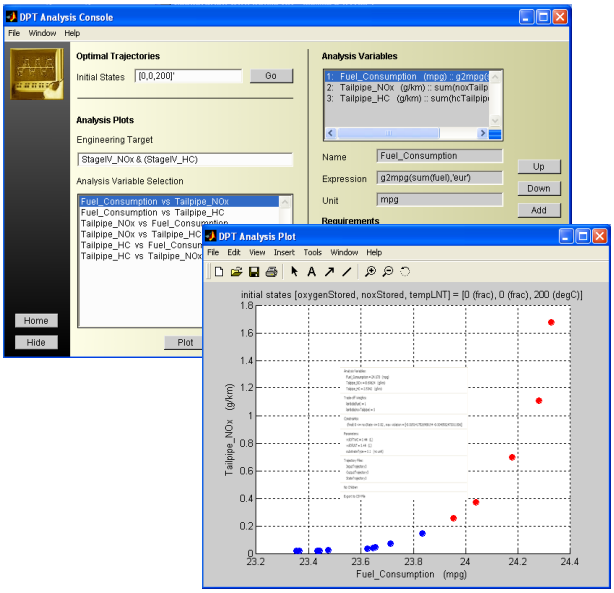

Fig. 6. The appearance of trade-off plots and trajectory analysis GUI. Analysis variables used to form the trade-off plots can be defined from trajectory data using valid MATLAB expressions and functions. The points are color-coded and shape-coded depending on whether they satisfy the specified requirements and pointwise-in-time constraints. The user can left click on individual points to access the Trajectory Plotting GUI, Trajectory Comparison GUI and Smoothing Utility GUI or right-click to get a summary information about the solution (shown).

they may be caused by model irregularities. The software implements a database to avoid recomputing the optimal dynamic programming solutions or simulated trajectories if these solutions or trajectories have been already computed and the problem formulation did not change.

III. IMPLEMENTATION OF PARAMETER OPTIMIZATION APPROACH

In the parameter optimization approach [17] only certain parameters, p, in a specified control policy, $u^0(t, x, p)$, are optimized. To specify a control policy within the software environment, the user is provided with a mechanism to define different operating modes. For each mode, an objective function and constraints are defined to select the control input at time t, *i.e.*, select $u^0(t, x, p) \in U_{ij}$, $j = 1, \cdots, K(i)$, $S_U(t) = i$. The objective functions and constraints can be functions of parameters and these parameters can be included into the vector p for the subsequent optimization. For example, for the DISI engine case study [9], different modes may correspond to stratified, homogeneous and purge operation.

To complete the definition of the control policy, the user defines the mode transition logic. Specifically, the logic-based conditions that enable or disable mode transitions to specified destination modes based on states, control inputs, and operating variables of the powertrain are defined. These transition rules may depend on threshold parameters which can be included into the vector q and subsequently optimized. Figure 7 illustrates a user interface for defining these parameter-dependent mode transition rules.

The parameter optimization of q can be performed in a number of different ways including exhaustive parameter sweeps, designed experiments, search algorithms, *etc*. The approach provides the user with a flexible and powerful framework to optimize parameters in specified control policies and analyze the results while keeping the overall computational effort containable even for large simulation models. In fact, at this stage the use of parameter optimization by internal customers has exceeded that of the dynamic programming. Still as Fig. 8 shows the dynamic programming based solution can be useful in delineating opportunities for improvement.

IV. INTERFACE TO MATLAB/SIMULINK MODELS

To correctly interface MATLAB models with the optimization software environment, the model must be accompanied by a model information file. The model information file identifies key variables and information important for the optimization such as states, inputs, outputs, parameters, units, variable ranges, default values, and constraints. The supported model types are the m-file (MATLAB function), the mdl-file (Simulink) and the mat-file (MATLAB mat files) models. The model information file is generated automatically based on the information the user enters into a model information file editor. Alternatively the model information file can be created manually. For the mdl-files and with the model information file editor open, the user can click directly on the outputs of Simulink blocks; then the model information editor record the path to the relevant variable into the model information file. The mat-files models are used in conjunction with the static optimization environment for generation of a finite set of control policies (2).

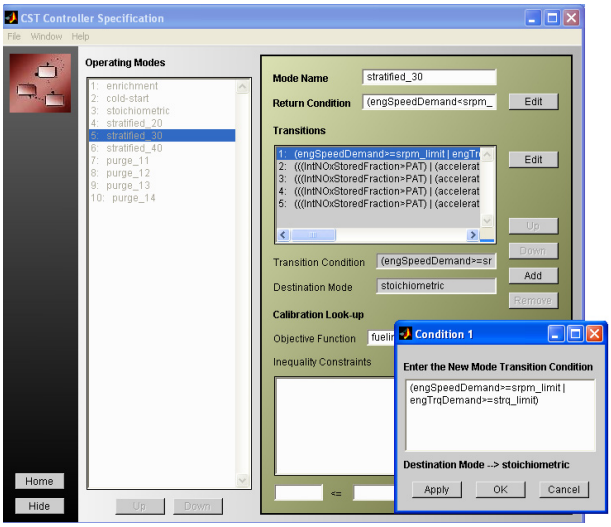

Fig. 7. User interface for defining mode transition logic.

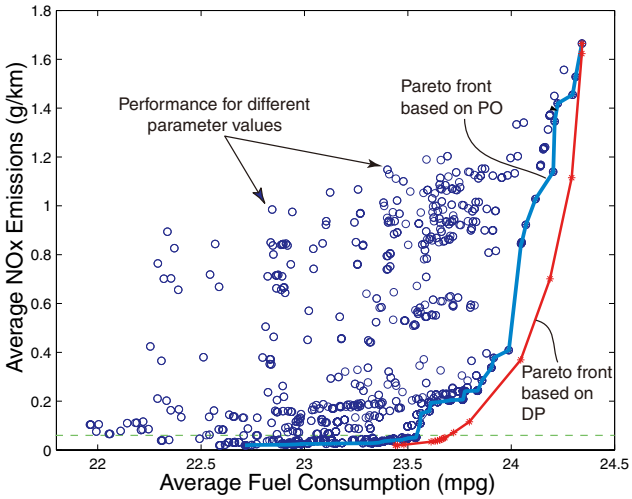

Fig. 8. Pareto fronts (Fuel consumption versus emissions) for DISI engine based on parameter sweeps for fixed structure policies (PO) and based on dynamic programming (DP). Parameters being swept are purge activation threshold and feedgas NOx emission index (see [17]).

In order to run optimizations faster, the plant model can be implemented in the form of an s-function or in C (as a C-MEX file called from MATLAB wrapper). The software environment can generate s-function automatically from a model with native Simulink blocks (with certain restrictions) using Real Time Workshop. Table 1 illustrates the computational speed improvements with this approach.

Table 1. Execution times in sec for s-functions and multiple parallel sessions for a model with native Simulink blocks and for automatically generated s-function.

MATLAB sessions	native Simulink blocks	s-function
1	1028	–
2	524	188
3	360	135

V. PARALLEL COMPUTATIONS

Parallel computations can reduce the time to complete weight sweeps and parameter sweeps for the dynamic programming-based optimization and for the parameter optimization. Additionally, parallel computing can be employed to build the output and state update maps faster. Either single processor computers connected over a network or a single multi-processor workstation can be used to implement this approach.

In the case of a multi-processor workstation, operating systems such as Windows 2000 automatically launch a new MATLAB session on a different processor if there is a MATLAB process running on one of the processors. So, multiple parameter sweeps can be completed faster using parallel sessions on a multi-processor machine as compared to a single processor machine. Tests were conducted on an eight processor Windows 2000 machine with 512MB of shared RAM. Standard demo example with settings for 5 parameter sweeps were run using 1, 2, and 3 MATLAB sessions on this machine. The results for an example with 5 parameter sweeps are illustrated in Table 1.

VI. EXAMPLES

In this section we present three additional examples illustrating the potential use of the software environment.

6.1. Engine idle speed control

We first consider an example of an engine operating near idle (in neutral) for 2 sec in the engine warm up process. The engine speed is constrained between 800 rpm and 1200 rpm. At the end of the time interval the engine speed must be between 980 and 1020 rpm. The control inputs are the indicated torque command to the engine and spark retard. There is a rate limit on the indicated torque that the engine can develop, which is no more than 5 Nm within 0.01 sec. The rate limit is handled by introducing an auxiliary state $z(t)$ which stores the past value of the indicated torque; then the rate limit becomes a pointwise-in-time state/control constraint that the software environment can handle.

To achieve best fuel economy, it is clear that the engine speed must stay close to 800 rpm for as long as possible. On the other hand, for the maximum exhaust heat generation the engine speed must stay close to 1200 rpm for as long as possible. However, if we minimize a weighted sum of total fuel consumption and minus the exhaust heat the optimal speed trajectory can be oscillatory, see Fig. 9.

6.2. Optimizing hydrocarbon injection for a diesel engine

Our second example is an application of the software to a diesel engine case study treated in [11]. The engine is equipped with an Active Lean NOx Catalyst (ALNC) as well as with a hydrocarbon (HC) injector upstream of ALNC in the exhaust system. The HC injector supplies hydrocarbons (*i.e.*, raw fuel) to the ALNC to ensure conversion of NOx emissions in the ALNC. The trade-off variables in this problem are the fuel consumption by the HC injector and the post-ALNC NOx emissions. The key dynamic states in this problem [11] are the temperature of the ALNC and the amount of HC stored in the ALNC.

In the original problem formulation, the total flow rate through the ALNC, the concentrations of NOx, HC, and oxygen at the inlet of the ALNC, as well as pressure and temperature at the inlet of the ALNC are prescribed as functions of time [11]. The hydrocarbon injection rate is the quantity being optimized. We partition the vector of conditions at the inlet to the ALNC as

$$u_d = [u_{d1}, u_{d2}^T]^T,$$

where u_{d1} is the hydrocarbon injection rate and u_{d2} is the vector of the total flow rate through the ALNC, the concentrations of NOx, HC, and oxygen at the inlet of the ALNC, and pressure and temperature at the inlet of the ALNC. The Control Sets are generated so that the elements of each of the Control Set differ from each other only in the u_{d1} component but not in u_{d2} (which is prescribed). The dynamic programming is applied to the resulting problem formulation.

Figure 10 illustrates the setup of the screens for analyzing and visualizing the dynamic programming optimization results. Figure 11 illustrates the Pareto front in terms of average fuel consumption by the HC injector and average NOx convergence efficiency of the ALNC. Figure 12 illustrates Graphic User Interface for plotting the trajectories corresponding to the selected point on the Pareto front as well as the trajectories of stored HC and HC injection rate.

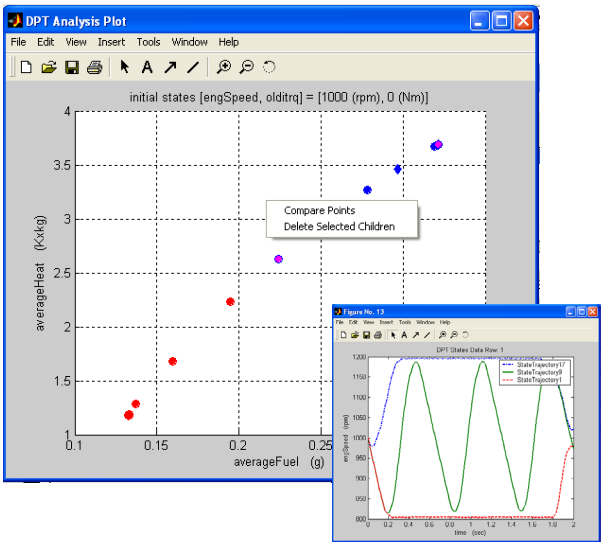

Fig. 9. Optimal speed control example with state constraints: Fuel consumption versus exhaust heat trade-off plot. The plot on the bottom shows engine speed trajectories corresponding to minimum fuel consumption, maximum exhaust heat generation and a minimum of weighted sum of fuel consumption and minus exhaust heat.

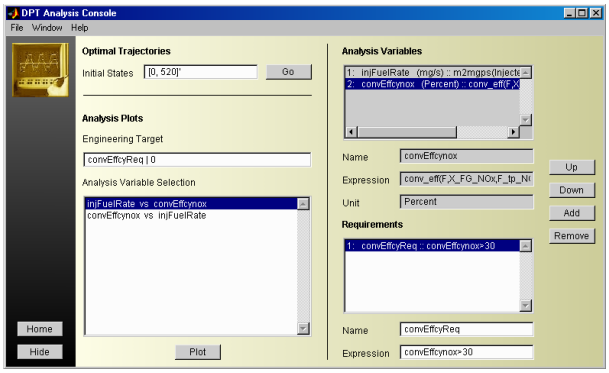

Fig. 10. The appearance of Graphic User Interface for analyzing and visualizing the dynamic programming optimization results.

6.3. Optimizing marketing strategy

The software is sufficiently flexible to also handle non-powertrain related dynamic optimal control problems. One such example is the dynamic optimization of manufacturer advertising and promotion strategies for a product (such as a car) when a retailer or a dealer first gets the product from the manufacturer and then sells this product to the customer. The manufacturer does not directly control the sales but can affect them and the retailer orders via discount and advertising. See Fig. 13.

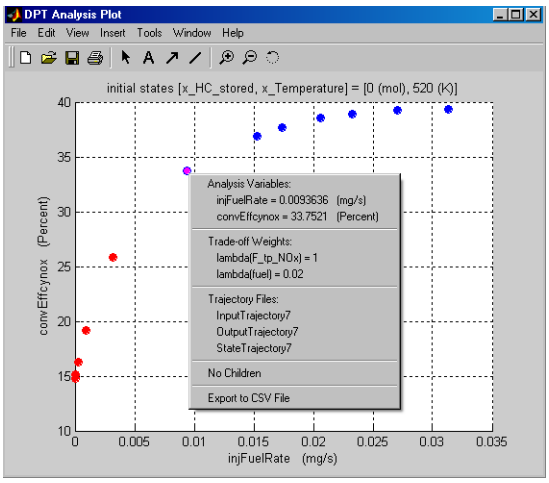

Fig. 11. The Pareto front plotted in terms of average convergence efficiency versus average hybrocarbon injection rate. Right clicking on a point on the Pareto front provides the information menu about the point.

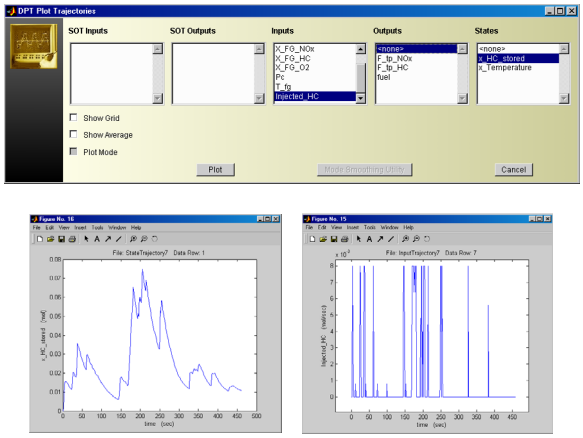

Fig. 12. The appearance of Graphic User Interface for plotting the trajectories corresponding to the selected point on the Pareto front (Top). The trajectories are the stored HC in the ALNC (Lower Left) and the fuel injection rate by the additional injector (Lower Right).

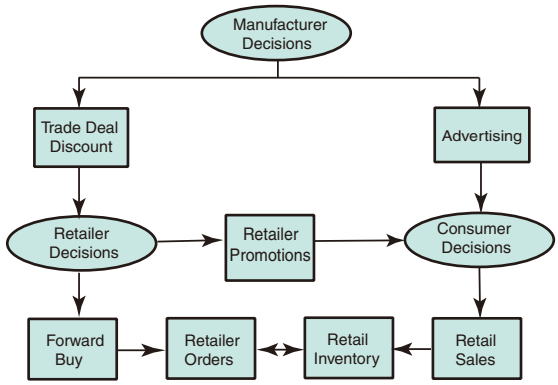

Fig. 13. The process of manufacturer, retailer, and consumer decisions from [18].

The model for this process from [18] includes the dynamics of the manufacturer, retailer and consumer. The manufacturer objective is to maximize revenues net of monthly advertising expenses ($MADV(t)$) and a per-case trade deal discount ($MDISC(t)$), *i.e.*,

$$\sum_{t=1}^{t=T} (MM - MDISC(t)) \cdot TRO(t) - MADV(t) \rightarrow \max ,$$

where MM is the manufacturer margin and $TRO(t)$ are the total retail orders (cases). The model has four states: retail beginning inventories (INV), promotion intensity ($PINT$), past manufacturer trade discount and past manufacturer advertsing expenditures. Figures 14-16 illustrate the dynamic programming based solution and confirm a periodic nature of the optimal promotion and discount strategies.

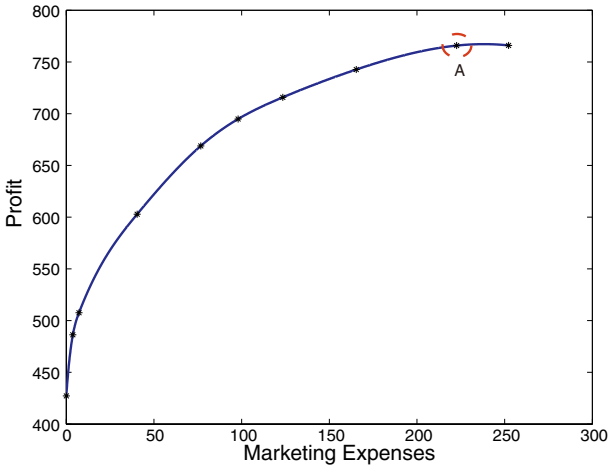

Fig. 14. Trade-off between revenues and marketing expenses (Pareto front).

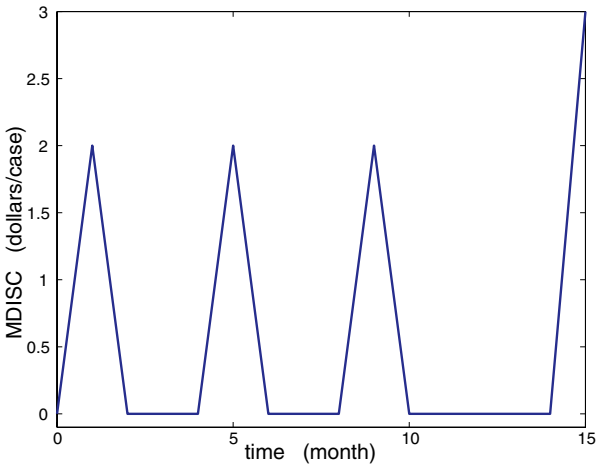

Fig. 15. Discount trajectory corresponding to point A on the Pareto front.

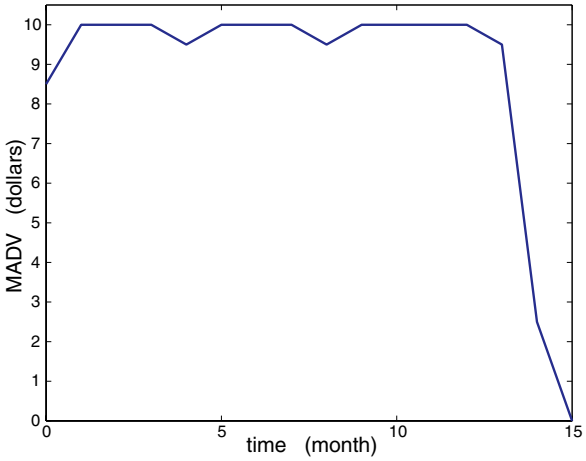

Fig. 16. Advertising trajectory corresponding to point A on the Pareto front.

VII. CONCLUSIONS

The paper discussed at a high level several computational and user interface solutions for implementing an optimal control based powertrain system assessment in a user-friendly software environment. A potential for future extensions exists and includes implementation of additional optimization algorithm and analysis options, ways to address variability and uncertainty in the inputs and model parameters, as well as automatic generation of parameter-dependent control policies from the optimal control solutions.

ACKNOWLEDGEMENTS

Alongkrit Chutinan and Gopalan Raghavachari of Emmeskay, Inc. have contributed to the development and implementation of the software environment discussed in this paper.

REFERENCES

1. Cassidy, J.F., "A Computerized On-Line Approach to Calculating Optimum Engine Calibration," SAE paper 770078, *SAE International* (1978).
2. Dohner, A., "Transient System Optimization of an Experimental Engine Control System Over the Federal Emissions Driving Schedule," SAE paper 780286, *SAE International* (1978).
3. Trella, T., "Spark Ignition Engine Fuel Economy Control Optimization-Techniques and Procedures," SAE paper 790179, *SAE International* (1979).
4. Rao, H., A. Cohen, J. Tennant, and K. Van Voorhies, "Engine Control Optimization via Nonlinear Programming," SAE paper 790177, *SAE International* (1979).
5. Rizzo, G. and C. Pianese, "A Stochastic Approach for the Optimization of Open-Loop Engine Control Sys-

tems," *Ann. Oper. Res.*, Vol. 31, pp. 545-568 (1991).

6. Schmitz, G., U. Oligschlager, G. Eifler, and H. Lechner, "Automated System for Optimized Calibration of Engine Management Systems," SAE paper 940151, *SAE International* (1994).
7. Cohen, A., K. Randall, C. Tether, K. van Voorhies, and J. Tennant, "Optimal Control of Cold Automobile Engines," SAE paper 840544, *SAE International* (1984).
8. Sun, J. and S. Sivashankar, "Issues in Cold Start Emission Control for Automotive IC Engines," *Proc. Amer. Contr. Conf.*, Philadelphia, PA, pp. 1372-1376 (1998).
9. Kang, J.M., I. Kolmanovsky, and J.W. Grizzle, "Dynamic Optimization of Lean Burn Engine Aftertreatment," *J. Dyn. Syst. Meas. Contr.*, Vol. 123, pp. 153-160 (2001).
10. Brahma, A., Y. Guezennec, and G. Rizzoni, "Dynamic Optimization of Mechanical/Electrical Power Flow in Parallel Hybrid Electric Vehicles," *Proc. 5th Int. Symp. Adv. Vehicle Contr.*, Ann Arbor, MI (2000).
11. Aswami, D.J., M.J. van Nieuwstadt, and J.A. Cook, "Control-Oriented Modeling of a Diesel Active Lean NOx Catalyst Aftertreatment System," *J. Dyn. Syst. Meas. Contr.*, Vol. 127, No. 11, pp. 1-12 (2005).
12. Lin, C.C., H. Peng, and J.W. Grizzle, "A Stochastic Control Strategy for Hybrid Electric Vehicles," *Proc. 2004 Amer. Contr. Conf.*, Boston, MA, pp. 4710-4715 (2004).
13. Kolmanovsky, I., I. Siverguina, and B. Lygoe, "Optimization of Powertrain Operating Policies for Feasibility Assessment and Calibration: Stochastic Dynamic Programming Approach," *Proc. Amer. Contr. Conf.*, Anchorage, AK, pp. 1425-1430 (2002).
14. Tate, E. and S. Boyd, "Finding Ultimate Limits of Performance for Hybrid Electric Vehicles," SAE paper 2000-01-3099, *SAE International* (2000).
15. Kolmanovsky, I. and I. Siverguina, "Feasibility Assessment of Operating Policy Optimization of Automotive Powertrains with Uncertainties Using Game Theory," *Proc. ASME Int. Mech. Eng. Cong. Exp.*, New York (2001).
16. Bertsekas. D.P., *Dynamic Programming and Optimal Control*, Athena Scientific, 2nd Ed. (2001).
17. Kolmanovsky, I., M. van Nieuwstadt, and J. Sun, "Optimization of Complex Powertrain Systems for Fuel Economy and Emissions," *Proc. IEEE Int. Conf. Contr. Appl.*, Vol. 1, Kohala Coast, Hawaii, pp. 833-839 (1999).
18. Neslin, S.A., S.G. Powell, and L.S. Stone, "The Effects of Retailer and Consumer Response on Optimal Manufacturer Advertising and Trade Promotion Strategies," *Manag. Sci.*, Vol. 41, No. 5, pp. 749-766 (1995).

APPENDIX
INTERPOLATION PROCEDURE

The static optimization may be performed directly over a set of static subsystem output response data stored in a .mat file. The data have a general form $v^q = f_s(w^q, u_s^q)$, $q = 1, \cdots, Q$, where Q is the total number of data points, w^q is the operating variable vector at the qth point, u_s^q is the vector of the static control variables at that point and v^q is the vector of output response variables of the static subsystem.

The static optimization seeks to optimize u_s^q for all w^* in the specified Operating Variable Set, W. To perform the optimization it is necessary to approximate the value of $f(w^*, u_s^q)$. Accomplishing this using the conventional interpolation can be intricate because the data set $\{(w^q, u_s^q),\ q = 1, \cdots, Q\}$ may not be in the suitable form.

The following procedure can be employed instead. First, all points in the data set with w^q sufficiently close to w^* are determined. For these points, $f(w^*, u_s^q)$ is approximated based on the value of $f(w^q, u_s^q)$. To compute the approximation, the output response variables (elements of the vector v) are classified into two types. The variables of the first type are scaled by the power ratio $(w_1^* \cdot w_2^*)/(w_1^q \cdot w_2^q)$ while the variables of the second type remain unchanged between w^q and w. The examples of variables of the first type are the flow rates which may be scaled by the indicated power ratio. The examples of variables of the second type are temperatures which remain unchanged.

An optional "belief" function, dependent on the distance from w^q to w^* may be utilized by the user to scale the trade-off variables. Such a "belief" function can discourage using the minima based on large degree of extrapolation.

Ilya V. Kolmanovsky studied as an undergraduate at Moscow Aviation Institute, Russia. He received the M.S. degree and the Ph.D. degree in aerospace engineering, in 1993 and 1995, respectively, and the M.A. degree in mathematics, in 1995, all from the University of Michigan, Ann Arbor. He is currently a Technical Leader in Powertrain Control at Ford Research & Advanced Engineering of Ford Motor Company, Dearborn, MI. In addition to expertise in the automotive engine and powertrain control, his research interests include potential of advanced control techniques as an enabling technology for advanced automotive systems, and several areas of control theory, which include constrained control, optimization-based and model-predictive control, and control of nonlinear mechanical, nonholonomic, and underactuated systems.

Dr. Kolmanovsky is the Chair of IEEE CSS Technical Committee on Automotive Control. He has served as an Associate Editor of IEEE CSS Conference Editorial Board (1997-1999), IEEE Transactions On Control Systems Technology (2002-2004), IEEE Transactions On Automatic Control (2005-present), and he was a Program Committee Member of American Control Conference in 1997, 1999, and 2004. He is the recipient of 2002 Donald P. Eckamn Award of American Automatic Control Council for contributions to nonlinear control and for pioneering work in automotive engine control of powertrain systems and of 2002 IEEE Transactions On Control Systems Technology Outstanding Paper Award.

Jing Sun received her Ph.D. degree from University of Southern California in 1989, and her B.S. and M.S. degrees from University of Science and Technology of China in 1982 and 1984, respectively. From 1989-1993, she was an assistant professor in Electrical and Computer Engineering Department, Wayne State University. She joined Ford Research Laboratory in 1993 where she worked in the Powertrain Control Systems Department. After spending almost 10 years in industry, she came back to academia and joined the faculty of the College of Engineering at University of Michigan in 2003 as an associate professor. Her research interests include system and control theory and its applications to marine and automotive propulsion systems.

She holds over 30 US patents and has co-authored a textbook on Robust Adaptive Control. She is an IEEE Fellow and one of the three recipients of the 2003 IEEE Control System Technology Award. She is also a subject editor for International Journal of Adaptive Control and Signal Processing.

Shiva N. Sivashankar received the B.Tech. degree from the Indian Institute of Technology, Madras in 1988, the M.S.E.E. degree from the University of Minnesota, Minneapolis in 1991 and the Ph.D. degree from the University of Michigan, Ann Arbor in 1993, all in Electrical Engineering.

During the summer of 1993, he was a visiting scholar in the Department of Aeronautics and Astronautics, Naval Postgraduate School, Monterey, CA where he worked on control systems for unmanned aerial vehicles. From 1993 to 2000, he was with Ford Research Laboratory working on several design aspects of powertrain control systems and processes for embedded control software development. He is currently the Vice President for Technology at Emmeskay Incorporated, a company that provides systems engineering based products and services. His research interests include modeling and control of automotive systems, application of systems engineering methods and tools to product development processes and multivariable robust control and estimation methods.

Dr. Sivashankar has served as an associate editor for the ASME Journal of Dynamic Systems, Measurement and Control.