

# A Database-Driven In-Cycle Engine Simulator for Control, Calibration and Robustness Testing

James C. Peyton Jones and Kenneth R. Muske

Villanova University

Copyright © 2008 SAE International

## ABSTRACT

Increasingly, advanced engine management systems incorporate high speed Digital Signal Processing (DSP) units for analyzing high-bandwidth, in-cycle signals such as those obtained from cylinder pressure, or knock sensors. In order to develop, calibrate and test the robustness of these algorithms, it is helpful to work in a simulation environment capable of simulating high-speed in-cycle data and its interaction with the engine management and DSP control strategies. Typically, however, in-cycle simulation is both deterministic and highly computationally intensive so a realistic, cyclically-varying simulation of in-cycle data is hard to generate. In this paper an alternative approach is used, based on initially recording files of high-speed, in-cycle data at different engine conditions. This database is then used to simulate the engine response as the specified engine condition varies, by playing back data from the appropriate files at each time instant. The process is straightforward in concept, but memory management is a significant issue due to the large size of the database. These issues have been addressed, and the simulation runs in near real-time in the Matlab / Simulink environment. Different noise processes, gains, and biases can also be applied to the data to simulate the effect of faults for robustness testing without the need for expensive dynamometer tests.

## 1. INTRODUCTION

The use of simulation is a key element in the design process, allowing optimization of system behavior before committing the design to a hardware implementation. The form of the simulation, and the underlying system model, are strongly dependent upon the application. Different applications require different levels of spatial and temporal accuracy. In-cycle simulation, as the name implies, seeks to model the processes of combustion and gas exchange to resolutions typically less than 1 degree of crank angle. Such models range from simple single-zone combustion models [1], to much more detailed spatially distributed models of the underlying thermo-fluid dynamic process, [2],[3]. Since

the computational burden generally increases as more detail is added to the model, realistic in-cycle simulations rarely operate in real time. Instead, such models are used mainly to study a particular phenomena, such as fuel injection, over a relatively short time span, generally within a single deterministic cycle, [4].

Control system simulation, however, normally occurs on a longer time span, with a temporal resolution of around 100Hz (or around 1 sample per cycle on an angle base), [5]. The interest in this case also lies less in the detailed and deterministic simulation of a single cycle, and more in the interaction of the control system and plant across many different cycles and operating conditions. In-cycle dynamics are therefore lumped into mean value engine models which are much more computationally efficient and which require relatively few model parameters, [6],[7]. The models are often coded in the Matlab and Simulink environment and the control system design is developed using the same toolset. As well as testing the performance aspects of the strategy, disturbances and malfunctions can also be introduced into the simulation in order to test the robustness of the system. This is particularly valuable since it is much harder to generate malfunctions on the physical system, and in some cases this might even damage the engine. Once satisfactorily tested in simulation, the strategy can be auto-coded, downloaded and validated on the engine management system target.

Traditionally, there has been little need to include in-cycle processes in control system simulations because of the sampling and processing rate limitations of the target hardware. Increasingly, however, advanced engine management systems incorporate high speed Digital Signal Processing (DSP) units for analyzing high-bandwidth, in-cycle signals such as those obtained from cylinder pressure or knock sensors, [8]. To develop, test and validate the new DSP algorithms requires some form of in-cycle data simulation which can also be integrated with existing control system simulation code. A new knock detection algorithm, for example, might operate in the kHz range but also needs to integrate with the spark advance and knock limitation strategies which run at much lower rates. The underlying models must run in near real time in order to allow the behavior

of the control system in response to the high bandwidth data signals to be studied and developed.

In this paper a new Simulink block is developed which outputs realistic in-cycle data in near real-time for control system development and testing. This is achieved by playing back high speed data from a database of recordings, where the file source is selected according to the current simulation operating conditions. Memory management issues are largely hidden from the user, but are a significant challenge. A single file containing 100 cycles of data recorded at 100kHz from an 8 cylinder engine, for example, is roughly 100 Mb in size. The database, which contains many such files, is therefore very large and efficient ways to handle the data are essential.

The organization of the paper is as follows. The concept of a database-driven engine simulator, and the internal structure of the new simulation block, are elaborated in more detail in section 2. Initial results are presented in section 3 and the user interface is also described. In section 4, the approach is extended, and a stochastic simulation technique is presented which reduces the data requirements by an order of magnitude. Finally brief conclusions are given in section 5.

## 2. A DATABASE DRIVEN IN-CYCLE SIMULATOR

The basic concept of the proposed database-driven engine simulator is shown in figure 1. The database is first populated by recording high speed, in-cycle engine data taken from across a grid of engine and control variable operating conditions. This is a lengthy and time-consuming process, but only has to be performed once for a given engine. The data is stored in files which are accessed during simulation in order to play back the high speed data corresponding to the current simulation condition. This data stream is fed to the DSP and control algorithms, which in turn adjust the engine operating condition. Playback data is then drawn from a different file source, reflecting the response of the engine to the given control action.

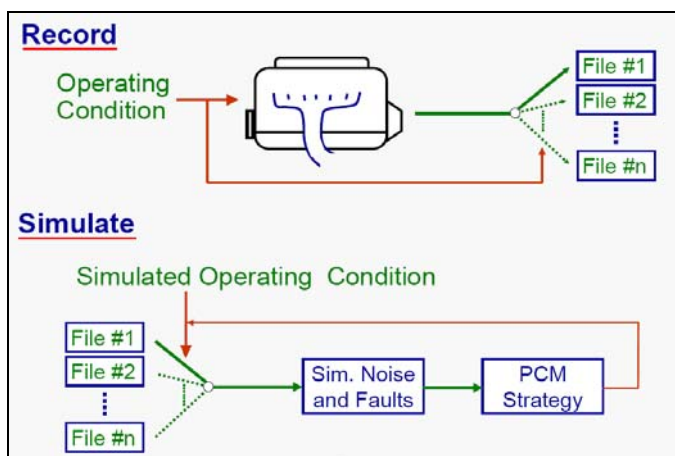


Fig 1. Database-driven simulator concept

One advantage of this approach is that it generates highly realistic, high bandwidth data, without the modeling effort normally associated with detailed in-cycle simulation. (The data, after all, originated from a real engine). The method is also much more computationally efficient than traditional in-cycle methods and operates in near real time. If desired, playback data can also be passed through additional simulation blocks, representing the effects noise or sensor faults, before being fed to the DSP and control algorithms. The tool, which is implemented in Simulink, can therefore be used to exploit the large datasets already held by many organizations, in order to develop combined DSP and control algorithms on the next generation of Engine Control Unit hardware.

Though simple in concept, a number of issues need to be addressed in order to implement the simulator. Foremost among these is memory. Data stored in files is normally accessed in Simulink using a 'from file' block. On initialization, the whole file is read into memory, and data points are then outputted as the simulation progresses. In this case, however, the data files are very large because data may well be sampled every 0.1 degree for several hundred cycles. Furthermore, there are many such files, (one for each operating point in the simulation search space), and loading all these files into memory is prohibitive.

In order to avoid these difficulties, embedded Matlab code was written so that data was only loaded into memory as it was needed. This required some data pre-processing because the compressed file format in which the data was originally stored did not allow random access. The file read process also required a fixed data structure, so data sampled at different rates was divided into separate files. The pre-processing task is highly automated, and iterates through every entry in a 'Database Description File' which describes the database of recorded data in a systematic way.

The operation of the simulator itself is illustrated in figure 2. The first part, called 'File Chooser', is essentially a lookup table of 'file identifiers' or 'fid's' which provide a reference to the data files, without actually loading any data into memory. The 'File Chooser' uses the database description file to determine the file that most closely matches the current operating conditions of the simulation. It then outputs the corresponding fid together with an 'end-of-file' variable which indicates the maximum length of the file.

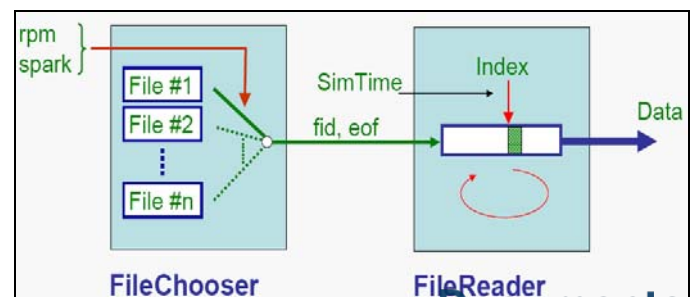


Fig 2. Internal Structure of the Simulator

These parameters are passed to the 'File Reader' component which reads just one cycle's worth of data from the given file and then outputs these data points sequentially as the simulation progresses. At the end of the cycle, the fid is checked once more and the next cycle of data is read according to the new fid value. As the operating condition changes, data from different cycles are therefore spliced together as illustrated in figure 3. Notice that data reading from a given file always resumes from the point at which it left off, unless the end of file is reached. In this case the process loops and data is read from the beginning of the file once more.

The transitions from one data file to another always occur at the end of a cycle, rather than the earliest instant at which the fid changes. This is to ensure that the small file switching discontinuity occurs during the gas exchange phase, rather than during combustion, when its effect would be much more noticeable. A Boolean variable is also outputted (for diagnostic purposes) in order to indicate when file switching occurs.

Another important function of the 'File Reader' component is to synchronize the data to the time step of the simulation. Data is typically recorded using an angular encoder to trigger sampling, and each sample is time-stamped. In some cases the simulation time step is also defined in crank angle degrees, in which case it is fairly straightforward to play back the data at the correct simulation instant. In most cases, however, simulation time is measured in seconds, and the fixed angular

steps with which the data was recorded appears as irregular sampling when translated into the time domain (see figure 4). The 'File Reader' therefore maps the data onto the new time base. Although this mapping can be achieved using linear interpolation, for reasons of computational efficiency the present system merely outputs the next recorded sample after a given time tick. This practice introduces a small phase jitter in the simulation, but its magnitude is very small because the raw data is sampled at a very high rate.

### 3. USER INTERFACE AND INITIAL RESULTS

The complexities of the 'File Chooser' and 'File Reader' components are hidden from the user using Simulink's masking capability. The database-driven simulator therefore appears as a single block. The inputs to the block define the current engine operating condition and simulation time, and the outputs yield the appropriate high speed in-cycle data. Double-clicking the block brings up a user dialog which allows the user to select the desired database, or subset of a database, by loading the specified 'Database Description File', (see figure 5). The dialog also displays the indexes of the grid of operating points defined in the description file.

In the example shown in figure 5, the inputs speed and spark advance define the operating conditions of interest. All the channels of data recorded across a grid of these operating conditions are available as outputs. However, the outputs are also grouped according to the sampling rates used in the original experiments and appear on 'fixed rate' and 'mixed rate' output buses. In practice, not all channels may be needed in a given simulation, or it may be desired extract a particular channel in order to modify it in some way so as to imitate a fault. The Bus Selector blocks shown in figure 5 provide this functionality. Double-clicking the selector block brings up a dialog which lists the available signals and allows the user to select the channels of interest. [The naming of the signals propagates automatically from the Database Description File].

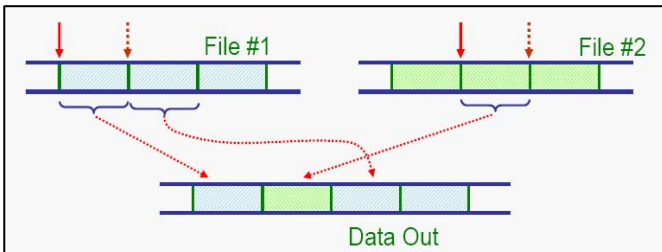


Fig 3. Interleaving of data from different files

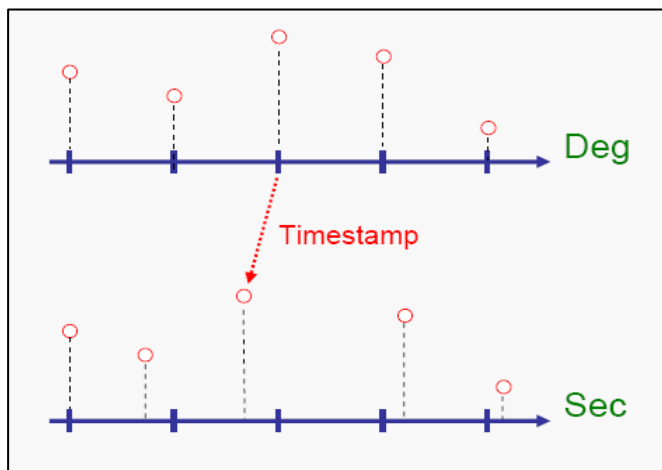


Fig 4. Mapping angle-based data to the time domain

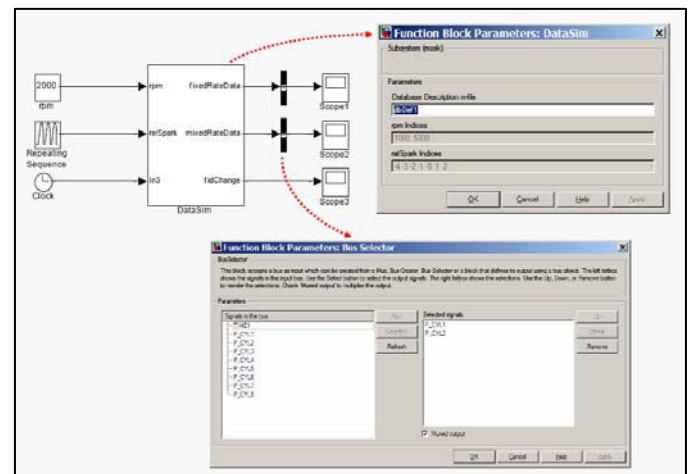


Fig 5. User Interface and Dialog boxes