2001-01-0707

The Development of Tools for the Automatic Extraction of Desired Information from Large Amounts of Engineering Data

Jianrong Dong
Gabriella Cerrato-Jay
Chih-Hung Chung
MTS Systems Corporation

ABSTRACT

Product development processes generate large quantities of experimental and analytical data. The data evaluation process is usually quite lengthy since the data needs to be extracted from a large number of individual output files and arranged in suitable formats before they can be compared. When the data quantity grows extremely large, manual extraction cannot be done in a limited timeframe. This paper describes a set of tools developed by MTS engineers to automatically extract the desired information from a large number of files and perform data post-processing. The tools greatly improved both speed and accuracy of the evaluation process during the development of a sound quality-based end-of-line inspection system for seat tracks [1]. It allowed engineers to quickly gather a comprehensive understanding of the relative importance of individual design parameters and of their correlation to the subjective perception of the sound quality of the seat track.

INTRODUCTION

The need to develop an automatic processing program came naturally during the development of a Sound Quality – Based End-of-Line inspection system. The goal of the project was to establish and implement pass/fail criteria based on certain standard metrics and other numerical values obtained from the recorded sound files. These metric values are defined in psychoacoustics theory [2], which is the interdisciplinary science between psychology and acoustics. It allows quantitative evaluation of subjective sound sensations. After the project started, numerous sound files of different seat tracks arrived. The transient segments of each sound file were trimmed off before a number of metric values and functions were computed. For each sound file, its metric values were saved in a metrics file and its corresponding metric functions were saved in a universal file. As part of the logic, we also computed some descriptors from the time variant metrics functions. Both the metric values and the descriptors of different seat tracks were put into an Excel file, which is a summary of the metric values and function descriptors for all the data.

Time was limited to develop the pass/fail criteria. Additionally, reprocessing the data manually to compute additional metrics could double or triple the effort. The job had to be done automatically.

Similar situations often arise in the auto industry since problems are often urgent and engineers would like feedback from numerous sources of test data as soon as possible. Manual processing of data is too slow. Test engineers spend a lot of their time doing data reduction after the data is acquired.

For as large as the data quantity is, the number of different data formats is very small. Automatic processing has two advantages here. First, automatic data processing tools can greatly increase an engineer's productivity by reducing both time and energy spent on repetitive work. The advantage of automatic processing over manual processing becomes more apparent when the data quantity grows. Second, the number of programs that needs to be developed per project is very small. With the experienced developer, the automatic processing tools can be developed in very short time.

The reason why there are very few automatic processing tools in the market is that different projects have different requirements on how the data should be processed and a cure-all program is unrealistic. Experience has shown that seldom can major parts of programs that are developed for previous projects be used on the current one. Different requirements often result in completely different data structures. Inserting part of the old programs into new ones is often ineffective and error prone.

An automatic processing tool is similar to a database in that both of them have a data structure to store the data and provide some processing functions. However, they are quite different in other aspects. First, the database has a query language like SQL to provide such functions as creating, searching, and updating records. This process is user-driven. The automatic processing tool, on the contrary, has preset tasks, which can only be changed by modifying the code. Second, to provide such comprehensive functionality plus a GUI, a database requires a large amount of code. An automatic processing tool is much smaller, requiring only several hundred lines of code. Third, using a commercial database to deal with post processing of data is also not feasible since the input format to the database is fixed and it has limited mathematical processing power. As a result, the engineer has to develop something independently in addition to using the database.

In this case, one has to develop something new for each different project. The following guidelines should be followed to reduce development time while generating robust code.

MINIMIZE DEVELOPMENT EFFORT – Writing one's own code is always the last method. Exploring the batch processing capability of commercial software and bridging their functions with interface programs saves both time and effort. Our experience showed that the following software is very useful in speeding up the process.

- **TCL/TK Scripting**. In the MTS-Sound Quality software, a very powerful programming tool called TCL/TK scripting can be used to write batch programs. TCL/TK can be used to pop up different menus, load parameters to different forms, start various processes like filtering, editing, transporting, etc., change display formats and compute metric values. Using scripting is just like using menus to generate sound quality metric values and function files. Generally, the only program that needs to be developed is the one that gathers information from the individual output files of metric values and metric function descriptors and puts the result into Excel.

- **MTS-IMAT Software**. Developed in MATLAB, MTS-IMAT is the interface software between I-DEAS Test and MATLAB. IMAT software can read in and write out I-DEAS Test files and general universal files. By providing new objects in MATLAB, IMAT can perform data management since the user can now access all the header information in the universal files and manipulate file storage more easily. IMAT is very capable and flexible at processing test FRF's because MATLAB has provided a huge library of matrix related routines. In addition, with the help of IMAT, MATLAB expands its processing objects to experimental data, which helps the development of hybrid models, correlation, and structural modification procedures. The graphics capability of IMAT is also very powerful.

It would be unwise and error prone to obtain these functionalities by developing the software from scratch.

KISS – Keep It Short and Simple. The shorter the program, the faster it can be developed. To be short, the programming language should have a good library to provide sufficient high-level support functions. In engineering applications, a good math library is essential. To be simple means the program should be modular and no fancy GUI's are needed. It also means that the data structure of the programming language should be simple to manipulate. For instance, all the data types in MATLAB are matrices, and it has a comprehensive library on string pattern recognition, matrix manipulation and sorting, etc. These capabilities make MATLAB a good candidate when selecting the programming language for the automatic processing tool.

COMMON ROUTINES – Some common procedures of these data processing programs should be isolated from the rest of the software. This eliminates the need for repetitive modifications in several places if the code is changed at one place.

EASY DEBUG CAPABILITY – The programming language should be easy to debug. Here execution efficiency is not so important because any automatic processing is much faster comparing with manual processing. However, the ability to easily step through programs and view the current values of different variables is highly desirable for debugging purposes. Therefore, a language that executes faster but needs compiling after each modification, such as C or C++, is not as convenient as some language running as an interpreter, such as MATLAB. Furthermore, if speed is really important, the MATLAB compiler can be used to compile the code after its correctness has been verified.

ROBUSTNESS WITH IMPERFECT INPUT – Unlike CAE output, experimental data is inconsistent in nature. The program should be able to handle the imperfectness in the data. It should warn the user and pause the execution in the case of unexpected input. For example, sometimes the RPM curves of the DC motors are so contaminated that certain metrics computation algorithms result in unreliable data. Sometimes the output is even incomplete. The subsequent automatic processing programs must be able to detect this error and act accordingly to stop error propagation.

MINIMAL MODIFICATION FOR DATA GROWTH – The program should be able to accommodate the growth of data. Since only a small portion of data is available at the beginning of coding, the input routine is separated from the rest of the program so that subsequent changes are limited to adding new file names to the input file list. Besides, variables should be used to describe the length, number of elements etc., of the input file list. The other part of the program should reference these variables instead of using numbers directly.

The end of line project described in the companion paper [1] is a good example of a case where these principles were applied. In this project the editing was done half automatically with the help of scripting, since all sounds also had to be listened to for subjective evaluation.

TCL, a programming language like LISP, was used to write scripts for filtering the data and computing the metrics values and functions. One of our tests shows that applying two filters and computing 15 metric values for 300 sound files takes only several hours without engineer's supervision while doing it manually takes more than a week. The average metrics values were output to the metrics files and the functions to universal files.

The major development work was limited to the programs that extract the metrics values and functions from individual metrics files and universal files, compute the descriptors of the metrics functions, store the results in a matrix, and process the matrix to yield a global picture on the distribution of all seat track data. The result was an ASCII file that can be read into Excel without modifications. The software is small in size, altogether 400 lines including a lot of comment lines, excluding the huge input file name list. However it processes the data in little time.

Figure 1 shows the overall process for post-processing the data. The ati files are from MTS I-DEAS Test; they hold multiple records, every one of which is either a sound pressure record or a tachometer record. The tachometer signal is embedded into the least significant bit of the sound record, and the sound format is changed into a special wave format for input of MTS Sound Quality Software. The wave files are further edited to trim off transient signals at both ends so that only the steady parts of the signals are kept. Using scripts, we can filter the wave and compute the metric values. There are two ways to go from this point. We can either manually extract all the information from individual metric value files (.mtrc files), and metrics function files (.unv files), or we can use the automatic processing tool developed in Matlab to get it done more quickly. The final results include the Excel file that summarizes the information and the metrics functions. They are arranged in two columns format, that is, one column for uneven x values, and one column for the corresponding y values. The metrics functions can be displayed easily using some software later.

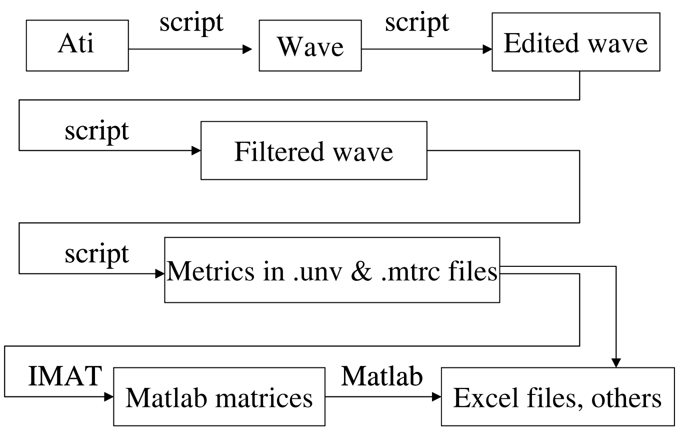

Figure 1. Overall Data Reduction Procedure

PROGRAM DETAILS

Small as it is, the software is designed to meet the growth requirement with minimal changes to the input file names. It consists of four modules. In the first one, **Head1_builder()**, the user can add different input file names, change the naming convention, choose the parameters that need to be processed, etc. The second module, **Digger()**, is an interpreter that scans all input files, recognizes different keywords and extracts the numbers after them. The third module, **Funlogics()** controls the overall process, consolidates all information extracted by the interpreter, calculates the pass-fail criteria based on user-defined algorithms and outputs the results using Excel format. The fourth module, **Newmetrics()**, outputs the user defined numerical descriptor according to the metrics file format so that they can be imported into MTS-Jury for sensory evaluation.

Figure 2 shows the corresponding calling relationship between the four modules. The main module is Funlogics(). It calls the other modules. Digger() further calls **Digfunction()** and **Digvalue()** to deal with the universal files and metrics files separately. For every batch of new data, what the user changes are two lists, the input file name list and the naming convention list in Head1_builder().

DATA STRUCTURE FOR STORAGE AND MANIPULATION – The core data structure is a four dimensional matrix. It stores and manipulates the extracted data. The first dimension is ROW, which represents different seat tracks. It grows larger when more and more seat track data are available. The second dimension is COLUMN, which represents the different metric values plus some derived numbers using the logic. Once set, this dimension is the same for all the sound files. The dimension is deliberately set a little larger so that new derived numbers can use the reserved space.

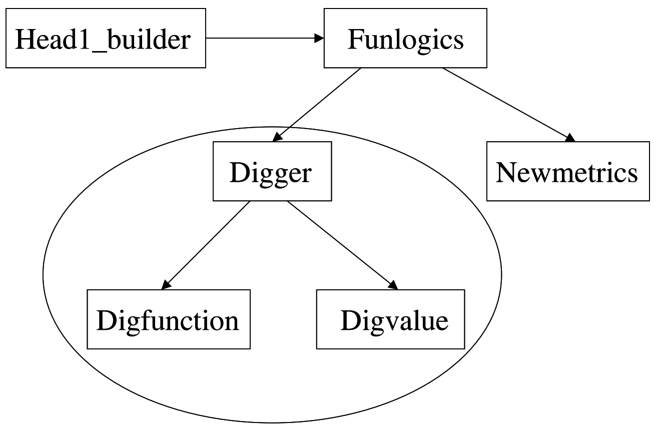

Figure 2. Structure of the Auto Processing Tool

The third dimension is <u>SHEET</u>, which represents all possible motion-ear combinations. The seat tracks have multiple motors and for each motor, two directions are possible, i.e., up/down, forward/rearward. Furthermore, for each sound file, there may be two sound tracks representing both ears of the binaural head.

If only the average values are needed, the above three dimensions suffice. However, if more metric function descriptors are used, a forth dimension, <u>DEPTH</u>, needs to be added to accommodate the additional data.

Using a single matrix to represent all the extracted and computed data makes it easy to process this matrix with MATLAB functions. To output the matrix results to Excel is straightforward. The DEPTH elements are moved into COLUMN by using more column titles.

INPUT AND GLOBAL DEFINITION MODULE – HEAD1_BUILDER() - In the input and global definition module, a name list is made of the input metrics file names and the input universal file names. The naming convention of these files is consistent. A typical name is “sn129_track_fvd.mtrc” or “sn129_track_fvd.unv”. Here *sn* is the abbreviation of serial number and 129 is serial number, *track* stands for track adjuster instead of a completed seat. *fvd* is the shorthand for “front vertical down”. Similarly *fvu* stands for “front vertical up”, and *rvd* stands for “rear vertical down”, etc. “.mtrc” is the suffix for metrics file while “.unv” is the suffix for universal file. Consistent naming facilitates information extraction, since string functions can perform pattern recognition on the names easily.

Variables like number_of_input_files, etc. can be obtained via some MATLAB functions every time the input file name list is updated. As a result, when the list expands, the rest of the program using these variables remains the same.

There are other lists in the input and global definition module that define the titles for the rows, columns, sheets, and depths in the final Excel file. Once the lists are changed, their dimensions are changed automatically and stored in variables. The lists and global definitions are saved in a .mat file and loaded by subsequent modules.

INTERPRETER MODULE – DIGGER() – The second module is an interpreter. It scans through all metric files and universal files, extracts the various metrics values and functions, and computes metrics function descriptors on the fly. It returns the metrics and descriptor values in several 1-dimensional arrays.

A typical sound metrics file generated by the MTS Sound Quality is as follows:

```
Metric Results for Sound File <sn296_track_frvertup.wav>
  Between 0.0000 and 11.4055 Seconds
Subject:        Aachen Head
Test Condition:
Test Engineer:
Test Date:
Created:
Correction:     NONE
MTS Sound Quality 3.6 Alpha 4
Tue Jul 18 10:32:05 2000


Metrics                         Units    Left     Right    Avg.

Linear SPL                      dB
A-weighted SPL                  dBA
B-weighted SPL                  dBB
C-weighted SPL                  dBC
D-weighted SPL                  dBD
Speechband SPL                  dB
Linear SPLT                     dB
Intelligibility                 %
Pref Speech Interference        dB
Speech Interference             dB
Spectrum Balance                dB
Composite Rating Preference              dB
Frame Kurtosis
Average Kurtosis
Zwicker Loudness (Sones)                 sone
Zwicker Loudness (Phons)                 phon
Sharpness                       acum
Transient Loudness (Sones)               sone     2.7      2.8
                2.8
Transient Loudness (Phons)               phon
Transient Sharpness             acum
Time Varying Loudness (Sones)            sone
Time Varying Loudness (Phons)            phon
Roughness                       asper
Fluctuation Strength            vacil
Tonality                                 0.119    0.151    0.135
Speed Variation
Speed Variation2                         1.72     1.72     1.72
FM Fluctuation
```

The metrics file starts with a header with the sound file name, length of recordings, test engineer name, and test date, etc., then follows a list of different sound quality metrics. Some metrics are selected for computation while others are neglected. Different numbers of metrics may be computed for different sound files in different development stages. Occasionally some metrics cannot be computed due to the contaminated RPM signals. Therefore, a fixed input format is unfeasible and not robust enough to deal with the metrics files. The program must recognize the keywords (metrics names), anticipate operands (metrics values), and act accordingly. This keyword based branch structure is exactly the idea of an interpreter.

The interpreter treats the metrics file as a string of characters that are separated by white spaces, carriage returns, etc., and ended with the End Of File (EOF) symbol. Several while loops with internal if-then-else branching statements are used to control the progress. The metrics names are treated as keywords. For instance, "A-weighted" is a keyword and "SPL" is a keyword. Since there may be many SPL's in a sound metrics file, different metrics can be differentiated by keeping track of the most recent four keywords. Some MATLAB functions are used to check the validity of the operand. If the operands corresponding to a metrics keyword are missing, a warning message is generated and the program suspends itself. Validity checks can also be put on the operands to see if they are out of reasonable ranges.

To deal with universal files, Digfunction() uses some functions of IMAT to read universal files to MATLAB matrices. The metrics functions extracted from universal files are never stored. They are extracted on the fly and saved into different file names. Only the function descriptors are calculated and stored in the storage matrix. The advantage is, no matter how different the function lengths are, the numbers of their descriptors are the same. So using a matrix to hold these descriptors is simplest, most economic and efficient for retrieval. The address computation for a matrix is the simplest of all data structures that hold large quantities of data.

COMPUTATION AND EXCEL OUTPUT MODULE – FUNLOGICS() – The computation and excel output module is the main module. First it calls the input and global definition module and use the dimensions defined in that module to initialize the storage matrix. In a loop that covers all the input metrics files and universal files, the interpreter module is called to dig out the metrics values and metrics functions. By pattern matching the metrics file and universal file names with a row title in the row title list and a motion title in the motion-ear title list, respectively, it locates a specific position in the storage matrix where it can put the metrics values and function descriptors. If the *m*th element in the row title list and the *k*th element in the motion-ear title list are matched, the storage index is (*m*, :, *k*, :). The second dimension index is determined by metrics. For instance, Kurtosis has index 1 and Loudness index 2, etc.

After the storage matrix is completed, all the data are printed out in Excel file format. There is some subtlety here. A new index EXTENDED COLUMN needs to be established in order to convert two indices COLUMN and DEPTH to one index EXTENDED COLUMN. As a result, an Extended Column Titles List can be formed so that the descriptor numbers in the fourth dimension DEPTH can have a title in the EXTENDED COLUMN. This list can be defined in the input and global definition module.

The output to Excel uses three levels of looping. The first level loop prints out 12 sheets, corresponding to the 12 motion-ear pairs (6 motions x 2 ears). The sheet title and the column titles are needed for every sheet. The second level loop is for different seat tracks, which are represented by the different rows on that sheet. The name of the seat track is printed. The third level loop is for different metrics and their descriptors in the EXTENDED COLUMN. It prints out the different metrics and function descriptor values in a line, but internally it converts one index EXTENDED COLUMN back into two indices, COLUMN and DEPTH in order to retrieve the data in the storage matrix.

MODIFIED METRICS FILES OUTPUT MODULE – NEWMETRICS() - The metric descriptor results can be output again to some modified metrics files. The reason is that the MTS-Jury Evaluation software needs a standard metrics file as its input format. Therefore, new titles can be used for the additional descriptors. Similar to the Funlogics() module, it prints out the EXTENDED COLUMN titles followed by the values of the function descriptors, that is, left, right and average values.

The pass/fail logic is based on these metrics values and function descriptors. One method to develop the pass/fail logic is to accumulate a sufficient amount of sound data and set acceptable thresholds on some specific metrics descriptors. The ability to provide a global picture very quickly is critical in this process. After the prototype is developed, the pass/fail logic needs to go through several thousands of seat tracks off the assembly line for final tune-up.

Another method is to use Jury to develop a regression equation relating the preference number obtained from some Jury tests to the different metrics descriptors in the modified metrics files. Traditionally, the logic used in the preference equation is based on the single or average number of the metrics such as loudness, kurtosis, speed variation etc. However, experience has shown that in some cases, it is not the average numbers that influence our subjective judgment. In many cases, the time variation of the metrics function also has a major role in determining our preference. Therefore it is desirable to use the functions instead of the single values in the equations.

CONCLUSION

The small set of tools discussed above was developed in only a few days. Yet it has greatly facilitated the engineer's daily work in processing. The engineer can then concentrate on the most important part of the project, that is, to develop, implement and validate different pass/fail logic criteria. However, the development rules discussed here have wide application in our daily practice, apart from its original goal in the end-of-line detection system project.

REFERENCES

1. T. Bernard, G. Cerrato-Jay, J. Dong, DJ Pickering, L. Braner, R. Davidson, *The Development of a Sound Quality-Based End-of-Line Inspection System for Powered Seat Adjusters*. SAE 01PC-205.
2. E. Zwicker, H. Fastl, *Psychoacoustics, Facts and Models*, Spring-Verlag, 1990.

CONTACT

For any discussion on the paper, please contact the authors. Jianrong Dong, PhD., Project Engineer, Jianrong.Dong@mts.com. Office: (248) 397-4964. Gabriella Cerrato-Jay, PhD., Principal Engineer, Gabriella.Cerrato-Jay@mts.com. Office: (248) 397-4953. Chih-Hung Chung, PhD., Senior Project Engineer. Jerry.Chung@mts.com. Office: (248) 397-4954. MTS Systems Corporation, Noise and Vibration Division, 800 East Whitcomb Ave. Madison Heights, MI 48071. Phone: 248-585-5000. Fax: 248-585-3013.