Structural Health Monitoring Tools (SHMTools)

Getting Started

LANL/UCSD Engineering Institute

LA-CC-10-032
LA-UR 10-01259

August 2, 2010

Contents

1 Introduction to SHMTools

SHMTools is a MATLAB package that facilitates the construction of structural health monitoring (SHM) processes. The package provides a set of functions organized into modules according to the three primary stages of Structural Health Monitoring: Data Acquisition, Feature Extraction, and Damage Detection. A modular function design and a set of standardized parameter formats make it easy to assemble and test customized SHM processes. A couple of assembly routines are provided along with SHMTools. The first is a general GUI interface called mFUSE (provided separately with its own documentation) that can be used to build an entire SHM process by simply specifying a sequence of functions from the various modules. The second assembly routine is more specialized: it is used to build a custom detector by combining pieces from within the Damage Detection module.

SHMTools is made available for free by the LANL/UCSD Engineering Institute. It is the beginning of a larger effort to collect and archive proven approaches to SHM for re-use by the research community. The package therefore includes various algorithms with source codes, along with structural data to serve as benchmarks for the evaluation of algorithms.

2 Installation

SHMTools download

Download the file *SHMTools.zip* from `http://institute.lanl.gov/ei/software-and-data/`. Unzip the files into a directory of your choice and add this directory including sub-directories to the MATLAB path (in MATLAB click File $\mapsto$ Set Path $\mapsto$ Add with subfolders).

At the top level directory one can find the *SHMFunctions* folder, *Documentation* and *Examples* folders, which are discussed in the following sections.

3 SHMFunctions

In this section we describe the various modules within the SHMFunctions folder of SHMTools in more detail. The Data Acquisition module deals with interfacing with existing commercial DAQ systems and returns raw data (e.g. time series) to the Feature Extraction module. Functions in the Feature Extraction module take the raw data and extract damage sensitive features to be used by the Feature Classification modules. The Feature Classification module implements various approaches to classification: given a set of training examples (examples are feature vectors), these functions build models that serve to classify future examples. The SHMTools package include additional functions implementing algorithms that support the SHM process, e.g. sensor diagnostic tools. These support functions are collected in an "Auxiliary module" which we discuss last.

3.1 Data Acquisition module

These functions provide basic services for common SHM related data acquisition tasks. Currently the software supports basic functionality for MATLAB DAQ Toolbox-supported hardware as well as interfacing with National Instruments high speed digitizers, function generators, and switches not otherwise supported with the DAQ Toolbox.

We note that the user of SHMTools has the flexibility to import his own (raw) data into the system and is not required to use the functions provided in the Data Acquisition module. This data can be used directly by the Feature Extraction module provided it's in the required matrix form. For instance for time series data collected from multiple channels, the data is expected in the form of a 3-dimensional matrix of time $\times$ channels $\times$ instances. See the *Parameter Specifications* document for more information.

3.2 Feature Extraction module

The functions in this group take the data (generally in the form of time series) and extract features that can be used in the detection phase. The feature extraction algorithms are divided into three groups:

- **ActiveSensing:** A set of functions for managing, visualizing, and extracting features for ultrasonic wave propagation-based active sensing.
- **ModalAnalysis:** These functions take in either a matrix of time series, or frequency response functions, and return modal properties such as mode shapes, natural frequencies, and damping ratios.
- **TimeSeries:** These functions take in a matrix of time series, and return a matrix of feature vectors.

3.3 Feature Classification module

Currently, the classification algorithms work in two phases. The training or learning phase takes feature vectors of *normal* conditions, and builds a model of the normal conditions. This model is subsequently used in the detection or scoring phase to flag future feature vectors as normal/undamaged or abnormal/damaged. The detection functions are organized in three groups:

- **Nonparametric:** No distributional assumption is made about the phenomena generating the undamaged data. Example: kernel density estimators.
- **SemiParametric:** Here the data space is partitioned into multiple cells using any of many possible procedures (e.g. k-means, k-d trees) and a parametric model (e.g. a Gaussian) is learned for each cell (group of feature vectors).
- **Parametric:** The algorithms here are built with an underlying assumption about the phenomena generating the data. For example, one might assume that the underlying undamaged distribution is a Gaussian, in which case we could just use the Mahalanobis distance between points (this is equivalent to using the log-likelihood of a fitted multidimensional Gaussian).

The algorithms from each group come in pairs of "learn" and "score" functions or "train" and "detect". For example in the Parametric group, the `learnMahalanobis_shm` function learns the parameters of a *Mahalanobis distance* function from the training data provided; the `scoreMahalanobis_shm` function will then use these parameters to evaluate the *similarity* of future points with the original training data. The difference between "learn" and "score" pairs or "train" and "detect" pairs, is that "train" and "detect" implies that a threshold is automatically determined by the detection routine for a specified confidence level.

As a Convention: The scores returned by a "score" function are interpreted as follows: the higher the score of a sample, the closer the sample is to the *normal* conditions. This way detection simply consists of thresholding the score values, i.e. every score under a certain threshold is indicative of damage.

Assembling detection routines:

The detection routines are customizable in that the user could pick from various sub-routines to implement a detection algorithm. For instance a Semiparametric routine can be assembled by picking from a choice of partitioning functions, a choice of parametric models (such as a Gaussian) to obtain a detector that consists of first partitioning the data space, then learning the particular parametric model on each cell of the partition.

Default mechanism: The user calls `assembleOutlierDetector_shm` which will navigate the various options available and produce a training routine called

`trainOutlierDetector_<...>` (with appended time stamp). This training routine will take in the training data (under normal conditions) and learn a model of the normal conditions according to the various choices made during assembly, and also produces a threshold for future detections. The learned model and threshold are subsequently used by `detectOutlier_shm` to flag future data as damaged or undamaged. A training routine (`trainOutlierDetector_shm`) is also included as an example of an assembled routine, which implements a semiparametric detector consisting of fitting a Gaussian mixture model to the training data after partitioning with the `kmeans_shm` function.

3.4 Auxiliary module

These functions provide basic services that support the SHM process. Currently these services are divided into two groups:

- **Plotting:** Functions for performing general plotting tasks not specific to any step in the SHM process.
- **Sensor Support:** Sensor Diagnostic functions for assessing piezoelectric sensor functionality through impedance-based methods and Optimal Sensor Placement functions for designing modal analysis-based sensing networks.

4 Documentation

All documentation can be found in the *Documentation* folder of the SHMTools directory. The four documents described below are provided in addition to this one.

Header Specifications

The document *HeaderrSpecs.pdf* provides the standard used for function headers in SHMTools functions.

Parameter Specifications

The document *ParameterSpecs.pdf* provides general parameter standards to facilitate communication between the various group of functions.

Function Library

The document *FunctionLibrary.pdf* serves as a central "help" file for all SHMTools functions.

Example Usages

Extensive documentation is provided as "example usage" documents that walk the user through various ways of using the software package. These are collected in a central document (*ExampleUsages.pdf*). We highlight a few of them below that are good starting examples.

- *exampleDAQ_ARModel_Mahalanobis.html*: Illustrates the full SHM process, from data acquisition to detection.
- *exampleDefaultThresholdingUsage.html*: Shows how to use the "train" and "detect" outlier functions described above.
- *exampleAssembleCustomDetector.html*: Explains how to assemble custom detectors of the various categories.
- *exampleModalFeatures.html*: Illustrates data normalization for outlier detection using modal properties.
- *exampleNLPCA.html*: Demonstrates detection based on the Associative Neural Network algorithm.

5 Examples

The *Examples* folder of the SHMTools directory contains examples for using the SHMTools functions in various forms. Examples are inluded as ".m" file scripts, mFUSE session files, and published scripts in HTML. Standard data sets are also included in the *ExampleData* folder. The folder is organized as follows:

- **ExampleUsageScripts:** Various ".m" files used to create the *Example Usages* described in the last section.
- **mFUSEexamples:** Examples of how to use the mFUSE GUI in conjunction withSHMTools.
- **ExampleData:** Standard data sets in ".mat" files along with import functions to be used with the mFUSE GUI.

6 Importing Data

Data used in SHM analysis can come in many different types and structures. To allow many different forms of data to be used with SHMTools, data import functions are used. Each data import function performs the following three tasks:

- Locate the dataset file's path.
- Load the dataset.
- Restructure data as necessary to match SHMTools parameter standards.

With data import functions, datasets can be stored in many different forms and still be compatible with SHMTools. When writing new data import functions, ensure that the outputs conform to the SHMTools parameter standards as defined in the Parameter Specifications manual. The *ExampleData* folder provides samples of datasets stored as .mat files along with their corresponding data import functions to use as templates.

7 Using mFUSE with SHMTools

mFUSE: Function Sequencer for MATLAB is a Java based graphical user interface for use with MATLAB. mFUSE facilitates the development of analytical processes by allowing users to quickly and intuitively connect MATLAB functions as steps in a sequence. mFUSE will help you easily develop and compare similar processes. Using mFUSE together with SHMTools can simplify your thinking while increasing productivity.

Tutorial for SHMTools and mFUSE

The following tutorial will guide you through the basic steps for building a process from SHMTools functions using the mFUSE interface. This tutorial provides step-by-step instructions to recreate the *Simple Complete Analysis* example provided with SHMTools. New SHMTools and mFUSE users should start here to become familiar with both packages.

1. Download SHMTools.zip and mFUSE.zip from `http://institute.lanl.gov/ei/software-and-data/SHMTools/`
2. Extract SHMTools.zip into desired directory
3. Extract mFUSE.zip into desired directory
4. Install and launch mFUSE (see Part II of mFUSE_Help.pdf manual)
5. Add SHMTools directories to mFUSE Function Library
   (a) Select *Function Library* then *Add Library Path* from mFUSE menu bar
   (b) Read warning and click *OK*
   (c) Navigate into *SHMTools\SHMFunctions* directory
   (d) Click *Open*
   (e) Repeat steps a-d for *SHMTools\Exampes\ExampleData* directory
6. Add *Import 3 Story Structure Dataset* function to sequence
   (a) Click on *Example Data* folder in Function Library to expand the folder
   (b) Double-Click *Import 3 Story Structure Dataset* function to add it to sequence
7. Add QuickStep to sequence to remove channel 1 input measurements
   (a) Click and hold on the *QS* logo just below the sequence list
   (b) Drag the mouse over the Sequence list below Step 1
   (c) Release the mouse to drop the new QuickStep into the sequence

(d) In *Body* box on right panel type:

```
outputChannels = allData(:,2:5,:);
```

(e) Click *OK*

8. Connect *allData* input for QuickStep to *Dataset* output from Step 1

(a) Make sure *QS: outputChannels = allData(:,2:5,:);* is highlighted in Sequence List

(b) Right-Click in Value column for *allData* variable under Inputs on right panel

(c) Move the mouse over *1: Import 3 Story Structure Dataset* to expand menu

(d) Click on *Dataset* output

9. Add and configure *Statistical Moments* function as Step 3 of sequence

(a) Navigate to *SHMFunctions\Feature Extraction\Time Series* in Function Library

(b) Add *Statistical Moments* function to sequence

(c) Connect *Time Series Data* input for Step 3 to *outputChannels* output from Step 2

10. Add and configure *Split Features Into Training and Scoring* function as Step 4 of sequence

(a) Navigate to *SHMFunctions\Feature Extraction* in Function Library

(b) Add *Split Features Into Training and Scoring* function to sequence

(c) Connect *Features* input for Step 4 to *Statistics Feature Vectors* output from Step 3

(d) Right-Click in Value column for *Training Indices* input for Step 4

(e) Click *Enter User Value*

(f) For the user value, type:

```
1:90
```

11. Add and configure *Train Outlier Detector* function as Step 5 of sequence

(a) Navigate to *SHMFunctions\Feature Classification\Outlier Detection* in Function Library

(b) Add *Train Outlier Detector* function to sequence

(c) Connect *Training Features* input for Step 5 to *Training Features* output from Step 4

12. Add and configure Detect Outlier function as Step 6 of sequence

(a) Navigate to *SHMFunctions\Feature Classification\Outlier Detection* in Function Library

(b) Add *Detect Outlier* function to sequence

(c) Connect *Test Features* input for Step 6 to *Scoring Features* output from Step 4

(d) Connect *Models* input for Step 6 to *Models* output from Step 5

13. Add and configure *Plot Scores* function as Step 7 of sequence

(a) Navigate to *SHMFunctions\Feature Classification* in Function Library

(b) Add *Plot Scores* function to sequence

(c) Connect *Scores* input for Step 7 to *Scores* output from Step 6

(d) Connect *States* input for Step 7 to *Results* output from Step 6

(e) Set *State Names* input to a user value of:

```
{'Undamaged','Damaged'}
```

(f) Connect *Thresholds* input for Step 7 to *Threshold* output from Step 6

(g) Set *Flip Signs* input to a user value of:

```
true
```

14. Add and configure *Plot Score Distributions* function as Step 7 of sequence

(a) Navigate to *SHMFunctions\Feature Classification* in Function Library

(b) Add *Plot Score Distributions* function to sequence

(c) Connect *Scores* input for Step 7 to *Scores* output from Step 6

(d) Connect *Damage States* input for Step 7 to *Damage States* output from Step 1

(e) Connect *Thresholds* input for Step 7 to *Threshold* output from Step 6

(f) Set *Flip Signs* input to a user value of:

```
true
```

15. Execute sequence by clicking *Execute* below the Sequence List

16. Open *Simple Complete Analysis* session and compare

(a) Select *File* then *Open Session* from mFUSE menu bar

(b) Save current session if desired

(c) Navigate into *SHMTools\Examples\mFUSEexamples\SimpleCompleteAnalysis* directory

(d) Click on *SimpleCompleteAnalysis.ses* file

(e) Click *Open*

(f) Execute *Simple Complete Analysis* example

