STDetect—Spectrogram Track Detection

User Guide

Thomas A. Lampert
Advanced Computer Architectures Group

Department of Computer Science
University of York
Deramore Lane
York
YO10 3GH

Email: tomalampert@hotmail.com

Website: sites.google.com/site/tomalampert

Contents

Introduction

This toolbox contains the spectrogram track detection algorithm outlined in several papers and my doctoral thesis (see bibliography). This high-level algorithm incorporates information such as temporal track continuity and harmonic structure to improve detection rates. As such, there are a number of steps to take before performing the detection and these are outlined below. The toolbox also contains a number of low-level feature detection algorithms which do not integrate this a priori information and are therefore applicable without any further effort (except to determine appropriate parameter values), the use of these methods is outlined in the section 'Low-Level Detection Algorithms'. The purpose of this release is to allow researchers recreate and verify published results, however, with a little effort these implementations should be generally applicable.

Requirements

The 64-bit version of Matlab does not include a C compiler and, if you are using this version, you need to install Microsoft Visual Studio C++ to compile some of the functions included in the package. Development and testing has been performed using Matlab 2008a/2008b/2009a (and it is expected to work in lower versions) and Microsoft Visual Studio 2005/2008/Visual C++ 2008 Express Edition. If Microsoft Visual Studio is not available, Microsoft Visual C++ 2008 Express Edition can be obtained free of charge from the Microsoft website (see below). The following instructions describe configuring Matlab to use the correct Microsoft Visual Studio compiler.

Microsoft Visual Studio 2005/2008

1. Install Microsoft Visual Studio 2005/2008.
2. Start Matlab, at the command prompt type "`mex -setup`".
3. When prompted "Would you like mex to locate installed compilers [y]/n" enter "`y`".
4. You will be presented with a numbered list of installed compilers. Enter the number corresponding to "Microsoft Visual C++ 2008" or "Microsoft Visual C++ 2005" depending on which is installed.
5. You will then be asked to verify your choices, enter "`y`".

Microsoft Visual C++ 2008 Express Edition

1. Download and install Microsoft Visual C++ 2008 Express Edition from
   http://www.microsoft.com/express/vc/.
2. Download and install the Microsoft Windows SDK from
   http://msdn.microsoft.com/en-us/windows/bb980924.aspx.
3. Place the following files in C:\Program Files\MATLAB\2009a\bin\win64\mexopts\
   .\setup\64-bit \msvc90freeengmatopts.bat
   .\setup\64-bit \msvc90freematopts.bat
   .\setup\64-bit \msvc90freematopts.stp.
4. Replace the following files in C:\Program Files\Microsoft Visual Studio 9.0\VC\bin\
   .\setup\64-bit\vcvars32.bat
   .\setup\64-bit \vcvarsx86_amd64.bat
   .\setup\64-bit\vcvars64.bat
5. Start Matlab and enter the following environment variables at the command prompt:

```
setenv('MSSdk','C:\Program Files\Microsoft SDKs\Windows\v6.1')
setenv('VS90COMNTOOLS','C:\Program Files\Microsoft Visual Studio
        9.0\Common7\Tools')
```

6. Type "`mex -setup`".
7. When prompted "Would you like mex to locate installed compilers [y]/n" enter "`y`".
8. You will be presented with a numbered list containing the compilers located. Enter the number corresponding to "Microsoft Visual C++ 2008 Express Edition".
9. You will then be asked to verify your choices, enter "`y`".

Additional information can be found at www.mathworks.com/matlabcentral/fileexchange/22689.

For the remainder of this user guide it is assumed that the current Matlab directory is set to ".\toolbox\" and that the paths have been initialised by entering `setpaths` at the command prompt.

Recreating Published Results

The toolbox contains a number of scripts that, along with the correct data set (this synthetic data set can be obtained at sites.google.com/site/tomalampert/data-sets), can be used to verify the results that have been presented in my doctorate thesis and publications. These scripts are contained in the './test_scripts' directory. The comments in the header of each describe in which section of the thesis, and which figure, the results of that particular script are presented.

Once you have obtained the synthetic data set used in my thesis unzip it to a suitable location and edit the file 'path.txt' located in the root directory of the toolbox to contain the location of the data set (this should be the location of the directories: 'training_set', 'test_set', and 'templates').

Furthermore, following the execution of these scripts will probably give you a good idea of how the toolbox works.

Example Detections

A number of example spectrograms are included which will allow you to start an analysis and extract some results. To perform these analyses enter the following:

Vertical and oblique track detection

```
setpaths;

spectrogram = load_Spectrogram('./examples/
                               ip_test_data_P1_V_1_4.5_6.dat', 1);
result_v = analyse_Spectrogram(spectrogram, 'pca');
```

Sinusoidal Track detection

```
spectrogram = load_Spectrogram('./examples/
                               ip_test_data_P1_S20_5_7_9.dat', 1);
result_sin = analyse_Spectrogram(spectrogram, 'pca');
```

Low-Level Detection Algorithms

A number of low-level track detection methods have been included in the toolbox, to execute these follow the instructions below. As these are low-level algorithms they do not use a priori information regarding harmonic structure and can therefore be applied to spectrograms without using the signature database (outlined below). The input spectrogram should be in the structure described in the section "Spectrogram File Structure".

An example spectrogram can be loaded by entering the following:

```
spectrogram = load_Spectrogram('./examples/
                               ip_test_data_P1_V_1_4.5_6.dat', 1);
```

Fixed-Scale Bar Detection

```
detection = run_bardetection(spectrogram, length, threshold);
```

where `length` is the length of the filter and `threshold` is the threshold value to be applied (proportional to the values within the spectrogram).

Multi-Scale Bar Detection

```
detection = run_bardetection(spectrogram, length, threshold);
```

as fixed-length bar detection but length can be a vector, i.e. length = [5, 7, 9, 11].

PCA Detection

```
detection = run_pca(spectrogram, threshold);
```

where `threshold` is between 0 and 1 (the most effective range is likely to be a very small number i.e. < 0.1). Note that this is a machine learning technique that requires a training set. This is provided as synthetic spectrograms but this may not be applicable to general problems and, therefore, it may be necessary to provide training data for specific applications (see section "Training Data"), which requires ground truth data in the form of binary images specifying track locations (see examples in './examples/templates').

Nayar detection

```
detection = run_nayar(spectrogram, windowsize, threshold);
```

where `windowsize` is the size of the line segments to detect and `threshold` is typically a value between 0 and 14 (but could be higher).

Training Data

The quality of the training data will ultimately dictate the performance of the algorithm. During development a number of 0 dB spectrograms were used to train the system and these are included in the '.\examples' directory within the toolbox. To change the training data used by the system edit lines 42-44 of the file '.\line_detection\pca\train_PCA_filters.m' and lines 25-28 of '.\line_detection\pca\classifier\train_Gauss.m' to point to the correct training data. As with any supervised learning problem, ground truth data is needed to identify the classes that will be used for training (examples of ground truth data used in STDetect are located in './examples/templates').

Signature Database

The signature database stores information regarding the harmonic relationships between the narrow-band tonals emitted by the sources to be detected. This information is used to enhance detection rates, however, the algorithm can also be used as an individual track detection mechanism. The file '.\@signatureDatabase\database.csv' is an example of a database layout that contains the required fields.
`analyse_Spectrogram.m`, located in the root directory, is the main detection function; you can specify which signature database is to be loaded in the 3rd line:

```
db = signatureDatabase(file_to_be_loaded);
```

The fields in the signature database are:

`Type` = a name corresponding to the signature.
e.g. 'Boat'

`Fundamental` = the expected fundamental frequency of the harmonic set in Hz.
e.g. '120'

`Search Range` = the expected variation around the fundamental frequency (percentage).
e.g. '10'

`Harmonic Set` = multipliers of the fundamental frequency which make up the signature (must start with 1 - corresponding to the fundamental frequency).
e.g. '1 2 2.5 4 5'

`Mask` = a Boolean set which can be used to ignore specific harmonic multipliers in the signature.
e.g. '1 1 0 1 1'

For example:

Type	Fundamental	Search Range	Harmonic Set	Mask
Boat	120	10	1 2 2.5 4 5	1 1 1 1 1

If individual tracks, as opposed to a signature, are to be detected, the harmonic set and mask should both be set to 1.

Spectrogram File Structure

The `load_Spectrogram` function reads the data from a binary spectrogram file. The file should have the following structure:

Data	**Data Format**
`Spectrogram Size (time (t) x frequency (f))`	`2 x uint32`
`Frequency Resolution`	`1 x float32`
`Time Resolution`	`1 x float32`
`Spectrogram Intensity Values`	`t x f x float32`

The output of the `load_Spectrogram` function is a Matlab structure containing the following fields:

`spectrogram`
- `|------z_spec` - a `t` x `f` matrix containing the spectrogram intensity values
- `|------t_res` - the resolution of the spectrogram's time axis
- `|------f_res` - the resolution of the spectrogram's frequency axis
- `|------template` - a `t` x `f` matrix containing the ground truth template*
- `|------size` - a 1 x 2 vector containing the size of the spectrogram [t f]
- `|------snr` - the signal-to-noise ratio of the spectrogram+.

* If there exists a ground truth template file describing the locations of the tracks in the spectrogram it can be loaded using the second input of the `load_Spectrogram` function, e.g. `spectrogram = load_Spectrogram(file_to_be_loaded, 1)` otherwise this input should be set to 0 and the template field will be empty, e.g. `spectrogram = load_Spectrogram(file_to_be_loaded, 0)`.

+ This is only available if a ground truth template is loaded with the spectrogram.

Algorithm Parameters

The parameters which control the detection process are stored in a class called `snake_param`, the default settings are initialised automatically when the `analyse_Spectrogram` function is called. These values are derived using synthetic data and therefore may not work with the real data. They can be changed by initialising an instance of the `snake_param` class in the workspace (or a script), such as

```
p = snake_param;
```

and passing the object to the `analyse_Spectrogram` function when initialising the analysis (see next section). The following describes changing parameters relevant to the detection process.

The amount of force that pushes the snake along is controlled by the parameter `WalkRate` and can be changed using the following command

```
p = set(p, 'WalkRate', value);
```

where

`value` is a real number between 0 and 1, the default value is 0.41.

If the scaling (and perhaps translation) parameters are tuned appropriately and the snake is still missing a signature then `WalkRate` can be reduced. In the opposite case, the snake is detecting too many signatures, this can be increased.

The snake has internal energies which control the shape of the track that the snake can model; these also give some noise invariance properties. There are two versions of these included, the first is taken from the original active contour algorithm and the second is the more up-to-date Perrin energy (recommended). These are changed through the command

```
p = set(p, 'InternalEnergy', option);
```

where

`option` = 'original'
'perrin' (default).

The amount of influence that these energies have on the snake is controlled by the parameters α and β for the original case and β for the Perrin case, these are changed through

```
p = set(p, parameter, value);
```

where

`parameter` = 'Beta'
'Alpha'

`value` is a real number between 0 and 1.

The values which have been used in experimentation are α = 0.1 and β = 0.2 for 'original' and β = 0.2 for 'perrin' - these are the default values.

The influence of that the external energy (the information taken from the spectrogram) is controlled by the parameter γ, this can be adjusted through:

```
p = set(p, 'Gamma', value);
```

where

`value` is a real number between 0 and 1.

The value used in experimentation is γ = 1 as this derives the maximum amount of information from the spectrogram. This value is recommended and is the default, however, under conditions where there is a large amount of noise the value could be reduced (this has not been tested).

The length of the snake, the amount of time steps integrated when performing a detection at a single time step, can be controlled through the `SnakeLength` parameter:

```
p = set(p, 'SnakeLength', value);
```

where

`value` must be a positive integer and 20 is the default value. The longer the snake is the more reliable the detection. However, this also increases the detection lag.

The size of the window (in pixels) that the external energy uses to calculate the local presence of a feature is controlled by two parameters `WindowHeight` and `WindowWidth`. These are set via the commands

```
p = set(p, 'WindowHeight', value);

p = set(p, 'WindowWidth', value);
```

where

> `value` must be a positive odd integer, the default values are `WindowHeight = 21` and `WindowWidth = 3`. The higher the values the more information that is integrated in the external energy and this results in more reliable detections. However, regarding the height parameter, detection will not be possible within the points in which the window falls out of the spectrogram. For example, if the window is set to a height of 21 pixels, detection will not be possible in the top and bottom 10 pixels of the spectrogram (the window is centred on each snake point).

Initialising the Analysis

Once the parameters and the training data described previously have been set the spectrogram analysis can be started. To do this, first ensure that the mex files have been compiled on your system, to do this enter `make`. Next ensure that a spectrogram is loaded by entering

```
spectrogram = load_Spectrogram(file_to_be_loaded);
```

and the analysis can be initialised using the `analyse_Spectrogram` function. This function takes the following inputs

```
result =

 analyse_Spectrogram(spectrogram, transformation, range, parameter_values);
```

the bare minimum that is needed start the function is

```
result = analyse_Spectrogram(spectrogram, 'pca');
```

which will call the analysis with the default parameter settings. If you would like to change the parameter values then they need to be changed as described in previous sections and then passed to the function, such as

```
result = analyse_Spectrogram(spectrogram, 'pca', [], p);
```

As noted previously, it is best to select sections of the spectrogram for classifier training which are at the beginning or end. To omit those chosen from the analysis use:

```
result = analyse_Spectrogram(spectrogram, 'pca', [start end], p);
```

where `start` is the position (in pixels) at which to start the spectrogram analysis and `end` is the position to end the analysis.

The output of `analyse_Spectrogram`, `result`, is a binary matrix the size of the input spectrogram which contains 1s at points of detection and 0s otherwise. The graphical user interface outputs the time and the type of source detected to the user.

License

Bibliography

For further information regarding the active contour track detection algorithm the following are recommended (these papers are available through the website http://sites.google.com/site/tomalampert/publications).

M. Kass, A. Witkin and D. Terzopoulos, “Snakes: Active Contour Models”, in: International Journal of Computer Vision 1 (4): 321-331, 1988.

T. Lampert and S. O’Keefe, “On the Detection of Tracks in Spectrogram Images”, Pattern Recognition: 46 (5): 1396–1408, 2013.

T. Lampert and S. O'Keefe, “A Detailed Investigation into Low-Level Feature Detection in Spectrogram Images”, Pattern Recognition 44 (9): 2076–2092, 2011.

T. Lampert and S. O’Keefe, “An Active Contour Model for Spectrogram Track Detection”, Pattern Recognition Letters 31 (10): 1201-1206, 2010.

T. Lampert and S. O’Keefe, “A Survey of Spectrogram Track Detection Algorithms”, Applied Acoustics 72 (2): 87-100, 2010.

T. Lampert, “Spectrogram Track Detection: An Active Contour Algorithm”, PhD Thesis, University of York, 2010.