

A Framework for Automatic Large-Scale Testing and Characterization of Signal Processing Algorithms

Kenneth Holladay
Southwest Research Institute®
San Antonio, TX 78238

Kay A. Robbins
University of Texas at San Antonio
San Antonio, TX 78249

ABSTRACT*

Performance analysis of signal processing algorithms should yield insight into expected performance as a function of all varying factors including algorithm parameters, signal characteristics, and transmission channel propagation effects. This paper presents a test framework for characterizing signal processing algorithm behavior. The framework provides functions to automate the evaluation process including creating large numbers of test files, running these files through the algorithm under test, collecting data, and analyzing the results. Automating this cycle allows rapid testing and evaluation of algorithm enhancements, as well as identifying the significant factors that affect performance. We demonstrate this technique by comparing and characterizing two different published symbol rate estimation algorithms.

INTRODUCTION

Analyzing the performance of a digital signal processing (DSP) algorithm should yield insight into its expected behavior as a function of all of the factors that can vary in its deployment environment. These factors may include algorithm tuning parameters (such as maximum likelihood thresholds), signal characteristics (such as symbol rate, modulation type, pulse shaping, etc.), and transmission channel propagation effects (such as additive white Gaussian noise or fading). For automated surveillance applications in non-cooperative environments, the list of applicable factors can be very large.

Mathematical analysis of an algorithm that accounts for all relevant factors is unlikely to yield a reasonable closed form solution, and the operational effect of any simplifying assumptions must be carefully considered. For this reason, most researchers include simulation testing in their evaluation. However, the significant variations in the planning and execution of published simulations renders it difficult to make direct quantitative comparisons.

The complexity of the interacting algorithm, signal and channel factors suggests that techniques pioneered in

experimental algorithmics [1] could yield useful insight into performance. Experimental algorithmic methods emphasize several key concepts that include clearly specifying the testing goal, articulating parameter variations and any hidden algorithmic factors, carefully constructing large test sets that span the problem domain, and systematically evaluating the performance results using statistical techniques.

Based on these principles, this paper presents an algorithm exploration framework (ALEF) for characterizing signal processing algorithm behavior. The framework provides functions to automate the evaluation process including creating large numbers of test files, running these files through the algorithm under test, and analyzing the results. Automating this cycle allows rapid testing and evaluation of algorithm enhancements, as well as identifying the significant factors that affect performance. We implemented the software framework in a commercially available programming and modeling environment (MATLAB® and Simulink®[2]) that includes a rich set of intrinsic digital communication functions.

After presenting an overview of the framework and its design philosophy, we demonstrate its capabilities by using it to quantitatively compare the performance of two Phase Shift Key (PSK) symbol rate estimation algorithms. This includes clearly delineating the expected signal domain, cataloging all of the algorithm parameters, collecting the large volume of test data, and analyzing the results. Visualization techniques based on the key factors portray the relative impact of the largest factors on overall algorithm reliability.

FRAMEWORK OVERVIEW

The automated test framework is based on a three-phase functional approach (test generation, test execution, and analysis) as illustrated in Fig. 1. Each phase stores its output in a format that facilitates systematic identification, retrieval, and manipulation by the next phase, including all relevant input factors and generated results.

The Test Generator uses a collection of fully specified domain factors to produce a full-factored set of test files. It records the domain properties of the test set in an XML

* This work was sponsored by the Southwest Research Institute Advisory Committee for Research (ACR).

signal library file [3], and it encodes the specific factor values associated with each signal in the individual file names.

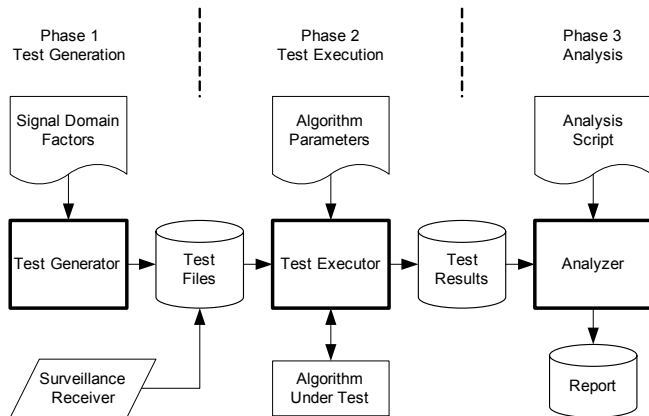


Figure 1 Test Framework Block Diagram

The Test Executor automatically invokes the algorithm against each test for every combination of the algorithm parameters. It collects all pertinent test input as well as the algorithm response for each test into a generalized, indexed structure that is saved in a single test results file.

In the final analysis phase, a simple script applies ALEF analysis tools to the test results and produces a report. The tools include functions to produce summaries of general performance, tables of results organized by test factors, and visualizations such as probability density curves and ROC (receiver operating characteristic) curves [4]. The framework also contains a series of simple functions that facilitate saving the analysis results as an HTML formatted report. There are functions for inserting section headings, data values, tables, and graphs.

EXAMPLE APPLICATION

To illustrate the application of the framework, we consider the selection of a PSK symbol rate algorithm for an existing surveillance system. The first candidate uses the derivative of the phase angle (DPDT) [5], while the second uses a multi-scale continuous wavelet transform (MCWT) [6]. When the system detects a PSK signal, it will deliver to the algorithm two seconds of data sampled at a rate of 8000 complex samples per second.

TEST GENERATION

First, we employ knowledge of the algorithm operational domain to establish quantitative bounds for all pertinent signal and transmission channel properties. Any property that is not constant is potentially a factor that can affect the algorithm performance. The combination of these factors

defines a multidimensional problem domain space. A comprehensive analysis of the algorithm requires a body of test signals whose range of factor values comprise a statistically significant population of this space [7]. These test signals can be either synthetic or recorded, as long as all factor values are known for every test signal.

The sheer volume of files required to cover every possible value of every signal parameter would be enormous. We addressed this problem in the framework by creating a series of signal generation functions loosely categorized by signal modulation type. Each function accepts vectors of signal property values and produces a full factorial set of baseband signal files containing all combinations of these values. If desired, the functions can also create several variations of each signal with different symbol sequences and noise. The output of this phase consists of simulated analytic (complex valued) signals stored in a standard WAV file format with I data in channel 1 and Q data in channel 2.

The research work that stimulated this investigation [8] focused on non-cooperative reception of digitally modulated signals in the High Frequency (HF) communication band. These signals are typically limited to channels with a bandwidth of 5 kHz or less. Table 1 lists some of the common signal properties that can affect the performance of a blind symbol rate estimation algorithm. The values used represent expected realistic ranges for the target surveillance system. The test set also includes complexities that arise in practice but that are often not considered in published studies, such as pulse shaping and symbol rates that are not divisors of the sample rate.

Table 1 - HF Digital Signal Properties

Property	Example HF Values	Values Used
Modulation type	FSK, MSK, PSK, DPSK, OQPSK, QAM, ASK	PSK
Pulse shape	None, raised cosine (RC), root RC (RRC), Gaussian	None, RC, RRC
Excess bandwidth (rolloff)	Limit: 0.00 to <1.00. Typical: 0.10 to 0.35	0.1, 0.2, 0.35
Symbol rate	Typical: 10 to 2400 symbols per second	50, 100, 300, 1280, 2400
Symbol states	2, 4, 8, 16	2, 4, 8
Signal to noise ratio ¹ (SNR)	Practical range: 0 to 60 dB	9, 12, 16, 20, 40
Frequency offset from baseband	Possible range: 0 to 500 Hz	0, 50, 100

For each combination of the six multiple-valued test signal factors we generated 10 random variations of noise and

¹ AWGN bandwidth measured at approximately the Nyquist sample rate.

symbol content, yielding an ensemble of 15,750 test signals. The framework manages frequency offset by heterodyning the baseband signal before calling the algorithm. Generating the test files requires a small MATLAB script to set up the parameters and options before calling the framework functions. An excerpt from this script might look like the following.

```
sLibPath = 'C:\SymRateLib';
kOptions.nVersions = 10;
kChannelParam.arSNRdB = [9 12 16 20 40];
kSignalParam.anStates = [2 4 8];
genPSKfiles(kSignalParam, kChannelParam, ...
            kOptions, sLibPath);
```

The generated signal files along with the XML signal library file are placed in the specified library path. The file name encoding allows framework functions to quickly sort through the large volume of files to select subsets for specific experiments. For example, the following file contains a PSK, 2 level, 50 baud, signal at 9dB SNR, 8000 Hz sample rate, no carrier offset, and no pulse shaping.

```
PSK_12_b50_s8000_c0_n9_pNone_r0_v20030515T193254F1.wav
```

The following script uses a framework function to select all files from a library directory that have 8000 Hz sample rate:

```
sFiles=fw_listdir( sLibPath );
sFiles=fw_listprune( sFiles, 'samplerate', '8000' );
```

TEST EXECUTION

In the Test Execution phase, a framework function processes the previously identified body of test signals through the algorithm under test. The input to this phase must include all algorithm specific parameters used to adjust performance. Incorporating these in the test process requires iteratively executing the algorithm against each test signal using an appropriate range of parameter values.

Most published symbol rate detection algorithms [9][10][11][12] have a similar structure consisting of two computational stages [13]. The first stage develops a feature space by applying a transform to the digitized signal. The goal of the transform is to locate and emphasize symbol transitions. The second stage analyzes this feature space to calculate a periodic value that is the estimate of the symbol rate. To determine the applicable algorithm specific parameters, we consider the two stages independently.

The first stage of the DPDT algorithm transforms the input signal by taking the magnitude of the derivative (first difference approximation) of the signal's instantaneous phase angle. As seen in this MATLAB expression, there are no adjustable parameters. In these snippets, x is the

analytical (complex) signal vector, and x_{1o} is the first stage output.

```
xphase = unwrap( angle( x(1:end) ) );
x1o = abs(diff(xphase));
```

The second algorithm uses the multi-scale continuous wavelet transform, expressed in MATLAB as follows.

```
c = cwt( x, scales, wavelet );
x1o = c.*conj(c);
if numel(scales) > 1, y = sum(x1o), end
```

Note that there are two parameters, the scales and the wavelet. Based on previous work with this algorithm [14], we selected the db6 wavelet and scale values of 2, 4, 6.

The second stage is a greater challenge. Most published descriptions determine the periodicity associated with the symbol rate by using a Fast Fourier Transform (FFT) of the first stage output and selecting a peak. Parameters associated with this hidden peak search algorithm are rarely disclosed, yet they affect practical performance and must be characterized. Rather than attempting to reproduce specific second stage implementations from sparse descriptions, we developed a common second stage for both algorithms.

Our final design of the second stage contains several adjustable parameters including the desired baud resolution (which affects the size of the FFT), whether to use a Hamming window prior to each FFT to reduce spectral leakage, and whether to use an overlap and add of the FFT results to enhance the desired peaks. This last option also requires a low pass filter on the final FFT accumulation to remove any trend at the low frequencies, which introduces a parameter for the width of the moving average.

When we used the framework to characterize these second stage parameters, the results indicated that the performance of the peak search scaled uniformly with the different first stage transforms. This was an important observation since it implied that the performance of the second stage would not mask the performance of any of the first stage transforms. It also implied that we could use the same fixed set of second stage parameters, and any variations in performance would result solely from the interaction of the first stage transform with the variations in signal factors.

Based on the above, an excerpt from the MATLAB script to process the test files through the DPDT algorithm (function name 'sr_dpdt') would look like this.

```
sFiles = fw_listdir( sLibPath );
rStartTimeSec = 0.6;
rDurationSec = 2.0;
rOffsetHz = [0 50 100];
```

```

kAlgParam(1).sPeakOpt = 'orh';
kAlgParam(1).rMinSymbolsPerSec = 20;
fw_runtest( sFiles, rStartTimeSec, rDuration,...
            rOffsetHz, 'sr_dpdt', kAlgParam );

```

```

pulseShape: 'None'
rolloff: 0
SNRdB: 12
rayleighCondition: 'None'
ricianCondition: 'None'

```

Note that the algorithm name is a string. The test executor uses a fixed algorithm calling convention of the form:

```

[kResult kAlgParam] =
    sAlgName(x, Fs, kAlgParam, kActual);

```

Where x is the input signal and Fs is the sample rate in Hz. This is generic enough that the test executor (fw_runtest) requires no explicit knowledge of the contents of the structures that carry algorithm parameters (kAlgParam), actual signal information (kActual), and test results (kResults). In our example, the algorithm returns not only the estimated symbol rate, but also several standard statistical measures of dispersion and central tendency for the first and second stage outputs, including the mean and standard deviation.

While each invocation of the test executor function may process thousands of tests, the results are saved in a single file. To create an organized format that is easy for the analysis functions to access and interpret, we use a three level hierarchical structure, fixing the field names only at the second level. The first level is a single variable (kTest) that holds all aggregate data from the single invocation. This variable contains the second level required structure names that are indexed by test number (kAlgParam, kActual, and kResults). The contents of these structures (the third level) consists of named fields, but by using run-time field name discovery, the framework does not impose any fixed names or data format for those fields. For example, the elements of kTest produced by a single run of the MCWT algorithm against 5250 signal files with variations to produce 15750 tests looks like this:

```

kTest =
    sAlgName: 'sr_wavelet'
    sAlgDesc: 'Multiscale Wavelet Transform'
    sTestDate: '29-Mar-2004 17:07:45'
    sFiles: {5250x1 cell}
    nFileIdx: [1x15750 double]
    kActual: [1x15750 struct]
    kResult: [1x15750 struct]
    kAlgParam: [1x15750 struct]

```

For each test, the actual values, parameters, and results are accessed by test index number:

```

kTest.kActual(1) =
    sampleRateHz: 8000
    durationSec: 2.0480000000000000
    startTimeSec: 0.6000000000000000
    offsetFreqHz: 0
    modulation: 'PSK'
    symbolsPerSec: 100
    numStates: 2
    modIndex: []
    phaseContinuity: 'Disc'

```

```

kTest.kResult(1) =
    symbolsPerSec: 99.97977557377107
    rComputeSec: 0.1000000000000000
    rXformNormMean: 0.05007506092908
    rXformNormStdDev: 0.07430819566016
    rBinWidthHz: 1.95407914020518
    idxBaudPeak: 52
    rBaudPeak: 2.652773560544409e+003
    rMaxPeak: 2.652773560544409e+003
    idxMaxPeak: 52
    rMeanAbsDev: 14.80192075274855
    rNormMean: 0.00316318933798
    rNormTrimMean10: 2.327396802381751e-004
    rNormStdDev: 0.03292030445029
    rNormIQR: 1.657681819373304e-004
    rNormMeanAbsDev: 0.00557979051545

kTest.kAlgParam(1) =
    rMinSymbolsPerSec: 20
    sPeakOpt: 'orh'
    sWavelet: 'db6'
    nScale: [2 4 6]
    sSignalOpt: 'c'

```

ANALYSIS PHASE

After executing the tests, another MATLAB script directs the framework analysis functions, which often require field names as part of their calling convention. The following script excerpt analyzes a kTest structure and produces a report summarizing test information and algorithm performance (% correct responses). This code deems a symbol rate estimate correct when the error is less than 2% of the actual value.

```

sField = 'symbolsPerSec';
sType = 'PctErr', rPass = 2;
[bPass, hfig, sReport] =
    fw_errsummary( kTest, sField, sType, rPass );
[rFactor, kInfo, sReport] = fw_factor( kTest );

```

After executing the above, the bPass variable is a Nx1 vector containing a 1 for each test that passed and a 0 for each test that failed (where N is the number of tests). rFactor is a NxM matrix with a column for each of the M multiple valued factors, and kInfo is a structure containing information about those factors.

To understand the flexibility afforded by this approach, consider that internally, the fw_errsummary function requires only a field name string to calculate the error for any continuous (non-discrete) value recorded by the test. After executing the following single line, rPctErr will be a Nx1 vector containing the symbol rate percent error for each test.

```

rPctErr = (kTest.kActual(:).(sFieldName) - ...
           kTest.kResult(:).(sFieldName) ) /
           (kTest.kActual(:).(sFieldName) * 100);

```

Table 2 combines the result tables from analyzing both algorithms with respect to one of the most significant performance factors: symbol pulse shaping. For the specified range of values, each of the algorithms demonstrated reasonable immunity to variations in offset frequency, number of symbol states, and noise.

Table 2 – Percent correct estimate as a function of symbol rate And pulse shape

Algorithm		DPDT			MCWT		
Pulse shape		None	RC	RRC	None	RC	RRC
Symbol Rate	50	79.6	77.8	85.0	100.0	2.15	18.9
	100	100.0	97.3	98.4	100.0	2.37	21.3
	300	100.0	100.0	100.0	100.0	13.7	21.3
	1280	100.0	100.0	100.0	100.0	99.8	100.0
	2400	100.0	100.0	100.0	100.0	98.9	100.0

DEVELOPING A PERFORMANCE PREDICTOR

Overall, it appears that DPDT performs better than MCWT. However, DPDT is not 100% successful under all conditions. We can use the statistical data collected by the framework to explore the possibility of a quality metric that would indicate the probability of a correct symbol rate estimate. Consider that since the first stage transform emphasizes symbol transitions, statistical conditions in the subsequent FFT might indicate the presence of a strong symbol rate peak. Intuitively, the normalized dispersion (standard deviation) and central tendency (mean) of the bin magnitude values should both be small. Using the framework functions, the following MATLAB script excerpt produces a probability density curve and an ROC curve based on the normalized mean of the FFT bins.

```
rMetric = fw_metric(kTest, 'val', 'rNormMean');
fw_plottedensity( rMetric, bPassed );
ROC = fw_plotroc(bPass, rMetric, rFactor, kInfo);
```

Fig. 2 shows the resulting probability density curves when using the normalized mean of the FFT bins to predict whether the algorithm has computed a correct symbol rate (pass) or an incorrect symbol rate (fail). Note that while the peak is large and sharp for the passed values, there is significant area overlap between the two curves.

By contrast, fig. 3 shows the probability curves for MCWT using the same pass/fail metric. While the passed curve has a long tail under the fail curve, the area under the tail is very small.

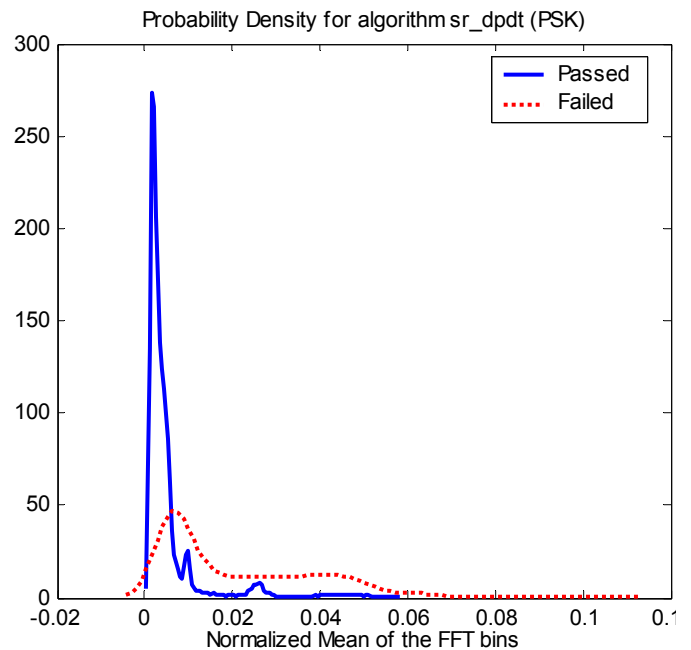


Fig. 2 – Probability density curves for DPDT using the normalized mean of FFT bins to predict pass/fail

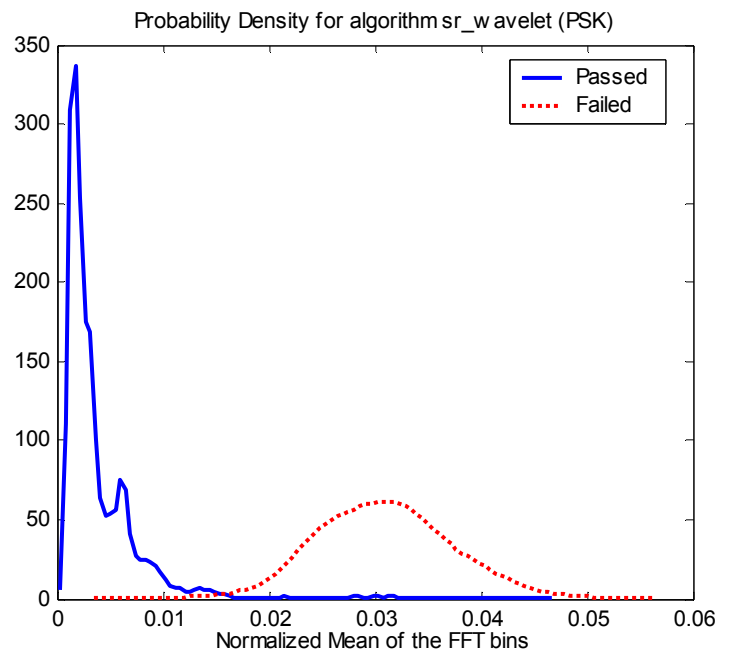


Fig. 3 – Probability density curves for MCWT using the normalized mean of FFT bins to predict pass/fail

To illustrate the potential operating effectiveness of this metric, fig. 4 shows a comparison of the resulting ROC curves for both algorithms.

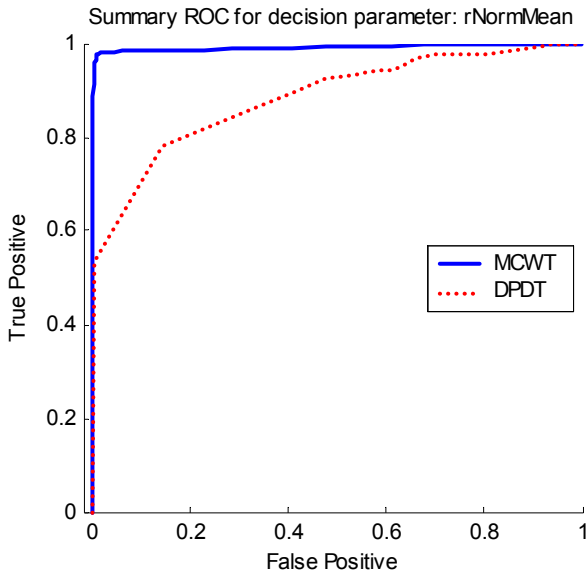


Fig. 4 – ROC curves for MCWT and DPDT using the normalized mean of FFT bins and the standard deviation of the FFT bins.

An example operating point might be a false positive rate below 1%. To achieve this with MCWT, we select a threshold value for the normalized mean of 0.016, which gives a true positive rate of 98%. The same operating point for DPDT requires a threshold of 0.0033 and yields a 65% true positive rate. This metric works very well for MCWT but only marginally for DPDT.

DISCUSSION

The criteria for selecting one algorithm over another to deploy in an actual system must include some measure that reflects performance across the entire anticipated problem domain. This necessarily implies conducting test and analysis iterations involving large numbers of test signals. This paper demonstrates the feasibility of using a generalized framework to conduct this type investigation. With its three-phase approach to the problem, the ALEF framework simplifies the iterative process of constructing test signals, executing the tests, and analyzing the results. The framework functions are general enough to support a wide variety of algorithm testing requirements, yet they contain sufficient high-level specificity to facilitate rapid data analysis.

For our example comparison of the two symbol rate algorithms, we have shown that using the ALEF tools to identify the most important signal and algorithm factors can lead to a better understanding of algorithm performance. For both algorithms, we were able to quickly validate the performance claims of [5] and [6]. However, both references excluded pulse shaping in their analysis, which seems to give MCWT an advantage over

DPDT at lower symbol rates. In practice, when pulse-shaped symbols are considered, DPDT does better overall, but lacks a simple metric for quality indication.

REFERENCES

- [1] B.M. Moret, "Towards a discipline of experimental algorithms," 5th Proceedings of the 5th DIMACS Challenge Workshop, 1996.
- [2] Mathworks, <http://www.mathworks.com/>, 2003.
- [3] K. Holladay, K. Robbins, D. Varner, "An XML schema for digital communication signal research," XML Conference 2002, Baltimore, MD, December 2002.
- [4] H. Van Trees, *Detection, Estimation, and Modulation Theory*, New York, NY: John Wiley & Sons, pp. 19-52, 2001.
- [5] R.J. Mammone, R.J. Rothaker, C.I. Podilchuk, "Estimation of carrier frequency, modulation type, and bit rate of an unknown modulated signal," IEEE International Conference on Communications (ICC '87), vol. 2, pp. 1006-1012, June 1987.
- [6] Y.T. Chan, J.W. Plews, K.C. Ho, "Symbol rate estimation by the wavelet transform," IEEE International Symposium on Circuits and Systems (ISCAS 97), vol. 1, pp. 177-180, June 1997.
- [7] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, New York, Springer-Verlag, pp 22-27, 2003.
- [8] K. Holladay, M. Koets, A. Burmeister, R. Dollarhide, "A configurable signal analyzer for embedded systems," Military Communications Conference Proceedings (MILCOM) 2003.
- [9] D.E. Reed, M.A. Wickert, "Performance of a fourth-power envelope detector for symbol-rate tone generation," IEEE International Conference on Communications (ICC'88), vol. 2, pp. 946-950, 1988.
- [10] B.S Koh, H.S. Lee, "Detection of symbol rate of unknown digital communication signals," Electronics Letters, vol. 29, issue 3, pp. 278-279, Feb. 1993.
- [11] J.A. Sills, J.F. Wood, "Application of the Euclidean algorithm to optimal baud-rate estimation," Military Communications Conference Proceedings (MILCOM '96), vol. 3, pp. 1015-1019, 1996.
- [12] W. Su, J. Kosinski, "A survey of digital modulation recognition methods," International Signal Processing Conference, March 2003.
- [13] A.W. Wegener, "Practical techniques for baud rate estimation," International Conference on Acoustics, Speech, and Signal Processing (ICASSP), vol. 4, pp. 681-684, 1992.
- [14] K. Holladay, K. Robbins, "Experimental analysis of wavelet transforms for estimating PSK symbol rate," IASTED Conference on Communication Systems and Applications, July 2004.