REAL-TIME RECOGNITION OF TIME-SERIES PATTERNS

Jeffrey P. Morrill
Jonathan Delatizky
BBN Systems and Technologies
A Division of Bolt, Beranek, and Newman Inc.
10 Moulton Street, Cambridge, MA 01238

KEYWORDS

pattern recognition, real-time analysis, signal processing, time-series data

ABSTRACT

This paper describes a real-time implementation of the pattern recognition technology originally developed by BBN [Delatizky et al] for post-processing of time-sampled telemetry data. This makes it possible to monitor a data stream for a characteristic shape, such as an arrhythmic heartbeat or a step-response whose overshoot is unacceptably large. Once programmed to recognize patterns of interest, it generates a symbolic description of a time-series signal in intuitive, object-oriented terms. The basic technique is to decompose the signal into a hierarchy of simpler components using rules of grammar, analogous to the process of decomposing a sentence into phrases and words.

This paper describes the basic technique used for pattern recognition of time-series signals and the problems that must be solved to apply the techniques in real time. We present experimental results for an unoptimized prototype demonstrating that 4000 samples per second can be handled easily on conventional hardware.

INTRODUCTION

When monitoring a real-time data stream, analysts often need to recognize patterns in a signal as part of their data reduction and analysis problem. The need may be particularly acute if the analyst's decisions are time-critical. In many cases, analysts fulfill this need using simple methods, such as threshold detection applied to a signal. In other cases, however, successful interpretation of the data depends on being able to recognize patterns in the data to extract the features of interest.

For the past several years, BBN has been developing an expert system called FAES for the Navy for post processing of telemetry data [Delatizky et al; Jeffreys]. One of the features that makes FAES unique is its use of pattern recognition to generate qualitative descriptions of the data that are natural and familiar to the user. We have found that pattern recognition is a powerful tool for data reduction that offers value for a wide variety of applications. Recently we have extracted this feature from FAES and modified its algorithms such that it can work in real time.

Two approaches to pattern recognition are in common use: discriminant and syntactic. Discriminant approaches extract a vector of characteristic features from a pattern; the recognition of the pattern is found by the location of its feature vector in feature space [Duda & Hart]. When applied to time series signals that have time-varying shape and structure, the number of characteristic features can be impractically large. Syntactic approaches solve the latter problem by describing the complex pattern in terms of a hierarchical composition of simpler subpatterns. Syntactic approaches use small sets of simple pattern primitives in combination with combinatorial rules (a grammar), so that it is possible to recognize an infinite set of patterns [Fu]. *Parsing* is the process of applying the grammar. In this paper, we discuss implementing a real-time pattern recognition unit called the Real-Time Parser that is based on a syntactic approach.

Most existing tools for pattern recognition implement discriminant methods. Because these are hard to apply to time-varying signals, common practice is either to develop solutions in-house or to employ people to recognize patterns by eye. An in-house solution may solve a particular need, but lacking reusable tools, it can be expensive to implement and it may not be general enough to solve future needs without periodic redesign. A simpler approach is to use human analysts to look at graphs of data, searching for the patterns of interest. Although people are good at finding patterns in noisy or complex data, this task can be tedious and labor-intensive. If the situation requires a response time within milliseconds, or the amount of data to be monitored is prohibitively large, a human solution is out of the question .

THE BASIC TECHNIQUE

The purpose of the Real-Time Parser is to examine a time series signal and summarize its shape as a symbolic data structure. It generates a high level abstract representation of the raw data in terms of shape features such as flat, ramp, step-response, and spike.

Prior to applying the parser in real time, there is an off-line, preparatory stage where an analyst must define the expected shape of the input signal, A library of existing shape definitions is available. Analysts combine library definitions with numerical

parameters to form new combinations (a grammar) that specify the characteristic patterns of their data.

We divide parsing into two independent tasks: segmentation and grouping. Segmentation is a process of dividing an interval of data into segments that are uniform with respect to some statistical quality. A simple example is flatness: a segment whose values lie within some threshold distance of a baseline. We use the term *primitive* to refer to a kind of shape that is recognized using segmentation. Grouping is a process of organizing contiguous primitives into coherent patterns that are pertinent to a particular problem-solving task. Once formed, a group may include other groups as subgroups. An example of a group is a (second-order) step response: either an incline or a decline, followed by an optional overshoot, followed by a flat. We use the term *pattern* to refer to a kind of shape that is recognized by grouping primitives and subgroups. We use the term *shape* to refer to both patterns and primitives.

Figures 1 and 2 illustrate the basic technique. **Figure 1a** shows a time series segment having the form of a classical step-response. **Figure 1b** shows the result of segmenting the data into a sequence of *Flats*, *Rises*, and *Drops*. **Figure 2** shows the result of grouping the primitive segments into a pattern. This structure is known as a parse tree and represents a hierarchical decomposition of the pattern. (Of course since this example is a classical step-response, we could have applied control systems theory to provide an equation that fits the data. The parser, however, can be applied to the far more common case where the analyst does not have a model but can describe the expected pattern in qualitative terms.)

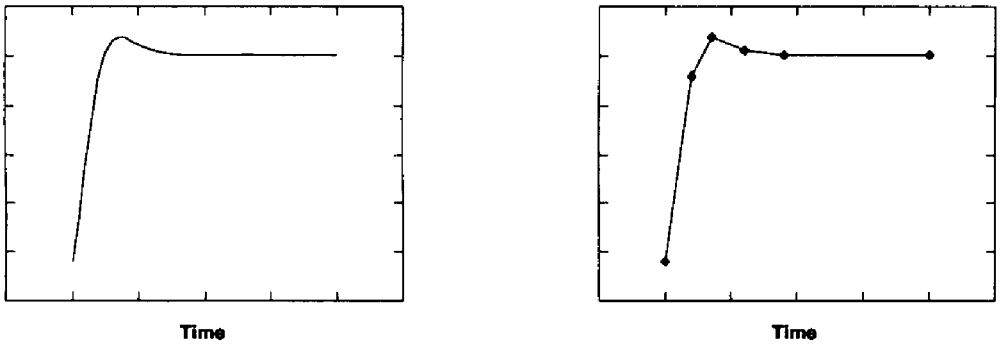

Figure 1. The parser segments raw data into pieces that are constant with respect to some statistical quality, such as slope. (a) Data illustrating a classical step-response curve. (b) Result of segmenting the data into a series of straight-line approximations.

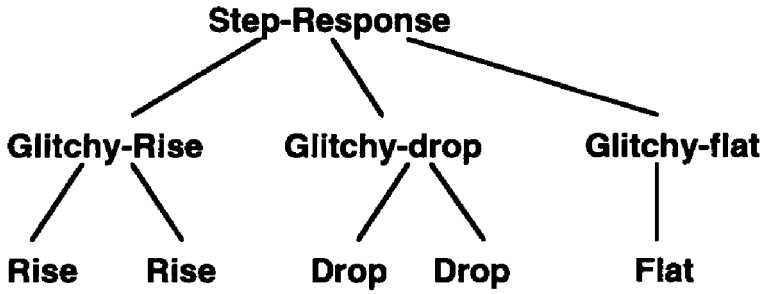

Figure 2. The parser produces results in the form of a parse tree, in this case grouping the segments shown in **Figure 1b** into a Step-Response pattern. The five leaf nodes of this tree correspond, consecutively, to the five line segments shown in **Figure 1b**.

The algorithms used for segmentation can be varied and heterogeneous. In our applications, we have worked mainly with flats and ramps. For flats, we employ the cumulative sum edge detection algorithm [Hinkley]. For ramps, we employ a variation of the same algorithm to detect changes in slope rather than changes in the mean. There are good reasons to investigate other algorithms, such as neural networks, that are sensitive to other signal statistics. Our goal is to develop a library of such algorithms that users can combine into arbitrary groupings.

The central algorithm used for grouping is Syntactic Pattern Recognition [Fu]. Syntax is the way in which words are put together to form phrases, clauses, or sentences. Rules of syntax, in the form of a grammar, are used to describe the way in which pattern primitives combine to form higher level patterns. People write grammars to describe the patterns in their data, and they express the grammars in a special pattern recognition language. Grammars are processed using a technique called *parsing*, hence the name, *Real-Time Parser*.

Our parser uses a technique known as an Augmented Transition Network (ATN) [Woods]. It is a top-down, goal-directed, or expectation-driven approach.[*] Thus the parser answers specific questions such as, "Does this signal match my definition for Step-Response?", rather than answering the general question, "What shape is this signal?" A goal-directed approach works as long as one knows from experience what patterns to expect. In monitoring real-time data, this is generally a valid assumption.

---

[*]There are also bottom-up or data-driven approaches. They identify each element in the input with one of the primitives in the grammar, and then proceed to group the primitives according to the grammar. Although bottom-up approaches can be more efficient, they require some impractical assumptions (the set of primitives must be mutually exclusive and exhaustive). For this reason, we have not pursued bottom-up approaches.

We have found it advantageous to parameterize shape definitions as much as possible. The parameters reflect for example how much noise to expect and how large the component shapes may be. Parameterization allows a certain amount of flexibility, since the same shape can be applied in two different situations if those situations differ only in superficial ways such as units along the y-axis or degree of noise.

The input to the parser includes:

1. A description of the signal to be parsed.
2. The name of a shape definition.
3. A list of values to use for the parameters of the shape.

The output from the parser is a symbolic description of the signal known as a *parse tree*. The parser produces these descriptions in real time as more data becomes available.

ARCHITECTURE

Among the wide variety of alternatives for implementing the basic technique, we have chosen a parallel, distributed approach. Parallelism allow s for greater efficiency, and distribution allows for greater flexibility. The basic architecture divides the information flow diagram into three layers of parallel processes: (1) data access, (2) segmentation, and (3) grouping. **Figure 3** illustrates this architecture for the case of a *Step-Response* parser, The parser creates the specifics of this layout at run time:

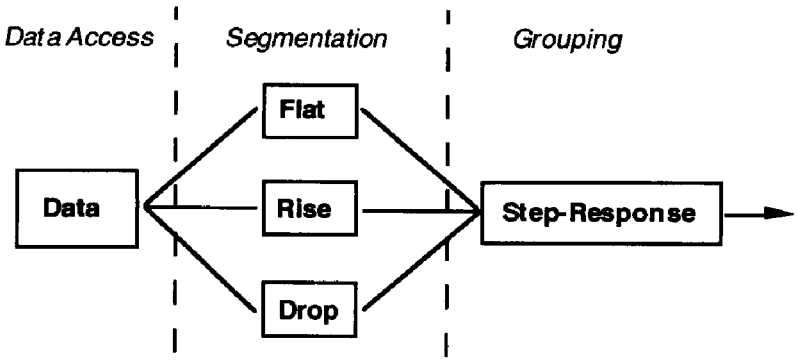

Figure 3. The Real-Time Parser architecture consists of three layers: data access, segmentation, and grouping. Each node is a parallel process that handles some data and passes the results along to the right. The parser creates the specifics of this layout at run time by consulting the definition of a **Step-Response**.

- A top-level function is called to request *Step-Response* information from the parser.
- When creating the *Step-Response process*, the parser consults the user's definition of Step-Response to determine that it requires the services of three

segmentation processes: *Rise*, *Drop*, and *Flat*. The parser establishes these processes in turn.

- Finally, the data access process is established, at which point it starts feeding data into the parser in real time.

Data flows from one layer to the next using data streams. A data stream is an abstract channel of communication. In different implementations, streams can be connections between threads of a process, between processes of a machine, and between machines. Our prototype implements data streams as connections between threads of a process.

For the data access layer, **Figure 4** shows some sample data relevant to a *Step-Response* parser. It comes from telemetry data describing the pitch of an underwater vehicle. Parsing can be used to measure the size of each overshoot. One could use this information to monitor for overshoots that are unacceptably large, indicating the feedback control loop is losing stability or that an unusually severe perturbation has occurred.

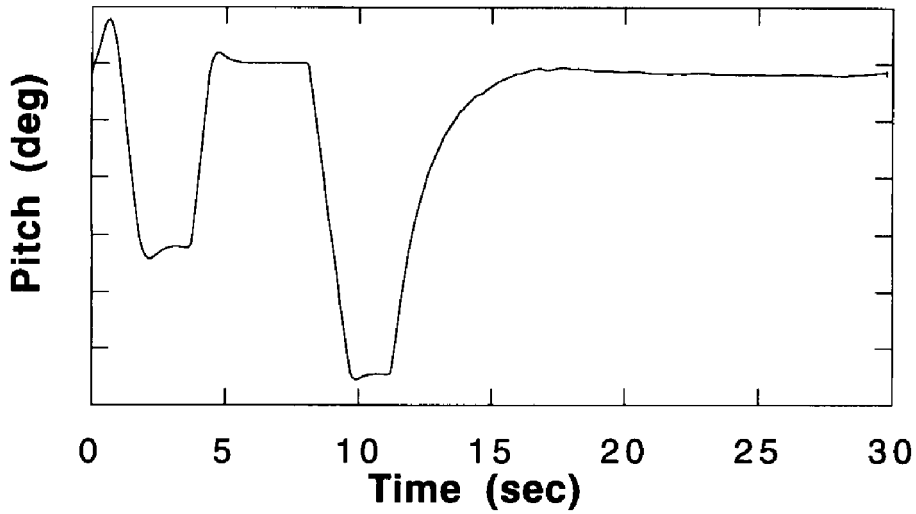

Figure 4. The figure shows some sample data relevant to a *Step-Response* parser, representing a series of step-responses in the pitch of an underwater vehicle. Overshoots can be seen at times 2.1, 4. 7 , and 10.0.

In the middle layer, each segmentation process works independently of the others to find which time intervals of the signal match its particular criteria and which intervals do not. Conceptually, the output of a primitive is a "1" for each data sample that matches and a "0" for each data sample that does not. We have found it more convenient, however, to describe a whole interval of the signal at once and send it as a composite message to the next layer. The primary advantage of composite messages is a considerable degree of data compression.

Figure 5 shows the output of the three segmentation primitives associated with the example data. Note in the figure that more than one primitive will occasionally match the data at the same time. For example, the signal appears to simultaneously match *Flat* and *Rise* at 3.0 seconds. The *Flat* monitor may or may not match a gently rising slope depending on the choice of the user-specified parameters that define flatness (*Flats* need only be approximately flat). In general, more than one of the segmentation algorithms may declare a match at a particular time. It is up to the higher-level *Step-Response* process to choose one, based on what it is expecting to come next in the pattern.

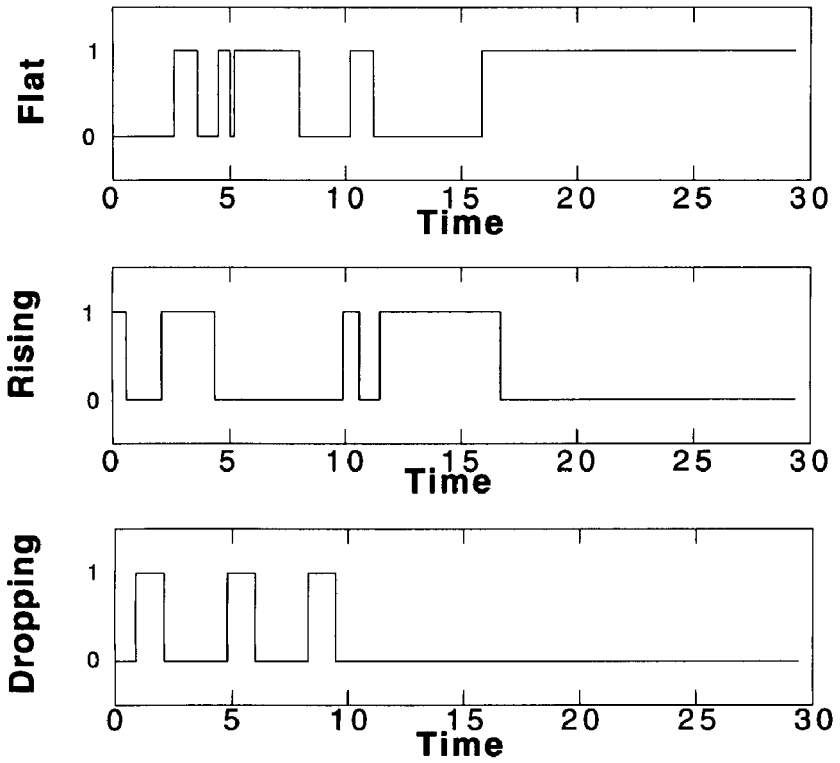

Figure 5. The three segmentation processes for *Step-Response* produce these three signals when fed the data of **Figure 4** as input. The output is "1" when the data matches and is "0" otherwise.

For the third layer, there is a single node for grouping primitives into *Step-Response*s. This node takes its input from each of the primitives in the preceding layer. It runs an ATN to group the primitives into the required pattern. The output of the third layer feeds a data stream directly to the client who has requested *Step-Response* information. Each object produced by this data stream is similar to that produced for segmentation primitives (interval descriptions), but a parse tree is also incorporated.

Table 1 shows the results of running the *Step-Response* monitor for the data in **Figure 4**. The first column is the size of the overshoot as a percentage of the size of the step. The second column is the time that the overshoot peaked. The third column is the time at which the parser has sufficient information to confirm that it has detected an

overshoot and report on its magnitude. The first and last *Step-Responses* have no overshoot.

Overshoot (%)	Time of Occurrence (sec)	Time of Report (sec)
0.0	-	-
4.6	2.1	3.6
5.8	4.7	6.4
2.5	10.0	11.2
0.0	-	-

Table 1. The parse tree results of the *Step-Response* parser can be used to extract the size of each overshoot. Each row shows the size of an overshoot, the time of its occurrence, and the time that the parser reports the information.

It is important to understand that some delay between the time of occurrence and the time of report is unavoidable. The delay occurs because the parser cannot report the information until the *Step-Response* reaches steady -state (*Flat*). This is appropriate because the height of an overshoot is a function of the

steady-state value of the signal. Of course the time required to process the data contributes to the delay, but in this case its contribution is small.

Two important real-time issues that were addressed in the detailed design and implementation were:

- What happens when a task cannot keep up with the data rate? If too much data is being received, the receiving process may choose not to buffer it all, effectively desampling the input data. This should lead to a graceful degradation in parser behavior rather than a sudden collapse.

- How is response time controlled when the duration of a pattern is long? We have implemented a mechanism for the parser to send intermediate status reports (part of a shape) at regular intervals, in addition to final results (the whole shape). Thus it may be possible to "see what's coming" even before the entire shape has been observed. This mechanism has a predictive flavor that we have not yet fully explored.

OBSERVATIONS

The basic technique of Syntactic Pattern Recognition has been in use for several years in the FAES project [Delatizky et al]. FAES combines the results of pattern recognition with traditional signal processing and an expert system to perform a data analysis task currently done by people. Using FAES, we have developed a knowledge

base that performs shutdown analysis of the ADCAP MK48 Torpedo.
Proof-of-concept work showed that FAES correctly analyzed 48 out of 49 test cases. We attribute a great deal of its success to the presence of pattern recognition, which significantly reduced the complexity of the rules needed for the expert system layer. The reason for this reduction is the ability of the expert to write rules using intuitively familiar language, such as "the start of the first flat" or "the duration of the decline". FAES users can factor out of their rules the difficult problem of identifying shapes. FAES is in its final stages of development prior to deployment into the Navy fleet.

The Navy has developed its own FAES knowledge base for *proofing* the ADCAP MK48 Torpedo. Proofing is a process of comparing weapon performance to the weapon specification. FAES has become an integral part of the proofing process.

Our success with FAES in a post-processing environment led us to implement the techniques described in this paper to achieve real-time processing. We wrote it in Common Lisp to take advantage of that language's superior prototyping flexibility, with a view to later implementation in some other object-oriented language, such as C++. The current prototype assigns each node in the parser architecture to its own thread within a multithreaded process. The multithreaded process runs on a Sun Sparcstation 2. The prototype simulates access to telemetry hardware by feeding data into the parser at a controlled rate.

The speed of a parser is difficult to quantify in a single number, because it depends on the sampling density of the data, on the available computing resources, and on the complexity of the pattern being recognized. As a lower bound, however, the unoptimized prototype can handle a real time signal having a sampling rate of over 4000 samples per second. Analysis shows the primary bottleneck is the speed of the segmentation primitives. The software was written for maximum prototyping flexibility at the expense of speed. We believe that by reversing our priorities, a new implementation could improve performance by a factor of 10. Of course a processor more powerful than a Sparc 2, or multiple parallel processors, would also improve performance,

SUMMARY

The Real-Time Parser makes it is possible to automate a variety of applications that either currently employ human analysts or that cannot be performed in real time. Additionally, the Real-Time Parser will be faster than human analysts. Our unoptimized prototype can handle a data stream having a sampling rate of over 4000 samples per second. One would expect a much greater level of performance by using multiple processors and by optimizing for speed rather than for prototyping flexibility.

Our syntactic pattern recognition algorithm has the benefit that its results are intuitive and easy to understand in comparison with traditional discriminant approaches. Where applicable, we believe that it offers the best trade-off between analysis power and ease of use for ordinary end-users. For signals that lack time-varying structure, however, discriminant methods may be the only viable approach.

Although numerous pattern recognition techniques have existed for some time, there is a clear absence of success stories bringing syntactic methods to the marketplace as a generic tool for real-time processing of time-series data. This technology has demonstrated its value in Navy post processing applications. The next step is to turn the prototype into an optimized, robust system to enable deployment of real-time applications.

ACKNOWLEDGEMENTS

The other two members of the FAES team, Karl Haberl and Steve Jeffreys, made many contributions to the ideas behind the Real-Time Parser. Thanks are also due to Ken Anderson for consulting on the design and implementation of this signal parser.

REFERENCES

Delatizky, J., J, Morrill, T. J. Lynch III, and K. Haberl, "Expert Analysis of Telemetry Data," *Proc. International Telemetering Conference,* Vol XXVII, 1991.

Duda, R., and P. Hart, *Pattern Classification and Scene Analysis*, Wiley, New York, 1973,

Fu, K. S., *Syntactic Pattern Recognition and Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

Hinkley, D. V., "Inference About the Change-Point from Cumulative Sum-Tests," *Biometrika*, Vol, 58, 1971, pp. 509-523.

Jeffreys, S., "Uniform Access to Signal Data in a Distributed Heterogeneous Computing Environment," *Proc. International Telemetering Conference,* Vol XXVIII, 1992.

Woods, W. A., "Transition Network Grammars for Natural Language Analysis," *Comm. ofACM*, Vol. 13, 1971, pp. 591-601.