Multiple Trellises and the Intelligent Cardiovascular Monitor

Michael Factor, David H. Gelernter, and Dean F. Sittig

YALEU/DCS/TR-847
February 1991

Multiple Trellises and the Intelligent Cardiovascular Monitor

Michael Factor [1], David H. Gelernter [1], and Dean F. Sittig [2]

Departments of Computer Science [1] and Anesthesiology [2]
Yale University, New Haven, CT

Abstract

A *stand-alone, intelligent medical monitor* is complex; it must continuously process multiple analog waveforms, recognize artifacts, extract pertinent parameters, recognize a patient's overall clinical state, analyze the problem and formulate a response. This paper presents a framework, *multiple trellises*, for building such a monitor. Multiple-trellises is a uniform hierarchical framework for heterogeneous program modules with widely varying run-time requirements that is efficient, predictable and usable. Our prototype intelligent cardiovascular monitor (ICM) consists of two trellises. The lower trellis contains code to process three different analog signals: the blood pressure from a non-invasive monitor and an arterial catheter, and the electrocardiographic waveform. The upper trellis contains processes to help detect evolving hemodynamic trends, identify abnormalities, and present a succinct summary to the clinician. Artifact detection processes are present in both trellises. Our long range goal is a complete, stand-alone intelligent medical monitor, this prototype shows the feasibility of constructing such a system. We conclude that the process trellis as extended by multiple trellises is a demonstrably useful software architecture for building these monitors.

Key words: real-time monitors, software architecture, artifact detection, signal processing

Please address all correspondence to:

Dean F. Sittig, Ph.D.
Department of Anesthesiology
Yale University School of Medicine
333 Cedar Street
New Haven, CT 06510

1.0 Introduction

A *stand-alone, intelligent medical monitor* is complex; it must continuously process multiple analog waveforms, recognize artifacts, extract pertinent parameters, recognize a patient's overall clinical state, analyze the problem and formulate a response. Not only do these tasks differ qualitatively, but they also have widely varying run-time requirements. For example, code to extract features from a sampled waveform must be very fast, executing for each sampled point (i.e., hundreds of times per second). By contrast, code to determine trends based upon the extracted features need execute much less frequently, perhaps once a second, but may require much longer to execute.

This paper presents a framework, *multiple trellises*, for building stand-alone monitors. Multiple trellises is a uniform framework for heterogeneous program modules with widely varying run-time requirements. In addition, a multiple-trellises-based monitor is easily extended to include new algorithms, (e.g., for feature extraction and artifact detection) taken directly from the literature.

Multiple trellises extends the *process trellis* software architecture for *heuristic, real-time monitors* [1,2]. The process trellis is a simple and uniform hierarchical framework for combining heterogeneous *processes* (i.e., modules) into a continuously executing, parallel program with real-time constraints. It is useful for building modular and flexible, understandable, predictable, and efficient programs. However, it is not particularly well-suited toward programs in which one module must run a hundred or more times more frequently than another. Multiple trellises provides the means of handling modules with widely varying run-time requirements.

The Intelligent Cardiovascular Monitor (ICM) is a trellis-based, prototype hemodynamic monitor [3]. The ICM's main goal is to present to the clinician a high-level analysis of the patient's condition (e.g., the patient is slightly hypovolemic) based upon low-level physiologic signals (e.g., blood pressure, heart rate, etc.). In its initial design, the ICM's inputs were pre-processed

physiologic signals (e.g., the heart rate is 78 beats per minute.). This paper describes an extended multiple-trellises-based design and an initial implementation of a stand-alone version of the ICM which receives sampled waveform data as input. Our new prototype mimics all capabilities seen in today's commercially available monitors, from processing digital waveforms to alarm generation, while still performing the higher-level "intelligent" tasks of the original version of the ICM. In addition, we have developed several processes using the multiple trellises framework which can recognize different monitoring artifacts.

This paper begins by motivating the need for a framework to build a stand-alone medical monitor consisting of modules with widely differing run-time requirements; we also discuss related research. After presenting the detailed goals of this research, we review those aspects of the process trellis architecture that are significant to multiple trellises. Using this review as a basis, we define the multiple trellis structure. We describe the benefits of this structure in the context of the ICM, showing how it has enabled us to design a stand-alone intelligent medical monitor. In particular, we describe some of the ICM's signal processing and artifact detection modules.

2.0 Motivation

There are two classes of medical monitors. *Low-level monitors*, made by commercial monitoring companies (e.g., Hewlett-Packard, Spacelabs, Siemens, Marquette, etc.) are common in today's intensive care units (ICUs) and operating rooms (ORs). They analyze a physiologic signal obtained from a transducer and present the clinician with the waveform and a simple summary of the signal's pertinent parameters. For example, a low-level monitor might receive as input the arterial blood pressure waveform sampled 200 times a second. The monitor will detect clinically important features of the waveform (i.e., the systolic, diastolic, and mean blood pressures from each heart beat) and present the clinician with the average (over several heart beats) numeric blood pressure values. Although some low-level monitors might present the data from

more than one signal on the same display, low-level monitors are not integrated in the truest sense since they do not contain logic that depends upon more than one physiologic signal. In addition, low-level monitors tend not to distinguish between data that accurately reflects the patient's condition and data that is artifactual [4].

Currently, in any ICU or OR, patients are surrounded by at least half-a-dozen low-level monitors, each of which contains special-purpose hardware for monitoring some signal. The American Society of Anesthesiologists, for example, encourages the monitoring of seven different signals [5] and it is not too difficult to imagine (or find information about in the literature) many more signals that may be important to monitor (e.g., multi-lead electroencephalogram (EEG), expired carbon dioxide fraction, muscle twitch response, etc.). Since these monitors are not integrated and only present simple summaries, they cannot draw a user's attention to more complicated and critical problems involving information from multiple data sources. This can easily lead to a situation of information overload in which a clinician has more information available than s/he can reasonably digest. Beyond this inability to present a high-level characterization of the patient's status, alarms to warn of untoward conditions that are built on top of low-level medical monitors generate many false alarms. Kestin et al. observed that the current generation of medical monitors experience false alarm rates of up to 75% [6]; O'Carrol observed similar high numbers of false alarms [7].

High-level monitors, the other class of medical monitors, are still, in general, in the experimental stage. A high-level, or intelligent, monitor integrates data from multiple sources and presents a high-level characterization of a patient's condition rather than just a simple summary of the data. Thus, instead of reporting separately that the patient's blood pressure and pulmonary artery diastolic pressure (PAD) are low, a high-level monitor could report (if conditions warranted) that the patient was hypovolemic. As others have remarked, a high-level characterization could greatly reduce the number of false alarms, as well as minimize information overload [6,8]. There are several research projects that have built prototype high-level monitors. For instance, VM [9],

COMPAS [10], SIMON [11], and VentPlan [12] all were designed to monitor and assist in the management of ventilator-dependent patients. In addition, Berger et al. [13] developed CADMO to assist healthcare professionals in the management of patients with insulin-dependent diabetes.

Like low-level monitors, there are several problems with state-of-the-art high-level monitors. These problems are at least part of the reason that most of these monitors have failed to advance beyond the research stage and into routine use in the OR or ICU (COMPAS is a notable exception [14,15]). The most significant problem is that these monitors assume their inputs are the (already-derived) features of the physiologic signals (e.g., the respiration rate) and not the signal itself (e.g., not the analog flow waveform collected from the pneumotach at the patient's mouth). Further, these monitors almost universally assume that the data streams are error-free.

Because they assume their inputs are the already-derived numeric values of the physiologic signals, the run-time requirements of a high-level monitor are much weaker than those of a low-level monitor. To derive the features of a signal from an analog waveform the signal must be sampled quite rapidly; for instance, in the typical low-level monitoring task of deriving a numeric heart rate from a lead of an electrocardiograph (ECG), the analog waveform should be sampled at least one hundred times a second to insure that all of the physiologically meaningful information is retained [16]. On the other hand, the high-level monitor's task of calculating the systemic vascular resistance (SVR) based upon the numeric blood pressure, the numeric central venus pressure (CVP), and the numeric cardiac output need be done much less frequently[1].

One might assume that a fully integrated, stand-alone monitor could be constructed by entering the output of commercially available low-level monitors into a high-level monitor. This approach, however, has problems. First, though it would be possible to package all of the hardware in a single "box" and present the results to the clinician using a single integrated display, the monitor's software would not be integrated. Each low-level monitor would be implemented

---

1 Assuming a maximum heart rate of 240 beats per minute this results in the program running 4 times a second.

using whatever technique its vendor chose while the high-level monitor's software would be implemented using still a different method. This *ad hoc*, non-integrated approach to software greatly decreases the ease of maintaining and extending the overall monitor's logic.

The second problem with this approach is that it does not address the (false) assumption made by most current high-level monitors that data streams are error-free. However, many lower-level artifacts might be detected within a truly integrated monitor by comparing independent measurements of the same physiological signals (e.g., the heart rate from the ECG and from an arterial catheter). This requires additional code which is not found today in either low-level or high-level monitors [17].

3.0 Goals

We desire a framework for combining the signal processing code that appears in today's commercially available low-level monitors, the diagnostic and prescriptive codes that appear in the experimental high-level monitors and the artifact detection and error correction codes to bridge between the noisy data produced by low-level monitors and the error-free inputs assumed by high-level monitors. To improve maintainability and to provide flexibility for extensions, as new medical algorithms are developed, we wish a uniform framework for the entire monitor that permits the ready incorporation of new algorithms. An application built with this framework must run predictably in real-time guaranteeing that the widely varying timing constraints of the low-level code and the high-level code are met. In addition, since such a monitor's computational requirements are high --- as mentioned above at least half-a-dozen signals will be sampled, analyzed and integrated --- the monitor's structure must be able to take advantage of parallel hardware, a proven way of meeting substantial computational requirements [18]. While this framework need not be limited to building medical monitors, it must be demonstrably useful in a medical setting [19].

The ICM, which is built on top of the process trellis, is a high-level monitor; it integrates data from numerous physiological signals, presenting a clinician with an analysis rather than a simple summary. The ICM's initial design assumed that its inputs were error-free numeric summaries of the various physiologic signals --- the blood pressure (systolic, diastolic and mean), the heart rate, the pulmonary artery diastolic blood pressure (PAD) and the central venous pressure (CVP) among others --- once a second. As described above this assumption is problematic. Our goal was to extend the ICM to be a *fully integrated, stand-alone monitor*, such that it could accept (potentially) noisy data sampled from analog waveforms at least one hundred times a second, while still performing all of the tasks performed by the pre-existing, high-level ICM. This extension provides an existence proof of the usefulness of the multiple trellises framework.

4.0 The Process Trellis

We have previously described the basic process trellis architecture in detail [see 2,20,21,22]. Here we present a brief and simplified description of those features of the process trellis architecture which are relevant to multiple trellises. We illustrate with examples drawn from the original high-level monitor version of the ICM.

The process trellis is an acyclic hierarchical network of heterogeneous decision processes whose structure mirrors the information-flow structure of the domain. The trellis's information-flow hierarchy bears a strong resemblance to a blackboard architecture's information-flow hierarchy [23]. One process dominates another in the hierarchy if the lower-level process calculates information that the higher-level process requires. Figure 1 is a generic trellis hierarchy. At the bottom, *raw-data* processes receive inputs from the external world; *data-filter* processes perform tasks such as validating data and converting into symbolic representations. Still higher, *correlation* processes look for simple correlations in the results of the data-filter level. Their outputs are used by *diagnosis* processes in characterizing the domain as a whole. Finally,

recommendation processes consider the results of the lower levels in recommending actions. Though not all trellis programs will have all of these levels, and additional layers are possible, this sketch captures the essence of a trellis hierarchy.

Figure 1 about here.

Every process has a set of *inferiors*, a set of *superiors* and a *state*. The value of a process's state, which can be a complex object, depends upon the states of the process's inferiors and upon the state's own prior value, making processes history sensitive. When a process executes, it generates a new state if it has sufficient new information; each process defines for itself in an arbitrary, domain-dependent manner what "sufficient new information" means. Thus, processes are heterogeneous. Whenever *any* non-empty subset of its inferiors have new states, a process is *enabled* and can execute. Unlike Petri-net or conventional data-flow models, processes do not require inputs from all of their inferiors to be enabled; thus, processes execute with partial information. When a process generates a new state, its superiors are enabled and attempt to generate new states in turn. To summarize, states flow up the trellis from inferiors to superiors.

Besides passively waiting for a new state from an inferior, a process can actively query any inferior. A query contains no data; it simply causes the lower level process to attempt to generate a new state. In fact, a process may not even be able to determine that it has been queried. A queried process executes and attempts to generate a new state. If the queried process has insufficient information to generate a new state it can in turn query its inferiors. A bottom-level process can, when queried, request inputs from a user. To summarize, queries flow down the trellis from superiors to inferiors.

To make this concrete, figure 2 shows a small and simplified portion of the high-level monitor version of the ICM. At the bottom are raw-data processes, such as *Raw BP* (blood pressure), which receives the numeric blood pressure from the external world. As mentioned,

these processes assume an error-free numeric data stream sampled once a second; we describe below how the data enters from the external world. In the data-filter level are processes that convert numeric values into qualitative values. For instance, *BP* might determine that the blood pressure is high with a slight downward trend. The third level of processes perform correlations and look for simple patterns such as *Hypotension*, which depends upon the qualitative blood pressure (*BP*) and heart rate (*HR*). At the top of the ICM are diagnostic processes which look for time ordered patterns in the states of lower-level processes; [23] describes the behavior of these processes in detail. The ICM does not currently have any recommendation-level processes, although earlier versions did.

As noted, when a process generates a new state, all of its superiors attempt to generate new states. Thus, when *HR* generates a new state, this state flows up the trellis to all of its superiors; in particular, *Vasoconstriction* executes. Note, that *Vasoconstriction* executes even if its other inferiors, *SVR* and *Cardiac Output*, have not generated new states. *Vasoconstriction* might, however, decide that knowing the patient's cardiac output would improve its own diagnostic accuracy; if so it can query its inferior *Cardiac Output*, which might then request, but not demand, a cardiac output measurement from a clinician.

Besides providing for uniform inter-process communication, the trellis provides a uniform mechanism for external-world interactions. *Probes* are an application-independent mechanism for dynamically interacting with *any* process in a running trellis program. Significantly, the user, *not* the program, controls when this interaction occurs and with which part of the program it occurs.

There are two symmetric probes. *Write probes* set the value of any portion of any process's state. *Read probes* read any process's state. Probes are either active or passive. An *active* probe can cause the process to execute; a *passive* probe cannot.

When a process gets prodded by an active write probe, it executes as if it had received a new state from an inferior. It can fully integrate the new data into its state and may generate a new state which its neighbors will see. When a process receives a passive write probe, it does not

execute. In either case, the next time this process executes, for whatever reason, its state reflects this write probe. While not required, base-level processes, i.e., processes with no inferiors, will generally have permanently attached active write probes that periodically funnel data from some transducer into the program. For example, in the high-level monitor version of the ICM, a permanently attached active write probe enters the numeric PAD into the process *Raw PAD* once a second [2].

When a process receives an active read probe, if it does not have a currently defined state, it executes as if it were queried. When a process receives a passive read probe, whether or not it has a currently defined state, it does not execute. In either case, when the process next has a state, the read probe will return this state.

4.1 Trellis Properties

The obvious way to implement a trellis program is to have a separate heavyweight process for each trellis "process" with the processes communicating in a message passing fashion. However, there are several reasons not to do this, relating to predictability and efficiency. Instead, a small number of active *worker* processes repeatedly sweep over the trellis graph, looking at all trellis processes and executing those that are enabled. A process is *enabled* if it has received new data, either as a new state from an inferior or from an active write probe, or if it has received a query, from either a superior or an active read probe. This *iterative execution scheme*, which is readily parallelized, guarantees that no process executes twice while another process which has been enabled equally as long does not execute even once.

When run using the iterative execution scheme, the process trellis has several nice properties. Here, we state the properties; [2,20, & 22] contain detailed discussions.

---

[2] Since the ICM is still a prototype, these values come from a patient simulator which uses data recorded during surgery and stored on disk.

4.1.1 Stability

Given three reasonable assumptions:

1) each invocation of an enabled process terminates,

2) no probes are executed during the period under consideration, and

3) processes do not generate new states without receiving new information in the form of a write probe or an inferior's state,

then in a fixed number of sweeps of the iterative execution scheme after receiving new write probes, any process trellis program will *stabilize*, i.e., there will be no enabled or executing trellis processes. Intuitively, the trellis processes initially enabled by probes will execute. Some of these will generate new states and/or query their inferiors, thereby enabling other processes. These process will then execute, potentially enabling a different subset of trellis processes; this will continue for a number of sweeps. The stability property states that eventually, there will be a sweep in which no trellis processes are enabled. Once stable, a trellis program remains stable until acted upon by an external agent executing a probe.

Stability leads to a natural definition of a trellis application meeting its real-time constraint: if the rate at which data enters a trellis program from the outside world, *tau*, is such that the trellis application stabilizes between receiving one set of inputs (via write probes) and the next, the trellis program meets its real-time constraint [3].

4.1.2 Efficiency

The trellis's processes can be efficiently and mechanically partitioned to a set of worker processes where the workers correspond to the processors of a parallel computer. For example, the ICM is over 92% efficient when run on ten processors [1].

---

[3] This is not the only definition of meeting a real-time constraint possible [20].

4.1.3 Predictability

An analytic model can, given a partitioning of processes to processors, predict how long, in the worst-case, a process trellis program requires to stabilize; this is a trellis program's *period.* Using this prediction, one can determine if a trellis application is guaranteed to meet its real-time constraint; a trellis program will always meet its real-time constraint if its period is less than *tau*, the time between external inputs.

Between each set of external inputs, the worker processes potentially execute each of the trellis processes. Each trellis process takes a certain amount of time to execute; this time can vary widely between the processes. The period of a trellis program is clearly limited by the time to execute the slowest process in the program; independent of the partitioning of trellis processes to worker processes, a trellis program cannot be guaranteed to stabilize in less time than it takes to execute this slowest process. Thus, if a trellis program contains a process that takes half a second to execute but is intended to receive new inputs from the external world four times a second, it is impossible to guarantee that the trellis always runs in real-time. This difficulty occurs since our definition of running a trellis program in real-time, the entire trellis stabilizes, is monolithic [4]; it does not provide for processes which may take longer to execute but do not need to execute as frequently. Multiple trellises, which we present in the next section, address this weakness.

4.1.4 Usability

We can construct high-level tools to aid in building a process trellis application. A process trellis shell implements all of the parallelism and inter-process communication along with numerous tools (including a scheduler that embodies the analytic model) to aid in guaranteeing a program meets its real-time constraint [1,3]. The shell has been used by individuals with no prior parallel programming experience to implement the ICM.

---

[4] The same difficulty occurs using alternate definitions of running in real time.

5.0 Multiple Trellises

Beyond providing a powerful, flexible mechanism for the external world to interact with a running trellis program, probes have an additional benefit; they can aggregate independent, logically disjoint, and potentially physically dispersed trellis programs into a single larger program, i.e., *multiple trellises*. Processes in the different trellises are connected via probes; each trellis treats the others as the external world. We can use probes to connect processes in different trellises since a process does not care where its inputs come from: transducers attached to the patient, low-level monitors or another process trellis. All of the properties discussed in the prior section hold for each of the individual trellises in a multiple trellis program.

Multiple trellises serve two main purposes. First, they provide another level of modularity. In the process trellis, each process solves a conceptually meaningful part of the monitoring problem; multiple trellises enable us to group together, into a single trellis, processes that serve a greater common end. For instance, a trellis application to monitor all of the patients in an ICU might consist of a separate trellis (such as the ICM) for each patient along with one or more trellises that performed more general tasks, such as monitoring patient care for quality control management, identifying potential sources of infections common to multiple patients, or even suggesting which patients would be the best candidates for transfer to another unit in the event of a staff or bed shortage.

The second reason to use multiple trellises was motivated by the need to include low-level, higher-frequency signal processing algorithms as well as computationally expensive but less frequent, higher-level logic in a single program. Multiple trellises allow programs to contain processes that execute at widely varying frequencies. For example, if some process, *the fast process*, receives an active write probe once every millisecond and requires only a half of a millisecond to execute while another process, *the slow process*, requires a tenth of a second per invocation but only needs to run once a second, there is no way the trellis could be scheduled such

that it was guaranteed to meet its real-time constraint. The trellis's period is at least 0.1 sec, but the time available to analyze each set of inputs from the external world, *tau*, is no more than 0.001 sec. However, by placing these two processes in independent trellises, the fast and slow process *can* be part of the same program. Each of the trellises runs on its own processors, which are independent (except for probes) of the other trellis's processors, and thus, the slow process does not interfere with the fast process.

5.1 The Extended Multiple Trellis ICM

Our initial version of the ICM is a high-level monitor [3]; it was designed to receive write probes containing error-free, numeric data from an external source once a second. This program contains 104 processes which perform a wide range of tasks including diagnosis [24] and trend detection [25]. While further work is necessary before planned clinical tests, the ICM does run and has received a small amount of off-line testing using data recorded during clinical cases [26].

We wished to use the multiple trellis structure to extend the high-level monitor version of the ICM to include the code normally found in low-level monitors, along with additional code to bridge between the noisy data produced by low-level monitors and the error-free data desired by high-level monitors. While our long range goal is a complete, stand-alone monitor, we wish with our current work to show the feasibility of constructing such a system.

The extended ICM consists of two trellises: a lower trellis and an upper trellis. The lower trellis contains processes that detect features in analog signals that are sampled one hundred times a second. The upper trellis contains all of the processes in the original high-level monitor version of the ICM. Artifact detection processes are present in both trellises. Figure 3 is a rough depiction of the ICM's multiple trellis structure.

Figure 3 about here: The Multiple Trellis Structure of the ICM

Information can be passed from the lower to the upper trellis either in an event-driven or time-driven mode; both modes are useful in a stand-alone intelligent medical monitor. In event-driven mode, a process in the lower trellis executes a write probe for a process in the upper trellis whenever an event occurs. For instance, a lower trellis process that measures the beat-to-beat heart rate might execute a write probe for the upper trellis process that performs an analysis of the numeric heart rate each time a beat is detected. However, in event-driven mode, no information is sent when an event does not occur; thus, the upper trellis processes are not notified when the lower trellis process does not detect a heart beat.

In time-driven mode, the lower level trellis process executes a write probe for a process in the upper trellis at fixed time intervals. In the heart rate example, the lower trellis process might execute a write probe for the upper trellis process, containing all completed beat information including (if necessary) information that no beat was detected, once a second. In this mode, the upper level trellis process executes once a second even if an event, the detection of a heart beat, does not occur. This is necessary since the absence of a detected heart beat, while possibly signifying a slow heart rate or a disconnected lead could also result from a heart which has stopped beating. Therefore, one must always make sure that the higher-level alarm generating code executes after some set time period and not solely as a result of events detected by the lower level trellis.

5.1.1 The Lower Trellis

The lower trellis in the stand-alone version of the ICM currently contains code to process three different signals: the blood pressure from a finapress [5], the blood pressure from an arterial

---

[5] A non-invasive blood pressure monitor that fits on the finger. It continuously displays the arterial pressure waveform, the systolic, diastolic and mean arterial blood pressures, and the heart rate. (2300 Finapress blood pressure monitor, Ohmeda, Englewood, CA) [27].

catheter, and the electrocardiograph waveform. While a complete stand-alone monitor would contain many more low-level signal processing modules, this initial prototype enables us to show the validity of multiple trellises as a structuring technique for stand-alone monitors. All three signals are sampled one hundred times a second and the sampled values are entered into the lower trellis by active write probes. In principle, there is no reason we could not sample these signals more often; the test data that we had available, however, was sampled at these rates. The lower trellis also contains an artifact detection process that performs a Fast Fourier transform (FFT) on the waveform from the arterial catheter; this process detects the monitoring artifact associated with inflation of the automatic blood pressure cuff proximal to the ABP (arterial blood pressure) recording site [6]. We briefly describe each of these processes.

5.1.1.1 ECG Waveform Analysis

The ICM determines a patient's heart rate by identifying the QRS complexes [7] from the ECG waveform; the waveform data enters the trellis process, by means of an active write probe, 100 times a second. Detecting the QRS complex is difficult for several reasons including 1) inherent physiologic variability of the signal, 2) noise from muscle contractions, electrode movement, power-line interference (60 Hz), baseline wander, etc. and 3) T waves with high-frequency components that can mimic the QRS complex to name a few. The logic of this process is based upon Pan and Tompkins' algorithm which uses digital filters, nonlinear transformations, and decision rule algorithms in an attempt to reduce the amount and influence of artifacts while maintaining the "true" signal [28].

Figure 4 about here.

---

[6] A catheter inserted in the radial artery just above the hand on the inner side of the arm.

[7] Large spikes resulting from ventricular depolarization of the cardiac muscle.

Figure 4 shows an ECG waveform and several of the parameters relevant to identifying the actual QRS complexes. The *signal peak* is the moving average of the 8 most recent peak values of each QRS complex. Likewise, the *noise peak* is the moving average of the 8 most recent peaks that were determined not to be QRS complexes. In this example, in which there is virtually no extraneous noise, these noise peaks correspond to the average height of the last 8 P-waves. Finally, the *QRS detection threshold* is the noise peak plus twenty-five percent of the difference between the signal peak and the noise peak.

Whenever the ECG waveform goes above the QRS detection threshold, a QRS complex is detected. After two QRS complexes have been detected (i.e., a complete heart cycle), the trellis process calculates the R-R interval (in milliseconds), the beat-to-beat heart rate (the inverse of the R-R interval measured in beats per minute), and the moving average of the previous 8 beats. The process also calculates a separate moving average of the last 8 beats which fell within a predetermined time window. Once a second, i.e., after every hundred samples of the waveform, the process executes a write probe to the process *Raw HR* in the upper level trellis (see figure 3); the probe contains all of the calculated information for each beat completed during the past second.

Preliminary testing of the QRS detection process (containing the Pan and Tompkins algorithm) on data collected off-line during surgery cases showed excellent agreement with human experts. Pan and Tompkins performed a much more thorough evaluation of this algorithm on the MIT/BIH (Massachusetts Institute of Technology/Beth Israel Hospital) database. They found that the algorithm correctly identified 99.3 percent of the QRS complexes [28]. Perhaps more important from our standpoint was the fact that our implementation of their algorithm (coded in C) was able to run within the time constraints we set (0.01 seconds).

5.1.1.2 Arterial Catheter and Finapress Waveforms Analysis

As with the ECG, difficulties arise in analyzing the arterial blood pressure waveform due to noise in the input signal. We have implemented an algorithm based upon Kinias et al.'s approach

to identifying important fiducial points (i.e., the systolic and diastolic pressures) [29]. The same basic method is useful in analyzing the blood pressure waveform whether it comes from an arterial catheter or from the finapress. Thus, the lower trellis contains two processes that perform identical analyses of independent waveforms: the blood pressure from the arterial catheter and from the finapress.

Insert Figure 5 about here: Detecting Important Fiducial Points in the Blood Pressure Waveform

Figure 5 is a simplified representation of Kinias et al.'s technique. In this example, an initial chord is drawn spanning k (in this example $k = 8$) points. The slope of the chord connecting the current data point with one k points back in time is compared to that of the chord connecting the next data point to one k points back in time from it. When there is a change in slope between two successive chords, the length of the chord is divided in half (i.e., $k/2$). This technique repeats until the single point, either the systolic or diastolic pressure, is identified. As Kinias et al. mention in their article, the length of the initial chord, k, is important for the filtering characteristics of the method. We experimented with several different chord lengths. By varying the chord length one can manipulate the number of false positive (i.e., extra systoles and/or diastoles) and/or the number of false negatives (i.e., missed systoles and/or diastoles).

To test our implementation of Kinias et al.'s algorithm, we collected digital ECG and arterial blood pressure waveforms from 10 patients (approximately 100 cardiac cycles per patient; total 1046 cycles) routinely monitored during surgery using a PC-based multi-channel data acquisition system [30]. The arterial and ECG waveforms were displayed on a high resolution monitor to allow easy manual identification of the appropriate systolic and diastolic blood pressure values, and their time of occurrence, from each heart beat; these points became our "gold" standard. Internal parameters of the blood pressure detection algorithm were manipulated in an attempt to increase sensitivity of the algorithm without compromising specificity. Manually

identified points were compared to those identified by the computer. A receiver operating characteristic (ROC) curve was constructed for each of these trials (see Figure 6). Optimal algorithm settings were chosen to maximize both sensitivity (True Positive Rate (TPR)) and specificity (1 - False Positive Rate (FPR)). At optimal settings, the computer generated values were compared to the manually detected points by calculating the mean absolute deviation (MAD) of corresponding values.

Insert Figure 6 about here.

For the optimal values, TPR = 0.80 and FPR = 0.003. The MAD of the systolic and diastolic arterial blood pressure values was 2.0 and 2.5 mmHg respectively (well within the normal range of clinical uncertainty). Calculation of the heart rate from the time interval between systolic peaks resulted in a MAD of 9.5 msec or 0.8 beats/min at a HR of 72 beats/min.

The Kinias et al. algorithm proved to be a reliable means of detecting both systolic and diastolic blood pressure values as well as for identifying complete cardiac cycles. From this information, we can calculate the beat-to-beat heart rate from the ABP waveform completely independent of that obtained from the ECG measurements.

5.1.1.3 Artifact Detection

An astute clinician can easily detect and identify numerous artifacts from the arterial blood pressure tracing; however, difficulties arise when attempting to build an intelligent monitor capable of duplicating a clinician's performance. We have designed a lower-level process to identify the arterial blood pressure (ABP) monitoring artifact associated with inflation of the automatic blood pressure cuff proximal to the ABP recording site. This process implements a fast Fourier transform (FFT) which is a valuable aid in identifying artifacts [31].

To develop the discrimination criteria for the process, we calculated a Fourier spectrum of selected "normal" arterial blood pressure (ABP) waveforms from 5 subjects routinely monitored during surgery. We averaged these spectra to develop a "normal" spectral pattern. We repeated this process for 5 ABP waveforms containing the artifact. Figure 7 shows a portion of a representative "normal" arterial blood pressure waveform (top left), along with a portion containing the artifact (bottom left), as well as their respective fourier spectra.

Insert Figure 7 about here.

By comparing the averaged fourier spectra patterns obtained from the normal and artifact containing waveforms, we hypothesized that a discrimination criterium could be based on the sum of the amplitudes of the first two normalized harmonics [8]. We calculated the mean and standard deviation of the sum of the first two normalized harmonics for each of the 5 normal waveforms ($\mu = 7.3$, SD = 3.0) and from the 5 artifact containing waveforms ($\mu = 49.0$, SD = 7.5) [32]. By setting the discrimination criterium (26.5) to be approximately 6 standard deviations above the "normal" mean and 3 standard deviations below the artifactual mean, we were able to obtain good separation between the spectra calculated from the normal and artifact containing waveforms.

To test this process, we selected arterial blood pressure tracings (50 seconds each) from 5 additional patients, two of which showed the artifact. The artifact detection process calculated the FFT for each newly sampled point (along with the previous 255 points), comparing the sum of the first two normalized harmonics of the FFT spectrum against the discrimination criterium. For each sampled point (4055 in each file), we recorded the process's determination (i.e., normal or artifact containing waveform) which we compared visually to the input waveform. The process accurately classified each tracing.

---

[8] We normalized the first two harmonics by dividing their sum by the sum of the amplitudes of the first 20 harmonics and multiplied the result by 100.

5.1.2 The Upper Trellis

The upper trellis of the multiple trellis ICM is little different from the original ICM [3,20]. We have added a handful of processes to this trellis to begin experimenting with detecting artifacts through comparison of the redundant information from multiple independent measurements of the same physiologic variable. We do not claim that we have solved the artifact detection problem; we do, however, present a start. This start can easily be extended to take advantage of new artifact detection algorithms as they are developed. We have also changed some of the data sources of processes in the upper trellis from data files to processes in the lower trellis which execute write probes for the appropriate upper trellis processes. We describe only these changes; those interested in knowing more about the logic of the higher trellis should see the references.

5.1.2.1 Higher-level Artifact Detection

We are experimenting with different methods of combining information within the context of a high-level integrated monitor. This is enabling us to test our approach to multi-channel artifact detection. Our goal is to detect and identify various high-level monitoring artifacts in real-time by correlating information from multiple independent measurements of the same physiologic variable. For example, the beat-to-beat heart rate as measured from the electrocardiogram and the arterial blood pressure tracing. In the simplest case, one might look for differences in the two measurements concluding that something is wrong if the two measurements differ. Unfortunately, if both measurements are still physiologically valid (i.e., within normal ranges), it is impossible to determine from this information alone which is in error. However, by including trend information from the recent past one might be able to determine which measurement is in error.

Insert figure 8 about here.

Towards this end, we have implemented a process, *Hr_arbitrator* (see figure 8) which takes as inputs the beat-to-beat heart rate values from the ECG and the arterial blood pressure waveform

identification processes along with the outputs from processes which implement the multi-state Kalman filtering algorithm[9]. The *Hr_arbitrator* process then determines the "correct" heart rate. Built into this *Hr_arbitrator* process is logic to identify double counting of the ECG waveform[10]. Figure 9 describes the logic, in pseudocode, that could be used to identify this particular type of monitoring artifact. Briefly, the outputs from the multi-state Kalman filters are used to identify abrupt changes within a particular heart rate measurement, or a consistent difference between the two beat-to-beat heart rate measurements (e.g., **Kalman filter for ECG**). Once an abrupt change, or a consistent difference, is detected in the HR from the ECG, then consecutive beat-to-beat R-R interval determinations (**ECG_RR-current** and **ECG_RR-previous**, which are available from the ECG HR Determination process [see fig. 8]) are added together to create a new hypothesized heart rate (**Hypothesized_HR**). This hypothesized heart rate value is then compared to the heart rate determined from the arterial blood pressure tracing (**ABP_measured_HR**).

As with all of the other processes in the upper trellis, the *Hr_arbitrator* process runs every time new data is entered into it. In the vast majority of instances, none of the conditions for double counting will be met and the *Hr_arbitrator* will send up the HR derived from the ECG. Therefore the *Hr_arbitrator* process will be exiting routinely after executing only one if-condition. On the other hand, it is also possible that this first if-condition could be met when double counting is not present. For example, we are currently in the process of developing a process specifically to

---

9 The Kalman filtering algorithm is a general state space modelling technique which can be applied to a wide variety of system models. We chose five independent data models which represent linear approximations to all of the important changes expected in the physiologic signals [see figure 8 and ref. 25 for a more complete description]. The first of these models STABLE reflects the state of the patient the vast majority of the time. Each of the other 4 data models reflects some type of abrupt change in the data stream: STEP, SLOPE, STEP+SLOPE, and TRANSIENT.

10 Double counting occurs when the QRS detection threshold is set too low. This leads to T-waves being mistaken for QRS complexes resulting in an abnormally high beat-to-beat heart rate determination.

identify bigeminy[11], which could easily be mistaken for double counting. Although neither of these processes (e.g., *Hr_arbitrator* or *bigeminy*) is likely to be relevant at any specific instance in time, when they are relevant they may be extremely important and therefore, can not be ignored.

This is but one example of the type of processes which could be implemented to find higher level artifacts within the multi-trellis framework using the integrated artifact detection methodology.

if (**Kalman filter for ECG <> Kalman filter for ABP**) then

calculate **Hypothesized_HR**

if (Absolute value [**Hypothesized_HR - ABP_measured_HR**] < Epsilon) then

if (High_ratio > [**ECG_RR-current / ECG_RR-previous**] > Low_ratio) then

DOUBLE_COUNTING HAS OCCURRED!

set "correct" **HR = ABP_measured_HR**

Where: $\textbf{Hypothesized\_HR} = \frac{60}{\textbf{ECG\_RR-current} + \textbf{ECG\_RR-previous}}$

and **ECG_RR-current > ECG_RR-previous**

High and low_ratios are based on "normal" variations in T-R / R-T intervals.

Figure 9. Pseudo-code showing double counting logic.

11 Bigeminal ventricular rhythm occurs when premature beats alternate with normal beats [33]. There are several theories as to the physiological cause of this problem.

6.0 Conclusions

Multiple-trellises is a uniform framework for heterogeneous program modules with widely varying run-time requirements. Since the multiple trellises framework inherits all of the properties of the process trellis, it is efficient, predictable and usable.

Programs built using the multiple trellis framework are extensible. As we showed with the ICM, they can readily incorporate algorithms developed by others. Since building a stand-alone intelligent monitor is a massive task, it is important that the software architecture provide the support necessary to take advantage of other researchers discoveries. In addition, a multiple-trellis-based monitor can contain the entire range of logic and codes that must be included in a stand-alone medical monitor: low-level signal processing, artifact detection, high-level diagnostic reasoning, and therapeutic advice generation. Thus, we conclude that the process trellis as extended by multiple trellises is a demonstrably useful structure for building stand-alone intelligent medical monitors.

Acknowledgements

This work was supported in part by grants from the National Library of Medicine (T15-LM07056) and the Whitaker Foundation.

References

[1] Factor M, Gelernter DH, Kolb C, Miller PL, Sittig DF. Real-time performance, parallelism and program visualization in medical monitoring. Yale University Research Report YALEU/DCS/RR-808; June 1990.

[2] Factor M. The Process Trellis Software Architecture for Real-Time Monitors. in Proceedings Second ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming (PPoPP), Mar, 1990: 147--155 Seattle, WA. (also in SIGPLAN Notices, 25(3)).

[3] Factor M, Sittig DF, Cohn AI, Gelernter D, Miller PL, Rosenbaum SH: A parallel software architecture for building intelligent medical monitors. *International Journal of Clinical Monitoring & Computing* 7:117-128; 1990.

[4] Wessling KH, Smith NT. Availability of intraarterial pressure waveforms from catheter-manometer systems during surgery. *J Clin Monit* 1985: 1: 11-16.

[5] The American Society of Anesthesiologists Newsletter. 50:12; 1986.

[6] Kestin IG, Miller BR, Lockhart CH. Auditory alarms during anesthesia monitoring. *Anesthesiology* 69: 106-109; 1988.

[7] O'Carrol TM. Survey of alarms in and intensive therapy unit. *Anesthesia* 41: 742-744; 1986.

[8] Poler SM, Dunford-Shore B, Lappas DG. Architecture of an intelligent anesthesia monitoring system: Concepts for distributed real-time multiprocessing. in Proceedings of the 1988 Rochester FORTH Conference. pgs 110-112; 1988.

[9] Fagan LM. Representing time-dependent relations in a medical setting. Ph.D. Dissertation, Stanford University; 1980.

[10] Sittig DF, Pace NL, Gardner RM, Beck E, Morris AH. Implementation of a computerized patient advice system using the HELP clinical information system. *Computers and Biomedical Research* 22(5):474-487; 1989.

[11] Dawant B, Uckum S. A framework for intelligent multi-channel biological signal interpretation. Vanderbilt University Center for Intelligent Systems Technical Report #CIS-90-08; 1990.

[12] Rutledge G, Thomsen G, Beinlich I, Farr B, Sheiner L, Fagan LM. combining qualitative and quantitative computation in a ventilator therapy planner. in Proceedings of the 13th Annual Symposium on Computer Applications in Medical Care (SCAMC-89). (ed. LC Kingsland) pgs 315-319; 1989.

[13] Berger MP, Gelfand RA, Miller PL. Combining statistical, rule-based, and physiological model-based methods to assist in the management of diabetes mellitus. Computers and Biomedical Research 23, 346-357; 1990.

[14] Henderson SE, Crapo RO, East TD, Morris AH, Wallace CJ, Gardner RM: Computerized clinical care protocols in an intensive care unit: How well are they followed? In Proceedings of the 14th Annual Symposium on Computer Applications in Medical Care (Ed. RA Miller) pgs. 284-288; 1990.

[15] Sittig DF, Gardner RM, Morris AH, Wallace CJ: Clinical evaluation of computer-based respiratory care algorithms. *International Journal of Clinical Monitoring & Computing* 7(3):177-185; 1990.

[16] Merri M, Fardin DC, Mottley JG, Titlebaum EL: Sampling frequency of the electrocardiogram for spectral analysis of the heart rate variability. *IEEE Trans. Biom. Engin.* Vol 37, No 1, 1990; 99-105.

[17] Beneken JEW, Gravenstein JS. Sophisticated alarms in patient monitoring: A methodology based on systems engineering concepts. in The Automated Anesthesia Record and Alarm Systems.(eds. Gravenstein JS, Newbower RS, Ream AK, Smith NT) Butterworth Publishers, 1987: 211-228.

[18] Carriero N, Gelernter DH. How to Write Parallel Programs; (MIT Press 1990)

[19] Factor M, Gelernter DH. Experience with the process trellis architecture. Yale University, Department of Computer Science Research Report YALEU/DCS/RR-818; 1990.

[20] Factor M, Gelernter DH, Kolb C, Miller PL, Sittig DF. Real-time data fusion in the ICU. (to appear in IEEE Computer, March 1991).

[21] Factor M, Gelernter DH. The Process Trellis: A Software Architecture for Intelligent Monitors. in Proceedings IEEE International Workshop on Tools for Artificial Intelligence: Architectures, Languages, & Tools (TAI89) IEEE Computer Society pgs.174-181;1989.

[22] Factor M. The Process Trellis Software Architecture for Parallel, Real-Time Monitors. PhD Dissertation Yale University, Department of Computer Science, 1990.

[23] Erman LD, Hayes-Roth F, Lesser VR, Raj Reddy D. The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty. *Computing Surveys;* 12(2): 213-253; 1980.

[24] Cohn AI, Rosenbaum SH, Factor M, Sittig DF, Gelernter D, Miller PL: Sequential clinical "scenes": A paradigm for computer-based intelligent hemodynamic monitoring. Proceedings of the 13th Symposium on Computer Applications in Medical Care, (ed. LC Kingsland) pgs. 5-10; 1989 .

[25] Sittig DF, Factor M: Physiologic trend detection and artifact rejection: A parallel implementation of a multi-state Kalman filtering algorithm. *Computer Methods and Programs in Biomedicine*, (31):1-10; 1990.

[26] Cohn AI, Rosenbaum SH, Miller PL. An alternative parallel-computing approach to intelligent hemodynamic monitoring. *Anesthesiology* 73(3A): A541; 1990.

[27] Boehmer RD. Continuous, real-time non-invasive monitor of blood pressure: Penaz method applied to the finger. *J Clin Monit* 3:282-287; 1987.

[28] Pan J, Tompkins WJ: A real-time QRS detection algorithm. *IEEE Trans. Biomed. Engin.* 32(3):230-236; 1985.

[29] Kinias P, Fozzard HA, Norusis MJ. A real-time pressure algorithm. *Comput. Bio. Med.* 11(4):211-220; 1981.

[30] Sittig DF, Clyman JI, Cheung KH, Miller PL: Receiver operating characteristic (ROC) curve helps optimize blood pressure detection algorithms. Anesthesiology Vol. 73(3A):A456; 1990.

[31] Rampil IJ: Intelligent detection of artifact. Chapter 17 pg 175-190; in The Automated Anesthesia Record and Alarm Systems. (eds. JS Gravenstein, RS Newbower, AK Ream, NT Smith) Butterworths, Boston, MA. 1987.

[32] Sittig DF. Characteristic fourier transform spectrum helps identify arterial blood pressure artifact. *Journal of Clinical Monitoring*, 7(1):130-132.

[33] Lipman BS, Massie E, Kleiger RE: Clinical Scalar Electrocardiography, 6th Edition. Year Book Medical Publishers, Inc. Chicago; 1979.

Figure Captions:

Figure 1. A generic trellis hierarchy.

Figure 2. Simplified version of the high-level trellis of the prototype intelligent cardiovascular monitor.

Figure 3. A simplified diagram of the multiple trellises-based intelligent cardiovascular monitor.

Figure 4. Diagram of an electrocardiographic waveform with several of the relevant parameters used to identify the QRS complexes superimposed.

Figure 5. Diagram describing how the important fiducial points are derived from the input waveform.

Figure 6. Example of the receiver operating characteristic (ROC) curve used to "tune" the blood pressure detection algorithm.

Figure 7. Examples of "normal" and artifactual arterial blood pressure waveforms along with their respective fourier spectra. The spectra are shown as the mean of 5 samples ± the standard deviation. The discrimination criteria was the sum of the first 2 harmonics. Values greater than 26.5 (6 SD's above the "normal" mean and 3 SD's below the artifactual mean) were determined to be artifacts.

Adapted from: Sittig DF. Characteristic fourier transform spectrum helps identify arterial blood pressure artifact. *Journal of Clinical Monitoring*, 7(1):130-132.

Figure 8. Diagram of the higher-level artifact detection portion of the ICM. The circles in the middle represent the types of data models for the Kalman filter.

Figure 9. Pseudo-code showing double counting logic.

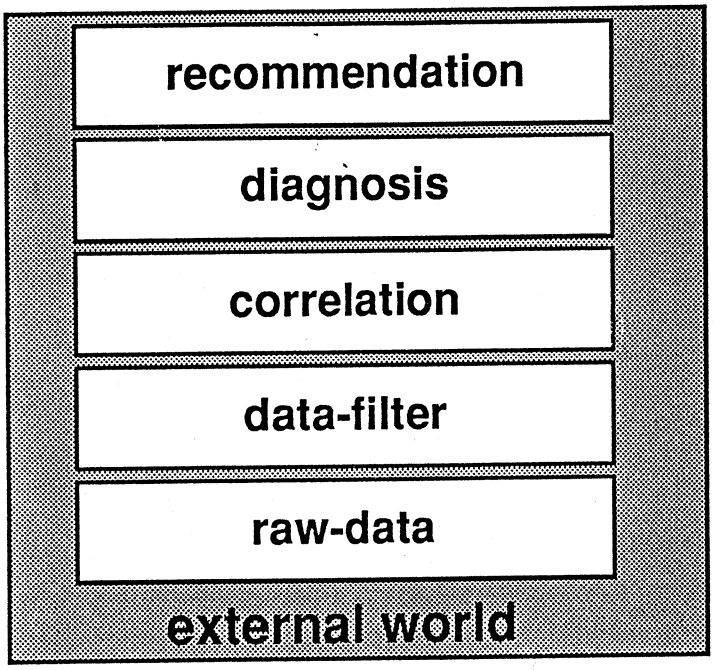


Figure 1. A generic trellis hierarchy.

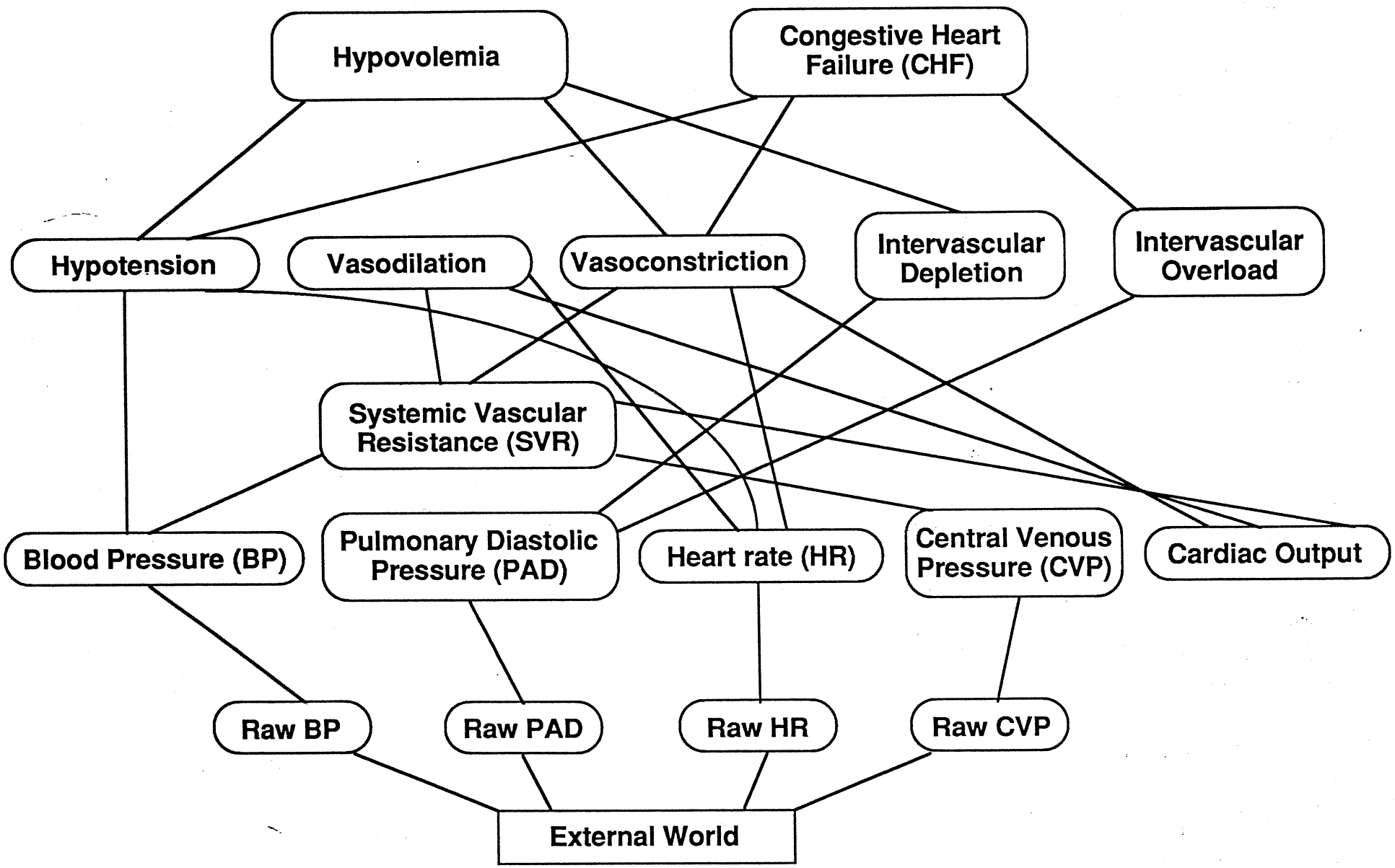


Figure 2. Simplified version of the high-level trellis of the prototype intelligent cardiovascular monitor.

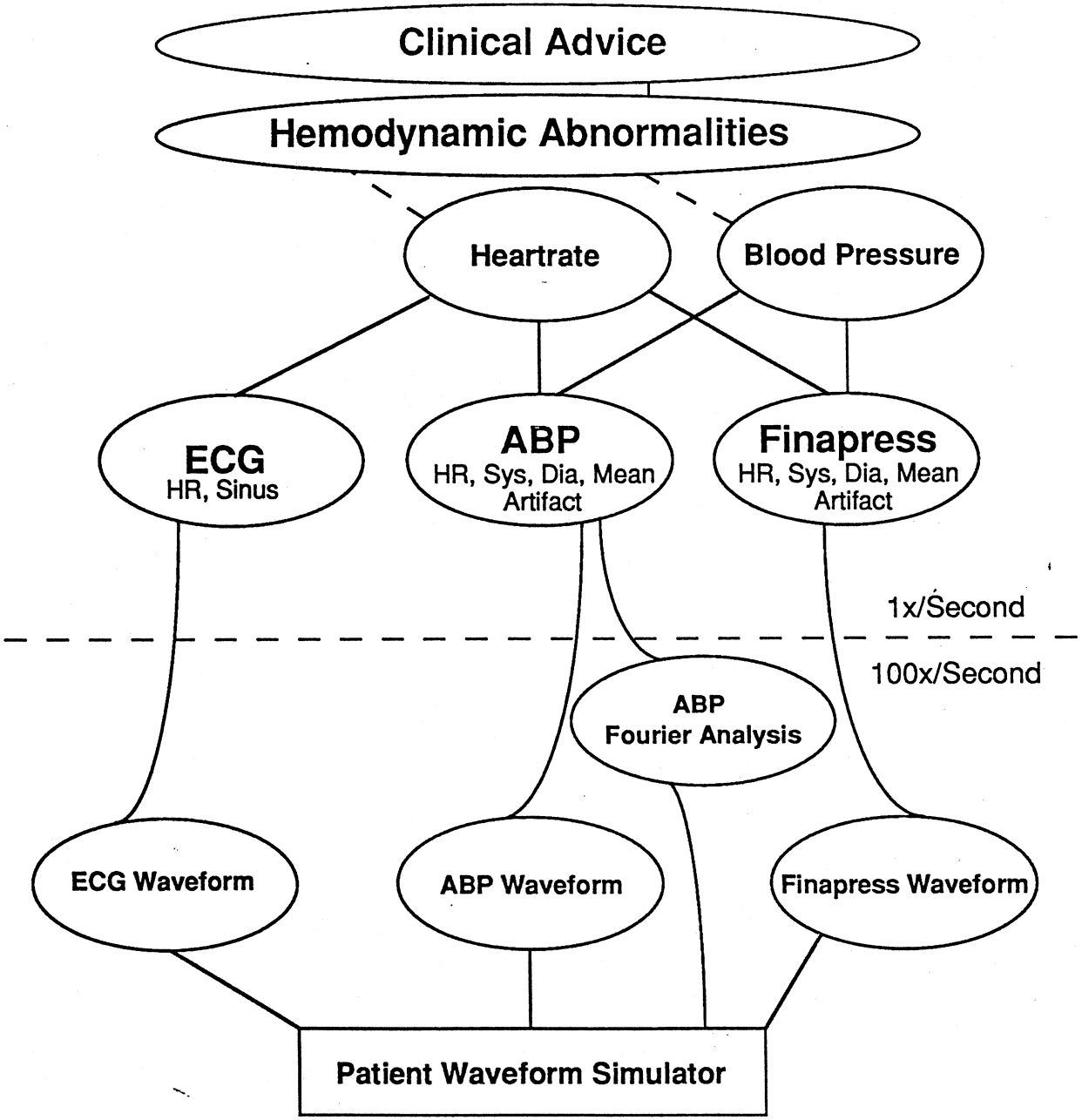

Clinical Advice
Hemodynamic Abnormalities
Heartrate
Blood Pressure
ECG
HR, Sinus
ABP
HR, Sys, Dia, Mean
Artifact
Finapress
HR, Sys, Dia, Mean
Artifact
1x/Second
100x/Second
ABP
Fourier Analysis
ECG Waveform
ABP Waveform
Finapress Waveform
Patient Waveform Simulator

Figure 3

Sample EKG Waveform Analysis

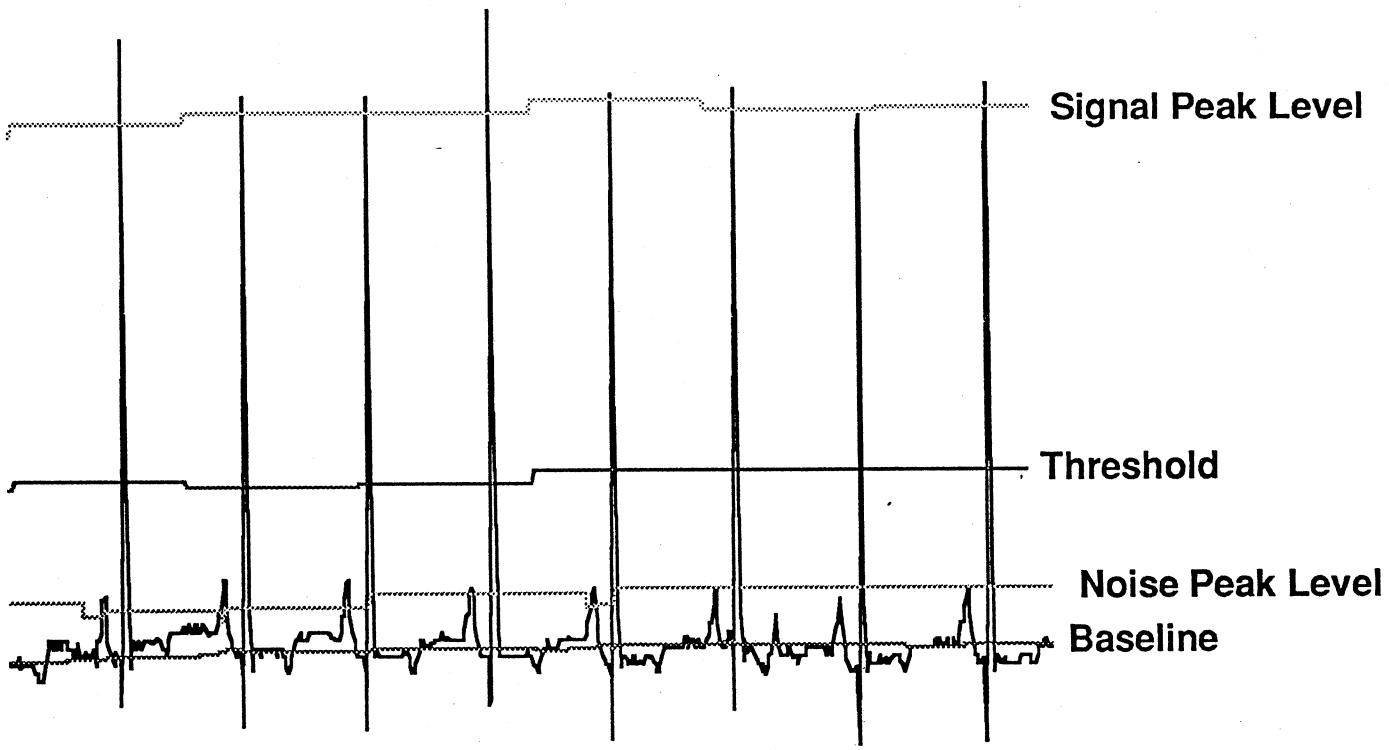
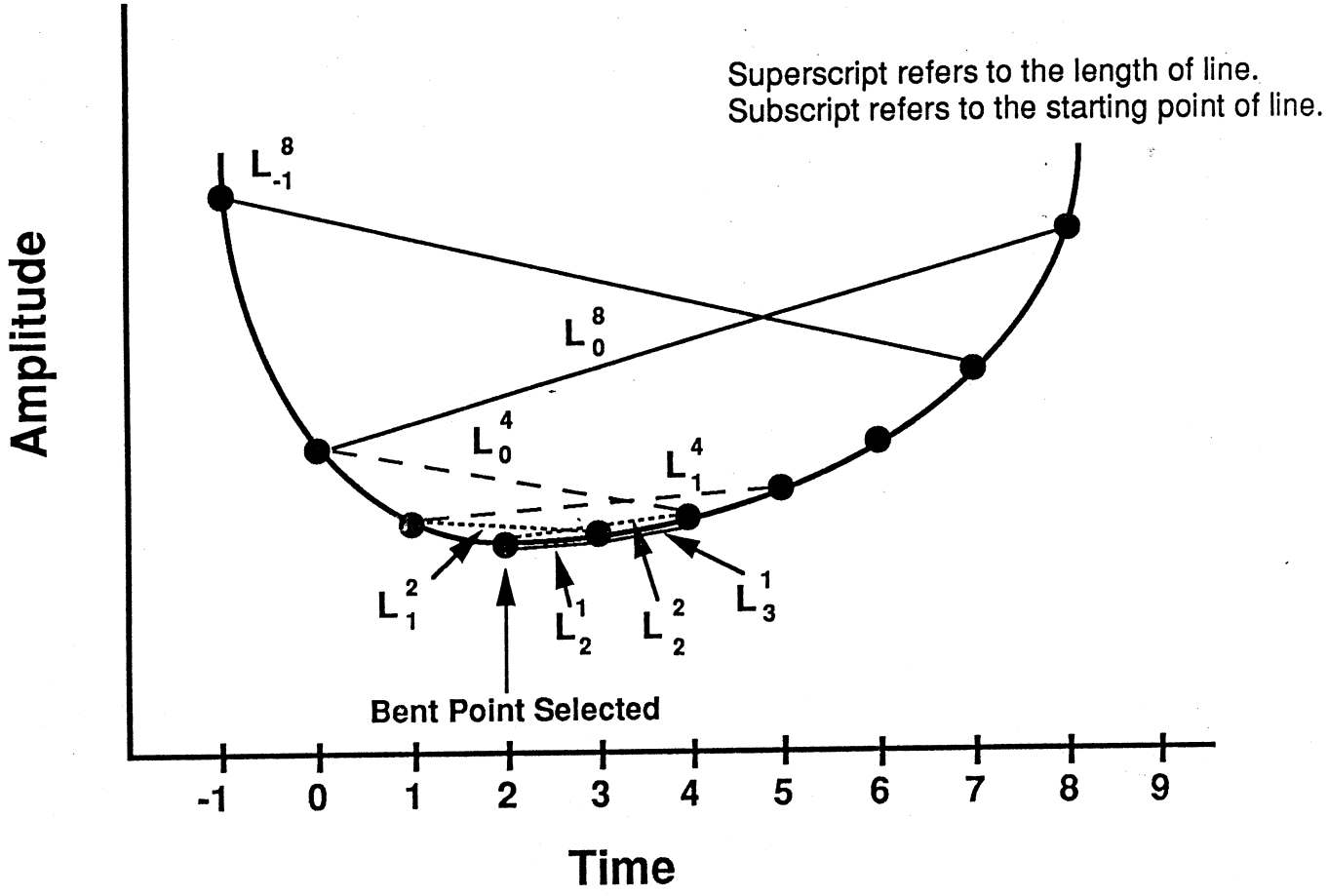


Figure 4

Determination of Critical Points on the Pressure Curve



To find bent points:

1. Calculate slope of line drawn between points 8 units apart.
2. Compare slope of this line to previous line. If similar, go to next pt. If different then
3. Divide interval in half and repeat procedure with new segment.
4. When line length = 1, if no slope change then first pt is bent pt, else middle pt. is bent pt.

Adapted from Kinias *et al.* Comput. Bio. Med. 11(4):211-220; 1981.

Figure 5

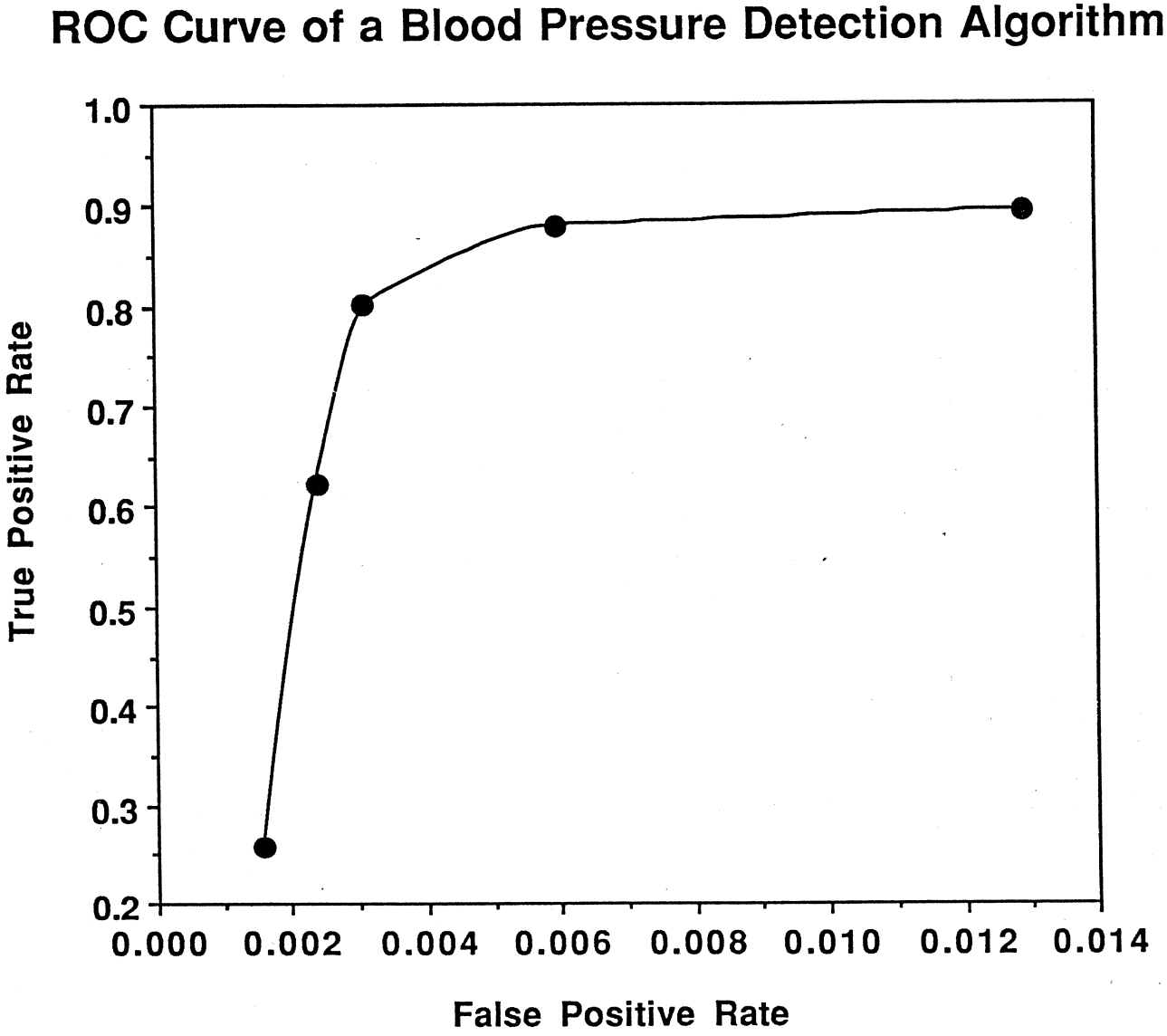
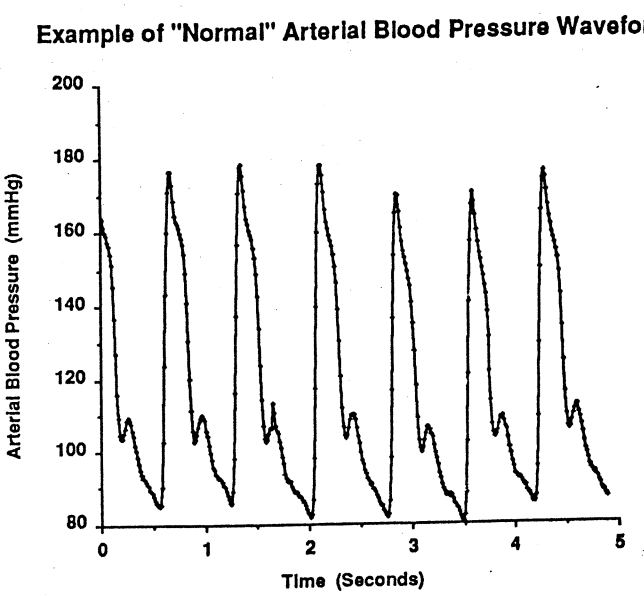
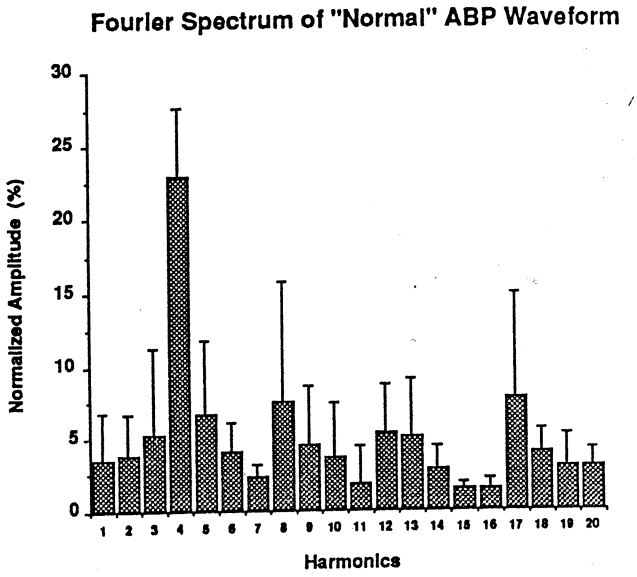
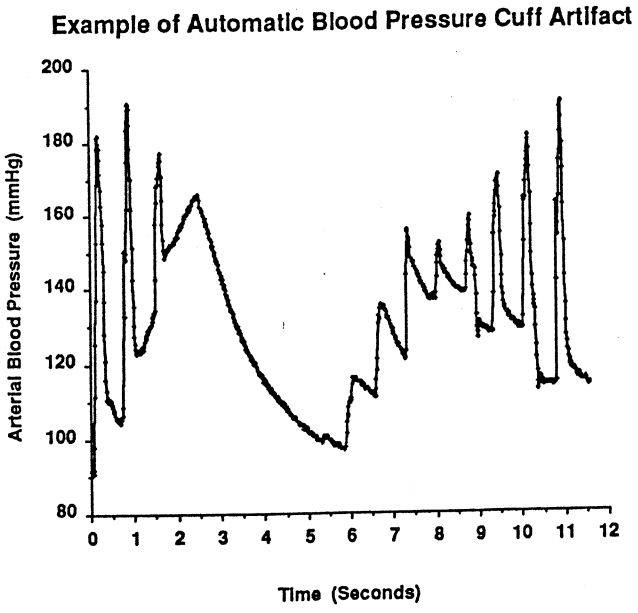

ROC Curve of a Blood Pressure Detection Algorithm
True Positive Rate
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0.014
False Positive Rate

Figure 6







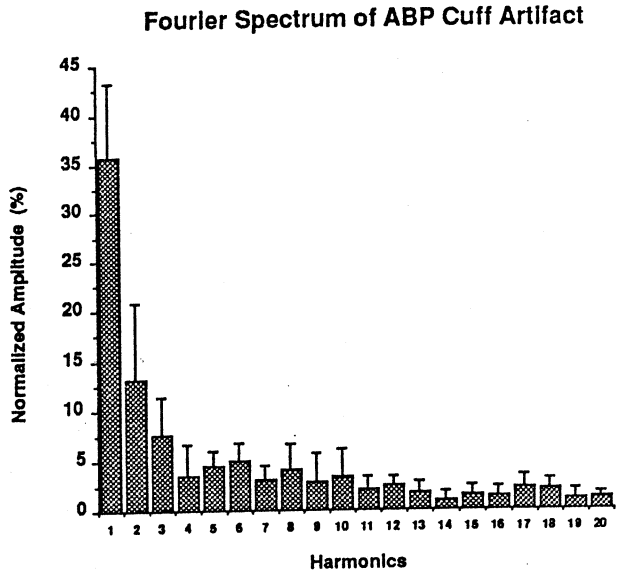


Figure 7: Examples of "normal" and artifactual arterial blood pressure waveforms along with their respective fourier spectra. The spectra are shown as the mean of 5 samples + the standard deviation. The spectra were normalized by dividing each harmonic by the sum of the amplitudes of the first 20 harmonics. The discrimination criteria was the sum of the first 2 harmonics. Values greater than 26.5 (6 SD's above the "normal" mean and 3 SD's below the artifactual mean) were determined to be artifacts.

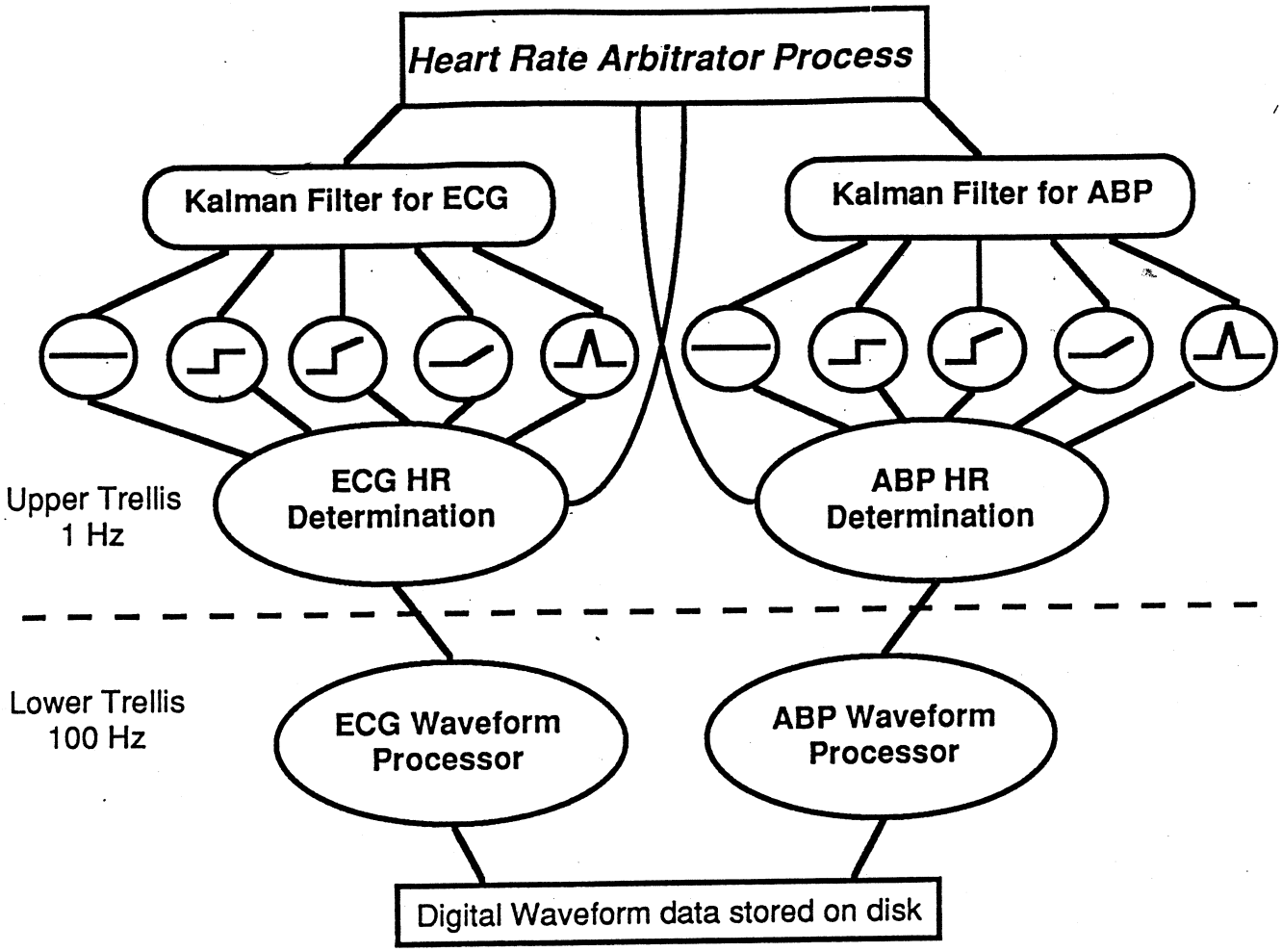


Figure 8

Figure 8. The circles in the middle represent the types of data models for the Multi-state Kalman filter:

Stable Step Step+Slope Slope Transient