GETTING STARTED WITH LABVIEW™ POINT-BY-POINT VIS

Contents

370152B-01 April 2001

LabVIEW offers a set of VIs that perform point-by-point analysis. Point-by-point analysis is ideally suited to real-time data acquisition. When your data acquisition system requires real-time, deterministic performance, you can build a program that uses Point-By-Point versions of array-based LabVIEW analysis VIs.

Real-time performance is a reality for data acquisition. With point-by-point analysis in LabVIEW, data analysis also can utilize real-time performance. The discrete stages of array-based analysis, such as buffer preparation, analysis, and output, might be too slow for higher speed, deterministic, real-time systems. Real-time systems might require continuous analysis, in which analysis occurs for each data point, point by point.

This document explains the concepts and some programming details of point-by-point data analysis. Using the LabVIEW Point-By-Point VIs offers the following benefits:

- You can track and respond to real-time events.
- The analysis process connects directly to the signal for speed and minimal data loss.
- You can perform programming tasks more easily, because you do not allocate arrays, and you make fewer adjustments to sampling rates.
- Analysis synchronizes automatically with data acquisition, because you work with a single signal instantaneously.

Using the LabVIEW Point-By-Point VI Libraries

LabVIEW Point-By-Point VIs correspond to each array-based analysis VI that is relevant to continuous data acquisition. However, you must account for programming differences. You usually have fewer programming tasks when you use point-by-point VIs. The following table describes the characteristic inputs and outputs of a point-by-point VI in LabVIEW.

Table 1. Characteristic Inputs and Outputs for Point-By-Point VIs

Parameter	Description
input data	Incoming data
output data	Outgoing, analyzed data
initialize	Routine that resets the internal state of a VI
sample length	Setting that best represents a significant portion of data for your data acquisition system or computation system

Refer to the *Case Study of Point-By-Point Analysis* section for an example of a point-by-point analysis system.

Initializing Point-By-Point VIs

This section describes when and how to use the point-by-point **initialize** parameter in many Point-By-Point VIs. This section also describes the LabVIEW First Call? function available in LabVIEW 6.0 and later.

Purpose of Initialization in Point-By-Point VIs

Using the **initialize** parameter, you can reset the internal state of VIs without interrupting the continuous flow of data or computation. You can reset a VI in response to events such as the following:

- A user changing the value of a parameter.
- The application generating a specific event or reaches a threshold.

For example, The Value has Changed PtByPt VI located on the **Functions» Point By Point»Other Functions PtByPt** palette, can respond to change events such as the following:

- Receiving the input data.
- Detecting the change.
- Generating a Boolean TRUE value that triggers initialization in another VI.
- Transfering the input data to another VI for processing.

Figure 1 shows the Value has Changed PtByPt VI, triggering initialization in another VI and transferring data to that VI. In this case, the input data is a parameter, value for the target VI.

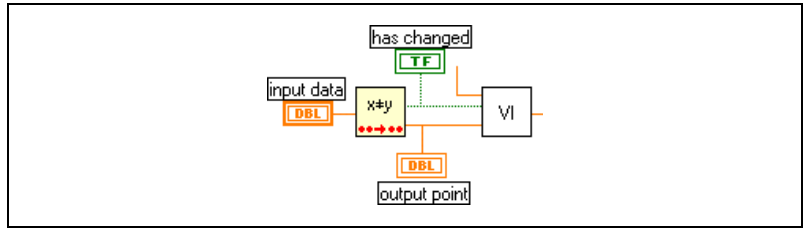


Figure 1. Typical Role of the Value Has Changed PtByPt VI

Using the First Call? Function

Many point-by-point applications do not require use of the **initialize** parameter because initialization occurs automatically whenever an operator quits an application and then starts again. For example, some

applications power down every 24 hours. Powering down constitutes an adequate initialization of the system.

Where necessary, point-by-point VIs contain the LabVIEW First Call? function. In a VI that includes the First Call? function, the internal state of the VI is reset once, the first time you call the VI. The value of the **initialize** parameter in the First Call? function is always TRUE for the first call to the VI. The value remains FALSE for the remainder of the time you run the VI. Use the First Call? function located on the **Functions»Advanced» Synchronization** palette to build Point-By-Point VIs. Figure 2 shows a typical use of the First Call? function with a LabVIEW While loop.

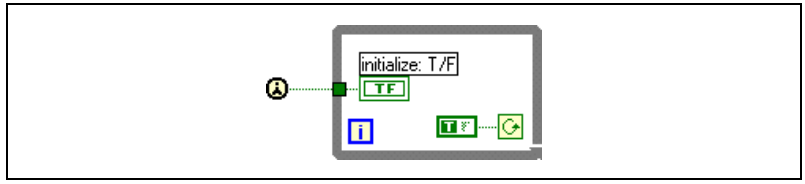


Figure 2. Using the First Call? Function in While Loop

Error Checking and Initialization

Point-By-Point VIs generate errors to help you identify flaws in the configuration of the applications that you build. Several point-by-point error codes exist in addition to the standard LabVIEW error codes.

Error codes usually identify invalid parameters and settings. For higher-level error checking, configure your program to monitor and respond to irregularities in data acquisition or in computation. For example, you create a form of error checking when you range check your data.

A Point-By-Point VI generates an error code once at the initial call to the VI or at the first call to the VI after you initialize your application. Because Point-By-Point VIs generate error codes only once, they can perform optimally in a real-time, deterministic application.

Point-By-Point VIs generate an error code to inform you of any invalid parameters or settings when they detect an error during the first call. In subsequent calls, Point-By-Point VIs set the error code to zero and continue running, generating no error codes. You can program your application to take one of the following actions in response to the first error:

- Report the error and continue running.
- Report the error and stop.
- Ignore the error and continue running. This is the default behavior.

The following programming sequence describes how to use the Value has Changed PtByPt VI to build a point-by-point error checking mechanism for point-by-point VIs that have an **error** parameter.

1. Choose a parameter that you want to monitor closely for errors.
2. Wire the parameter value as **input data** to the Value has Changed PtByPt VI.
3. Transfer the **output data**, which is always the unchanged **input data** in Value has Changed PtByPt VI, to the target VI.
4. The Value has Changed PtByPt VI also outputs a TRUE value whenever the input parameter value changes. Pass the TRUE event to the target VI to trigger initialization, as shown in Figure 1.
5. For the first call that follows this initialization, LabVIEW checks for errors. This initialization and error checking loop runs every time the input parameter changes.

Frequently Asked Questions

This section answers frequently asked questions about point-by-point analysis.

What Are the Differences between Point-By-Point Analysis and Array-Based Analysis in LabVIEW?

Tables 2 and 3 compare array-based LabVIEW analysis to point-by-point analysis from multiple perspectives. In Table 2, the differences between two automotive fuel delivery systems, carburation and fuel injection, demonstrate the differences between array-based data analysis and point-by-point analysis.

Table 2. Comparison of Traditional and Newer Paradigms

Traditional Paradigm	Newer Paradigm
Automotive Technology	
Carburation • Fuel accumulates in a float bowl. • Engine vacuum draws fuel through a single set of metering valves that serve all combustion chambers. • Somewhat efficient combustion occurs.	**Fuel Injection** • Fuel flows continuously from gas tank. • Fuel sprays directly into each combustion chamber at the moment of combustion. • Responsive, precise combustion occurs.

Table 2. Comparison of Traditional and Newer Paradigms (Continued)

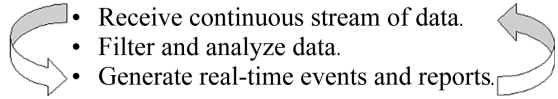
Traditional Paradigm	Newer Paradigm
Data Analysis Technology	
Array-Based Analysis • Prepare a buffer unit of data. • Analyze data. • Produce a buffer of analyzed data. • Generate report.	**Point-by-point Analysis**  • Receive continuous stream of data. • Filter and analyze data. • Generate real-time events and reports.

Table 3 presents other comparisons between array-based and point-by-point analysis.

Table 3. Comparison of Array-Based and Point-By-Point Data Analysis

	Array-Based Analysis	Data Acquisition & Analysis with Point-By-Point VIs
Compatibility	Limited compatibility with real-time systems	Compatible with real-time systems; backward compatible with array-based systems
Data typing	Array-oriented	Scalar-oriented
Interruptions	Interruptions critical	Interruptions tolerated
Operation	You observe, offline	You control, online
Performance and programming	Compensate for start-up data loss (4–5 seconds) with complex “state machines”	Start-up data loss does not occur; initialize the data acquisition system once and run continuously
Point of view	Reflection of a process, like a mirror	Direct, natural flow of a process
Programming	Specify a buffer	No explicit buffers
Results	Output a report	Output a report and a event in real time
Run-time behavior	Delayed processing	Real time
Run-time behavior	Stop	Continue
Run-time behavior	Wait	Now
Work style	Asynchronous	Synchronous

Why Use Point-By-Point Analysis?

Point-by-point analysis works well with computer-based real-time data acquisition. In array-based analysis, the input-analysis-output process takes place for subsets of a larger data set. In point-by-point analysis, the input-analysis-output process takes place continuously, in real time.

What is New about Point-By-Point Analysis?

When you perform point-by-point analysis, keep in mind the following concepts:

- Initialization—You must initialize the point-by-point analysis application to prevent interference from settings you made in previous sessions of data analysis.
- Re-Entrant Execution—You must enable LabVIEW re-entrant execution for point-by-point analysis. Re-entrant execution allocates fixed memory to a single analysis process, guaranteeing that two processes that use the same analysis function never interfere with each other.

Note If you create custom VIs to use in your own point-by-point application, be sure to enable re-entrant execution. Re-entrant execution is enabled by default in almost all Point-By-Point VIs.

- Deterministic Performance—Point-by-point analysis is the natural companion to many deterministic systems, because it efficiently integrates with the flow of a real-time data signal.

What Is Familiar about Point-By-Point Analysis?

Point-by-point analysis in LabVIEW is familiar because the approach for most analysis operations remains the same. Use filters, integration, mean value algorithms, and so on, in the same situations and for the same reasons that you use these operations in array-based data analysis. In contrast, the computation of zeroes in polynomial functions is not relevant to point-by-point analysis, and point-by-point versions of these array-based VIs are not necessary.

How is it Possible to Perform Analysis without Buffers of Data?

Analysis functions yield solutions that characterize the behavior of a data set. In array-based data acquisition and analysis, you might analyze a large set of data by dividing the data into 10 smaller buffers. Analyzing those 10 sets of data yields 10 solutions. You can further resolve those 10 solutions into one solution that characterizes the behavior of the entire data set.

In point-by-point analysis, you analyze an entire data set in real-time. A sample unit of a specific length replaces a buffer. The point-by-point sample unit can have a length that matches the length of a significant event in the data set that you are analyzing. Refer to the *Case Study of Point-By-Point Analysis* section for an example that acquires a few thousand samples per second to detect defective wheels. In this case, the signal comes from a train that is moving at 60 to 70 km per hour. The sample length for this application corresponds to the minimum distance between wheels.

A typical point-by-point analysis application analyzes a long series of sample units, but you are likely to have interest in only a few of those sample units. To identify those crucial samples of interest, the application focuses on transitions, such as **end of relevant signal**.

In the *Case Study of Point-By-Point Analysis* section, a sample train wheel detection application uses **end of signal** to identify crucial samples of interest. The instant that the application identifies this transition point, the application captures the maximum amplitude reading of the current sample unit. This particular amplitude reading corresponds to the complete signal for the wheel on the train whose signal has just ended. You can use this real-time amplitude reading to generate an event or a report about that wheel and that train.

Why is Point-By-Point Analysis Effective in Real-Time Applications?

In general, when you must process continuous, rapid data flow, point-by-point analysis can respond. For example, in industrial automation settings, control data flows continuously, and computers use a variety of analysis and transfer functions to control a real-world process. Point-by-point analysis can take place in real time for these engineering tasks.

Some real-time applications do not require high-speed data acquisition and analysis. Instead, they require simple, dependable programs. Point-by-point analysis offers simplicity and dependability, because you do not allocate arrays explicitly, and data analysis flows naturally and continuously.

Do I Need Point-By-Point Analysis?

You can continue to work without point-by-point analysis as long as you can control your processes without high-speed, deterministic, point-by-point data acquisition. However, if you dedicate resources in a real-time data acquisition application, use point-by-point analysis to achieve the full potential of your application. As you increase the samples-per-seconds rate by factors of ten, the need for point-by-point analysis increases.

The point-by-point approach simplifies the design, implementation, and testing process, because the flow of the application closely matches the natural flow of the real-world processes that you want to monitor and control.

What is the Long-Term Importance of Point-By-Point Analysis?

Real-time data acquisition and analysis continue to demand more streamlined and stable applications. Point-by-point analysis is streamlined and stable, because it directly ties into the acquisition and analysis process. Streamlined and stable point-by-point analysis allows the acquisition and analysis process to move closer to the point of control in FPGA (field programmable gate array) chips, DSP chips, embedded controllers, dedicated CPUs, and ASICs.

Examples

Real-time analysis and control are just some of the advantages of the Point-By-Point paradigm. The following examples demonstrate advantages to using Point-By-Point VIs and Functions.

Point-By-Point and Array Based Filter Example

The PtbyPt and Array Based Filter VI, shown in Figure 3, generates a signal that contains undesired noise. The VI filters the signal to remove the noise and provide a better representation of the actual signal. The filter blocks unwanted frequencies in the signal and allows the important frequencies to pass. The VI uses two methods for filtering the unwanted noise from the signal. The first method removes unwanted noise by using the Point-By-Point version of the filter. The second method uses the array-based version of the filter.

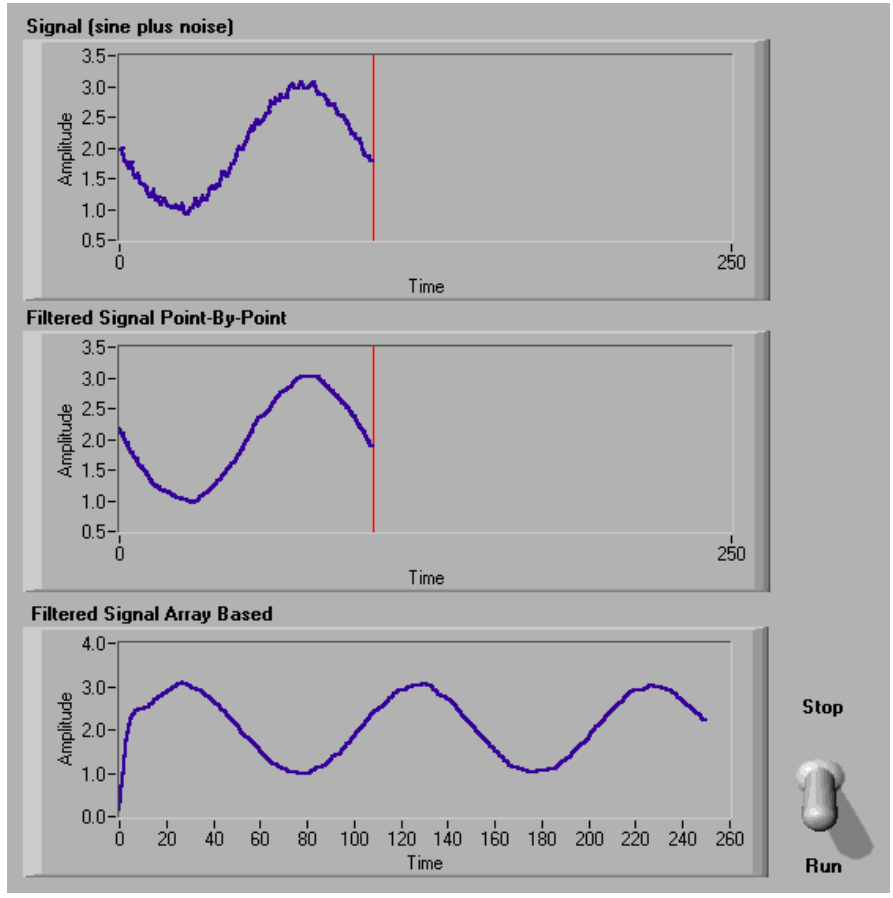


Figure 3. Point-By-Point versus Array Based Filter

Point-By-Point Based Filter

In point-by-point LabVIEW programming, a VI acquires a single point of data, analyzes the point, and then makes it available for reporting. At the same time, the VI acquires another single point of data and repeats the process, point-by-point. In the point-by-point filter, the VI analyzes each point of the incoming signal and reports one after the other in real time.

Array Based Filter

In array-based LabVIEW programming, a VI acquires a set of data, analyzes the data, and generates a report. In the array-based filter, the VI acquires and analyzes a length of the signal. After analysis, the filter removes the unwanted frequencies from the data set, and the VI displays a report of the filtered data set in the graph. The graph shows the entire

filtered signal. The VI then waits until another length of signal is acquired and repeats the process.

Realtime Amplitude Spectrum Example

The Realtime Amplitude Spectrum VI, shown in Figure 4, generates a sample signal and adds random noise. The VI analyzes the signal and displays a power spectrum of the signal on a graph.

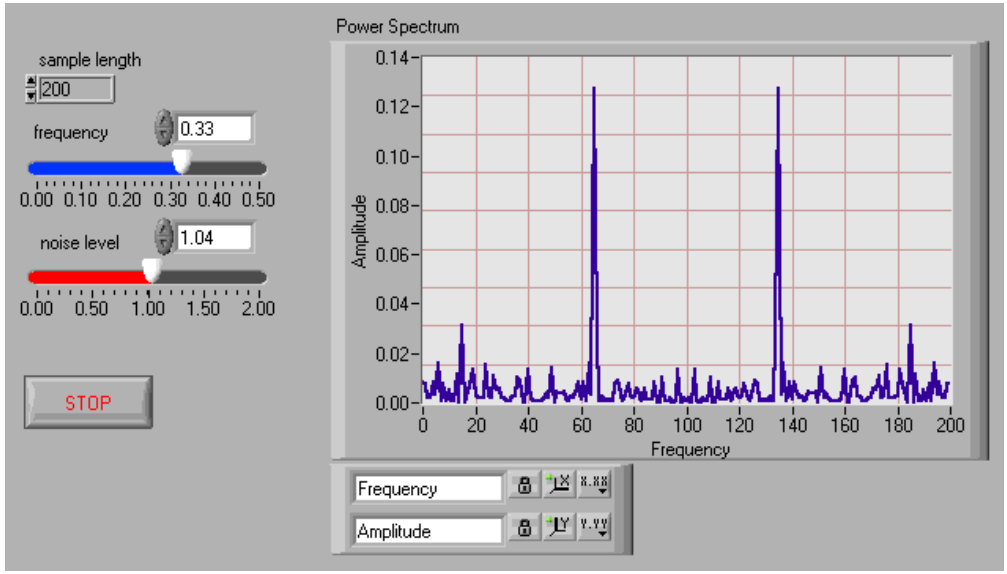


Figure 4. Power Spectrum Analysis with Real Time Controls

This VI uses Point-By-Point VIs so you can change the frequency, noise level, and sample lengths in real time. These changes effect the power spectrum in real time. The Point-By-Point paradigm is very useful in data acquisition and analysis that demand a real-time level of detail.

Moving Histogram PtByPt

The Moving Histogram PtByPt VI, shown in Figure 5, generates a signal, analyzes it, and then creates a histogram. You can adjust intervals and sample lengths in real time.

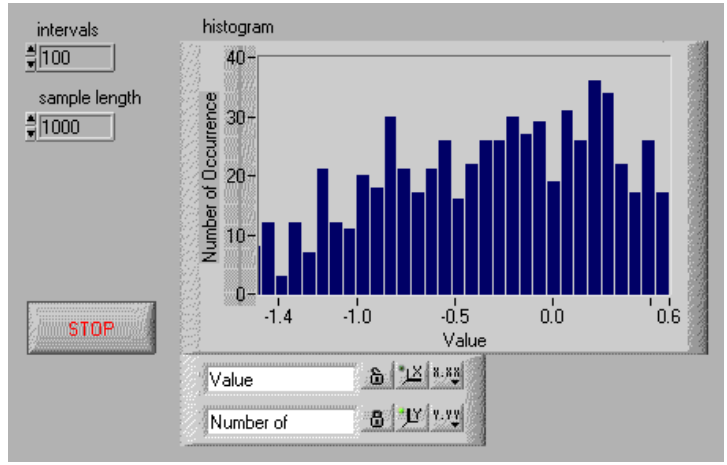


Figure 5. Moving Histogram Created with Point-By-Point VIs

Creating a moving histogram is possible in the array-based paradigm of LabVIEW, but it is much more difficult. The Point-By-Point paradigm allows you to create VIs with real-time behavior very easily.

Case Study of Point-By-Point Analysis

The case study in this section shows a complete point-by-point analysis application built in LabVIEW. A real-time data acquisition application that detects defective train wheels demonstrates the simplicity and flexibility of point-by-point data analysis. The Train Wheel PtByPt VI uses Point-By-Point VIs.

Point-By-Point Analysis of Train Wheels

In this example, the maintenance staff of a train yard must detect defective wheels on a train. A railroad worker can strike a flawed wheel with a hammer and hear a different resonance that identifies a flaw. Automated surveillance must replace manual testing, because manual surveillance is too slow, too prone to error, and too crude to detect subtle defects. An automated solution also adds the power of dynamic testing, because the train wheels can be in service during the test, instead of standing still.

The solution to detect potentially defective train wheels must detect even subtle signs of defects quickly and accurately. The application should gather data when a train travels during a normal trip. The application should collect and analyze data in real time to simplify programming and to increase speed and accuracy of results.

The Train Wheel PtByPt VI uses the point-by-point analysis capabilities of LabVIEW to construct a solution for detecting defective train wheels.

Figures 3 and 4 show the front panel and the block diagram for the Train Wheel PtByPt VI, respectively. You can download the Train Wheel PtByPt VI and other point-by-point programming example VIs used in this manual from the National Instruments Developer Zone at `ni.com/zone`

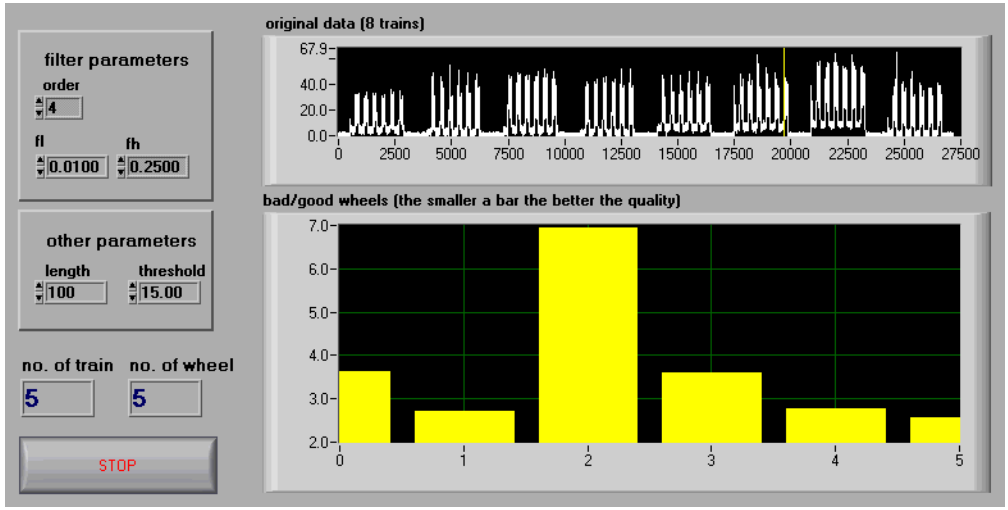


Figure 6. Front Panel of the Train Wheel PtByPt VI

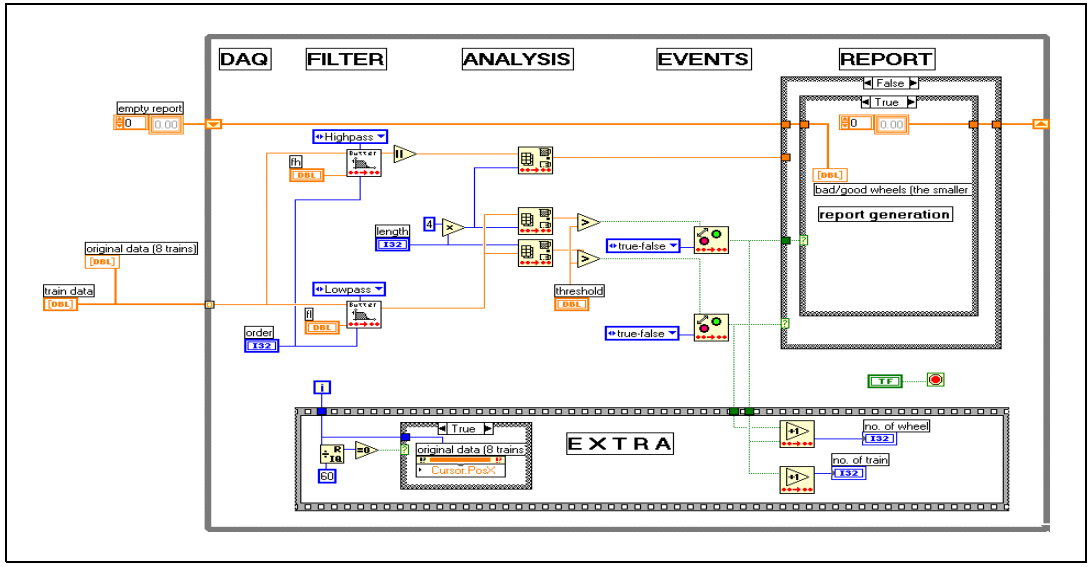


Figure 7. Train Wheel PtByPt VI

Note This example focuses on implementing a point-by-point analysis program in LabVIEW. The issues of ideal sampling periods and approaches to signal conditioning are beyond the scope of this example.

Overview of the LabVIEW Point-By-Point Solution

The data that the train wheel detection application acquires flows continuously through a While Loop. The following process describes what occurs in the loop:

1. DAQ—Flow of waveform data.
2. Filter—Separation of low- and high-frequency components.
3. Analysis—Detection of train, wheel, and energy level of waveform for each wheel.
4. Events—Response to signal transitions of wheels and trains wheels.
5. Report—Logging of trains, wheels, and trains that might have defective wheels.

The detection application requires standard LabVIEW programming objects, such as Case structures, While Loops, numeric controls, and numeric operators. The Train Wheel PtByPt VI program also requires the VIs listed in the following sections.

Characteristics of a Train Wheel Waveform

The characteristic waveform that train wheels emit determines how you analyze and filter the waveform signal point-by-point. A train wheel in motion emits a signal that contains low- and high-frequency components. If you mount a strain gauge in a railroad track, you detect a noisy signal, similar to a bell curve. Figure 8 shows the low- and high-frequency components of this curve.

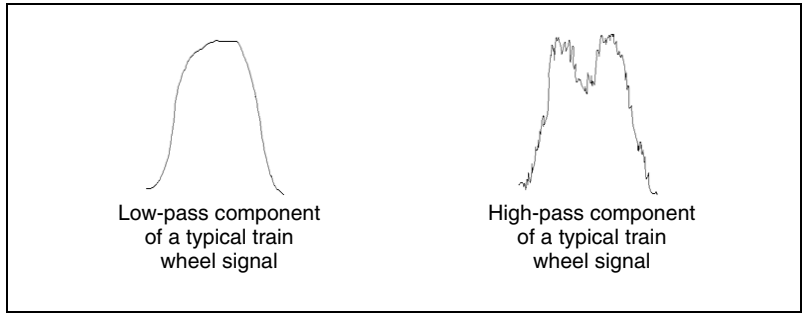


Figure 8. Low- and High-Frequency Components of a Train Wheel Signal

The low-frequency component of train wheel movement represents the normal noise of operation. Defective and normal wheels generate the same low-frequency component in the signal. The peak of the curve represents the moment when the wheel moves directly above the strain gauge. The lowest points of the bell curve represent the beginning and end of the wheel, respectively, as the wheel passes over the strain gauge.

The signal for a train wheel also contains a high-frequency component that reflects the quality of the wheel. In operation, a defective train wheel generates more energy than a normal train wheel. In other words, the high frequency component for a defective wheel has greater amplitude.

Parameters of Waveforms in the Train Wheel PtByPt VI

The waveform of all train wheels, including defective ones, falls within predictable ranges. This predictable behavior allows you to choose the parameters that appear in Table 4. These parameters apply to the five stages described in the *Overview of the LabVIEW Point-By-Point Solution* section.

Note The parameters in the table can help you understand the details of programming with the point-by-point approach. However, you can understand point-by-point acquisition and analysis without studying the parameters in the table. Remember, you must adjust parameters for any implementation of the Train Wheel PtByPt VI, because the characteristics of each data acquisition system differ.

Table 4. Acceptable Parameters for Train Wheel Waveforms in the Train Wheel PtByPt VI

Parameter	Name	Stage and VI Affected	Purpose in Train Wheel PtByPt
Data input	**input data**	Data Acquisition Stage/ DAQ In.VI	Data source, device, channel, samples per second, and scaling.
Filter resolution	**order**	Filtering Stage/ Butterworth Filter PtByPt	Amount of the waveform data that the VI filters at a given time. `2` is acceptable for Train Wheel PtByPt. **order** applies to both Butterworth filters in Train Wheel PtByPt.
Low cutoff frequency	**fl**	Filtering Stage/ Butterworth Filter PtByPt	Minimum signal strength that identifies the departure of a train wheel from the strain gauge. `0.01` is acceptable for Train Wheel PtByPt.

Table 4. Acceptable Parameters for Train Wheel Waveforms in the Train Wheel PtByPt VI (Continued)

Parameter	Name	Stage and VI Affected	Purpose in Train Wheel PtByPt
High cutoff frequency	**fh**	Filtering Stage/ Butterworth Filter PtByPt	Minimum signal strength that identifies the end of high frequency waveform information. `0.25` is acceptable for Train Wheel PtByPt.
Length of waveform to analyze	**sample length**	Analysis Stage/ Array Max & Min PtByPt	Size of the portion of the waveform that Train Wheel PtByPt analyzes. To calculate the ideal length of sample, consider the speed of the train, the minimum distance between wheels, and the number of samples you receive per second. `100` is acceptable for Train Wheel PtByPt. The Train Wheel PtByPt VI uses **sample length** to calculate values for all three Array Max & Min PtByPt VIs.
Initialization	**initialize**	All point-by-point VIs	Routine that resets a VI for a new session of continuous data acquisition.
(Multiplier)	(none)	Analysis Stage/ Array Max & Min PtByPt	Sets a longer portion of waveform to analyze. When this longer portion fails to display signal activity for train wheels, the VI identifies the end of the train. `4` is acceptable for Train Wheel PtByPt.
(Threshold)	(none)	Analysis Stage/ Array Max & Min PtByPt	Provides a comparison point to identify when no train wheel signals exist in the signal that you are acquiring. `3` is acceptable for Train Wheel PtByPt.

Sampling Rate in the Train Wheel PtByPt VI

The point-by-point detection application operates on the continuous stream of waveform data that comes from the wheels of a moving train. For a train moving at 60 to 70 kilometers per hour, a few hundred to a few thousand samples per second are likely to give you sufficient information to detect a defective wheel.

Filtering Requirements of the Train Wheel PtByPt VI

The point-by-point detection application must filter low- and high-frequency components of the train wheel waveform. Two point-by-point Butterworth filters perform the following tasks:

- Extract the low-frequency components of the waveform.
- Extract the high-frequency components of the waveform.

Analysis Components of the Train Wheel PtByPt VI

The point-by-point detection application must analyze the low- and high-frequency components separately. In this case a point-by-point version of an Array Max & Min VI extracts waveform data that reveals the level of energy in the waveform for each wheel, end of train, and end of wheel.

Three separate point-by-point Array Max & Min VIs perform the following discrete tasks:

- Identify the maximum high-frequency value for each wheel.
- Identify the end of each train.
- Identify the end of each wheel.

Note The name Array Max & Min PtByPt VI contains the word Array only to match the name of the array-based form of this VI. You do not need to allocate arrays for the PtByPt version of this VI.

After the Analysis stage identifies maximum and minimum values, a separate Events stage detects when these values cross a threshold setting.

Event Components of the Train Wheel PtByPt VI

The point-by-point detection application logs every wheel and every train that it detects. The Boolean Crossing VI generates an event every time the Array Max & Min PtByPt VI detects the end of a train wheel. The data acquisition system also generates an event every time the Array Max & Min PtByPt VI detects the end of a train. Analysis of the high-frequency signal identifies which wheels, if any, might be defective. When the VI encounters a potentially defective wheel, the VI passes the event directly to the report at the moment the report receives the end-of-wheel event.

Two point-by-point Boolean Crossing VIs perform the following tasks:

- Detect the transition point in the signal that indicates the end of wheel.
- Detect the transition point in the signal that indicates the end of train.

The Boolean Crossing PtByPt VI responds to transitions. When the amplitude of a single wheel waveform falls below a specific level, the end of the wheel has arrived at the strain gauge. For this data acquisition system, 3 is a good threshold level to identify the end of wheel. When the signal strength falls below this level, the Boolean Crossing VI recognizes a transition event and passes that event to a report.

Report Components of the Train Wheel PtByPt VI

The point-by-point detection application reports on all wheels for all trains that pass through the data acquisition system. The system also reports any wheels that are likely to be defective.

Every time a wheel passes the strain gauge, the application captures its waveform, analyzes it, and reports the event. Table 5 describes the components of a report on a single train wheel.

Table 5. Example Report on a Single Train Wheel

Information Source	Meaning of Results
Counter mechanism for waveform events	Stage One: Wheel number four has passed the strain gauge.
Analysis of high-pass filter data	Stage Two: Wheel number four has passed the strain gauge and the wheel might be defective.
Counter mechanism for end-of-train events	Stage Three: Wheel number four in train number eight has passed the strain gauge and the wheel might be defective.

The Train Wheel PtByPt VI uses point-by-point analysis to generate a report, not to control an industrial process. However, this data acquisition system acquires data in real time, and you can modify the application to generate real-time control responses, such as stopping the train when it encounters a potentially defective wheel.

Conclusion

When acquiring data with real time performance, point-by-point analysis helps you analyze data in real time. Point-by-point analysis occurs continuously and instantaneously. While you acquire data, you filter and analyze it, point by point, to extract the information you need and to make an appropriate response. This case study demonstartes the effectiveness of the point-by-point approach for generation of both events and reports in real time.