Rotating Machinery Analysis Module

Command reference and application guide

Rotating Machinery Analysis Module, Command reference and application guide

Version 1.00

Microstar Laboratories, Inc.
2265 116 Avenue N.E.
Bellevue, WA 98004
Tel: (425) 453-2345
Fax: (425) 453-3199
http://www.mstarlabs.com

Part Number: MSROTMMOD100

Contents

1. Introduction

The downloadable Rotating Machinery Analysis Module (ROTM.DLM) for xDAP systems provides a new way to capture and analyze measurements from rotating machines.

Traditional Approaches

Previously, there were two options for measuring a rotating machine.

1. *Capture your data according to the timing from a fixed-rate sampling clock.*

This approach gives high-resolution data relative to a consistent elapsed-time reference, but this is not always the most convenient, often not useful at all. Things like power transfer rates and resonant mechanical frequencies are best observed in time-based data, but these are often not of special interest. There is no direct information in the data set about position. The number of rotations covered in any one data set will vary inversely with the speed. Suppose for example that you are studying harmonic distortion for a generator under load. An FFT data analysis is relatively straightforward if the number of samples captured per rotation happens to be the exact number of samples that an FFT data block requires. But this is a rarity. The slightest variation in speed will cause misalignment between the rotation angles and the ends of the data block, leading to some severe complications for later data analysis.

2. *Capture your data under hardware control of an external timing reference signal.*

With sampling slaved to the signals from an external encoder device that rotates at the speed of the equipment, the number of samples per rotation and the angular positions for data capture will be strictly fixed, making a data analysis relatively straightforward. But this approach is inherently limited. You can't know the elapsed time or actual frequency, because the time between samples varies with speed. The slower the speed, the lower the time-resolution of measurements. Important information could be missed, and high frequencies would not be properly resolved. Even though there is never a problem with a poorly aligned or improperly-sized data block, the encoder hardware could restrict you to an inconvenient data block size.

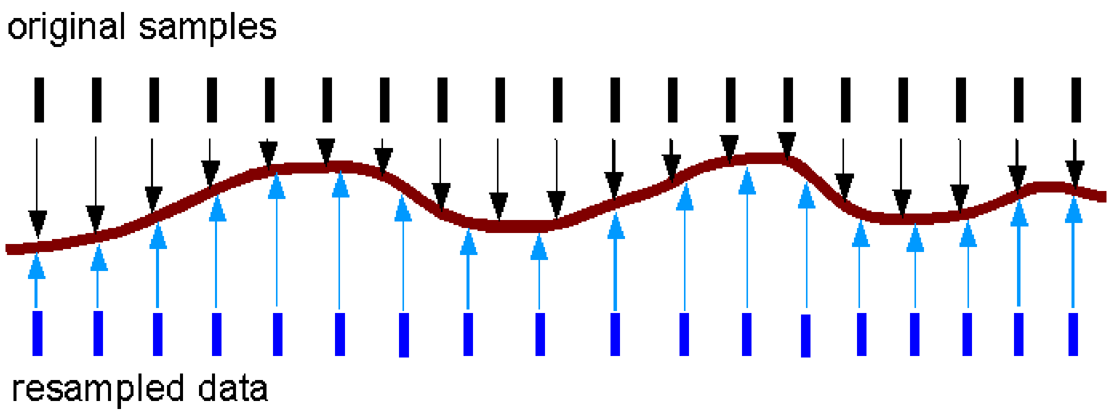
The *“Rotating Machinery Analysis”* alternative

The *Rotating Machinery Analysis* module combines the best features of time-based sampling and encoder-based sampling. The data are captured at a very high rate, based on a precise, stable digital oscillator. The sampling process measures not only your signals, but also your encoder timing pulses, as data. A relationship is established between locations in time and locations in rotation angle. Data can then be obtained from the high resolution data stream at desired locations relative to the encoder pulses. For example, if you have a 1 degree encoder (360 pulses-per-rotation), you can sample at 1024 locations-per-rotation, even though the hardware doesn't support this directly.

The relationship to “*resampling*”

This document makes frequent references to “resampling.” Sampling is virtually instantaneous – measurements that you obtain with an xDAP represent the value of the input signals at extremely narrow sampling instants. There is no guarantee that instants at which samples are captured using a high-resolution sampling clock will line up perfectly with the instants at which an encoder signal transition occurs. Quite the opposite – sample values that would be captured under encoder hardware control rarely “line up” exactly.

Given the abundance of samples captured, however, there are many samples available in the vicinity of every encoder pulse edge. Using well-known DSP techniques, there is plenty of information to reconstruct the value that would have been measured *if the signal had been measured exactly at the encoder pulse edge*. This is sometimes called “interpolation” or “time-shifting.”



The *resampling technology* is well-established and time-proven, used in all kinds of digital processing applications every day, even if you haven't heard much about it before. Though resampling has known limitations, it turns out that the hardest question is *where should the resampling be done* in the presence of continuously changing rates. For a quick introduction, see **Appendix I** in this manual.

2. Configuring the xDAP

The Rotating Machinery Analysis commands are run within the DAPL 3000 system environment, by the xDAP processor, at the time that the samples are captured. Your data arrive at your PC host application ready to use, with all of the calculations completed.

> When you first receive your Rotating Machinery Analysis module, first install your DAP software and your xDAP unit. Run the *Data Acquisition Processor* control panel application in your Windows system, and select the *Modules* tab. Click on the Add button in the lower left. In the pop-up dialog, click the DAPL3000 button, and click the Browse button in the lower right corner to locate the copy of the module on your system. When you find it, select it and click *Open*. Use the default options. Back in the "Adding a module" dialog, click the *OK* button in the lower left corner. Your module should now be included in the DAPL system software, and it will be loaded automatically each time the DAPL system is initialized on your xDAP unit.
>
> If you are going to record data at both high-rate clocked and encoder-clocked rates, passing the data streams to the host through separate channels to avoid rate problems, you should use the *Browser* tab in the *Data Acquisition Processor* control panel application to set up the `Cp2Out` pipe as a second channel from the DAP to the host PC for binary data transfers. Right-click on the *Compipes* node and then the *Create* option. Your added pipe will be available each time the DAPL system is initialized.

An xDAP application typically uses a text file to deliver the configuration commands, and the commands are downloaded from the file all at once to the xDAP unit. Using the DAPstudio software, this script file is generated on-the-fly when needed. The results of running this configuration are *preprocessing* operations necessary to obtain the best possible representation of your data. That is not the end of the story: various kinds of *application programming* are usually required in the host system to extract the final information from the data sets. This manual will not go into the details of application programming – for which every application is different.

The only thing special about Rotating Machinery Analysis is the use of some special commands in the processing configuration sent to your xDAP. The following is a typical application configuration, with explanations to follow:

```
// DAPL configuration for rotating machinery analysis

// Clear any remnants from past configurations
reset

// Configure the required data pipes
pipes  pEnc double, pOutMux word

// Configure the input sampling
idefine  highrate
  channels  8
  set  ip0   d0      // encoder data pulses
  set  ip1   d2      // encoder reference pulses
  set  ip2   d4      // first data channel
  ...
  set  ip7   d14     // last data channel
  scan  10.0
end

// Configure resampling
pdefine  resampling
  // Analyze encoder signal to determine sample locations
  ENCDRATE(720, ip0, ip1, 0, 14750, ANALOG, 1000, pEnc)
  // Resample selected signals at encoder rates
  ADJRESAMP(ipipe(2,3,4,5), 4, pEnc, FAST, pOutMux)

  // Transfer resampled data to host
  // Data source: channels 2, 3, 4, 5 (resampled)
  COPY(pOutMux,$BinOut)

  // Transfer high-rate time-based data to host (raw)
  // Data source: channels 2, 3, 6, 7 (raw time-based)
  COPY(ipipe(2,3,6,7), Cp2Out)
end
```

Analyzing this configuration example line by line:

-
  ```
  reset
  ```
  Clears any residual configurations or data left over from past activity.

-
  ```
  pipes  pEnc double , pOutMux word
  ```
  Reserves pipes for intermediate data movement, related to timing analysis and resampled data.

-
  ```
  idefine  highrate
    channels  8
    ...
  end
  ```
  Define an input sampling configuration capturing 8 channels each scan interval

-
  ```
    set  ip0   d0
    set  ip1   d2
  ```
  Use one channel to observe encoder data timing pulses. Use one channel to observe encoder reference pulses.

-
  ```
    set  ip2   d4
    set  ip3   d6
    set  ip4   d8
    set  ip5   d10
    set  ip6   d12
    set  ip7   d14
  ```
  Configure six more channels to capture various other input signals. Hardware channels are assigned in such a way that all six channels can be captured simultaneously, along with the encoder pulse observations.

-
  ```
    Scan  2.0
  ```
  Configure the channel scan rate so that every channel and the encoder pulses are scanned sufficiently fast. At 8000 RPM, this configuration with a 720 pulse-per-revolution encoder would have about 5 samples per encoder pulse, which is about bare minimum.

-
  ```
  pdefine  resampling
    ...
  end
  ```
  Define the data processing that goes between the sample capture and the data transfers to the PC host.

-
  ```
    ENCDRATE(720, ip0, ip1, 0, 14750, ANALOG, 1000, pEnc)
  ```
  Apply the ENCDRATE processing command to analyze the time relationship between encoder pulses and samples captured on the sampler clock. The encoder hardware has 720 pulses per rotation, but samples are desired at 1000 equally-spaced angular positions per rotation. The `ip0` and `ip1` channels from the input sampling configuration observe the encoder pulses. Voltage levels on the logic signals are approximately 0V for logic low, approximately 4.5 volts for logic high, equivalent to digitized levels 0 and 14750 on the 10-volt digitizer. The input channels monitoring the rotation are *ANALOG*. The timing information is delivered through pipe *pEnc*.

-
  ```
    ADJRESAMP(ipipe(2,3,4,58), 4, pEnc, FAST, pOutMux)
  ```
  Apply the ADJRESAMP processing command to resample the data from the four signals on input channels `ip(2,3,4,5)` at the 1000 points per rotation established previously. The timing information used to locate the resampling positions is taken from pipe `pEnc`. The resulting data are placed into pipe `pOutMux`.

-
  ```
  COPY(pOutMux,$BinOut)
  COPY(ipipe(2,3,6,7), CpOut2)
  ```
  Transfer the resampled data from pipe `pOutMux` through pre-defined communication pipe `$BinOut` to the host. Transfer time-based samples directly from the four channels `ipipe(2,3,6,7)` through the extra communication pipe `CpOut2`. Separate streams are used so that the distinct data rates through the two channels do not interact and cause an unintended backlog of inaccessible samples.

The only part missing from this story is the command to make it all start to happen – the `start` command. Typically, you will not tell the xDAP to start its processing until your machine under test is operating and ready to be measured.

3. Applications

This section describes some typical applications using the *Rotating Machinery Analysis Module.* Keep in mind that these are *pre-processing* applications, executed by embedded xDAP on-the-fly, so that any data selection and transformation operations are already completed when your application program receives the the data. You can record both transformed and raw data if you wish, with suitable care.

Each application described in this section has a corresponding DAPstudio configuration file. You can begin configuring new applications by connecting real signals into these applications and then incrementally adapting them. Or if you prefer to work in a different environment, you can use the DAPstudio *Save DAPL* option to export the configuration in a text script file form, which can then be modified or used directly with any software environment.

Using features of the *Rotating Machinery Analysis* package does not take away any of the other processing options you might wish to use in the embedded xDAP environment. For example, if you wish to collect data from multiple blocks and average them to reduce measurement noise, you can certainly add this processing. This section, however, will concentrate on core features related to rotating machinery and timing.

Here are some of the added capabilities you get from the *Rotating Machinery Analysis Module.*

1. Obtain data sets equivalent to the ones you would get using encoder-driven sampling directly. (That is... you don't give up anything just because encoders don't drive sampling directly.)
2. Obtain samples at different angular resolution than provided by the hardware encoder.
3. Retain both time-based and encoder-based data blocks from an experiment.
4. Locate encoder-based data sets from events detected in high-resolution sampling data.
5. Locate time-based data sets from events detected in encoder-aligned data.
6. Extract profiles of instantaneous rotation speed.

Application 0: Test effects of resampling (simulation)

If you are ready to start with real applications, you can skip this part about simulation and validation. But this is something ready to run “out of the box” before you have anything real devices wired up and attached to your xDAP system.

Resampling seems almost magical. How is it possible to reconstruct a waveform to full accuracy with only 5 samples per waveform cycle? Even if the internal details are not clear, the results should be demonstrable.

In this self-testing application, you compare resampled values to ideal mathematical values, to demonstrate that the resampling calculations are reliable. While this is only a mathematical exercise, it is a self-contained test that you can run before any physical devices are set up.

It takes relatively elaborate mathematics to predict the time duration for each pulse from an encoded device, for the general case of smooth but unknown rotational speed variations. Since this simulation is somewhat artificial anyway, it considers only a special case of steady rotation. For this case, we can predict exactly how many sample intervals will span each encoder pulse interval observed.

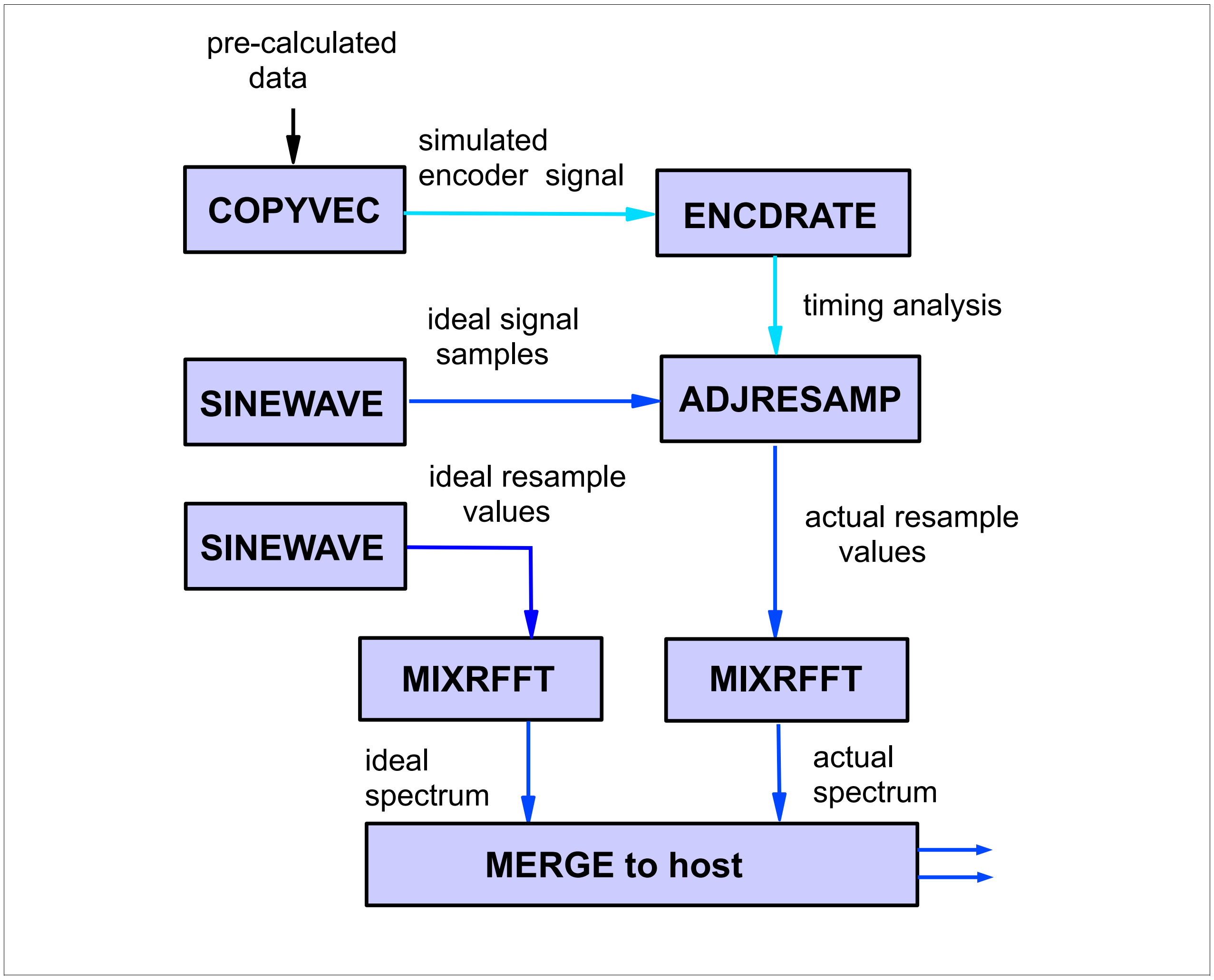
This exercise assumes a device with 360 sample locations per rotation, but 256 sample locations are desired for an FFT frequency analysis. Let's set up the sampling clock such that there are 20 samples per encoder pulses at the desired new rate. There will be 20 x 256/360 = 14.222222 samples observed per encoder pulse at the original pulse rate, on the average. But since the encoder is digital, what you will actually get is sometimes a count of 14 samples during a pulse, sometimes a count of 15, resulting in the appropriate 14.222222 samples average. We will simulate this by generating sequences of 14 or 15 samples between pulses, in the appropriate 7-to-2 ratio.

Suppose that we have a sinewave signal with 4 cycles per rotation spanning each block of 360 encoder pulses. After resampling, the original groups of 90 pulses per waveform cycle will be reduced to 64 locations per waveform cycle at the new resolution. We can generate sine wave data having these periods analytically, as a baseline for evaluating the quality of the resampled signal values.

Solution: Use a `COPYVEC` task to generate the digital logic bit pattern simulating an encoder pulse sequence as observed on a digital port. Apply this to an an `ENCDRATE` command to apply a timing analysis for the resampling. Use one `SINEWAVE` command on the xDAP to generate ideal, band-limited sampled data prior to resampling, and another `SINEWAVE` command to calculate the theoretical ideal results for the wave at a modified frequency, as it should appear after resampling. Use an `ADJRESAMP` command to combine the timing information and the original sampled data stream in an attempt to construct a data stream that matches the theoretically ideal resampled data stream.

There are some extra padding samples placed into the input data sequence. These align the phase of the data stream and the phase of the ideal signal so they start aligned when the encoder analysis synchronizes to the measured data. In general this is hard to predict, but we can do it for this configuration with its special case.

One way to determine whether there are systematic, repeating errors such as harmonic distortion is to apply an FFT analysis. Since the input is a pure (sampled) sine wave, the FFT of the resampled data should also be a pure sine wave. Look for significant magnitude attenuation, significant harmonics from distortion, and phase displacement.



Result:

FFT values not aligned to the original input sampling are artifacts of the fixed-point numbers and the resampling. However, the largest of these distortions is 0.27, corresponding in magnitude to approximately ½ of one encoder count. The resampling did not introduce significant nonlinear harmonic distortion, and the noise levels are in line with what you would expect from numerical rounding alone.

Comparing the fourth location in the FFT for the mathematically pure data and the resampled data stream at the frequency of the input signal, the FFT terms found there have the following polar form.

```
Term 4  =   1.414197E+04 | -1.569986E+00
Ideal   =   1.414216E+04 | -1.570796E+00
```

This is a close match, but not perfect, and you can see the effects of resampling here. The magnitude `1.414197E+04` is off by about `0.019` units, which corresponds to an error of about ½ converter tick. The phase match here is within `0.0002` radians, and since this is at a frequency 4 times the rotation rate, the overall rotation estimate is off by approximately `0.00005` radians.

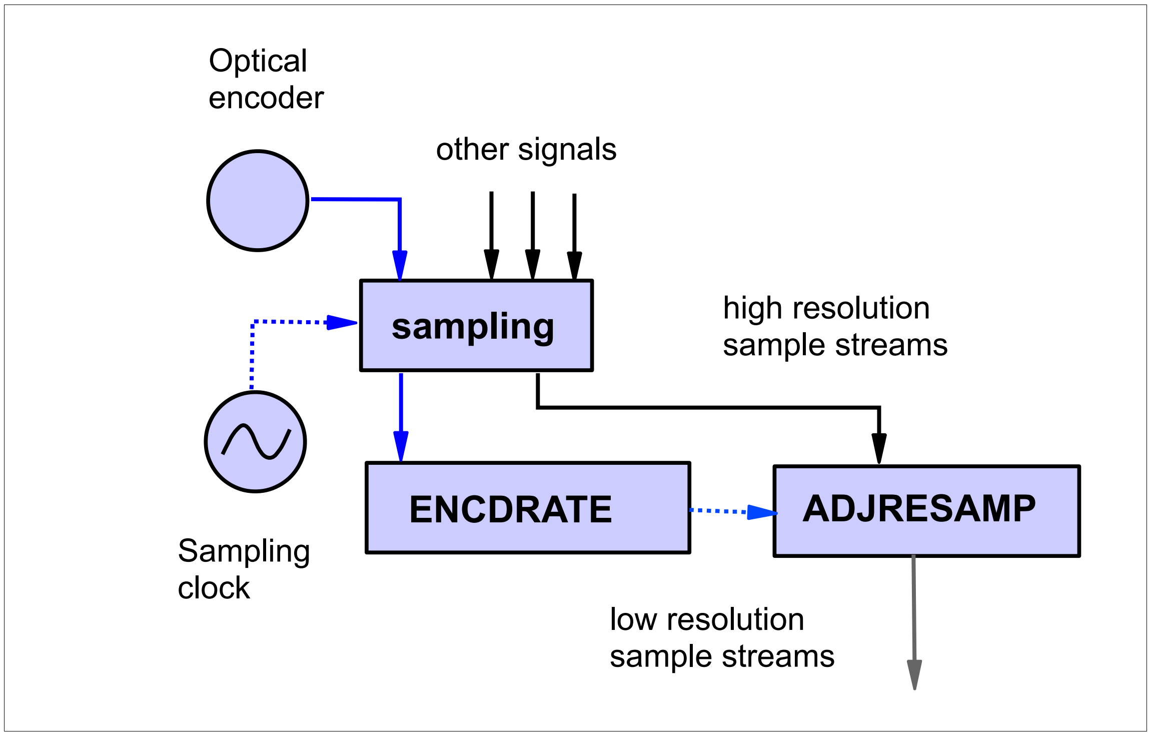
DAPL configuration:

```
VECTOR  vPulses  word = (
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1 )
PIPE    pTiming  double
PIPES   pRepPulses word
PIPES   pOldSine, pFastSine, pNewSine, pIdealSine
PIPES   pIdMag float, pIdAng float, pNewMag float, pNewAng float

// Values that we can predict will be skipped as encoder analysis
// synchronizes at the end of the first complete encoder pulse
// it sees. This is easier than adjusting phase on the ideal wave.
FILL    pFastSine  0,0,0,0,0,0,0,0,0,0,0,0,0,0
FILL    pFastSine  0,0,0,0,0,0,0,0,0,0,0,0

PDEFINE  simulate
  // Generate the continuous encoder pulse stream
  COPYVEC(vPulses, pRepPulses)
  // Analyze pulse stream to generate resample timing data
  ENCDRATE(360, pRepPulses, $0001, DIGITAL, 256, pTiming)
  // Generate mathematically ideal waveforms
  SINEWAVE( 20000, 1280, pFastSine )
  SINEWAVE( 20000, 64, pIdealSine )
  // Resample from the original waveform
  ADJRESAMP(pFastSine, 1, pTiming, ACCURATE, pNewSine)
  // Post-resampling analysis of signal differences.
  // Send the four output pipes of the FFTs to the PC host.
  MIXRFFT(256, FORWARD, pNewSine, HALF, POLAR, pNewMag, pNewAng )
  MIXRFFT(256, FORWARD, pIdealSine, HALF, POLAR, pIdMag, pIdAng )
END
```

Application 1: Downsampling rates

This application is basically the same as Application 0, except that it is the real thing, using actual encoder input measurement channel signals. For making some measurements at a very high speed, you might still want to collect samples "on a per revolution basis" but you don't need all of the samples you would get at every encoder timing pulse. In this example, the encoder has 512 pulses per revolution, but you only wish to record 10 measurements in each channel per rotation. Besides the timing channel, there are four signal channels where you want to capture measurements. All channels are sampled simultaneously.

Solution: Use an `ENCDRATE` command to analyze the incoming timing sequence from analog encoder pulses. The input levels are set so the nominal high voltage level is 5 volts and the nominal low voltage level is 0 volts on the encoder input channel. There is no reference timing pulse. Use an `ADJRESAMP` command to combine the timing information and the original sampled data streams to construct new data streams, synchronized to the rotations, but with the data rates reduced to the desired per-rotation rate.



Hazards: Be careful about signal bandwidths. Downsampling in this way has a potential, in general, for introducing aliasing effects into your measurements. If this happens, there is no way to remove those effects from your data sets later. You will be safe if you are measuring signals that simply cannot change fast, hence potentially dangerous frequencies are not present.

DAPL configuration:

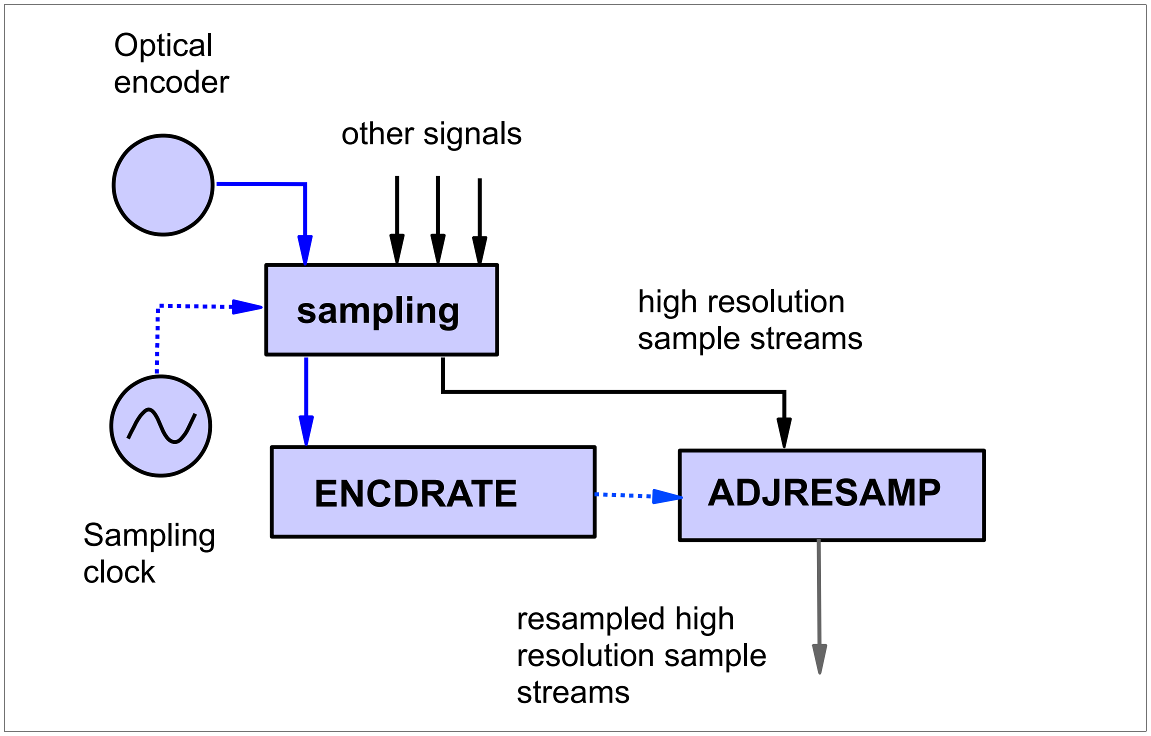
```
idefine MslInput
   channels 5
   set IP0 d0      // Encoder channel
   set IP1 d2      // Four analog channels
   set IP2 d4
   set IP3 d6
   set IP4 d8
   scan  10.000    // 100000 samp/sec, each channel
end

pipes pResampled word
pipes pTiming double

pdefine downsample
  // Analyze timing from encoder signal
  ENCDRATE(512, ip0, 0, 16000, ANALOG, 10, pTiming)
  // Resample from the original waveform
  ADJRESAMP(ip(1,2,3,4), 4, pTiming, FAST, pResampled)
  // Merge pipes to PC.
  merge(pResampled, $BinOut)
end
```

Application 2: Upsampling rates

This application is similar to Application 1, except that instead of reducing the number of sampling positions, it increases the number. This can be useful when rotation is slow – timing pulses from the encoder signal do not arrive very often, and would not provide a sufficient number of samples if used directly. But since the resampling is a mathematical operation that uses high-rate, time-based samples, it is perfectly valid to generated multiple sample values per hardware encoder pulse.

In this example, a 360 position encoder is used to study an engine rotating at 600 RPM. That is only 10 rotations per second, so the time between hardware samples is about 300 microseconds. To observe some variables, this is not fast enough. Resampling increases the apparent sample rate from 360 to 3600 samples per rotation.

Solution: Use an `ENCDRATE` command to analyze the incoming timing sequence from analog encoder pulses. The input levels are set so the nominal high voltage level is 5 volts and the nominal low voltage level is 0 volts on the encoder input channel. There is no reference timing pulse. Use an `ADJRESAMP` command to combine the timing information and the original sampled data streams to construct new data streams, synchronized to the rotations, but with the data rates increased to the desired per-rotation rate. This configuration is virtually the same as the one in Application 1, except for the differences in the resampling rate parameters.


Optical encoder
other signals
sampling
high resolution sample streams
Sampling clock
ENCDRATE
ADJRESAMP
resampled high resolution sample streams

Hazards: Upsampling cannot generate information that is not already present in your original signals. If the original high-resolution time-based sampling could not capture certain information, that information will continue to be missing in an upsampled data stream – in effect, the resampling becomes an elaborate curve interpolator that fills gaps smoothly where the signal originally was not smooth. If upsampling computes many sample values per encoder pulse, and if the pulse rates are very fast, this could overload the xDAP processor, and unprocessed data will backlog in memory.

DAPL configuration:

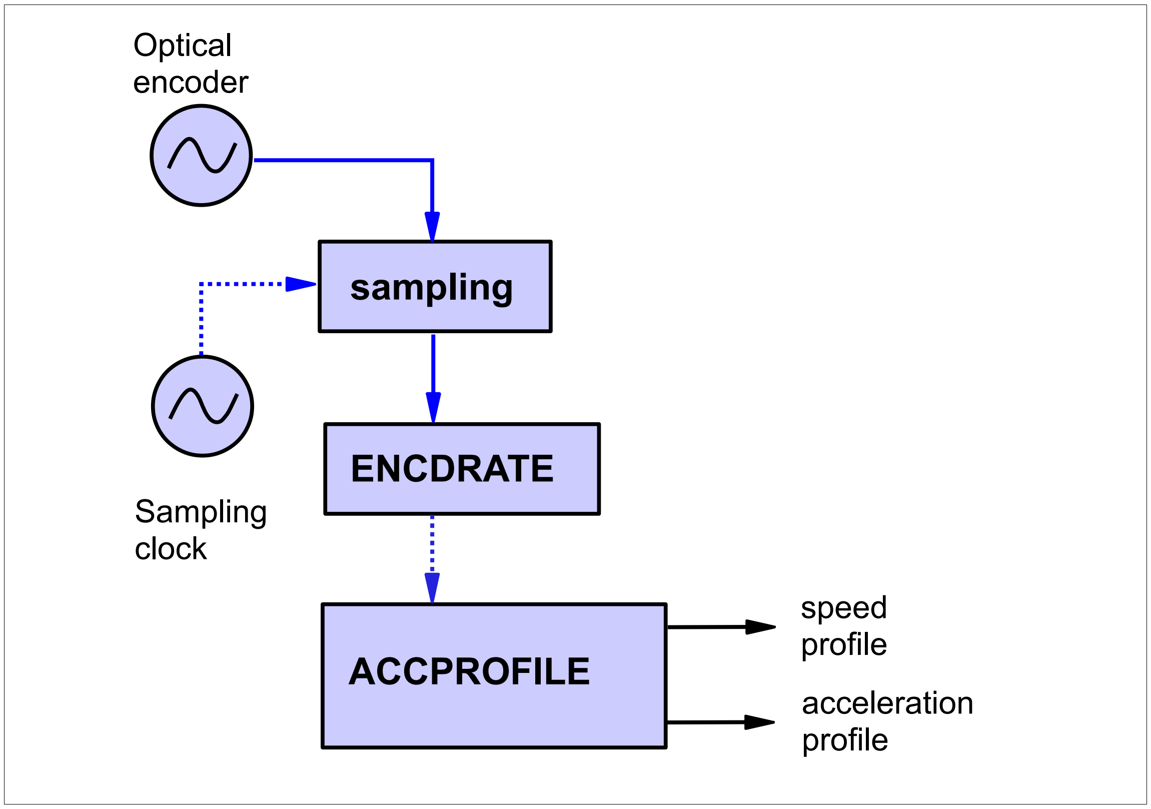
```
idefine MslInput
   channels 5
   set IP0 d0      // Encoder channel
   set IP1 d2      // Four analog channels
   set IP2 d4
   set IP3 d6
   set IP4 d8
   scan  20.000    // 50000 samp/sec, each channel
end

pipes pResampled word
pipes pTiming double

pdefine upsample
  // Analyze timing from encoder signal
  ENCDRATE(360, ip0, 0, 16000, ANALOG, 3600, pTiming)
  // Resample from the original waveform
  ADJRESAMP(ip(1,2,3,4), 4, pTiming, FAST, pResampled)
  // Merge pipes to PC.
  merge(pResampled, $BinOut)
end
```

Application 3: Speed profile

To perform a timing analysis, the `ENCDRATE` command must determine which time-based high-resolution samples correspond to which encoder pulse positions. Since the time-based samples correspond to equal time intervals, the information is equivalent to a correspondence between rotation angle and equal increments of time. Under assumptions that the profile is sufficiently smooth, there is enough information to generate a profile of instantaneous velocity at each angle of rotation. From this, a derivative estimator can generate the corresponding profiles of instantaneous acceleration.

In this example, the effects of a switching load are studied on a large synchronous generator. The speed of the generator will be regulated somewhat by the interconnected power grid and the dynamics of the load, but strain due to switching transients or resonant oscillations can be indicated by local acceleration disturbances. The generator operates at a nominal 600 RPM, and is monitored with a 1024-point optical encoder, for a nominal rate of 10240 encoder points per second.

Solution: To get a good time resolution, the sampling rate of the xDAP is set to 51200 samples per second, a factor of 5 higher than the encoder pulse rate. The encoder signal is analyzed by an `ENCDRATE` command in a normal way to derive timing information at resampling locations. The resampling produces data for the same 1024 locations per rotation that correspond to hardware encoder pulse positions. An `ACCPROFILE` processing command extracts timing information to reconstruct the speed profiles — no data streams are resampled for this processing, but you are not excluded from adding additional processes to the configuration if you wish.



Hazards: *You cannot depend on the absolute accuracy of the profile values.* The appendix section at the end of this manual, and the descriptions of individual commands, discuss how instantaneous velocity and the pulse-by-pulse digital chatter are “aliased.” If the digital bit-chatter is allowed to leak through, it dominates the profile data sets, which are then little better than pure noise; but removing the chatter also removes some of the bandwidth necessary to track rapid speed transitions.

An acceleration profile is obtained by differentiating the velocity profile. This is only a mathematical approximation, since the functions being differentiated are unknown. In effect, the speed samples are used to construct a smoothed approximation curve, and the slope of that curve becomes the derivative estimate. The derivative estimates cannot know which variations are caused by residual noise and which are physically meaningful. Derivatives will amplify any noise passing through into the velocity profile, and the numerical estimates will behave very much like true derivatives would. Filtering is required to prevent noise effects from dominating, but this filtering unavoidably impairs the ability of the profile estimates to track very fast but physically real effects such as impulsive acceleration spikes.

DAPL configuration:

```
 idefine MslInput
   channels 1
   set IP0 d0     // encoder signal
   scan 19.532    // approximately 51200 samples/second
 end

pipes pTiming double
pipes pVel float
pipes pAcc float

pdefine  speedtrak
  // Analyze timing from encoder voltage signal
  ENCDRATE(1024, ip0, 0,16000,ANALOG, 1024, pTiming)
  // Construct velocity and acceleration profiles
  ACCPROFILE(PTiming, 51200.0, 1024, pVel, pAcc)
end
```

Application 4: Retaining time-based and rotation-based data

The number of samples in data streams will depend on the sampling rates and the rotation speeds. It will be almost impossible to tell from the data which time-based samples correspond to which angle-based samples. That is why most applications that need to use data at both rates will involve software triggering. The software triggering identifies specific positions in data streams, and these positions can be used to determine the correspondence.

Most applications that care about positions will also connect a “reference timing” signal, also called an “index” or “top dead center” (TDC) signal, to establishes an angular position reference.

Since there is no fixed relationship between the sampling and rotation rates, sufficient data block sizes must be chosen so that every desired time interval is adequately spanned in both data sets. If you need to cover full rotations, allow plenty of time. If you need to cover a certain time interval, collect sufficient rotation data.

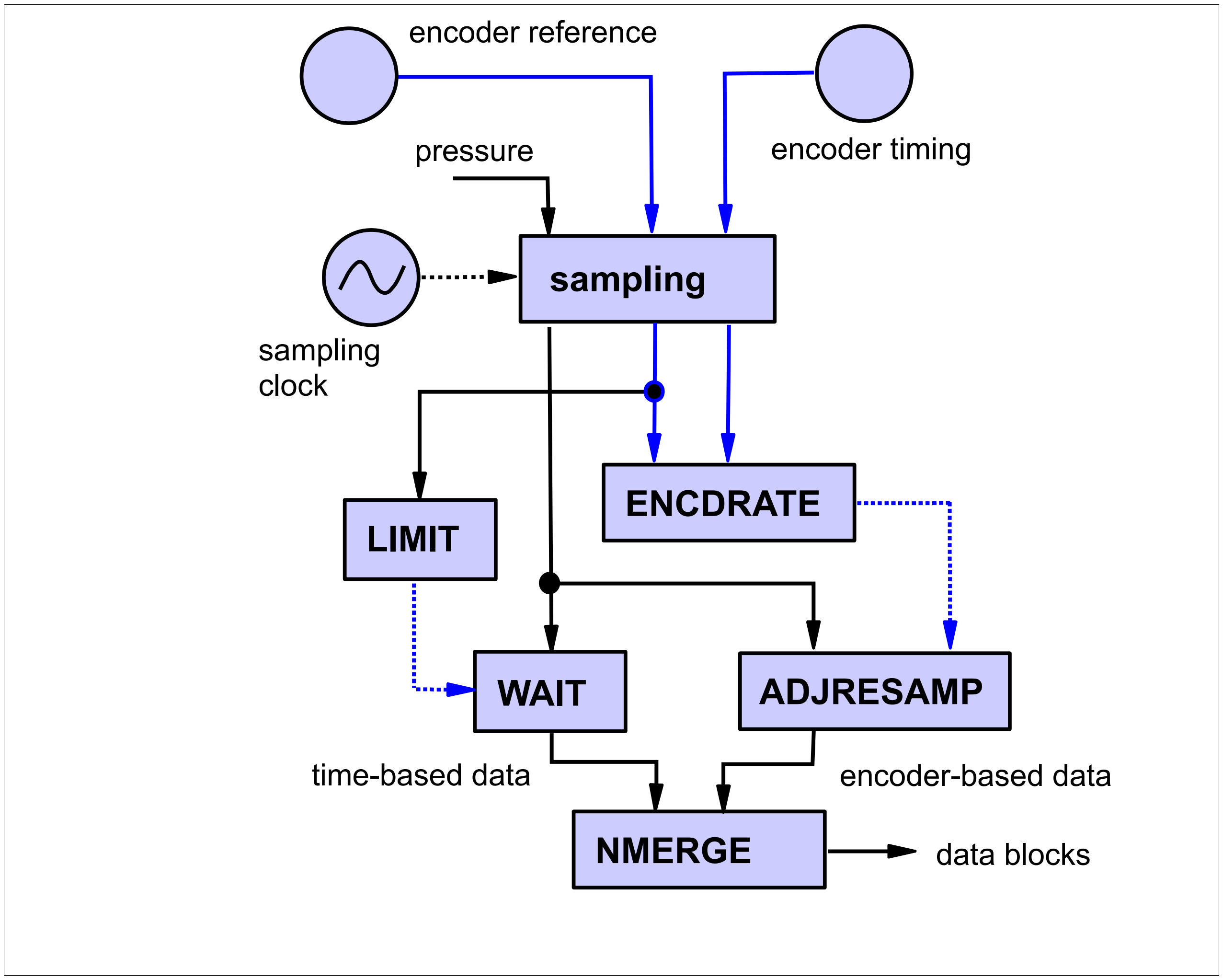
In this application, piston pressure is monitored. Pressure can change rapidly, so we want to preserve high-rate samples, but we also want to record pressure as a function of rotation angle, so that the two data sets can be related to each other. The resampled measurements start at a TDC reference position. Data capture continues long enough to cover the desired time span, allowing for speed variations. Data are selected once at encoder-aligned positions, using resampling, and then again in the ordinary time-based data using software triggering.

Solution: An `ENCDRATE` command observes the encoder timing and reference signals. The results of this analysis are used by an an `ADJRESAMP` command to build a resampled data set with pressure values corresponding to the encoder's hardware timing locations. This example is configured for a rotation rate around 1000 RPM, with an encoder having 720 pulses per rotation. Sampling at a rate of 120000 samples per second will yield approximately 10 samples per encoder timing pulse.

Given a TDC reference signal, an `ENCDRATE` command will locate its first position aligned to a TDC pulse. (This has the effect of bypassing an indeterminate initial number of samples, which are not forgotten, but no resampling positions are located there.) Starting from this “sync” location, each group of 720 samples will represent one rotation. Though it is possible to locate these blocks by triggering, it is easier just to “count samples” and take groups of 720 samples from the resampled data stream.

Software triggering analysis, applied to the raw, high-resolution time samples of the TDC signal, can be performed by a `LIMIT` command to detect TDC pulse edges, in parallel with other processing. There is one event per TDC pulse. A `WAIT` command then responds to the software triggering events by selecting the corresponding time-based samples from the high-resolution stream of pressure measurements. Retaining data for ½ rotation will require a block of about 3600 samples.

The number of samples in the time-based and encoder-based blocks is different. To avoid transfer rate problems, these values are first collected into a pipe, and then transferred to the host using an `NMERGE` command. (See the “general example” at the end of Chapter 2 for the slightly more complicated but more robust approach that sends data at differing rates through separate communication channels.)



Hazards: Always be careful when transmitting data to a PC host at two different rates. It is easier to get into trouble with this than you might suppose – the default data transfer method configured by DAPstudio is vulnerable to this. DAPstudio does not know when you collect data at multiple rates, and assumes constant and equal flow rates of data in all channels. But suppose that the time-based sample block is 10 times as large as the encoder-based block. Under the default strategy, with samples delivered one-for-one from each source, only 10% of the time-based samples have been transferred at the time that the rotation-based block completes, leaving 90% of the time-based data backlogged in memory and waiting for more angle-based samples. Repeat this lots of times and the data backlog can grow into a major problem.

There are two solutions that you can use to avoid these rate problems.

1. If you know the number of samples in each block, force the transfers into these sizes to keep the data rates consistent.
2. Move the two kinds of data blocks to the host through different communication channels, so that the streams are processed independently at their own separate rates.

This example application uses the first approach.

Data sets collected by triggering cannot overlap When you use software triggering to retain some data, those values are removed from the original data pipe and are not available for another triggering event. If you need to be sure that your time-based data completely span a rotation, you will need to retain data that might span more than one rotation – and in that case, data blocks captured once per rotation with resampling will not correspond one-for-one with time-based blocks.

DAPL configuration:

```
idefine MslInput
  channels   3
  set IP0    d0
  set IP1    d2
  set IP2    d4
  scan   8.334    // 120000 samples/second
end

pipes pPressEnc word
pipes pPressRaw word
pipes pTiming double
pipes pTransfer word
triggers tTDCevent mode = normal

pdefine twoways
  // Analyze encoder timing and reference signals
  ENCDRATE(720, ip0, ip1,  0, 16000, ANALOG, 720, pTiming)
  // Resample the pressure measurements at timing pulses
  ADJRESAMP(ip2, 1, pTiming, FAST, pPressEnc)
  // Detect TDC reference pulse in time-based data
  LIMIT(ip1,  INSIDE,14000,32767,  tTDCevent, INSIDE,12000,32767)
  // Collect about 1/2 rotation of time-based pressure data
  WAIT(ip2, tTDCevent, 0,3600, pPressRaw)
  // Deliver data at balanced rates to host system
  NMERGE( 720, pPressEnc, 3600, pPressRaw, pTransfer)
  // Transfer to PC host
  merge(pTransfer, $BinOut)
end
```

Application 5: Selecting rotational data from time-events

Time events can arise from a variety of activities.

- Manually triggered activity from a GUI control interface
- Events that are too fast to detect in resampled data streams
- Events related to external timing control

This application studies how machine performance varies with operating temperature. We don't know exactly when the system will be at a temperatures that we want to measure, but we do know that the temperature cannot change very fast, so it is sufficient to select data every few seconds.

You can't tell how much time elapses within a resampled data stream – this varies with rotation speed. But it is relatively easy to determine regular time intervals in time-based data sets, simply by counting samples. The TGEN processing command will do this for you, generating trigger events corresponding to regular time intervals that you specify.

Time intervals could end with the rotations at any angle, but the retained data blocks should start at a "top dead center" timing reference pulse, and cover complete rotations. This requires a two-step process:

1. Convert the timing-event location to an angle-based location relative to encoder timing pulses.
2. Determine within the encoder-based data stream the location where a rotation starts.

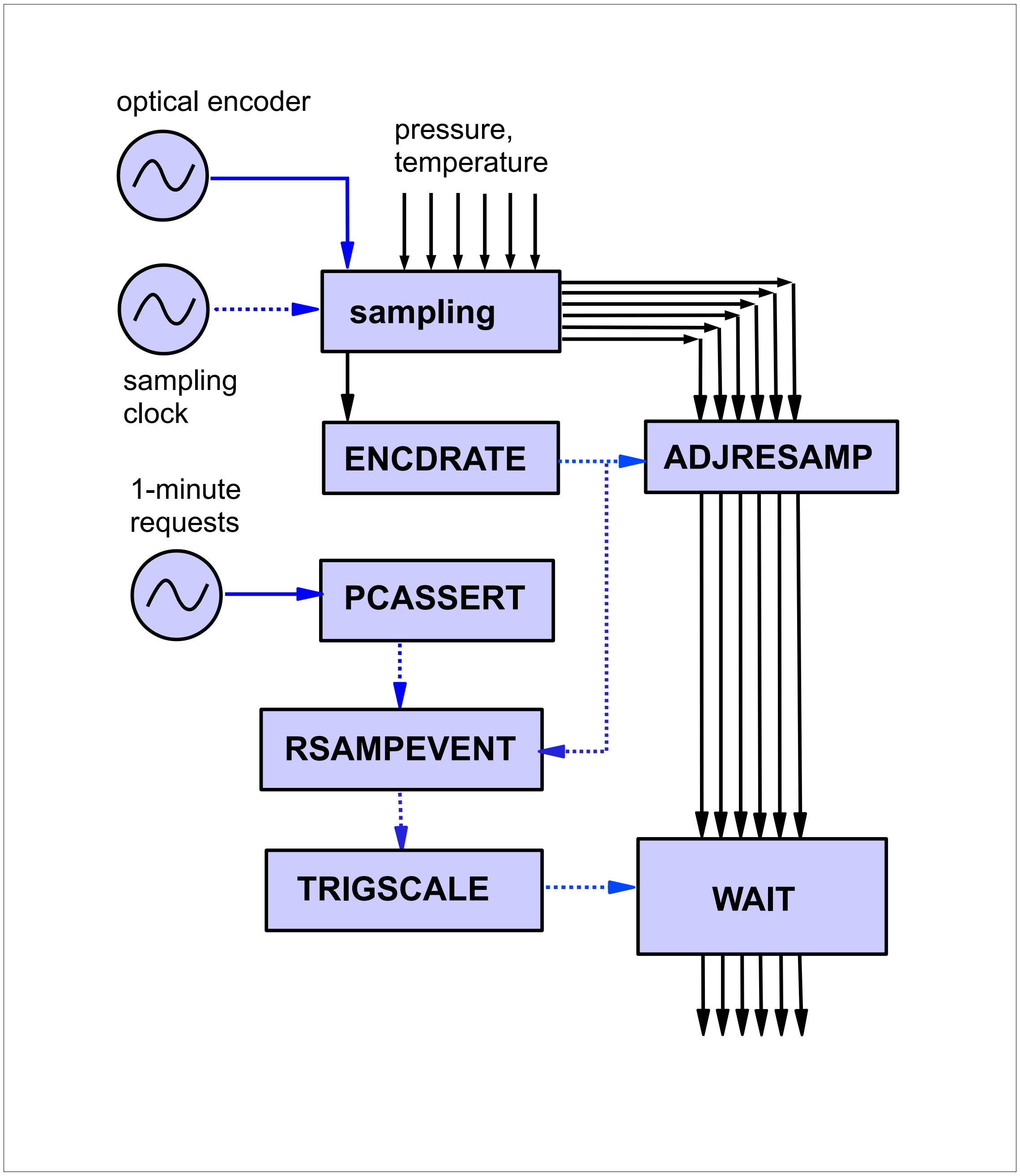
Solution: Measurements are taken using 12 sensors, recording 32 positions per rotation, and retaining 10 rotations in a cluster. The encoder has 256 timing pulses per rotation. You determine that 40000 samples per second on each channel provides sufficient time resolution for the resampling. Use an `ENCDRATE` command to analyze the timing pulses and position reference pulses from the encoder. Apply an `ADJRESAMP` command, in the usual way, so that the desired 32 positions of equal rotation angle are located. The total number of samples to retain is

```
 12 sensors  x  32 positions  x  10 rotations  =  3840 samples
```

This process is repeated once every 10 seconds. During these 10 seconds, the number of time-based samples that will pass per channel is

```
 40000 samples/sec * 10 seconds  = 400000
```

Configure a `TGEN` processing command to generate trigger events at 400000 time-sample intervals. Take these events (at 10 second intervals), and convert them to the corresponding rotational positions using a `TRIGMAP` command. Use a `TRIGSCALE` command to realign these to the beginning of 32-sample groups at a TDC reference pulse. Use a software triggering `WAIT` command to select the 3840 retained samples at each event.


optical encoder
pressure, temperature
sampling
sampling clock
ENCDRATE
ADJRESAMP
1-minute requests
PCASSERT
RSAMPEVENT
TRIGSCALE
WAIT

DAPL configuration:

```
idefine MslInput
  channels   14
  set IP0    d0   // Timing
  set IP1    d2   // Reference
  set IP2    d4   // Six sensor channels
  set IP3    d6
  set IP4    d8
  set IP5    d10
  set IP6    d12
  set IP7    d14
  set IP8    d1   // Remaining six sensor channels
  set IP9    d3
  set IP10   d5
  set IP11   d7
  set IP12   d9
  set IP13   d11
  scan   25.0     // 40000 samples/second each channel
end

pipes pTiming double
pipes pPressEnc word
pipes pRSSensors word
pipes pTransfer word
triggers tTenSec  mode=normal
triggers tRotational   mode=normal
triggers tKeep   mode=normal

pdefine twoways
  // Analyze encoder timing and reference signals
  ENCDRATE(256, ip0, ip1,  0, 16000, ANALOG, 32, pTiming)
  // Resample the 12 sensor measurements 32 times per rotation
  ADJRESAMP(ip(3..13), 12, pTiming, FAST, pRSSensors)
  // Generate 10-second timing intervals
  TGEN(400000, tTenSec)
  // Map these interval events into rotational positions
  TRIGMAP(tTenSec, pTiming, TOEXTERNAL, tRotational )
  // Adjust to start of 3840-term blocks covering 12 channels
  TRIGSCALE(tRotational, 0, 3840, 32,   tKeep)
  // Retain desired 3840 sample blocks
  WAIT(pRSSensors,tKeep, 0,3840, pTransfer)

  // Transfer to PC host
  merge(pTransfer, $BinOut)
end
```

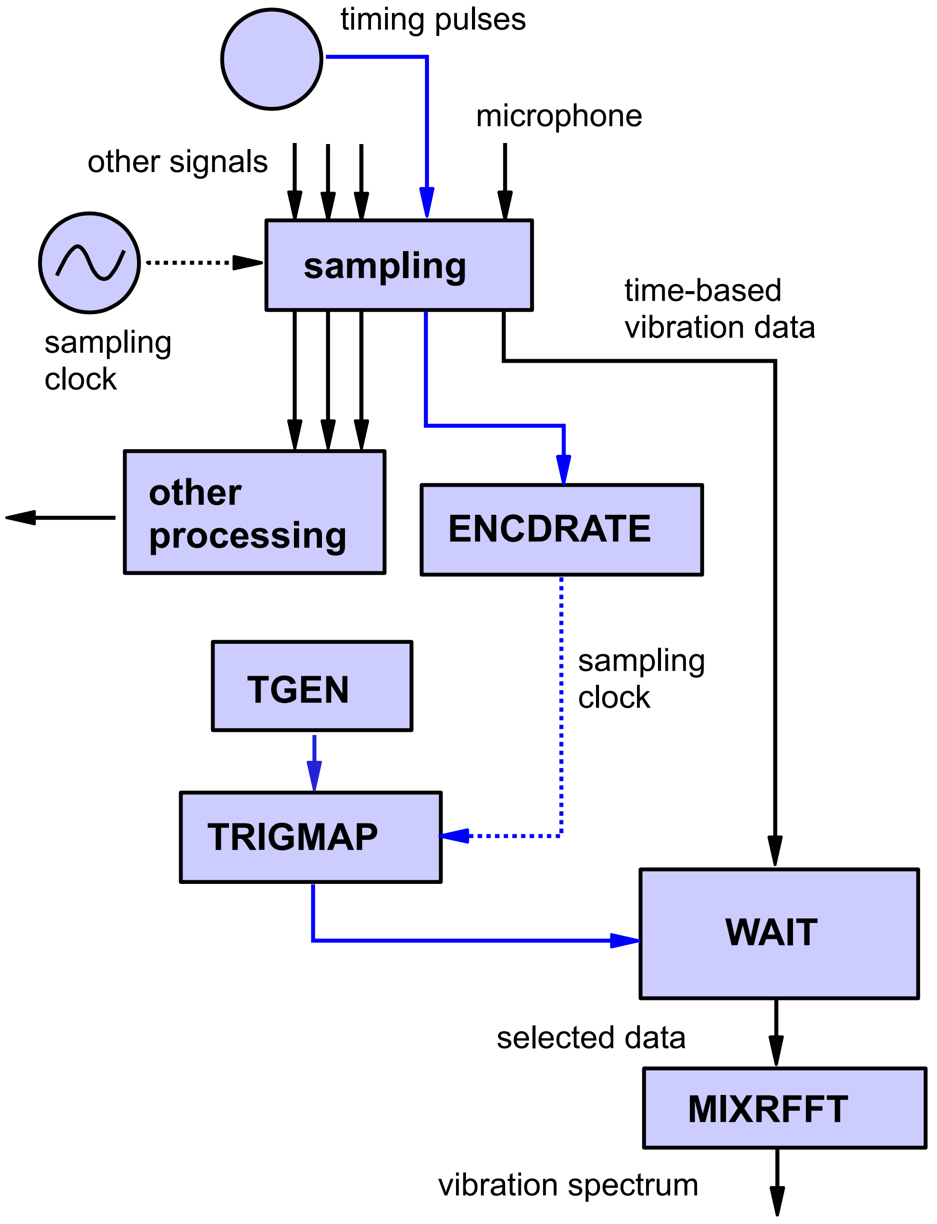
Application 6: Selecting time-based data from rotation events

Time-based data can be useful to resolve high-frequency or short-duration features that would be missed in resampled data sets. It can also observe elapsed time, which is not possible in data sets slaved to the rotation speed.

In this application example, the goal is to select blocks of data for a frequency analysis once every 10000 rotations. There is no TDC reference pulse, but the encoder resolution is known to be 360 pulses per rotation. A microphone capture vibrations for analysis. It is possible to do this application with the time-based data alone, triggering on each pulse so that it can be counted, counting the pulses until there are 3600000 of them, then triggering again to collect the desired sample blocks. This is very tricky, however.

It is much simpler to do this in terms of rotations. We know that there will be 3600000 encoder pulses between events where blocks are analyzed, so the locations of these events are predetermined. These locations are then mapped to the corresponding time-based sample locations where data should be captured.

Solution: Measurements are taken using 3 regular sensors plus one more special microphone sensor. The encoder timing pulses are analyzed by an `ENCDRATE` command that retains 360 samples per rotation, which is a configuration selected to facilitate other analysis. A `TGEN` processing command generates artificial trigger events once every 3600000 encoder pulses. The timing information from the `ENCDRATE` command and the trigger events from `TGEN` are routed to a `TRIGMAP` command, which operates in the *TOINTERNAL* mode to map events from an encoder pulse location to a time-based sample location. A fixed 2048-sample block is captured at that location, using software triggering and a `WAIT` processing command. The blocks are then passed to an FFT for analysis, and returned to the host system via a separate data communication channel.


timing pulses
microphone
other signals
sampling
sampling clock
time-based vibration data
other processing
ENCDRATE
sampling clock
TGEN
TRIGMAP
WAIT
selected data
MIXRFFT
vibration spectrum

DAPL configuration:

```
idefine MslInput
  channels   5
  set IP0    d0   // Timing
  set IP1    d2   // Three other channels
  set IP2    d4
  set IP3    d6
  set IP4    d8   // Microphone, for vibration analysis
  scan   20.0     // 50000 samples/second each channel
end

pipes pTiming    double
pipes pSelected  word
pipes pRreal     float, pImag float
pipes pSelected  word

triggers tRotations  mode=normal
triggers tSelect  mode=normal

pdefine getspectra
  // Analyze encoder timing
  ENCDRATE(360, ip0, 0, 16000, ANALOG, 360, pTiming)
  // Omit other resampling and related processing...

  // Generate events every 10000 rotations (3600000 samples)
  TGEN(3600000, tRotations)
  // Map these interval events into time-based positions
  TRIGMAP(tRotations, pTiming, TOINTERNAL, tSelect )
  // Retain 2048 sample blocks of microphone data
  WAIT(iPipe4,tSelect, 0,2048, pSelected)

  // Transfer spectrum to PC host via separate pipe
  MIXRFFT( 2048, FORWARD, HAMMING, pSelected, FULL, PARTS, \
      pReal, pImag )
  merge(pReal, pImag, Cp2Out)
end
```

4. Command Reference

This section provides details about the special processing commands provided by the ROTM command module.

- **ENCDRATE**
  Analyze timing pulse data received from a digital encoder, to determine a sequence of positions for resampling relative to rotary encoder timing pulses. This is the most common way of tracking the rotation angle for rotating machinery such as generators or engines. Sampling can take place with higher or lower resolution than that provided by the encoder.

- **ADJRESAMP**
  Perform the numerical calculations to obtain resampled signal values at selected positions in one or more channels, as determined previously by the ENCDRATE command.

- **TRIGMAP**
  Convert events in resampled data streams into locations in the original high-resolution time-based sample streams. Convert events in the original high-resolution time-based sample streams to corresponding positions in resampled data streams.

- **ACCPROFILE**
  Apply a supplementary rate analysis to obtain instantaneous speed and acceleration information from the timing information previously generated the ENCDRATE processing command. The values are with respect to time, but associated point-by-point with the resample locations determined relative to the encoder.

Commands are listed in alphabetical order, by name, in the pages that follow.

Processing Command

Module ROTM :: ACCPROFILE

Computes rotational velocity and acceleration profiles from encoder timing data.

Syntax

```
ACCPROFILE( RATEDATA, SAMPRATE, RESOLUTION, VELPIPE, ACCPIPE )
```

Parameters

RATEDATA
: Timing information generated by a previous `ENCDRATE` command
  `DOUBLE PIPE`

SAMPRATE
: Samples per second used to observe raw encoder signals
  `FLOAT CONSTANT`

RESOLUTION
: Encoded stream resolution in positions per rotation
  `WORD CONSTANT`

VELPIPE
: Velocity data output stream
  `FLOAT PIPE`

ACCPIPE
: Acceleration data output stream
  `FLOAT PIPE`

Description

The `ACCPROFILE` command analyzes a timing data stream, produced by an `ENCDRATE` processing command and delivered via the `RATEDATA` pipe. Rate information is extracted without actually performing any resampling — so no sampled signals are used directly. So that the `ACCPROFILE` command can interpret the rate data in terms of RPM units, you must provide the `SAMPRATE` parameter, which specifies the sampling rate (in samples per second) used to monitor the encoder signal, and the `RESOLUTION` parameter, which specifies the number of sample positions per rotation that the `ENCDRATE` command produces. The results of the analysis are two streams of data, each with one output term per resampling position indicated by the input timing data. The first output stream is the instantaneous speed in RPM units, placed into the `VELPIPE` output stream. The second output stream is the instantaneous acceleration in units of RPM change per second, placed into the `ACCPIPE` output stream. Provided that you used a "top dead center" reference pulse signal with your `ENCDRATE` processing, the output streams begin at a location that corresponds to the arrival of a reference pulse.

Both the velocity and the acceleration profiles are derived numerically. Raw encoder data arrive in the form of integer counts of samples, yet speed of rotation is not limited to integer values. For example, if the number of samples for a sequence of observed encoder pulses is `7, 7, 8, 7, 7, 8, ...` this is unlikely to mean that there are quantum jumps in speed. Smoothing the velocity profile is critical to estimating an acceleration profile — discrete jumps in velocity look like extreme impulses of acceleration, yielding very high noise levels in the trajectory estimates. The `ACCPROFILE` command makes a tradeoff between bandwidth (how closely it can track speed change) and how much noise will get through.

Some hazards to keep in mind:

- The rate at which data are produced will depend on the rotation speed, just as it would if your sampling was controlled directly by the encoder hardware.
- At very high speeds, there are lots of encoder pulses, but few samples per pulse, so the time resolution with respect to angular position is relatively poor.
- At very low speeds there are lots of samples per encoder tick for good angular resolution, but lots of things can be missed during the time intervals between encoder ticks. *Aliasing* can cause spurious effects that look like noise or velocity variations.
- The filtering necessary to produce reasonable acceleration profiles has the side effect of producing subtle alterations of the data stream, so if you attempt to integrate the acceleration data numerically you will not regenerate the velocity profile exactly.

Examples

```
ACCPROFILE(PProfile, 120000.0, 1000, pVel, pAcc)
```

A separate encoder timing analysis, performed by an `ENCDRATE` command, is received through pipe `PProfile`. The encoder signal was observed by sampling it 120000 times per second. The result of the `ENCDRATE` analysis is a data stream defining 1000 locations per rotation. The timing information is analyzed, and 1000 velocity profile values per rotation (in units of RPM) and 1000 acceleration profile values per rotation (in units of RPM change per second) are delivered to the `pVel` and `pAcc` pipes, respectively.

See Also

Processing Command

Module ROTM :: ADJRESAMP

Adjustable-rate resampling of a data stream.

Syntax

ADJRESAMP(PINMUX, NCHAN, RSRATE, [METHOD,] POUTMUX)

Parameters

PINMUX
: Input data stream with multiplexed sampled signals
: `WORD PIPE`

NCHAN
: Number of multiplexed channels in the PINMUX stream
: `WORD CONSTANT`

RSRATE
: Resampling position information from other processing
: `DOUBLE VARIABLE | DOUBLE PIPE`

METHOD
: Optional interpolation method keyword parameter
: `NEAREST | FAST | ACCURATE`

POUTMUX
: Output data with multiplexed resampled signals
: `WORD PIPE`

Description

An `ADJRESAMP` task performs an adjustable interpolation analysis, so that equally spaced samples captured at a sampling rate determined by a Data Acquisition Processor sampling clock are transformed into samples at a dynamically adjustable rate. The `ADJRESAMP` command uses an adjustable rate established by a separate `ENCDRATE` task. This allows the effective resampling rate to lock to speed variations of the external encoder timing signal.

The `ADJRESAMP` command receives multiplexed input data from the `PINMUX` pipe. The number of multiplexed channels, common to the input and output streams, is specified by the `NCHAN` parameter. Each group of `NCHAN` channels represents the data captured in one pass through the sampling channel list, with one value from each input channel. In all channels, the samples are assumed to occur at equal intervals of time. No assumptions are made about relative time or phase relationships between channels. Each channel is processed independently. The new sample values are placed into the `POUTMUX` pipe, in the same multiplexed sequences as received.

The timing source specified by the `RSRATE` parameter. For purposes of rotating machinery where maintaining alignment to a reference signal is critical, only the `DOUBLE PIPE` form of the `RSRATE` parameter should be used. (This command can accept a simple rate parameter via a shared variable, but this is not sufficient to maintain position alignment, and never recommended for rotating machine analysis.) The encodings received from the `RSRATE` output pipe includes rate, duration, and curvature information. This is enough information to lock to the phase of a continuously changing rate, somewhat in the manner of a phase-locked loop. No resampling is done until the associated resampling data arrive in the `RSRATE` pipe.

The positions of the input and output data streams are aligned so that the first sample in the output stream aligns in time exactly with the sample captured at the rising edge of an encoder timing signal. If the resampling analysis was previously done taking a TDC timing reference pulse into account, that first location also aligns to the TDC timing pulse. After that, because the input and output streams are at different sample rates, the samples in the input and output data streams do not line up one-for-one over time. Beware, this difference can lead to data backlog problems when attempting mixed data transfers, for example, sending both raw and resampled data through the same MERGE processing command supplied by a DAPstudio software configuration.

The optional `METHOD` parameter is one of the following keywords, in all capital letters but without any quote characters.

- *FAST* This option is appropriate for most applications, particularly those having some observable random noise, processing data at high rates, applying additional processing such as anti-alias filtering, or performing spectrum analysis. In theory it is slightly less accurate, making measurements appear slightly more "noisy". In practice, you will find it difficult to see any difference in your results.
- *ACCURATE* This is the option that you should use if you have very precise, very clean signals, and you want the best possible reconstruction of every sample, or the best possible representation of very high frequencies. This requires roughly twice the computation of the *FAST* method, using a two-stage interpolation. If you have a surplus of CPU capacity, there is no harm from using this method even if you don't otherwise need it.
- *NEAREST* This can be a useful option when the data are originally sampled at a much higher rate than the rate at which samples are retained — for example, when a machine is rotating very slowly there could be hundreds of samples observed for each optical encoder position. For such cases, there should be no harm from simply ignoring fractions of sample positions. The resampled value is taken as the value of the *nearest neighbor* position, as produced directly by sampling hardware, without additional computations.

If you omit the `METHOD` parameter, you will get the *FAST* option by default.

For the resampling methods that use interpolation, the interpolation formula is balanced, using an equal time-horizon of older and newer samples surrounding each evaluation position. This introduces some potential problems at the start and end of a data run, where not enough older or newer samples are available to perform interpolation calculations. At task start-up, artificial "padding" values force the correct alignment of the data streams, but as a result, the first few values you receive in the `POUTMUX` stream are not meaningful data. To avoid a "glitch" in your analysis, be careful to bypass any artificial initial samples (if present) before applying your data analysis. In a similar manner, to avoid missing relevant samples at the end of a run, sample some extra raw data at the end.

Some additional restrictions:

1. For best results, the frequencies present in every channel should be band-limited to below 40% of the Nyquist frequency (at least 5 samples captured per cycle). If your data sets have high-frequency noise or step edges, some noise artifacts will remain in the resampled data. The resampling is somewhat analogous to fitting a smooth curve to the data, and then evaluating along the curve. If the input data set is not smooth, the meaning of the curve fitting operation is unclear. This is not disastrous, with results similar to noise removal by lowpass filtering. One way to get a suitably oversampled data stream is to double your sampling rate, then decimate back to the sample rate you want later in your resampling.
2. Be careful about using `ADJRESAMP` command to lower the effective sample rate. Exactly like any other sampling based on an encoder, the `ADJRESAMP` command has no defense against aliasing. For rate shifts that are only a few percent, the aliasing can only affect the few highest frequencies, which should be empty anyway – if high frequencies are not present, there is nothing to alias and no damage done. If it is critical to avoid all aliasing, use electronic or digital filtering on your signal to establish appropriate band-limiting before resampling.
3. If subjected to abrupt steps (which would violate the bandwidth condition above), the interpolation process exhibits a transient behavior similar to a frequency-selective digital filter in a neighborhood of the discontinuity. Because the interpolation is balanced forwards and backwards in time, and applied to data in a buffer, the transient can appear to have a counter-intuitive *"before the disturbance arrived"* behavior.
4. There is limited local resolution for locating resampling points. For the *FAST* interpolation method, the timing uncertainly results in an apparent timing jitter of up to ±1/1000 sample interval at the initial high-resolution. Some effects can be observed on very high-level signals near the useful range limit at 40% of the Nyquist rate, appearing as seemingly random "noise"errors approximately -60dB below the signal level. These errors decrease at lower signal amplitudes and lower frequencies, and are typically masked by ordinary signal noise. For the *ACCURATE* interpolation method, the results of timing uncertainty are indistinguishable from integer rounding effects, approximately -100dB below the signal level.
5. Timing alignment depends on continuous sampling starting from sample 0. Burst mode processing will carry over stale samples from previous operation in data buffers, resulting in some invalid initial values each time the processing resumes. The time shift at the initial sample of each new burst is indeterminate. It is better to use software triggering, which will watch continuous streams of input data between detected events.

Examples

```
PIPE    rate_control  DOUBLE;

ADJRESAMP(IPipe(2,4,6,8), 4, rate_control, ACCURATE, pinterp)
```

The application needs highly precise data, aligned in rate and phase over a very long term to an encoder signal. The timing analysis is performed by a prior `ENCDRATE` task. The resulting timing information is received through the `rate_control` pipe. The four input channels from input pipe `Ipipe(2,4,6,8)` are resampled independently at the predetermined positions using the `ACCURATE` interpolation strategy. This expends extra computation to obtain the best possible time-resolution for locating each resampled point – the `FAST` method is preferable with less extreme requirements. The resampled data for the four channels are placed in the `pinterp` output pipe.

See Also

Processing Command

Module ROTM :: ENCDRATE

Compute resampling positions equally spaced in rotation angle using an encoder.

Syntax

ENCDRATE([NENCODE,] ANALOGIN, [TDCIN,] LOW, HIGH,
  SOURCE, NRESAMP, OUTPIPE)

ENCDRATE([NENCODE,] DIGITALIN, TMMASK, [REFMASK,]
  SOURCE, NRESAMP, OUTPIPE)

Parameters

NENCODE
  Predetermined number of encoder pulse cycles per rotation
  `WORD CONSTANT | LONG CONSTANT`

ANALOGIN
  Analog input timing signal
  `WORD PIPE`

TDCIN
  Optional analog input reference signal
  `WORD PIPE`

LOW
  Nominal digitized level for typical "inactive low" analog input
  `WORD CONSTANT`

HIGH
  Nominal digitized level for typical "active high" analog input
  `WORD CONSTANT`

DIGITALIN
  Digital input port value with timing and reference lines
  `WORD PIPE`

TMMASK
  Bit mask to isolate timing pulses from digital input
  `WORD CONSTANT`

REFMASK
  Bit mask to isolate reference position pulses from digital input
  `WORD CONSTANT`

SOURCE

Keyword, no quotes, indicating type of input signals

`ANALOG | DIGITAL`

NRESAMP

The number of resampling positions to record per rotation

`DOUBLE CONSTANT | LONG CONSTANT`

OUTPIPE

Pipe for transferring encoded rate information to resampling tasks

`DOUBLE PIPE`

Description

An `ENCDRATE` task observes pulses from an encoder mounted on a rotating machine, to determine the locations for resampling data. The pulses from the encoder are observed by sampling at short fixed time intervals, noting how many samples occur within each of the encoder timing pulse intervals. The inertia of the rotating machinery will not allow the speed to vary more than a few percent through the course of a single rotation, so the sequence of pulses can be interpreted as resulting from a smoothly-varying speed trajectory. The smoothed trajectory is then used to determine positions within the sampled data stream corresponding to equal steps in angular position. A separate analysis can use this information to extract resampled data values from the data originally captured on the high-resolution, fixed-interval sampling clock.

Ordinarily, when measuring rotating equipment, a choice must be made between one of two sampling strategies:

1. *Use a stable oscillator as a time-reference to clock the samples.* This gives good control of time resolution, but as the speed of rotation changes, the relationship between samples in the data set and angular positions becomes unclear.
2. *Use an optical encoder mounted on the shaft to produce digital pulses at equal increments of angular position.* The relationship between samples and angular positions is then known precisely, but the time interval spanned by the analysis becomes unclear, dependent on the unknown speed.

The `ENCDRATE` command provides a bridge between these two strategies. Measurements are captured on a high-resolution time-reference clock, so that the time location of each measurement is known. Then the `ENCDRATE` command observes the passage of encoder pulses, relating these to locations within the captured measurement data streams.

The `NENCODE` parameter is optional, and specifies how many timing pulses the digital encoder will generate during one shaft rotation. Timing pulses begin inactive low, and a pulse is deemed to arrive when a low-to-high transition is detected. If specified, the `NENCODE` value is assumed to characterize the *angular resolution* of the encoder, the number of timing pulses per one revolution. If the resolution is not specified, but a reference input signal is provided, the resolution property is determined automatically by counting the timing pulses through one revolution. If there is no specification and also no reference input signal, the resolution property is indeterminate and an arbitrary default of 360 (one timing pulse per degree rotation) is assumed.

The ENCDRATE command accepts one or two input signals from the encoder. If received as analog, they arrive through two separate channels; if received as digital, they arrive through two separate bit lines within one port-group of 16 bits. The timing signal provides high-resolution pulses at equal angles of rotation. The reference signal (sometimes called a *top dead center* signal) provides one pulse per rotation, aligned with one of the timing signal pulses.

Digital or analog signals can be used as inputs to the ENCDRATE command. The type of signals is specified by the SIGNALS keyword. This keyword must be specified in all capital letters, without quotation marks. You can't use a mix of timing signal types. The keyword must be one of the two following choices:

1. ANALOG If the signals are routed through analog channels, specify this keyword. The analog inputs are specified by parameters ANALOGIN and TDCIN. Since each analog channel uses all 16 bits, there must be two input channels when you use the reference signal. If you are not using a reference signal, omit the TDCIN parameter.
2. DIGITAL If the signals are routed through digital lines, specify this keyword. Both digital signal lines must be routed through the same digital port address, so the two values can be captured simultaneously and delivered through one DIGITALIN source pipe. Specify the TMMASK parameter to indicate which bit is the timing bit, and the REFMASK parameter to indicate which bit is the reference bit. The binary representation of the integer masks must be such that there is a "1-bit" at one bit position corresponding to the signal, and all of the other bit positions are "0-bits". For example, $0002 would select bit position 1, the next-to-low-order bit. If you are not using a reference signal, omit the REFMASK parameter.

Rotary encoders typically provide a reference output for "top dead center" (TDC) pulses. When the secondary encoder input TDCIN is specified, timing pulses are observed but skipped prior to the first TDC pulse received through the TDCIN channel, and the first resampling position will align with the TDC pulse. Suppose for example that the NENCODE parameter is 1024, and that 44 previous encoder pulses spanned by 248 samples were passed before the TDC pulse arrived. Then the 44 encoder pulses are disregarded. As far as the encoder processing is concerned, the first resampling position is at the rising edge of the 45th encoder pulse at the 249th sample; this will be “sample position 0” in resampled data sets. The TDC input is not used again, since the ratio of timing pulses per TDC reference pulse is known.

The NRESAMP parameter specifies how many sample positions you want to a later resampling analysis to retain per rotation. Usually, NRESAMP is integer-valued, but it is not restricted to integer values.

- Setting NRESAMP equal to the NENCODE parameter, the resampling results are essentially the same as if the data were obtained by directly clocking the sampling from encoder pulses, but without the specialized clocking connections.
- Setting NRESAMP to a value that is lower than NENCODE allows you to reduce the amount of resampled data produced, useful when rotation rates are very high.
- Setting NRESAMP to a value higher than NENCODE will give you more resampling positions per rotation than the encoder hardware is able to provide. The expanded data sets can provide you improved time resolution for analyzing a machine rotating at slow rates.

The analysis of the encoder signals results in a representation of locations along a piecewise smooth trajectory, with the results placed into the pipe OUTPIPE. A separate ADJRESAMP processing command can take the encoded information produced by the ENCDRATE command, and apply it to various maeasurement streams to determine sample values from those measured signals at the prescribed resampling positions.

When the command first starts, it does not know which encoder pulses are which. It can take up to one rotation to first locate a reference pulse, and to auto-detect the encoder's pulse-per-rotation resolution requires an additional rotation. An indeterminate number of samples might be bypassed to reach the aligned condition – and this can make it awkward to determine the point of initial alignment between time-based and encoder-based streams based on stream values alone.

Examples

```
CONSTANT  vLow   WORD  =  -16384
CONSTANT  vHigh  WORD  =   16384
PIPE      encoder_data       WORD
PIPE      tdc_data           WORD
PIPE      resamp_data        DOUBLE
CONSTANT  enc_resolution     LONG = 512
CONSTANT  resamp_positions   DOUBLE = 100.00

ENCDRATE( enc_resolution, encoder_data, tdc_data, vLow, vHigh, \
   ANALOG, resamp_positions, resamp_data )
```

An encoder produces analog signals ranging from -5 volts (logic low) to +5 volts (logic high). The two signals are delivered via analog inputs. The xDAP is configured for input ranges -10V to +10V range, so the nominal digitized signal voltages specified by vLow and vHigh cover half the range of the analog input channels. The signal with the high-rate timing pulses is delivered via the pipe encoder_data and the signal with the lower-rate "top dead center" reference pulses is provided via the pipe tdc_data. The enc_resolution parameter for the encoder hardware is 512 locations per rotation, which is too much data when rotating at a high speed, so the resamp_positions parameter is set to reduce the number of captured points per rotation to 100. The encoded rate information produced by the analysis goes to the resamp_data pipe, and this can be delivered to an ADJRESAMP task for resampling calculations.

See Also

ADJRESAMP, for resampling that uses ENCDRATE results

Processing Command

Module ROTM :: TRIGMAP

Converts triggering locations between time-based and encoder-based streams.

Syntax

TRIGMAP(INTRIG, RATEDAT, MODE, OUTTRIG)

Parameters

INTRIG
  Trigger pipe to read original untranslated event locations
  `TRIGGER`

RATEDAT
  Stream of sample timing data from independent timing analysis
  `DOUBLE PIPE`

MODE
  Keyword indicating the domain for translated events
  `TOINTERNAL | TOEXTERNAL`

OUTTRIG
  Trigger to receive the translated event locations
  `TRIGGER`

Description

A `TRIGMAP` task receives information about events from the `INTRIG` trigger pipe, uses information associating internal sample locations to external timing signal locations from the `RATEDAT` pipe , and generates the corresponding translated event location in the `OUTTRIG` trigger pipe.

The direction of the translation is indicated by the `MODE` keyword parameter. It must be specified as one of two keyword values, *not* enclosed in quote characters.

- `TOINTERNAL` In this mode, the mapping converts events detected relative to an independent external timing reference such as a rotary encoder, converting these locations to the corresponding locations relative to the internal sampling clock. For example, events corresponding to certain angles of rotary position, as determined by an encoder, can be mapped to a corresponding block of high-resolution time-samples.
- `TOEXTERNAL` In this mode, the mapping converts events detected in a high-resolution internal data stream, based on a fixed internal sampling clock, into corresponding positions in a data stream aligned to an external timing reference, such as a time-base standard or rotary encoder. For example, when using a rotary encoder, mapping an event detected in high-resolution samples determines the angle of rotation where this event occurred.

Sample locations in the original high-resolution data stream and timing events in the externally-driven timing stream do not always align perfectly. While sample values can be interpolated along continuous trajectories, events are discrete and cannot be interpolated. Consequently, if there is any time displacement between internal and external locations for an event, the translation will report the *next subsequent* sample location relative to the new stream. Or another way of saying this: a translated event always corresponds to a specific sample that is at the same time or immediately following the event in the originating stream, never earlier.

Coordinating event locations in a data stream based on a regular sampling clock and also on an adjustable-speed encoder signal can lead to some complications that do not exist for ordinary triggering, and you might need to be aware of these when analyzing your data.

- Events observed in one stream might not exist in the other stream. Data events could occur during periods of stall, or map onto a location where another event was previously mapped.
- Very small alignment differences during the timing analysis can result in an event that is not reported until the next discrete sample location.

Examples

```
TRIGGER   internTrig
TRIGGER   irigTrig
PIPE      pEncRate
...

TRIGMAP(internTrig, pEncRate, TOEXTERNAL, encTrig)
```

Timing events are observed in high-rate samples received via the `internTrig` trigger. The `TRIGMAP` task uses the mapping information from the `pEncRate` pipe to establish the association between position of samples in the high-rate clocked streams and externally-aligned streams. The `TOEXTERNAL` option specifies that the `internTrig` input events are based on the high-rate internal sampling clock, and the `encTrig` trigger receives output events that are aligned to timing pulses from the external timing signal. The timing information received from the `pEncRate` pipe is generated by a separate task (such as `ENCDRATE`).

See Also

`ENCDRATE`, for processing that establishes timing information relative to a rotation encoder
`TRIGSCALE` (DAPL system), for subsequent modification of trigger events

Appendix I.   Resampling Technology

What is resampling?

Resampling is the application of well-established DSP techniques to convert a stream of samples, captured under control of one timing clock, into another stream of samples referred to a different timing signal. You can think of resampling as accurately reconstructing the continuous signal from the available samples, and then selecting a new set of sample positions along this reconstructed signal. Resampling is closely related to the mathematical problem of *numerical interpolation.*

Resampling is widely used in digital music recording, synthesis, and playback. It has been much less commonly used for test and measurement applications, but it is no less valid for these.

When is resampling applicable?

Resampling is appropriate when there is a secondary timing reference, and you want samples aligned to that time reference, rather than the original sampling clock where measurements were captured.

How are resampled values computed?

The resampling calculations are very similar to the calculations that would be done to interpolate between two known points along a straight line segment. For the case of interpolating along a continuously varying signal, many known points (time-sample pairs) are used instead of just two. And the interpolation curve is a higher-order curve, not a straight line.

It is not immediately obvious how to reconstruct a signal with continuously varying curvature at any arbitrary location from discrete samples. However, the *Whittaker-Shannon Interpolation Theorem* gives an explicit formula for doing exactly this. Any continuous signal (subject to bandwidth restrictions) can be reconstructed *perfectly* from its samples.

The practical difficulty in applying the *Whittaker-Shannon Interpolation Theorem* is that a *perfect* reconstruction requires an infinite number of operations using arithmetic with infinite precision. Fortunately, perfect reconstruction is not necessary. There is no such thing as perfect measurements, and attempting an absolutely perfect reconstruction would only reproduce measurement noise in complete detail. Approximations to the *Whittaker-Shannon* formulas with a finite number of terms and finite numerical precision yield excellent results, with the numerical errors masked by the roundoff, nonlinearity, and noise effects that you will experience in real measurement data.

What is a "resampling filter"?

The theoretical *Whittaker-Shannon Interpolation Theorem* formula has the form of a *Finite Impulse Response* (FIR, transversal) digital filter — except for the infinite number of terms involved. A practical filter is obtained by reducing that filter to a finite length by *windowing*, a standard DSP technique. After this, the calculation process is the same as those for an ordinary FIR filtering operation. A slight complication is that a different filter must be constructed for every point to be evaluated.

What is the impact on signal amplitudes?

Just as ordinary frequency-selective filters have frequency-dependent gain properties, interpolation filters have these too, but the intent is to make these side-effects as small as possible.

Low frequencies should see no detectable gain error effects.

The interpolation filters in the Resampling Analysis Modules deliberately sacrifice accuracy at frequencies higher than 40% of the Nyquist frequency of the time-based sampling rate, in favor of best performance at 40% of the Nyquist frequency or below. This restriction does not necessarily compromise an application — the sample rate of the high-density clocked data stream is *oversampled* significantly compared to the time-rate of the encoder edges. This means that the practical bandwidth limitations of the resampling occur at high frequencies beyond those that should be present in the signals you are measuring.

Amplitudes of signals in the low frequency band are preserved within 0.0002 dB — that is, the attenuation effects on a signal of maximum amplitude are less than the bit chatter that you will get anyway from digitizing to 16 bits precision. Frequencies higher than the 40% of Nyquist frequency can still carry useful information, but you must keep in mind that the higher the frequency the greater the attenuation effects, so you cannot depend on the magnitudes to be accurate.

What is the impact on signal phase?

An interpolation filter has absolutely no phase distortion effects. However, practical implementation restrictions will sometimes select resampling locations that only an approximation to the ideal locations, causing small erratic effects on phase that look like a lot like *time jitter*.

Low frequency waveforms can't change very much from one sample to the next, so time displacements that are a tiny fraction of one sample at the original high-resolution have no observable effect. For a wave that is very high frequency and therefore changing very fast, time-quantization effects produce localized small time shifts can result in signal variations appearing very similar to random noise. Unlike true random noise, the magnitude of these effects decrease both as frequency and as amplitude decrease, so for typical real signals, the chatter is masked by natural background noise and is difficult to detect.

What are the "interpolation modes" and what is their effect?

Because time-jitter effects can sometimes be large enough to be observable, the *Rotating Machinery Analysis* module provides three operating modes for selecting a resampling location, given a mathematical determination of the desired location.

- Nearest neighbor mode. There are no numerical compuations. If the position of the resampled value is different from the position of an available time-based sample, the closest time-based sample position is selected, and its numerical value used unmodified. This is called the *Nearest Method* in the *Rotating Machinery Analysis* software. This approach is very efficient, but it can produce an observable "phase jitter" in high-frequency signals.
- Pre-calculated mode. The resampling location is rounded to one of a finite set of locations at which pre-computed interpolation filters are available. The phase shift is approximately 1/1000 of the phase shift that would result from one sample displacement at the high internal sampling rate. To put this in perspective, if a signal has maximum possible amplitude of 32767 counts, at the maximum 40% of the Nyquist limit, the difference in a sample value due to the phase jitter could be as much as 16 converter counts — approximately -42 dB relative to the signal level. At lower amplitudes, and at lower frequencies, the effects are proportionally smaller. This approach is called the *Fast Method* in the *Rotating Machinery Analysis* software. For signals with normal levels, bandwidth, and background noise, or that are subjected to a frequency spectrum analysis, this is the method of choice.
- Phase-interpolated mode. The resampling is calculated at two bounding locations for which pre-computed interpolation filters are available, and then the final sample value is obtained by linearly interpolating between these two bounding filters. Using this method, *timing jitter* becomes undetectable. However, this computation requires twice the CPU resources of the pre-calculated mode. This approach is called the *Accurate Method* in the *Rotating Machinery Analysis* software. For extremely clean signals and absolute minimal measurement error, this is the method of choice. In practice, it rarely adds any meaningful improvement to measurement quality.

Appendix II. Hazards and Limitations

Resampling is a numerical approximation, with known limitations. In practice, however, these details are buried in the last-digit roundoff of your 16-bit integer sample values. With properly bandlimited input signals, you are unlikely to have a world-class statistical expert who could determine whether or not your signals were obtained by resampling or captured directly under hardware control.

It turns out that there are other considerations that are much more mundane but much more significant to the quality of your results than the mysterious interpolation calculations.

Tradeoff: tracking accuracy vs. digital chatter

In contrast to numerical problem of interpolation calculations, it turns out that determining where and when to apply the calculations is much more difficult. Suppose that the following sample counts are observed during a sequence of encoder pulses.

```
   10   10   10   10   11   10   10   10   10   11   10   10 …
```

One possible interpretation of this is that the machine was rotating at (for example) 4000 RPM for the duration of 4 encoder pulses, suddenly dropped to 3600 RPM for the duration of 1 encoder pulse, and then one pulse later returned to the 4000 RPM rate, repeating this pattern. While this is perfectly consistent with the data, it is extremely unlikely that a massive machine is going to make such large changes in velocity over the course of a few microseconds. What is more likely is that the machine is maintaining a steady rate and the integer number of samples observed for the duration of any one encoder pulse varies according to how the clocked samples and the encoder pulses happen to align. By smoothing the discrete sequence, we can get a reasonable estimate of the steady rate of rotation.

Yet, what happens in a realistic situation where the rate of rotation is not really steady, but varying continuously? In that case, an unexpected increase or decrease in the number of samples-per-pulse could be an important indicator of a speed change. These two goals of ignoring the digital chatter from sample counting, but also responding decisively to real speed changes, are at cross-purposes. There is a compromise, and ultimately, the curve reconstruction is not perfect. Vibrations with cycles spanning 12 or fewer encoder pulses per vibration cycle cannot be tracked at all. There are small local deviations from the theoretical ideal where the smoothing is locally too aggressive or not aggressive enough to separate the true speed trajectory from the chatter. Typically, the chatter-induced deviations will appear random and less than 1 encoder position in magnitude — but they can be larger if noise levels are high and speed variations are large and sudden. Results are best when changes in the machine operating state are relatively slow and steady. This will be true in almost all practical situations, due to the large inertia of the machine.

Data rate inconsistency

There are potential problems when you try to relate a data stream captured at the original sample time intervals to another resampled stream using a different time reference. For example, suppose you resample a stream that originally had 1001 samples per second. After resampling, it has 1000 samples per second. If you transfer "1000 samples of this stream, 1000 samples of that stream" to your host system using a DAPL system `BMERGE` command, every time you transfer one pair of blocks this way, one sample of data from the original sample stream remains backed-up in buffer memory. Do this for a million blocks, and you get a million samples backlog. Eventually, something has to break.

If you transfer data from the two streams through separate communications pipes — and remember to receive different numbers of samples from the two channels — there is no problem.

Event locations

The DAPL system locates data according to their positions in the data stream. Events detected in a stream that is resampled will have positions that are not clearly related a position in the original input sample stream, since resampling locations vary with the rotation speed. Or the reverse: an event detected in the original sampled data stream will not be clearly related to any specific position in a resampled data stream. To detect events in one kind of data stream, and retain the corresponding data from another kind of stream at a different data rate, you must use the TRIGMAP command to appropriately realign the event locations.

Events observed in one stream might not exist in the other stream. While samples based on an internal time clock are guaranteed to move forward in a consistent manner, samples based on an external encoder device can slow, or in the extreme case, go all the way to zero. Data events cannot be mapped during time intervals in which the external timing source stalls, and can sometimes be lost immediately before and immediately after stalls.

At most one event can be posted for a data stream at any given location; if there are more events than locations, not every event detected in an input stream will result in an event in the output stream. An event occurring before synchronizing to an external timing stream cannot be mapped. Data collected under more normal conditions, without stalling or rapid bursts of events, should not show anomalous behaviors.

Aliasing

As is the case with any sampling, when resampling reduces the data rate, there is a possibility that high-frequencies representable in the original data set become aliased onto lower frequencies. The only way to avoid this is to keep the potentially troublesome frequencies out of the original signal at the time that sampling or resampling is applied.

Half-sample shift

There is a fundamental and unavoidable time shift of ½ sample relative to the high-rate sampling oscillator. This applies to *any* detection of *any* position in *any* sampled data stream, and resampling has no immunity from this.

Suppose that something produces a detectable change in a data signal (such as a timing pulse edge). When can you detect this change? Well, certainly no sooner than the event actually occurred. But after that, you can't detect it until the change appears in the next sample. In the best case, when the change occurs *just before* sampling, the delay is approximately zero. In the worst case, the change occurs *just barely too late*, delay for detection is nearly 1 full sample. The worst and best cases average out to a ½ sample shift in the entire data stream, with variations distributed around that. If multiple channels are resampled according to this timing, all of these channels will show this same ½ sample delay, and will show no time delay with respect to each other.

Bandwidth limitations

It is not possible to preserve the full theoretical bandwidth through to the Nyquist limit *and* to preserve full accurcy at every frequency – not without filters of impractical length. There must be a compromise somewhere. The Rotational Analysis functions make a deliberate choice to preserve frequencies up to 40% of the Nyquist limit.

It is advisable to apply simple electronic filtering to cut off very high frequencies above the band you must preserve accurately. If the cutoff of this filter is below 40% of the Nyquist frequency of high-rate sampling, conditions are ideal for applying a resampling analysis. If you are going to further decimate your data streams, your smoothing filters can be more aggressive, and your results will improve.

If some high-frequency noise gets through, the imperfect reconstruction of these higher frequencies is not necessarily bad. The effects are much the same as applying a weak lowpass filter to the data first, and then resampling. If no high frequencies were supposed to be present anyway, the reconstruction has helped to suppress noise and can be viewed as an improvement.

Startup dead time

The Rotating Machinery analysis cannot detect a rising pulse edge until it first sees a signal at a low logic level and then sees the signal at a high level, indicating edge arrival. Consequently, the first few samples can never be used by the analysis. They correspond to a fragment of a complete timing pulse, and unknown fragments cannot be used for anything.

When a timing reference “top dead center” pulse is available, any pulses or pulse fragments prior to the rising edge of that timing pulse will be ignored. The first timing pulse at or subsequent to the reference timing pulse will be “initial sample zero” in the resampled data set. The resampled stream will be as if no encoder activity existed prior to the rising edge of the reference pulse. The corresponding samples in the clocked sample stream will still be accounted for, but not used for anything.

These start-up delays can make a direct comparison between the original sample stream and the resampled stream tricky. However confusing, the difference is just a delay to pick the appropriate starting point.

Transient Behaviors

Like other filters, resampling filters have a transient response characteristic. Ordinarily, it is difficult to know this. However, you can see the transient response if you violate the "40% of Nyquist frequency" guideline and include an abrupt "step transient" in your signal. The response of a resampling filter is very much like the response of any FIR filter: there is a small, artificial, "transient" wobble in the output data stream. This is not necessarily harmful as long as you recognized that it is an artifact of the filtering, and not something real.

The step transient can seem unusual. The interpolation filter is two-sided, applied to samples both ahead of and behind the desired evaluation point, using data held in buffer storage. Consequently, the "transient wobble" can look like it is propagating both forward and backward in time, a curious illusion.

------------ End of Document ----------