

MsPinky's Vinyl Tracking Object

Software Specification Version 0.0.4

by Scott Wardle. April 1, 2004

updated March 18, 2006

1. Introduction

The MsPinky Vinyl Tracking Object "MPVT" is designed to take a stereo stream of PCM audio samples and produce measurements of the velocity, direction, signal power, and absolute position of the stylus playing a MsPinky vinyl record on a standard DJ turntable. Alternately, the same software object can calculate the pitch, direction of playback, volume level, and position of the read point of a CD player which is playing back a CD recording of the MsPinky scratch control signal. Figure 1 shows a graphical representation of the MPVT object responding to inputs and producing outputs.

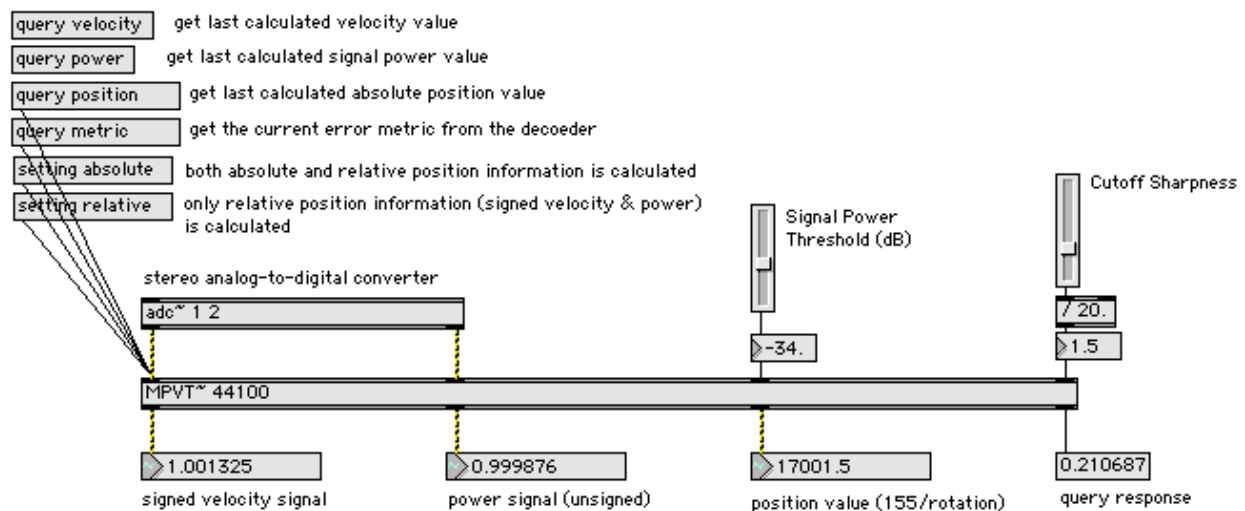


Figure 1. MPVT object inputs & outputs.

2. Creating a new instance of the MPVT Object

A single instance of MPVT corresponds to just one turntable playing the MsPinky vinyl control disc, or one CD player playing a recording of the MsPinky control signal. For systems that use multiple turntables, multiple instances of the MPVT object must be created. The MPVT object must be explicitly created by the client, and it does nothing unless it receives a command from the client. MPVT operates on buffers of stereo PCM audio samples. It does not work with compressed sample data. Its internal processing requires the use of temporary arrays that must be pre-allocated to be large enough to hold the maximum number of samples which the calling client will ever request to be processed by a single call to its processing functions. The client who wishes to use MPVT must therefore provide this maximum buffer size to MPVT when it creates a new instance of the object.

An instantiation procedure for MPVT will take two parameters: 1) an integer-valued parameter which specifies the maximum buffer size, and 2) a double floating-point value specifying the sample rate (Hertz) of the samples that will be provided to the MPVT object.

```
void* MPVT_CreateNew(int max_buffer_size, double sample_rate);
```

The function `MPVT_CreateNew(int,double)` returns a `void*` pointer which can be used appropriately to access the services of MPVT.

3. Destructing an instance of the MPVT Object

Each instance of the MPVT object should be explicitly destructed when it is no longer needed.

```
void MPVT_Destruct(void *the_object);
```

The pointer returned by `MPVT_CreateNew` is the same one that should be passed in as the single argument to `MPVT_Destruct`, and likewise as the first argument to all other MPVT functions except `MPVT_CreateNew`.

4. Processing Audio Samples using the MPVT Object

The MPVT object requires a stereo stream of PCM audio samples formatted as 32-bit floating-point values in the range `[-1.01.0]`. For each 64 input samples provided, MPVT provides one each of the following measurements:

A. A signed Velocity measurement. The sign indicates direction of motion, forward being positive, backwards being negative. The magnitude represents the ratio of the speed of the playback device to the nominal 33-1/3 rpm. Hence, if the turntable playing the MsPinky vinyl control disc is turning at 45 rpm in the forward direction, the velocity value returned by MPVT will be approximately $1.35 = 45/33.333333\dots$. Format: double-precision floating point.

B. An unsigned Signal Power measurement. This is a value between 0.0 and 1.0 which corresponds roughly to an estimate of the signal power produced by the playback of the Vinyl Control Disc (alternately the CD recording of the control signal). This power measurement is very useful as a muting signal to combat the spurious incorrect measurements which gradually creep in as the velocity of the turntable approaches zero. Format: double-precision floating point.

C. Absolute Position measurement. This measurement gives the absolute position of the playback read point (the phonograph stylus or the CD read head) as a value starting at zero, and increasing at a rate of 155.0390625 per rotation of the MsPinky Vinyl Control Disc. This can be expressed alternately as 86.1328125 per second of playback at the nominal rate of 33-1/3 turntable rpm. For the Ms Pinky CD-specific control signal, the absolute position values are returned at a rate of 172.265625 per second of playback at a nominal 0% pitch setting.

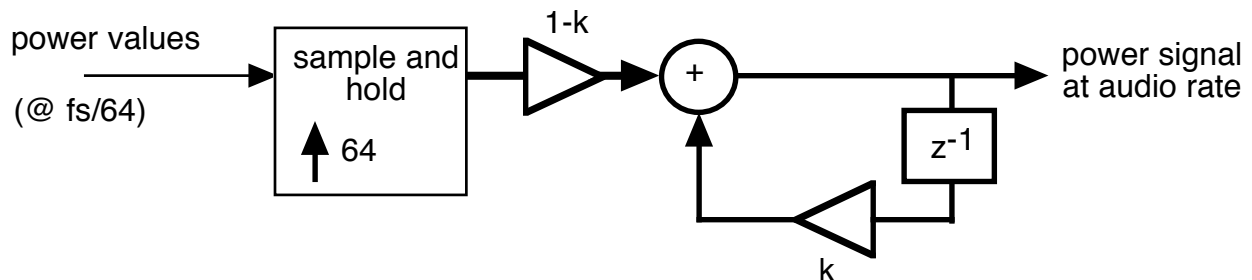
```
void MPVT_ProcessBuffer(void *the_object, Float32 *inBufferLeft,  
    Float32 *inBufferRight, long num_samps, Float64 *velocityVals,  
    Float64 *powerVals, Float64 *positionVals, long &num_measurements);
```

where again for the first argument clients should pass in the same pointer returned by `MPVT_CreateNew`; `inBufferLeft` and `inBufferRight` point to arrays of samples from left and right stereo channels each of length `num_samps`; `velocityVals`, `powerVals`, and `positionVals` each point to arrays (allocated by the calling client) that are large enough to hold the number of measurements that will be produced for the input sample arrays provided by the

calling routine (approximately one measurement per 64 input samples), and *num_measurements* on return will hold the number of measurements returned in the *velocityVals*, *powerVals*, and *positionVals* arrays. Note that the arrays pointed to by *velocityVals*, *powerVals*, and *positionVals* should each be allocated to hold one value per 64 input samples, plus a safety margin of one additional element. IMPORTANT: the value of *num_measurements* that is passed into the routine when calling it should be equal to the length of the arrays pointed to by *velocityVals*, *powerVals*, and *positionVals*. MPVT_ProcessBuffer will only return as many values into *velocityVals*, *powerVals*, and *positionVals* as is specified by the value of *num_measurements* when the routine is called.

The application of the velocity values is very straightforward. They can be used to control, for example, the resampling ratio of a sample interpolator; where the resampling ratio is updated only once for each new value returned from MPVT_ProcessBuffer.

The application of the power values is slightly more complicated. For most purposes, the power values should be converted to a smooth signal at the audio rate for application as a direct modulator signal on an audio rate signal such as the output of a sample interpolator. To create from the power values a signal at the audio rate which is also smooth, it is recommended that a sample-and-hold upsampling-by-64 conversion be used, followed by some appropriate form of low-pass filtering to remove the sharp transitions which would cause zippering artifacts in any audio signal to which the power values signal was applied. A suitable system for creating from the power values an audio rate signal which could be used to mute other signals at the audio rate is shown here:



5. Setting Parameters of the MPVT Object

5.1. Signal Power Threshold

The MPVT object operates on the audio input differently according to the signal level it receives. Because of the velocity-sensitive nature of the standard DJ phonograph stylus, no useful measurements can be made when the MsPinky Vinyl Control Disc is not moving on the turntable. As the turntable velocity approaches zero, the ability of the MPVT object to produce accurate measurements diminishes. In order to gracefully transition from high-confidence measurements to complete oblivion, a power threshold level is set in units of decibels below which MPVT applies a muting function to its power level output. Whenever the input signal power is measured and found to be below the specified threshold, MPVT basically assumes that no useful measurements can be made and therefore produces a value approaching zero as the Signal Power measurement discussed in 4-B.

```
void MPVT_SetSignalPowerThreshold(void *the_object, Float64 inThresh);
```

5.2. Cutoff Sharpness

As discussed in the previous section (5.1), MPVT gradually reduces its Signal Power measurement towards zero as it transitions from a state of high-confidence measurements to a state of oblivion. The sharpness of this transition is governed by the Cutoff Sharpness parameter, a double-precision floating-point value typically in the range [0.0 ... 10.0] . Setting Cutoff Sharpness to approximately 1.0 is a good starting point for finding the optimal setting. Higher values give sharper cutoff. Too low a value will cause MPVT not to do any useful thresholding.

```
void MPVT_SetCutoffSharpness(void *the_object, Float64 inSharpness);
```

5.3. Absolute Mode vs. Relative Mode

When set to operate in absolute mode, MPVT always attempts to calculate Velocity, Signal Power, and Absolute Position. When set to operate in relative mode, only Velocity and Signal Power are calculated. In relative mode, if MPVT were to be used to control an audio file playing back, then when the user lifted the needle and moved it to another spot on the MsPinky vinyl control disc, the audio file playing back would just continue playback at the point it left off when the needle was lifted. In absolute mode, the file would re-cue to the new position.

```
void MPVT_SetAbsoluteMode(void *the_object, Boolean inMode);
```

Passing the value “false” sets the operation of MPVT to relative mode. Passing the value “true” sets the operation to absolute mode.

5.4. Vinyl Generation

There are currently four versions, or generations, of MsPinky vinyl. There is also a new fifth generation signal for use with CD scratchers. Use the MPVT_SetVinylGeneration function to set the MPVT object to respond appropriately to the vinyl (or CD) you’re using.

```
void MPVT_SetVinylGenerations(void *the_object, long generation);
```

Passing a value of 1 sets the MPVT to respond to first generation vinyl, likewise a value of 2 sets it to respond to second generation vinyl, also known as “2nd Mutation” vinyl. Passing a value of 3 sets it to respond to “3rd Generation” vinyl, etc. To set MPVT to respond to the CD control signal, pass in a value of 5 for the generation.

5.5. Sample Rate

Should the sampling rate of the samples you’re providing to MPVT change during operation, you should call this function

```
void MPVT_SetSampleRate(void *the_object, Float64 inSampleRate);
```

in order to renormalize the MPVT object’s internal processing to the new sampling rate. As with all other MPVT functions, the first argument is a pointer to the MPVT object itself. The second argument is simply the new sampling rate in units of Hertz.

6. Querying the MPVT Object

6.1. Querying the Error Metric

The MPVT object decodes the patterns of tones on the record into binary numbers that represent positions on the surface of the vinyl. In the process of decoding, it produces an “error metric” which is a measure of the amount of distortion and noise present in the signal. When setting up the system initially, and when troubleshooting a system that is malfunctioning, querying for the error metric is a useful check to see if the decoder is happy.

```
Float64 MPVT_Query_ErrorMetric(void *the_object);
```

The function simply returns a 64-bit floating point value indicating the error metric. If the value is much higher than 0.5, the decoder is not happy. The cause of this could be anything from an ungrounded turntable to a buildup of dust around the stylus. So it doesn't tell you specifically what the problem is, but at least it provides a warning that something needs checked. In normal, healthy MsPinky vinyl tracking scenarios, the error metric should range from about 0.1 to about 0.2.

6.1. Querying the Velocity, Signal Power, and Absolute Position

The MPVT object can also be queried asynchronously from the calls to **ProcessBuffer** for the three measured quantities described in 4-A through 4-C (Velocity, Signal Power, and Absolute Position).

```
Float64 MPVT_Query_Velocity(void *the_object);
```

```
Float64 MPVT_Query_SignalPower(void *the_object);
```

```
Float64 MPVT_Query_AbsolutePosition(void *the_object);
```

These functions return the last calculated value for each measurement. It is not a good idea to use these querying methods to control the playback of an audio file, for instance, in a virtual scratching application. The resulting granularity of the measurements applied to the file control will not be sufficient to provide realistic virtual scratching. These methods are, however, very useful for updating display values in your GUI or for otherwise monitoring your MsPinky system performance.

7 Summary

The MPVT object exposes the following callable functions

7.1 - Creation/Destruction:

```
void* MPVT_CreateNew(int max_buffer_size);
```

```
void MPVT_Destruct(void *the_object);
```

7.2 - Processing Sample Data:

```
void MPVT_ProcessBuffer(void *the_object, Float32 *inBufferLeft,  
    Float32 *inBufferRight, long num_samps, Float64 *velocityVals,  
    Float64 *powerVals, Float64 *positionVals, long &num_measurements);
```

7.3 - Setting Parameters:

```
void MPVT_SetSignalPowerThreshold(void *the_object, Float64 inThresh);  
void MPVT_SetCutoffSharpness(void *the_object, Float64 inSharpness);  
void MPVT_SetAbsoluteMode(void *the_object, Boolean inMode);  
void MPVT_SetVinylGeneration(void *the_object, long generation);  
void MPVT_SetSampleRate(void *the_object, Float64 inSampleRate);
```

7.4 - Querying:

```
Float64 MPVT_Query_ErrorMetric(void *the_object);  
Float64 MPVT_Query_Velocity(void *the_object);  
Float64 MPVT_Query_SignalPower(void *the_object);  
Float64 MPVT_Query_AbsolutePosition(void *the_object);
```