[audioordeal.co.uk/how-to-build-a-vst-lesson-2-autopanner/](https://audioordeal.co.uk/how-to-build-a-vst-lesson-2-autopanner/) First create manual stereo panner. download tutorial 1 from [github.com/aRycroft/JuceTutorial1](https://github.com/aRycroft/JuceTutorial1)

**Stereo Panner**

To pan audio we attenuate the audio from one channel while boosting the other. To control plugin we modify gain slider we created in tutorial 1.

At top of PluginProcessor.cpp there’s addParameter line we added last time. Change the default slider value to 0.5 by changing the final float to 0.5f.

addParameter(gain = new AudioParameterFloat(“gain”, “Gain”, 0.0f, 1.0f, 0.5f));

Now go to the process block in PluginProcessor.cpp, it should look like this.

A screen shot of a computer

Description automatically generated with medium confidence

Currently gSlider multiplies both audio channels by same value. To pan audio one channel needs to be multiplied by inverse of other, e.g

inputL = inputL \* 0.3;

inputR = inputR \* 0.7;

inputL = inputL \* 1;

inputR = inputR \* 0;

With gSlider variable we can multiply left channel by 1 – gSlider, and other just with gSlider.

A screenshot of a computer

Description automatically generated with medium confidence

Replace this

//inputL = inputL \* gSlider;  
//inputR = inputR \* gSlider;

With this

inputL = inputL \* (1 – gSlider);  
inputR = inputR \* gSlider;

Build the plugin and move the VST to your DAW’s VST folder. When changing the gain slider, you should see the audio pan left or right. When panned to the centre you will notice a dip in volume of -3dB. We use a different panning algorithm later to fix this problem.

**AutoPan**

Now we can pan audio left and right, we need to find a way to modulate the gSlider value over time. To achieve this we are going to find the value of a Sin wave at different points, then multiply these values with the channel data.  To start, we will modulate the value between 0-1 every second.

To modulate the levels the correct amount we need to do a bit of maths.

**Every cycle of Sin = 2 \* pi radians.**

**The sample rate of the DAW means that a certain number of samples happen every second, so**

**2pi radians / sample rate = Radians per Sample**

For every sample, we can find Sin (Radians per Sample) then add another ‘Radians per Sample’ length to find a value for the next sample. These values can be multiplied to each channel of audio instead of the gSlider variable.

A screen shot of a computer

Description automatically generated with medium confidence

const int numberSamples = getSampleRate();

const float radsPerSample = (2 \* double\_Pi) / numberSamples;

This finds sample rate and how many radians required for each sample length. Use const for these values as they won’t change after each iteration.

Now add float to hold value of the radians for a given sample. We don’t want to initialise this variable every buffer length so we will add it to the header file. Go to PluginProcessor.h and add this line in the private section.

A screenshot of a computer program

Description automatically generated with medium confidence

float nextRad = 0;

Now add the following code to your ProcessBlock function in PluginProcessor.cpp.

float sinValue = 0.5 \* std::sin(nextRad) + 0.5; //1

inputL = inputL \* (1 – sinValue); //2  
inputR = inputR \* sinValue;

nextRad += radsPerSample; //3

A screenshot of a computer

Description automatically generated with medium confidence

1.Sets sinValue as Sin at value of nextRad, Sin scaled to value between 0-1.

2. Replaces gSlider with sinValue.

3. Adds radsPerSample to nextRad.

Now audio will pan between left and right channel every second.

Currently, the nextRad variable will get bigger forever while the plugin is active,

A screenshot of a computer program

Description automatically generated with medium confidence

to solve this we can add an if statement to the bottom of the process block

if (nextRad > numberSamples) nextRad = 0;

This resets nextRad after it gets larger than numberSamples.

Add new audio parameter float to change how often audio pans between L & R. I called it mS and set range to 10 – 5000. Now add this to process block.

A screen shot of a computer

Description automatically generated with medium confidence

float mSeconds = mS->get() / 1000;

//Replace this  
//const int numberSamples = getSampleRate();

//With this  
int numberSamples = getSampleRate() \* mSeconds;

This finds the value of the mS slider and divides it by 1000, to find the value in seconds, then multiplies this to the sample rate. **Make sure to remove the const from the numberSamples variable.**

Build the plugin, and you can use the mS slider to change the autopanner rate.

There is a better algorithm for panning audio. We can use a constant power panning algorithm by changing some code in the process block.

A screenshot of a computer program

Description automatically generated with medium confidence

//Replace This

//float sinValue = 0.5 \* std::sin(nextRad) + 0.5; //1

//inputL = inputL \* (1 – sinValue); //2

//inputR = inputR \* sinValue;

//With This

float sinValue = std::sin(nextRad) + 1;

sinValue = (sinValue \* double\_Pi) / 4;

inputL = inputL \* cos(sinValue);

inputR = inputR \* sin(sinValue);

After adding this code and building plugin you should see a smoother transition between the left and right channels.

we developed a simple autopanner plugin. If you make mS very small you should hear some amplitude modulation effects.