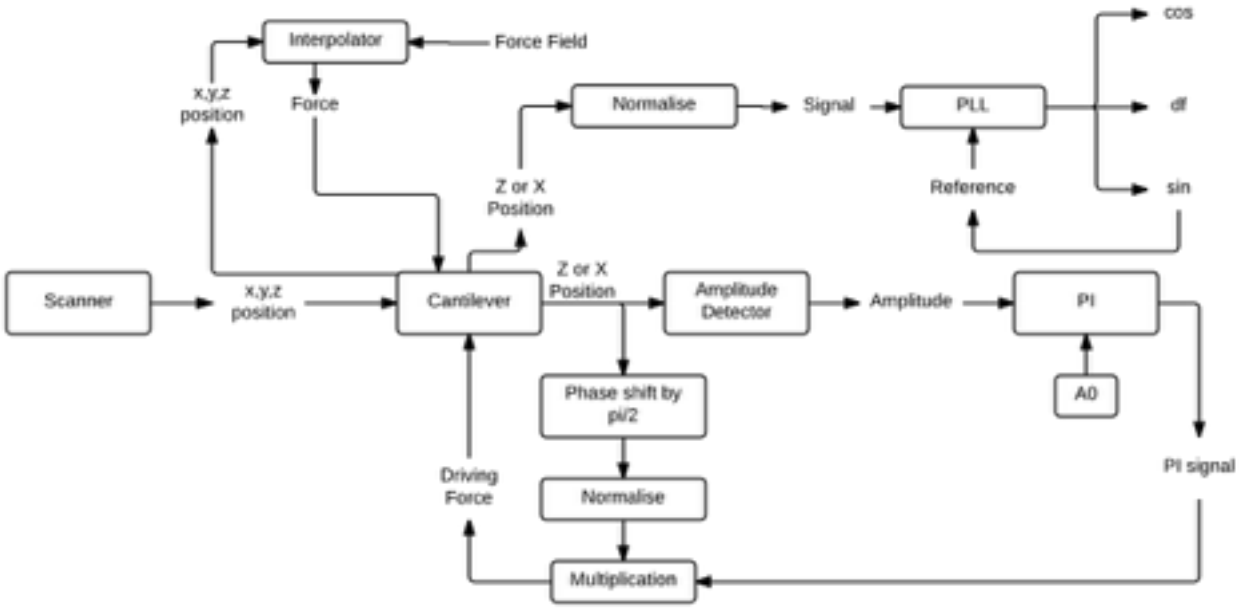
PyVAFM Tutorial

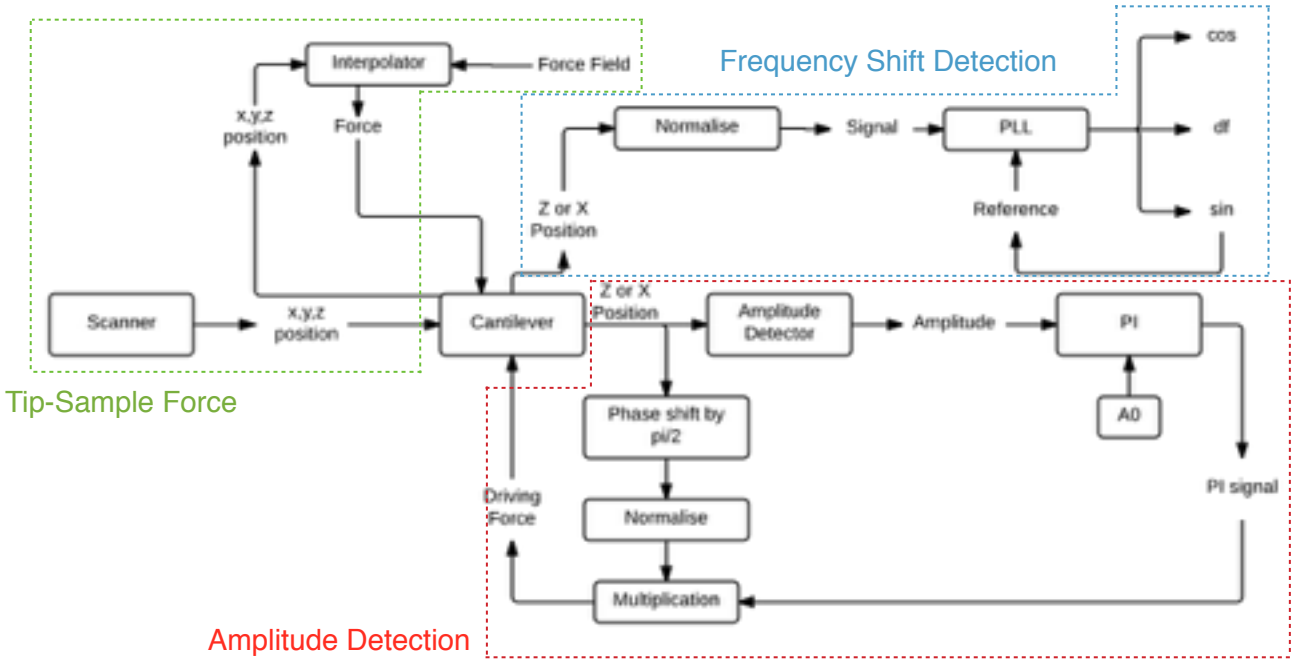
Introduction

In this tutorial you will simulate an FM-AFM simulation of a surface in water. The presentation will have described most of the basic understanding you will require to build a simulation although if something is not clear or you want more information on a circuit I suggest you take a look at our online manual (http://singroup.github.io/PyVAFM/). The online manual contains all the information required to use the PyVAFM including extra tutorials and how to use each circuit.

Anatomy of an FM-AFM simulation



Above is the schematic for our FM-AFM simulation. It may look a bit complicated at first but we shall go through it in parts. The AFM simulation can be split into 3 specific parts: Amplitude detection, Frequency shift detection and Tip-Sample force.



As can be seen form the above diagram all 3 parts are linked together through the cantilever circuit. The cantilever circuit has the name suggests simulates the dynamics of the cantilever with user defined properties.

In order to keep our cantilever oscillating at a constant amplitude we require the Amplitude Detection part. We take our cantilever signal and pass it through an amplitude detector (More on this later) in order to obtain the amplitude. Now we need to decide if we should increase or decrease the driving signal. We do this using the PI (proportional integral) circuit. Basically this circuit compares two signals and produces an error based on how different they are. By setting one of these signals to the amplitude we want and the other to the detected amplitude we can generate such an error signal. Finally we must multiply our error signal with a wave before supplying it as a driving force. This can be done by taking our cantilever signal, phase shifting it by π/2 (so our final cantilevers phase remains unchanged) and multiplying it with our error signal. Now by using this signal that we have generated as a driving signal the cantilevers amplitude will be kept constant.

We need to measure our cantilevers frequency change and we do this using the Frequency Shift Detection part. As with the amplitude detection system we take our cantilever signal and normalise it. Next we pass this normalised signal to the PLL. The PLL is a device that compares the phase of two signals and hence can calculate the frequency shift between them. By supplying the normalised cantilever signal and by connecting the PLLs reference signal to one of the PLLs output a frequency shift (df) can be obtained.

Finally the Tip-Sample interactions are governed by the force field as described in the presentation. By passing the cantilevers position to an interpolation circuit a force can be found at any location in the force field. Lastly in order to move the cantilever around the force field we must use the scanner circuit allowing us to set positions, scan speeds etc.

FM-AFM Input Script

For a new user building the entire FM-AFM script may take a fair bit of time, since we have limited time in this tutorial you have been provided with a mostly complete script. Although 3 things (the most important things) are missing. You must add the amplitude detection circuit, the Phase locked loop as well as tune these components, as well as scripting the cantilevers movement in order to produce an image.

Step 0 - Starting Cygwin, using Gnuplot and finding the vafm

In order to start using the PyVAFM we must use the Cygwin terminal, this is a program that resembles a linux terminal in windows. To start Cygwin just start the application that is located in the desktop. You can find the VAFM by entering the following command:

cd ../vafm/MainzTutorial

Now your terminal is located in the correct folder so we should also make sure you can edit all the files required. In windows file explorer navigate to C:/Cygwin64/Home/vafm/MainzTutorial. In this file you should find the input script called MainzTutorial.py, open it using whatever text editor you would like although I suggest you use sublime text that is installed.

Now we need a plotting program, Gnuplot is installed on your computer so open it with the desktop shortcut. In the Gnuplot select the chdir button at the top and navigate to C:/Cygwin64/Home/vafm/MainzTutorial. Incase you haven't used Gnuplot before in order to plot a data file use the following command:

plot “Filename” using x:y

Where Filename is the name of the file you want to plot and x and y are the columns you want to plot. So for example if you want to plot a file named data.dat and columns 1 and 2 then use:

plot “data.dat” using 1:2

Now we have all the programs open we should just leave them open since we will be using them alot. Now we can move on to setting up the vafm.

Step 1 - Choosing Simulation Parameters

At the top of the script 4 variables are required to be defined.

- F0, The resonance frequency of the cantilever.
- Az, The starting amplitude of the cantilever.
- Q factor of the cantilever.
- k, the spring constant of the cantilever.

These settings will heavily influence the contrast you get (This is why tools such as the vafm can be useful). You are tasked with choosing the variables here. In order to help get you started I shall go through the effect of each of these parameters as well as roughly the ranges you should be choosing in.

f0 or the resonance frequency of the cantilever will have an effect on the response time which is proportional to 1/f0 . In addition to this a high frequency oscillation will require a lower time step in order to integrate successfully. I would suggest a resonance frequency in the 10s of kHz range and a time step of approximately 10 microseconds (1e-5 seconds).
The amplitude of the system has an effect on the range of df you will obtain, since the cantilever averages forces over the range of its oscillation a large amplitude would result in the cantilever spending a lot of time far from the surface. A typical amplitude would be somewhere around a few tenths of a nm.

the Q factor describes how quickly the cantilever will lose energy. This also can have an effect on the response time which is proportion to Q. A typical Q factor for AFM in liquid would be less than 10.

The k factor has an effect on the force sensitivity of the cantilever. Typical k you could use range from 10 to 100 N/m.

TASK

Set k factor
Set Q factor
Set Az
Set f0
Select an appropriate time step

Step 2 - Adding and connecting circuits

Next we must add some circuits, as mentioned above most of the input script has already been done for you. All that is left is to add and connect the Automatic gain controller to obtain constant amplitude and the Phase Locked Loop to measure df.

First lets add the Amplitude detector. We will use a Proportional Integral circuit to achieve this (http://singroup.github.io/PyVAFM/classvafmcircuits__control_1_1PI.html). The previous link contains all the information about the initialisation parameters for a PI circuit.

The order that you add circuits does matter, in order to make thinks simpler there is a comment in the input file “#Add PI Circuit Here”, you should add the PI circuit under that comment.

TASK

Add a PI circuit
Set name to agc
Assign Ki to 0
Assign Kp to 0
Assign set to Az (the variable not the number)
Set pushed to True

Now that the the PI circuit has been added lets add the analogue phase locked loop (PLL) (http://singroup.github.io/PyVAFM/namespacecustoms.html#a04dc7cf766120fc9d488b8c625d5c5cd).

As with before add the circuit under the comment “#Add PLL here”.

TASK

Add a PLL circuit
Set type to Machine
Set name to pll
Set assembly to aPLL
Set filters to [0,0,0]
Set gain to 0
Set f0 to f0 (The variable not the number)
Set Kp to 0
Set Ki to 0
Set pushed to True

Now that we have our circuits added we must connect them up. Unlike adding circuits the order we connect circuits doesn't matter. Although in an effort to keep things tidy I suggest you add the following connections to the end of section 2 in the input script.

TASK

Connect the following circuits and channels:

Output Circuit name	Output Channel	Input Circuit name	Input Channel	Effect
canti	ztip	amp	signal	Connect the cantilever to the amplitude detector.
amp	amp	agc	signal	Connect the detected amplitude to the age.
agc	out	exc	in1	Connect the AGC signal to a multiplication circuit.
pll	sin	inv	in1	Connect the PLL sin signal to a inversion circuit.
inv	out	exc	in2	Connect inverted PLL signal to the multiplication circuit.
amp	norm	pll	signal1	Connect the normalise cantilever signal to the PLL
pll	sin	pll	signal2	Connect the reference signal of the PLL to one of its inputs

Step 3 - Tune the circuits

Step 3.1 - Amplitude Detector

Now that all the circuits have been added we shall tune each of the circuits individually.

We should start with the amplitude detection circuit. Our amplitude detection system uses a form of envelope detection based (https://en.wikipedia.org/wiki/Envelope_detector). If you look for the circuit named ‘amp’ located under the cantilever circuit you will notice it contains a fcut variable. This is controls the frequency cut off for the low pass filter. This must be adjusted based on the incoming signal frequency.

Before we start tuning this lets make some adjustments to allow us to visualise this easier. Firstly since we want to be able to compare the measured amplitude with the cantilevers real signal lets set the Q factor to a very large number so we get little energy loss.

Next we don’t want this cantilever to be driven (we haven’t tuned the AGC yet). So lets comment out (using # before the command) :

machine.Connect(‘amp.norm’,’pll.signal1').

Also uncomment the following line:

machine.Connect(‘testwave.cos’,'pll.signal1')

Now our cantilever will oscillate freely for a long time so we can measure its amplitude. Now adjust the fcut of the amp circuit a good guide would be to try 1% of the f0 you set earlier.

We have to also run the PyVAFM for some given amount of time, we can do this by using the command machine.Wait(0.5). This command should be entered in section 4 of the input script.

We can now execute the script by entering python MainzTut.py into the terminal.

You can now plot the Debug.dat file to check if the amplitude detection circuit is working correctly. Do this by plotting the cantilever vs time (first and third column of debug.dat) and comparing it to Amplitude vs time (first and fourth column of debug.dat). If you find the amplitudes to be very different then adjust fcut and try again. A quick tip is you can use the replot command in Gnuplot to quickly plot the last graph.

Don’t forgot to reset the Q factor back to what it was before.

TASK

Set the Q factor to a very large number
Comment out machine.Connect(‘amp.norm’,’pll.signal1’)
Uncomment machine.Connect(‘testwave.cos’,’pll.signal1’)
Adjust the value of fcut
Run the simulation for 0.5 seconds
Compare the amplitude of the cantilever with the detected amplitude
Reset the Q factor back to what it was before

Step 3.2- Automatic Gain Control

In order to maintain constant amplitude we have to take our computed amplitude and compare it to a set point. This is done using a Proportional Integral (PI) circuit (http://singroup.github.io/PyVAFM/classvafmcircuits__control_1_1PI.html). Essentially a PI circuit compares a signal with a set point and produces an error signal. We can multiply this signal with a driving signal to produce an excitation signal which will keep the cantilever at constant amplitude. Although our PI circuit must be tuned so it will respond quickly to any changes in amplitude that occur. The equation governing the PI circuit is as follows:

$$Out = K_p(Set - Signal) + K_I \int (Set - Signal)dt$$

As you can see from the above equation there is two terms, K_p and K_I. These are the two values you must adjust in the PI circuit to obtain a lock. By adjusting these values you can increase or decrease the size of the error signal. This error signal then adjusts our amplitude either increasing it or decreasing it allowing us to maintain amplitude. Tuning this can be somewhat challenging and requires a little bit of trial and error.

A good starting value is set K_p to 0 and set K_I to something fairly small like 1 or 2. Now if we run the simulation for around 1.5 second (machine.Wait(1.5)) and then plot the first and fourth column of debug.dat. You should see the amplitude slowly increase to the Az value over the course of a second or so. If not try adjusting the K_I value until you the amplitude is constant at Az.

As you can probably guessed this is far to slow to be useful in an AFM simulation. So now try adjusting the K_p and K_I values and you should find it it obtains constant amplitude much quicker. Try adjusting each of these values until Az is reached in around 0.4 seconds. Generally K_I values are around 100 times more than K_p.

We can now execute the script by entering python MainzTut.py into the terminal.

TASK

Set K_p to 0
Set K_I to 1
Run the simulation
Observe the amplitude, if it doesn't reach Az adjust K_I
Adjust the value of K_p and K_I to obtain a faster constant amplitude
Once you are happy with the result remove the machine.Wait command

Step 3.3- Phase Locked Loop

In order to measure the frequency shift of the cantilever we must use a Phase Locked Loop (PLL). The PLL contains two inputs, one for our signal and one for a reference signal that are out of phase by $\pi/2+\phi$. Our PLL works by multiplying these two signals, this generates the following trigonometric identity:

$$sin(\theta+\phi)\, cos(\theta) = 0.5\, (sin(2\theta+\phi) + sin(\phi))$$

If we passing this function through a low pas filter we can remove the high frequency component resulting in :

$$0.5\ sin(\phi)$$

Since we can approximate small values of sin as a linear function we can obtain $\phi/2$ from the above expression. Since the PLL is a feedback loop the reference signals frequency is then adjusted until our ϕ value is minimised. Hence we can obtain the frequency shift required to make these waves in phase and hence the df of the signal.

So now we have several parameters we must change in order to obtain a lock. We must choose our filter chain in order to remove our high frequency component. Hence it has to have a cut off less than 2 times the resonance frequency. Generally since our df << f0 it is good practice to set the values of the filters to quite a bit lower than 2*f0. In order to add multiple filters to chain set filters = [fcut1,fcut2, … ,fcutN].

Next is our gain which helps speed up a lock with the pll. Although if the gain value is to high the signal may be extra noisy or may not lock at all. Gain values are usually in the order of a few 100s.

The K_p and K_I work in the exact same way as with the PI (The PLL contains a PI circuit). So try setting K_p to 1 and adjusting K_I in order to obtain a lock. Try a initial value of 1 for your K_I.

Now we have to test it and adjust, which as with the AGC circuit requires a little bit of trial and error. A few steps back we uncommented this line machine.Connect('testwave.cos','pll.signal1'). This is because we wanted to attach a test wave to our PLL for tuning purposes. If you scroll down to the bottom of the input script you can see a section named "PLL testing commands". Uncomment out this portion (remove both the '"). What these commands do is adjust the frequency of our test wave and

You probably see that your PLL locks onto a frequency or maybe doesn't even lock on to the high frequency shifts but it takes in order of a few seconds which is far to slow for AFM. So try adjusting the K_p in order to obtain a better lock.

We can now execute the script by entering python MainzTut.py into the terminal. You can observe the df by plotting the 1st and 6th column of debug.dat.

Keeping altering these parameters till you can obtain a lock for the 3 frequency shifts within 0.1 - 0.2s.

Once you are happy it has locked on reconnect the cantilever to the PLL by commenting out machine.Connect('testwave.cos','pll.signal1') and uncommenting machine.Connect('amp.norm','pll.signal1')

Now that everything is tuned run the simulation for a half a second or so (machine.Wait(0.5)) and observe the df and amplitude. Our amplitude should still be constant and our df should be around 0 although it is worth noting that the cantilever will introduce noise into our PLL signal so our df will most likely lock within a few 10s of Hz of the 0 mark. Although this df offset is very heavily influenced by the time step so if your time step is around 10 microseconds expect a offset of 100s of Hz. This will cause an offset of 300 hz in your images although the contrast won't change. Generally in AFM sims we use a time step of 10ns although it would take around a hour to produce

an image. Hence in this tutorial i suggest you maintain a larger time step in order to produce an image much faster despite the offset.

TASK

Set the filter chain as described above
Set K_p to 1
Set K_I to 1
Run the simulation
Observe the df
Adjust the value of each parameter to obtain a faster constant lock
Once you are happy that it locks on fast enough comment out the PLL testing lines again
UnComment out machine.Connect('amp.norm','pll.signal1')
Comment machine.Connect('testwave.cos','pll.signal1')
Run for 0.5 seconds to test everything is working as we expect

Step 4 - Script the cantilever and obtain an image

So now we have everything working we need to actually scan our cantilever. We have already placed the cantilever at at position 0,0,1.5, which is 1.5 nm above the surface just now. Before we start a big scan let suppress the debug output so we don't slow down a sim and end up a massive file. At the top of section 4 add the following line:

```
out1.Stop()
```

Near the bottom of the script you can see the "Scripting Commands" section. in this section enter the commands we will discuss in this step. First let the cantilever settle by running the simulation with the cantilever far from the surface for around half a second machine.Wait(0.5).

Now the cantilever is settled down, lets move the scanner down to closer to the surface by using the command scanner.Move(x=0,y=0,z=0). Where the z value must be adjusted to move the cantilever down by that value. Again let the cantilever settle for a little while by using machine.Wait(0.5).

Now lets get the scanner to automatically scan over the surface to obtain an image, using the following commands :

```
scanner.Recorder = imager
scanner.BlankLines = True
scanner.Resolution = [64,64]
scanner.ImageArea(3,3)
scanner.ScanArea()
```

We can now execute the script by entering python MainzTut.py into the terminal. This will take around a minute to finish depending on resolution and time step.

Now you can plot the image my running the Gnuplot script “Plot.plt” located within the same folder as the vafm is installed. You can run this script by clicking open at the top of the Gnuplot window and navigating to “plot.plt”.

Now you should see your df image, Can you find out what surface it is?

TASK

Suppress the debug output circuit
Let the cantilever settle far away from the surface
Move the cantilever down and let it settle
Set up the scanning circuit
Produce an image using the Gnuplot script
Can you guess what surface it is?

Step 5 - Challenge step , Static deflection

If you manage to get everything working with time to spare here is a bonus extra challenge. It is possible to measure the static deflection of the cantilever in the same manner as they do with the experiment. In experiments they use a low pass filter with a low cut off so that when the cantilever signal is put through it only outputs the DC part of the cantilevers oscillation.

Try and build this static deflection by adding a low pass filter to the simulation then observe how the static deflection changes through out the approach to the surface.

Take a look at the online manual for more information on how the low pass filters should be added to the simulation and don’t forget to register the low pass output channel in the output circuit, more information on how the output circuit works can be found in the manual.