The arts, media, and engineering behind MOCAP

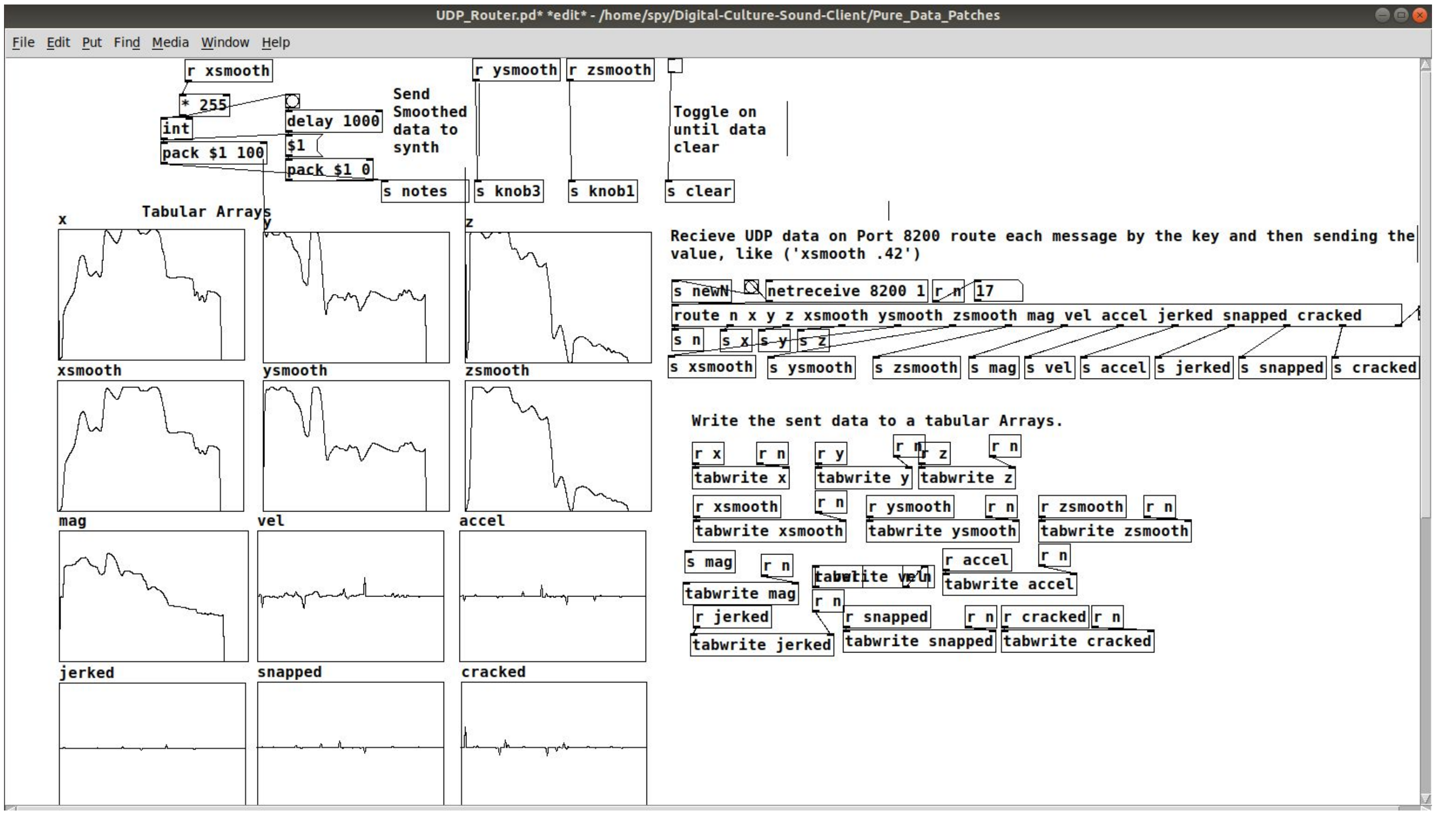
Introduction/Context:

With the handful of calculations and the amount of data produced, over time, from a motion capture system, this projects aim is to produce a system and visual interface that will mitigate this potentially overwhelming and difficult to understand subject. While brainstorm over ideas on what to do with MOCAP data, and at the same time creating a visual interface to plot the features produced from this system geographic analysis, a problem presented itself--and this project is the solution.

Motivations:

This project is first interested in extracting the basic physical features from the rigid bodies--magnitude, velocity, and amplitude. But, how do I know that the results of the calculations are correct? How do I understand the data enough to be able to map it to any control parameter? The answer is a visual interface that plots the data, and an information system that simplifies the extraction of these features.

The Visual Interface:



Send Smoothed data to synth

Toggle on until data clear

Recieve UDP data on Port 8200 route each message by the key and then sending the value, like ('xsmooth .42')

Write the sent data to a tabular Arrays.

With the MOCAP data being time series, its essential to provide some context to each individual data point--a graph of each amplitude, over time, achieves this context and allows for an understanding of the motion in amplitude. This understanding drives the decisions when making aesthetic choices about how to use these features. For example, I was looking for enough change in the amplitude so that it produced a variety of tones, and I wanted the change in decay rate to be subtle--so that was ideally mapped to a more slowly changing feature set.

The software:

Written in python, the motion capture data is processed and then streamed over UDP, to Pure Data. The geometry of the data is processed so it produces the feature data form the rigid bodies.

<u>The feature data that is produced includes:</u>

- Normalized x,y,z vectors
- Smoothed x,y,z vectors
- Magnitude
- Velocity
- Acceleration

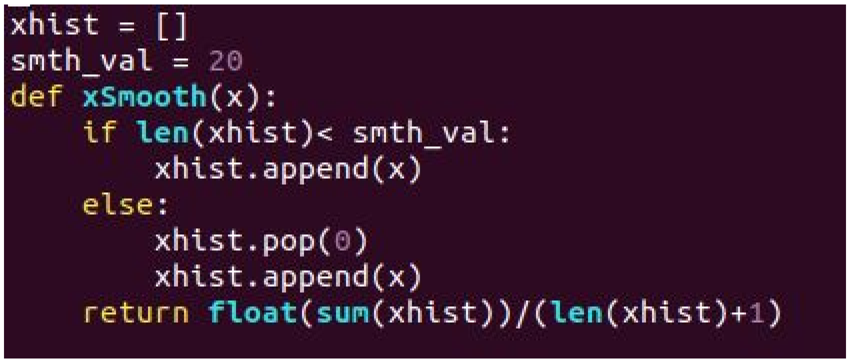
- Jerked, Snapped, Cracked

<u>The streamed UDP data is in this format:</u>

```
'n 16'
'time change:', 0.0
'x 0.0147695012951'
'y 0.97633833709'
'z 0.126799593613'
'xsmooth 0.0090075117708'
'ysmooth 0.920055816486'
'zsmooth 0.0656637245342'
'mag 0.984648578722'
'vel 0.00189304712224'
'accel 6.77379737011e-05'
'jerked 1.5677462081e-05'
'snapped -5.85875046111e-06'
'cracked -0.000932718641299'
```

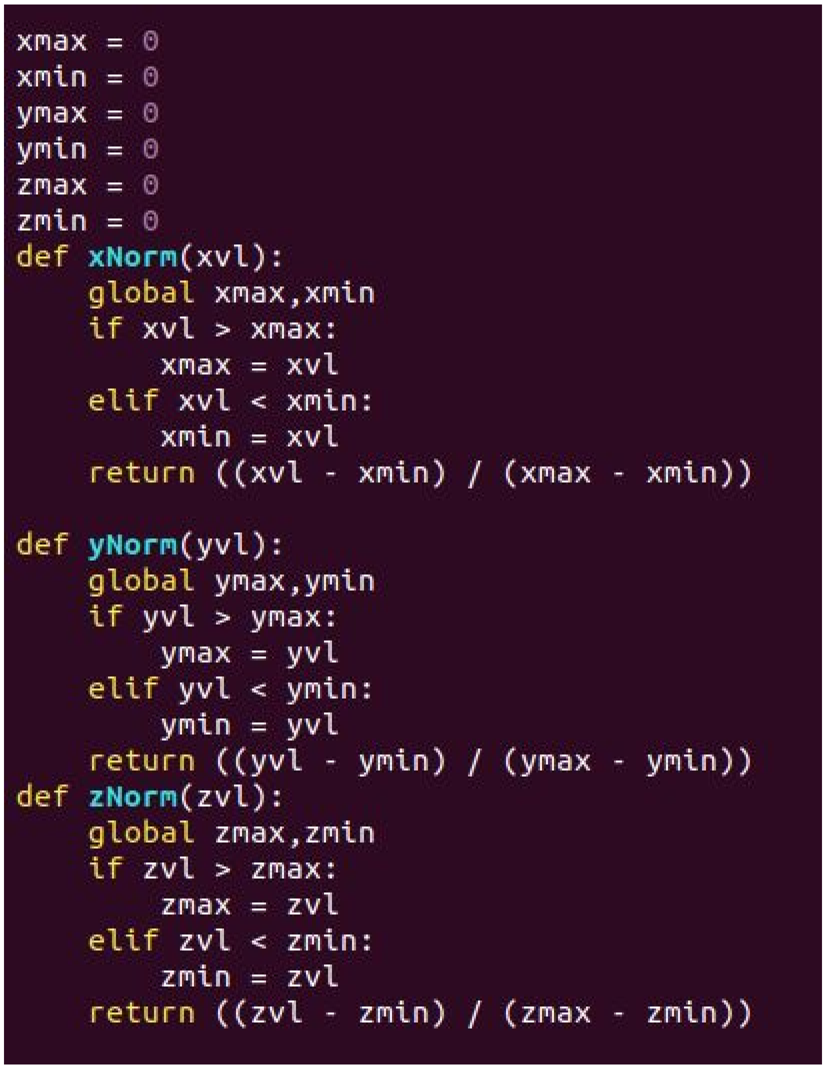
**Please see the attached python script to see this code.

Smoothing Algorithm: Note the pop and append of the array.



```
xhist = []
smth_val = 20
def xSmooth(x):
    if len(xhist)< smth_val:
        xhist.append(x)
    else:
        xhist.pop(0)
        xhist.append(x)
    return float(sum(xhist))/(len(xhist)+1)
```

Normalization: Note the self-regulating max and min. This give less control over the end value, but makes this better adaptable for longer sampling and with un-known max and mins.



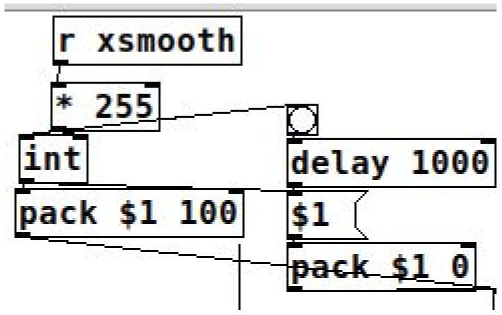
```python
xmax = 0
xmin = 0
ymax = 0
ymin = 0
zmax = 0
zmin = 0
def xNorm(xvl):
    global xmax,xmin
    if xvl > xmax:
        xmax = xvl
    elif xvl < xmin:
        xmin = xvl
    return ((xvl - xmin) / (xmax - xmin))

def yNorm(yvl):
    global ymax,ymin
    if yvl > ymax:
        ymax = yvl
    elif yvl < ymin:
        ymin = yvl
    return ((yvl - ymin) / (ymax - ymin))
def zNorm(zvl):
    global zmax,zmin
    if zvl > zmax:
        zmax = zvl
    elif zvl < zmin:
        zmin = zvl
    return ((zvl - zmin) / (zmax - zmin))
```

Now that the data is understood and the features properly extracted:

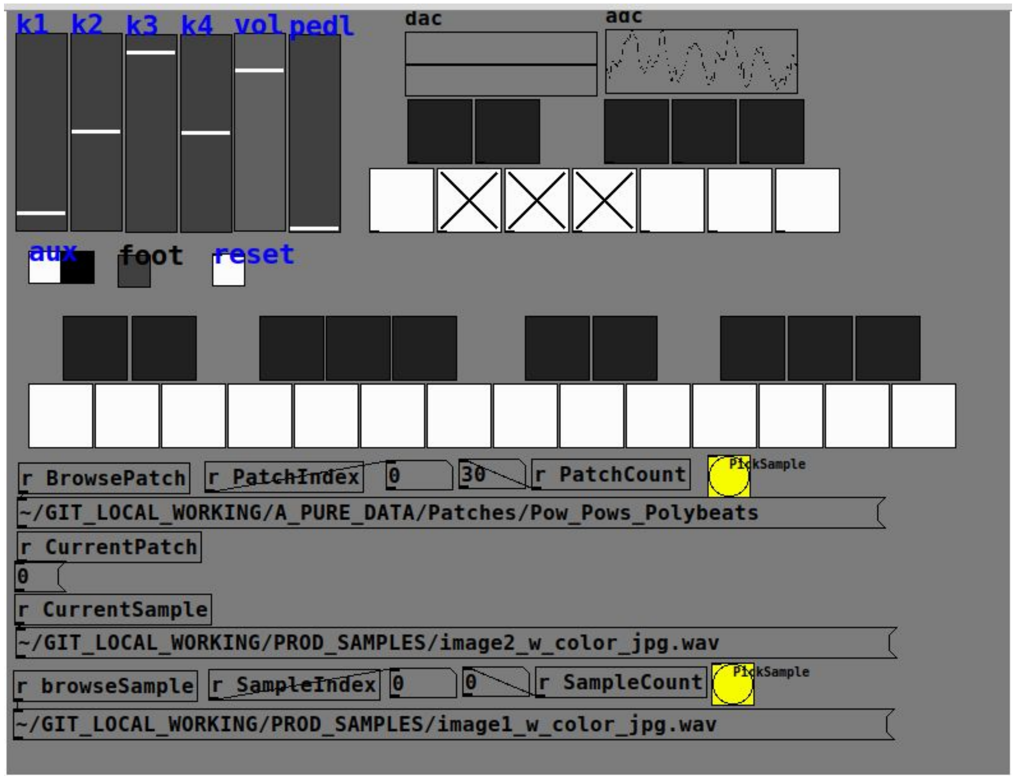
One problem with using time series data is how to determine where the trigger is. And, if we trigger something with this data, how do we de-activate the trigger?

Here, I use a delay object. The time series data is made into 2 note values, one for turning on the note--an amplitude of 100--and another note for turning off the note--an amplitude of 0. Reflecting on this now, I see that instead of using static velocity of the notes I could parameterize the amplitude of the note at its strike--rather than modulating the synthesizer at more of a global scale.

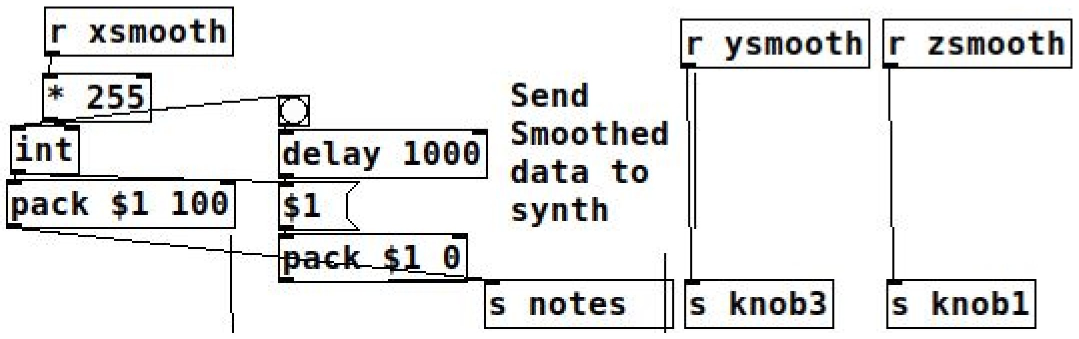
This loads 2 notes, one with 100 amplitude and another with 0 amplitude, with a sustain of 1000 milliseconds.



Feedback


k1
k2
k3
k4
vol
pedl
dac
adc
aux
foot
reset
r BrowsePatch
r PatchIndex
0
30
r PatchCount
PickSample
~/GIT_LOCAL_WORKING/A_PURE_DATA/Patches/Pow_Pows_Polybeats
r CurrentPatch
0
r CurrentSample
~/GIT_LOCAL_WORKING/PROD_SAMPLES/image2_w_color_jpg.wav
r browseSample
r SampleIndex
0
0
r SampleCount
PickSample
~/GIT_LOCAL_WORKING/PROD_SAMPLES/image1_w_color_jpg.wav

As a way to more easily use the synthesizers I produce, I follow the frame work of the Critter & Guitari Organelle, here is their product page: https://www.critterandguitari.com/organelle . I have been working in this frame work for over a year and produced and adapted a variety of synthesizers. A benefit for working in this framework is the 4 knobs that are on the synthesizer. These are easily accessible through the 'send' and 'recieve' object. And this is how I accomplish my mapping of features to synthesizer parameters, like this:



```
r xsmooth
* 255
int
pack $1 100
delay 1000
$1
pack $1 0
s notes
r ysmooth
s knob3
r zsmooth
s knob1
```

Final Thoughts:

Its interesting to be the only student in this class that is graduating; I'm the only student in the class that has taken all the required classes and have built all the required projects, and more. And here, I decide to build something simple--and people enjoy the simplicity of the visual interface. Some times less is more.