ChordRunner: A Leap Motion Musical Loop Generator

A project made for the Cognitive Systems course in Jagiellonian University by Burak Çetin

Introduction

ChordRunner is a simple real-time loop generator with a separate melody layer and drum layer. The hand motion is captured via a Leap Motion and transformed into musical parameters that define the loop content.

Theory

The program outputs 2 layers of sound: a melody line which is generated by oscillators and a sample line which is generated by playing samples. These layers are run in sync by a timer that determines the time of each beat.

Melody Line

4 Fingers of the left hand controls the 4 notes that will be generated by a sine wave oscillator on each beat. A greater angle between the horizontal plane creates a higher pitched note in the major scale.

Sample Line

4 Fingers of the right hand controls the 4 samples that will be played on each beat. The angle between the horizontal plane selects which sample is played. In this version there are 2 samples: a percussion hit and a snare drum hit. Each hit is either silent, a percussion hit, a snare hit or percussion and snare hits at the same time.

Scale Control

The thumb of the left hand controls the tonic of the major scale that the notes should be played on. The angle between the vertical plane and the thumb is used to select between tonic options. These options are sorted in the circle of fifths order. This

decision is made to stop the user from playing dissonant notes when they change the tonic.

Beat Speed

The thumb of the right hand controls how fast the sample line is running in with respect to the melody line. The angle between the vertical plane and the thumb is used to select between speed options. With this control the user can change the sample line being 4 times slower than the melody line to being 4 times faster. This control is discrete with changes only happening as multiples of 2.

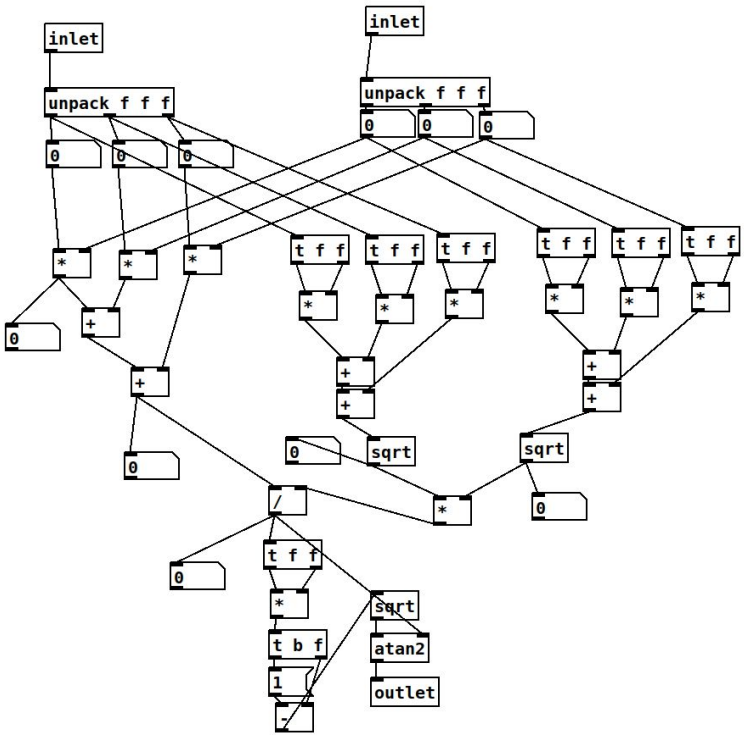
All of these controls are happening at the same time whıle someone is using ChordRunner. There are also options implemented to turn off beat speed controls or scale controls together or separately.

Implementation

The project was realized in Pure Data. Pure data is an open source visual programming language for multimedia. In order to get information from the Leap Motion device a Pure Data external called leapmotion by Chikashi Miyama was used.
There are 4 pd files in the project:

- vec3angle.pd
- angle-to-disc.pd
- miditofreq.pd
- ChordRunner.pd

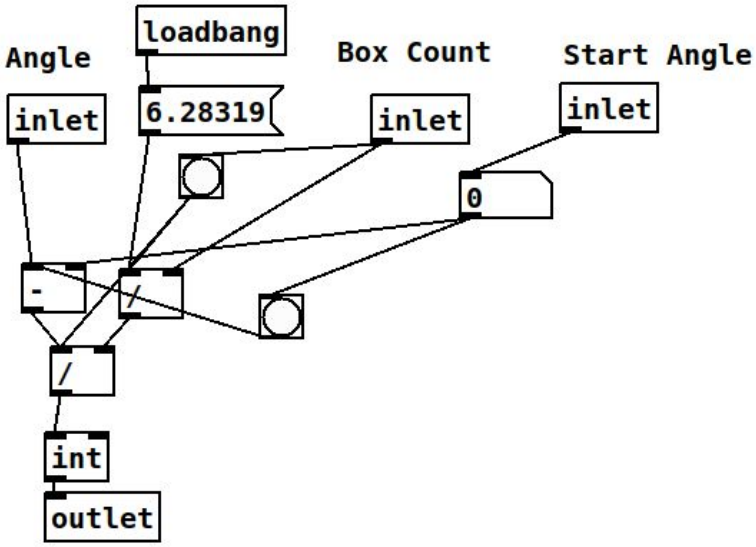
vec3angle.pd



A screenshot of vector-angle3.pd

This abstraction takes two 3d vectors from its inlets and sends the angle between the vectors to its outlet. It uses the cosine formula but since Pd Vanilla only has arctangent as an inverse trigonometric formula. This abstraction calculates the arccosine by utilising the sine squared plus cosine squared identity.

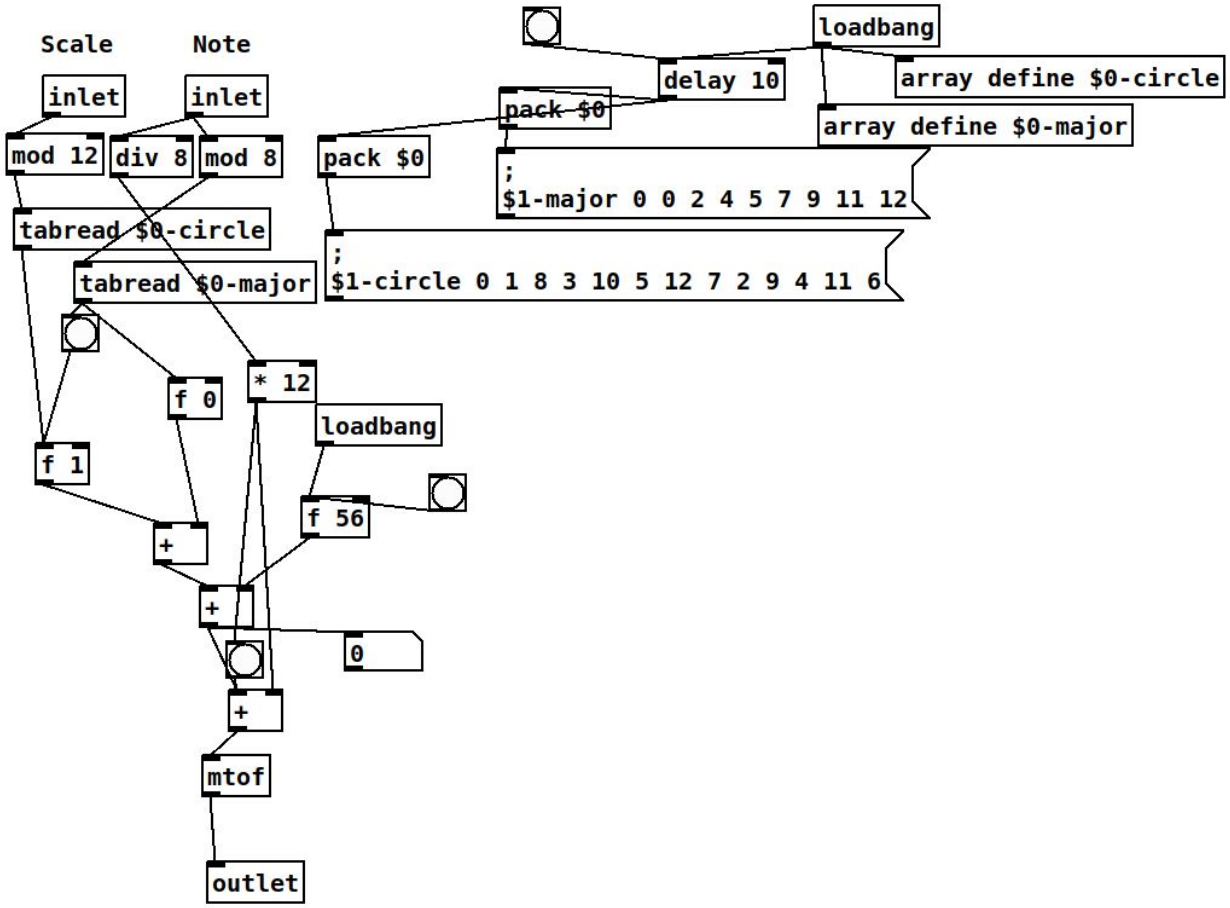
angle-to-disc.pd



A screenshot of angle-to-disc.pd

This abstraction takes in an angle, the number of desired discrete intervals in one full rotation and another angle as the zero angle. It outputs the index of the interval that the given angle would fall into. The math is really simple because it just divides the number with the size of each interval.

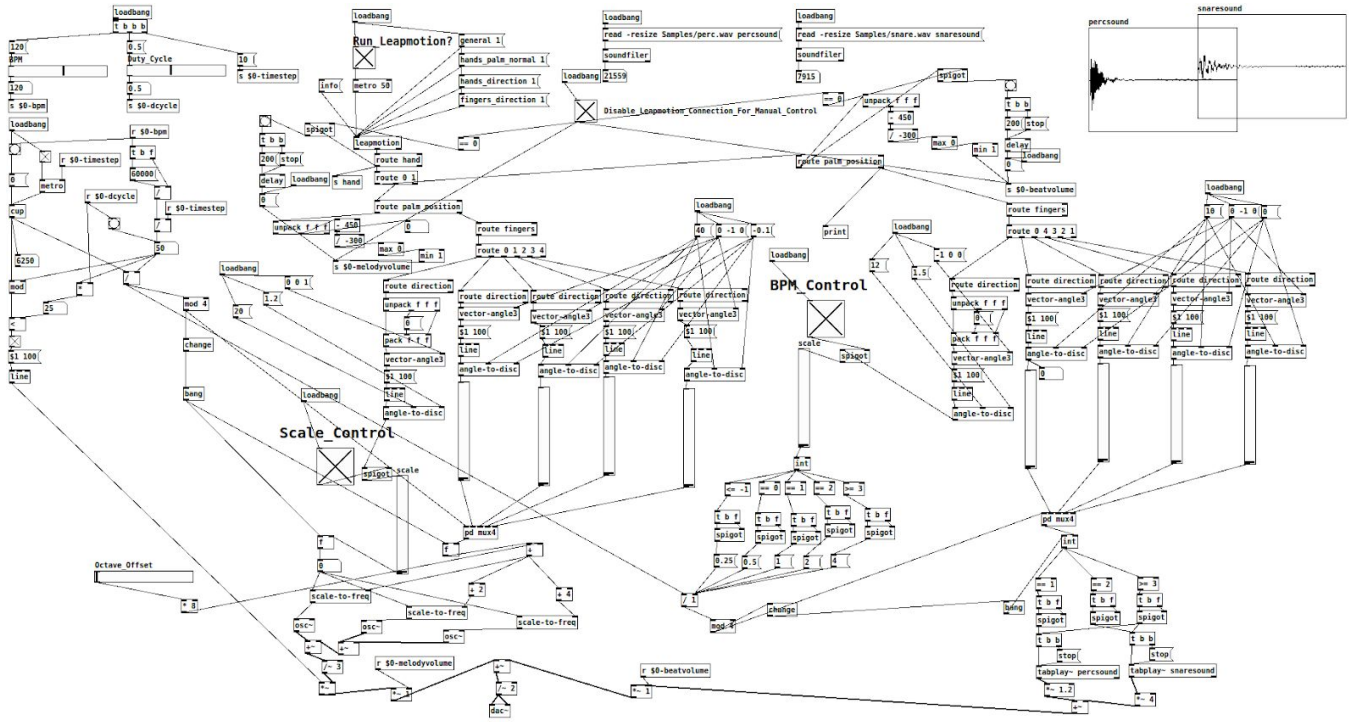
chordtofreq.pd



A screenshot of chordtofreq.pd

This abstraction takes in two integers which determines the tonic of the major scale you want to play on and which note you want to play on that scale. In western music we have 12 semitones in a single octave. Each of these semitones can be picked as the tonic of the scale. As a result the information about the tonic is processed in modulo 12. When the tonic input is increased by one instead of going a semitone higher the system is following the circle of fifths.This is done so that small differences in the tonic input do not cause dissonant note changes. The information about circle of fifths is stored in the array *$0-circle*. The note input is simply used to iterate over the 7 notes in the major scale. The differences (measured in semitones) between a note and the tonic in a major scale is stored in the array *$0-major*. At the end the index of the note that we are trying to output is calculated. Then this is converted to actual sound frequency by utilising the mtof object provided by Pure Data which converts midi keyboard button indexes to their frequencies.

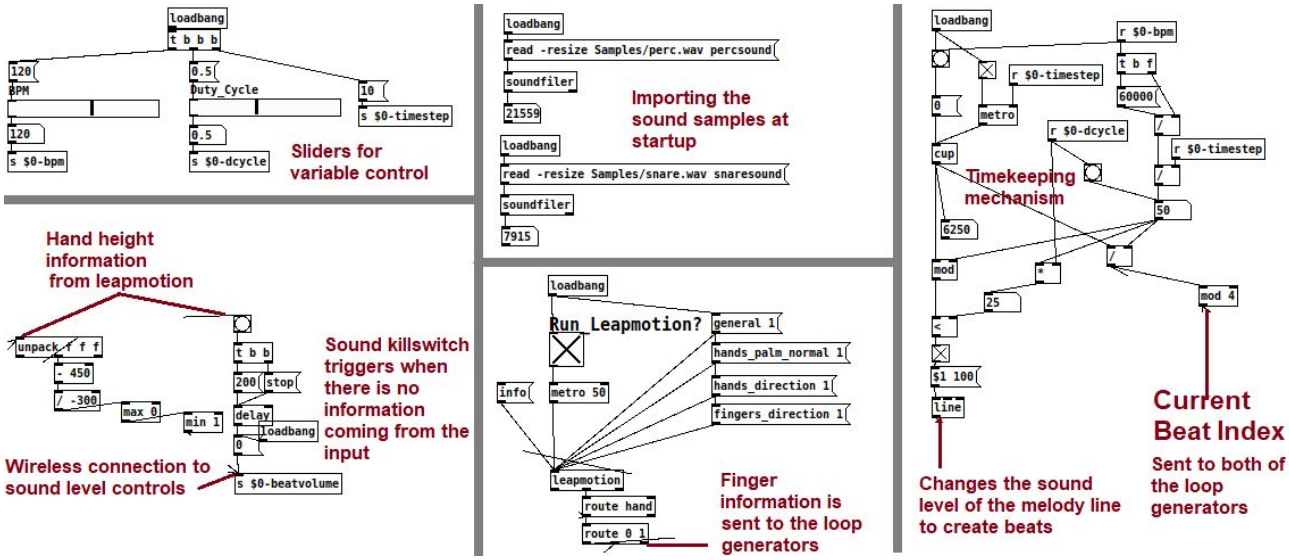
ChordRunner.pd



A screenshot of chordtofreq.pd

This is the main code of ChordRunner. Since the code is pretty complicated on a general glance I will go over some of the relevant small parts of the code. Then I will explain the main loop generation system. In these diagrams I edited out some of the more technical parts of the code from the image for clarity.

Miscellaneous Important Parts



A diagram explaining some parts of chordtofreq.pd

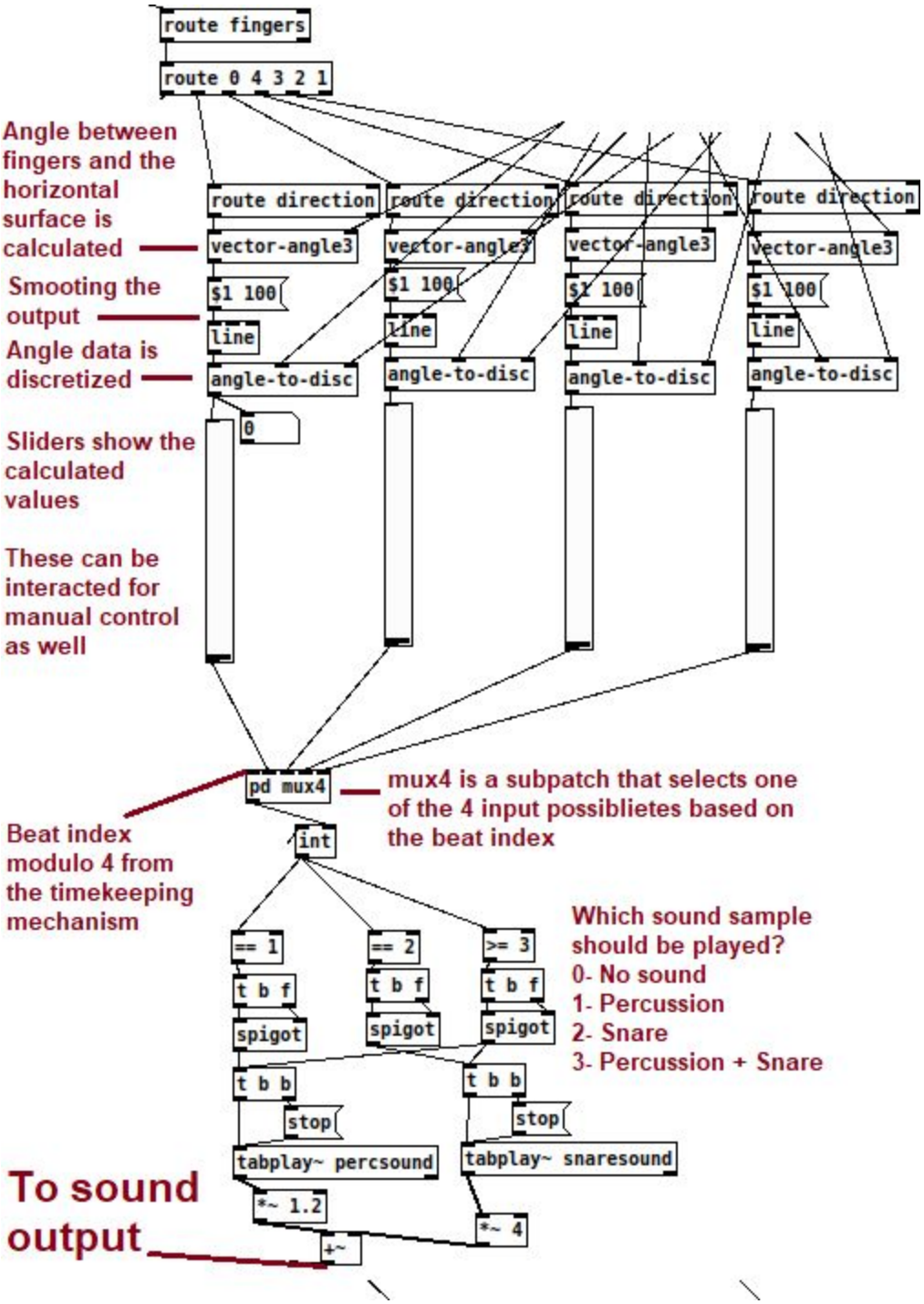
On the top left corner we have the global parameters which can be changed in real time. BPM determines the number of beats in the melody line. Duty cycle determines how long does a beat take in relation to the time difference between beats.

On the bottom left corner we have the sound level control which is based on hand height. Also the killswitch that turns off the sound that a hand controls if the hand is not detected by the device anymore.

Middle diagrams show the initializations of sample sounds and the *leapmotion* external.

The diagram on the right is the main timekeeping mechanism. It counts up by using the *cup* object in each time step. Then this number is divided by the time difference between two beats to calculate the current beat index. Then the modulo 4 of this number is taken to determine which beat should be played by the loop generators.

Loop Generator



A diagram explaining the loop generation for samples

The loop generator for the sample playing part of the code is shown above. There are initial values that I calculated for the finger angle calculations in the code that I omitted

here for clarity. The melody line loop generator follows the same structure. It only uses oscillators for sound generation instead of playing samples. As a result it will not explained as a separate entity in this report.

Setup and Execution

Pd patches in this project and the sample sounds can be found in the Github repository of this project.
Project was realized in an Ubuntu operating system with the linux version of the *leapmotion* external. I am aware of a windows version as well. I did not test the code on windows but it does not contain any linux exclusive content so it should work on windows as well.

For the linux version you need Pure Data. You can install it using *apt-get* or compiling it from the source code following the instructions here.
Afterwards you need to compile the *leapmotion* external. For this you need the LeapSDK from the Leap Motion website. You also need to install *flext*. For all of these steps you can follow this tutorial.
After making sure that Pure Data, LeapSDK and *leapmotion* external working you can open the *ChordRunner.pd* patch on Pure Data.
Note: If you are using the vanilla distribution of Pd you will need to install the *cup* external through the *Find Externals* menu inside Pure Data.

Evaluation Tasks

For evaluating and seeing the later improvements in this project I designed a simple evaluation system. A person unfamiliar with ChordRunner is given a basic control explanation infographic and 5 to 10 minutes of time to experiment with it.

Then they are asked to play some predetermined melody and sample lines made with ChordRunner that will be played to them. There will be 6 tasks:

1. A 4 beat sample line.
2. A 4 beat melody line.
3. An 8 beat sample line with changing beatspeed.
4. An 8 beat melody line with changing tonic.
5. Tasks 1 and 2 simultaneously.
6. Tasks 3 and 4 simultaneously.

On each task the amount of time spent to achieve the desired loop is recorded.
After doing this experiment with a control group on the initial version, the intuitiveness of the future versions of the system can be compared to earlier versions to evaluate success.

Possible Improvements

1. ChordRunner.pd needs a better UI that do not show any of the complicated inner workings for ease of use.
2. The patches themselves can be commented for clarity for future developments.
3. Finger data discretization can be improved by extra fine tuning.
4. A noise cancellation system can be implemented to the data collected from the Leap Motion device.
5. More instrument and sample options that are easily customizable.

Results and Discussion

To see ChordRunner in action you can watch this short demo video I made.

The inspiration for this project was firstly Reactable then similar Leap Motion based music projects I saw on the internet.
As I am implementing this project I learned how compile the source code of a program I want to use in a Linux operating system. Then I learned how to use Pure Data for creating sounds and playing samples. I also gained experience in using Pure Data for complex controls. I believe this project added to me as a person a lot. I think I will continue developing patches in Pure Data for future sound engineering or hobby projects.