Zhuowen (Charles) Lin

Electrical Engineer, Music Lover, Analog Enthusiast

I am a graduate student pursuing my Master of Science degree in Georgia Institute of Technology in Atlanta, United States. I graduated from Southern University of Science and Technology in Shenzhen, China in July 2019 with a Bachelor of Engineering in Electrical Engineering. Within my degree programs, I took classes focused on digital signal processing, audio engineering, circuit design, and acoustics.

During my university life, I have had the opportunity to work on numerous interesting projects, a few of which are showcased in this portfolio.

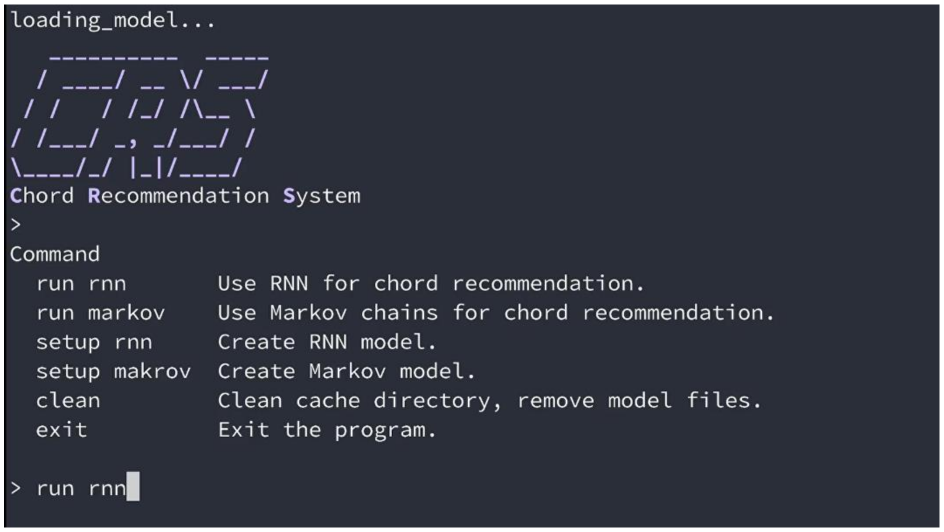
zlin343@gatech.edu

(+1)404-358-6784

Me graduating as an undergrad :)

Automatic Chord Recognition and Recommendation | Computational Music Analysis Project 2019

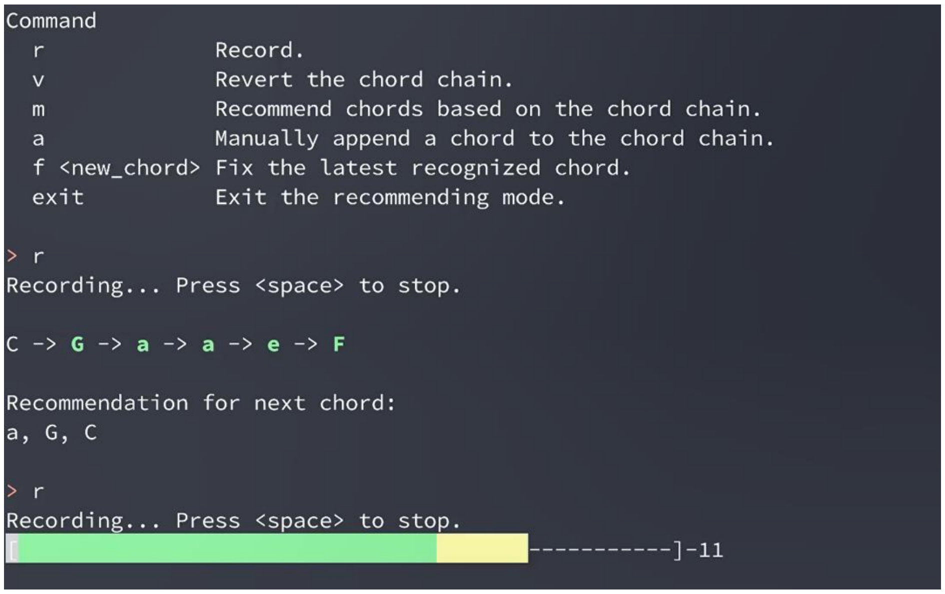
My team and I designed and built a python-based functional tool that can be used to fulfill guitarists' need. This tool can record multiple chords played by guitar player, recognize the played chords, and give recommendations to what chords to be played next that are more musically favored. We also provided functions of reverting the chord chain, appending a chord to the existed chain manually, and change the latest recognized chord manually, to handle different situations like the tool gave incorrect chord recognition results.



```
loading_model...
   __________  _____
  / ____/ __ \/ ___/
 / /   / /_/ /\__ \
/ /___/ _, _/___/ /
\____/_/ |_|/____/
Chord Recommendation System
>
Command
  run rnn       Use RNN for chord recommendation.
  run markov    Use Markov chains for chord recommendation.
  setup rnn     Create RNN model.
  setup makrov  Create Markov model.
  clean         Clean cache directory, remove model files.
  exit          Exit the program.

> run rnn
```

I was in charge of the chord recognition part.

Onset Detection



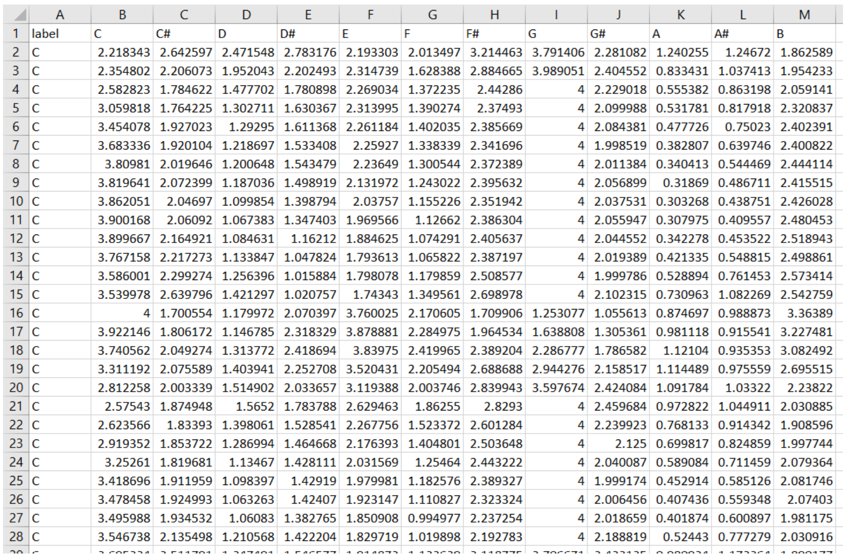
```
Command
  r              Record.
  v              Revert the chord chain.
  m              Recommend chords based on the chord chain.
  a              Manually append a chord to the chord chain.
  f <new_chord>  Fix the latest recognized chord.
  exit           Exit the recommending mode.

> r
Recording... Press <space> to stop.

C -> G -> a -> a -> e -> F

Recommendation for next chord:
a, G, C

> r
Recording... Press <space> to stop.
[                                          -----------]-11
```

We wanted to detect multiple chords played, so onset detection was an essential part to slice the raw input audio file into several pieces. In order to detect the onset, namely the starting time, of each chord, I first computed the short-time root mean square (RMS) of the input raw audio file. This step can be comprehended as using an RMS energy to represent all samples in a certain frame. Then, I took dB scale to the short-time RMS and lowpass filtered it. The last step was to calculate the difference of filtered RMS between neighboring frames. When the difference was larger than a certain threshold, we could say that there was an energy jump between these neighboring frames and thus there was an onset.

Chromagram

Chromagram was the audio feature I used for chord recognition. Chromagram can be regarded as the frequency spectrum on specific frequencies that correspond to the 12 semitones in an octave. I used constant Q transform (CQT) to compute chromagram but not short-time Fourier transform (STFT) as in most DSP applications. This is because CQT can be interpreted as computing a DFT only for specific, logarithmically spaced, frequency bins. One positive side effect of CQT is increasing the frequency resolution in low frequency range. The Q parameter in CQT can control frequency resolution per octave.

Dataset



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	label	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
2	C	2.218343	2.642597	2.471548	2.783176	2.193303	2.013497	3.214463	3.791406	2.281082	1.240255	1.24672	1.862589
3	C	2.354802	2.206073	1.952043	2.202493	2.314739	1.628388	2.884665	3.989051	2.404552	0.833431	1.037413	1.954233
4	C	2.582823	1.784622	1.477702	1.780898	2.269034	1.372235	2.44286	4	2.229018	0.555382	0.863198	2.059141
5	C	3.059818	1.764225	1.302711	1.630367	2.313995	1.390274	2.37493	4	2.099988	0.531781	0.817918	2.320837
6	C	3.454078	1.927023	1.29295	1.611368	2.261184	1.402035	2.385669	4	2.084381	0.477726	0.75023	2.402391
7	C	3.683336	1.920104	1.218697	1.533408	2.25927	1.338339	2.341696	4	1.998519	0.382807	0.639746	2.400822
8	C	3.80981	2.019646	1.200648	1.543479	2.23649	1.300544	2.372389	4	2.011384	0.340413	0.544469	2.444114
9	C	3.819641	2.072399	1.187036	1.498919	2.131972	1.243022	2.395632	4	2.056899	0.31869	0.486711	2.415515
10	C	3.862051	2.04697	1.099854	1.398794	2.03757	1.155226	2.351942	4	2.037531	0.303268	0.438751	2.426028
11	C	3.900168	2.06092	1.067383	1.347403	1.969566	1.12662	2.386304	4	2.055947	0.307975	0.409557	2.480453
12	C	3.899667	2.164921	1.084631	1.16212	1.884625	1.074291	2.405637	4	2.044552	0.342278	0.453522	2.518943
13	C	3.767158	2.217273	1.133847	1.047824	1.793613	1.065822	2.387197	4	2.019389	0.421335	0.548815	2.498861
14	C	3.586001	2.299274	1.256396	1.015884	1.798078	1.179859	2.508577	4	1.999786	0.528894	0.761453	2.573414
15	C	3.539978	2.639796	1.421297	1.020757	1.74343	1.349561	2.698978	4	2.102315	0.730963	1.082269	2.542759
16	C	4	1.700554	1.179972	2.070397	3.760025	2.170605	1.709906	1.253077	1.055613	0.874697	0.988873	3.36389
17	C	3.922146	1.806172	1.146785	2.318329	3.878881	2.284975	1.964534	1.638808	1.305361	0.981118	0.915541	3.227481
18	C	3.740562	2.049274	1.313772	2.418694	3.83975	2.419965	2.389204	2.286777	1.786582	1.12104	0.935353	3.082492
19	C	3.311192	2.075589	1.403941	2.252708	3.520431	2.205494	2.688688	2.944276	2.158517	1.114489	0.975559	2.695515
20	C	2.812258	2.003339	1.514902	2.033657	3.119388	2.003746	2.839943	3.597674	2.424084	1.091784	1.03322	2.23822
21	C	2.57543	1.874948	1.5652	1.783788	2.629463	1.86255	2.8293	4	2.459684	0.972822	1.044911	2.030885
22	C	2.623566	1.83393	1.398061	1.528541	2.267756	1.523372	2.601284	4	2.239923	0.768133	0.914342	1.908596
23	C	2.919352	1.853722	1.286994	1.464668	2.176393	1.404801	2.503648	4	2.125	0.699817	0.824859	1.997744
24	C	3.25261	1.819681	1.13467	1.428111	2.031569	1.25464	2.443222	4	2.040087	0.589084	0.711459	2.079364
25	C	3.418696	1.911959	1.098397	1.42919	1.979981	1.182576	2.389327	4	1.999174	0.452914	0.585126	2.081746
26	C	3.478458	1.924993	1.063263	1.42407	1.923147	1.110827	2.323324	4	2.006456	0.407436	0.559348	2.07403
27	C	3.495988	1.934532	1.06083	1.382765	1.850908	0.994977	2.237254	4	2.018659	0.401874	0.600897	1.981175
28	C	3.546738	2.135498	1.210568	1.422204	1.829719	1.019898	2.192783	4	2.188819	0.52443	0.777279	2.030916

With a program that could compute chromagram feature, we could build a dataset by ourselves. My team member and I played each major chord and minor chord for more than 40 times with guitar, extracted their chromagram, then formed our dataset with chord names as labels and chromagram as features. I trained and tested three classifiers including kNN, SVM and Gaussian Naive Bayes with our dataset, all had more than 92% accuracy. With a trained classifier model, the whole process of chord recognition would be: 1. record chord sequences, 2. slice the audio file with onset detection, 3. extract chromagram for each slice, 4. classify to chord names with classifier.

	kNN	Gaussian NB	SVM
Accuracy	0.9819	0.9294	0.9864
Time spent	0.3084	0.0215	0.0290

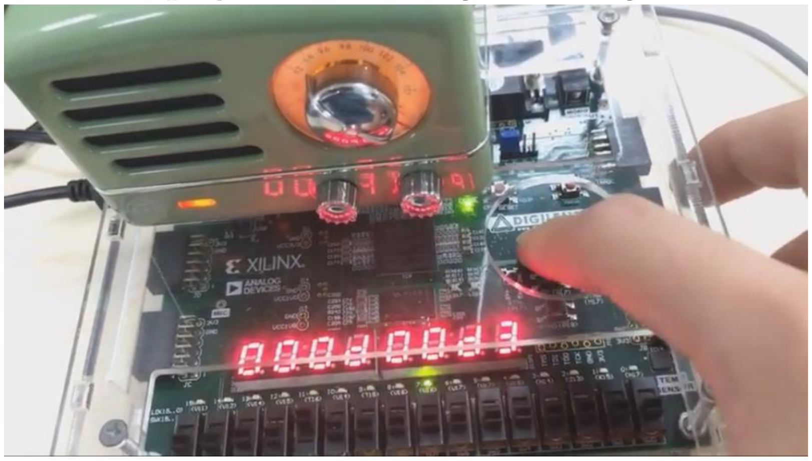
Chord recommendation

The chord progression could be modeled as a Markov process, where the transitions from state to state are based on the observations of limited numbers of previous states. We built a Markov chain model by extracting the chord-to-chord transitions from the McGill Billboard dataset and obtain the transition probabilities represented by a multi-dimensional matrix.

16-Step FPGA Music Sequencer and Synthesizer | Digital System Design Project

2018

Music sequencer is a device that can edit, record, and play back music notes for a certain period. For example, for a 16-step sequencer, we can edit the music notes at each step, record our editing, and then the sequencer would play the 16 notes again and again.



In this project, my teammate and I made a hardware music sequencer on Nexys 4 DDR FPGA board using VHDL as programming language. I was responsible for the sequencer circuit, implementing functions of counting step progression, music notes selection and memorization, and a "play & pause" button.

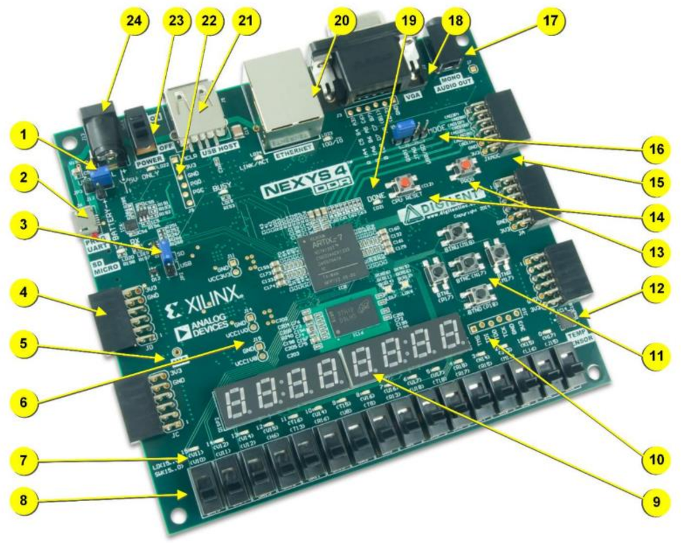


Figure 1. Nexys4 DDR board features.

The picture beside is the features and layout of the board.

Counting 16-step Progression

Our music sequencer used No.8 switches as the main control. There were 16 switches in total, with each one represented a music note or a beat, so 16 switches corresponded to the 16 steps in the sequencer. Turning a certain switch up meant that the corresponding step was activated, then the music note on this step could be edited.

No.7 are LEDs. It showed the beat that was currently playing. If there was a sound on the current step, the corresponding LED would be on.

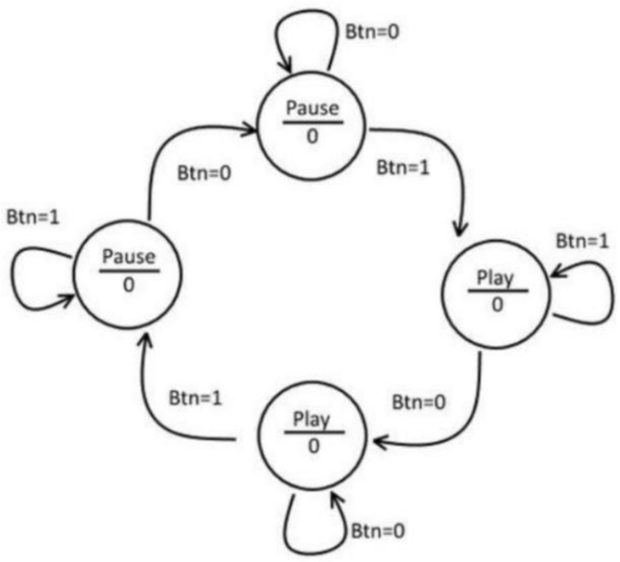
The 16-step progression was implemented with a 4-bit T flip-flop. The 4-bit T flip-flop could count from 0 to 15 and then came back to 0, so I used it as a 4-bit counter to keep track of the step progression. 16 switches corresponded to the 16 values in 4-bit counter, so the speaker output could know which switch was on and played out its music note.

Selecting And Memorizing Music Notes

No. 11 the five buttons were used to edit the beats. Every beat had 16 available pitches preset in the program, selected by pressing the up button. After selection, we could press right button to save the pitch. I used 16 D flip-flops to remember the chosen pitches for each step respectively. Same as counting the progression, I used a 4-bit T flip flop to store the 16 preset pitches in the circuit. They were in a pentatonic scale, with three octaves of C, D, E, G, A, and a C in the fourth octave.

Play & Pause Button

The center button of No.11 was utilized as a play & pause button. Since we could not press and release the button so rapidly that only one trigger was given to the button under the hardware clock signal, the sequencer would switch between play and pause. In order to overcome this problem, I designed and implemented a finite state machine into the circuit, so that I could toggle between play and pause states in finite state machine exactly once when the button was pressed.



Synthesizer circuit

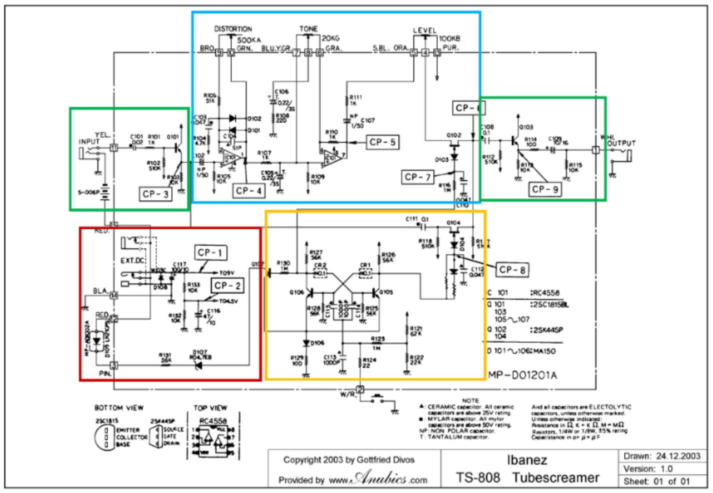
The Nexys 4 DDR FPGA board could only generate square wave, which was not as musically favored as sine wave, so we converted square wave to sine wave using pulse width modulation.

Production of Guitar Distortion Effector Ibanez TS808 | Analog Circuit Project

2017

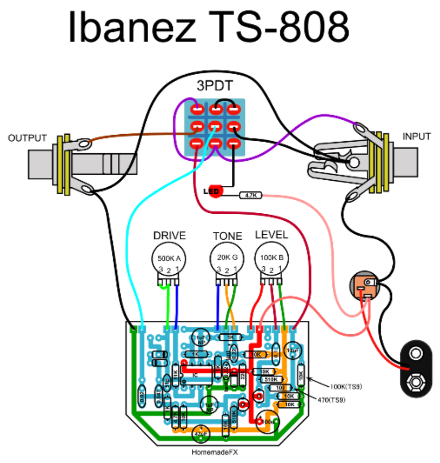
Ibanez Tube Screamer 808 (TS808) was a vintage analog guitar distortion effector that was famous among guitarists. In this project, my teammate and I analyzed, tested and finally built a TS808 effector from scratch that could accomplish all the functions of the "Legendary TS808" including distortion effect, volume and tone adjustment.

PCB layout design



Our work started from a circuit diagram of TS808. The circuit could be divided into 4 parts: the red power supply, the blue sound effect circuit, the green IO buffer, and the orange switching circuit. The core of this circuit was two clipping diodes that could clip the signal above certain level.

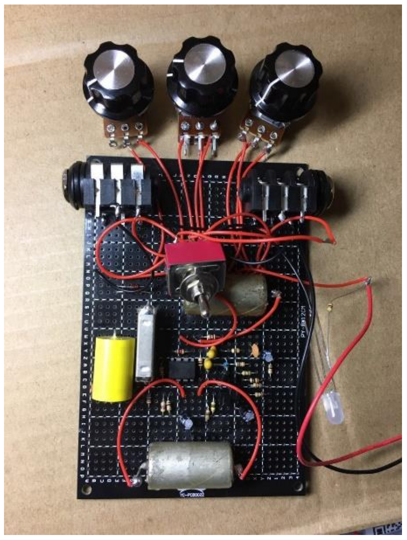
We designed a PCB layout from the circuit diagram. The color notation in this layout was the same as that in circuit diagram.



Notice that the switching circuit was accomplished by a 3PDT switch and the IO connectors are 6.5mm connectors compatible with guitar.

With the PCB layout in hand, we purchased most of the circuit components from local electronics supply stores. Some rare components that was used specifically in audio-related circuits were bought online.

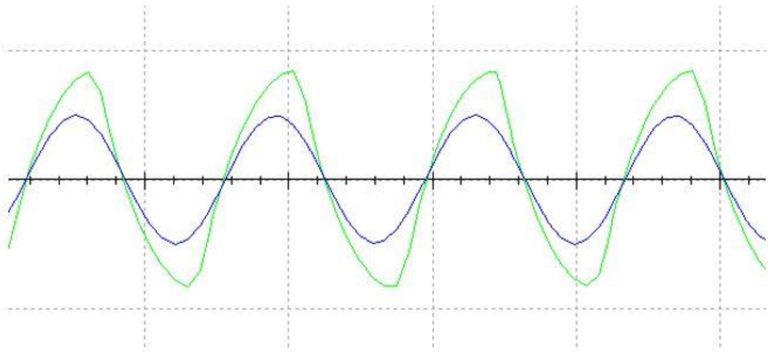
Soldering the circuit

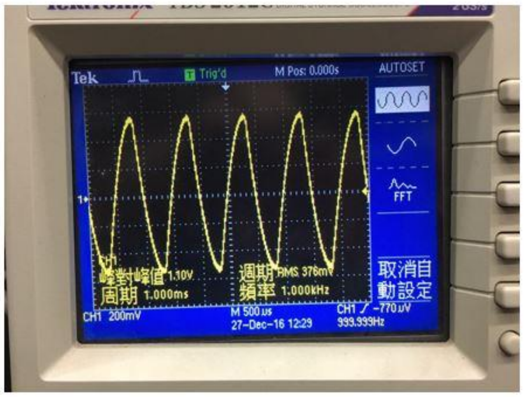


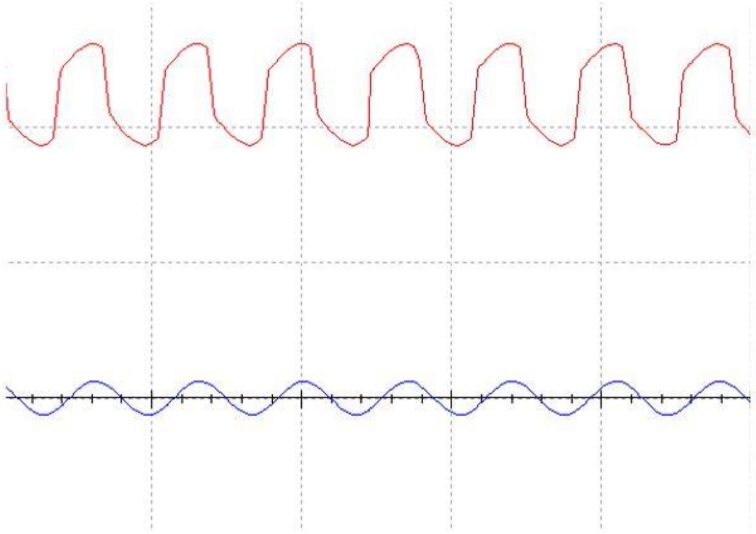
We soldered the whole circuit by ourselves. Some capacitors were too large to be fit on the PCB board, so we had to change our wiring tactic sometimes in practice.

Circuit debugging and testing

We built the same circuit on software Multisim to simulate the signals. After finishing soldering the circuit, we used an oscilloscope to check signals at certain testing points in our circuits to see whether they were consistent with the simulation. The output signal waves in our circuit were almost identical to the software simulation results. We also tested the circuit with a guitar and an amplifier, it worked just as the legendary rock & roll distortion sound.







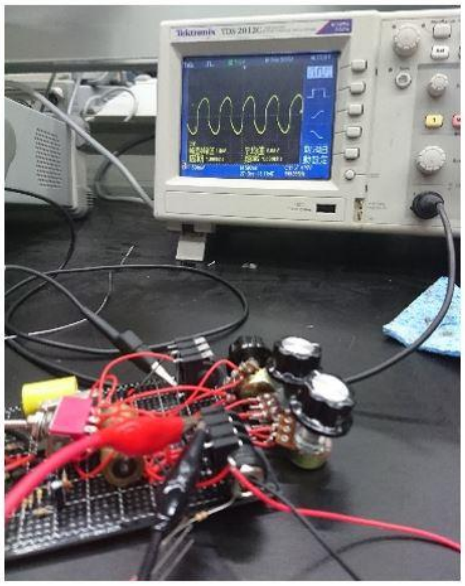


Image Haze Removal using Dark Channel Prior | Digital Image Processing Project

In hazy weather, outdoor images suffer from poor quality due to the insufficient luminance and noise brought by the atmospheric particles in haze. Haze removal, also known as dehazing, is considered as a significant process as clear, haze-free images are not only favored by human eyes, but also essential for improving the performance of computer vision systems.

In this project, I first implemented the dark channel prior (DCP) haze removal algorithm with MATLAB. Then I improved the algorithm by introducing a depth estimation to adaptively control the level of dehazing in DCP algorithm.

Dark Channel Prior

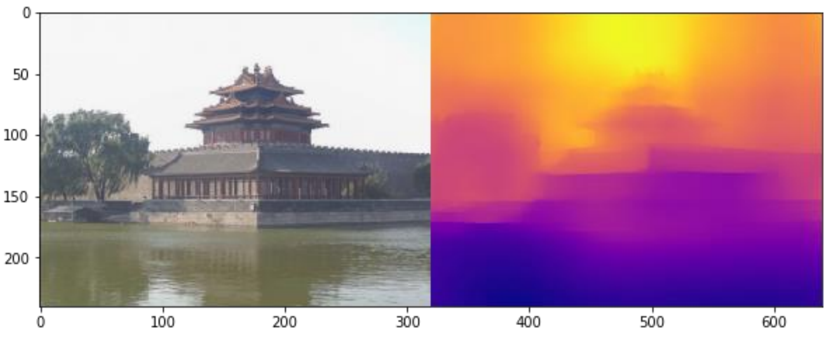
The dark channel is defined as the pixel whose intensity is close to zero for at least one of the color channels within a local image patch. For a haze-free image, the value of dark channel is close to 0 with a high probability, while for a hazy image, the value of dark channel is dramatically greater than 0. The above observation is called dark channel prior (DCP). Based on this observation, we can not only distinguish between hazy image and haze-free image, but also develop a system to remove haze.

The traditional DCP haze removal method has 5 parts: dark channel estimation, atmospheric light estimation, transmission map estimation, transmission map refinement, and image reconstruction, in which transmission map estimation is an essential part, because transmission map records the information about haze density in image.

I first implemented the traditional DCP haze removal method.

Depth Estimation

There is a pre-determined global parameter ω in transmission map estimation setting the level of dehazing for the whole image but using a same level of dehazing for ground and

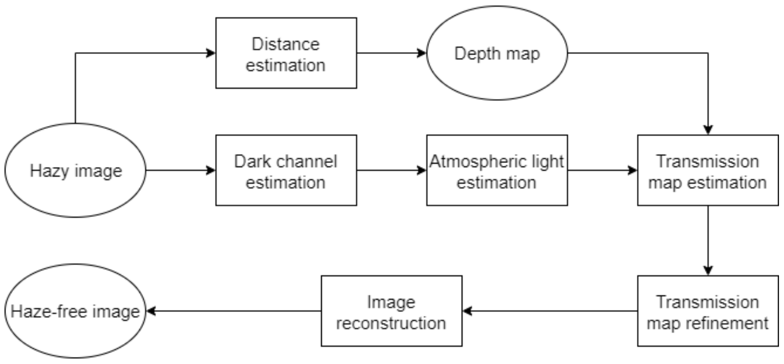


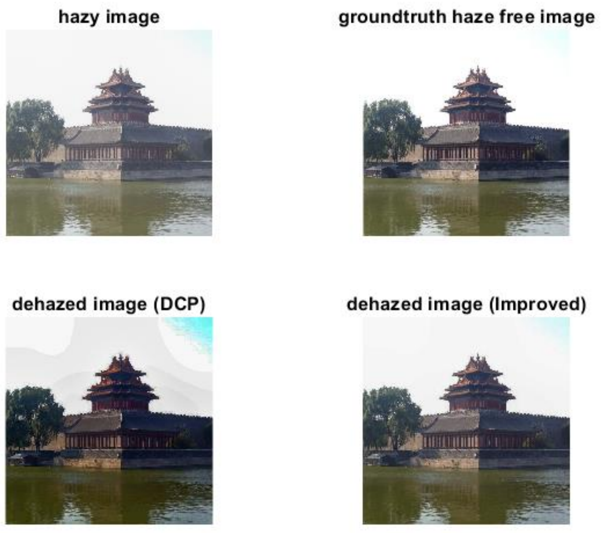
sky would leave strong artifacts in sky region in dehazed image. This is because the human visual system tends to rely on a low level of haze in the sky to preserve aerial perspective. Therefore, instead of using a global value of ω and setting a same level of dehazing for the whole image, I decided to set ω according to the depth, or distance to the camera in the image. For objects in small depth or close to camera, I used large value of ω and stronger dehazing; for objects in deep depth or far from camera (largely sky region), I applied small value of ω and weaker dehazing.

The depth estimation was achieved by using a deep learning neural network called DenseDepth. The image above is an example of depth estimation. In order to combine the DenseDepth with the existed DCP method, I sent the original hazy image into the neural network model, and did a sigmoid normalization on the distance estimation result to ensure that the parameter ω in transmission map estimation had a reasonable range and smooth changing trend.

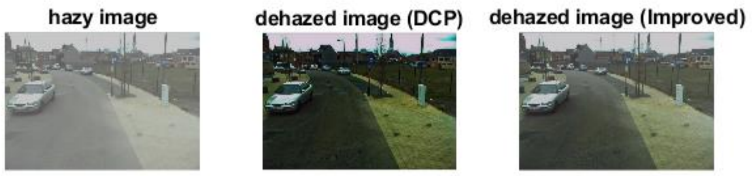
My Haze Removal Procedure

By introducing the depth estimation into haze removal, the flow chart of my method is shown here. I ran my dehazing program over a dataset called REalistic Single Image DEhazing (RESIDE) because it had ground truth haze-free image with hazy image, providing good reference for the assessment of dehazing effect. I also ran my program over another dataset called CURE-TSD because it was a dataset used largely in image object detection, where image dehazing could be an essential preprocessing part.





Besides the subjective visual assessment, I also ran my program through the RESIDE dataset to calculate an objective assessment criterion called FSIMc. If I only used the traditional DCP haze removal method, the FSIMc was 0.5277. If I used my improved haze removal method, the FSIMc increased to 0.5428.



Guitar Musical Instrument Digital Interface (MIDI) Controller | Digital Circuit Project

2017

In this project, my teammate and I built a guitar-based MIDI controller. By using a modified acoustic guitar, we could generate standard MIDI signal by a connected Arduino board, and could further modified the timbre of guitar sound.

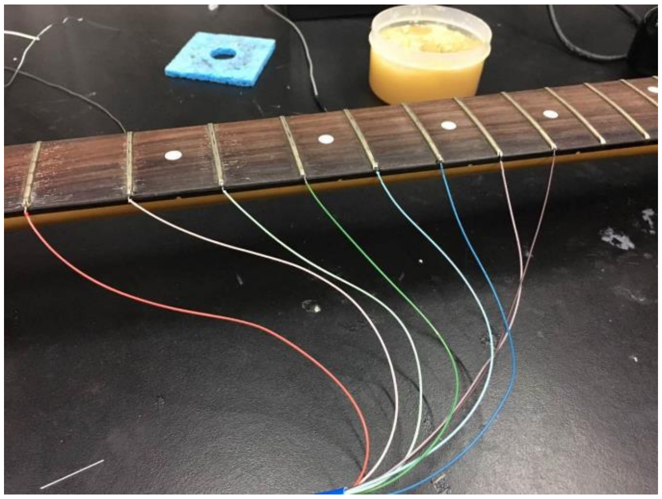
I was in charge of modifying the structure of acoustic guitar.

<u>Insulating guitar strings</u>

Because the strings and frets in guitar are made of metal, they can conduct electrical signal. When we play the guitar, our fingers press strings onto frets, the strings and frets are connected, and thus enable electrical signal to flow through. If a scanning signal is generated to the metal plectrum we used and detected at strings and frets with Arduino, the signal can let the program in Arduino know which string we are playing on, which fret is pressed onto, and what sound would be generated.

Because the strings on guitar may be electrically connected at metal tuning keys or metal bridge, the first modification I did was to insulate the strings. I disassembled the strings from guitar, and wrapped the heads and tails of the strings with insulation electrical tape, then placed them back to the guitar.

<u>Connecting with Arduino</u>



I attached the electrical wire carrying scanning signal output by Arduino to the plectrum, and attached the wire connected to the signal detector at Arduino to the strings and frets. Real-time standard MIDI signal could be generated as we played and pressed the strings onto the frets. One application of this MIDI controller is timbre transfer. There is a channel in MIDI to control timbre, so we changed the value at this channel to change the timbre of the sound produced by guitar. For example, we can play drum kit or trumpet with a guitar.

Personal Hobbies

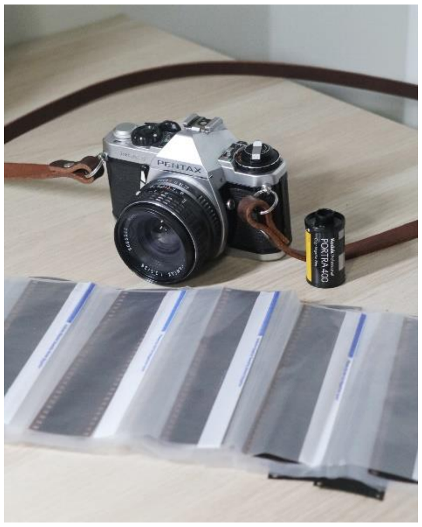
Besides studying, my hobbies make sure I am always working on something, playing with interesting stuffs, embracing the beauty of engineering design, learning the engineers' design principles, and communicating with different kinds of people.

“Industry at its best is the intersection of science and art.” – Edwin H. Land

Analog Synthesizer

I love playing with different analog synthesizers, I own a Korg Monologue and a Teenage Engineering PO-14 Sub Bass. Tweaking different settings and parameters, making new sounds and music sequences, combining the principles of synthesizers and my knowledge in DSP really fascinate me. Recently, I have a plan of implementing my Korg Monologue as a software program with C++ this summer.

Film Cameras & Lenses



I use a Pentax camera produced in the 1970s to take photos with film. I often have to disassemble my camera and lubricate its mechanical components like shutter and dials. I like the feeling of seeing some old stuffs revive on my hands. Some of my works can be viewed here: https://www.instagram.com/spacecatoddity/

Driving

I enjoy driving my Scion FR-S (Toyota 86 in North America market in early years). It is a very "crude" car with few electronic driving assist systems, a good teacher for those who desire to improve their driving skills. I join an amateur racing team in Atlanta and usually drive my car onto the mountain roads in North Georgia with the teammates and exchange thoughts on driving skills and car tuning.

The white one is mine :)