EECS 3213

LAB 1 REPORT

NAME: MICHAEL WILLIAMS

STUDENT ID: 211087798

EECS USERNAME: MW1992

INTRODUCTION

In this lab, we develop and analyze different ways of utilizing a piano to transmit data. More specifically, we create 3 different ways to transmit data, based on the given restrictions. We explore how these different schemes were created, the logic behind it and the maximum bit rates they can transmit, as well as why these are the maximum bit rates that can be transmitted, given the restrictions. Furthermore, we compare the last modulation scheme created to other modulation schemes introduced both in lectures and in the lab and make comparisons to different instruments as well.

In this lab, the piano's sound produced will be used for data transmission. A piano contains a total of 88 different keys, all of which produce different sounds at different frequencies. We assume that any bit sequences in data are equally likely to be transmitted. The main idea throughout the modulation schemes is to utilize a way of determining the maximum number of different combinations of sounds that can be produced, given the restrictions. With these combinations known, we can use information from the textbook and slides, such as Nyquist theorem, to determine the maximum bit rate.

PART 1

In this part of the lab, we assume that a human is playing the piano. Furthermore, the player hits keys at a rate of 1 single key per second, known as "one-finger" play. We also assume that the notes last for at most 1 second. Note bandwidth does not need to be considered in this case.

The logic used here is simple: given the restrictions above, we can only transmit data 1 key at a time per second. That is, every second we can only transmit the amount of bits representable in a single key. To determine the number of bits representable in each key we use the following formula:

$$r = \log_2 L,$$

Where r is the number of data elements (or number of bits) carried in one signal element, and L is the number of different signal elements. In this case, the signal element refers to the piano key.

We make L equal 88, or the number of keys on the piano, to obtain a value of 6.459, which is rounded up to 7. The reason for this rounding up is because we cannot have a decimal

proportional of a bit and because rounding down to 6 would only encompass 64 keys. In other words, the remaining 24 keys would not transmit any information. Rounding up, we can transmit all possible bit sequences obtainable from 88 keys or signal levels. Therefore, given the restrictions above, the maximum number of bits transmittable is 7 bits per key stroke. Because these key strokes occur every second, the max bit rate is 7 bps.

PART 2

For this part of the lab, we allow chords to be played. A chord refers to the sound produced when the player plays up to 5 notes with each hand at once. Therefore, a chord can comprise of up to 10 notes. However, some restrictions are made. First, the notes of each hand should reside within the same octave. An octave refers to a group of 12 notes. Next, the lowest notes playable for each hand should be at least 1 octave apart. For example, if the left handed octave consists of keys 30-42, the right hand octave must begin at least at key 43. We also assume that the right hand is always to the right of the left hand. Last, we assume that 1 chord is played per second and that the notes do not last longer than 1 second.

The logic used here is far more complex than the previous part of the lab. Therefore, we need to break the problem down into smaller problems and then combine them later to obtain a result. For this portion of the lab, the signal element will refer not to the key being pressed, but to the chord being played. Therefore, we must find the total combination of chords that can be played to figure out how many bits can be represented in each signal element. We begin by considering the number of different chords that can be played with 1 hand.

Remembering that each hand can only play notes of a single octave, we use binomial distribution to obtain the number of combinations of chords that any octave can produce. We technically use 5 binomial distributions to encompass all combinations involving 1-5 fingers. Then, we can simply add all 5 up to obtain the total combinations that can be played by a hand, within an octave, as shown below:

$$Total\ number\ of\ combinations = \binom{12}{5} + \binom{12}{4} + \binom{12}{3} + \binom{12}{2} + \binom{12}{1}$$

$$Total\ number\ of\ combinations = 792 + 495 + 220 + 66 + 12$$

$$Total\ number\ of\ combinations = 1585$$

Where 12 refers to the size of the octave and the number below refers to the number of fingers being used.

The next problem involves overlap when trying to count the total number of combinations on the piano. Consider the case where the left hand uses an octave comprised of keys 1-12 and thus can create 1585 different chords. Now imagine that the hand is shifted to the right by one key, so that the new octave is comprised of keys 2-13. With this hand, we can also create 1585 combinations of chords and combining the two octaves gives us a total of 3170 chords. However, the problem in doing so is that many of the chords from the first octave are repeated in the second octave. One solution would be shifting the hand over 12 keys, to an entirely new octave. However, the problem here is that this would mean dropping many combinations of new chords that comprise partially of the previous octave.

Therefore, we must figure out how many new chords can be created when shifting an octave by a single key. It's worth mentioning that any new chords created after this shift will all involve the new key. Therefore, we just need to figure out how many chords are created, using this key. Assuming we are holding down the new key in our octave, we use binomial distribution to determine the number of combinations our other 4 fingers can have to form a new chord:

$$Total\ number\ of\ combinations = \binom{11}{4} + \binom{11}{3} + \binom{11}{2} + \binom{11}{1} + \binom{11}{0}$$

$$Total\ number\ of\ combinations = \ 330 \ + \ 165 + \ 55 \ + \ 11 \ \ + \ \ 0$$

$$Total\ number\ of\ combinations = \ 561 + 1 = 562$$

Where 11 refers to the possible number of keys that can still be played in the octave, assuming the new key is being held down already, and the numbers below refer to the additional number of fingers being used to form the chord. We add 1 at the end to represent the case where only the new key is being pressed.

With the above information addressed, we can now begin to consider the total number of combinations made on the piano. We break this down into 3 parts. First, we consider the initial case where the left hand and right hand are placed to the leftmost octaves of the piano. That is, the left hand covers keys 1-12 and the right hand covers keys 13-24. For the moment, we only look at the right hand. As explained above, it can currently create 1585 possible chords. The right hand can shift an additional 64 times to the right and thus create 64 new octaves. Because each new shift introduces 562 new chords, this means that all 64 shifts create a total of 35,968 new chords, when shifting the right hand alone.

$$New\ chords\ created = Number\ of\ shifts\ X\ chords\ created\ per\ shift$$

$$New\ chords\ created = 64\ X\ 562$$

$$New\ chords\ created = 35968$$

We then add the original 1585 chords from the rights hand's initial position at keys 12-24 to obtain a total of 37553 chords.

$$New\ chords\ created = 35968 + 1585$$
$$New\ chords\ created = 37553$$

Last, we multiply these combinations of right handed chords with the combinations of left handed chords, assuming the left hand always remains at keys 1-12. This gives us a total of 59,521,505 combinations.

$$New\ chords\ created =$$
$$Total\ combination\ of\ right\ handed\ chords\ X\ number\ of\ left\ handed\ chords$$

$$New\ chords\ created = 37553\ X\ 1585$$
$$New\ chords\ created = 59{,}521{,}505$$

For the second part, we now consider each time the left hand shifts over by 1 key, while incorporating the number of combinations the right hand can make. To calculate the number of combinations created from the left hand shifts, we use the following summation:

$$\sum_{i=1}^{64}(Number\ of\ chords\ created\ per\ left\ hand\ shift\ X\ Total\ number\ of\ right\ hand\ combinations)$$

$$= \sum_{i=1}^{64}\left(562\ \ X\ \ \left[1585\ \ +\ \ \left(562\ X\ (64-i)\right)\right]\right)$$

Sum:

$$\sum_{i=1}^{64} 562\,(562\,(64-i)+1585) = 693\,750\,784$$

This summation is done a total of 64 times, to represent the 64 shifts of the left hand, until the right hand is at the right most octave (keys 76-88). For each of these shifts, the total number of new chords created by the left hand is 562 as explained previously. The number of chords created by the right hand for every shift of the left hand is equal to 562 (the number of new chords per shift) multiplied by the number of shifts, which is the equivalent of 64 – i. The total number of chords created through these shifts is 693, 750, 874.

All that is left to incorporate now is the number of single handed combinations that produce chords. This is easily done using the shifting technique described earlier, only now, the number of shifts will be 76 instead of 64. That is:

$$New\ chords\ created = Number\ of\ shifts\ X\ chords\ created\ per\ shift$$

$$New\ chords\ created = 76\ X\ 562$$

$$New\ chords\ created = 42712$$

We then add the original 1585 chords from the left hand's initial position at keys 1-12 to obtain a total of 44297 chords.

$$New\ chords\ created = 42712 + 1585$$
$$New\ chords\ created = 44297$$

Finally, all that is left to do is add up the 3 components discussed above, to get the total number of combinations possible, given the restrictions. This results in a total of 753,316.586.

$$Total\ number\ of\ combinations = Component\ 1\ combinations + Component\ 2\ combinations + Component\ 3\ combinations$$

$$Total\ number\ of\ combinations = 753{,}316{,}586$$

With the total number of combinations, or total number of different signal elements, we can now use the equation from part 1 to obtain the maximum bit rate:

$$r = \log_2 L,$$

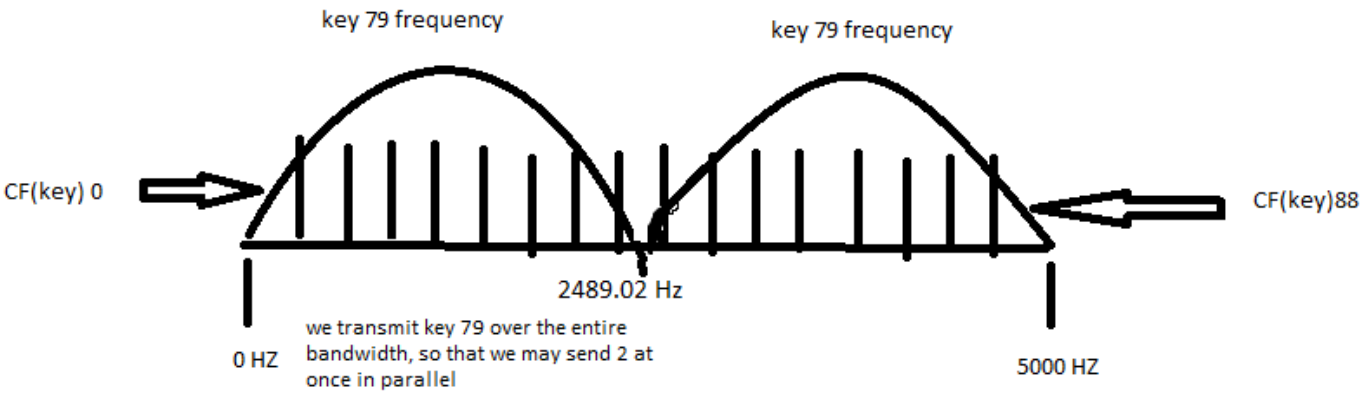
Making L equal the total number of combinations, we obtain a bit rate of 29.489, which we round up to 30. Therefore, the max bit rate is 30 bps.

PART 3

In this part of the lab, we remove the human component entirely. That is, any number of notes can be played for any duration in any combination. We assume the notes are pure sign waves with specified frequencies and constant amplitude. We also assume that the bandwidth of the produced sound does not exceed 5000 Hz. Similarly to the other 2 parts of this lab, it seems that again the main goal is to find all possible combinations of chords (now comprised of any number of keys) that can be played given the restriction that the produced sound cannot exceed 5000 Hz. Our solution uses elements of OFDM, Spread spectrum and PPM. We make note immediately that the number of possible chords on a piano does have an upper limit. The number of possible chords is 2^{88} . Many of these possible chords created include sounds that have a frequency of over 5000 Hz. Therefore, it makes sense that the number of combinations, before considering duration, must be less that that upper limit. Our solution attempts to encompass both every one of these possible chords, as well as the additional amount created when factoring durations.

We first need to think about the elements we are including when figuring out this new combination. First, we include the element of spread spectrum, where we are essentially stretching out our signal in the frequency domain to cover a larger bandwidth. Without doing this, our sounds would not use the entire bandwidth available and thus remove possible combinations from being counted. We also include elements of PPM and OFDM to transmit some keys over another, in order to maximize our available region. We use PPM in the sense that the number of times a key is hit (or its frequency) can be used to transmit different amounts of information (for example, sending pulses at different rates and listening to how far apart they are spaced). We use OFDM in the sense that by transmitting certain keys over others, we can send multiple bits in parallel, provided the bandwidth we transmit certain keys over is sufficient.

Consider using key 79. This key contains a frequency of 2489.02 Hz. If we were to transmit this key alone, over the entire bandwidth of 5000 Hz, we could use OFDM to transmit the same frequency twice over the bandwidth, in parallel, without interference. We could use a form of PSK and elements of PPM to theoretically shift the phase of the frequencies sent, and distinguish them. Therefore, using key 77 alone, we could effectively send 2 different signal elements in parallel and thus transmit 1 bit of information over a given time duration.



-CF refers to carrier frequency

In order to maximize the amount of different signal elements we can send, we effectively try to do the above with all other carrier frequencies or keys, as well as in combination of one another. One example would be spanning key 1 to transmit over 900 HZ and using the regular bandwidth of key 88. This would effectively create a way of sending 34 signal elements at once. In order to find the maximum number of different ways in which we can send signal elements we think of all the possible combinations as a set. The set contains all numbers of occurrences of each key that can effectively be spread out over the bandwidth. For example, the set would contain the frequency of key 63 (being 97.767 HZ) a total of 5 times, as 5 separate elements, because that is the max number of times it could be transmitted over the entire bandwidth, using the methodology described above. It would also contain five occurrences of key 46's frequency for the same reasons. Using this set, we find all possible combinations of different frequencies that do not exceed 5000 HZ, using a modified version of a program online built to do so.

```
public class SumSet {
    public static int counter;
    static void sum_up_recursive(ArrayList<Integer> numbers, int target, ArrayList<Integer> partial) {
       int s = 0;
       for (int x: partial) s += x;
      if (s == target){
            counter++;
      }
       if (s >= target)
            return;
       for(int i=0;i<numbers.size();i++) {
             ArrayList<Integer> remaining = new ArrayList<Integer>();
             int n = numbers.get(i);
             for (int j=i+1; j<numbers.size();j++) remaining.add(numbers.get(j));
             ArrayList<Integer> partial_rec = new ArrayList<Integer>(partial);
             partial_rec.add(n);
             sum_up_recursive(remaining,target,partial_rec);
      }
    }
    static void sum_up(ArrayList<Integer> numbers, int target) {
        sum_up_recursive(numbers,target,new ArrayList<Integer>());
    }
    public static void main(String args[]) {
       // Integer[] numbers = {3,9,8,4,5,7,10}; //has to include all values
        Integer[] numbers = new Integer[513];
            int target = 1;
         counter = 0;
      while (target <=5000){
        //int target = 15;
        sum_up(new ArrayList<Integer>(Arrays.asList(numbers)),target);
        target = target + 1;
      }//while loop
      System.out.println("counter=" + counter);

    }
}
```

https://stackoverflow.com/questions/4632322/finding-all-possible-combinations-of-numbers-to-reach-a-given-sum?noredirect=1&lq=1) shows original code used to solve for values. Actual code is much larger to include all possible array elements

The program gives a result of about 3.817 X 10^{12}.

Using the same formula described in parts 1 and 2, we let L equal this value to obtain a rate of 41.79 bps, or 42 bps. However, because we now factor bandwidth into our question, we must refer to the Nyquist theorem for our final result. According to the Nyquist theorem, the bit rate is twice the bandwidth, multiplied by the value we obtained, as shown below:

$$\text{BitRate} = 2 \times \text{bandwidth} \times \log_2 L$$

Using a bandwidth of 5000 Hz, we obtain a total bit rate of 420,000 bps.

The biggest factor limiting the bit rate is the bandwidth. The reason for this is evident from the fact that our total number of combinations calculated is only a small fraction of the 2^{88} possible combinations achievable without even considering duration of notes. Many of the possible combinations conceivable are prevented because of the limitations on the bandwidth. To understand this more clearly, consider if we were to add up each key's frequencies, from the lowest to the highest without considering durations. Assuming that each

key can only be pressed once, the maximum number of keys that can be simultaneously played without the sound exceeding 5000 HZ is 42. With less than half of the keys being able to be played simultaneously under these conditions, it's clear that the level of bandwidth available greatly affects the bit rate possible. This bit rate is also to some regards, limited by the number of different signals achievable. Obviously, a larger number of signal elements equates to a larger bit rate.

PART 4

In this part of the lab, we further analyze the modulation scheme used in part 3. First, we consider if controlling the individual note durations can help to achieve a higher bit rate. The answer is yes. By controlling the note durations within a specified time period, we can utilize elements of pulse-position modulation (PPM). If we utilize 2^m different time shifts, or durations of key strokes in a specified time, we can effectively encode m message bits. This effectively increases the possible number of combinations you can create and thus allows us to create more signal elements. More signal elements equates to a higher bit rate. Therefore controlling the individual note durations must help to achieve a higher bit rate in part 3.

Next part of this question is to consider the differences between our modulation and a typical OFDM. A typical OFDM encodes digital data on multiple carrier frequencies and utilizes a large number of sub carrier signals to carry data on multiple parallel channels, without interference. Our modulation uses these similar features of OFDM with a few exceptions. For example, although our modulation also takes advantage of parallel transmission, our signals do not always do so using a large number of sub carrier signals, but rather as many subcarrier signals that can be generated given the bandwidth limit. More so, our modulation uses aspects of PPM discussed above to take advantage of the note durations, and elements of spread spectrum to utilize the entire bandwidth made available, rather than just the bandwidth of the signal. Last, our modulation is not guaranteed to provide all the security and quality related advantages of an OFDM, such as transmission without interference, coping with attenuation and SNR rate improvement.

The last part of this question is to imagine using a different instrument, in particular a drum set. Using a drum set would mean making some changes to the modulation scheme, so that it may be used for data communication. The biggest issue is that the bit rate would be significantly smaller, because there would be less separate drums than keyboard keys. Therefore, the signal rate would be substantially smaller, and so the bit rate would as well. Another issue that would need addressing is the noise. After a drum is hit, the pressure from the strike causes a quick jump in the frequency content, creating a pulse that looks very similar

to a square wave. However, the fading sound made after the drum is hit creates possible lingering noise which may disrupt other signals. This can result in some cases, where different drum beats actually end up sounding the same. In other words, it would be harder to distinguish individual drum beats, which translates to having a harder time interpreting and distinguishing signal elements. In order to adapt to this, our modulation scheme would have to adapt more elements of the OFDM, to ensure that any frequency never interferes with other frequencies. More so, we could increase the amount of signal elements by how a drum is hit. By recording the pressures at which the drum is hit, it would be possible to divide them into different groups, depending on the distinguishable strengths at which the drum is struck. Distinguishing this would mean being able to form more combinations of different signal elements, much like when considering the duration of piano keys, which would result in a higher bit rate. Overall however, I believe that the bit rate of this scheme would not be as large as the bit rate of the scheme used in part 3, primarily because of the number of different signal elements that can be created. A piano contains 88 keys, whereas a typical drum set contains only contains 5 drums. More so, distinguishing pressures and dealing with noise problems could increase the bit rate, but most likely not so much as to cover the gap created simply by the number of instruments that can be struck, being drums versus piano keys.