EE 126 Project Proposal

Akilesh Bapu Prashanth Ganesh Nitin Manivasagan

March 3, 2017

1 Introduction

For our project, we are planning on creating the Digital Communication system. As seen in lecture, the Transmitter follows this model:

$$\begin{tikzcd}
\boxed{\text{Source}} \arrow[r] & \boxed{\text{Source Encoder}} \arrow[r] & \boxed{\text{Channel Encoder}} \arrow[r] & \boxed{\text{Modulator}}
\end{tikzcd}$$

The Receiver follows a similar model:

$$\begin{tikzcd}
\boxed{\text{Demodulator}} \arrow[r] & \boxed{\text{Channel Decoder}} \arrow[r] & \boxed{\text{Source Decoder}} \arrow[r] & \boxed{\text{Destination}}
\end{tikzcd}$$

2 Transmitter

2.1 Source Encoder

We will use simple ascii conversion to convert text to binary. Python has a module called **binascii** which would be useful.

```
import binascii
bin(int(binascii.hexlify('hello'), 16))
#'0b110100001100101011011000110110001101111'
```

2.2 Channel Encoder

A Reed-Solomon code is a popular error correcting code that we intend to use in this application. We can break up a message of a certain number of bits into 8-bit "symbols". Then if a message consists of **k** symbols, a Reed-Solomon code will send **k+2s** bits as a message. The parameter **s** is the number of parity bits, and in general, the Reed Solomon code can detect up to $\frac{s}{2}$ errors in the transmitted symbol. This makes the channel quite resistant to noise and erasures.

Finding the number of parity bits will involve some tuning on our part. One proposed method is to collect sample data and create a random variable **X** that corresponds to the number of errors we would encounter in a particular message. We can then bound the value of **X** within some appropriate confidence interval, and use the bound to set the value of **s**.

Another important design consideration is the bitrate. If we want to achieve a bitrate of 100 $\frac{bits}{sec}$, then s cannot be too large, since we would need to send more bits and would thus negatively impact our relative bitrate. This will involve some more optimization on our part.

2.3 Channel Modulator

The idea is to map frequencies to bit sequences of certain lengths (packets). Packet size and packet rate are variables that depend on the channel. We estimate $\frac{bits}{sec}$ of the data (not including encoding) to be $num\frac{packets}{sec} \times num\frac{bits}{packet}$. When factoring in the encoding to find bitrate, let **n** be the length of the message prior to Reed Solomon encoding and **nk** be the length after encoding, then the total bit-rate can be found by dividing non-encoding bit rate by k.

The only requirements for the frequency map is that all frequencies must be orthogonal to one another and that the frequencies chosen are within the min/max frequencies of both the speaker and the microphone.

We can calculate Reed Solomon parity parameter and the confidence that the signal can be completely decoded without errors once we know on average how many packets will have erasures/bit-flips. We can come up with a good estimate by testing in multiple different scenarios (hand claps/loud rooms/different frequencies of noise) and once we have a maximum value of erasures, we can then optimize accordingly.

3 Receiver

3.1 Channel Demodulator

Have a lookup table of size 2^k where k is the packet size which maps frequencies to bit sequences. Each time a signal is received, we use Discrete Fourier Transform to convert the received bit sequence into a sequence of frequencies. Then we can take the most prevalent frequency, look up the bit sequence in the frequency table, and use that as our demodulation.

One consideration was to use a filter to reduce some of the higher frequency noise but the drawback is that we have to be careful in what frequencies we use for the transmitter and make sure they donâĂŹt overlap with the designated noise frequencies.

3.2 Channel Decoder

Using the Reed-Solomon scheme, we can decode the received signal and correct up to $\frac{s}{2}$ errors. Then, we can proceed to the Source Decoding stage.

3.3 Source Decoder

binascii can be used to convert binary to text as well

```
import binascii
n = int('0b110100001100101011011000110110001101111', 2)
binascii.unhexlify('%x' % n)
#'hello'
```

4 Challenges

Using Reed Solomon, we can meet a pretty high $\frac{erasure}{bit-flip}$ rate by just picking a suitable parity parameter but this will decrease the bitrate linearly, so we need to find an optimal value for **s**. If this does not work, we can try phase modulation instead of frequency modulation to see if phase modulation tends to be more resistant to noise. We might also have to mix in other encoding methods so that we can keep the parity parameter as small as possible.