NARI PPS Bits on Air – Bericht

Céline Schönenberger – [sceline@student.ethz.ch](mailto:sceline@student.ethz.ch)

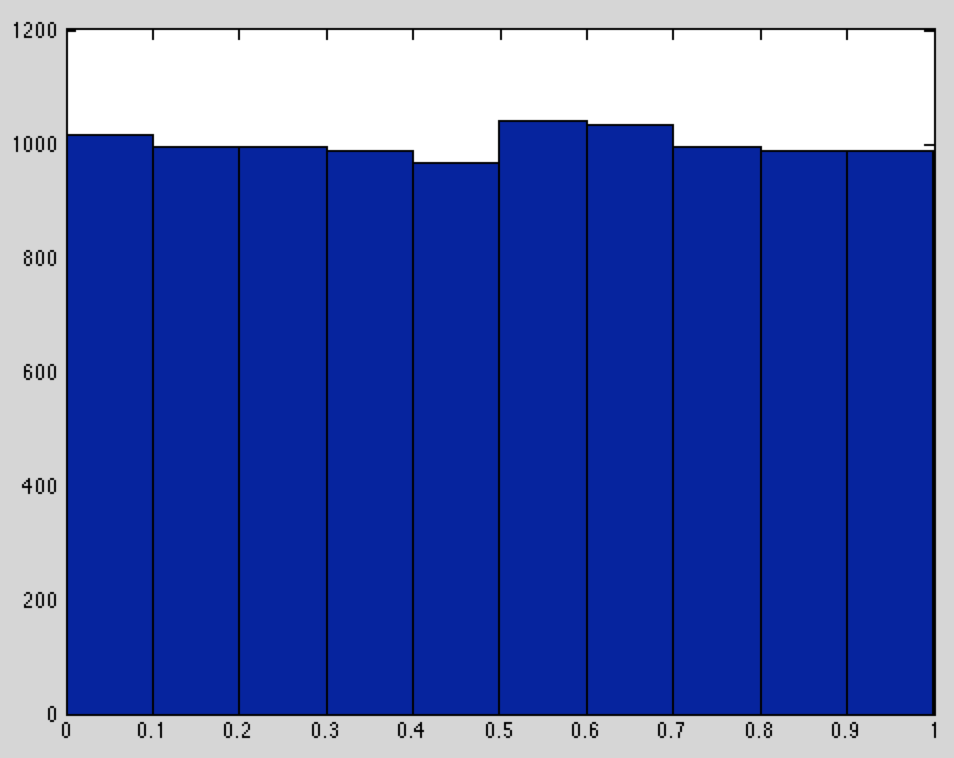
Jonathan Müller – [jo@student.ethz.ch](mailto:jo@student.ethz.ch)

# Nachmittag 1 – Programmierung eines einfachen Kommunikationskanals

## Zufallsgenerator

hist(rand(10000, 1))

Produziert folgenden Graphen:



Man kann daher davon ausgehen, dass die Zahlen gleichm¨s̄sig verteilt sind.

## Sender

Der Sender generiert eine zufällige Bitsequenz, wobei die 0 mit Wahrscheinlichkeit p auftritt.

function bitsequence = source( sequence\_length, p )

bitsequence = rand(1, sequence\_length) > p;

end

## Empfänger/Drain

Die Funktion Drain berechnet die Standardabweichung und den Mittelwert einer Bitsequenz.

function [ mittelwert, sigma ] = drain( bitsequenz )

mittelwert = mean(bitsequenz);

sigma = std(bitsequenz);

end

## Mittelwert und Standardabweichung messen

In einer Schleife soll jetzt source.m und drain.m mehrere male ausgeführt werden, um ein Histogramm der Stanndardabweichung auszugeben.

function [sigma\_total] = loop( n, loopsize )

p = 0.5;

mittelwert\_v = zeros(loopsize, 1);

sigma\_v = zeros(loopsize, 1);

for k = [1:loopsize]

[ mittelwert, sigma ] = drain(source(n,p));

mittelwert\_v(k) = mittelwert;

sigma\_v(k) = sigma;

end

figure(1);

hist(mittelwert\_v, 100)

figure(2);

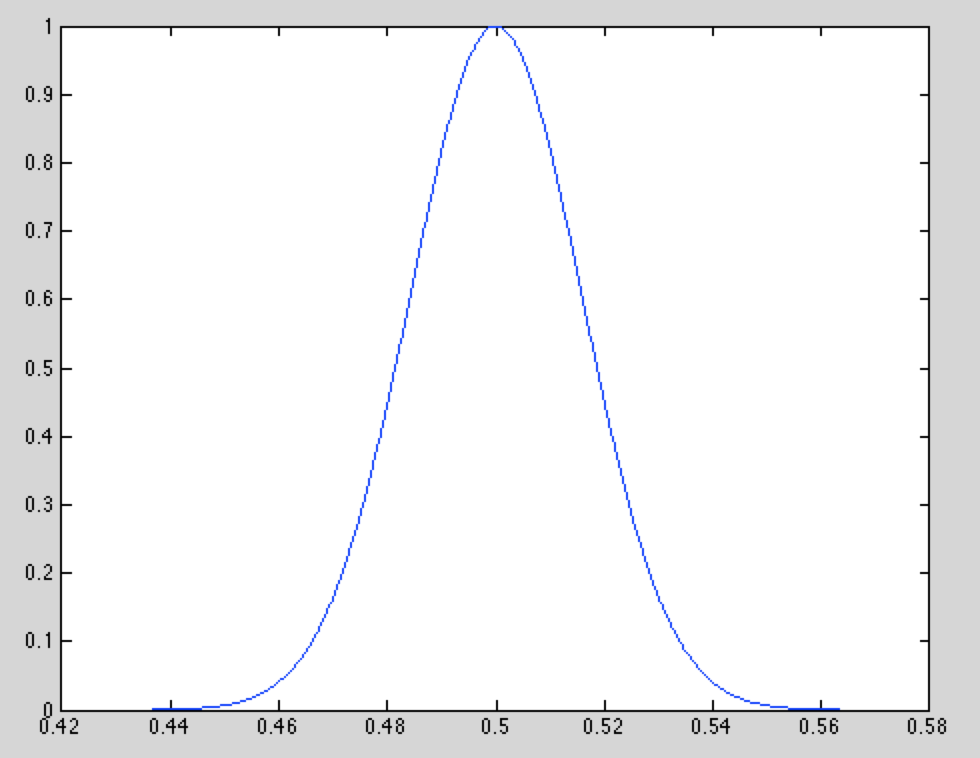
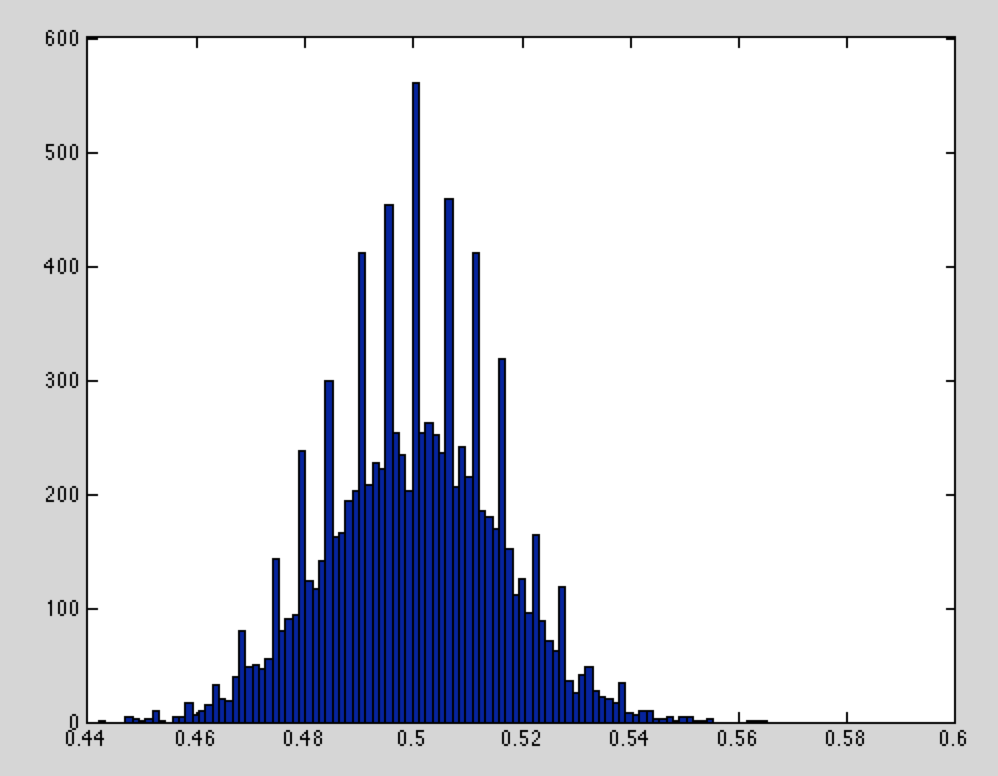
hist(sigma\_v, 100)

gauss(1-p, std(mittelwert\_v))

sigma\_total = std(mittelwert\_v);

end

Der Mittelwert konvergiert gegen . für grosse Werte von .



Histogramm des Mittelwerts, mit der Gausskurve daneben.

Unser Programm zur Erzeugung der Gausskurve:

function [ output\_args ] = gauss( u, sigma )

x = (-sigma \* 4 + u:sigma \* 8 / 1000:sigma \* 4 + u)';

y = gaussmf(x, [sigma u]);

figure(3);

plot(x, y)

ylim([0 1]);

end

und hängen voneinander ab:

## Übertragungskanal

Nun soll ein fehlerhafter Übertragungskanal simuliert werden, der mit der Wahrscheinlichkeit Übertragunsfehler (Bit-Flips) verursacht. Wir verwenden dazu xor.

function [ channel\_bit\_sequence ] = BSC\_channel( bit\_sequence, q )

channel\_bit\_sequence = xor( source(length(bit\_sequence), 1-q), bit\_sequence);

end

Die Bit-Error-Rate gibt an, wie viele Bits sich im Vergleich zum Ursprungssignal verändert haben.

function [ ber ] = calcBEr(bit\_sequence, channel\_bit\_sequence)

ber = mean(xor(bit\_sequence, channel\_bit\_sequence))

end

## Fehlerkorrektur mit Repetitionscode

Mit einem einfachen Code sollen Übertragungsfehler korrigiert werden können. Dazu wird jedes bit einfach mal wiederholt.

**Encodierung:**

function [ bitsequence\_encoded ] = repencode(bitsequence, n )

bitsequence\_encoded = repmat(bitsequence, n, 1);

bitsequence\_encoded = bitsequence\_encoded(:)';

end

**Decodierung:**

function [ bitsequence\_decoded ] = repdecode( bitsequence, n )

bitsequence\_decoded =

reshape(bitsequence', n, length(bitsequence) / n);

bitsequence\_decoded = mean(bitsequence\_decoded, 1) > 0.5;

end

Wenn die Fehlerrate kleiner ist, funktioniert der Code natürlich besser, da es weniger Fehler zu korrigieren gibt. Wenn per Zufall mehr wie die Hälfte der Bits einer Wiederholung geflipt werden, kommt das Bit am Ende falsch heraus.

# Nachmittag 2 – Modulation und Demodulation

## Modulation

ist die Sprungfunktion, die vor dem Zeitpunkt immer und danach immer 1 ist.

Die gezeigte Formel gibt die modulierte Version einer Bitsequenz wieder, d.h. jeweils Abschnitte von mit den beiden Übertragungsfrequenzen .

Bei unsere Implementierung werden die Parameter nicht aus einem .mat-File geladen, sondern direkt als Parameter der Funktion übergeben.

function y = modulate( seq, ts, t0, t1 )

seq1 = repencode(seq, ts);

f0 = repmat(cos([0:ts - 1] / t0 \* 2 \* pi), 1, length(seq));

f1 = repmat(cos([0:ts - 1] / t1 \* 2 \* pi), 1, length(seq));

y = (seq1 .\* f1) + (~seq1 .\* f0);

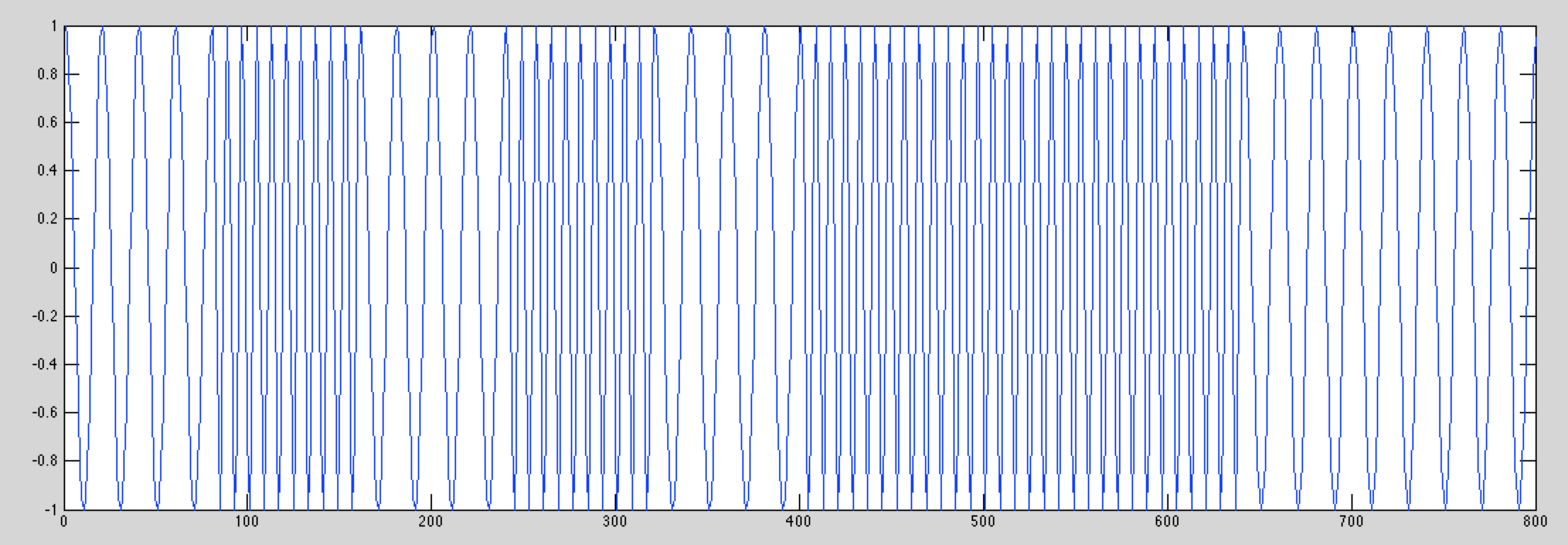
plot(y)

end

Unser Programm test\_modulate.m:

modulate([0 1 0 1 0 1 1 1 0 0 ], 80, 20, 8)

Dies erzeugt folgenden Plot:



## Demodulation

Aus den beiden korrelierten Sequenzen soll die originale Bitsequenz wiederhergestellt werden. Das empfangene Signal wird aufgeteilt in Abschnitte, deren Länge jeweils entsprechen. Jeder Abschnitt wird separat mit den Trägerfrequenzen korreliert. Um herauszufinden, ob nun eine 0 oder eine 1 empfangen wurde, wird das Integral des Betrags der beiden Korrelationen verglichen (Grösser als), diskret betrachtet also sum(abs(abschnitt)) . Diese Schreibweise findet sich jedoch nicht direkt so in unserem Code, da wir mit reshape und Matrizen arbeiten.

function seq = demodulate( y, ts, t0, t1 )

x0 = repmat(cos([0:ts - 1] / t0 \* 2 \* pi), 1, length(y) / ts);

x1 = repmat(cos([0:ts - 1] / t1 \* 2 \* pi), 1, length(y) / ts);

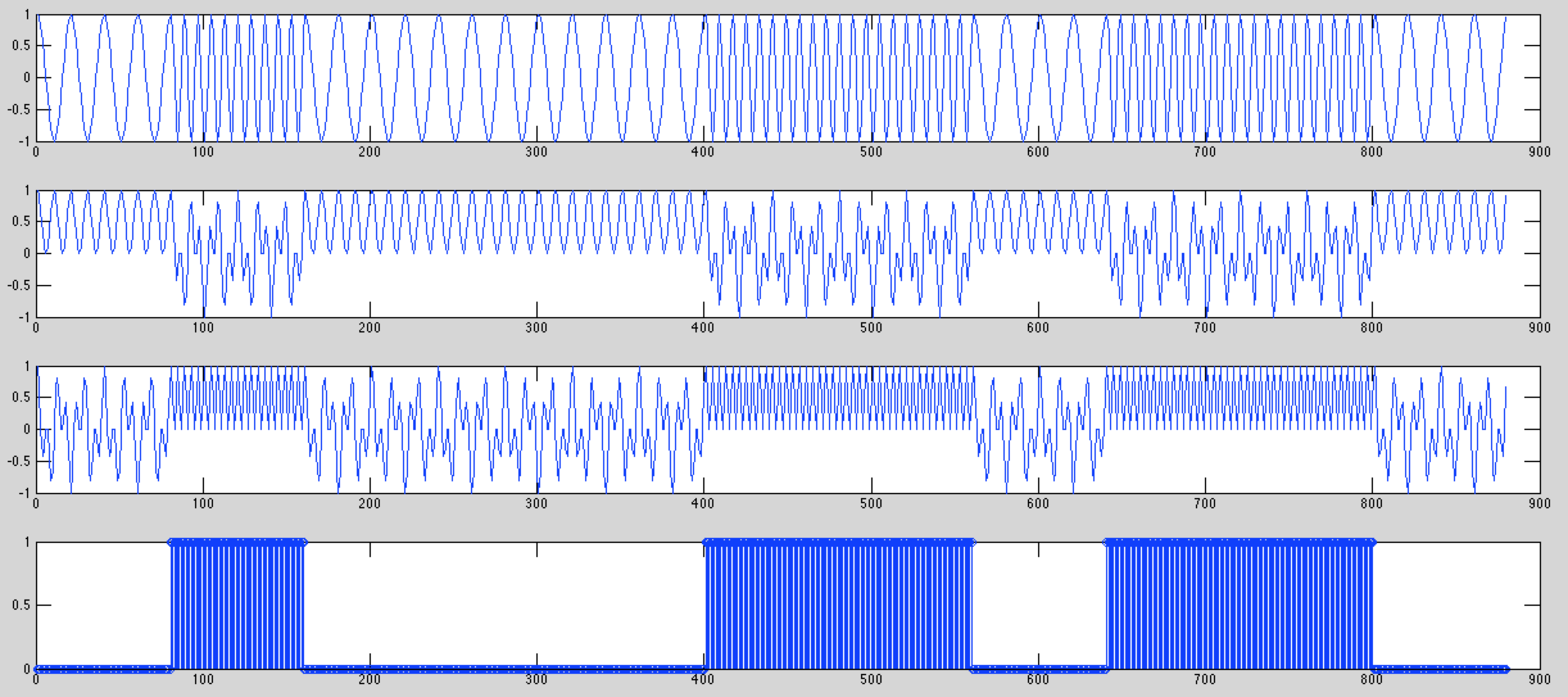
plot(x0 .\* y)

seq0 = sum(reshape(x0 .\* y, ts, length(y) / ts));

seq1 = sum(reshape(x1 .\* y, ts, length(y) / ts));

seq = abs(seq0) < abs(seq1);

end



Plots für das Decoding der Bitsequenz [0 1 0 0 0 1 1 0 1 1 0]:

1. Moduliertes Signal
2. Korrelation mit
3. Korrelation mit
4. Berechnete Bitsequenz

**Durchsatz erhöhen**: Dies kann man über mehrere Wege erreichen:

* Die beiden Trägerfrequenzen erhöhen, und gleichzeitig die Zeitabschnitte verkleinern
* Nur verkleinern
* Ein besserer Fehlerkorrekturalgorithmus wie „Repencode“

## Additive White Gaussian Noise Channel (AWGN)

Um die Modulation und Demodulation zu testen, wird bei der Simulation des Übertragungskanals nun White Noise auf das Signal addiert. Unser Code dazu:

clear

repenc = 10

seq = source(100, 0.5);

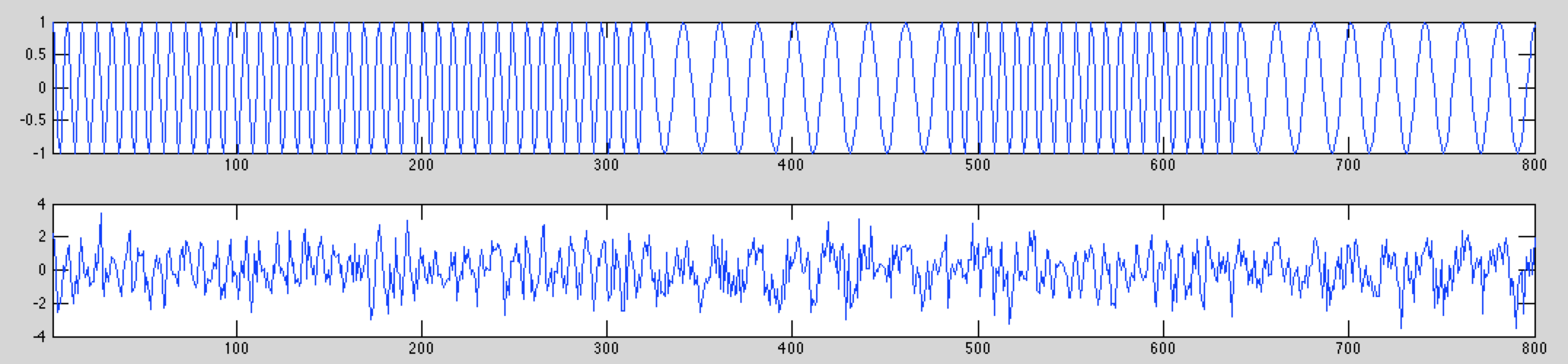
enc = repencode(seq, repenc);

y = modulate(enc, 80, 20, 8);

r = awgn(y, 0)

dec = repdecode(demodulate(r, 80, 20, 8), repenc);

calcBER(seq, dec)



* Oben: Originalsignal
* Unten: Signal mit Noise

Wir haben eine Repetitionscode mit verwendet, und eine Signallänge von 10000 Bits.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| SNR [dB] | 0 | -10 | -20 | -25 | -30 | -50 | -100 |
| BER | 0 | 0.0043 | 0.3527 | 0.4584 | 0.4937 | 0.5053 | 0.4923 |

Ab ca. beginnt die BER sich immer um 0.5 zu bewegen.

# Nachmittag 3 – Synchronisation

## Kreuzkorrelation von Hand

Wir korrelieren den Vektor mit dem Vektor .

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | -1 | -1 | 1 | -1 | 1 | Summe |
|  | -1 | 1 | 0 | 0 | 0 | 0 |
|  | 0 | -1 | -1 | 0 | 0 | -2 |
|  | 0 | 0 | 1 | 1 | 0 | 2 |
|  | 0 | 0 | 0 | -1 | -1 | -2 |
|  | 0 | 0 | 0 | 0 | 1 | 1 |
|  | 0 | 0 | 0 | 0 | 0 | 0 |

Diese Korrelation zeigt uns, das der Vektor mit dem Offset im Vektor vorkommt.

## Synchronisation

Dies ist unsere allgemeine Sync-Funktion, die den Offset eines Signals b im Signal a angibt. Dabei sollte b kürzer wie a sein.

function offset = sync( a, b )

[c, lag] = xcorr(a, b);

[m, i] = max(c);

offset = lag(i) + 1;

end

Es werden anstatt Werte die Werte zur Repräsentation verwendet. Weil das Signal zur Kreuzkorrelation mit Nullen aufgefüllt wird, würde sich sonst das Signal nicht mehr vom Null-Signal abheben, und dieses könnte somit einen störenden Einfluss haben.

Wenn die Sequenz b mehrere Male in a vorkommt, wird das letzte Vorkommnis davon zurückgegeben. Dies liegt daran, dass max() bei mehreren Maxima den Index des letzten Maximums zurückgibt.

Die Vereinfachung length(a) >> length(b) ist nicht nötig, length(a) >= length(b) genügt.

## Symbolsynchronisation

Bei der Symbolsynchronisation geht es nun darum, den Beginn der Zeitabschnitte der Länge zu synchronisieren. Dafür wird zuerst ein -Übergang gesucht, und anschliessend die nötige Anzahl Nullen vorne angehängt.

function [ output ] = symbolsync( a, ts, t0, t1 )

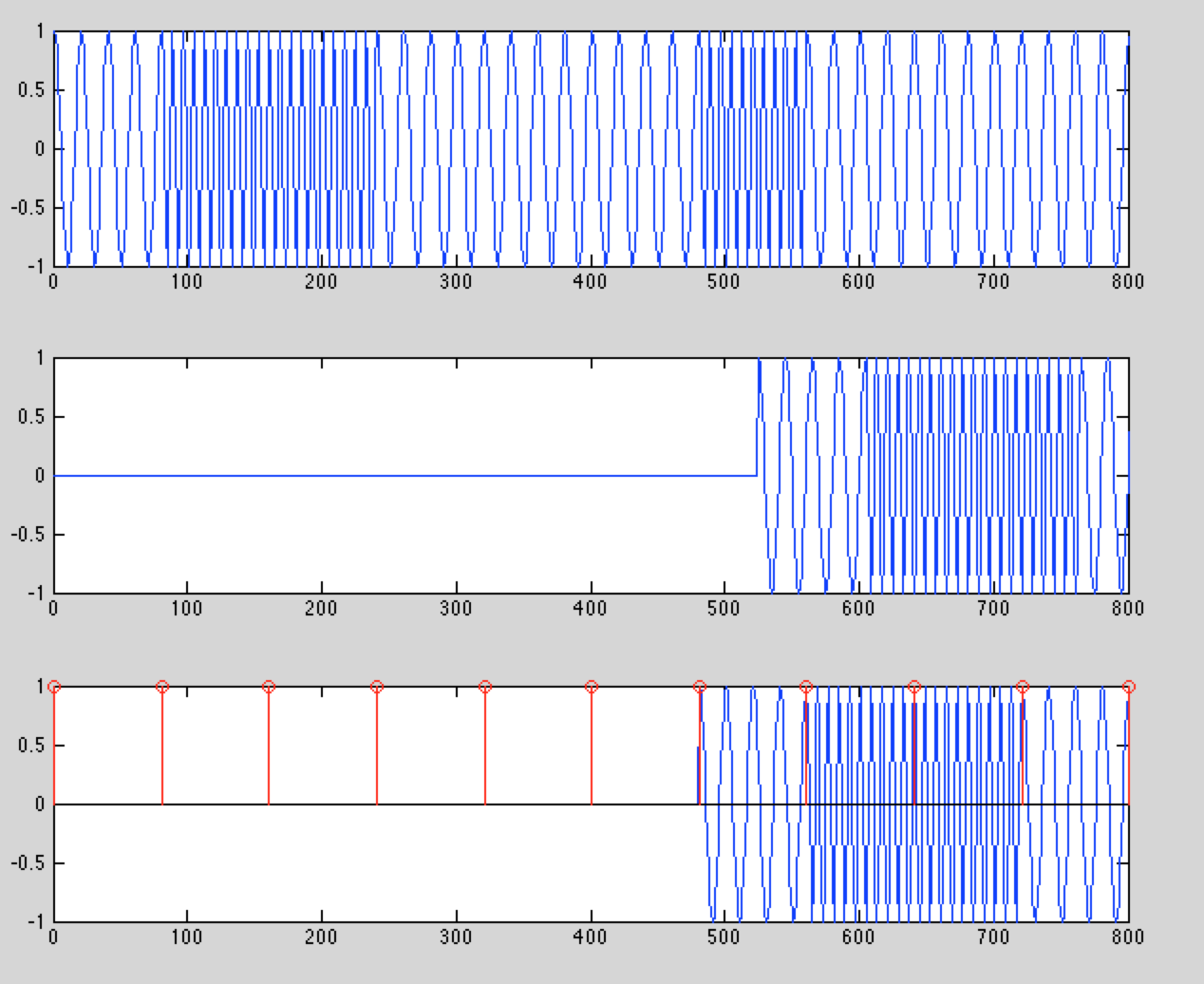
b = modulate([0 1], ts, t0, t1);

offset = sync(a, b);

output = a(mod(offset, ts):length(a));

output = [output, zeros(1, (ts - mod(length(output), ts)))];

end



In diesem Plot wird illustriert, wie Symbolsync arbeitet.

1. Das Originalsignal
2. Das Signal, mit einer zufälligen Anzahl Nullen vorangesetzt. Dies simuliert die Desynchronisation bei der Übertragung
3. Das synchronisierte Signal (Ende abgeschnitten auf dem Plot), mit den eingezeichneten Zeitschlitzen (rot)

## Rahmensynchronisation

Um den Beginn und das Ende einer Übertragung zu finden, muss nun eine Rahmensynchronisation durchgeführt werden. Dabei wird bei jeder gesendeten Nachricht eine Prä- und Postambel angehängt, die auch dem Empfänger bekannt ist, welcher dann danach sucht.

Unsere Prä- und Postambel sind jeweils 50 Bit lang, und wurden nach völlig unwissenschaftlichen Kriterien zufällig ausgewählt:

function [pre, post] = preamble( )

pre = [

0 0 1 0 1 1 0 1 0 1

1 0 0 1 1 0 1 1 0 0

1 0 1 0 1 1 1 1 0 0

1 0 1 0 1 0 1 0 1 1

1 0 1 0 0 1 1 1 0 0

];

post = [

1 0 1 1 0 0 1 1 0 0

1 1 1 0 0 1 1 0 1 1

0 0 1 1 1 1 0 0 1 1

0 0 1 1 0 0 0 1 0 1

0 1 1 0 1 0 0 1 1 0

];

end

Anhängen ans Signal:

function signal = addkeys( source )

[pre, post] = preamble();

signal = [pre, source, post];

end

Die framesync Funktion sucht nun nach der Prä- und Postambel und schneidet das Signal entsprechend zu.

function output = framesync( signal )

[pre, post] = preamble();

pre = pre \* 2 - 1;

post = post \* 2 - 1;

signal\_ = signal \* 2 - 1;

offset = sync(signal\_, pre);

offset\_end = sync(signal\_, post);

output = signal(offset+length(pre):offset\_end - 1);

end

Unser Test-Code für die Synchronisation:

repenc = 2;

signal = source(10, 0.5);

seq = addkeys(signal);

enc = repencode(seq, repenc);

y = modulate(enc, 80, 20, 8);

shifted = awgn([zeros(1, 100 + floor(rand(1, 1) \* 1000)), y, zeros(1, floor(rand(1, 1) \* 1000))], 10);

synced = symbolsync(shifted, 80 \* repenc, 20, 8);

demodulated = demodulate(synced, 80, 20, 8);

fsynced = framesync(repdecode(demodulated, repenc));

Und ein Beispieldurchlauf:

>> test\_framesync

signal =

1 0 1 0 0 1 1 0 1 1

fsynced =

1 0 1 0 0 1 1 0 1 1

# Nachmittag 4 – Senden und Empfangen

## Audioplayer- und Rekorder

function [] = playsound( snd )

samplerate = 22044;

seconds = length(snd) / samplerate;

ap = audioplayer(snd', samplerate, 16, -1);

play(ap);

pause(seconds);

stop(ap);

end

function snd = recordsound( seconds )

ar = audiorecorder(22044, 16, 1, -1);

record(ar);

pause(seconds);

stop(ar);

snd = getaudiodata(ar);

snd = snd';

end

## Send

function modulated = send( data, repenc )

seq = addkeys(data);

enc = repencode(seq, repenc);

modulated = modulate(enc, 30, 10, 3);

end

Unser Test-Programm test\_send.m:

data = source(50, 0.5);

repenc = 2;

tosend = send(data, repenc);

tosend = [zeros(1, 800) tosend zeros(1, 8000)];

samplerate = 22044;

ap = audioplayer(tosend', samplerate, 16, -1);

playblocking(ap);

pause(2)

## Receive

function data = receive( modulated, repenc )

synced = symbolsync(modulated, 30 \* repenc, 10, 3);

demodulated = demodulate(synced, 30, 10, 3);

decoded = repdecode(demodulated, repenc);

data = framesync(decoded);

end

test\_receive.m:

seconds = 10

ar = audiorecorder(samplerate, 16, 1, -1);

record(ar);

pause(seconds)

stop(ar);

recv = getaudiodata(ar);

recv = recv';

dataout = receive(recv, repenc);

playsound(recv)

calcBER(data, dataout);

## Kombination von send und receive

Wir sind nun dazu in der Lage, eine Nachricht in ein Audiosignal zu verwandeln, dieses abzuspielen und gleichzeitig über ein Mikrofon aufzunehmen, und es dann zu decodieren.

Testprogramm zur Übermittlung einer zufälligen 50Bit-Sequenz:

data = source(50, 0.5);

repenc = 2;

tosend = send(data, repenc);

tosend = [zeros(1, 800) tosend zeros(1, 8000)];

samplerate = 22044;

seconds = length(tosend) / samplerate;

ar = audiorecorder(samplerate, 16, 1, -1);

record(ar);

pause(2)

ap = audioplayer(tosend', samplerate, 16, -1);

playblocking(ap);

pause(2)

stop(ar);

recv = getaudiodata(ar);

recv = recv';

length(tosend)

length(recv)

dataout = receive(recv, repenc);

calcBER(data, dataout);

Die Übertragung funktioniert so sehr zuverlässig. Wir haben auch die Übertragung über mehrere Meter versucht, was auch funktioniert hat.

Störungen wie Klatschen oder ins Mikrofok blasen bleiben meist erfolglos, ausser die Prä- oder Postambel werden gestört.

Was auffällt ist, wie langsam die Übertragung stattfindet.

## Übertragung von Bildern

Um Übertragungsfehler zu visualisieren, haben wir die Übertragung von monochromen (Schwarz/Weiss) Bitmaps versucht. Entsprechender Code:

fid = fopen('image.bmp', 'r');

data = fread(fid, 'ubit1')';

fclose(fid)

repenc = 1;

tosend = send(data, repenc);

tosend = [zeros(1, 800) tosend zeros(1, 8000)];

samplerate = 22044;

seconds = length(tosend) / samplerate;

ar = audiorecorder(samplerate, 16, 1, -1);

record(ar);

pause(2)

ap = audioplayer(tosend', samplerate, 16, -1);

playblocking(ap);

pause(2)

stop(ar);

recv = getaudiodata(ar);

recv = recv';

dataout = receive(recv, repenc);

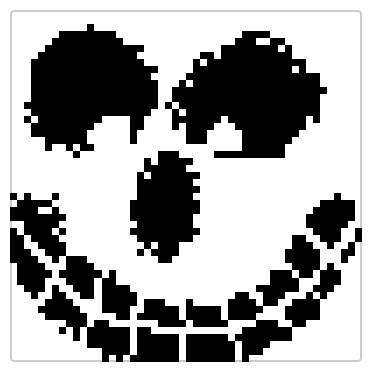
fid = fopen('image-recv.bmp', 'w');

fwrite(fid, dataout, 'ubit1')';

fclose(fid);

Ein Monochrom-Bitmap mit 50x50 Pixeln ist 463 Bytes gross, d.h. 3.7kbit.

Unser original-Bild, erzeugt mit MS Paint:



Am besten lassen sich visualisierbare Störungen erzeugen, wenn man repencode auf 1 setzt. Jedoch wird dann manchmal auch die Präambel oder der Bitmap-Header falsch übertragen, was zum vollständigen Verlust des Signals führt.

|  |  |  |
| --- | --- | --- |
| Macintosh HD:Users:joni:Desktop:Screen Shot 2013-12-16 at 15.50.58.png | Macintosh HD:Users:joni:Desktop:Screen Shot 2013-12-16 at 15.51.20.png | Macintosh HD:Users:joni:Desktop:Screen Shot 2013-12-16 at 15.51.14.png |
| Beinahe fehlerfreie Übertragung | Viele Fehler | Vermutlich kaputter Header |