Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für Kommunikationstechnik
Prof. Dr. H. Bölcskei

P&S "Bits on Air"
5. Teil

Markus Gärtner, Felix Kneubühler & Felix Wermelinger

Revidierte Version vom 29. April 2022

1 Einleitung

In den vorangegangenen vier Praktikumsteilen wurden `matlab`-Funktionen zur Erzeugung des abgetasteten informationstragenden Signals am Sender-PC und zum Empfang der Datenfolge am Empfänger-PC einschliesslich Synchronisation, Demodulation und Detektion implementiert. Nachdem diese Funktionen erfolgreich getestet worden sind, kann nun die eigentliche Datenübertragung mit Lautsprecher und Mikrofon angegangen werden.

2 Senden und Empfangen der Signale

Um die modulierten Signale abspielen zu können gibts es zwei vorgeschlagene Implementationsvarianten, die benutzt werden können. Man kann die Matlab internen Funktionen `audiorecorder` und `audioplayer` verwenden oder sich via Matlab-funktion `unix` den unix Konsolenkommandos `rec` und `play`. Wir werden uns hier auf die Matlab-variante beschränken
Hinweis: `audioplayer` und `audiorecorder` funktionieren auch auf Windows. Als Mikrophon kann auch ein einfaches Headset verwendet werden.

Aufgaben:

- Machen Sie sich als erstes mit den oben genannten Funktionen vertraut. Schreiben Sie die Funktionen `recordsound` und `playsound`, wobei erstere eine Sprachaufnahme durchführen soll und letztere in einem Vektor gespeicherte Samples abspielen soll.

  Hinweise:

  - Die Befehle `audioplayer` und `audiorecorder` erstellen ein Objekt und geben jeweils einen Handler dieses Objekts zurück. Mit diesem Handler können Methoden des Objekt ausgeführt werden. Welche Methoden es für ein bestimmtes

Objekt gibt erfährt man mittels der Funktion `methods`. Die vorerst spannenden Methoden sind: `record, play, stop`

- Um eine korrekte Aufnahme durchzuführen, muss beim `audiorecorder` nach dem Ende der Übertragung die stop-Methode ausgeführt werden um ihm mitzuteilen, dass die Aufnahme beendet werden soll. Das Warten auf das Ende der Übertragung kann entweder mit `pause` oder `input` gemacht werden.
- Alternativ können Sie auch die Methode `playblocking` verwenden, welche die Programmausführung pausiert, bis alle Samples abgespielt wurden. Fügen Sie nach Ihren Samples noch Nullen ein, um ein Abschneiden des Signals zu vermeiden.
- Beachten Sie, dass die play-Methode nicht direkt am Ende eines Programms stehen darf, da sonst Matlab am Ende des Programms das `audioplayer`-Objekt aufräumt, bevor alles Samples abgespielt wurden. Fügen Sie deshalb ein `pause` am Schluss ein.
- Beachten Sie die Frequenzen, die in den help-files spezifiziert sind. Die Erfahrung hat gezeigt, dass die Funktion `audiorecorder` je nach Frequenz dazu neigen abzustürzen. Sollte sich ein Prozess über `ctrl+c` nicht mehr abbrechen lassen, so kann Matlab im Terminal mit `killall matlab` beendet werden. Starten Sie danach Matlab neu.
- Die Objekte `audiorecorder` und audioplayer müssen mit den richtigen Parametern initialisiert werden. Wir verwenden hier die Werte $f = 22044$ für die Frequenz resp. $bits = 16$ für die Auflösung. Dies ergibt dann zusammen mit den Parametern $\tau_0 = 8$, $\tau_1 = 20$ und $\tau_s = 80$ aus dem zweiten Teil des Praktikums eine Trägerfrequenz von ungefähr $\frac{f}{\tau_0} \approx 2756$ Hertz bzw. $\frac{f}{\tau_1} \approx 1102$ Hertz und eine Bitrate von $\frac{f}{\tau_S} = 276$ bit/s.
- Beachten Sie, dass die Funktionen jeweils mit Spalten- und nicht mit Zeilenvektoren arbeiten.

Diese Funktionen zum Abspielen und Aufnehmen von Signalen müssen nun noch in unser Matlab-Programm integriert werden. Hierzu ist die Funktion `getaudiodata` hilfreich. Beachten Sie, dass Probleme auftreten können, wenn direkt nach dem Start der Aufnahme bereits abgespielt wird. Wirken Sie diesem Problem mit einer Zeitverzögerung entgegen, z.B. indem Sie die zu sendenden Sample-Vektoren am Anfang und am Schluss grosszügig mit Nullen ergänzen.

Aufgaben:

- Der von der Funktion `send` erzeugte Vektor mit dem abgetasteten Sendesignal ist über den Lautsprecher abzuspielen.

- Am gleichen PC ist das abgespielte Geräusch aufzunehmen und zu speichern. Wird die Datei in Stereo abgespeichert, wird eine nx2 Matrix anstatt eines nx1 Spaltenvektors ausgegeben.
- Das empfangene Signal soll in `receive` gespeichert werden (beachte die Dimension des Vektors). Erstellen Sie Plots nach allen Teilschritten, um möglichst anschaulich die Verarbeitung des Signals mitverfolgen und eventuelle Problem erkennen zu können.

3 Weiterführende Aufgaben

Nachdem Sie auf diese Weise eine erste Datenübertragung über die Luft durchführen können, sind nun folgende Untersuchungen und Weiterentwicklungen interessant:

3.1 Alphabeterweiterung

Im Moment ist unser Alphabet binär, wir können also nur ein Symbol 0 oder 1 pro Symboldauer senden. Eine Möglichkeit, die Bitrate zu erhöhen ohne die Symboldauer zu verkürzen, wäre neben den zwei vorhandenen Frequenzen f_0, f_1 mit weiteren Frequenzen zu arbeiten, die weitere Symbole repräsentieren, z.b. würden die Frequenzen $f_{00}, f_{01}, f_{10}, f_{11}$ jeweils 2 Bit pro Symbol übertragen. Das Alphabet ist dann statt $\{0, 1\}$ neu $\{00, 01, 10, 11\}$.

3.2 Fehlerkorrektur

Ein weiteres Ziel ist auch, die Zuverlässigkeit der Übertragung zu gewährleisten. Eine Methode dies zu erreichen haben Sie im 2. Teil des Praktikums mit dem Repetitionscode bereits angewendet. Diese Methode ist jedoch sehr teuer, da sich die zu übertragende Datenmenge bereits im einfachsten Fall verdreifacht. Eine weitere Möglichkeit sind die aus der Digitaltechnik bekannten Paritätsbits, die in jedem gesendeten Paket angeben ob sich eine gerade (even parity) oder ungerade (odd parity) Anzahl Einser darin befinden. Wird dabei nur am Ende jedes Paketes (sogenanntes Wort) ein Paritätsbit gesetzt ist erst eine Fehlerdetektion möglich, nicht jedoch eine Korrektur. Letztere wird erst machbar, wenn n Wörter zusammen wieder mit einem Paritätsbit versehen werden und so eine $(n+1)$x$(m+1)$-Matrix (bei einer Wortlänge von m Bits) konstruiert wird. Die Matrix enthält dann in der letzten Zeile bzw. letzten Spalte die Paritätsbits:

$$\mathbf{A} = \begin{pmatrix} b_{11} & \dots & b_{1m} & P_{11} \\ \vdots & \vdots & \vdots & \vdots \\ b_{n1} & \dots & b_{nm} & P_{n1} \\ P_{21} & \dots & P_{2m} & P_{n+1,m+1} \end{pmatrix}$$

Wird nun ein Bit falsch übertragen, kann es durch einen Vergleich der Einser und den erhaltenen Paritybits in der letzten Zeile und Spalte genau lokalisiert werden und damit, d der Wertebereich nur $[0, 1]$ ist, auch korrigiert werden. Werden zwei Bits in diesem Array falsch übertragen, kann dies immer noch in jedem Fall detektiert, aber nicht mehr in jedem Fall korrigiert werden. Bei drei und mehr Fehlern ist auch eine Detektion nicht mehr in jedem Fall möglich.

Eine in der Praxis häufig eingesetzte Methode zur Fehlerkorrektur ist diejenige mittels Hamming Code. Dabei werden - schlicht gesagt - nur gewisse Wörter als erlaubt definiert und falls ein nicht definiertes Wort eintrifft, so wird es demjenigen erlaubten, sogenannten Codewort, angeglichen, dem es am ähnlichsten ist. Dieses Vorgehen ist als *Prinzip des nächsten Nachbarn* bekannt. Eine wichtige Eigenschaft solcher Codes ist der *Abstand* zwischen zwei Codewörtern bzw. des Codes. Er ist definiert durch die Anzahl Bits, die geändert werden müssen, bis man das nächste Codewort erhält.

Als Beispiel soll eine Wortlänge von 3 Bits dienen: Codewörter seien $[000]$ und $[111]$. Der Abstand δ ist also $\delta = 3$. Wird nun das Wort $[010]$ empfangen wird es nach dem Prinzip des nächsten Nachbarn als $[000]$ interpretiert. Analog wird ein empfangenes Wort $[110]$ als Codewort $[111]$ interpretiert. Will man nun e Fehler korrigieren können, so muss der Abstand des Codes

$$\delta \geq 2e + 1$$

sein. Dies ist leicht ersichtlich, da jedes Codewort e Wörter davor und danach braucht, die man nur grade diesem Codewort zuordnen kann. Die Addition von 1 kommt daher, dass benachbarte Wörter bereits einen Abstand von 1 haben, sich also in einem Bit unterscheiden. Im obigen Beispiel sind also $e = (\delta - 1)/2 = 1$ Fehler korrigierbar.

Es gilt nun das Verhältnis der Datenbits und der Bits zur Fehlerkorrektur zu optimieren. Dieses wird durch die sogenannte *Coderate* beschrieben. Sie gibt an, welcher Anteil der übertragenen Bits tatsächliche Nutzdaten enthalten. Im Beispiel sind mit 3 Bits 2 mögliche Codewörter realisiert worden, was auch mit einem Bit ohne Fehlerkorrektur möglich gewesen wäre. Somit sind von diesen 3 Bits eines für die Datenübertragung und zwei dienen zur Fehlerkorrektur, und somit beträgt die Coderate in diesem Beispiel $1/3$.

3.3 Paketbildung zur Übertragung grosser Dateien

Bei der Übertragung grosser Dateien treten mehrere Probleme auf:

- Die Sampling-Frequenz von Mikrophon und Lautsprecher stimmt meist nicht ganz überein. Dies führt dazu, dass nicht alle Symbole schön synchronisiert sind.
- Die Länge der Keys (Präambel, Postambel) wird im Verhältnis zur Paketlänge klein und die Wahrscheinlichkeit, dass sie irgendwo zufällig auftreten wird immer grösser.

Aus diesem Grund werden Daten in einzelnen Datenpaketen kleiner Länge übertragen, wobei jedes Paket jeweils mit den Keys übertragen wird.

3.4 Übertragung von Dateien und Bildern

Beachten Sie, dass die folgenden Aufgaben z.T. grosse Rechenleistung erfordern. Arbeiten Sie deshalb Ihren `Matlab`-Code zuerst nochmals durch und optimieren Sie die Routinen mit `matlab`-gerechten Befehlen. Schleifenoperationen sind in `Matlab` generell wann immer nur möglich zu vermeiden und durch Matrizen- oder Vektoroperationen zu ersetzten. Für die Bildübertragung sollten Sie zudem höchstens ein Plot pro Übertragung ausgeben lassen, da sonst die Aufzeichnungen sehr lange dauern.

- Implementieren Sie einen Dateitransfer, indem Sie eine beliebige Datei binär lesen, die Bits übertragen und im Empfänger wieder in eine Datei schreiben. Hierzu sind die `matlab`-Funktionen `fopen`, `fread`, `fwrite` und `fclose` nützlich (beachte auch die Tipps unten).

Bemerkung: Eine Datei mit 1kB erzeugt ohne Fehlerkorrektur bereits eine Wav-Datei von etwa 30 Sekunden. Testen Sie die Routine erst mit einfachen Textdateien. Bei fehlerfreiem Ablauf können auch kleine Bilder ($<$ 1kB) übertragen werden. Besonders interessant ist die Übertragung von Bitmap-Dateien, da Übertragungsfehler - sofern sie nicht im Header der Datei sind - sofort durch falschfarbene Pixel sichtbar werden.

Noch einige nützliche Tipps zu einzelnen Schritten:

Eine Text- oder Bilddatei kann folgendermassen bitweise gelesen werden:

```
fid = fopen(filename, 'r');
bitsequence = fread(fid, 'ubit1')';
fclose(fid);
```

Das bitweise Schreiben in eine Datei erfolgt ähnlich wie das Lesen:

```
fid = fopen(filename, 'w');
fwrite(fid, bitsequence, 'ubit1');
fclose(fid);
```

Zum Versenden eignen sich kleine Textdateien (`.txt`, mit Textedior erstellen) mit wenigen Sätzen Text. Bei der Aufnahme sollten Sie unbedingt Amplitudensättigung am Mikrophon vermeiden. Wenn die Übertragung klappt, können Sie es auch mit kleinen Bitmaps (maximal

20x20 Pixel) ausprobieren. Ihr Code sollte so ausgelegt sein, dass man wahlweise die Codierung an- und ausschalten kann und auch die Coderate festlegen kann, falls die Übertragung ohne Codierung nicht erfolgreich ist.

Grundsätzlich können beliebige Dateitypen verschickt werden, bei grösseren Datenmengen dauert die Übertragung einfach entsprechend lange.