Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für Kommunikationstechnik
Prof. Dr. H. Bölcskei

P&S "Bits on Air"
3. Teil

Markus Gärtner & Felix Kneubühler

Revidierte Version vom 29. April 2022

1 Einleitung

Das Ziel dieses Praktikums ist die Realisierung einer drahtlosen Datenübertragung zwischen zwei PC-Stationen, wobei als Übertragungsmedium die Luft dient. Die Information wird senderseitig mittels eines Lautsprechers in die Luft transportiert und empfängerseitig mit einem Mikrofon aus dem Übertragungsmedium extrahiert. Nun ist bekannt, dass das menschliche Ohr nur zur Rezeption von Tönen mit Frequenzen zwischen ungefähr 20 Hertz und 20 Kilohertz fähig ist. Folglich werden auch Lautsprecher und Mikrofone nur für höchstens diesen Frequenzbereich dimensioniert. Die Übertragung der Information kann also nur über akustische Signale bestehend aus innerhalb eines eingeschränkten Bereichs liegenden Frequenzanteilen erfolgen. Die geeignete Aufbereitung des informationstragenden Signals vor dem Lautsprecher besteht aus der Bereitstellung des abgetasteten Signals in digitaler Form, gefolgt von einer Digital/Analog-Konvertierung. Umgekehrt muss das vom Mikrofon aufgenommene Signal im Empfänger-PC zuerst in eine digitale Form umgewandelt und danach die Information zurückgewonnen werden. In diesem Teil des Praktikums werden die digitale Aufbereitung und Auswertung behandelt und entsprechende Programmteile in `Matlab` implementiert.

Beim Programmieren sollten Sie vor allem darauf achten, dass Sie den Code so strukturieren, dass man ihn in den kommenden Teilen des PPS ohne Probleme wieder einsetzen kann. Das bedeutet, dass Sie alle relevanten Funktionen, die Sie aufrufen, in einem Verzeichnis speichern oder die Pfade der Dateien mit `Set Path` entsprechend setzen. Ausserdem sollten alle wichtigen Aufrufparameter auch in späteren Funktionen übernommen werden, die Sie in den kommenden Teilen des PPS entwickeln werden.

2 Modulation

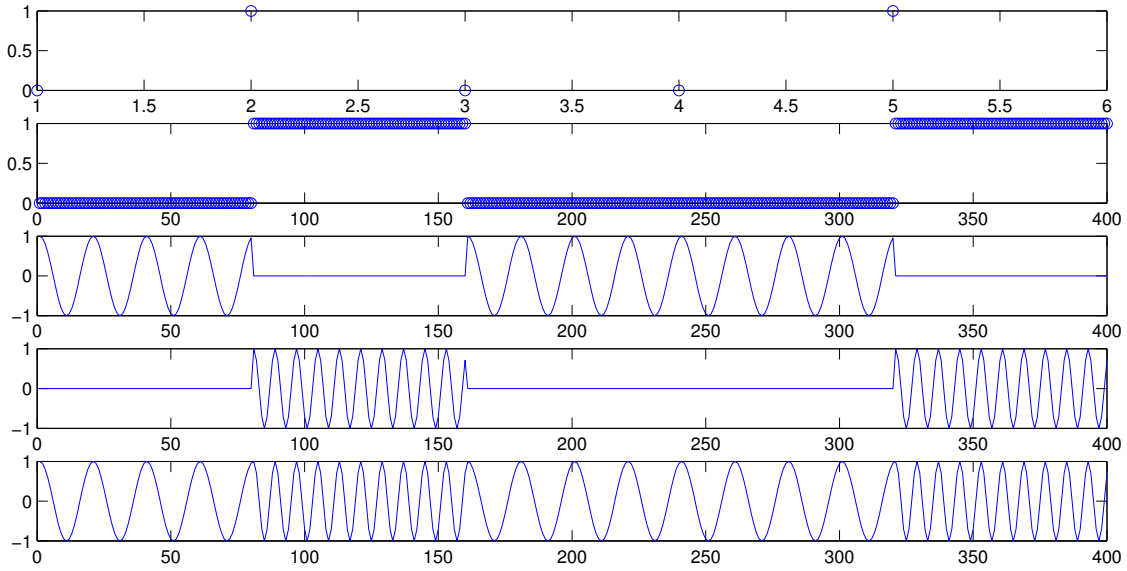

Abbildung 1: Modulation der Bitfolge $\{0, 1, 0, 0, 1\}$. Von oben nach unten:
① diskrete Bitsequenz $B[n]$ ② verlängerte Bitsequenz $b(t) = B[\lfloor \frac{t}{\tau_S} \rfloor]$
③ Träger $x_1(t) \cdot (1 - b(t))$ ④ Träger $x_2(t) \cdot b(t)$ ⑤ moduliertes Signal $y(t)$

Diskrete Signale können nicht direkt über die Luft übertragen werden, sondern sie müssen in Schwingungen umgewandelt werden, welche sich in der Luft ausbreiten können. Die Grundschwingung einer solchen Übertragung nennt man *Träger* (engl. *Carrier*).

$$x_i(t) = A \cos(2\pi f \cdot t)$$

Wobei f die Frequenz in Hertz ist. Da wir Einsen und Nullen übertragen wollen und diese am Empfänger wieder erkennen wollen, wählen wir für jedes Bit entweder eine Trägerfrequenz f_0 (falls das Bit 0 ist), oder f_1 (falls das Bit 1 ist) wobei $f_0 \neq f_1$. Man nennt diese Art der Modulation *Frequency Shift Keying (FSK)*. Zusätzlich wählen wir eine *Symboldauer* τ_S, welche angibt welche Zeitdauer für die Übertragung eines Symbols verwendet wird, bevor das nächste gesendet wird. Da wir wir in unserem Beispiel nur zwei verschiedene Frequenzen benutzen, entspricht ein Symbol genau einem Bit, also einer Eins oder einer Null (Binary-FSK). Im Allgemeinen kann ein Symbol aber M Bits repräsentieren, wir müssten dann bei FSK entsprechend 2^M verschiedene Symbole / Frequenzen verwenden, um die entsprechende Anzahl Bits darzustellen.

Damit das Signal stetig ist, muss die Symboldauer τ_S ein ganzzahliges Vielfaches der beiden Schwingungsdauern $\tau_0 = \frac{1}{f_0}$ und $\tau_1 = \frac{1}{f_1}$ sein. Ein unstetiges Signal wird eine sehr grosse Bandbreite belegen und ist deshalb im Allgemeinen nicht erwünscht.

Sei nun $\{B_n : n = 0, 1, \ldots, N-1\}$, $B_n \in \{0, 1\}$ die zu übertragende Bitsequenz der Länge N, so kann das modulierte Signal $y(t)$ wie folgt geschrieben werden

$$y(t) = \sum_{n=0}^{N-1} \cos\Big(2\pi\big(f_1 \cdot B_n + f_0 \cdot (1 - B_n)\big) \cdot t\Big) \cdot p(t - n\tau_S),$$

wobei

$$p(t) = \begin{cases} 1, & t \in [0, \tau_S) \\ 0, & t \notin [0, \tau_S). \end{cases}$$

Alternativ kann auch geschrieben werden:

$$y(t) = \sum_{n=0}^{N-1} \Big(B_n \cdot \cos(2\pi f_1 \cdot t) + (1 - B_n) \cdot \cos(2\pi f_0 \cdot t)\Big) \cdot p(t - n\tau_S).$$

Aufgaben:

- Beschreiben Sie in Worten, was die oben gezeigte Formel (beziehungsweise die alternative Schreibweise) tut. (Tipp: Finden Sie heraus, was $p(t - n\tau_S)$ für verschiedene n und t darstellt).

- Erstellen Sie eine Funktion `function y = modulate(b)`, welche ein mit der Bitsequenz `b` moduliertes Signal zurückgibt. Dabei soll $\tau_S = 80$, $\tau_0 = 20$ und $\tau_1 = 8$ gewählt werden. Erstellen Sie zu Debugging-Zwecken eine Grafik, die ähnlich wie Abbildung 1 aussieht.

  Wichtig: Versuchen Sie nicht, obige Gleichung eins zu eins mit einer for-Schleife zu implementieren, sondern nutzen Sie die Matrix-Operationen von Matlab. Abbildung 1 kann hierbei sehr hilfreich sein.

  Tipp: Die Parameter $\tau_S = 80$, $\tau_0 = 20$ und $\tau_1 = 8$ werden von mehreren Funktionen in unserem System verwendet. Sie können deshalb zentral in einer .mat-Datei gespeichert werden. Der Befehl `save('data.mat','tauS', 'tau0', 'tau1')` speichert die im Workspace gespeicherten Variablen, in die Datei `data.mat`. Alternativ können Sie mit einem Rechtsklick auf den Workspace den gesamten Workspace in einer .mat-Datei speichern. Achten Sie dabei darauf, dass Sie ungenutzte Variablen davor aus dem Workspace entfernen. Später können einzelne Variablen verändert werden z.B. mittels `tauS=40; save('data.mat', 'tauS', '-append')`, oder direkt im Workspace. Vergisst man das `'-append'` werden alle anderen Variablen im .mat file gelöscht! Mit der Funktion `load('data.mat')` können die gespeicherten Variablen geladen werden. Sie können auch analog zum Befehl `save` nr bestimmte Variablen laden. Die Befehle für die Plots heissen `subplot`, `figure` und `plot(y,'bo')`.

3 Demodulation

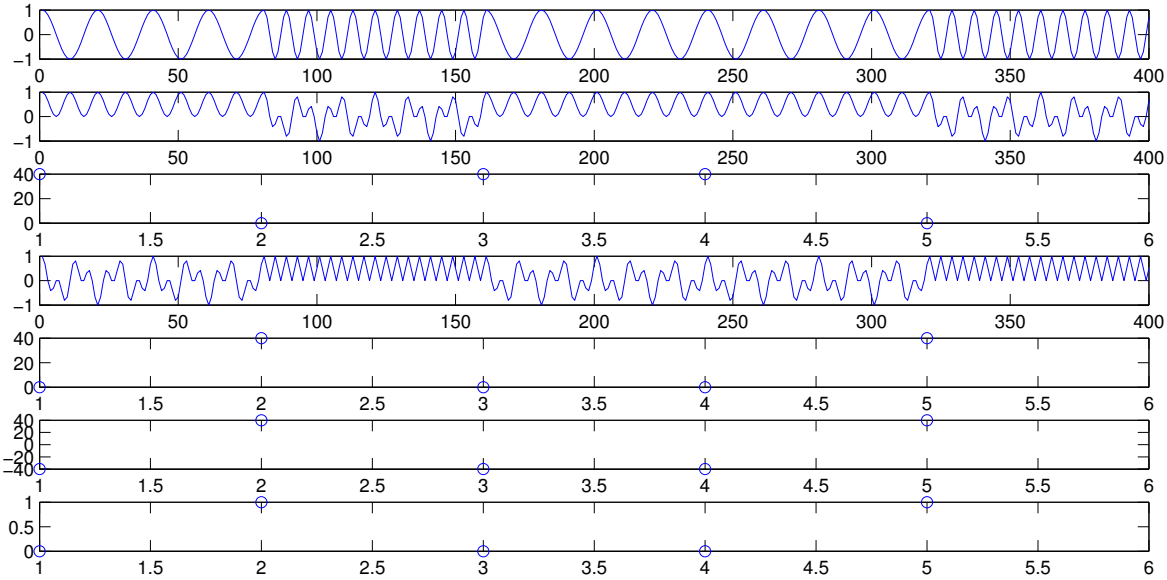

Abbildung 2: Einzelne Schritte der Demodulation der Bits aus Abbildung 1:
① Empfangenes Signal ② Multiplikation mit $x_1(t)$ ③ $z_1(t) =$ Korrelation mit $x_1(t)$
④ Multiplikation mit $x_2(t)$ ⑤ $z_2(t) =$ Korrelation mit $x_2(t)$
⑥ Differenz zwischen $z_2(t)$ und $z_1(t)$ ⑦ Delektierte Bitsequenz

Das akustische Signal wird bei der Empfängerstation vom Mikrofon empfangen und steht in abgetasteter Form zur Verfügung. Die quarzstabilisierten Taktgeber in den PCs stellen sicher, dass die Abtastfrequenzen in Sender und Empfänger übereinstimmen. Die übertragenen Daten werden mit dem in Abbildung 2 beschriebenen Verfahren rekonstruiert. Zuerst wird durch eine Korrelation des empfangenen Signals $r(t)$ mit den beiden möglichen Frequenzen f_1 und f_2 die Bits aus dem hochfrequenten Signal extrahiert, was als *Demodulation* bezeichnet wird. Für jedes Datensymbol n ergibt sich ein Wert $z_i[n]$, welcher die Korrelation des i-ten Trägers mit dem empfangenen Symbol angibt, das zur Zeit $t = n \cdot \tau_S$ beginnt. Darauf basierend wird dann für jedes übertragene Bit ein Schätzwert $\hat{B}_n$ gebildet (*Detektion*).

Die Korrelation zwischen dem empfangenen Signal $r(t)$ und dem t-ten Träger $x_i(t)$ während der Dauer des n-ten Bits ist

$$z_i[n] = \int r(t) \cdot x_i(t) \cdot p(t - n\tau_S)dt,$$

wobei

$$p(t) = \begin{cases} 1, & t \in [0, \tau_S) \\ 0, & t \notin [0, \tau_S). \end{cases}$$

Aufgaben:

- Berechnen Sie von Hand $z_i[n], i \in \{0,1\}$, für den rauschfreien Kanal ($r(t) = y(t)$) für ein einzelnes übertragenes Bit $B_0 \in \{0,1\}$. Nehmen Sie an, dass τ_0, τ_1 ganzzahlige Teiler von τ_S sind (das heisst, die Symboldauer ist ein ganzzahliges Vielfaches der einzelnen Schwingungsdauern). Grafische Darstellung des Integrationsgebiets ($r(t) \cdot x_i(t))|t \in [0, \tau_S]$) kann die Aufgabe vereinfachen. Verifizieren Sie die Resultate mittels Matlab oder Taschenrechner. Es ist sehr wichtig, dass τ_0, τ_1 ganzzahlige Teiler von τ_S sind.

  Tipp: Es gelten folgende Gesetze:

$$\begin{aligned}
\int \cos(a \cdot t) \cdot \cos(b \cdot t) dt &= \frac{\sin(a \cdot t - b \cdot t)}{2a - 2b} + \frac{\sin(a \cdot t + b \cdot t)}{2a + 2b} \\
\sin(x + y) &= \sin(x)\cos(y) + \sin(y)\cos(x) \\
\sin(x - y) &= \sin(x)\cos(y) - \sin(y)\cos(x) \\
\cos^2(a \cdot t) &= \frac{1}{2}(\cos(2a \cdot t) + 1) \\
\int \cos^2(a \cdot t) &= \frac{\sin(2a \cdot t)}{4a} + \frac{t}{2}
\end{aligned}$$

- Wie muss nun die geschätzte Bitsequenz $\hat{B}_n$ von $z_i[n]|i \in \{0,1\}$ abhängen, um eine optimale Schätzung $\hat{B}_n = B_n$ zu erhalten?

- Implementieren Sie eine `Matlab`-Funktion `function bhat = demodulate(r)`, welche mit der im Zeilenvektor `r` enthaltenen abgetasteten Version des Signals $r(t)$ gemäss Abbildung 2 die Demodulation und Detektion durchführt und die geschätzten Bits zurückgibt. Der Rückgabeparameter `bhat` soll ein Zeilenvektor mit $\lfloor$`length(r)`$/\tau_s \rfloor$ Elementen sein. Als Test für diese Funktion kann verifiziert werden, ob für jede beliebige Bitfolge `demodulate(modulate(b)) = b` gilt. Geben Sie zusätzlich auch einen Plot ähnlich Abbildung 2 aus.

- Überlegen Sie sich, wie man den Durchsatz erhöhen könnte, also mehr Information (bzw. Bits) in derselben Zeit über den Kanal senden könnte.

4 Aufräumen

Inzwischen haben Sie sich schon relativ gut in Matlab eingearbeitet und sich an die Syntax gewöhnt. Dies ist der perfekte Zeitpunkt, den bisher erstellten Code nochmals kurz zu überarbeiten. Einerseits für Optimierungen (ersetzen Sie for-schleifen) und Benutzerfreundlichkeit (Was ist Eingabeparameter? Was kann ich in eine Datei auslagern?) als auch um den Code etwas zu kommentieren. Dabei sollten Sie vor allem Dinge ergänzen, die Ihnen später

selbst helfen, also bei einer Funktion angeben, ob gewisse Anforderungen an den Input existieren (z.b. in `demodulate`), in welcher Reihenfolge die Eingabeargumente sind und was genau man rausgibt. Versuchen Sie das ganze Programm wie in Abbildung 3 aufzuteilen und kontrollieren Sie, ob die Ergebnisse eines Durchlaufs von `loop` sinnvoll sind.

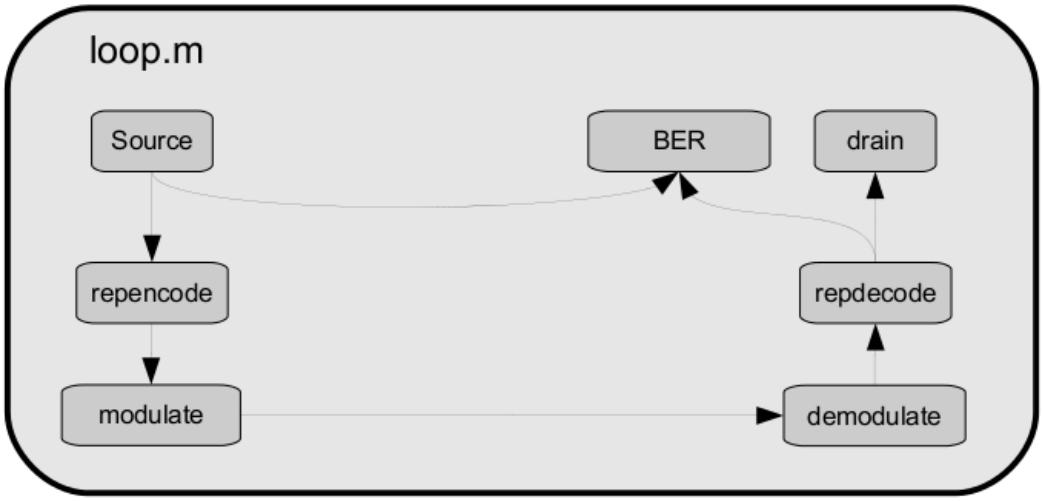

Abbildung 3: So soll der generelle Flow des Programmes ablaufen.

Abschliessend können Sie das Programm auf Fehlerstabilität testen. Da wir nun ein moduliertes Signal haben, geht dies nicht mehr mit dem BSC channel aus der letzten Lektion (dort werden diskrete Werte benutzt). Wir modellieren den Kanal deshalb als $r(t) = y(t) + w(t)$, wobei $w(t)$ weisses Rauschen ist. Dieses Kanalmodell nennt man auch AWGN-Kanal (additive white gaussian noise channel). Suchen Sie mittels der Matlab-Funktion `lookfor` nach einer Funktion, die Ihnen ermöglicht einen AWGN-Kanal zu simulieren. Testen Sie das Programm mit verschiedenen SNR. SNR ist die Signal to Noise Ratio, also das Verhältnis von Signalenergie zu Energie des Störsignals. Verändern Sie zu Testzwecken auch die abgespeicherten Schwingungsdauern, um etwas die verschiedenen Effekte zu sehen, und betrachten Sie jeweils die Grafiken.