Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für Kommunikationstechnik
Prof. Dr. H. Bölcskei

P&S "Bits on Air"
6. Teil

Revidierte Version vom 9. Mai 2022

1 Einleitung

Das *Automatic Packet Reporting System* (APRS) ist ein Kommunikationsprotokoll zum Senden und Empfangen von Nachrichten. Es wird grösstenteils im Bereich des Amateurfunks zur Übermittelung von kleinen Nachrichten, GPS- oder Wetterdaten eingesetzt.

APRS verwendet als Modulationsform *Frequency Shift Keying* (FSK), also das gleiche Verfahren wie wir in diesem P&S verwenden. Das Ziel des heutigen Nachmittags ist es deshalb, einen APRS Empfänger zu programmieren. Dieser funktioniert im Grunde fast gleich wie das im bisherigen P&S implementierte Programm es ist jedoch noch etwas komplexer, da es beispielsweise eine Prüfsumme oder eine spezielle Codierung (`NRZI`) verwendet. Deshalb wurde das Gerüst des Demodulators bereits erstellt und Sie werden nur noch die Kernfunktionen implementieren.

Der Einfachheit halber unterscheiden wir hier nicht zwischen APRS und dem zugrunde liegenden Protokoll AX.25 (vgl. [1]), sondern sprechen im Folgenden nur von APRS.

APRS Pakete werden per Funk auf der Frequenz 144.800 MHz gesendet. Da wir in diesem P&S jedoch keine Antenne zur Verfügung haben, verwenden wir `http://websdr.org`. Hier befindet sich eine Liste von Stationen welche auf der genannten Frequenz empfangen und das Signal ins Audio-Basisband übertragen, so dass es im hörbaren Bereich vorliegt, ähnlich wie im bisherigen P&S. Dieses Audiosignal wird nun über das Internet gestreamt.

Ebenfalls interessant ist die Webseite `https://aprs.fi`, hier laden spezielle Stationen, welche über einen Internetanschluss verfügen (sogenannte iGates) ihre empfangenen Pakete hoch.

Aufgaben:

- Verschaffen Sie sich einen kurzen Eindruck von APRS. Gehen Sie dazu auf die Website `https://aprs.fi`. Was für verschiedene Nachrichtentypen gibt es? Welche Signale gibt es im Raum Zürich?
- Suchen sie auf `http://websdr.org` nach der Station Friedrichshafen und versuchen sie das empfangene Audiosignal zu hören. Stellen sie dazu auf das `2m Band` um und klicken sie dann in der Spektrumsanzeige auf `APRS` (144.800 MHz).

2 Implementation

Bevor wir auf die genaue Implementation eines APRS-Empfängers eingehen, soll zuerst ein Gesamtüberblick vermittelt werden. Betrachten Sie dazu den Programmablauf nach Abb. 1 auf der nächsten Seite.

Als erstes wird ein `Audiorecorder` aufgesetzt. Dieser speichert die via Streaming empfangenen Daten kontinuierlich in einem Zwischenspeicher. Die gelesenen Samples werden mit den Signalen $s_0(t)$ und $s_1(t)$ korreliert. Anschliessend wird nach einer Präambel (im APRS Kontext Flag genannt) gesucht. Wird kein Flag gefunden werden neue Samples von der Soundkarte eingelesenen und der Prozess wiederholt sich. Wird jedoch ein Flag gefunden, wird für eine bestimmte Zeit, welche der Sendedauer von 500 Bytes entspricht (dies ist etwas länger als die Maximale Paketlänge in APRS), alles zwischengespeichert und weiter verarbeitet. Zuerst wird in der Rahmensynchronisation das Flag abgeschnitten, danach werden die einzelnen Samples zu Symbolen demoduliert und zum Schluss werden alle APRS spezifischen Codierungen rückgängig gemacht, so dass die Nachricht in von Menschen lesbarem Format vorliegt.

Heute werden Sie die Funktionen schreiben, welche die Korrelation bestimmt (vgl. Abschnitt 2.1). Weiter können Sie wahlweise die Funktion zur Flag Erkennung selber implementieren oder die bestehende Funktion verwenden (vgl. Abschnitt 2.2 auf Seite 7). In Abschnitt 3 auf Seite 10 werden Sie schliesslich die verschiedenen Elemente des Programms zusammenführen und ein Signal in Echtzeit empfangen und dekodieren.

2.1 Einlesen und korrelieren

Obwohl APRS ebenfalls FSK verwendet, gibt es im Vergleich zu den bisherigen Aufgaben wichtige Änderungen, vgl. [2].

APRS verwendet wie wir am 3. Nachmittag des P&S' die Signalform $(i = 0, 1)$ gegeben durch:

$$s_i(t) = \begin{cases} A\cos(2\pi f_i t), & 0 \leq t \leq T_b \\ 0, & \text{sonst} \end{cases}, \tag{1}$$

wobei $s_0(t)$ (bzw. $s_1(t)$) für das Symbol `0` (bzw. `1`) steht. Allerdings sind die Frequenzen gewählt als:

$$f_0 = 2200\,\text{Hz}$$
$$f_1 = 1200\,\text{Hz}$$

Da eine Symboldauer $T_b \coloneqq \frac{1}{f_1} \approx 833\,\mu\text{s}$ ist und somit kein Vielfaches einer Periode von $s_0(t)$ beträgt, hat jedes gesendete Symbol eine andere Phase (vgl. Abb. 2 auf Seite 4). Das Problem

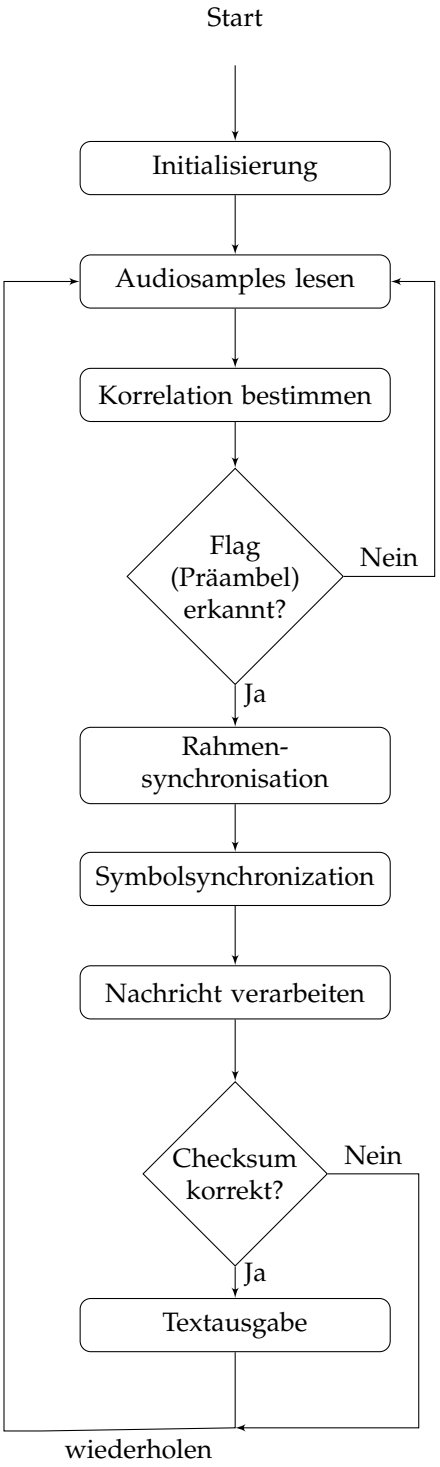

Abbildung 1: Programmablauf zum Empfangen von APRS-Signalen

der unbekannten Phase wäre durch das Akzeptieren von Unstetigkeiten beim Übergang zwischen den einzelnen Symbolen einfach zu lösen. Das wird in APRS jedoch nicht realisiert, da Unstetigkeiten zu einer erheblichen Vergrösserung der Bandbreite führen.

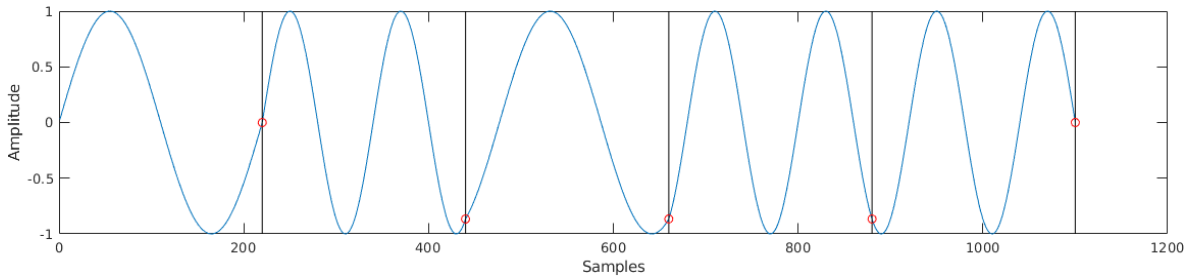

Abbildung 2: Die Nachricht `10100` als Signalgewertet nach (1) repräsentiert. Die Phasensprünge sind rot markiert.

Dies führt zu einigen neuen Herausforderungen bei der Implementierung: Anders als mit dem bisherigen Programm kann die Symbolsynchronisation nicht zu Beginn der Entschlüsselung durchgeführt werden, da jedes Symbol eine andere Phase hat. Deshalb muss ein anderer Weg gefunden werden. Leider ist es nicht mehr möglich, die Präambelsequenz (Flag) zu modulieren und im empfangenen Signal mittels `xcorr` danach zu suchen, wie es im Nachmittag 4 gemacht wurde. Es gibt nämlich keine Vorschrift, mit welchem Phasenversatz das erste Symbol des Flags gesendet wird. Will man also Nachrichten eines fremden Senders entschlüsseln, dessen Phasenversatz man nicht kennt, muss die Position des Flags auf andere Weise bestimmt werden.

Wir berechnen dazu die Korrelation zwischen der empfangenen Nachricht und den beiden Cosinussignalen $s_0(t)$ und $s_1(t)$ mit beliebigem Phasenversatz. Wie das funktioniert, folgt später. Damit kann nun für jedes Sample ein Wert, im Folgenden `symbolDecision` genannt, berechnet werden. Dieser bestimmt, ob das Sample eher zu $\cos(2\pi f_1 t)$ (positive Werte) oder zu $\cos(2\pi f_0 t)$ (negative Werte) gehört (vgl. Abb. 3 auf der nächsten Seite).

Nun wollen wir für jedes Sample bestimmen, ob es eher zu dem Signal $s_0(t)$ oder $s_1(t)$ gehört. Dafür stellen wir die beiden Hypothesen

$$\begin{aligned} H_0 &: \quad \text{Das Sample korrespondiert eher zu } s_0(t). \\ H_1 &: \quad \text{Das Sample korrespondiert eher zu } s_1(t). \end{aligned}$$

auf. Es gilt nun für jedes Sample zu prüfen, welche Hypothese akzeptiert und welche verworfen wird.

Wie oben bereits erwähnt, kann die Korrelation nicht so einfach berechnet werden wie im Nachmittag 3, da die Phasen unbekannt sind. Deshalb bedienen wir uns eines mathematischen Tricks, der komplexen Korrelation. Hier wird nicht mit $\cos(2\pi f_i t)$ korreliert, sonder mit dem komplexen Signal $e^{2\pi j f_i t}$. Die zu berechnende Korrelation für ein bestimmtes Sam-

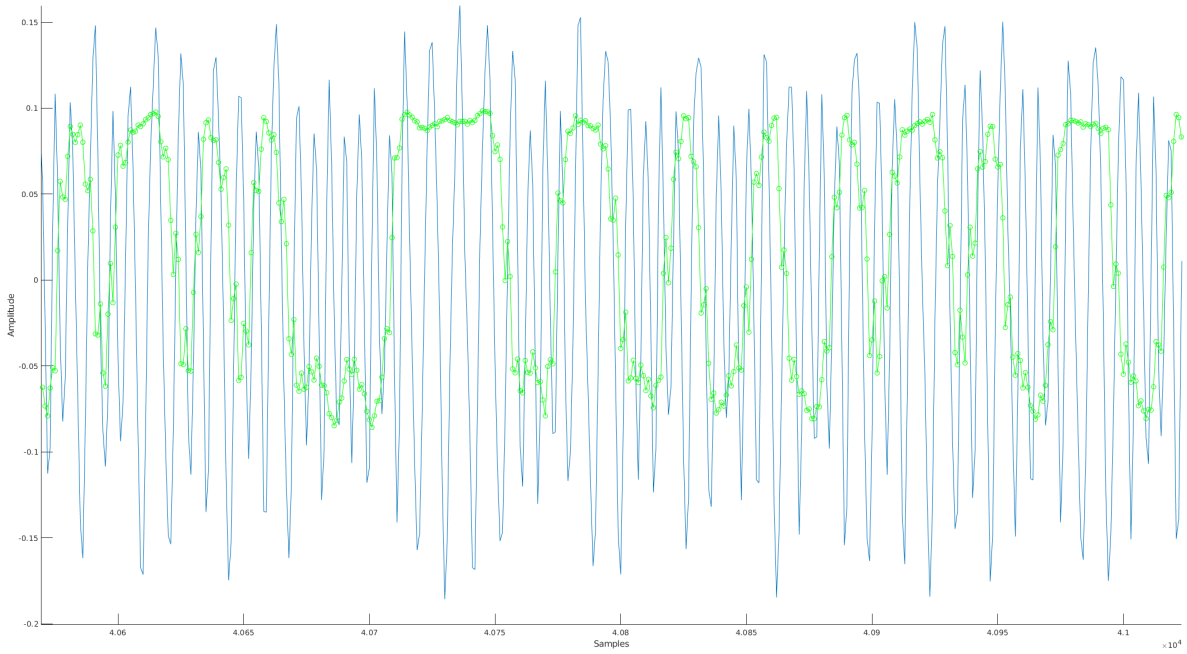

Abbildung 3: APRS Nachricht als Audiosignal (blau) und `symbolDecision` (grün) normalisiert auf den Bereich $[-0.1, 0.1]$

ple ist ähnlich wie in Nachmittag 4 definiert als:

$$\varphi_i = \lim_{T\to\infty} \frac{1}{T} \int_{-\infty}^{\infty} x(t) e^{2\pi j f_i t}\, d\, t \qquad i = 0, 1$$

wobei

$$x(t) = cos(2\pi f_k t + \theta)$$

Mit den trigonometrischen Identitäten

$$\cos(2\pi f_k t + \theta) = \cos(2\pi f_k t)\cos\theta - \sin(2\pi f_k t)\sin\theta$$
$$e^{2\pi j f_i t} = \cos(2\pi f_i t) + j\sin(2\pi f_i t)$$

wird der Integrand zu

$$\cos(2\pi f_k t)\cos(2\pi f_i t)\cos(\theta) + j\cos(2\pi f_k t)\sin(2\pi f_i t)\cos(\theta)$$
$$-\sin(2\pi f_k t)\cos(2\pi f_i t)\sin(\theta) - j\sin(2\pi f_k t)\sin(2\pi f_i t)\sin(\theta)$$

Aufgrund der Linearität des Integrals kann das Integral in vier Summanden aufgeteilt werden. Die Mischtherme mit $\cos(2\pi f_{\{i,k\}} t)\sin(2\pi f_{\{i,k\}} t)$ verschwinden falls $i = k$, da $\sin$ und $\cos$ bei gleicher Frequenz zueinander orthogonal sind und die entsprechenden Integrale so-

mit 0 ergeben. Übrig bleibt der Term

$$z_i = \lim_{T\to\infty} \left(\cos(\theta) \underbrace{\frac{1}{T}\int_{-\infty}^{\infty} \cos(2\pi f_k t)\cos(2\pi f_i t)\,dt}_{=1,\text{ falls } i=k} + \sin(\theta) \underbrace{\frac{1}{T}\int_{-\infty}^{\infty} j\sin(2\pi f_k t)\sin(2\pi f_i t)\,dt}_{=1,\text{ falls } i=k} \right)$$

wobei z_i die komplexe Korrelation von $x(t)$ und $e^{2\pi j f_i t}$ ist. Wir interessieren uns jedoch nur für den Betrag von z_i, dieser ist mit dem genormten und perfekt korrelirenden Signal $x(t)$ definiert durch

$$|z_i| = \underbrace{\cos(\theta)^2 + \sin(\theta)^2}_{1} = 1$$

Wir sehen nun, dass der Betrag von z_i unabhängig von θ ist und somit vom Phasenversatz. Er hängt nun nur noch von der Amplitude und dem effektiven Korrelationswert ab. Es reicht also einfach die Korrelation mit $\cos(2\pi f_i t)$ und $\sin(2\pi f_i t)$ separat zu berechnen und sie dann zu einer komplexen Zahl z zusammen zu fügen (siehe Abb. 4).

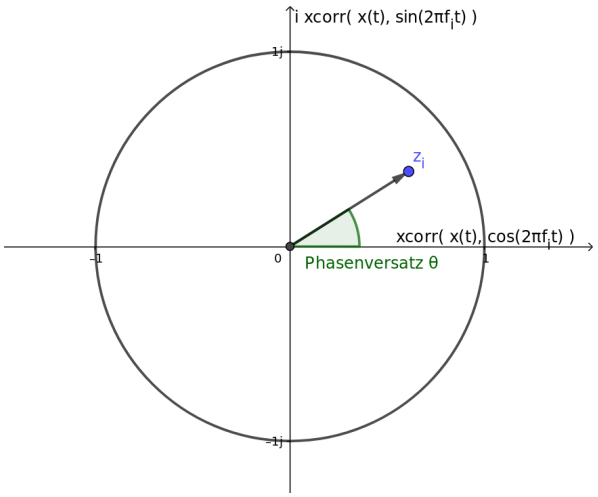

Abbildung 4: Komplexe Korrelationswerte

Wenn man nun z_i für beide Frequenzen berechnet und die beiden erhaltenen Werte voneinander abzieht und normalisiert auf das Intervall $[-1, 1]$, erhält man den Wert von `symbol-Decision`.

Dann wird jeweils folgende Hypothese akzeptiert:

$$\begin{aligned} H_0, &\qquad \text{falls } |z_0| > |z_1| \\ H_1, &\qquad \text{falls } |z_0| < |z_1| \end{aligned}$$

Aufgaben:

- Implementieren Sie im Template `decideCorrelationStudents.m` die Funktion `decideCorrelationStudents`, welche aus zuvor bereits gefilterten Audiosamples den `symbolDecision` Vektor erstellt.
- Der Vektor `symbolDecision` soll jedem Sample das vermutete Signal zuordnen. Korrespondiert also ein Sample perfekt zu dem Signal $s_1(t)$ (bzw. $s_0(t)$), so soll der `symbolDecision` den Wert 1 (bzw. -1) haben. Korrespondiert das Sample zu keinem der beiden Signalen, so soll der Wert nahe bei Null sein.

Hinweise:

- Nutzen Sie dafür die komplexe Korrelation.
- Beachten Sie die Länge der Ausgabe von `xcorr`.
- Achten Sie darauf, dass `symbolDecision` am Ende der Funktion wieder die gleiche Länge hat wie die eingelesenen Samples.
- Lesen sie die Kommentare in der vorgefertigten Funktion.

Testing:

- Öffnen Sie das Skript `Testbench.m`. Im Bereich *set parameters* können Sie einstellen, welche Tests durchgeführt werden sollen.
- Führen Sie Test 1 und 2 durch, die weiteren Tests folgen in der nächsten Aufgabe. Falls Sie diese zeitlich nicht mehr drin liegt sind sie an dieser Stelle fertig mit dem Programmierteil und können nun versuchen eine echte APRS Nachricht zu entschlüsseln. Führen Sie dazu die Anweisungen am Ende des Dokuments aus.

2.2 Flag erkennen: Rahmensynchronisation

Um eine Nachricht dekodieren zu können, ist das Wissen um die Struktur elementar. Es soll deshalb an dieser Stelle ein kurzer Überblick darüber gegeben werden. Betrachten Sie dazu Abb. 5. Dieser Teil ist optional.

Flag	Address	Control	PID	Info	FCS	Flag
0111 1110	112/224 Bits	8/16 Bits	8 Bits	$N \cdot 8$ Bits	16 Bits	0111 1110

Abbildung 5: Die Struktur einer Nachricht im APRS. N steht für eine natürliche Zahl.

Da im Folgenden auf die Erkennung der Flags fokussiert wird, seien die anderen Felder nur kurz erwähnt:

Address enthält sowohl die Adresse des Senders als auch die des Adressaten.

Control identifiziert den Nachrichtentyp.

PID (*Protocol Identifier*) spezifiziert das verwendete Protokoll.

Info enthält die eigentliche Nachricht beliebiger Länge.

FCS steht für *Frame Check Sequence* und ist eine Prüfziffer.

Eine Schwierigkeit in der Nachrichtenverarbeitung liegt in der Erkennung des Anfangs und Endes einer Nachricht. Dazu verwendet APRS (ähnlich wie in diesem P&S in Nachmittag 4) eine Präambel (*Flag*). Diese ist immer gleich und gegeben als `0x7E` bzw. `0111 1110`. Zum Anzeigen des Endes der Nachricht verwendet APRS eine mit der Präambel identische Postambel.

In der vorherigen Aufgabe wurden den Audiosamples bereits Werte zwischen -1 und 1 zugeordnet, welche jeweils die Zugehörigkeit zu s_0 beziehungsweise s_1 zeigen. Um die Position eines Flags zu finden, können Sie also die Korrelation zwischen `symbolDecision` und dem Flag bestimmen. Überschreitet der Korrelationswert eine bestimmte Schwelle, hier Threshhold genannt, befindet sich das Flag an dieser Stelle. Es ist allerdings zu beachten, dass die Nachricht *Non-Return-to-Zero-Inverted* (NRZI) gesendet wird. In NRZI wird eine zu sendende Null als Übergang zwischen 0 und 1 codiert und eine zu sendende Eins als halten des vorherigen Bits (vgl. Abb. 6 auf der nächsten Seite).

$$\begin{aligned} \ldots 1 \ldots \quad &\longrightarrow \quad \{\ldots 00 \ldots, \ldots 11 \ldots\} \\ \ldots 0 \ldots \quad &\longrightarrow \quad \{\ldots 01 \ldots, \ldots 10 \ldots\} \end{aligned}$$

Wir können deshalb nicht nach der Sequenz `0111 1110` suchen, sondern müssen nach den NRZI Flags suchen. Da das Umkehren von NRZI nicht eindeutig ist, muss nach zwei Flags gesucht werden. Diese sind gegeben als:

$$\begin{aligned} \texttt{Flag}_{\texttt{NRZI,0}} &= 1\ 0000\ 000\ 1 \\ \texttt{Flag}_{\texttt{NRZI,1}} &= 0\ 1111\ 111\ 0 = \overline{\texttt{Flag}_{\texttt{NRZI,0}}} \end{aligned}$$

In der Praxis werden oftmals mehrere Flags ($\approx$ 5 - 100) am Anfang der Nachricht gesendet. Wir werden deshalb auch gleich auf die Korrelation drei aneinander gehängter Flags prüfen.

Dass jeweils mehrere Flags zu Beginn einer Nachricht gesendet werden, verkompliziert die Sache etwas. Wir wollen deshalb nicht jedes Flag als Beginn einer Nachricht ansehen, sondern nur jeweils ein Flag in einer Sequenz aufeinander folgenden Flags. Wir führen deshalb noch einige Variablen mit. Die erste ist `flagPositions`, sie ist ein Vektor in welchem pro Flagsequenz jeweils der Index des Flags mit dem höchsten Korrelationswert gespeichert wird. Die dazugehörigen Korrelationswerte werden im Vektor `flagValues` gespeichert.

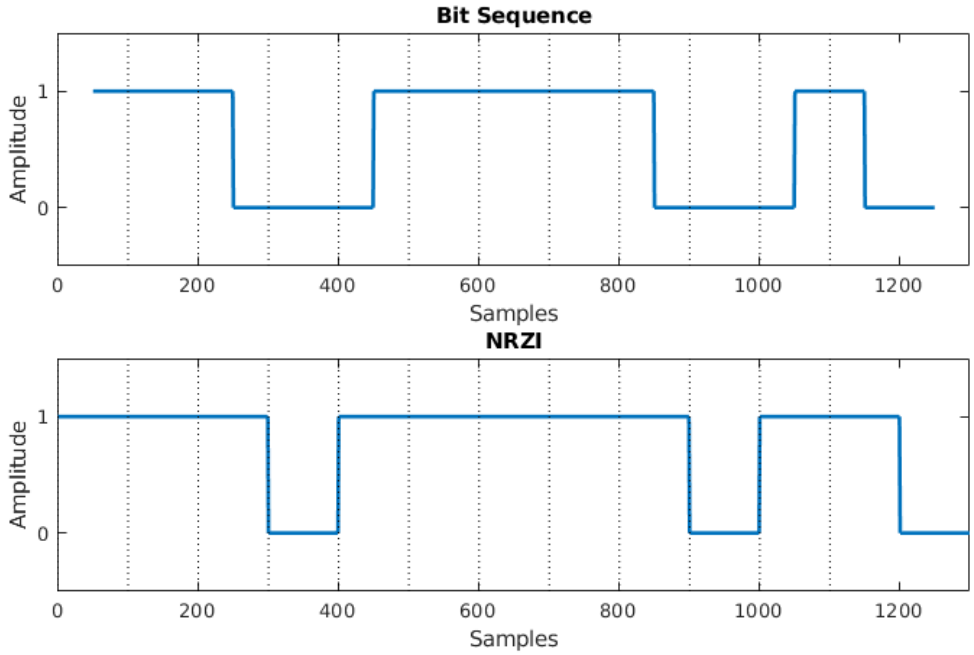

Abbildung 6: NRZI Codierung

Um zu bestimmen ob wir uns noch in der selben Flagsequenz befinden oder bereits den Beginn einer neuen Nachricht sehen, führen wir noch die Variable `lastFlagPosition` mit. Hier wird der Index gespeichert, an welchem der Korrelationswert zuletzt höher war als der Threshold.

Erreicht nun ein Korrelationswert einen höheren Wert als der Treshold (dies wird im Folgenden als „Match“ bezeichnet), so wird `lastFlagPosition` aktualisiert. Falls wir zuvor lange keinen Match hatten, d.h. eine arbiträre Mindestzeit, hier die Sendedauer von 10 Bytes, überschritten wurde gehen wir davon aus, dass wir uns in einer neuen Flagsequenz befinden. Das bedeutet, dass eine neue Nachricht beginnt. Deshalb werden nun die neuen Werte hinten an die Arrays `flagPositions` und `flagValues` angefügt. Wurde die Mindestzeit nicht überschritten, so muss bei besserem Korrelationswert `flagPositions` und `flagValues` aktualisiert werden.

Aufgaben:

- Schreiben Sie ausgehend von der vorherigen Aufgabe eine Funktion, welches aus dem `symbolDecision` die Position des besten Flags bestimmt. Verwenden Sie dazu die vorgefertigte Funktion `checkFlagOnlineStudents`

Vorgehen:

- Skizzieren Sie sich das oben beschriebene Verfahren auf Papier. Dies sollte die Aufgabe wesentlich vereinfachen.

- Überlegen Sie sich, wie Sie die Korrelation von `symbolDecision` mit `flagExtendedNRZI` berechnen können. Beachten Sie die Länge des Rüchgabevektors.
- Implementieren Sie die beiden Fallunterscheidungen in der Iterationsschleife, um den Anfang einer neuen Flagsequenz oder den höchsten Korrelationswert zu erkennen.

Hinweise:

- Die Vervielfachung der Flags wurde bereits implementiert, da es etwas verwirrend sein kann: Die Flags werden nicht einfach aneinander gehängt, sondern sie teilen sich aufgrund der NRZI Codierung jeweils ein Bit. Verwenden Sie `flagExtendedNRZI` für ihre Korrelation.
- Beachten Sie die Länge des von der Funktion `xcorr` zurückgegebenen Vektors!
- Überlegen Sie sich, wie mit der Uneindeutigkeit der Flags (NRZI) umgegangen werden kann. *Tipp:* Korreliert eine Nachricht perfekt mit einem Flag, so korreliert sie negativ nicht mit dem anderen.
- Lesen sie die Kommentare in der vorgefertigten Funktion.

Testing:

- Führen Sie wie zuvor die Tests aus, dieses Mal Test 3 - 5
- Falls die Tests funktionieren können Sie nun das Main Programm ausprobieren. Öffnen Sie dazu `BoAPRS.m` und passen Sie im obersten Abschnitt *Configuration* Ihre Einstellungen an. Verändern Sie am Rest des Programms nichs. Setzen Sie `online` auf 0 und `filePath` auf `'Students/Samples/finalTest.wav'` Wenn Sie das Skript nun ausführen wird eine Testnachricht decodiert. Falls Sie sie lesen können haben Sie alles richtig gemacht, ansonsten müssen die vorherigen Funkionen nochmals verbessert werden.

3 Nachrichten in Echtzeit empfangen

- Falls Sie die vorherige Aufgabe gelöst haben können Sie nun in der `BoAPRS.m` Datei im oberen Bereich die Variable `checkFlagSolution` auf 0 setzen. So wird auch effektiv ihre Lösung verwendet.
- Nun sind Sie bereit für die online Version. Rufen Sie mit Firefox wieder die Webseite vom Empfänger Friedrichshafen auf. `http://dk0te.dhbw-ravensburg.de:8901/` Stellen Sie den Empfänger wieder auf `2m Band` und `144.800 MHz`. Überlegen Sie sich auch kurz warum es 2m Band heisst.

- Falls auf dem genannten Empfänger keine Daten ankommen, können Sie auch den Empfänger in Nürnburg probieren. `http://nbgsdr.ddns.net/`
- Starten Sie danach `PulseAudio Volume Control`, ein vorinstalliertes Linux Programm. Damit übermitteln Sie die Audioausgabe von Firefox direkt an Matlab. Testen Sie PulseAudio mit dem Skript `AudioInput.m` Falls Sie Probleme haben, finden Sie hier eine detaillierte Erklärung:
  `https://unix.stackexchange.com/questions/82259/how-to-pipe-audio-output-to-mic-input`
- Wenn alles funktioniert setzt Sie in `BoAPRS.m` `online` auf 1 und starten Sie das Skript. Pakete werden nun automatisch empfangen und decodiert. Es können nicht ganz alle Pakete empfangen werden, aufgrund der Uneindeutigkeit des APRS Protokolls.

Literatur

[1] W. A. Beech, D. E. Nielsen und J. Taylor, *AX.25 Link Access Protocol for Amateur Packet Radio*, Version 2.2, Tucson Amateur Packet Radio Corporation, Juli 1998.

[2] I. Wade, *APRS Protocol Reference*, Approved Version 1.0.1, The APRS Working Group, Aug. 2000.