Hochschule Konstanz Technik, Wirtschaft und Gestaltung (HTWG)
Fakultät Informatik
Rechner- und Kommunikationsnetze
Prof. Dr. Dirk Staehle

Labor zur Vorlesung Kommunikationstechnik

Laborübung
Linear-systematische Blockcodes

Prof. Dr. Dirk Staehle

Die Abgabe erfolgt durch Hochladen der bearbeiteten Word-Datei sowie der Matlab-Skripte in Moodle.

Bearbeitung in Zweier-Teams

Team-Mitglied 1:

Team-Mitglied 2:

1 Einleitung

In der Vorlesung wurde die Fehlererkennung und Fehlerkorrektur mit Hilfe von linear-systematischen Blockcodes vorgestellt. In dieser Laborübung soll die Anwendung von Kanalcodierung in Python umgesetzt werden. Eine erste Simulation einer digitalen Übertragungsstrecke soll implementiert und erste Simulationen durchgeführt werden.

2 Implementieren in Python

Implementieren Sie eine Python-Klasse BlockCode mit den Attributen

- k: Anzahl Nutzbits
- n: Anzahl CodeBits
- p: Anzahl Prüfbits
- max_corr_bits: maximale Anzahl korrigierter Bits (bis zu 2 soll unterstützt werden)
- G: Generatormatrix
- P: Teil-Matrix
- H: Kontrollmatrix
- S: Syndrom-Tabelle

und den Methoden

- codeword=encode(message)
- message, corrected_errors =decode(codeword)
  - Eine Korrektur soll nur bei einem eindeutigen Fehlersyndrom mit maximal zwei Fehlern durchgeführt werden
  - Rückgabewerte: korrigierte Nachricht, Anzahl korrigierter Fehler (0-2),
    - falls keine Korrektur durchgeführt werden kann, wird statt der korrigierten Nachricht None zurückgegeben

Dem Konstruktor wird die Matrix P sowie max_corr_bits, die Obergrenze der korrigierbaren Bitfehler übergeben. Die übrigen Attribute können aus den Argumenten bestimmt werden. Bei der Syndrom-Tabelle ist es nicht notwendig, eine allgemeine Variante zu implementieren. Es genügt, die Syndrom-Tabelle für einen oder zwei korrigierbare Fahler zu implementieren.

Tipp 1: Verwenden Sie bei der Implementierung Arrays (Matrizen, Vektor) aus dem Modul numpy und verwenden Sie Matrix-Multiplikationen (Funtion numpy.matmul).

Tipp 2: Binären Vektor (Array) in Integerwert umwandeln:

```
int(''.join([f'{b}' for b in v]),2)
```

3 Bitfehler / Kanalmodell

Implementieren Sie zwei Modelle, um Bitfehler zu erzeugen:

1. einen binär-symmetrischen Kanal (BSC), der jedes Bit unabhängig mit Wahrscheinlichkeit p umkippt
   Tipp: rand() aus numpy.random
2. einen Kanal mit fester Anzahl Bitfehler pro Codewort, die an zufälligen Stellen auftreten
   Tipp 1: sample(rand(n)) mit sample aus random liefert Bitpositionen
   Tipp 2: Funktion flipbits(bit_vector, position_list) ist hilfreich

Die Kanalmodell sollen als Klassen implementiert werden. Die Übertragung eines Pakets (das Einfügen der Bitfehler) soll durch Überladen der Methode __call__ erfolgen, so dass Sie die Übertragung durch y=Channel(x) durchführen können, wobei Channel ein Objekt eines Kanalmodells ist, x die gesendete und y die empfangene Nachricht. Nachrichten sind jeweils binäre Vektoren (numpy.array)

4 Test

Testen Sie ihre Implementierung mit den code aus der Vorlesung:

1. der Matrix P des (7,4)-Codes aus der Vorlesung mit max_corr_bits=1
2. mit $P = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$ und max_corr_bits=2
3. mit $P = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$ und max_corr_bits=2

Verwenden Sie das Bitmodell mit der festen Fehlerzahl, um die Anzahl korrigierbarer Bits zu überprüfen.

5 Simulationsstudie

5.1 Übertragung einer Nachricht

Implementieren Sie die Simulation der Übertragung einer Nachricht über eine digitale Übertragungsstrecke, die folgende Komponenten beinhaltet:

1. **Quelle**: Erzeugung einer zufälligen binären Nachricht aus k Bits.
2. **Kanalcodierung** (Sender): Codieren Sie die Nachricht mit einem linear-systematischen Code
3. **Übertragungskanal**: Simulieren Sie die Übertragung der Nachricht, in dem Sie mit einem Kanalmodell Bitfehler hinzufügen.
4. **Kanalcodierung** (Empfänger): Dekodieren/korrigieren Sie die Nachricht am Empfänger.
5. **Auswertung:** Bestimmen Sie,
   a. die Anzahl der Bitfehler vor der Korrektur
   b. ob eine Fehlerkorrektur durchgeführt wurde (ob Fehlersyndrom in Syndrom-Tabelle enthalten ist)
   c. die Anzahl korrigierter Bits
   d. die Anzahl der Bitfehler im Codewort nach der Korrektur

Tipp: Implementieren Sie alle Elemente der Übertragungsstrecke so, dass sie mit der __call__ Funktion aufgerufen werden können. Für den BlockCode bedeutet das, dass es eine Klasse Encoder und eine Klasse Decoder gibt, die z.B. als Attribut die gleiche Instanz der Klasse BlockCode haben.

5.2 Simulationsstudie

In der Simulationsstudie sollen N Nachrichten übertragen werden und dann eine statistische Auswertung mit den folgenden Ergebnissen durchgeführt werden:

1. Anzahl/Anteil fehlerfrei übertragener Nachrichten
2. Anzahl/Anteil korrigierter Nachrichten
3. Anzahl/Anteil fehlerbehafteter aber nicht korrigierter Nachrichten (zu viele Bitfehler, Fehlersyndrom nicht in der Syndrom-Tabelle enthalten)
4. Anzahl/Anteil erfolgreich korrigierter Nachrichten (Bitfehler vorhanden und erfolgreich korrigiert)
5. Anzahl/Anteil fehlerhaft korrigierter Nachrichten (Bitfehler nach der Korrektur noch vorhanden)
6. die Verteilung der Anzahl Bitfehler vor der Korrektur (als Counter)
7. die Verteilung der Anzahl korrigierter Bits (als Counter)
8. die Verteilung der Anzahl Bitfehler nach der Korrektur (als Counter)

5.3 Szenario 1

Betrachten Sie den (7,4)-Hamming-Code mit maximal einem korrigierten Bitfehler. Die Nachrichten werden über einen binär-symmetrischen Kanal mit einer Bitfehlerwahrscheinlichkeit von 5% übertragen.

Erstellen Sie die folgenden Grafiken:

1. einen Bar-Plot mit
    a. dem Anteil fehlerfrei übertragener Nachrichten
    b. dem Anteil korrigierter Nachrichten
    c. dem Anteil fehlerbehafteter aber nicht korrigierter Nachrichten
2. einen Bar-Plot mit
    a. dem Anteil fehlerfrei übertragener Nachrichten
    b. dem Anteil erfolgreich korrigierter Nachrichten
    c. dem Anteil fehlerhaft korrigierter Nachrichten
3. einen Bar-plot mit
    a. der Verteilung der Anzahl Bitfehler vor der Korrektur
    b. der Verteilung der Anzahl korrigierter Bits
    c. der Verteilung der Anzahl Bitfehler nach der Korrektur

5.4 Szenario 2

Betrachten Sie den Code mit $P = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$ und max_corr_bits=2. Die Nachrichten werden über einen binär-symmetrischen Kanal mit einer Bitfehlerwahrscheinlichkeit von 15% übertragen. Erstellen Sie die gleichen Grafiken wie in der vorigen Aufgabe.

5.5 Szenario 3

Untersuchen Sie jetzt den Einfluss der Bitfehlerwahrscheinlichkeit. Führen Sie Simulationen mit je 1000 Nachrichten und selbstgewählten Bitfehlerwahrscheinlichkeiten zwischen 0,1% bis 20% durch. Vergleichen Sie die Restfehlerwahrscheinlichkeit für die Code mit den Matrizen (7,4)-Hamming-Code, den (11,4)-Hamming-Code

mit $P = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$. Bei letzterem Code sollen bis zu zwei Bitfehler korrigiert werden.