TU DRESDEN

Fakultät Informatik
Institute of Systems Architecture
Dr.-Ing. Dagmar Schönfeld

Leistungsnachweis

Faltungskodierer und Quantisierung

– Kanalkodierung –

vorgelegt von:

Student:	Peter Süttner
Studiengang:	Dipl. Medieninformatik
Matrikelnummer:	3144238
Geburtsdatum:	11.03.1984
E-Mail:	Peter.Suettner@mailbox.tu-dresden.de

Inhalt

1 Einleitung

1.1 Aufgabenstellung

Zum Erhalt des Leistungsnachweises Kanalkodierung wurde aus dem Themenkomplex **Faltungskodes** die **1.Aufgabe** zur Bearbeitung ausgewählt. Weiterhin wurde als Zusatz die Untersuchung des Einflusses einer 2-Bit Quantisierung betrachtet. Es folgt ein kurze Zusammenfassung der Aufgabenstellung:

- programmtechnische Realisierung VITERBI-Algorithmus

- dekodieren der Empfangsfolge

  b = (1111100110101110110010101110100100110000001110011110110110000000111110101110000111110100000010001000000010000111100010)

- Faltungskodierer:

$$G = (47_8, 53_8, 75_8)$$

- Vergleich Ergebnis mit **freier Distanz** des Faltungskodes

- Experimentieren mit Quantisierungsalphabet **Q = {-2, -1, 1, 2}**

1.2 Aufbau des Dokumentes

Im Rahmen des Leistungsnachweis ist ein Programm zur experimentellen Untersuchung von Faltungskodes unter Berücksichtigung einer 2-Bit Quantisierung entstanden. Der folgende Abschnitt gibt eine kurze Einführung in das Programm und dessen Bedienung. Daraufhin folgt eine zusammengefasste Präsentation der Ergebnisse und abschließend wird auf die softwaretechnische Realisierung eingegangen.

2 Das Programm

Ausgehend von der Umsetzung des VITERBI Algorithmus wurde eine komplette Anwendung zur Analyse von Faltungskodes entwickelt. Die Beschreibung der Hauptfunktionen dieser Anwendung sollen anhand der grafischen Benutzerschnittstelle erläutert werden. Diese ist in Abbildung 1 dargestellt.

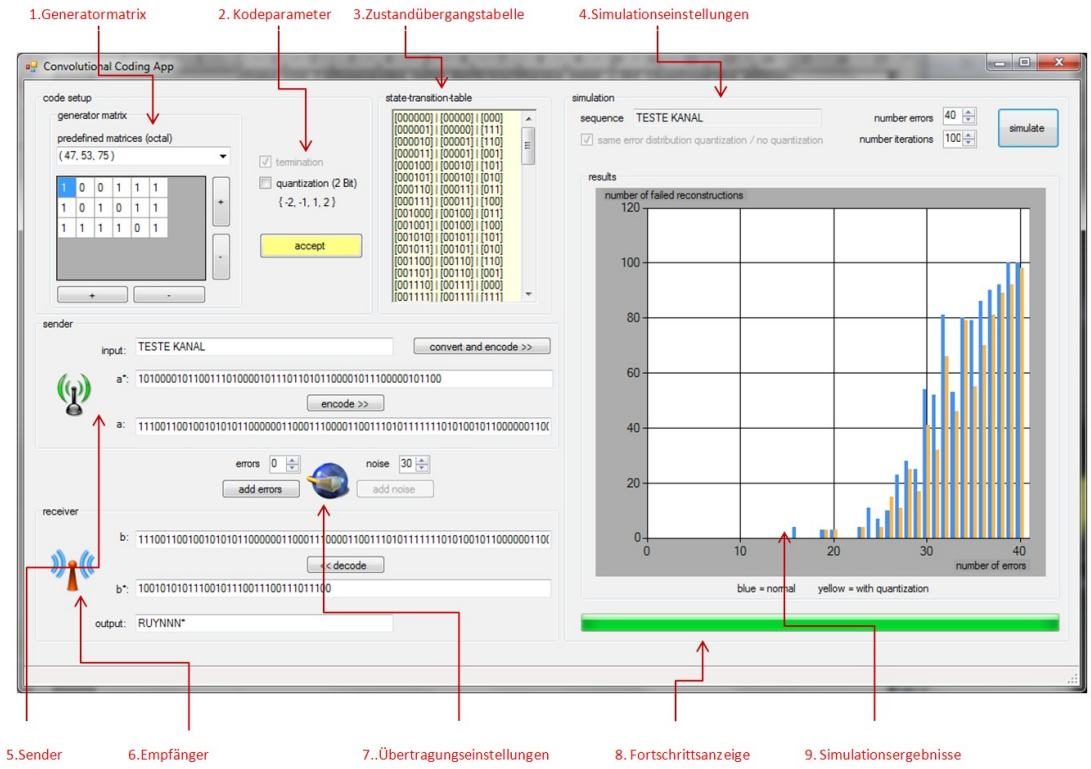

Abbildung 1: Grafische Benutzeroberfläche der Applikation

1. **Generatormatrix** Dieser Teil der Oberfläche erlaubt das Festlegen einer beliebigen Generatormatrix. Einige Matrizen von OFD Faltungskodierern sind bereits voreingestellt. Bislang findet keine Überprüfung auf katastrophale Eigenschaften statt. Daher wird zur Vorsicht bei der Eingabe neuer Matrizen ausdrücklich geraten.
2. **Kodeparameter** Hier lassen sich sich diverse Kodeparameter festlegen, die in Zukunft noch ergänzt werden könnten. Bislang ist nur Quantisierung realisiert. Terminierung wurde vorläufig nicht implementiert, da das weglassen dieser zu einer Verschlechterung der Kodeeigenschaften führt (hintere Stellen weniger gut vor Fehlern geschützt).

Achtung: eine neue Generatormatrix wird erst nach drücken des 'accept' Buttons übernommen und Kodierer sowie Dekodierer generiert.

3. **Zustandsübergangstabelle** Zeigt die zum aktuellen Faltungskodierer gehörige Zustandsübergangstabelle.

4. **Simulationseinstellungen** Hier können diverse Einstellungen die Simulation betreffend vorgenommen werden. Als Quellwort dient das jeweils zuletzt kodierte Wort, welches auch in der Textbox hinter dem Label 'sequence' noch einmal auftaucht. Einige Einstellungen werden im Folgenden kurz erläutert

   number errors: Anzahl der Fehler, die zufällig in die Quellkodefolge gestreut werden.

   number iterations: Anzahl der Iterationen für den Simulationsvorgang; entspricht der Anzahl, wie oft eine Sequenz kodiert und dekodiert wird pro Fehler. Pro Iteration wird die Kanalkodefolge mit zufälligen neuen Fehlern, entsprechend der aktuellen Fehlerzahl, überlagert (z.B bei 40 Fehlern und 100 Iterationen wird 40 * 100 mal kodiert, überlagert, dekodiert).

   Daher sollte man die Größe dieser Parameter auch hinsichtlich der aktuell eingestellten Generatormatrix mit Bedacht wählen.

   Die Simulation kann anschließend über den Button 'simulation' gestartet werden. Für eine gegebenenfalls lange Bearbeitung ist weniger die Implementierung des VITERBI Algorithmus verantwortlich, als vielmehr die relativ komplizierte Überlagerung der Folgen mit zufälligen Fehlern und der Wechsel zwischen der Bitfolge mit Quantisierung und ohne. Um einen direkten Vergleich zwischen Quantisierung und normaler Dekodierung zu ermöglichen war dieses Vorgehen leider nötig.

5. **Sender** Hier kann ein Quellwort in alphanumerischen Zeichen oder direkt eine Quellkodefolge (a*) eingegeben, und über die jeweiligen Buttons kodiert werden (a).

6. **Empfänger** Hier wird die zuvor kodierte und gegebenenfalls mit Fehlern und /oder Rauschen überlagerte Folge empfangen (b).

7. **Übertragungseinstellungen** Hier befindet sich die Repräsentation eines möglicherweise gestörten Übertragungskanal. Auf der linken Seite des Icons kann eine Anzahl Fehler festgelegt werden, mit der b anschließend (zufällig) überlagert wird. Auf der rechten Seite befindet sich ein Steuerelement um gegebenenfalls mehr Rauschen in die Folge einzufügen. Dieses ist selbstverständlich nur aktiv, wenn zuvor eine 2-Bit Quantisierung gewählt wurde. Dies ist im Zusammenspiel mit Fehlern eine Art Simulation des Demodulators. Das Prinzip ist das Folgendes:
   Ausgangspunkt: 2-Bit Quantisierung mit Q = {-2, -1, 1, 2}
   Zunächst wird jede 0 → -2 und jede 1 → 2 abgebildet. Die Addition von Rauschen (sowohl zufällig verteilt als auch zufällig festgelegte Werte) sorgt dafür, dass jede -2 entweder eine -2 bleibt oder eine -1 wird (-2 → -1 | -2). Durch Rauschen werden also noch keine Fehler in die Folge eingebracht. Dies geschieht lediglich mit der linken Seite des Steuerelements. Die Fehleranzahl wird wiederum zufällig auf die Empfangsfolge verteilt mit der Wirkung, dass an betreffender Stelle das Vorzeichen gekippt und ein Wert zwischen |1| und |2| zufällig gewählt wird. Das Prinzip ist ohne Quantisierung dasselbe, nur dass statt des Vorzeichens eine 0 in eine 1 und umgekehrt gewandelt wird
   Abschließend kann über den entsprechenden Button die (nun gestörte) Empfangsfolge dekodiert werden. In diesem Zuge wird, falls möglich, die dekodierte Folge auch in die alphanumerische Zeichenfolge umgewandelt.

8. **Fortschrittsanzeige** Da, wie bereits zuvor erwähnt, die Simulation eine gewisse Zeit in Anspruch nehmen kann, wird der Nutzer hier nach dem Start über den Fortschritt informiert.

9. **Simulationsergebnisse** Nach Abschluss der Simulation werden in dieser Ansicht die Ergebnisse präsentiert. Dabei stehen gelbe Balken für Dekodierung mit Quantisierung, blaue Balken für Dekodierung mit harten Werten. Die X-Achse zeigt die Anzahl der Fehler, die zufällig eingestreut wurden, während die Y-Achse die Anzahl der fehlgeschlagenen Korrekturen repräsentiert.

3 Ergebnisse

Wie zu erwarten lieferte die Simulation mit Quantisierung (soft input) bessere Dekodierergebnisse. Die Umsetzung fand mit Hilfe der Minimum Distance (MD) Metrikberechnung statt. In Abbildung 2 und Abbildung 3 sind beispielhaft die Simulationsergebnisse zweier Testläufe unter gleichen Bedingungen (gleiche Sequenz 'KANAL TEST', gleiche Parameter – 40 Fehler bei 100 Iterationen, gleiche Generatormatrix laut Aufgabenstellung) gezeigt.

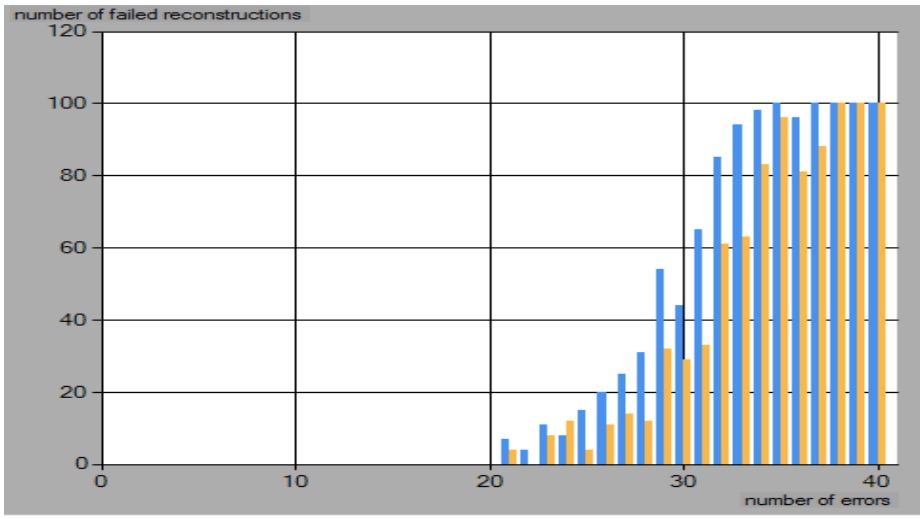

Abbildung 2: Ergebnis 1

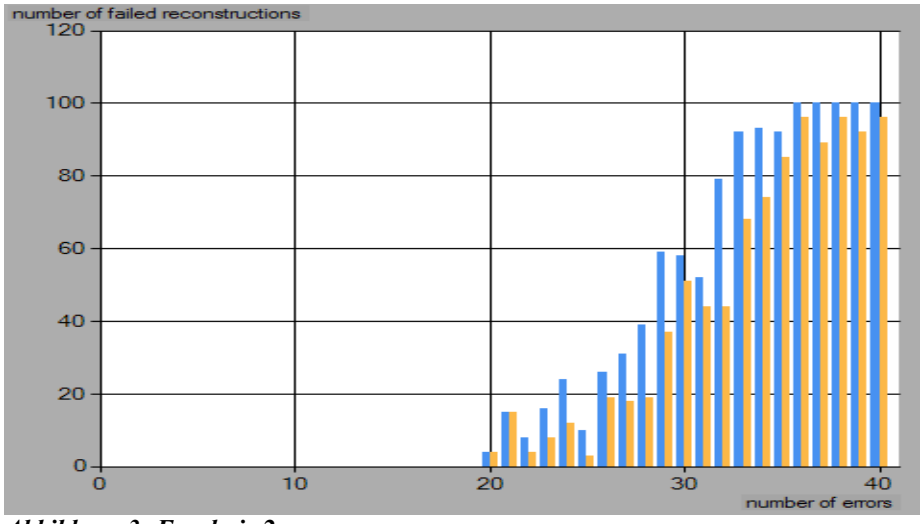

Abbildung 3: Ergebnis 2

Das Quellkodewort der Aufgabenstellung konnte leider nicht rekonstruiert werden, es sei denn es lautete [RUYNNN*].

4 Softwaretechnische Umsetzung

Für die Implementation wurde die Programmiersprache C# in Kombination mit Visual Studio 2010 verwendet. Die grafische Benutzeroberfläche wurde mit WindowsForms realisiert. Daher sollte die Anwendung auf jedem MS Windows Betriebssystem mit aktuellem .NET Framework ausführbar sein.

Da aus einer relativ simplem Implementierung des VITERBI Algorithmus eine ganze Anwendung (um nicht zu sagen ein kleines Framework) entstanden ist, soll hier grob auf die Architektur eingegangen werden. Abbildung 4 zeigt die Klassenstruktur der Applikation als UML Klassendiagramm.

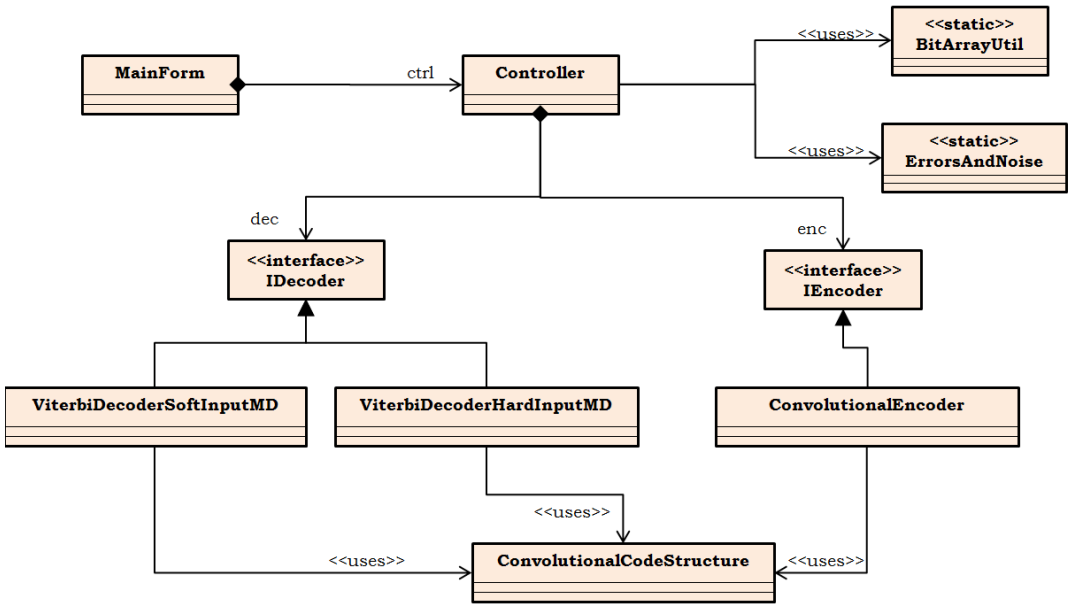

Abbildung 4: Klassendiagramm der Anwendung

Zentrales Element ist die Klasse *Controller*, der das Zusammenspiel zwischen der grafischen Benutzerschnittstelle (*MainForm*) und den Kodierern (*IEncoder*) sowie Dekodierern (*IDecoder*) steuert. Unter den Decodern sind momentan VITERBI für hard und soft (nach MD) implementiert. Weitere könnten leicht in die bestehende Struktur eingehängt werden. Die Kodierer und Dekodierer beziehen die Datenstrukturen, mit denen sie arbeiten, aus der Klasse *ConvolutionalCodeStructures*. Die beiden Hilfsklassen *BitArrayUtil* und *ErrorAndNoise* sind beide statisch und stellen hilfreiche Methoden zum Umgang mit BitArrays (um eine effiziente Umsetzung bei Hard-Decision zu gewährleisten) und zur Überlagerung von Signalfolgen mit Fehlern und Störungen bereit.

5 Fazit / Ausblick

Im Rahmen dieses Leistungsnachweises wurde eine komplette Anwendung für Simulationen des VITERBI Algorithmus implementiert. Dieser Algorithmus wurde sowohl in der in der harten als auch weichen MD Variante umgesetzt. Mit dem Simulationtool kann ein effektiver Vergleich der beiden Varianten, nicht zuletzt durch die grafische Umsetzung, stattfinden. Als Ergebnis bleibt festzuhalten, dass mit Quantisierung, im Normalfall, deutlich Dekodierungsgewinne erzielt werden können. Das Programm darf und kann von Interessierten jederzeit erweitert und verändert werden. Auf die Einstellung des Quellcodes in dieses Dokument wurde aufgrund der Größe verzichtet.