Untersuchung potenzieller Wellenformen zur Broadcast-Datenübertragung mittels Pulsradar

Bachelor-Thesis

zur Erlangung des akademischen Grades B.Sc.

Michael Allinger

[redacted]

Hochschule für Angewandte Wissenschaften Hamburg

Fakultät Design, Medien und Information

Department Medientechnik

Erstprüfer: Prof. Dr.-Ing. Jan Mietzner

Zweitprüfer: Prof. Dr. Robert Mores

Hamburg, 02.07.2019

I. Abstract

“Radar Communication”, short „RadComm“, is a concept of a joint radar and wireless system. Todays research is focusing on short-range radars used in the automotive industry. The technology used there differs from the technology used in sea- and air-based radars, where typically pulse-doppler-radars are installed. Therefor this thesis is investigating the suitability of RadComm as a concept for pulse-doppler-radars. For radar technology is no special field in “Medientechnik” it had to be researched in advance. Based on this research this thesis contains a short overview over radar technology and derived requirements for data transmission. Additionally, it contains a chapter to “fountain codes” found during research, their complexity in practical use and a simulation to check the performance of fountain codes. The conclusion includes a simulation of a sampled bit-sequence to check for radar compatibility.

Keywords: “RadComm, radar technology”, Fountain Codes, pulse-doppler, MATLAB”

I. Zusammenfassung

Das Themengebiet der „Radar Communication“ („RadComm“) befasst sich mit dem Konzept eines vereinheitlichten Systems von Radar und Kommunikation. Dabei liegt der Schwerpunkt der aktuellen Forschungen im Bereich der Automobilwirtschaft. Die dort verwendete Radartechnik unterscheidet sich jedoch stark von der im aeronautischen oder maritimen Bereich. Hier kommen meist Puls-Doppler-Radare zum Einsatz. Daher befasst sich diese Arbeit mit der Untersuchung, ob „RadComm“ auch in diesen Bereichen Anwendung finden könnte. Da die Radartechnik kein Fachbereich im Bereich Medientechnik darstellt, wurde dieser zuerst eigenständig erarbeitet. Darauf basierend wird in dieser Arbeit ein kurzer Überblick über die Radartechnik gegeben und Anforderungen an die Nachrichtentechnik formuliert, die diese für eine Datenübertragung erfüllen muss. Zudem werden die in der Recherche entdeckten Fountain Codes erläutert und auf Basis einer Simulation auf ihre Leistungsfähigkeit untersucht, sowie die Schwierigkeiten einer praktischen Umsetzung erläutert. Am Ende wird beispielhaft eine so kodierte Sequenz auf ihre Funktionalität für die Radartechnik überprüft.

Schlagwörter: „RadComm, Radartechnik, Fountain Codes, Puls-Radar, MATLAB“

II. Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelor-Thesis selbständig und nur mit den angegebenen Hilfsmitteln verfasst habe. Alle Passagen, die ich wörtlich aus der Literatur oder aus anderen Quellen wie z. B. Internetseiten übernommen habe, habe ich deutlich als Zitat mit Angabe der Quelle kenntlich gemacht.

(Datum) **(Unterschrift)**

Inhaltsverzeichnis

0. Die vorgegebene Aufgabenstellung

Radarsensoren kommen heutzutage in den verschiedensten Anwendungsgebieten zum Einsatz, beispielsweise in den Bereichen Zielobjektdetektion, Messtechnik, Fernerkundung sowie Bildgebung. Allgemein senden Radare aktiv hochfrequente Signale aus und erfassen Informationen über ihre Umgebung, indem sie entsprechende Echosignale auswerten. Einige praktische Anwendungen benötigen darüber hinaus gleichzeitig auch eine Möglichkeit zur drahtlosen Kommunikation. In aktuellen technischen Lösungen werden typischerweise zwei getrennte Systeme für die Radar- und die Kommunikationsfunktion realisiert, was zusätzliche Installations- und Wartungskosten nach sich zieht. Außerdem kann es bei einer solchen Lösung zu wechselseitigen Interferenzen kommen, wenn beide Systeme in ähnlichen Frequenzbereichen arbeiten. Daher ist es von Interesse, die Radar- und Kommunikationsfunktion zu einem gemeinsamen System zusammenzuführen (Stichwort „Joint radar and wireless communications, RadComm“).

Eine vorstellbare Anwendung eines solchen gemeinsamen Systems wäre ein Überwachungsradar, wie zum Beispiel ein Küstenradar zur Überwachung des Schiffsverkehrs oder ein Flugüberwachungsradar, welches gleichzeitig zum Radarbetrieb relevante Broadcast-Informationen aussendet. Inhalte könnten Wetterinformationen sein, aber auch Notfallmeldungen jeglicher Art, das heißt Dienste mit eher geringen Anforderungen hinsichtlich der Datenrate. Dies könnte auch für viele weitere praktische Anwendungen einen gewissen Mehrwert darstellen.

Im Rahmen dieser Arbeit sollen potenzielle Wellenformen für einen gemeinsamen Radar- und Kommunikationsbetrieb auf ihre Eignung untersucht werden. Der Schwerpunkt soll dabei auf einem pulsbasierten, rotierenden Radar zur 360°-Überwachung liegen.

Teil dieser Arbeit ist eine ausgiebige Literaturrecherche zum Thema „Radartechnik“ und „Radar Communication“. Basierend auf der Recherche sollen dann mögliche Wellenformen für die obige Anwendung identifiziert und hinsichtlich ihrer Eignung für den Radar- und gleichzeitigen Kommunikationsbetrieb untersucht werden. Die Leistungsfähigkeit der ausgewählten Wellenformen für den Radarbetrieb kann beispielsweise basierend auf der resultierenden Ambiguity-Funktion in den Dimensionen Entfernung und ggf. Doppler bewertet werden.

1. Anmerkung zu dieser Arbeit

Radartechnik ist kein Fachbereich des Studienganges Medientechnik. Daher befasst sich ein Abschnitt dieser Arbeit damit, dieses Themengebiet zu erarbeiten und entsprechende Recherchen anzustellen. Zum allgemeinen Verständnis gebe ich zu Beginn eine grundlegende technische Einführung in die Radartechnik und konkretisiere das Thema anhand des Puls-Doppler-Radars. Anschließend stelle ich die für diese Arbeit relevanten Grundlagen der Nachrichtentechnik vor.

Da sich diese Bachelorarbeit auf aktuelle Forschungsarbeiten im Bereich der „Codierungstheorie“ und auf Veröffentlichungen rund um das Thema „joint radar and wireless communications“ bezieht, ist es möglich, dass es bereits zum Zeitpunkt der Fertigstellung meiner Arbeit aktuellere Fachliteratur zu diesen Themen gibt, als die hier von mir angeführte und verwendete. Es wurden Publikationen bis zum 11.06.2019 berücksichtigt.

2. Grundlagen

Der Begriff „Radar“ stammt ursprünglich aus dem Englischen und wird hauptsächlich als Abkürzung verwendet für „*RAdio Detection And Ranging“*, aber auch „*Radio Aircraft Detection And Ranging“* bzw. *„RAdio Direction And Ranging“*.[4,25] Je nach Bezug kann man den Begriff somit als „funkbasierte (Flugzeug) Detektion und Entfernungsmessung“ oder als „funkbasierte Richtungs- und Entfernungsmessung“ übersetzen. Damit werden die grundlegenden Funktionen von Radarsystemen gut zusammengefasst, wobei im Laufe der Zeit weitere Funktionen hinzugekommen sind.

Physikalisch nutzen die meisten Radarsysteme drei grundlegende Eigenschaften elektromagnetischer Wellen aus [19]:

a) die quasi konstante Ausbreitungsgeschwindigkeit in der Luft mit annähernder Lichtgeschwindigkeit von 299 792 458 m/s
b) die geradlinige Ausbreitung in den höheren radarspezifischen Frequenzbereichen
c) die Reflexionen der Wellen beim Auftreffen auf elektrisch leitende Körper

Allein durch die Nutzung dieser drei Aspekte lassen sich Hindernisse und deren Entfernung detektieren. Kombiniert mit moderner Antennentechnik und digitaler Signalverarbeitung lassen sich außerdem die Größe, der Höhenwinkel bzw. die Höhe eines Objektes, ggf. die Bewegungsrichtung und sogar die Struktur eines Objektes erkennen. Ausnahmen sind Radar-Systeme für spezielle Anwendungsfälle wie das sogenannte „Over The Horizon Radar“ [4], das sehr niedrige Frequenzen nutzt, die nicht die oben genannten Eigenschaften besitzen, um damit weit entfernte Objekte wahrzunehmen. Zu Gunsten der enormen Reichweite verzichtet dieses Radar auf den Versuch einer exakten Ortung.

Anwendung finden Radarsysteme in heutiger Zeit in vielen Bereichen. Sie reichen von der klassischen Luft-, Raum- und Schifffahrt, Militäranwendungen wie radarbasierte Bildaufklärung, bis hin zum alltäglichen Leben in Fahrassistenzsystemen von Autos, dem Wetterradar oder automatisierten Produktionsprozessen der Wirtschaft. Je nach Kontext ordnet man die Systeme nach Anwendungsgebiet oder nach Plattform.

2.1 Technische Einführung

Grundlegend gibt es zwei Typen von Radarsystemen, die aktiven und die passiven[1,4,19]. Während ein aktives Radar elektromagnetische Wellen erzeugt und ausstrahlt, detektiert das passive Radar nur Reflektionen von anderen Sendequellen, wie DVB-T (*Digital Video Broadcast-Terrestrial*, eine funkbasierte Form des digitalen Fernsehens), DAB (Digital Audio Broadcast, Digitales Funk-Radio) und ähnlichen Systemen (siehe schematisch in Abbildung 1). Dadurch sparen sie zum einen die zum Senden benötigte Energie, zum anderen ist es mit Passiv-Radaren möglich, durchgehend zu detektieren, was bei einigen aktiven Langstreckenradaren, wie dem Puls-Doppler-Radar, nicht der Fall ist.
Damit eine genaue Ortung bei passiven Radaren möglich ist, müssen dem System sowohl Standort als auch Frequenzen und Modulationsarten der anderen genutzten Systeme bekannt sein. Eingesetzt werden passive Radarsysteme sowohl im militärischen als auch zivilen Bereich, meist als Überwachungssysteme zum Beispiel im Flugverkehr.

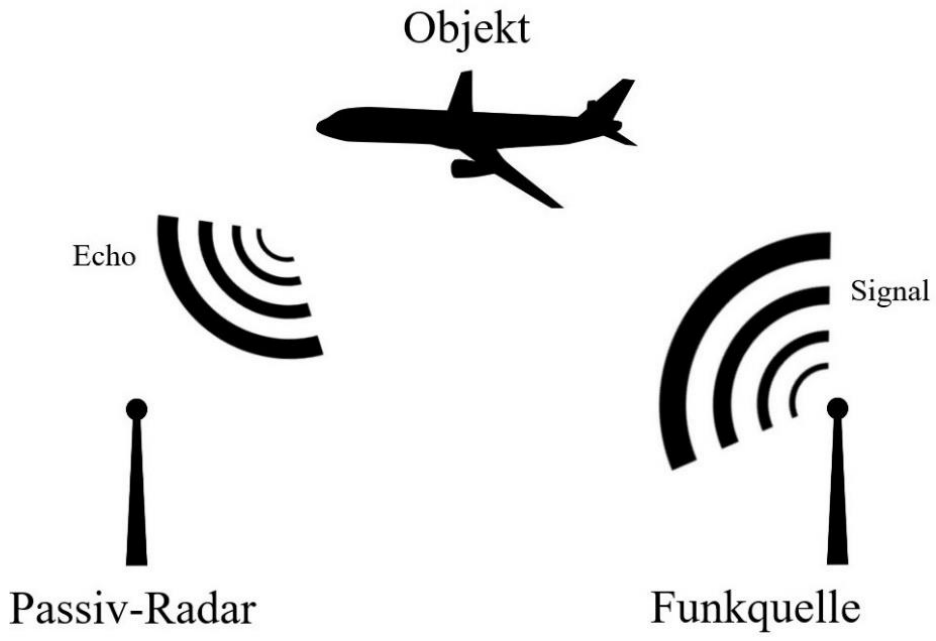


Abbildung 1 Skizzierte Funktionsweise eines Passiv-Radars

Aktiv-Radare kommen überall dort zum Einsatz, wo entweder keine anderen nutzbaren Strahlungsquellen zur Verfügung stehen (offenes Meer, unbewohnte Gebiete) oder der Anwendungsfall eine besonders feine Ortungsauflösung benötigt, um entweder sehr präzise zu messen oder sehr kleine Objekte zu erfassen.[] Dies gelingt vor allem durch eine höhere

Betriebs- bzw. Trägerfrequenz des Radarsystems. Je höher die Frequenz, desto kleiner die Wellenlänge (vergleiche Formel 1.1.1) und desto kleiner können die zu detektierenden Objekte sein.

$$\lambda = \frac{c_0}{f} \qquad \text{mit } c_0 \text{ als } Lichtgeschwindigkeit$$

Formel 2.1.1, Verhältnis Wellenlänge zu Frequenz bei elektromagnetischen Wellen

So nutzen radargestützte Körperscanner Frequenzen >100 GHz [18], um Objekte von unter 5mm Größe zu erfassen (nach Formel 2.1.1), während die meisten Flug- und Schiffssysteme von circa 100 MHz im militärischen Bereich bis zu 18 GHz im zivilen Schiffsverkehr [19] arbeiten.

Grundsätzlich ist die Funktionsweise eines Aktiv-Radars wie folgt darstellbar:

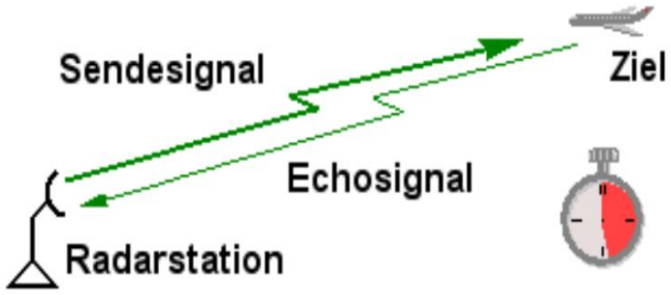


Abbildung 2 Vereinfachte Darstellung der Radar-Funktionsweise

Dabei lässt sich der Aufbau vereinfacht in fünf Bauteilgruppen aufteilen. [4] Folgt man der Entstehung eines Sendesignals, so beginnt es beim Sender. Dieser erzeugt anhand von Signal- und Taktgeneratoren das ursprüngliche Sendesignal, das dann an die Sendeantenne mit vorgeschaltetem Verstärker weitergegeben wird. Hier wird dem Signal Energie zugeführt und über die Antenne ausgesendet.

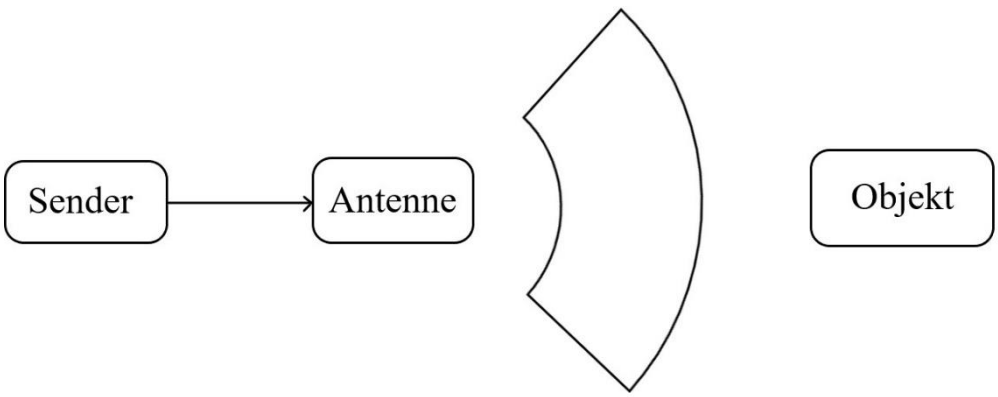


Abbildung 3 Skizzierte Darstellung Sender (1), Antenne (2) und das getroffene Objekt

Trifft das Signal ein leitendes Objekt, wird ein Echo zurückgeworfen, welches dann an einer Empfangsantenne empfangen wird. Je nach Radartyp kann es sich um die ursprünglich zum Senden genutzte Antenne oder eine andere handeln. Diese ist mit dem Empfänger verbunden,

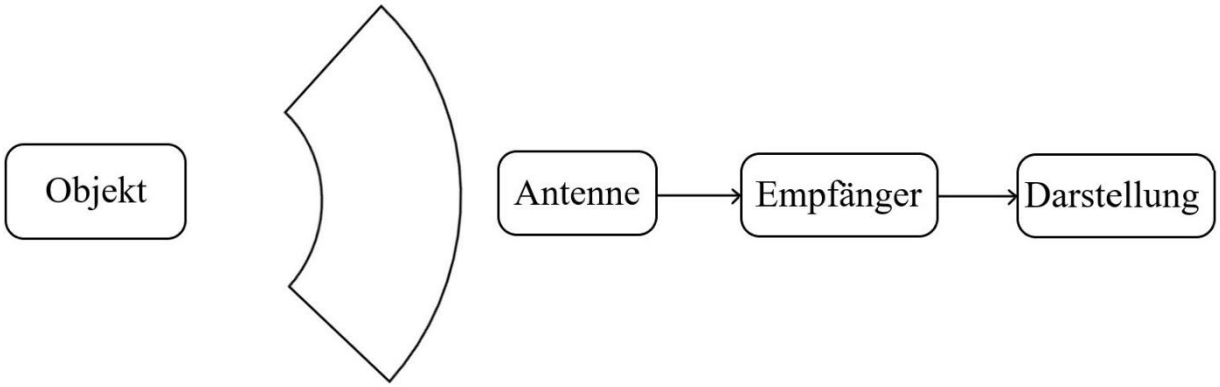


Abbildung 4 Skizzierte Darstellung Objekt, Antenne (3), Empfänger (4) und Darstellung (5)

der das Signal zunächst analog filtert und in seiner Energie begrenzt, um die folgenden, empfindlichen Komponenten vor Beschädigung zu schützen. Danach wird das Empfangssignal digitalisiert und digital gefiltert.
Im Anschluss kann das Signal dann entweder gespeichert oder mit Hilfe der digitalen Signalverarbeitung dargestellt werden. Ein Beispiel hierfür wäre der Überwachungsmonitor eines Flug- oder Schiffsradars.

Es gibt verschiedene Arten von aktiven Radarsystemen. Sie unterscheiden sich im verwendeten Sendesignal und in der verwendeten Sendetechnik, wobei teilweise Kombinationen eingesetzt werden, um die Funktionalität zu optimieren.

Der Unterschied in der Sendetechnik liegt erstens in der Sendeleistung und zweitens in der Anzahl der verwendeten Antennen und/oder des verwendeten Antennentyps. [4,19,25]
Rundstrahler geben ihre Leistung nahezu vollumfänglich um die Antenne herum ab, müssen für gesteigerte Reichweiten jedoch wesentlich mehr Energie aufwenden als andere Antennentypen.
Richtantennen bündeln die Leistung und geben sie gezielt ab. Dadurch ist eine gesteigerte Reichweite in Ausrichtung der Antenne möglich, gleichzeitig wird die Abstrahlung in andere Richtungen stark gedämpft.

Radare mit geringer Sendeleistung, wie sie zum Beispiel für die Nahbereichsdetektion in Automobilen eingesetzt werden, sind in der Lage, gleichzeitig zu senden und zu empfangen. Sie benötigen dafür jedoch separate Antennen.
Radare mit hoher Sendeleistung können nicht gleichzeitig senden und empfangen, da eine hohe Sendeleistung bei einem nahegelegenen Empfänger zu Interferenzen und damit zu eingeschränkter Funktionalität bis hin zu einem kompletten Funktionsausfall führen kann. Solche Systeme verwenden meist eine Antenne, die im Wechsel zum Senden und Empfangen genutzt wird.
Phased-Array-Antennensysteme werden dafür genutzt, dasselbe Sendesignal von mehreren nebeneinanderliegenden Antennen auszustrahlen, jedoch phasen- bzw. zeitversetzt. Ähnlich wie in der Akustik bei Schallwellen zu beobachten, lassen sich auch elektromagnetische Wellen auf diese Weise formen und bündeln. Dies nennt man Beamforming. Es entsteht eine Richtwirkung, ohne eine speziell dafür ausgelegte Antenne einsetzen zu müssen.
Im Nahbereich wird bevorzugt das sogenannte MIMO-Radar eingesetzt. MIMO steht für „Multiple-Input-Multiple-Output“. Dieses nutzt mehrere Antennen, sowohl zum Senden als auch Empfangen, wobei jede Sendeantenne ein anderes Sendesignal besitzt. Die Empfänger nehmen alle Signalechos auf und verarbeiten diese. Da an jeder Empfangsantenne alle Echos empfangen werden, können anhand der Signalabweichungen zwischen den Empfangsantennen feine Unterschiede detektiert und für die Ortung genutzt werden. Diese Form des Radars ist daher besonders präzise.

Unterscheidet man Radare nach ihrem Sendesignal, so gibt es zwei Haupttypen:
das Puls-Radar und das *„Continuous Wave Radar“*, kurz CW-Radar. [19,25]

Das CW-Radar, im Deutschen auch „Dauerstrichradar“ genannt, sendet durchgehend eine Frequenz aus und empfängt gleichzeitig ebenfalls durchgehend. In seiner ursprünglichen Funktionsweise war es nicht zur Entfernungsbestimmung geeignet, sondern diente lediglich zur Detektion von Bewegung und gegebenenfalls zur Bestimmung der Geschwindigkeit dieser Bewegung. Bewegungsmelder sind ein Anwendungsbeispiel für dieses Radar. In seiner Weiterentwicklung, zum Beispiel zum FMCW-Radar durch Hinzufügen einer Frequenzmodulation, wurde die Entfernungsmessung realisiert und findet heute Einsatz im Bereich der Kraftfahrzeug-Radare.

Das Puls-Radar nutzt, wie der Name aussagt, einen kurzen Impuls als Sendesignal. Meist in Langstreckensystemen mit hoher Sendeleistung eingesetzt, wird bei diesem Verfahren ein Puls abgegeben, die Antenne wird vom Sender auf den Empfänger umgeschaltet und das System „hört“ auf die ankommenden Echos.

2.2 Aufbau eines Puls-Doppler-Radars

Das Puls-Doppler-Radar lässt sich grundsätzlich in seinem Aufbau wie in Kapitel 2.1 beschrieben darstellen. Im Folgenden werde ich, angelehnt an die Übersicht aus dem „Radar Handbook“ [4, S. 4 -- 11], etwas genauer auf die einzelnen Komponenten eines typischen Puls-Doppler-Radars nach heutigem Technikstand eingehen und dabei auch dessen Funktion beschreiben. Dazu dient das folgende Schaltbild:

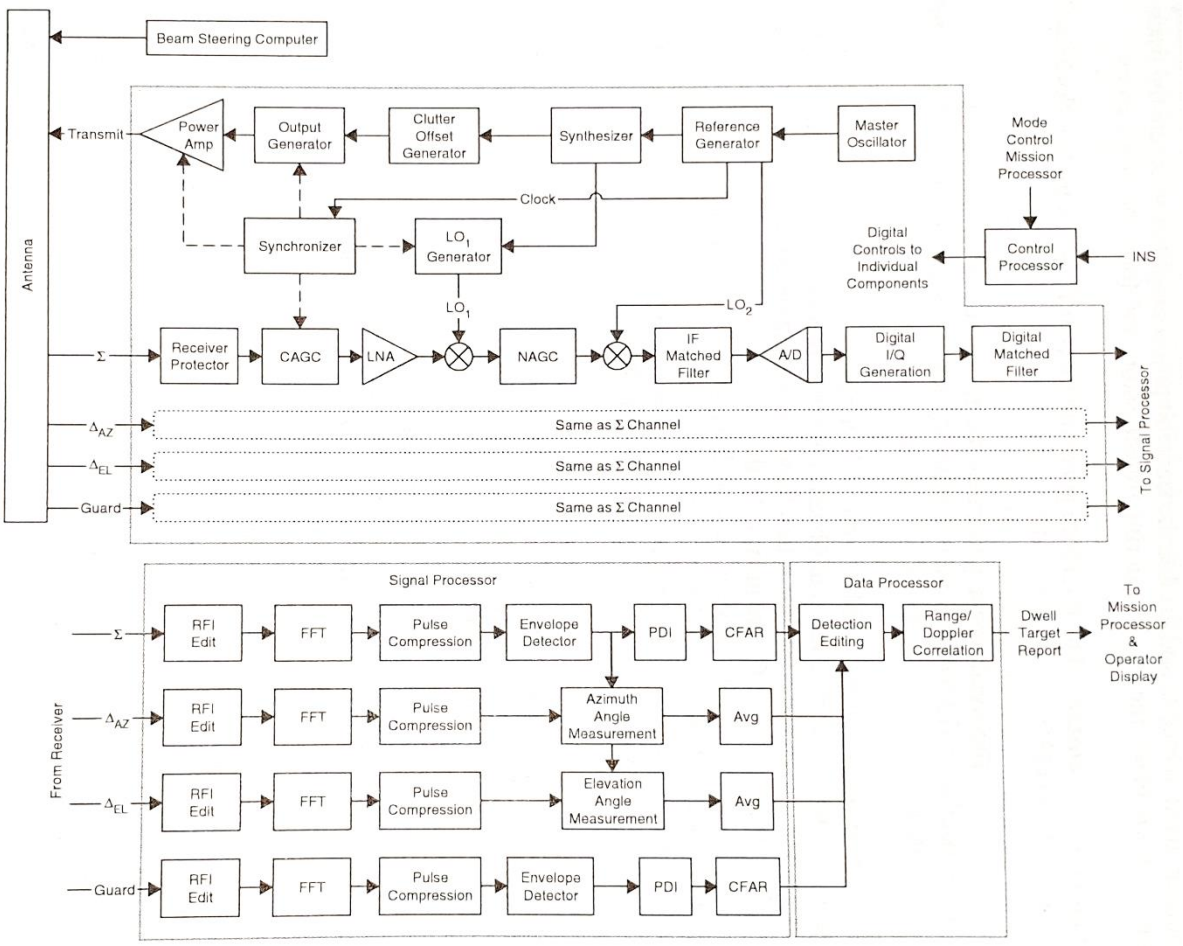


Abbildung 5 Gesamtschaltbild eines typischen Puls-Doppler-Radars

2.2.1 Übergreifende Komponenten

Um eine präzise Ortung zu ermöglichen, bedarf es äußerst genauer Messungen und Berechnungen. Essenziell für eine solche Zusammenarbeit einzelner Komponenten ist ein gemeinsamer Arbeitstakt (auch Referenztakt, im englischen *„reference clock“*). Dafür wird von einem Präzisions-Oszillator („*Master Oscillator“)* eine Sinuskurve erzeugt, die von dem eigentlichen Referenzgenerator („*Reference Generator*“) zur Erzeugung des Arbeitstaktes verwendet wird. Dieser wird sowohl im Empfangsblock als auch im Sendeblock genutzt. Zusätzlich speist der Oszillator den sogenannten „Synchronisator“ („*Synchronizer*“), der die Funktion hat, sämtliche Komponenten für den eigentlichen Sende- und Empfangsvorgang synchron mit einem stabilen Arbeitstakt zu versorgen.

2.2.2 Sender

Die Sendeseite eines Puls-Doppler-Radars besteht aus vier Komponenten. Dem Synthetisierer („*Synthesizer*“), dem sogenannten „Clutter-Offset Generator“, dem Ausgangs-Generator („*Output Generator*“) und dem Ausgangsverstärker („*Power Amp*“).

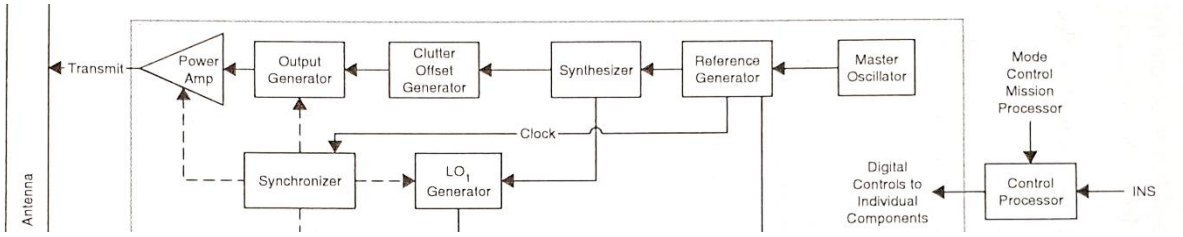


Abbildung 6 Sender; Ausschnitt aus dem vorherigen Gesamtschaltbild (Abb5)

Der Synthetisierer erzeugt die Trägerfrequenz des Systems. Dieser Prozess muss sehr präzise ausgeführt werden, da dies die Frequenz ist, auf der die Empfangsseite nach Echos sucht. Der Clutter-Offset Generator verschiebt die Trägerfrequenz leicht, mit dem Ziel, spätere Störungen im Empfangssignal besser filtern zu können. Dieses nachrichtentechnische Konzept wird in Kapitel 2.4.3 Rausch- und Störunterdrückung noch näher erläutert. Der Ausgangssignal-Generator erzeugt mithilfe der Trägerfrequenz und, je nach System, einem Funktionssignal den eigentlichen Sendeimpuls. Dieser wird dann von dem Ausgangsverstärker auf die gewünschte Sendeleistung verstärkt.

2.2.3 Duplexer

Der verstärkte Impuls aus dem Ausgangsverstärker wird nicht direkt an die Antenne weitergegeben, sondern erst durch den sogenannten Duplexer geleitet. Da das Puls-Doppler-Radar mit hohen Sendeleistungen für Langstreckenüberwachung arbeitet, kann es, wie in der Einführung (Kapitel 2.1) bereits erwähnt, nicht gleichzeitig senden und empfangen. Eine separate Empfangsantenne würde durch die Sendeantenne gestört werden. So wird nur eine Antenne für das Senden und Empfangen verwendet. Diese eine Antenne muss entsprechend umgeschaltet werden, damit beim Senden das Signal nur zur Antenne hingeleitet wird und nicht auch zum Empfänger. Diesen Umschaltvorgang übernimmt der Duplexer. Die Zeit, in der der Duplexer eine Schaltposition hält, in der die Antenne sendet oder empfängt oder das Radar sonstige Funktionen (z.B. Wartungsfunktionen) ausführt, ist abhängig von den gewünschten Ortungseigenschaften. Diese Zeitspannen beeinflussen unter anderem die minimale und maximale Ortungsentfernung (siehe Kapitel 2.4.2 Grundlagen der Detektion).

2.2.4 Antenne

Die Antenne ist das offensichtlichste Bauteil eines Radargerätes. Es ist auch das Bauteil, das den größten Einfluss auf die weiteren Eigenschaften des Systems hat. Sie erzeugt aus der generierten Signalenergie die eigentliche elektromagnetische Welle oder entnimmt einer solchen bei Empfang Energie, die sie an den Empfänger weitergibt. Je nach Anwendungsgebiet, gewünschtem Abstrahlverhalten und der gewünschten Empfangs- beziehungsweise Sendefrequenz ist ihre Bauform verschieden. Daher gehe ich hier, im Gegensatz zu den anderen Bauteilen des Puls-Dopplers, etwas genauer auf dieses Bauteil ein. Zum Vergleich der verschiedenen Antennenbauarten wird meist eine rein theoretische Antenne als Bezugspunkt genutzt: der Isotropstrahler, auch Kugelstrahler genannt. Es handelt sich um einen fiktiven Punkt im Raum, der als Antenne angesehen wird. Sein Abstrahlverhalten entspricht dem einer Kugel um diesen Punkt, die sich vergrößert und die Sendeenergie in alle Richtungen gleichmäßig abstrahlt. In einigen Gebieten der Messtechnik werden Antennen auch mit realen Antennen verglichen, die dem Kugelstrahler in ihren Eigenschaften am nächsten kommen.

Ziel des Vergleichs mit dem idealen Kugelstrahler ist es, den relativen Energiegewinn durch die Richtwirkung einer Antenne sowie das Streuverhalten der Antenne in andere Richtungen zu ermitteln. Den Energiegewinn in Richtung des Ziels nennt man „Antennengewinn“ (Kapitel 2.3.1). Er wird in Dezibel (isotrop) [dBi] angegeben.

Eine allgemein bekannte Antennenform ist die Parabolantenne.[19] Sie wird in zahlreichen Bereichen eingesetzt, die bekannteste Anwendung ist die erdgebundene Satellitenkommunikation. Die Parabolantenne besitzt einen Reflektor, der speziell gebogen ist, um ausgesendete und empfangene elektromagnetische Wellen phasenrichtig zu bündeln. Je nach Ausstattung der Sendeeinheit der Antenne ist der Reflektorschirm zweidimensional in Form eines Zylinderausschnittes oder dreidimensional in Form eines Kugelausschnittes gebogen. Diese Form der Antenne besitzt einen der höchsten Antennengewinne von >20 dBi und wird in der Radartechnik am häufigsten für große Distanzen eingesetzt. Je nach Größe wird der Reflektor meist nicht solide, sondern als Metallgitter gefertigt, dessen Lücken kleiner als ein Zehntel der Wellenlänge der Betriebsfrequenz sein müssen. Dies verhindert die Reflektion niedrigerer Frequenzen. Trotz ihrer guten Bündelung besitzen auch Parabolantennen ein eher keulenartiges Abstrahlverhalten, jedoch deutlich schmaler als das anderer Antennentypen.

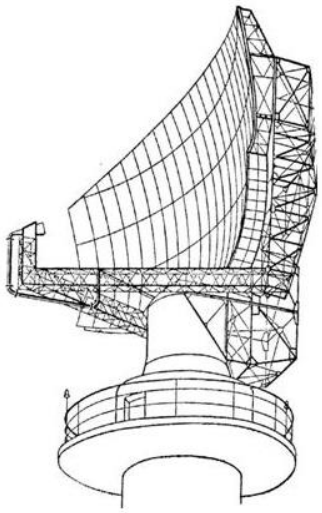


Abbildung 7 Darstellung eines typischen Parabolantennen-Radars

Eine im maritimen Einsatzgebiet verbreitete Antennenform ist der Schlitzstrahler [19], bekannt durch seine typischen Rotationen auf Schiffen. Der Schlitzstrahler wird häufig in Navigationsradaren mit Frequenzen zwischen 300 MHz und 25 GHz eingesetzt.

Charakteristisch ist sein Aufbau. Er besteht in der Regel aus einem elektrisch leitenden Hohlleiter, in den mehrere schmale Schlitze geschnitten sind. Die Größe der Schlitze richtet sich nach der Betriebsfrequenz. Während die Länge des Schlitzes auf die Hälfte der

Abbildung 8 Kommerzielles Beispiel eines Schlitzstrahlers

Wellenlänge der Betriebsfrequenz festgelegt ist, wird die Breite des Schlitzes auf ein Zehntel der Wellenlänge bemessen.

Abbildung 9 Foto einer Cosecans²-Antenneinheit

Im Bereich der Flugsicherheit wird häufig eine besondere Form der Parabolantenne eingesetzt, die sogenannte Cosecans²-Antenne [19]. Durch Modifizierung der ursprünglichen Parabol-Form kann eine wesentlich bessere Raumabtastung erreicht werden. Meist werden dabei die Krümmung des Reflektors einer Antenne angepasst und die Anzahl der Sendeeinheiten erhöht, indem man ein Antennen-Array erzeugt. Damit ist die gleichzeitige Ausstrahlung mehrerer Sendesignale und eine genaue, dreidimensionale Ortung möglich.

Im stationären Einsatz, jedoch hauptsächlich im maritimen Bereich, wird noch eine weitere Antenne verwendet: die Krähennestantenne [19]. Dabei handelt es sich im eigentlichen Sinne um ein Antennensystem. Entwickelt und patentiert vom Fraunhofer-Institut, nutzt diese „Antenne“ eine große Anzahl einzelner Strahler, die räumlich verteilt in einer Kugel platziert sind und senkrecht auf einer Bodenplatte angeschlossen sind. Aufgrund der hohen Anzahl an Einzelantennen und deren dreidimensionaler Verteilung kann das Antennendiagramm digital gesteuert werden. Mit Hilfe digitaler Signalverarbeitung ist es möglich, unter anderem anhand von Phasen- und Laufzeitunterschieden, alle Objekte im Umkreis gleichzeitig zu detektieren. Das bedeutet, dass - mit ausreichender Rechenleistung - alles in der horizontalen 360° Sicht und vertikal oberhalb der Höhe der Bodenplatte gleichzeitig erfasst werden kann.
Auch hier leiten sich verschiedene Formen und Größen des Antennensystems der Betriebsfrequenz ab. Eine Krähennestantenne kann Maße von einem Meter bis hin zur Höhe eines Containerschiffes haben. Die Anzahl der Einzelstrahler beträgt in der circa zwei Meter hohen Variante für das X-Band knapp 2000.

2.2.5 Empfänger

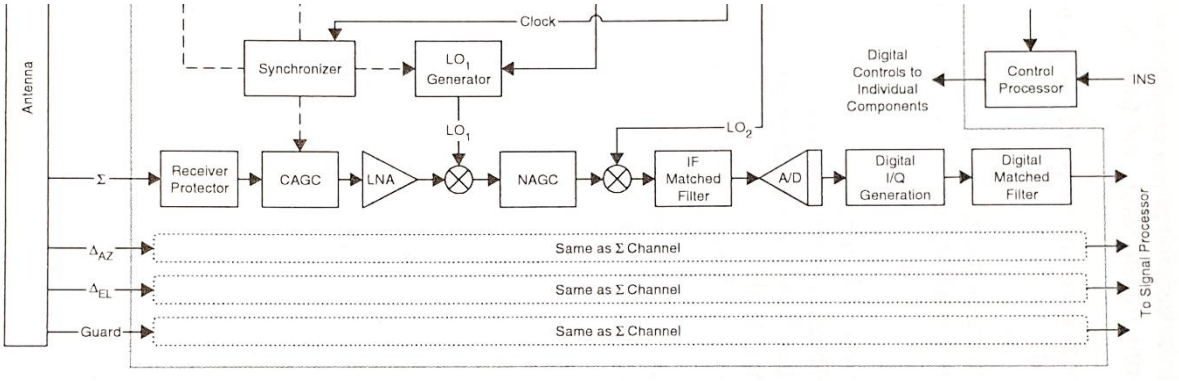


Abbildung 10 Empfänger; Ausschnitt aus dem Gesamtschaltbild (Abb5)

Der Empfänger besteht aus einem analogen Teil hinter dem Duplexer zum Schutz der empfindlichen Komponenten im weiteren Signalverarbeitungsverlauf sowie einem digitalen Teil zur Vorbereitung der Signalverarbeitung im nächsten Schritt. Dieser unterscheidet sich jedoch in Abhängigkeit der verwendeten Antenne und der gewünschten Auswertungsmöglichkeiten, wie zum Beispiel zweidimensionaler oder dreidimensionaler Detektion. Daher gehe ich hier nur auf die generell vorhandenen Komponenten ein.

Da der Duplexer nicht exakt zwischen Senden und Empfangen umschalten kann, kommt es immer wieder vor, dass die hohe Sendeleistung, auch Gain genannt, des Senders für kurze Schaltzeiten auch am Empfänger ankommt. Diese wird durch den sogenannten „Receiver Protector“ sowie durch zwei weitere „Gain-Control“-Einheiten limitiert und das Empfangssignal auf das Arbeitsniveau des Empfängers heruntergeregelt. Das eigentliche Sendesignal von der Trägerfrequenz ins Basisband gebracht. Das Signal durchläuft dann einen analogen Matched-Filter (Kapitel 2.3.5) zur Optimierung des SNR und wird anschließend analog-digital gewandelt. Meist wird in digitalen Puls-Doppler-Radaren im Anschluss an die Analog-Digital-Wandlung noch einmal ein Matched-Filter zur weiteren Optimierung genutzt. Im Anschluss beginnt die digitale Signalverarbeitung.

2.2.6 Signalauswertung und -darstellung

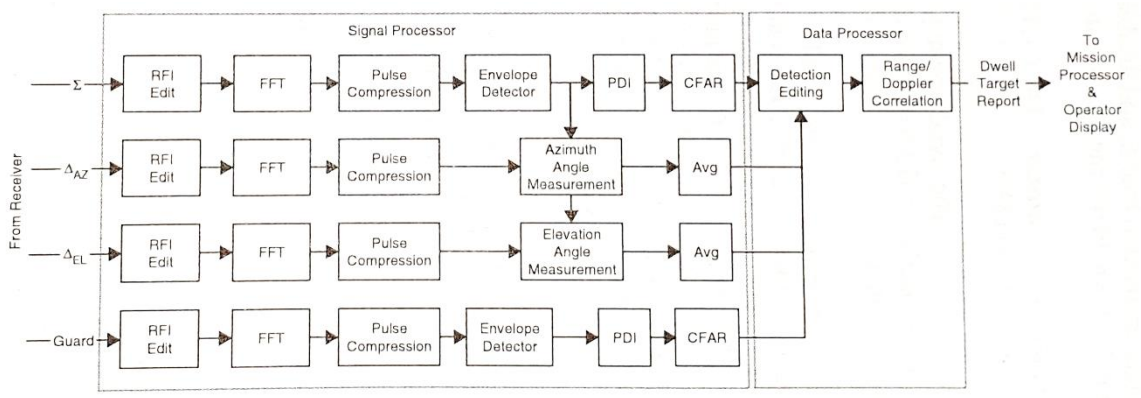


Abbildung 11 Signal-Prozessor zur digitalen Signalverarbeitung; Ausschnitt aus dem Gesamtschaltbild (Abb5)

Der Signalprozessor des Radargeräts ist entsprechend seiner Aufgaben unterschiedlich aufgebaut. Hier wird das Empfangssignal aus dem Zeitbereich in den Frequenzbereich transformiert, durchläuft eine Puls-Kompression zur Verbesserung der Entfernungsauflösung (behandelt in Kapitel 2.4.3) und wird dann an die eigentliche Detektoreinheit weitergegeben. In der obigen Abbildung ist ein Puls-Doppler mit einer dreidimensionalen Detektion zu sehen. Hier werden Daten mehrerer Empfangsantennen, zum Beispiel eines Cosecans²-Antennensystems, in verschiedenen Kanälen verarbeitet, um Azimut (horizontaler Winkel) und Elevation (Höhe bzw. Höhenwinkel) zusätzlich zur Geschwindigkeit zu detektieren. Er erkennt die Präsenz eines Objektes und leitet diese Informationen an die nächste Einheit weiter zur eigentlichen Bestimmung der Entfernung eines Objektes sowie, mit Hilfe einer Doppler-Filterung, auch dessen Geschwindigkeit.

Diese Informationen werden dann an das Betriebssystem zur Speicherung und/oder Darstellung weitergegeben.

2.3 Grundlagen der Nachrichtentechnik

2.3.1 Antennengewinn und Abstrahlverhalten

Wie in Kapitel 2.2.4 „Antenne“ beschrieben, besitzen die unterschiedlichen Antennenformen ein unterschiedliches Abstrahlverhalten, welches sich aus dem dazugehörigen Antennendiagramm ermitteln lässt [1,4,19]. Daraus kann die sogenannte Richtwirkung D der Antenne abgeleitet werden. Dabei kann das Antennendiagramm sowohl im kartesischen als auch im Polar- Koordinatensystem in verschiedenen grafischen Darstellungsformen abgebildet werden.
Die Richtwirkung D ermittelt sich aus der Leistungsdichte einer Antenne im Vergleich zum theoretischen Isotropstrahler (idealer Kugelstrahler). Der Antennengewinn G errechnet sich aus der Richtwirkung D der Antenne und deren Effizienzgrad η und gibt damit die tatsächlich abgestrahlte Leistung als Faktor in dBi (deziBel isotrop) an.

$$G = D * \eta$$

Formel 2.3.1.1, der Antennengewinn G

Die gewünschte Richtcharakteristik respektive der gewünschte Antennengewinn kann durch gezielte Planung beim Entwurf einer neuen Antenne erreicht werden. Hierbei werden sowohl mechanische als auch elektrische Optimierungen angewendet, unter anderem in Form von analogen und digitalen Filtern. Bei einer stark gerichteten Antenne wird als Strahlbreite der Bereich bezeichnet, in dem noch mindestens die Hälfte der maximalen Leistung abgestrahlt wird. Die Strahlbreite Θ wird auch als Öffnungswinkel bezeichnet [19]. Die Grenzen dieser „Hauptkeule“ werden als die Punkte im Raum bzw. im Antennendiagramm definiert, bis zu denen die maximale Leistung im Vergleich zur Hauptlinie um 3 dB geringer ist. Ein Beispiel für die Darstellung der Strahlbreite Θ in einem Antennendiagramm findet sich in Abbildung 12 auf der folgenden Seite.

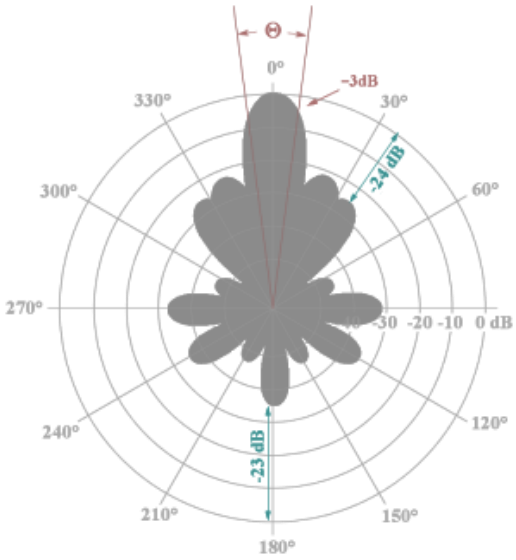


Abbildung 12: Ein typisches Antennendiagramm einer gerichteten Antenne

Wie aus dieser Abbildung ersichtlich wird, gibt es häufig nicht nur eine Strahlrichtung. In Antennendiagrammen finden sich immer sogenannte „Nebenkeulen“. Diese sind bei Richtantennen unerwünscht, da sie Sendeleistung in eine unbeabsichtigte Richtung abgeben. Das Leistungsverhältnis zwischen Hauptkeule und größter Nebenkeule nennt man Nebenkeulendämpfung. Vorteilhaft ist ein möglichst hoher Wert, da eine Antenne damit nur gering in andere Richtungen streut. In der oben dargestellten Grafik sind zwei Beispiele für Nebenkeulen mit ihrem jeweiligen Dämpfungswert abgebildet.

Betrachtet man ein Übertragungsmodell mit Sender und Empfänger, so lässt sich die am Empfänger ankommende Signalleistung P_s mit folgender Formel [25] berechnen:

$$P_S = P_{T_x} * G_1 * G_2 * D_s$$

Formel 2.3.1.2, Signalleistung P_S am Empfänger

Hierbei stellen G_2 den empfangsseitigen Antennengewinn, P_{T_x} die Leistung des Senders und D_s einen allgemeinen Dämpfungsfaktor äußerer Einflüsse dar.

2.3.2 Bandbreite, Datenrate, Informationsübertragung und der Übertragungskanal

Die Informationsübertragung in der Nachrichtentechnik basiert auf der zu übermittelnden Informationsmenge I und dem gewählten Übertragungskanal (kurz „Kanal“) [25]. Dieser bestimmt einige Faktoren der Datenübertragung wie zum Beispiel Übertragungseigenschaften und Störeinflüsse.

Für drahtlose Kommunikation wird überwiegend der AWGN-Kanal gewählt, der „Additive White Gaussian Noise“- Kanal (im Deutschen: Additives Weißes Gaußsches Rauschen).[3] Dieser kombiniert das „Weiße Rauschen“ mit seiner, über sämtliche Frequenzen betrachtet, konstanten Leistungsdichte mit einer Gaußschen Verteilungskurve als Amplitude und simuliert dabei ein überlagerndes Störsignal.

Ein Kanal besitzt zudem eine sogenannte Kanalkapazität C [25]. Sie gibt die maximal mögliche Datenmenge an, die pro Zeiteinheit übertragen werden kann. Sie errechnet sich aus der möglichen Bandbreite B und dem „Signal-zu-Rauschabstand“ (SNR, engl. „Signal to Noise Ratio“), dem Leistungsunterschied zwischen Nutzsignal und Rauschen.

$$C = B * log_2(1 + SNR)$$

Formel 2.3.2.1, Kanalkapazität

Damit ergibt sich für die maximale Informationsmenge I, die über die Zeit T übertragen werden kann, folgende Gleichung:

$$I \leq C * T$$

Formel 2.3.2.2, Informationsmenge bei T Übertragungszeit

Dabei ist zu beachten, dass ein ursprüngliches bit Information nur im seltensten Fall durch ein einzelnes Bit übertragen werden kann. Daher gilt, dass die Informationsrate J meist kleiner ist als die Datenrate R, die physisch übertragenen Bit pro Zeit.

$$J = \frac{I}{T} \text{ und } R \geq J$$

Formel 2.3.2.3, Informationsrate J und die Datenrate R

Dabei ist es mit Hilfe sogenannter Kanalcodes (siehe Kapitel 2.3.4) jedoch möglich, die Informationsrate J dicht an die Kanalkapazität C anzunähern ($J \sim C$). Dabei kann es vorkommen, dass die Datenrate R die Kanalkapazität C übersteigt, ohne dass Übertragungsfehler auftreten.

Die Bandbreite B besitzt eine obere und untere Grenzfrequenz, in deren Intervall sich die wesentlichen Frequenzen eines Signals befinden. Es gibt keine einheitliche Definition zur Festlegung der oberen und unteren Grenzfrequenz.

Antennen besitzen ebenfalls eine Bandbreite. Sie wird an dem Frequenzbereich bemessen, in dem die Antenne ihre geforderten Eigenschaften, wie zum Beispiel ihre Richtwirkung, erfüllt. [19] Aufgrund ihrer Bauart besitzen die meisten Antennentypen eine geringe Bandbreite von etwa 5% bis 10% ihrer Betriebsfrequenz. Für größere Bandbreiten sind spezielle Antennen erforderlich.

2.3.3 Doppler-Effekt

Der Doppler-Effekt, auch Doppler-Verschiebung genannt, ist eine Veränderung der Frequenz, die bei sich bewegenden Sendern und/oder Empfängern auftritt [1,4,19,25]. Ein geläufiges Beispiel hierfür sind die Sirenen von Einsatzfahrzeugen. Bewegt sich eines dieser Fahrzeuge (Sender) mit eingeschalteter Sirene auf den Hörer (Empfänger) zu, so steigt die Frequenz und damit die Tonhöhe. Entfernt sich im umgekehrten Fall das Fahrzeug, nimmt die Frequenz respektive die Tonhöhe ab. Dabei bleibt die ausgesendete Frequenz des Senders gleich. Dieser Effekt gilt nicht nur für die Akustik, sondern auch für elektromagnetische Wellen. Er ist wesentlicher Bestandteil der Signalverarbeitung im Mobilfunk, aber auch in der Radartechnik, wie zum Beispiel in Kraftfahrzeugen oder in der Flugüberwachung.

Mathematisch wird der Doppler-Effekt als Frequenzabweichung von der Grundfrequenz beschrieben. Davon leitet sich auch der Begriff der Doppler-Verschiebung (f_D) ab. Die

Faktoren für die Verschiebung sind die Grund- oder Trägerfrequenz (f_T) sowie die Differenz in der Geschwindigkeit (v_D) und in der Bewegungsrichtung (α) des Senders und Empfängers. Daraus ergibt sich folgende Gleichung:

$$f_D = f_T \cdot \frac{v_D}{c_0} \cdot cos(\alpha)$$, mit c_0 als Lichtgeschwindigkeit

Formel 2.1.1, Verhältnis Wellenlänge zu Frequenz bei elektromagnetischen Wellen

2.3.4 Kanalcodes und Modulationsarten

Wird eine Information übertragen, so wird diese im ersten Schritt meist in ein Dateiformat oder in einen Datencontainer kodiert. Diese Art der Kodierung nennt man Quellenkodierung [25]. Ziel ist es, redundante Daten und damit die zu übertragende Datenmenge zu reduzieren. Diese quellenkodierte Information wird dann erneut kodiert. Diesen Vorgang nennt man Kanalkodierung [3,25]. Das Prinzip ist, gezielt Information hinzuzufügen, mit deren Hilfe Fehler erkannt und korrigiert werden können. Es gibt mehrere unterschiedliche Arten von Kanalcodes. Aus der Vorlesung sind die folgenden Kanalcode-Typen bekannt:

a) Blockcodes
b) Faltungscodes
c) Turbo-Codes

Sie unterscheiden sich darin, wann und wie die zusätzliche Information hinzugefügt wird.

Im Anschluss an die Kanalkodierung findet die sogenannte Modulation statt. Dabei wird die kodierte Information in die Übertragungsfrequenz eingebettet. Es gibt verschiedene Arten ein Nutzsignal oder eine binäre Information zu modulieren.
Ein Signal besteht aus drei modifizierbaren Komponenten: der Amplitude, der Frequenz und der Phase oder auch Phasenlage. In der analogen Radioübertragung werden zum Beispiel mit der Amplituden- und Frequenzmodulation zwei dieser Komponenten genutzt.
In der digitalen Signalverarbeitung können alle drei Komponenten für eine digitale Modulation verwendet werden. Dies wird dann als „Amplitude Shift Keying“ (ASK), “Frequency Shift Keying” (FSK) oder “Phase Shift Keying” (PSK) bezeichnet. Für jede dieser drei

Grundmodulationsarten gibt es weitere Unterarten. Ein Beispiel für ein einfaches PSK-Verfahren ist das BPSK-Verfahren, das „Binary Phase Shift Keying“, bei dem das Signal in der Phase um 180° gedreht wird, sobald sich das zu übertragende Bit im Wert von 0 zu 1, bzw. 1 zu 0 ändert.

2.3.5 Matched-Filter

Der Name „Matched-Filter“ wurde aus dem Englischen in den deutschen Sprachgebrauch übernommen und hat den dort bislang verwendeten, deutschen Namen, „Optimalfilter“, verdrängt. Der Matched-Filter oder auch „Signal-angepasste Filter“ hat die Aufgabe, den Signal-zu-Rausch-Abstand zu erhöhen.[1,4,19]
Wird ein Signal über einen Kanal übertragen, überlagert es sich mit dem Grundrauschen des Kanals. Dadurch wird es verzerrt und ist schwerer zu detektieren.
Ist dem Empfänger die verwendete Struktur der Übertragung, vor allem die Modulationsart, bekannt, so kann empfängerseitig ein entsprechend angepasster Filter konstruiert werden, der das Rauschen unterdrückt und das Nutzsignal hervorhebt. Dies geschieht mit Hilfe der Autokorrelation zwischen Sendesignal und Filter.
Beim Einsatz eines normalen Puls-Doppler-Radars wird der „Matched-Filter“ für diesen Zweck teilweise sogar mehrfach eingesetzt.

2.4 Nachrichtentechnische Grundlagen des Puls-Doppler-Radars

Im folgenden Abschnitt werden einige nachrichtentechnische Aspekte des Puls-Dopplers näher erläutert. Dies bildet mit eine der wichtigsten Grundlagen zur Untersuchung möglicher Wellenformen für eine gemeinsame Radar- und Kommunikationsfunktionalität.

2.4.1 Der Impuls, die Pulse-Repetition-Frequency und die Trefferzahl

Die “Pulse Repetition Frequency” (PRF), im Deutschen auch Impulsfolgefrequenz genannt, gibt die Anzahl gesendeter Impulse pro Sekunde an. Aus ihr ergibt sich die „Pulse Repetition Time“ (PRT) oder auch das „Pulse Repetition Interval“ (PRI), die Zeit, die zwischen zwei Sendeimpulsen liegt. [1,4,19,25] Diese bestimmt die maximale Reichweite eines Radars für eine genaue Ortung. Für eine eindeutige Ortung muss das Signalecho vor dem nächsten abgegebenen Sendeimpuls ankommen.
Ein Sendeimpuls lässt sich allgemein in drei Abschnitte einteilen: die Sendezeit τ, die Empfangszeit t_r und die sogenannte Totzeit t_d. Dies ist in der folgenden Formel dargestellt und aus der Grafik ersichtlich.

$$PRT = \tau + t_r + t_D$$

Formel 2.4.1.1, Pulse Repetition Time

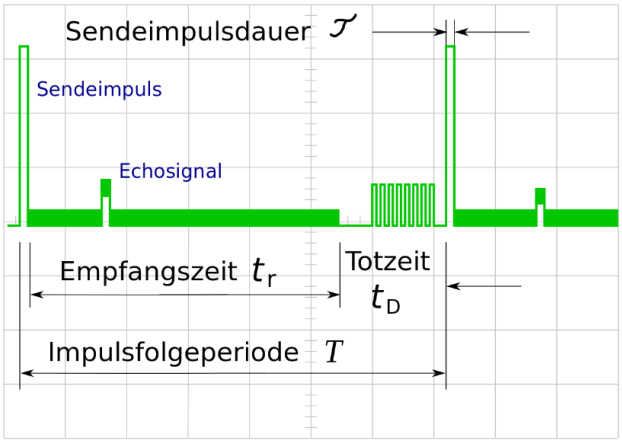


Abbildung 13: Zeitliche Darstellung eines Impulsintervalls

Für die Detektion ist eine volle Reflektion eines Sendeimpulses notwendig, somit muss das Echo vollständig innerhalb der Empfangszeit liegen. Damit gilt grundsätzlich für die maximale eindeutige Reichweite r_{max} [in km][1,4,19,25]:

$$r_{max} = \frac{c_0 * \left(PRT - (\tau + t_D)\right)}{2} \qquad oder\ \ vereinfacht: \qquad r_{max} = \frac{c_0 * t_r}{2}$$

Formel 2.4.1.2, maximal Detektionsreichweite r_{max}

Die Totzeit t_D verringert dabei die maximale eindeutige Detektionsreichweite, da sie die Empfangszeit t_r verkürzt. Jedoch besitzt nicht jedes Radar eine solche Totzeit. In diesem Fall lässt sich die Formel daher auch wie folgt darstellen:

$$r_{max} = \frac{c_0 * (PRT - \tau)}{2}$$

Formel 2.4.1.3, Spezialfall für r_{max} mit t_D=0

Die Sendezeit entspricht der Zeit, in der die Antenne das Sendesignal abstrahlt. Sie hat Auswirkung auf den sogenannten Blindbereich des Radargeräts, auch Entfernungsauflösung genannt, dessen Radius die zur Detektion benötigten Mindestentfernung r_{min} ist.

$$r_{min} = \frac{c_0 * \tau}{2}$$

Formel 2.4.1.4, Bestimmung des Bildbereichs

Die Trefferzahl ist ein essenzieller Teil der Detektion und leitet sich von der sogenannten Beleuchtungszeit T_D ab. Diese ist abhängig von der Umdrehungsgeschwindigkeit n [u/min] des Radars und beschreibt die Zeit, in der ein Punkt im Raum innerhalb der Strahlbreite (Hauptkeule) des Radars bleibt. Nehmen wir an, dieser Punkt im Raum ist ein Objekt, so steht ein Treffer für einen Sendeimpuls, der auf dieses Objekt trifft und wieder reflektiert wird. Damit lässt sich die Trefferzahl als Anzahl an Echos definieren, die in der Beleuchtungszeit reflektiert werden können.

Die Beleuchtungszeit T_D lässt sich mit folgender Formel berechnen:

$$T_D = \frac{\Theta}{\frac{360° * \mathrm{n}}{60}}$$

Formel 2.4.1.5, Beleuchtungszeit T_D

$\frac{n}{60}$ ist dabei die Umwandlung von Umdrehungen pro Minute in Umdrehungen pro Sekunde, da T_D in Sekunden gemessen wird. Ersetzt man die Einheit von n durch Umdrehungen pro Sekunde, so ergibt sich folgende übersichtlichere Formel:

$$T_D = \frac{\Theta}{360° * n}$$

Formel 2.4.1.6, Beleuchtungszeit T_D für n [1/s]

Aus den Impulsen pro Sekunde (PRF) kann die Anzahl an möglichen Treffern und damit die Trefferzahl m_s [19] abgeleitet werden. Das „s“ ist aus dem Englischen abgeleitet und steht für „scan“ – „(ab-)suchen“.

$$m_s = T_D * PRF \quad oder\ auch \quad m_s = \frac{T_D}{PRT}$$

Formel 2.4.1.7, Trefferzahl m_s

Die Trefferzahl m_s hat einen großen Einfluss auf die Detektionsfähigkeit eines Radars. Je größer sie ist, desto besser kann detektiert werden.

2.4.2 Grundlagen der Detektion

Die Radar-Detektion erfüllt vier grundlegende Aufgaben: alle vorhandenen (und gewünschten) Objekte wahrzunehmen und für jedes dieser Objekte die Entfernung, die Höhe und den horizontalen Winkel zu bestimmen.
Dafür sendet das Radar einen elektromagnetischen Impuls aus. Dieser wird an einem Objekt reflektiert und das Echo von der Antenne empfangen. Dabei werden grundsätzlich zunächst der horizontale Winkel und die Entfernung gemessen.

Das Radargerät wird horizontal in 360° aufgeteilt. Dabei entspricht der Bezugswinkel 0° geografisch Nord. [19] Die Antenne des Radars dreht sich im Uhrzeigersinn. Anhand der Antennenstellung lässt sich der horizontale Winkel eines erfassten Objektes bestimmen.

Die Entfernung eines Objektes wird anhand der Laufzeit t des ausgestrahlten und reflektierten Impulses gemessen. Dieser legt vom Sendezeitpunkt bis zum Empfang die doppelte Entfernung zurück. Daraus ergibt sich für die Entfernung r eines Objektes folgende Formel:

$$r = c_0 * \frac{t}{2} = \frac{c_0 * t}{2}$$

Formel 2.4.2.1, Bestimmung der Entfernung eines Objektes

Zu beachten ist dabei, dass je nach vertikalem Winkel α der Antenne, in der Entfernung r auch ein Höhenanteil h enthalten sein kann. Der Winkel α wird dabei oberhalb des Horizontes positiv und unterhalb des Horizontes negativ angegeben. Dieser Höhenanteil h lässt sich, mit r_h als realer Entfernung, durch folgende vereinfachte, geometrische Formel eliminieren:

$$r_h = r * cos(\alpha)$$

Formel 2.4.2.2 horizontale Entfernung r_h

In der Flugüberwachung beispielsweise ist diese Höheninformation jedoch gewünscht.
Die eigentliche Höhe h berechnet sich dann aus:

$$h = r * sin(\alpha)$$

Formel 2.4.2.2 Bestimmung der Höhe h

Die in der Praxis angewendeten Formeln sind dabei etwas komplexer, berücksichtigen sie unter anderem die Erdkrümmung zur Bestimmung der Entfernung und der Höhe.
Wird ein Antennensystem wie die Cosecans²-Antenne verwendet, können zur Berechnung dieser Informationen mehrere Radarimpulse gleichzeitig ausgewertet werden [19].

Die oben angeführten Formeln lassen sich nur bei einer stationären Radarstation mit stationären Zielen direkt anwenden. Sobald es sich um ein mobiles Radar und/oder bewegte Ziele handelt, kommen weitere einzubeziehende Faktoren, wie die Doppler-Verschiebung, zur Detektion hinzu. Mit Hilfe der Doppler-Frequenz lässt sich beispielsweise auf die relative Geschwindigkeit zwischen Radar und Objekt schließen[1,4,19,25]. Durch die Bewegung des Objektes beziehungsweise des Radars werden gleichzeitig die empfangenen Echosignale unschärfer. Zur Berechnung der gewünschten Informationen, wie Entfernung, Höhe, Richtung

und Geschwindigkeit, muss daher das eingehende Echosignal intensiv gefiltert, untersucht und verarbeitet werden.

Ein weiterer Aspekt bei der Detektion ist die Trefferzahl m_s, da selbst moderne Radargeräte je nach Anwendungsgebiet und Radartyp bis zu 20 Treffer zur genauen Detektion benötigen. Dabei werden die Informationen dieser Treffer teilweise gemeinsam verarbeitet, um genauere Ergebnisse zu erzielen.

2.4.3 Rausch- und Störunterdrückung

Bei einer Signalübertragung kommt es quasi immer zu Störungen, die die Signalleistung verringern und das Signal verzerren[25]. Mögliche Störquellen sind einzelne Komponenten, Umwelteinflüsse sowie andere Signalquellen. Die gängigste Störquelle in Form einer Signaldämpfung ist die atmosphärische Dämpfung als Teil der Kanaldämpfung bei drahtloser Kommunikation [19,25]. Sie besteht aus einer relativ stabilen Grunddämpfung und einer sogenannten „Wetter-Komponente“ (Regen, Nebel) [19]. Andere Leistungsverluste entstehen durch das Auftreten eines Rauschens. Während allgemeine Dämpfungen die Signalleistung vermindern, Rauschquellen das sowieso vorhandene Grundrauschen und reduzieren so den SNR und damit den Leistungsunterscheid zwischen Nutzsignal und Störsignalen. Jedes Gerät bzw. jede Komponente besitzt ein eigenes Grundrauschen. Bei qualitativ hochwertigen Bauteilen ist dieses jedoch einzeln betrachtet meist zu vernachlässigen. Mit steigender Komponentenzahl summieren sich diese Störeinflüsse allerdings auf und können letztendlich doch einen erheblichen Leistungsverlust auf der Empfängerseite bewirken. Daher ist es wichtig, jeden beeinflussbaren Störfaktor zu eliminieren oder zumindest zu minimieren. Dafür wird auf Seiten des Empfängers eine Reihe verschiedener Filter realisiert. [1,4,19]

Ein spezielles Filter ist die sogenannte Puls-Kompressions-Schaltung [1,4,19], die ursprünglich für ältere Radargeräte entwickelt wurde. Diese waren technisch noch nicht in der Lage, die für hohe Messentfernungen benötigte Sendeleistung in einer kleinen Impulsdauer τ zu erzeugen. Daher stieg τ und damit der Blindbereich des Radars. Um dies zu kompensieren, wurde die Puls-Kompressions-Schaltung entwickelt, eine Form des Matched Filter. Die Besonderheit im

Vergleich zu einem normalen Matched Filter ist die Kenntnis über das genaue Sendesignal. Um das zu realisieren, muss also sendeseitig ein genau definiertes Signal erzeugt werden. Hier kommt die sogenannte Intrapuls-Modulation zum Einsatz. Sie moduliert den Sendeimpuls mit einer Bitsequenz. Empfangsseitig sind sowohl die Modulationsart als auch die Bitsequenz bekannt. So lässt sich empfangsseitig ein nahezu perfekt abgestimmtes Filter konstruieren. Die am meisten verbreitete Bitsequenz findet sich in Form des sogenannten Barker-Codes in Kombination mit der BPSK-Modulation [19]. Der Barker-Code hat für eine Sequenzlänge n einen im Voraus bekannten Signal-Nebenkeulen-Abstand in dB. Dabei sind die Bitsequenzen in unterschiedlicher Länge n einzigartig, bei gleicher Länge n jedoch für jeden neuen Impuls identisch. Dieses Prinzip lässt sich noch weiter steigern, indem jedes auf den Sendeimpuls modulierte Symbol des Barker-Codes als eigener Teilimpuls betrachtet wird und dieser dann erneut mit dem Barker-Code gleicher Länge moduliert wird. Für die Länge n=13, eine typische und effiziente Sequenzlänge des Barker-Codes, ergeben sich dementsprechend aus 13 Teilimpulsen mit jeweils 13 eigenen Symbolen eine Anzahl von insgesamt 169 Symbolen. Diesen speziellen Aspekt der Intrapuls-Modulation möchte ich im Rahmen dieser Arbeit nutzen, indem ich den Barker-Code durch eine selbst generierte Bit-Sequenz als Informationsträger ersetze. Es können jedoch auch Formern der Frequenzmodulation zum Einsatz kommen, der Einfachheit der geplanten Simulation zuliebe und in Anlehnung an die größere Verbreitung der Barker-Code basierten BPSK-Modulation werden diese im Folgenden nicht weiter betrachtet.

Die Pulsintegration [1,4,19] ist eine weitere Möglichkeit, den SNR zu steigern. Das Konzept nutzt einen Zwischenspeicher, in dem die Amplituden mehrerer Echosignale gesichert und addiert werden. Der Pulsintegrator arbeitet dabei mit dem Zielverfolger zusammen, um die Amplituden bewegter Objekte besser zu verarbeiten. Dadurch wird die Wahrscheinlichkeit erhöht, dass ein Objekt entdeckt wird. Dieses Prinzip wird auch dazu genutzt, den in Kapitel 2.2.2 erwähnten Clutter zu filtern. Bei Clutter handelt es sich um anwendungsabhängige Störsignale, die bei der Radarauswertung fälschlicherweise als Objekte detektiert werden. [1,4,19,25] Aufgrund der Anwendungsabhängigkeit ist nicht festgelegt, welche Signale als Clutter gelten. In der Seefahrt zum Beispiel wird als sogenanntes „Sea-Clutter“ die von Wellen erzeugten Echos bezeichnet. Allgemein lässt sich also sagen: Clutter ist all das, was nicht für die angestrebte Detektion relevant ist. Für ein stationäres Radar ist es möglich, anhand einer

erstellten „Karte“ konstante Fehlerkennungen zu filtern. [19] Für andere Arten von Fehlern sowie für mobile Radare wird meist ein gemeinsamer Filter zur Clutter-Reduktion und Doppler-Filterung eingesetzt.

2.4.4 Doppler-Filterung

Bei sich bewegenden Radaren und/oder Zielen kann der in Kapitel 2.3.3 beschriebene Doppler-Effekt in Form einer Verschiebung der Arbeitsfrequenz um den Wert f_D auftreten. Da dieser Wert von der Differenz der Geschwindigkeit zwischen Radar und Ziel abhängt, schwankt er zwischen positiven und negativen Werten. Für eine genaue Detektion ist es jedoch nötig, dass das empfangene Echosignal trotz Frequenzverschiebung weiterhin analysier- und filterbar bleibt.

Hier kommt das Prinzip einer sogenannten Filterbank zum Einsatz.[19] Sie enthält für quasi jede Doppler-Verschiebung einen eigenen Matched Filter. Zu analogen Zeiten bedeutete dies einen hohen Schaltungsaufwand. In der heutigen Digitaltechnik kann eine solche Filterbank per Software generiert werden. Da jedes Filter auf eine spezielle Doppler-Verschiebung zugeschnitten ist, ist dessen Effektivität bei einer abweichenden Dopplerfrequenz gering. Dadurch sind die Signalamplituden am Ausgang jedes Filters unterschiedlich. Der Filter, dessen in der Konzeption berücksichtigten Dopplerfrequenz am dichtesten an der vorhandenen liegt, gibt am Ausgang die stärkste Signalamplitude aus. Ein sogenannter „Greatest-Of“ Filter leitet nur die stärkste Signalamplitude weiter.

Parallel kommen zwei weitere Filterarten zum Einsatz, der Clutter-Doppler-Filter [19] und der Zero-Doppler-Filter [19]. Sie arbeiten mit dem Pulsintegrations-Verfahren und vergleichen die Phasenlagen zweier aufeinanderfolgenden Echos. Dabei ist zu beachten, dass der Zero-Doppler-Filter nur stationäre Ziele erkennt, während der Clutter-Doppler-Filter sich bewegende Ziel erfasst. Ein Vergleich der drei Filterausgänge ermöglicht anschließend die exakte Filterung und Verarbeitung eines Signals, und damit verbunden, die genaue Ortung eines Ziels.

3. Untersuchung möglicher Wellenformen

Bei der Betrachtung der Grundlagen aus der Nachrichten- und Radartechnik sowie der Eigenschaften typischer Puls-Radare, lassen sich zur Wahl einer möglichen Wellenform für ein gemeinsam genutztes System sehr spezielle Anforderungen definieren.
Vorrangig aus Sicht der Radartechnik muss die ursprüngliche Radarfunktion erhalten bleiben. Damit ist zu fordern, dass die finale Wellenform weiterhin die grundlegende Detektion und damit die nötigen Filterschritte ermöglicht, wie die Rausch- und Störunterdrückung. Dazu muss die Datenübertragung auf mehrere Sub-Pulse aufgeteilt werden können. Während die Doppler-Filterung im maritimen Hochsee-Einsatz eher irrelevant ist, sollte diese Funktion für mögliche Einsätze im aeronautischen Bereich jedoch erhalten bleiben.
Aus Sicht der Nachrichtentechnik steht die zu übertragende Information und deren Ziel, der Empfänger, im Vordergrund. Diese Information muss codiert, übertragen und decodiert werden. Dabei ist zu beachten, dass aufgrund der rotierenden Antenne eine durchgehende Übertragung an ein oder mehrere Ziele nicht möglich ist. Zudem gibt es viele äußere Einflüsse auf die Übertragungsqualität, die teilweise zu einer starken Verschlechterung bis hin zu einem vollständigen Verlust eines Teils der Information führen kann. Bis dato vorhandene, konventionelle Fehlerkorrektur-Verfahren können dabei an ihre Grenzen gelangen [3]. Gesucht wird entsprechend ein Kanalcode, der mit einer für die Radartechnik möglichen Modulationsart arbeiten kann, dabei den Verlust ganzer Datenpakete zu kompensieren vermag sowie hierbei keine durchgehende Verbindung oder eine Rückmeldung des Empfängers benötigt.

3.1 Ergebnisse der Literaturrecherche

Ein großer Teil dieser Arbeit bestand in der Recherche zum Thema „Radar-kompatible Kanalcodes und Modulationsarten“. Während meiner Recherche stieß ich bis auf eine Ausnahme ausschließlich auf Veröffentlichungen zum Thema „RadComm“ im Anwendungsfall der Car-to-Car Kommunikation. Hier lag der Fokus fast vollständig auf der Verwendung der „Orthogonal Frequency-Division Multiplexing“-Modulation

(OFDM), einer Mehrträger-Modulation. Diese ist jedoch aufgrund des beabsichtigten Übertragungskanals und der oben genannten benötigten Funktionen nicht für die Verwendung mit einem Puls-Doppler-Radar geeignet. Der Sinn einer OMDF-Modulation liegt in der Verteilung der Information auf mehrere Trägerfrequenzen. Dies würde zum einen den technischen Schaltungsaufwand und die benötigte Rechenleistung für die Ausführung der Radarfunktionen deutlich steigern, zum anderen würde damit die verwendete Bandbreite des Impulses vergrößert. Je größer die Bandbreite des Impulses, desto mehr Rauschen ist in einem Empfangssignal enthalten und führt zu einer schlechteren Detektion auf Seiten des Empfängers. [19]

Da für die aktuelle Radarfunktion häufig auf einen Barker-Code mit einer BPSK-Modulation zurückgegriffen wird, scheint diese für den ersten Lösungsansatz gut geeignet. Vor diesem Hintergrund und nach einem Gespräch mit meinem Dozenten und Betreuer Prof. Dr.-Ing. Jan Mietzner verlagerte sich die Recherche daraufhin auf die Suche nach einem möglichen Kanalcode mit Fokus auf die „Fountain Codes“. Ursprünglich für den Binary Erasure Channel (BEC) entwickelt [3,10], in dem Bit entweder korrekt oder gar nicht übertragen werden, werden diese Codes den Anforderungen an einen zeitunabhängigen, gegenüber hohen Verlustraten toleranten und dabei mit geringem Rechenaufwand schnell zu berechnenden Code gerecht. 1998 wurden sie zum ersten Mal als Theorie der sogenannten „Digital Fountain“ (engl. für „digitaler Springbrunnen“) unter dem Titel „A Digital Fountain Approach to Reliable Distribution of Bulk Data“ am International Computer Science Institute der Universität Berkley in Kalifornien vorgestellt und 2002 offiziell veröffentlicht [5]. Entwickler Michael Luby gründete unter dem Namen „Digital Fountain Inc.“ seine eigene Firma (inzwischen von Qualcomm Inc. übernommen) und entwickelte diesen Ansatz weiter. Aus der Theorie des „Digital Fountain“ leitete sich später der Name einer eigenen Code-Klasse ab, die Fountain-Codes. Während der Recherche wurde ersichtlich, dass bis heute Fountain-Codes im Fokus von Veröffentlichungen stehen [6,7,8,9,11,12,13,14,15,16,17]. Sie werden stetig weiterentwickelt. Dabei wurde die Performanz über Binary Erasure Channels bereits mehrfach aufgezeigt und dokumentiert. Der Einsatz in AWGN-Kanälen ist für vereinzelte Anwendungsfälle ebenfalls untersucht worden [13]. Die meisten dieser Codes sind von der Qualcomm Inc. Patentiert [21,22,23]. Auffällig ist, dass zwar die theoretischen Ansätze und Ideen gut dokumentiert sind, Beispiele für praktische Umsetzungen jedoch fast komplett fehlen. Dies könnte auf die Patent-Situation zurückzuführen sein. Im Folgenden werden die Fountain Codes und eine Auswahl ihrer Variationen vorgestellt.

3.2 Fountain-Codes

Fountain Codes fallen unter die Kategorie der ratenlosen Codes [3,10…]. Sie basieren auf dem Bild eines Springbrunnens. Aus einer gegebenen Menge Informations-Bit (n) werden mit Hilfe eines entsprechenden Algorithmus eine quasi beliebige Anzahl codierter Bit (k) größer oder gleich der Anzahl an Informations-Bit generiert.

$$k \geq n$$

Formel 3.2.1, Zusammenhang unkodierte, kodierte Bit-Zahl

Die Besonderheit dieser Codes liegt in ihrem Dekodierpotential. Aus einer beliebigen Auswahl an codierten Bit lässt sich die ursprüngliche Information wiederherstellen, solange der Empfänger mindestens die Anzahl an codierten Bit empfängt, die der Anzahl an Informations-Bit entspricht [10,20].
Betrachtet man als Beispiel ein typisches Textdokument mit 30KB, so ergibt sich als Quellinformation eine Anzahl von 245.760 einzelner Bit. Fountain-Codes können aus diesen Bit nun theoretisch beliebig viele codierte Bit erzeugen. Werden aus diesen zum Beispiel eine Million codierte Bit generiert und übertragen, reichen dem Empfänger im Idealfall exakt diese 245.760 Bit, um die ursprüngliche Datei zu dekodieren. Es können also bis zu ~75% der codierten Bit verloren gehen, ohne dass die eigentliche Information beschädigt wird.
In den heutigen Versionen sind Fountain-Codes in der Lage, dieses Prinzip nicht nur auf einzelne Bit, sondern auf ganze Bit-Sequenzen[10,21,22,23], sogenannte Symbole, anzuwenden. Dies erhöht die Performance drastisch. Aus den mir zur Verfügung stehenden Patenten konnte jedoch kein konkreter mathematischer Algorithmus abgeleitet werden.

3.2.1 Tornado-Code

Der Tornado-Code zählt eigentlich nicht zu den Fountain Codes, er ist jedoch die Grundlage für die heutigen Raptor-Codes[6,7,9]. Er erzeugt eine festgelegte Anzahl an encodierten Bit. Dabei wird die ursprüngliche Information in Blöcke aufgeteilt und mit einem Fehler-Korrektur-Code, wie zum Beispiel einem LDPC-Code, ergänzt. Dies ergibt eine festgelegte Block-Länge,

die dem Empfänger bekannt ist. Nun werden die Bit XOR verknüpft, wieder in kleinere Blöcke aufgeteilt und erneut mit einem LDPC-Code ergänzt. Dies wird für eine festgelegte Anzahl an Schritten wiederholt. Sobald die festgelegte Anzahl an Wiederholungen erreicht ist, wird die Information mit einem Reed-Solomon Code ein letztes Mal vor Fehlern geschützt und übertragen. Zwar ist der Reed-Solomon Code relativ langsam im Enkodieren und Dekodieren, durch die bis dahin jedoch deutlich kleineren Blocklängen lässt sich dennoch eine gute Performance erzielen.
Empfängerseitig werden die Schritte in umgekehrter Reihenfolge durchgeführt. Allerdings lässt sich dabei keine absolut sichere Decodierung erreichen. Eine gewisse Fehlerwahrscheinlichkeit bleibt erhalten. Dies war einer der Hauptkritikpunkte an den Tornado-Codes und der Grund für deren Weiterentwicklung in die Rapid-Tornado Codes oder auch Raptor Codes.

3.2.2 LT-Codes

Die LT-Codes [5,8,9,10,14,15,17,20] sind die erste Form der Fountain-Code. Die Abkürzung steht für „Luby-Transform“, nach dem Namen einer der Entwickler.
LT-Codes sind ratenlos. Je nach Konzeptionierung können aus einer Anzahl an Quellensymbolen beliebig viele encodierte Bit erzeugt werden. Die Besonderheit der LT-Codes ist, dass sie im optimalen Fall zur Dekodierung nur eine beliebige Auswahl an codierten Bit benötigen, die der Menge der Quellsymbole entsprechen. Dies entspricht einer nahezu idealen Codierung.
In der Praxis und in Modellen mit realen Kanalbedingungen kann diese Performance jedoch nicht gehalten werden.
Eine besondere Stärke der LT-Codes liegt in ihrer Effizienz. Da der eigentliche Kodier- und Dekodiervorgang nur aus logischen Operatoren besteht, ist die Arbeitslast relativ gering und steigt auch mit größeren Quelldateien nur linear an, während andere Codes ein eher exponentielles Ansteigen des Rechenaufwandes mit steigender Quellengröße zeigen.
Die Grundidee des Kodierers besteht darin, jedem codierten Bit eine Anzahl an Quellsymbolen zuzuteilen und diese mit der XOR Operation zu verbinden. Wie viele Bit einem codierten Bit zugeteilt werden, wird anhand einer Wahrscheinlichkeitsverteilung festgelegt, die von der Länge K der eingehenden Information abhängt. Zur Bestimmung der am besten geeigneten Verteilung gibt es mehrere Studien und Dissertationen []. Am weitesten verbreitet ist die

„Robust Soliton Distribution“. Sie berechnet sich teils aus der „Ideal Soliton Distribution“, dabei wird zusätzlich eine akzeptable Fehlerwahrscheinlichkeit δ bestimmt. Diese gibt an, mit welcher Wahrscheinlichkeit ein Fehler im Dekodiervorgang auftritt. Die „Ideal Soliton Distribution“ wird mit k als Anzahl kodierter Bit mathematisch definiert als []:

$$\mathrm{p(i)} = \begin{cases} 1/\mathrm{k} & \mathrm{i} = 1 \\ 1/i * (i-1) & \mathrm{i} = 2, \ldots, \mathrm{k} \end{cases}$$

Formel 3.2.2.1: Ideal Soliton Distribution

Für die „Robust Soliton Distribution“ [] wird jeder Wert der „Ideal Soliton Distribution“ mit dem Wert T_i aus folgender Formel addiert und mit dem Faktor β normalisiert.

$$\tau(\mathrm{i}) = \begin{cases} R/ik & \mathrm{i} = 1, \ldots, \mathrm{k/R} - 1 \\ R * ln(R/\delta)/k & i = k/R \\ 0 & i = k/R + 1, \ldots, k \end{cases}$$

$$\beta = \sum_{i=1}^{k} p(i) + \tau(i)$$

Formel 3.2.2.2: Robust Soliton Distribution und der Normalisierungsfaktor β

Für die finale Verteilungskurve ergibt sich dann:

$$\mu(\mathrm{i}) = \big(\rho(\mathrm{i}) + \tau(\mathrm{i})\big)/\beta$$

Formel 3.2.2.2: Robust Soliton Distribution

Die entstehende Funktion gibt an, mit welcher Wahrscheinlichkeit ein codiertes Bit die angegebene Ordnung i hat. Die Ordnung entspricht dabei der Anzahl an Quell-Bit, mit denen das codierte Bit verknüpft ist. Die Ordnung beträgt dabei mindestens „1“.

Der eigentliche Kodierer arbeitet nach folgendem Schema:

- Unter Berücksichtigung der Wahrscheinlichkeitsverteilung wird zufällig die Ordnung des nächsten codierten Bit ermittelt
- Anschließend wird zufällig aus den Quell-Bit eine der Ordnung entsprechende Anzahl Bit ausgewählt
- Diese werden XOR verknüpft
- Das Ergebnis ist das codierte Bit

Dieser Prozess kann auch über eine Matrix gesteuert werden, die mit den Quell-Bit, als Vektor gesehen, multipliziert wird. In ihrer ursprünglichen Form orientiert sich die Anzahl der Einsen in den Spalten und Zeilen der Matrix ebenfalls an der gewählten Wahrscheinlichkeitsverteilung. In abgewandelter Form kann eine maximale Ordnung der kodierten Bit festgelegt werden und damit die maximale Anzahl an Einsen pro Spalte.
Bei der Kodierung wird implizit davon ausgegangen, dass aufgrund der Abhängigkeit zur Eingangslänge K der Wahrscheinlichkeitsverteilung, sämtliche Quell-Bit verarbeitet werden.
In praktischen Anwendungen wird dabei meist die Kodier-Matrix vor Einsatz diesbezüglich einmal geprüft.

Der Dekodierprozess beginnt dann, wenn sich mindestens k empfangene Symbole im Speicher des Dekodierers befinden, wobei diese nicht aus derselben Quelle stammen müssen. Kodieren zwei Sender die gleiche Information mit dem gleichen Verfahren, können codierte Bit von beiden Quellen genutzt werden, um die ursprüngliche Information wiederherzustellen. Der Dekodierer benötigt zusätzlich entweder die Information, welche Bit benachbart sind (diese Information muss mit übertragen werden) oder es muss sowohl im Kodierer als auch im Dekodierer die gleiche Matrix verwendet werden.

Das Dekodierverfahren verläuft anschließend in folgenden Schritten:

- Zuerst werden alle Bit gesucht, die nur mit einem Quellsymbol verknüpft sind
- Diese werden dann dazu verwendet, mit ihnen verknüpfte Bit zweiter Ordnung zu dekodieren
- Dieses Vorgehen wird wiederholt, bis sämtliche Bit dekodiert wurden.

Dabei ist es wichtig, dass in jedem Durchgang Bit übrigbleiben, die nur mit einem (weiteren) unbekannten Bit verknüpft sind. Andernfalls schlägt der Dekodierprozess fehl. Eine weitere Schwierigkeit des Dekodierverfahrens ist die Wahrscheinlichkeitsverteilung, die anhand der Eingangslänge generiert wird. Diese erzeugt, zwar mit geringer Wahrscheinlichkeit, codierte Bit, die mit bis zu allen Quell-Bit verknüpft sein können. Die Wahrscheinlichkeit, dass ein so codiertes Bit decodiert werden kann, ist sehr gering. Dies war einer der Gründe für die Einschränkung der Ordnung der Kodiermatrix und die Entwicklung der Raptor-Codes.

3.2.3 Raptor-Codes

Raptor-Codes, kurz für Rapid-Tornado, [3,6,7,21,22,23] sind inzwischen eine eigene Unterart der Foutain-Codes und weisen je nach Einsatzgebiet feine Unterschiede auf, zugeschnitten auf die Bedürfnisse des Anwenders. Am bekanntesten sind der R10 Code, die finale Form des ersten Raptor-Codes, der RaptorQ, einer der aktuellsten Raptor-Codes aus dem Hause „Digital Fountain Inc.“, und der OpenRQ, ein Open Source Projekt für den Einsatz eines RaptorQ ähnlichen, lizenzfreien Raptor-Codes.
Ihren Ursprung haben diese Codes in dem ersten Raptor-Code, einer Weiterentwicklung des Tornado-Codes mit Aspekten des Luby-Transform Codes. Bei diesem Code wurde der Fokus auf die Effizienz des Kodierens und des Dekodierens gelegt, die mit Hilfe von im Voraus festgelegter Tabellen und Verteilungskurven gesteigert wurde. Bis auf das Open-Source Projekt sind alle Raptor-Codes, die ich gefunden habe, von der Digital Fountain Inc. patentiert.

Bei diesen Codes werden erst, ähnlich dem Tornado-Code, zusätzliche Prüf-Bit an die Quell-Bit angehängt. Anschließend wird eine leicht abgewandelte Form des LT-Codes für die finale Kodierung angewendet. Der wesentliche Unterschied besteht in der Begrenzung der maximalen Ordnung der codierten Bit. Dadurch werden die oben genannten Probleme des Dekodierens umgangen, das Erstellen einer Kodier-Matrix erleichtert und die Geschwindigkeit des Codes erhöht.

3.2.4 Online (Fountain) Code

Der Online Fountain Code, häufig auch nur als Online Code bezeichnet, [12,16] ist eine Abwandlung des Raptor Codes und wurde speziell für Online Dienste entwickelt. Während die traditionellen Fountain Codes meist mit festen Bit-Längen pro Kodiervorgang oder zumindest mit dem Wissen um die Informationslänge arbeiten, ist der Online Code darauf ausgelegt, ohne dieses Wissen in Echtzeit mit wenigen zusätzlichen codierten Bit (ergo geringem Overhead) zu arbeiten. Ein Beispiel aus der Praxis ist ein Live-Stream. Dem Kodierer ist nicht bekannt, wie lange der Stream laufen wird, gleichzeitig muss die aktuelle Information jedoch ausgesendet werden. Der Online Code generiert also einen Fountain-codierten Bitstrom parallel zum eingehenden Informationsstrom und sendet diesen aus.

4. Simulation

Im Folgenden werde ich auf Basis des von mir gewählten Codes eine MATLAB Simulation entwerfen und durchführen, die im späteren Verlauf zum einen die Funktionalität des Codes als Fehlerkorrekturcode und zum anderen auch seine Kompatibilität mit den Radarfunktionen überprüft.

4.1 Code Entscheidung

Aufgrund mangelnder mathematischer Genauigkeit und des Schwierigkeitsgrades der zur Verfügung stehenden Informationen, habe ich mich im ersten Schritt entschlossen, für die Simulation den Luby-Transform-Code zu verwenden. Dieser ist seit seiner Veröffentlichung im Jahr 2002 immer wieder Thema in Dissertationen und Publikationen im Fachbereich „Codierungstheorie“. Zur Überprüfung der Radar-Kernfunktion verwende ich, mit ausdrücklicher Genehmigung meines Dozenten und Betreuers Prof. Dr.-Ing. Jan Mietzner, eine von ihm entwickelte Version der Ambiguity-Funktion.
Im Verlauf der Entwicklung der MATLAB Simulation wurde deutlich, dass die im Rahmen meines Studienganges „Medientechnik“ erworbenen mathematischen Fähigkeiten für die Umsetzung des ursprünglichen Luby-Transform-Dekodierers nicht ausreichen. Grund hierfür ist, dass dem Dekodierer bekannt sein muss, an welcher Stelle das empfangene Bit in der Codesequenz stehen muss und welches die benachbarten Bit sind [10,20]. Dies lässt sich entweder durch synchronisierte Zufallsgeneratoren realisieren oder durch die Erzeugung eines Pakets aus codiertem Symbol und zusätzlichen Informationen. Andernfalls müssen mit Hilfe anderer Methoden, wie der „Belief Propagation“ oder „Maximum Liklihood“, und der Wahrscheinlichkeitsverteilung diejenigen Bit identifiziert werden, die erster Ordnung sind. Dies war in dieser Arbeit nicht zu realisieren. Der erste, inzwischen verworfene Codeentwurf mit direktem Bezug auf den ursprünglichen LT-Code ist dieser Arbeit angehängt.
In Anlehnung an online veröffentlichte Entwürfe wählte ich stattdessen für das vereinfachte Simulationsmodell eine auf einer Generatormatrix basierende Form des Luby-Transform-Code[10]. In diesem Modell ist die Matrix dem Empfänger bereits bekannt.

4.2 Modellentwicklung und Parameter

Vorrangiges Ziel dieses Modells ist die Entwicklung eines Luby-Transform-(De-)Kodierers um anschließend eine Überprüfung der Funktionalität und Leistungsfähigkeit des Luby-Transform-Codes durchführen zu können. Aus dieser Untersuchung lässt sich zur Analyse der Radarfunktionalität dann eine kodierte Bit-Sequenz einsetzen. Um den möglichen Umfang dieser Simulation und damit verbunden den Umfang dieser Arbeit nicht zu überschreiten, wird auf eine vollständige Simulation des Radars in dieser Arbeit verzichtet. Die Simulation wird auf folgende Komponenten beschränkt:

- Quellinformation
- Kodierer
- Kanal
- Dekodierer
- Bitfehlerrate

Die ursprüngliche Entwicklung des LT-Codes basierte auf dem Binary Erasure Channel (BEC), über den Bit entweder korrekt oder gar nicht übertragen werden. Dieses Prinzip wird nachfolgend übernommen.
Die wichtigsten Simulationsparameter können für eine bessere Übersichtlichkeit der Tabelle auf der folgenden Seite entnommen werden:

Variable	Bedeutung/Sinn/Wert
msg	Erzeugte binäre Nachricht
K	Länge der zu kodierenden Nachricht = Länge 1 Frames
N	Länge der kodierten Nachricht; vorgegeben
o_head	Anzahl an zusätzlichen Bit (Fehlerschutz)
o_head_perc	Prozentualer Anteil an o_head im Vergleich zu N
F	Anzahl der Frames, Berechnet aus msg-Länge/K
m_s	Trefferzahl, einzige Radar-Variable; Anzahl an Frames gleichen Inhalts (aktuell nur mit m_s=3 verwendbar)
frame	Ein K-bit langer Vektor
matrix	Generierte Matrix zum Kodieren, funktionsintern auch als G bezeichnet
encTxBits	Matrix aus dem Kodierer, enthält die kodierten Bit und den jeweilig dazugehörenden Index
encRxBits	Matrix nach der Kanalsimulation, enthält die kodierten Bit und den jeweilig dazugehörenden Index, mit ggf. fehlenden Spalten
decBits	Entschlüsselte Bit in Form eines Vektors
bec_error_p	Gibt die Wahrscheinlichkeit an, dass ein Bit auf dem Kanal verloren geht
part_failure	Gibt die Anzahl an verlorenen Frames an, die durch die wiederholte Übertragung trotz Kanal dekodiert werden konnten
Temp"X"_decBits	Temporär dekodierter Vektor, aus Vergleich wird wahrscheinlich richtiger Vektor ausgewählt
„channel_failure"	Gibt die Anzahl an Frames an, die trotz wiederholter Übertragung verloren gegangen sind
„perc_packet_loss"	Gibt temp_failure als Prozentzahl aus
„count_matrix_errors"	Gibt die Anzahl der Dekodier-Abbrüche an. Grund: Die LT-Dekodiervorgaben sind nicht (mehr) gegeben (Bsp. nach Ordnung 2 nur Ordnung 4 Bit)
„diff_msg_dec"	Vergleicht die Länge der ursprünglichen Nachricht mit der Länge der dekodierten Nachricht und gibt die Differenz an => verlorene Bit

4.3 Simulationsaufbau in MATLAB

In diesem Kapitel wird zuerst das Grundgerüst der Simulation vorgestellt, sowie geplante und bereits implementierte Ergänzungen erläutert. Grundsätzlich gilt dabei, dass nur die relevanten Abschnitte hier in Quellcode-Form abgebildet werden. Der komplette Code ist der Arbeit angehängt. Dir ursprüngliche geplante zusätzliche Simulink-Variante konnte aufgrund einer Inkompatibilität mit der Kodier- und Dekodierfunktion nicht durchgeführt werden. Die auf die Kapitel 4.2 genannten Komponenten reduzierte Form der Simulation trägt den Namen „Simulation_V1_Matrix.m“, in „V2“ befinden sich bereits Erweiterungen, wie in Kapitel 4.5 angegeben.
Die Simulation zur Überprüfung der Ambiguity („Mehrdeutigkeit) Range- und Doppler-

4.3.1 Die Simulation

Die grundsätzliche Simulationsablauf besteht in seiner Grundform aus sechs Schritten:

1. Erstelle eine zufällige Nachricht mit festgelegter Länge:

```
%generiere Msg
info_length = K*F;
for (aa=1:info_length+1)
    bit=round(rand);
    inf=[inf bit];
end
msg=inf;
```

   Im ersten Schritt wird die Länge der Information anhand der gewünschten Anzahl an Frames und der Größe K errechnet. Für diese Länge wird ein Vektor erstellt, der an jeder Stelle durch die verschachtelte Verwendung der Befehle „round()“ – „runde“ - und „rand“ – „generiere eine zufällige Zahl zwischen 0 und 1“ - zufällig eine 0 oder 1 erhält. Am Ende wird dieser Vektor in den „msg“-Vektor (kurz für „message“) kopiert, da er sonst durch die spätere Aufteilung in kleinere Sub-Nachrichten („Frames“) gelöscht würde und nicht mehr für den Vergleich mit dem dekodierten Vektor zur Verfügung stünde.

2. Generiere anhand der Nachricht und festgelegter Parameter die zum (De-)Kodieren benötigte Matrix:

```
%Erzeuge Matrix
if (RSD_flag == 0)
    matrix=GenerateGMatrix(K,N);
elseif (RSD_flag == 1)
    matrix=GenerateG(K,N);
elseif (RSD_flag == 1)
    [matrix,b]=genRSD_Matrix(K,N,p_dec_fail);
end
```

Im nächsten Schritt wird anhand einer sogenannten „flag“ (engl. für Flagge) entschieden, welche Funktion zur Generation der Kodiermatrix genutzt werden soll (siehe unter „Hauptfunktionen“). Diese Funktion erhält dann die Information über die Länge K des Eingangsvektors und die gewünschte Länge N des Ausgangsvektors.

3. Kodiere die Nachricht mit Hilfe der Matrix:

```
%erzeuge Frame
for (kk=1:K)
  new_bit=msg(1);
  msg(1)=[];
  frame=[frame new_bit];
end

%Kodierer
encTxBits=encLT(frame,matrix);
```

Zum Kodieren der Nachricht muss diese zuerst in kleinere Einzelteile, sogenannte „Frames“, zerlegt werden. Diese werden dann zusammen mit der verwendeten Matrix nacheinander an den Kodierer übergeben,.

4. Simuliere den Übertragungskanal (BEC):

Zur Simulation des Übertragungskanals wird anschließend nur die entsprechende Funktion aufgerufen und die kodierten Bit sowie die eingestellte Verlustwahrscheinlichkeit eingespeist.

```
encRxBits=simBEC(encTxBits,p_bec_error);
```

5. Dekodiere die Nachricht:

```
%Überprüft Dekodierbarkeit anhand der dem Kodierer bekannten Matrix
  Rx_length=size(encRxBits,2);
  known_K=size(matrix, 1);
  if (Rx_length<known_K)
      temp_failure=temp_failure+1;          %zählt verlorene Pakete
  else
      %Dekodierer
      temp_decBits,matrix_failure]=decLT(encRxBits,matrix);

      %Speichert jeden Durchlauf in einer anderen Variable
      count_matrix_failure=count_matrix_failure+matrix_failure;
      if (ll==1)
          temp1_decBits=temp_decBits;
      elseif (ll==2)
          temp2_decBits=temp_decBits;
      elseif (ll==3)
          temp3_decBits=temp_decBits;
      end
  end
```

Der Dekodierer „empfängt“ m_s mal den über den Kanal gesendeten kodierten Vektor entsprechend der Trefferzahl m_s (im aktuellen Stand auf 3 festgelegt). Er prüft zunächst, ob die Länge des empfangenen Vektors ausreicht, um generell eine Dekodierung durchführen zu können. Sollte dies nicht möglich sein, wird der Vorgang primär als „`temp_failure`“ – temporärer Fehler – gespeichert. Anschließend versucht der Dekodierer, die empfangenen Bit zu dekodieren und speichert jedes Ergebnis in einer anderen Variablen ab. Aufgrund der aktuellen Art der Implementierung können diese Empfangsvektoren eine unterschiedliche Länge aufweisen, da trotz Matrixfehler ein Vektor (kürzerer Länge) ausgegeben wird. Dies hat Auswirkungen auf eine geplante Erweiterung in Kapitel 4.4. Die Fehler werden vom Dekodierer gezählt. Anschließend werden die (drei) temporären Vektoren verglichen (hier nicht abgebildet, siehe Code im Anhang) und der Vektor ausgewählt, der die höchste Wahrscheinlichkeit hat, der richtige zu sein. Seine Bit werden in den dekodierten Vektor kopiert.

6. Vergleiche das Eingangs- und Ausgangssignal/Zähle die Fehler:

```
if (temp_failure>0)                          %zählt verlorene Pakete
      if (temp_failure<3)
          part_failure=part_failure+1;
      elseif (temp_failure==3)
          channel_failure=channel_failure+1;
      end
end
```

```
[…]

comp_decBits=size(decBits,2);
comp_msg=size(inf,2);

%Gibt am Ende Werte aus


part_failure
channel_failure
count_matrix_failure
diff_msg_decBits=(comp_msg-comp_decBits)
```

Im letzten Schritt wird als weitere Analyse der Längenunterschied zwischen ursprünglicher und dekodierter Vektornachricht bestimmt. Daraus ergeben sich die verlorenen Bit. Anschließend werden alle Fehler ausgegeben. Normalerweise würde dafür ein eigener Befehl genutzt. Da diese Simulation jedoch über die vorliegende Arbeit hinaus weiterentwickelt wird, sind die letzten Zeilen bereits eine Vorbereitung auf diese Weiterentwicklung und haben den gleichen Effekt wie ein Ausgabebefehl.

4.3.2 Funktionen

Folgende Funktionen finden in der Simulation ihren Einsatz:

GenerateG.m

```
function G = GenerateG(K, N, ColTH, TryNum)
```

Diese Funktion generiert eine binäre Kodier-Matrix anhand der zu kodierenden Informationslänge, der gewünschten Anzahl kodierter Bit, die entstehen sollen, sowie der Beschränkung der maximalen Ordnung. Es werden so viele Matrizen generiert, wie in „TryNum“ angeben. Anschließend werden die Matrizen verglichen und die bestmögliche Matrix ausgewählt. [24]

```
%maximize weight per row and minimize weight per col
[V, loc] = max(MeanSummerRow);
G = TryG{loc};
G = full(G);
```

Diese Funktion basiert auf dem Verfahren des Raptor-Codes.

GenerateGMatrix.m

```
function G = GenerateGMatrix(K, N)
```

Die GenerateGMatrix-Funktion erstellt eine zufällige Binärmatrix [24]. Dabei wird die zu erzeugende Matrix spaltenweise durchgegangen und ein Spaltenvektor mit einer zufälligen Anzahl an Einsen und Nullen erzeugt.

genRSD_Matrix.m

```
function [G,b]= genRSD_Matrix(K,N,p_dec_fail)
%Nutzt genRSD,pick_d um Spalten der Matrix zu erzeugen
```

Die hier angegebene Form der Matrix-Generierung berechnet anhand der erlaubten Fehlerwahrscheinlichkeit eine Wahrscheinlichkeitsverteilung nach der „Robust Soliton Distribution“. Mit dieser bestimmt sie den Grad jeder Spalte der Matrix und fügt zufällig eine entsprechende Anzahl an Einsen und Nullen zufällig hinzu.

```
%Geht die zu generierende Matrix spaltenweise durch und generiert anhand
%der gewählten Ordnung Einsen und Nullen
for n = 1: N
    degree=pick_d(rsd);
    create_for_column=[ones(1,degree) zeros(1,(K-degree))];
    for_column = randperm(create_for_column);
    column=for_column';
    G(:,N)=column;
end

end
```

Dabei ist zu beachten, dass diese Funktion eine weitere Ausgabe hat, die Variable b (für β). Diese Variable ist der Normierungsfaktor der Wahrscheinlichkeitsverteilung und dient nach Michael Luby [] als Richtlinie für die Bestimmung des Overheads.

genRSD.m

```
function [RSD_M,b]= genRSD(seq_length, error_delta)
```

„genRSD“ erzeugt auf Basis der Formeln [] zur Erstellung der R.S.D Wahrscheinlichkeitsverteilung in Abhängigkeit der Eingangsgröße (Länge des Informationsvektors).

pick_d.m

```
function pickdegree = pick_d(RSD_M)
```

Diese Funkion gibt anhand einer Wahrscheinlichkeitsverteilung eine Ordnung aus.

encLT.m

```
function [TxBits] = encLT(seq,matrix)
```

Nach der Formel zur Atrix-basierten Kodierung führt diese Funktion eine Vektor-Matrixmultiplikation Modulo 2.
Daraus entsteht der kodierte Vektor „TxBits“. Ein ähnlicher Ansatz findet sich auch im „Fountain-Toolkit“ [24].

decLT.m

```
function [decBits, matrix_failure] = decLT(encRxBits,Matrix)
```

Diese Funktion baut auf einer LT-basierten Dekodiererversion des „Fountain-Toolkit“ auf [24]. Die Abwandlung besteht in der Erstellung einer temporären Kodier-Matrix auf Basis der ursprünglichen Matrix, um über den BEC verlorene Bit zu kompensieren. Außerdem wird eine weitere Variable ausgegeben, um die Anzahl der Dekodier-Abbrüche aufgrund eines Ordnungsproblems der Matrix zu zählen. (siehe nächste Seite)

```
%Binärer LT-Decoder

%Bestimme anhand der eingegangen Empfangsmatrix das neue N
N = size(encRxBits, 2);


%erstelle und initialisiere temporäre Variable für die Matrix => angepasst
an eingegangener encRxBits-Matrix
temp_matrix =zeros(K,N);
%Nimmt den Index-Vektor der Empfangsmatrix und erstellt daran die temporäre
%Dekodiermatrix
chose_column=encRxBits(2,:);

for (i=1:N)
    cc=chose_column(i);                     % wähle den i. Index aus
    temp_matrix(:,i)=Matrix(:,cc);          % nimm die cc. Spalte der
ursprünglichen Matrix und nimm sie als i. Spalte
end
```

simBEC.m

```
function encRxBits = simBEC(encTxBits,p_bec_error)
```

Diese Funktion simuliert die Eigenschaft des BEC, dass übertragene Bit nur korrekt oder gar nicht übertragen werden. Dafür habe ich eine Variable „p_bec_error" eingeführt. Sie gibt die Wahrscheinlichkeit an, mit der ein einzelnes Bit bei der Übertragung verloren geht. Darauf aufbauend werden die Wahrscheinlichkeiten für „0 bis N Bit gehen verloren" berechnet, wobei N der Länge des empfangenen, kodierten Vektors entspricht.

```
%errechne Wahscheinlichkeit für X verlorene Bits
p=[];
q=(1-p_bec_error);
N=size(temp_encTxBits,2);
p(1)=q^N;                               %p(1) ist die Wahrscheinlichkeit, dass
kein Bit verloren geht
p(2)=(p_bec_error)*(q^(N-1))*16;        %p(2) ist die Wahrscheinlichkeit,
dass 1 Bit verloren geht//16 verschiedene Stellen!
for (i=2:(N-2))
    summe=0;
    a=1:(N-i+1);                        %Die Anzahl an Möglichkeiten entspricht
der Summe von 1 bis N-1 im ersten Druchlauf => +1!
    a=1./a;
    summe=sum(a);
    p_temp=((p_bec_error)^i)*(q^(N-i))*(summe);
    p=[p p_temp];
end
p(N)=((p_bec_error)^(N-1))*q*16;        %Vorletzte Wahrscheinlichkeit => 1
Bit leibt übrig
p(N+1) = (p_bec_error)^N;               %letze Wahrscheinlichkeit => kein Bit
bleibt übrig

sum_p=sum(p);
P_error=p/sum_p;                        %Normieren
```

Anschließend wird anhand dieser Wahrscheinlichkeit entschieden, wie viele Bit in diesem Durchlauf verloren gehen. Im Anschluss wird ein Vektor konstruiert, der die gleiche Länge wie der Eingangsvektor besitzt und entsprechend der Anzahl verlorender Bit Einsen enthält. Dieser wird mit dem „randperm“-Befehl in eine zufällige neue Reihenfolge gebracht. Daraufhin wird jede Spalte mit einer Eins bestimmt. Diese entsprechen den Spalten, die im Eingangsvektor gelöscht werden.

```
for (ii=1:chosen_errors)
    z=find(delete_array,1);
    temp_encTxBits(:,z)=[];
    delete_array(z)=[];
end
encRxBits=temp_encTxBits;
end
```

Sim_AF_test.m

[25]

Dies ist die von meinem Betreuer zur Verfügung gestellte Simulation zur Berechnung der Ambiguity-Funktion in den Dimensionen Entfernung und Doppler. Da bis kurz vor Ende dieser Arbeit nicht feststand, ob eine funktionierende Version der Code-Simulation entstehen wird, wurden in dieser Funktion nur minimale Änderungen vorgenommen.

```
%fs_adc = 200e6;            % Assumed sampling rate of ADC in Hz
fs_adc = 9e9;               % Angenommene Frequenz /9GHz Hafenradar / 1GHz
Flugradar

% Tx signal
Test_sequenz_ambiguity=[
```

[…]

```
data=Test_sequenz_ambiguity';           %nimmt eine kodierte Sendesequenz
und moduliert sie (BPSK)
bpskModulator = comm.BPSKModulator;
modData = bpskModulator(data);

s_tx1=modData';
%s_tx1 = randn(1,10000);     % Beispiel: Rauschsignal

% Ambiguity function in range & Doppler
%dR = dt/2*3e8;              % Range sample size in m
dR = dt/fs_adc;              % Range an Frequenz angepasst!!!
```

Diese bestehen in der Anpassung der Betriebsfrequenz bzw. Samplefrequenz, des verwendeten Eingangssignals in Form eines BPSK-gewandelten Bit-Vektors und der

Anpassung der Reichweitenfunktion an die Sampler-Frequenz. Dabei hat sich jedoch die Achsendimension der ausgegebenen Grafiken verändert, weshalb nur ein indirekter Vergleich von Rauschen und Signal möglich geworden ist.
Ziel weiterer Untersuchungen wäre dann die Anpassung der Funktion, sodass die Auswertung für verschiedene Frequenzen und Signalstärken erfolgen kann.

4.4 Mögliche Erweiterungen

Aufgrund des gescheiterten ersten Versuchs, einen Dekodierer zu programmieren, blieb für die endgültige Simulation nicht mehr so viel Zeit, wie ursprünglich eingeplant wurde. Somit sind einige der geplanten Funktionen noch nicht umgesetzt. Dennoch habe ich auf Basis der Simulation in der ersten Version (V1) die Simulation der zweiten Version bereits weiterentwickelt. Ziel war es, automatisch mehrere Messwerte für zwei (oder mehr) Variablen zu erzeugen. Die erste zu untersuchende Variable ist der prozentuale Overhead. So sollten die Fehlerarten in Abhängigkeit zur gewählten Differenz von Eingangslänge und Ausgangslänge simuliert werden. Die zweite Variable, die es zu untersuchen galt, war die Bit-Verlustwahrscheinlichkeit des BEC Kanals und deren Einfluss auf die Fehlerarten. Im Idealfall könnte so eine dreidimensionale Matrix entstehen, die in einem Simulationsablauf sämtliche benötigten Daten für eine 3D-Darstellung enthält. Auf der X-Achse würde dann die Bit-Verlustwahrscheinlichkeit, auf der y-Achse der prozentuale Overhead und auf der Z-Achse die unterschiedlichen Fehlerarten abgebildet werden. Um dies zu erreichen, war eine Anpassung der Simulation notwendig, da es mit Version 1 notwendig ist, für jeden Messwert eine separate Simulation zu starten. In Version 2 der Simulation ist es bereits möglich, für den prozentualen Overhead einen kompletten Messdurchlauf durchzuführen. Dabei wurde ebenfalls die Funktion der Wiederholungen implementiert. Diese soll für jeden Prozentwert eine festgelegte Anzahl an Durchläufen dazu nutzen, einen Mittelwert zu bilden. Dadurch sollen verlässlichere Werte entstehen.
Die Umsetzung der dritten Dimension scheiterte jedoch bisher, da der Dekodierer, wie bereits angesprochen, unterschiedliche Werte ausgibt. Dabei kam es zu dem Phänomen, dass die hinzuzufügende Dimension andere Ausmaße annahm. Dies ist programmtechnisch aber nicht umzusetzen. Ziel der weiteren Entwicklung wird die Erschließung der dritten Dimension sein, um eine dreidimensionale Ausgabe zu erzeugen.

Eine weitere Möglichkeit zur Erweiterung der Simulation ist die Umwandlung der Übertragungsart. Aktuell wird eine Sendematrix als Übertragung genutzt. Denkbar wäre jedoch auch ein einfacher Vektor, der als erstes Bit das kodierte Bit enthält. Die restlichen Bit enthalten die zusätzlich benötigten Informationen, wie den Index zum Dekodieren. Dies würde der Realität noch etwas genauer entsprechen und die Performanz besser reflektieren. Dafür ist jedoch eine weitere Funktion erforderlich, nämlich ein zusätzliches Fehlerkorrektur-Verfahren sowohl vor der Fountain-Kodierung als auch nach Generierung der kodierten Sequenz. Dies entspricht dann eher dem Vorgehen eines Raptor-Codes. In SIMULINK sind bereits einige Fehlerkorrekturverfahren implementiert.

Eine andere Möglichkeit der Simulationserweiterung ist das Hinzufügen weiterer Übertragungskanäle, die dann per „Flag“ in MATLAB oder per Simulationsblock in SIMULINK ausgewählt werden können. Gerade bei Betrachtung der Radartechnik sind Übertragungskanäle, wie zum Beispiel der AWGN-Kanal, interessant. Durch die Veränderung des Übertragungskanals könnten dann zusätzlich noch verschiedene Modulationsarten in Verbindung mit dem dann gewählten Fountain Code getestet werden.

In letzter Instanz könnte so ein komplettes Radarsystem mit integrierter Kommunikationsfunktion in MATLAB und SIMULINK erstellt und simuliert werden. Hier ist das Buch „MATLAB Simulations for Radar Systems Design“ von B. R. Mahafza und A. Z. Elsherbeni zu empfehlen, die darin verschiedene Ansätze zur Konzeptionierung und Simulation von Radarsystemen in MATLAB aufzeigen. [3]
Die Testfunktion zur Überprüfung der Ambiguity-Funktion sollte für die Verwendung verschiedener Trägerfrequenzen und unterschiedlicher Sendeleistungen sowie Kanalbetrachtungen ergänzt werden.

4.5 Ergebnisse

Die Realisierung eines Fountain Code-basierten Kodierers und Dekodierers ist für Anwender außerhalb des Fachbereichs „Codierungstheorie“ aufgrund der Komplexität schwer und nur unter Eingehen von Kompromissen umsetzbar. Dies hat das verwendete Simulationsmodell stark beschränkt und die Aussagekraft der Ergebnisse begrenzt.

Mit Hilfe der durchgeführten Simulation in Version 1 war es jedoch möglich, für einige ausgewählte Werte die gängigsten Fehlerarten zu simulieren.
In mehrfachen Durchläufen der Simulation wurde allerdings ersichtlich, dass die Wahl der Kodiermatrix bei sonst gleichen Parametern deutliche Unterschiede erzeugt, vor allem bei Wahl eines geringen Overheads (<=5%).
Weitere Untersuchungen und Erweiterungen waren aufgrund des begrenzten Zeitrahmens nicht möglich. Auf mögliche perspektivische Untersuchungsoptionen wurde hingewiesen.

Trotz intensiver Einarbeitung und Recherche konnte die ursprünglich geplante, komplexe Simulation mit selbst programmiertem Dekodierer nicht durchgeführt werden, da die hierfür erforderlichen mathematischen Anforderungen im Studiengang „Medientechnik“ nicht vermittelt wurden. Eine vertiefte Einarbeitung in dieses mathematische Themengebiet war ebenfalls aufgrund des begrenzten Zeitrahmens nicht möglich.

Die in der Aufgabe angeführte Überprüfung des Wellensignals mit Hilfe der Ambiguity-Funktion ist grundsätzlich möglich gewesen, wie in der Vorstellung der Funktion aber bereits erwähnt, war eine Anpassung der Reichweiten-Achse durch den Fokus auf die Umsetzung des Luby-Transformcodes gelegten Simulation leider nicht mehr möglich. Dennoch lässt sich das Signal grundlegend mit einem Rauschsignal vergleichen.

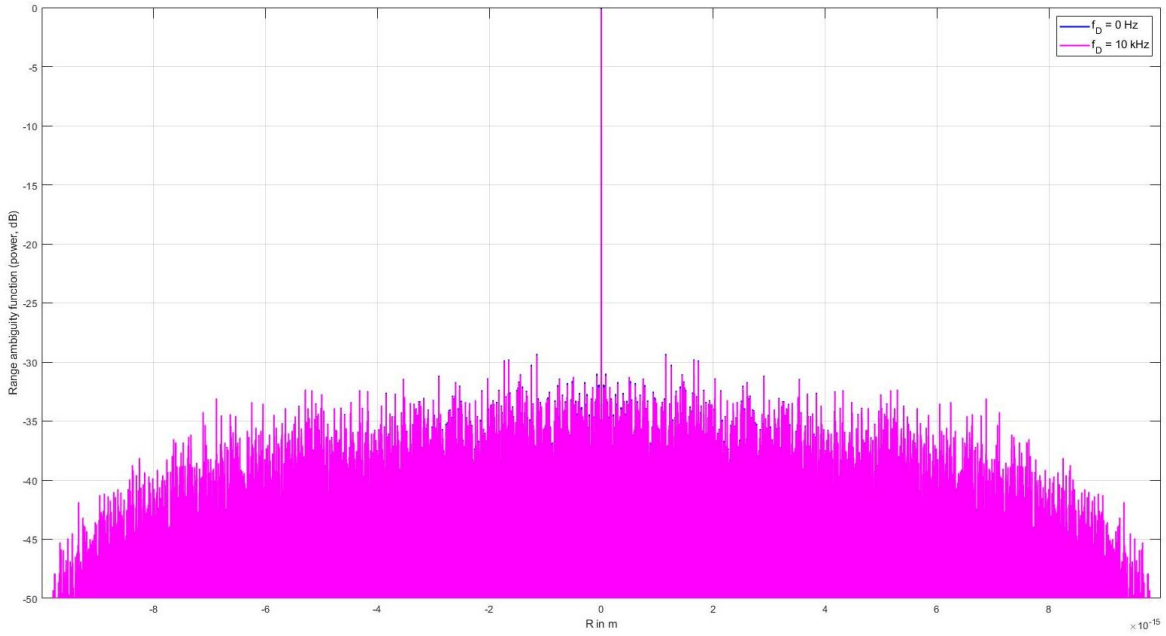


Abbildung 14Ambiguity 1GHz Rauschsignal Entfernung zu Dämpfung

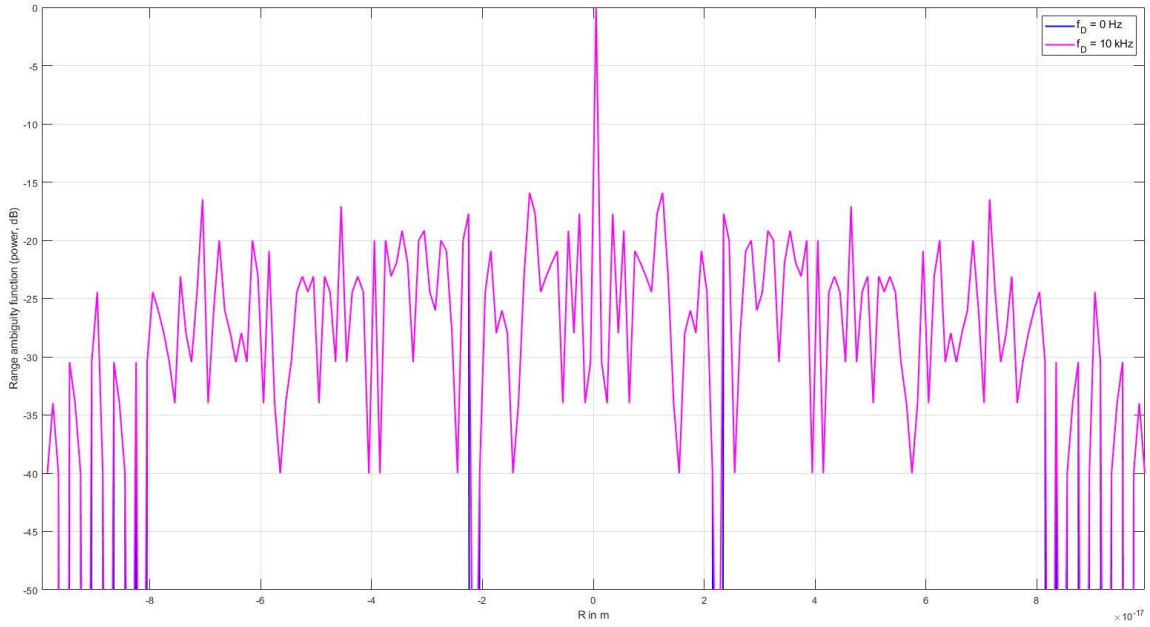


Abbildung 15 Ambiguity 1GHz Nutzsignal Entfernung zu Dämpfung

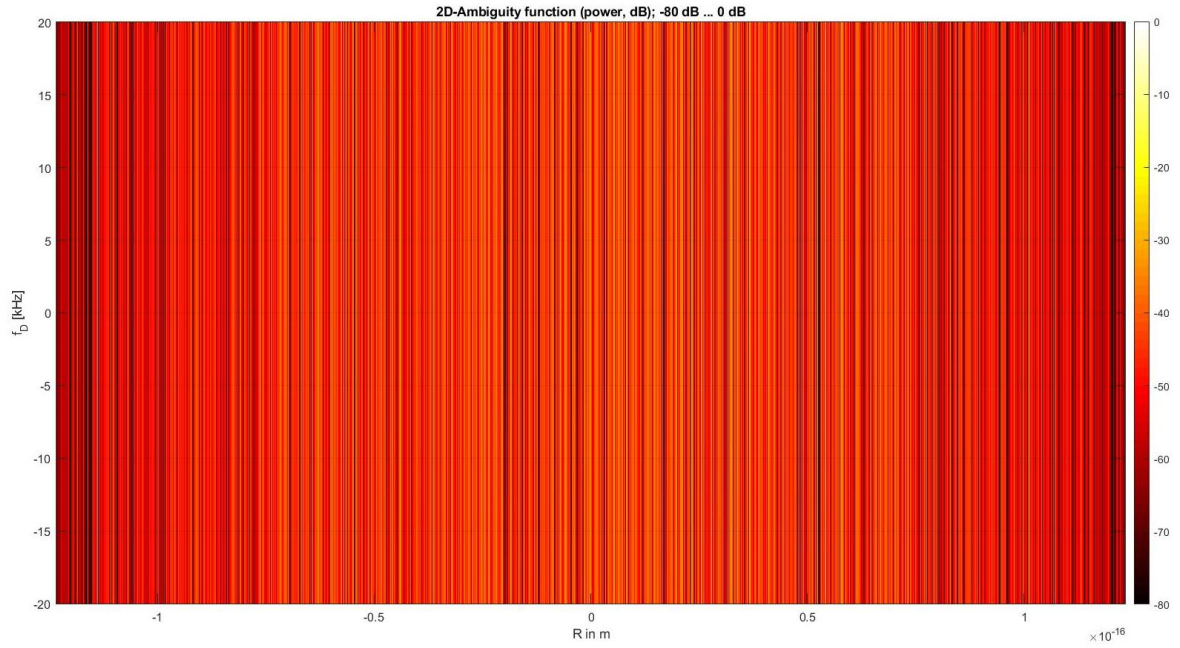


Abbildung 16 Ambiguity 1GHz Rauschsignal Entfernung Doppler

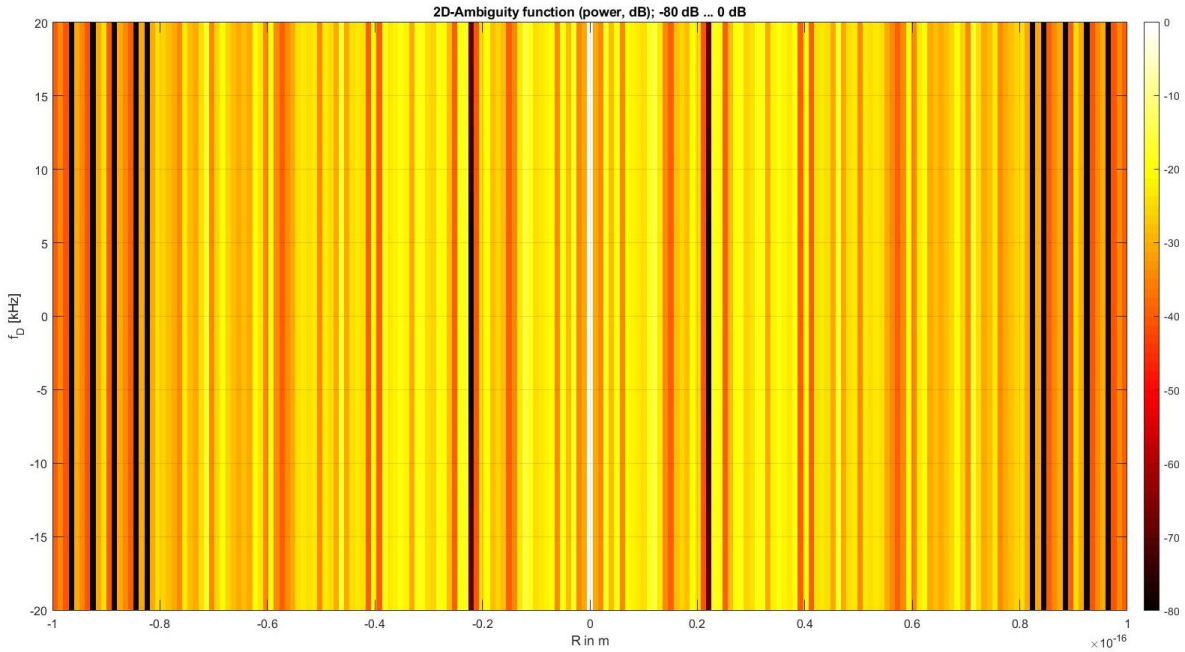


Abbildung 17 Ambiguity 1GHz Nutzsignal Entfernung Doppler

Ziel des Radarsignals ist trotz Entfernung und möglichen Dopplereffekts möglichst wenig Signaldämpfung zu haben, sodass eine Signalauswertung weiterhin möglich ist. Wie in den Grafiken zu erkennen ist, hat das generierte BPSK-modulierte Bitsignal grundsätzlich eine geringe Signaldämpfung. Ein Ideales Signal wäre in der oberen Darstellung eine konstante bei 0dB und eine reine weiße Fläche in der zweiten.
Daher lässt sich eine präzise Auswertung erst nach Anpassung der Funktion vornehmen.

5. Fazit

Ziel dieser Bachelorarbeit war es, eine Recherche rund um das Thema „RadComm“ anzustellen und anhand dieser Informationen ein Konzept zur gemeinsamen Verwendung eines Puls-Doppler-Radars für die Radardetektion und Kommunikation zu erstellen und per Simulation zu überprüfen.

Nach aktuellem Stand der Literatur stehen Puls-Radare nicht im Fokus der Entwicklung eines gemeinsamen Systems zur Detektion und Kommunikation. Dieser liegt eher im Bereich der Automobil-Industrie und deren Einsatzgebieten. Dennoch kann die Umsetzung einer solchen Lösung durchaus für den Einsatz in zum Beispiel Küstenradaren einen Mehrwert bieten.

Auf Basis der radartechnischen Anforderungen an die Modulationsart und die angenommenen Systemumstände lag der weitere Fokus dieser Arbeit auf der Suche und Auswahl einer passenden Kanalkodierung, die zur Betrachtung der Fountain Codes führte. Eine finale Bewertung für deren Einsatz in einem kombinierten System aus Radar und Kommunikation kann aufgrund fehlender spezifischer Untersuchungen nicht abgegeben werden. Hierzu müssten in Simulationen Faktoren wie die Antenneneigenschaften (Rotation, Richtwirkung, Nebenkeulen), ein der Realität entsprechender Kanal inklusive Umwelteinflüsse sowie unterschiedliche Modulationsarten für die Puls-Kompression als auch Puls-Integration genauer untersucht werden.

Die Performanz der Fountain Codes über Binary Erasure Channels generell wurde in Veröffentlichungen bereits mehrfach aufgezeigt und dokumentiert. Der Einsatz in AWGN-Kanälen ist für vereinzelte Anwendungsfälle ebenfalls untersucht worden, jedoch lässt sich bislang keine generelle Aussage zu ihrer Eignung treffen. Jedoch spricht allein ihre theoretische Performanz und ihre Eigenschaften, insbesondere die Ratenlosigkeit in Form von theoretisch unbegrenzten kodierten Bit pro zu übertragender Information, für ihre Verwendung in störanfälligen Übertragungskanälen und bei weiten Übertragungsstrecken mit entsprechenden Dämpfungen. Die Ergebnisse dieser Arbeit können trotz der Tatsache, dass viele Aspekte dieser Art der Systemzusammenlegung vernachlässigt werden mussten, für eine erste Einschätzung genutzt werden.

Grundsätzlich gilt, dass aufgrund der kurzen Impulsdauer des Sendesignals nur wenig Daten pro Impuls übertragen werden können. Kombiniert mit der redundanten Übermittlung und der Antennenumdrehung wird in vielen Anwendungsfällen die letztendliche „Datenrate“ nur für

geringe Informationsmengen ausreichen. Ob diese Performanz den erforderlichen technischen Entwicklungsaufwand rechtfertigt, müssen weitere Simulationen zeigen.

In dieser Arbeit wird deutlich, dass die Realisierung eines Fountain Code-basierten Kodierers und Dekodierers für Anwender außerhalb des Fachbereichs „Codierungstheorie“ schwer und nur unter Eingehen von Kompromissen umsetzbar ist. Dies hat das verwendete Simulationsmodell stark beschränkt und die Aussagekraft der Ergebnisse begrenzt. Daher wäre in weiteren Untersuchungen zu erwägen, bereits einsatzfähige Varianten eines kommerziellen Fountain Code, wie den RaptorQ Codes der Qualcomm Inc., einzusetzen, um die Aussagekraft zu erhöhen. Letztendlich wurde in dieser Arbeit die Umsetzung eines Kodierers und Dekodierers auf Basis des ursprünglichen Luby-Transform Codes realisiert.

A Verwendete Abkürzungen + Begriffserklärungen

ADC: Analog-to-Digital Converter, deutsch: Analog-Digital-Wandler. Sie tasten das Eingangssignal in bestimmten Intervallen Ts= 1/ fs ab und setzen die Spannung in einen digitalen Wert um.

Ambiguity: deutsch: Mehrdeutigkeit. Der „Ambiguity-Test“ dient der Prüfung der Radarkompatibilität eines Signals anhand von Dämpfungswerten in Abhängigkeit der auftretenden Doppler-Verschiebung.

ASK: Amplitude-Shift Keying, deutsch: Amplitudenumtastung, digitale Modulationsart

AWGN: Additive White Gaussian Noise, deutsch: additives weißes Gaußsches Rauschen, ein Kanalmodell in der drahtlosen Kommunikation

Azimut: aus dem Arabischen, ursprünglich aus der Astronomie. In der Antennentechnik auch als Horizontalwinkel bezeichnet. Horizontale Ausrichtung einer Antenne bzw. Winkel des detektierten Objektes im Vergleich zu geografisch Nord.

Barker Code: Binärcode mit minimaler Autokorrelation, wird im Bereich der Synchronisation und Radartechnik angewendet. 1953 von R. H. Barker entwickelt.

Belief Propagation: Dekodierung mittels Unterdrückung des Austausches unsicherer Informationen

Binary Erasure Channel, BEC: deutsch: binärer Auslöschungskanal, in der Informationstheorie erstmals von Peter Elias verwendet. Bezeichnet einen gedächtnisfreien Kanal, der Bit korrekt oder gar nicht überträgt. Gibt als Ausgang entsprechend das Bit oder ein E für „Error“ aus.

Bit: englisch binary digit, „binäre Ziffer“ oder auch Binärziffer. Wird in der Informatik, der Informationstechnik, der Nachrichtentechnik sowie verwandten Fachgebieten als Maßeinheit für die digitale Datenmenge verwendet.

bit: Pseudoeinheit des Informationsgehalts, je nach Sprache und Literatur auch Bit, wobei es zu Verwechslungen mit der obigen Definition kommen kann.

BPSK: Binary Phase Shift Keying, deutsch: Zweiphasenumtastung, Phasenumtastung

Clutter: deutsch: Unordnung. In der Radartechnik Begriff für ungewollt detektierte Objekte

Clutter-Doppler Filter: Der Filter vergleicht aus mindestens zwei Impulsperioden die Phasenlagen der Echosignale miteinander. Es erkennt sich langsam bewegende Objekte (z.B. Regenwolken).

Clutter-Offset Generator: Bauteil eines Puls-Radar-Senders. Ist eine Vorbereitung auf die Echo-Filterung. Verschiebt die Frequenz des Radars minimal.

CW-Radar: Continuous Wave Radar, deutsch: Dauerstrichradar. Radarsystem, bei dem der Sender während der Dauer des Messvorganges ununterbrochen arbeitet.

DAB: Digital Audio Broadcasting, deutsch: digitales Funk-Radio

DVB-T: Digital Video Broadcast-Terrestrial, deutsch: Digitale Videoübertragung terrestrisch, Antennenfernsehen.

Elevation: lateinisch: elevare – erheben. Höhenwinkel oder Vertikalwinkel, vertikaler Winkel zwischen Horizont und Antennenrichtung bzw. detektiertem Objekt**.**

FMCW-Radar: Frequency-Modulated Continuous Wave Radar, deutsch: frequenzmoduliertes Dauerstrichradar. Sendet und empfängt gleichzeitig.

Fountain Codes: gehört zur Klasse der ratenlosen Codes. U.a von Michael Luby entwickelt.

FSK: Frequency Shift Keying, deutsch: Frequenzumtastung,

Greatest Of-Filter: Filterschaltung, die aus einer Auswahl an Eingangssignalen jenes auswählt, das die stärkste Signalamplitude aufweist.

Ideal Soliton Distribution: Eine Form einer diskreten Wahrscheinlichkeitsverteilung.

Kartesisches Koordinatensystem: orthogonales Koordinatensystem. Nach dem latinisierten Namen Cartesius des französischen Mathematikers René Descartes benannt. Im zwei- und dreidimensionalen Raum das am häufigsten verwendete Koordinatensystem (x,y,z).

LDPC-Codes: Low-Density-Parity-Check-Codes, auch Gallager-Codes. Lineare Blockcodes zur Fehlerkorrektur. Entwickelt 1962 von Robert Gray Gallager.

LT-Codes: Luby Transform Codes, deutsch: Luby-Transformationscodes, veröffentlicht 1998 von Michael Luby.

Matched-Filter: Früher auch Optimalfilter. Optimiert das Signal-Rausch-Verhältnis.

MATLAB: MATrix LABoratory, kommerzielle Software des US-amerikanischen Unternehmens The MathWorks zur Lösung mathematischer Probleme und grafischen Darstellung der Ergebnisse.

MIMO: Multiple-Input Multiple-Output. In der Antennentechnik eingesetztes Verfahren zum gleichzeitigen Verwenden mehrerer Empfangs- und Sendeantennen.

MIMO -Radar: Multiple-Input Multiple-Output Radar. Verwendet die MIMO-Technik.

OFDM: Orthogonal Frequency-Division Multiplexing, deutsch: orthogonales Frequenzmultiplexverfahren. Verwendet mehrere orthogonale Träger zur digitalen Datenübertragung.

Online Fountain Code: Abwandlung des Raptor Code speziell für Online Dienste.

Over the Horizon Radar: deutsch: Überhorizontradar. Kann Radarechos ohne optischen Sichtkontakt über die Erdkrümmung hinaus erfassen. Die verwendeten Frequenzen liegen weit unterhalb der üblichen Radarfrequenzen.

Polar-Koordinatensystem: auch Kreiskoordinatensystem. Zweidimensionales Koordinatensystem mit Aufteilung in Winkel.

PRF: Pulse Repetition Frequency, deutsch: Impulsfolgefrequenz. Anzahl der gesendeten Impulse bezogen auf eine Sekunde. Standardmäßig in Hz angegeben.

PRI: Pulse Repetition Interval, deutsch: Impulsfolgeintervall. Zeitspanne eines Impulses.

PRT: Pulse Repetition Time, deutsch: Impulsfolgeperiode. Entspricht PRI.

PSK: Phase-Shift Keying, deutsch: Phasenumtastung. Digitales Modulationsverfahren in der Nachrichtentechnik.

Radar: Je nach Bezug „Radio Detection and Ranging“, „Radio Aircraft Detection and Ranging“, „Radio Direction and Ranging“, frei übersetzt: funkgestützte Ortung.

RadComm: Radar Communications, deutsch: radargestützte Kommunikation. Konzept zur Zusammenlegung der sonst getrennten Radarsysteme und funkbasierter Kommunikation.

Raptor Codes: Rapid-Tornado Codes. Unterart der Fountain Codes. Weiterentwicklung des Tornado Codes. Beispiele: R10 Code, RaptorQ, OpenRQ.

Reed-Solomon Codes: kurz RS-Codes, fehlerkorrigierende Codes. Um 1960 von Irving S. Reed und Gustave Solomon am MIT Lincoln Laboratory entwickelt.

Robust Soliton Distribution: eine Form einer diskreten Wahrscheinlichkeitsverteilung

SIMULINK: Software des Herstellers The MathWorks zur Modellierung von Systemen, Zusatzprodukt zu MATLAB, benötigt MATLAB zum Ausführen

SNR: Signal-to-Noise Ratio, deutsch: Signal-Rausch-Verhältnis, Störabstand /Signal-Rauschabstand, Maß für die technische Qualität eines Nutzsignals, das von einem Rauschsignal überlagert ist.

Treffer: Ein Treffer entspricht dem Auftreffen und der Reflektion eines elektromagnetischen Radarimpulses auf ein Objekt.

Trefferzahl: Bewertungskriterium für die Arbeit von Radargeräten. Sie gibt an, wie viele Echosignale pro Ziel während jeder Antennendrehung oder -schwenkung erzeugt werden.

X-Band: Bezeichnung für ein Frequenzband im Mikrowellenfunkbereich des elektromagnetischen Spektrums, wird häufig in modernen Radaren verwendet.

XOR Operator: eXclusive OR ‚deutsch: exklusives Oder', „entweder oder" Operator

Zero-Doppler-Filter: Meist Filter zweiter Ordnung, erkennt nur Festziele bzw. Ziele ohne Doppler-Frequenz.

B Anhang

Finaler Programmcode:

Simulation_V1_Matrix.m

```
% Simulation V1 - Luby-Transform-Kodierung: Matrix-basiert /
% Übertragungsmatrix
%
% Ansatz:
%
% Binary Erasure Channel!(BEC) Simuliert durch die Wahrscheinlichkeit
% p_dec_fail Gibt an, mit welcher Wahrscheinlichkeit ein Bit+Inex gelöscht
% wird.
%
% Das Sendesignal besteht aus einer Matrix mit den codierten Bits in der
% oberen Zeile und den dazugehörigen Indizes in der unteren Zeile.
%
% Es gibt eine festgelegte Frame-Länge (N). Diese ist dem Sender und
% Empfänger bekannt.
%
% Der Overhead ist ebenfalls Sender und Empfänger bekannt. (Anzahl
% zusätzlicher Bits im Frame im Vergleich zu der Menge der dafür genutzten
% Quellbits)
%
% Es wird eine zufalls-generierte Nachricht übertragen.
%
% N:            Länge der kodierten Nachricht
% K:            Eingangslänge zu kodierender Nachricht == Länge 1 Frames
% F:            Anzahl der Frames
% m_s:          Trefferzahl des Radars==Anzahl der Wiederholungen eines
Frames
% encTxBits:    SendeBits kodiert
% encRxBits:    empfangene Bits kodiert
% decBits:      dekodierte% Bits
%
% Dieser Code gibt als Auswertung ...
% -die Anzahl der komplett verlorenen Pakete (channel_failure)
% -die Anzahl an verlorenen Pakete, die aufgrund des wiederholten Sendens
dennoch dekodiert werden konnte
% -die insgesamt verlorenen Bits
% -die daraus entstehende Fehlerrate
% -eine Info, für den Fall, dass die generierte Matrix die Luby-Transform-
Dekodierregeln nicht erfüllt
% ...aus
%
% Da eine Weiterführung dieser Untersuchung inkl Simulation nach Abschluss
% der Bachelor-Arbeit geplant ist, können in dieser Simulation bereits
% Funktionen vorgemerkt sein, die noch nicht fertig implementiert sind.
%
% Michael Allinger, HAW Hamburg, 02.07.19
```

```
close all;
clc;


RSD_flag = 1;
%gibt an, ob die Matrix auf Basis der Robust-Soliton-Distribution erstellt
wird
% 0 = rein zufällige Matrix (online gefundener Code)    => in über 90% der
Fälle nicht geeignet!
% 1 = zufällige Matrix mit Beschränkung der maximalen Anzahl an 1-en =>
Vorstufe RAPTOR(online gefundener Code)
% 2 = eigene Matrix-Funktion komplett anhand RSD - aktuell nicht
implementiert

m_s=3;
% Trefferzahl des Radars => Anzahl Übertragungen der gleichen Information
%Beispielhaft auf drei gesetzt und auch nur dafür konzipiert!

N=100;
% Anzahl codierter bits N = info_length + overhead
% Da das Radar nur kurze Sub-Impulse sendet:
% Würde ein verschachtelter Barker-Code mit n=13 eingesetzt, wäre die
% Sequenzlänge insgesamt 169 Phasensprünge enthalten = 169Bit (BPSK) =>
N=169
% Am dichtesteb an einer 2 Potenz => 128

bec_error_p=1;
p_bec_error=bec_error_p/100;
%p_bec_error = 0.5;
%Wahrscheinlichkeit, dass der Kanal ein Bit (+dessen Index) verliert


o_head_perc=50;
%=> N=K+o_head
%wie viele Bit soll der Fountaincode zusätzlich generieren?
%Je höher, desto sicherer gelingt die Dekodierung und desto weniger
%Information kann übertragen werden
%Michael Luby empfiehlt N=K*beta => N sollte ein vielfaches von K sein
%=> Overhead >=50%

inf=[];
msg=[];
%zu kodierenden Bitvektor initalisieren

frame=[];
%initialisiere N-Bit Vektor zur Übertragung

decBits=[];
temp1_decBits=[];
temp2_decBits=[];
temp3_decBits=[];
%initialisiere N-Bit Vektor nach der Dekodierung

% p_dec_fail=0.01; - aktuell nicht genutzt
% Fehler-Akzeptanz für Empfangsseite (Prozent in Dezimalform)
% Ergo Prozentualer Anteil an Paketen, die verloren gehen können und vom
% inneren Fehlerkorrekturcode kompensiert werden können.
```

```
F = 100;
% Anzahl an N-Bit-Übertragungen
% entspricht inkl Redundanz bei 169 Bit/Burst circa:
% F=100:        16kB
% F=1000:       165kB
% F=10000:      1,6MB
% F=100000:     16,1MB



% Bestimmung der Anzahl an zu kodierenden Quellsymbolen K pro Übertragungs-
Impuls
% mit o_head >= 0

o_head=round(N*o_head_perc/100);
K = N - o_head;

%initalisiere verschiedene Fehlerzähler
temp_failure=0;
part_failure=0;
channel_failure=0;
count_matrix_failure=0;



%generiere Msg
info_length = K*F;
for (aa=1:info_length+1)
    bit=round(rand);
    inf=[inf bit];
end
msg=inf;

%info1=['Nachricht erzeugt']

%Erzeuge Matrix
if (RSD_flag == 0)
    matrix=GenerateGMatrix(K,N);
elseif (RSD_flag == 1)
    matrix=GenerateG(K,N);
elseif (RSD_flag == 1)
    [matrix,b]=genRSD_Matrix(K,N,p_dec_fail);
end

%info2=['Matrix erzeugt']

for (jj=1:F)    %für Anzahl an Durchläufen (ein File durchlaufen)


    %erzeuge Frame
    for (kk=1:K)
        new_bit=msg(1);
        msg(1)=[];
        frame=[frame new_bit];
    end

    %Kodierer
    encTxBits=encLT(frame,matrix);
```

```
    for (ll=1:m_s)

        %Bit-Erasure-Channel
        %hier werden per Zufallsgenerator anhand der Wahrscheinlichkeit
        %Einträge aus der encTxBits-Matrix entfernt
        %zur Matrix_überprüfung erst einmal ohne BEC
        encRxBits=simBEC(encTxBits,p_bec_error);
        %encRxBits=encTxBits;

        %Überprüft Dekodierbarkeit anhand der dem Kodierer bekannten Matrix
        Rx_length=size(encRxBits,2);
        known_K=size(matrix, 1);
        if (Rx_length<known_K)
            temp_failure=temp_failure+1;                %zählt verlorene
Pakete
        else
            %Dekodierer
            [temp_decBits,matrix_failure]=decLT(encRxBits,matrix);
%Speichert jeden Durchlauf in einer anderen Variable
            count_matrix_failure=count_matrix_failure+matrix_failure;
            if (ll==1)
                temp1_decBits=temp_decBits;
            elseif (ll==2)
                temp2_decBits=temp_decBits;
            elseif (ll==3)
                temp3_decBits=temp_decBits;
            end
        end
    end
        if (temp_failure>0)                          %zählt verlorene Pakete
            if (temp_failure<3)
                part_failure=part_failure+1;
            elseif (temp_failure==3)
                channel_failure=channel_failure+1;
            end
        end


  if  isequal(temp1_decBits,temp2_decBits,temp3_decBits)  %Vergleiche
Zwischenspeicher und wähle decodierte Bits mit höhster Übereinstimmung aus
        decBits=decBits+temp1_decBits;
  else
      if isequal(temp1_decBits,temp2_decBits)
            decBits= [decBits temp1_decBits];
      elseif isequal(temp1_decBits,temp3_decBits)
            decBits= [decBits temp1_decBits];
      elseif isequal(temp2_decBits,temp3_decBits)
            decBits= [decBits temp2_decBits];
      else
        if (temp1_decBits>temp2_decBits && temp1_decBits>temp3_decBits)
            decBits=[decBits temp1_decBits];
        elseif (temp2_decBits>temp1_decBits && temp2_decBits>temp3_decBits)
            decBits=[decBits temp2_decBits];
        else
            decBits=[decBits temp3_decBits];
        end

 end
 temp1_decBits=[];
 temp2_decBits=[];
```

```
 temp3_decBits=[];
 frame=[];
                                                %nach der Übertragung muss der
"Frame" gelöscht werden. Ein neuer Frame wird erzeugt!
end
comp_decBits=size(decBits,2);
comp_msg=size(inf,2);

%Gibt am Ende Werte aus

part_failure
channel_failure
count_matrix_failure
diff_msg_decBits=(comp_msg-comp_decBits)
```

genRSD.m

```
function [RSD_M,b]= genRSD(seq_length, error_delta)
% ?1 = 1/k,  ?i = 1/i(i?1) for all i = 2,...,k.
c = 1; % c > 0 als Konstante für Robust Soliton Distribution
k=seq_length;
d=error_delta;
theta=[];
theta_1 = 1/k;
theta=[theta theta_1];
Ti=[];
%R = c·ln(k/?)?k
R= c*log(k/d)*sqrt(k); % log = ln; log10 = log!
M=[];
RSD_M = zeros(k, 2);

%1. Ideal Soliton Distribution
for (i=2:k)
    newtheta = 1/(i*(i-1));
    theta=[theta newtheta];
end

%2. Robust Soliton Distribution
for (j=1:(k/R))
    newTi = R/(i*k);
    Ti=[Ti newTi];
end
Tinew=(R*log(R/d)/k);
Ti=[Ti Tinew];

for (ii=(k/R):(k))
    Ti=[Ti 0];
end

for (jj=1:k)
    M(jj)=theta(jj)+Ti(jj);
end
    b=sum(M);
for (iii=1:k)
    N(iii)=M(iii)/b;
    RSD_M(iii,:)=[N(iii)  iii];
end
%save(RSD,RSD_M); Überlegung RSD_M in seperater Datei zu speichern.
end
```

pick_d.m

```
function pickdegree = pick_d(RSD_M)

    %erst Spaltenvektor, durch " ' " Zeilenvektor
    degree=(RSD_M(:,2))';
    probability=(RSD_M(:,1))';
    n_a=size(degree);
    n=n_a(2);

    %folgenden Teil gefunden unter:
    %https://de.mathworks.com/matlabcentral/answers/23319-easy-question-
with-probability
    %Wähle anhand der Wahrscheinlichkeit für den jeweiligen Wert X einen
    %Wert X aus

    p = cumsum([0 probability(:).'/sum(probability(:))]);
    p(end) = 1e3*eps + p(end);
    % histcveraltet, aber neue Funktion führt keine identische Durchläufe
aus.
    [a a] = histc(rand,p);
    chosen_degree = degree(a);

pickdegree = chosen_degree;
```

GenerateG.m [24]

```
function G = GenerateG(K, N, ColTH, TryNum)
%function G = GenerateG(K, N, ColTH, TryNum)
%
%Decription:
% This function generates the binary matrix for Fountain codes
%
%Input:
% K - number of rows
% N - number of columns
% ColTH - lower bound for number of 1's per col (multiple of K/S)
% TryNum - number of tests to choose from
%
%Output:
% G - a K X N binary matrix
%
%Formed by: Roee Diamant, UBC, July 2012

if nargin < 3
    ColTH = 1.3; %bound for number of 1's in each column
end
if nargin < 4
    TryNum = 10;
end

TryG = {};
MeanSummerRow = zeros(1, TryNum);
MinSummerRow = zeros(1, TryNum);
for TryInd = 1: TryNum
    while(1)
        %-----------------
        %create pdf
        c = 0.2;
        delta = 0.3;
        S = c*log(K/delta)*sqrt(K);

        rho = zeros(1, K);
        thu = zeros(1, K);
        d = 1: K;

        rho(1) = 1/K;
        rho(2:K) = 1 ./ (d(2:end) .* (d(2:end)-1));

        loc = floor(K/S-1);
        thu(1: loc) = S/K ./ d(1: loc);
        thu(loc+1) = S/K*log(S/delta);

        Z = sum(rho + thu);
        mu = (rho + thu) / Z;
        %-----------------
        %threshold for maximum number of ones per column
        MaxColTH = K/S * ColTH;

        G = zeros(K, N);
        for n = 1: N
            while(1)
                %-----------------
```

```
                %draw a number
                Cmu = cumsum(mu)/sum(mu);
                u = rand(1,1);
                diff = abs(u - Cmu);
                [M, loc] = min(diff);
                Num = d(loc);
                %-----------------
                if Num <= MaxColTH
                    break;
                end
            end
            row = randperm(K);
            G(row(1: Num), n) = 1;
        end

        SummerRow = sum(G, 2);
        if min(SummerRow) > 0
            MeanSummerRow(TryInd) = mean(SummerRow);
            MinSummerRow(TryInd) = min(SummerRow);
            TryG{TryInd} = sparse(G); %#ok<AGROW>
            break;
        end
    end
end

%maximize weight per row and minimize weight per col
[V, loc] = max(MeanSummerRow);
G = TryG{loc};
G = full(G);
```

GenerateGMatrix.m [24]

```
function G = GenerateGMatrix(K, N)

%Decription:
% This function generates the binary matrix for Fountain codes
%
%Input:
% K - number of rows
% N - number of columns
%
%Output:
% G - a K X N binary matrix
%
%Formed by: Roee Diamant, UBC, Oct 2012

Sigma = ceil(2*log(K));
if Sigma/2 == floor(Sigma/2)
    Sigma = Sigma + 1;
end

W = ceil(2 * (sqrt(K)-1)*(Sigma-1)/(Sigma-2));

G = zeros(K, N);
for n = 1: N
    loc = round(rand(1,1)*K);
    PosVec = [loc+1: min(K, loc+W), 1: W-(K-loc)];
    pos = randperm(W);
    G(PosVec(pos(1: Sigma)), n) = 1;
end
```

genRSD_Matrix.m

```
function [G,b]= genRSD_Matrix(K,N,p_dec_fail)
%Nutzt genRSD,pick_d um Spalten der Matrix zu erzeugen
%

%Erzeuge Variablen
[rsd,b]=genRSD(N,p_dec_fail);
G=zeros(K,N);
column=[];
degree=0;

%Geht die zu generierende Matrix spaltenweise durch und generiert anhand
%der gewählten Ordnung Einsen und Nullen
    for n = 1: N
        degree=pick_d(rsd);
        create_for_column=[ones(1,degree) zeros(1,(K-degree))];
        for_column = randperm(create_for_column);
        column=for_column';
        G(:,N)=column;
    end
end
```

encLT.m

```
function [TxBits] = encLT(seq,matrix)

% Kodiere nach Matrix_Luby-Transform
% reiner Binär-Kodierer

%interne Variablen
uncoded_bits=seq;
code_matrix=matrix;

%Bestimmung von K,N
K = size(code_matrix, 1);
N = size(code_matrix, 2);

%Zur Performance-Steigerung, vorab Festlegung der Größe der Matrix
TxBits=zeros(2,N);

%eigentlicher Codiervorgang
% Modulo 2, da binär!
math_encoded=uncoded_bits*code_matrix;
encoded_symbols=mod(math_encoded,2);

%Erzeugung der Sendematrix mit encodierten Bits und ihrem jeweiligen Index
for (i=1:N)
    TxBits(1,i)=encoded_symbols(i);
    TxBits(2,i)=i;
end
end
```

decLT.m

```
function [decBits, matrix_failure] = decLT(encRxBits,Matrix)
%Binärer LT-Decoder
%Basiert auf  dem Code Von Roee Diamant, UBC, July 2012
%(siehe Datei-Ordner)
%Namen von Variablen verändert zum besseren Verständnis
%Einzelne Funktionen dokumentiert.


%Erzeuge aus Empfangsmatrix den Empfangsvektor!
temp_encRxBits=encRxBits(1,:);

%Bestimme anhand der ursprünglichen Matrix K
K = size(Matrix, 1);

%Initialisiere Variablen
decBits = -ones(K, 1);
add = zeros(1, K);
matrix_failure= 0;   %Info, ob der Dekodiervorgang aufgrund der Matrix
versagt hat. 0 = nein; 1 = ja;

%Bestimme anhand der eingegangen Empfangsmatrix das neue N
N = size(encRxBits, 2);


%erstelle und initialisiere temporäre Variable für die Matrix => angepasst
an eingegangener encRxBits-Matrix
temp_matrix =zeros(K,N);
%Nimmt den Index-Vektor der Empfangsmatrix und erstellt daran die temporäre
%Dekodiermatrix
chose_column=encRxBits(2,:);

for (i=1:N)
    cc=chose_column(i);                    % wähle den i. Index aus
    temp_matrix(:,i)=Matrix(:,cc);         % nimm die cc. Spalte der
ursprünglichen Matrix und nimm sie als i. Spalte
end


% nach dem ursprünglichen Luby-Transform-Decoder-Verfahren
% suche in der Matrix eine Spalte mit einer einzigen 1 => codediertes
% Symbol hat nur ein verknüpftes Quellensymbol!
% Wenn keines vorhanden => Fehler! Abbruch!

while(1)                                   %wiederhole ewig!
    loc = find(sum(temp_matrix,1) == 1);   %finde Spalten mit einzelner 1,
nimm deren Index
    if isempty(loc)                        %wenn keiner
        matrix_failure=1;                  %Dekodiervorgang schlägt fehl
        break;
    else

    for (ind = 1: length(loc))                      %für alle Spalten mit
nur einer 1, führe nacheinander aus...
        pos = find(temp_matrix(:, loc(ind)) == 1);  %gibt Stelle im
dekodierten Vektor an
```

```
        loc2 = find(temp_matrix(pos, :) == 1);      %findet alle Spalten,
die ebenfalls dieses Quellbit verwendet haben
        temp_matrix(pos, :) = 0;                    %setze die 1 dieser
Spalten auf 0, Bit ist gelöst
        add(pos) = 1;                               %erzeugt einen Vektor,
jeder decodierte Index wird zu 1, zum späteren Vergleich, ob alle Symbole
dekodiert wurden
        decBits(pos) = temp_encRxBits(loc(ind));    %setzt das decodierte
Bit in den endgültigen Dekodiert-Vektor (decBits)
        temp_encRxBits(loc2) = mod(temp_encRxBits(loc2)-decBits(pos),
2^(2));
    end
    end
    if (sum(add) == K)                              %Wenn Länge K
erreicht(ergo alle Quellsymbole decodiert sind), stoppe und gebe Bits aus!
        break;
    end

end
```

simBEC.m

```
function encRxBits = simBEC(encTxBits,p_bec_error)
%Diese Funktion simuliert eine reine Wahrscheinlichkeit ein
%Bit(+dazugehörigen Index) zu verlieren, anhand der einzugebenen
%Fehler-Wahrscheinlichkeit

%Erstelle temporäre Variable

temp_encTxBits=encTxBits;

%errechne Wahscheinlichkeit für X verlorene Bits
p=[];
q=(1-p_bec_error);
N=size(temp_encTxBits,2);
p(1)=q^N;                           %p(1) ist die Wahrscheinlichkeit, dass
kein Bit verloren geht
p(2)=(p_bec_error)*(q^(N-1))*16;      %p(2) ist die Wahrscheinlichkeit,
dass 1 Bit verloren geht//16 verschiedene Stellen!
for (i=2:(N-2))
    summe=0;
    a=1:(N-i+1);                    %Die Anzahl an Möglichkeiten entspricht
der Summe von 1 bis N-1 im ersten Druchlauf => +1!
    a=1./a;
    summe=sum(a);
    p_temp=((p_bec_error)^i)*(q^(N-i))*(summe);
    p=[p p_temp];
end
p(N)=((p_bec_error)^(N-1))*q*16;      %Vorletzte Wahrscheinlichkeit => 1
Bit leibt übrig
p(N+1) = (p_bec_error)^N;             %letze Wahrscheinlichkeit => kein Bit
bleibt übrig

sum_p=sum(p);
P_error=p/sum_p;                    %Normieren

%P_error                            %zur Überprüfung der Normierung und
%sum_p=sum(P_error)                 %Wahrscheinlichkeitsverteilung
P_error_M=zeros((N+1),2);
for (j=1:(N+1))
    P_error_M(j,1)=P_error(j);
    P_error_M(j,2)=j-1;
end

errors=(P_error_M(:,2))';
probability=(P_error_M(:,1))';
n_a=size(errors);
n=n_a(2);

%folgenden Teil gefunden unter:
%https://de.mathworks.com/matlabcentral/answers/23319-easy-question-with-
probability
%
%Wähle anhand seiner Wahrscheinlichkeit einen Wert X aus...
%=> gewählte Anzahl an Fehlern

choose_p = cumsum([0 probability(:).'/sum(probability(:))]);
choose_p(end) = 1e3*eps + choose_p(end);
```

```
% histcveraltet, aber neue Funktion führt keine identische Durchläufe aus.
[a a] = histc(rand,choose_p);
chosen_errors = errors(a);

gen_delete_array=[ones(1,chosen_errors) zeros(1,N-chosen_errors)];
rand_delete_array=randperm(N);
[~,sort_rand_delete_array]=sort(rand_delete_array);
delete_array=gen_delete_array(sort_rand_delete_array);


%Ab hier wieder eigener Code

for (ii=1:chosen_errors)
    z=find(delete_array,1);
    temp_encTxBits(:,z)=[];
    delete_array(z)=[];
end

encRxBits=temp_encTxBits;



end
```

```
% Simulation V2 - Luby-Transform-Kodierung: Matrix-basiert /
% Übertragungsmatrix
%
% Ansatz:
%
% Binary Erasure Channel!(BEC) Simuliert durch die Wahrscheinlichkeit
% p_dec_fail Gibt an, mit welcher Wahrscheinlichkeit ein Bit+Inex gelöscht
% wird.
%
% Das Sendesignal besteht aus einer Matrix mit den codierten Bits in der
% oberen Zeile und den dazugehörigen Indizes in der unteren Zeile.
%
% Es gibt eine festgelegte Frame-Länge (N). Diese ist dem Sender und
% Empfänger bekannt.
%
% Der Overhead ist ebenfalls Sender und Empfänger bekannt. (Anzahl
% zusätzlicher Bits im Frame im Vergleich zu der Menge der dafür genutzten
% Quellbits)
%
% Es wird eine zufalls-generierte Nachricht übertragen.
%
% N:            Länge der kodierten Nachricht
% K:            Eingangslänge zu kodierender Nachricht == Länge 1 Frames
% F:            Anzahl der Frames
% m_s:          Trefferzahl des Radars==Anzahl der Wiederholungen eines
Frames
% encTxBits:    SendeBits kodiert
% encRxBits:    empfangene Bits kodiert
% decBits:      dekodierte% Bits
%
% Dieser Code gibt als Auswertung ...
% -die Anzahl der komplett verlorenen Pakete (channel_failure)
% -die Anzahl an verlorenen Pakete, die aufgrund des wiederholten Sendens
dennoch dekodiert werden konnte
% -die insgesamt verlorenen Bits
% -die daraus entstehende Fehlerrate
% -eine Info, für den Fall, dass die generierte Matrix die Luby-Transform-
Dekodierregeln nicht erfüllt
% ...aus
%
% Da eine Weiterführung dieser Untersuchung inkl Simulation nach Abschluss
% der Bachelor-Arbeit geplant ist, können in dieser Simulation bereits
% Funktionen vorgemerkt sein, die noch nicht fertig implementiert sind.
%
% Michael Allinger, HAW Hamburg, 02.07.19

close all;
clc;

part_loss=[];
perc_packet_loss=[];
count_matrix_errors=[];
diff_msg_dec=[];


%for (bec_error_p=1:50)

bec_error_p=1

    RSD_flag = 1;
```

```
    %gibt an, ob die Matrix auf Basis der Robust-Soliton-Distribution
erstellt wird
    % 0 = rein zufällige Matrix (online gefundener Code)    => in über 90%
der Fälle nicht geeignet!
    % 1 = zufällige Matrix mit Beschränkung der maximalen Anzahl an 1-en =>
Vorstufe RAPTOR(online gefundener Code)
    % 2 = eigene Matrix-Funktion komplett anhand RSD - aktuell nicht
implementiert

    m_s=3;
    % Trefferzahl des Radars => Anzahl Übertragungen der gleichen
Information
    %Beispielhaft auf drei gesetzt und auch nur dafür konzipiert!

    N=100;
    % Anzahl codierter bits N = info_length + overhead
    % Da das Radar nur kurze Sub-Impulse sendet:
    % Würde ein verschachtelter Barker-Code mit n=13 eingesetzt, wäre die
    % Sequenzlänge insgesamt 169 Phasensprünge enthalten = 169Bit (BPSK) =>
N=169
    % Am dichtesteb an einer 2 Potenz => 128

    p_bec_error=bec_error_p/100;
    %p_bec_error = 0.5;
    %Wahrscheinlichkeit, dass der Kanal ein Bit (+dessen Index) verliert

    part_loss_i=[];
    perc_packet_loss_i=[];
    count_matrix_errors_i=[];
    diff_msg_dec_i=[];

    %o_head_perc;
    %=> N=K+o_head
    %wie viele Bit soll der Fountaincode zusätzlich generieren?
    %Je höher, desto sicherer gelingt die Dekodierung und desto weniger
    %Information kann übertragen werden
    %Michael Luby empfiehlt N=K*beta => N sollte ein vielfaches von K sein
    %=> Overhead >=50%

    max_o_head_perc=90;

    for (o_head_perc=1:max_o_head_perc)

        iter=10;

        summer_dif_msgdec=int64(0);
        summer_part_failure=int64(0);
        summer_channel_failure=int64(0);
        summer_matrix_failure=int64(0);

        for (repeat=1:iter)

            inf=[];
            msg=[];
            %zu kodierenden Bitvektor initalisieren

            frame=[];
            %initialisiere N-Bit Vektor zur Übertragung
```

```
            decBits=[];
            temp1_decBits=[];
            temp2_decBits=[];
            temp3_decBits=[];
            %initialisiere N-Bit Vektor nach der Dekodierung

            % p_dec_fail=0.01; - aktuell nicht genutzt
            % Fehler-Akzeptanz für Empfangsseite (Prozent in Dezimalform)
            % Ergo Prozentualer Anteil an Paketen, die verloren gehen
können und vom
            % inneren Fehlerkorrekturcode kompensiert werden können.


            F = 100;
            % Anzahl an N-Bit-Übertragungen
            % entspricht inkl Redundanz bei 169 Bit/Burst circa:
            % F=100:        16kB
            % F=1000:       165kB
            % F=10000:      1,6MB
            % F=100000:     16,1MB

            % Bestimmung der Anzahl an zu kodierenden Quellsymbolen K pro
Übertragungs-Impuls
            % mit o_head >= 0

            o_head=round(N*o_head_perc/100);
            K = N - o_head;

            %initalisiere verschiedene Fehlerzähler
            temp_failure=0;
            part_failure=0;
            channel_failure=0;
            count_matrix_failure=0;

            %for(iter=1:10000)

            %generiere Msg
            info_length = K*F;
            for (aa=1:info_length+1)
                bit=round(rand);
                inf=[inf bit];
            end
            msg=inf;

            %info1=['Nachricht erzeugt']

            %Erzeuge Matrix
            if (RSD_flag == 0)
                matrix=GenerateGMatrix(K,N);
            elseif (RSD_flag == 1)
                matrix=GenerateG(K,N);
            elseif (RSD_flag == 1)
                matrix=genRSD_Matrix(K,N,p_dec_fail);
            end

            %info2=['Matrix erzeugt']

            for (jj=1:F)    %für Anzahl an Durchläufen (ein File
durchlaufen)
```

```
                %erzeuge Frame
                for (kk=1:K)
                    new_bit=msg(1);
                    msg(1)=[];
                    frame=[frame new_bit];
                end

                %Kodierer
                encTxBits=encLT(frame,matrix);

                for (ll=1:m_s)

                    %Bit-Erasure-Channel
                    %hier werden per Zufallsgenerator anhand der
Wahrscheinlichkeit
                    %Einträge aus der encTxBits-Matrix entfernt
                    %zur Matrix_überprüfung erst einmal ohne BEC
                    encRxBits=simBEC(encTxBits,p_bec_error);
                    %encRxBits=encTxBits;

                    %Überprüft Dekodierbarkeit anhand der dem Kodierer
bekannten Matrix
                    Rx_length=size(encRxBits,2);
                    known_K=size(matrix, 1);
                    if (Rx_length<known_K)
                        temp_failure=temp_failure+1;                %zählt
verlorene Pakete
                    else
                        %Dekodierer

[temp_decBits,matrix_failure]=decLT(encRxBits,matrix);  %Speichert jeden
Durchlauf in einer anderen Variable

count_matrix_failure=count_matrix_failure+matrix_failure;
                        if (ll==1)
                            temp1_decBits=temp_decBits;
                        elseif (ll==2)
                            temp2_decBits=temp_decBits;
                        elseif (ll==3)
                            temp3_decBits=temp_decBits;
                        end
                    end
                end
                    if (temp_failure>0)%zählt komplett verlorene Pakete
                        if (temp_failure<3)
                            part_failure=part_failure+1;
                        elseif (temp_failure==3)
                            channel_failure=channel_failure+1;
                        end
                    end


                if  isequal(temp1_decBits,temp2_decBits,temp3_decBits)
%Vergleiche Zwischenspeicher und wähle decodierte Bits mit höhster
Übereinstimmung aus
                    decBits=decBits+temp1_decBits;
                else
                    if isequal(temp1_decBits,temp2_decBits)
                        decBits=decBits+temp1_decBits;
```

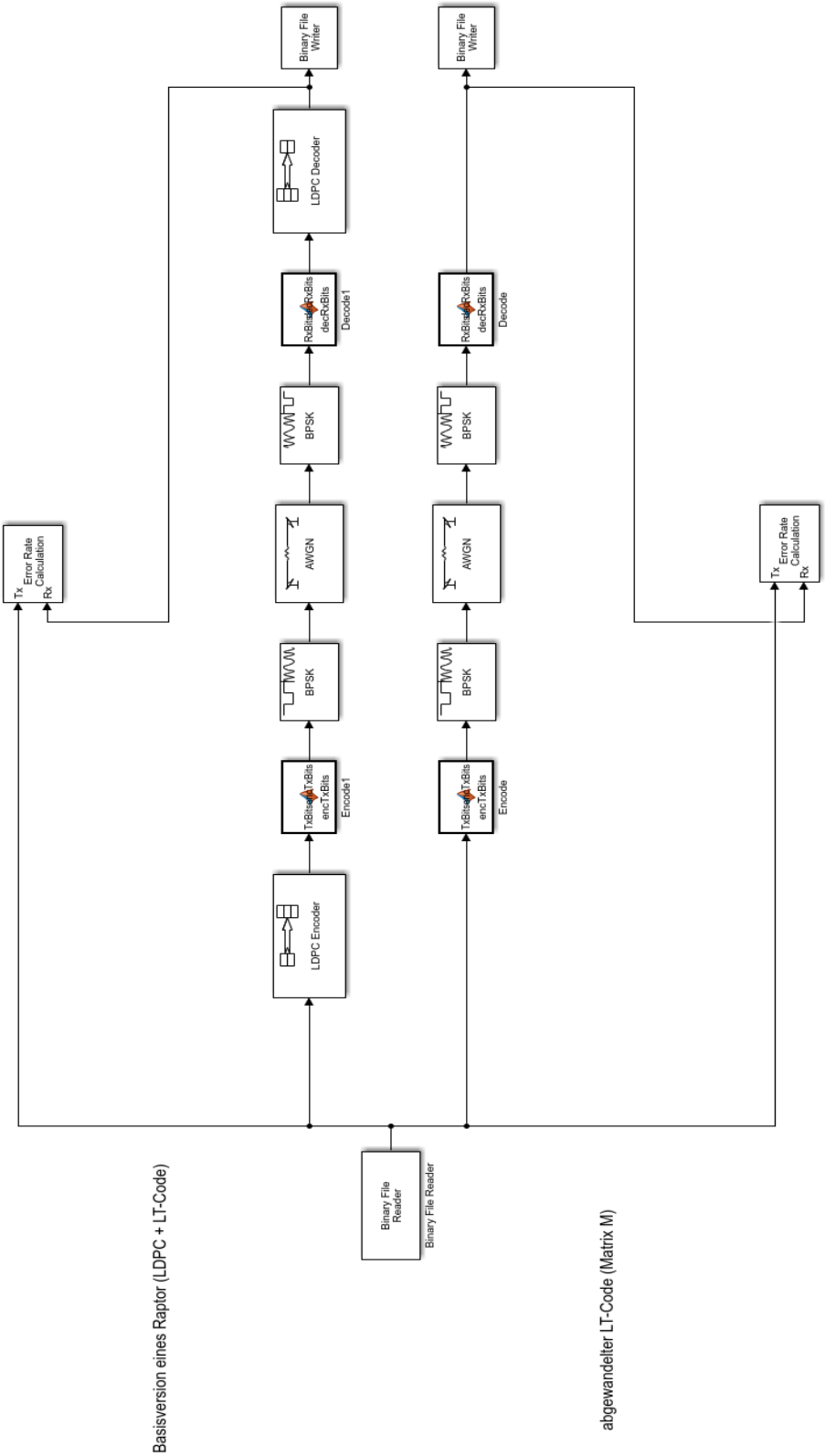
```
                    elseif isequal(temp1_decBits,temp3_decBits)
                        decBits=decBits+temp1_decBits;
                    elseif isequal(temp2_decBits,temp3_decBits)
                        decBits=decBits+temp2_decBits;
                    end
                end
                temp1_decBits=[];
                temp2_decBits=[];
                temp3_decBits=[];
                frame=[];
                                                        %nach der Übertragung
muss der "Frame" gelöscht werden. Ein neuer Frame wird erzeugt!
            end
            comp_decBits=size(decBits,2);
            comp_msg=size(inf,2);
            diff_msg_decBits=comp_msg-comp_decBits;
            summer_part_failure=summer_part_failure+part_failure;
            summer_channel_failure=summer_channel_failure+channel_failure;

summer_matrix_failure=summer_matrix_failure+count_matrix_failure;
            summer_dif_msgdec=summer_dif_msgdec+diff_msg_decBits;
        end

        avg_part_loss_i=summer_channel_failure/iter;

avg_perc_packet_loss_i=(summer_part_failure+summer_channel_failure)/iter;
        avg_count_matrix_errors_i=summer_matrix_failure/iter;
        avg_diff_msg_dec_i=summer_dif_msgdec/iter;

        part_loss_i=avg_part_loss_i;
        perc_packet_loss_i=avg_perc_packet_loss_i;
        count_matrix_errors_i=avg_count_matrix_errors_i;
        diff_msg_dec_i=avg_diff_msg_dec_i;

        save('part_loss-BEC_P1.mat', 'part_loss_i' );
        save('perc_packet_loss-BEC_P1.mat', 'perc_packet_loss_i' );
        save('count_matrix_errors-BEC_P1.mat', 'count_matrix_errors_i' );
        save('diff_msg_dec-BEC_P1.mat', 'diff_msg_dec_i' );

    end
%
%     part_loss=cat(bec_error_p,part_loss,part_loss_i);
%
perc_packet_loss=cat(bec_error_p,perc_packet_loss,perc_packet_loss_i);
%
count_matrix_errors=cat(bec_error_p,count_matrix_errors,count_matrix_errors
_i);
%     diff_msg_dec=cat(bec_error_p,diff_msg_dec,diff_msg_dec_i);
%
% end
%
% save('part_loss.mat', 'part_loss' );
% save('perc_packet_loss.mat', 'perc_packet_loss' );
% save('count_matrix_errors.mat', 'count_matrix_errors' );
% save('diff_msg_dec.mat', 'diff_msg_dec' );
```

Sim_AF_test.m

```
% 2D-ambiguity function (AF) for analyzing radar signal performance
%
% Mietzner, 09.05.2019
%
% ************************************************************************
close all
j = sqrt(-1);

% Parameters for 2D-AF
fd_max = 20000;             % Maximum Doppler shift in Hz to be analyzed
                            % (20 kHz is 150 m/s in K-band)
                            % @9GHz => 1200m/s für Schiffsverkehr Hafen
                            % @1GHz => ~10km/s!!! Flugverkehr
delta_fd = 100;             % Step size for Doppler analysis in Hz
%fs_adc = 200e6;            % Assumed sampling rate of ADC in Hz
fs_adc = 9e9;               % Angenommene Frequenz /9GHz Hafenradar / 1GHz
Flugradar

% Tx signal
Test_sequenz_ambiguity=[1,1,0,1,0,0,0,1,1,1,0,1,1,1,1,1,1,0,0,1,0,0,1,1,0,0
,0,1,1,0,0,1,1,0,0,1,0,0,1,0,1,0,1,0,0,1,1,1,1,1,0,0,1,0,1,1,1,1,1,0,0,1,
0,0,1,0,0,1,0,1,0,1,1,1,1,1,1,0,0,0,1,0,0,0,0,0,0,0,1,0,0,1,0,1,1,1,0,1,1];
dt = 1/fs_adc;              % Time base is s (sample duration)
disp('Sendesignal einfügen')

data=Test_sequenz_ambiguity';           %nimmt eine kodierte Sendesequenz
und moduliert sie (BPSK)
bpskModulator = comm.BPSKModulator;
modData = bpskModulator(data);

s_tx1=modData';
%s_tx1 = randn(1,10000);     % Beispiel: Rauschsignal



% Ambiguity function in range & Doppler
%dR = dt/2*3e8;             % Range sample size in m
dR = dt/fs_adc;             % Range an Frequenz angepasst!!!
fd = [-fd_max:delta_fd:+fd_max];
dim_dp = length(fd);

AF_rg_dp1 = [];                         % Ambiguity function for Tx-signal
s_tx1
for jj = 1:dim_dp
    dp_phase = exp(j*2*pi*fd(jj)*[0:length(s_tx1)-1]*dt);
    AF_cur = conv(s_tx1.*dp_phase,fliplr(conj(s_tx1)));
    AF_rg_dp1 = [AF_rg_dp1 ; AF_cur];
end
% Normalization to maximum = 1 (0 dB)
AF_rg_dp_pow1 = abs(AF_rg_dp1).^2;
AF_rg_dp_pow1 = AF_rg_dp_pow1/max(max(AF_rg_dp_pow1));
rg_shift_axis = [-size(AF_rg_dp_pow1,2)/2 : size(AF_rg_dp_pow1,2)/2]*dR;

% Figure
figure
AF_dB1 = 10*log10(AF_rg_dp_pow1);
```

```
AF_dB1(AF_dB1 < -80 ) = -80;
h = imagesc(rg_shift_axis,fd/1000,AF_dB1);
hold on
grid on
h=xlabel('R in m');
set(h,'FontSize',12)
h=ylabel('f_D [kHz]');
set(h,'FontSize',12)
h=title('2D-Ambiguity function (power, dB); -80 dB ... 0 dB');
set(h,'FontSize',12)
set(gca,'FontSize',12)
axis tight
%axis([-1000 +1000 -fd_max/1000 fd_max/1000])
axis xy
colormap('Hot')
colorbar

figure
idx = find(fd==0);
idx2 = find(fd==fd_max/2);
h = plot(rg_shift_axis(2:end),10*log10(AF_rg_dp_pow1(idx,:)),'b');
set(h,'Linewidth',1.5)
hold on
h = plot(rg_shift_axis(2:end),10*log10(AF_rg_dp_pow1(idx2,:)),'m');
set(h,'Linewidth',1.5)
grid on
h = legend('f_D = 0 Hz',['f_D = ' num2str(fd_max/2000) ' kHz']);
set(h,'FontSize',11)
axis([rg_shift_axis(1) rg_shift_axis(end) -50 0])
xlabel('R in m')
ylabel('Range ambiguity function (power, dB)')
```

Ursprünglich geplante SIMULINK-Schaltung



Erster Entwurf des LT-Kodierers:

Hauptfunktion/Test-File

```
%Test LT-Code (Fountain)
clear;
clc;

%Fehler-Akzeptanz für Empfangsseite (Prozent in Dezimalform)
error_delta=0.05

%generiere Nachricht (bit-vector)
msg= ;
msg_lenght=size(msg,2);

%Distribution (Wahrscheinlichkeitsverteilung)
[RSD_M b]=genRSD(msg_length,error_delta);

%Encoder
TxLT_enc=FountainEnc(msg,RSD_M,b);

%BPSK Modulation
TxLT_BPSK=LTmodBPSK(TxLT_enc);

%AWGN Simulation


%BPSK Demodulation
RxBit=LTdemodBPSK(RxLT_BPSK);

%Decoder


%Fehlerrate
```

Im ersten Anlauf der Simulation basierte der Kodierer auf dem ursprünglichen LT-Code, vorgestellt von Michael Luby.

Die Funktion „fountainEnc“ zur Erstellung der codierten Bit wird gespeist durch die Quellsymbole, der Wahrscheinlichkeitsverteilung als Matrix und dem Skalierungsfaktor [beta].

```
function TxLT_enc = fountainEnc(ScrBit,RSD_M, b)
seq_length=size(ScrBit,2);
%nach Luby für LT Code empfohlene Anzahl an CodeBit für Wiederherstellung
to_gen_CodeBit = seq_length*b;
%initialisiere leeren Vektor
LT_enc_Bit = [];

for (1:to_gen_CodeBit)
    temp_degree=pick_d(RSD_M);  %zufällig "degree" bestimmen
    chosen_ScrBit=ScrBit(randperm(numel(ScrBit),temp_degree));
    xor_size=size(chosen_ScrBit,2);
    %verknüpft die zufällig gewählten Bit mit der XOR Operation
    for (i=1:xor_size)
        xorBit=[];
        xorBit=bitxor(xorBit,chosen_ScrBit(i));
    end
   %das encodierte Symbol wird in den codierten Bittrom gepackt
    LT_enc_Bit=[LT_enc_Bit xorBit];
end

TxLT_enc=LT_enc_Bit
```

Diese Funktion geht die in Kapitel 3.2.2 „Luby-Transform Code“ beschriebenen Schritte für jedes Codebit durch.

Zuerst wird eine Ordnung gewählt, dann eine zufällige Anzahl an Quellsymbolen entsprechend der Ordnung gewählt und diese XOR-verknüpft. Dann wird das codierte Bit in den Sende-Vektor geschrieben.

C Abbildungsverzeichnis

D Quellenverzeichnis

Fachliteratur (Bücher)

[1] Albrecht Ludloff: *Praxiswissen Radar und Radarsignalverarbeitung*, 4. ergänzte und erweiterte Auflage. Wiesbaden: Vieweg und Teubner, 2009

[2] Bassem R. Mahafza und Atef Z. Elsherbeni: MATLAB *Simulations for Radar Systems Design.* USA: CHAPMA & HALL/CRC Press LLC, 2004

[3] K. Deergha Rao: *Channel Coding Techniques for Wireless Communications*. Springer, 2015

[4] Merrill Skolnik: *Radar Handbook*, Third Edition. McGraw-Hill Education, 2008

Publikationen

[5] Byers, J. W., Luby, M., Mitzenmacher, M., & Rege, A. *A digital fountain approach to reliable distribution of bulk data*. *ACM SIGCOMM Computer Communication Review*, *28*(4), 1998, pp56-67

[6] Demir, Ufuk und Aktas, Ozlem: *Raptor versus Reed Solomon Forward Error Correction Codes.* Dokuz Eyluil University Izmir, Turkey, 2006, pp 264-269

[7] Ho, Tracey: *Summary of Raptor Codes*. October 29, 2003

[8] Joshi, Gauri, Rhim, Joong Bum, Sun, John, Wang, Da: *Fountain Codes* In: *Global telecommunications conference (GLOBECOM 2010).* Dezember 2010

[9] Khisti, Ashish: *TornadoCodes and Luby Transform Codes.* October 22, 2003

[10] Luby, M.: *LT codes*. Proc. 43rd Ann. IEEE Symp. on Foundations of Computer Science, 16–19 November 2002, pp. 271–282

[11] MacKay, D.J.C.: *CAPACITY APPROACHING CODES DESIGN AND IMPLEMENTATION SPECIAL SECTION Fountain codes.* IEE Proc.-Commun., Vol. 152, No. 6, December 2005, 1062-1068

[12] Maymounkov, Petar: *Online codes (Extended Abstract).* Secure Computer Systems Group, New York University, November 2002

[13] Palanki, Ravi und Yedidia, Jonathan S.: *Rateless Codes on Noisy Channels.* Mitsubishi Electric Research Laboratories, Inc., Cambridge, Massachusetts, 2004

[14] Tirronen, Tuomas: *Optimal Degree Distributions for LT Codes in Small Cases.* HELSINKI UNIVERSITY OF TECHNOLOGY Networking Laboratory S-38.138 Networking Technology, Special Assignment, 25.11.2005

Abschlussarbeiten/Dissertationen:

[15] Blasco, Francisco Lázaro: *Fountain Codes under Maximum Likelihood Decoding.* Technische Universität Hamburg-Harburg, 2017

[16] Dönicke, Martin: *Betrachtung von ratenlosen Codes und prototypische Implementierung von Online Codes.* Hochschule für Technik und Wirtschaft Dresden, University of Applied Sciences, 9. Dezember 2013

[17] Sejdinovi¢, Dino: *Topics in Fountain Coding.* University of Bristol, Department of Electrical and Electronic Engineering, September 2009

Internetquellen:

[18] Bundesamt für Strahlenschutz “ELEKTROMAGNETISCHE FELDER: Strahlenschutzaspekte bei Ganzkörperscannern” bfs.de, zul. geändert 07.18 [Online].

Verfügbar:http://www.bfs.de/DE/themen/emf/hff/quellen/ganzkoerperscanner/ganzkoerperscanner_node.html , [zuletzt aufgerufen: 27.06.19].

[19] Dipl. Ing. (FH) Christian Wolff "Radartutorial" radartutorial.eu, 1998, zul. geändert 05.19 [Online]. Verfügbar: http://www.radartutorial.eu/index.html , [zuletzt aufgerufen: 26.06.19].

[20] User: ELE ELE "Fountain Codes - Michael Luby - 2004" youtube.com, 28.06.2016. [Online]. Verfügbar: https://youtu.be/s3lrmBczBTc . [zuletzt aufgerufen: 26.06.19].

Technische Standards:

[21] Michael Luby, Amin Shokrollahi, Mark Watson , Thomas Stockhammer: *Raptor Forward Error Correction Scheme for Object Delivery* Internet Engineering Steering Group, October 2007

[22] M. Luby, Amin Shokrollahi, Mark Watson , Thomas Stockhammer: *RaptorQ Forward Error Correction Scheme for Object Delivery.* Internet Engineering Steering Group, August 2011

[23] QUALCOMM Incorporate, *RaptorQ™ Technical Overview,* 2010

Matlab-Code:

[24] Diamant, Roee, "*Raptor and Fountain Codes"*, [MATLAB File Exchange] University of British Columbia, July 2012 [zuletzt aufgerufen: 26.06.19].

Personen [Vorlesung/Betreuung/Code]

[25] Prof. Dr.-Ing Jan Mietzner, Hochschule für angewandte Wissenschaften Hamburg, 2019.
=> Vorlesung: Nachrichtentechnik/Digitale Signalverarbeitung/Nachrichten- und Telekommunikationstechnik
=> Vorlesung: "An Introduction to Radar Signal Processing and Algorithms"
=> MATLAB Code: "Sim_AF*"*