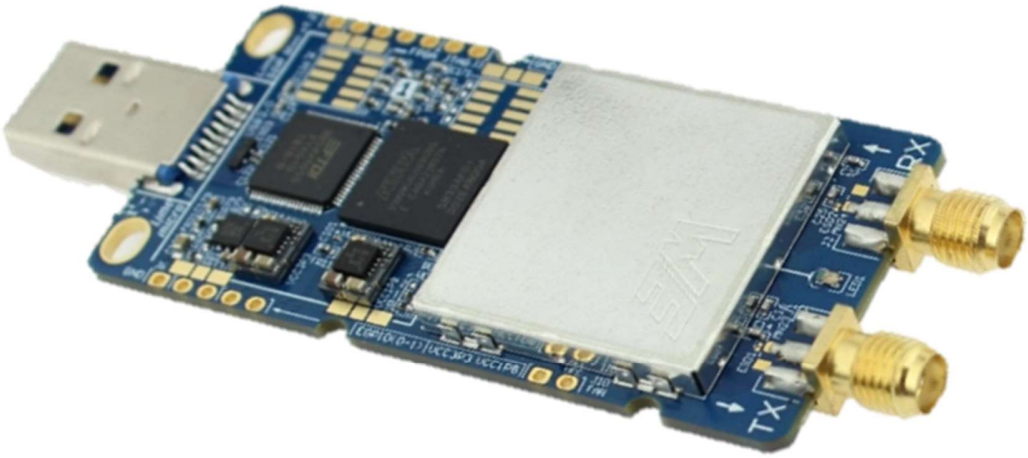
Software-Defined Radio

Bachelorarbeit HS 2019



LimeSDR

GNURadio
THE FREE & OPEN SOFTWARE RADIO ECOSYSTEM

python

Tobias Mailänder

Bachelor-Thesis an der Hochschule Luzern - Technik & Architektur

Titel	**Datenübertragung mit Rückkanal in Software-Defined Radio**
Diplomandin/Diplomand	**Mailänder, Tobias**
Bachelor-Studiengang	**Bachelor Elektrotechnik und Informationstechnologie**
Semester	**HS19**
Dozentin/Dozent	**Prof. Dr. Thomas Hunziker**
Expertin/Experte	**Werner Scheidegger**

Abstract Deutsch
Die Hochschule Luzern untersucht die Möglichkeit mit SDRs eine Datenübertragung mit Rückkanal zu entwickeln. Für die Untersuchung werden die Modelle LimeSDR mini von Lime microsystems verwendet.

Mit GNU Radio können die Funkmodule programmiert werden und ermöglicht viele verschiedene Realisierungen der Übertragung. Die Signalverarbeitung von GNU Radio kann mit Python ergänzt werden. Das gesendete Signal wird QPSK moduliert und bei einer Frequenz von 868.1 MHz übermittelt. Die Empfängerseite filtert das Signal mit einem root raised cosine filter und schreibt es in einen Socket. Ein Python Programm greift auf den Socket zu und verarbeitet die empfangenen Signale.
Die Pakete werden durch eine Korrelation der empfangenen Signale mit der Präambel erkannt und die angehängten Symbole rekonstruiert. Mit dem Maximum Likelihood Estimation Verfahren wird der Time Offset und Phase Drift geschätzt und die Symbole korrigiert. Nach der Korrektur wird mit einem PLL den Oscillator Drift kompensiert und die Symbole geschätzt. Nach der Auswertung der geschätzten Symbole des Pakets wird eine Antwort gebildet und zurückgesendet.

Eine Recherche im Internet hat gezeigt, auf welche Weise eine bidirektionale Kommunikation koordiniert werden kann. Damit sich die Kommunikation nicht selber stört, wird der Kanal als time division duplex (TDD) verwendet. Als ARQ Protokoll steht das Stop and Wait Verfahren im Einsatz. Zur Kontrolle, ob ein Paket korrekt ankommt, wird mit CRC ein Prüfwert an das Ende des Pakets angefügt.

Mit einer Simulation der Kommunikation und Funkverbindung, konnte das Konzept geprüft und bestätigt werden. Nachdem die Kommunikation mit den Funkmodulen unter realen Bedingungen ausgeführt wurde, wurden viele Pakete nicht erkannt oder nicht akzeptiert. Was die genauen Quellen dieser Fehler sind, konnte nicht entdeckt werden. Die Signalverarbeitung und das Konzept ist durch die Simulation bestätigt.

Damit die Kommunikation verwendet werden kann, werden die Pakete mit einem fehlerkorrigierenden Code ergänzt. Durch eine Recherche wurde der Zyklischer Hamming Code entdeckt und umgesetzt. Die vom Block Code generierte Bitfolge besteht aus 4 Datenbits und 3 Kontrollbits.
Das Ergebnis zeigt, dass die Pakete öfters akzeptiert werden und die Kommunikation besser vorankommt. Es ist jedoch zu berücksichtigen, dass die Funkverbindung nicht verbessert wurde. Das Konzept wurde ergänzt um die Fehler möglichst gut zu kompensieren.

Abstract Englisch
The Lucerne University of Applied Sciences and Arts is investigating the possibility of using SDRs to develop data transmission with a return channel. For the investigation the models LimeSDR mini from Lime microsystems are used.

With GNU Radio the radio modules can be programmed and allows many different realizations of the transmission. The signal processing of GNU Radio can be completed with Python.
The transmitted signal is QPSK modulated and transmitted at a frequency of 868.1 MHz. The receiver side filters the signal with a root raised cosine filter and writes it into a socket. A Python program accesses the socket and processes the received signals.
The packets are recognized by a correlation of the received signals with the preamble and the attached symbols are reconstructed. With the Maximum Likelihood Estimation procedure the time offset and phase drift is estimated and the symbols are corrected. After the correction the oscillator drift is compensated with a PLL and the symbols are estimated. After evaluation of the estimated symbols of the packet a response is formed and sent back.

A research on the Internet has shown how bidirectional communication can be coordinated. In order to prevent the communication from interfering itself, the channel is used as time division duplex (TDD). As ARQ protocol the stop and wait procedure is used. To check whether a packet arrives correctly, a check value is added to the end of the packet with CRC.

With a simulation of the communication and radio link, the concept could be tested and confirmed. After the communication with the radio modules was carried out under real conditions, many packets were not recognized or not accepted. What the exact sources of these errors are could not be discovered. The signal processing and the concept is confirmed by the simulation.

In order to use the communication, the packets are supplemented with an error correcting code. Through research the cyclic Hamming code was discovered and implemented. The bit sequence generated by the block code consists of 4 data bits and 3 control bits.
The result shows that the packets are accepted more often and the communication progresses better. However, it should be noted that the radio link has not been improved. The concept has been extended to compensate the errors as good as possible.

Ort, Datum Horw, 06. Januar 2020

1 Inhalt

2 Aufgabenstellung

Die Bachelorarbeit basiert auf zwei abgeschlossenen Industrieprojekten, welche sich mit der Datenübermittlung mit dem LimeSDR mini befassen.

In der Arbeit von Warisa Kulcharoenwirat geht es um das Senden und Empfangen einer Präambel und in der Arbeit von Peerawit Khetsakun um die Signalverarbeitung.

In dieser Arbeit geht es darum, die Möglichkeiten des LimeSDR Mini zur Datenübertragung mit Rückkanal zu betrachten. Bei einer OFDM-Übertragung beispielsweise kann der Sender über einen Rückkanal über den Kanalzustand informiert und dementsprechend die Modulation angepasst werden. Die Latenz muss sich dafür allerdings in Grenzen halten.

2.1 Software-Defined Radio

Unter Software Defined Radio (SDR) fasst man Konzepte für Hochfrequenz-Sender und -Empfänger zusammen, bei denen kleinere oder größere Anteile der Signalverarbeitung mit Software verwirklicht werden. Vor allem Selektion und Modulation/Demodulation werden bei einem SDR mittels digitaler Signalverarbeitung verwirklicht. [1]

2.1.1 LimeSDR mini

In dieser Bachelorarbeit wird mit dem LimeSDR mini gearbeitet, welches ein günstiges Software defined Radio Board ist und für die Entwicklung von RF Designs und Prototypen eingesetzt wird. Der LimeSDR mini verfügt über ein Intel MAX 10 FPGA, Lime Microsystems LMS7002M RF transceiver und kann in einem Frequenzbereich von 10MHz bis 3.5GHz arbeiten. Das LimeSDR mini Board wird über eine USB 3.0 Schnittstelle programmiert. [2]

Tabelle 1 zeigt eine Übersicht der wichtigsten Daten des LimeSDR mini Moduls.

Tabelle 1 Board Eigenschaften

RF Transceiver	Lime Microsystems LMS7002M
RF Frequenz Bereich	10MHz bis 3.5GHz
FPGA	Intel MAX 10
EEPROM Memory	2x 128Kb
FLASH Memory	4Mb
USB 3.0 Kontroller	FTDI FT601
RF Schnittstelle	2x Coaxial RF (SMA female)
Clock System	40.00MHz onboard VCTCXO
Board Grösse	69mm x 31.4mm

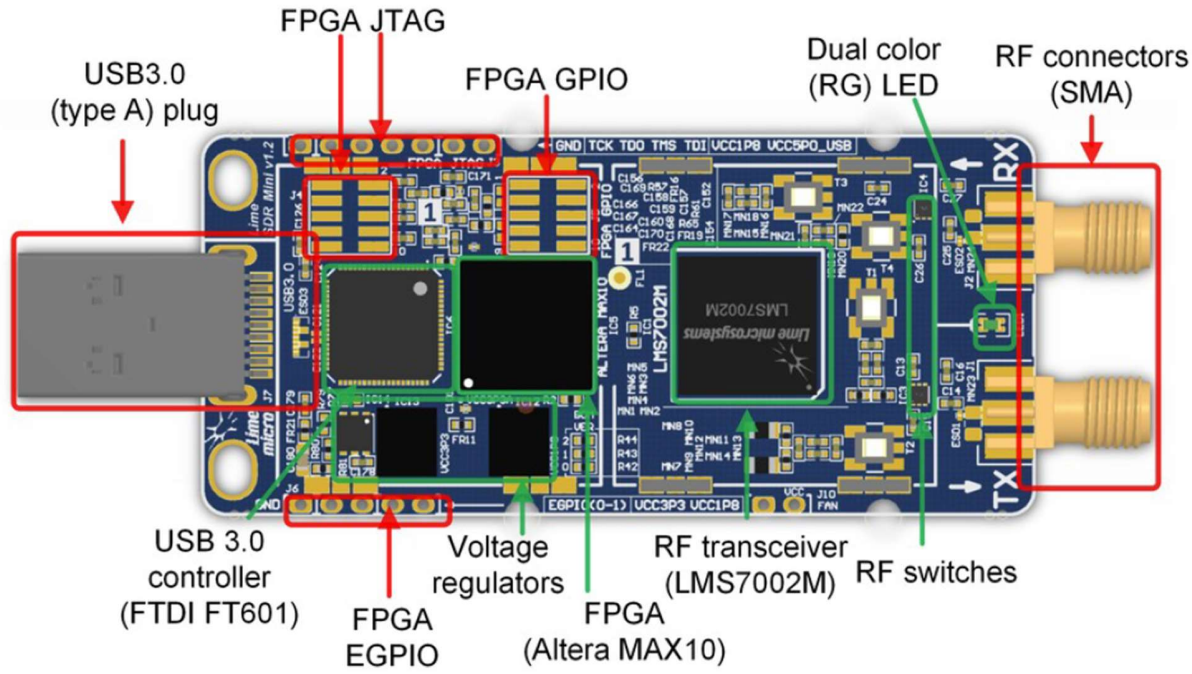


Abbildung 1 LimeSDR-Mini v1.2 board top connectors and main components

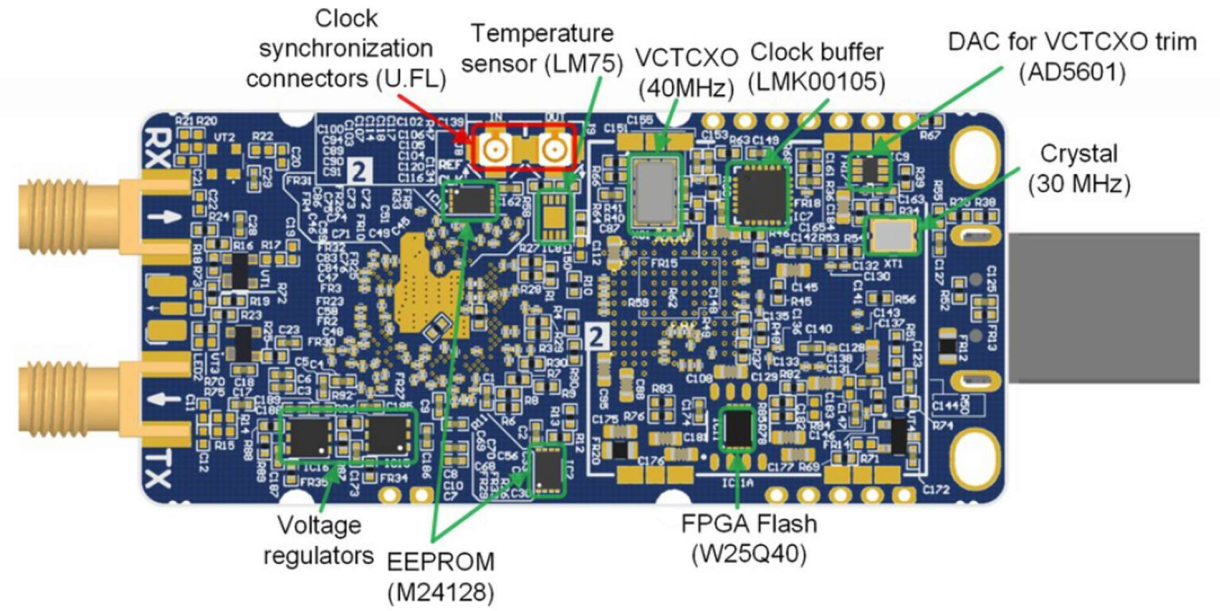


Abbildung 2 LimeSDR-Mini v1.2 board bottom connectors and main components

Hinweis: Es gibt verschiedene Versionen der Module und nicht alle aufgezeigten Bauteile sind bestückt.

3 Konzept

In diesem Abschnitt werden die möglichen Umsetzungen erläutert und welches Konzept in der Arbeit umgesetzt wird.

3.1 Rückkanal

Mit dem Rückkanal wird ein Signalaustausch in beide Richtungen ermöglicht. Der Empfänger kann den Sender über die empfangenen Daten informieren und so das Senden des nächsten Pakets beeinflussen.

In Abbildung 3 Konzept Rückkanalist das Konzept für die Kommunikation mit Rückkanal dargestellt. Der Pfad des Senders ist dem Pfad des Empfängers sehr ähnlich. Der einzige Unterschied liegt in den Python Files, welche die gesamte Kommunikation koordinieren und die Signale verarbeiten.

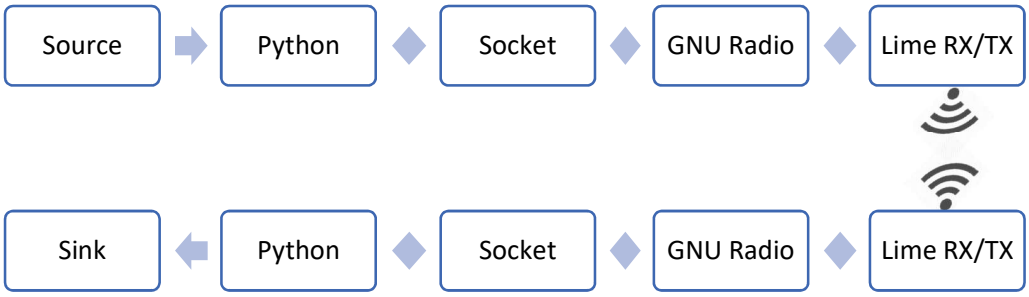


Abbildung 3 Konzept Rückkanal

3.2 FDD/TDD

Der Rückkanal darf den Kanal des Senders nicht beeinflussen. Um die zwei Kanäle sauber voneinander zu trennen, gibt es zwei Möglichkeiten.

Mit FDD (frequency division duplex) kommuniziert der Rückkanal auf einer anderen Frequenz und ist somit vom Kanal des Senders getrennt. Abbildung 4 FDDzeigt, dass ein genügend grosses Guard Band benötigt wird, damit sich die zwei Trägerfrequenzen nicht stören

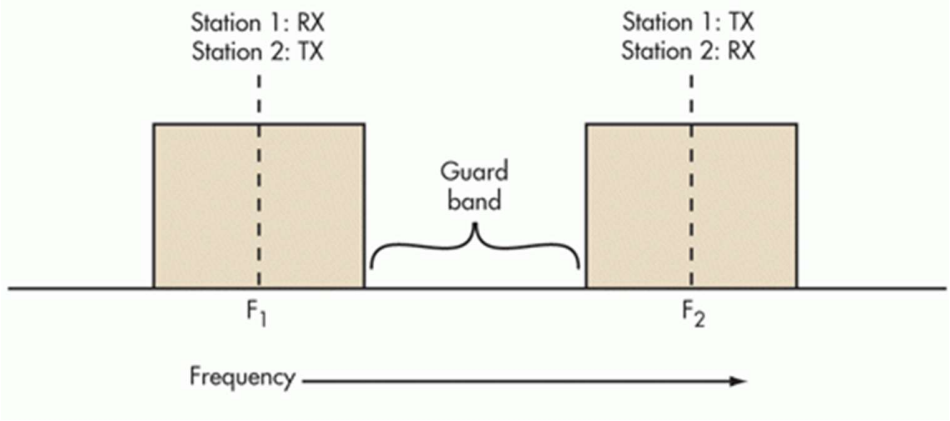


Abbildung 4 FDD [3]

Mit TDD (time divison duplex) wird für beide Kanäle dieselbe Frequenz verwendet. Das Senden und Empfangen wird wie in Abbildung 5 TDDgezeigt, abgewechselt und so koordiniert, dass nicht zur selben Zeit beide Funkmodule senden oder empfangen. Es muss beachtet werden, dass ein Paket erst gesendet wird, wenn der Empfänger bereit ist. Damit beim Abtausch von Sender und Empfänger keine Kollisionen entstehen, wird eine Guard Time benötigt. [3]

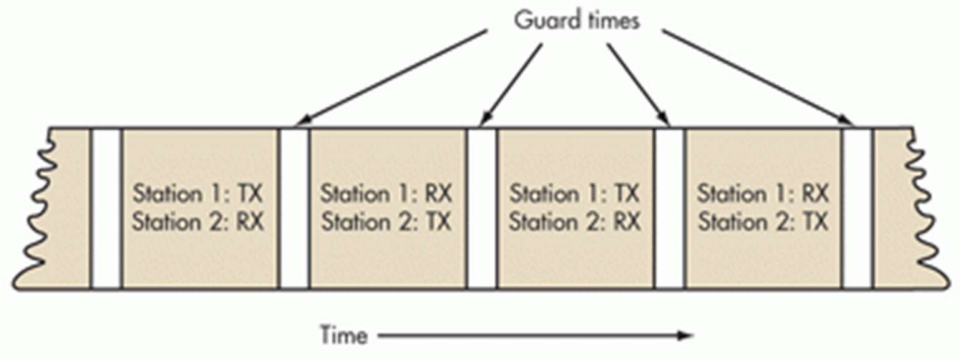


Abbildung 5 TDD [3]

Wenn FDD verwendet wird, kann in beide Richtungen zeitgleich kommuniziert werden. Dies ist ein effizienterer Ansatz, jedoch müssen die Pakete vorsichtig koordiniert werden. Der Sender muss stets wissen, welche erhaltene Information zu welchem gesendeten Paket gehört. Für dies werden die Pakete nummeriert und in Listen eingetragen. Gemäss den Informationen vom Empfänger, werden die Pakete aus der Liste gestrichen, oder nochmals gesendet.

Mit der Verwendung von TDD wechselt die Kommunikation ab und es wird eine einzelne Trägerfrequenz verwendet. Es ist zu beachten, dass ein Funkmodul sein eigenes gesendete Signal empfangen kann und ignorieren muss. Aus diesem Grund erhält jedes Funkmodul seine eigene Identität. Die Identität des gewählten Empfängers, wird als Präambel dem Paket hinzugefügt. So wird jedes Paket nur vom Empfänger mit der entsprechenden Identität erkannt und ausgewertet.

3.3 Datenpakete

Die zu sendende Datei wird in Pakete zerteilt und einzeln übermittelt.

Das gesendete Paket des Senders sieht wie folgt aus:

Empfänger Id	Daten	Prüfwert

In der Präambel wir die Identität des Empfängers eingefügt. Anschliessend wird ein Teil der zu sendenden Datei hinzugefügt und mit einem Prüfwert ergänzt. Der Prüfwert wird mit den gesendeten Daten berechnet und ist für jedes weitere Paket anders.

Das gesendete Paket des Empfängers sieht wie folgt aus:

Sender Id	Bestätigung

In der Präambel wir die Identität des Senders eingefügt und mit der Bestätigung ergänzt. Die Bestätigung kann positiv oder negativ ausfallen. Bei einer positiven Bestätigung wird das nächste Paket gesendet und bei einer negativen Bestätigung wird das Paket wiederholt gesendet.

3.4 Paketüberprüfung

Das erhaltene Paket wird vom Empfänger, mit Hilfe des Prüfwerts, auf Fehler überprüft und entscheidet, ob das Paket erneut gesendet werden muss. Der Prüfwert wird mit dem zyklischen Redundanzprüfverfahren berechnet (englisch cyclic redundancy check, daher meist CRC). Im Idealfall kann das Verfahren sogar die empfangenen Daten selbständig korrigieren, um eine erneute Übertragung zu vermeiden.

Wenn ein Paket fehlerhaft empfangen wurde, wird es über ARQ-Protokolle (englisch Automatic Repeat reQuest, dt. Automatische Wiederholungsanfrage) neu angefordert.

Gewöhnlicherweise geschieht dies durch Übertragung sogenannter ACK/NAK Signale (Acknowledgement bzw. Negative Acknowledgement, d.h. korrekter Empfang bestätigt bzw. Wiederholungsanfrage). Gegebenenfalls wird eine gestörte Nachricht solange erneut übertragen, bis sie den Empfänger ohne Fehler erreicht hat.
Folgende Protokolle werden als grundlegend betrachtet: [4]

3.4.1 Stop-and-Wait

Stop-and-Wait (engl. für *Halte an und Warte*, auch als Send-and-Wait bezeichnet) stellt das einfachste Verfahren dar und ist in Abbildung 6 dargestellt. Nachdem ein Paket gesendet wurde, muss der Sender auf die Bestätigung warten, bevor das nächste Paket gesendet wird.

Falls der Sender keine Bestätigung innerhalb eines bestimmten Zeitrahmens (Timeout) empfängt, muss das Paket noch einmal gesendet werden. Dies kann passieren, wenn wie in Abbildung 7 gezeigt ein Paket verloren geht.

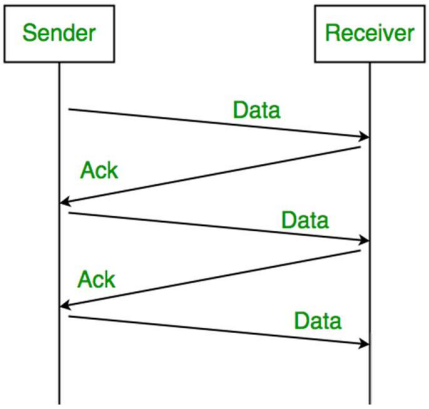


Abbildung 6 Stop and Wait [5]

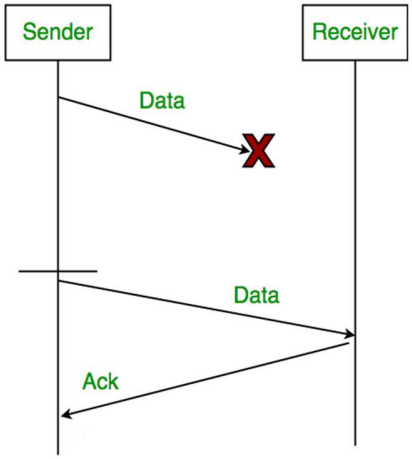


Abbildung 7 Timeout [5]

Es ist zu beachten, dass das Datenpakete wie auch die Bestätigung verloren gehen kann. Wenn eine positive Bestätigung verloren geht, wird dasselbe Paket wieder gesendet, obwohl der Empfänger das nächste Paket erwartet.
Wenn der Empfänger erkennt, dass ein Paket gesendet wird, welches bereits erfolgreich empfangen wurde, wird das Paket ignoriert und eine positive Bestätigung gesendet.

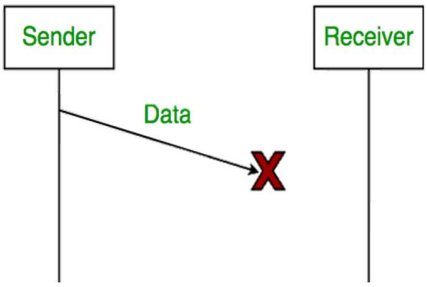


Abbildung 8 Datenpaket verloren [5]

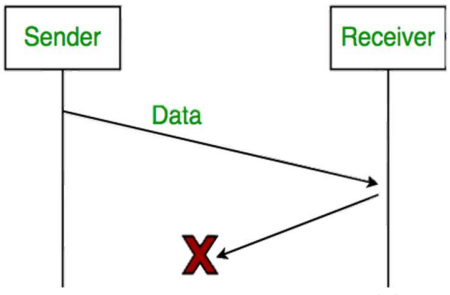


Abbildung 9 Bestätigung verloren [5]

3.4.2 Go-Back-N

Go-Back-N (engl. für *Gehe zu N zurück*) stellt ein Verfahren dar, das im Gegensatz zu Stop-and-Wait einen deutlich größeren Durchsatz ermöglicht.
Der Sender kann dabei mehrere Pakete senden, ohne auf eine Bestätigung warten zu müssen. Wie viele das sind, hängt von der sogenannten Fenstergröße ab.

Wenn z.B. das Paket mit der Nummer 2 verloren geht und nicht bestätigt werden kann, kommt es beim Warten auf die Bestätigungen zu einem Timeout. Der Sender geht zurück zum letzten unbestätigten Paket und sendet alle Pakete in diesem Fenster neu.

Da es der Fall sein kann, dass lediglich ein Paket im Fenster nicht ordnungsgemäß übertragen wurde, werden dennoch alle danach gesendeten Pakete erneut übertragen. So wird an dieser Stelle Übertragungskapazität verschwendet.

3.4.3 Selective Repeat

Selective Repeat (engl. für *Selektive Wiederholung*) ist eine weitere allgemeine Fehlerbehandlungsstrategie, welche das Go-Back-N Verfahren erweitert. Geht bei der selektiven Wiederholung ein Paket verloren, wird nicht wie bei Go-Back-N ab dieser Stelle alles neu gesendet, sondern lediglich das verlorene Paket erneut dem Fenster hinzugefügt. Die Reihenfolge der empfangen Pakete kann so durcheinanderkommen und muss mit Hilfe eines Buffers korrigiert werden.

[5]

3.5 Umsetzung

Zu Beginn liegt der Fokus darin eine funktionierende Kommunikation zu implementieren. Sobald die Kommunikation besteht kann Sie in Effizienz und Geschwindigkeit optimiert werden.

Umgesetzt wird die Kommunikation mit TDD. Als ARQ-Protokoll wird das Stop-and-Wait Verfahren angewendet. Dieses Konzept ist das einfachste zur Umsetzung und Überprüfung der Funktion.

4 Funktion

In diesem Abschnitt wird der Ablauf und die Funktion der Datenübermittlung mit Rückkanal beschrieben.

4.1 GNU Radio Transceiver

Der Transceiver in GNU Radio verbindet das Python File über den Socket mit dem Funkmodul.

Die verwendeten Funktionsblöcke sind in Abbildung 10 ersichtlich.

Das Paket wird über den Socket als binäre Datei gesendet und mit dem Constellation Modulator QPSK moduliert. Die Kommunikation verläuft bei einer Trägerfrequenz von 868.1MHz. Die Sample Rate liegt bei 50k was mit einer Samples/Symbol Rate von 4 zu einer Symbolrate von 12.5k führt. Mit der QPSK Modulation entspricht dies einer Bitrate von 25kbit/s.
Das empfangene Signal wird mit einem RRC Filter gefiltert und über einen Stream zum Socket geschrieben. Über den Socket erhält das Python File das empfangene Signal und wertet es aus.

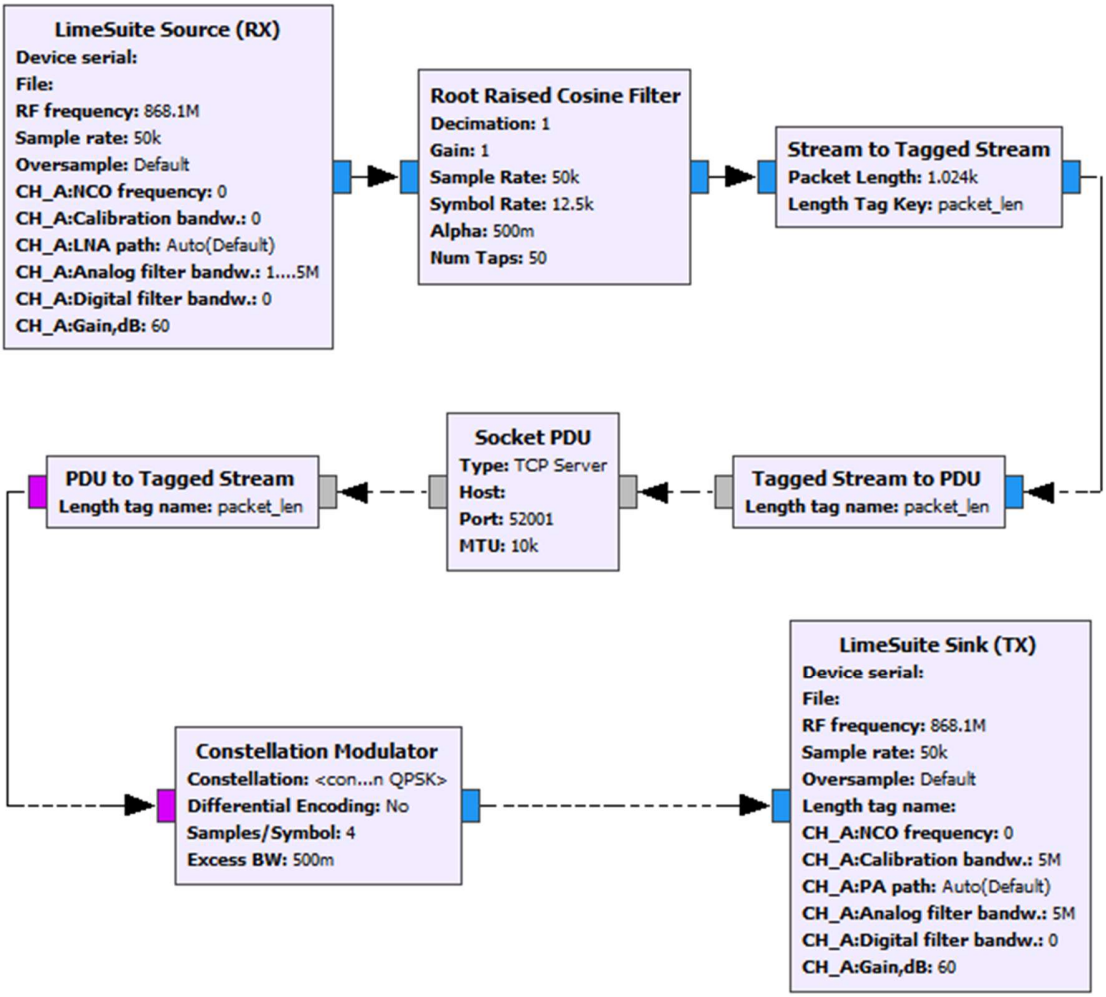


Abbildung 10 GNU Radio Transceiver

4.2 Datenpakete

Die Daten, welche gesendet werden, werden in 256 Byte grosse Pakete aufgeteilt. Diese Grösse entspricht 1024 Symbolen. Damit der Empfänger das Paket erkennt, wird die 16 Byte grosse Präambel angefügt. Für die Überprüfung des Pakets, wird die Prüfsumme am Ende des Pakets angefügt und hat eine Grösse von 10 Byte. Dies ergibt eine gesamte Paketgrösse von 282 Bytes oder 1128 Symbolen.

```
amt_of_pkg = (len(source)/msg_byte_len) + 1       # amount of packages
for pkg_cnt in range(amt_of_pkg):                 # for all packages
    # create empty package
    package = bytearray([])
    # add receiver id
    package.extend(r_id[0:pre_byte_len])
    # add data from file
    package.extend(source[pkg_cnt*msg_byte_len:(pkg_cnt+1)*msg_byte_len])
    # calculate crc control value
    msg_symb = s_symb[(pkg_cnt*msg_len):((pkg_cnt+1)*msg_len)]
    crc_value = bytes(crc32_func("".join(str(i) for i in msg_symb)))
    # add crc control value
    package.extend(crc_value)
    # sending created package
    s.sendall(package)
```

4.3 Timeout

Nachdem das Paket gesendet wurde, wird der Timout Counter gestartet und auf die Bestätigung des Pakets gewartet. Wenn der Timout Counter den Maximalwert von timout_value erreicht, wird das Paket erneut gesendet und der Timout Counter zurückgesetzt. Dies wiederholt sich solange, bis eine Bestätigung erkannt wird.

```
# reset timeout counter
timeout_cnt = 0
ack = False
while not ack:                                    # waiting for acknowledgment
    timeout_cnt = timeout_cnt + 1
    if timeout_cnt > timeout_value:
        print ("\nacknowledge timed out")
        s.sendall(package)
        timeout_cnt = 0
```

4.4 Präambel erkennen

Damit der Empfänger das gesendete Paket vom Sender erkennt, muss das Programm wissen, wonach es sucht. Für dies wird die Präambel vom Python File gelesen, QPSK moduliert und mit 4 Samples/Symbol gestreckt. Das zu suchende Muster in den empfangenen Daten ist in pus4 gespeichert.

```
r_id = np.fromfile(open("receiver_id.bin"),dtype=np.uint8)
r_symb = np.zeros(4*len(r_id),dtype=int)
for i in range(len(r_id)):
    for j in range(4):
        if (('0000000'+bin(r_id[i])[2:])[-8+2*j]=='1'):
            if (('0000000'+bin(r_id[i])[2:])[-7+2*j]=='1'):
                r_symb[4*i+j] = 2
            else:
                r_symb[4*i+j] = 3
        else:
            if (('0000000'+bin(r_id[i])[2:])[-7+2*j]=='1'):
                r_symb[4*i+j] = 1
            else:
                r_symb[4*i+j] = 0

# QPSK modulation
preamble = np.exp(0.5j*np.pi*(r_symb[0:pre_len]+0.5))
# 4-fold oversampled version
pus4 = np.vstack((np.array(preamble),np.zeros((3,len(preamble)))))
pus4 = np.reshape(pus4,4*len(preamble),order='F')
```

Die empfangenen Daten werden paketweise gelesen und mit der Präambel verglichen.

Es werden 8192 Bytes empfangen und zu complex64 umgewandelt, was 1024 Symbolen entspricht. Diese 1024 Symbole werden in `rx` gespeichert. Vom alten `rx` wird der hintere Bereich von der Länge `pus4+15` in das neu generierte `rx` übernommen. So wird sichergestellt, dass die mögliche empfangene Präambel nicht durch die Pakete zerteilt wird.

Das `rx` erhält so eine Grösse von 1295 Symbolen worin `pus4` mit der Grösse von 256 Symbolen gesucht wird.

```
while (len(data)<8192):
    data = data+s.recv(4096)
rx = np.concatenate((rx[rx.shape[0]-pus4.shape[0]-15:rx.shape[0]],
        np.frombuffer(data[0:8192],dtype=np.complex64)))
data = data[8192:len(data)]
```

In cp wird die Korrelation von rx und pus4 gespeichert. Wenn in den empfangenen Daten pus4 vorkommt, entsteht an dieser Stelle ein sehr hoher Wert. Der entstehende Peak ist in Abbildung 11 gut zu erkennen.

```
cp = np.abs(np.correlate(rx,pus4,"valid"))
```

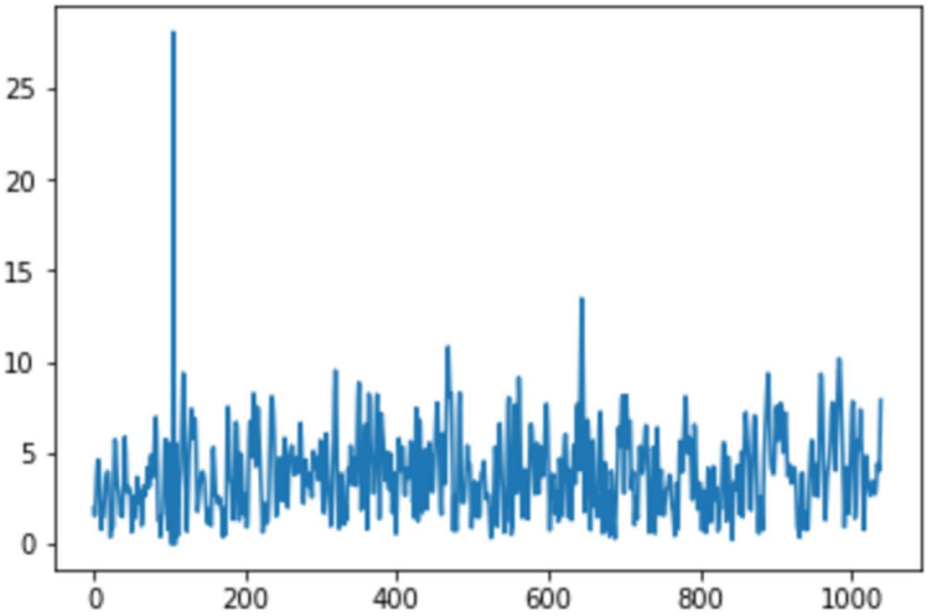


Abbildung 11 Korrelation von rx und pus4 mit erkannter Präambel

Als Kriterium für die korrekte Präambel, darf dieser Peak im gesamten Paket, welches untersucht wird, nur einmal vorkommen. Um dies zu überprüfen wird der Peak und die benachbarten Werte auf null gesetzt.
Im ganzen cp darf kein weiterer Wert vorkommen, welcher grösser ist, als 50% vom Peak. Wird dieses Kriterium erfüllt, gilt die Präambel als gefunden und wird in rxp gespeichert.

```
pi0 = np.argmax(cp)    # peak index
vp = cp[pi0]
if ((pi0>=8) and (pi0<cp.shape[0]-8)):
    cp[pi0-3:pi0+4] = 0
    if (np.max(cp) < 0.5 * vp):
        cp[pi0] = vp
        rxp = rx[pi0:pi0 + 4 * pre_len]
```

4.5 Korrektur

Die empfangenen Symbole haben aufgrund der Funkverbindung einige Fehler und Ungenauigkeiten, welche für die korrekte Datenerfassung korrigiert werden müssen.

Anhand der empfangenen Präambel wird der time offset und den phase drift mit dem Maximum Likelihood Estimation Verfahren ermittelt.

Nachdem die Korrekturen ermittelt wurden, kann das gesamte Paket gelesen und korrigiert werden.

```
# --- symbol time offset estimation ------------------------------------------------
vmax = 0
for dt in np.linspace(-0.125, 0.125, num=21):
    vdt = np.abs(np.sum(np.matmul(rrcscorr(lm - dt), preamble) * np.conj(rxp)))
    if (vdt > vmax):
        dt0 = dt
        vmax = vdt

# --- coarse phase drift estimation ------------------------------------------------
vmax = 0
for df in np.linspace(-0.1, 0.1, num=41):
    vdf = np.abs(np.sum(np.matmul(rrcscorr(lm - dt0), preamble) * np.exp(
        0.25j * df * np.arange(4 * pre_len)) * np.conj(rxp)))
    if (vdf > vmax):
        df0 = df
        vmax = vdf

# --- initial phase reference ------------------------------------------------------
ph0 = np.angle(np.sum(rxp * np.conj(
    np.matmul(rrcscorr(lm - dt0), preamble) * np.exp(0.25j * df0 * np.arange(4 *
pre_len)))))


# --- receive whole package --------------------------------------------------------
while (rx.shape[0] - pi0 < 4 * pkg_len + l):
    while (len(data) < 4096):
        data = data + s.recv(4096)
    rx = np.concatenate((rx, np.frombuffer(data[0:4096], dtype=np.complex64)))
    data = data[4096:len(data)]

# --- sampling time offset correction ----------------------------------------------
rxf = np.matmul(minv, np.transpose(rrcscorr(0.25 * np.arange(-l, l + 1) - dt0)))
r = np.zeros(pkg_len, dtype=complex)
k = pi0
for i in np.arange(r.shape[0]):
    r[i] = np.sum(
        rx[np.arange(k - l, k + l + 1)] * np.exp(
            -1j * df0 * np.arange(i - l, i + l + 1)) * rxf) * np.exp(
        -1j * ph0)
    k = k + 4
```

Weil der Oszillator von Sender und Empfänger nicht synchronisiert sind, entsteht ein oscillator drift. Um diesen oscillator drift zu korrigierten, wird ein phase lock loop verwendet.

In einer Schlaufe wird der abweichende Winkelwert ephi erfasst und integriert. Das nächste Symbol wird mit einem proportionalen und einem integrierten Teil des Fehlers korrigiert. Dieses Verfahren entspricht einem PI-Regler und ist in Abbildung 12 mit Hilfe eines Blockdiagramms dargestellt.

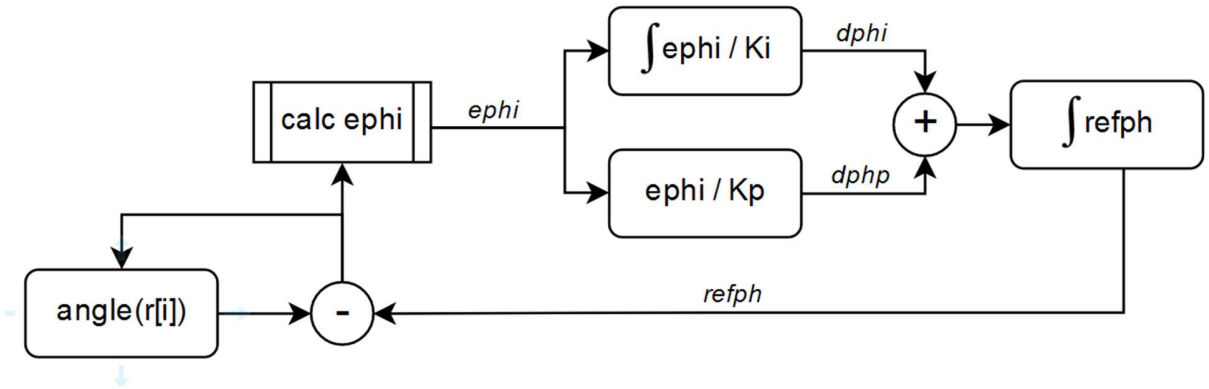


Abbildung 12 Blockdiagramm PLL

```
# --- fine oscillator drift compensation (PLL) -------------------------------------
refph = 0
dphi = 0
Ki = 1000
Kp = 50
for i in np.arange(r.shape[0]):
    r[i] = r[i] * np.exp(-1j * refph)
    ephi = (np.angle(r[i]) + np.pi) % (0.5 * np.pi) - 0.25 * np.pi
    dphi = dphi + ephi / Ki
    dphp = ephi / Kp
    refph = refph + dphi + dphp
```

4.6 Auswertung des Pakets

Nachdem das Paket empfangen und korrigiert wurde, kann die Auswertung beginnen. Für dies werden die Symbole anhand des Winkels geschätzt und in estsymb gespeichert.

```
# --- estimate received symbols ----------------------------------------------------
estsymb = np.zeros(r.shape[0], dtype=int)
for i in np.arange(r.shape[0]):
    estsymb[i] = int((np.angle(r[i]) + 2 * np.pi) / (0.5 * np.pi)) % 4
```

Von dem Paket werden die Daten und der Prüfwert entnommen.

```
received_msg = estsymb[pre_len:pre_len + msg_len]
received_crc = estsymb[pre_len + msg_len:pkg_len]
```

Falls die positive Bestätigung des letzten Pakets verloren geht, wird dasselbe Paket nochmals gesendet, obwohl das nächste erwartet wird. Um dies zu verhindern wird das letzte akzeptierte Paket gespeichert und mit dem empfangenen Paket verglichen. Entspricht es demselben Paket, wird eine positive Bestätigung gesendet.

Handelt es sich um ein neues Paket, wird die Prüfsumme berechnet und mit der empfangenen Prüfsumme verglichen. Sind die beiden Prüfsummen identisch, wird eine positive Bestätigung gesendet. Wenn die Prüfsummen unterschiedlich sind, wird eine leere Bestätigung gesendet. Als positive Bestätigung wird ein binäres File von der Grösse 10 Byte angefügt.

```
crc_value_calc = crc32_func("".join(str(i) for i in received_msg))

# --- acknowledge received package -------------------------------------------------
ack = bytearray([])                                     # prepare acknowledgement
ack.extend(t_id[0:pre_byte_len])                        # add transmitter id
if sum((last_accepted_msg - received_msg) != 0) < 5:    # package already accepted
    ack.extend(p_ack)                                   # add p acknowledgement

elif (crc_value_calc - crc_value_recv) == 0 :
    print ("package accepted")
    last_accepted_msg = received_msg                   # save the accepted message
    pkg_cnt = pkg_cnt + 1                              # expect next package
    ack.extend(p_ack)                                  # add p acknowledgement
else:
    print ("\nplease send package again.")

ack.extend(bytearray(np.zeros(10)))             # add zeros to flush
time.sleep(2)                                   # delay for presentation
s.sendall(ack)                                  # send acknowledgement
```

4.7 Bestätigung

Wenn die Bestätigung vom Empfänger erhalten wird, wird untersucht, ob es sich um eine positive oder negative Bestätigung handelt. Die erhaltene Bestätigung wird vor der Auswertung auf dieselbe Weise wie beim Empfänger korrigiert und die Symbole abgeschätzt.

Der Sender verfügt über dasselbe File, welches bei einer positiven Bestätigung angefügt wird. Die daraus berechneten Symbole werden mit den erhaltenen Symbolen verglichen. Wenn die erhaltenen Symbole nahezu identisch mit den berechneten Symbolen sind, wird die Bestätigung als positiv gewertet und das nächste Paket kann gesendet werden. Gibt es jedoch mehr als 5 Abweichungen, wird die Bestätigung als negativ gewertet und das Paket wiederholt gesendet.

```
received_ack = estsymb[pre_len:pre_len+p_ack_len]
if sum((received_ack-p_ack_symb) != 0) < 5:
    ack = True
else:
    s.sendall(package)  # sending created package
    timeout_cnt = 0
```

5 Ergebnisse

In diesem Abschnitt sind die Ergebnisse und Erkenntnisse der Entwicklung beschrieben.

5.1 PLL

In Abbildung 13 sind die Symbole nach der Korrektur des PLLs angezeigt. Es ist gut zu erkennen, wo sich die vier Winkel der QPSK Modulation befinden. Des Weiteren ist zu erkennen, dass der Verlauf dieser Winkel starke Wellen aufweist und über den gesamten Verlauf viele Ausreisser entstehen.

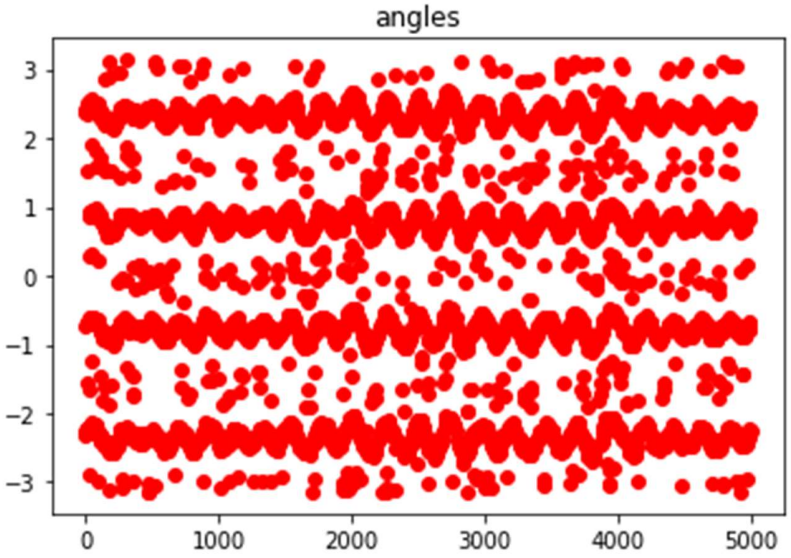


Abbildung 13: Winkel vom alten PLL

Der PLL der erhaltenen Arbeit wurde verbessert und das Ergebnis ist in Abbildung 14 dargestellt.
Die Wellen konnten sehr stark reduziert werden. Die vielen Ausreisser sind jedoch weiterhin vorhanden und muss genauer untersucht werden. Der verbesserte PLL ist im Kapitel 4.5 beschrieben.

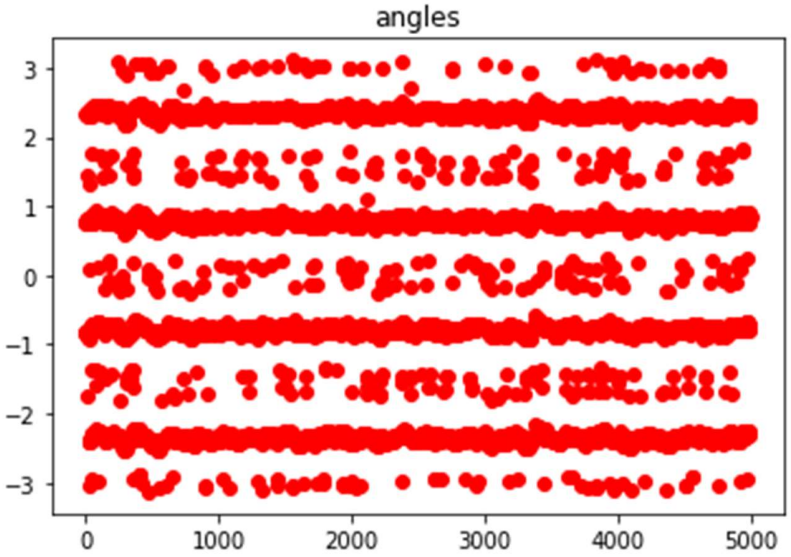


Abbildung 14 Winkel vom verbesserten PLL

5.2 Antennen

Weil die Präambel nicht immer gut erkannt wurde und viele Versuche Instabilitäten aufwiesen, werden die Antennen genauer untersucht und verglichen.

Als Antennen stehen die GSM 3G 4G LimeSDR mini Antennen von Lime Microsystems und die SDR smart Antenne von NooElec zur Verfügung.

Abbildung 15 LimeSDR mini antenna

Abbildung 16 NooElec SDR smart antenna

5.2.1 Rauschen

Zu Beginn wird das Rauschen der Antennen untersucht. Es wird erwartet, dass die Werte sich nahe an dem Ursprung der komplexen Ebene befinden. GNU Radio empfängt die Signale von Lime RX und schreibt die RRC gefilterten Signale in einen Socket Server.

Das Python File verbindet sich mit dem Socket Server von GNU Radio und liest das empfangene Signal aus dem Socket heraus. Die gelesenen Daten werden in Abbildung 17 dargestellt.

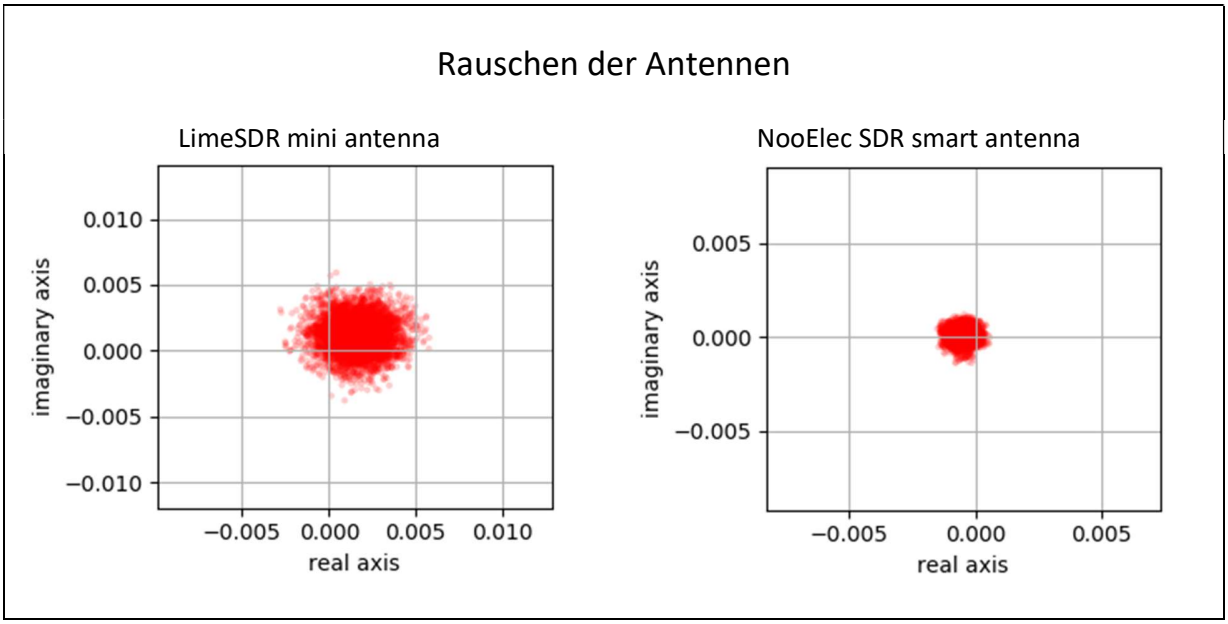


Abbildung 17 Rauschen der Antennen

Es ist sehr gut zu erkennen, dass das Rauschen von dem NooElec viel schwächer ist und somit das Signal mit weniger Störungen empfängt.

5.2.2 Distanztest

Um die korrekte Distanz der Antennen zu ermitteln, wird kontinuierlich ein QPSK moduliertes Signal gesendet und von einem zweiten LimeSDR mini empfangen. Der Empfänger wird bei unterschiedlichen Distanzen aufgestellt und das Gain des Senders variiert.

Bei jedem Testversuch werden 5120 Symbole empfangen und angezeigt. Als Antennen sind die Lime SDR mini Antennen im Einsatz.

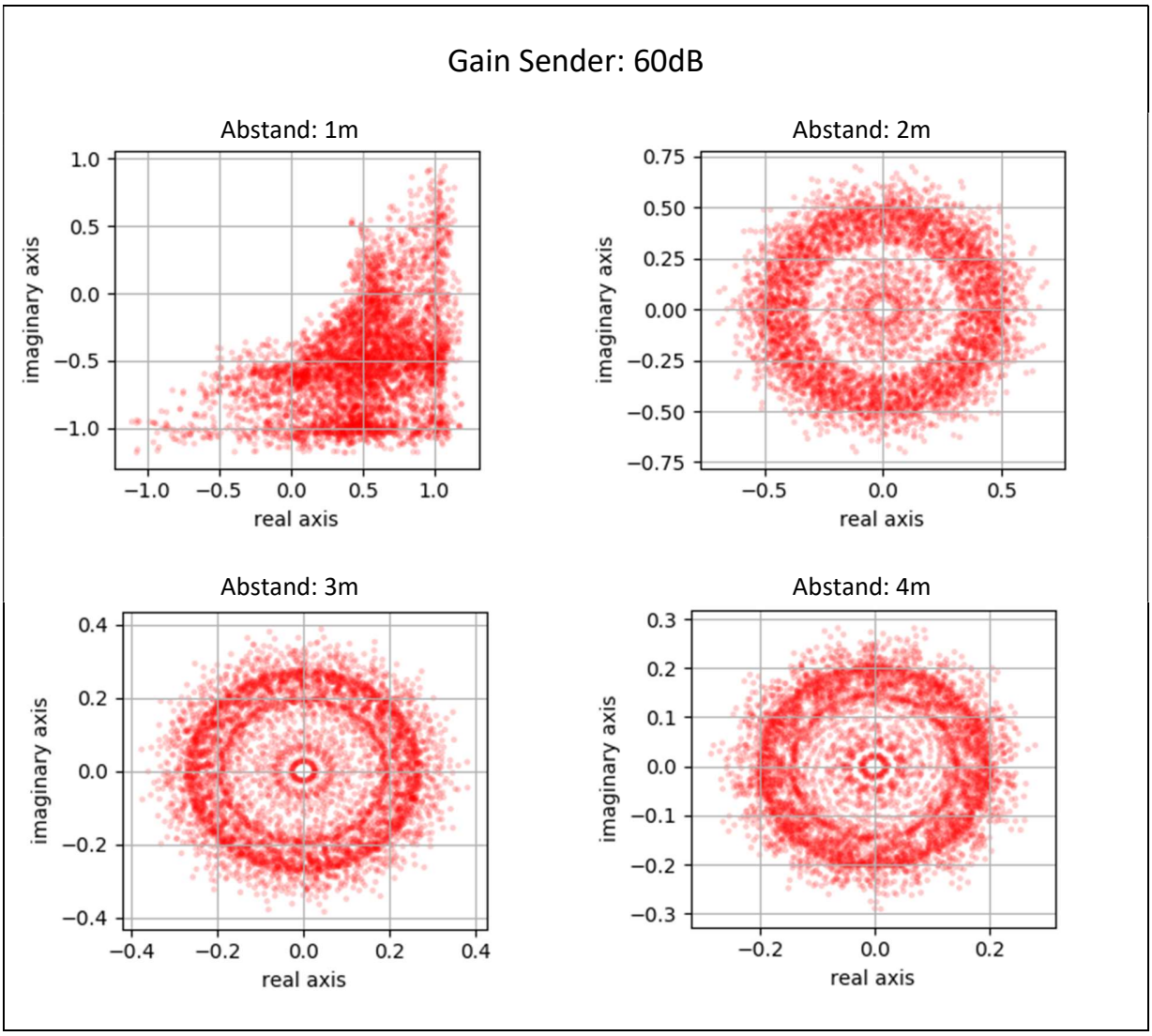


Abbildung 18 Distanztest TX Gain 60dB

Es ist ersichtlich, dass die Distanz von einem Meter nicht geeignet ist bei einem Sender Gain von 60dB. Bei zwei Meter ist das Signal deutlich besser. Wird die Distanz erhöht, zeigt es weiterhin ein gutes Signal mit abnehmender Stärke.

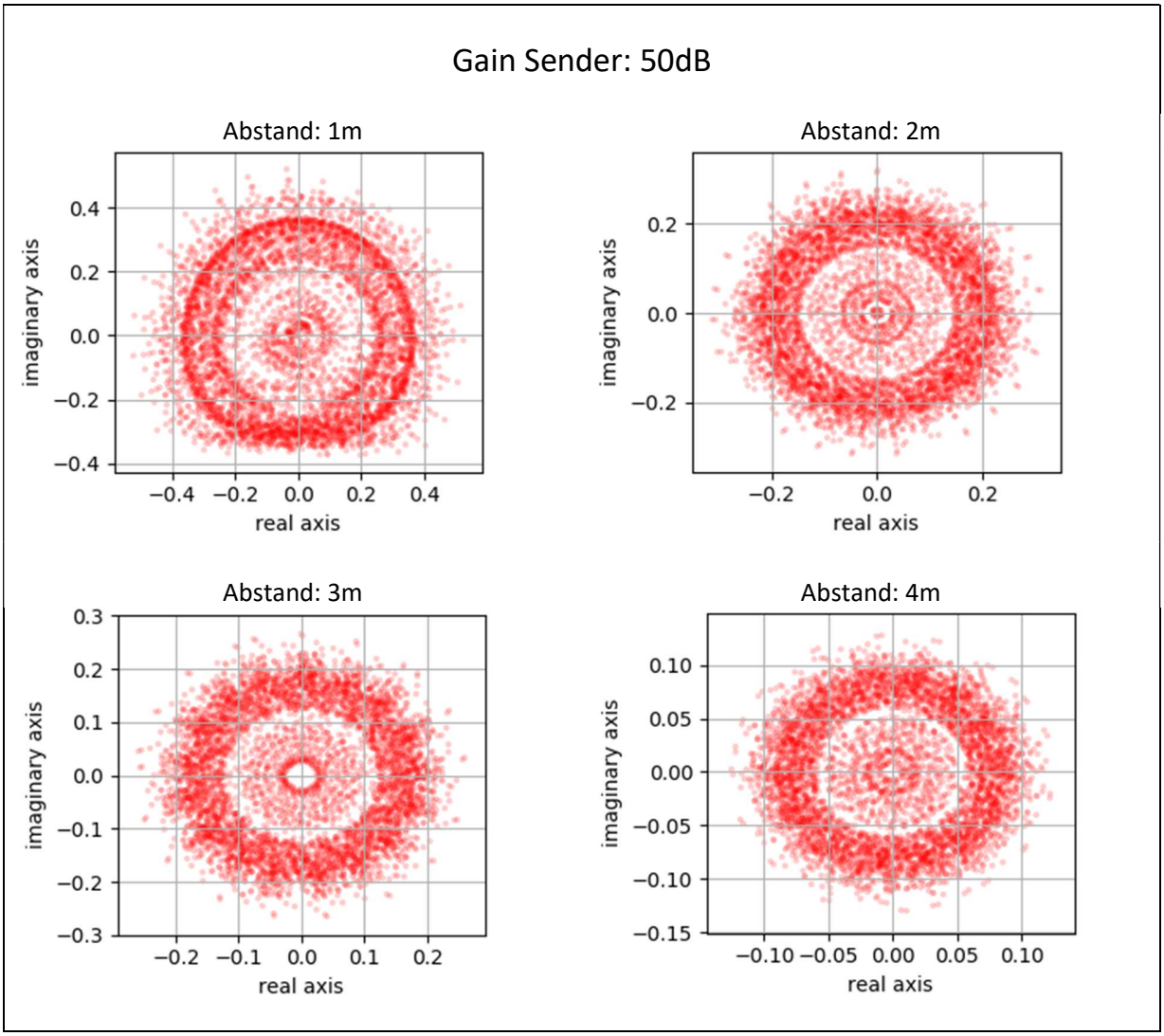


Abbildung 19 Distanztest TX Gain 50dB

Wird das Gain von 60dB auf 50dB reduziert, zeigt sich bei einer Distanz von einem Meter ein deutlich besseres Signal. Mit der Erhöhung der Distanz, wird weiterhin ein gutes Signal mit abnehmender Stärke empfangen.

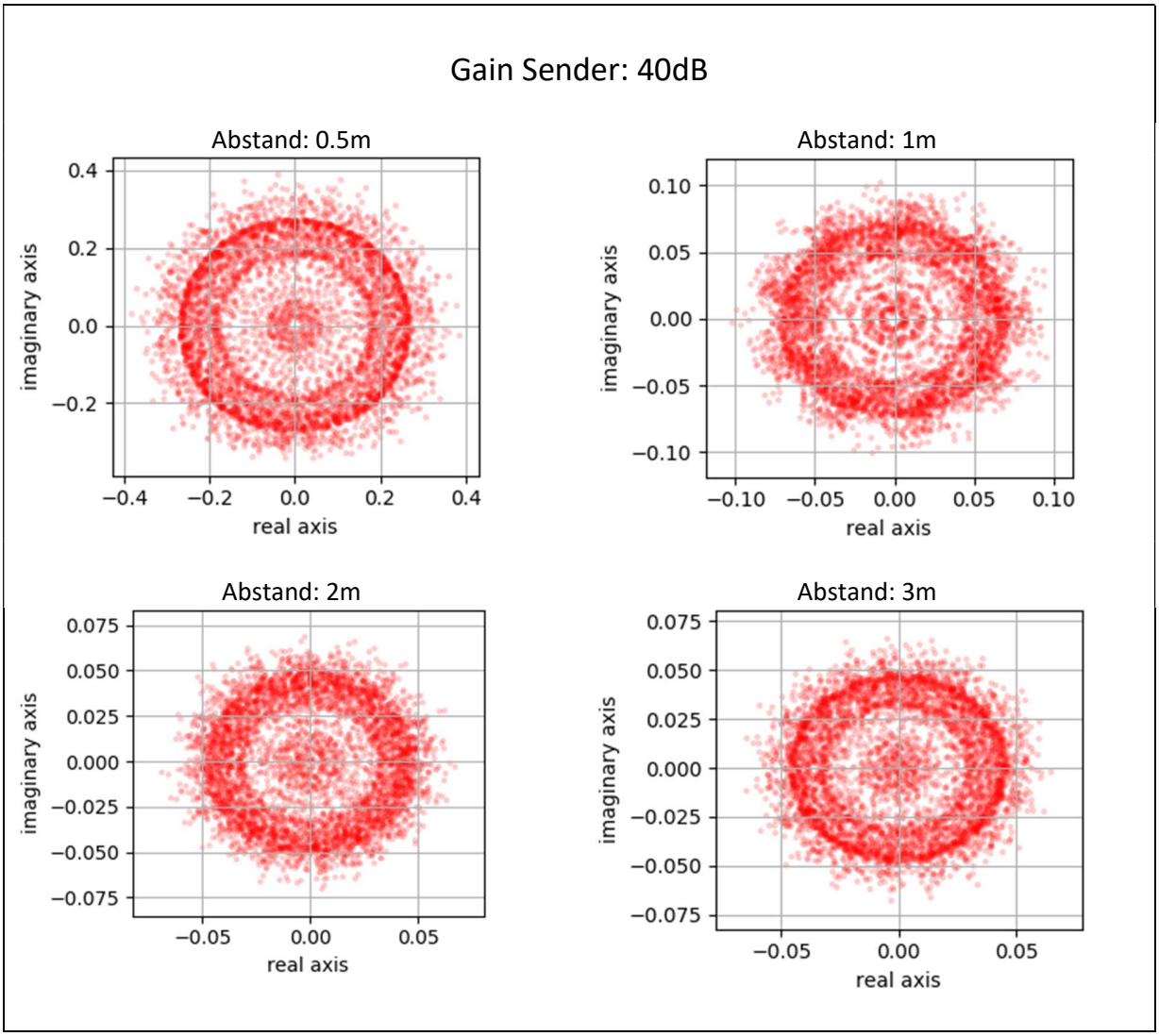


Abbildung 20 Distanztest TX Gain 40dB

Wird das Gain des Senders auf 40dB reduziert, kann die Distanz auf einen halben Meter verkürzt werden. Die Stärke des Signals nimmt jedoch mit der Erhöhung der Distanz stark ab.

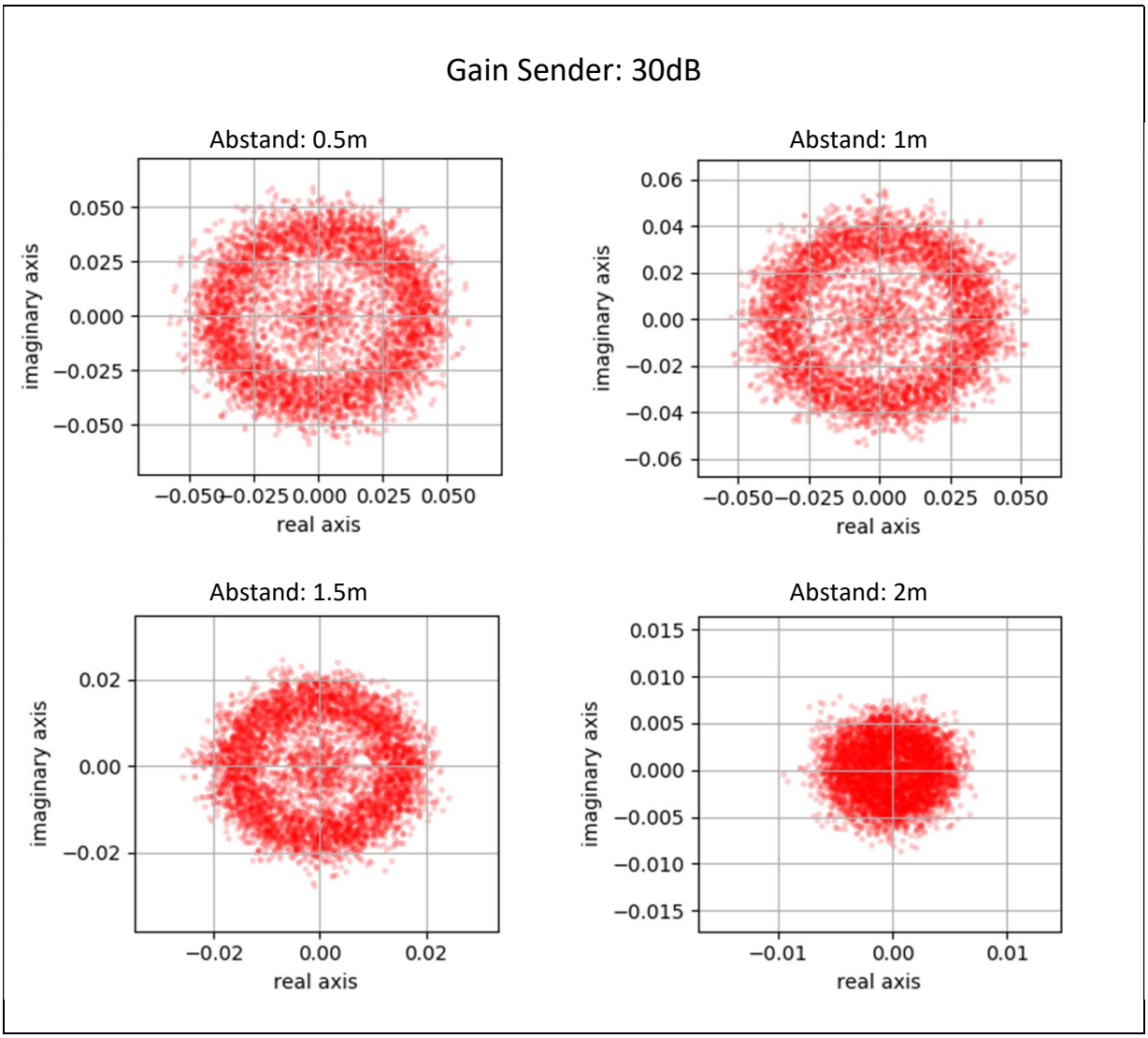


Abbildung 21 Distanztest TX Gain 30dB

Mit einem Sender Gain von 30dB werden nur noch schwache Signale empfangen und bei einer Distanz ab zwei Meter zeigt es das annähernd identische rausch Bild.

Aufgrund des Distanztest werden folgende Sender Gain empfohlen:

Tabelle 2 Distanz und Gain Empfehlung

Distanz [m]	Sender Gain [dB]
2 – 4	60
1 – 2	50
0.5 – 1	40

5.2.3 Vergleich

Um den Einfluss der Antenne zu untersuchen, wird ein Distanztest bei 60dB mit der NooElec SDR smart Antenne wiederholt.

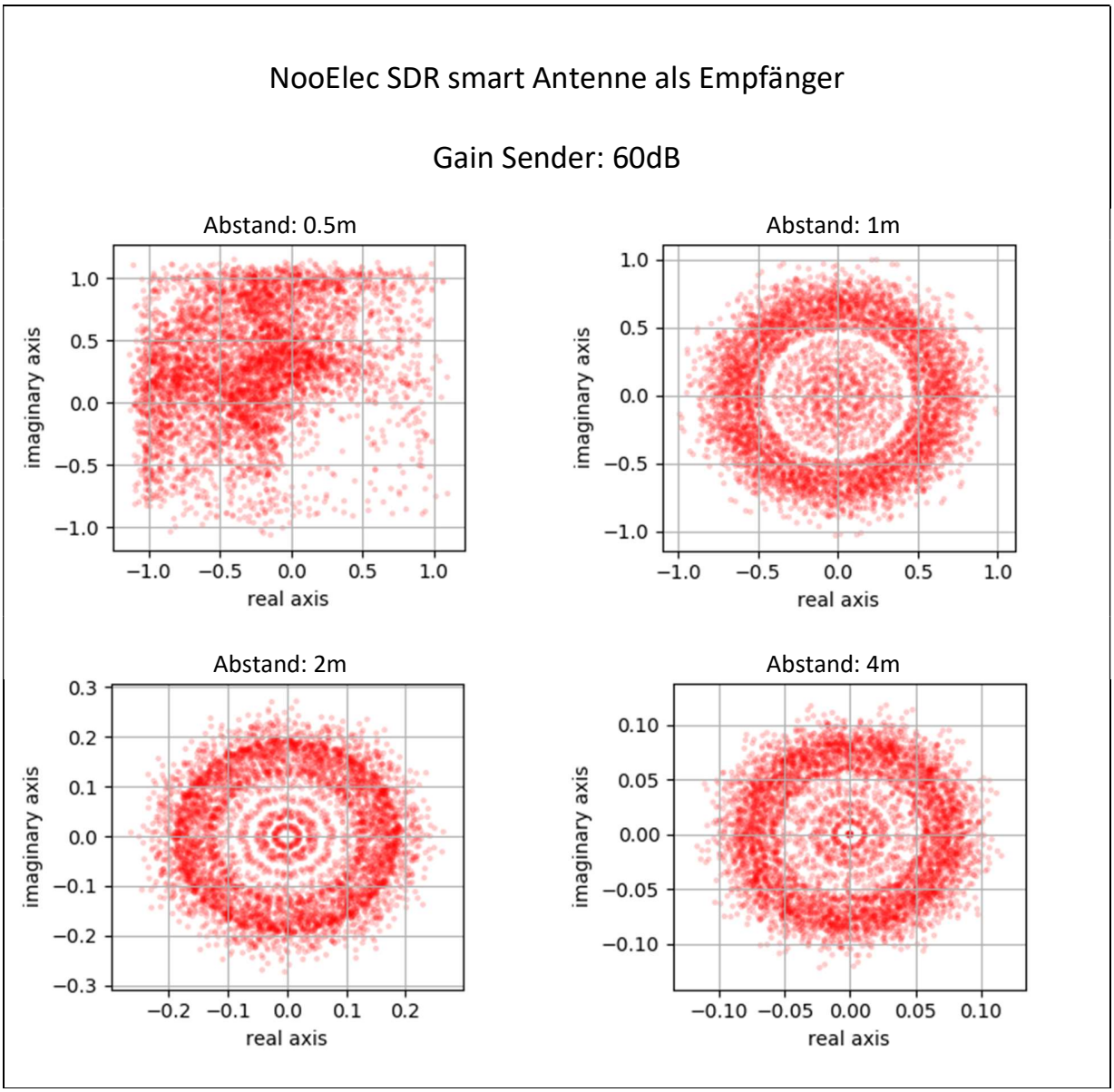


Abbildung 22 Distanztest NooElec SDR smart Empfänger

Die NooElec SDR smart Antenne konnte bei 60dB bereits ab einer Distanz von 1m ein gutes Signal empfangen. Ansonsten sind die Ergebnisse sehr ähnlich und es sind kaum Unterschiede zur LimeSDR mini Antenne zu erkennen.

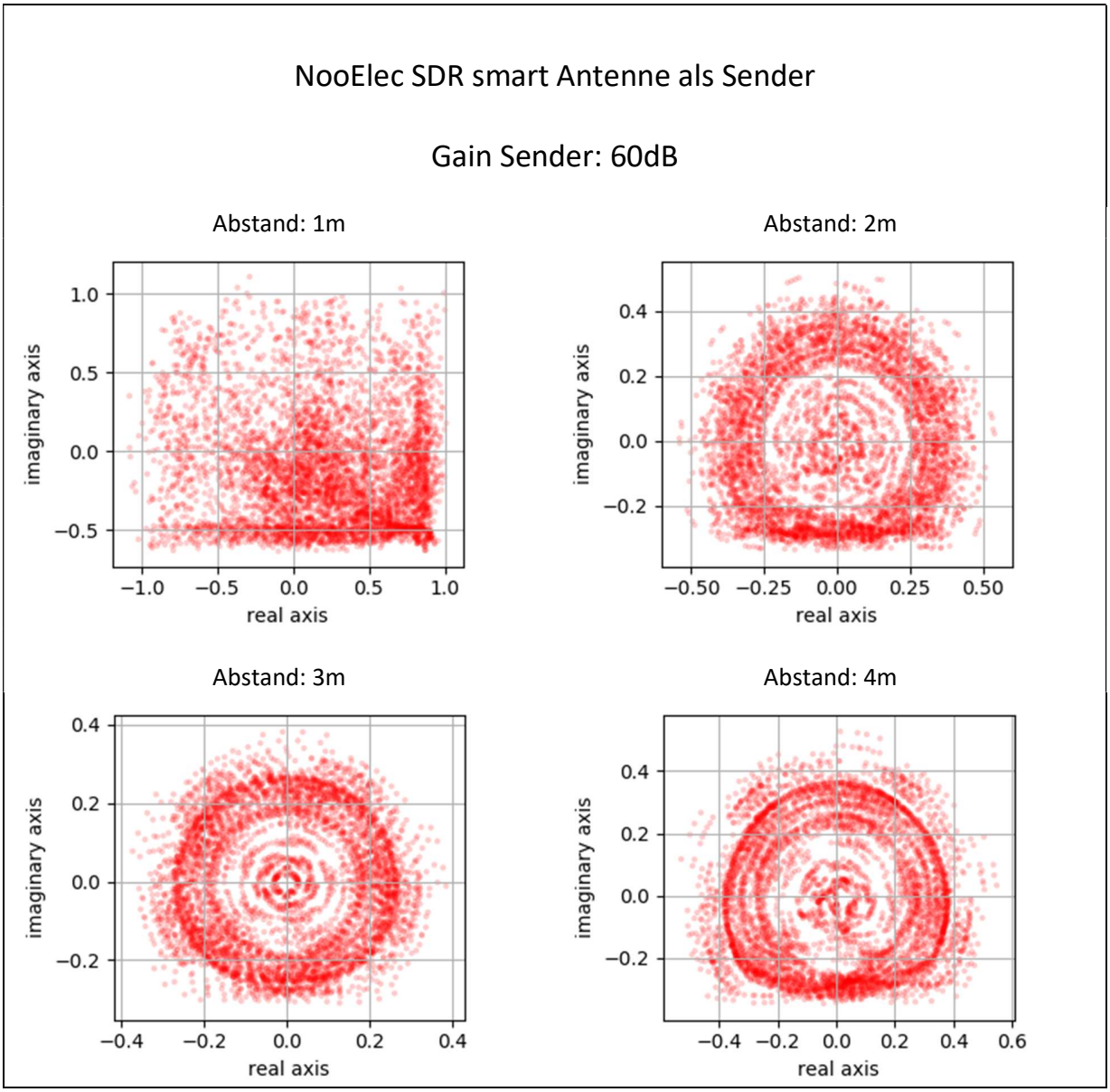


Abbildung 23 Distanztest NooElec SDR smart Sender

Werden die Signale mit dem Ergebnis in Abbildung 18 Distanztest TX Gain 60dB verglichen, zeigen sich Ähnlichkeiten.

Bei 2m kann ein relativ gutes Signal empfangen werden, jedoch ist ersichtlich das der untere Teil des Kreises eingedrückt ist. Dasselbe ist bei 4m zu sehen und kann nicht begründet werden.

Das Signal bei 3m sieht gut aus und die Antenne kann grundsätzlich als Sender eingesetzt werden, trotzdem zeigt das Ergebnis unerwünschtes Verhalten, welches genauer untersucht werden muss.

5.3 Präambel

Weil in einigen Experimenten das Detektieren der Präambel nicht funktioniert hat, wurde der Algorithmus für das detektieren der Präambel genauer untersucht.

In Abbildung 24 ist die Korrelation von rx (das empfangene Signal) und pus4 (Vergleichsmuster der Präambel) abgebildet. Das empfangene Signal ist ein kontinuierliches wiederholen der Präambel. Die Wiederholungen der Präambel ist sehr gut zu erkennen.

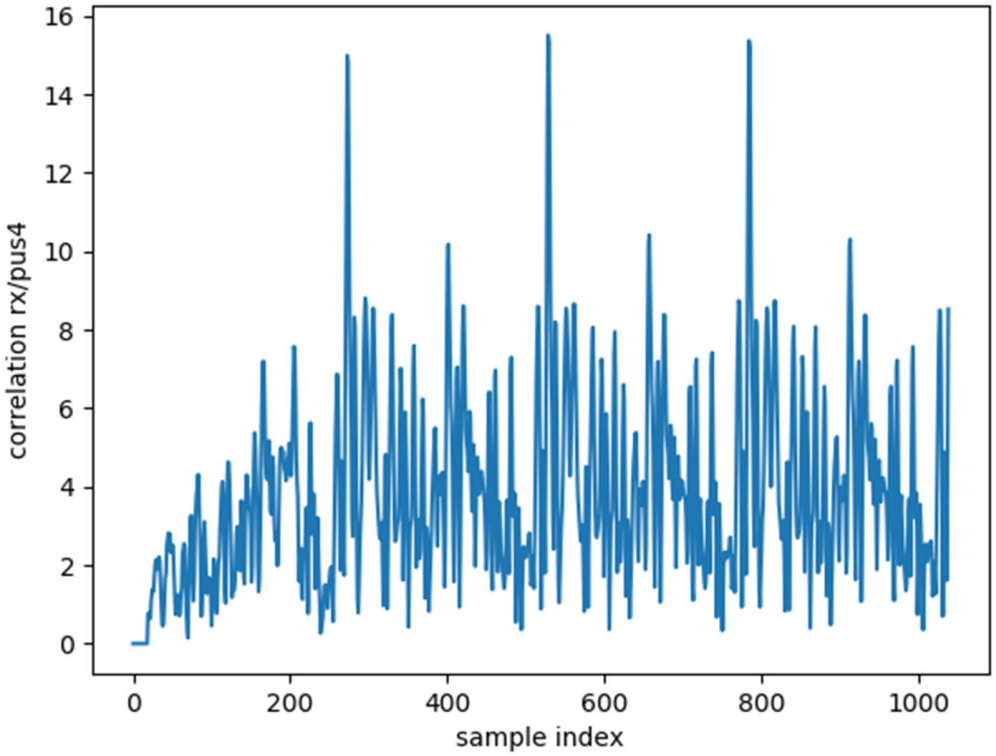


Abbildung 24 wiederholende Korrelation einer korrekten Präambel

Der Algorithmus sucht in Datenpaketen von 1295 Symbolen den höchsten Wert in der Korrelation mit pus4. Sofern keine weiteren Werte der Korrelation grösser als die Hälfte des höchsten Wertes sind, ist die Präambel gefunden. Die weiteren Werte der Korrelation sind sehr nahe an der Hälfte des höchsten Werts, was nicht erwünscht ist und somit keine saubere Korrelation vorliegt.

Abbildung 25 zeigt die Korrelationswerte einer einzigen Präambel. Der höchste Wert liegt bei knapp 16. Damit die Präambel erkannt wird, darf kein weiterer Wert grösser als die Hälfte von 16 sein. Dieser Wert wird mit der roten Linie, welche zuunterst ist, markiert. Weil sich mehrere Werte über dieser Linie befinden, wird die Präambel nicht erkannt.

Die Grafik wird mit der gelben Linie in der Mitte und der grünen Line zuoberst ergänzt. Die gelbe Linie zeigt die Grenze von 0,6 * Maximalwert und die grüne Linie zeigt die Grenze von 0,7 * Maximalwert.

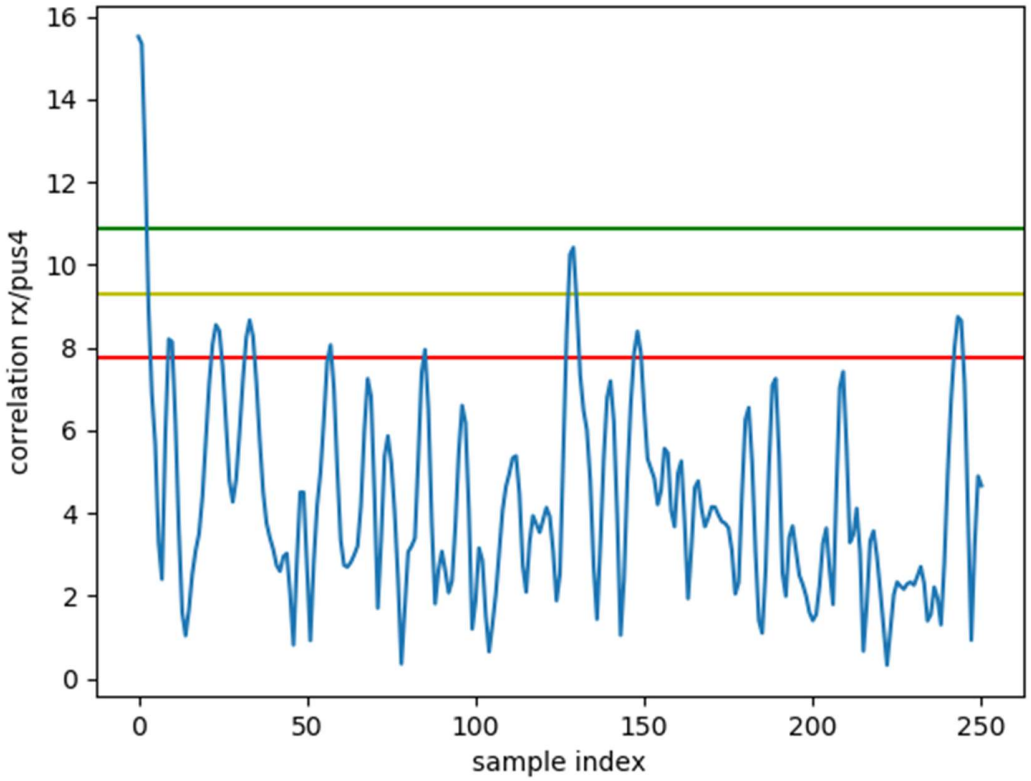


Abbildung 25 Korrelation von der Länge einer Präambel

Die Erweiterung der Grenze auf 0,6 oder 0,7 des Maximalwerts vereinfacht das Erkennen der Präambel, jedoch zeigt dies, dass das empfangene Signal Störungen enthält und verbessert nicht die Qualität der Verbindung. Wird die Grenze zu hoch gesetzt, kann ein Signal fälschlicherweise als Präambel interpretiert werden. Es wird nicht empfohlen eine höhere Grenze als 0,6 zu wählen.

Durch wiederholende Versuche wurde entdeckt, dass wie in Abbildung 26 gezeigt, eine sehr gute Korrelation möglich ist.

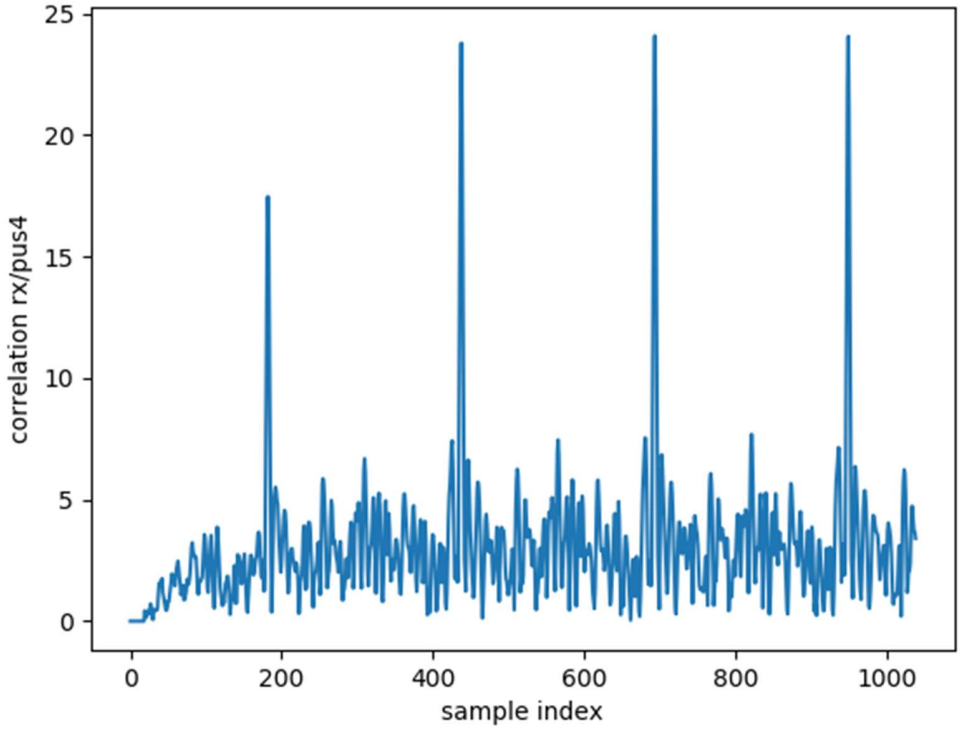


Abbildung 26 sehr gute Korrelation einer Präambel

Was die unerwünschten Störrungen erzeugt, konnte nicht herausgefunden werden. Es wurde jedoch beobachtet, dass die ersten Versuche positiv verliefen und die Störrungen nach längerem Einsatz der Funkmodule auftraten. Die Untersuchung vom Einfluss der Wärmeentwicklung und Rauschen konnten keine Anhaltspunkte liefern.

5.4 Signalverarbeitung

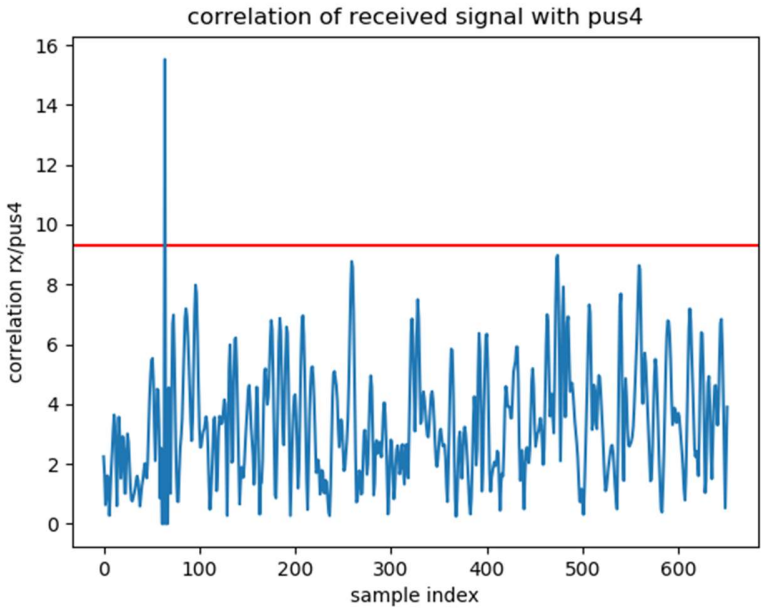
Pakete können hin und her gesendet werden, jedoch sind viele empfangene Pakete fehlerhaft.
Um die mögliche Ursache der Ausreisser und Instabilitäten einzugrenzen, wird im folgenden Abschnitt die Signalverarbeitung der Präambel eines Pakets untersucht. Als Vergleich wird eine Simulation über GNU Radio verwendet. Für weitere Vergleichsmöglichkeiten wird die mittlere quadratische Abweichung der Winkel berechnet. Die rote Linie liegt bei 60% des Maximalwerts und entspricht der Grenze für das Erkennen der Präambel.

Tabelle 3 Testparameter

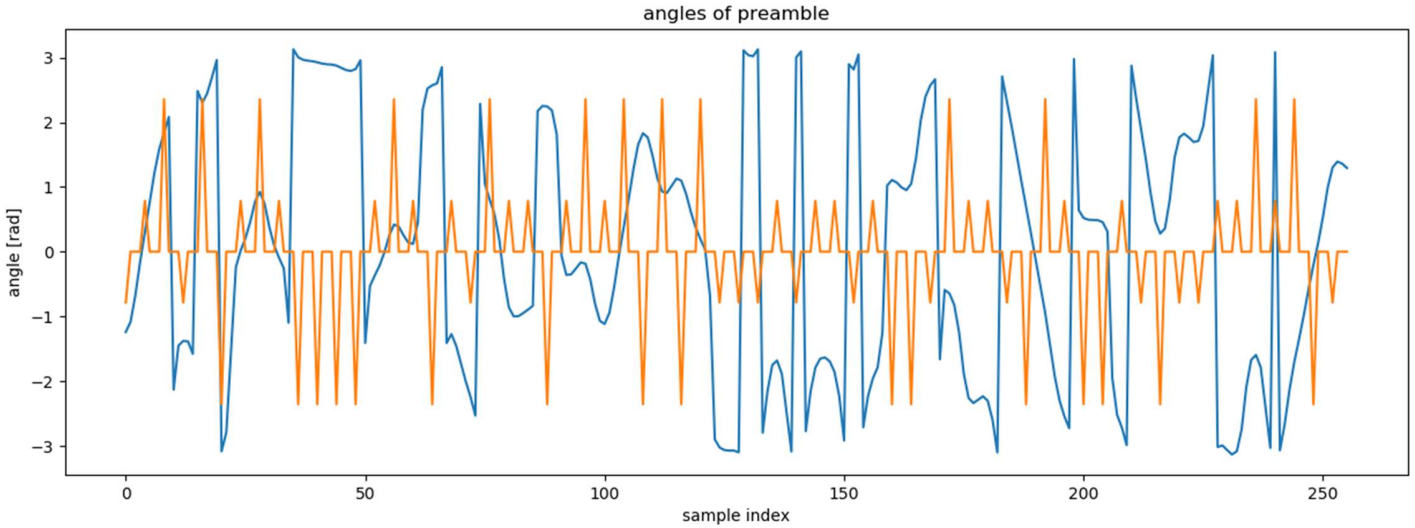
Abstand	2m
Sender Gain	60dB
Sender Antenne	LimeSDR mini Antenne
Empfänger Gain	60dB
Empfänger Antenne	NooElec SDR smart Antenne
Präambel Grenze	0,6 (rote Linie)

Testergebnis 1

mse of preamble angle [rad]: 1.53437579155
mse of received message angle [rad]: 2.24982825772

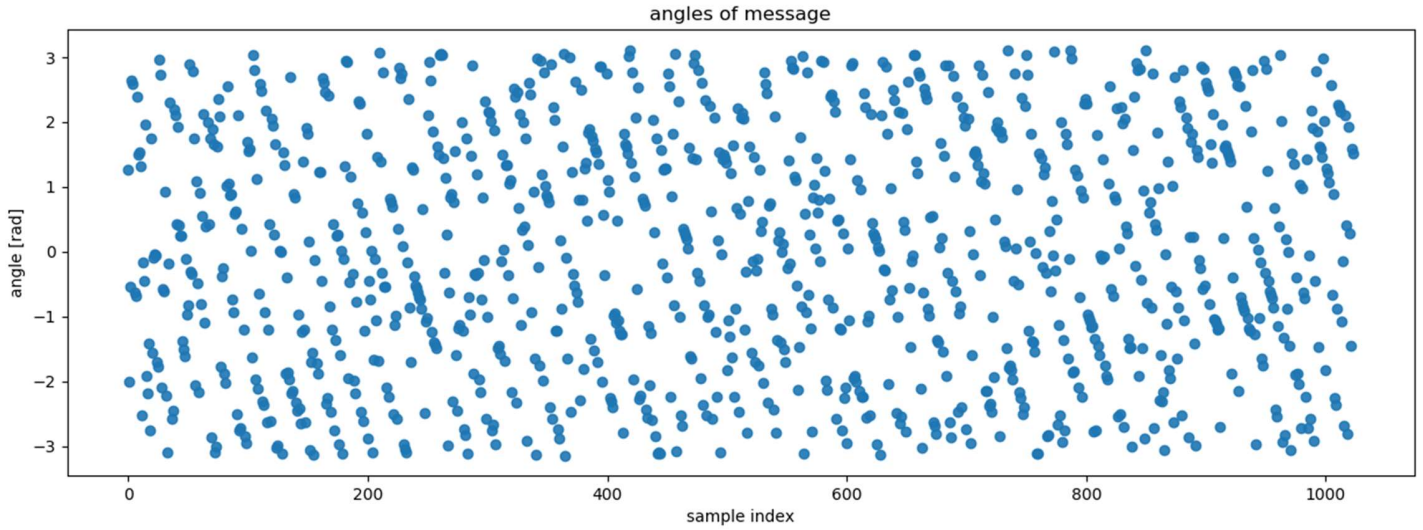


60% des Maximalwerts



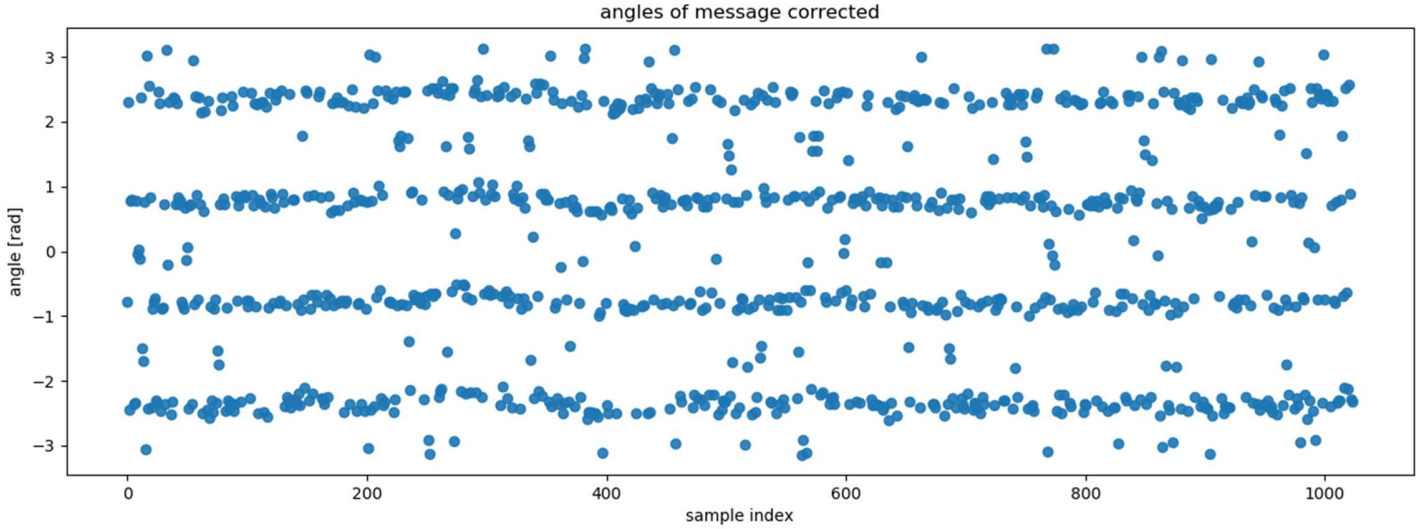
orange: pus4
blau: rx

Tobias Mailänder



Erhaltene Symbole nach dem RRC Filter und Time Offset Estimation.

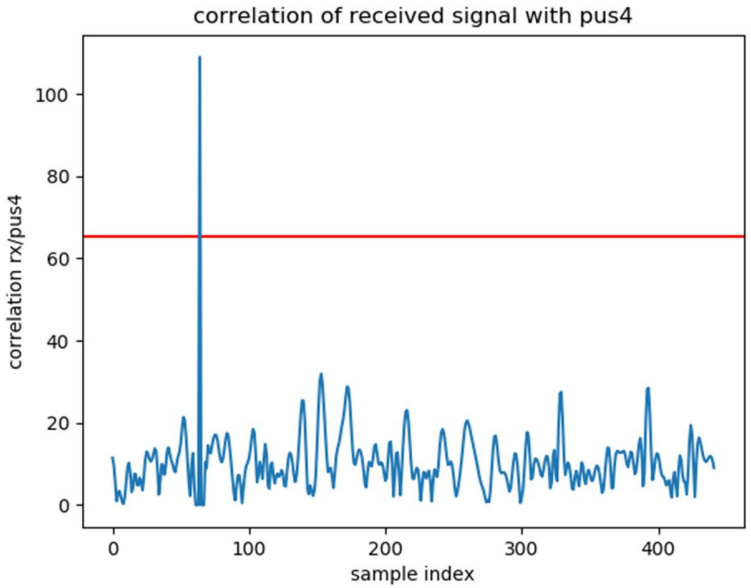
Es wird jeder vierte Sample dargestellt.



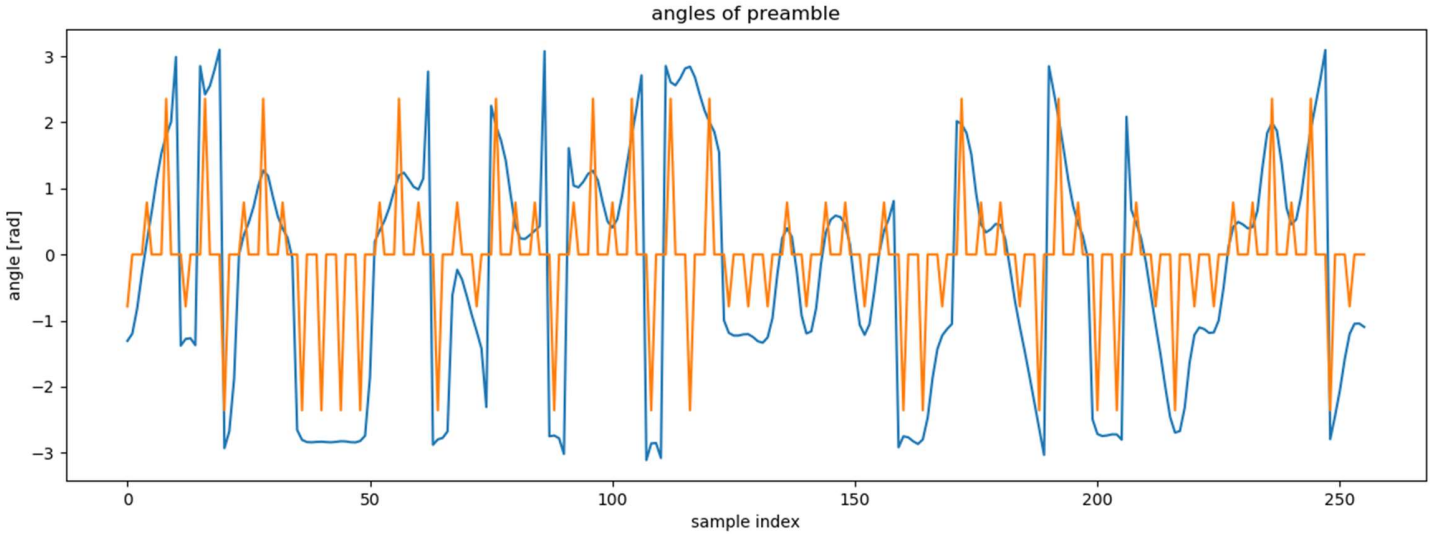
Erhaltene Symbole nach der Korrektur in Python.

Testergebnis 2

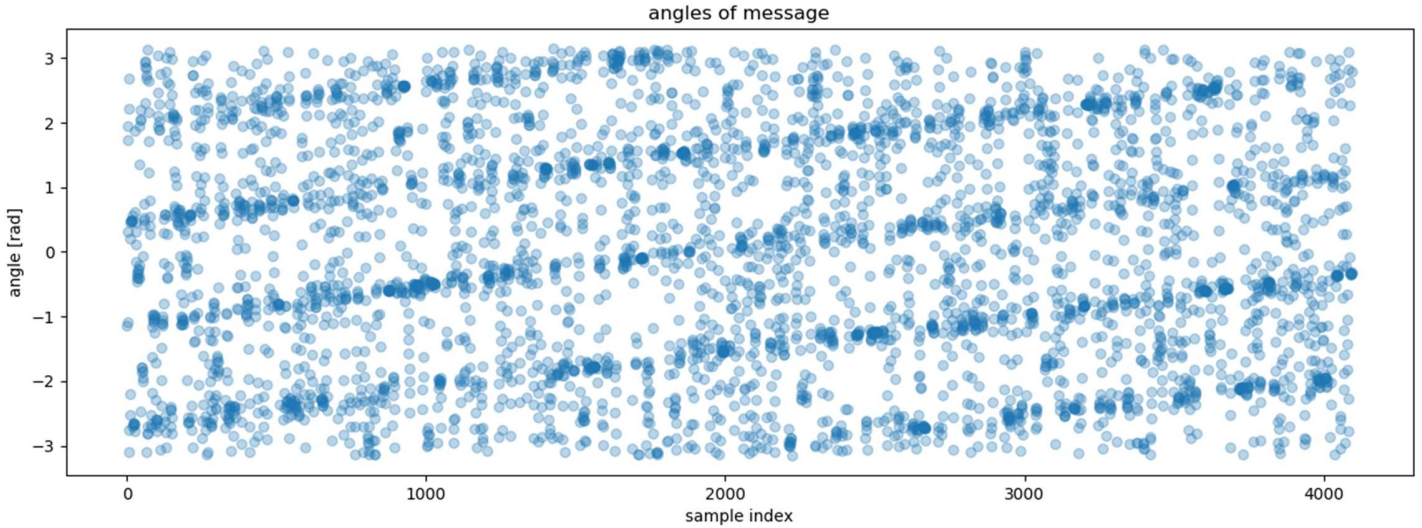
mse of preamble angle [rad]: 0.815653588964
mse of received message angle [rad]: 1.03462860522



60% des Maximalwerts

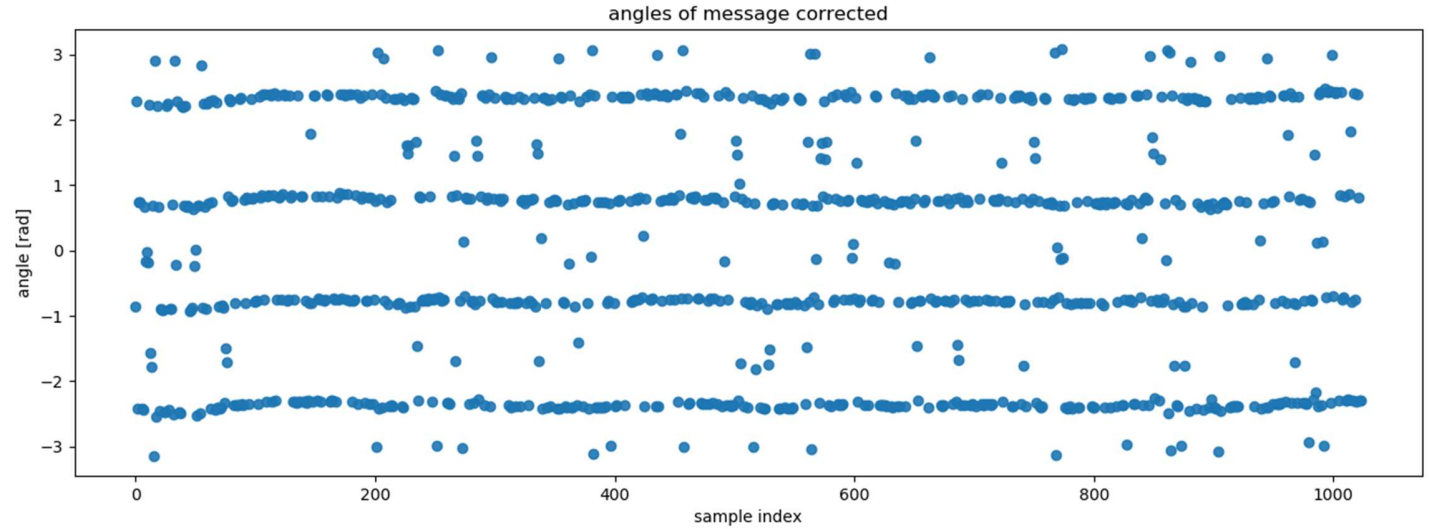


orange: pus4
blau: rx



Erhaltene Symbole nach dem RRC Filter.

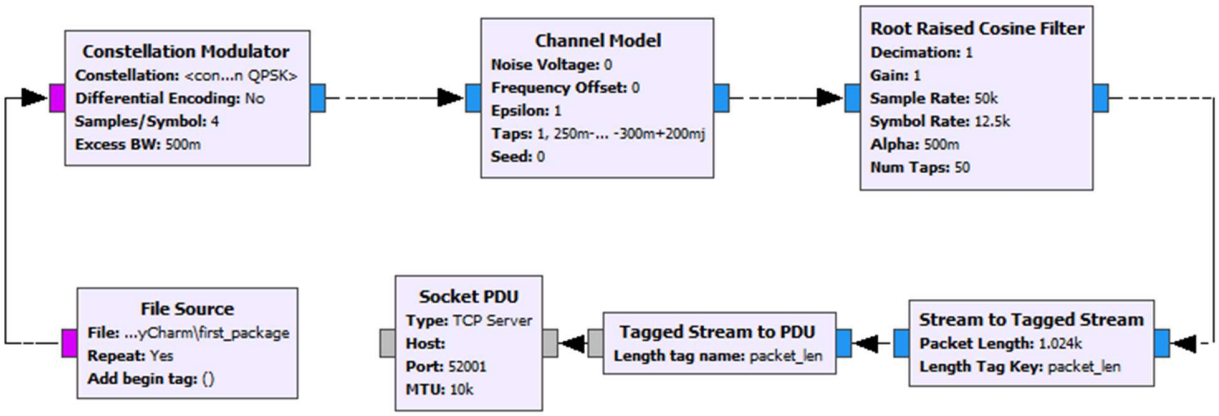
Es werden alle Samples dargestellt.



Erhaltene Symbole nach der Korrektur in Python.

5.4.1 Simulation

Um die Signalverarbeitung zu kontrollieren wird die Funkverbindung mit einem Channel Model simuliert und das erste Paket der Kommunikation gesendet. [6]



Mit Reglern kann die gewünschte Störung im Kanal eingestellt werden.

Simulation 1: Es werden keine Störrungen eingebaut und erwartet, dass die vier Winkel der QPSK Modulation gerade und ohne Ausreisser auftreten.

Simulation 2: Es wird ein Echo über Multichannel eingebaut, was zu einem Rauschen führt und die Winkel leicht streut.

Simulation 3: Zusätzlich zum Echo wird ein Frequency Offset hinzugefügt und erwartet, dass der Offset korrigiert wird und zu einem identischen Ergebnis führt wie in Simulation 2.

Simulation 4: Das Rauschen wird verstärkt und die Winkel werden noch mehr gestreut.

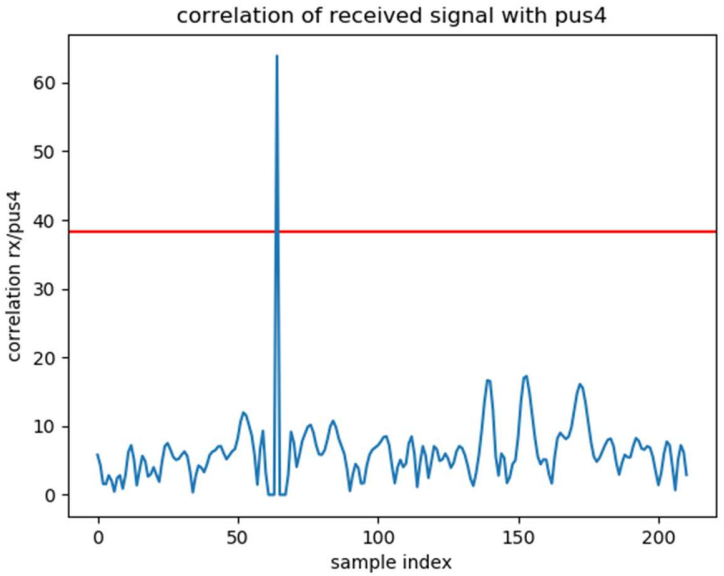
Simulation 1

Multichannel [1.0]

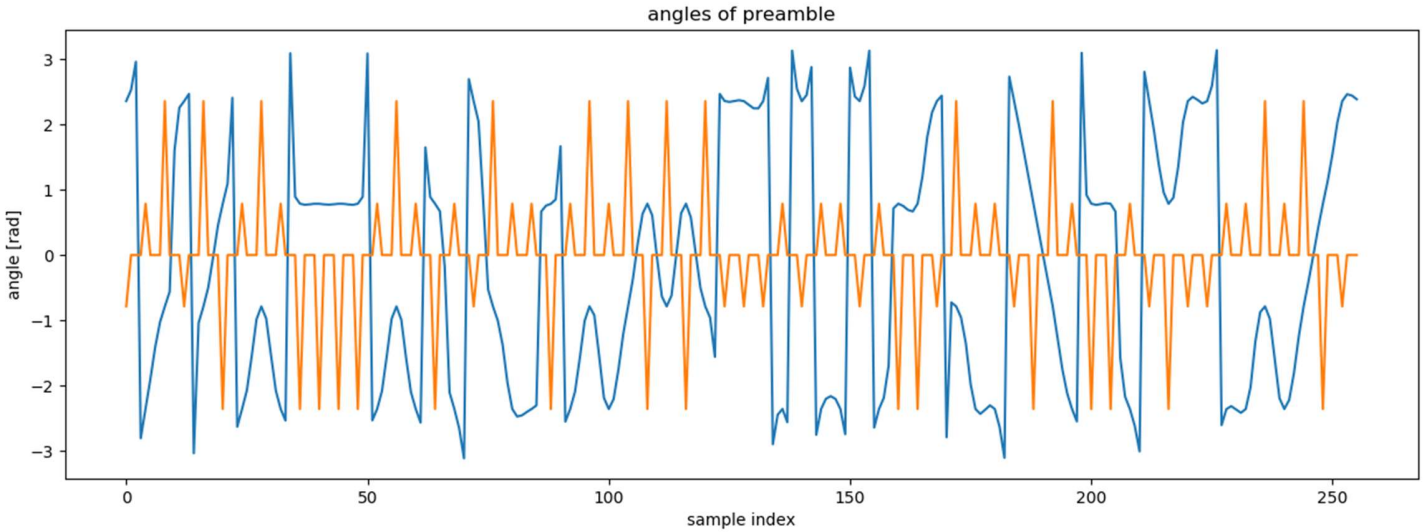
mse of preamble angle [rad]: 1.59107705388
mse of received message angle [rad]: 1.4066785842
mse of received message angle corrected [rad]: 0.820093651359

time offset: 0.0125
phase drift: 0.0
initial phase: 3.13924068261

wrong symbols in preamble: 0
wrong symbols in message: 0

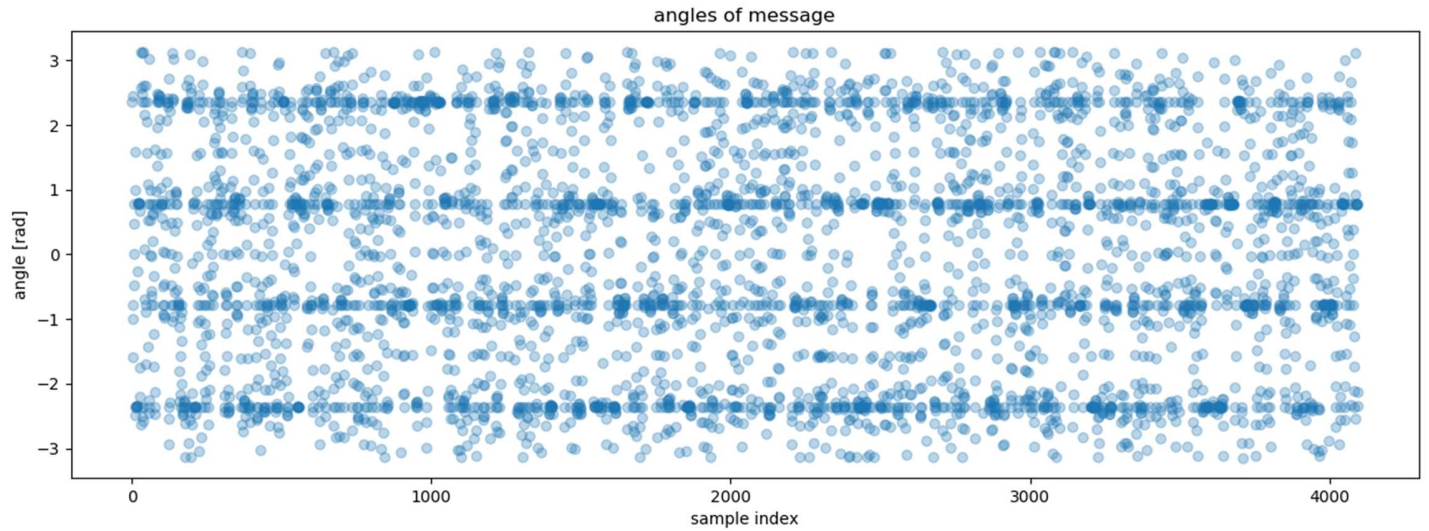


60% des Maximalwerts



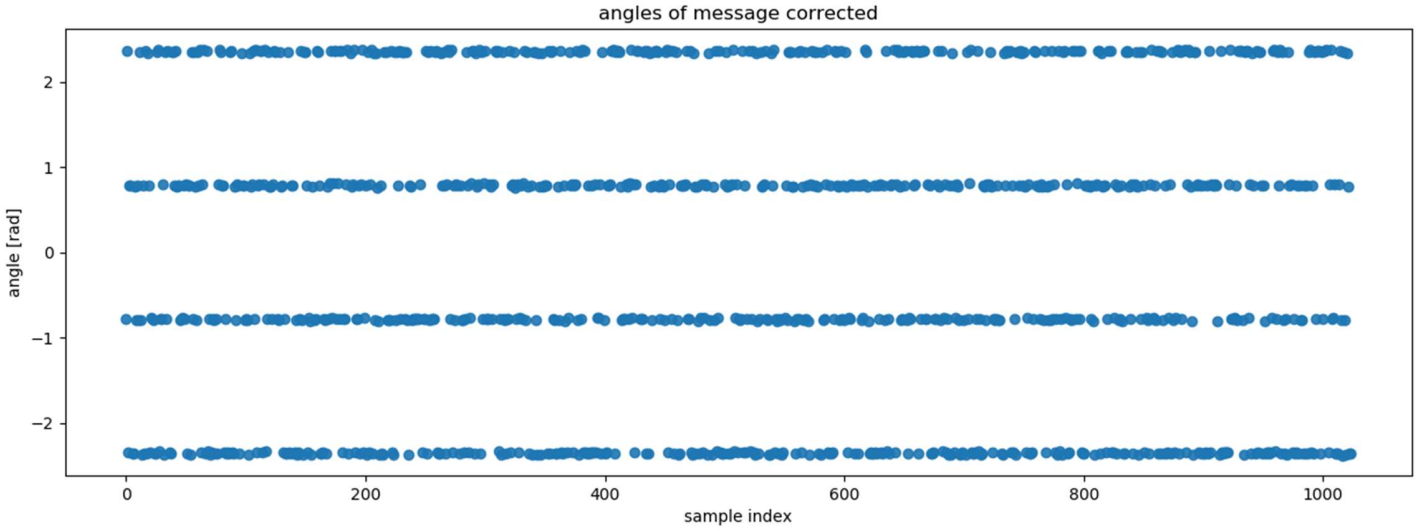
orange: pus4
blau: rx

Tobias Mailänder



Erhaltene Symbole nach dem RRC Filter.

Es werden alle Samples dargestellt.



Erhaltene Symbole nach der Korrektur in Python.

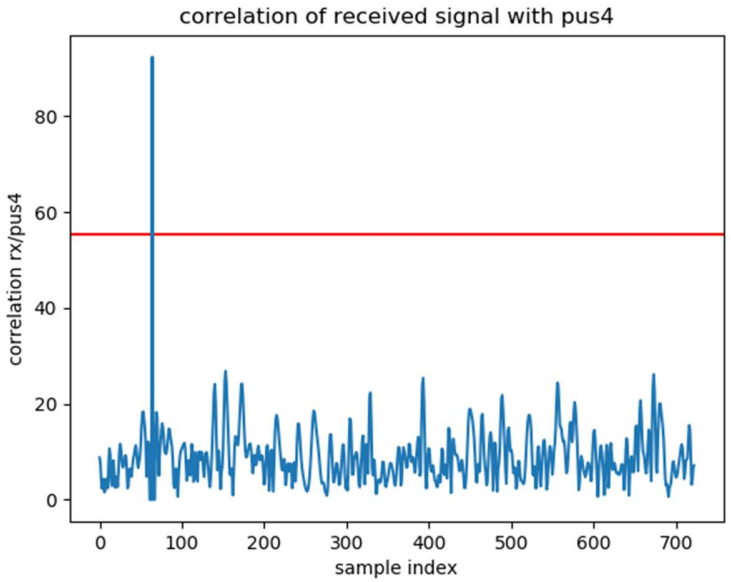
Simulation 2

Multichannel [1.0, 0.25-0.25j, 0.50 + 0.10j, -0.3 + 0.2j]

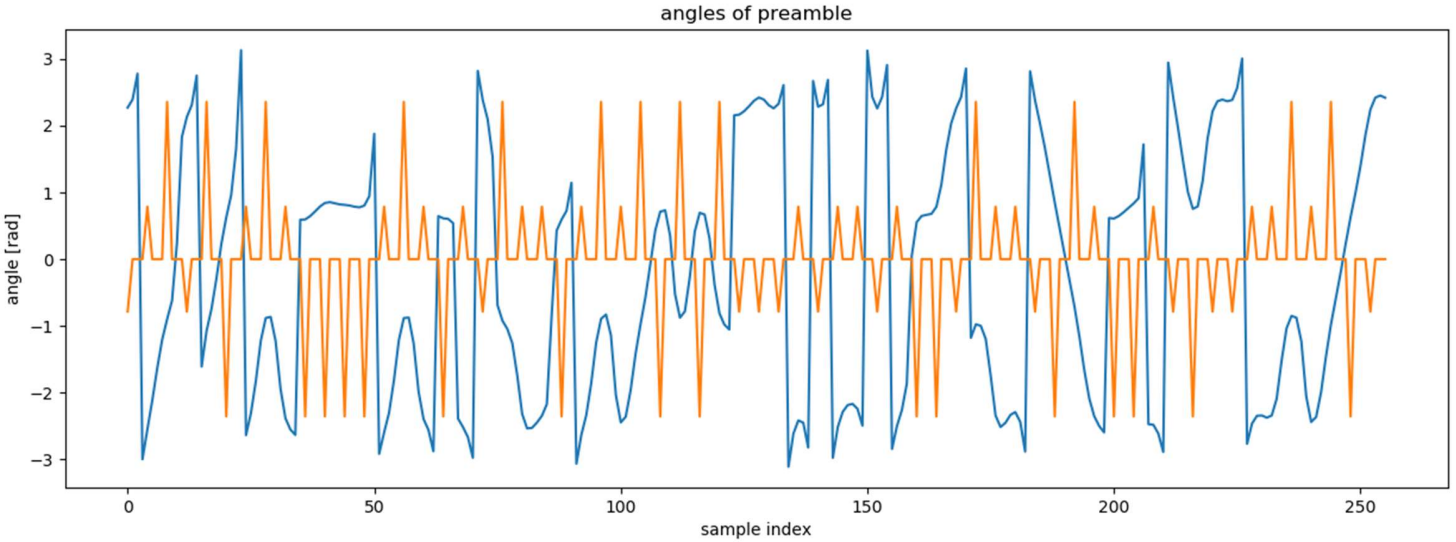
mse of preamble angle [rad]: 1.53002240352
mse of received message angle [rad]: 1.2004449255
mse of received message angle corrected [rad]: 1.20452077191

time offset: 0.0875
phase drift: 0.0
initial phase: 3.09158036872

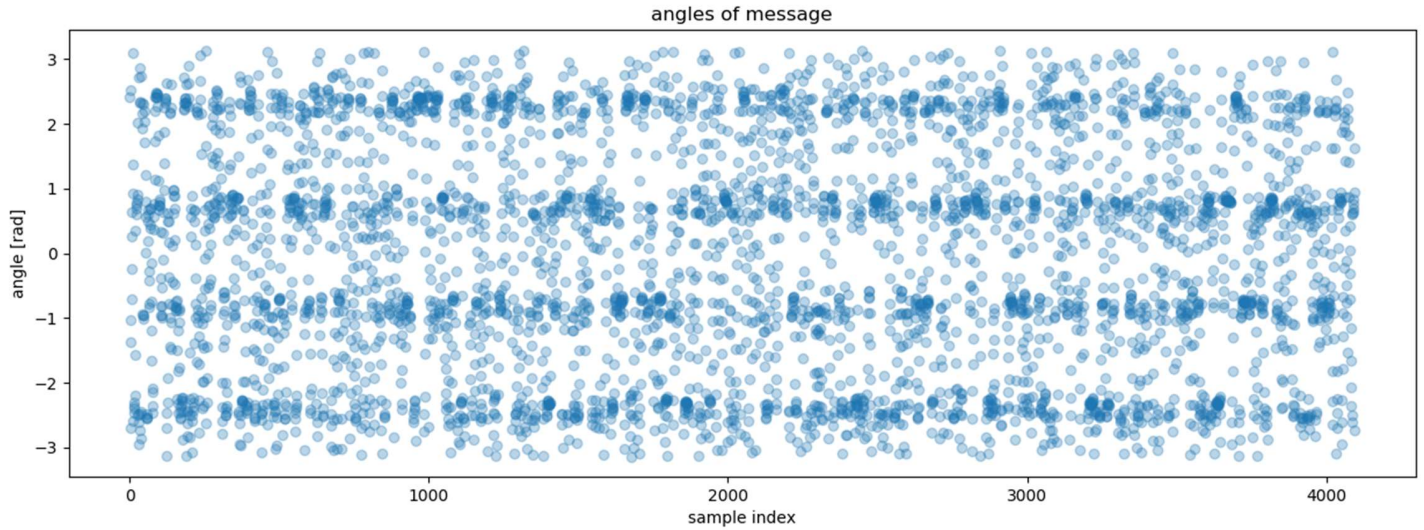
wrong symbols in preamble: 0
wrong symbols in message: 0



60% des Maximalwerts

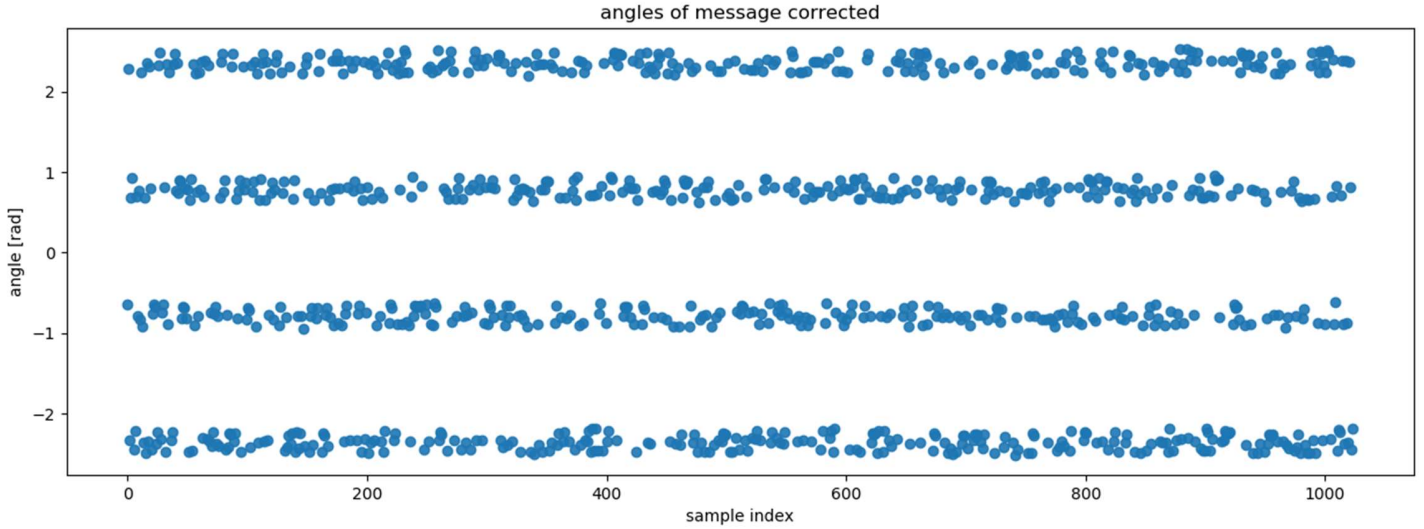


orange: pus4
blau: rx



Erhaltene Symbole nach dem RRC Filter.

Es werden alle Samples dargestellt.



Erhaltene Symbole nach der Korrektur in Python.

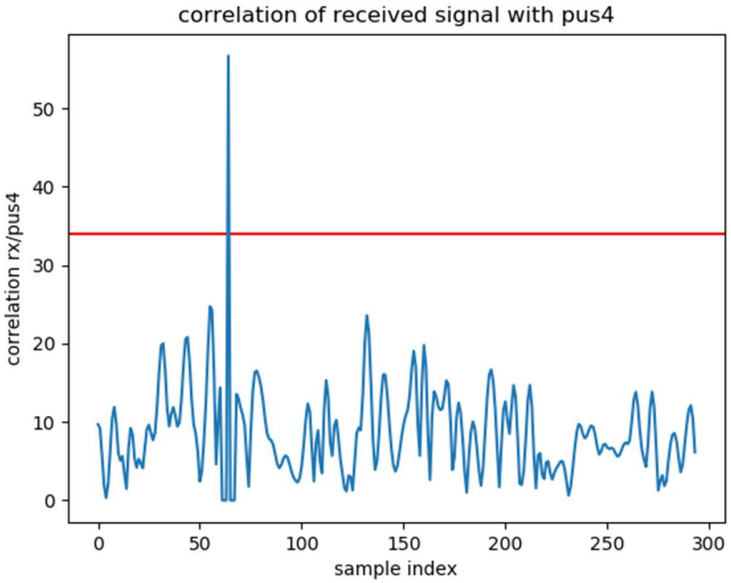
Simulation 3

Multichannel [1.0, 0.25-0.25j, 0.50 + 0.10j, -0.3 + 0.2j]
Frequency Offset: 0.0020

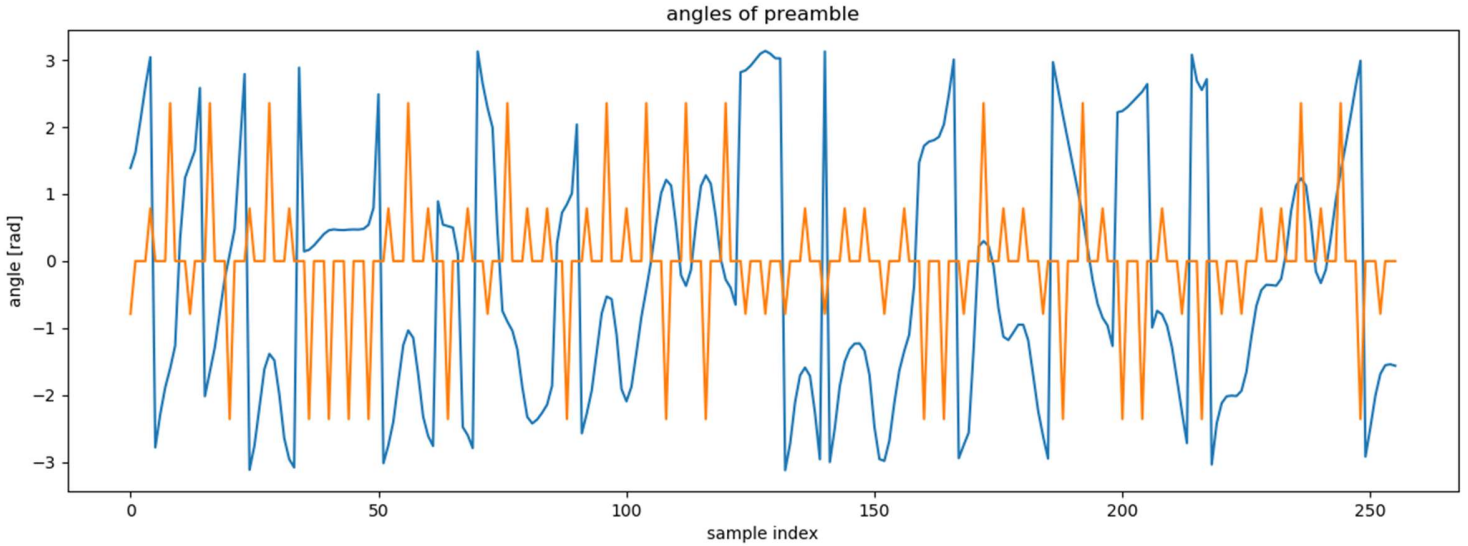
mse of preamble angle [rad]: 1.7780055346
mse of received message angle [rad]: 2.0539361409
mse of received message angle corrected [rad]: 1.19081758341

time offset: 0.05
phase drift: 0.05
initial phase: 2.21570040288

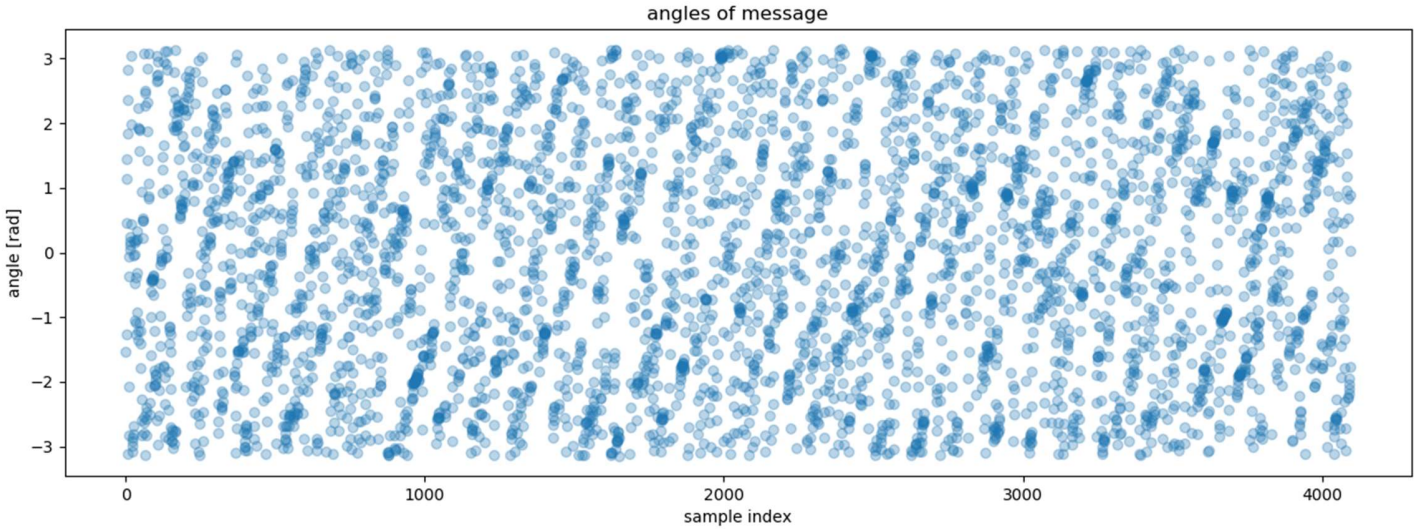
wrong symbols in preamble: 0
wrong symbols in message: 0



60% des Maximalwerts

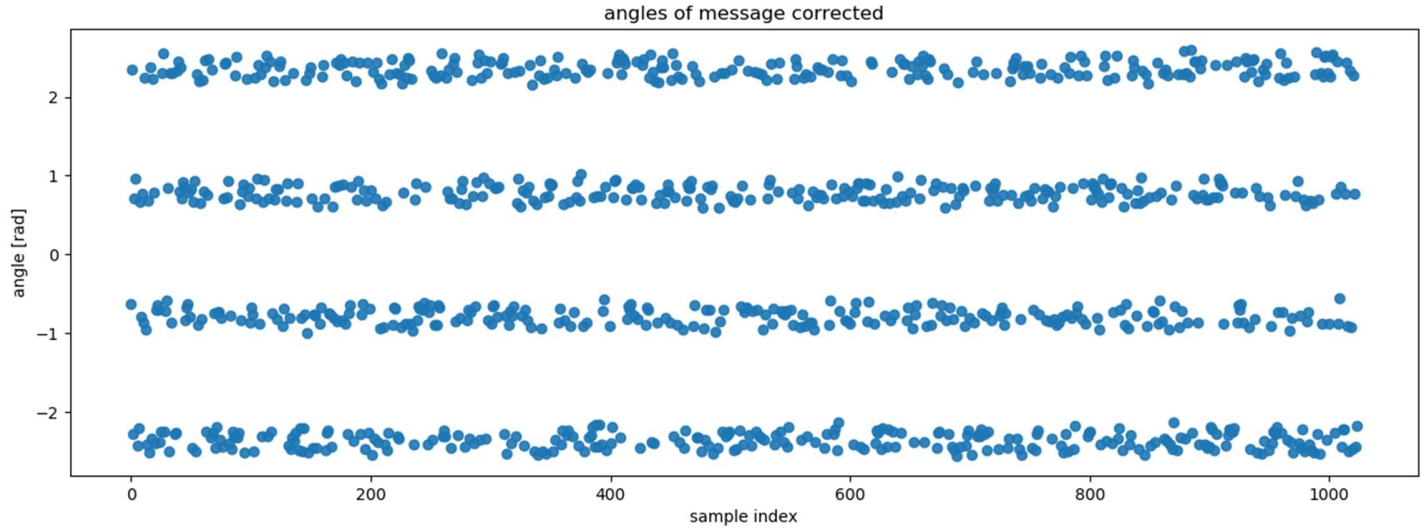


orange: pus4
blau: rx



Erhaltene Symbole nach dem RRC Filter.

Es werden alle Samples dargestellt.



Erhaltene Symbole nach der Korrektur in Python.

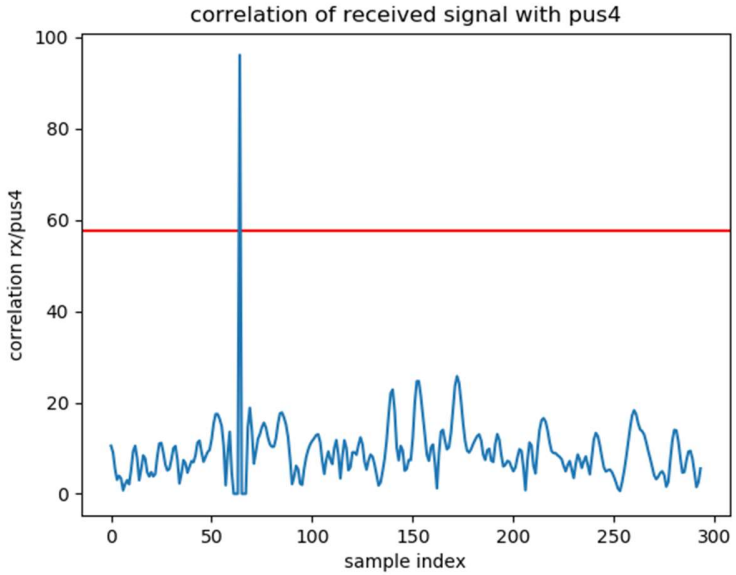
Simulation 4

Multichannel [1.0, 0.25-0.25j, 0.50 + 0.10j, -0.3 + 0.2j]
Noise Voltage: 0.5000

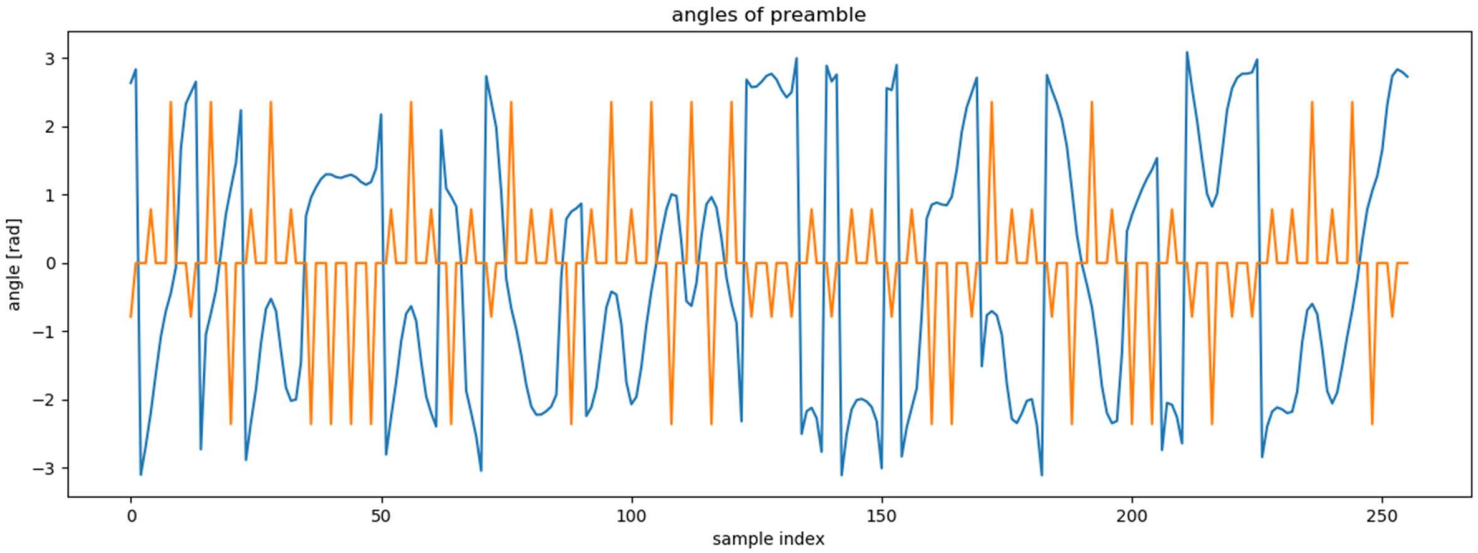
mse of preamble angle [rad]: 2.01513051941
mse of received message angle [rad]: 1.84622022397
mse of received message angle corrected [rad]: 1.522656498

time offset: 0.0125
phase drift: 0.0
initial phase: -2.90391108535

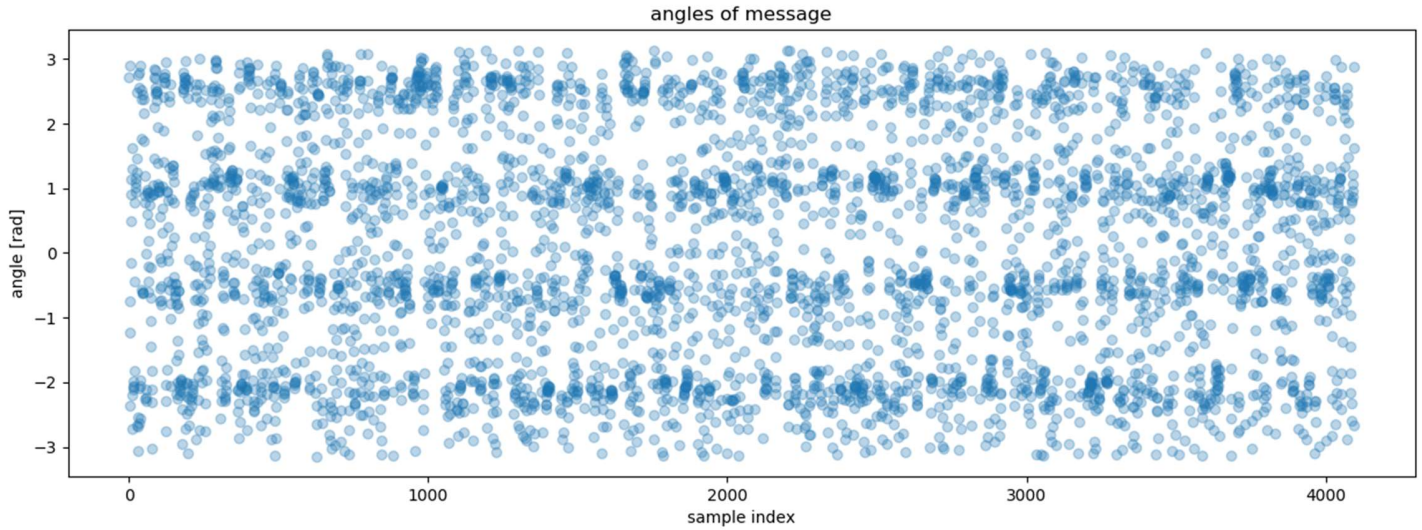
wrong symbols in preamble: 0
wrong symbols in message: 0



60% des Maximalwerts

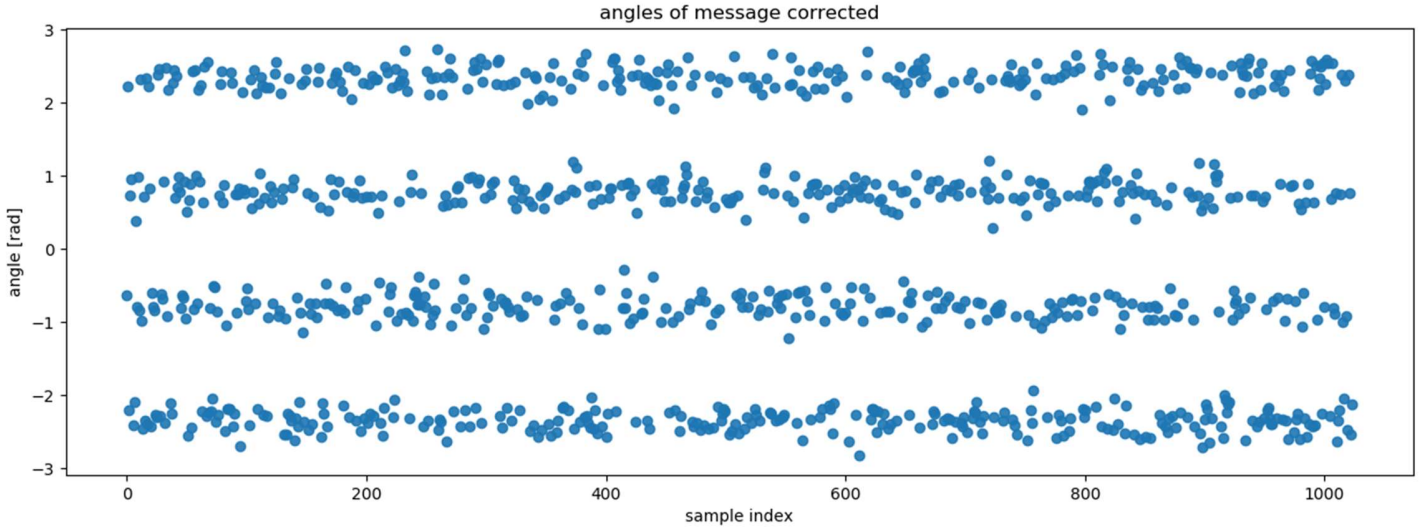


orange: pus4
blau: rx



Erhaltene Symbole nach dem RRC Filter.

Es werden alle Samples dargestellt.



Erhaltene Symbole nach der Korrektur in Python.

5.4.2 Auswertung

Beim ersten Test wird die Präambel knapp erkannt und die korrigierten Symbole zeigen ein kleines Rauschen mit mehreren Ausreissern.

Das zweite Testergebnis zeigt eine sehr gute Korrelation und die korrigierten Symbole verfügen kaum über ein Rauschen. Trotzdem sind mehrere Ausreisser klar ersichtlich.

Werden diese Testergebnisse mit den Simulationen verglichen, liegt der Hauptunterschied beim Auftreten der Ausreisser. Die erste Simulation zeigt das Ergebnis eines idealen Funkkanals. Bei den erhaltenen Symbolen sind weder Ausreisser noch ein Rauschen zu erkennen. In den weiteren Simulationen treten ebenfalls keine Ausreisser auf und müssen somit durch die Hardware erzeugt werden.

Der Fehler durch das Echo zeigt sich in der zweiten Simulation durch ein Rauschen. Das Rauschen kann nicht korrigiert werden und darf nicht zu stark sein.

Wenn in einer weiteren Simulation ein Frequency Offset hinzugefügt wird, ist die Korrelation der Präambel weniger deutlich und das Bild vor der Korrektur ändert sich stark. Im Vergleich zur zweiten Simulation, wird das Rauschen nach der Korrektur nicht stärker und der Offset kann korrigiert werden.

In der vierten Simulation wird das Rauschen erhöht und verglichen. Die Präambel wird sehr gut erkannt, jedoch sind die Winkel der Symbole nahe beieinander und können falsch interpretiert werden.

Zusammenfassend kann folgendes gesagt werden:

- Die Ausreisser werden nicht durch die Signalverarbeitung erzeugt.
- Bei einem starken Rauschen wird die Präambel gut erkannt. Sind die Winkel der Symbole durch das Rauschen zu nahe beieinander, können die Symbole falsch interpretiert werden.
- Gibt es Schwankungen in den Frequenzen, kann die Präambel weniger gut erkannt werden. Die entstehenden Fehler können in der Signalverarbeitung korrigiert werden.

6 Optimierung

Die Funkmodule können miteinander kommunizieren und Pakete werden hin und her gesendet. Jedoch kommt es öfters vor, dass ein Paket nicht fehlerfrei ist oder nicht erkannt wurde. Dies hat zufolge, dass Pakete sehr oft wiederholt gesendet werden. Um dies zu verbessern, wurde ein zyklischer Hamming Code entworfen, welcher im Idealfall die fehlerhaften Pakete korrigieren kann.

6.1 Zyklischer Hamming Code

Ein zyklischer Hamming Code berechnet mit einem Generatorpolynom ein Prüfwert und fügt ihn den Datenbits hinzu. Verwendet wird das Generatorpolynom $z^3 + z + 1$, welches aus 4 Datenbits ein Prüfwert von 3 Bits erzeugt. Zusammen ergibt dies ein Frame von 7 Bits. Diese 7 Bits werden als ein gesamtes Byte gesendet. Die Bitkapazität des Funkkanals wird nicht komplett genützt, jedoch wird die Umsetzung und Kontrolle vereinfacht. So entsteht aus einem Daten Byte zwei Frames und es müssen doppelt so viele Pakete gesendet werden, um die gesamte Datei zu übermitteln. [7]

Aufbau eines Frames:

Source	Daten	Prüfwert	Frame	Daten	Prüfwert	Frame
0110 1001	0110	001	x011 0001	1001	110	x100 1110

Die Codierten Pakete sind wie folgt aufgebaut:

Empfänger ID	Codierte Frames

Ein Test hat gezeigt, dass die Pakete erfolgreich korrigiert werden konnten, jedoch ist dies nicht bei jedem Paket der Fall. Sind mehrere Bits falsch, wird das Frame falsch korrigiert und das Paket muss wiederholt gesendet werden.

Durch die Codierung wurde der Funkkanal nicht verbessert, die Kommunikation verläuft jedoch erfolgreicher und kann die Datei schneller übermitteln.

7 Inbetriebnahme

Diese Inbetriebnahme soll helfen eine erfolgreiche Kommunikation aufzubauen und falsch funktionierende Elemente zu erkennen.

7.1 GNU Radio starten

Schrauben Sie die Antennen an die Schnittstellen vom LimeSDR mini. Verbinden Sie das Funkmodul über USB mit dem Computer.

Starten Sie GNURadio Companion und öffnen Sie die Datei transceiver.grc

Starten Sie mit dem grauen Play Button den Flowgraph und warten Sie bis die Graphiken des Transmitters und Receivers erscheinen.

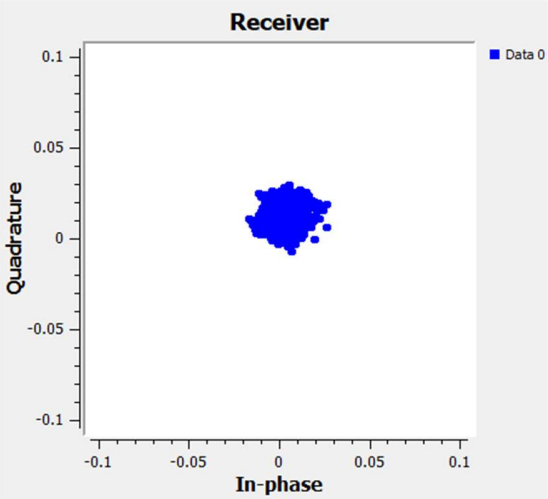


Abbildung 27 empfangenes Rauschsignal

Beim Transmitter Graph sehen Sie das Signal, welches gesendet wird. Dieser Graph soll ein kleiner Punkt im Ursprung zeigen, da kein Signal gesendet wird.

Im unteren Graph sehen Sie das Signal, welches über den Receiver empfangen wird. Dieser Graph soll wie in Abbildung 27 ein Rauschen im Ursprung zeigen.

Die Graphiken können mit dem Mausrad vergrössert werden.

Falls kein Rauschen detektiert wird:
Schliessen Sie GNU Radio, stecken Sie den LimeSDR aus und wiederholen Sie die Inbetriebnahme.

7.2 Empfänger prüfen

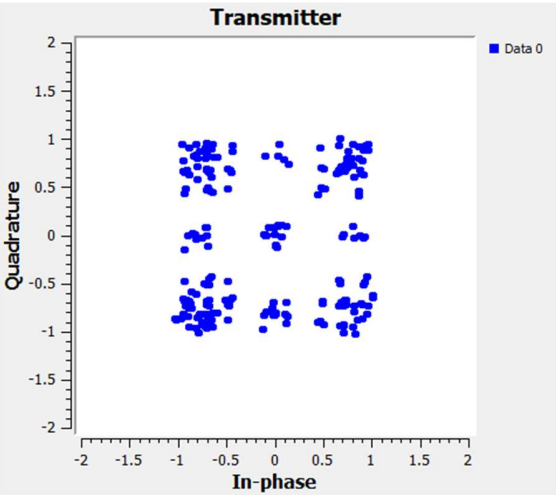


Abbildung 28 gesendetes QPSK Signal

Um die Antenne des Empfängers zu prüfen, senden Sie mit einem zweiten LimeSDR mini ein wiederholendes QPSK moduliertes Signal.

Nehmen Sie einen weiteren Computer und bereiten Sie Ihn wie der erste vor.

Starten Sie das Python File send_preamble und vergleichen Sie die Graphik des Transmitters mit der Abbildung 28. Das Muster entspricht einem QPSK moduliertem Signal.

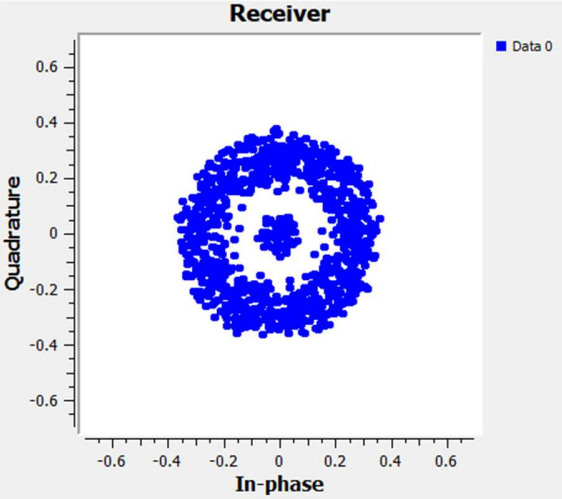


Abbildung 29 empfangenes QPSK Signal

Richten Sie die Empfänger Antenne des ersten Computers, bis Sie ein Signal wie in Abbildung 29 abgebildet erhalten.

Falls das Signal zu stark ist, kann das Gain des Empfängers reduziert werden. Versuchen Sie einen möglichst grossen und gleichmässigen Kreis zu empfangen.

Um zu überprüfen, ob vom Signal die Präambel erkannt wird, starten Sie das Python File detect_preamble.py.

Wenn eine Präambel erkannt wird, wird der in Abbildung 30 abgebildete Graph angezeigt. Die wiederholenden Peaks der erkannten Präambel ist gut zu erkennen. Je stärker der Peak ersichtlich ist, desto besser ist die Verbindung.

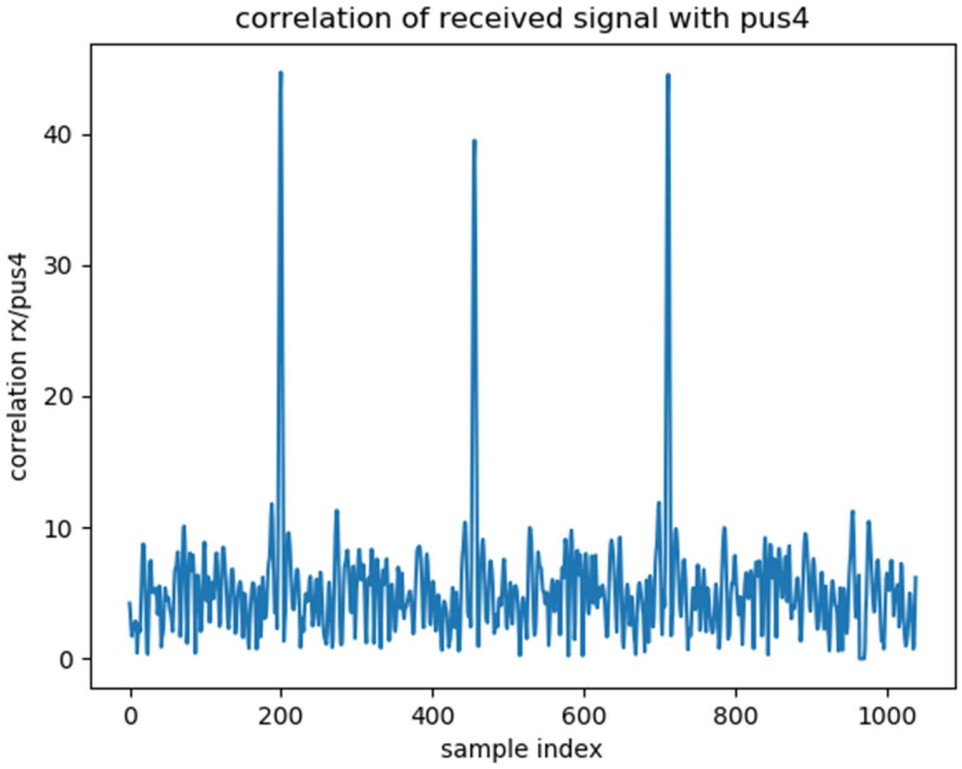


Abbildung 30 Korrelation von rx und pus4

Um den Rückkanal zu testen, wechseln Sie die Python Files und machen Sie denselben Ablauf. Bevor Sie den zweiten Test starten, warten Sie bis wieder das Rauschen empfangen wird und die Buffer leer sind.

8 Reflektion

In dieser Arbeit durfte ich sehr viel neues lernen und entdecken. Meine Erfahrungen wurden im technischen wie auch im praktischen Bereich erweitert. Ich habe erkannt, dass die organisatorischen Fähigkeiten ebenso wichtig sind wie die technischen und dürfen nicht vernachlässigt werden.

Zu Beginn des Projekts wurde ein optimistischer Ansatz umgesetzt und die Arbeitsschritte waren zu gross. Konzepte wurden geprüft, bevor die relevanten Funktionen genügend genau verstanden wurden. Dies hatte zu folge, dass die Ergebnisse nicht erfolgreich ausfielen und die Arbeit immer mehr unterteilt werden musste. Durch die gezielteren Untersuchungen konnten die kritischen Elemente erkannt und verbessert werden. Es lohnt sich von Beginn an kleinere dafür gezieltere Entwicklungsschritte zu nehmen.

Im technischen Bereich habe ich einen Einblick in die Schwierigkeiten der Funkkommunikation erhalten. Eine Kommunikation verfügt über viele Elemente und sobald ein Element dieser Kommunikationskette nicht sauber funktioniert, besteht keine gute Verbindung.
So gibt es den Bereich Software, Hardware und der Einfluss der Umgebung. Software kann sehr einfach kontrolliert und direkt verbessert werden. Die Hardware kann ebenfalls verbessert werden, Veränderungen können jedoch nicht so direkt umgesetzt werden wie bei der Software. Die Einflüsse der Umgebung kann lediglich erkannt und über Software oder Hardware korrigiert werden. Leider habe ich in meiner Arbeit Inkonsistenzen der Hardware entdeckt und konnte deren Ursache nicht herausfinden. Die Untersuchung der Hardware war frustrierend und dennoch eine wertvolle Erfahrung.

Die Fehler mit Hilfe der Software zu korrigieren war sehr interessant und hat mein Wissen wie auch meine Erfahrung sehr gut bereichert..

9 Anhang

9.1 Aufgabenstellung

Bachelor Thesis im Studiengang Elektrotechnik und Informationstechnologie

Aufgabe für Herrn Tobias Mailänder

Datenübertragung mit Rückkanal in Software-Defined Radio
Fachliche Schwerpunkte

Signalverarbeitung & Kommunikation

Einleitung
Mit dem LimeSDR Mini, einem Software-Defined Radio (SDR) Modul, lassen sich Radiosignale von 10 MHz bis 3.5 GHz breitbandig senden und empfangen. Zusammen mit GNU Radio bieten sich viele Möglichkeiten zur Realisierung unterschiedlicher Anwendungen der Nachrichtenübertragung in Python, die ansonsten nur mit aufwendiger Hardware möglich wären.

Aufgabenstellung
In dieser Arbeit geht es darum, die Möglichkeiten des LimeSDR Mini zur Datenübertragung mit Rückkanal zu betrachten. Bei einer OFDM-Übertragung beispielsweise kann der Sender über einen Rückkanal über den Kanalzustand informiert und dementsprechend die Modulation angepasst wer-den. Die Latenz muss sich dafür allerdings in Grenzen halten.

Die genauen Ziele werden zu Beginn des Projektes besprochen und festgelegt.

Termine

Start der Arbeit:	Montag 16.9.2019
Zwischenpräsentation:	Zu vereinbaren im Zeitraum 21.10. – 22.11.2019
Abgabe Schlussbericht:	Montag 6. Januar, vor 15:00 im Sekretariat
Abgabe Digitale Doku:	Gemäss separater Anweisung der Studiengangleitung
Abschlusspräsentation:	Zu vereinbaren im Zeitraum 6.1. – 24.1.2020
Diplomausstellung:	Freitag 3. Juli 2020 (Teilnahme obligatorisch!)

Dokumentation

Der gebundene Schlussbericht enthält keine Selbständigkeitserklärung und ist in 3-facher Ausführung zu erstellen. Er enthält zudem zwingend
- einen englischen Abstract mit maximal 2000 Zeichen.
- Ein Titelblatt, gleich hinter dem Deckblatt, gemäss Weisungen der Studiengangleitung
- Eine SD-Hülle, innen, auf der Rückseite des Berichtes für den Betreuer

Alle Exemplare des Schlussberichtes müssen komplett und termingerecht gemäss Angaben der Studiengangleitung abgeben werden. Zusätzlich muss eine SD-Speicherkarte mit dem Bericht (inkl. Anhänge), dem Poster und den Präsentationen, Messdaten, Programmen, Auswertungen, usw. un-mittelbar nach der Präsentation abgeben werden. Die gesamte Dokumentation ist zudem gemäss Anweisungen der Studiengangleitung elektronisch auf einen Server zu laden. Sämtliche abzugebende Teile der Dokumentation sind Bestandteile der Beurteilung.

Fachliteratur/Web-Links/Hilfsmittel

Geheimhaltungsstufe: Öffentlich

Verantwortlicher Dozent/Betreuungsteam, Industriepartner

Dozent	Prof. Dr. Thomas Hunziker	thomas.hunziker@hslu.ch
Experte	Werner Scheidegger	werner.scheidegger@sbb.ch

Hochschule Luzern
Technik & Architektur

Prof. Dr. T. Hunziker

10 Quellen

[1] Wikipedia (2019). «Software Defined Radio»
https://de.wikipedia.org/wiki/Software_Defined_Radio

[2] MYRIAD RF LimeSDR Mini Hardware Description
https://wiki.myriadrf.org/LimeSDR-Mini_v1.2_hardware_description

[3] cablefree.net (2019). «FDD vs TDD Technology»
https://www.cablefree.net/wirelesstechnology/fdd-vs-tdd/

[4] Wikipedia (2019). «ARQ Protokoll»
https://de.wikipedia.org/wiki/ARQ-Protokoll

[5] Tutorialspoint (2019). «Stop and Wait ARQ»
https://tutorialspoint.dev/computer-science/computer-network-tutorials/stop-and-wait-arq

[6] GNU Radio (2019). «Guided Tutorial PSK Demodulation»
https://wiki.gnuradio.org/index.php/Guided_Tutorial_PSK_Demodulation

[7] Wikipedia (2019). «Zyklische Redundanzprüfung»
https://de.wikipedia.org/wiki/Zyklische_Redundanzpr%C3%BCfung