GNUradio Workbook – Teil 2

1. Einleitung in die digitale Nachrichtentechnik

Nachdem wir im ersten Teil analoge Signale (wie Sprache oder Musik) empfangen und demoduliert haben, widmen wir uns nun der Königsdisziplin: Der digitalen Signalverarbeitung.

In der digitalen Welt übertragen wir keine stufenlosen Spannungen mehr, sondern diskrete Zustände – Nullen und Einsen. Diese Zustände nennen wir **Symbole**. Die große Herausforderung bei der digitalen Demodulation besteht darin, dass ein Funksignal durch die Luft verzerrt, abgeschwächt und mit Rauschen überlagert wird. Aus unseren perfekten, eckigen digitalen Nullen und Einsen werden am Empfänger verschmierte, wackelige Sinuswellen.

Unser Computer (und GNU Radio) muss nun drei fundamentale Fragen klären, um das Signal wiederherzustellen:

1. Signalpegel: Wie laut ist das Signal aktuell? (Amplitude / Gain)
2. Timing: **Wann** genau beginnt ein Symbol und wann endet es? Wo ist die Mitte des Symbols, um es optimal abzutasten? (Clock Recovery)
3. Phase: Wenn die Information in der Verschiebung der Welle steckt, wo ist dann der „Nullpunkt“ der Welle? (Carrier Recovery)

GNU Radio bietet für all diese Probleme hochspezialisierte Algorithmen. Wir werden diese nun Schritt für Schritt entzaubern.

2. Frequency Shift Keying (FSK) und ASCII-Dekodierung

Die robusteste Form der digitalen Modulation ist die Frequenzumtastung (Frequency Shift Keying, FSK). Hierbei werden digitale Nullen und Einsen durch den Wechsel zwischen zwei eng beieinander liegenden Frequenzen gesendet.

Wir schauen uns dies an einem digitalen Fernschreib-Verfahren an. Statt des klassischen 5-Bit Baudot-Codes (RTTY) nutzen wir hier ein **ASCII-FSK** Signal (8-N-1). Das bedeutet, wir übertragen ganz normale Computer-Buchstaben (8 Bit pro Zeichen), getrennt durch Start- und Stoppbits, genau wie bei einer seriellen UART-Schnittstelle eines Arduinos oder eines alten Modems.

Es gibt zwei Töne:

- **Mark (Logisch 1):** Die höhere Frequenz.
- **Space (Logisch 0):** Die tiefere Frequenz.
- **Baudrate:** Wir senden mit gemächlichen 1200 Baud (1200 Symbole pro Sekunde).

2.1. Schritt 1: Das Frequenz-Wackeln sichtbar machen

Um FSK zu demodulieren, können wir uns eines alten Bekannten bedienen: der Frequenzmodulation (FM). Wenn sich bei FSK die Frequenz ändert, können wir einen FM-Demodulator nutzen. Dieser gibt eine positive Spannung (orange, Float) aus, wenn die Frequenz steigt (Mark), und eine negative Spannung, wenn die Frequenz sinkt (Space).

2.1.1. Block-Fokus: Low Pass Filter (Complex)

- **Funktion:** Lässt nur Frequenzen unterhalb eines bestimmten Schwellenwertes passieren und blockiert alles darüber.

- **Wann man ihn braucht:** Bei fast jedem I/Q-Empfang! Wenn wir ein zentriertes Signal haben (auf 0 Hz gemischt), befindet sich links und rechts im Spektrum Rauschen oder benachbarte Sender. Bevor wir demodulieren, müssen wir das nützliche Signal isolieren.
- **Wie er funktioniert:** Als komplexer FIR-Filter (Finite Impulse Response) berechnet er gewichtete Durchschnitte aus vergangenen Samples. Bei Typ `Complex->Complex` wendet er diese Mathematik simultan auf den In-Phase- (I) und Quadrature-Kanal (Q) an, ohne die Phasenbeziehung zu zerstören.

2.1.2. Block-Fokus: Quadrature Demodulator

- **Funktion:** Wandelt Frequenzänderungen eines I/Q-Signals in reelle Spannungswerte um (FM-Demodulation).
- **Wann man ihn braucht:** Immer dann, wenn die gesuchte Information in der Frequenzänderung steckt (UKW-Radio, FSK-Daten, APT-Wettersatelliten).
- **Wie er funktioniert:** Der Block vergleicht das aktuelle I/Q-Sample mit dem vorherigen Sample. Er misst den Winkel zwischen diesen beiden komplexen Zeigern. Dreht sich der Zeiger schneller gegen den Uhrzeigersinn (höhere Frequenz), wird der Winkel größer und der Block gibt einen positiven, orangen Float-Wert aus. Dreht er sich langsamer, wird der Wert negativ.

Aufgabe 1 — Das rohe Basisband extrahieren

Wir haben dir die Datei `BeispielA1.iq` vorbereitet. Sie enthält ein zentriertes FSK-Signal bei einer Sample Rate von 48.000 Hz. Wir wollen daraus zunächst ein analoges Signal gewinnen, das im Takt der Bits auf und ab springt.

a) *Lade die I/Q-Datei mit einer **File Source** in deinen leeren Flowgraph.*

b) *Füge zwingend einen **Throttle** Block hinzu. Setze dann die Sample Rate auf `48000`, damit dein PC nicht überlastet. Verbinde ihn mit der Quelle.*

c) *Da das Funksignal von Rauschen umgeben ist, müssen wir es isolieren. Schalte einen **Low Pass Filter** (Typ: Complex->Complex) hinter den Throttle. Setze die `Cutoff Freq` auf `1500` Hz und die `Transition Width` auf `500` Hz. Alles außerhalb dieses Bereichs wird nun gelöscht.*

d) *Leite das gefilterte, blaue I/Q-Signal in einen **Quadrature Demodulator**. Den Parameter `Gain` kannst du zunächst auf `1` belassen. Anstelle eines blauen Ports erhältst du nun einen orangen Float-Datenstrom.*

e) *Hänge eine **QT GUI Time Sink** im Float-Modus an den Ausgang des Demodulators. Starte den Flowgraph. Drücke in der Time Sink das „Middle Mouse Button“ Menü und schalte `Trigger` auf `Auto`. Du solltest nun ein Rechtecksignal sehen, das zwischen einer positiven und negativen Spannung springt!*

> **Hint**
>
> Was sehen wir hier? Das Rechtecksignal in der Time Sink ist unser sogenanntes **Baseband-Signal** (Basisband). Die oberen Plateaus sind unsere logischen Einsen, die unteren unsere Nullen. Durch das Rauschen sind die Kanten jedoch ausgefranst und wackelig.

2.2. Schritt 2: Symbol-Synchronisation (Timing Recovery)

Wir haben nun eine Spannung, die auf und ab springt, aber GNU Radio weiß noch nicht, **wann** es dieses Signal abtasten muss. Wenn wir exakt an der Flanke (während das Signal gerade von +1 auf –1 wechselt) abtasten, erhalten wir Zufallswerte. Wir müssen das Signal exakt in der **Mitte** eines jeden Rechtecks abtasten, wo der Pegel am stabilsten ist!

2.2.1. Block-Fokus: Symbol Sync

Funktion: Findet den perfekten Takt in einem kontinuierlichen Datenstrom und tastet jedes Symbol genau einmal exakt in seiner zeitlichen Mitte ab.

- **Wann man ihn braucht:** Zwingend bei **jeder** digitalen Demodulation. Ohne Takt-Synchronisation gibt es keine korrekten Bits.
- **Wie er funktioniert:** Der Block nutzt einen TED (Timing Error Detector). Dieser Algorithmus sucht nach den Flanken (Wechsel von 0 auf 1). Da er weiß, wie viele Samples idealerweise ein Symbol lang sein sollte (Parameter: `Samples per Symbol`), platziert er seinen virtuellen „Abtast-Finger“ genau in die Mitte zwischen zwei erwarteten Flanken.
- **Wichtige Parameter: `Loop Bandwidth`: Bestimmt, wie träge oder aggressiv der Algorithmus auf Timing-Abweichungen reagiert. Klassische Werte liegen zwischen 0.02 und 0.06.** `TED Type`: Verschiedene Signalarten brauchen verschiedene Erkenner. Für rohe, ungefilterte Rechtecksignale (wie FSK) nutzt man meist **Mueller and Müller**.

Aufgabe 2 — Den perfekten Takt finden

Wir integrieren nun die Taktrückgewinnung, um aus dem wackeligen analogen Strom perfekte, isolierte Symbole zu machen.

a) *Füge den **Symbol Sync** Block im Float-Modus hinter deinem Quadrature Demodulator ein.*

b) *Die Konfiguration dieses Blocks ist die größte Hürde in GNU Radio! Wähle als `Timing Error Detector (TED)` die Option `Mueller and Müller`.*

c) *Setze den Parameter `Samples per Symbol`. Dies ist das Verhältnis aus Sample Rate und Baudrate. Berechne $\frac{48000}{1200}$ und trage das exakte Ergebnis (`40.0`) hier ein.*

d) *Setze die `Loop Bandwidth` auf `0.04`. Das ist die „Trägheit“ des Algorithmus.*

e) *Verbinde den **obersten** Ausgang des Symbol Syncs (den modulierten Datenstrom) mit einer neuen **QT GUI Time Sink**. Starte den Graphen. Klicke in der Time Sink auf das Rädchen (Settings), wähle `Line 1` und setze `Marker` auf `Circle` und `Line` auf `None`. Du siehst*

nun keine Linien mehr, sondern perfekte rote Punkte, die sich in zwei horizontalen Linien formieren: Unsere extrahierten Einsen und Nullen!

2.3. Schritt 3: Eigene Blöcke programmieren (Der Python Block)

Jetzt haben wir perfekte Einsen und Nullen (Magentafarbener Port). Normalerweise bräuchten wir nun einen UART-Receiver, um Start- und Stoppbits zu erkennen und die Bits in ASCII-Text zu wandeln. GNU Radio bietet standardmäßig jedoch keinen fertigen UART-Block an!

In der echten Signalverarbeitung ist dies der Moment, in dem Ingenieure eigene Blöcke schreiben. Wir nutzen dafür das mächtigste Werkzeug in GNU Radio: Den **Python Block**.

2.3.1. Block-Fokus: Python Block

- **Funktion:** Erlaubt es dir, das Verhalten eines Blocks in Echtzeit mit einem eigenen Python-Skript zu programmieren.
- **Wann man ihn braucht:** Immer dann, wenn die Standard-Bibliothek von GNU Radio an ihre Grenzen stößt (z.B. bei speziellen Netzwerkprotokollen, exotischen Dekodern oder eben UART).
- **Wie er funktioniert:** Der Block stellt ein Grundgerüst (Template) zur Verfügung. Du definierst, welche Datentypen vorne reingehen und hinten rauskommen. In der `work()`-Funktion iterierst du über den ankommenden Datenstrom und programmierst deine eigene Logik (z.B. eine State-Machine).

Aufgabe 3 — Den UART-Decoder selbst programmieren

Wir verwandeln die Datenpunkte in Bytes und programmieren unseren eigenen Interpreter, der die geheime Nachricht ausgibt.

a) *Füge den Block **Binary Slicer** hinter den Symbol Sync ein. Er macht aus reellen Zahlen harte `0` oder `1` Werte (Typ: Byte).*

b) *Suche in der Bibliothek den **Python Block** und füge ihn hinzu. Mache einen Doppelklick darauf und klicke auf **Open in Editor**. Es öffnet sich der integrierte Code-Editor.*

c) *Lösche den vorgegebenen Code und ersetze ihn durch unseren eigenen UART-Decoder (siehe Kasten unten). Speichere die Datei (Strg+S / Cmd+S) und schließe den Editor. Der Block hat sich nun optisch verändert und heißt „UART Decoder“!*

d) *Verbinde den magentafarbenen Ausgang des Binary Slicers mit dem Eingang deines neuen Python Blocks.*

e) *Füge den Block **Message Debug** hinzu. Verbinde den gestrichelten `out_msg` Port deines Python Blocks mit dem `print` Port des Message Debugs.*

f) *Starte den Flowgraph. Schau unten links in dein Konsolen-Fenster. Dein eigener Code schnürt nun die Bits zusammen und druckt live einen lesbaren Text aus! Notiere den Satz hier:*

```python
import numpy as np
from gnuradio import gr
import pmt

class blk(gr.sync_block):
    def __init__(self):
        # Wir nehmen Bytes (int8) als Eingang und haben keinen direkten Ausgang,
        # da wir Nachrichten (Messages) senden wollen.
        gr.sync_block.__init__(self, name='UART Decoder', in_sig=[np.int8], out_sig=None)
        self.message_port_register_out(pmt.intern('out_msg'))
        self.state = 'WAIT_START'
        self.bits = []

    def work(self, input_items, output_items):
        for bit in input_items[0]:
            if self.state == 'WAIT_START':
                if bit == 0:  # Startbit gefunden!
                    self.state = 'READ_BITS'
                    self.bits = []

            elif self.state == 'READ_BITS':
                self.bits.append(bit)
                if len(self.bits) == 8: # 8 Datenbits gelesen
                    self.state = 'WAIT_STOP'

            elif self.state == 'WAIT_STOP':
                if bit == 1:  # Stoppbit korrekt?
                    # Bits in eine Zahl umwandeln (LSB first)
                    val = sum([b << i for i, b in enumerate(self.bits)])

                    # Nur druckbare ASCII-Zeichen ausgeben
                    if 32 <= val <= 126:
                        msg = pmt.intern(chr(val))
                        self.message_port_pub(pmt.intern('out_msg'), msg)

                self.state = 'WAIT_START' # Zurück auf Anfang

        return len(input_items[0])
```

Listing 1: Der Code für unseren maßgeschneiderten Python Block. Er nutzt eine einfache Zustandsmaschine (State Machine), um das UART-Timing auszuwerten.

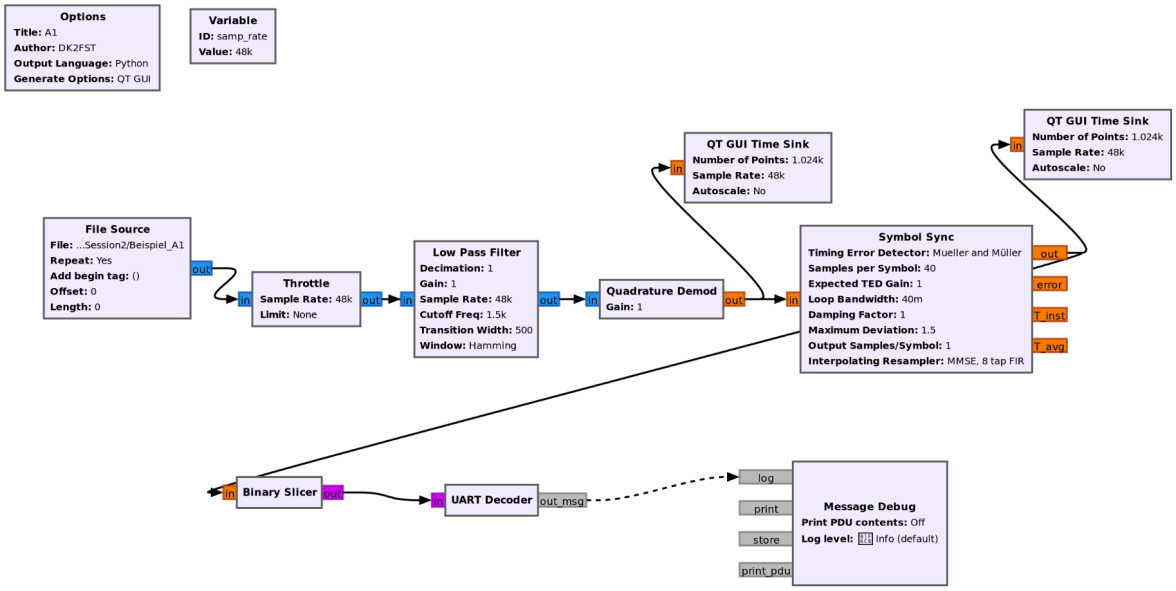

Abbildung 1: Die komplette Kette für die Demodulation von FSK zu ASCII. Der wichtigste Schritt ist die Taktrückgewinnung in der Mitte, gefolgt von unserer eigenen Logik.

3. Phase Shift Keying (QPSK) und das Augendiagramm

FSK ist extrem robust, verschwendet aber viel Frequenzbandbreite für sehr geringe Datenraten. Wenn wir heute Daten per Satellit, WLAN oder DVB-T übertragen, nutzen wir **Phase Shift Keying (PSK)** oder **Quadrature Amplitude Modulation (QAM)**.

Hier wackeln wir nicht an der Frequenz, sondern manipulieren gezielt den **Winkel** (Phase) und die **Länge** (Amplitude) unseres I/Q-Zeigers. Bei QPSK (Quadrature Phase Shift Keying) kann unser Symbol 4 verschiedene Winkel annehmen (z.B. 45°, 135°, 225°, 315°). Da $2^2 = 4$ ist, übertragen wir mit jedem Phasenumschlag direkt **2 Bits gleichzeitig**!

3.1. Schritt 1: Vorbereitung und automatische Vorverstärkung

Wir laden nun eine Telemetrie-Aufzeichnung eines Satelliten (`BeispielA2.iq`). Wenn sich ein Satellit am Himmel dreht, schwankt die Signalstärke extrem (Fading).

3.1.1. Block-Fokus: Automatic Gain Control (AGC2)

- **Funktion:** Regelt die Amplitude (Lautstärke) eines Signals automatisch auf einen konstanten Zielwert (z.B. 1.0).
- **Wann man ihn braucht:** Bei Signalen, deren Stärke auf dem Funkweg schwankt (Fading) oder wenn man nicht genau weiß, wie stark das Eingangssignal ist. Mathematische Algorithmen (wie QAM-Decoder) benötigen oft einen exakten Referenzpegel, um richtig zu funktionieren.
- **Wie er funktioniert:** Der Block misst kontinuierlich die Durchschnittsamplitude des Eingangssignals. Ist sie zu niedrig, dreht er einen internen Multiplikator hoch. Ist sie zu stark, dreht er ihn herunter. Parameter wie `Attack Rate` bestimmen, wie schnell er auf plötzliche Signalspitzen reagiert.

3.1.2. Block-Fokus: Root Raised Cosine Filter (RRC)

- **Funktion:** Begrenzt die Bandbreite eines digitalen Signals dramatisch und verhindert Intersymbolinterferenz (ISI).
- **Wann man ihn braucht:** RRC-Filter sind der weltweite Standard in der digitalen Signalverarbeitung (Satellit, Mobilfunk). Wenn der Sender das Signal durch einen RRC-Filter gejagt hat, **muss** der Empfänger exakt denselben Filter als „Matched Filter“ anwenden.
- **Wie er funktioniert:** Harte digitale Rechteck-Pulse würden beim Senden unendlich viel Frequenz-Bandbreite belegen (aufgrund extrem steiler Flanken). Der RRC-Filter am Sender rundet diese Ecken ab – das Signal verschmiert scheinbar. Am Empfänger durchläuft das Signal denselben RRC-Filter noch einmal. Durch die mathematische Faltung beider RRC-Hälften entsteht ein perfekter „Raised Cosine“-Impuls. Genau im Abtastzeitpunkt des Symbols heben sich die Echos der Nachbarsymbole magisch auf (Nyquist-Kriterium)!

Aufgabe 4 — Pegelanpassung und Matched Filter

Wir bauen das Front-End unseres Satelliten-Empfängers.

a) *Erstelle einen neuen Flowgraph. Lade die I/Q-Datei mit einer **File Source** (Sample Rate: `100000` Hz) und setze einen **Throttle** dahinter.*

b) *Füge den Block **AGC2** (Complex) ein. Setze die `Reference Rate` auf `1.0`, den `Attack Rate` auf `1e-3` und den `Decay Rate` auf `1e-4`.*

c) *Füge den Block **Root Raised Cosine Filter** (RRC) ein. Wähle `Complex->Complex`.*

d) *Konfiguriere den RRC-Filter: Decimation: `1`, Gain: `1`, Sample Rate: `100000`, Symbol Rate: `25000` (Das Signal hat 25k Baud), Alpha (Roll-off): `0.35`. Unter Taps trägst du `11` ein. Dies definiert die Länge des Filters.*

e) *Schließe eine **QT GUI Time Sink** an. Du solltest nun kontinuierliche blaue (I) und rote (Q) Schwingungen sehen, die sehr weich und sinusförmig verlaufen. Das ist der Effekt des RRC-Filters!*

3.2. Schritt 2: Das Augendiagramm (Eye Diagram)

Um die Qualität unseres digitalen Signals zu beurteilen, bevor wir es demodulieren, nutzt der Ingenieur das Augendiagramm. Dabei legen wir viele Symbole auf dem Bildschirm des Oszilloskops übereinander.

3.2.1. Block-Fokus: QT GUI Eye Diagram

- **Funktion:** Visualisiert den Datenstrom nicht über die Zeit weglaufend, sondern plottet exakt die Länge von 1 oder 2 Symbolen immer wieder übereinander auf dieselbe Stelle.
- **Wann man ihn braucht:** Zur optischen Beurteilung der Signalqualität, der Filtereinstellungen und zur Erkennung von Jitter oder Rauschen, bevor das Signal vom Symbol Sync abgetastet wird.
- **Wie er funktioniert:** Indem er den Graph alle X Samples (X = Samples pro Symbol) neu auf der linken Seite zu zeichnen beginnt, überlagern sich die Signalkurven. Ist das Signal

störungsfrei und der RRC-Filter passt, treffen sich alle Kurven genau an einer Stelle (dem idealen Abtastzeitpunkt), und dazwischen entsteht ein großer Freiraum – das offene Auge. Ist das Auge „geschlossen“, schlägt die Demodulation sehr wahrscheinlich fehl.

Aufgabe 5 — Einen Blick auf das Auge werfen

Wir nutzen das Augendiagramm, um zu prüfen, ob unser RRC-Filter richtig eingestellt ist.

a) *Füge eine **QT GUI Eye Diagram** Sink hinzu und verbinde sie mit dem Ausgang des RRC-Filters.*

b) *Das Augendiagramm braucht die Anzahl der Samples pro Symbol. Berechne* $\frac{100000}{25000} = 4$ *und trage* `4` *unter* `Samples per Symbol` *ein.*

c) *Starte den Flowgraph. Du siehst nun ein Diagramm, das aussieht wie aufgerissene Augen. Sind die „Augen“ weit geöffnet, bedeutet das, dass wir in der Mitte des Auges störungsfrei abtasten können.*

d) *Experimentiere kurz: Öffne den RRC-Filter und ändere den* `Alpha`*-Wert auf* `0.9` *und starte neu. Beobachte, wie das Auge verschmiert. Setze den Wert danach wieder auf* `0.35` *zurück!*

e) *Füge nun den uns bekannten **Symbol Sync** hinter den RRC-Filter ein. Diesmal wählst du unter TED Type* `Polyphase Clock Sync` *(oder* `Gardner`*), da dieser bei QPSK und RRC-gefilterten Signalen wesentlich besser funktioniert als Mueller & Müller. Samples per Symbol wieder auf* `4.0` *setzen.*

3.3. Schritt 3: Carrier Recovery (Phasen-Korrektur)

Hänge eine **Constellation Sink** hinter deinen Symbol Sync. Du wirst sehen: Statt 4 Punkten siehst du einen rotierenden Ring! Warum?

Die Oszillatoren von Satellit und Empfänger schwingen niemals exakt auf die Nachkommastelle gleich. Ein winziger Frequenzfehler sorgt dafür, dass sich unser I/Q-Zeiger permanent weiterdreht. Das Konstellationsdiagramm dreht sich wie ein Karussell.

3.3.1. Block-Fokus: Costas Loop

Funktion: **Ein geschlossener Regelkreis, der minimale Frequenz- und Phasenfehler zwischen Sender und Empfänger erkennt und das rotierende Signal ausgleicht, bis es „still steht“. Wann man ihn braucht:** Bei BPSK, QPSK oder 8-PSK Signalen, bevor man den Constellation Decoder anwendet. Ohne Phasenkorrektur ist PSK wertlos! Wie er funktioniert: **Die Costas Loop nimmt das Eingangssignal und vergleicht die Phase mit einem idealen Referenzraster (bei QPSK: vier Punkte im Abstand von 90°). Liegt das empfangene Symbol knapp daneben, berechnet die Loop einen Fehlerwert („Phase Error“). Dieser Fehler wird genutzt, um einen internen Oszillator anzupassen, der das Signal gegensteuert. Sobald Fehler = 0 ist, rastet die Schleife ein („Lock“). Wichtige Parameter:** `Order`. Dies gibt an, wie viele Phasenlagen erwartet werden. BPSK = 2, QPSK = 4, 8-PSK = 8.

Aufgabe 6 — Das Karussell anhalten

Wir bauen einen Empfänger, der den Träger fängt und das Konstellationsdiagramm stabilisiert.

a) *Füge den **Costas Loop** Block zwischen den Symbol Sync und die Constellation Sink ein.*

b) *Da wir QPSK (4 Punkte) haben, setze die `Order` zwingend auf `4`.*

c) *Setze die `Loop Bandwidth` auf `0.05`.*

d) *Starte den Flowgraph erneut. Wenn die Costas Loop einrastet („lock“), friert das Karussell fast sofort ein! Aus dem rotierenden Ring werden vier scharfe, klar abgegrenzte Wolken in den Ecken des Diagramms.*

e) *Füge eine **QT GUI Scatter Plot** Sink hinzu, um das Rauschen der Wolken genauer zu analysieren. Je enger die Punkte beisammen liegen, desto besser ist dein Signal-Rausch-Verhältnis (SNR)!*

3.4. Schritt 4: Konstellation-Decoder

Das Signal ist perfekt, die 4 Wolken stehen still. Jetzt müssen wir den Punkten Bytes zuordnen. Punkt oben rechts = `00`, oben links = `01` usw.

3.4.1. Block-Fokus: Constellation Rect Object & Constellation Decoder

- **Funktion:** Das **Object** ist eine Art Blaupause oder Wörterbuch, das dem System sagt, welche Koordinate (I/Q) zu welchen Bits gehört. Der **Decoder** führt diese Übersetzung auf Basis der Blaupause live für jedes eintreffende Symbol durch.
- **Wann man sie braucht:** Am Ende jeder M-PSK oder QAM Empfangskette, um die physikalischen Symbole wieder in reine Informatik (Bytes) zu übersetzen.
- **Wie sie funktionieren:** Im Constellation Object sind ideale Punkte und ihre entsprechenden Bit-Werte (Symbols) definiert. Der Decoder nimmt ein blaues I/Q-Sample entgegen (z.B. von der Costas Loop), misst den Abstand zu allen bekannten Punkten aus dem Wörterbuch und wählt den nächstgelegenen („Hard Decision“). Danach spuckt er den magentafarbenen Bytewert aus.

Aufgabe 7 — Die Wolken lesen

Wir extrahieren die Telemetrie-Nachricht.

a) *Füge den Block **Constellation Decoder** hinter die Costas Loop ein.*

b) *Der Block braucht ein `Constellation Object`. Lege im leeren Raum deines Graphen einen Block vom Typ **Constellation Object** an.*

c) *Konfiguriere das Object: Wähle als Typ `QPSK`. Das System weiß nun automatisch, wie die Punkte im Raum liegen und welche Bitpaare sie repräsentieren.*

d) *Trage in den Eigenschaften des **Constellation Decoders** unter „Constellation“ die ID deines Constellation Objects ein (meist `variable_constellation_0`). Der Decoder wandelt nun die blauen I/Q-Werte in magentafarbene Bytes um.*

e) *Da wir nur 2 Bits auf einmal decodieren, aber ganze Bytes ausgeben wollen, müssen wir diese „Packen“. Dies tut der Block **Unpacked To Packed**. Füge einen solchen hinter dem Constellation Decoder ein. Die Standardeinstellungen sollten funktionieren.*

f) *Füge nun hinter dem Unpacked to Packed einen sog. File Descriptor Sink ein. Dieser kann unser Ergebnis direkt auf die Konsole schreiben. Wähle wie gewohnt den Typ Byte aus und verbinde die Blöcke. Für den **File Descriptor** setzt du **0** ein.*

3.4.2. Bit Delay

Wir haben nun noch das Problem, dass wir wegen des verrauschten Signals nicht perfekt beim 1. Bit der Übertragung anfangen, das Signal zu decodieren. Die einzelnen Bits, die hierdurch verschoben werden, sorgen leider dafür dass unsere Ausgabe komplettes Kauderwelsch ist. Wir müssen also noch die verschobenen Bits korrigieren. Dies können wir mittels Delay tun.

Aufgabe 8 — Bits verschieben

a) *Erstelle eine QT GUI Range und gib ihr eine aussagekräftige ID.*

b) *Da wir maximal einen Byte an Verschiebung erreichen können wähle einen sinnvollen Bereich und Default-Wert für die Range.*

c) *Füge nun zwischen Constellation Decoder und Unpacked to Packed einen Delay-Block ein. Gib ihm für den Delay-Wert die ID der Range.*

d) *Führe den Flow aus. Probiere verschiedene Werte für den Delay aus, bis ein sinnvolles Ergebnis rauskommt.*

Herzlichen Glückwunsch. Du hast eben ein QPSK-Signal erfolgreich dekodiert!

4. Die Kür: Bilder aus Tönen (SSTV Robot36)

Willkommen im Finale! Du hast nun gelernt, wie man analog mischt, FM demoduliert, digitale Symbole synchronisiert und Phasenverschiebungen korrigiert. Jetzt bringen wir alles zusammen und decodieren ein Format, das in der Raumfahrt (von der ISS) und bei Funkamateuren noch heute genutzt wird, um Bilder zu übertragen: **Slow Scan Television (SSTV)**.

Wir haben dir die Datei `BeispielA3.wav` zur Verfügung gestellt. Unser Ziel ist es, das darin verborgene Bild nicht mit fertigen Apps, sondern auf der puren Byte-Ebene mit GNU Radio und dem Bildbetrachter GIMP sichtbar zu machen.

4.1. Wie funktioniert Robot36?

Robot36 ist ein analoges Farb-SSTV-Format, das extrem robust ist. Es überträgt Bilder nicht als digitalen Nullen-und-Einsen-Stream, sondern nutzt Frequenzmodulation (FM) im Audio-Bereich. Die Tonhöhe bestimmt die Helligkeit des aktuellen Pixels!

- **Frequenzen:** Der Ton schwankt zwischen 1500 Hz (Schwarz) und 2300 Hz (Weiß).
- **Sync-Puls:** Um der Software zu sagen „Hier beginnt eine neue Zeile“, sendet der Sender am Anfang jeder Zeile einen extrem kurzen, tiefen Ton auf exakt **1200 Hz**.
- **Bildaufbau:** Robot36 nutzt den YUV-Farbraum. Es sendet zuerst die Helligkeit (Luminanz, Y) für eine ganze Zeile. Dann sendet es abwechselnd die Farbunterschiede (Chrominanz, R-Y und B-Y) für die halbe Auflösung.
- **Dauer:** Ein komplettes Bild mit 320x240 Pixeln benötigt exakt 36 Sekunden. Eine einzelne Bildzeile dauert **exakt 120 Millisekunden** ($0,12$ s).

4.2. Schritt 1: Demodulation und Basisband-Gewinnung

Wir bauen nun einen Flowgraph, der das Audiosignal auspackt und die Frequenzen in proportionale Spannungen umwandelt.

4.2.1. Block-Fokus: Band Pass Filter

- **Funktion:** Eine Kombination aus Low-Pass und High-Pass. Er lässt nur Frequenzen in einem eng definierten „Fenster“ passieren. Alles, was zu hoch oder zu tief ist, wird abgeschnitten.
- **Wann man ihn braucht:** Wenn wir unser gesuchtes Signal (hier: 1200 - 2300 Hz) aus einem breiteren Frequenzband isolieren wollen, in dem möglicherweise tieffrequentes Rumpeln (z.B. 50 Hz Netzbrummen) oder hohes Rauschen stört.

4.2.2. Block-Fokus: Hilbert

- **Funktion:** Der Hilbert-Transformator nimmt ein reelles, oranges Signal und generiert daraus ein blaues, komplexes I/Q-Signal.
- **Wann man ihn braucht:** Wenn wir Audiodaten (z.B. aus WAV-Dateien oder der Soundkarte) in Blöcke leiten wollen, die zwingend komplexe Zahlen fordern (wie der Quadrature Demodulator).
- **Wie er funktioniert:** Er leitet das Originalsignal 1:1 an den realen (I) Kanal weiter. Parallel dazu verschiebt er die Phase des gesamten Signals um exakt 90 Grad und legt dieses verschobene Signal auf den imaginären (Q) Kanal. Das resultierende Signal verhält sich nun wie ein einseitiges Spektrum (ohne Spiegelfrequenzen) und kann demoduliert werden.

Aufgabe 9 — Das FM-Bildsignal aufbereiten

Wir extrahieren die Frequenzänderungen und stückeln sie in digitale Bildinformationen.

a) *Lade das WAV-File mit einer **Wav File Source**. (Kein Throttle nötig, da die WAV-Datei den Takt von 44.100 Hz vorgibt).*

b) *Schalte einen **Band Pass Filter** (Float) nach der Quelle (Cutoff Low: `1000`, Cutoff High: `2500`).*

c) *Füge den Block **Hilbert** hinzu und verbinde ihn mit dem Filter.*

d) *Füge den **Quadrature Demodulator** hinzu und hänge ihn hinter den Hilbert-Block. Setze `Gain` auf `1`.*

e) *Schließe eine **QT GUI Time Sink** an. Pausiere die Ansicht im laufenden Betrieb. Du solltest extrem tiefe Spikes nach unten sehen (Das sind unsere 1200 Hz Sync-Pulse!) und ein unregelmäßiges Wellenmuster dazwischen (die Bildpixel).*

4.3. Schritt 2: Skalierung und Analog-Digital-Wandlung (ADC)

Unsere Werte aus dem Demodulator schwanken nun im negativen und positiven Fließkommabereich (z.B. von −0.3 bis +0.8). Ein Bildpixel in GIMP erwartet aber einen Wert (Byte) von exakt `0` (Schwarz) bis `255` (Weiß).

4.3.1. Block-Fokus: Add Const & Multiply Const

- **Funktion:** Einfache mathematische Grundrechenarten. **Add Const** addiert auf jedes Sample einen festen Wert. **Multiply Const** multipliziert jedes Sample mit einem Faktor.
- **Wann man sie braucht:** Zur Skalierung und Anpassung von Pegeln. Add Const dient dem Entfernen oder Hinzufügen eines Gleichspannungsanteils (DC-Offset). Multiply Const ist ein klassischer Lautstärkeregler oder Verstärker.

4.3.2. Block-Fokus: Float to UChar

- **Funktion:** Ein Datentyp-Konverter. Er wandelt orangene 32-Bit Fließkommazahlen (Float) in magentafarbene 8-Bit Unsigned Characters (UChar/Bytes) um.
- **Wann man ihn braucht:** Beim Übergang von reeller Signalverarbeitung zur Speicherung in Roh-Datenformaten (Bilder, Bitstreams).
- **Wie er funktioniert:** Der Block schneidet gnadenlos alle Kommastellen ab (Trunkierung). Aus `204.8` wird das Byte `204`. Werte, die unter `0` fallen, werden zu `0`. Werte über `255` führen oft zu Überläufen (Rollover, 256 wird zu 0). Es ist daher **zwingend** erforderlich, das Signal vorher mit Add und Multiply exakt in den Bereich `0.0` bis `255.0` zu verschieben!

Aufgabe 10 — Das Signal in Bytes gießen

Wir verschieben und strecken das Signal mathematisch auf unsere 8-Bit-Grenzen.

a) *Lies in der Time Sink mit der Maus den exakten Y-Wert ab, an dem der tiefe Sync-Puls (1200 Hz) liegt (z.B. `-0.2`). Notiere diesen Wert.*

b) *Füge einen **Add Const** (Float) Block ein. Addiere exakt die positive Variante deines notierten Wertes (z.B. `0.2`), um das gesamte Signal „anzuheben“, sodass der tiefste Punkt nun bei exakt `0.0` liegt.*

c) *Lies nun den höchsten Punkt im Bild ab (z.B. `0.8`). Da wir von 0 bis 255 wollen, müssen wir verstärken. Füge einen **Multiply Const** Block ein. Multipliziere mit einem Faktor (z.B. `255 / 0.8 = 318`).*

d) *Füge den Block **Float to UChar** hinzu.*

e) *Speichere das Ganze mit einer **File Sink** als `robot36_bild.raw` ab. Lass den Graphen bis zum Ende der Sounddatei durchlaufen.*

4.4. Schritt 3: Das Bild in GIMP zusammensetzen

Wir haben nun eine Datei voller Millionen von Bytes. Sie hat keinen Header wie ein JPG. Wir müssen das Bild nun „manuell“ in GIMP zusammensetzen, indem wir die horizontale Ablenkung des Monitors simulieren.

Da wir wissen, dass die Sample-Rate der WAV-Datei **44.100 Hz** (Samples pro Sekunde) beträgt und eine Robot36-Zeile exakt **0,12 Sekunden** dauert, berechnen wir die exakte Pixel-Breite:

$$44100\ \text{Hz} * 0,12\ \text{s} = 5292\ \text{Pixel pro Zeile}$$

Aufgabe 11 — Das Rohbild inspizieren und ausrichten

Wir öffnen unsere Bytes als ungefiltertes Graustufenbild. Da Robot36 YUV nutzt, ignorieren wir die Farbe vorerst und erfreuen uns am messerscharfen Schwarz-Weiß-Bild (dem Y-Kanal).

a) *Öffne GIMP. Gehe auf `Datei -> Öffnen` und wähle deine `robot36_bild.raw` Datei aus.*

b) *Es öffnet sich der Dialog „Rohbilddaten laden“. Wähle als Bildtyp **Graustufen**.*

c) *Trage unter Breite (`Width`) den berechneten Wert von `5292` ein. Klicke auf Öffnen.*

d) *Du siehst nun ein breites, mehrfach nebeneinander liegendes Bild. Auf der linken Seite solltest du einen perfekt geraden, tiefschwarzen vertikalen Strich sehen: Das sind unsere übereinander liegenden Sync-Pulse! Direkt daneben ist das Motiv gestochen scharf zu erkennen.*

e) *Ist der schwarze Balken leicht diagonal? Das liegt an winzigen Timing-Fehlern der Oszillatoren (Doppler-Effekt der ISS). Variiere die Breite in GIMP um 1 oder 2 Pixel nach oben oder unten (z.B. auf 5293), bis der Balken exakt gerade steht („Slant Correction“).*

Hint

Die Farben: Wenn du dir das breite Bild in GIMP genau ansiehst, siehst du rechts neben dem Hauptbild (Luminanz, Y) noch schmalere Versionen des Bildes. Das sind die Farbkanäle (Chrominanz, U und V), die abwechselnd gesendet werden. Ein echtes SSTV-Programm schneidet dieses Graustufenbild an den Sync-Pulsen auseinander und rechnet die Farbwerte mathematisch auf die Helligkeitswerte, um das finale Farbbild anzuzeigen.

Abbildung 2: Das decodierte Roh-Bild. Die Bildbreite von 5292 Pixeln entspricht exakt der Dauer einer Robot36-Bildzeile.

Glückwunsch! Du hast die absolute Meisterklasse der Signalverarbeitung erreicht: Von der unsichtbaren Funkwelle, über komplexe Carrier-Recovery, Bit-Slicing und Matched-Filter bis hin zum rohen, dekodierten Bild-Pixel. Das GNUradio-Diplom gehört dir!