Proseminar Datenkompremierung

Skalare Quantisierung

Codierung durch Einteilung in Werteklassen

Reinhard Dietrich
dietrich@informatik.tu-muenchen.de

Skalare Quantisierung:

Unter skalarer Quantisierung versteht man die Repräsentation eines großen Wertebereichs (möglicherweise auch unendlich groß) durch einen kleineren (begrenzten) Bereich von Werten. Dies führt zwangsweise zu einer nicht ganz verlustfreien Komprimierung. Die Motivation ist jedoch, Daten möglichst komprimiert zu übertragen oder zu speichern, oder sie überhaupt erst reproduzierbar zu machen.
Die Grundeigenschaften der skalaren Quantisierung und einen Ausblick auf einige Anwendungen möchte ich anhand von Beispielen zeigen.

1. Beispiel: 3-Bit-Digitalübertragung analoger Werte

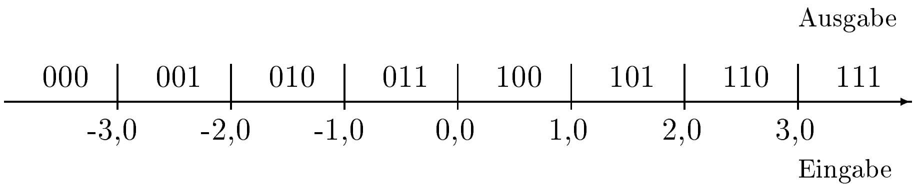
Nehmen wir an, wir haben einen Eingangswert x aus einem analogen Eingangsbereich $X := [-10V, 10V]$. Dazu gehört eine Verteilungsfunktion $f_X(x)$, die angibt, mit welcher Wahrscheinlichkeit die Quelle X einen bestimmten Wert $x \in X$ emittiert. Dabei ist die Verteilungsfunktion immer so beschaffen, daß $\int_{-\infty}^{\infty} f_X(x)\, dx = 1$ gilt. Somit ist die Quelle vollständig bestimmt durch:

$x \in [X_{min}, X_{max}]$ mit einer Wahrscheinlichkeitsverteilung $f_X(x)$

Nun kann man die analog/digital Übersetzung formulieren:
Die Anzahl m der codierten Werte c_i, die bei der Codierung angenommen werden können, ist gegeben durch:

$m := 2^n = 2^3 = 8 \;\Rightarrow\; \{c_1, c_2, \cdots, c_8\}$
n ist die Anzahl der zur Verfügung stehenden Bits.

Die Graphik zeigt eine mögliche Einteilung von X.



Zur eindeutigen Einteilung in die Werteklassen c_i müssen noch die Intervallgrenzen der c_i: $\{b_i\}_{i=0}^{m}$ festgelegt werden. Allgemein definiert man $b_0 = -\infty$ und $b_m = \infty$. Dies deutet an, daß die Eingangswerte, die nicht im Intervall X liegen - sowas kann in der Praxis schon mal vorkommen - auch mit dem Code für den kleinsten bzw. den größten Wert übertragen werden.

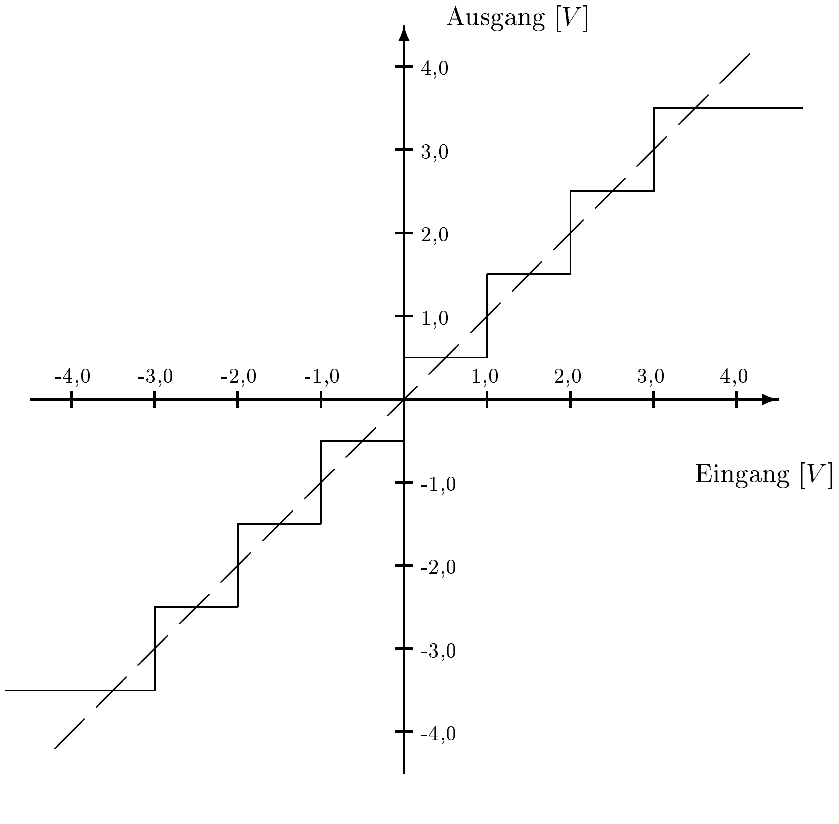
Es ist also der Wertebereich in kleine Abschnitte $[b_{i-1}, b_i] \quad 1 \leq i \leq m$ eingeteilt. Diese werden durch den Code c_i repräsentiert. Die Quantisierungsfunktion läßt sich schreiben als:

$$Q(x) = c_i \quad \text{wenn} \quad b_{i-1} < x \leq b_i$$

Diese Codierungsfunktion wird sicherlich nicht eindeutig umkehrbar sein, da sie zwar surjektiv, aber nicht injektiv ist. Die zugehörige Dekomprimierungsfunktion ordnet dem Code c_i den Wert y_i zu, der so gewählt ist, daß er den Bereich, dem c_i angehört, am besten repräsentiert (z.B. $y_i = \frac{b_{i-1}+b_i}{2}$). Eine mögliche Decodierungsfunktion für unser Beispiel läßt sich somit als Zuordnungstabelle schreiben.

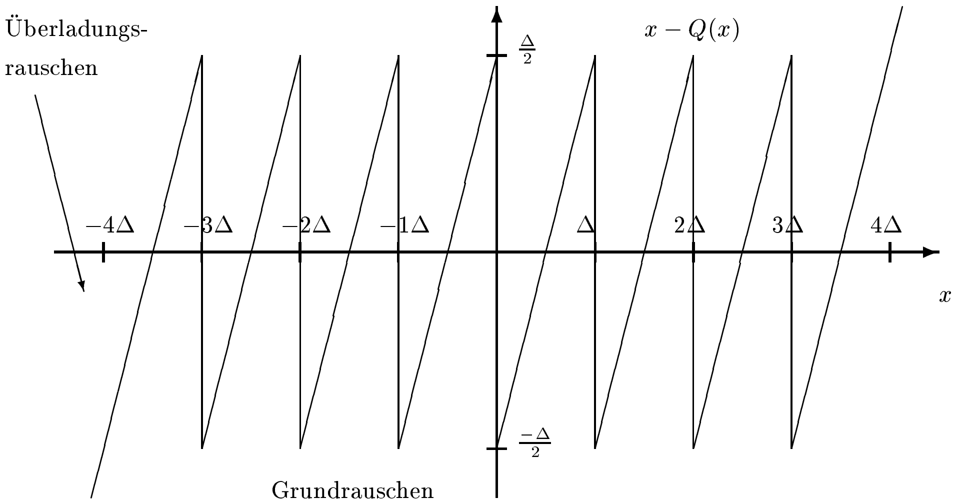
Eingabe c_i	000	001	010	011	100	101	110	111
Ausgabe $[V]$	-3,5	-2,5	-1,5	-0,5	0,5	1,5	2,5	3,5

Daß diese Art der Komprimierung sehr verlustbehaftet ist, erkennt man leicht. Man braucht nur mal zu überlegen was passiert, wenn man einen Wert b_i so darstellen will. Der übertragene Code wird c_i, und der Repräsentationswert y_i sein. Der Fehler $b_i - y_i = \frac{b_{i-1}+b_i}{2}$ ist beträchtlich.
Der Quantisierungsfehler der gesamten Übertragungsstrecke ist gut in einem Graphen, der den Ausgang Y in Abhängigkeit vom Eingang X zeigt, darstellbar.



Bei der Betrachtung dieses einen Graphen sollte man nicht vergessen, daß dieser sich aus zwei nacheinander ausgeführten Funktionen ergibt, die sich eigentlich aufheben sollten. Ein

Optimum wäre also eine Gerade durch den Ursprung mit der Steigung von eins.
Das folgende Bild stellt den Quantisierungsfehler in Abhängigkeit zur Eingangsspannung dar.



Wenn bei Quantisierern von Rauschen die Rede ist, ist das eigentlich nicht präzise ausgedrückt. Denn Rauschen ist normalerweise eine Störung einer Einheit von außen (z.B. durch elektromagnetische Wellen). Hier steckt die Störung aber im System. Sie ist zugunsten der Komprimierung mit eingeplant. Da sich aber bei der Anwendung kein Unterschied ergibt, wovon das Rauschen kommt, nennt man den Quantisierungsfehler auch Rauschen.
Nun gilt es ein Maß für den Quantisierungsfehler $|x - Q(x)|$ der Übertragungsstrecke zu finden. Dabei muß auch die Abhängigkeit von der Verteilungsfunktion $f_X(x)$ berücksichtigt werden. Eine Möglichkeit den Quantisierungsfehler allgemein zu definieren, ist ihn quadratisch anzugeben:

$$\sigma_q^2 = \int_{-\infty}^{\infty} (x - Q(x))^2 \; f_X(x) \; dx = \sum_{i=1}^{m} \int_{b_{i-1}}^{b_i} (x - y_i)^2 \; f_X(x) \; dx$$

Interessant dabei ist das Verhältnis:

$$\frac{\text{Eingangssignal}}{\text{Rauschen}}$$

Dieser Signal-Rausch-Abstand (SNR, engl. signal-to-noise-ratio) berechnet sich aus:

$$\begin{aligned} \text{SNR} &= \frac{\sigma_s^2}{\sigma_q^2} \\ \text{SNR [dB]} &= 10 \log_{10} \frac{\sigma_s^2}{\sigma_q^2} \end{aligned}$$

Dabei ist σ_q^2 der Quantisierungsfehler wie definiert, und σ_s^2 ist die gewichtete quadrierte Eingangsspannung. Das Ziel der Fehlerbetrachtungen ist es, zu einer bestimmten Eingangsverteilung $f_X(x)$ und einer vorgegebenen Anzahl von Bit für die Übertragung, die Intervallgrenzen b_i so zu legen, daß der Fehler σ_q^2 minimal (und somit der SNR möglichst groß) wird.

2. Beispiel: der Fehler eines Uniform Quantizer

Der Quantisierungsfehler einer gleichverteilten Quantisierungsfunktion an einer gleichverteilten Quelle.
Da dieser Ansatz für einen Quantisierer wohl der häufigste ist, lohnt es sich, ihn speziell zu betrachten. In diesem Fall wird die Quantisierung mit gleichen Abständen der b_i durchgeführt. Es gilt: $b_i - b_{i-1} = \Delta$.
Dabei kann man noch eine Unterscheidung treffen: zwischen "midrise" und "midtread quantizer". Man sieht leicht, daß das abhängig davon ist, ob die Anzahl der Intervalle $[b_{i-1}, b_i]$ gerade oder ungerade ist. Der Quantisierer im 1. Beispiel ist ein "midrise quantizer". Betrachten wir den in der Digitaltechnik häufiger auftretenden Fall des "midrise quantizers". Der Eingangsbereich X soll zudem symmetrisch und gleichverteilt sein. Somit gilt:

$$f_X(x) = \begin{cases} \frac{1}{2X_{max}} & : \ x \in [-X_{max}, X_{max}] \\ 0 & : \ \text{sonst} \end{cases}$$

Die Länge des Intervalls Δ ergibt sich zu:

$$\Delta = \frac{2X_{max}}{m}$$

Um den durchschnittlichen quadrierten Quantisierungsfehler berechnen zu können, müssen wir $Q(x)$ in Abhängigkeit von dem Intervall i, indem wir uns mit x befinden, darstellen.

$$Q(i) = \frac{2i-1}{2}\Delta$$

Jetzt können wir den Fehler ausrechnen:

$$\begin{aligned}
\sigma_q^2 &= \int_{-\infty}^{\infty} (x - Q(x))^2 \ f_X(x) \ dx \\
&= \sum_{i=1}^{m} \int_{(i-1)\Delta}^{i\Delta} \left(x - \frac{2i-1}{2}\Delta\right)^2 \ \frac{1}{2X_{max}} \ dx \\
&= \frac{m}{2X_{max}} \int_0^{\Delta} \left(x - \frac{1}{2}\Delta\right)^2 \ dx \\
&= \frac{m}{2X_{max}} \left[\int_0^{\Delta} x^2 \ dx \ + \int_0^{\Delta} \Delta x \ dx \ + \int_0^{\Delta} \frac{\Delta^2}{4} \ dx\right] \\
&= \frac{m}{2X_{max}} \left(\frac{1}{3}\Delta^3 + \frac{1}{2}\Delta^3 + \frac{1}{4}\Delta^3\right) \\
&= \frac{m}{2X_{max}} \frac{1}{12}\Delta^3 \\
&= \frac{1}{\Delta}\frac{1}{12}\Delta^3 \\
&= \frac{1}{12}\Delta^2
\end{aligned}$$

Als Vergleichswert σ_s^2:

$$\sigma_s^2 \ := \ \sum_{i=1}^{m} \int_{b_{i-1}}^{b_i} x^2 \ f_X(x) \ dx$$

$$
\begin{aligned}
&= \int_{-X_{max}}^{X_{max}} x^2 \frac{1}{2X_{max}}\, dx \\
&= \frac{1}{2\, X_{max}} \left[\frac{1}{3}x^3\right]_{-X_{max}}^{X_{max}} \\
&= \frac{(2X_{max})^2}{12}
\end{aligned}
$$

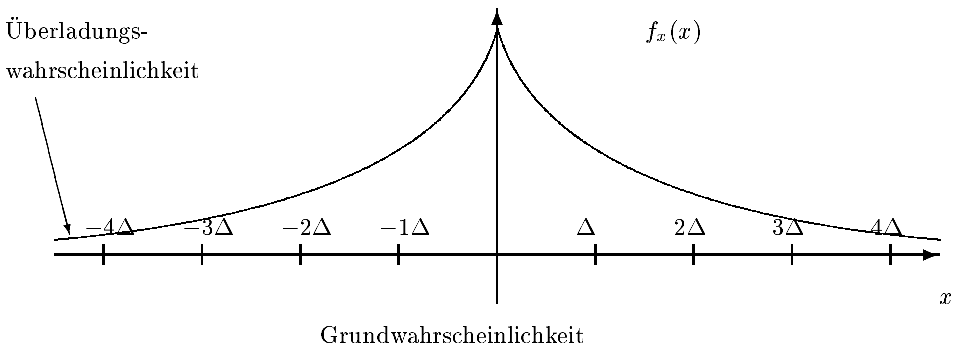
Und das in die Formel für den Signal-Rausch-Abstand eingesetzt:

$$
\begin{aligned}
\text{SNR [dB]} &= 10\ \log_{10} \frac{\sigma_s^2}{\sigma_q^2} \\
&= 10\ \log_{10} \left(\frac{(2X_{max})^2}{12} \cdot \frac{12}{\Delta^2}\right) \\
&= 10\ \log_{10} \left(\frac{(2X_{max})^2}{12} \cdot \frac{12}{\left(\frac{2X_{max}}{m}\right)^2}\right) \\
&= 10\ \log_{10} m^2 \\
&= 20\ \log_{10} 2^n \\
&= 6{,}02n \text{ dB}
\end{aligned}
$$

Bei einer Musik-CD, die mit 16 Bit aufgenommen ist, ergibt sich theoretisch ein Signal-Rausch-Abstand von ca. 96 dB. In der Praxis werden diese Werte jedoch nur schwer erreicht, da ja zu dem systembedingten Rauschen noch das echte, von außen verursachte Rauschen dazukommt.

3. Beispiel: Huffman-Codierung

Bei einer ungleich verteilten Quelle kann man die Repräsentationswerte c_i nach Huffman codieren.



Code variabler Länge (nach Huffman):

c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8
1110	1100	100	00	01	101	1101	1111

Dabei ergibt sich die Wahrscheinlichkeit für ein Zeichen zu:

$$P(c_i) = \int_{b_i - 1}^{b_i} f_X(x)\ dx$$

Man beachte, daß aus der Forderung $\int_{-\infty}^{\infty} f_X(x)\ dx = 1$ folgt: $\sum_{i=1}^{m} P(c_i) = 1$. Damit ist die durchschnittliche Codelänge:
(l_i ist die Anzahl der Bits von c_i.)

$$\begin{aligned} R &= \sum_{i=1}^{m} l_i P(y_i) \\ &= \sum_{i=1}^{m} l_i \int_{b_i - 1}^{b_i} f_x(x)\ dx \end{aligned}$$

Nun können wir die Problemstellung der Quantisierung allgemein formulieren:

> Finde zu einem maximal zulässigem Fehler $\sigma_q^2 \leq D^*$ alle Intervallgrenzen, Rekonstruktionswerte und Binärcode, so daß R möglichst klein wird,

oder,

> finde zu einer maximal zulässigen Länge von $R \leq R^*$ alle Intervallgrenzen, Rekonstruktionswerte und Binärcode, so daß σ_q^2 nicht zu groß wird.

Hierzu gibt es viele Ansätze die bezüglich einer konkreten Anwendung die Probleme lösen. Oft werden dazu numerische Näherungsverfahren verwendet.
Eine einfachere Möglichkeit möchte ich im nächsten Beispiel aufzeigen.

4. Beispiel: adaptive Komprimierung

Dabei wird die Quelle X eine bestimmte Zeit untersucht und der Quantisierer dann auf die Verteilung $f_X(x)$ angepaßt. Man unterscheidet in Vorwärts- und Rückwärtsadaption. Bei der Vorwärtsadaption wird die Quelle erst untersucht, und dann für den untersuchten Bereich der Quantisierer angepaßt. Das hat den Vorteil, daß die Quantisierung richtig angepaßt erfolgt. Dazu muß aber der zu untersuchende Bereich zwischengespeichert werden.
Die Rückwärtsadaption ist dagegen echtzeitfähig. Bei ihr wird der Quantisierer nach den bisher eingegangenen Werten eingestellt. Die zur Anpassung des Decodierers notwendige Information muß mit gespeichert werden. Dadurch wird der Code länger. Es liegt an der Qualität des Adaptionsverfahrens, wie stark sich das Signal-Rausch-Verhältnis verbessert, ohne daß der erzeugte Code zu lang wird. Ein sehr einfaches Beispiel sei hier kurz vorgestellt: Die Quantisierung eines Bildes mit 256 Grauwerten.

Originalbild mit 8 Bit

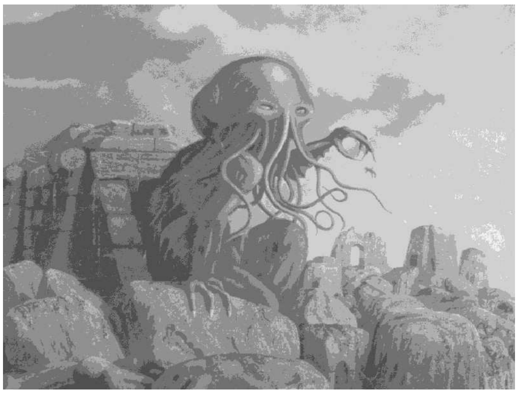
Der Eingangsbereich $X = \{0, \cdots, 255\}$. Hierfür werden 8 Bit pro Pixel benötigt. Will man das Bild nun mit nur einem Bit pro Pixel darstellen, so wird man den Eingangsbereich X in zwei Hälften aufteilen: $[0, 127]$ und $[128, 255]$. Jede Hälfte wird nun durch einen Grauwert repräsentiert, der in der Mitte des Intervalls liegt. In diesem Fall 64 und 196. Das Ergebnis:

Mit einem Bit quantisiert

Man sieht, daß hier sehr viel Information verloren geht. Um mehr Information über das Bild zu erhalten, versuchen wir dasselbe mit zwei Bit pro Pixel. Die Repräsentationswerte des in vier gleichverteilte Intervalle zerlegten Eingangsbereichs sind dann die vier Graustufen $32, 96, 160$ und 224.

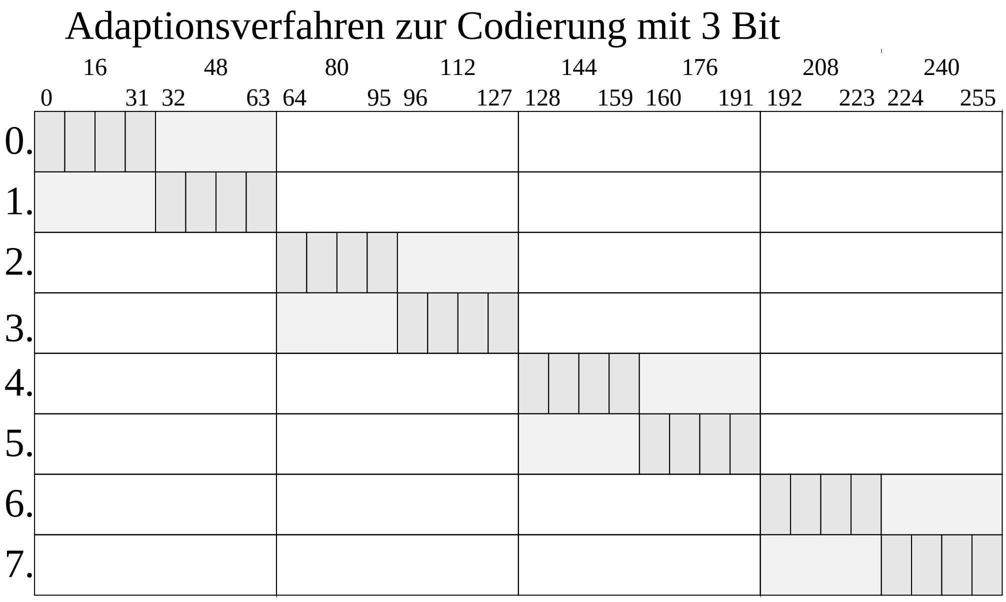
Mit zwei Bit quantisiertes Bild

Da die Qualität dieses Bildes immer noch zu wünschen übrig läßt, ein weiterer Versuch mit drei Bit pro Pixel. Die Einteilung der Quantisierungsbereiche und deren Repräsentanten sind in der Graphik auf der folgenden Seite ersichtlich.



Mit drei Bit quantisiertes Bild

An diesem Bild sieht man den Schwachpunkt der gleichmäßigen Quantisierung sehr gut. In den Bereichen, in denen die Werte sehr dicht liegen, wie z.B. in den Wolken, geht die Information vollständig verloren. Hier setzt die Idee der adaptiven Codierung an. Die folgende Graphik zeigt eine andere mögliche Einteilung der Werteklassen. Dabei werden acht verschiedene Einteilungen zurechtgelegt.



Einteilung der Werteklassen für Drei-Bit- und adaptive Drei-Bit-Quantisierung

Diese sind so aufgabaut, daß sie bestimmte Bereiche der Grauverteilung besonders genau, andere dafür weniger genau darstellen. So wird z.B. bei der Verteilung 0 der Eingangsbereich [0..255] in die Interwalle [0..7], [8..15], [16..23], [24..31], [32..63], [64..127], [128..191] und [192..255] eingeteilt. Insgesamt bleiben es aber immer acht Intervalle - drei Bit. In der Graphik grau schattiert sieht man die Schwerpunkte der Quantisierungsgenauigkeit. Die Information, welche Verteilung der Bereiche gewählt wurde, muß natürlich auch mit abgespeichert werden. Würde man nun für jedes Pixel eine andere Quantisierung wählen, so bekäme man sechs Bit pro Pixel, was dem Speicherplatz des Orginalbildes mit acht Bit doch schon sehr nahe käme. Es ist aber bei Bildern immer so, daß größere Bereiche (z.B. ein Pinselstrich) die gleiche, oder eine ähnliche, Farbe haben. Somit kann man einen größeren, festgelegten Bereich mit der selben Quantisierungsmethode bearbeiten. Um nun herauszufinden nach welcher Verteilungsfunktion quantisiert werden soll, führt man einfach eine Statistik, in welchem Bereich am meisten Grauwerte liegen. Eine Implementierung in C könnte folgendermaßen aussehen:

```
#define BUFFERLENGTH 64
int sb[BUFFERLENGTH];         /* "searchbuffer" */

int suchprio (int sb[]){                        /* Statistik */
  int prio[8];                /* Werte der Intervalle zählen */
  int i, aktprio, w;                  /* aktuelle Prioritaet */

  for(i = 0; i <= 7; i++)
    prio[i] = 0;                          /* Startbedingung */

  for(i = 0; i < BUFFERLENGTH; i++)
      prio[sb[i]/32]++;       /* Auswertung des searchbuffer */
```

```
  w = 0;
  for(i = 0; i <= 7; i++)
    if (prio[i] > w){
      aktprio = i;
      w = prio[i];
    };              /* jetzt zeigt aktprio in welchen Inter- */
    /* vall die meisten Grauwerte liegen */
  return aktprio;
};
```

Ist nun für ein Intervall bestimmter Länge (hier 64 Werte) die Verteilungsfunktion ausgewählt, können die Pixel quantisiert werden. Dieses Verfahren arbeitet somit vorwärtsadaptiv. Die folgende C-Funktion simuliert die Quantisierung der Pixel. Sie gibt für ein Pixel den Repräsentationswert nach einer gewählten Verteilung zurück. Dies entspricht der Repräsentation der quantisierten Werte.

```
int codiere (int x, int prio){
/* codiert nach wählbarer Prioritaet */

  /* Intervallbegrenzungen */
  int b[8][9] = {
    0,  7,  15,  23,  31,  63, 127, 191, 255,
    0, 31,  39,  47,  55,  63, 127, 191, 255,
    0, 63,  71,  79,  87,  95, 127, 191, 255,
    0, 63,  95, 103, 111, 119, 127, 191, 255,
    0, 63, 127, 135, 143, 151, 159, 191, 255,
    0, 63, 127, 159, 167, 175, 183, 191, 255,
    0, 63, 127, 191, 199, 207, 215, 223, 255,
    0, 63, 127, 191, 223, 231, 239, 247, 255 };
  int pos;

  for (pos=1; b[prio][pos] < x; pos++);
  return (b[prio][pos]+b[prio][pos-1])/2;
};
```

Das Resultat der adaptiven Quantisierung ist im Vergleich zu dem nicht adaptiv quantisiertem Bild beträchtlich verbessert. Probleme entstehen jedoch bei sprunghaftem Wechsel der Grautöne (das Bild wirkt gestreift). Hier sollten bessere Adaptionsverfahren, z.B. mit Rechtecken arbeitende Verfahren, eine weitere Verbesserung bringen. Die 3,05 Bit pro Pixel entstehen durch die mitgespeicherte Adaptionsinformation. Zu je $64 * 3$ Bit kommen 3 Bit, die die Verteilung angeben, hinzu.

$$
\begin{aligned}
R &= \frac{64 * 3\text{Bit} + 3\text{Bit}}{64\text{Pixel}} \\
&\approx 3.05 \frac{\text{Bit}}{\text{Pixel}}
\end{aligned}
$$

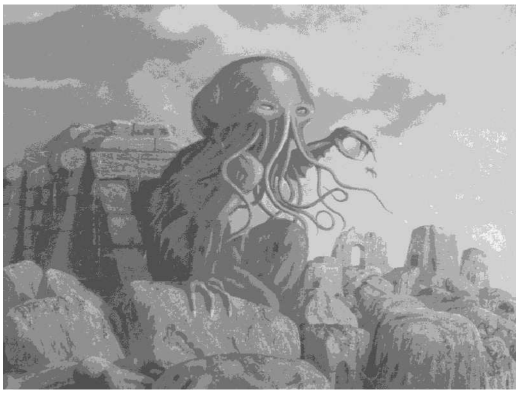


Bild mit drei Bit

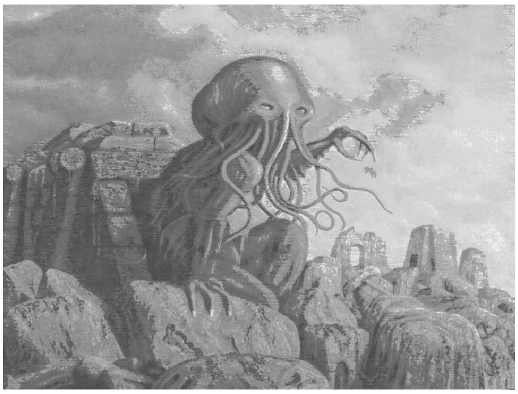
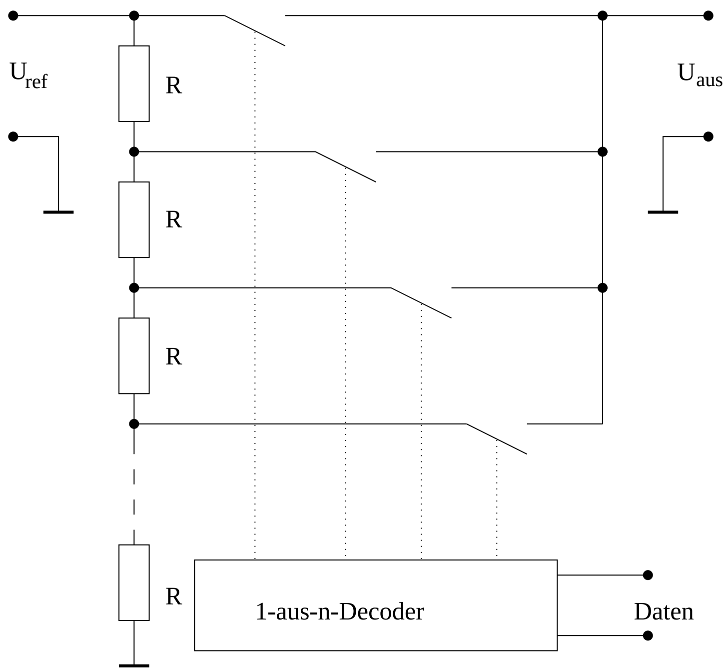


Bild mit 3,05 Bit (adaptiv)

Ein Ausblick: technische Realisierungen

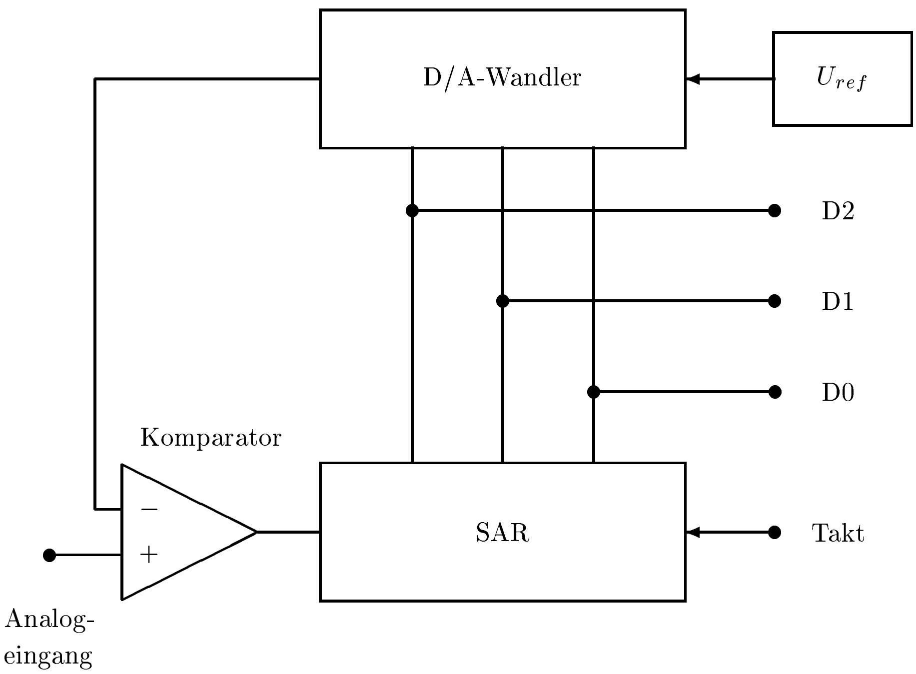
Zum Schluß möchte ich einen kleinen Ausflug in die Praxis machen. Wie können Analog/Digital- und Digital/Analog-Wandler technisch realisiert werden?
Sei mit der Digital/Analog-Wandlung begonnen. Die einfachste Möglichkeit ist das Parallelverfahren.



Parallelverfahren

Dabei werden über eine Serienschaltung von Widerständen alle möglichen Ausgangsspannungen erzeugt. Durch einen 1-aus-n-Decoder wird immer genau ein Spannungsniveau über einen der Schalter auf den Ausgang geschaltet. Für einen n-Bit-Umsetzer werden also 2^n Widerstände und Schalter benötigt. Dies ist technisch jedoch schwer zu realisieren.
Eine bessere Möglichkeit bietet das Wägeverfahren, daß durch geschickte Anordnung der Widerstände und Schalter mit n Schaltern und $2n$ Widerständen auskommt. Diejenigen, die jetzt die Neugierde gepackt hat, möchte ich auf [DEM91] verweisen, da ein tieferer Einblick in das Thema der Realisierung von Wandlern Bücher füllt.

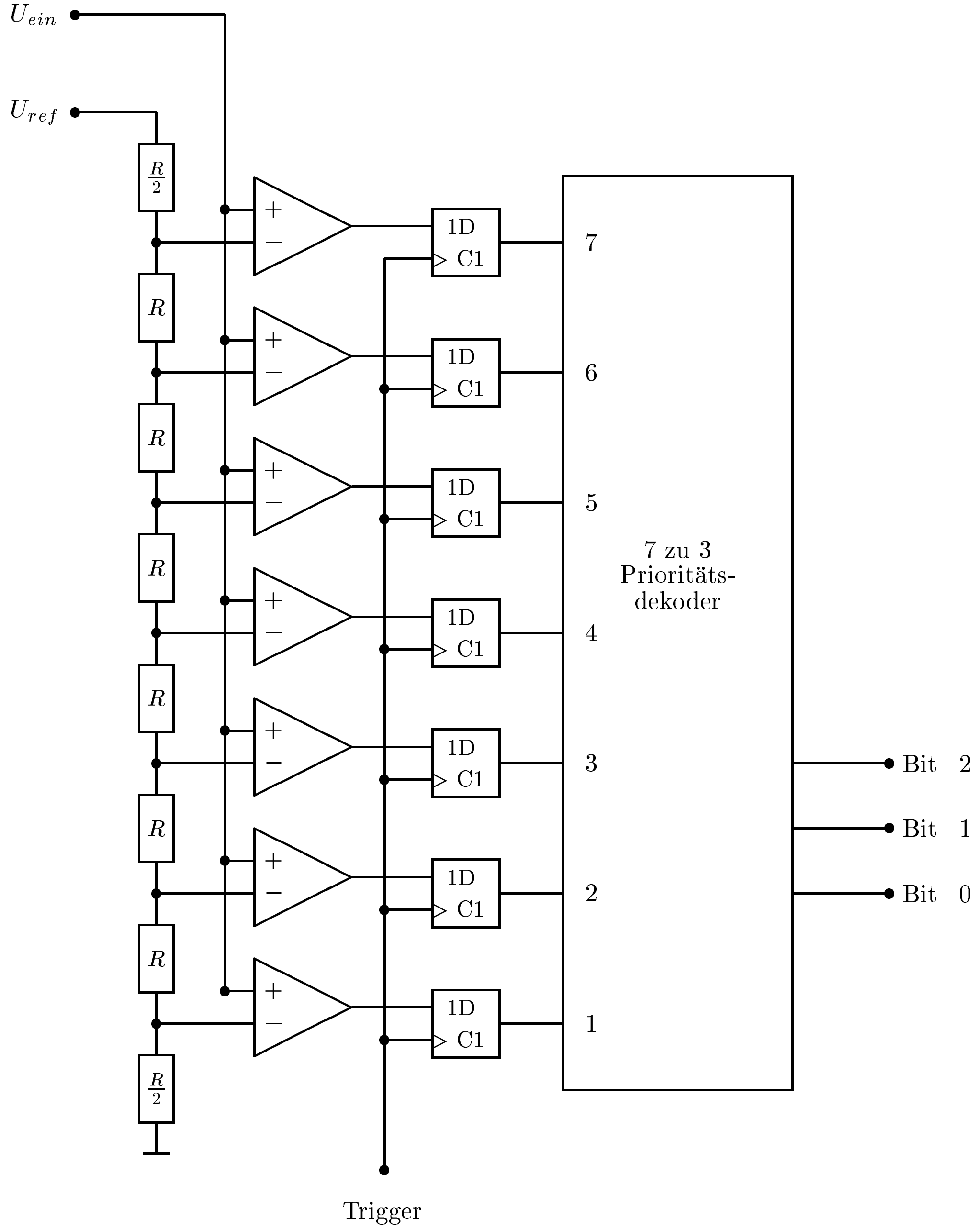
Das Gegenstück ist der Analog/Digital-Wandler. Dieser kann z.B. nach dem Prinzip der sukzessiven Approximation gebaut werden. In der Funktionsweise entspricht das einem Wäge- oder Annäherungsverfahren. Dabei wird die Eingangsspannung mit einer, über einen D/A-Wandler erzeugten Spannung verglichen. Stimmen die Spannungen überein, entspricht die Ansteuerung des D/A-Wandlers der Eingangsspannung. Die Annäherung der intern erzeugten Spannung erfolgt über ein spezielles Register zur sukzessiven Approximation (SAR). Dazu wird mit dem höchstwertigen Bit (MSB) begonnen. Ist die erzeugte Spannung im Vergleich zur Eingangsspannung nach dem Setzen dieses Bits immer noch kleiner, so bleibt es gesetzt, ansonsten wird es wieder gelöscht. Dieses "Abwägen" wird mit den niederwertigeren Bits bis zum LSB fortgesetzt. Danach stehen die Digitalausgänge zur Verarbeitung bereit.



A/D-Wandler nach dem Prinzip der sukzessiven Approximation

Es ist klar, daß dieses Verfahren zur Wandlung einige Zeit benötigt, denn die stufenweise Approximation geschieht in einzelnen Schritten. Dadurch erreichen Bausteine dieser Bauart in der Praxis nur eine Wandlungszeit größer ca. 10μs.

Eine andere Bauart stellt der Flash-Wandler dar. Dieser vergleicht die Eingangsspannung mit den, an dem internen Spannungsteiler anstehenden Referenzspannungen. Das Ergebnis des Vergleichs wird dann mit einem Prioritätsdekoder in ein binäres Ausgangssignal gewandelt. Das Ausgangswort eines Prioritätsdekoders entspricht immer dem höchstwertigen Eingangsbit, daß high ist.



Drei-Bit-Flash-Wandler

Diese Wandler sind sehr schnell. Sie erreichen Wandlungsraten von bis zu 200 Msps (Millionen Samples pro Sekunde). Doch sie haben die gleichen bauartbedingten Probleme wie die Parallel-D/A-Wandler. Sie benötigen für eine Wandlung in n Bit 2^n Komparatoren und Widerstände. Auch hierfür gibt es Lösungen, wie z.B. den Semi-Flash-Wandler. Dazu, und zu vielen weiteren Verfahren, möchte ich jedoch wieder auf [DEM91] verweisen.

Literatur

[SAY96] K. Sayood. *Data Compression.* Morgan Kaufmann, 1996

[DEM91] Klaus Dembowski. *PC-gesteuerte Meßtechnik.* Markt Technik Verlag AG, 1991

[WAT91] John Watson. *MASTERING ELECTRONICS.* MACMILLAN EDUCATION LTD, 3rd ed. 1991