Dr. Frank Schöpfer

Einführung in Matlab

Übung 8

Aufgabe 1:

(a) Schreiben Sie (analog zu Aufgabe 1 aus Übung 7) eine Funktion `cw=codewort2(t,z)` zum **Erzeugen der elementaren Codewörter**, wobei diesesmal der **Huffman-Baum als Matrix** `t` sowie die Zeichen `z` übergeben werden sollen.

Testen Sie Ihr Programm an dem Beispiel aus der Vorlesung.

(b) **Anwendung bei einem `uint8`-Grauwertbild**:

- Bild laden und zu einem langen Vektor machen, z.B.

```
>> X=imread('coins.png');
>> v=X(:);
```

- Relative Häufigkeiten `w` der Grauwert-Intensitäten `z=0:255` in `v` bestimmen.
- Zugehörigen Huffman-Baum `t` sowie die Codewörter `cw` erstellen.
- Vektor `v` damit kodieren.
- Anzahl an Bits zur Speicherung des Original-Bildes und des Huffman-kodierten Bildes vergleichen; dabei kommen noch die Bits zum Speichern des Huffman-Baumes hinzu, z.B. als `uint16`, denn `uint8` genügt hier nicht (wieso?)
- Zur Kontrolle auch wieder dekodieren.

```
>> vd=dekodiere2(c,t,z);  % dekodiere2 aus Vorlesung für t als Matrix
>> Xd=reshape(vd,size(X));  % macht aus langem Vektor eine Matrix
                            % mit gleicher Dimension wie X
>> image(Xd); colormap(gray(256))
```

(c) Um mehr Speicher sparen zu können, kann man das **Bild gröber Quantisieren um mehr gleiche Zeichen zu erzeugen** (das ist dann zwar mit Verlusten behaftet, aber das Auge sieht oft keine großen Unterschiede). Dadurch werden insgesamt auch weniger verschiedene Zeichen benötigt. Hier fassen wir immer zwei aufeinanderfolgende Zahlen zusammen, d.h. wir bilden 0,1 auf 0 und 2,3 auf 1 und ... 254,255 auf 127 ab, so dass wir auch nur noch die Zeichen 0 bis 127 verwenden müssen. Zuvor müssen wir noch aus dem `uint8`-Vektor `v` einen Vektor mit `double`-Werten machen, da wir beim Rechnen den `uint8` Bereich verlassen.

```
>> zq=0:127; vq=round((double(v)+1)/2)-1;
```

Verfahren Sie nun ananlog zu (b) (genügt nun `uint8` für `t`?). Um nach dem Dekodieren wieder Werte in 0 bis 255 zu erhalten, dann z.B. einfach hochskalieren und wieder `uint8` daraus machen.

```
>> vqd=dekodiere2(cq,tq,zq); vqd=uint8(round(vqd*255/127));
```

Vergleichen Sie dann auch das Original mit dem quantisierten Bild.

(d) Um noch mehr Speicher zu sparen, kann man einfach noch mal quantisieren auf 0 bis 63, usw. Schreiben Sie dafür eine Funktion `x=quantisiere(x,n)` um n-mal nacheinander ein (`uint8`-)Array `x` zu quantisieren. Testen Sie auch an einem `uint8`-**RGB-Bild**, z.B. beim Trailer-Bild `X=imread('trailer.jpg')`. Vergleichen Sie hier einfach nur die quantisierten Bilder mit dem Original, um zu sehen wie sich **mehrfaches Quantisieren** auf ein Bild auswirkt, denn hier dauert das (De-)Kodieren zu lange.