FFHS
Fernfachhochschule Schweiz
Zürich | Basel | Bern | Brig
Mitglied der SUPSI

Fernfachhochschule Schweiz
BSc Informatik

Signal-/Bildkomprimierung mit der disktreten Kosinustransformation

Semesterarbeit im Modul LinAlg, des Studiengangs BSc Informatik

von

Tobias Gasche

Dozent:
Prof. Dr. Matthias Dehmer

4554 Etziken, 08. Dezember 2024

Zusammenfassung

In der Informatik wird oft das Ziel verfolgt, eine Information mit möglichst wenig Aufwand zu verarbeiten.
Sei dies möglichst wenig CPU-Zeit zum Berechnen einer Darstellung, möglichst wenig Platzbedarf auf einer Speicherplatte (oder einem Cloud-Speicher), oder möglichst wenig zu übertragende Information in einer Netzwerk-Datenübertragung.
Um dieses «möglichst wenig» zu erreichen gibt es diverse Vorgehen, wie eine Datei komprimiert werden kann. In dieser Arbeit wird die diskrete Kosinus Transformation (DCT) vorgestellt und behandelt. Diese wird zum Beispiel im bekannten JPEG-Dateiformat verwendet, um grosse Bilddateien zu komprimieren.
Eine Kompression bedeutet in diesem Zusammenhang, dass die Datei durch Weglassen von Informationen verkleinert wird. Die Frage stellt sich, welche Informationen in einer Datei entsprechend vernachlässigbar sind, dass ihr Weglassen nicht bemerkt wird.

Inhaltsverzeichnis

1 Einleitung

Bereits im Jahr 1992 wurde durch das «International Telegraph and Telephone Consultative Commitee» (CCITT) und die «International Telecommunication Union» (ITU) die CCITT Empfehlung T.81 veröffentlicht [1].
Das Paper beinhaltet das Vorgehen zur Komprimierung von Bildern, genauer zur «digitalen Komprimierung und Kodierung von Halbton-Standbildern».
Die Empfehlung ist das Resultat der Forschung einer 1986 gegründetend Expertengruppe, bestehend aus der «Studiengruppe VII» des CCITT und der «Joint Photographic Experts Group» der internationalen Standardisierungsorganisation ISO. Da die Gruppe aus Mitgliedern der CCITT und der ISO bestand, wurde der selbe Text auch als ISO 10918-1 veröffentlicht.
Die Abkürzung der «Joint Photographic Experts Group» ist bis heute als Dateiname für gewisse komprimierte Bilder geblieben: JPEG.
Bei der JPEG-Dateiendung handelt es sich also nicht um eine Abkürzung der Programmiersprache einer Datei («.java», «.php», «.js» usw.), oder das Format («.pdf» für «Portable document format»), sondern um die Abkürzung der Forschungsgruppe, die an der Findung des Komprimierungsvorgehens beteiligt war.

Um ein Bild zu komprimieren, lautet die Empfehlung, das Originalbild in Blöcke von 8x8 Pixel zu zerlegen, dann durch den «DCT-basierten encoder» zu senden, was zu einem komprimierten Bild führt (Abbildung 1). Die gleiche Idee funktioniert soll auch rückwärts funktionieren: Ein komprimiertes Bild wird durch einen «decoder» wieder zur Originaldatei.

Im DCT-basierten encoder wird, ziemlich vereinfacht definiert, berechnet, welche Informationen des Bildes unnötig sind, weil sie vom menschlichen Auge nicht erfasst werden können (oder eine bestimmte Grenze unterschreiten). Diese Informationen werden dann im komprimierten Bild einfach weggelassen.

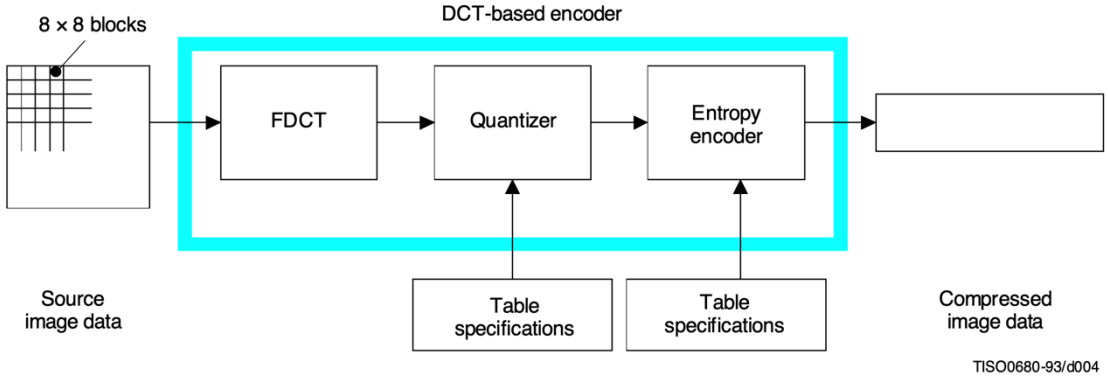

Abbildung 1 Vorgehen nach CCITT T.81 [1]

2 Theorie

Die JPEG-Komprimierung bedient sich der diskreten Kosinustransformation, die in 2.1 näher vorgestellt wird.

2.1 Diskrete Kosinustransformation (DCT)

2.1.1 Grundsatz Idee

Die Idee der Diskreten Kosinustransformation besteht darin, ein gegebenes Signal in einen Frequenzbereich umzuwandeln.
Zum Einstieg sei sei eine Kosinusfunktion $f(x) = \cos(x)$ gegeben. In Abbildung 2 wird die Funktion zwischen 0 und π betrachtet. Sie hat den höchsten Wert an der Stelle $x = 0$ und den tiefsten Wert an der Stelle $x = \pi$.

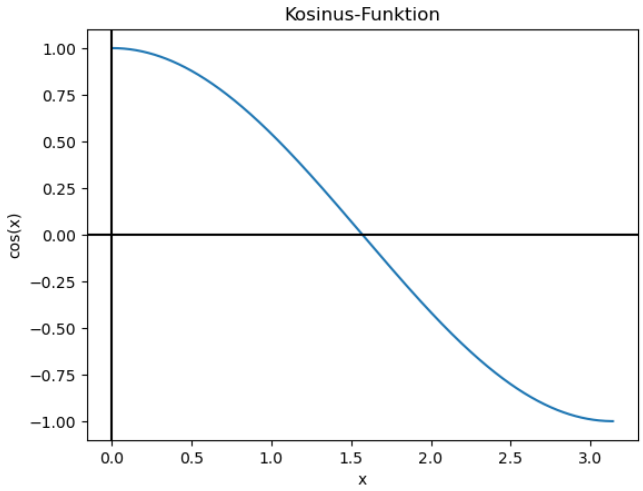

Abbildung 2 Kosinusfunktion

Diese Kurve wird anschliessend mit 3 Stützstellen «gesampelt», also abgetastet. Dazu wird das Intervall von 0 bis π in drei gleichgrosse Teile unterteilt und die Stützstellen in den Mitten dieser Teile definiert. Es resultieren $x_1 = \frac{\pi}{6}, x_2 = \frac{\pi}{2}, x_3 = \frac{5\pi}{6}$ $(x_k = \frac{2k+1}{2n} * \pi)$ und daraus die zugehörigen y-Werte $f(x_1) = \sim 0.866, f(x_2) = 0, f(x_3) = \sim -0.866$.
Stelle die Kosinuskurve den Farbraum von schwarz bis weiss mit einschliessenden Graustufen dar, so entspricht $f(x_2) = 0$ dem «mittlersten» Grau, welches man sich vorstellen kann. $f(x_3)$ ist nahe an tiefschwarz und $f(x_1)$ entsprechend ein leichtes Grau, nahe weiss.
Wird dasselbe Vorgehen für die Funktion $f(x) = \cos(0)$ angewendet, resultiert an allen drei Stützstellen $f(x_1) = f(x_2) = f(x_3) = 1$, also immer weiss.
Eine dritte Funktion soll $f(x) = \cos(2 * x)$ sein. Hier resultieren die Werte $f(x_1) = \cos\left(\frac{\pi}{3}\right) = 0.5\,, f(x_2) = \cos(\pi) = -1\,, f(x_3) = \cos\left(\frac{5\pi}{3}\right) = 0.5$
Mit diesen drei Vektoren $v_1 = \left(\cos\left(\frac{\pi}{6}\right), 0, \cos\left(\frac{5\pi}{6}\right)\right), v_2 = (1,1,1), v_3 = (0.5, -1, 0.5)$ kann jedes Bild, das aus drei Graustufen-Pixel besteht, dargestellt werden. Sie bilden also eine Basis für einen dreidimensionalen Raum.
Abbildung 3, Abbildung 4 und Abbildung 5 zeigen diese Kurven mit ihren Stützstellen.

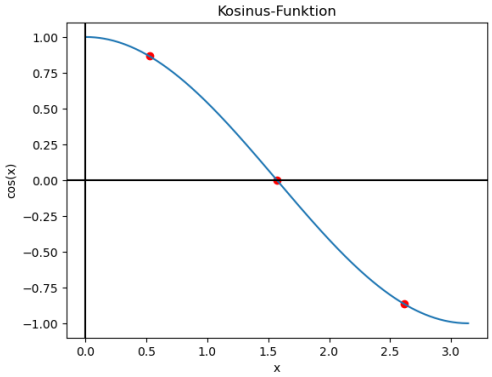

Abbildung 3 f(x) = cos(x)

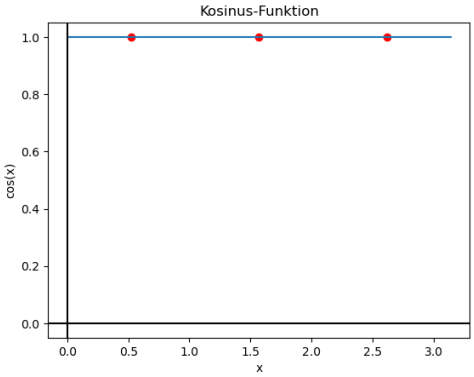

Abbildung 4 f(x) = cos(0)

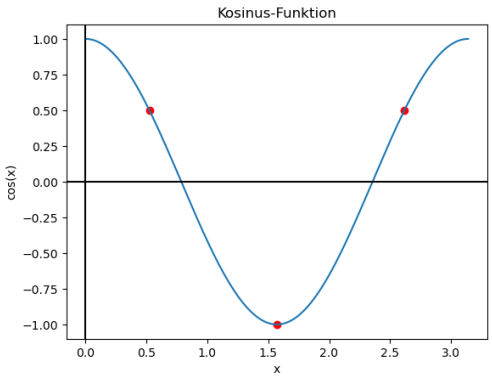

Abbildung 5 f(x) = cos(2x)

2.1.2 Allgemeine Definition

Die in 2.1.1 definierten Basisvektoren durch $f(x) = \cos(0), f(x) = \cos(x), f(x) = \cos(2x)$ sind nicht das Ende der Definition der DTC. Sie dienen lediglich der Anschauung.
Die allgemeine Form der DTC lautet nach [2]

$$C(u) = a(u) \sum_{x=0}^{N-1} f(x) \cos\left[\frac{\pi(2x+1)u}{2N}\right] \quad \text{für } x \in \{0,1,,2,\dots,N-1\}$$

$$a(u) = \begin{cases} \sqrt{\frac{1}{N}}, & u = 0 \\ \sqrt{\frac{2}{N}}, & u \neq 0 \end{cases}$$

Konkret bedeutet dies, dass jeder gesampelte Wert eines Signals als Summe verschieden schnell schwingender Kosinuskurven dargestellt wird. u stellt dabei den Frequenzindex dar.

Wird also ein Signal genau einmal gesampelt (an einer einzigen Stelle abgetastet), so ist $N = 1$.

Die Berechnung lautet dann gemäss obiger Formel

$$C(0) = 1\sum_{x=0}^{0} f(0)\cos\left[\frac{\pi(2x+1)*0}{2}\right]$$
$$= f(0)*\cos(0) = f(0)*1$$
$$C(0) = f(0)$$

Bei genau einer Samplestelle resultiert also wieder der eingegebene Wert.

Für $N = 2$ resultiert

$$C(0) = \sqrt{\frac{1}{2}}\sum_{x=0}^{1} f(x)\cos\left[\frac{\pi(2x+1)*0}{4}\right]$$

Der $\cos$ []-Teil wird wegen der 0 im Zähler immer $\cos(0) = 1$ sein. Die Summe daher $f(0) + f(1)$ und demnach $C(0) = \sqrt{\frac{1}{2}}\big(f(0) + f(1)\big)$

Für $C(1)$ gilt

$$C(1) = \sqrt{\frac{2}{2}}\sum_{x=0}^{1} f(x)\cos\left[\frac{\pi(2x+1)*1}{4}\right]$$

Der $\cos$ []-Teil lautet für $x = 0$: $\cos\left[\frac{\pi(2*0+1)*1}{4}\right] = \cos\left[\frac{\pi}{4}\right] = \frac{1}{\sqrt{2}}$

Und für $x = 1$: $\cos\left[\frac{\pi(2*1+1)*1}{4}\right] = \cos\left[\frac{3\pi}{4}\right] = -\frac{1}{\sqrt{2}}$

Daraus folgt $C(1) = 1\left(f(0)*\frac{1}{\sqrt{2}} + f(1)*-\frac{1}{\sqrt{2}}\right)$ respektive $C(1) = \frac{1}{\sqrt{2}}\big(f(0) - f(1)\big)$

Soll die Transformation hingegen mit einer Transformationsmatrix berechnet werden, so muss diese nach der Regel

$$M[k,n] = \begin{cases} \sqrt{\frac{1}{N}}, & k = 0 \\ \sqrt{\frac{2}{N}} * \cos\left(\frac{\pi(2n+1)*k}{2N}\right), & x \neq 0 \end{cases}$$

errechnet werden.

Um etwas mehr Klarheit zu schaffen, sei eine Pixelfolge von drei Pixeln mit den Werten 255, 128, 45 gegeben. (Abbildung 6)

Abbildung 6 Graustufen Pixelfolge

Da sich die Kosinuskurve nicht zwischen 0 und 255 bewegt, sondern zwischen -1 und 1, werden die gegebenen Graustufen-Werte zuerst um 0 normiert. Dies geschieht, indem

von jedem Wert 128 subtrahiert wird. Der höchstmögliche Wert lautet danach 127, der tiefste -128.
Die Eingabewerte lauten daher $f(0) = 127, f(1) = 0, f(2) = -83$
Die DCT gemäss oben definierter Formel:

$$C(0) = a(0)\sum_{x=0}^{2} f(x) * \cos\left(\frac{\pi(2 * x + 1) * 0}{6}\right)$$

$$= \sqrt{\frac{1}{3}}\big(f(0) + f(1) + f(2)\big) = \sqrt{\frac{1}{3}}(127 + 0 - 83) = \sqrt{\mathbf{\frac{1}{3}}} * \mathbf{44}$$

$$C(1) = a(1)\sum_{x=0}^{2} f(x) * \cos\left(\frac{\pi(2 * x + 1) * 1}{6}\right)$$

Für die Summe:

$$f(0) * \cos\left(\frac{\pi(2 * 0 + 1) * 1}{6}\right) + f(1) * \cos\left(\frac{\pi(2 * 1 + 1) * 1}{6}\right) + f(2) * \cos\left(\frac{\pi(2 * 2 + 1) * 1}{6}\right)$$

$$= 127 * \cos\left(\frac{\pi}{6}\right) + 0 * \cos\left(\frac{3\pi}{6}\right) - 83 * \cos\left(\frac{5\pi}{6}\right)$$

Und daher

$$C(1) = \sqrt{\frac{2}{3}} * 127 * \frac{\sqrt{3}}{2} + 0 * 0 - 83 * -\frac{\sqrt{3}}{2}$$

$$= \sqrt{\frac{2}{3}} * \left(127 * \frac{\sqrt{3}}{2} + 83 * \frac{\sqrt{3}}{2}\right) = \sqrt{\frac{2}{3}} * 210 * \frac{\sqrt{3}}{2} = \mathbf{105\sqrt{2}}$$

Bleibt noch $C(2)$:

$$C(2) = a(2)\sum_{x=0}^{2} f(x) * \cos\left(\frac{\pi(2 * x + 1) * 2}{6}\right)$$

Für die Summe: (auf die Notation des Kosinusterms für $f(1)$ wird verzichtet, da $f(1) = 0$

$$f(0) * \cos\left(\frac{\pi(2 * 0 + 1) * 2}{6}\right) + f(1) * \cos(\dots) + f(2) * \cos\left(\frac{\pi(2 * 2 + 1) * 2}{6}\right)$$

$$= 127 * \cos\left(\frac{2\pi}{6}\right) + 0 - 83 * \cos\left(\frac{10\pi}{6}\right) = 127 * \cos\left(\frac{\pi}{3}\right) - 83 * \cos\left(\frac{5\pi}{3}\right)$$
$$= 127 * 0.5 - 83 * 0.5 = 22$$

Und deshalb:

$$C(2) = \sqrt{\mathbf{\frac{2}{3}}} * \mathbf{22}$$

Durch diese, bereits mit drei Pixeln doch etwas aufwändige Berechnung gewinnt man die Gewichtung der 2.1.1 erwähnten Kosinusfunktionen. Für die Pixel in Abbildung 6 gilt demzufolge $f(x) = \sqrt{\frac{1}{3}} * 44 * \cos\left(\frac{\pi(2x+1)*0}{6}\right) + 105\sqrt{2} * \cos\left(\frac{\pi(2x*+1)*1}{6}\right) + 22\sqrt{\frac{2}{3}} * cos\left(\frac{\pi(2x+1)*2}{6}\right)$.
Zur Kontrolle sei diese Berechnung durchgeführt:

$$f(0) = \sqrt{\frac{1}{3}} * 44 * \cos\left(\frac{\pi(2*0+1)*0}{6}\right) + 105\sqrt{2} * \cos\left(\frac{\pi(2*0+1)*1}{6}\right) + 22\sqrt{\frac{2}{3}} * \cos\left(\frac{\pi(2*0+1)*2}{6}\right)$$

$$= \sqrt{\frac{1}{3}} * 44 * \cos(0) + 105\sqrt{2} * \cos\left(\frac{\pi}{6}\right) + 22\sqrt{\frac{2}{3}} * \cos\left(\frac{2\pi}{6}\right)$$

$$= \frac{44}{\sqrt{3}} + 105\sqrt{2} * \frac{\sqrt{3}}{2} + 22 * \sqrt{\frac{2}{3}} * 0.5 = \mathbf{162.98}$$

$$f(1) = \frac{44}{\sqrt{3}} + 105\sqrt{2} * \cos\left(\frac{3\pi}{6}\right) + 22 * \sqrt{\frac{2}{3}} * \cos(\pi) = \mathbf{7.44}$$

$$f(2) = \frac{44}{\sqrt{3}} + 105\sqrt{2} * \cos\left(\frac{5\pi}{6}\right) + 22 * \sqrt{\frac{2}{3}} * \cos\left(\frac{10\pi}{6}\right) = \mathbf{-114.1}$$

Da es sich bei den Resultaten um normierte Werte handelt, wird zu jedem Resultat 128 addiert. Es resultieren $f(0) = 290.98, f(2) = 135.44, f(3) = 13.9$

Abbildung 7 Zurückgerechnetes Bild

Abbildung 7 zeigt das Ergebnis, zum Vergleich nochmals Abbildung 6 darunter. Obschon die kalkulierten Werte leicht abweichen, ist kein Unterschied wahrnehmbar.
Diesen Effekt macht sich die JPEG-Komprimierung zu Nutze, die im Folgenden beschrieben wird.

3 Die JPEG-Komprimierung

Im vorangegangenen Kapitel wurde die diskrete Kosinustransformation anhand eines 3x1 Pixel Bildes beschrieben.

Da in der Realität Bilder meist aus mehr als 3x1 Pixel bestehen, hat die Joint Photographic Experts Group zur Kompression von Bildern in [1] folgendes definiert:

- Ein Bild wird in 8x8 Pixel Blöcke zerlegt.
- Jeder dieser Blöcke wird mit der DCT in 64 DCT-Koeffizienten transformiert.
- Jeder dieser 64 Werte wird mit einer Quantisierungs-Tabelle quantisiert
- Die quantisierten Werte werden in einem Zig-Zig-Muster notiert

In einem 8x8 Block werden im Gegensatz zum 3x1 Block horizontale, wie auch vertikale Kosinuswellen verwendet. Es wird also jede Zeile und jede Spalte analog der Idee aus 2.1.2 berechnet.

3.1 Formel

Da jede Zeile und jede Spalte berechnet wird, berechnen sich die Werte jetzt in Abhängigkeit von x und y und nicht mehr nur eines Wertes.

$$S_{vu} = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \sum_{y=0}^{7} s_{xy} \cos\left(\frac{\pi(2x+1)u}{16}\right) \cos\left(\frac{\pi(2y+1)v}{16}\right)$$

Wobei gilt

$$C_u, C_v = \begin{cases} \frac{1}{\sqrt{2}}, & u, v = 0 \\ 1, & sonst \end{cases}$$

In 2.1.1 wurden die Basisvektoren anhand der Kosinuskurven für $f(x) = \cos(0)$, $f(x) = \cos(x)$, $f(x) = \cos(2x)$ definiert.
Für die DCT eines 8x8 Pixel Blocks ergeben sich 64 Basis-Kurven.
Abbildung 8 zeigt diese Kurven.
Oben links, die Kurve für $S_{0,0}$ nach rechts mit zunehmend starker Schwingung in horizontaler Richtung, nach unten mit zunehmend starker Schwingung in vertikaler Richtung. In der restlichen Matrix die Überlagerungen der jeweiligen Schwingungen.

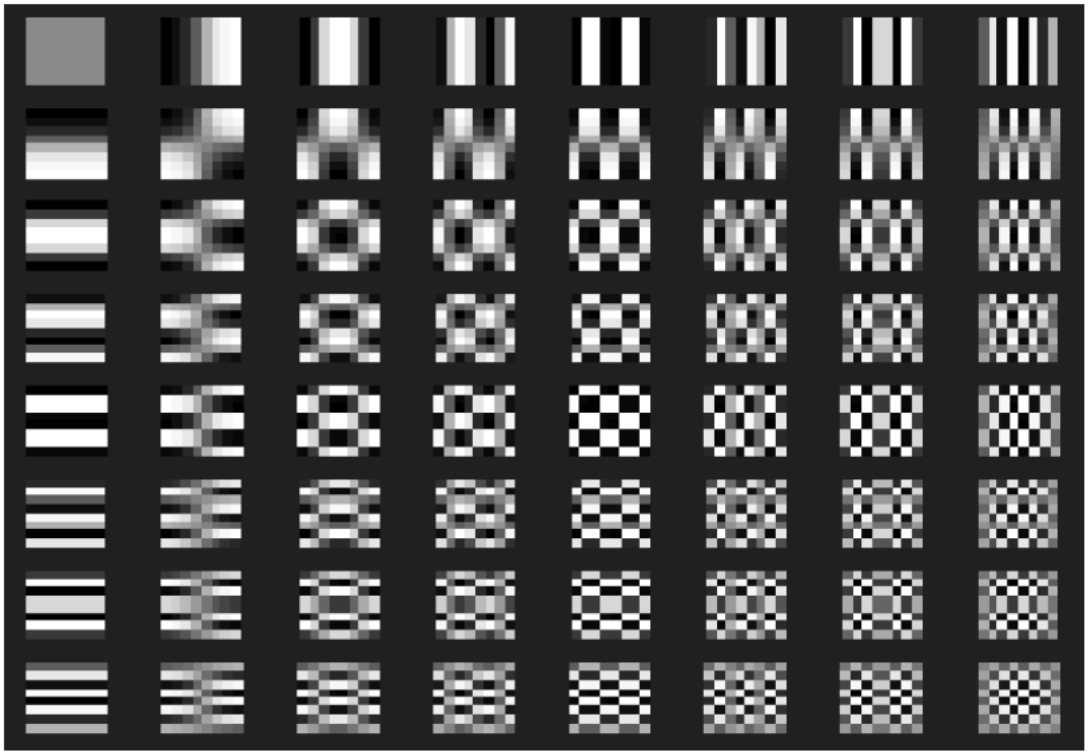

Abbildung 8 Basis Kosinuskurven 8x8 DCT

3.2 Beispiel

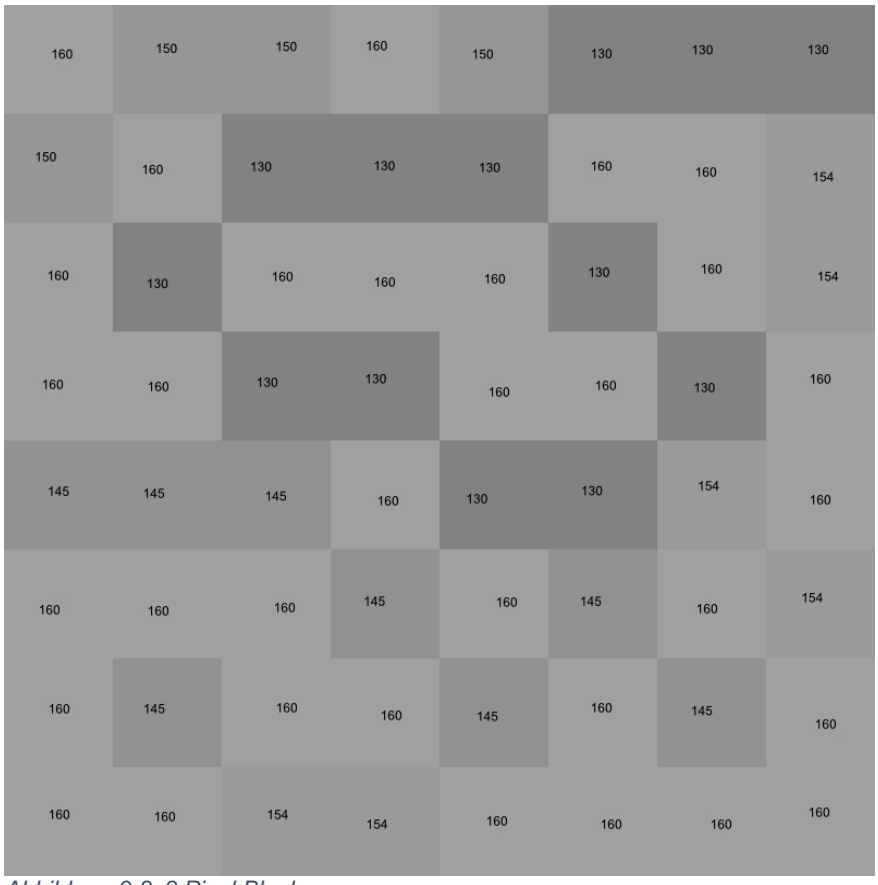

Abbildung 9 8x8 Pixel Block

Abbildung 9 zeigt schematisch einen 8x8 Pixel Block. Die Grauwerte variieren zwischen 145 und 160. Die Nummern sollen nur der Anschauung dienen und gehören nicht zum Bild.

Wird dieser 8x8 Block mit der in 3.1 definierten Funktion berechnet resultiert

1206	7.11	18.2	0.18	13.5	1.4	-1.05	1.07
-29.9	9.31	1.39	5.51	4.17	1.41	-7.8	0.009
7.95	11.36	-9.78	5.21	-7.14	5.39	2.00	-3.85
-10.63	16.27	-3.74	-11.67	-3.30	5.52	-4.62	-3.32
-10	10.21	-4.04	0.96	9.25	-11.58	-4.54	-9.77
2.02	16.10	-15.83	5.97	17.89	-9.04	16.45	4.05
12.67	11.89	-16.49	-8.05	12.15	9.81	-8.71	28.59
-5.78	0.96	-13.4	-29.19	14.62	34.64	0.78	3.90

Tabelle 1 Berechnete DCT-Werte

Diese Werte definieren, wie stark die jeweilige Kurven aus Abbildung 8 im Bild vertreten sind.

Der nächste Schritt der JPEG-Komprimierung besteht darin, die berechneten Werte zu quantisieren.

Abbildung 10 zeigt die Basis-Quantisierungstabelle aus [3]. Je nach JPEG-Qualität, mit welcher ein Bild komprimiert werden soll, gilt es eine andere Quantisierungstabelle anzuwenden.

Luminance Quantization Table							
16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Abbildung 10 Basis Quantisierungstabelle

Die Quantisierung erfolgt, indem ein Wert aus Tabelle 1 mit durch den entsprechenden Wert aus Abbildung 10 dividiert wird und anschliessend auf die nächste ganze Zahl gerundet wird.

$$B_{0,0} = round\left(\frac{A_{0,0}}{Q_{0,0}}\right) = round\left(\frac{1206}{16}\right) = 75$$

Für das gesamte Bild resultiert

75	1	2	0	1	0	0	0
-2	1	0	0	0	0	0	0
1	1	-1	0	0	0	0	0
-1	1	0	0	0	0	0	0
-1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Tabelle 2 Resultate nach Quantisierung

Durch diese Berechnungen, die wohlgemerkt für jeden 8x8 Pixel Block eines Bildes gemacht werden, wird auf den ersten Blick nichts gewonnen. Aus einer 8x8 Matrix mit Graustufenwerten wird eine 8x8 Matrix mit anderen Werten gewonnen.
Die eigentliche Kompression geschieht nun in der Notation dieser berechneten 64 Werte. Wie Tabelle 2 zu entnehmen ist, resultieren die meisten Werte zu einer 0.

Abbildung 11 zeigt die Zig-Zag-Sequenz, nach welcher die errechneten Werte anschliessend notiert werden.

Das Resultat lautet $\{75, -2,1,2,1,1, -1, -1,1, -1,0,1,52 * 0\}$.
Dieses Resultat ist beachtlich kürzer als eine 8x8 Matrix. Von 64 Werten werden 51 eingespart (52*0 ist ja auch eine Information).
Würde ein HD-Bild mit 1280x720 Pixeln aus 160x90 solcher 8x8 Blöcke bestehen, wäre die Einsparung 14400*51 Werte, also 734400 Werte, die nicht gespeichert oder übertragen werden müssen. Da $1280 * 720 = 921600$ entspricht dies einer Einsparung von 79.6%. Ein Bild mit der Grösse von 45MB wäre nach der Komprimierung also nur noch rund 9.1MB gross.

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

Abbildung 11 Zig-Zag Sequenz (Quelle: [1])

3.2.1 Datenverlust

Sollte der Einwand aufkommen, dass durch dieses Verfahren Daten verloren gehen, so ist dieser korrekt.
Die JPEG-Komprimierung ist eine verlustbehaftete Komprimierung. Allerdings werden hochfrequente Daten aus dem Bild entfernt, welche vom menschlichen Auge schlecht, bis gar nicht wahrgenommen werden können.
Sollte zu stark quantisiert werden, würde irgendwann nur noch der Wert in $B_{0,0}$ bestehen bleiben, der im Grunde genommen die Basis-Helligkeit des Blocks definiert.
Man kann sich die Komprimierung ungefähr vorstellen, als würde man die Augen zukneifen. Je mehr man die Augen schliesst, desto weniger sind starke Übergänge erkennbar, bis schliesslich nur noch eine einfarbige Fläche erkennbar ist.

3.2.2 Rückwärtsberechnung

Wird das Bild aus 3.2 zurückportiert, werden zuerst die quantisierten Werte wieder zu gewichteten DCT-Werten. Die Quantisierung erfolgte mit $\frac{A_{x,y}}{Q_{x,y}} = B_{x,y}$. Im Umkehrschluss resultiert $A'_{x,y} = B_{x,y} * Q_{x,y}$

1200	11	20	0	24	0	0	0
-24	12	0	0	0	0	0	0
14	13	-16	0	0	0	0	0
-14	17	0	0	0	0	0	0
-18	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Tabelle 3 Zurückgerechnete DCT-Werte

Tabelle 3 zeigt die Resultate für $A_{x,y}$.
Diese werden anschliessend mit der Formel für die Inverse DCT berechnet, woraus die Werte für $f'(x,y)$ resultieren.
Die Formel für die IDCT lautet

$$f'(x,y) = \frac{1}{4}\sum_{u=0}^{7}\sum_{v=0}^{7} C_u C_v S_{vu} \cos\left(\frac{\pi(2x+1)u}{16}\right)\cos\left(\frac{\pi(2y+1)v}{16}\right)$$

Wobei analog der DCT-Formel gilt

$$C_u, C_v = \begin{cases} 1/\sqrt{2}, & u,v = 0 \\ 1, & sonst \end{cases}$$

157.87	150.25	147.22	149.19	144.73	134.53	131.25	135.46
159.76	151.95	149.08	152.24	150.35	143.68	143.88	150.24
157.28	148.81	145.43	149.34	150.04	147.43	151.80	160.81
152.91	143.41	138.66	141.77	142.88	141.81	148.13	158.47
155.30	145.28	139.54	141.37	141.09	138.75	144.08	153.89
162.05	152.83	148.07	150.17	148.92	144.51	147.50	155.77
162.58	155.27	153.32	157.68	157.09	151.62	152.74	159.60
157.60	151.92	152.46	159.10	159.70	154.16	154.46	160.60

Tabelle 4 Rückportierte Graustufen

Abbildung 12 zeigt das Rückportierte Bild.
Zum Vergleich ist daneben nochmals Abbildung 9 gezeigt, welche das Ursprungsbild darstellt.

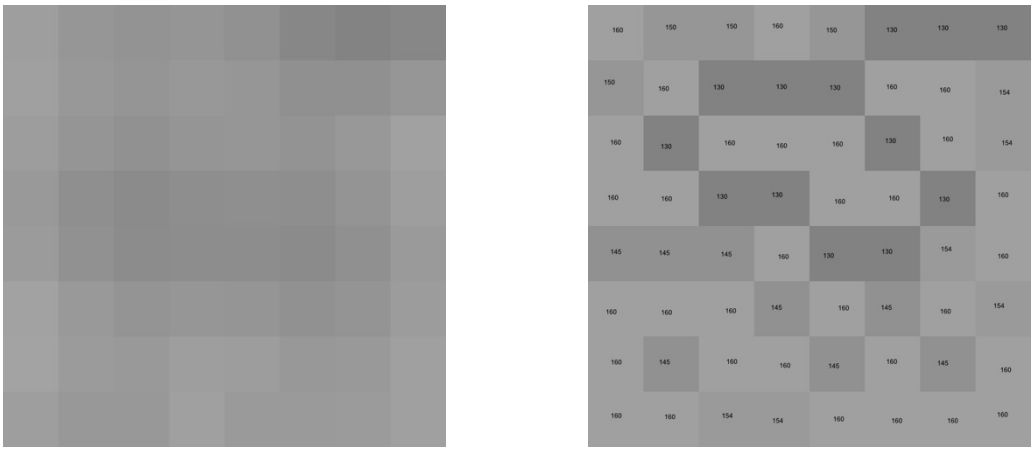

Abbildung 12 8x8 Block Rückportiert

Es ist festzuhalten, dass die genauen Übergänge zwischen den einzelnen Pixeln nicht mehr so klar definiert sind. Betrachtet man den Block allerdings in seiner Originalgrösse (8x8 Pixel) so ist dies mit blossem Auge zwar bei gutem Sehvermögen noch knapp wahrnehmbar aber in einem HD-Bild mit 1280x720 Pixeln wird die Differenz nicht auffallen. Abbildung 13 zeigt die beiden Blöcke in Originalgrösse.

Abbildung 13 8x8 Blöcke in Originalgrösse

Es sei an dieser Stelle festgehalten, dass die demonstrierten Berechnungen im JPEG-Verfahren selbstverständlich nicht auf Graustufen-Werte, sondern auf die drei Komponenten Helligkeit (Y), Blauanteil (U) und Rotanteil (V) angewendet werden. Liegt ein Bild im RGB-Format vor, muss es zuerst nach YUV umgerechnet werden. RGB und YUV stehen im folgenden Verhältnis zueinander:

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1 & 0 & \frac{8}{5} \\ 1 & -\frac{1}{3} & -\frac{4}{5} \\ 1 & 2 & 0 \end{pmatrix} \begin{pmatrix} Y \\ U \\ V \end{pmatrix}$$

4 Umsetzung in Python

Der im Anhang befindliche Code ist auch unter https://github.com/tobias-hu/ffhs_linalg auffindbar.

Die Vorwärts-DCT-Funktion wird im File «DCT.py» definiert.
Als Parameter nimmt die Methode N entgegen, was der Einheit für die NxN-Transformationsmatrix entspricht.

```
def fdct(N):
    M = np.zeros((N, N))
    for k in range(N):
        for n in range(N):
            M[k, n] = np.sqrt(1 / N) if k == 0 else np.sqrt(2 / N) *
sp.cos((sp.pi * (2 * n + 1) * k) / (2 * N))

    return M
```

In 2.1.2 wurde die Normalisierung um 0 erwähnt, welche in der vorliegenden Python-Umsetzung mit der Methode «normalize» umgesetzt wurde. Die Methode nimmt ein Array als Parameter entgegen, subtrahiert von jedem Wert 128 und retourniert die berechneten Werte als Array.

In «main.py» wird zuerst die Transformationsmatrix «M» mit *fdct(3)* erstellt

```
M = fdct(3)
```

Anschliessend das «Bild» erstellt und dessen Werte normalisiert

```
img = normalize(np.array([255,128,45]))
```

Die Matrix wird mit *print(M)* angezeigt, sie lautet

$$\begin{matrix} \sqrt{\frac{1}{3}} & \sqrt{\frac{1}{3}} & \sqrt{\frac{1}{3}} \\ \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{6}} & -\frac{2}{\sqrt{6}} & \frac{1}{\sqrt{6}} \end{matrix}$$

Das Resultat der DCT entspricht der Multiplikation von *img* mit der Matrix *M*

```
dct_res = M @ img
```

mit *print(dct_res)* wird es angezeigt:
[25.40341184 148.49242405 17.96292478]

Diese Werte entsprechen den kalkulierten Werten aus 2.1.2

$$\sqrt{\frac{1}{3}} * \mathbf{44}\ ,\ \mathbf{105}\sqrt{\mathbf{2}},\ \sqrt{\frac{2}{3}} * \mathbf{22}.$$

Für die Rückwärtsberechnung werden diese Werte mit der IDCT-Matrix multipliziert, anschliessend gerundet und «denormalisiert», was lediglich ein addieren von 128 je Wert ist.

```
res = denormalize(np.round(T @ dct_res))
```

Das Resultat lautet [255, 128, 45], was zu erwarten und zu hoffen war, denn das waren die Ursprungswerte des Bildes.

In der Aufgabenstellung zu dieser Arbeit wurde gefordert *dct* und *idct* aus *scipy.fftpack* ebenfalls zu verwenden.
Mit dieser Bibliothek kann die DCT und die IDCT auf Bilder angewendet werden, die Resultate werden anschliessend mit «matplotlib» geplottet.

Es wurde mit mehreren Bildern experimentiert. Die grösste Herausforderung, welche im Rahmen dieser Arbeit nicht gelöst werden konnte, war die Anzeige in Graustufen für gewisse Bilder.
Es werden Stock-Images aus der «skimage»-Bibliothek verwendet. Für die Bilder «astronaut» und «coffee», um nur zwei Beispiele zu nennen, war es mit der Anweisung «cmap='gray'» in *plt.imshow()* nicht möglich die Bilder in Graustufen anzuzeigen.

Um dennoch mit der Bibliothek zu arbeiten und Resultate der Funktionen zu erhalten wurden Bilder gesucht, die in Graustufen angezeigt werden konnten. Für das erste Bild wurde «camera» gewählt.

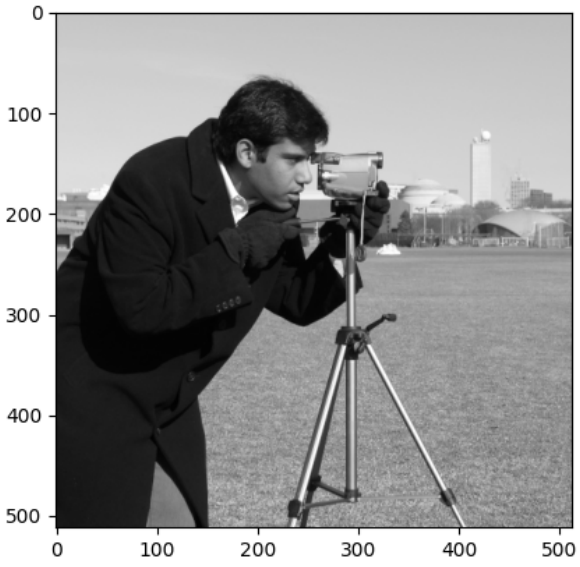

Abbildung 14 Camera

Die DCT wird zuerst über die Zeilen durchgeführt.

```
dctRows = dct(img, axis=1, norm='ortho')
```

anschliessend über die Spalten

```
dctCols = dct(dctRows, axis=0, norm='ortho')
```

Der kalkulierte Wert S_{00} entspricht 66079, weitere Werte sind im einstelligen Bereich. Um diese Differenz «sichtbar» darzustellen, werden die Werte logarithmisch dargestellt.

```
plt.imshow(np.log(np.abs(dctCols)), cmap='gray')
```

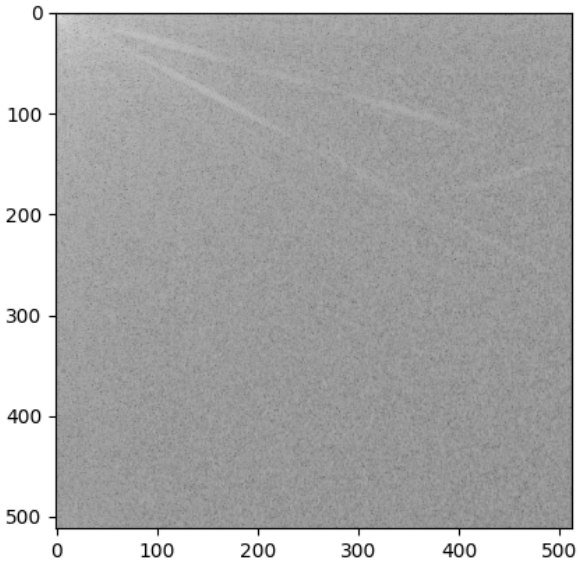

Abbildung 15 DCT von Camera

Die Rücktransformation geschieht analog der DCT mit der Funktion *idct.*
Erst die Spalten, dann die Zeilen:

```
idctCols = idct(dctCols, axis=0, norm='ortho')
idctImg = dct(idctCols, axis=1, norm='ortho')
```

Das Resultat:

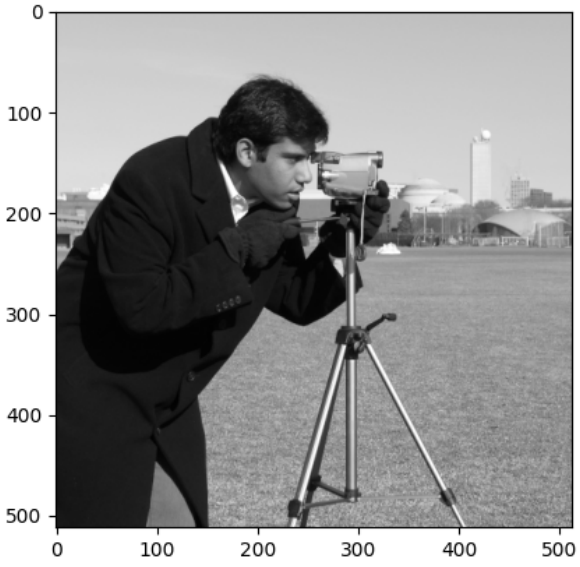

Abbildung 16 IDCT Camera

Ein zweites Beispiel wurde mit einem eher linienartigen Bild «brick» der «skimage»-Bibliothek unternommen.
Die folgenden Abbildungen zeigen die Resultate.

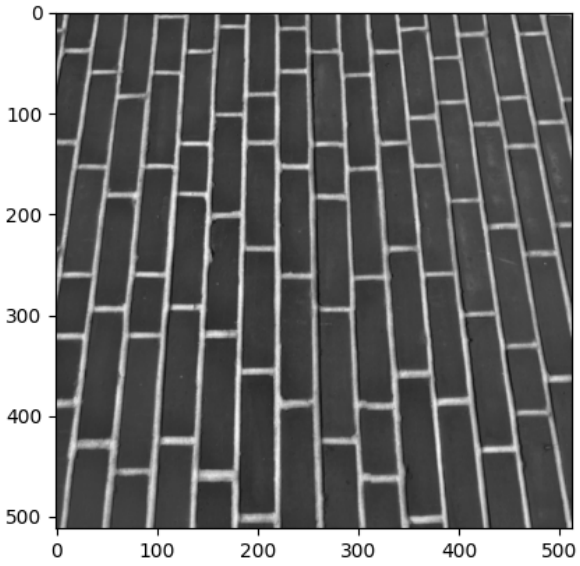

Abbildung 17 Brick

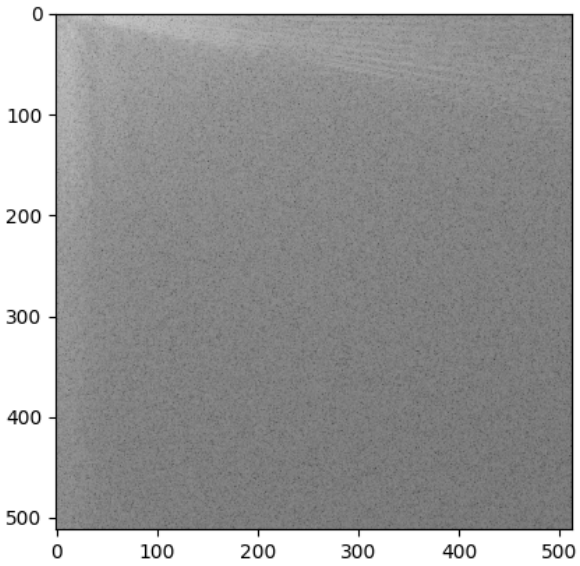

Abbildung 18 DCT Brick

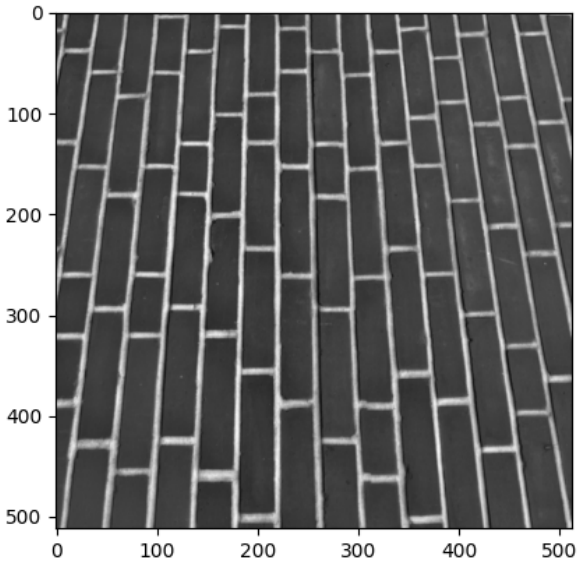

Abbildung 19 IDCT Brick

5 Diskussion

5.1 Fehlerkorrektur

Bevor weiter auf die DCT eingegangen wird, soll an dieser Stelle kurz ein Fehler besprochen werden.
Auf Seite 19 wird die Rücktransformation mit «scipy.fftpack» angewendet. Die anschliessend gezeigten Bilder sind nicht die Resultate dieser zwei Zeilen Code

```
idctCols = idct(dctCols, axis=0, norm='ortho')
idctImg = dct(idctCols, axis=1, norm='ortho')
```

Hier wird nämlich für «idctImg» die Funktion *dct* verwendet, was natürlich falsch ist. Die Resultate mit diesem falschen Code sehen wie folgt aus:

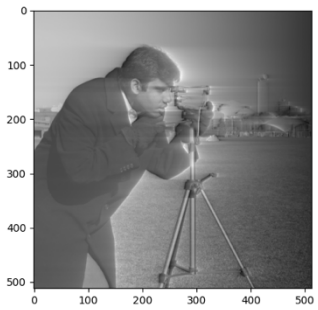

Abbildung 20 Falsche IDCT für Camera

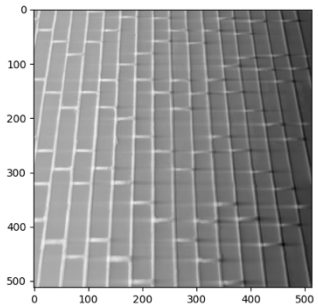

Abbildung 21 Falsche IDCT für Brick

Beim Bild «Camera» war ich mir nicht sicher, ob meine Anwendung so falsch ist, ob ein Problem mit den Graustufen vorliegt, oder ob die verwendete Bibliothek nicht ausreichend ist. Beim zweiten Bild «brick» fällt auf, dass das Bild von Hell zu dunkel läuft, was dem Verlauf der DCT-Basiswellen entspricht. Dadurch entstand die Idee, die Kalkulation nochmals genau zu prüfen und deshalb konnte der Fehler gefunden und korrigiert werden. In der aktuellen Version des Codes ist dies behoben.

5.2 Diskussion der DCT

Die DCT ist ein wichtiges Vorgehen bei der Komprimierung von Daten.
Die Transformation selbst ist rechenintensiv, gerade im Fall von Bildkomprimierungen wie JPEG ist der Rechenaufwand enorm. Dennoch lohnt es sich, diesen Aufwand zu betreiben, da dadurch Speicherplatz gespart werden kann, oder Bilder als Anhänge in E-Mails

oder anderen Online-Kommunikationsmitteln verwendet werden können, ohne unsagbar lange Übertragungszeiten zu verursachen.
Im Rahmen dieser Arbeit war es nicht leicht, einschlägige Literatur zur Thematik zu finden. In vielen gefundenen und anschliessend nicht verwendeten Unterlagen und Videos wird die DCT lediglich als «Mittel zum Zweck» erwähnt, aber nicht mathematisch grundlegend erläutert.
Für eine mathematische Arbeit sind entsprechende Quellen sinnfrei.
Nach Meinung des Autors dieser Arbeit bestünde der Bedarf, das Thema der Diskreten Kosinustransformation leichter verständlich und öffentlich zugänglich zu dokumentieren. Es scheint wahrscheinlich, dass durchaus Interesse daran bestehen könnte; und wenn es nur seitens Studenten ist, welche sich mit der Thematik auseinandersetzen dürfen/müssen.

6 Anhang

main.py

```
from DCT import fdct, idct, normalize, denormalize
import numpy as np


M = fdct(3)
T = idct(3)

img = normalize(np.array([255,128,45]))

print(M)

dct_res = M @ img

print(dct_res)

res = denormalize(np.round(T @ dct_res))

print(res)
```

DCT.py

```
import numpy as np
import sympy as sp


def fdct(N):
    M = np.zeros((N, N))
    for k in range(N):
        for n in range(N):
            M[k, n] = np.sqrt(1 / N) if k == 0 else np.sqrt(2 / N) *
sp.cos((sp.pi * (2 * n + 1) * k) / (2 * N))

    return M


def idct(N):
    T = np.zeros((N, N))
    for n in range(N):
        for k in range(N):
            T[n, k] = np.sqrt(1 / N) if k == 0 else np.sqrt(2 / N) *
sp.cos((sp.pi * (2 * n + 1) * k) / (2 * N))

    return T


def normalize(data):
    return data - 128


def denormalize(data):
    return data + 128
```

FFTPack.py

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from scipy.fftpack import dct, idct
from skimage import data
# 'cause of some weird issues with the matplotlib backend
matplotlib.use('TkAgg')

img = data.brick()

plt.imshow(img, cmap='gray')
plt.show()

dctRows = dct(img, axis=1, norm='ortho')

dctCols = dct(dctRows, axis=0, norm='ortho')
print(dctCols)

plt.imshow(np.log(np.abs(dctCols)), cmap='gray')

plt.show()

idctCols = idct(dctCols, axis=0, norm='ortho')
idctImg = idct(idctCols, axis=1, norm='ortho')

plt.imshow(idctImg, cmap='gray')
plt.show()
```

Literatur- und Quellenverzeichnis

[1] ITU, «Information Technology - Digital Compression And Coding of Continuous-Tone Still Images - Requirements and Guidelines,» CCIT, The International Telegraph and Telephone Consultative Committee, 1993.

[2] S. A. Khayam, «The Discrete Cosine Transform (DTC), Theory and Application,» 2003.

[3] S. Dhanani and M. Parker, Digital Video Processing for Engineers, Elsevier, 2013.

[4] E. Weitz, «Weitz.de,» [Online]. Verfügbar: https://weitz.de/dct/. [Zugriff am 16 12 2024].

[5] E. Weitz, «youtube.com,» 17 Mai 2017. [Online]. Verfügbar: https://www.youtube.com/watch?v=7fhHQgu2OcY. [Zugriff am 9 Dezember 2024].

[6] T. Hanselmann, "Hochschule Mittweida," 2016. [Online]. Verfügbar: https://www.cb.hs-mittweida.de/fileadmin/verzeichnisfreigaben/haenselm/dokumente/dbv_2016ss_transform.pdf. [Accessed 27 Dezember 2024].

[7] M. Hellmund, «Uni-Leipzig,» [Online]. Verfügbar: https://www.math.uni-leipzig.de/~hellmund/Vorlesung/Jpeg.pdf. [Zugriff am 29 Dezember 2024].

[8] U. Schulz, «Youtube,» 22 Juni 2020. [Online]. Available: https://www.youtube.com/watch?v=8WakHe-rY_w. [Zugriff am 16 Dezember 2025].

Bildverzeichnis

Tabellenverzeichnis