

11B0033 AuV: Meilenstein 05

Video- und Audiokompression, Multimedia-Container

Julius Schöning
j.schoening@hs-osnabrueck.de

Abgabedatum: 07. Januar 2021 23:59:59

In diesem Meilenstein werden Sie sich mit der Digitalisierung von Videosignalen, Multimedia-Containern und Audiokompression beschäftigen. Alle Aufgaben dieses Aufgabenblattes können Sie wahlweise mit Matlab oder der Scriptssprache Python bearbeiten.

Matlab steht Ihnen als Total Headcount Lizenz der Hochschule Osnabrück zur Verfügung. Installationsinformationen finden Sie unter <https://wiki.hs-osnabrueck.de/pages/viewpage.action?pageId=8749400>.

Gruppen

Der Meilenstein *Video- und Audiokompression, Multimedia-Container* wird in Gruppenarbeit mit den bereits bekannten Mitgliedern umgesetzt. Dieser Meilenstein ist so ausgelegt, dass Sie diesen an Ihrem eigenen Computer bearbeiten können.

Eine Veränderung der Gruppenzusammensetzung ist nicht mehr möglich.

Abgabe

Die Ergebnisse aller Teilaufgaben sind schriftlich in einem strukturierten Praktikumsbericht im PDF-Format zusammenzufassen. Ergänzen Sie Ihre Ausführungen durch aussagekräftige Screenshots, Abbildungen, Plots und Tabellen. Mögliche Word und L^AT_EX Vorlagen für Praktikumsberichte finden Sie im Downloadbericht dieses Praktikum.

Den strukturierten Praktikumsbericht und alle Quellcodedateien, die Sie in diesem Meilenstein erzeugen oder auf die Sie sich in Ihrem Praktikumsbericht beziehen müssen in Ihrem Abgabearchiv vorhanden sein.

Für die Abgabe komprimieren Sie alle erstellen Matlab bzw. Python Quellcodedateien sowie Ihren strukturierten Praktikumsberichts als PDF in ein zip-Archive mit dem Namen *05_UserName1_UserName2_UserName3.zip*. Laden Sie das Zip-File bis spätestens zum 07. Januar 2021 23:59:59 im Abgabebereich AuV Praktikum im OSCA als hoch.

Testat

Bereiten Sie sich für ein Testat mit ggf. schriftlichen Kurztest von ca. 10 Minuten vor. Inhalt dieses Testats werden die Themen der Meilensteine 05 und 06 sein. Ihre Gruppe erhält einen persönlichen Termin für die KW02/2021 zugeteilt.

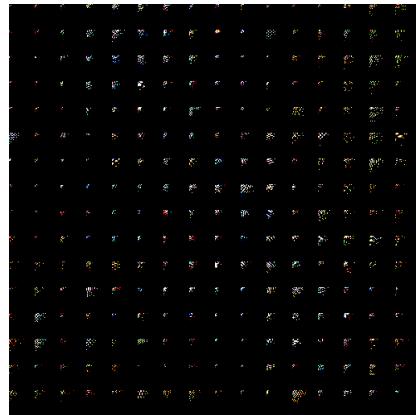
1 Aufgabe: Diskrete Kosinustransformation (DCT)

Die diskrete Kosinustransformation wird beispielsweise beim Dateiformat JPEG verwendet, um Bilddaten zu komprimieren. Dazu wird das Bild in $32 \times 32px$ Blöcke aufgeteilt und die Pixelwerte jedes Blocks in Frequenzkoeffizienten transformiert, die eine bessere Komprimierung ermöglichen. In den transformierten Blöcken stehen die wichtigsten Werte in der linken oberen Ecke und können so genauer quantisiert werden.

Programmieren sie ein Script, das diesen Prozess durchführt. Testen Sie ihr Script mit dem Bild `space.png`.



Bild 'space.png'



DCT Kodiert



Rekonstruiertes Bild

(`space.png` by Chuck Ayoub, Creative Commons Attribution-Share Alike 4.0 International Lizenz)

Speichern Sie Ihr Script unter `auv_05_Aufgabe_1.m` wenn Sie Matlab oder unter `auv_05_Aufgabe_1.py` wenn Sie Python verwenden.

1. lesen Sie das Bild `space.png` mit `double(imread('space.png'))` ein und machen Sie einen Intervallshift auf $[-128, 127]$.

2. teilen Sie es je Farbkanal in $32 \times 32px$ Blöcke ein.

3. erstellen Sie nun die Quantisierungsmatrix. Gehen Sie dazu wie in der Vorlesung besprochen vor. Verwenden Sie eine Qualität von 6.

4. iterieren Sie nun über alle $32 \times 32px$ Blöcke: Transformieren Sie jeden Block mit `dct2(<block>)`, quantisieren Sie ihn mit der zuvor erstellten Quantisierungsmatrix. Fügen Sie in die kodierte Ergebnismatrix ein. Decodieren Sie außerdem den Block mit `idct2(<block>)` und fügen sie diesen in die dekodierte Ergebnismatrix ein.

Hinweis: Mit `.*` und `./` können Sie elementweise multiplizieren und dividieren.

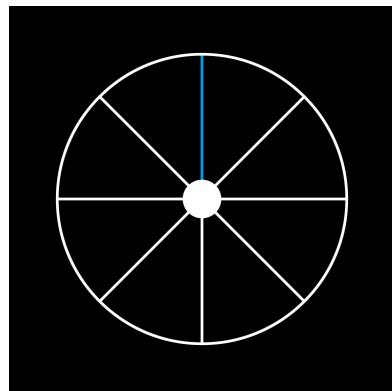
5. Bringen Sie die Ergebnismatrizen wieder in ihre Ursprungsgröße und visualisieren Sie diese Bilder in einem Plot mit Subplots.

6. Handelt es sich um Entropie-, Quellen-, oder Hybridkodierung? Begründen Sie!

7. Vergleichen Sie die Dateigrößen und Bildqualität der Bilder. Wie unterscheidet sich diese Methode zur Quantisierung der Farbwerte?

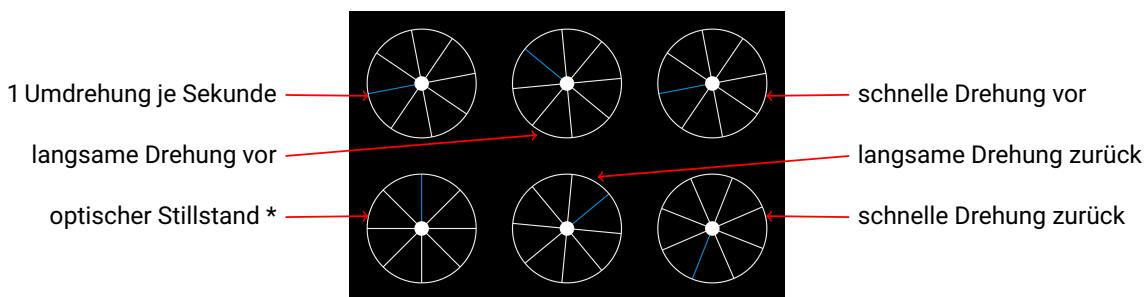
2 Aufgabe: Digitalisierung von Videosignalen

In diesem Aufgabenteil sollen Sie die Auswirkung des Nyquist-Shannon-Abtasttheorems auf die Wiedergabe von Videodateien mit unterschiedlicher Geschwindigkeit untersuchen. Sie sollen die Wiedergabe eines Speichenrades bei unterschiedlich großen Geschwindigkeiten verstehen und erläutern.



rad.png

- a) Erstellen Sie ein Matlab- oder Python-Script, das ein drei Sekunden Video erstellt, in dem sich das Bild *rad.png* in verschiedenen Geschwindigkeiten dreht.



Beispiel Frame des Videos *auv_05_Aufgabe_2.mp4'* (3 Sekunden bei 25fps)

Nutzen sie dafür in Matlab den *VideoWriter* um das Video *auv_05_Aufgabe_2.mp4'* mit H.264 Encodierung zu erstellen. Mit *imrotate(<Bild>, <Drehung>, 'bilinear', 'crop')* können Bilder gedreht werden. Diese können dann beispielsweise mit *imtile* angeordnet werden. Mit *writeVideo* kann das zusammengesetzte Bild als Frame in das Video eingefügt werden. Die Beschriftung ist Teil einer späteren Aufgabe und ist in diesem Aufgabenteil nicht zu implementieren.

Die Räder sollen sich wie von den Beschriftungen angegeben unterschiedlich schnell drehen. Die Originalgeschwindigkeit soll dabei eine Umdrehung pro Sekunde sein.

Die besondere Herausforderung: Auch wenn sich die Räder scheinbar in unterschiedliche Richtungen drehen, dürfen Sie die Räder mit *imrotate* nur in die gleiche Richtung drehen. *Außerdem darf die Drehung nicht 0 sein.

Das Video soll 3 Sekunden lang sein bei 25 Frames pro Sekunde.

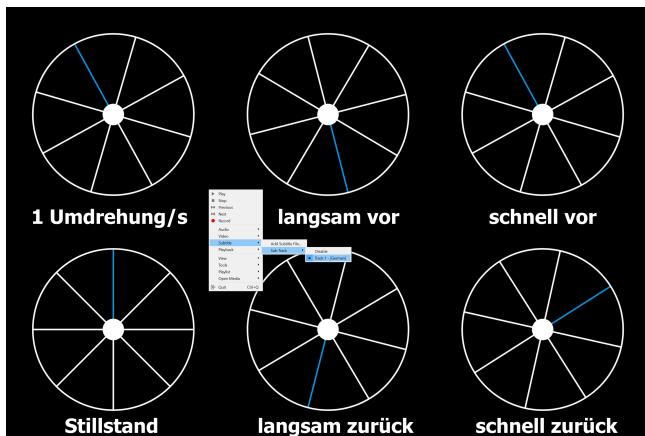
Speichern Sie Ihr Script unter *auv_05_Aufgabe_2.m* wenn Sie Matlab oder unter *auv_05_Aufgabe_2.py* wenn Sie Python verwenden.

- b) Beschreiben Sie, wie Sie mit Drehungen in die gleiche Richtung scheinbar unterschiedliche Drehrichtungen erreicht haben. Stellen Sie dabei auch einen Bezug zum Nyquist-Shannon-Abtasttheorem her.

3 Aufgabe: Multimedia-Container

Nun sollen die Beschriftungen der Räder hinzugefügt werden. Dies wird über Untertitel in einem MKV-Container realisiert.

- Informieren Sie sich über das ASS (Advanced SubStation Alpha) Untertitel-Format (<https://www.matroska.org/technical/subtitles.html#ssaass-subtitles>). Erstellen Sie eine deutschsprachige Untertitel-Datei, die die Räder passend beschrifft.
- Nutzen Sie die Open-Source-Software MKVToolNix (<https://mkvtoolnix.download/>), um die in Aufgabe 2 erstellte Videodatei und die Untertitel in einen MKV-Container zusammenzuführen. Für den Fall, dass Sie in Aufgabe 2 kein Video erstellen konnten, verwenden Sie das Video `aув_05_Aufgabe_2_failsave.mp4`.



Video mit deutschen Untertitel

Speichern Sie das Video als `aув_05_Aufgabe_3.mkv`.

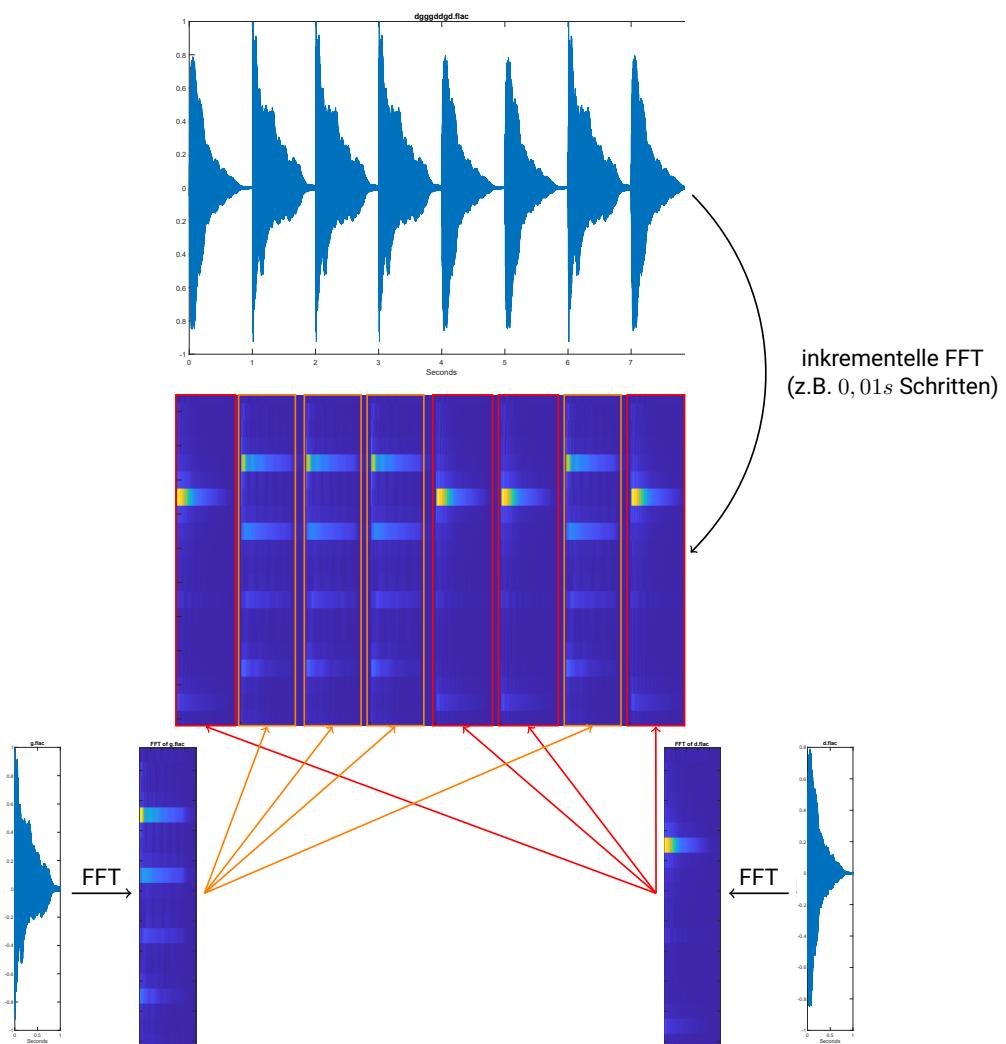
Hinweise: Vergessen Sie nicht diese Aufgabe in Ihrem Praktikumsbericht zu beschreiben. Nicht jeder Multimedia-Player unterstützt das ASS Untertitel-Format; der VLC Multimedia-Player unterstützt dieses Format.

4 Aufgabe: Audiokompression

Einer der besten Wege Audiodaten zu komprimieren, ist den Inhalt der Daten zu erkennen und diesen mittels Symbolen zu speichern. D.h. eine Audiodatei mit gesprochenem Wörter können erkannt und als Textdatei abgespeichert werden.

In dieser Aufgabe sollen sie ein Skript entwickeln, das in einer Audiodatei gegebene Audio-Fragmente erkennen kann. Zur Vereinfachung handelt es sich dabei nicht um gesprochenen Text, sondern Töne auf einem Klavier.

- Lesen Sie `dgggddgd.flac`, `d.flac` und `g.flac` mit `audioread` ein. Ihr Skript soll in `dgggddgd.flac` die im Dateinamen angegebene Reihenfolge der Audio-Fragmente "Ton d" und "Ton g" erkennen. Transformieren Sie dazu in z.B. 0,01 Sekunden Schritte die Audiosignale `d.flac` und `g.flac` mit `fft()` in den **Frequenzraum**. Überprüfen Sie dann, ob die Frequenzen von `d.flac` oder `g.flac` mit den Frequenzen eines gleich langen Ausschnitts aus `dgggddgd.flac` übereinstimmen. Geben Sie das Ergebnis in der Form `d g g g d d g d` aus. (Mit `imagesc()`, können Sie sich die FFT-Matrizen visualisieren, vgl. Abbildung unten.)



Visualisierung der Zuordnung der Audio-Fragmente "Ton d" und "Ton g" zur Audiodatei *dggddgd.flac*

- b) Versuchen Sie jetzt, das Script aus a) so zu erweitern, das die Audio-Fragmente "ja" (*ja_f.flac*) und "nein" (*nein_f.flac*) in der Audiodatei *synthetic_Ja_Nein_Nein_Nein_Ja_Ja_Nein_Nein_Ja.flac* richtig erkannt werden. Geben Sie das Ergebnis in der Form *ja nein nein nein ja ja nein nein ja* aus.
- c) Versuchen Sie jetzt, das Script aus a) und b) so zu erweitern, das Audio-Fragmente eines anderen Sprechers (*ja_m.flac* bzw. *nein_m.flac*) benutzt werden können, um in der Audiodatei *synthetic_Ja_Nein_Nein_Nein_Ja_Ja_Nein_Nein_Ja.flac* die Wörter "ja" und "nein" richtig zu erkennen.
- d) Versuchern Sie jetzt in der Audiodatei *voice_Ja_Nein_Nein_jein_Ja_Ja_Nein_Nein_Ja_Nein.flac* die Wörter "ja" und "nein" richtig zu erkennen indem Sie die Audio-Fragmente *ja_f.flac*, *nein_f.flac*, *ja_m.flac* und *nein_m.flac* verwenden. Dieser Aufgabenteil kann ggf. nicht mit dem richtigen Ergebnis gelöst werden.
- e) Reflektieren Sie die Aufgabenteile a), b) und c). Welche Probleme treten beim Anwenden des oben beschriebenen Vorgehens auf real gesprochene Worte auftreten? Wie könnte dieses Verfahren verbessert werden?

Speichern Sie Ihr Script unter *auv_05_Aufgabe_4.m* wenn Sie Matlab oder unter *auv_05_Aufgabe_4.py* wenn Sie Python verwenden.