

Maschinelles Sehen

Bildmerkmale und Bildsuche (13 Punkte)

SoSe 2025

Ziel dieser Übungsaufgabe ist es, sich ein tieferes Verständnis für markante Bildmerkmale (Image features/descriptors), ihre Implementierung und Anwendungsbereiche zu erarbeiten und dabei NumPy und OpenCV Kenntnisse zu erweitern. Laden Sie den vorgegebenen Skelett-Sourcecode aus Moodle und implementieren Sie die notwendigen Funktionen. Erweitern und verändern Sie den Sourcecode nach Belieben. Der vorgegebene Sourcecode enthält viele weitere Anmerkungen.

Aufgabe 1: SIFT in OpenCV (1 Punkt)

Bitte setzen Sie sich mit dem SIFT descriptor in OpenCV auseinander (https://docs.opencv.org/4.5.4/da/df5/tutorial_py_sift_intro.html). Bauen Sie in Ihr bisheriges Videostream-Beispiel die SIFT feature Extraktion ein und visualisieren Sie die Keypoints wie im Bild 1 dargestellt. Hinweis: Die Kamerakalibrierung brauchen wir im Moment erstmal nicht mehr. Nutzen Sie das mitgelieferte features.py zur Implementierung, **machen Sie einen Screenshot und geben diesen bitte mit ab**.

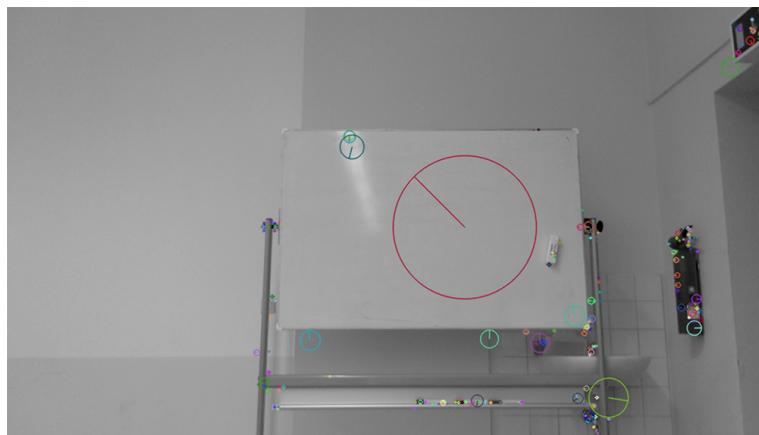


Abbildung 1: Bild des Videostreams mit der Visualisierung der SIFT Features

Aufgabe 2: Simple Content-based Image Retrieval (4 Punkte)

Das Ziel einer inhaltsbasierten Bildsuche ist, ähnliche Bilder bezogen auf ein Eingabebild zu finden und dabei ausschliesslich Bildmerkmale zu nutzen (also keine Keywords oder Ähnliches). Diese Suche basiert ebenfalls häufig auf der Verwendung von lokalen Featurevektoren, wie z.B. die Bildmerkmale von SIFT. Bei der Suche müssen diese Vektoren verglichen werden, um Ähnlichkeiten festzustellen. Dafür können unterschiedliche Distanzmetriken benutzt werden. In diesem sehr reduzierten Beispiel sollen Sie eine Suche implementieren, die die Vektoren auf Basis der L2-Norm vergleicht (also der euklidische Distanz der Vektoren). Gehen Sie dabei wie folgt vor und nutzen image_retrieval.py:

- Laden Sie alle Bilder der mitgelieferten 'Datenbank'. Es gibt insgesamt 3 unterschiedliche Bildklassen mit etwa 5-6 Bildern.
- Erzeugen Sie eine Menge von Keypoints die in einem uniformen Gitter über dem Bild verteilt sind und an denen Sie die SIFT-Featurevektoren berechnen. **Hinweis:** Die Suche nach einem



Abbildung 2: Ergebnisse der Bildsuche von links oben nach rechts unten.

gitterförmigen Sampling ist eine Hyperparametersuche. D.h. Sie müssen eine sinnvolle Menge an Keypoints finden, um die Suche präzise zu machen. Wir nutzen hier explizit nicht die Keypoints von SIFT, sondern einen gitter-basierten Ansatz. Das funktioniert für unsere kleine Datenbank besser.

- c) Benutzen Sie die Keypoints, um ausschliesslich an diesen Stellen die SIFT-Featurevektoren zu berechnen. Das erzeugt genau einen Descriptor für jedes Bild in der Datenbank. Diese sollten in einem Array zwischengespeichert werden. Hinweis: Wir benutzen hier nicht den SIFT-Keypointdetector.
- d) Laden Sie ein Anfragebild und extrahieren Sie für die gleichen Keypoints einen Bildbeschreiber. Vergleichen Sie den Beschreiber gegen die Descriptoren aus der Datenbank. Nutzen Sie dafür die L2-Norm.
- e) Speichern Sie das Ergebnis in eine *PriorityQueue*.
- f) Geben Sie die Ergebnisse der Anfrage in der Reihenfolge der kürzesten Distanz aus. Das Ergebnis sollte für die Gesichter so aussehen wie in Abbildung 2 dargestellt, **kann aber auch leicht abweichen. Eine 100%ige Übereinstimmung ist nicht notwendig, aber die erste 3-4 Suchergebnisse sollten mit der Anfragekategorie übereinstimmen.**

Aufgabe 3: Write your own image descriptor (4 Punkte)

Implementieren Sie ihren eigenen lokalen Featurevektor und testen ihn mit den mitgelieferten Eingabebildern (simple_hog.py). Der Deskriptor soll ein Histogramm von Gradientenrichtungen, wie Sie von einem einfachen Kantendetektions-Filter erzeugt werden (z.B. Sobel-Filter), abspeichern. Im speziellen Fall soll der Deskriptor nur für einen einzigen Keypoint (in der Mitte des Bildes) und einer vorgegebenen Fentergröße (z.B. 11) extrahiert werden. Hinweis: Schauen Sie sich die Funktionen *cv2.Sobel*, *cv2.phase* und *cv2.magnitude* an. Die Ergebnishistogramme sollen **in etwa** so aussehen wie in Abbildung 3 dargestellt. Das Histogramm enthält 8 Buckets. Denken Sie über einen sinnvollen Wertebereich nach.

Aufgabe 4: Harris Corner Detector (4 Punkt)

Wie in der Vorlesung besprochen, benötigt man zur Berechnung des Harris-Corner Detektors die Matrix

$$M = \sum_{x,y}^{3,3} w(x,y) \begin{bmatrix} I_{xx} & I_x I_y \\ I_y I_x & I_{yy} \end{bmatrix}$$

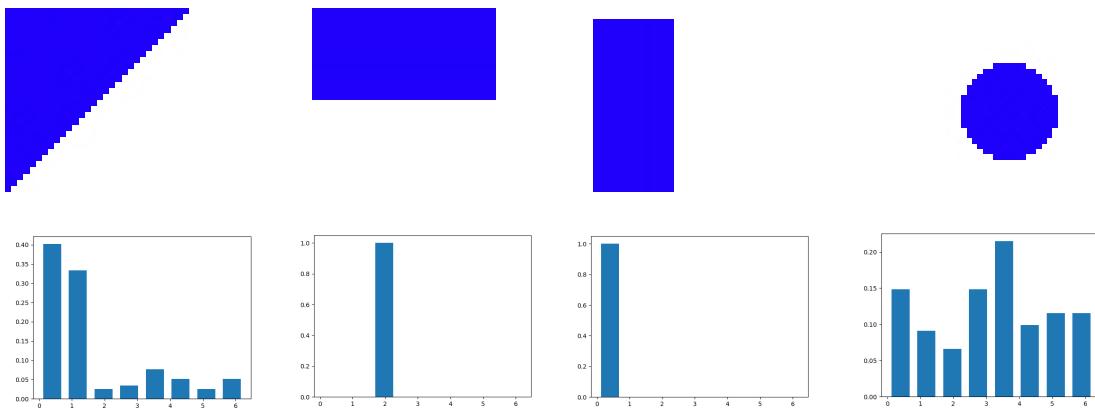


Abbildung 3: Histogram of Oriented Gradients Testauswertung. Obere Reihe zeigt eine Reihe von mitgelieferten Beispielbildern. Untere Reihe zeigt das Histogramm für die Gradienten im Bild.

Die Matrix enthält in I_{xx}, I_{yy}, I_{xy} die quadrierten Gradienten aus x, y , die über eine Nachbarschaft von 3×3 aufsummiert wurden. Der Ablauf ist also:

- a) Berechne Sobel in x- und y-Richtung (G_x, G_y).

Die Eigenwerte dieser Matrix unterscheiden sich dann für Kanten, glatte Regionen und Ecken in ihren Wertebereichen. Für glatte Regionen im Bild sind beide Eigenwerte sehr klein. Bei Kanten ist ein Eigenwert um mindestens eine Größenordnung größer als der andere und bei Ecken sind beide in der gleichen Größenordnung.

Zur effizienten Berechnung der Harris-Corners kann auf die Berechnung der Eigenwerte verzichtet werden. Es reicht aus die Determinante (\det) und die Spur Tr der Matrix zu berechnen:

$$H = \det(M) - k \cdot Tr(M)^2$$

Wie in der Vorlesung besprochen sind dafür folgende Punkte notwendig, die Sie in dieser Aufgabe implementieren sollen:

- a) Berechnung der Gradienten im Bild (x- und y-Richtung).
- b) Berechne I_{xx}, I_{yy}, I_{xy} als elementweise Multiplikation von G_x, G_y .
- c) Summiere I_{xx}, I_{yy}, I_{xy} über eine 3er-Nachbarschaft und schreibe diese in ein neues ND-Array $sumI_{xx}, sumI_{yy}, sumI_{xy}$. Dieser Schritt lässt sich mit einem Convolution2D Filter lösen. Nutzen Sie dafür bitte $cv2.filter2D$
- d) Daraus kann man dann die Matrix M aufbauen.
- e) Berechnung der Determinante von M durch $\det(M) = I_{xx} \cdot I_{yy} - I_{xy} \cdot I_{xy}$
- f) Berechnung der Spur von M über $Tr(M) = I_{xx} + I_{yy}$
- g) k ist ein Parameter und sollte zwischen 0.04 ... 0.06 liegen.
- h) Das finale Bild H enthält dann die Harris Eck-Merkmale die man wie oben errechnen kann. In einem finalen Schritt werden diese mit einem Schwellwert gefiltert.

`harris.py` enthält weitere Informationen zur Implementierung. Zum Test ihres Algorithmus können Sie wie im SourceCode angegeben die OpenCV Implementierung verwenden und die Ergebnisse vergleichen. Die Summe der Pixeldifferenzen zwischen ihrer und der OpenCV Implementierung sollte 0.0 sein (ggf. gibt es Floatingpoint Ungenauigkeiten). Bei höheren Abweichungen kommt es zum Punktabzug, ebenfalls bei uneffizienter Implementierung (Nutzung von unnötigen Loops). Ihre Ergebnisse sollten dann wie in Abbildung 4 aussehen.



Abbildung 4: Beispieldgebnisse des Harris-Corner Detectors bei voreingestellten Parameterwerte wie im SourceCode angegeben.

Aufgabe 5: Readings (freiwillig)

Bitte lesen sie die Veröffentlichungen, die im Bereich *Readings* im Moodle zu finden sind und rekapitulieren Sie die Vorlesung.

Abgabe: Dies ist *Übungsaufgabe 2*; die Bearbeitungszeit der Aufgabe ist für ca. 3 Wochen ausgelegt. **Die Abgabe erfolgt via Moodle zum angegebenen Zeitpunkt. Bitte fügen Sie alle notwendigen Bilder ein, sodass jede Aufgabe direkt ausführbar ist. Andernfalls behalte ich mir Punktabzüge vor.** Verspätete Abgaben werden mit einem Abschlag von 5 Punkten je angefangener Woche Verspätung belegt.