

Interactive Systems

Augmented Reality Marker Tracking

WiSe 2017/18

Aufgabe 1: Approximate Nearest Neighbors in OpenCV (2 Punkt)

Bitte setzen Sie sich mit dem *Fast Library for Approximate Nearest Neighbors (FLANN)* Modul in OpenCV auseinandern (http://opencv-python-tutorials.readthedocs.org/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html). Bauen Sie in Ihr bisheriges Videostream-Beispiel aus (vorheriges Aufgabenblatt, Aufgabe 1) und visualisieren Sie die in der Vorlesung besprochenen Feature Matches wie im Bild 1 dargestellt. Nutzen Sie für die Darstellung der Matches die Funktion `draw_matches` aus der Datei `ImageStitcher.py`.

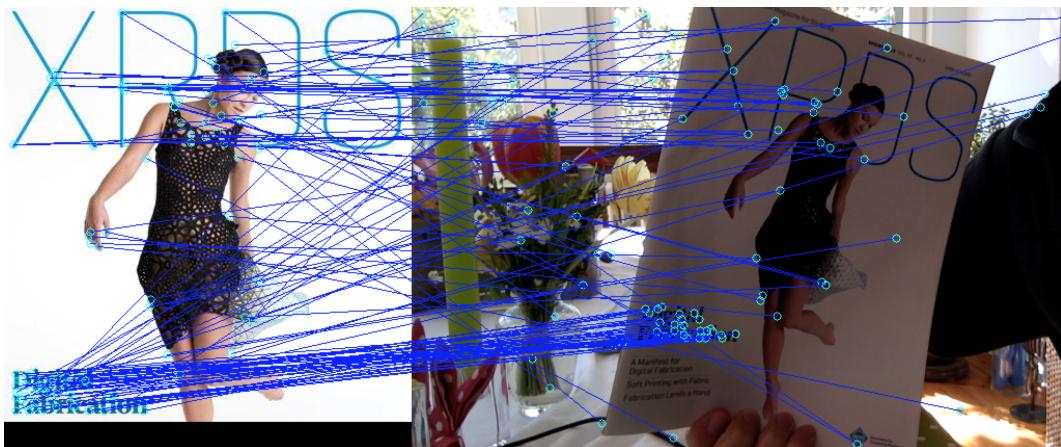


Abbildung 1: Links: Marker, Rechts: Videostream. Visualisierung der Feature Matches

Aufgabe 2: k-means (4 Punkte)

Die *Fast Library for Approximate Nearest Neighbors (FLANN)* benutzt – wie in der Vorlesung besprochen – unter anderem eine Clustering oder Quantisierungs-Methode namens k-means. Dabei wird aus einer Menge von ähnlichen Datenpunkten eine vorab bekannte Anzahl von k Gruppen gebildet. Diese Standardmethode ist eine der häufigsten verwendeten Techniken zur Gruppierung von Datenpunkten, z.B. RGB-Pixelwerte. Dieser soll im folgenden für eine farbbasierte Segmentierung bzw. Quantisierung von Ihnen implementiert werden (siehe Abbildung 2).

Die Idee des *Color-based Segmentation* auf Basis des k-means Algorithmus ist, eine Quantisierung/Segmentierung der Farben im Bild vorzunehmen und dabei aber ähnliche Farben im Bild zu einer Gruppe zusammenzufassen. Die Ähnlichkeit zwischen Pixelwerten lässt sich wieder über die euklidische Distanz zwischen RGB, LAB oder HSV Werten bestimmen.

Implementieren Sie auf Basis des vorgegebenen Sourcecodes und der Kommentare den k-means Algorithmus und testen diesen für unterschiedliche k und unterschiedliche Farträume wie in Abbildung 2 dargestellt. Nutzen Sie dafür entweder die vorgegebenen Farben (`cluster_colors`) oder jeweils die Farben

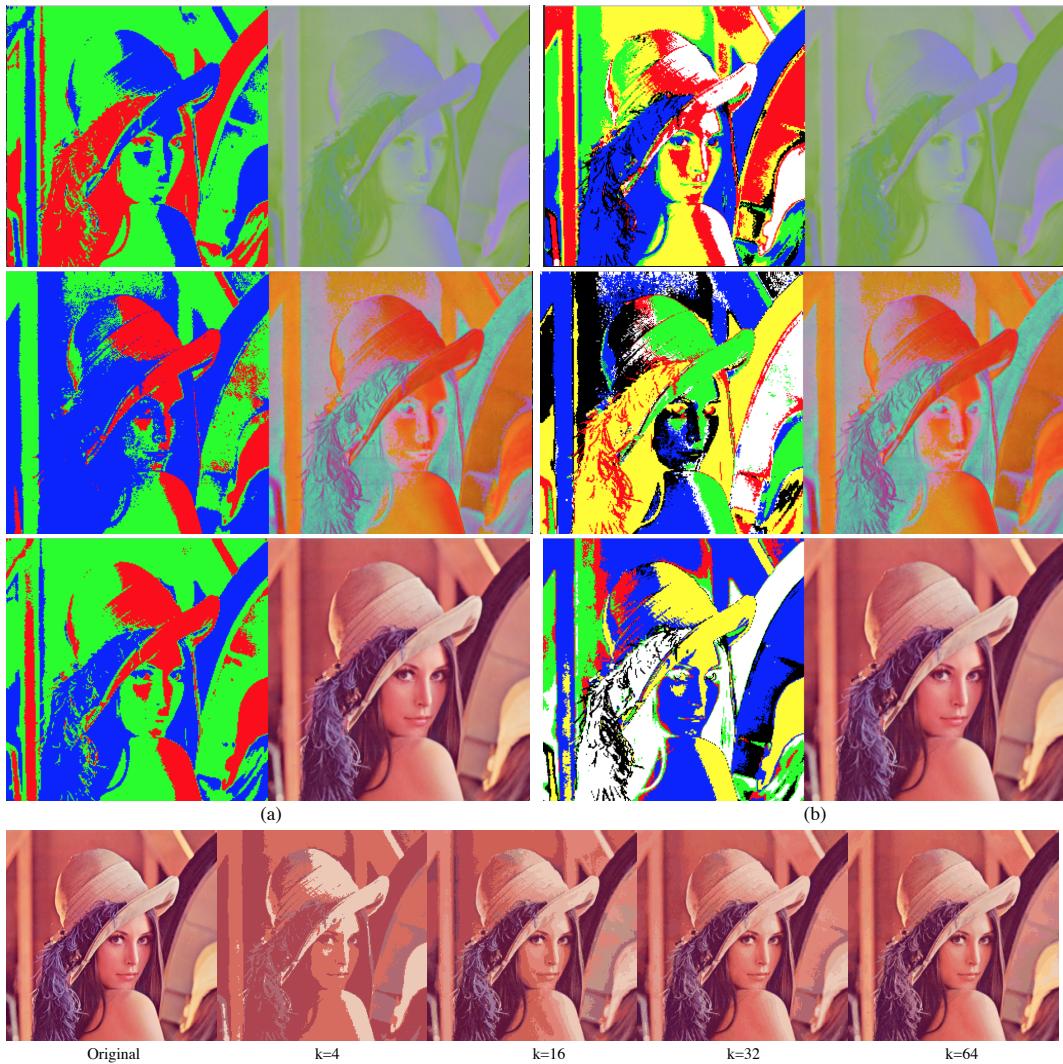


Abbildung 2: k-means basiertes Farbclustering. Oben: (a) k-means mit $k = 3$ über das Eingabebild im LAB-, HSV-, und RGB-Farbraum. (b) k-means mit $k = 6$ über die gleichen Farbräume. Unten: Farbquantisierung mit wachsendem $k = 4 \dots 64$

des Clusterzentrums (mean color). **Berechnen Sie zu jedem Ergebnis bitte den Gesamtfehler und geben diesen aus.**

Bitte denken Sie ein wenig über die Ergebnisse ihres Clustering nach und beantworten (zur Abnahme) folgende Fragen:

- Welche Probleme hat dieser Clustering Algorithmus?
- Wie kann ich die Ergebnisse verbessern?

Zusatzaufgabe (1 Punkt): Implementieren Sie ein Verbesserung des Algorithmus. **Hinweis:** Wenn es Ihnen notwendig erscheint, können Sie den vorgegebenen Sourcecode gern abändern.

Aufgabe 3: RANSAC Outlier Filtering (5 Punkt)

Der RANSAC Algorithmus *Random Sample Consensus* ist ein Standard Algorithmus zur Schätzung eines Modells einer Messreihe mit Ausreißern, Rauschen und Fehlern. RANSAC wird wegen seiner Robustheit in vielen Bereichen, insbesondere in der Computer Vision und des Maschinellen Sehens eingesetzt, da hier Ausgleichsverfahren wie die Methode der kleinsten Quadrate, die bei einer größeren Anzahl von Ausreißern meist versagen. In dieser Übungsaufgabe sollen Sie den RANSAC Algorithmus für eine 2D Linienmodell selbst implementieren. Die Klasse *RansacPointGenerator* erzeugt Ihnen eine Menge von

Punkte inkl. einer Reihe von Ausreißern. Vervollständigen Sie die Klasse *Ransac*. Erzeugen Sie eine Reihe von Plots (mindestens 5) ähnlich wie in Abbildung 3 mit unterschiedlichen Punktmengen (3) und unterschiedlichen Ransac Parametern (2) und seien Sie zur Abnahme in der Lage die Ergebnisse sinnvoll zu interpretieren. **Hinweis:** Installieren Sie für diese Aufgabe ggf. das python Paket *matplotlib* nach (*conda install matplotlib*).

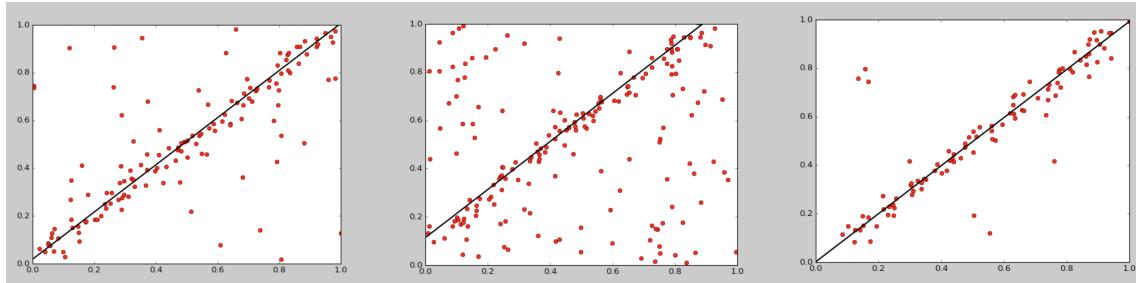


Abbildung 3: RANSAC - Beispielhafter Ergebnisplots für unterschiedliche Punktverteilungen

Aufgabe 4: Homography und Pose Estimation (4 Punkte)

Ziel dieser Aufgabe ist die Implementierung eines einfachen markerbasierten Augmented Reality Tracking in OpenCV. Dafür können Sie den Source-Code der Aufgabe 1 wiederverwenden. Allerdings können Sie die *drawMatches*-Methode wieder herausnehmen und auch die Splitscreen-Darstellung brauchen wir nicht mehr. Gehen Sie am besten zurück zu der einfachen Videodarstellung. Einiges an Sourcecode ist bereits in *ar.py* vorgegeben.

Geben Sie zusätzlich eine kleine Textdatei ab, in der die folgenden Fragen beantwortet werden:

- Was passiert, wenn Sie das Distanz-Filtering aus dem gegebenen Sourcecode nicht nutzen?
- Wie sieht die OpenCV Datenstruktur der Matches aus?
- Was passiert wenn Sie den Thresholdwert in *cv2.findHomography* ändern und warum?

Abschließend muss nur noch das virtuelle Objekt an der richtigen Stelle im Bild gezeichnet werden. Dafür müssen alle 3D Punkte des Objektes an der korrekten Stelle im Bildraum dargestellt werden, also 3D nach 2D Korrespondenzen auf Basis des transformierten Markers gefunden werden. Der vorgegebene



Abbildung 4: Virtuelles Objekt auf vorher definiertem Marker.

Sourcecode enthält bereits die Funktion `render_virtual_object`. Implementieren Sie hier zunächst eine Liste von Punkten und Kanten, die einen Quader bilden sollen (so wie in Abbildung 4 dargestellt). Füllen Sie dann die weiteren Lücken im gegebenen Sourcecode.

Aufgabe 5: Readings

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

Abgabe: Dies ist *Übungsaufgabe 3*; die Bearbeitungszeit der Aufgabe ist für ca. 2 Wochen ausgelegt.
Die Abgabe erfolgt via Moodle zum angegebenen Zeitpunkt.