4 Science Challenge, 23. Wettbewerb

Aufgabe 4: Bauteile vermessen mittels Computer Vision

Diese Aufgabe wird vom **SFB 1153 – Prozesskette zur Herstellung hybrider Hochleistungsbauteile durch Tailored Forming der** Fakultät für Maschinenbau gestellt. Falls ihr Interesse am Studiengang Maschinenbau habt, findet ihr hier weitere Informationen: https://www.maschinenbau.uni-hannover.de

Alle Teile der Aufgabe beinhalten einen Programmieranteil. Daher ist es wichtig, dass ein Python-Interpreter zur Verfügung steht. Zu empfehlen ist Google Colab (https://colab.research.google.com), da es alle erforderlichen Python-Pakete für die Bearbeitung der Aufgaben bereitstellt.

Aufgabe 1 (10 Punkte) – Grundlagen: Wie sehen Computer?

a) Recherchiert die Grundlagen der Bildaufnahme und wie Computer Bilder verarbeiten. Fokussiert euch insbesondere auf die Erfassung von Farbinformationen in Bildern und deren Interpretation durch Computer. In der Recherche sollten die folgenden Aufgaben behandelt werden:
   a. Erkundet die Python-Entwicklungsumgebung, um mit grundlegenden Bildverarbeitungsfunktionen vertraut zu werden. Dabei solltet ihr erlernen, wie Python-Funktionen aufgerufen werden. Schaut euch zudem die Python-Bibliothek OpenCV an. Erklärt anhand eines theoretischen Beispiels, wie Farben in digitalen Bildern repräsentiert werden.
   b. Ladet ein Bild eurer Wahl in Python hoch. Erstellt für dieses Bild ein Histogramm mithilfe der im Anhang befindlichen Funktionen.
   c. Konvertiert das Bild in Graustufen und erstellt ebenfalls ein Histogramm davon. Vergleicht beide Histogramme und beschreibt die Unterschiede.

b) Informiert euch im Internet, welche Herausforderungen bei der Bildaufnahme auftreten können und welche Schwierigkeiten entstehen, wenn Computer diese Bilder verarbeiten sollen. Schreibt fünf Beispiele auf, welche Einflüsse eine Bildaufnahme komplizierter machen.

Aufgabe 2 (10 Punkte) – Mathematische Grundlagen der thermischen Ausdehnung

a) In einer Internetrecherche sollt ihr die mathematischen Beschreibungen für die Ausdehnung von metallischen Bauteilen herausfinden. Beschreibt die in der Recherche ermittelten Formeln und welchen Einfluss einzelne Parameter/Variablen haben.

b) Im Folgenden soll die thermische Ausdehnung einer Eisenkugel berechnet werden. Gegeben sind die folgenden Parameter:
   - Durchmesser einer Eisenkugel: 20 cm
   - Längenausdehnungskoeffizient α von Eisen: 12,2 * 10^{-6} 1/K
   - T_0 (vor der Erwärmung): 20 °C
   - T_1 (in einem Schmiedeofen): 1300 °C
   - T_2 (nach dem Herausnehmen aus dem Schmiedeofen): 1220 °C

   Berechnet basierend auf den gegebenen Parametern sowohl die Längen- als auch die volumetrische Ausdehnung für die Zeitpunkte T_1 und T_2.

Aufgabe 3 (10 Punkte) – Messen der metallischen Ausdehnung anhand von Bildern mit Python

Jetzt sollt ihr die beiden vorher gelernten Konzepte kombinieren. Auf der Website der 4 Science Challenge findet ihr zwei Bilder: Bild 1 zeigt die bereits erwähnte Eisenkugel, bevor sie in den Schmiedeofen gelegt wird, während Bild 2 die erhitzte Kugel nach dem Herausnehmen aus dem Schmiedeofen zeigt.

a) Ladet – ähnlich wie in Aufgabe 1 – das Bild 1 in Python mithilfe der bereitgestellten Funktionen hoch. Konvertiert das Bild zunächst in ein Graustufenbild und erstellt ein Histogramm davon. Basierend auf dem Graustufenbild und dem Histogramm soll ein Binärbild erstellt werden. Pixel in einem Binärbild können nur zwei Werte annehmen: Null oder Eins. Dies soll durch eine von euch erstellte Segmentierungsfunktion umgesetzt werden. Informiert euch im Internet über hilfreiche Methoden. Das Ziel dieser Funktion soll sein, alle Pixel der Eisenkugel auf Eins zu setzen und alle umgebenden Pixel auf den Wert Null.

b) Basierend auf dem Graustufenbild soll nun der Radius ermittelt werden. Dazu soll die im Anhang bereitgestellte Funktion "ermitteln_des_radius" verwendet werden. Diese Funktion liefert den Radius eines in dem Graustufenbild erkannten Kreises zurück. Mit diesem ermittelten Wert soll nun die theoretische Längenausdehnung berechnet werden. Dafür soll eine weitere Funktion erstellt werden, die den Radius als Eingabeparameter erhält. Anhand dieses Radius wird mithilfe der Parameter aus Aufgabe 2b die Ausdehnung berechnet.

c) Führt den Prozess der Aufgabenteile 3a und 3b auch für das zweite Bild durch. Vergleicht die berechnete Ausdehnung aus Aufgabenteil 2b mit dem gemessenen Radius aus dem zweiten Bild. Berechnet die prozentuale Abweichung zwischen dem berechneten und dem gemessenen Wert. Überlegt euch, aus welchen Gründen mögliche Abweichungen entstehen könnten.

Viel Erfolg bei der vierten Aufgabe!

Allgemeine Hinweise
Einsendeschluss: Sonntag, 04. Februar 2024, 19:59 Uhr

Gebt eure Lösungen über Stud.IP ab: https://studip.uni-hannover.de

Das zulässige Dateiformat für die zusammengeschriebene Lösung (mit eingebetteten Bildern) ist PDF. Bitte ladet eure Dateien rechtzeitig hoch.

Gebt innerhalb der Datei euren Teamnamen, die Namen der Teammitglieder sowie deren Schulen an. Benennt eure Datei nach folgendem Schema: „Teamname_Aufgabe4".

Das Hochladen funktioniert wie folgt:

Loggt euch mit den bei eurer Anmeldung zur 4 Science Challenge angelegten Zugangsdaten auf der Stud.IP-Seite ein (bitte nutzt dazu den „Login ohne WebSSO"). Geht dann auf „Meine Veranstaltungen" und auf die 4 Science Challenge 2023/2024. Geht dann oben auf „Dateien" und auf den Ordner „Upload Aufgabe 4". Dort könnt ihr entweder über „Dokument hinzufügen" oder über „Dateien hochladen" eure Lösungsdatei hochladen.

Wenn ihr die Datei hochgeladen habt, öffnet sich ein Fenster, in dem u. a. nach Lizenzinformationen gefragt wird. Dieses braucht ihr nicht weiter zu beachten und könnt einfach auf „Speichern" klicken.

Bitte achtet darauf, dass ihr eure Dateien wirklich innerhalb des Ordners „Upload Aufgabe 4" hochladet und nicht außerhalb davon, da ansonsten die anderen Teams eure Dateien sehen können.

Die Teilnahmebedingungen und weitere Informationen findet ihr unter
www.uni-hannover.de/4sciencechallenge

Der Rechtsweg ist ausgeschlossen.

Anhang:
Notwendige Python-Pakete:

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
import matplotlib.pyplot as plt
```

Einlesen eines Bildes:
Diese Funktion "einlesen_eines_bildes" nimmt einen Dateipfad zu einem Bild als Eingabe und verwendet die OpenCV-Bibliothek, um das Bild von der angegebenen Datei zu lesen. Das eingelesene Bild wird als eine große Matrix repräsentiert und am Ende der Funktion zurückgegeben.

```
def einlesen_eines_bildes(pfad_zum_bild):
    bild = cv2.imread(pfad_zum_bild)
    return bild
```

Anzeigen eines Bildes:
Wenn kein Google-Colab verwendet wird, kann die folgende Funktion verwendet werden. Diese Funktion "anzeigen_eines_bildes" nimmt eine Matrix als Bildrepräsentation ("matrix_bild") als Eingabe und verwendet die OpenCV-Bibliothek, um das Bild in einem Fenster mit dem Titel "Bild" anzuzeigen. Die Funktion blockiert die Ausführung des Programms, bis der Benutzer eine Taste drückt, und schließt dann das Bildfenster, wenn eine Taste gedrückt wird. Dies ermöglicht es dem Benutzer, das Bild anzuzeigen und es zu schließen, wenn er fertig ist.

```
def anzeigen_eines_bildes(matrix_bild):
    cv2.imshow('Bild', matrix_bild)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Wird Google Colab verwendet, kann die folgende Funktion genutzt werden:

```
def anzeigen_eines_bildes(matrix_bild):
    # Hier wird das Bild angezeigt.
    cv2_imshow(matrix_bild)
```

Konvertieren eines Bildes:
Diese Funktion "konvertieren_in_grauwertbild" nimmt eine Matrix als Eingabe, die ein Farbbild repräsentiert ("matrix_als_farbbild"). Sie verwendet die OpenCV-Bibliothek, um diese Farbmatrix in ein Grauwertbild umzuwandeln. Das resultierende Grauwertbild wird dann von der Funktion zurückgegeben.

```
def konvertieren_in_grauwertbild(matrix_als_farbbild):
    matrix_grauwert = cv2.cvtColor(matrix_als_farbbild, cv2.COLOR_BGR2GRAY)
    return matrix_grauwert
```

Erstellen eines Histogramms
Die Funktion "erstellen_eines_histogramms" erstellt ein Histogramm für ein gegebenes Bild. Zuerst werden die Häufigkeiten der Bildwerte gezählt und in einer Liste "hist_bild" gespeichert. Falls es sich um ein Farbbild handelt, werden Histogramme für die Farbkanäle (Blau, Grün, Rot) erstellt und in einem Diagramm dargestellt. Für Grauwertbilder wird ein Histogramm erstellt und dargestellt. Die Funktion verwendet NumPy für die Histogrammberechnung und Matplotlib zur Visualisierung der Histogramme. Am Ende werden die Histogramme je nach Bildtyp angezeigt.

```
def erstellen_eines_histogramms(matrix_bild):
    hist_bild = []
    bins = np.linspace(0, 255, 256, endpoint=True, dtype=int)
    # Hier wird gezählt, wie häufig Zahlen im Bild auftreten.
    if len(matrix_bild.shape) == 3:  # Farbbild
        for i in range(matrix_bild.shape[-1]):
            farb_kanal = np.matrix.flatten(matrix_bild[:, :, i])
            haufigkeiten = np.histogram(farb_kanal, bins)[0]
            haufigkeiten = np.append(haufigkeiten, 0)
            hist_bild.append(haufigkeiten)

        plt.figure(figsize=(6, 6))
        plt.title('Farbbild')
        plt.ylabel('Häufigkeit')
        plt.xlabel('Wert')
        colors = ['blue', 'green', 'red']
        for n in range(len(hist_bild)):
            plt.plot(bins, hist_bild[n], color=colors[n], alpha=0.3)
            plt.fill_between(bins, hist_bild[n], color=colors[n], alpha=0.3)
        plt.legend(['Blau', 'Grün', 'Rot'])
    else:  # Grauwertbild
        farb_kanal = np.matrix.flatten(matrix_bild)
        haufigkeiten = np.histogram(farb_kanal, bins)[0]
        haufigkeiten = np.append(haufigkeiten, 0)
        hist_bild.append(haufigkeiten)

        plt.figure(figsize=(6, 6))
        plt.title('Grauwertbild')
```

```
        plt.ylabel('Häufigkeit')
        plt.xlabel('Wert')
        plt.plot(bins, hist_bild[0], color='grey', alpha=0.3)
        plt.fill_between(bins, hist_bild[0], color='grey', alpha=0.3)
        plt.legend(['Grauwert'])

    plt.tight_layout()
    plt.show()
```

Binarisierung eines Bildes

Diese Funktion "bild_binarisierung " führt eine Binarisierung auf einem gegebenen Bild ("matrix_bild") durch, wobei Werte unterhalb einer unteren Schwelle ("untere_grenze") auf Weiß und Werte oberhalb einer oberen Schwelle ("obere_grenze") ebenfalls auf Weiß gesetzt werden. Alle Werte dazwischen werden auf Schwarz gesetzt. Das Ergebnis ist ein binäres Bild, bei dem nur zwei Werte (Weiß und Schwarz) vorhanden sind, und es wird als Matrix zurückgegeben.

```
def bild_binarisierung(untere_grenze, obere_grenze, matrix_bild):
    matrix_bild = np.where(matrix_bild < untere_grenze, 0, matrix_bild)
    matrix_bild = np.where(matrix_bild > obere_grenze, 0, matrix_bild)
    matrix_bild = np.where(matrix_bild != 0, 255, matrix_bild)
    return matrix_bild
```

Ermitteln des Radius der Kugel:

Diese Funktion "ermitteln_des_radius" nimmt ein Grauwertbild ("grau_bild") und ein Farbbild ("farb_bild") sowie einen optionalen Parameter "kreise_Zeigen" entgegen. Zuerst wird das Grauwertbild mit einem Medianfilter geglättet, um Rauschen zu reduzieren. Dann wird die Hough-Transformation verwendet, um Kreise im Bild zu erkennen.

Wenn mehr als zwei Kreise erkannt werden, wird eine Meldung ausgegeben und die Funktion gibt None zurück. Wenn überhaupt keine Kreise erkannt werden, wird ebenfalls eine Meldung ausgegeben, und die Funktion gibt None zurück.

Wenn "kreise_Zeigen" auf True gesetzt ist, werden die erkannten Kreise im Farbbild markiert und das Bild wird angezeigt.

Wenn nur ein Kreis erkannt wird, wird der Radius dieses Kreises verwendet, um den Durchmesser, den Abstand zum Bauteil, die Brennweite der Kamera, die Breite des Sensors und die Breite des Bildes zu berücksichtigen, um den Radius der Kugel in Metern zu berechnen. Dieser Radius wird zurückgegeben.

In jedem Fall wird der ermittelte Radius oder None je nach Ergebnis ausgegeben.

```
def ermitteln_des_radius(grau_bild,farb_bild,kreise_Zeigen=True):

    grau_bild = cv2.medianBlur(grau_bild, 3)
    circles = cv2.HoughCircles(grau_bild, cv2.HOUGH_GRADIENT, dp=2, minDist=800,
                               param1=150, param2=70, minRadius=250, maxRadius=500)
    print(len(circles[0]))
    if len(circles[0]) > 2:
      print(circles[0])
```

```
    print('Es wurden mehrer Kreise gemessen, binaeres Bild anpassen')
    return None
if circles is None:
    print('Kein Kreis gefunden -> Es kann kein Radius ermittelt werden')
    return None

if kreise_Zeigen:
        circles = np.uint16(np.around(circles))
        for i in circles[0, :]:
            # Kreismittelpunkt zeichnen
            cv2.circle(farb_bild, (i[0], i[1]), 2, (255, 255, 0), 3)
            # Kreisumfang zeichnen
            cv2.circle(farb_bild, (i[0], i[1]), i[2], (255, 0, 255), 3)

        anzeigen_eines_bildes(farb_bild)

circle = circles[0, 0]
radius = circle[2]
durchmesser = radius * 2
abstand_bauteil = 0.50
brennweite = 35
sensor_breite = 36
bild_breite = grau_bild.shape[1]
radius_kugel = (durchmesser * abstand_bauteil) / ((brennweite * bild_breite) / sensor_breite)
print('Radius der Kugel in m: ', radius_kugel)
return radius_kugel
```