Geometrische Transformationen und Grauwertinterpolation

Aufgabenstellung:

1) Es ist ein Python-Programm zu schreiben, mit dessen Hilfe das Quellbild (kalib.bmp) "*affin transformiert*" wird, also mit den Formeln:

$$x_q = a_0 + a_1 x_z + a_2 y_z$$
$$y_q = b_0 + b_1 x_z + b_2 y_z$$

Hierzu soll die Target-to-Source-Methode verwendet werden, d.h. ausgehend von den Zielbildkoordinaten werden die korrespondierenden Quellbildkoordinaten berechnet. Mit Hilfe der errechneten Quellbildkoordinaten wird dann der Zielgrauwert bestimmt.

Zwei Varianten sind zu realisieren. Zur Berechnung des Zielgrauwertes soll

a) der Grauwert des nächstliegenden Quellbildpunktes verwendet werden (*Nearest-Neighbor-Interpolation*),
b) die *bilineare Interpolation* verwendet werden.

Da es sein kann, dass die berechnete Quellbildkoordinate nicht im Bild liegt, ist mit der Funktion *outsideImage*(…) (s.u.) zu prüfen, ob ein Quellbildpunkt existiert. Wenn nicht, ist der Zielbildpunkt schwarz zu setzen.

2) Es ist ein weiteres Python-Programm zu schreiben, so dass das Quellbild mit der 4-Punkte-Transformation in das Zielbild transformiert wird.

$$\hat{x}_q = \sum_{i=1}^{4} \Phi_i(\hat{x}_z, \hat{y}_z) \cdot \hat{x}_{q_i} \quad \text{und}$$
$$\hat{y}_q = \sum_{i=1}^{4} \Phi_i(\hat{x}_z, \hat{y}_z) \cdot \hat{y}_{q_i}$$

$$\Phi_1(\hat{x}_z, \hat{y}_z) = (1-\hat{x}_z)\cdot(1-\hat{y}_z)$$
$$\Phi_2(\hat{x}_z, \hat{y}_z) = \hat{x}_z \cdot (1-\hat{y}_z)$$
$$\Phi_3(\hat{x}_z, \hat{y}_z) = \hat{x}_z \cdot \hat{y}_z$$
$$\Phi_4(\hat{x}_z, \hat{y}_z) = (1-\hat{x}_z)\cdot \hat{y}_z$$

Wenden Sie auch hier
a) die Nearest-Neighbor-Interpolation
b) die bilineare Interpolation an.

Geometrische Transformationen und Grauwertinterpolation

Vorzubereiten:

Zu 1) Für die affine Transformation sind diejenigen Transformationsparameter zu berechnen (mit der Determinantenmethode), mit deren Hilfe die angegebenen Quellbildkoordinaten auf die korrespondierenden Zielbildkoordinaten abgebildet werden:

Quellkoordinaten		Zielkoordinaten
(98, 465)	=>	(5, 500)
(150,268)	=>	(5, 100)
(415, 269)	=>	(605,100)

Zu 2) Führen Sie die 4-Punkte-Transformation so aus, dass die abgebildete Kalibrierplatte näherungsweise perspektivisch korrigiert wird (senkrechte Aufsicht). Als Parameter können z.B. die Koordinaten der 4 kreisförmigen Eckmarken verwendet werden :
(196, 100), (583, 95), (666, 471), (100, 466)

Hinweise zur Aufgabenstellung:

Die Zielbilder werden durch Iteration über die Zielbildpunkte berechnet (zwei geschachtelte for-Schleifen). Für alle Bilder und sonstige Größen sind numpy-Arrays und -Funktionen zu verwenden.

Python-Listen, -Tupel und -Dictionaries sollen nicht verwendet werden. Vorgefertigte Funktionen zur Bildtransformation, wie sie z.B. in den Python-Bibliotheken PIL, scikit-image oder opencv enthalten sind, dürfen nicht verwendet werden.

Zwei Hauptfunktionen sind zu entwickeln:

1)
```
def AffineTransformation(src, a, method=0)
        ...
        return dst
```

Parameter: Quellbild `src`
Parametermatrix a = [(a_0, a_1, a_2), (b_0, b_1, b_2,)] für affine Transf.
`method=1` : bilineare Interpolation, sonst Interpol. durch Rundung

Rückgabe: transformiertes Zielbild `dst`

Geometrische Transformationen und Grauwertinterpolation

2) `def VierPunkteTransformation(src, eckp, method=0)`
`...`
`return dst`

Parameter: Quellbild `src`
Parametermatrix eckp = $[(x_1, y_1), \ldots (x_4, y_4)]$ für 4-Punkte-Transf.
`method=1` : bilineare Interpolation, sonst Interpol. durch Rundung

Rückgabe: transformiertes Zielbild `dst`

Sowohl für die Affine Transformation als auch für die 4-Punkte-Transformation werden zwei Hilfsfunktionen benötigt, die ebenfalls zu entwickeln sind:

a) `def outsideImage(coordinate, maxsize)`
`...`
`return isOutside`

Rückgabe: `True`, wenn `coordinate` nicht im Interval `[0, maxsize)` liegt.
`False` sonst

b) `def bilinInterpol(Xtilde, Ytilde, src)`
`...`
`return interpolatedValue`

Zweck: Berechnet mit Hilfe der bilinearen Interpolation anhand der Quellbildkoordinate (`Xtilde,Ytilde`) (beides reelle Zahlen) sowie des Quellbildes `src` (Datentyp `uint8`) den interpolierten Grauwert.

Rückgabeparameter: Bilinear interpolierter Grauwert.

Abnahme:

Abgenommen werden die (gut kommentierten) Jupyter-Notebooks *AffineTransformation.ipynb* und *VierPunkteTransformation.ipynb*.

- handschriftliche Berechnung (mit Determinantenmethode) der Transformationsparameter für die affine Transformation (Screenshot)
- Ergebnisbilder der affinen Transformation (beide Interpolationsverfahren)
- Ergebnisbilder der 4-Punkte-Transformation (beide Interpolationsverfahren)

Geometrische Transformationen und Grauwertinterpolation

Python/Numpy-Tips und Beispiele:

Anlegen von numpy-Arrays: `Arr = np.array([[1, 2, 3],[4, 5, 6]])`

Typkonversion von Zahlen (s. numpy-Doku):

```
x1 = np.uint8(167.0)  # 8-bit unsigned
x2 = np.int(167.0)    # signed integer
X3 = np.float(16)     # float
```

Für die Datentypen der Ergebnisse numerischer Operationen (Op) gilt (Beispiele):

uint8 Op uint8 → uint8
uint8 Op uint16 → uint16
uint8 Op int → int

Soll beispielsweise die Operation „uint8 Op uint8“ im Integerformat (also vorzeichenbehaftet) durchgeführt werden, müssen die Operanden vorher konvertiert werden.

Den Datentyp einer Zahl oder Variable erhält man z.B. mit:
`type(12), type(a), type(src[0][0]), …`

Die Form eines Arrays A erhält man z.B. mit `A.shape`

Rundung auf eine ganze Zahl: `np.rint(17.634)` → 18.0

Abrunden auf eine ganze Zahl: `np.floor(17.634)` → 17.0