Podstawy cyfrowego przetwarzania obrazów

- praca z myszką oraz transformacje geometryczne obrazów

Autor: dr inż. Jakub Markiewicz

Spis treści

Wykorzystanie funkcji myszy w przetwarzaniu obrazów

Jedną z najważniejszych funkcji, która jest wykorzystywana podczas pracy z przetwarzaniem obrazu jest zdarzenie myszy. OpenCV wspiera wykrywanie zdarzeń myszy. W tym rozdziale dowiesz się jak ją wywołać, wybrać lewy róg i prawy dolny róg prostokąta i narysować go na obrazie.

W funkcji 0 został pokazany kod do rysowania prostokąta. Najpierw należy raz kliknąć lewym przyciskiem myszy, aby zainicjalizować funkcję i wybrać lewy górny róg, a następnie dwa razy lewym przyciskiem myszy, aby określić współrzędne prawego dolnego rogu i narysować prostokąt.

```
import cv2
import numpy as np

points = []
# mouse callback function
def draw_rectangle(event, x, y, flags, param):

#Storage point coordinates in the global variables
    global points
    if event == cv2.EVENT_LBUTTONDBLCLK:
        points = [(x,y)]
    elif event == cv2.EVENT_RBUTTONDBLCLK:
        points.append((x,y))

    #Draw rectangle on the image
    if np.size(points, 0) == 2:
        cv2.rectangle(image, points[0], points[1], (0, 255, 0), 2)
    else:
        print('Not enough number of points')

%Main code
img = cv2.imread(file, arg)
cv2.namedWindow('image')
cv2.setMouseCallback('image', draw_rectangle)

While True:
    cv2.imshow('image',img)
    if cv2.waitKey(20) & 0xFF == 27: #Esc button
        break
cv2.destroyAllWindows()
```

(0)

Spróbujmy przygotować znacznie bardziej rozbudowaną aplikację, która wykorzystuje funkcję interaktywnego przeciągania, jak w aplikacji Paint (0.1):

```
import cv2
import numpy as np

drawing = False #True if mouse is pressed
s_x, s_y = -1,-1

# mouse callback function
def draw_rectangle(event, points,flags,param):

#Storage point coordinates in the global variables
    global s_x, s_y,drawing
    if event == cv2.EVENT_LBUTTONDBLCLK:
        drawing = True
        s_x, s_y = x,y
    elif event == cv2.EVENT_MOUSEMOVE:
```

(0.1)

```
        if drawing = True:
                    cv2.rectangle(img,(ix,iy),(x,y),(0,255,0),2)
    elif event == cv2.EVENT_LBUTTONUP:
        drawing = False
                    cv2.rectangle(img,(ix,iy),(x,y),(0,255,0),2)
%Main code
img = cv2.imread(file, arg)
cv2.namedWindow('image')
cv2.setMouseCallback('image', draw_rectangle)

while(1):
    cv2.imshow('image',img)
    if cv2.waitKey(20) & 0xFF == 27: #Esc button
        break
cv2.destroyAllWindows()
```

Przedstawiony przykład może być bardzo pomocny w tworzeniu i zrozumieniu niektórych interaktywnych aplikacji, takich jak śledzenie obiektów, segmentacja obrazu itp.

Lista dostępnych *event'ów:*

- EVENT_MOUSEMOVE
- EVENT_LBUTTONDOWN
- EVENT_RBUTTONDOWN
- EVENT_MBUTTONDOWN
- EVENT_LBUTTONUP
- EVENT_RBUTTONUP
- EVENT_MBUTTONUP
- EVENT_LBUTTONDBLCLK
- EVENT_RBUTTONDBLCLK
- EVENT_MBUTTONDBLCLK

oraz flag:

- EVENT_FLAG_LBUTTON
- EVENT_FLAG_RBUTTON
- EVENT_FLAG_MBUTTON
- EVENT_FLAG_CTRLKEY
- EVENT_FLAG_SHIFTKEY
- EVENT_FLAG_ALTKEY

Kombinacja tych parametrów pozwala na zwiększenie funkcjonalności aplikacji:

```
import cv2
# mouse callback function
def mouse_callback_function:
    if flags == (cv2. EVENT_FLAG_ALTKEY + EVENT_RBUTTONDOWN:
        print('Right mouse button is clicked while pressing ALT
key - position', x,' ' ,y))

%Main code
img = cv2.imread(file, arg)                                                    (0.2)
cv2.namedWindow('image')
cv2.setMouseCallback('image', mouse_callback_function)

while(1):
    cv2.imshow('image',img)
    if cv2.waitKey(20) & 0xFF == 27: #Esc button
        break
cv2.destroyAllWindows()
```

Transformacje geometryczne obrazów

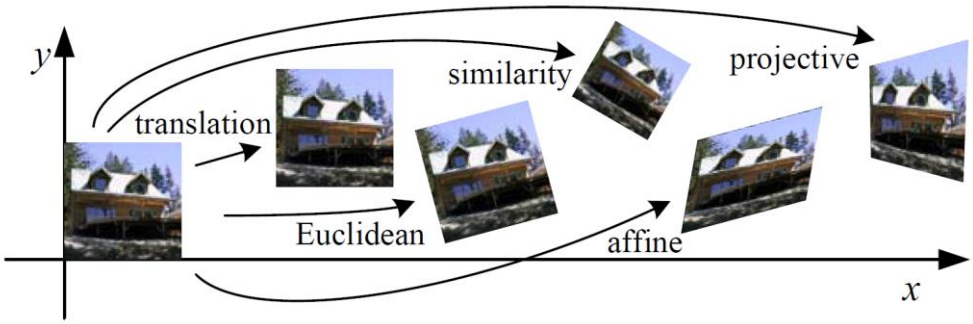
Transformacje geometryczne obrazów służą do przekształcenia współrzędnych pikseli zdjęcia wejściowego *I* do postacie zdjęcia wynikowego *I'* przy wykorzystaniu zależności geometrycznych opisanych ogólnym wzorem:

$$I(x,y) \rightarrow I'(x',y')$$

gdzie wartość funkcji obrazu Iw pierwotnym położeniu (x, y) przenosi się na nową pozycję (x', y') w przekształconym obrazie *I'* , a wartości elementów obrazu nie zmieniają się - jedynie ich położenie ulega zmianie. W celu matematycznego opisu tych zależności wykorzystuje się transformację 2D funkcją mapowania geometrycznego *T:*

$$T: R^2 \rightarrow R^2$$

$$T(x,y) = (x',y')$$



Rys. 1 Przykłady transformacji geometrycznych obrazów [1]

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	2	orientation	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	3	lengths	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	4	angles	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{2\times3}$	6	parallelism	
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{3\times3}$	8	straight lines	

Rys. 2 Hierarchia przekształceń współrzędnych 2D. Każde przekształcenie zachowuje również właściwości wymienione w rzędach *Preserve*, tzn. podobieństwo zachowuje nie tylko kąty, ale także równoległość i proste. Macierze 2 × 3 są rozszerzone o trzeci rząd $[0^T\ 1]$, tworząc pełną macierz 3×3 dla jednorodnych transformacji współrzędnych.[1]

Podstawowymi przekształceniami obrazów są: skalowanie, przemieszczanie, obracanie, transformacja afiniczna, transformacja perspektywiczna (Rys. 1). W zależności od wykorzystywanego przekształcenia

zachowuje różne właściwości i różną postać macierzy transformacji. Generalnie do przekształceń (poza samym skalowaniem) wykorzystywane są funkcje:

- Cv2.remap() – transformacja w oparciu o mapę owych pikseli,
- cv2.warpAffine() - transformacja afiniczna oparta na macierzy 2x3,
- cv2.warpPerspective() - transformacja perspektywiczna oparta na macierzy 3x3.

Przy transformacjach obrazu wykorzystywane są funkcje *resamplingu* (wyznaczania nowych wartości stopni szarości) a domyślnie przyjmowany jest sposób **cv2.INTER_LINEAR**). Dostępne są również metody **cv2.INTER_AREA** lub **cv2.INTER_CUBIC** .

Transformacja przy wykorzystaniu funkcji remap

Funkcja *remap* jest jedną z podstawowych funkcji biblioteki OpenCV wykorzystywaną do transformacji pikseli przy wykorzystaniu dowolnej funkcji geometrycznej lub dozwolonego sposobu przekształcania pikseli. Ilość przekształceń geometrycznych, które możesz wykonać za pomocą funkcji *remap* jest praktycznie nieograniczona i wszystko zależy od wykorzystywanych funkcji transformacji. Pozwala to wykonanie dowolnego przekształcenia geometrycznego, które można opisać funkcją matematyczną:

$$zdjecieWynikowe = zdjecieOryginalne(mapaPrzeksztalcenX(x,y), mapaPrzeksztalcenY(x,y))$$

gdzie wartości pikseli o współrzędnych innych niż całkowite są obliczane przy użyciu jednej z dostępnych metod interpolacji. $mapaPrzeksztalcenX$ i $mapaPrzeksztalcenY$ mogą być zakodowane jako oddzielne mapy zmiennoprzecinkowe. W OpenCV odpowiada za to funkcja:

zdjeciePoPrzeksztalceniu = cv2.remap(zdjecie, mapaPrzeksztalcenX, mapaPrzeksztalcenY, metodaInterpolacji)

gdzie:

- zdjęcie – zaimportowane zdjęcie w formacie *numpy*,
- zdjeciePoPrzeksztalceniu – zdjęcie po zastosowaniu transformacji w formacie *numpy.* Taki sam rozmiar jak zdjęcie wejściowe i mapaPrzeksztalcenX,
- mapaPrzeksztalcenX - pierwsza mapa punktów (x,y) lub tylko wartości x typu CV_16SC2 , CV_32FC1 lub CV_32FC2,
- mapaPrzeksztalcenY - druga mapa wartości y mająca typ CV_16UC1, CV_32FC1.
- metodaInterpolacji – metoda interpolacji; standardowa wersja **cv2.INTER_LINEAR.**

Przykład zastosowania funkcji *remap* do odbicia lustrzanego obrazu (1) oraz wprowadzania dystorsji (2):

```
import cv2
import numpy as np

zdjęcie = wczytane zdjęcie

#Przygotowanie mapy  przekształcającej zdjęcie
mapaPrzeksztalcenX = np.zeros((zdjecie.shape[0], zdjecie.shape[1]), dtype=np.float32)
mapaPrzeksztalcenY = np.zeros((zdjecie.shape[0], zdjecie.shape[1]), dtype=np.float32)

for i in range(mapaPrzeksztalcenX.shape[0]):
   mapaPrzeksztalcenX [i,:] = [x for x in range(mapaPrzeksztalcenX.shape[1])]
for j in range(mapaPrzeksztalcenY.shape[1]):
   mapaPrzeksztalcenY [:,j] = [mapaPrzeksztalcenY.shape[0]-y for y in range(mapaPrzeksztalcenY.shape[0])]
```

(1)

```
#Przekształcenie obrazu
wynikowyObraz = cv2.remap(zdjecie, mapaPrzeksztalcenX, mapaPrzeksztalcenY, cv2.INTER_LINEAR)

#Wyświetalnie obrazu

cv2.imshow('Oryginalne', zdjęcie)
cv2.imshow('Nowy obraz', wynikowyObraz)
cv2.waitKey(0)
```

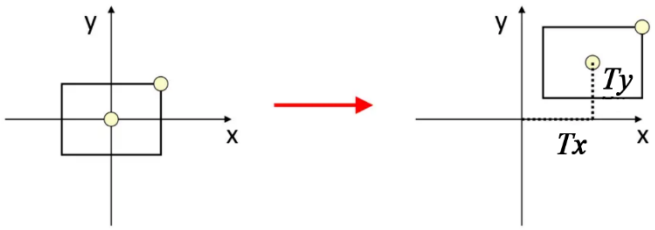
```
zdjecie = wczytane zdjęcie
zniekształcenie = 3
# Zapis do zmiennej wymiarów obrazu
(wysokosc, szerokosc, _) = zdjecie.shape
# Zdefiniowanie map przekształceń współrzędnych w formacie float32
mapaPrzeksztalcenX = np.zeros((wysokosc, szerokosc), np.float32)
mapaPrzeksztalcenY  = np.zeros(wysokosc, szerokosc), np.float32)

wspolrzędnaXsrodka = szerkosc/2
wspolrzędnaYsrodka = wysokosc/2
promienNormalizujacy = w/2
for y in range(wysokosc):
   deltaY = (y - wspolrzędnaYsrodka)
   for x in range(szerokosc):
      deltaX = (x - wspolrzędnaXsrodka)
      odlegloscOdSrodka = np.power(deltaX,2) + np.power(deltaY,2)
      if odlegloscOdSrodka >= np.power(promienNormalizujacy,2):
         mapaPrzeksztalcenX [y, x] = x
         mapaPrzeksztalcenY  [y, x] = y
      else:
         wspolczynnikZniekształcajacy = 1.0
         if odlegloscOdSrodka > 0.0:
            wspolczynnikiZniekształcenia = math.pow(math.sin(math.pi * math.sqrt(odlegloscOdSrodka) / promienNormalizujacy / 2),
zniekształcenie)
         mapaPrzeksztalcenX[y, x] = wspolczynnikiZniekształcenia * deltaX + wspolrzędnaXsrodka
         mapaPrzeksztalcenY[y, x] = wspolczynnikiZniekształcenia * deltaY + wspolrzędnaYsrodka
dst = cv2.remap(zdjecie, mapaPrzeksztalcenX, mapaPrzeksztalcenY, cv2.INTER_LINEAR)
```

(2)

Translacja obrazów

Translacja obrazu to prostoliniowe przesunięcie obrazu z jednego miejsca do drugiego przy wykorzystaniu następujących zależności::

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



Rys. 3 Przykład wykorzystania translacji obrazu

Translację przy wykorzystaniu funkcji OpenCV można wykonać przy wykorzystaniu macierzy 2 x 3 (kod 3; bez dolnego wiersza [0,0,1]) i funkcji *cv2.warpAffine* lub macierzy 3x3 i funkcji *cv2.warpPerspective* (4).

```
import cv2
import numpy as np
```

(3)

```
zdjęcie = wczytane zdjęcie

wysokosc, szerokosc, _ = zdjęcie.shape
macierzTranslacji = np.float32([[1,0,50],
                    [0,1,150]])
print(macierzTranslacji)
zdjeciePoTranslacji = cv2.warpAffine(zdjęcie, macierzTranslacji,(wysokosc, szerokosc))
cv2.imshow('Oryginalne', zdjęcie)
cv2.imshow('Nowy obraz', zdjeciePoTranslacji)
cv2.waitKey(0)
```

```
zdjecie = wczytane zdjęcie

# Zapis do zmiennej wymiarów obrazu
(wysokosc, szerokosc, _) = zdjecie.shape
macierzTranslacji = np.float32([[1,0,50],
                    [0,1,150],
                    [0,0,1]])
zdjeciePoTranslacji = cv2.warpPerspective(zdjecie, macierzTranslacji, (wysokosc, szerokosc))
cv2.imshow('Oryginalne', zdjęcie)
cv2.imshow('Nowy obraz', zdjeciePoTranslacji)
cv2.waitKey(0)
```

(4)

Skalowanie obrazów

Skalowanie obrazu to proces wykorzystywany do zmiany rozmiaru obrazu cyfrowego. OpenCV posiada wbudowaną funkcję *cv2.resize()*, która umożliwia przeskalowanie obrazów w oparciu o współczynniki skalujące lub do zdefiniowanego obszaru:

zdjeciePoPrzeksztalceniu = cv2.resize(zdjecie, (szerokosc, wysokosc), wspolczynnikSkalujacyX, wspolczynnikskalujacyY, metodaInterpolacji)

gdzie:

- zdjęcie – zaimportowane zdjęcie w formacie *numpy*,
- zdjeciePoPrzeksztalceniu – zdjęcie po zastosowaniu transformacji w formacie *numpy*,
- (szerokosc, wysokosc) - rozmiar obrazu wyjściowego,
- wspolczynnikSkalujacyX - współczynnik skali wzdłuż osi poziomej
- wspolczynnikskalujacyY - współczynnik skali wzdłuż osi pionowej
- metodaInterpolacji – metoda interpolacji; standardowa wersja **cv2.INTER_LINEAR.**

Przykład skalowania przy definiowaniu rozmiaru (5) oraz współczynników skalujących (6):

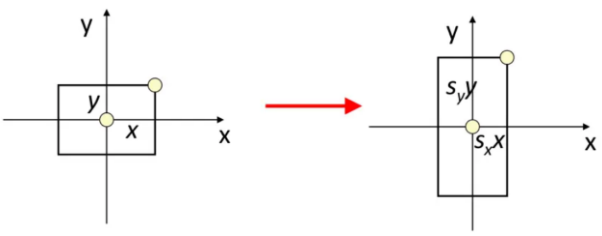
```
wysokosc, szerokosc, _ = zdjęcie.shape
zdjeciePoPrzeksztalceniu = cv2.resize(zdjecie,(szerokość/5, wysokość/5))
```

(5)

```
zdjeciePoPrzeksztalceniu = cv2.resize(zdjęcie,None,fx=0.2,fy=0.2,interpolation = cv2.INTER_CUBIC)
```

(6)

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



Rys. 4 Przykład wykorzystania skalowania obrazu

Ogólny sposób (bez wykorzystywania wbudowanych funkcji OpenCV) skalowania można wykorzystać poprzez zastosowanie macierzy transformacji (Rys. 4):

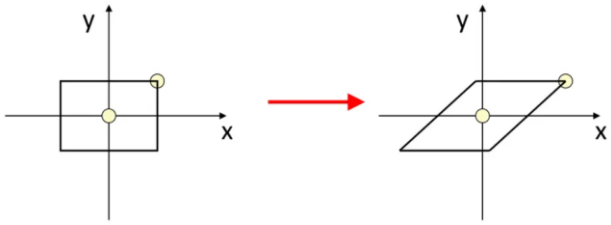
```
import cv2
import numpy as np

zdjęcie = wczytane zdjęcie

(wysokosc, szerokosc, _) = zdjecie.shape
macierzTransformacji = np.float32([[0.2,0,0],                                    (7)
                    [0,0.2,0],
                    [0,0,1]])
zdjeciePoPrzeksztalceniu = cv2.warpPerspective(zdjęcie, macierzTranslacji, (wysokosc, szerokosc))
cv2.imshow('Oryginalne', zdjęcie)
cv2.imshow('Nowy obraz', zdjeciePoPrzeksztalceniu)
cv2.waitKey(0)
```

Transformacja shear mapping

Shear mapping to transformacja (pochylanie obrazu), która przesuwa każdy punkt w ustalonym kierunku, zastępuje każdy punkt w poziomie lub w pionie o określoną wartość proporcjonalnie do jego współrzędnych x lub y:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x + s_x y \\ y \end{bmatrix} = \begin{bmatrix} 1 & s_x \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ s_y x + y \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ s_y & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & s_x & 0 \\ s_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



Rys. 5 Przykład wykorzystania transformacji shear

W celu wykorzystania shear mapping należy nie tylko zastosować macierz transformacji, ale również wyznaczyć wymiary nowego zdjęcia:

```
import cv2
import numpy as np

zdjęcie = wczytane zdjęcie

(wysokosc, szerokosc, _) = zdjecie.shape
macierzTransformacji = np.float32([[1,0.5,0],
                    [0,1,0],                                                     (8)
                    [0,0,1]])

wysokosc = wymiary[0]
szerokość = wymiary[1]
zdjeciePoPrzeksztalceniu = cv2.warpPerspective(zdjęcie, macierzTransformacji, (int(wysokosc), int(szerokosc)))
cv2.imshow('Oryginalne', zdjęcie)
cv2.imshow('Nowy obraz', zdjeciePoPrzeksztalceniu)
cv2.waitKey(0)
```

Odbicie lustrzane

Odbicie obrazu (lub odbicie lustrzane) jest wykorzystywane do odwrócenia obrazu, zarówno w pionie jak i w poziomie. Jest to szczególnym przypadek skalowania -dla odbicia wzdłuż osi x ustawiamy wartość Sy na -1, a Sx na 1, i odwrotnie dla odbicia w osi y.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & \text{wysokosc} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & \text{szerokosc} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Przykład implementacji odbicia lustrzanego został przedstawiony w kodzie nr 9:

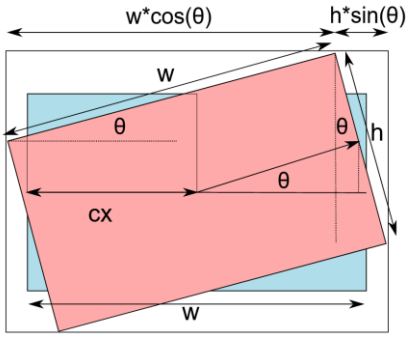
```
import cv2
import numpy as np

zdjęcie = wczytane zdjęcie

(wysokosc, szerokosc, _) = zdjecie.shape
# Odbicie względem osi X
macierzTransformacji = np.float32([[1,0,0],
                    [0,-1, wysokosc],
                    [0,0,1]])
# Odbicie względem osi Y
macierzTransformacji = np.float32([[-1,0, szerokosc],
                    [0,1,0],
                    [0,0,1]])

zdjeciePoPrzeksztalceniu = cv2.warpPerspective(zdjecie, macierzTranslacji, (wysokosc, szerokosc))
cv2.imshow('Oryginalne', zdjęcie)
cv2.imshow('Nowy obraz', zdjeciePoPrzeksztalceniu)
cv2.waitKey(0)
```

(9)

Obrót obrazów

Rotacja to pojęcie w matematyce, które jest ruchem pewnej przestrzeni w oparciu o jeden niezmienny punkt. Obrót obrazu jest powszechnie stosowany w dopasowaniu, wyrównaniu i innych algorytmach opartych na obrazie. Jest również szeroko stosowany w procesie *data augumentation* (rozszerzaniu zbioru uczącego), zwłaszcza jeśli chodzi o klasyfikację obrazu.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



Rys. 6 Przykład wykorzystania obrotu [https://cristianpb.github.io/blog/image-rotation-opencv]

Przykład implementacji obrotu zdjęcia został przedstawiony w kodzie nr 10:

```
import cv2
import numpy as np
```

(10)

```
zdjęcie = wczytane zdjęcie

(wysokosc, szerokosc, _) = zdjecie.shape

# Definiowanie macierzy obrotu
katObrotuWRadianach = np.radians(katWStopniach)
macierzTransformacji = np.float32([[np.cos(katObrotuWRadianach),-np.sin(katObrotuWRadianach), 0],
                    [np.sin(katObrotuWRadianach), np.cos(katObrotuWRadianach), 0],
                    [0,0,1]])

zdjeciePoPrzeksztalceniu = cv2.warpPerspective(zdjecie, macierzTranslacji, (wysokosc, szerokosc))
cv2.imshow('Oryginalne', zdjęcie)
cv2.imshow('Nowy obraz', zdjeciePoPrzeksztalceniu)
cv2.waitKey(0)
```

W bibliotece OpenCV zaimplementowana jest funkcja do wyznaczania macierzy obrotów. W przeciwieństwie, do klasycznego podejścia do wyznaczania macierzy obrotu, dodatkowo jest uwzględniana zmiana położenia osi obrotu:

$$\text{macierzTransformacji} = \begin{matrix} a & b & (1-a)*centr.x - b*centr.y \\ -b & a & b*centr.x + (1-a)*centr.y \end{matrix}$$

$$\text{macierzTransformacji} = \begin{bmatrix} a & b & (1-a)*centr.x - b*centr.y \\ -b & a & b*centr.x + (1-a)*centr.y \\ 0 & 0 & 1 \end{bmatrix}$$

gdzie:

- centr - środek obrotu
- a = skala * cos(k)
- b = skala * sin(k)

W celu szybkiego wyznaczenia elementów macierzy, stosuje się funkcję cv2.getRotationMatrix2D((centr.x, centr.y), kąt, skala), która tworzy macierz, przekazywaną następnie do cv2.warpAffine().

Do zapisu pełnego obrazu transformacji poprzez obrót należy na nowo wyznaczyć wielkości zdjęcia według kodu nr 12:

```
import cv2
import numpy as np

zdjecie = wczytane zdjęcie

(wysokosc, szerokosc, _) = zdjecie.shape

# Wyznaczenie środka zdjęcia
# of an image
centrY, centrX = wysokosc/2, szerokosc /2

katObrotuWRadianach = np.radians(katWStopniach)                                    (12)

# Wyznaczenie macierzy rotacji
macierzTransformacji = cv.getRotationMatrix2D((centrY, centrX), katObrotuWRadianach, 1.0)

# Wyznaczenie wartości bezwzględnych funkcji sin I cos
cosMacierzObrotu = np.abs(macierzTransformacji [0][0])
sinMacierzObrotu = np.abs(macierzTransformacji [0][1])

# Wyznaczenie nowych wymiarów zdjęcia celem wykorzystania wszystkich piskeli
nowaWysokosc = int((wysokosc * sinMacierzObrotu) +
                    (szerokosc * cosMacierzObrotu))
```

```
nowaSzerokosc = int((wysokosc * cosMacierzObrotu) +
                    (szerokosc * sinMacierzObrotu))

# Aktualizacja macierzy transformacji
macierzTransformacji [0][2] += (nowaSzerokosc/2) - centrX
macierzTransformacji [1][2] += (nowaWysokosc/2) - centrY

# Zastosowanie transformacji obrazu
zdjeciePoPrzeksztalceniu = cv.warpAffine(
   zdjecie, macierzTransformacji, (nowaSzerokosc, nowaWysokosc))
```

Transformacja afiniczna

Transformacja afiniczna - zachowująca równoległość linii równoległych na obrazie pierwotnym, potrzebuje danego położenia trzech punktów na obrazie wejściowym i odpowiadających im trzech w układzie obrazu wyjściowego.

Parametry transformacji 2x3 uzyskiwane są za pośrednictwem funkcji: **cv2.getAffineTransform(zestaw_wejściowy, zestaw_wyjściowy**), gdzie zestawy oznaczają zbiory punktów, wskazane jako listy list (macierze) współrzędnych. Funkcja ta tworzy macierz, przekazywaną **do funkcji cv2.warpAffine().**

```
import cv2
import numpy as np

zdjęcie = wczytane zdjęcie

(wysokosc, szerokosc, _) = zdjecie.shape

punkty1 = np.float32([[0,0],[100,0],[0,100]])
punkty2 = np.float32([[10,10],[150,20],[30,170]])
macierzTransformacji = cv2.getAffineTransform(punkty1,punkty2)
zdjeciePoPrzeksztalceniu = cv2.warpAffine(zdjecie, macierzTransformacji,(szerokosc, wysokosc))
```

(12)

Transformacja perspektywiczna

Transformacja perspektywiczna jest wykonywana na podstawie czterech odpowiadających sobie punktów, z których żadne trzy nie mogą być współliniowe. Do przekształcenia potrzebne jest wyznaczenie macierzy 3x3, którą tworzy funkcja: **cv2.getPerspectiveTransform(zestaw_wejściowy, zestaw_wyjściowy),** gdzie zestawy oznaczają zbiory punktów, wskazane jako listy list (macierze) współrzędnych.

Funkcja ta tworzy macierz, przekazywaną do funkcji **cv2.warpPerspective(obraz, Macierz, (kolumny, wiersze))**, która tworzy nowy, przekształcony obraz.

```
import cv2
import numpy as np

zdjęcie = wczytane zdjęcie

(wysokosc, szerokosc, _) = zdjecie.shape

punkty1 = np.float32([[0,0],[100,0],[0,100],[100,100]])
punkty2 = np.float32([[10,10],[150,20],[30,170],[130,190]])
macierzTransformacji = cv2.getPerspectiveTransform(punkty1,punkty2)
```

(12)

```
zdjeciePoPrzeksztalceniu = cv2.warpPerspective(zdjecie, macierzTransformacji,(600,700)) #tym razem bez wymiarów obrazu wejściowego
```