Zaawansowne algorytmy wizyjne

materiały do ćwiczeń laboratoryjnych

Piotr Pawlik, Tomasz Kryjak

PUBLISHED BY AGH

First printing, March 2018

Contents

1 — Kalibracja kamery i stereowizja

1.1 Kalibracja pojedynczej kamery

Kalibracja kamery ma na celu eliminację zniekształceń (dystorsji) wprowadzanych przez układ optyczny. Zalicza się do nich zniekształcenia: radialne (beczkowe, poduszkowe) oraz tangensoidalne. Szersze omówienie – Distortion_(optics).

Po pierwsze przekonajmy się na czym polegają te zniekształcenia (choć pewnie każdy z nas je widział w przypadku kamer sportowych z bardzo szerokim kątem obiektywu). Otwórz w programie Gimp dowolny obraz z katalogu *images_left* dostępny w archiwum na stronie kursu. Użyj narzędzia *Miarka* i wyrysuj prostą łącząca dwa narożniki szachownicy (najlepiej daleko od środka obrazu). Efekt zniekształceń powinien być łatwy do zaobserwowania.

Korekcja zniekształceń wymaga wyliczenia pewnych współczynników. Jest to możliwe w procedurze kalibracji, gdzie rejestruje się obraz wzorca o znanych parametrach (wzajemne odległości między punktami i ich rozmiary) w różnych położeniach względem kamery. Najczęściej stosuje się wzorzec w postaci szachownicy, na którym łatwo wykryć narożniki (tak jak na analizowanym obrazie). Praktyka wskazuje, że obrazów powinno być co najmniej 10.

Poniższy kod z komentarzami pozwala wykryć i zapisać wymagane punkty.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

#  kryternia przetwania obliczen (blad+liczba iteracji)
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

# przygotowanie punktow 2D w postaci: (0,0,0), (1,0,0), (2,0,0) ....,(6,7,0)
objp = np.zeros((6*7,3), np.float32)
objp[:,:2] = np.mgrid[0:7,0:6].T.reshape(-1,2)

# tablice do przechowywania punktow obiektow (3D) i punktow na obrazie (2D)
    dla wszystkich obrazow
objpoints = [] # punkty 3d w przestrzeni (rzeczywsite)
imgpoints = [] # punkty 2d w plaszczyznie obrazu.
```

```
for i in range(1,13)::
  # wczytanie obrazu
  img = cv2.imread('images_left/left%02d.jpg' % i)
  # konwersja do odcieni szarosci
  gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

  # wyszukiwanie naroznikow na planszy
  ret, corners = cv2.findChessboardCorners(gray, (7,6), None)

  # jesli znaleniono na obrazie punkty
  if ret == True:
    #dolaczenie wspolrzednych 3D
    objpoints.append(objp)

    # poprawa lokalizacji punktow (podpiskelowo)
    corners2 = cv2.cornerSubPix(gray,corners, (11,11), (-1,-1), criteria)
    # dolaczenie poprawionych punktow
    imgpoints.append(corners2)

    # wizualizacja wykrytych naroznikow
    cv2.drawChessboardCorners(img, (7,6), corners2, ret)
    cv2.imshow("Corners",img)
  cv2.waitKey(0)
```

Mając wyliczone współrzędne punktów można przystąpić do kalibracji:

```
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints,
    gray.shape[::-1], None, None)
```

Uwagi:

- trzeci parametr `gray.shape[::-1]` to rozmiar obrazka,
- `ret` – wartość błędów średniokwadratowego projekcji wstecznej (opisuje jakość dopasowania),
- `mtx` – macierz parametrów wewnętrznych kamery w postaci:

$$A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{1.1}$$

  gdzie: (f_x, f_y) – to długości ogniskowych, a (c_x, c_y) to współrzędne środka obrazu.
- `dist` – współczynniki zniekształceń,
- `rvecs` – wektory rotacji,
- `tvecs` – wektory translacji (wraz z wektorami rotacji umożliwiają przekształcenie położenia wzorca kalibracyjnego z współrzędnych modelu do współrzędnych rzeczywistych)

Proszę dodać funkcję do kodu i wyświetlić zwracane przez nią parametry.

Mając wyliczone parametry, możemy przystąpić do korekcji obrazu. Zanim jednak tego dokonamy należy poprawić wyznaczoną macierz kamery.

Służy do tego funkcja:

```
newcameramtx, roi=cv2.getOptimalNewCameraMatrix(mtx, dist, (w,h), 1, (w,h))
```

gdzie: (w,h) – to szerokość i wysokość obrazu (`h, w = img.shape[:2]`), a 1 to parametr alfa (przyjmuje on wartości z zakresu 0-1. 0 oznacza, że wszystkie piksele w nowym obrazie są poprawne, a 1, że wszystkie ze starego zachowane /taka sama rozdzielczość/).

Samą korekcję można przeprowadzić na dwa sposoby (z takim samym rezultatem końcowym):

```
dst = cv2.undistort(img, mtx, dist, None, newcameramtx)
```

albo:

```
mapx, mapy = cv2.initUndistortRectifyMap(mtx, dist, None, newcameramtx, (w,h
    ), 5)
dst = cv2.remap(img, mapx, mapy, cv.INTER_LINEAR)
```

Ostatnim krokiem jest wycięcie ROI (obraz po korekcji jest większy niż oryginalny). Można wykorzystać taki kod:

```
x, y, w, h = roi
dst = dst[y:y+h, x:x+w]
cv2.imwrite('calibresult.png', dst)
```

Do wykonania:

- sprawdzić korekcję dla jednego z obrazów ze zbioru (test na linię prostą),
- sprawdzić jak działa parametr alfa.

Warto wiedzieć, że istnieje także alternatywny wzorzec kalibracyjny – macierz kółek i funkcja w OpenCV do detekcji ich środków (`findCirclesGrid`). Przykład na rysunk 1.1

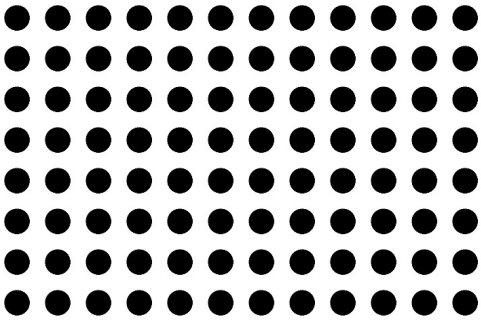

Figure 1.1: Wzorzec kalibracyjny – macierz kółek.

1.2 Kalibracja układu kamer

W największym uproszczeniu, kalibracja układu kamer (dla uproszczenia dwóch) ma umożliwić łatwe obliczenie map głębi. Zagadnienie to można rozumieć na różne sposoby: ustawienie identycznych parametrów obu kamer, eliminację zniekształceń, sprowadzenie do „wspólnego" układu współrzędnych oraz rektyfikację. Szczególnie interesujące jest ostatnie zagadnienie, które polega na takim przekształceniu obrazów, aby na obu te same piksele leżały na tych samych liniach (tzw. liniach epipolarnych). Znakomicie ułatwia to wyznaczenie korespondencji (mapy głębi) – trzeba prowadzić poszukiwania tylko w jeden płaszczyźnie.

Początkowe operacje przeprowadza się podobnie jak dla pojedynczej kamery. Najlepiej przekopiować stworzony kod i dodać „obsługę" drugiego obrazu.

W efekcie powinniśmy uzyskać rezultaty kalibracji (wyjście z `calibrateCamera`) dla kamery lewej i prawej.

Następnie wykonujemy trzy funkcje:

```
retval, cameraMatrix1, distCoeffs1, cameraMatrix2, distCoeffs2, R, T, E, F =
      cv2.stereoCalibrate(objpoints,imgpoints_L,imgpoints_R,mtx_L,dist_L,
    mtx_R,dist_R,gray_l.shape[::-1])
```

```
R1, R2, P1, P2, Q, validPixROI1, validPixROI2 = cv2.stereoRectify(
    cameraMatrix1, distCoeffs1, cameraMatrix2, distCoeffs2, gray_r.shape
    [::-1], R, T)

map1_L, map2_L = cv2.initUndistortRectifyMap(cameraMatrix1, distCoeffs1, R1,
     P1, gray_l.shape[::-1], cv2.CV_16SC2)
map1_R, map2_R = cv2.initUndistortRectifyMap(cameraMatrix2, distCoeffs2, R2,
     P2, gray_l.shape[::-1], cv2.CV_16SC2)
```

Odpowiednio: kalibracja stereo, rektyfikacja oraz obliczenie współczynników mapowania dla funkcji `repmap` (była pokazana wcześniej).

Na końcu pozostaje nam wczytać dwa obrazy ze zbioru (dowolne) i wykonać mapowanie:

```
dst_L = cv2.remap(img_l, map1_L, map2_L, cv2.INTER_LINEAR)
dst_R = cv2.remap(img_r, map1_R, map2_R, cv2.INTER_LINEAR)
```

Aby sprawdzić działanie rektyfikacji warto zestawić oba obrazy obok siebie i wyrysować poziome linie, co pozwoli sprawdzić, czy faktycznie te same piksele leżą na tych samych liniach. Można to np. zrobić tak:

```
N, XX, YY = dst_L.shape[::-1] # pobranie rozmiarow obrazka (kolorowego)

visRectify = np.zeros((YY,XX*2,N),np.uint8) # utworzenie nowego obrazka o
    szerokosci x2
visRectify[:,0:640:,:] = dst_L      # przypisanie obrazka lewego
visRectify[:,640:1280:,:] = dst_R   # przypisanie obrazka prawego

# Wyrysowanie poziomych linii
for y in range(0,480,10):
        cv2.line(visRectify, (0,y), (1280,y), (255,0,0))

cv2.imshow('visRectify',visRectify)  #wizualizacja
```

Warto wiedzieć, że w OpenCV dostępna jest również funkcja rektyfikacji obrazów nieskalibrowanych `stereoRectifyUncalibrated`. Uruchomienie dla chętnych.

1.3 Obliczenia korespondencji stereo

Samo obliczenie korespondencji stereo (mapy głębi, mapy dysparcyji) jest stosunkowo proste (przynajmniej w teorii i dla odpowiednich zdjęć). Celem jest znalezienie o ile piksel $I_L(x,y)$ jest przesunięty na drugim obrazie $I_R(x+d,y)$. Dla przypomnienia – szukamy tylko w liniach poziomych, bo zakładamy, że obrazy poddane zostały rektyfikacji. Można też zauważyć, że zadanie w pewnym sensie jest podobne do tego, przy obliczaniu przepływu optycznego. Tylko tam mieliśmy dwie kolejne ramki z sekwencji zarejestrowane jedną kamerą, a tu dwa obrazy z dwóch kamer. Nota bene, warto wiedzieć, że na podstawie sekwencji z pojedynczej kamery też można odtworzyć głębię – o ile kamera ta jest ruchoma – zagadnienie *structure from motion*.

W OpenCV dostępne są dwie metody obliczenia map głębi: dopasowanie bloków (*Block Matching*) i SGM (*Semi-Global Matching*). Działanie pierwszej jest dość oczywiste, w przypadku drugiej dokonuje się również analizy globalnej mapy (w uproszczonej formie), co pozwala poprawić ciągłość mapy.

Proszę samodzielnie, na podstawie dokumentacji oraz ew. przykładów wyznaczyć mapę dyspracycji dla obrazów *aloe*. Pochodzi one ze zbiory Middlebury. Są tam dostępne mapy referencyjne (*ground truth*). Można porównać metodę z mapa np. wskaźnikiem *bad2.0* (procent pikseli o błędzie dysprycji większym niż dwa) lub *rms* (*root mean square*).

1.3.1 Zadanie dla dociekliwych (nieobowiązkowe)

Prostą, ale dość dobrą metodą jest tzw. transformata Censusa. Bierzemy blok NxN np. $5x5$. Binaryzujemy go z progiem w postaci wartości środkowej z kontekstu. Otrzymujemy ciąg binarny. Na drugim obrazie robimy to samo, ale w d położeniach (od 0 do $d_max - 1$). Porównanie dwóch kontekstów to operacja $xor(C1, C2)$. Zliczmy liczbę różnic i szukamy minimum – to będzie nasza wartość d. Proszę zaimplementować opisaną metodą i porównać działanie z BM i ew. SGM (przynajmniej wizualnie).

Bibliography