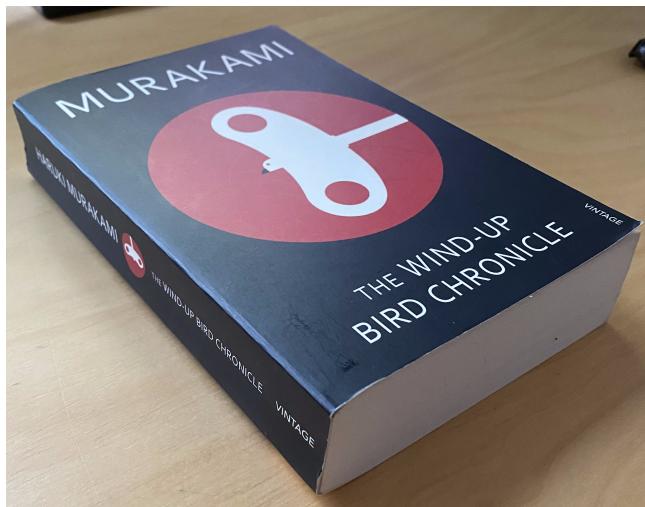


Sprawozdanie 3 - Estymacja pozy obiektu w oparciu o PnP

Celem laboratorium jest zapoznanie z algorytmem Perspective-n-Point (PnP) do estymacji pozy obiektu.

W pierwszej części sprawozdania przedstawię estymację pozy za pomocą algorytmu EPnP (ulepszoną wersją algorytmu PnP).

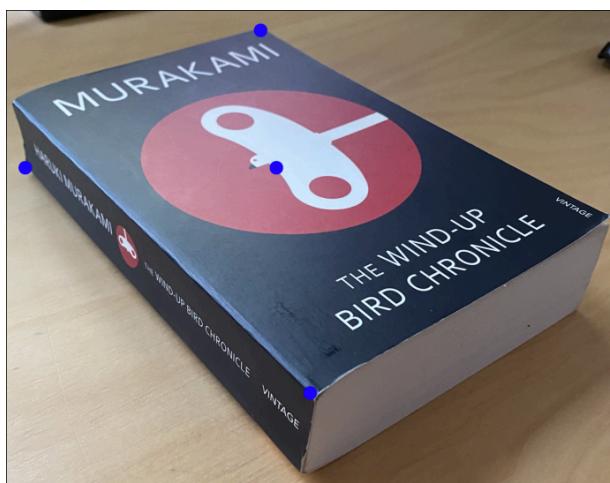
W tym celu wykorzystam zdjęcie książki:



Poniższa część skryptu nanosi kluczowe punkty na książkę:

```
points_2d = np.array([(1200, 700), (1130, 90), (1350, 1700), (85, 700)])
for p in points_2d:
    cv2.circle(im, (int(p[0]), int(p[1])), 30, (255, 0, 0), -1)
plt.imshow(cv2.cvtColor(im, cv2.COLOR_BGR2RGB))
plt.show()
```

I zdjęcie wygląda następująco:



Poniższa część skryptu służy do ustawienia pozycji kamery:

```
focal_length = image_size[1]
center = (image_size[1]/2, image_size[0]/2)
camera_matrix = np.array([
    [focal_length, 0, center[0]],
    [0, focal_length, center[1]],
    [0, 0, 1]]).astype(float)
```

Poniższa część służy do wyznaczenia pozy:

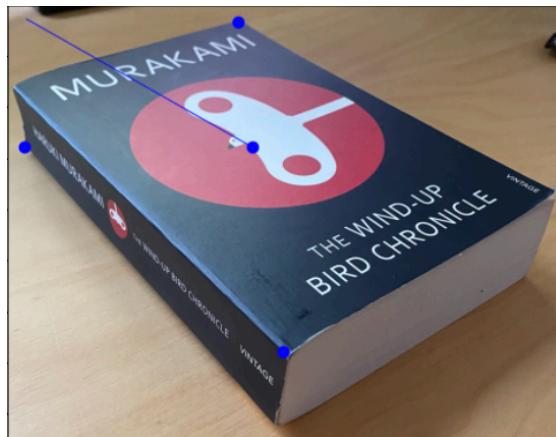
```
distortion_coefficients = np.zeros((4,1))
_, rotation_vector, translation_vector = cv2.solvePnP(points_3d,
                                                       points_2d,
                                                       camera_matrix,
                                                       distortion_coefficients,
                                                       flags=cv2.SOLVEPNP_EPNP)

def draw_axis(rotation_vector, translation_vector, camera_matrix, distortion_coefficients, points_2d):
    end, _ = cv2.projectPoints(np.array([(0.0, 0.0, 200.0)]), rotation_vector,
                               translation_vector,
                               camera_matrix,
                               distortion_coefficients)

    for point in points_2d:
        cv2.circle(image, (int(point[0]), int(point[1])), 5, (0,0,255), 1)
    point1, point2 = (int(points_2d[0][0]), int(points_2d[0][1])), (int(end[0][0][0]), int(end[0][0][1]))
    cv2.line(image, point1, point2, (255,0,0), 3)
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))

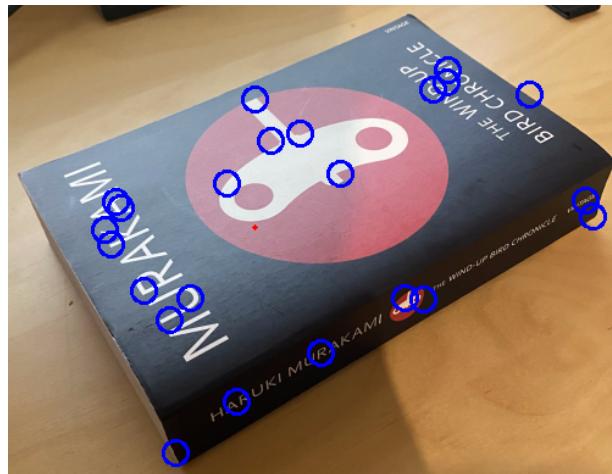
draw_axis(rotation_vector, translation_vector, camera_matrix, distortion_coefficients, points_2d)
plt.show()
```

Efekt jest następujący:

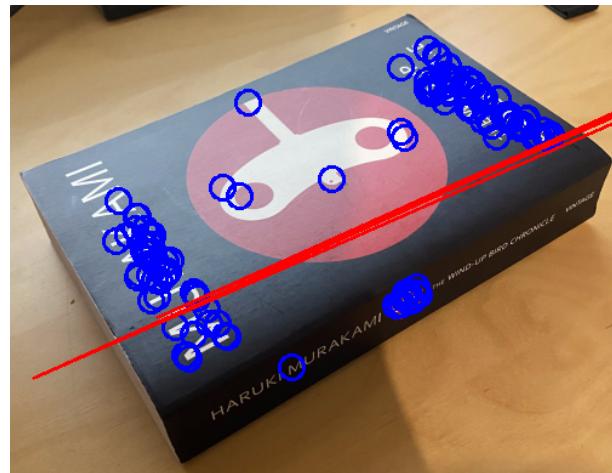


W tej części zajmę się wyznaczaniem pozy obiektu metodami SIFT i ORB. Wykorzystam do tego poprawiony załączony kod. Na początku usuwam elementy występujące w standardzie Python wyższym niż 3.5 (m.in. usuwam odwołania do form __future__ import annotations oraz typing z funkcji w klasie Camera).

Metoda SIFT:



Metoda ORB:



Jak widać obie metody poradziły sobie inaczej (wyznaczony został inny kierunek pozy).

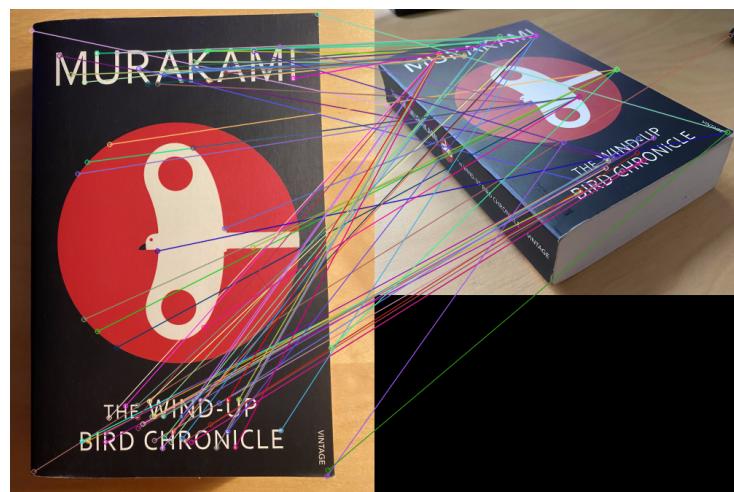
W kolejnej części zajmę się strategiami wyznaczania korespondencji w oparciu o keypoints. Wykorzystam do tego również poprawiony załączony skrypt `estimate_cube_pose.py`.

Teraz zajmę się estymacją pozy metodą SIFT. W tym celu przygotowałem dwie pary zdjęć książki.

Pierwszym badanym podejściem będzie SIFT + Brute-Force Matching. Wykorzystam funkcję BRMatcher.knnMatch(), aby otrzymać k najlepszych dopasowań. Dla k=2 efekt jest następujący:



Dla innej pary zdjęć efekt jest następujący:

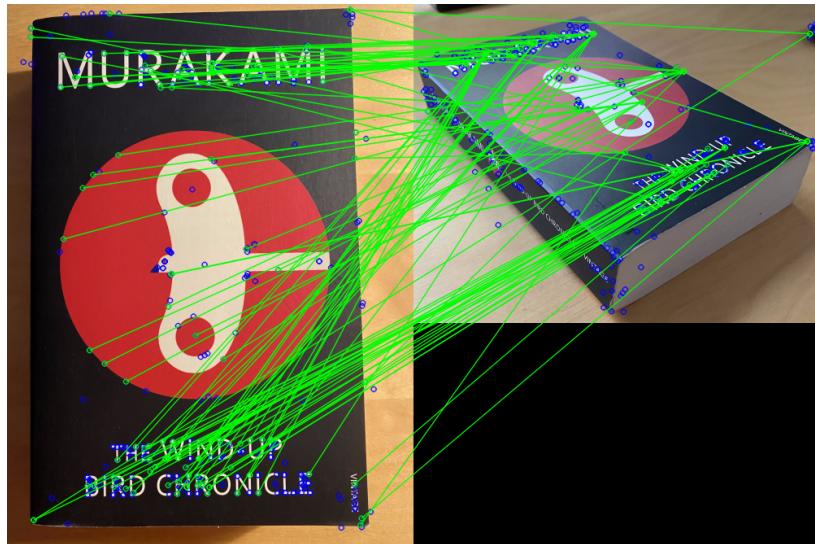


Jak widać w przypadku obu par zdjęć algorytm poradził sobie dość dobrze, jednak da się znaleźć pusty błędnie przyporządkowane.

Kolejną badaną strategią będzie SIFT + Fast Library for Approximate Nearest Neighbors. Dla pierwszej pary zdjęć efekt jest następujący:

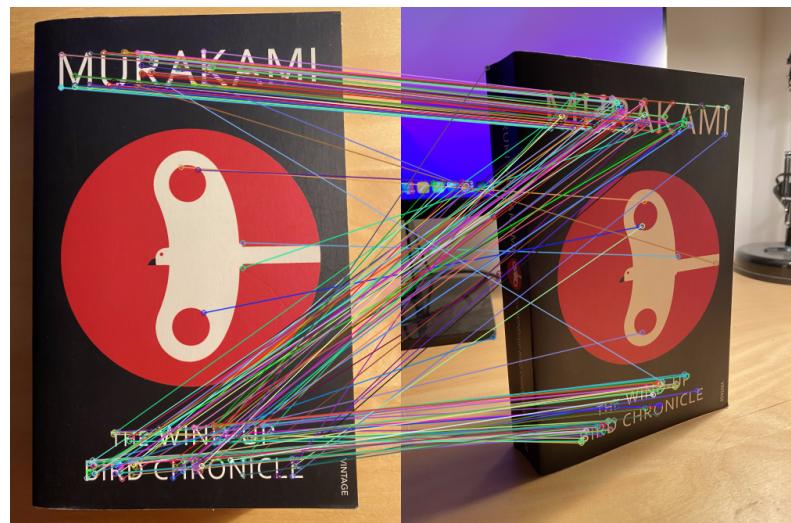


I dla drugiej pary zdjęć:

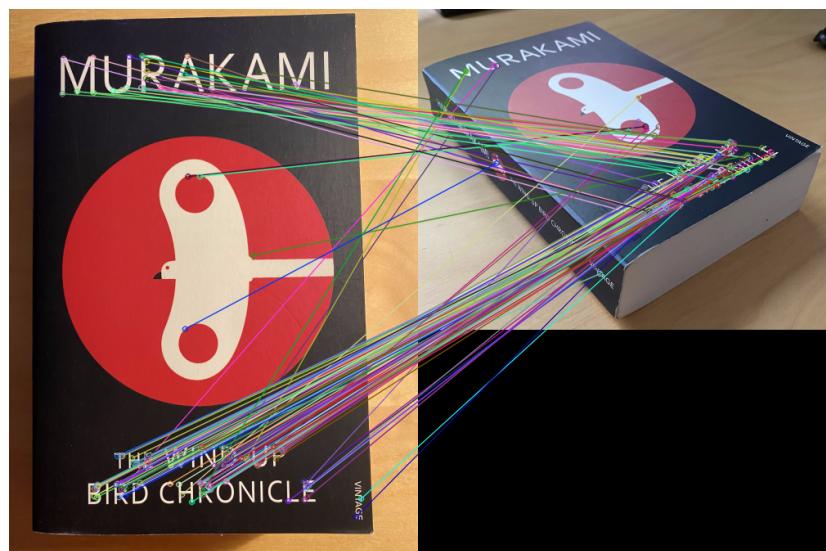


Tutaj widać również algorytm poradził sobie dość dobrze.

Kolejną metodą będzie ORB + Brute-Force Matching. Dla pierwszej pary zdjęć efekt jest następujący:



Dla drugiej pary zdjęć wygląda to następująco:



Dla pierwszej pary zdjęć algorytm poradził sobie dość dobrze, dla drugiej znacznie gorzej (cały górnny napis został źle przyporządkowany).