To use the functionality of working with transformation, filtering and image corrections, you need to connect the file «combined\_imgproc.py».

**Image transformation**

The following functions are designed to obtain transformation matrices between two images using a set of key points. The resulting matrix can be used to combine images into a panorama, estimate camera position, remove/correct image perspective, etc. To use the following functions, you need to obtain a set of key points on a pair of images using opencv:

orb = cv2.ORB\_create(nfeatures=500)  
# Find keypoints and descriptors  
kp1\_cv, des1\_cv = orb.detectAndCompute(image1\_bgr, None)  
kp2\_cv, des2\_cv = orb.detectAndCompute(image2\_bgr\_transformed, None)

Compare the obtained points using opencv:

bf = cv2.BFMatcher(cv2.NORM\_HAMMING, crossCheck=False) # Use NORM\_HAMMING for ORB  
knn\_matches\_cv = bf.knnMatch(des1\_cv, des2\_cv, k=2)

Then transform into two lists (matched keypoints of the source and target images) of the Point class as follows:

src\_pts\_for\_transform = [ips.Point.from\_cv\_keypoint(kp1\_cv[m.queryIdx]) for m in good\_matches\_cv]  
dst\_pts\_for\_transform = [ips.Point.from\_cv\_keypoint(kp2\_cv[m.trainIdx]) for m in good\_matches\_cv]

**Homography matrix calculation function**

It is designed to find the homography matrix between two images. The homography matrix is ​​a 3×3 matrix that describes the projective transformation between two planes. The calculation requires at least 4 pairs of key points. With its help you can: 1) rotate; 2) scale; 3) shift; 4) shear; 5) perspective distortion.

To call the function, it is enough to pass key points on the source and target images, as well as the method for filtering intermediate results:

H\_custom\_ransac = ips.find\_homography\_py\_custom\_ransac(  
 src\_pts\_for\_transform, dst\_pts\_for\_transform,  
 method\_filter="RANSAC)

The result is the final homography matrix between the two images, which can be applied to the image and output as follows:

img1\_warped\_cv\_h = cv2.warpPerspective(image1\_bgr, H\_cv\_ransac,  
 (image1\_bgr.shape[1], image1\_bgr.shape[0]))  
cv2.imshow("Image1 Warped (CV RANSAC H)", img1\_warped\_cv\_h)

**Function for calculating the matrix of affine transformations**

Designed to find the matrix of affine transformations between two images. An affine matrix is ​​a 2x3 matrix that describes an affine transformation, i.e. a transformation of a plane (or space) that preserves lines and parallelism, but can change lengths and angles. A minimum of 3 pairs of key points are required for the calculation. With its help you can: 1) rotate; 2) scale; 3) shift; 4) shear;

To call the function, it is enough to pass key points on the source and target images, a method for filtering intermediate results:

A\_custom\_ransac = ips.estimate\_affine\_partial2d\_py\_custom\_ransac(  
 src\_pts\_for\_transform, dst\_pts\_for\_transform,  
 method\_filter="RANSAC)

The result is a final matrix of affine transformations between the two images, which can be applied to the image and output as follows:

img1\_warped\_custom\_a = cv2.warpAffine(image1\_bgr, A\_custom\_ransac,  
 (image1\_bgr.shape[1], image1\_bgr.shape[0]))  
cv2.imshow("Image1 Warped (Custom RANSAC A)", img1\_warped\_custom\_a)

**Color correction of images**

The next group of functions allows you to adjust the contrast of images in order to equalize the overall brightness.

**Histogram correction function**

This method performs histogram correction. The algorithm is based on the transformation of the brightness distribution function (histogram) so that the resulting cumulative distribution approaches uniformity. This allows for an expansion of the dynamic range of intensities in the image and an increase in contrast, especially in areas of weakly expressed gradations, while maintaining the overall structure of the image.

To call the function, it is enough to pass the original image:

hist\_eq\_result\_custom = ips.apply\_histogram\_py(image1\_bgr.copy())

The output returns the image after correction. To evaluate the result, you can display the resulting image:

cv2.imshow("Custom Global HistEq", hist\_eq\_result\_custom)

**CLAHE correction function**

Improved histogram equalization option, thanks to contrast gain limitation and adaptive block-by-block image processing.

To call, you need to pass the original image, the size of one correction block and the relative limitation on the height (maximum value) of the histogram in each block:

clahe\_result\_custom = ips.apply\_histogram\_clahe\_py(image1\_bgr.copy(), tile\_size\_x=80, tile\_size\_y=80, relative\_clip\_limit=3.0)

The output returns the image after correction. To evaluate the result, you can display the resulting image:

cv2.imshow("Custom CLAHE", clahe\_result\_custom)