МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА

Звіт

до лабораторної роботи №3

з предмету Комп’ютерне бачення та аналіз зображень

Роботу виконала:

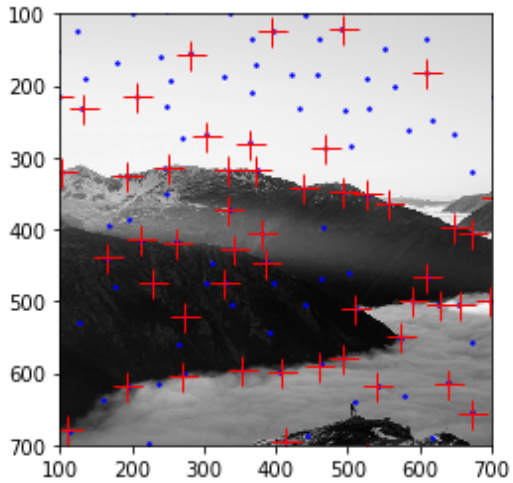
Мерцало Ірина Ігорівна,

студентка групи ПМІМ-11

Львів – 2022

Відкрила зображення. Виконала визначення кутів Хаара (знаходить всі ребра на зображенні, а потім попарно перевіряє, чи перетинаються краї):

```
image = imread('mountains.jpg')
image = rgb2gray(image)
#Compute the Harris corners in the image. This returns a corner measure respons
corners = corner_harris(image)
#Using the corner response image we calculate the actual corners in the
image
coords = corner_peaks(corners, min_distance=25)
# This function decides if the corner point is an edge point or an isolated pea
coords_subpix = corner_subpix(image, coords, window_size=13)
fig, ax = plt.subplots()
ax.imshow(image, interpolation='nearest', cmap=plt.cm.gray)
ax.plot(coords[:, 1], coords[:, 0], '.b', markersize=3)
ax.plot(coords_subpix[:, 1], coords_subpix[:, 0], '+r', markersize=15)
ax.axis((100, 700, 700, 100))
plt.show()
```



Застосувала каскад локальних бінарних шаблонів (для кожного пікселя зображення створюється восьмирозрядний двійковий вектор ознак враховуючи вісім сусідніх пікселів (верхній лівий, верхній правий, лівий, правий, нижній лівий та нижній правий). Для кожного сусіднього пікселя є відповідний біт, якому призначається а значення 1, якщо значення пікселя більше, ніж значення центрального пікселя, інакше воно дорівнює 0. Восьмибітовий вектор ознак розглядається як двійкове число (пізніше перетворює його на десяткове значення) і використовуючи десяткові значення для кожного пікселя, обчислюється гістограма на 256 бінів. Ця гістограма використовується як представлення зображення.) :

In [34]:
```
from skimage.transform import rotate
from skimage.feature import local_binary_pattern
from skimage import data
from skimage.color import label2rgb
import numpy as np

# Get three different images to test the algorithm with
brick = data.brick()
grass = data.grass()
wall = data.clock()

# Calculate the LBP features for all the three images
brick_lbp = local_binary_pattern(brick, 16, 2, 'uniform')
grass_lbp = local_binary_pattern(grass, 16, 2, 'uniform')
wall_lbp = local_binary_pattern(wall, 16, 2, 'uniform')

# Next we will augment these images by rotating the images by 22 degrees
brick_rot = rotate(brick, angle = 22, resize = False)
grass_rot = rotate(grass, angle = 22, resize = False)
wall_rot = rotate(wall, angle = 22, resize = False)

# Let us calculate the LBP features for all the rotated images
brick_rot_lbp = local_binary_pattern(brick_rot, 16, 2, 'uniform')
grass_rot_lbp = local_binary_pattern(grass_rot, 16, 2, 'uniform')
wall_rot_lbp = local_binary_pattern(wall_rot, 16, 2, 'uniform')
```

```
# We will pick any one image say brick image and try to find
# its best match among the rotated images
# Create a list with LBP features of all three images
bins_num = int(brick_lbp.max() + 1)
brick_hist = np.histogram(brick_lbp, normed=True, bins=bins_num, range=(0,
bins_num))
lbp_features = [brick_rot_lbp, grass_rot_lbp, wall_rot_lbp]

min_score = 1000 # Set a very large best score value initially
winner=0
idx = 0 # To keep track of the winner

for feature in lbp_features:
    histogram, _ = np.histogram(feature, normed=True, bins=bins_num, range=(0,b
    p = np.asarray(brick_hist)[0]
    q = np.asarray(histogram)
    filter_idx = np.logical_and(p != 0, q != 0)
    score = np.sum(p[filter_idx] * np.log2(p[filter_idx] / q[filter_idx]))
    if score < min_score:
        min_score = score
        winner = idx
    idx = idx + 1

if winner == 0:
    print('Brick matched with Brick Rotated')
elif winner == 1:
    print('Brick matched with Grass Rotated')
elif winner == 2:
    print('Brick matched with Wall Rotated')
```

```
Brick matched with Brick Rotated
```

Навчання та тестування відбулося швидше з каскадами LBP, а отже, цей метод є кращим при розробці вбудованих програм. Порівняно з каскадами Хаара,

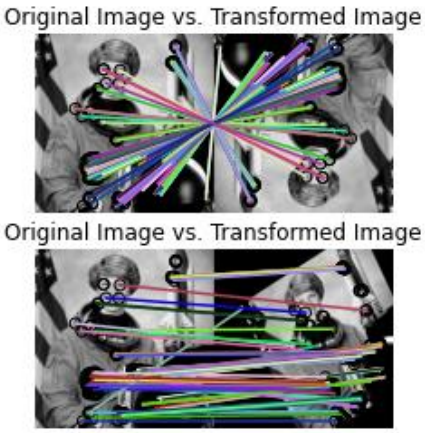
каскади LBP мають справу з цілими числами, а не з подвійними значеннями, тому що ми просто встановлюємо значення або 0, або 1.

Застосувала ORB (який спирається на FAST детектор ключових точок і дескриптор BRIEF):

In [36]:
```
from skimage import data
from skimage import transform as tf
from skimage.feature import (match_descriptors, corner_harris,
 corner_peaks, ORB, plot_matches)
from skimage.color import rgb2gray
import matplotlib.pyplot as plt

#Read the original image
image_org = data.astronaut()
#Convert the image gray scale
image_org = rgb2gray(image_org)
#We prepare another image by rotating it. Only to demonstrate feature mathcing
image_rot = tf.rotate(image_org, 180)
#We create another image by applying affine transform on the image
tform = tf.AffineTransform(scale=(1.3, 1.1), rotation=0.5,
 translation=(0, -200))
image_aff = tf.warp(image_org, tform)
#We initialize ORB feature descriptor
descriptor_extractor = ORB(n_keypoints=200)
#We first extract features from the original image
descriptor_extractor.detect_and_extract(image_org)
keypoints_org = descriptor_extractor.keypoints
descriptors_org = descriptor_extractor.descriptors
descriptor_extractor.detect_and_extract(image_rot)
keypoints_rot = descriptor_extractor.keypoints
descriptors_rot = descriptor_extractor.descriptors
descriptor_extractor.detect_and_extract(image_aff)
keypoints_aff = descriptor_extractor.keypoints
descriptors_aff = descriptor_extractor.descriptors
```

```
matches_org_rot = match_descriptors(descriptors_org, descriptors_rot,
cross_check=True)
matches_org_aff = match_descriptors(descriptors_org, descriptors_aff,
cross_check=True)
fig, ax = plt.subplots(nrows=2, ncols=1)
plt.gray()
plot_matches(ax[0], image_org, image_rot, keypoints_org, keypoints_rot,
matches_org_rot)
ax[0].axis('off')
ax[0].set_title("Original Image vs. Transformed Image")
plot_matches(ax[1], image_org, image_aff, keypoints_org, keypoints_aff,
matches_org_aff)
ax[1].axis('off')
ax[1].set_title("Original Image vs. Transformed Image")
plt.show()
```



У алгоритмі ORB відбувається додавання швидкого та точного компонента орієнтації до FAST, ефективне обчислення орієнтованих BRIEF функцій, аналіз дисперсій та кореляції орієнтованих BRIEF ознак, метод навчання для декореляції BRIEF об'єктів за ротаційної інваріантності, що призводить до кращої продуктивності в програмах.

Виконала накладання (з’єднання) зображень (знято кілька зображень, що перекриваються, об’єднуються разом загальні частини зображень) :


mountains-1.png


mountains-2.png


output.png

In [47]:
```
from skimage.feature import ORB, match_descriptors
from skimage.io import imread
from skimage.measure import ransac
from skimage.transform import ProjectiveTransform
from skimage.color import rgb2gray
from skimage.io import imsave, show
from skimage.color import gray2rgb
from skimage.exposure import rescale_intensity
from skimage.transform import warp
from skimage.transform import SimilarityTransform
import numpy as np

image0 = imread('mountains-1.png')
image0 = rgb2gray(image0)
image1 = imread('mountains-2.png')
image1 = rgb2gray(image1)
orb = ORB(n_keypoints=1000, fast_threshold=0.05)
orb.detect_and_extract(image0)
keypoints1 = orb.keypoints
descriptors1 = orb.descriptors
orb.detect_and_extract(image1)
keypoints2 = orb.keypoints
descriptors2 = orb.descriptors
matches12 = match_descriptors(descriptors1, descriptors2, cross_check=True)
src = keypoints2[matches12[:, 1]][:, ::-1]
dst = keypoints1[matches12[:, 0]][:, ::-1]
```

```
transform_model, inliers = \
    ransac((src, dst), ProjectiveTransform, min_samples=4, residual_threshold=2
r, c = image1.shape[:2]
corners = np.array([[0, 0],[0, r],[c, 0],[c, r]])

warped_corners = transform_model(corners)
all_corners = np.vstack((warped_corners, corners))
corner_min = np.min(all_corners, axis=0)
corner_max = np.max(all_corners, axis=0)
output_shape = (corner_max - corner_min)
output_shape = np.ceil(output_shape[::-1])
offset = SimilarityTransform(translation=-corner_min)
image0_warp = warp(image0, offset.inverse, output_shape=output_shape, cval=-1)
image1_warp = warp(image1, (transform_model + offset).inverse, output_shape=out
image0_mask = (image0_warp != -1)
image0_warp[~image0_mask] = 0
image0_alpha = np.dstack((gray2rgb(image0_warp), image0_mask))
image1_mask = (image1_warp != -1)
image1_warp[~image1_mask] = 0
image1_alpha = np.dstack((gray2rgb(image1_warp), image1_mask))
merged = (image0_alpha + image1_alpha)
alpha = merged[..., 3]
merged /= np.maximum(alpha, 1)[..., np.newaxis]
imsave('output.png', merged)
```