CV2 HW0

Qianhuang Li
ql2465

Code:

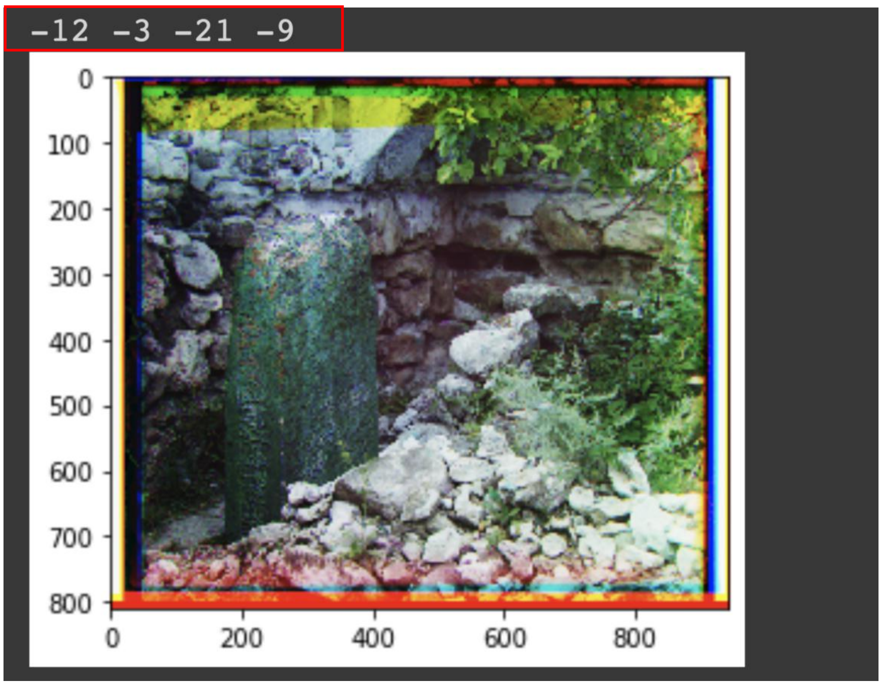
```
import cv2 as cv
# Store the height and width of the images
height, width = images[0].shape

# Pad each image with black by 30 pixels. You do not need to use this, but
# padding may make your implementation easier.
pad_size = 30
images_pad = [np.pad(x, pad_size, mode='constant') for x in images]
# Given two matrices, write a function that returns a number of how well they are aligned.
# The lower the number, the better they are aligned. There are a variety of scoring functions
# you can use. The simplest one is the Euclidean distance between the two matrices.
def score_function(im1, im2):
score = np.linalg.norm(im1 – im2)
return score
# Given two matrices chan1 and chan2, return a tuple of how to shift chan2 into chan1. This
# function should search over many different shifts, and find the best shift that minimizes
# the scoring function defined above.
def align_channels(chan1, chan2):
best_offset = (0,0)
best_score = 1000000
index = 0
for i in range(–30,30):
for j in range(–30,30):
index +=1
offset = (i, j)
chan1_n = chan1[200:600, 300:600]
chan2_n = chan2[(200 – i):(600 – i), (300 – j):(600 – j)]
score = score_function(chan1_n, chan2_n)
if best_score > score:
    best_score = score
    best_offset = offset
print(index," ",best_score)
return best_offset
rg_dy, rg_dx = align_channels(images_pad[0], images_pad[1])
rb_dy, rb_dx = align_channels(images_pad[0], images_pad[2])
print(rg_dy, rg_dx,rb_dy, rb_dx)
# Use the best alignments to now combine the three images. You should use any of the variables
# above to return a tensor that is (Height)x(Width)x3, which is a color image that you can visualize.
def combine_images():
#TODO
obj = np.float32([[1, 0, rg_dx], [0, 1, rg_dy]])
image_g = cv.warpAffine(images[1], obj, (width, height))
obj = np.float32([[1, 0, rb_dx], [0, 1, rb_dy]])
image_b = cv.warpAffine(images[2], obj, (width, height))
final_image = np.stack((images[0],image_g,image_b),axis = -1)
return final_image
final_image = combine_images()
if final_image is not None:
show_image(final_image)
```

Results:

I got two similar results:

The first one used the central area from [200:600, 300:600] in Red Image, it took two seconds to complete the task.

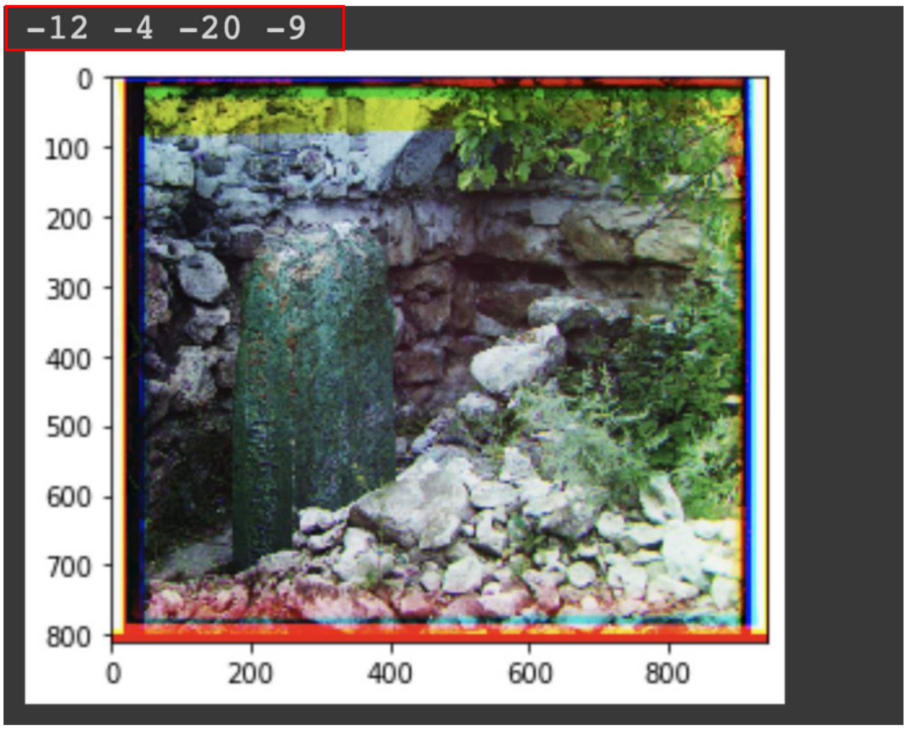


```
-12 -3 -21 -9
```

Picture1 Upper line is the shift pixels of Green and Blue Images

Below is the combined image

The second one used the central area from [200:500, 300:500] in Red Image, it took one seconds to complete the task.



```
-12 -4 -20 -9
```

Picture1 Upper line is the shift pixels of Green and Blue Images

Below is the combined image

My colab link: https://colab.research.google.com/drive/1e-8ByD2b-M1ZLGcKd-8KX3WZdxoWqImm?usp=sharing