82023140 Francesco Leone

Final Report: algorithm for image stitching

Program

```
import math
import cv2
import numpy as np

def pointsDistance(point1, point2):
    vector = (point1[0] - point2[0],point1[1] - point2[1])
    return math.sqrt(vector[0] ** 2 + vector[1] ** 2)

def resize(image, fact):
    width = int(image.shape[1] * fact)
    height = int(image.shape[0] * fact)
    dim = (width, height)
    imageRes = cv2.resize(image, dim, interpolation=cv2.INTER_AREA)
    return imageRes

def highlightRandom(image, points):
    for point in points:
        color = np.random.randint(0, 255, 3)
        color = (int(color[0]), int(color[1]), int(color[2]))
        x = int(point[0])
        y = int(point[1])
        image = cv2.circle(image, (x, y), radius=3, color=color, thickness=-1)
    return image

def highlightRed(image, points):
    for point in points:
        x = int(point[0])
        y = int(point[1])
        image = cv2.circle(image, (x, y), radius=3, color=(0, 0, 255), thickness=-1)
    return image

def centrate(image1, image2, a, b, factx, facty):
    width = int(max(image1.shape[1], image2.shape[1]) * factx)
    height = int(max(image1.shape[0], image2.shape[0]) * facty)
    dim = (width, height)
    x = int((width-width/factx))/2
    y = int((height-height/facty))/2
    M3 = np.float32([[1, 0, x], [0, 1, y]])
    image1 = cv2.warpAffine(image1, M3, dim)
    image2 = cv2.warpAffine(image2, M3, dim)
    for i in range(4):
        a[i] = (a[i][0] + x, a[i][1] + y)
        b[i] = (b[i][0] + x, b[i][1] + y)
    return image1, image2, a, b


def bestFeaturesFirst(matches, kp1, kp2):
    a1 = (int(kp1[matches[0].queryIdx].pt[0]), int(kp1[matches[0].queryIdx].pt[1]))
    a2 = (int(kp1[matches[1].queryIdx].pt[0]), int(kp1[matches[1].queryIdx].pt[1]))
    a3 = (int(kp1[matches[2].queryIdx].pt[0]), int(kp1[matches[2].queryIdx].pt[1]))
    a4 = (int(kp1[matches[3].queryIdx].pt[0]), int(kp1[matches[3].queryIdx].pt[1]))
    b1 = (int(kp2[matches[0].trainIdx].pt[0]), int(kp2[matches[0].trainIdx].pt[1]))
    b2 = (int(kp2[matches[1].trainIdx].pt[0]), int(kp2[matches[1].trainIdx].pt[1]))
    b3 = (int(kp2[matches[2].trainIdx].pt[0]), int(kp2[matches[2].trainIdx].pt[1]))
    b4 = (int(kp2[matches[3].trainIdx].pt[0]), int(kp2[matches[3].trainIdx].pt[1]))
    a = np.int16([a1, a2, a3, a4])
    b = np.int16([b1, b2, b3, b4])
    return a, b

def bestFeaturesQuadrant(dim, matches, kp1, kp2):
    a = [0,0,0,0]
    b = [0,0,0,0]
    q1 = (0, int(dim[0]/2), 0, int(dim[1]/2))
    q2 = (0, int(dim[0]/2), int(dim[1]/2), int(dim[1]))
    q3 = (int(dim[0]/2), int(dim[0]), 0, int(dim[1]/2))
    q4 = (int(dim[0]/2), int(dim[0]), int(dim[1]/2), int(dim[1]))
    q = [q1, q2, q3, q4]
    i = 0
    j = 0
    while(i<4):
        x1 = (int(kp1[matches[j].queryIdx].pt[0]), int(kp1[matches[j].queryIdx].pt[1]))
        y1 = (int(kp2[matches[j].trainIdx].pt[0]), int(kp2[matches[j].trainIdx].pt[1]))
        if((x1[1] in range(q[i][0], q[i][1])) and (x1[0] in range(q[i][2], q[i][3]))):
            a[i] = x1
            b[i] = y1
            j = 0
            i = i+1
        else:
            j = j+1
    return a, b

def bestFeaturesDistance(distance, matches, kp1, kp2):
    a = []
    b = []
    i = 0
    j = 1
    while(i<4):
        if i==0:
            x = [(int(kp1[matches[0].queryIdx].pt[0]),
int(kp1[matches[0].queryIdx].pt[1]))]
            y = [(int(kp2[matches[0].trainIdx].pt[0]),
int(kp2[matches[0].trainIdx].pt[1]))]
            a = a + x
            b = b + y
            i = i+1
        else:
            while(True):
                ok = True
                x1 = (int(kp1[matches[j].queryIdx].pt[0]),
int(kp1[matches[j].queryIdx].pt[1]))
                y1 = (int(kp2[matches[j].trainIdx].pt[0]),
int(kp2[matches[j].trainIdx].pt[1]))
                for k in range(len(a)):
                    if(pointsDistance(x1, a[k])<distance):
                        ok = False
                        break
                if(ok):
                    a = a + [x1]
                    b = b + [y1]
                    j = j+1
                    i = i+1
                    break
                else:
                    j = j+1
    return a, b

def findFeaturesORB(image1, image2):
    orb = cv2.ORB_create()
    kp1, des1 = orb.detectAndCompute(image1, None)
    kp2, des2 = orb.detectAndCompute(image2, None)
    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    matches = bf.match(des1, des2)
    # Sort matches in the order of their distance.
    matches = sorted(matches, key=lambda x: x.distance)
    img1 = image1
```

```python
    img2 = image2
    # a, b = bestFeaturesFirst(matches, kp1, kp2)
    a, b = bestFeaturesDistance(60, matches, kp1, kp2)
    # width = int(min(img1.shape[1], img2.shape[1]))
    # height = int(min(img1.shape[0], img2.shape[0]))
    # dim = (width, height)
    # a, b = bestFeaturesQuadrant(dim, matches, kp1, kp2)
    # img1 = highlightRandom(img1, a)
    # img2 = highlightRandom(img2, b)
    # img1 = highlightRed(img1, a)
    # img2 = highlightRed(img2, b)
    return img1, img2, a, b

def mergePixelsAvg(pixel1, pixel2):
    x1 = pixel1[0]
    y1 = pixel1[1]
    z1 = pixel1[2]
    x2 = pixel2[0]
    y2 = pixel2[1]
    z2 = pixel2[2]
    if(x1==0 and y1==0 and z1==0):
        pixel = (x2, y2, z2)
    elif(x2==0 and y2==0 and z2==0):
        pixel = (x1, y1, z1)
    else:
        x = (int(x1) + int(x2)) / 2
        y = (int(y1) + int(y2)) / 2
        z = (int(z1) + int(z2)) / 2
        pixel = (x,y,z)
    return pixel

def mergePixelsPrior(pixel1, pixel2):
    x1 = pixel1[0]
    y1 = pixel1[1]
    z1 = pixel1[2]
    x2 = pixel2[0]
    y2 = pixel2[1]
    z2 = pixel2[2]
    if(x1==0 and y1==0 and z1==0):
        pixel = (x2, y2, z2)
    else:
        pixel = (x1, y1, z1)
    return pixel

def mergeImages (image1, image2):
    width = int(image1.shape[1])
    height = int(image1.shape[0])
    mergedImage = np.zeros((height, width, 3), np.uint8)
    for x in range(width):
        for y in range(height):
            # mergedImage[y, x] = mergePixelsAvg(image1[y, x], image2[y, x])
            mergedImage[y, x] = mergePixelsPrior(image2[y, x], image1[y, x])
    return mergedImage

def translation(image1, image2):
    img1, img2, a, b = findFeaturesORB(image1, image2)
    a1 = a[0]
    b1 = b[0]
    pts1 = a1[0] - b1[0]
    pts2 = a1[1] - b1[1]
    width = int((max(img1.shape[1], img2.shape[1]) + abs(pts1)) *1)
    height = int((max(img1.shape[0], img2.shape[0]) + abs(pts2)) *1)
    dim = (width, height)
    imga = img2
    imgb = img1

#the translations occur so that the images are not cropped
    if pts1 < 0:
        pts1 = -pts1
        M1 = np.float32([[1, 0, pts1], [0, 1, 0]])
        imgb = cv2.warpAffine(img1, M1, dim)
    else:
        M1 = np.float32([[1, 0, pts1], [0, 1, 0]])
        imga = cv2.warpAffine(img2, M1, dim)
    imgc = imga
    imgd = imgb
    if pts2 < 0:
        pts2 = -pts2
        M2 = np.float32([[1, 0, 0], [0, 1, pts2]])
        imgd = cv2.warpAffine(imgb, M2, dim)
    else:
        M2 = np.float32([[1, 0, 0], [0, 1, pts2]])
        imgc = cv2.warpAffine(imga, M2, dim)
    M3 = np.float32([[1, 0, 0], [0, 1, 0]])
    imgc = cv2.warpAffine(imgc, M3, dim)
    imgd = cv2.warpAffine(imgd, M3, dim)
    return imgc, imgd

def perspective(image1, image2):
    img1, img2, a, b = findFeaturesORB(image1, image2)
    img1 ,img2, a, b = centrate(img1, img2, a, b, factx=1.8, facty=1.25)
    c0 = ((a[0][0] + b[0][0]) / 2, (a[0][1] + b[0][1]) / 2)
    c1 = ((a[1][0] + b[1][0]) / 2, (a[1][1] + b[1][1]) / 2)
    c2 = ((a[2][0] + b[2][0]) / 2, (a[2][1] + b[2][1]) / 2)
    c3 = ((a[3][0] + b[3][0]) / 2, (a[3][1] + b[3][1]) / 2)
    c = (c0, c1, c2, c3)
    a = np.float32(a)
    b = np.float32(b)
    c = np.float32(c)
    width = int(max(img1.shape[1], img2.shape[1]))
    height = int(max(img1.shape[0], img2.shape[0]))
    dim = (width, height)
    imga = img1
    imgb = img2

# # these can be used to warp only one image
#     # M1 = cv2.getPerspectiveTransform(a, b)
#     # imga = cv2.warpPerspective(imga, M1, dim)
# #this second one is better
#     M2 = cv2.getPerspectiveTransform(b, a)
#     imgb = cv2.warpPerspective(imgb, M2, dim)

    M1 = cv2.getPerspectiveTransform(a, c)
    imga = cv2.warpPerspective(imga, M1, dim)
    M2 = cv2.getPerspectiveTransform(b, c)
    imgb = cv2.warpPerspective(imgb, M2, dim)
    return imga, imgb


#perspective, 2 images
def testPerspective(name1, name2):
# name1 and name2 are the names of the two images to stitch
# here the names are turned into paths
    name1 = 'photos/perspective/'+name1+'.jpeg'
    name2 = 'photos/perspective/' + name2 + '.jpeg'
    image1 = cv2.imread(name1)
    image2 = cv2.imread(name2)
    image1 = resize(image1, 0.25)
    image2 = resize(image2, 0.25)

    img1 = image1
    img2 = image2
    # imga, imgb = translation(img1, img2)
    imga, imgb = perspective(img1, img2)
    fimg = mergeImages(imga, imgb)
```

```python
#uncomment to see all steps
    # cv2.imshow('i1', img1)
    # cv2.imshow('i2', img2)
    # cv2.waitKey(0)
    # cv2.destroyAllWindows()
    # cv2.imshow('wi1', imga)
    # cv2.imshow('wi2', imgb)
    # cv2.waitKey(0)
    # cv2.destroyAllWindows()
    cv2.imshow('fi', fimg)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

def testTranslation():
    #translation, 4 images
    image1 = cv2.imread('photos/translation/4a.jpeg')
    image2 = cv2.imread('photos/translation/4b.jpeg')
    image3 = cv2.imread('photos/translation/4c.jpeg')
    image4 = cv2.imread('photos/translation/4d.jpeg')
    image1 = resize(image1, 0.25)
    image2 = resize(image2, 0.25)
    image3 = resize(image3, 0.25)
    image4 = resize(image4, 0.25)

    imga, imgb = translation(image1, image2)
    fimg1 = mergeImages(imga, imgb)
# uncomment to see all steps
    # cv2.imshow('i1', imga)
    # cv2.imshow('i2', imgb)
    # cv2.waitKey(0)
    # cv2.destroyAllWindows()
    # cv2.imshow('fi1', fimg1)
    # cv2.waitKey(0)
    # cv2.destroyAllWindows()

    imgc, imgd = translation(image3, image4)
    fimg2 = mergeImages(imgc, imgd)
# uncomment to see all steps
    # cv2.imshow('i3', imgc)
    # cv2.imshow('i4', imgd)
    # cv2.waitKey(0)
    # cv2.destroyAllWindows()
    # cv2.imshow('fi2', fimg2)
    # cv2.waitKey(0)
    # cv2.destroyAllWindows()

    imge, imgf = translation(fimg1, fimg2)
    fimg = mergeImages(imge, imgf)
# uncomment to see all steps
#     cv2.imshow('wi1', imge)
#     cv2.imshow('wi2', imgf)
#     cv2.waitKey(0)
#     cv2.destroyAllWindows()
    cv2.imshow('fi', fimg)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

test = int(input("Want to test perspective(1) or translation(2)?"))
if(test==1):
    image1, image2 = input("what couple of images must be used?").split()
    testPerspective(image1, image2)
elif(test==2):
    testTranslation()
else:
    print("command not valid")
```

Results

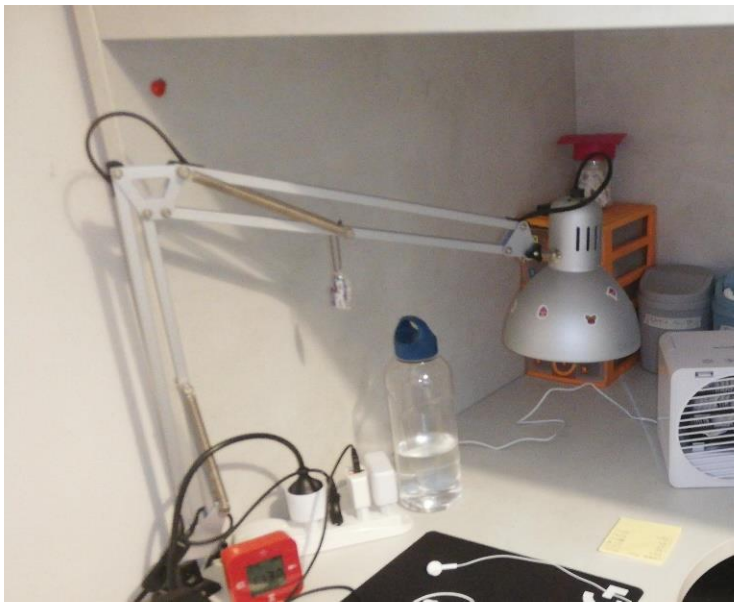
For the implementation, the algorithm has been divided into two functions: one for alignment and one for image merging. Both functions have two variants: the alignment can be done with translation or with perspective warping, whereas the merging can be done overlapping or averaging the two images.

The alignment process is based on feature point matching. In the current implementation of the algorithm, brute-force matching with ORB Descriptors was used, because according to the OpenCV-Phyton documentation is one of the simplest and more generally applicable matching techniques. However, it is probably not the best one for this application, so for a better result different matching technique should be tested.

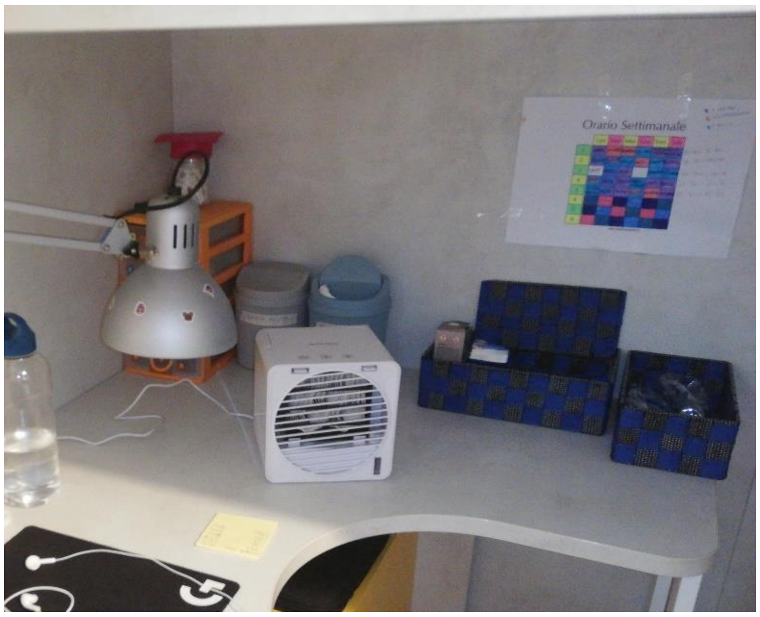
Translation

Starting images

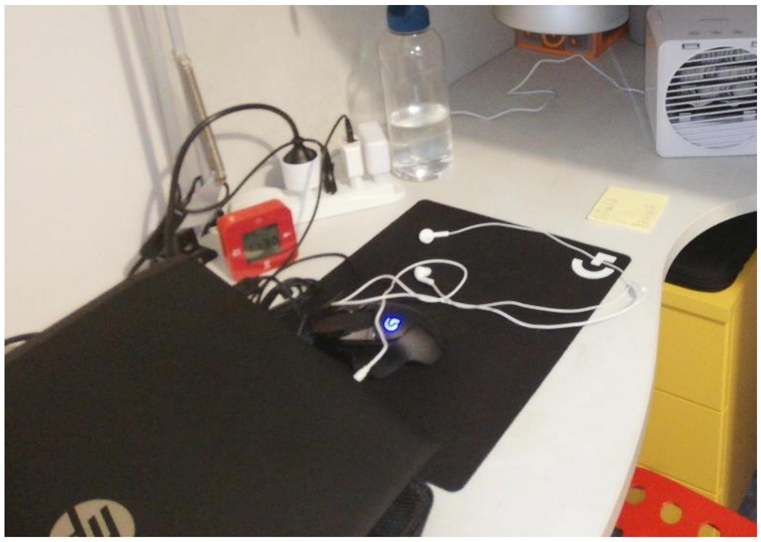
1a



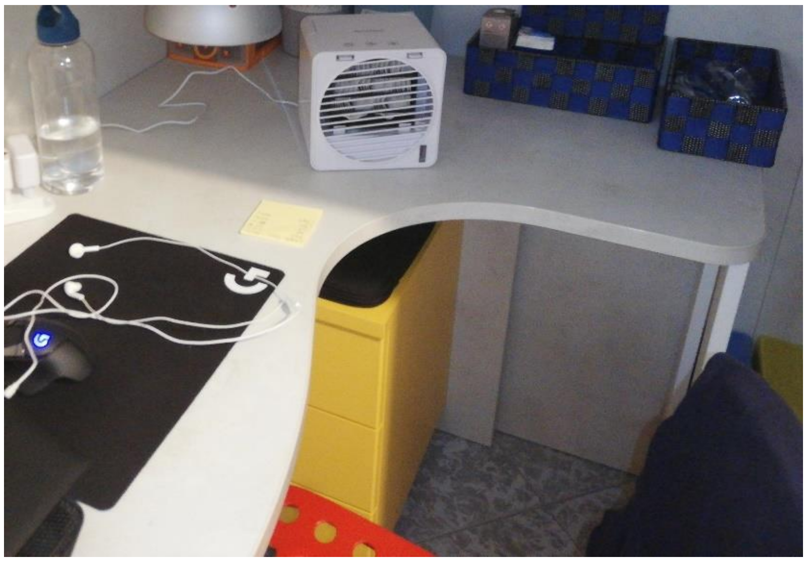
1b



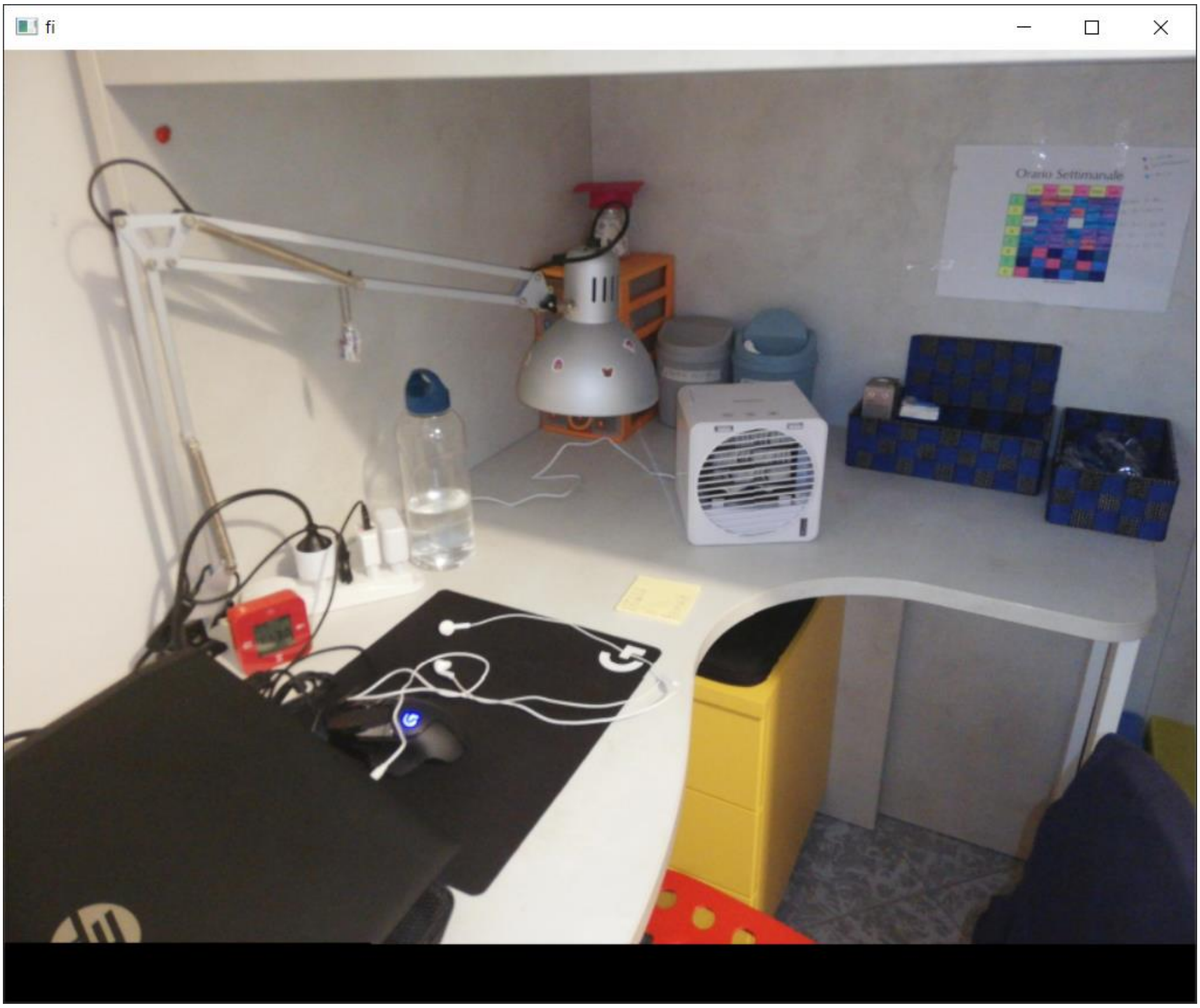
1c



1d



My result



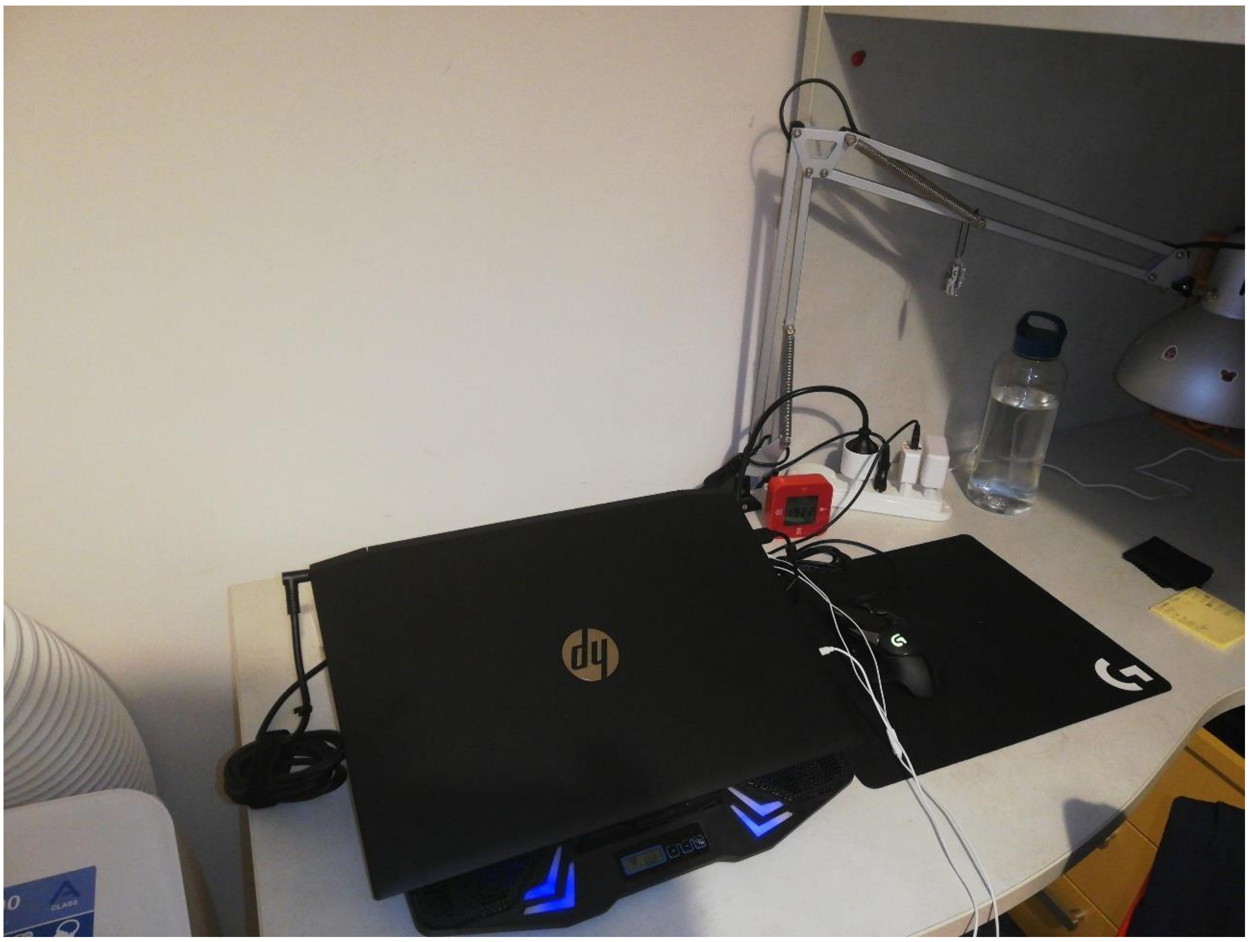
The alignment is done with translation for images that represent objects very distant from the camera (e.g., panoramas), or for recomposing fragmented images, as in this case. The implementation is based on the common feature points in the two images: the best matching points are found and used to calculate the translation vector of the images. Also, the translation is done always on the image that would require a positive translation for each axis. In this way, the images are never cropped.

Open-source tool result

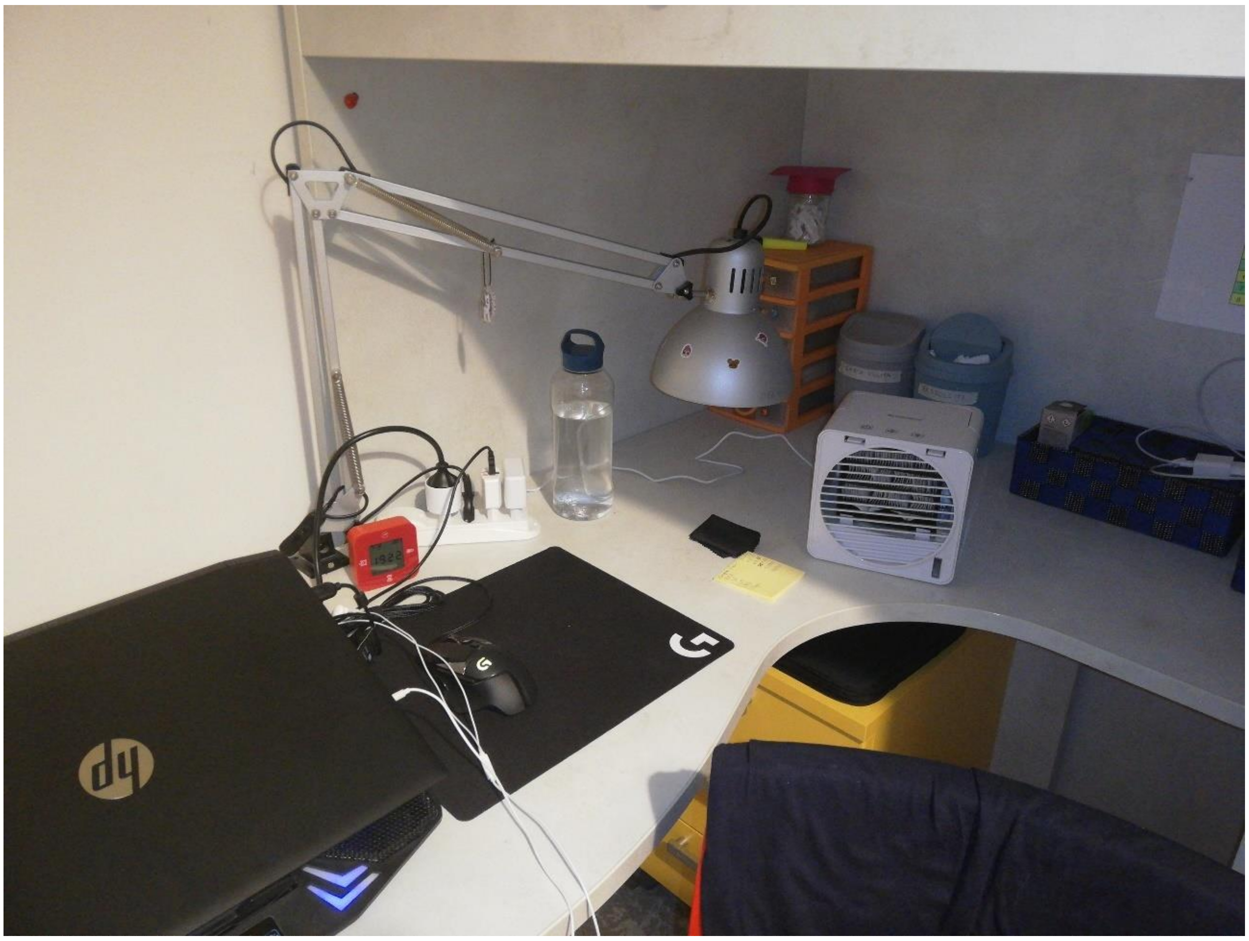
For the pure translations, the algorithm works well: the only difference with the compared open-source tool stands in the black bands that may be added at the borders of the final image because of the previous translations. This result was expected, since when the images are simply translated the pixels have a direct correspondence in the two images and no distortion is introduced with the alignment.

Perspective transformation: smaller camera movement

Starting images

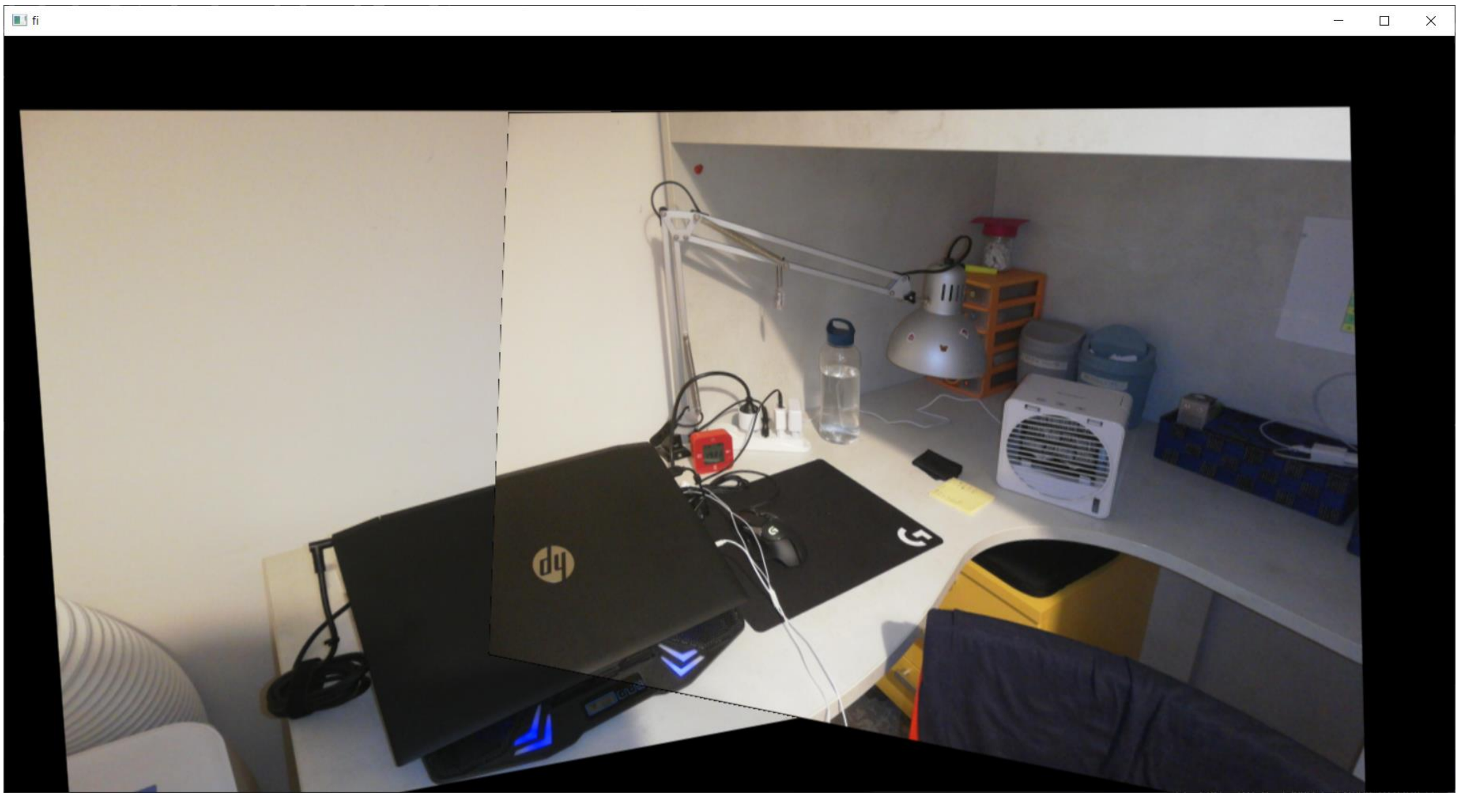


2a



2b

My result

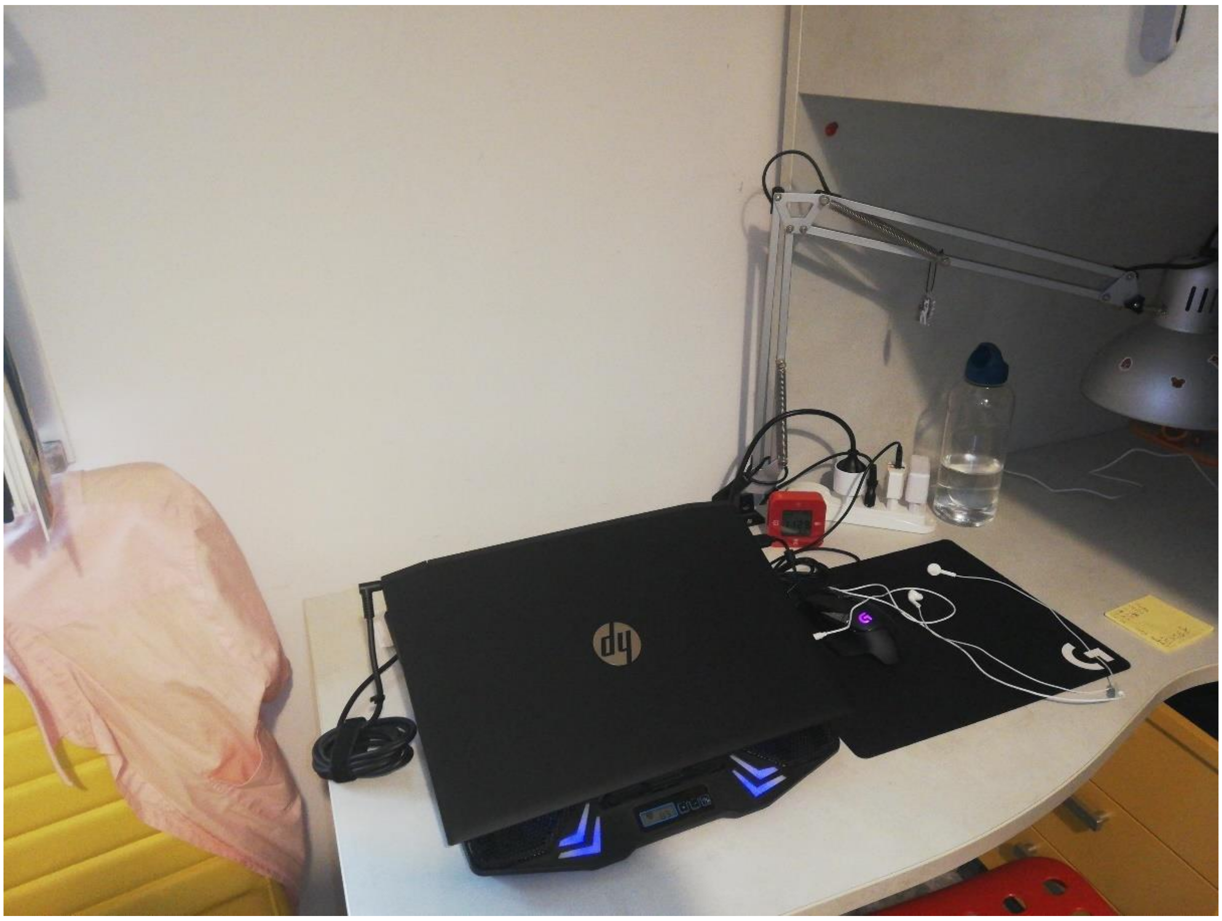


When the images are not simply translated, the perspective transformation must be used. In order to find the warping matrix, the four best matching points with a certain distance between each other are chosen. The distance between the matching points is needed to avoid situations in which the points are too close and so the perspective transformation introduces an important distortion far from these points. For the final implementation, the distance is 60 (this is the vectorial distance between two points using a pixel as a unit), found empirically. This is not the best value for each image but seems to be the most suitable in general.

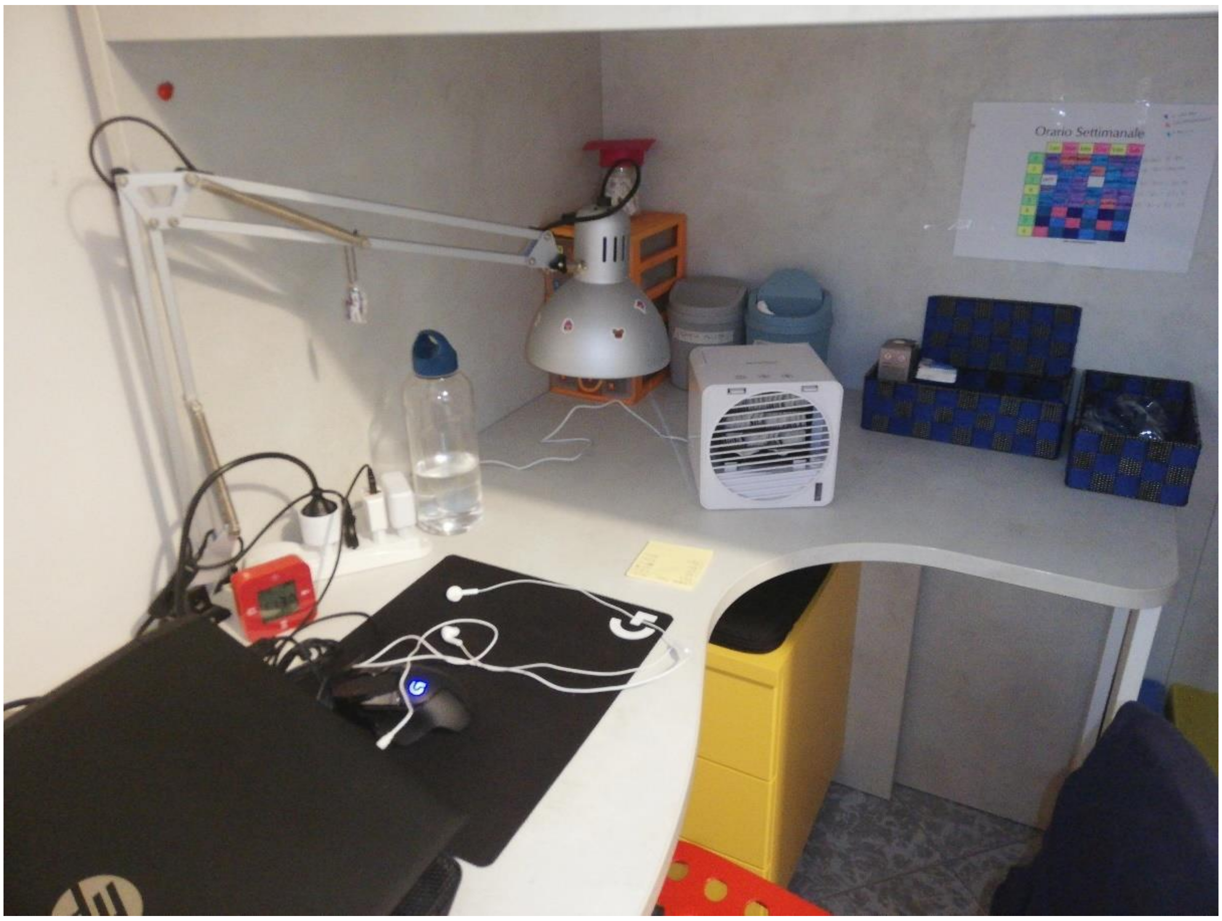
The perspective transformation has been tested in two cases: when the camera performs a small movement, and when the camera performs a more complex movement. In the first case the result seems acceptable. The alignment does not introduce weird distortion and the image is quite clear. The result could be easily improved with some border detection and adjustment.

Perspective transformation: bigger camera movement

Starting images

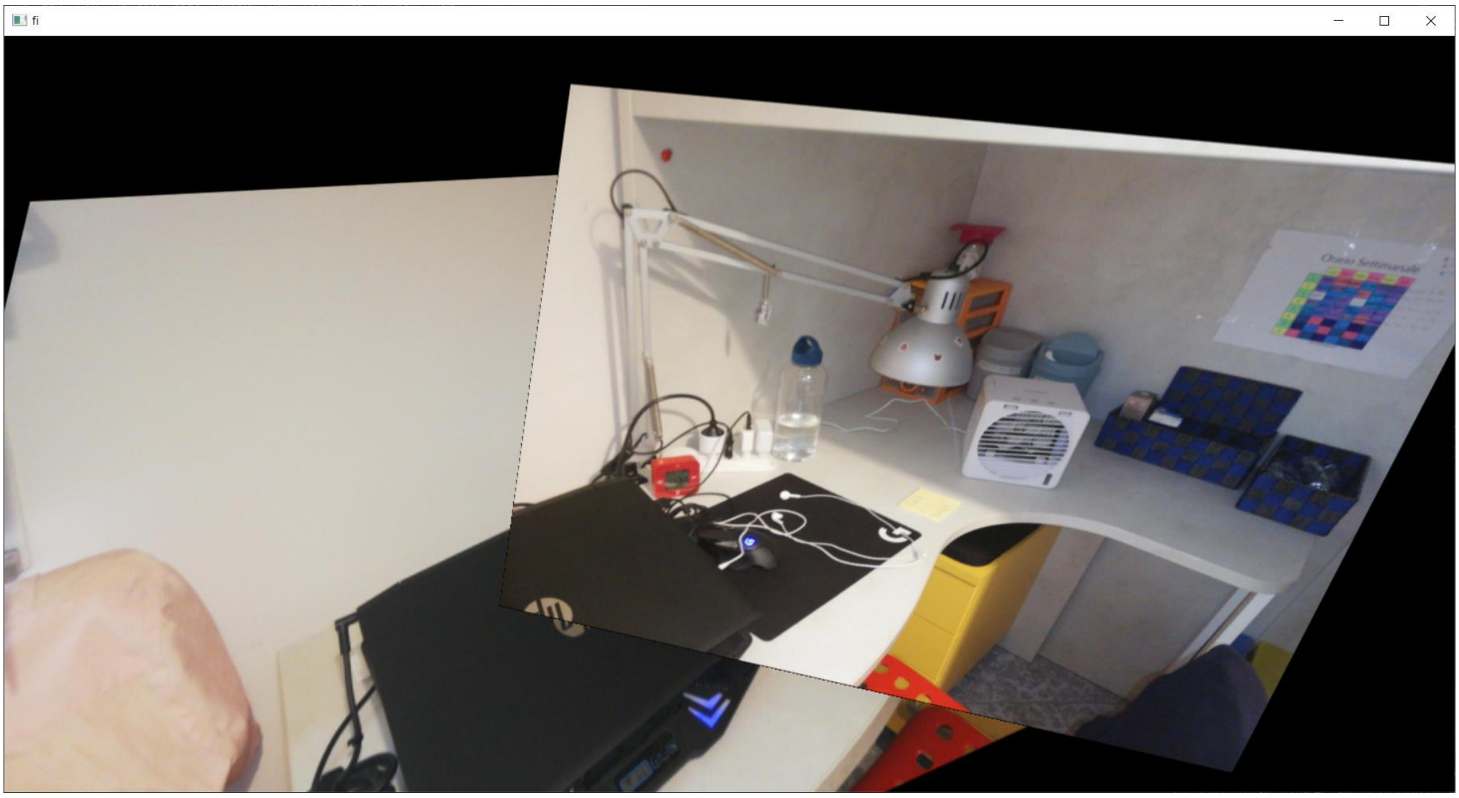


3a



3b

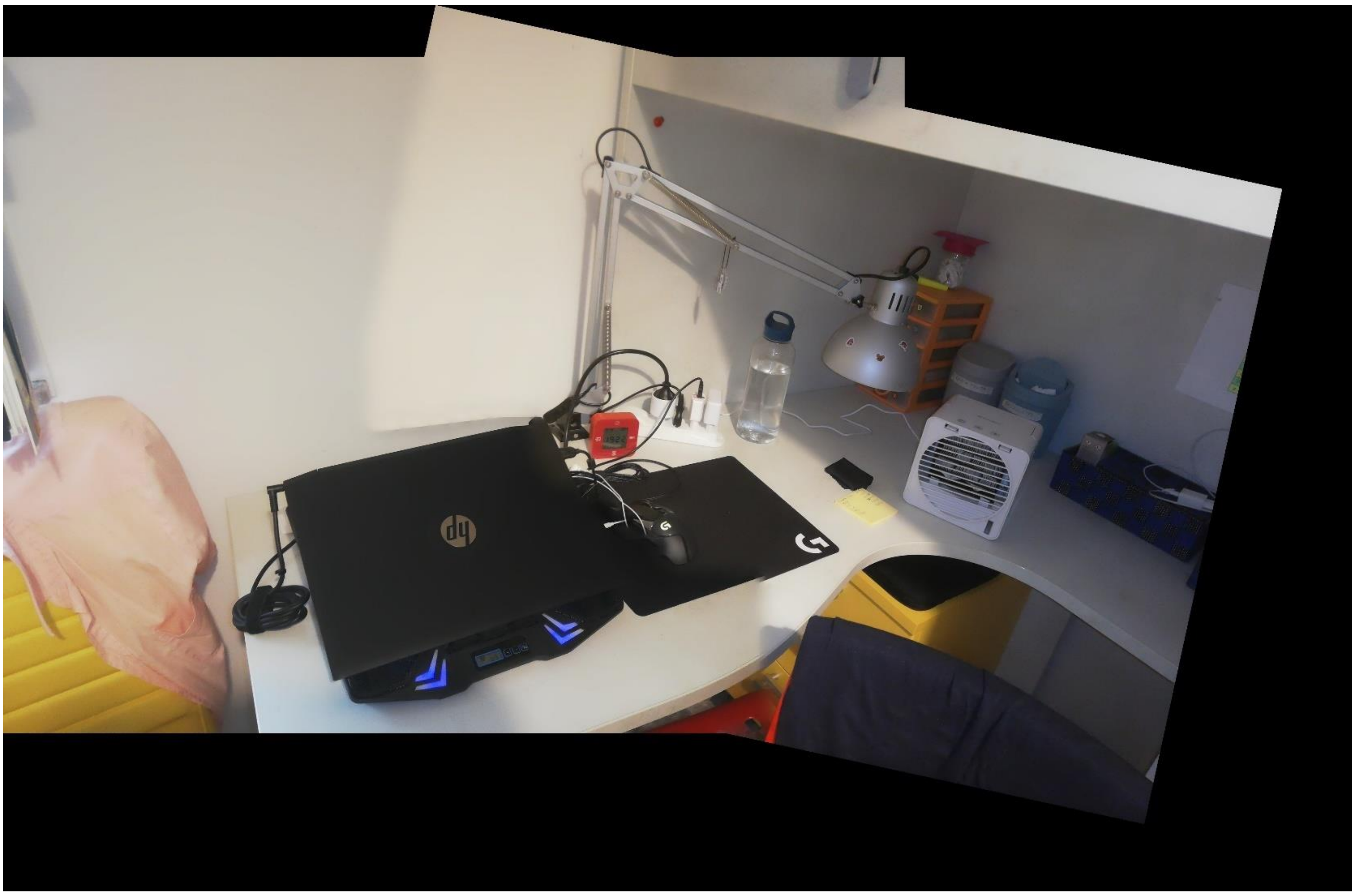
My result



In the case of a more complex camera movement, the result can still be understood by a person, since the objects in the scene are still in the right position, but the information about the borders is severely compromised and the distortion at the corners of the images is sometimes too intrusive. This happens because the complex movement of the camera could not be simply represented through a perspective matrix. In this case, working on border blending is not sufficient: to improve the result, a different warping approach should be used, maybe dividing the complex movement into several simpler movements, and warping the images according to this new set of movements.

Perspective transformation: Open-source tool result

The open-source tool has been able to merge only the images ***2b*** and ***3a***. Any other combination of images could not be merged

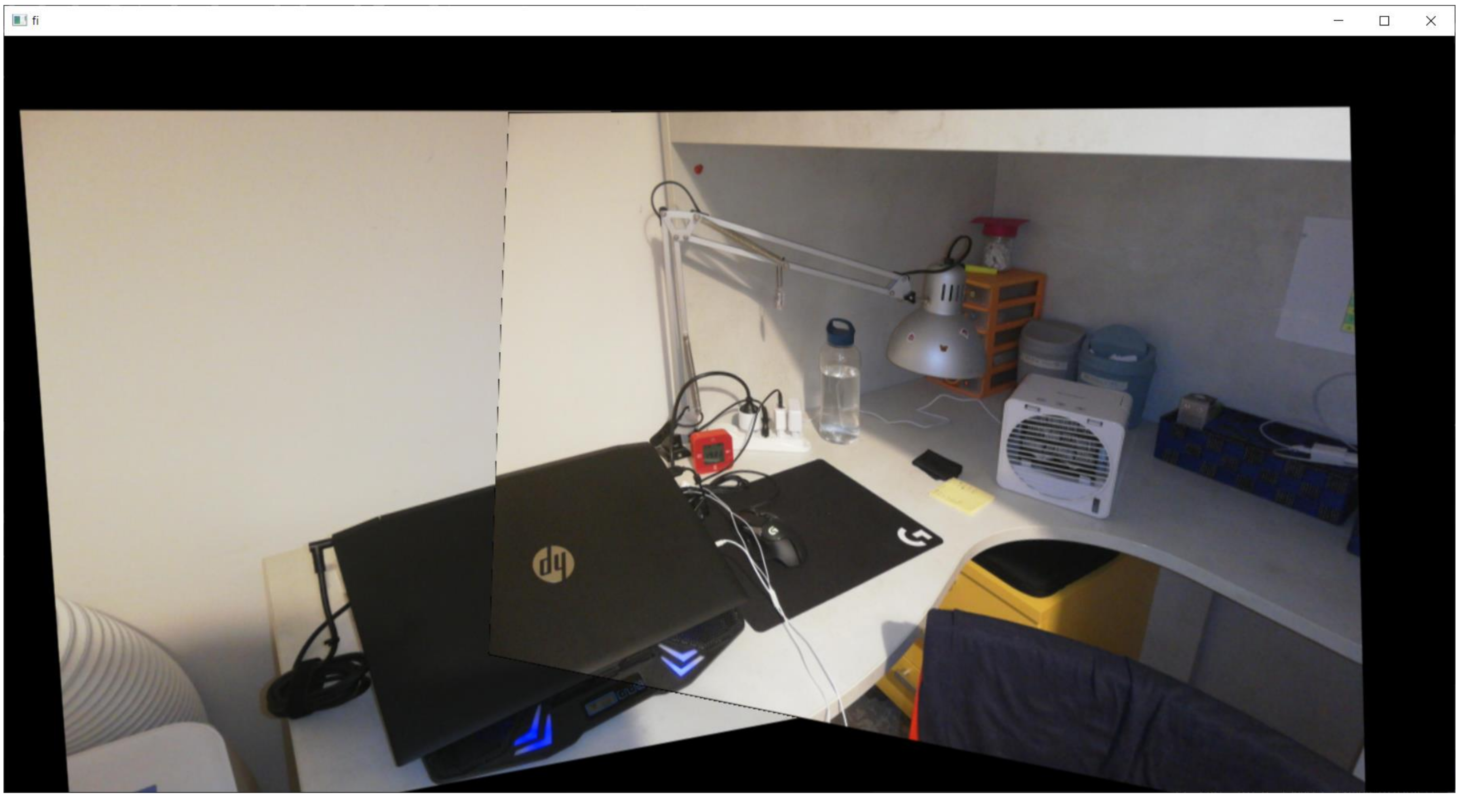


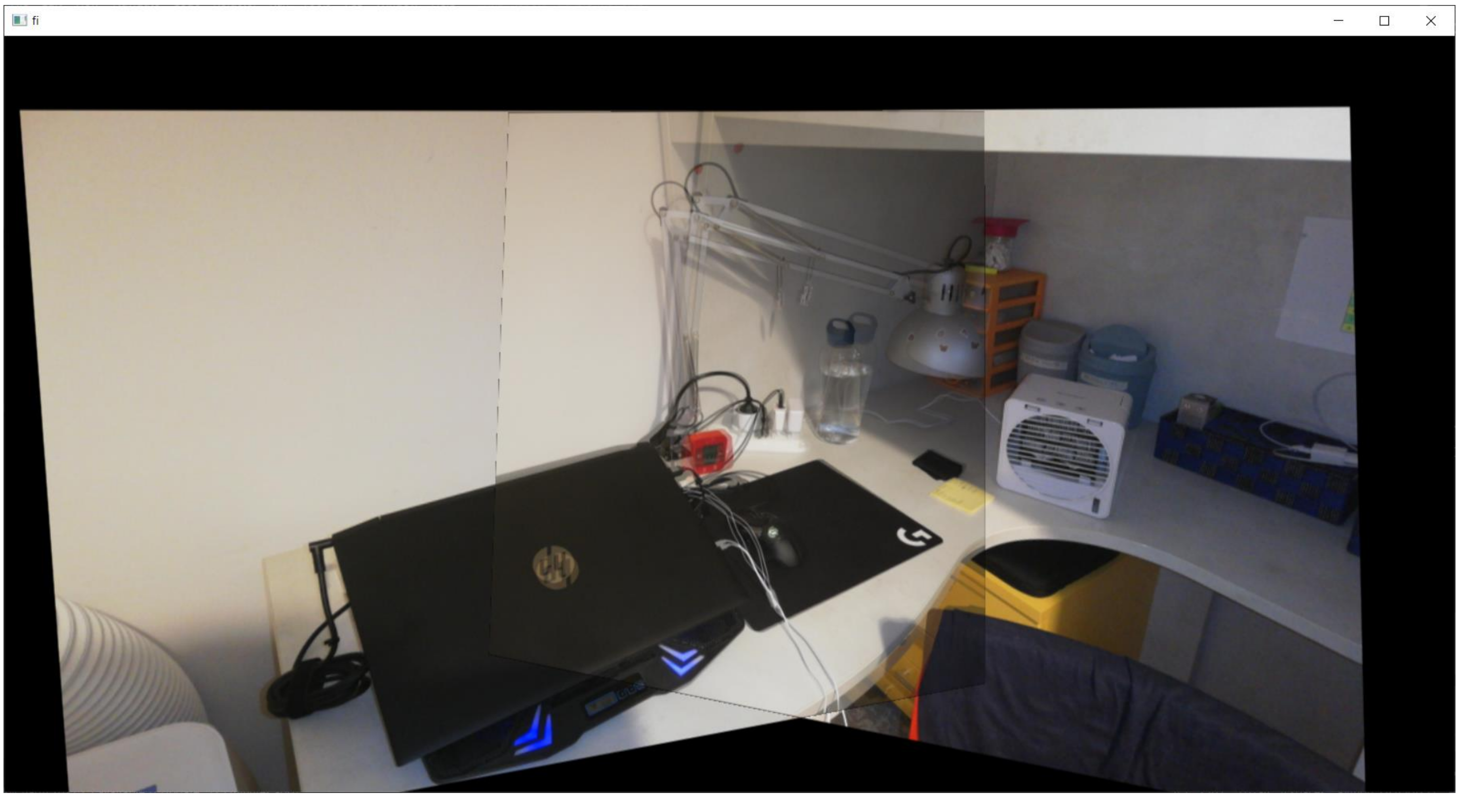
In the case of a simple camera movement, the compared open-source tool is much better. It detects the borders and adjusts them: border blending is almost perfect. Also, there is almost no distortion in the final image and the proportion of the different objects looks very real. The result is an image where almost all the objects are perfectly represented.

On the other hand, the compared open-source tool could not find a solution at all for the second case, when the camera movement is more complex. It seems that the tool prefers to produce a result only if it could be almost perfect, otherwise it asks for more images to better calculate the camera movements

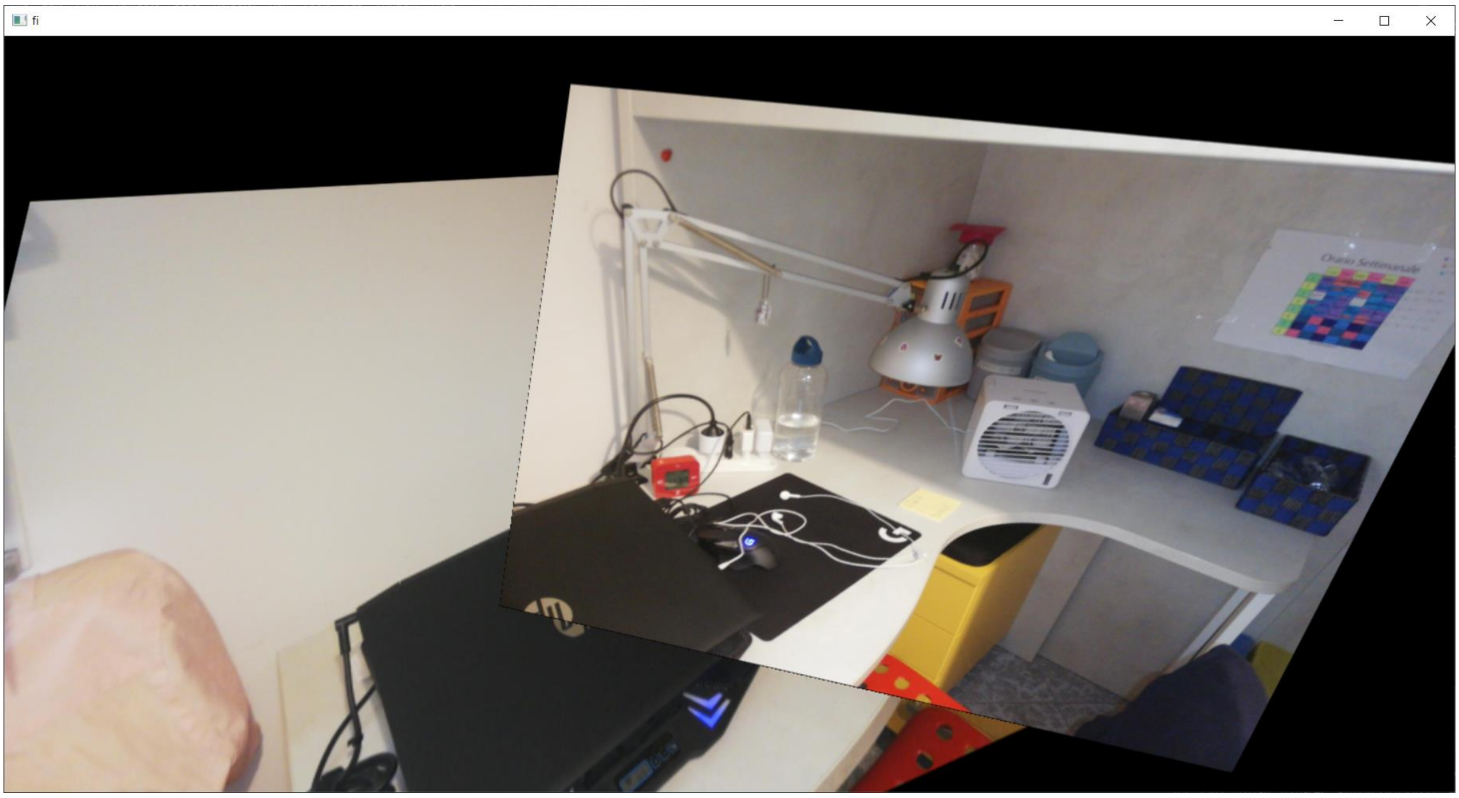
Overlapping vs average

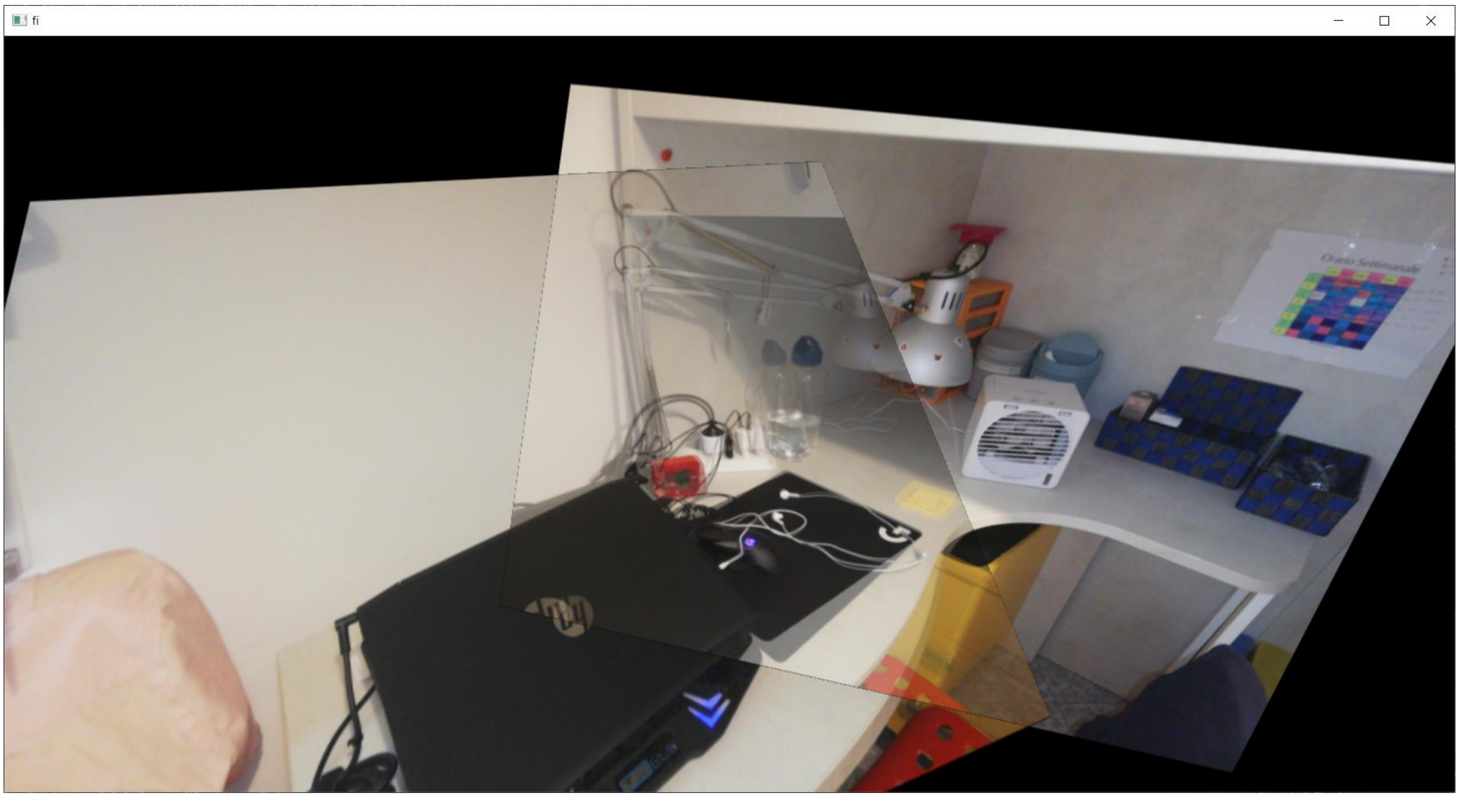
Perspective transformation: smaller camera movement





Perspective transformation: bigger camera movement





Finally, overlapping and averaging the images have been tested. The overlapping makes the borders much more evident, but the final image is clearer than with averaging. On the other hand, averaging preserves information about the objects in the two images, that could be used for a better blending. However, averaging introduces a ghosting effect that can be very troublesome in case of complex camera movements, because several objects appear twice. For this reason, I would prefer overlapping and used it in the current implementation of the algorithm.

References

Compared open-source tool: Image stitching with OpenCV in Python

https://github.com/opencv/opencv/blob/master/samples/python/stitching.py