Name: Chican Costin-Andrei

Group: 442Ca

Assignment 3

-FIPCV-

Subsection 1 - Assigned Set : set08.jpg

Your task is to stitch all images in your image set*x into one big panorama image. Do NOT use the OpenCV Stitcher class! Read the working images from your image set. Compute the keypoints and descriptors for every image in the image set using ORB algorithm. Match the corresponding points between every pair of images using a DescriptorMatcher and choose an appropriate number of good matches. Sort the scores in descending order and only take top 15% of the matches as corresponding points for the next step. Display the good matches obtained for every image pair.

First find two images that have the best matches and work further with those two. Compute the homography between those two images using findHomography and RANSAC. Apply the previously found perspective transformation to all pixels in one image to map it to the other image. Stitch the first image to the second aligned image. Display the stitched output image. Fundamentals of Image Processing and Computer Vision 2 Repeat the previous steps to stitch the previously obtained image to a third image in your image set. Do this until you have stitched together all images in your image set.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

img1 = cv2.imread('shanghai-11.png')
img2 = cv2.imread('shanghai-12.png')
img3 = cv2.imread('shanghai-13.png')

plt.figure(1)
plt.subplot(131), plt.imshow(img1, cmap='gray'), plt.title("Image 1")
plt.subplot(132), plt.imshow(img2, cmap='gray'), plt.title("Image 2")
plt.subplot(133), plt.imshow(img3, cmap='gray'), plt.title("Image 3")
plt.show()

orbObj = cv2.ORB_create()  # Create ORB Object
kp1, descriptors1 = orbObj.detectAndCompute(img1, None)  # Computing
descriptors and keypoints
```

```
kp2, descriptors2 = orbObj.detectAndCompute(img2, None)
kp3, descriptors3 = orbObj.detectAndCompute(img3, None)

matcher =
cv2.DescriptorMatcher_create(cv2.DESCRIPTOR_MATCHER_BRUTEFORCE_HAMMING)  #
Creating Matcher Object
GOOD_MATCH_PERCENT = 0.15
MAX_MATCHES = 25

# Image 1 - Image 2 Matcher
matches1 = matcher.match(descriptors1, descriptors2, None)
matches1 = sorted(matches1, key=lambda x: x.distance, reverse=False)
good_matches1 = int(len(matches1) * GOOD_MATCH_PERCENT)
matches1 = matches1[:good_matches1]  # Remove weak matches

# Image 1 - Image 3 Matcher
matches2 = matcher.match(descriptors1, descriptors3, None)
matches2 = sorted(matches2, key=lambda x: x.distance, reverse=False)
good_matches2 = int(len(matches2) * GOOD_MATCH_PERCENT)
matches2 = matches2[:good_matches2]  # Remove weak matches

# Image 2 - Image 3 Matcher
matches3 = matcher.match(descriptors2, descriptors3, None)
matches3 = sorted(matches3, key=lambda x: x.distance, reverse=False)
good_matches3 = int(len(matches3) * GOOD_MATCH_PERCENT)
matches3 = matches3[:good_matches3]  # Remove weak matches

# Drawing Matches
img_matches1 = cv2.drawMatches(img1, kp1, img2, kp2, matches1[0:MAX_MATCHES],
None)
img_matches2 = cv2.drawMatches(img1, kp1, img3, kp3, matches2[0:MAX_MATCHES],
None)
img_matches3 = cv2.drawMatches(img2, kp2, img3, kp3, matches3[0:MAX_MATCHES],
None)

plt.figure(2)
plt.subplot(311), plt.imshow(img_matches1, cmap='gray'), plt.title("Image 1 -
Image 2")
plt.subplot(312), plt.imshow(img_matches2, cmap='gray'), plt.title("Image 1 -
Image 3")
plt.subplot(313), plt.imshow(img_matches3, cmap='gray'), plt.title("Image 2 -
Image 3")
plt.show()
# Best Match Result : Img1 - Img2
points1 = np.zeros((len(matches1), 2), dtype=np.float32)
points2 = np.zeros((len(matches1), 2), dtype=np.float32)

for i, match in enumerate(matches1):
    points1[i, :] = kp1[match.queryIdx].pt
    points2[i, :] = kp2[match.trainIdx].pt

h, mask = cv2.findHomography(points2, points1, cv2.RANSAC)  # Compute
Homography
img1_height, img1_width, img1_channels = img1.shape
img2_height, img2_width, img2_channels = img2.shape
im2Aligned = cv2.warpPerspective(img2, h, (img2_width + img1_width,
img2_height))  # Align Img2
```

```
stitched_Image12 = np.copy(im2Aligned)  # Stitch Img 1 with Aligned Img 2
stitched_Image12[0:img1_height, 0:img1_width] = img1

plt.figure(3)
plt.imshow(stitched_Image12, cmap='gray'), plt.title("Image 1 - Image 2
Stiched")
plt.show()

# Repeating the above steps for Stiching: stitched_Image12 - Image 3
stitched_Image12 = stitched_Image12[:, :1343]
kp12, descriptors12 = orbObj.detectAndCompute(stitched_Image12, None)
matches12 = matcher.match(descriptors12, descriptors3, None)
matches12 = sorted(matches12, key=lambda x: x.distance, reverse=False)
good_matches12 = int(len(matches12) * GOOD_MATCH_PERCENT)
matches12 = matches12[:good_matches12]  # Remove weak matches

# Drawing Matches
img_matches12 = cv2.drawMatches(stitched_Image12, kp12, img3, kp3,
matches12[0:MAX_MATCHES], None)
plt.figure(4)
plt.imshow(img_matches12, cmap='gray'), plt.title("Stiched Image 12 - Image 3
Matches")
plt.show()

# Stiching Images:
points1 = np.zeros((len(matches12), 2), dtype=np.float32)
points2 = np.zeros((len(matches12), 2), dtype=np.float32)
for i, match in enumerate(matches12):
    points1[i, :] = kp12[match.queryIdx].pt
    points2[i, :] = kp3[match.trainIdx].pt

h123, mask123 = cv2.findHomography(points2, points1, cv2.RANSAC)  # Compute
Homography
img12_height, img12_width, img12_channels = stitched_Image12.shape
img3_height, img3_width, img3_channels = img3.shape
im3Aligned = cv2.warpPerspective(img3, h123, (img12_width + img3_width,
img3_height))
stitched_Image123 = np.copy(im3Aligned)  # Stitch Img 1 with Aligned Img 2
stitched_Image123[0:img12_height, 0:img12_width] = stitched_Image12

plt.figure(5)
plt.imshow(stitched_Image123, cmap='gray'), plt.title("Image 1 - Image 2 -
Image 3 Stiched")
plt.show()
stitched_Image123_noblck = np.copy(stitched_Image123)
stitched_Image123_noblck = stitched_Image123_noblck[:, :1830]
cv2.imwrite('stiched_image.jpg', stitched_Image123_noblck,
[cv2.IMWRITE_JPEG_QUALITY, 80])

plt.figure(6)
plt.subplot(2, 1, 1), plt.imshow(stitched_Image123, cmap='gray'),
plt.title("Image 1 - Image 2 - Image 3 Stiched")
plt.subplot(2, 1, 2), plt.imshow(stitched_Image123_noblck, cmap='gray'),
plt.title("No black -Image 1 - Image 2 - "
"Image 3 Stiched")
plt.show()
```

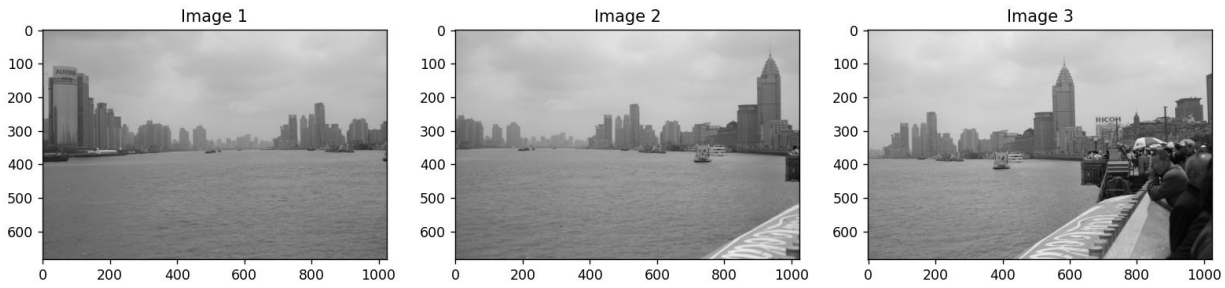

Figure 1 – Assigned Images

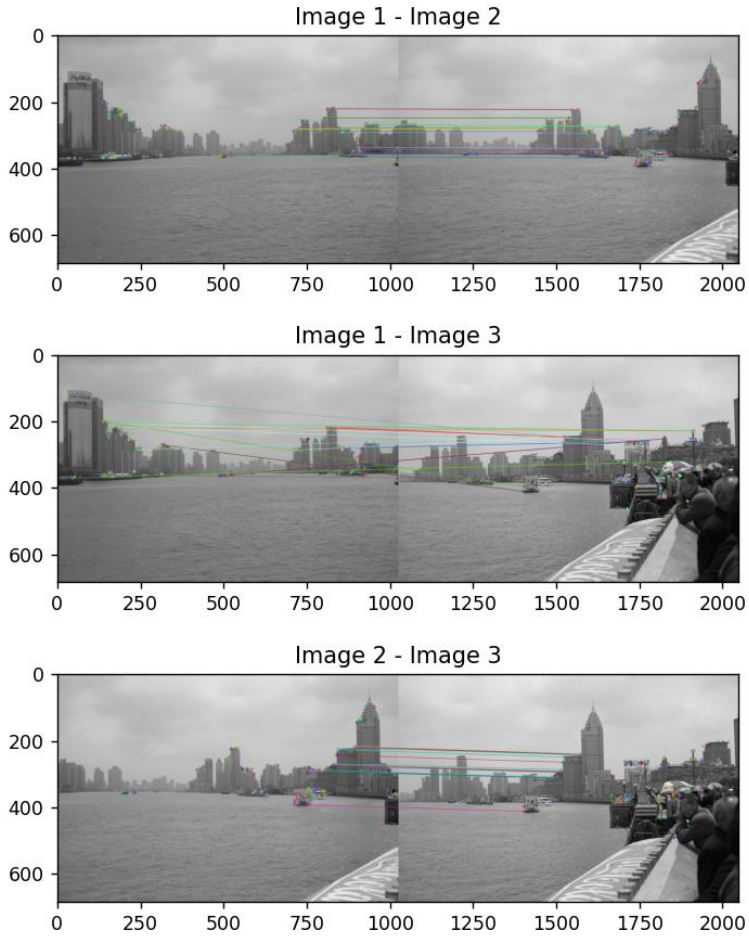

Figure 2 – Found Matches

By looking at the matches in Figure 2 we can tell right from the start that the matches between Image1 and Image 3 are not accurate at all, so we can rule out this case. The matches between Image 1 – Image 2 are almost perfect, most of the essential buildings being detected, along with the boat in the two pictures. As a result, the matches between Image 1 and Image 2 had been used for the following parts of the exercise.

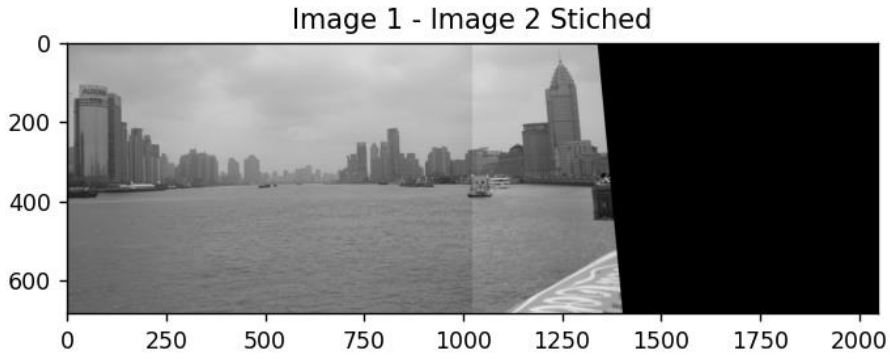

Figure 3 – Stitching of Image 1 and Image 2

The choice of matches made in the previous part of the exercise is confirmed by the stitching process, Image 1 and Image 2 had been combined perfectly, with no details being lost and no alignment issues.

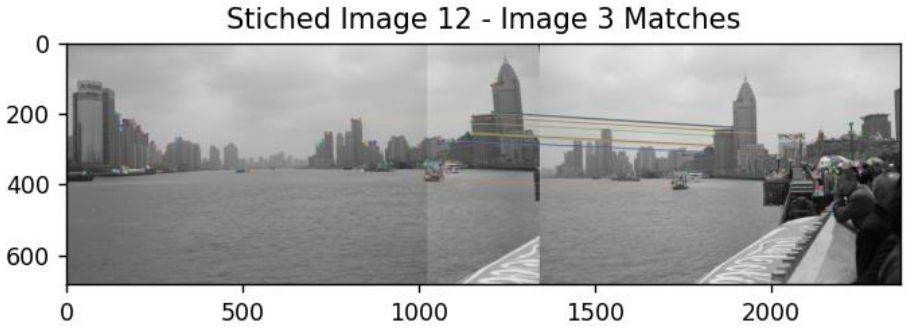

Figure 4 – Found matches

In order to stitch the third image to the first two, the above process needs to be repeated. Firstly, a search of matches between the Stitching of Image1 – Image 2 and Image 3 needs to take place. As we can see in Figure 4, the main common buildings between the pictures were detected, so the final stitching can take place.

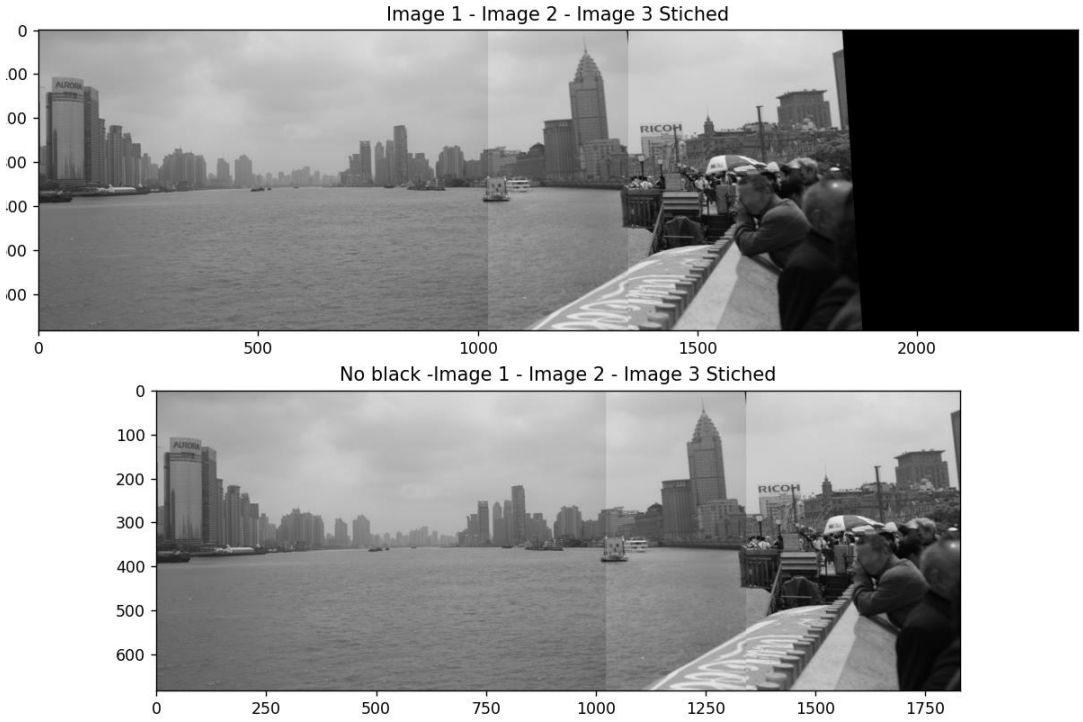

Figure 5 – Stitching of Images 1, 2 and 3

Figure 5 shows the final result, the panorama made by stitching Images 1, 2 and 3. The alignment between the 3 photos is almost perfect, so the matches detected were correct. The black box in the top picture had been eliminated by reducing the size of the image to the highest size that doesn't contain any black borders

Subsection 2 - Assigned Set : set03.jpg

Your task is to find the object presented in object.jpg, in all cluttered images from set*y. Read the working images from your image set set*y. Compute the keypoints and descriptors for all images in the image set using SIFT or ORB. Match the corresponding points between 2 images (object & clutter) using the flann-based knnmatcher and choose only good matches (the first match for a keypoint should be significantly better than the second best match for the same keypoint). Set a condition that at least 10 good matches must be there to find the object. Otherwise simply show a message saying not enough matches are present.

If enough good matches have been found, display the matches between the two images and compute the homography between them. Draw a red box around the detected object. To do this, find the points in the cluttered image corresponding to the object in the first image. The object in object.jpg is defined by the 4 corners of this image, so you have to find those corners in the clutter image and draw the corresponding polygon in the clutter image. Display the matches and the detected object for all clutter images.

```
# SECTION 2

obj = cv2.imread('object.jpg')
img1 = cv2.imread('clutter01.jpg')
img2 = cv2.imread('clutter02.jpg')
img3 = cv2.imread('clutter03.jpg')
img4 = cv2.imread('clutter04.jpg')

plt.figure(1)
plt.subplot(151), plt.imshow(img1[:, :, ::-1]), plt.title("Clutter 1")
plt.subplot(152), plt.imshow(img2[:, :, ::-1]), plt.title("Clutter 2")
plt.subplot(153), plt.imshow(img3[:, :, ::-1]), plt.title("Clutter 3")
plt.subplot(154), plt.imshow(img4[:, :, ::-1]), plt.title("Clutter 4")
plt.subplot(155), plt.imshow(obj[:, :, ::-1]), plt.title("Object")
plt.show()

# Computing descriptors and keypoints using SIFT
siftObj = cv2.SIFT_create()  # Create SIFT Object
kp1, descriptors1 = siftObj.detectAndCompute(img1, None)
kp2, descriptors2 = siftObj.detectAndCompute(img2, None)
kp3, descriptors3 = siftObj.detectAndCompute(img3, None)
kp4, descriptors4 = siftObj.detectAndCompute(img4, None)
kp5, descriptors5 = siftObj.detectAndCompute(obj, None)


# Too many descriptors and keypoints => define function

def flann_with_sift(image1, d1, keyp1, image2, d2, keyp2):
    FLANN_INDEX_KDTREE = 1
    index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
```

```
    search_params = dict(checks=100)
    flann_obj = cv2.FlannBasedMatcher(index_params, search_params)
    matches = flann_obj.knnMatch(d1, d2, 2)

    # Filter matches with Lowe's ratio test(src: opencv.com -
tutorial_feature_flann_matcher)
    thresh = 0.67
    valid_matches = []
    for m, n in matches:
        if m.distance < thresh * n.distance:
            valid_matches.append(m)

    if len(valid_matches) < 10:
        print(f"Valid matches: {len(valid_matches)} - Not enough matches")
    else:
        img_matched = np.empty((max(image1.shape[0], image2.shape[0]),
image1.shape[1] + image2.shape[1], 3),
                               dtype=np.uint8)
        cv2.drawMatches(image1, keyp1, image2, keyp2, valid_matches,
img_matched,
                        flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
        plt.figure(), plt.imshow(img_matched[:, :, ::-1]), plt.title(
            f"Matched images with Flann and SIFT")  # Draw Matches

        points1 = np.zeros((len(valid_matches), 2), dtype=np.float32)
        points2 = np.zeros((len(valid_matches), 2), dtype=np.float32)
        for i, match in enumerate(valid_matches):
            points1[i, :] = keyp1[match.queryIdx].pt
            points2[i, :] = keyp2[match.trainIdx].pt
        h, mask = cv2.findHomography(points2, points1, cv2.RANSAC)
        imHeight, imWidth, channels = image1.shape
        imAligned = cv2.warpPerspective(image2, h, (imWidth, imHeight))
        #Draw Rectangle
        cont_image = np.copy(imAligned)
        cont_image = cv2.cvtColor(cont_image, cv2.COLOR_BGR2GRAY)
        contours, hierarchy = cv2.findContours(cont_image, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)
        cv2.drawContours(image1, contours, -1, (0, 0, 255), 5)
        plt.figure()
        plt.subplot(1, 2, 1), plt.imshow(image1[:, :, ::-1]),
plt.title("Original")
        plt.subplot(1, 2, 2), plt.imshow(imAligned[:,:,::-1]),
plt.title("Homography")
        plt.show()


flann_with_sift(img1, descriptors1, kp1, obj, descriptors5, kp5)
flann_with_sift(img2, descriptors2, kp2, obj, descriptors5, kp5)
flann_with_sift(img3, descriptors3, kp3, obj, descriptors5, kp5)
flann_with_sift(img4, descriptors4, kp4, obj, descriptors5, kp5)
```

In this exercise, the SIFT algorithm was used for the computation of the keypoints and descriptors for all images in the set. The SIFT algorithm was chosen due to its invariation when subject to scale, rotation, illumination and viewpoint changes. This fact makes it the perfect choice for our images in which a carton of milk must be found while being place in multiple positions and perspectives. Also, the ORB algorithm only detects strong features, such as the big text on the carton. This fact might lead to wrong detections in a case in which we have two different products made by the same brand. The main advantage of the ORB algorithm is its speed, but that is not necessary in our exercise due to only having five pictures to compare.

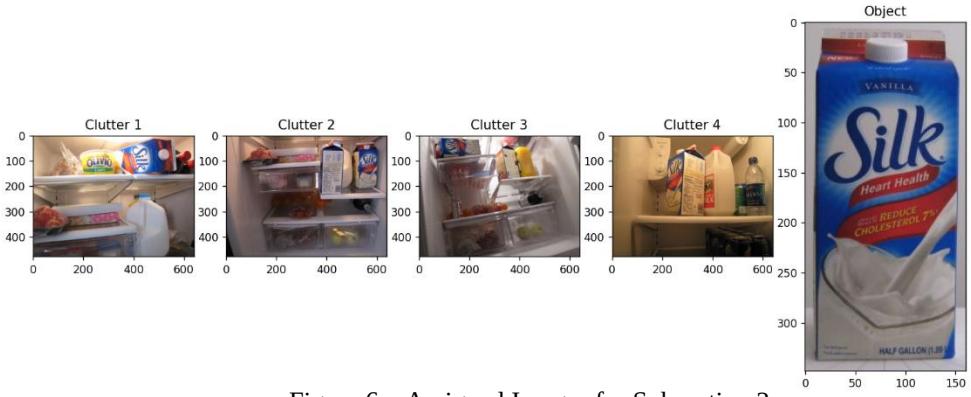

Figure 6 – Assigned Images for Subsection 2

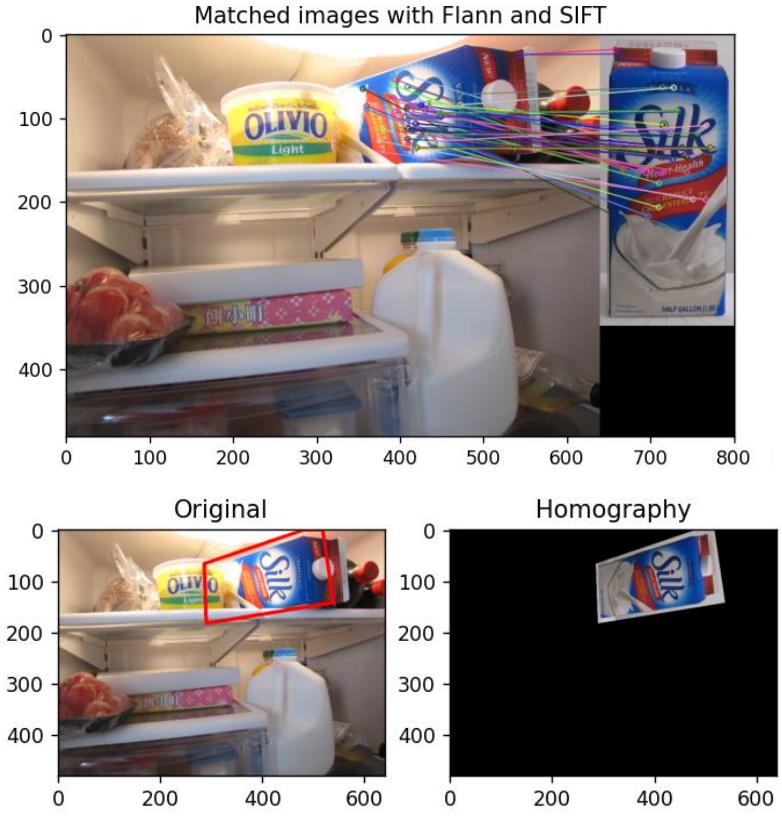

Figure 7 – Object Detected in Clutter 01

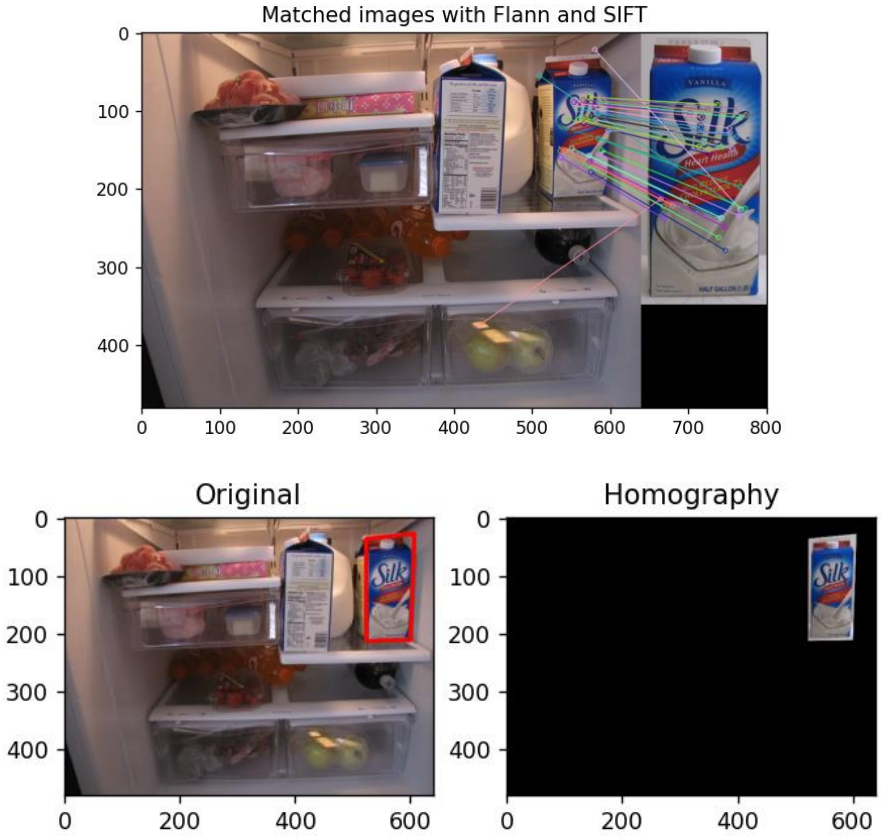

Figure 8 – Object Detected in Clutter 02

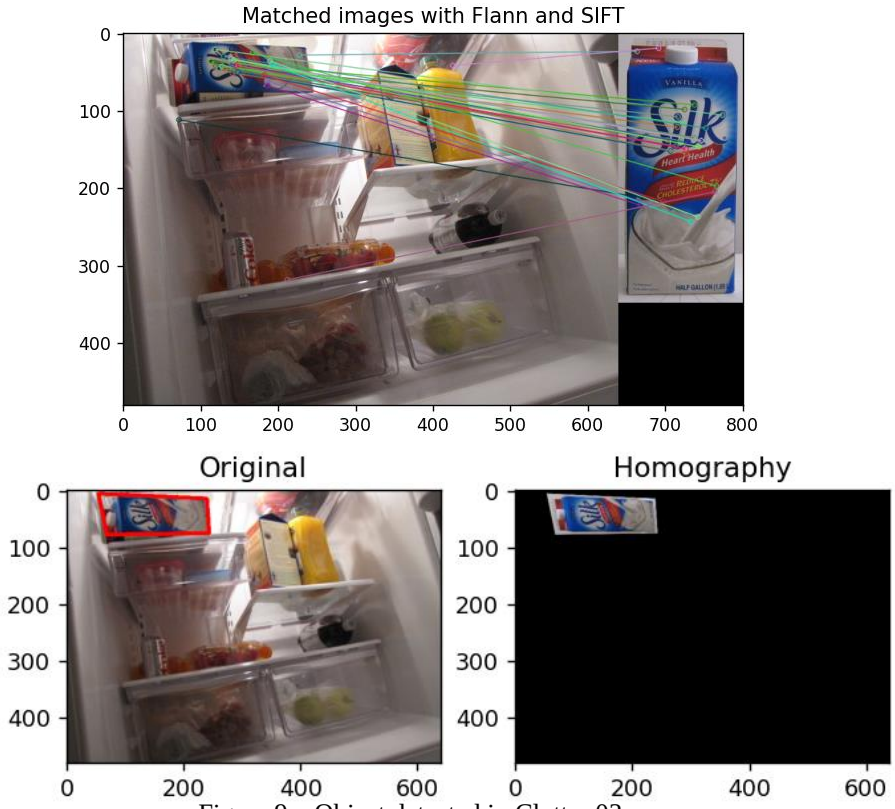

Figure 9 – Object detected in Clutter 03

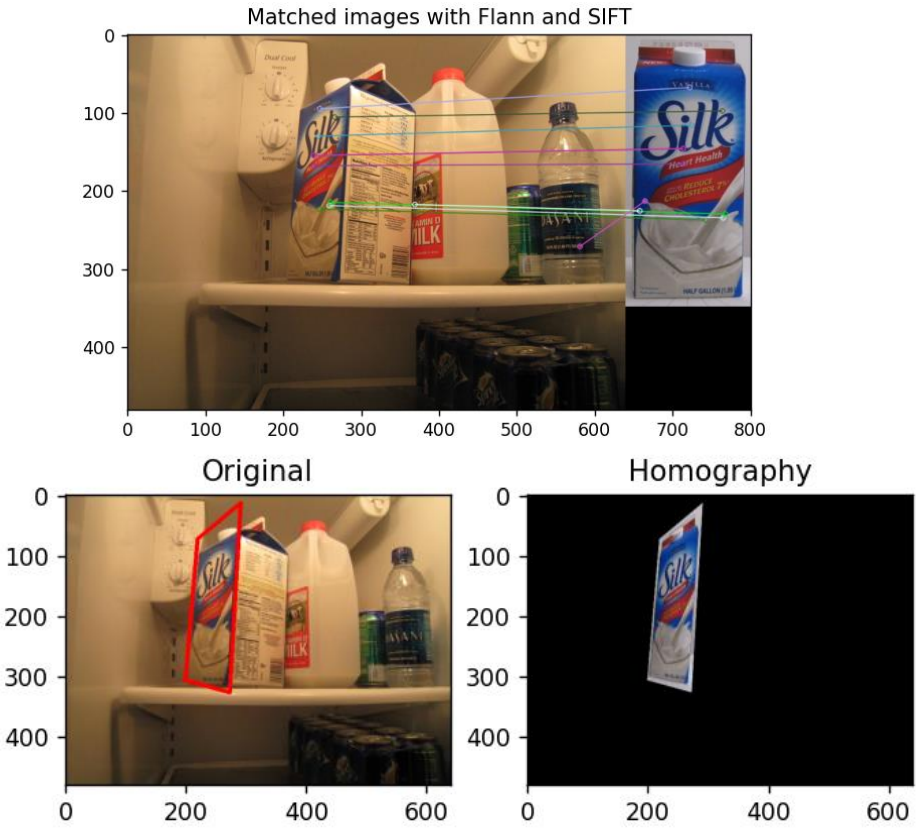

Figure 10 – Object detected in Clutter 04

As the figures listed above show, all the objects were correctly identified and their homography was perfectly drawn. This confirms the decision of going with the SIFT algorithm, each case having enough matches (minimum of 10) to properly identify the object.