In [4]:

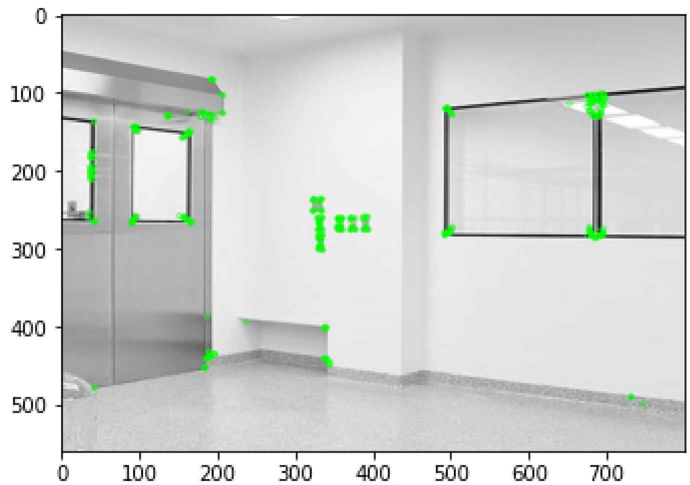
```
## conda install -c pytorch torchvision cudatoolkit=10.1 pytorch
import torchvision as tv
import phototour
import torch
from tqdm import tqdm
import numpy as np
import torch.nn as nn
import math
import tfeat_model
import torch.optim as optim
import torch.nn.functional as F
import torch.backends.cudnn as cudnn
import os
%matplotlib inline
import tfeat_utils
#init tfeat and load the trained weights
tfeat = tfeat_model.TNet()
models_path = 'pretrained-models'
net_name = 'tfeat-liberty'

device_id = 'cuda' if torch.cuda.is_available() else 'cpu'
device = torch.device(device_id)
use_gpu = device_id == 'cuda'

tfeat.load_state_dict(torch.load(os.path.join(models_path,net_name+".params"), map_
location=device))
if torch.cuda.is_available():
    tfeat.cuda()

tfeat.eval()
```

Out[4]:

```
TNet(
  (features): Sequential(
    (0): InstanceNorm2d(1, eps=1e-05, momentum=0.1, affine=False, track
_running_stats=False)
    (1): Conv2d(1, 32, kernel_size=(7, 7), stride=(1, 1))
    (2): Tanh()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil
_mode=False)
    (4): Conv2d(32, 64, kernel_size=(6, 6), stride=(1, 1))
    (5): Tanh()
  )
  (descr): Sequential(
    (0): Linear(in_features=4096, out_features=128, bias=True)
    (1): Tanh()
  )
)
```

In [5]:

```
## getting kp using ORB
import cv2
from matplotlib import pyplot as plt
img_room = cv2.imread('imgs/room1.jpg',0)

orb = cv2.ORB_create()
# find the keypoints with ORB
kp = orb.detect(img_room, None)

# compute the descriptors with ORB
kp, desc_orb = orb.compute(img_room, kp)

# draw only keypoints location,not size and orientation
img2 = cv2.drawKeypoints(img_room, kp, img_room, color=(0,255,0), flags=0)
plt.imshow(img2),plt.show()
print(f'length of descriptors found using ORB: {len(desc_orb)}')
```




```
length of descriptors found using ORB: 500
```

In [6]:

```
# mag_factor is how many times the original keypoint scale
# is enlarged to generate a patch from a keypoint
mag_factor = 3
desc_tfeat = tfeat_utils.describe_opencv(tfeat, img_room, kp, 32, mag_factor, use_g
pu=use_gpu)
# print(f'length of descriptors found using TFeat: {len(desc_tfeat)}')

## length of descriptors is the same since we are using the same keypoints generate
d by ORB
print(f'desc_orb[0]: {desc_orb[0]}') ## 1st keypoint descriptor
print(f'desc_tfeat[0]: {desc_tfeat[0]}') ## 1st keypoint descriptor
print(f'desc_orb[1]: {len(desc_orb[1])}') ## 2nd keypoint descriptor length (32)
print(f'desc_tfeat[1]: {len(desc_tfeat[1])}') ## 2nd keypoint descriptor length (12
8)
```

```
desc_orb[0]: [ 91  58  53 213 111 148 153  55  91 232 146 118  60  30 2
12 230 196  63
  62 141 133 218 241 195 149  76 187 162   9   0  33 253]
desc_tfeat[0]: [ 0.10877752  0.99007213  0.9983085  -0.9966855   0.9962
847   0.9998375
  0.95581603  0.02040958 -0.9984609   0.6303016   0.2885018  -0.1933585
7
 -0.9996648  -0.99982536 -0.7804964  -0.997913   -0.16521657 -0.9850716
6
 -0.4402254   0.8593178   0.2205745   0.09464724 -0.99989724 -0.9858746
5
  0.94623405  0.9994752  -0.60619026 -0.7560385   0.09357091  0.9631967
 -0.11786427 -0.99634    -0.28793883  0.99968004  0.06620529  0.9968847
6
  0.9979051  -0.9968203   0.3207367  -0.8607095  -0.99637556  0.9858191
 -0.9998301   0.7088365   0.8932801   0.8571288  -0.40608186 -0.642607
 -0.9743517  -0.4772837  -0.9698842   0.96350074  0.9994706   0.9999108
  0.9816929  -0.9999762   0.8254248  -0.9769926  -0.18002385 -0.8426741
 -0.5276419  -0.90724254 -0.99955255  0.9909258  -0.5205326  -0.9500755
7
 -0.9994314   0.67885244 -0.80177504 -0.47418317  0.24662878 -0.8930310
6
  0.6504253  -0.21874818  0.868391   -0.98790896  0.9830543   0.9479453
 -0.98937255  0.9185085   0.95979977  0.88604295 -0.44588706  0.4041138
6
  0.9491763  -0.22015972 -0.32336372 -0.98582983 -0.6558997  -0.9840244
7
 -0.53964937  0.27146447  0.44582868 -0.89402705  0.9956143   0.9052976
  0.13848957 -0.5801624  -0.03408958  0.9578376  -0.2035427   0.9474056
4
 -0.0843036  -0.7340471   0.9896767  -0.9017346   0.9715651  -0.6919059
 -0.7406972   0.78603697  0.0125647  -0.9512517  -0.9900483   0.9560787
 -0.71579856  0.98908675 -0.7822163   0.845755    0.9974358  -0.7752567
 -0.89704096 -0.05875105 -0.4666269  -0.28016764  0.9645858   0.7613034
 -0.9993822  -0.9983862 ]
desc_orb[1]: 32
desc_tfeat[1]: 128
```

Some important things for descriptors are:

- They should be independent of keypoint position

> If the same keypoint is extracted at different positions (e.g. because of translation) the descriptor should be the same.

- They should be robust against image transformations:

> Some examples are changes of contrast (e.g. image of the same place during a sunny and cloudy day) and changes of perspective (image of a building from center-right and center-left, we would still like to recognize it as a same building). Of course, no descriptor is completely robust against all transformations (nor against any single one if it is strong, e.g. big change in perspective). Different descriptors are designed to be robust against different transformations which is sometimes opposed to the speed it takes to calculate them.

- They should be scale independent

> The descriptors should take scale in to account. If the "prominent" part of the one keypoint is a vertical line of 10px (inside a circular area with radius of 8px), and the prominent part of another a vertical line of 5px (inside a circular area with radius of 4px) -- these keypoints should be assigned similar descriptors.

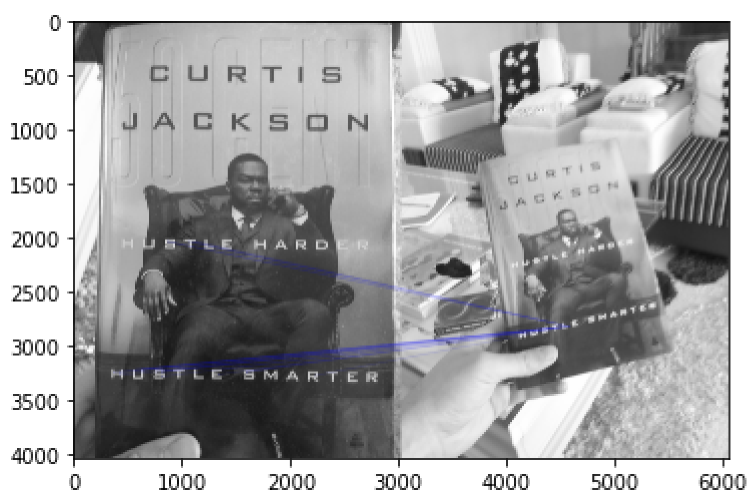
Testing the "accuracy" of our descriptors in image/feature matching

Check here [opencv docs feature matching (https://docs.opencv.org/master/dc/dc3/tutorial_py_matcher.html)](https://docs.opencv.org/master/dc/dc3/tutorial_py_matcher.html)

> Now, that you calculated descriptors for all the keypoinst, you have a way to compare those keypoints. For a simple example of image matching (when you know the images are of the same object, and would like to identify the parts in different images that depict the same part of the scene, or would like to identify the perspective change between two images), you would compare every keypoint descriptor of one image to every keypoint descriptor of the other image. As the descriptors are vectors of numbers, you can compare them with something as simple as Euclidian distance. There are some more complex distances that can be used as a similarity measure, of course. But, in the end, you would say that the keypoints whose descriptors have the smallest distance between them are matches, e.g. same "places" or "parts of objects" in different images.

In [8]:

```
#############################################################
### feature matching using tFeat descriptors
#############################################################
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
img1 = cv.imread('imgs/book.jpg',cv.IMREAD_GRAYSCALE)
img2 = cv.imread('imgs/book_room.jpg',cv.IMREAD_GRAYSCALE)

orb = cv.ORB_create()
kp1 = orb.detect(img1,None)
kp2 = orb.detect(img2,None)

mag_factor = 3
des1 = tfeat_utils.describe_opencv(tfeat, img_room, kp1, 32, mag_factor, use_gpu=use_gpu)
des2 = tfeat_utils.describe_opencv(tfeat, img_room, kp2, 32, mag_factor, use_gpu=use_gpu)

## NOTE: we cant use NORM_HAMMING for tFeat
bf = cv.BFMatcher(cv.NORM_L2)

matches = bf.match(des1,des2) ## match descriptors
matches = sorted(matches, key = lambda x:x.distance) ## sort descriptors
best_matches = matches[:20] ## get 20 best matches

img_matches_tfeat = cv.drawMatches(img1, kp1, img2, kp2, best_matches, None, flags=2, matchColor = (255,0,0))
plt.imshow(cv2.cvtColor(img_matches_tfeat, cv2.COLOR_BGR2RGB))
plt.show()
```



In [9]:

```
############################################################
### feature matching using SIFT descriptors
# ############################################################
# import numpy as np
# import cv2 as cv
# import matplotlib.pyplot as plt
# img1 = cv.imread('imgs/book.jpg',cv.IMREAD_GRAYSCALE)
# img2 = cv.imread('imgs/book_room.jpg',cv.IMREAD_GRAYSCALE)

# sift = cv.xfeatures2d.SIFT_create()

# kp1, des1 = sift.detectAndCompute(img1,None)
# kp2, des2 = sift.detectAndCompute(img2,None)

# bf = cv.BFMatcher(cv.NORM_L2)
# matches = bf.match(des1,des2)
# matches = sorted(matches, key = lambda x:x.distance)
# best_matches = matches[:20] ## get 20 best matches

# img_matches_orb = cv.drawMatches(img1, kp1, img2, kp2, best_matches,None,flags=2,
matchColor = (255,0,0))
# plt.imshow(cv2.cvtColor(img_matches_orb, cv2.COLOR_BGR2RGB))
# plt.show()
```

Based on the original paper of TFeat:

> When using Harris-Affine keypoints our descriptor still outperforms the others, although with a smaller margin. In the case of the DoG keypoints, our networks outperform all the others in terms of mAP.

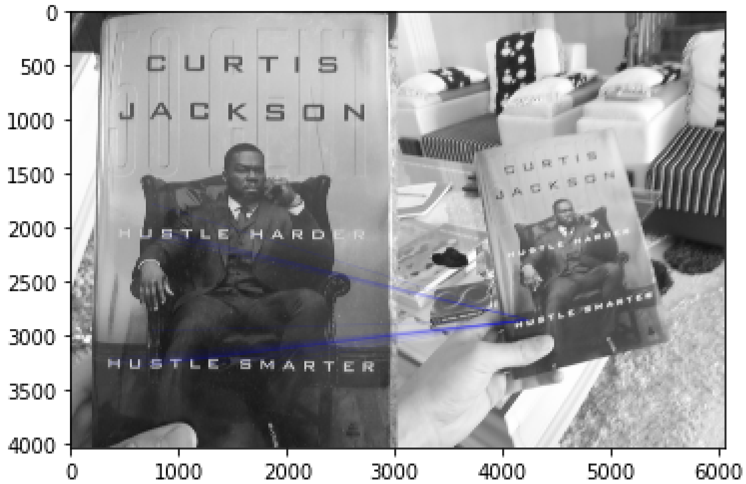
> Since we have to use DoG based keypoint detector we may need to use SIFT which is part of opencv-contrib because it needs a license to be used so we have to install opencv-contrib as well

Harris Affine Region Detector

> In the fields of computer vision and image analysis, the Harris affine region detector belongs to the category of feature detection. Feature detection is a preprocessing step of several algorithms that rely on identifying characteristic points or interest points so to make correspondences between images, recognize textures, categorize objects or build panoramas

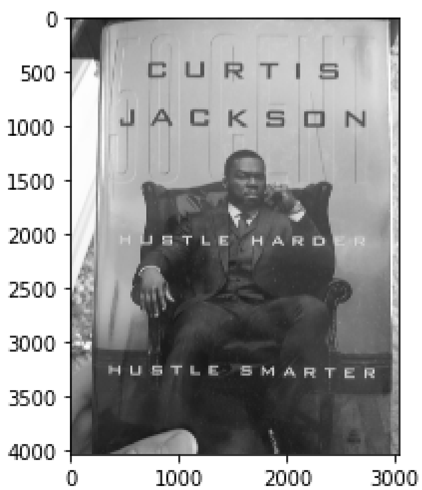
In [12]:

```
######## Using goodFeaturesToTrack to get keypoints
import numpy as np
import cv2
from matplotlib import pyplot as plt

img1 = cv.imread('imgs/book.jpg',cv.IMREAD_GRAYSCALE)
img2 = cv.imread('imgs/book_room.jpg',cv.IMREAD_GRAYSCALE)

corners1 = cv2.goodFeaturesToTrack(img1,25,0.01,10)
corners2 = cv2.goodFeaturesToTrack(img2,25,0.01,10)

def corners_to_keypoints(corners):
    """function to take the corners from cv2.GoodFeaturesToTrack and return cv2.Key
Points"""
    if corners is None:
        keypoints = []
    else:
        keypoints = [cv.KeyPoint(kp[0][0], kp[0][1], 1) for kp in corners]

    return keypoints

kp1 = corners_to_keypoints(corners1)
kp2 = corners_to_keypoints(corners2)

mag_factor = 3
des1 = tfeat_utils.describe_opencv(tfeat, img_room, kp1, 32, mag_factor, use_gpu=us
e_gpu)
des2 = tfeat_utils.describe_opencv(tfeat, img_room, kp2, 32, mag_factor, use_gpu=us
e_gpu)

## NOTE: we cant use NORM_HAMMING for tFeat
bf = cv.BFMatcher(cv.NORM_L2)

matches = bf.match(des1,des2) ## match descriptors
matches = sorted(matches, key = lambda x:x.distance) ## sort descriptors
best_matches = matches[:20] ## get 20 best matches

img_matches_tfeat = cv.drawMatches(img1, kp1, img2, kp2, best_matches, None, flags=
2, matchColor = (255,0,0))
plt.imshow(cv2.cvtColor(img_matches_tfeat, cv2.COLOR_BGR2RGB))
plt.show()
```



In [24]:

```
#### Drawing only keypoints to compare ORB, SIFT and goodFeaturesToTrack
img_with_kp = cv.drawKeypoints(img1, kp1, None, flags=cv.DRAW_MATCHES_FLAGS_DRAW_RI
CH_KEYPOINTS)
plt.imshow(img_with_kp)
plt.show()
```



In [ ]: