### General imports

In [ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt

from PIL import Image
import numpy as np
import cv2
from skimage.metrics import mean_squared_error as mse
from skimage.metrics import structural_similarity as ssi
from scipy.signal import convolve2d as conv2
```

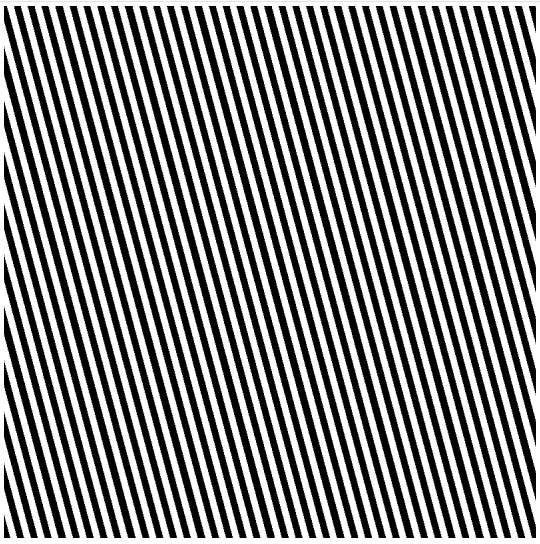
### Auxiliary functions

In [ ]:
```
def render_images(images):
    """Renders images

    Args:
        images: dictionary with images
    """

    # TODO: Develop function to render images. Make sure
    for img_name, imgs in images.items():
        # plt.figure()
        plt.imshow(imgs)
        plt.title(img_name)
        plt.show()

    pass
```

In [ ]:
```
def load_images(names, render=True):
    """Loads images from files and renders them, if neces

    Args:
        names: list of image files to load
        render: flag that defines if image should be ren

    Returns:
        Dictionary with loaded images
    """

    # TODO: Develop function to load images from files
    img_dic = {}

    for names_list in names:
        img_read = Image.open(names_list)
        img_read = img_read.convert('1')
        img_dic[names_list] = img_read

    images = render_images(img_dic)

    return images
```

In [ ]:
```
# TODO: Show generated images with periodic structure
# Total: 4 images
# 1st
Image.fromarray(make_pattern(10, 15, 1000))
```

Out[43]:



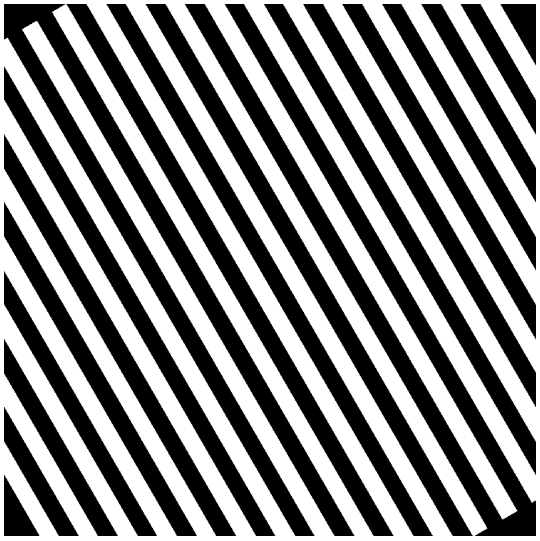
## 1 Fourier filtering

### 1.1 Preparation of dataset

In [ ]:
```
def make_pattern(dx, theta, shape):
    """Generates image with periodic structure (`stripes`

    Args:
        dx: stripe thickness
        theta: angle between stripes and x-axis
        shape: shape of resulting image

    Returns:
        Image with required pattern as numpy array
    """

    # TODO: Develop function to generate image with requ

    # Generate empty matrix & line strips
    gen_mat = np.zeros((shape,shape), np.uint8)
    lines = np.linspace(0, shape, shape//dx, dtype=int)

    for i in range(0, len(lines)-1, 2):
        gen_mat[:, lines[i]:lines[i+1]] = 255

    pad = 200
    img = np.pad(gen_mat, 100, mode='constant')
    img = np.array(Image.fromarray(img).rotate(theta))
    image = img[pad:-pad, pad:-pad]

    return image
```

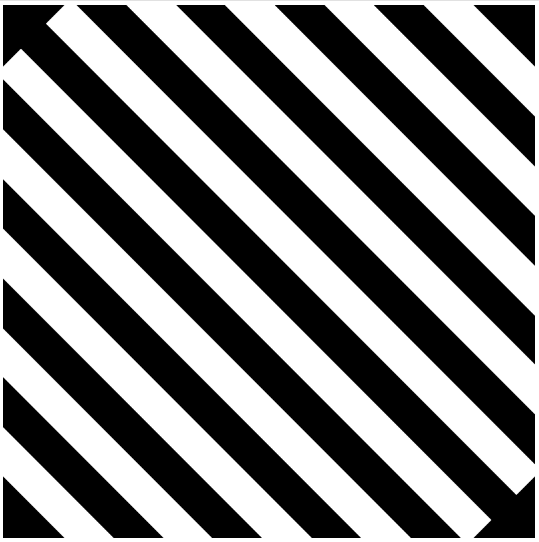
In [ ]:
```
# 2nd
Image.fromarray(make_pattern(25, 30, 1000))
```

Out[44]:



In [ ]:
```
# 3rd
Image.fromarray(make_pattern(50, 45, 1000))
```

Out[45]:

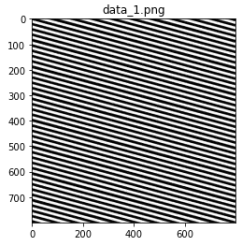


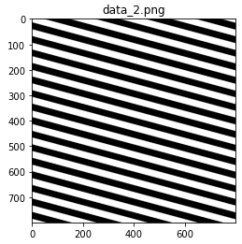
In [ ]:
```
# 4th
Image.fromarray(make_pattern(100, 90, 1000))
```

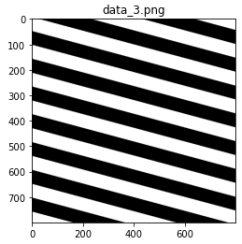
Out[46]:

In [ ]:
```
# TODO: Save generated images with periodic structure on
# Total: a dozen of images

dx = [10, 25, 50, 100, 5, 10, 15, 20, 30, 35, 40, 45]
theta = [15, 30, 45, 90, 125, 145, 180, 250, 270, 320, 6

for i in range(len(dx)):
    for j in range(len(theta)):
        img = Image.fromarray(make_pattern(dx[i], theta[
        img.save(f'data_{i+1}.png')
```

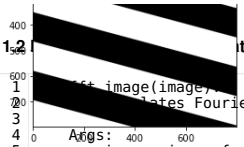
In [ ]:
```
# TODO: Load and render captured photos
# Total: 4 images

load_images(['data_1.png', 'data_2.png', 'data_3.png', '
```







**1.2 [illegible]taset**

In [ ]:
```
    [illegible]ft image(image):
        [illegible]lates Fourier Transform image

    Args:
        image: image for which Fourier Transform image is

    Returns:
        Fourier Transform image
    """

    # TODO: Develop function to calculate Fourier Transfo

    # Reading an image in default mode
    image = plt.imread(image)
    image = image[:, :, :3].mean(axis=2)  # Convert to g
    plt.set_cmap("gray")
    ft = np.fft.ifftshift(image)
    ft = np.fft.fft2(ft)
    result = np.fft.fftshift(ft)

    return result
```

In [ ]:
```
# TODO: Calculate Fourier Transform image of captured ph
# Total: 4 Fourier Transform images

print(fft_image('data/DSC_0229.JPG'))
Image.open('data/DSC_0229.JPG')
```

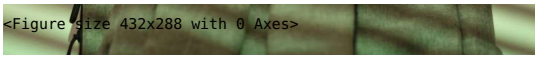
```
[[-128139.33333333+2.27373675e-12j   -6520.90476455+9.68318
055e+03j
    11850.95072477-3.16730217e+03j ...  -12728.59548276-1.8
8877828e+03j
    11850.95072477+3.16730217e+03j   -6520.90476455-9.68318
055e+03j]
 [  -3064.87126203+3.62661958e+04j   -3954.48010303-4.55682
057e+02j
    -7097.32670549-6.95142314e+03j ...    -641.38633336+1.2
3469000e+04j
    -1428.75885317-1.31614898e+04j  -15184.29840222+1.50421
411e+03j]
 [   8957.43627179-3.19055488e+03j   -6796.82846489-1.05410
870e+02j
    -5235.34866509+5.76569141e+03j ...    6071.37435483-3.6
5384614e+02j
   -12231.15865671-8.62153406e+02j   -6309.99297768+1.10903
912e+04j]
 ...
 [  -4734.01424578+5.83612563e+03j    6345.75265536-4.72080
393e+03j
     206.46375169-1.16310776e+04j ...    -477.9882519 +1.1
1448012e+03j
    5849.97799613-7.59236619e+03j    -489.51648344-8.21689
597e+03j]
 [   8957.43627179+3.19055488e+03j   -6309.99297768-1.10903
912e+04j
   -12231.15865671+8.62153406e+02j ...    7841.55443192+6.3
0544649e+03j
    -5235.34866509-5.76569141e+03j   -6796.82846489+1.05410
870e+02j]
 [  -3064.87126203-3.62661958e+04j  -15184.29840222-1.50421
411e+03j
    -1428.75885317+1.31614898e+04j ...    5977.73505648-1.1
8209233e+04j
    -7097.32670549+6.95142314e+03j   -3954.48010303+4.55682
057e+02j]]
```

Out[83]:


In [ ]:
```
print(fft_image('data/DSC_0230.JPG'))
Image.open('data/DSC_0230.JPG')
```

```
[[-113922.66666667-3.86535248e-12j   -4464.50466498+6.81584
827e+03j
    10326.36309279-3.40199623e+03j ...  -13940.72541512-3.7
5443489e+03j
    10326.36309279+3.40199623e+03j   -4464.50466498-6.81584
827e+03j]
 [  -8529.3850613 +3.32714908e+04j   -1733.43538408+3.14310
530e+03j
    -5594.26229391-5.46360023e+03j ...   -2735.78432696+1.1
8693194e+04j
     2009.49327538-1.37923007e+04j  -15383.50770135-2.04412
305e+03j]
 [   7700.06505983+2.64833445e+03j   -5359.09309315+7.42890
770e+03j
    -2300.13854149+8.65915463e+03j ...    6464.15543025+1.5
0789704e+03j
   -13281.16238174-1.15608319e+03j   -6227.17269346+7.65842
953e+03j]
 ...
 [  -1017.76786755+9.16064986e+02j    4211.02351091-5.22870
501e+03j
    -1814.76979744-1.37869245e+04j ...     309.23114891-1.9
9402239e+03j
     5352.96346108-6.50219649e+03j    4228.07367833-1.01771
745e+04j]
 [   7700.06505983-2.64833445e+03j   -6227.17269346-7.65842
953e+03j
   -13281.16238174+1.15608319e+03j ...    5764.23116356-2.9
5217711e+03j
    -2300.13854149-8.65915463e+03j   -5359.09309315-7.42890
770e+03j]
 [  -8529.3850613 -3.32714908e+04j  -15383.50770135+2.04412
305e+03j
     2009.49327538+1.37923007e+04j ...    -151.68610828-1.2
1297159e+04j
    -5594.26229391+5.46360023e+03j   -1733.43538408-3.14310
530e+03j]]
```

Out[84]:



In [ ]:
```
print(fft_image('data/DSC_0231.JPG'))
Image.open('data/DSC_0231.JPG')
```

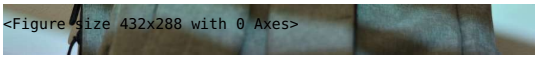
```
[[-1.21246000e+05-6.59383659e-12j -9.80330307e+03+5.9250523
9e+03j
   7.55165695e+03+7.83728035e+02j ... -1.49151082e+04-4.754
83945e+03j
   7.55165695e+03-7.83728035e+02j -9.80330307e+03-5.9250523
9e+03j]
 [-8.37901307e+03+3.05619213e+04j -1.12151406e+04+5.0086170
4e+03j
  -6.54351041e+03-6.97573470e+03j ... -8.31262865e+01+2.912
60664e+03j
  -2.85074701e+03-1.24424880e+04j -1.61614444e+04+1.6816180
7e+03j]
 [-1.23700731e+03+9.62551376e+01j -7.18244119e+03+1.0467811
4e+04j
  -8.11477273e+03+9.88357024e+03j ...  1.13645746e+03-9.514
22917e+03j
  -6.05531501e+03-2.69834331e+03j -8.72567383e+03+7.0830454
0e+03j]
 ...
 [-1.01209470e+04-8.12228064e+03j  3.08606693e+03-2.3575531
5e+03j
  -1.96857114e+03+3.08480041e+03j ... -3.37206124e+03-3.688
15827e+03j
   7.17291047e+03-4.15318044e+03j  1.73043442e+03-7.3843444
6e+03j]
 [-1.23700731e+03-9.62551376e+01j -8.72567383e+03-7.0830454
0e+03j
  -6.05531501e+03+2.69834331e+03j ...  4.41001694e+03+4.277
39302e+03j
  -8.11477273e+03-9.88357024e+03j -7.18244119e+03-1.0467811
4e+04j]
 [-8.37901307e+03-3.05619213e+04j -1.61614444e+04-1.6816180
7e+03j
  -2.85074701e+03+1.24424880e+04j ...  5.73433987e+03-1.303
89729e+04j
  -6.54351041e+03+6.97573470e+03j -1.12151406e+04-5.0086170
4e+03j]]
```

Out[85]:

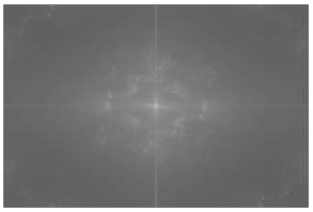
In [ ]:
```
print(fft_image('data/DSC_0232.JPG'))
Image.open('data/DSC_0232.JPG')
```

```
[[-135829.66666667-6.36646291e-12j  -10197.14748988+8.35865
054e+03j
      766.23890052-5.58874596e+02j ...  -15411.85107728+1.3
6177740e+03j
      766.23890052+5.58874596e+02j  -10197.14748988-8.35865
054e+03j]
 [ -14703.80577212+2.72753011e+04j  -12883.49083657+3.53506
258e+03j
    -7381.6564382 -3.81779654e+03j ...   -6843.21618054+1.8
2368801e+04j
    -8373.93509599-8.51082795e+03j  -19041.68404944-3.90389
440e+03j]
 [  -3945.67071819-3.21152838e+03j  -10007.58858312+9.81525
468e+03j
    -6628.85685425+7.51146477e+03j ...    6523.61362586+8.6
3506767e+03j
   -11985.43450268+2.29547133e+03j  -15718.16177855+1.36399
643e+04j]
 ...
 [ -11949.75542457+8.34790189e+02j   -8689.98555663-5.59177
573e+03j
    -4635.42682061-1.20825017e+04j ...     780.24120704-4.1
7277965e+03j
     3941.61204104-8.04741095e+03j    4394.53052215-7.59116
734e+03j]
 [  -3945.67071819+3.21152838e+03j  -15718.16177855-1.36399
643e+04j
   -11985.43450268-2.29547133e+03j ...    5445.74073496+3.4
5798787e+03j
    -6628.85685425-7.51146477e+03j  -10007.58858312-9.81525
468e+03j]
 [ -14703.80577212-2.72753011e+04j  -19041.68404944+3.90389
440e+03j
    -8373.93509599+8.51082795e+03j ...     989.15190382-1.1
3064032e+04j
    -7381.6564382 +3.81779654e+03j  -12883.49083656-3.53506
258e+03j]]
```

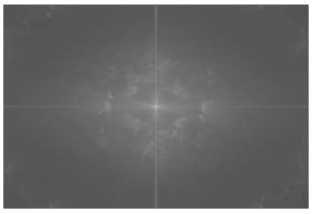
Out[86]:



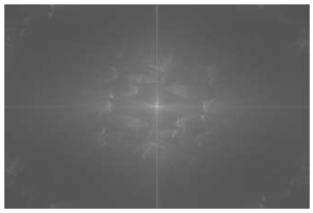
In [ ]:
```
# TODO: Display Fourier Transform image of captured photo
# Total: 4 images
ft = fft_image('data/DSC_0229.JPG')
plt.imshow(np.log(abs(ft)))
plt.axis("off")
plt.show()
```



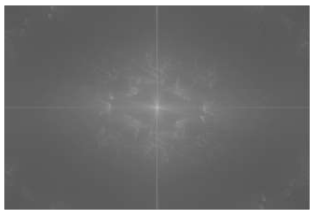
In [ ]:
```
ft = fft_image('data/DSC_0230.JPG')
plt.imshow(np.log(abs(ft)))
plt.axis("off")
plt.show()
```



In [ ]:
```
ft = fft_image('data/DSC_0231.JPG')
plt.imshow(np.log(abs(ft)))
plt.axis("off")
plt.show()
```



In [ ]:
```
ft = fft_image('data/DSC_0232.JPG')
plt.imshow(np.log(abs(ft)))
plt.axis("off")
plt.show()
```



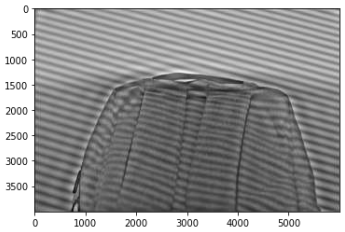
In [ ]:
```
# Differenct delta x and theta leads to the different br
```

*TODO* : Qualitatively describe your observations of the changes to the Fourier spectrum as a function of Δx and θ

In [ ]:
```
def fouirer_filter(source):
    """Filters image to get rid of `striped` illuminatio

    Args:
        source: original image with hardware overlayed `
        theta: angle between stripes and x-axis
        shape: shape of resulting image

    Returns:
        Filtered image
    """

    # TODO: Develop function to filter image
    image = plt.imread(source)
    image = image[:, :, :3].mean(axis=2)
    plt.set_cmap("gray")
    ft = np.fft.fft2(image)
    fft = np.fft.fftshift(ft)

    # Try to change this!!!
    pad = 16

    x_cut = fft.shape[0]//2 - pad, fft.shape[0]//2 + pad
    y_cut = fft.shape[1]//2 - pad, fft.shape[1]//2
    chunk = fft[x_cut[0]:x_cut[1], y_cut[0]:y_cut[1]]

    fft[x_cut[0]:x_cut[1], y_cut[0]:y_cut[1]] = 0

    x_cut = fft.shape[0]//2 - pad, fft.shape[0]//2 + pad
    y_cut = fft.shape[1]//2+2, fft.shape[1]//2 + pad
    chunk = fft[x_cut[0]:x_cut[1], y_cut[0]:y_cut[1]]
    fft[x_cut[0]:x_cut[1], y_cut[0]:y_cut[1]] = 0

    fourier = np.fft.ifft2(np.fft.ifftshift(fft))
    image = np.real(fourier)

    return image
```

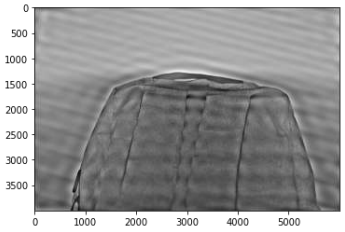
In [ ]:
```
# TODO: Test fouirer_filter() using captured photos
# Total: Render 4 filtered images

remove_1 = fouirer_filter('DSC_0229.JPG')
plt.imshow(remove_1, cmap = 'gray')
```

Out[3]: <matplotlib.image.AxesImage at 0x7f7ddbfe8a90>



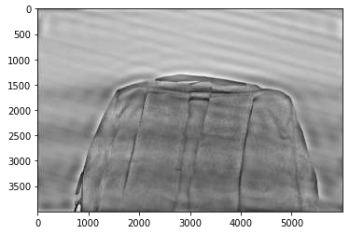

In [ ]:
```
remove_2 = fouirer_filter('DSC_0230.JPG')
plt.imshow(remove_2, cmap = 'gray')
```

Out[4]: <matplotlib.image.AxesImage at 0x7f7ddb2a5710>



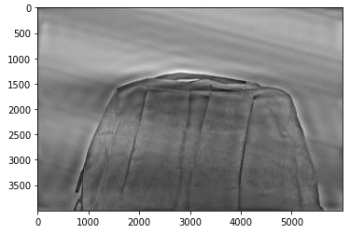

In [ ]:
```
remove_3 = fouirer_filter('DSC_0231.JPG')
plt.imshow(remove_3, cmap = 'gray')
```

Out[5]: <matplotlib.image.AxesImage at 0x7f7ddb2254d0>




In [ ]:
```
remove_4 = fouirer_filter('DSC_0232.JPG')
plt.imshow(remove_4, cmap = 'gray')
```

Out[6]: <matplotlib.image.AxesImage at 0x7f7ddb1a0190>



In [ ]:
```
# TODO: Compare filtered images to thier originals using
# Total: 4 comparisons

ref_1 = plt.imread('DSC_0229.JPG')
ref_1 = ref_1[:, :, :3].mean(axis=2)

ref_2 = plt.imread('DSC_0230.JPG')
ref_2 = ref_2[:, :, :3].mean(axis=2)

ref_3 = plt.imread('DSC_0231.JPG')
ref_3 = ref_3[:, :, :3].mean(axis=2)

ref_4 = plt.imread('DSC_0232.JPG')
ref_4 = ref_4[:, :, :3].mean(axis=2)

ssim_1 = ssim(ref_1, remove_1)
ssim_2 = ssim(ref_2, remove_2)
ssim_3 = ssim(ref_3, remove_3)
ssim_4 = ssim(ref_4, remove_4)

print('{0:<8}({1:<12}| {2:<12}): {3:.8f}'.format('SSIM',
print('{0:<8}({1:<12}| {2:<12}): {3:.8f}'.format('SSIM',
print('{0:<8}({1:<12}| {2:<12}): {3:.8f}'.format('SSIM',
print('{0:<8}({1:<12}| {2:<12}): {3:.8f}'.format('SSIM',
```

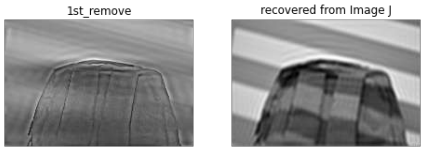
```
SSIM    (Original 1  | Filtered 1  ): 0.94804709
SSIM    (Original 2  | Filtered 2  ): 0.71529326
SSIM    (Original 3  | Filtered 3  ): 0.81875488
SSIM    (Original 4  | Filtered 4  ): 0.86953612
```

*TODO* : Qualitatively describe your observations and suggest a concept to eliminate the artifacts in the filtered images.

Ans: I found that the method that I use (remove only the center of FT) is not good enough. It would be great if can apply other removal function (e.g. sine) or remove other part of FT because the source code pattern has other parameter (e.g. theta). The result should be better but loose sharpness.

In [30]:
```
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(8, 5),

plt.gray()

fft_pic = Image.open("fft_pic.png")
fft_pic = fft_pic.convert('1')

ax[0].imshow(remove_4, cmap = 'gray')
ax[0].axis('off')
ax[0].set_title('1st_remove')

ax[1].imshow(fft_pic)
ax[1].axis('off')
ax[1].set_title('recovered from Image J')
```

Out[30]: Text(0.5, 1.0, 'recovered from Image J')




## 2 Bokeh deconvolution

### 2.1 Image recovery from software-originated Bokeh effect

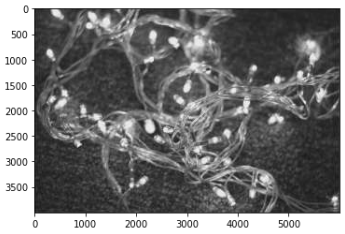
In [ ]:
```
# TODO: Load photo of decoration lights
# Total: 1 image

Image.open('DSC_0421.JPG')
```

Out[15]:

In [42]:
```
def make_synthetic_psf(shape):
    """Generates synthetic PSF of specified shape

    Args:
        shape: shape of synthetic PSF

    Returns:
        Synthetic PSF as numpy array
    """

    # TODO: Place code for synthetic PSF generation  int
    psf =  np.array([[0, 0, 1, 0, 0],
                     [0, 1, 1, 1, 0],
                     [1, 1, 1, 1, 1],
                     [0, 1, 1, 1, 0],
                     [0, 0, 1, 0, 0]], dtype="float64")/

    return psf
```

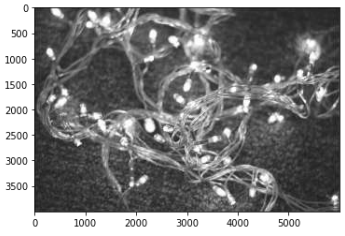
In [43]:
```
# TODO: Calculate convolution of original photo of decor
# TODO: Bokeh effect should be perceptible
# Total: 1 image

from skimage import restoration

bokeh_1 = Image.open("DSC_0421.JPG")
bokeh = bokeh_1.convert('1')
psf = make_synthetic_psf(200)
convo = conv2(bokeh, psf, 'same')

rng = np.random.default_rng()
convo += 0.1 * convo.std() * rng.standard_normal(convo.sh

plt.imshow(convo, cmap = 'gray')
```

Out[43]: <matplotlib.image.AxesImage at 0x7f7dc3dd1c90>

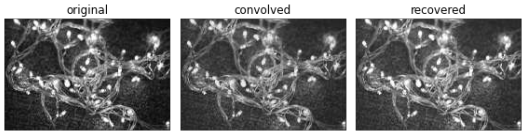



In [44]:
```
# TODO: Recover the original image from the convolved ima
# Total: 1 image

deconvolved, _ = restoration.unsupervised_wiener(convo, p
plt.imshow(deconvolved)
```

Out[44]: <matplotlib.image.AxesImage at 0x7f7dc3dc9690>



In [45]:
```
# TODO: Display the original image, the convolved Bokeh
# Total: a row of 3 images

fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(8, 5),

plt.gray()

orig = Image.open("DSC_0421.JPG")
orig = orig.convert('1')

ax[0].imshow(orig)
ax[0].axis('off')
ax[0].set_title('original')

ax[1].imshow(convo, cmap = 'gray')
ax[1].axis('off')
ax[1].set_title('convolved')

ax[2].imshow(deconvolved)
ax[2].axis('off')
ax[2].set_title('recovered')

fig.tight_layout()

plt.show()
```




*TODO* : Describe what the custom shape of the PSF did to the original image and what are the differences in the recovered and the original photos

Ans: The custom shape of psf is rhombus and when we apply to the original image, the picture is brightened. The difference between recovered and original images is sharpness. Recovered image leads to the loss in sharpness.

### 2.2 Image recovery from hardware-originated Bokeh effect

In [ ]:
```
# TODO: Load and render the captured photo of decoration
# Total: 1 image
Image.open('DSC_0422.JPG')
```

Out[17]:

In [ ]:
```
def recover_from_hw_Bokeh(image, psf):
    """Recovers clean image from image with hardware-orig

    Args:
        image: image with Bokeh-effect
        psf: PSF of known shape (may required scaling or

    Returns:
        Clean image
    """

    # TODO: Place code for recovery of the clean image o

    return recovered_image

#### Using imageJ ####
```

I find FT and select only the white circle inside rhombus. The result will be shown in the next cell (loss sharpness).

In [29]:
```
# TODO: Display the clean image side-by-side with origina
# Total: a row of 2 images
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(8, 5),

plt.gray()

bok_rec_ref = Image.open("DSC_0422.JPG")
bok_rec_ref = bok_rec_ref.convert('1')

bok_rec_rec = Image.open("recovered_1.png")
bok_rec_rec = bok_rec_rec.convert('1')


ax[0].imshow(bok_rec_ref)
ax[0].axis('off')
ax[0].set_title('original')

ax[1].imshow(bok_rec_rec)
ax[1].axis('off')
ax[1].set_title('recovered')
```

Out[29]: Text(0.5, 1.0, 'recovered')