averaging_methods

January 19, 2018

1 Image averaging methods

This notebook implements some methods to image noise reduce by averaging. The noise is made by:

- shifting image in axis x and y
- rotating image by its center
- adding a gaussian random noise
- downscale factor

Original image:

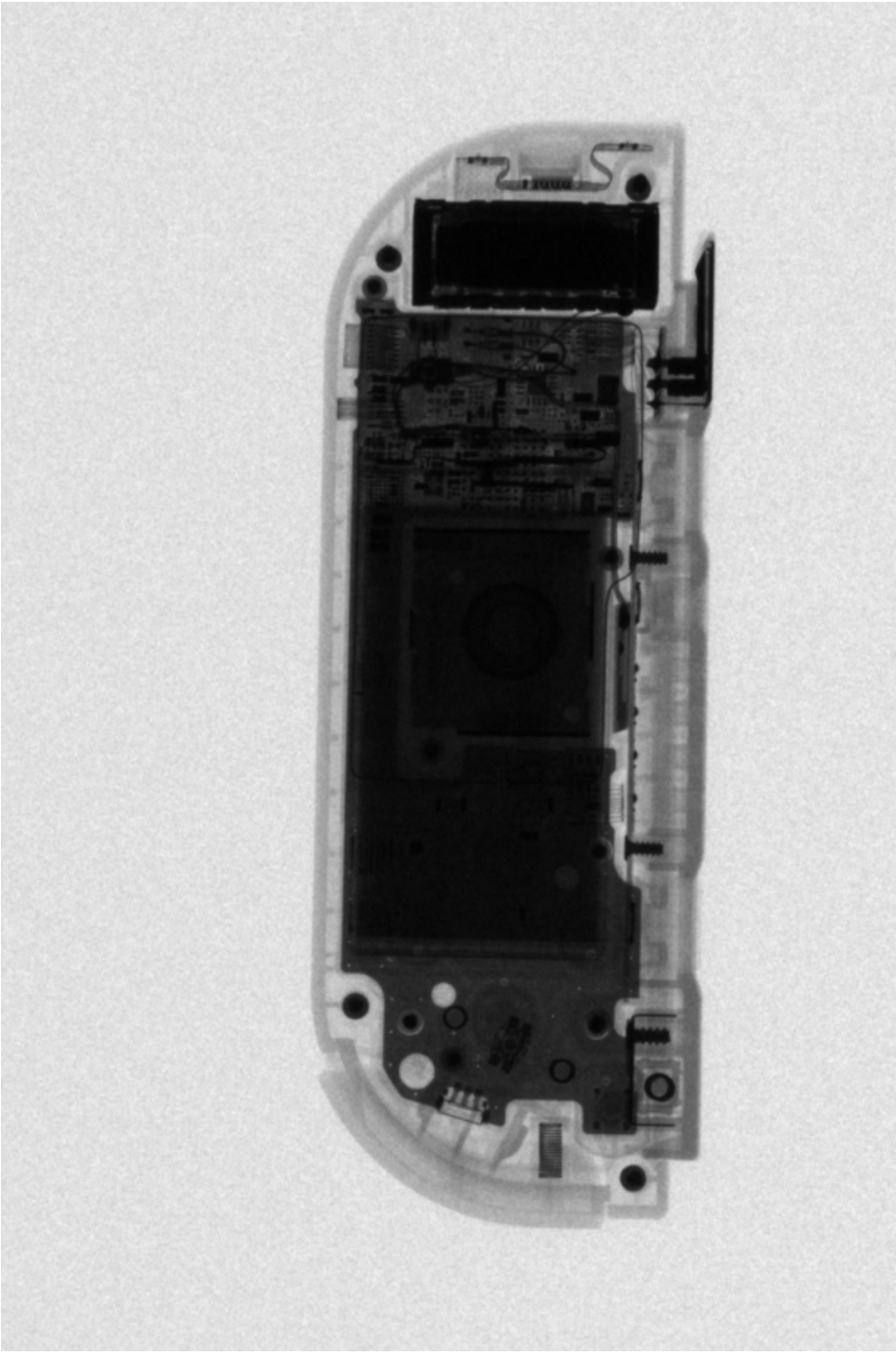
1.1 Results

Parameters:

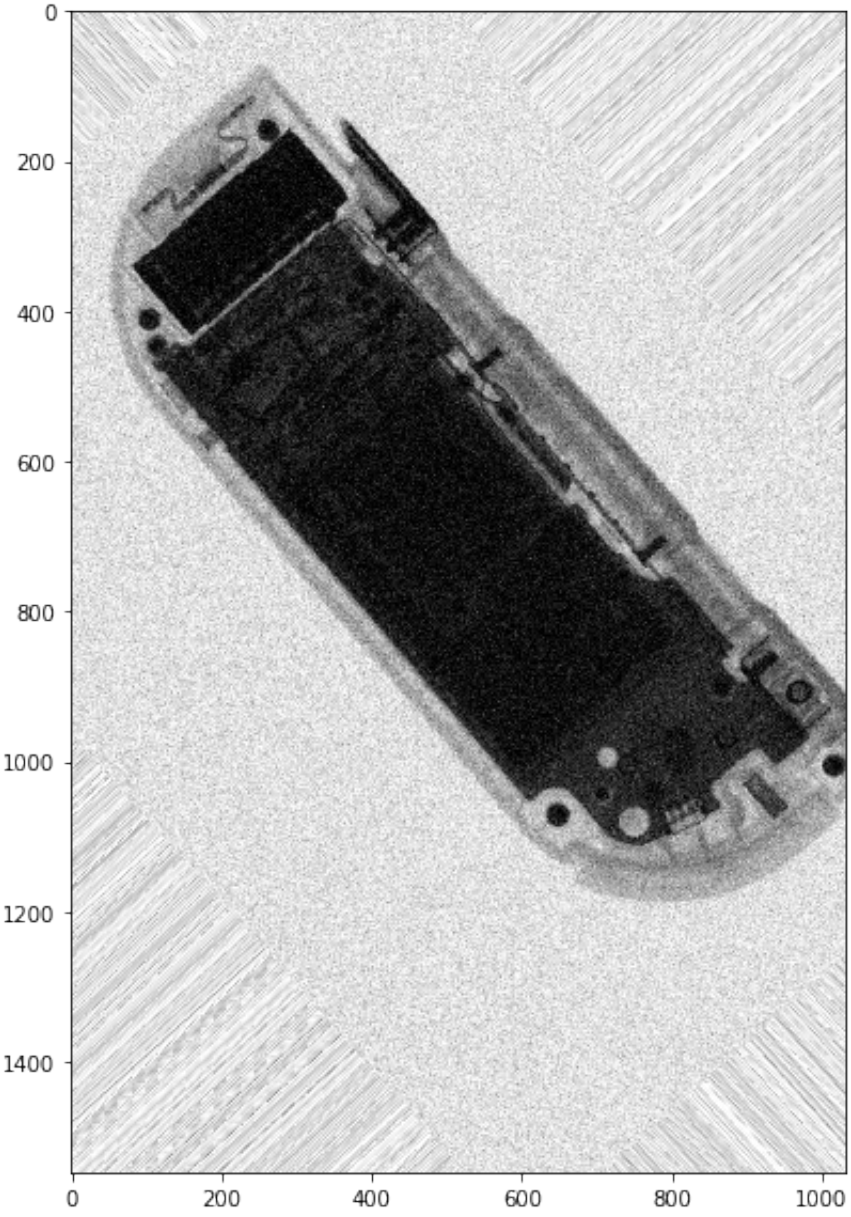
- random x and y deviation in range <-200, 200>
- random rotation deviation in range <-190, 190>
- gaussian noise with variance 0.01
- downscale factor 1

Noised image:
Noised images average:
Noised images average after images alignment:
Parameters:

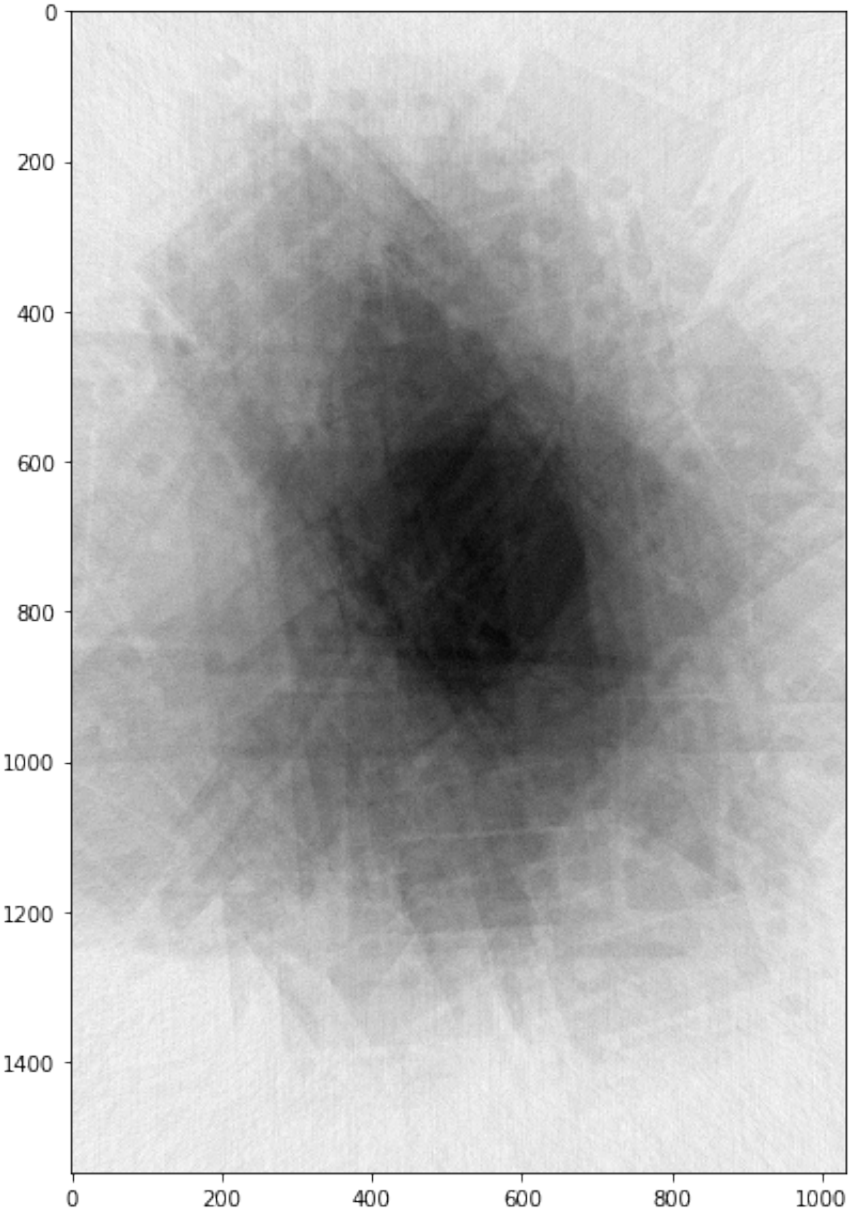
- random x and y deviation in range <-20, 20>
- random rotation deviation in range <-20, 20>
- gaussian noise with variance 0.01
- downscale factor 1



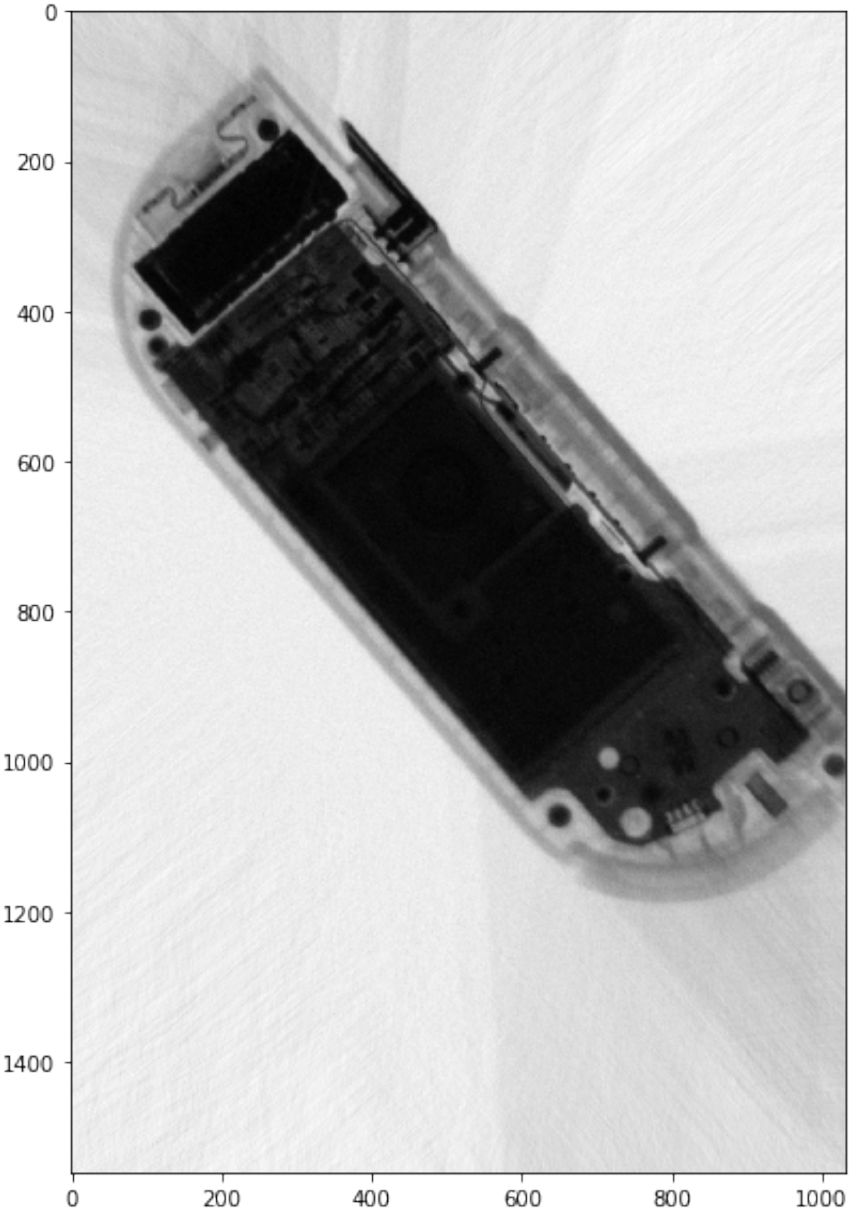
Original image



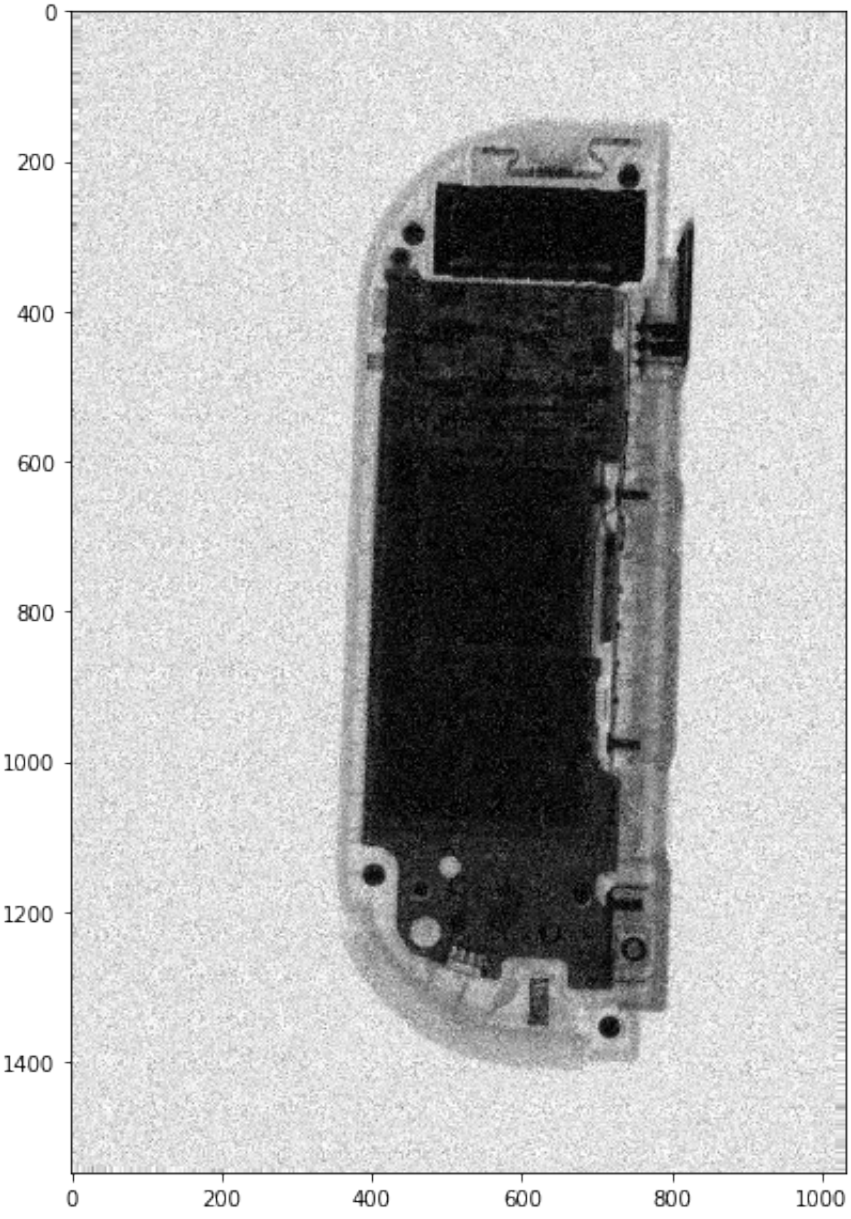
Noised image



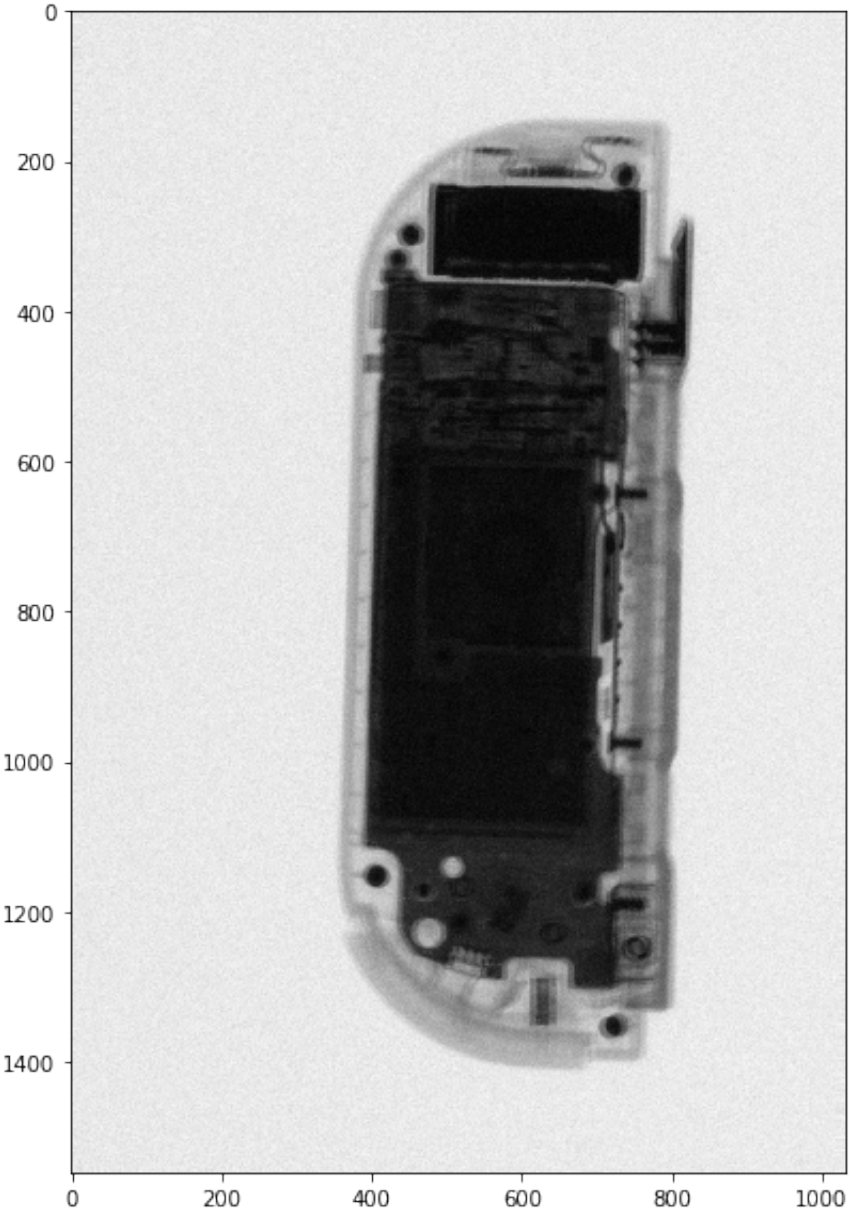
Noised images average



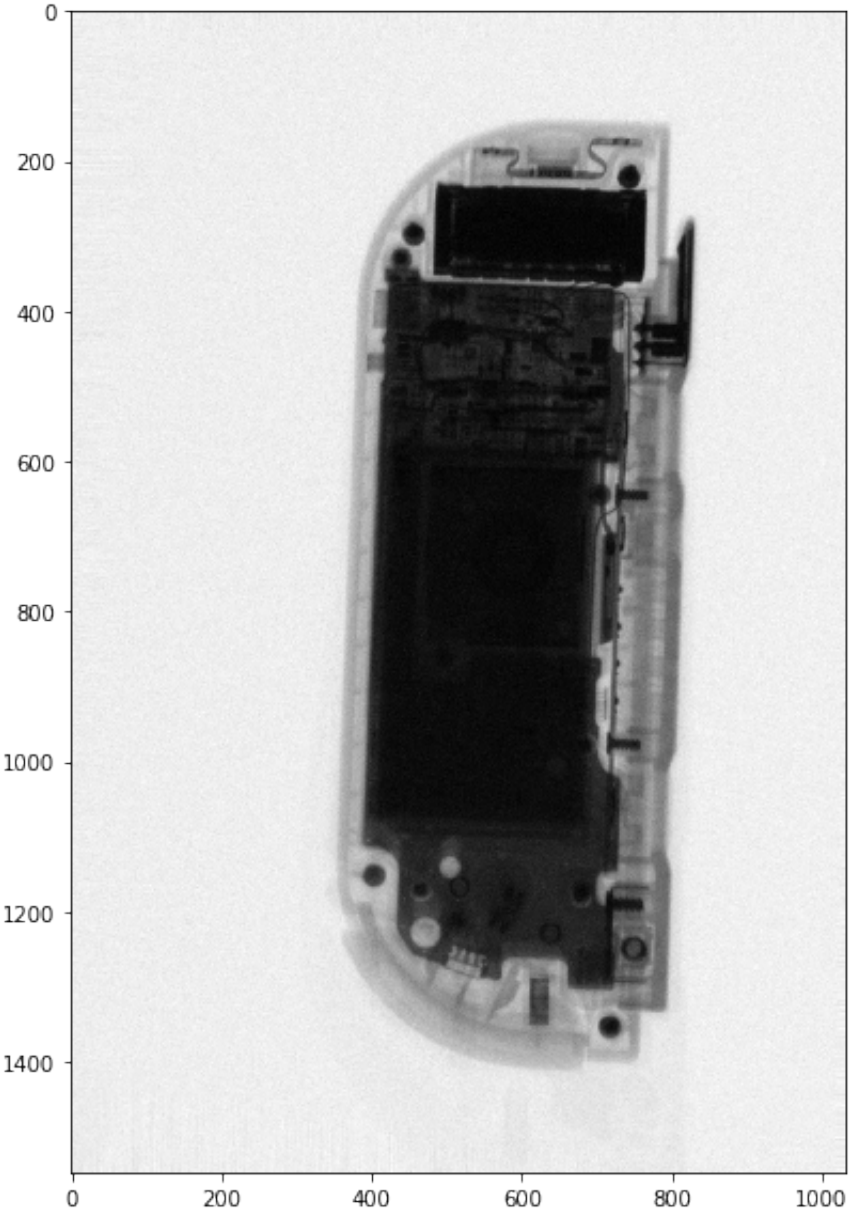
Noised image average after images alignment



Noised image



Noised images average



Noised image average after images alignment

Noised image:
Noised images average:
Noised images average after images alignment:

In [14]:
```
%matplotlib inline

        import glob
        import random

        import matplotlib.pyplot as plt
        from ipykernel.pylab.backend_inline import flush_figures

        from skimage import img_as_uint
        from skimage.io import imread
        from skimage.feature import ORB, match_descriptors, CENSURE
        from skimage.transform import downscale_local_mean, ProjectiveTransform
        from skimage.transform import warp, SimilarityTransform, rotate
        from skimage.measure import ransac
        from skimage.util import random_noise

        from ipywidgets import interact, interactive, fixed, interact_manual
        from IPython.display import display, clear_output
        import ipywidgets as widgets

        import numpy as np

        import cv2

        plt.rcParams['image.cmap'] = 'gray'
        plt.rcParams['image.interpolation'] = 'none'

        images = []
        shifted_images = []

        for filename in glob.glob('./static_image_series/*.tiff'):
            image = imread(filename, as_grey=True)
            images.append(image)


          File "<ipython-input-14-3f2bd524140a>", line 13
        ProjectiveTransform, warp,
        ^
    IndentationError: unexpected indent
```

2 Noisy data generation

Using sliders bellow you can generate noisy data which will be used in following algorithms.

```
In [10]: noise_slider = widgets.FloatSlider(min=0.0,
                                            max=1,
                                            step=0.05,
                                            value=0.1,
                                            continuous_update=False)
         x_max_noise_slider = widgets.IntSlider(min=1,
                                                max=500,
                                                step=1,
                                                value=1,
                                                continuous_update=False)
         y_max_noise_slider = widgets.IntSlider(min=1,
                                                max=500,
                                                step=1,
                                                value=1,
                                                continuous_update=False)
         rot_max_noise_slider = widgets.IntSlider(min=0,
                                                  max=359,
                                                  step=1,
                                                  value=1,
                                                  continuous_update=False)
         downscale_slider = widgets.IntSlider(min=1,
                                              max=20,
                                              step=1,
                                              value=1,
                                              continuous_update=False)
         gaussian_var_slider = widgets.FloatSlider(min=0.01,
                                                   max=1,
                                                   step=0.01,
                                                   value=0.01,
                                                   continuous_update=False)
         generate_button = widgets.Button(description='Generate')

         def downscale(imgs, scale):
             downscaled_images = []
             for image in imgs:
                 image = downscale_local_mean(image, (scale, scale)).astype('uint16')
                 downscaled_images.append(image)
             return downscaled_images

         def apply_noise(imgs, x_max_deviation, y_max_deviation, rot_max_deviation):
             noised_images = []
             for image in imgs:
                 random_noised_image = img_as_uint(
                     random_noise(image, mode='gaussian', var=gaussian_var_slider.value))
```

```
                tform = SimilarityTransform(
                    translation=(random.randrange(-x_max_deviation, x_max_deviation),
                                 random.randrange(-y_max_deviation, y_max_deviation)))
                shifted_image = warp(random_noised_image, tform, mode='edge')
                noised_images.append(rotate(shifted_image, random.randrange(
                    -rot_max_deviation, rot_max_deviation), mode='edge'))
            return noised_images

        def on_button_clicked(b):
            global shifted_images
            scaled_images = downscale(images, downscale_slider.value)
            shifted_images = apply_noise(scaled_images,
                                         x_max_noise_slider.value,
                                         y_max_noise_slider.value,
                                         rot_max_noise_slider.value)
            fig = plt.figure(figsize=(10, 10))
            plt.imshow(shifted_images[0])

        @interact(x_max_deviation=x_max_noise_slider)
        def x_noise(x_max_deviation):
            return x_max_deviation

        @interact(y_max_deviation=y_max_noise_slider)
        def y_noise(y_max_deviation):
            return y_max_deviation

        @interact(rotation_max_deviation=rot_max_noise_slider)
        def rot_noise(rotation_max_deviation):
            return rotation_max_deviation

        @interact(downscale_ratio=downscale_slider)
        def rot_noise(downscale_ratio):
            return downscale_ratio

        @interact(gaussian_variance=gaussian_var_slider)
        def x_noise(gaussian_variance):
            return gaussian_variance

        generate_button.on_click(on_button_clicked)

        display(generate_button)
```

```
A Jupyter Widget
```

```
A Jupyter Widget
```

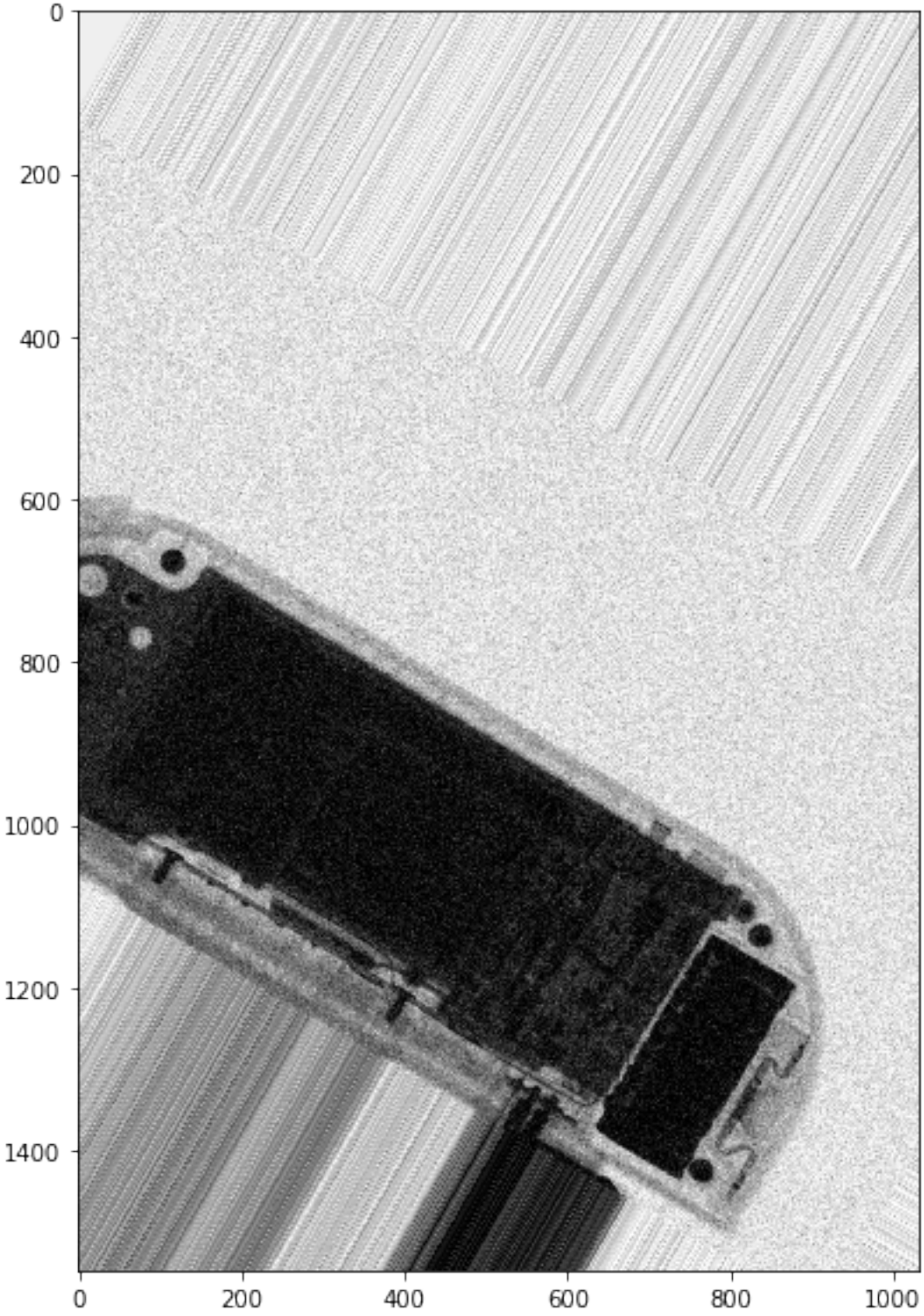
```
A Jupyter Widget
```

```
A Jupyter Widget
```

```
A Jupyter Widget
```

```
A Jupyter Widget
```

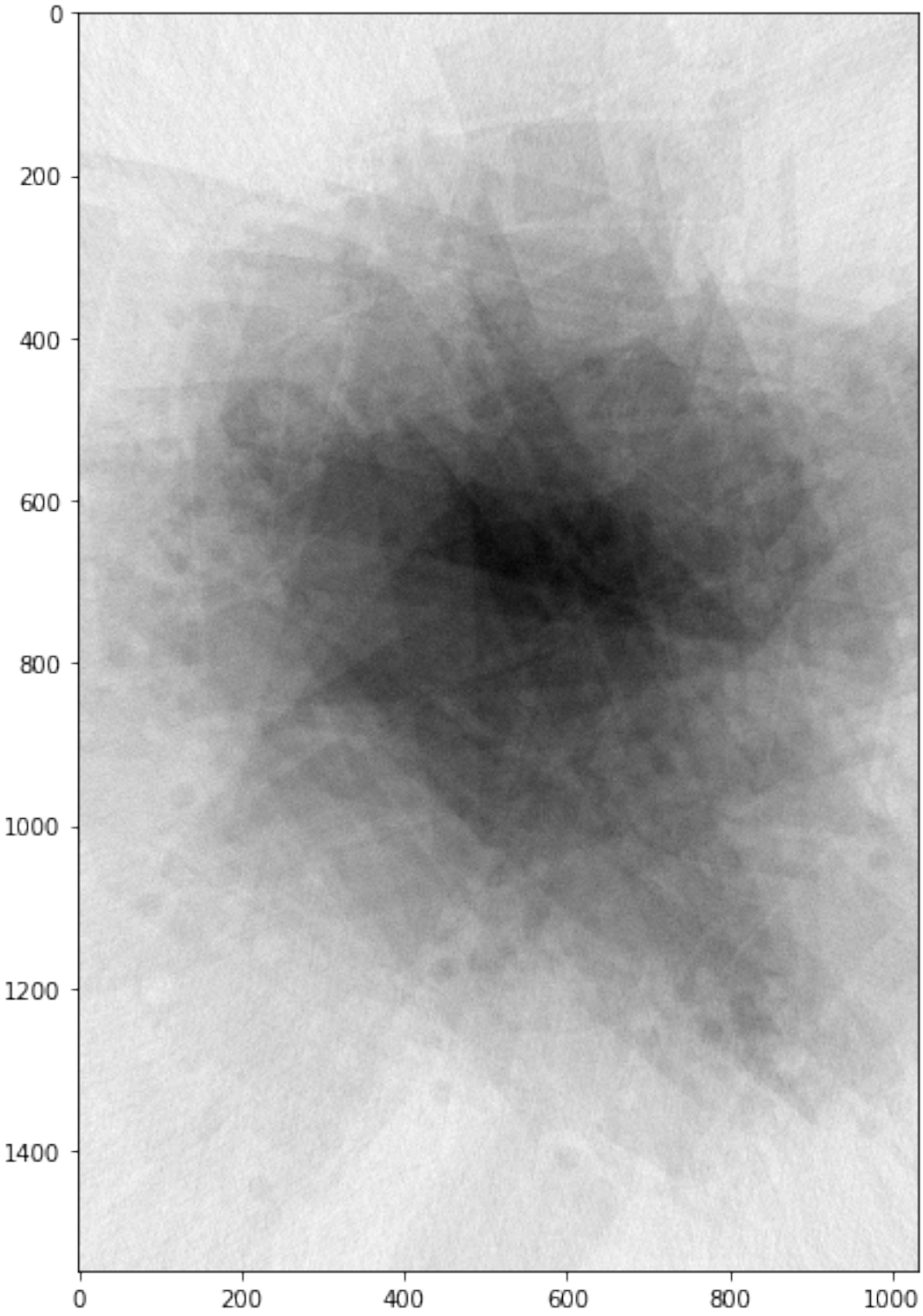
```
C:\Users\Petr\Anaconda3\lib\site-packages\skimage\util\dtype.py:122: UserWarning: Possible pre
  .format(dtypeobj_in, dtypeobj_out))
```


0
200
400
600
800
1000
1200
1400
0
200
400
600
800
1000

2.1 Basic average

This method just average all of the images.

```
In [11]: averaged_image = np.mean(shifted_images, axis=0)
         plt.figure(figsize=(10,10))
         plt.imshow(averaged_image)

Out[11]: <matplotlib.image.AxesImage at 0x20f24941390>
```


0
200
400
600
800
1000
1200
1400
0
200
400
600
800
1000

3 Hybrid Accumulation After Registration Algorithm

This algorythm is based on this paper: https://hal.archives-ouvertes.fr/hal-00510866/document

In my case I've implemented just keypoint and feature detection followed by image matchig and averaging of matched images. I've not implemented searching for the best homography according to the algorithm above.

3.1 Keypoint and feature detection and matching

Following code match keypoints of pairs of images (I0, Ij) where j = 1, ... n, where n is number of images * This topic is helpful: https://stackoverflow.com/questions/35877193/detecting-similar-points-between-two-pictures-and-then-overlaying-them-python * Also this implementation of panorama matchich is helpful: https://github.com/scikit-image/scikit-image-paper/blob/master/skimage/pano.txt

Code above finds keypoints of features and setup its binary descriptors. After keypoint serach it takes all of the keypoints and match them to first image. Result of this code is array of matched keypoints for all (I0, Ij) pairs

```
In [12]: def intersect_of_point_arrays(a, b):
             return np.array([x for x in set(tuple(x) for x in a) & set(tuple(x) for x in b)])

         orb = ORB(n_keypoints=1000, fast_threshold=0.05)

         orb.detect_and_extract(shifted_images[0])
         keypoints = orb.keypointsh

         descriptors = []
         keypoints = []
         matches_idx = []

         descriptors.append(orb.descriptors)
         keypoints.append(orb.keypoints)

         for image in shifted_images[1:]:
             orb.detect_and_extract(image)
             matches_idx.append(match_descriptors(descriptors[0], orb.descriptors))
             descriptors.append(orb.descriptors)
             keypoints.append(orb.keypoints)
```

4 Finding best homography points using RANSAC, warping and averaging

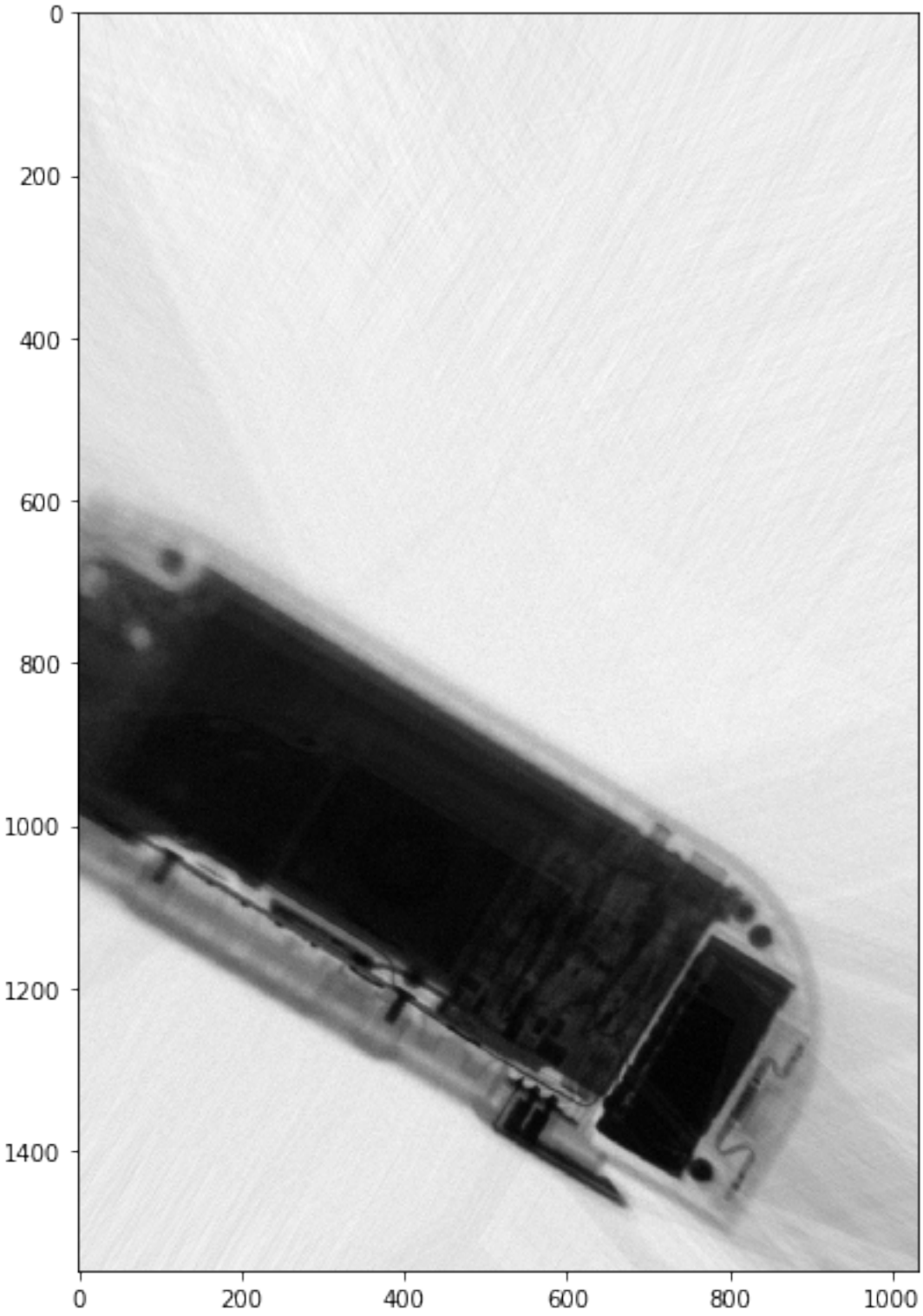
In this part I'm finding an homography points for images pairs

(I0, Ij) where j = 1, ..., n. n = number of samples

using RANSAC algorithm. Based on discovered points all of the images are transformed according to first image I0. After transformation all of the images are averaged.

In [13]:
```
number_of_agreed = 0
random_points = []

warped_images = []

for idx, image in enumerate(shifted_images[1:]):
    src = keypoints[idx + 1][matches_idx[idx][:, 1]][:, ::-1]
    dst = keypoints[0][matches_idx[idx][:, 0]][:, ::-1]

    model_robust, inliers = ransac((src, dst),
                                   ProjectiveTransform,
                                   min_samples=4,
                                   residual_threshold=2)

    # wrap image
    y, x = image.shape[:2]
    corners = np.array([[0, 0], [0, y], [x, 0], [x, y]])
    warped_corners = model_robust(corners)

    warped_images.append(warp(image, model_robust.inverse, mode='edge'))

plt.figure(figsize=(10, 10))
plt.imshow(np.mean(warped_images, axis=0))
```

Out[13]:
```
<matplotlib.image.AxesImage at 0x20f249eca58>
```


0
200
400
600
800
1000
1200
1400
0
200
400
600
800
1000