MRI_k-space-derived_details_edges

k-space based details/edges detection in MRI images with optional k-space based denoising and detail control

1 Introduction

Edge detection methods are fundamental to computer vision, as detecting edges is often the first stage of an image interpretation process. Boundary extraction and image segmentation are widely used in the field of biological and medical image analysis. For example, edge detection can be used as a pre-processing technique to identify points of interest in MRI (Magnetic Resonance Imaging) images.

Raw MRI data is stored in k-space. Every point in k-space represents a wave in the reconstructed image - a different spatial frequency at a different angle. The center of the k-space hosts low spatial frequencies (these data points in k-space give only large waves on the image - information about large areas of bright and dark, i.e. contrast information). The farther from the center the higher spatial frequency. This means that edges and detail information is located on the periphery of k-space. Reconstruction of an image from k-space with blank center results in the image of edges, details and (naturally for real-life imaging) noise. Having said that, it is worth remembering that simple cutting out the center of k-space can cause artifacts on the resulting image due to sudden signal zeroing. The solution to avoid the abovementioned artifacts is graduate dimming of the k-space towards the center.

This software uses previously published k- space denoising [1] and detail control (blurring) [2] methods. For more details see https://github.com/BeataWereszczynska/k-space_wght_msk_for_MRI_denoising/blob/main/Description.pdf and https://github.com/BeataWereszczynska/k-space_masking_for_MRI_denoising/blob/main/Description.pdf, respectively.

In this software, binary image of details/edges can be produced using one of three binarization methods: (1) automatic (utilising Otsu's binarization), (2) simple thresholding with single threshold value and (3) adaptive thresholding with the threshold value being a gaussian-weighted sum of the neighbourhood values minus the constant.

2 The sample data

2.1 The specimen

Fixation of the mouse specimen was performed by immersion in a 70% ethanol aqueous solution and soaking for over a month at room temperature (see https://doi.org/10.1016/j.fri.2021.200449 [3] for more details).

2.2 Magnetic Resonance Imaging

Spin echo MRI experiment was performed at 17°C on a 9.4 T horizontal bore Agilent MRI scanner using a 40 mm diameter Agilent millipede coil for transmission and reception. Six slices were acquired. Slice dimensions were 30 mm x 30 mm with slice thickness of 1 mm and data matrix size 512 x 512. To image the specimen, NMR signal of 1H nuclei present in methyl groups was used and other signals were saturated with single 90 degree pulse, as described in https://doi.org/10.1016/j.fri.2021.200449 [3].

3 The software

This section presents the code and describes how it works step by step.

The script (kspace_det_edg.py) contains 7 functions: 'import_kspace' for data import, 'wght_msk_kspace' for denoising [1], 'grad_mask_kspace' for detail control [2], 'FFT_2D' for image reconstruction, 'kspace_det_edg' for creating details/edges image, 'visualize' for creating summarising figure and 'main' which provides input parameters for the above functions, runs them and retrieves the results.

3.1 Importing the necessary libraries

In [1]:
```
import nmrglue as ng
import numpy as np
import cv2
from skimage import morphology
import matplotlib.pyplot as plt
```

3.2 Input parameters

3.2.1 Import parameters:

1. .fid folder location: **path** (string variable, e.g. "D:/sems_20190203_03.fid" or just "sems_20190203_03.fid" if located in one folder with the script),
2. total number of slices in the imaging experiment: **number_of_slices** (integer, e.g. 6)
3. selected slice number: **picked_slice** (integer, e.g. 4).

3.2.2 Denoising parameters:

1. switching denoising on/off: **denoise** (boolean value; 1 = on, 0 = off),
2. exponent in the signal weighting equation: **weight_power** (floating point number, e.g. 0.09),
3. restoring contrast on/off: **contrast** (boolean value; 1 = on , 0 = off).

3.2.3 Detail control parameters:

1. switching detail control on/off: **detail_contr** (boolean value: 1 = on , 0 = off),
2. radius for k-space masking in pixels: **r** (integer, e.g. 200).

3.2.4 Details/edges detection parameters:

1. smalest radius for k-space masking in pixels: **radius_min** (integer, e.g. 4),
2. largest radius for k-space masking in pixels: **radius_max** (integer, e.g. 45),
3. step betwen the two above values: **radius_step** (integer, e.g. 1),
4. thresholding option: **threshold** (tuple: (type, value), e.g. ('auto',), ('manual', 36), ('adaptive', -18))

- 'auto' thresholding uses Otsu's binarization and does not require any value to be provided,
- 'manual' thresholding uses basic thresholding and requires explicit threshold value (integer from 0 to 255),
- in 'adaptive' thresholding the threshold value is a gaussian-weighted sum of the neighbourhood values minus the constant C (integer), which has to be specyfied (in this case the size of the neighbourhood area is a fixed value (= 31) optimized for the experimental data in my reach; it can be amended in line 139 of the script).

3.3 Data import: 'import_kspace' function

The function is suitable for Agilent FID files.

Imports data from .fid folder specified in **path**. Single .fid file contains raw data for the whole imaging experiment. The function needs **number_of_slices** to downsample the data to extract the k-space corresponding to the **picked_slice**.

In [2]:
```
def import_kspace(path, number_of_slices, picked_slice):
    """
    Imports k-space corresponding to the picked slice from Agilent FID data.
    Input:
        .fid folder location: path [str],
        total number of slices in the MRI experiment: number_of_slices [int],
        selected slice number: picked_slice [int],
    """

    # import k-space data
    echoes = ng.agilent.read(dir=path)[1]
    kspace = echoes[picked_slice - 1 : echoes.shape[0] : number_of_slices, :]  # downsampling to one slice

    # return data
    return kspace
```

3.4 Denoising: 'wght_msk_kspace' function [1]

If **contrast** is set to 0, each element of the k-space is multiplied by its absolute value to the power of **weight_power** value.

If **contrast** is set to 1, each element of the k-space is multiplied by its absolute value to the power of **weight_power** value times 3. Subsequently, contrast is restored by finding k-space values with absolute value greater than absolute value of the maximal k-space value devided by 6. Those big values are divided by their absolute value to the power of **weight_power** value.

Next, all the k-space values outside the circle inscribed in the k-space are changed to 0.

In [3]:
```
def wght_msk_kspace(kspace, weight_power, contrast):
    """
    k-space weighting and masking for denoising of MRI image without blurring or losing contrast,
    as well as for brightening of the objects in the image with simultaneous noise reduction.
    Input:
        k-space data: kspace [array]
        exponent in the signal weighting equation: weight_power[float]
        restoring contrast: contrast [bool] (1 for denoising, 0 for brightening).

    From https://github.com/BeataWereszczynska/k-space_wght_msk_for_MRI_denoising/blob/main/wght_msk_kspace.py
    """

    # k-space weighting
    if contrast:
        kspace = kspace * np.power(abs(kspace), 3*weight_power)
        # contrasting
        a = kspace[abs(kspace) > abs(np.max(kspace))/6]
        a = a / np.power(abs(a), weight_power)
        kspace[abs(kspace) > abs(np.max(kspace))/6] = a
        del a
    else:
        kspace = kspace * np.power(abs(kspace), weight_power)
    del contrast, weight_power

    # k-space masking
    r = int(kspace.shape[0]/2)
    mask = np.zeros(shape=kspace.shape)
    cv2.circle(img=mask, center=(r,r), radius = r, color =(1,0,0), thickness=-1)
    kspace = np.multiply(kspace, mask)

    # return data
    return kspace
```

3.5 Detail control: 'grad_mask_kspace' function [2]

Creates masked version of k-space by multiplying it by a mask gradually dimming the edges outside of the given radius **r**.

In [4]:
```
def grad_mask_kspace(kspace, r):
```

```
    """
    Graduate k-space masking for MRI image blurring.
    Input:
        k-space data: kspace [array],
        radius for k-space masking in pixels: r [int].

    From https://github.com/BeataWereszczynska/k-space_masking_for_MRI_denoising/blob/main/grad_mask_kspace.py
    """

    # graduate k-space masking
    mask_denoise = np.zeros(shape=kspace.shape)
    mask = np.zeros(shape=kspace.shape)
    for value in range(r, r+15, 2):
        cv2.circle(img=mask, center=(int(kspace.shape[0]/2),int(kspace.shape[1]/2)),
                   radius = value, color =(1,0,0), thickness=-1)
        mask_denoise = mask_denoise + mask
    kspace = np.multiply(kspace, mask_denoise)

    # return data
    return kspace
```

3.6 MRI image reconstruction: 'FFT_2D' function

Reconstructs MRI image from k-space using 2-dimensional Fast Fourier Transform (2D FFT).

In [5]:

```
def FFT_2D(kspace):
    """
    Performs two-dimentional Fast Fourier Transform on k-space to reconstruct MRI image
    (for Agilent FID data).
    """

    # reconstruct MRI image
    ft = np.fft.fft2(kspace)              # 2D FFT
    ft = np.fft.fftshift(ft)              # fixing problem with corner being center of the image
    ft = np.transpose(np.flip(ft, (1,0))) # matching geometry with VnmrJ-calculated image (still a bit shifted)

    # return data
    return ft
```

3.7 Details/edges detection: 'kspace_det_edg' function

The function starts with creating masked version of k-space by multiplying it by a mask gradually dimming the center of the k-space in accordance with **radius_min**, **radius_max** and **radius_step** parameters. **radius_min** defines the circle in the k-space center that will be zeroed out. **radius_max** defines the furthest points of k-space impacted by the graduate dimming. **radius_step** defines distance in pixels from one dimming weight to another. The value of 1 results in the smoothest dimming (weight of dimming changing at each radius).

Please notice that **radius_step** value impacts the actual **radius_max** value, e.g. if **radius_min** = 4, **radius_max** = 40 and **radius_step** = 10, the actual **radius_max** value is 34 (4 + 10 + 10 + 10) as it is the last value within the chosen range.

In the next step the details/edges image is reconstructed by 'FFT_2D' function. The image is a matrix of complex numbers, so absolute values of pixels are calculated and the image gets normalized.

Binary image of the details/edges is produced using one of three binarization methods (in accordance with **threshold** parameter) and gets denoised by removing objects smaller than 3 pixels.

In [6]:

```
def kspace_det_edg(kspace, radius_min, radius_max, radius_step, threshold):
    """
    k-space based details/edges detection in MRI images with optional k-space based denoising
    and detail control.
    Input:
        k-space data: kspace[array],
        denoising option: denoise [tuple (1=on 0=off [bool], weight_power [float], contrast [bool])
        radii for k-space masking in pixels: radii [tuple (min [int], max [int], step [int])],
        detail control option: detail_contr [tuple (1=on 0=off [bool], radius in pixels [int])]
        threshold option: threshold [tuple (type [str], value [int])].
    Threshold options are: ('auto', ), ('manual', threshold value), ('adaptive', C value)
    """

    # masking k-space for creating an image of k-space-defined details/edges
    masks = np.zeros(shape=kspace.shape)
    mask = np.ones(shape=kspace.shape)
    for value in range(radius_min, radius_max+1, radius_step):
        cv2.circle(img=mask, center=(int(kspace.shape[0]/2),int(kspace.shape[1]/2)),
                   radius = value, color =(0,0,0), thickness=-1)
        masks = masks + mask
    kspace = np.multiply(kspace, masks)

    # reconstructing the image of k-space-defined details/edges
    ft2 = FFT_2D(kspace)
    del kspace

    # normalizing the image of k-space-defined details/edges (0-255)
    ft2 = cv2.normalize(abs(ft2), None, 0, 255, cv2.NORM_MINMAX, dtype=cv2.CV_8U)

    # k-space-defined details/edges to binary image
    if threshold[0] == 'adaptive':
        im_bw = cv2.adaptiveThreshold(ft2, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                      cv2.THRESH_BINARY, 31, threshold[1])
```

```
    elif threshold[0] == 'manual':
        im_bw = cv2.threshold(ft2, threshold[1], 255, cv2.THRESH_BINARY)[1]
    elif threshold[0] == 'auto':
        im_bw = cv2.threshold(ft2,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]
    else:
        print("Threshold options are: ('auto', ), ('manual', threshold value), ('adaptive', C value).")
        im_bw = np.zeros(ft2.shape)

    del threshold

    # denoise the binary image (remove small objects)
    im_bw = morphology.remove_small_objects(im_bw != 0, min_size=3, in_place=True, connectivity=2)

    # return data
    return ft2, im_bw
```

3.8 Visualization of the results: 'visualize' function

Creates summarising figure showing: the MRI image, the image of k-space-derived details/edges and the binary image of details/edges.

In [7]:

```
def visualize(ft1, ft2, im_bw):
    """
    Creates summarising figure.
    """

    plt.rcParams['figure.dpi'] = 1200
    plt.subplot(131)
    plt.title('MRI image', fontdict = {'fontsize' : 8}), plt.axis('off')
    plt.imshow(abs(ft1), cmap=plt.get_cmap('gray'))
    plt.subplot(132)
    plt.title('k-space-derived details/edges', fontdict = {'fontsize' : 8}), plt.axis('off')
    plt.imshow(ft2, cmap=plt.get_cmap('gray'))
    plt.subplot(133)
    plt.title('binary image of details/edges', fontdict = {'fontsize' : 8}), plt.axis('off')
    plt.imshow(im_bw, cmap=plt.get_cmap('gray'))
    plt.tight_layout(pad=0, w_pad=0.2, h_pad=1.0)
    plt.show()
```

3.9 Making all the above work together: 'main' function

Provides the input parameters for the above functions, runs them accordingly to **denoise** and **detail_contr** parameters and retrieves the results. It also creates global variables for the results to be available after the run completion.

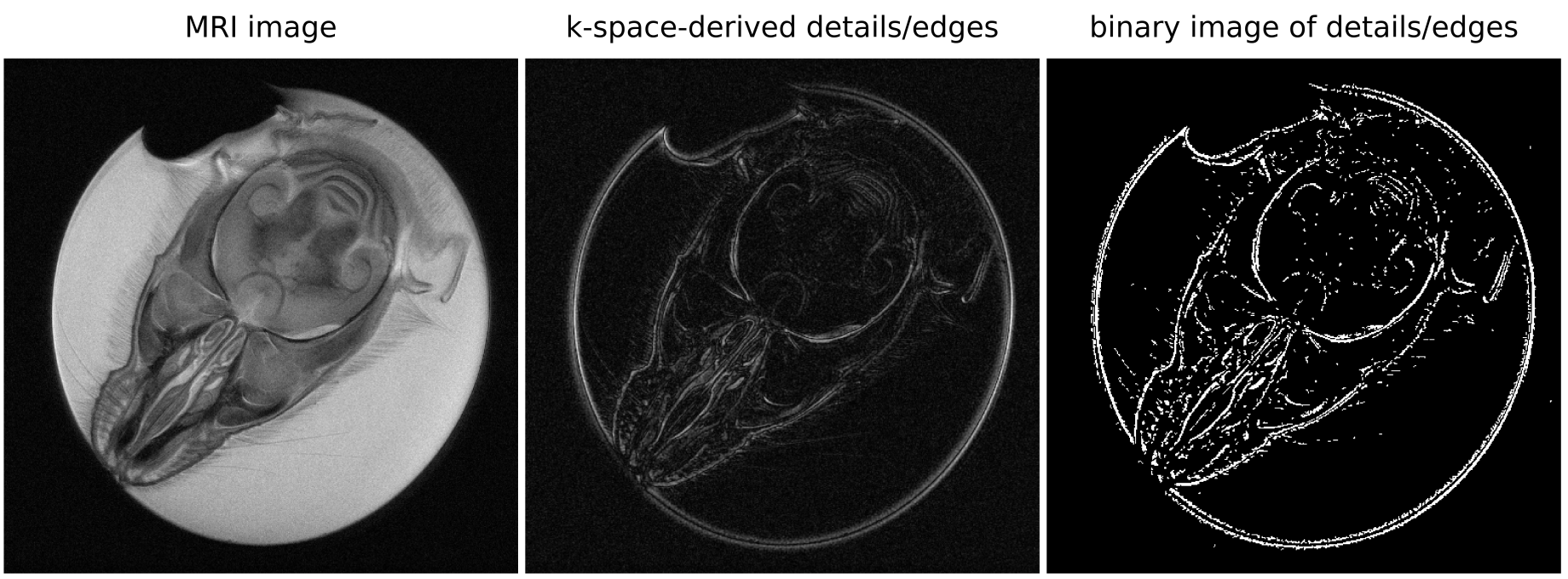
In [8]:

```
def main():

    # import parameters
    path = 'sems_20190203_03.fid'         # .fid folder location [str]
    number_of_slices = 6                  # total number of slices in the MRI experiment [int]
    picked_slice = 4                      # selected slice number [int]

    # denoising parameters
    denoise = 1                           # 1=on 0=off [bool]
    weight_power = 0.02                   # exponent in the signal weighting equation [float]
    contrast = 0                          # restoring contrast: 1=on 0=off [bool]

    # detail control parameters
    detail_contr = 1                      # 1=on 0=off [bool]
    detail_r = 200                        # radius for graduate masking in pixels [int]

    # details/edges detection parameters
    radius_min = 4                        # smalest radius for k-space masking [int]
    radius_max = 40                       # largest radius for k-space masking [int]
    radius_step = 1                       # step betwen the two above values [int]
    threshold = ('adaptive', -18)         # threshold option [tuple (type, value)]
    # threshold options: ('auto', ), ('manual', threshold value), ('adaptive', C value)

    # import
    kspace = import_kspace(path, number_of_slices, picked_slice)
    del path, number_of_slices, picked_slice

    # reconstructing the original MRI image
    ft1 = FFT_2D(kspace)

    # denoising
    if denoise:
        kspace = wght_msk_kspace(kspace, weight_power, contrast)
    del weight_power, contrast, denoise

    # detail control
    if detail_contr:
        kspace = grad_mask_kspace(kspace, detail_r)
    del detail_r, detail_contr

    # creating the image of details/edges and its binary version
    ft2, im_bw = kspace_det_edg(kspace, radius_min, radius_max, radius_step, threshold)
    del kspace

    # visualization
    visualize(ft1, ft2, im_bw)

    # creating global variables to be available after the run completion
```

```
    global MRI_complex_img
    MRI_complex_img = ft1
    global details_img
    details_img = ft2
    global binary_details
    binary_details = im_bw
```

3.10 The end of the script and sample result

In [10]:
```
if __name__ == "__main__":
    main()
```



4 Customizable results

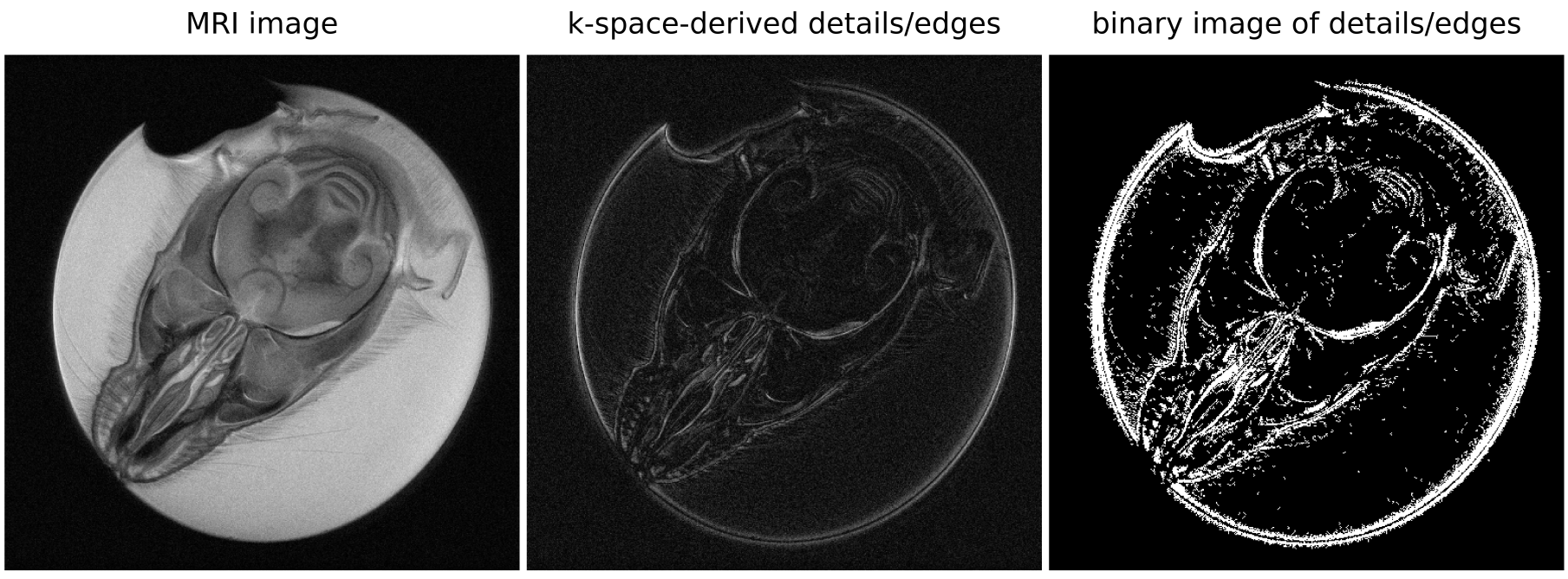
Different sets of input parameters will produce different results. It is important to understand the impact of the key input parameters.

4.1 Radii for details/edges detection

The easiest way to start is to set denoising and detail control off, choose auto thresholding and try different values for **radius_min** and **radius_max** with **radius_step** = 1. Then the **radius_step** can be changed to slightly higher value to speed up calculations (if the effect of **radius_step** > 1 is acceptable).

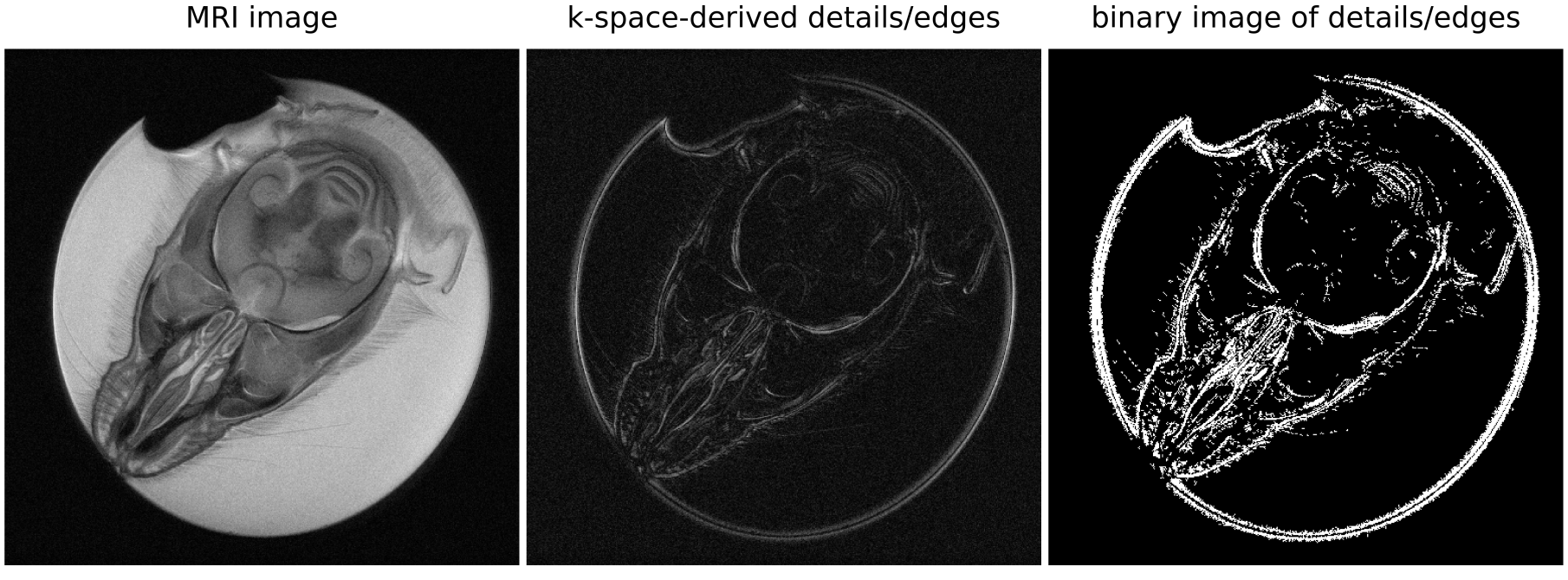
4.1.1 Too small radius_min (and optimal radius_max)

Not enough of k-space center is zeroed out - visible residual image shading (from low spatial frequencies).



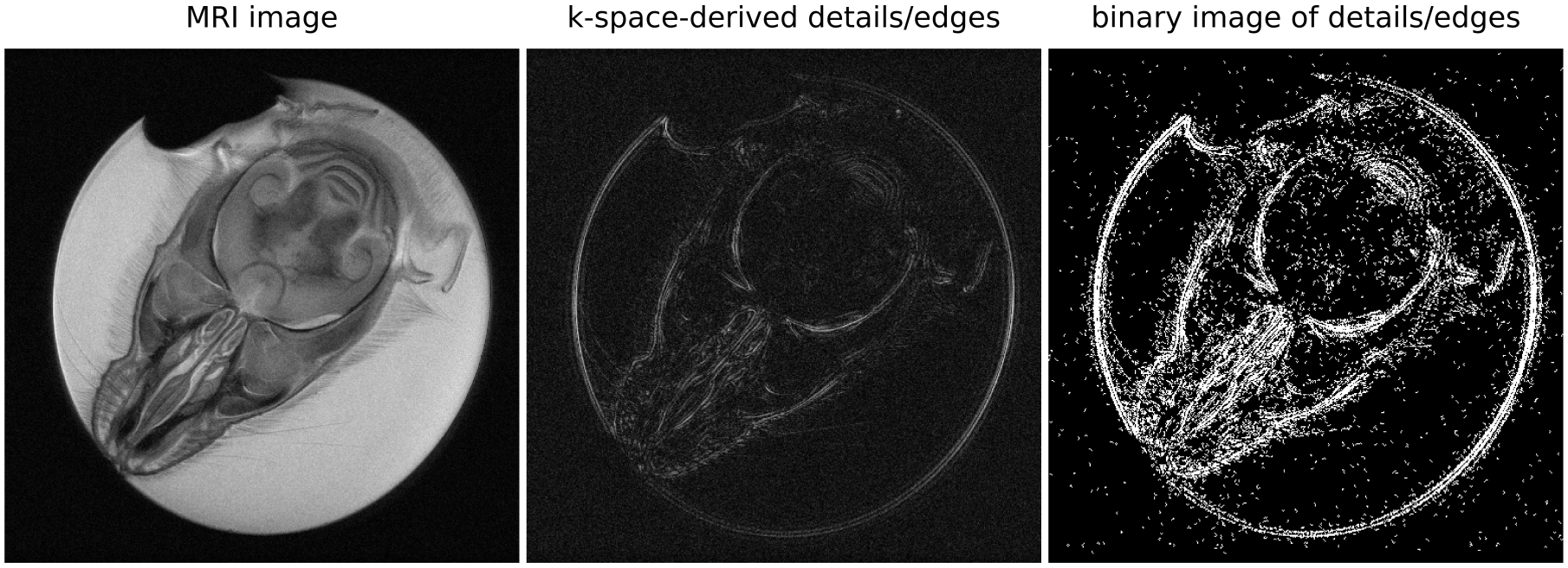
4.1.2 Optimal radius_min (and optimal radius_max)

No image shading or excess noise.



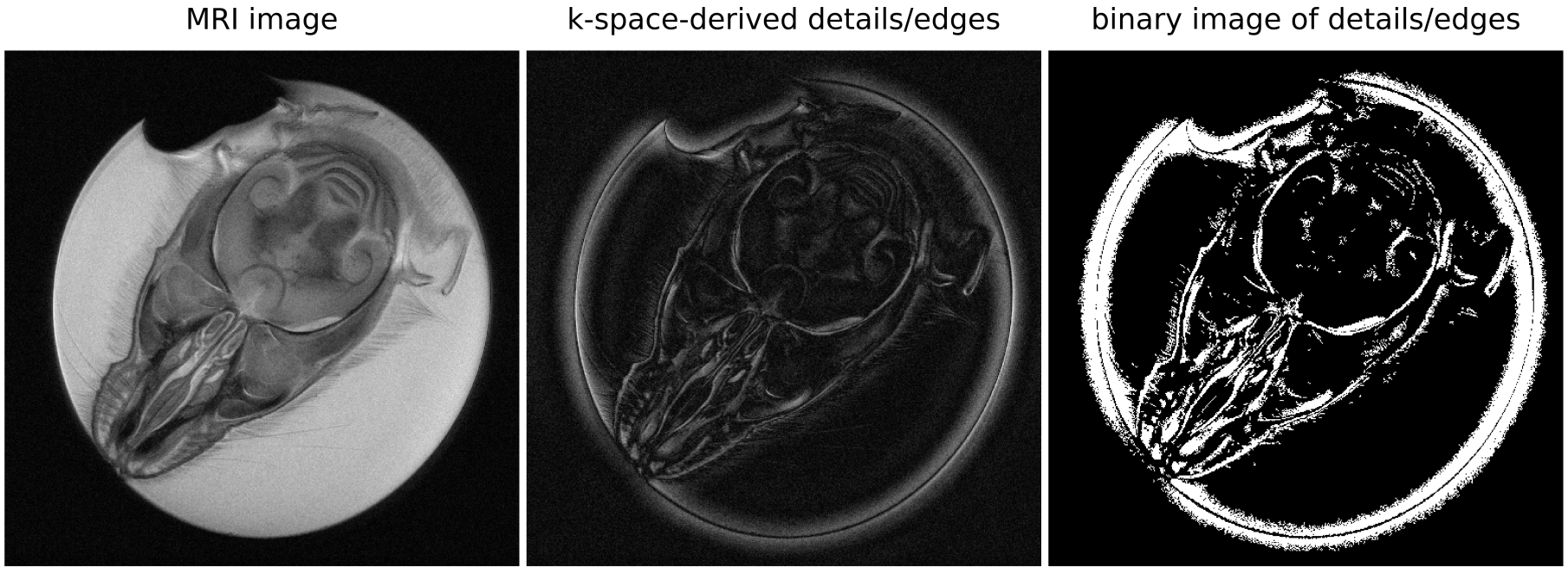
4.1.3 Too large radius_min (and optimal radius_max)

Too much zeroed out signals resulting in low signal-to-noise ratio (SNR).



4.1.4 Too small radius_max (and optimal radius_min)

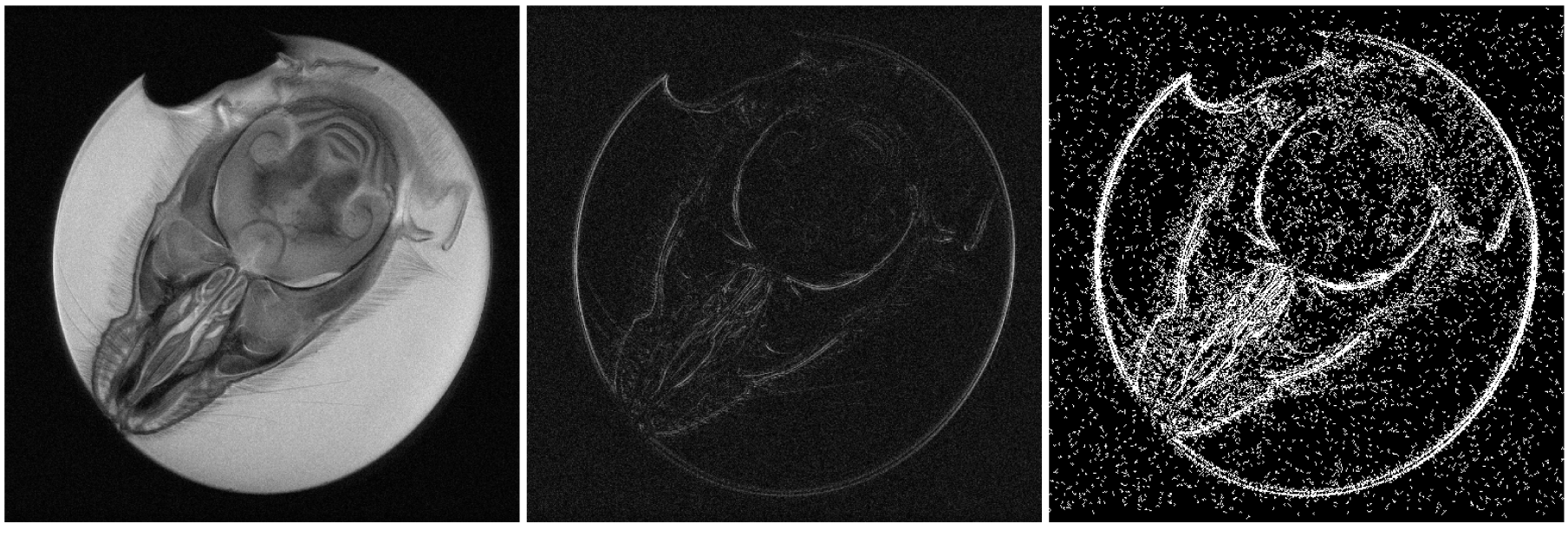
Not enough of k-space is covered by the masking - visible residual image shading (from medium spatial frequencies).



4.1.5 Too large radius_max (and optimal radius_min)

Too much k-space is covered by the masking - reducing too many signals results in noticeable SNR decrease.

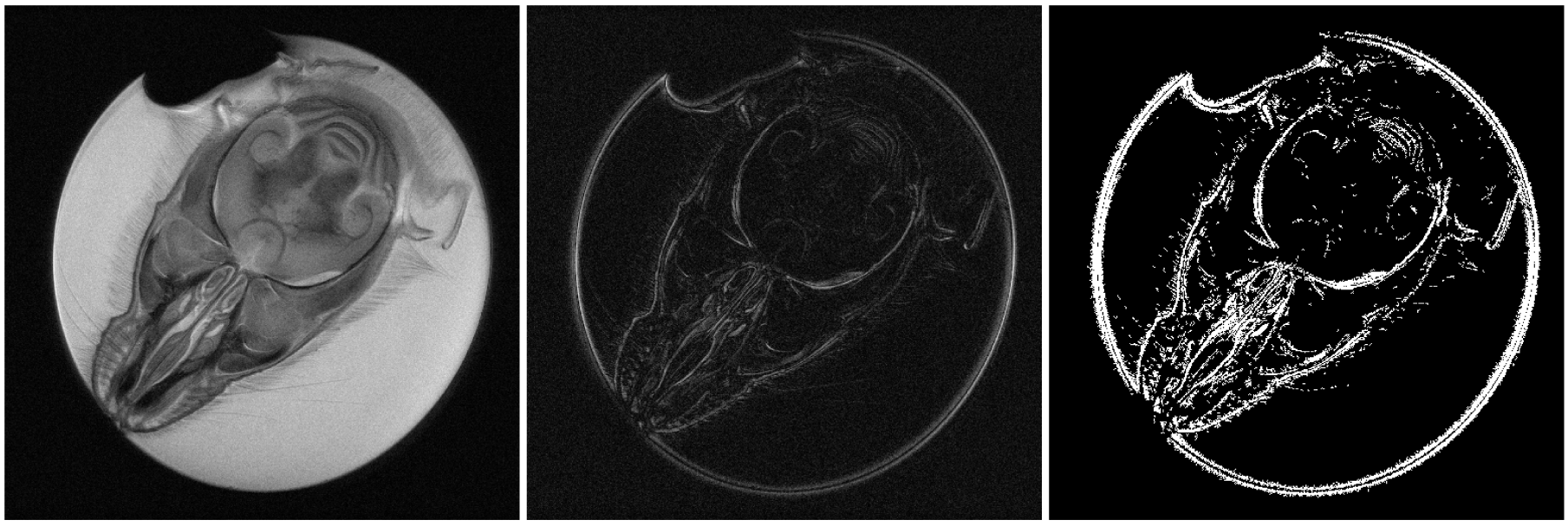
MRI image | k-space-derived details/edges | binary image of details/edges



4.1.6 radius_step > 1

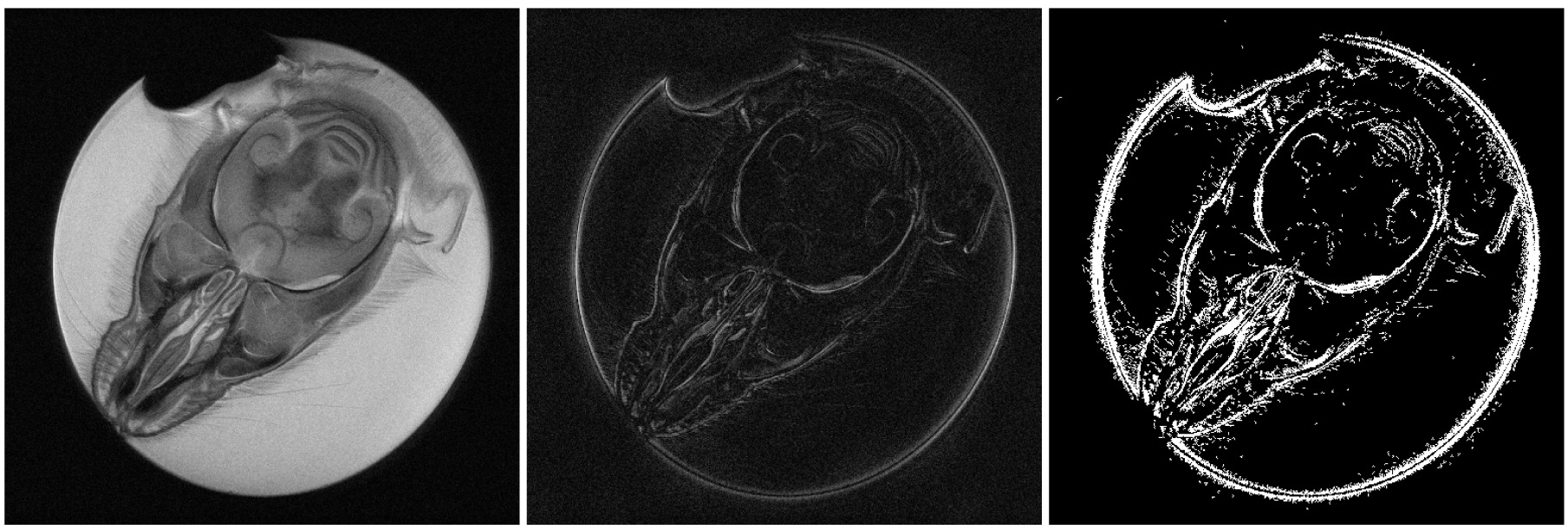
An example of reasonable **radius_step** (= 3):

MRI image | k-space-derived details/edges | binary image of details/edges



and too large **radius_step** (= 12):

MRI image | k-space-derived details/edges | binary image of details/edges



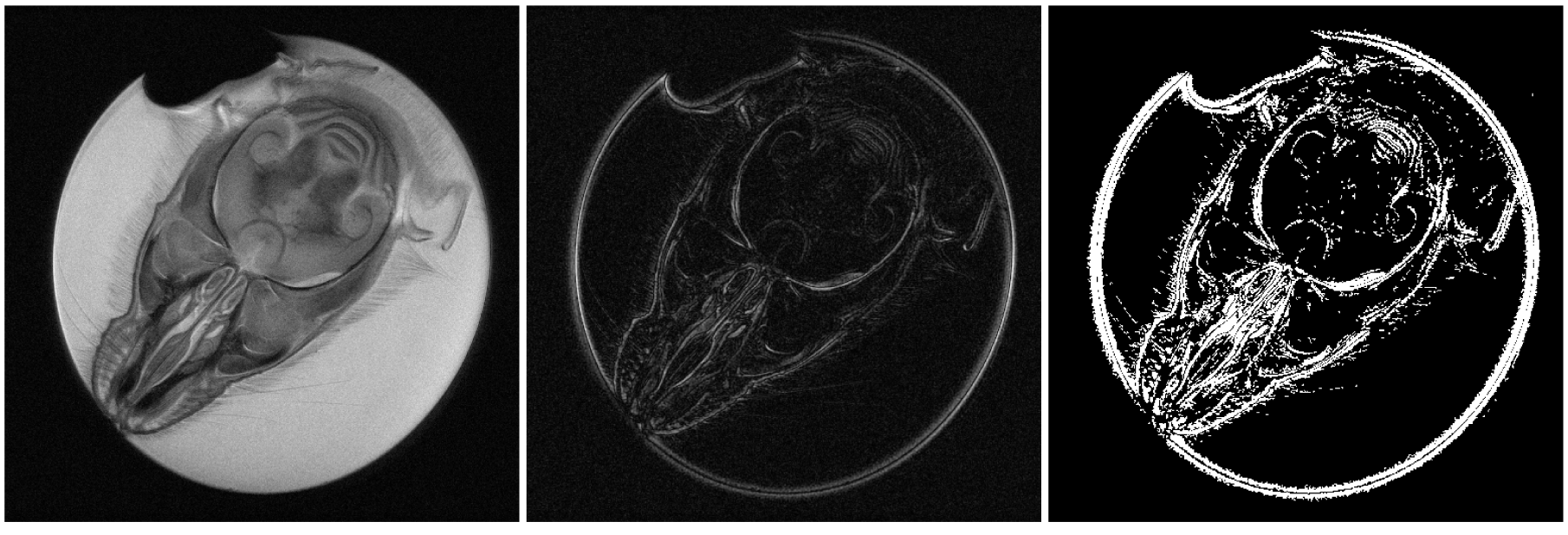
4.2 Implementing denoising and detail control

With denoising and/or detail control implemented, automatic thresholding may not produce satisfactory results. Using manual or adaptive thresholding is recomended.

For detailed information about denoising and detail control implementation see https://github.com/BeataWereszczynska/k-space_wght_msk_for_MRI_denoising/blob/main/Description.pdf [1] and https://github.com/BeataWereszczynska/k-space_masking_for_MRI_denoising/blob/main/Description.pdf [2], respectively.

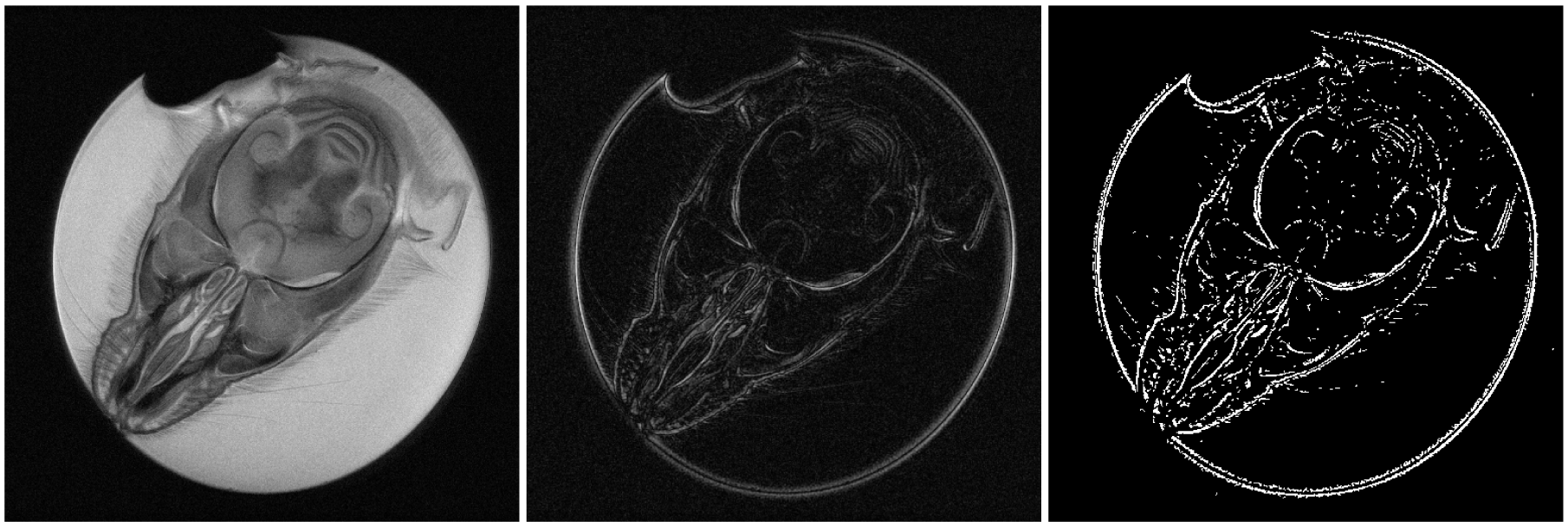
4.2.1 Details/edges detection with denoising, detail control and manual thresholding

MRI image | k-space-derived details/edges | binary image of details/edges



4.2.2 Details/edges detection with denoising, detail control and adaptive thresholding

MRI image | k-space-derived details/edges | binary image of details/edges



5 References

Denoising:

[1] Beata Wereszczyńska, *k-space weighting and masking for denoising of MRI image without blurring or losing contrast, as well as for brightening of the objects in the image with simultaneous noise reduction*, 2022, https://doi.org/10.5281/zenodo.7367057. (https://github.com/BeataWereszczynska/k-space_wght_msk_for_MRI_denoising)

Detail control:

[2] Beata Wereszczyńska, *Graduate k-space masking for MRI image denoising and blurring*, 2022, https://doi.org/10.5281/zenodo.7359195. (https://github.com/BeataWereszczynska/k-space_masking_for_MRI_denoising)

Collection of the sample data:

[3] Beata Wereszczyńska, *Alcohol-fixed specimens for high-contrast post-mortem MRI*, Forensic Imaging, Volume 25, 2021, 200449, ISSN 2666-2256, https://doi.org/10.1016/j.fri.2021.200449. (https://www.sciencedirect.com/science/article/pii/S2666225621000208)

6 About the author

Beata Wereszczyńska, PhD

e-mail: b.e.wereszczynska@gmail.com

GitHub: https://github.com/BeataWereszczynska

ORCID: https://orcid.org/0000-0003-3013-212X

ResearchGate: https://www.researchgate.net/profile/Beata-Wereszczynska

LinkedIn: https://www.linkedin.com/in/beata-wereszczynska/

7 License