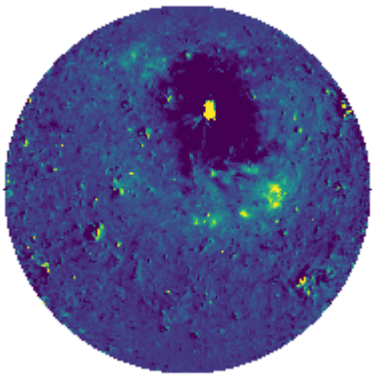
Detection of coronal dimming to warn about coronal mass ejections.



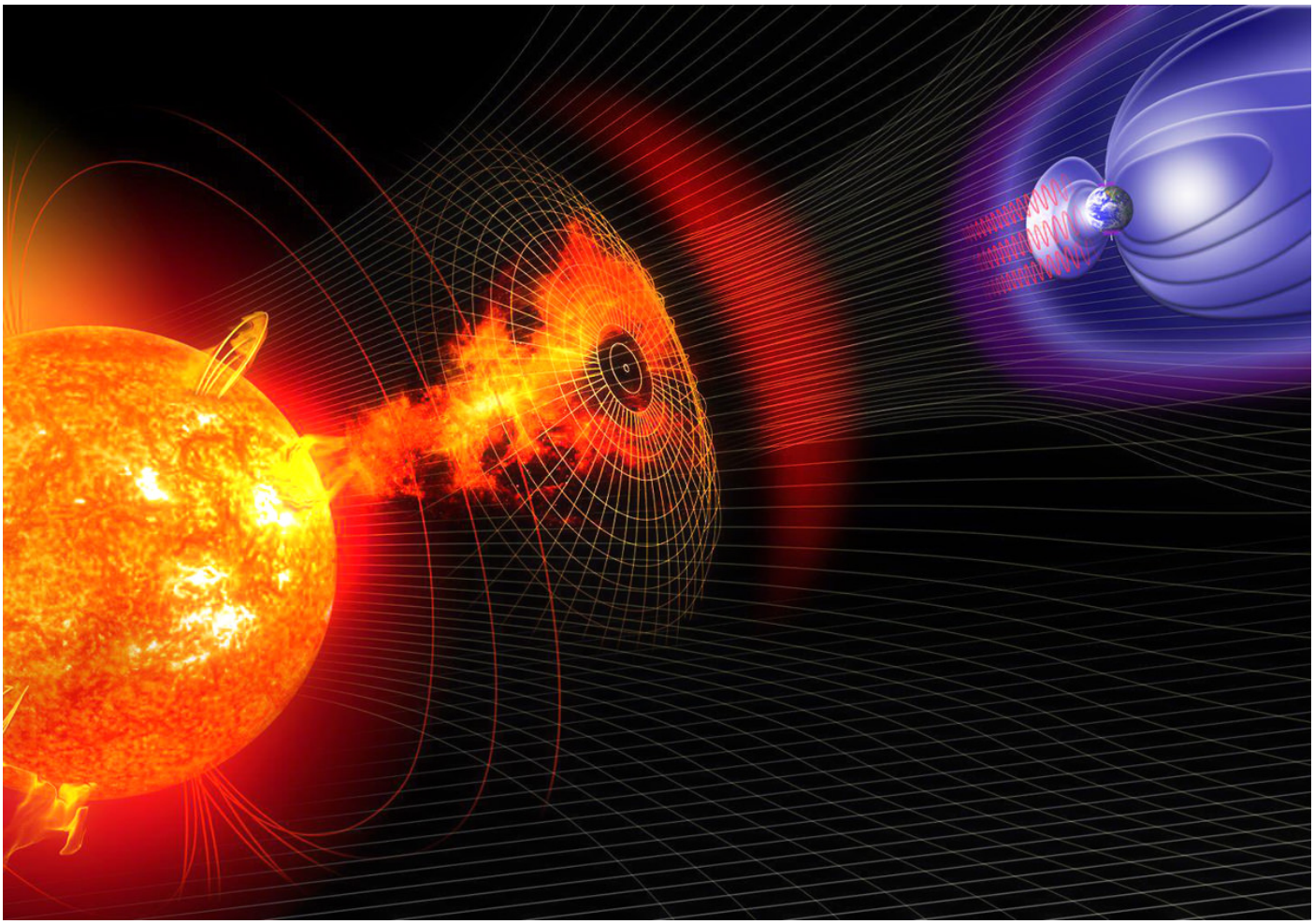
Vasilyev Artem, Gavrilov Mihail, Konubayev Ruslan, Lioznov Anton

Experemental Data Processing, 2018, Term 1B

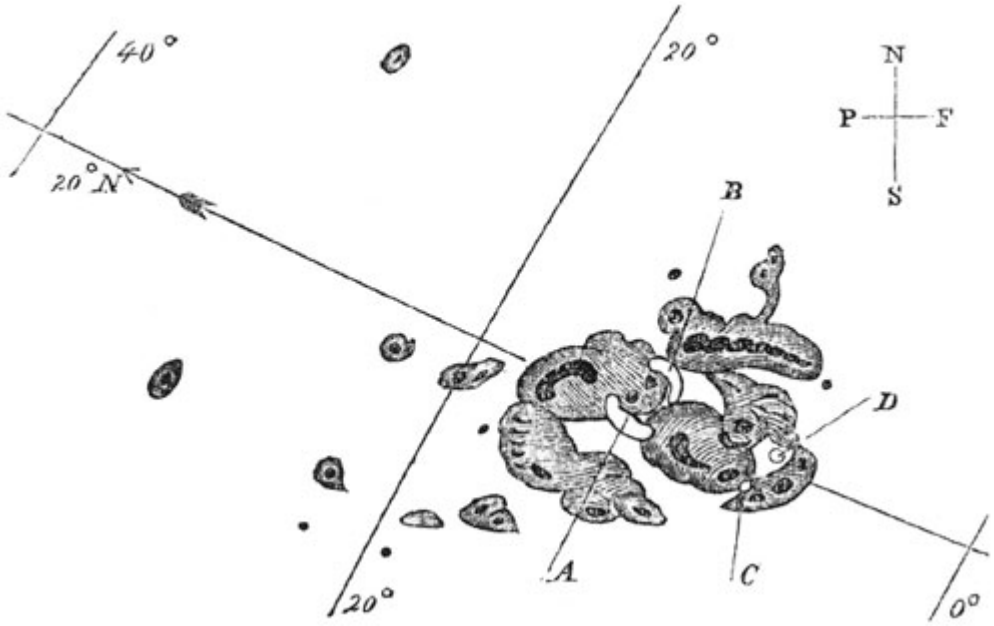
https://lavton.github.io/EDP_reveal/#/ (https://lavton.github.io/EDP_reveal/#/)
https://github.com/Lavton/EDP_reveal (https://github.com/Lavton/EDP_reveal)

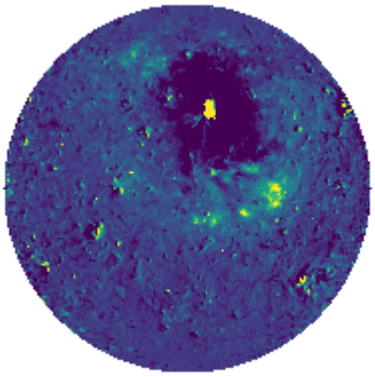
Background

- Coronal mass ejection (CME) is a release of great amount of plasma and electromagnetic radiation from coronal area
- Possible impact on the Earth - aurorae, disrupted radio transmission, power outages
- Possible impact on the space objects - damaged sattelites, health problems for cosmonauts (still under discussion)
- CME can be a serious problem. And once it was.



- One of the most well-known CME events was the solar storm of 1859 (also known as Karrington event)
- Telegram systems in Europe and the United States were out of order, several operators got electric shock
- Current estimates state that similar storm nowadays may cost up to $2.6 trillion (sic!)
- The analysis and prediction of such events is **very** important.





This image was taken at 4:50 UT 12.05.1997 from SOHO satellite

Realization

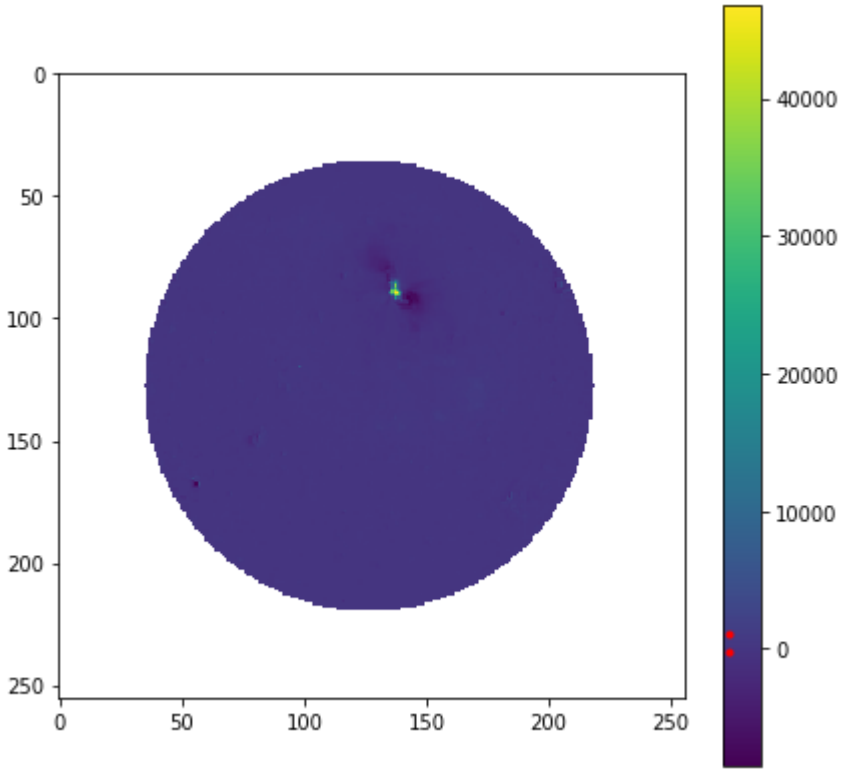
In [37]:
```
from matplotlib import pyplot as plt
import scipy.io
import numpy as np
import cv2
```

In [38]:
```
plt.rcParams['figure.figsize'] = (7, 7)
```

In [39]:
```
mat = scipy.io.loadmat('solar_image.mat')
raw_data = mat["Dif"]
```

Intensitivity range

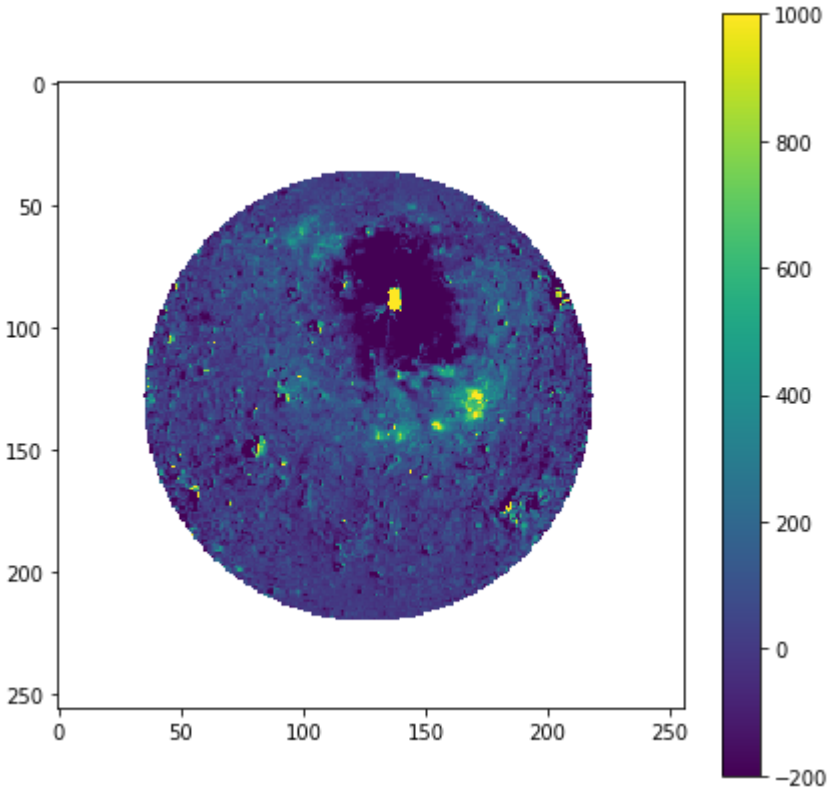
In [40]:
```
plt.imshow(raw_data)
cbar = plt.colorbar()
cbar.ax.plot([0], [-200]*1, 'w.', color="r")
cbar.ax.plot([0], [1000]*1, 'w.', color="r")
plt.show()
```



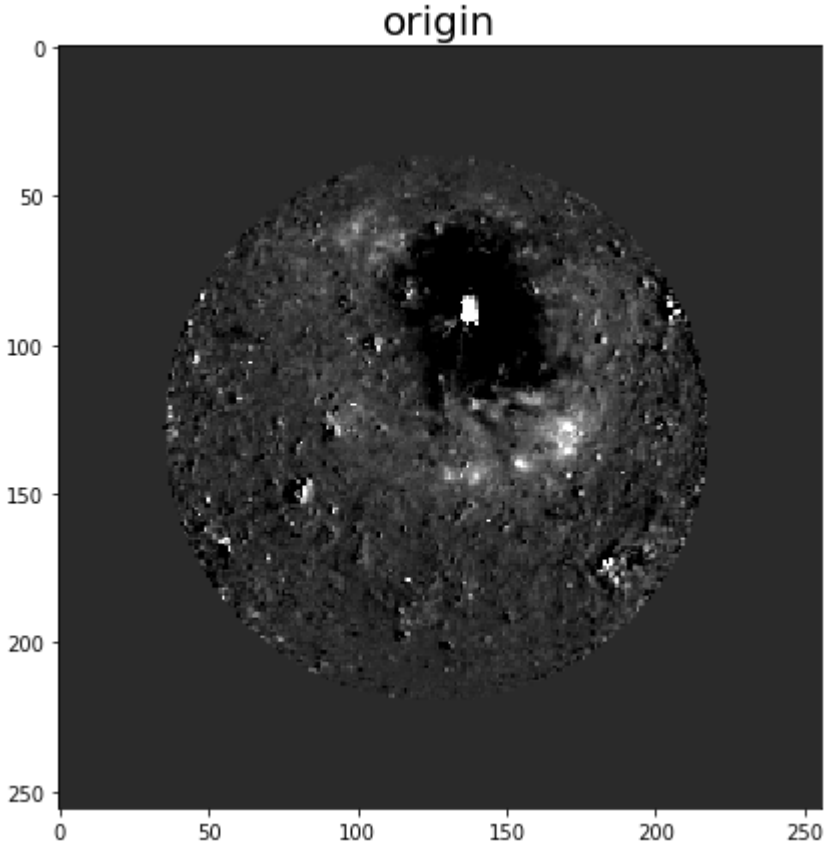
No useful information at all:(

Changing scale

In [41]:
```
plt.imshow(raw_data, vmin=-200, vmax=1000)
plt.colorbar()
plt.show()
```

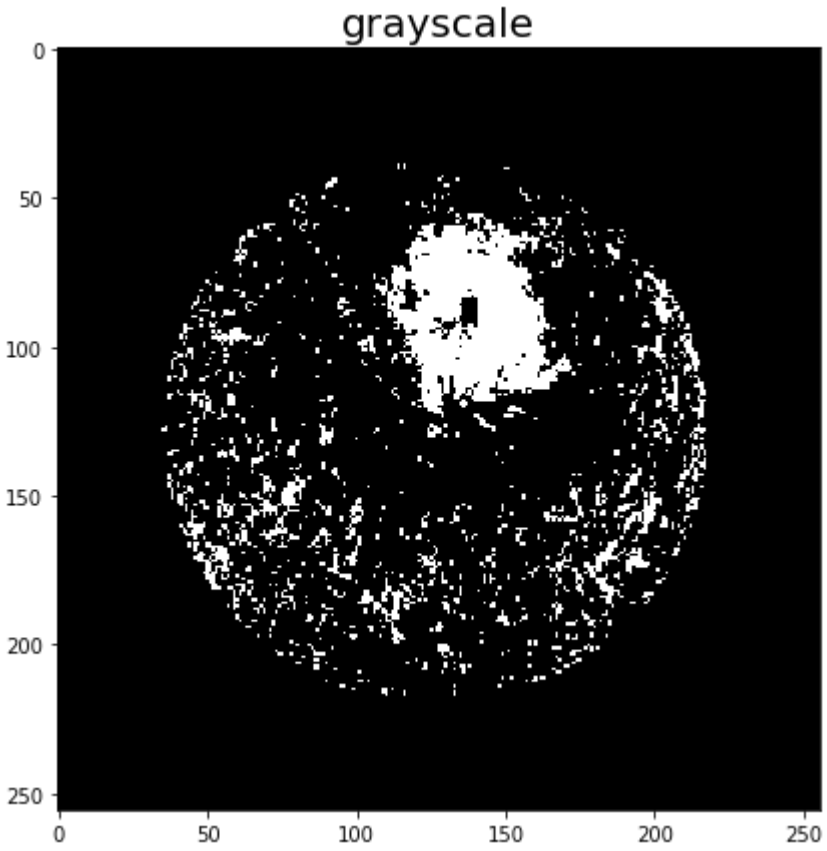


In [42]:
```
plt.imshow(np.nan_to_num(raw_data), "gray", vmin=-200, vmax=1000)
plt.title("origin", fontsize=20)
plt.show()
```



Selecting area with low intensity

In [7]:
```
_,low_int_img = cv2.threshold(raw_data,-65,255,cv2.THRESH_BINARY_INV)
low_int_img = low_int_img.astype("int16")
plt.title("grayscale", fontsize=20)
plt.imshow(low_int_img, "gray")
plt.show()
```

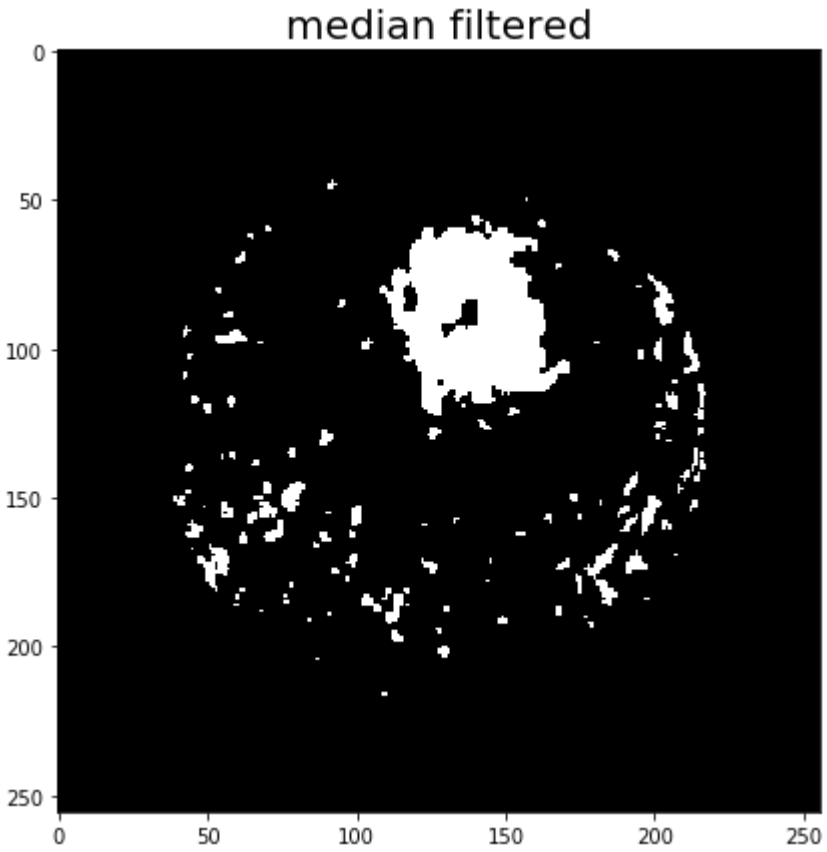


Median filtration

This will remove the small white areas

In [8]:
```
median_filtrd_img = cv2.medianBlur(low_int_img, 3)
```

In [33]:
```
plt.title("median filtered", fontsize=20)
plt.imshow(median_filtrd_img, "gray")
plt.show()
```

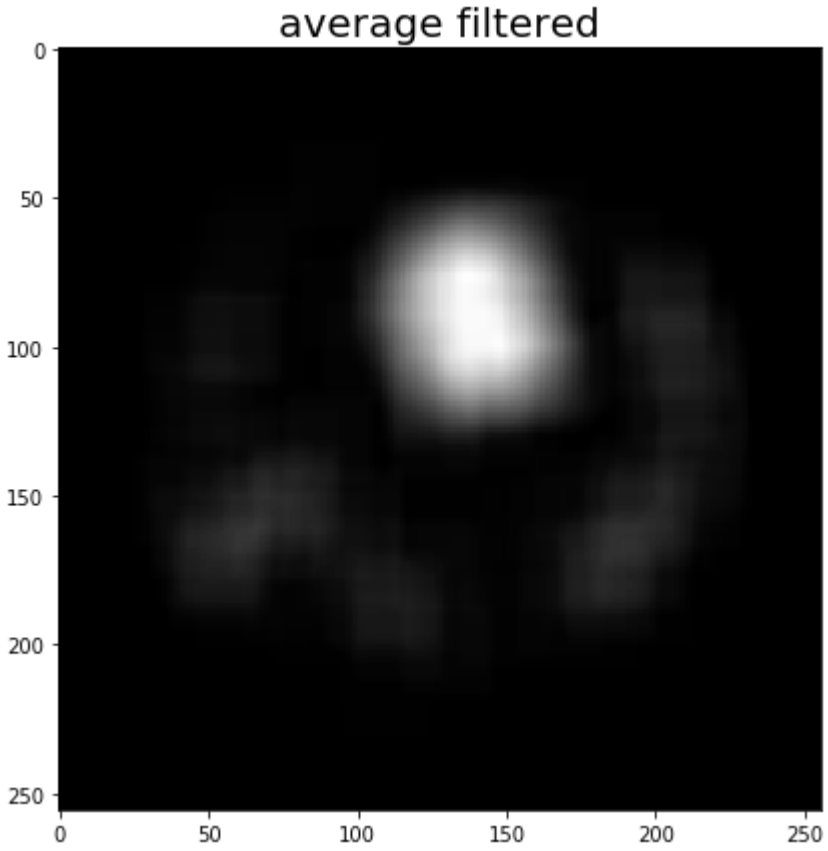


Addition filtration

1. Applying averaging filter with kernel size = 30
2. Using a mask: the filtration results are kept only where pixels are white. Black pixels stay black
3. Binarizing the new image: so only the big areas will be still white

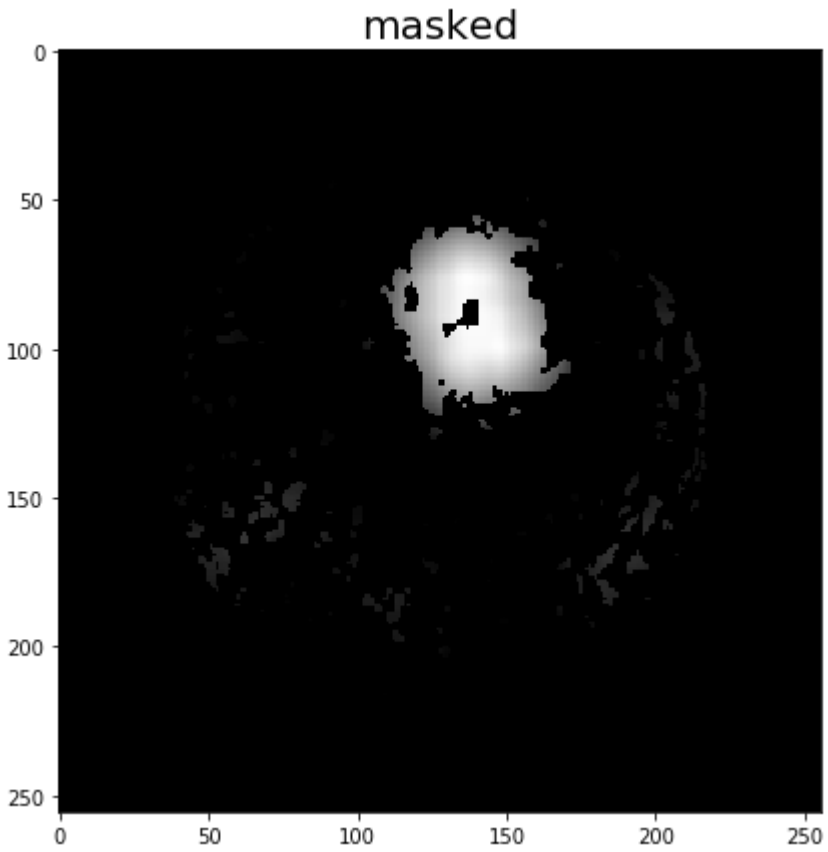
In [10]:
```
kernel = np.ones((30, 30))/(30*30)
averaged_img = cv2.filter2D(median_filtrd_img, -1, kernel)
```

In [13]:
```
plt.title("average filtered", fontsize=20)
plt.imshow(averaged_img, "gray")
plt.show()
median_filtrd_img = cv2.normalize(src=median_filtrd_img, dst=None, alpha=0, be
ta=255, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8UC1)
```



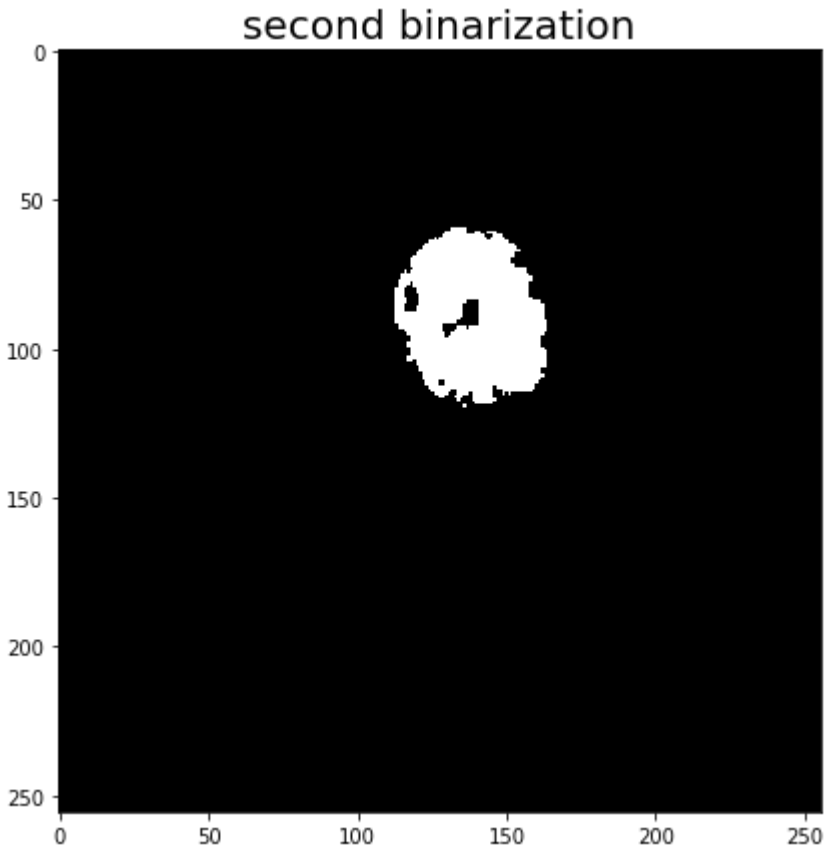
In [15]:
```
masked_img = cv2.bitwise_and(averaged_img, averaged_img,  mask=median_filtrd_i
mg)
```

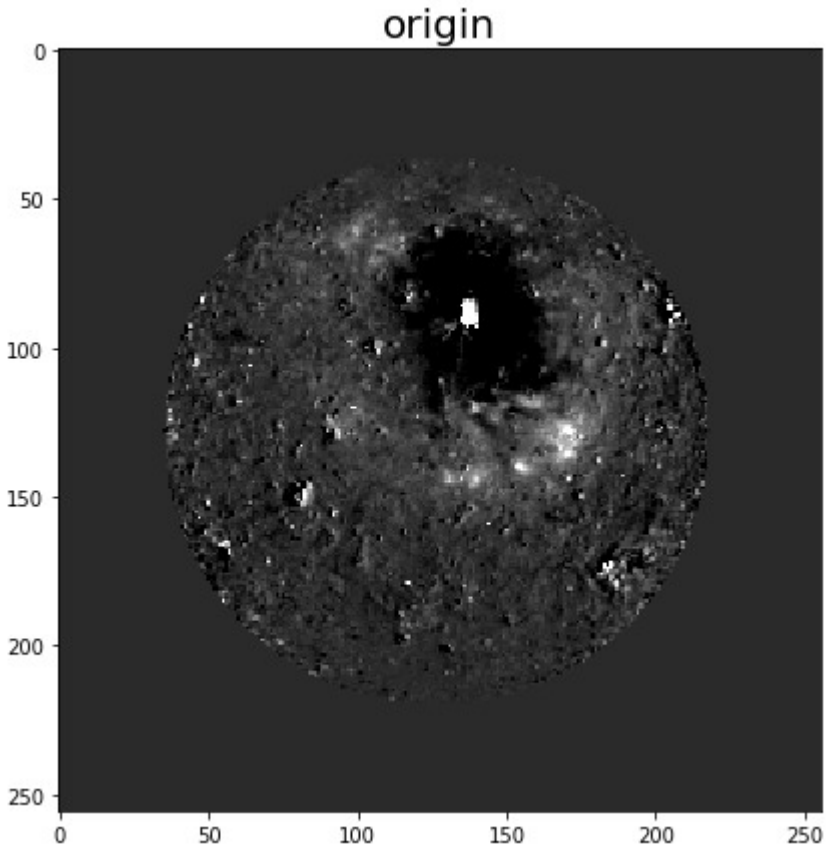
In [18]:
```
plt.title("masked", fontsize=20)
plt.imshow(masked_img, "gray")
plt.show()
```



In [19]:
```
_, second_bin_img = cv2.threshold(masked_img, masked_img.max()*0.45, 255, cv2.
THRESH_BINARY)
```

In [20]:
```
plt.title("second binarization", fontsize=20)
plt.imshow(second_bin_img, "gray")
plt.show()
```





Results

In [52]:
```
radius_sun = np.sqrt(np.count_nonzero(~np.isnan(raw_data))/np.pi)
sun_area = 4*np.pi*radius_sun**2
```

The area of dimming regions is

In [53]:
```
second_bin_img.sum()//255
```

Out[53]: 2156

In [56]:
```
print("which is {}% of the Sun area!".format(int(second_bin_img.sum()/255 / sun_area * 100)))
```

```
which is 2% of the Sun area!
```

Dynamics

There were numerous images for one event. Let's analyze them!

In [57]:
```
import csv
import numpy as np
import matplotlib.pyplot as plt
from os import listdir
from os.path import isfile, join
import cv2
```

read

In [58]:
```
mypath = "txt/"
onlyfiles = [f for f in listdir(mypath) if isfile(join(mypath, f))]
onlyfiles.sort()
```

In [59]:
```
onlyfiles
```

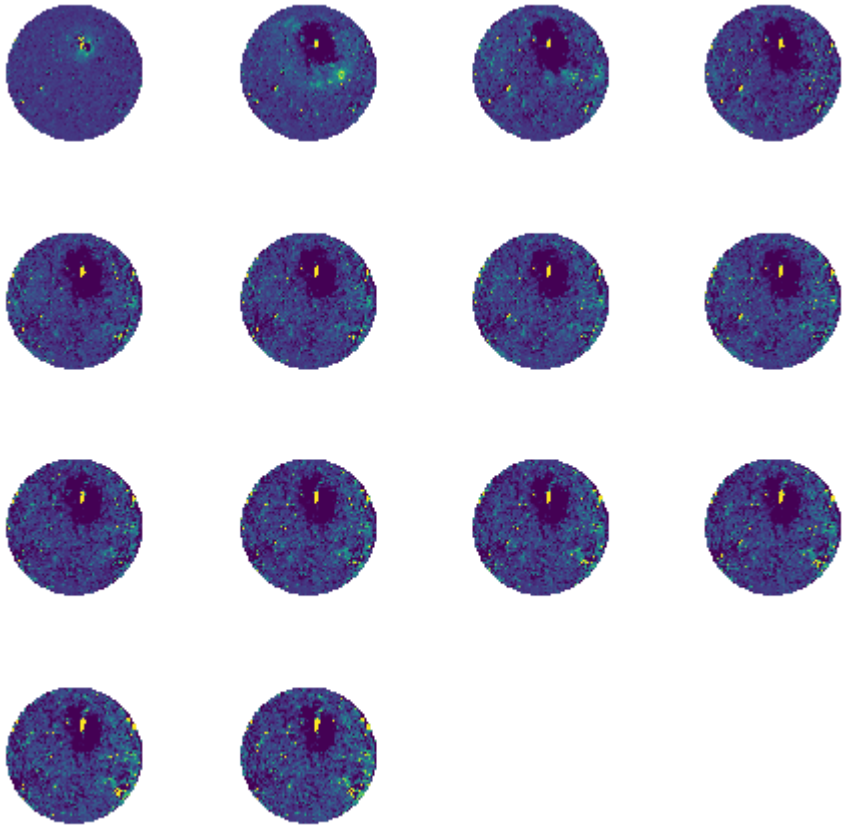
Out[59]:
```
['solar_image16.txt',
 'solar_image17.txt',
 'solar_image18.txt',
 'solar_image19.txt',
 'solar_image20.txt',
 'solar_image21.txt',
 'solar_image22.txt',
 'solar_image23.txt',
 'solar_image24.txt',
 'solar_image25.txt',
 'solar_image26.txt',
 'solar_image27.txt',
 'solar_image28.txt',
 'solar_image29.txt']
```

In [60]:
```
raw_images = []
for f in onlyfiles:
    data = []
    with open('txt/'+f) as csvfile:
        rdata = csv.reader(csvfile, delimiter=" ")
        for row in rdata:
            real_c = []
            for c in row:
                if c:
                    real_c.append(float(c))
            data.append(
                real_c
            )
    data = np.array(data)
    raw_images.append(data)
```

In [61]:
```
for f, data in zip(onlyfiles, raw_images):
    plt.imsave("steps/01/"+f+".png", data, vmin=-200, vmax=1000)
```

In [62]:
```
i = 1
plt.subplots_adjust(wspace=0, hspace=0)
plt.figure(figsize=(8,8))
for i, img in enumerate(raw_images):
    plt.subplot(4, 4, i+1)
    plt.axis('off')
    plt.imshow(img, vmin=-200, vmax=1000)
plt.show()
```

```
<Figure size 504x504 with 0 Axes>
```



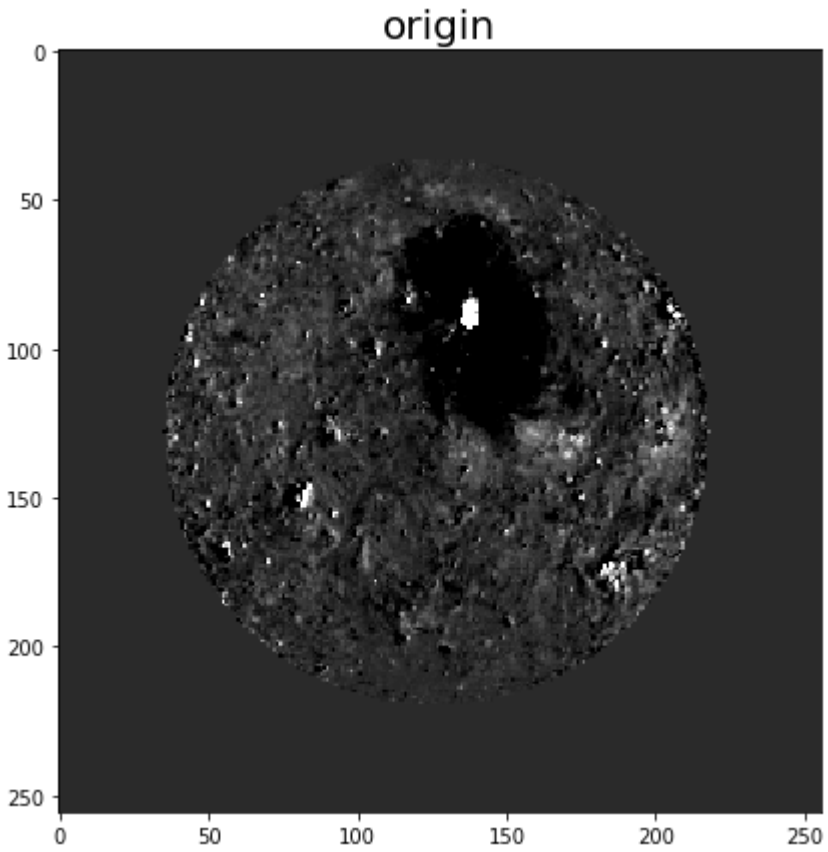
Steps:

- Binarization
- Morphing
- Find the components of connection
- Find the biggest one

Binarization

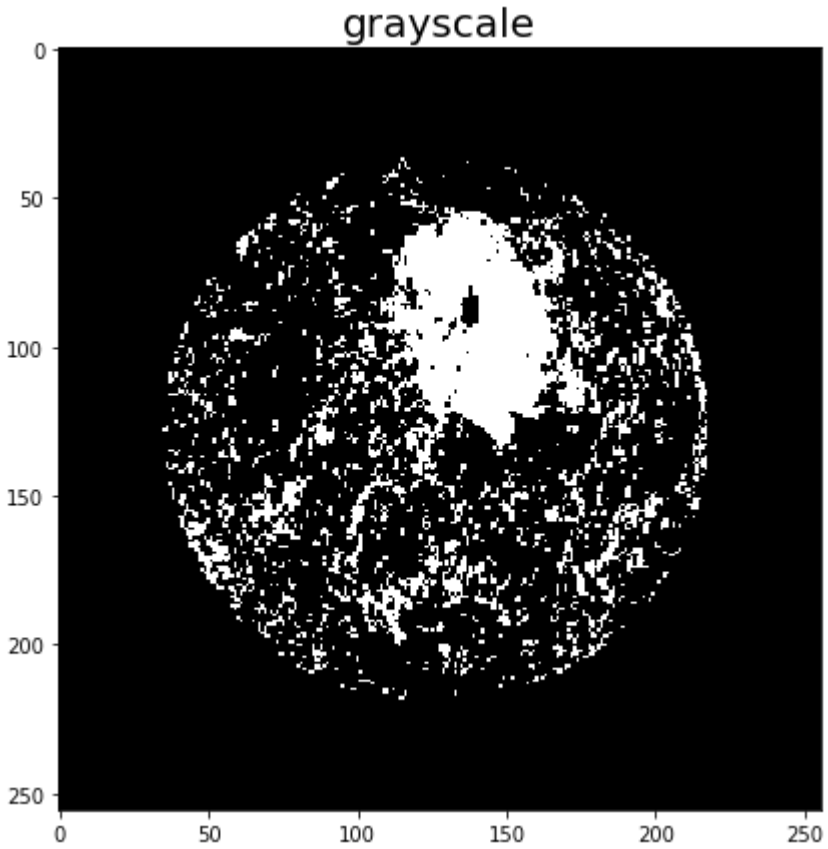
In [63]:
```
raw_data = raw_images[2]
```

In [64]:
```
plt.imshow(np.nan_to_num(raw_data), "gray", vmin=-200, vmax=1000)
plt.title("origin", fontsize=20)
plt.show()
```



In [65]:
```
_,low_int_img = cv2.threshold(raw_data,-65,255,cv2.THRESH_BINARY_INV)
```

In [66]:
```
low_int_img = low_int_img.astype("int16")
plt.title("grayscale", fontsize=20)
plt.imshow(low_int_img, "gray")
plt.show()
```



In [67]:
```
bin_images = []
for f, data in zip(onlyfiles, raw_images):
    _,thresh1 = cv2.threshold(data,-65,255,cv2.THRESH_BINARY_INV)
    thresh1 = thresh1.astype("int16")
    bin_images.append(thresh1)

    cv2.imwrite("steps/02/"+f+".png", thresh1)
```

Morphing

verbose description is avaliable on OpenCV docs
(https://docs.opencv.org/3.4/d9/d61/tutorial_py_morphological_ops.html)

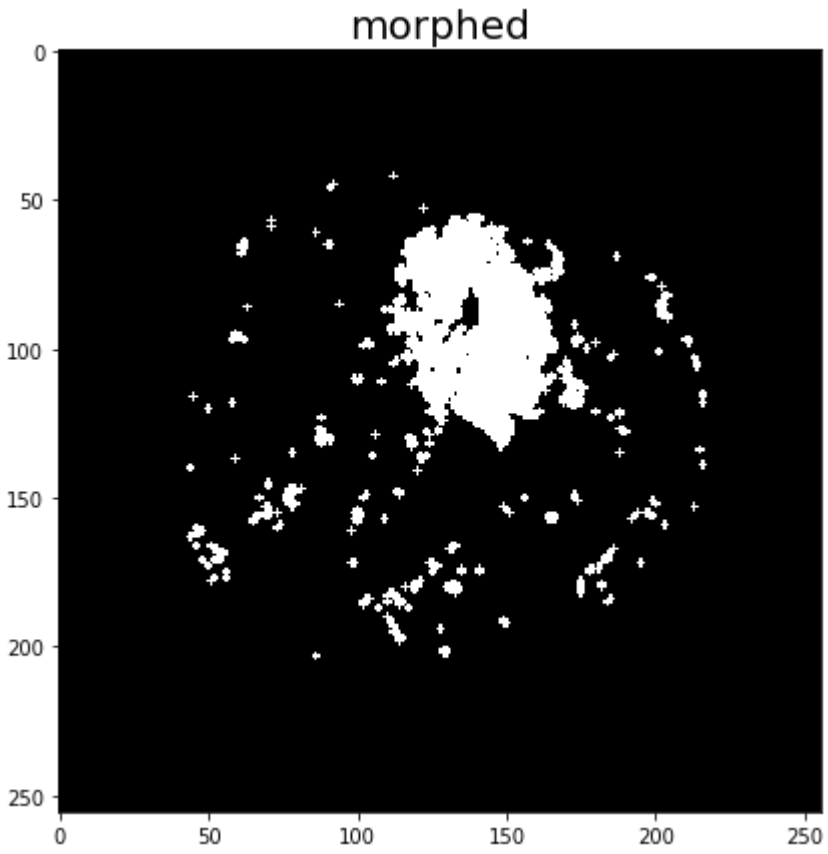
Closing

Opening

In [68]:
```
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(3,3))
closing = cv2.morphologyEx(low_int_img, cv2.MORPH_OPEN, kernel)
```

In [69]:
```
opening = cv2.morphologyEx(closing, cv2.MORPH_OPEN, kernel)
```

In [70]:
```
plt.title("morphed", fontsize=20)
plt.imshow(opening, "gray")
plt.show()
```

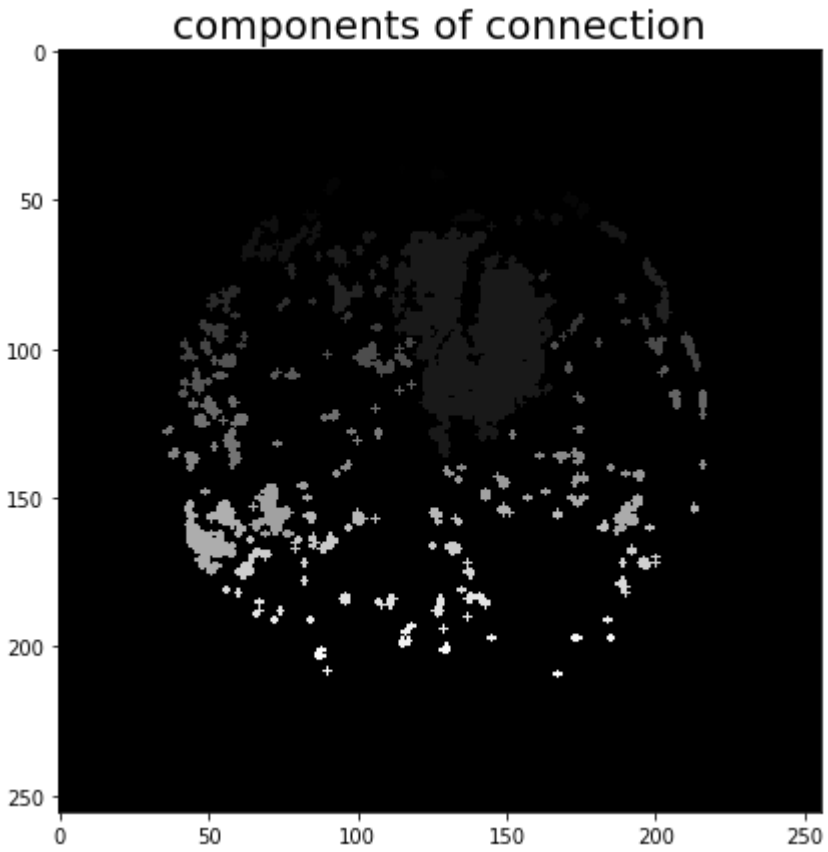


In [71]:
```
morf_transformed_images = []
for f, data in zip(onlyfiles, bin_images):
    closing = cv2.morphologyEx(data, cv2.MORPH_OPEN, kernel)
    opening = cv2.morphologyEx(closing, cv2.MORPH_OPEN, kernel)
    morf_transformed_images.append(opening)
    cv2.imwrite("steps/o03/"+f+".png", opening)
```

Connected components

In [72]:
```
opening = cv2.normalize(src=opening, dst=None, alpha=0, beta=255, norm_type=cv
2.NORM_MINMAX, dtype=cv2.CV_8UC1)
n_c, comp_imag = cv2.connectedComponents(opening)
```

In [73]:
```
plt.title("components of connection", fontsize=20)
plt.imshow(comp_imag, "gray")
plt.show()
```



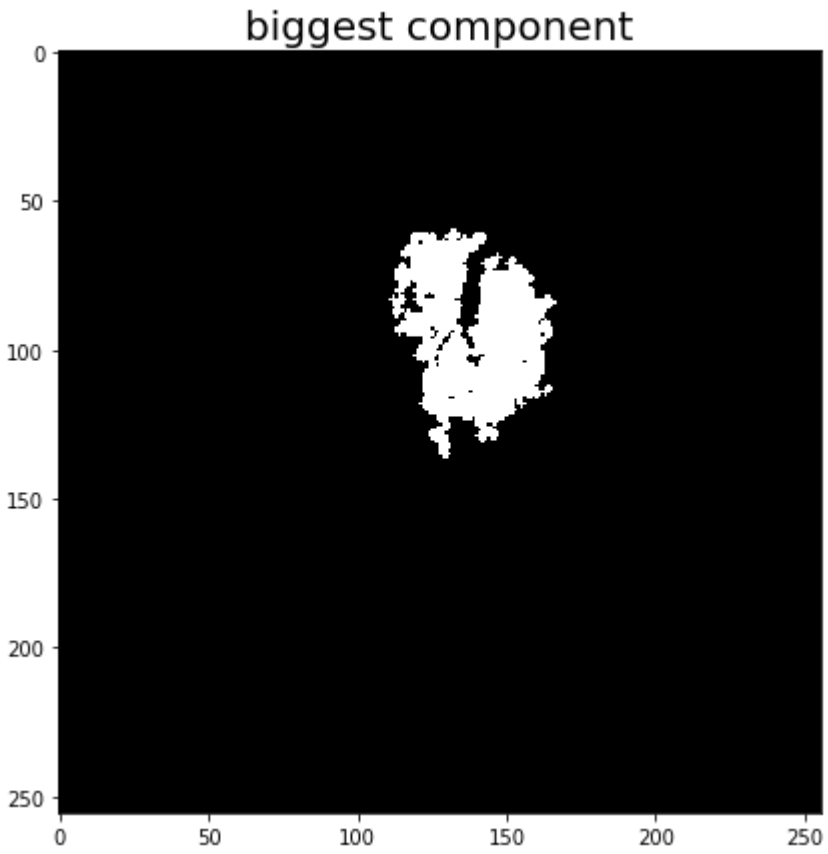
In [74]:
```
components_images = []
for f, data in zip(onlyfiles, morf_transformed_images):
    data = cv2.normalize(src=data, dst=None, alpha=0, beta=255, norm_type=cv2.
NORM_MINMAX, dtype=cv2.CV_8UC1)
    n_c, comp_imag = cv2.connectedComponents(data)
    components_images.append(comp_imag)
    plt.imsave("steps/o04/"+f+".png", comp_imag)
```

Allocating the biggest component

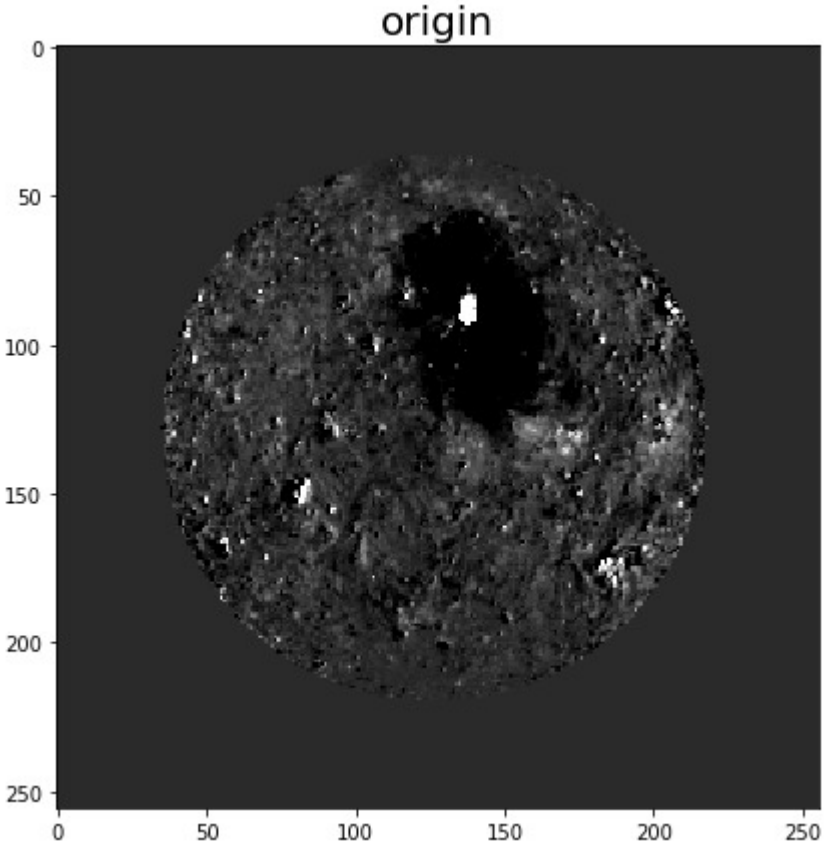
In [75]:
```
from collections import Counter

def count_pix(img):
    counter = Counter()
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            counter[img[i,j]] += 1
    return counter
```

In [76]:
```
c = count_pix(comp_imag).most_common()
new_img = (np.array([comp_imag == c[1][0]]).astype("int16")*255)[0]
```

In [77]:
```
plt.title("biggest component", fontsize=20)
plt.imshow(new_img, "gray")
plt.show()
```



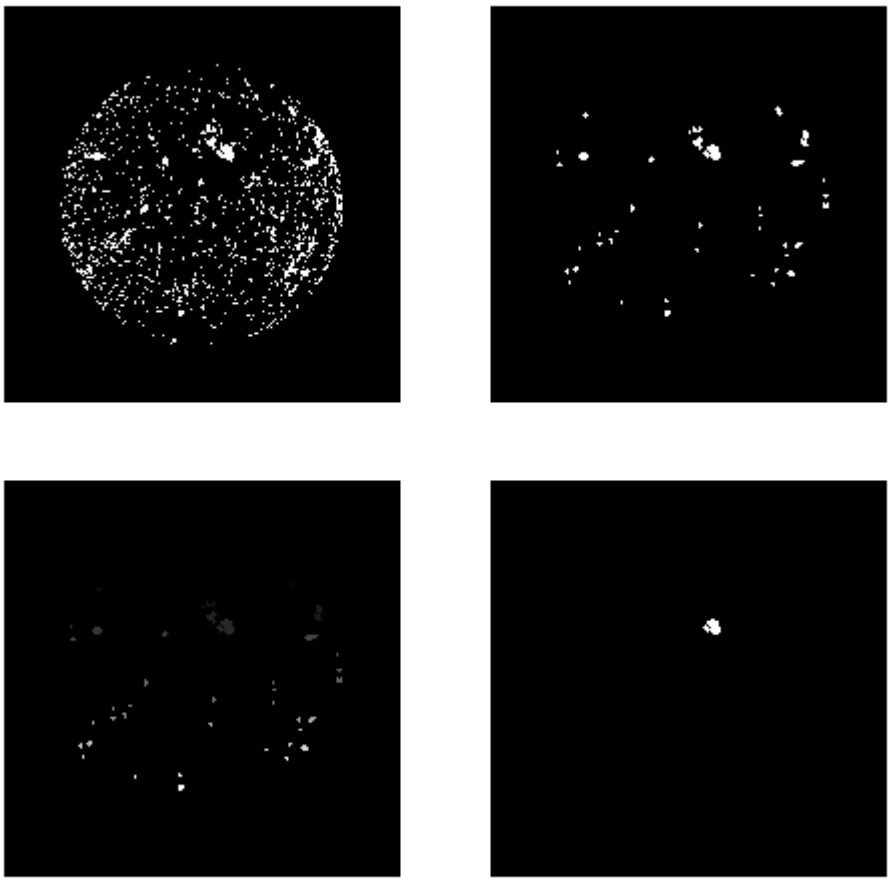
In [78]:
```
final_new_images = []
for f, data in zip(onlyfiles, components_images):
    c = count_pix(data).most_common()
    new_img = (np.array([data == c[1][0]]).astype("int16")*255)[0]
    final_new_images.append(new_img)
    cv2.imwrite("steps/o05/"+f+".png", new_img)
```


origin
0
50
100
150
200
250
0
50
100
150
200
250

In [79]:
```
plt.subplots_adjust(wspace=0, hspace=0)
plt.figure(figsize=(8,8))
for i in range(len(bin_images)):
    plt.subplot(221)
    plt.axis('off')
    plt.imshow(bin_images[i], "gray")

    plt.subplot(222)
    plt.axis('off')
    plt.imshow(morf_transformed_images[i], "gray")

    plt.subplot(223)
    plt.axis('off')
    plt.imshow(components_images[i], "gray")

    plt.subplot(224)
    plt.axis('off')
    plt.imshow(final_new_images[i], "gray")
    plt.savefig("imgs/01/l{num:02d}.png".format(num=i))
    plt.cla()
    plt.clf()
```

```
<Figure size 504x504 with 0 Axes>
```

```
<Figure size 576x576 with 0 Axes>
```



The images with the CME

In [80]:
```
bin_images_c = []
for f, data in zip(onlyfiles, raw_images):
    ret,thresh1 = cv2.threshold(data,1000,255,cv2.THRESH_BINARY)
    thresh1 = thresh1.astype("int16")
    bin_images_c.append(thresh1)

    cv2.imwrite("steps/i02/"+f+".png", thresh1)
```

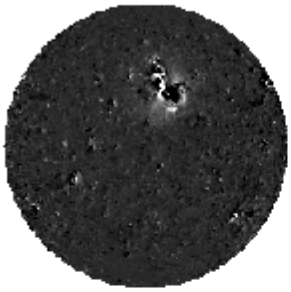
In [81]:
```
injection_images = []
for f, data in zip(onlyfiles, bin_images_c):
    data = cv2.normalize(src=data, dst=None, alpha=0, beta=255, norm_type=cv2.
NORM_MINMAX, dtype=cv2.CV_8UC1)
    n_c, comp_imag = cv2.connectedComponents(data)
    c = count_pix(comp_imag).most_common()
    new_img = (np.array([comp_imag == c[1][0]]).astype("int16")*255)[0]
    injection_images.append(new_img)
    cv2.imwrite("steps/i03/"+f+".png", new_img)
```

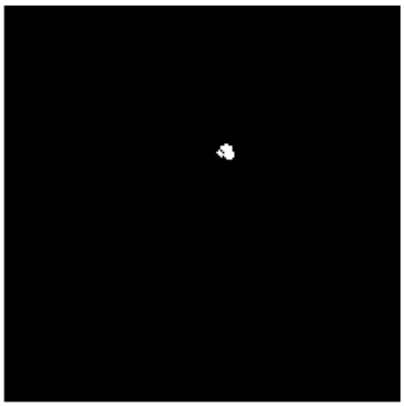
Intensity inside the flash area

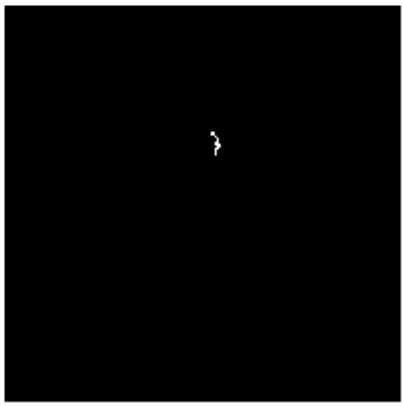
In [82]:
```
intensity = []
for raw, inj in zip(raw_images, injection_images):
    intensity.append(raw[inj==255].sum())
```

In [83]:
```
plt.subplots_adjust(wspace=0, hspace=0)
plt.figure(figsize=(8,8))
for i in range(len(bin_images)):
    plt.subplot(221)
    plt.axis('off')
    plt.imshow(raw_images[i], "gray", vmin=-200, vmax=1000)

    plt.subplot(223)
    plt.axis('off')
    plt.imshow(final_new_images[i], "gray")

    plt.subplot(224)
    plt.axis('off')
    plt.imshow(injection_images[i], "gray")
    plt.savefig("imgs/02/l{num:02d}.png".format(num=i))
    plt.cla()
    plt.clf()
```

```
<Figure size 504x504 with 0 Axes>

<Figure size 576x576 with 0 Axes>
```

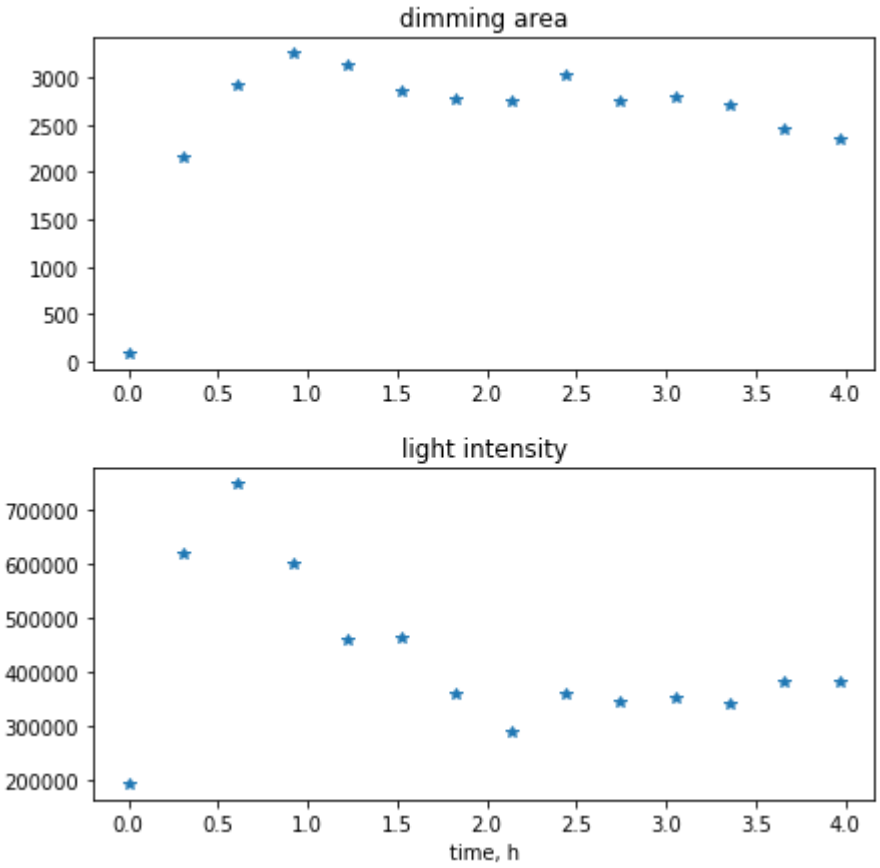






Results

In [84]:
```
time_ax = np.linspace(0, len(final_new_images)*17, len(final_new_images))/60
plt.subplots_adjust(wspace=0, hspace=0.3)
plt.subplot(211)
plt.title("dimming area")
plt.plot(time_ax, [img.sum()//255 for img in final_new_images], "*")

plt.subplot(212)
plt.title("light intensity")
plt.plot(time_ax, [intensity_ for intensity_ in intensity], "*")
plt.xlabel("time, h")
plt.show()
```



Conclusions

- CME is crucial phenomenon that may cause deteriorating consequences for the humankind, thus, the prediction is vital;
- We found out that CME causes temperutare decrese in the area near to ejection;
- The conclusion above was made with the use of trivial image processing methods;
- The analysis of CME dynamics shows that the size of CME and dimming areas are related;
- Despite the model of the process requires profound theoretical knowledge in the sphere of space weather,the background is not needed for data processing - only common sense, programming and basic maths.