Pre-Process Image

In [44]:
```
# Import necessary packages
import numpy as np
import matplotlib.pyplot as plt

from scipy.ndimage import gaussian_filter
from skimage import data
from skimage import img_as_float
from skimage.morphology import reconstruction
from skimage.color import rgb2gray
```

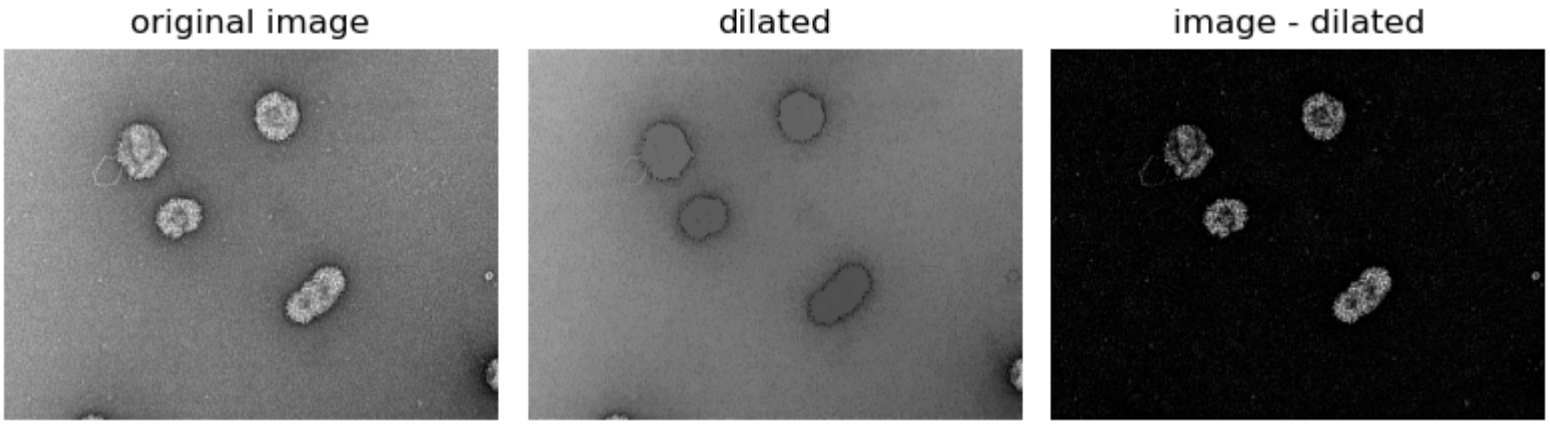
Test on Local Data

In [35]:
```
# Convert to float: Important for subtraction later which won't work with uint8
from skimage import io
image = img_as_float(io.imread('/home/ananda/Downloads/vesicles.png'))
```

In [36]:
```
#image = img_as_float(data.coins())
image = gaussian_filter(image, 1)

seed = np.copy(image)
seed[1:-1, 1:-1] = image.min()
mask = image

dilated = reconstruction(seed, mask, method='dilation')
```

In [47]:
```
# Subtracting the dilated image leaves an image with just the coins and a flat, black background, as
fig, (ax0, ax1, ax2) = plt.subplots(nrows=1,
                                    ncols=3,
                                    figsize=(8, 2.5),
                                    sharex=True,
                                    sharey=True)

ax0.imshow(image, cmap='gray')
ax0.set_title('original image')
```

```
ax0.axis('off')

ax1.imshow(dilated, vmin=image.min(), vmax=image.max(), cmap='gray')
ax1.set_title('dilated')
ax1.axis('off')

final_image = image - dilated

#ax2.imshow(image - dilated, cmap='gray')
ax2.imshow(final_image, cmap='gray')
#ax2.imshow(final_image, cmap='gray')
ax2.set_title('image - dilated')
ax2.axis('off')

fig.tight_layout()
```



In [49]:
```
# Obtain the final image
greyFinalImage = ax2.imshow(final_image, cmap='gray')
greyImage = greyFinalImage.get_array()
```

In [52]:
```
# Obtain the dilated image
dilImageplot = ax1.imshow(dilated, vmin=image.min(), vmax=image.max(), cmap='gray')
dilImage = dilImageplot.get_array()
```

Implementing Trainable Segmentation with Local Features and Random Forests

Because we already implemented similar technique directly on Ilastik (vesicle detection), we can skip this (although it's a good technique, it might be computationall expensive to be executed on other OS with weaker processor). Next up, we can implement object detection based on Otsu thresholding.

Segment Locally without Training (Otsu Thresholding)

In [31]:
```
# Import necesary packages
import plotly
import plotly.express as px
import plotly.graph_objects as go
from skimage import data, filters, measure, morphology
```

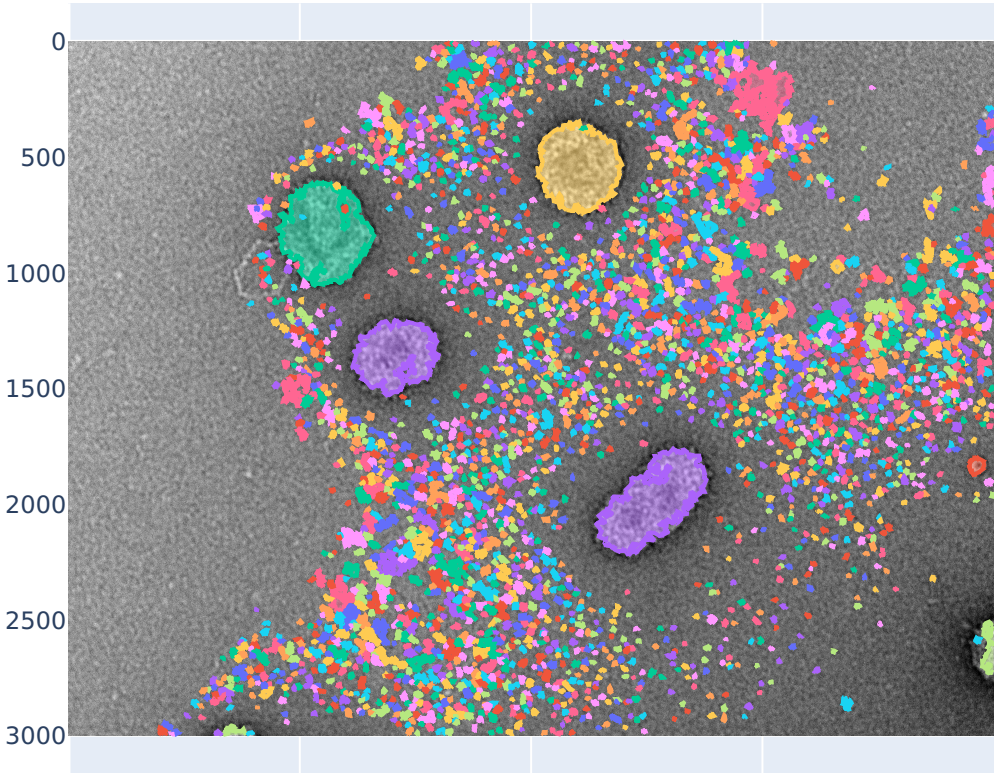
Test on Local Data

In [57]:
```
# Load image
from skimage import io
img = io.imread('/home/ananda/Downloads/vesicles.png')
```

In [60]:
```
# Binary image, post-process the binary mask and compute labels
threshold = filters.threshold_otsu(img)
mask = img > threshold
mask = morphology.remove_small_objects(mask, 50)
mask = morphology.remove_small_holes(mask, 50)
labels = measure.label(mask)

fig = px.imshow(img, binary_string=True)
fig.update_traces(hoverinfo='skip') # hover is only for label info

props = measure.regionprops(labels, img)
properties = ['area', 'eccentricity', 'perimeter', 'intensity_mean']
```

In [61]:
```
# For each label, add a filled scatter trace for its contour,
# and display the properties of the label in the hover of this trace.
for index in range(1, labels.max()):
    label_i = props[index].label
    contour = measure.find_contours(labels == label_i, 0.5)[0]
```

```
    y, x = contour.T
    hoverinfo = ''
    for prop_name in properties:
        hoverinfo += f'<b>{prop_name}: {getattr(props[index], prop_name):.2f}</b><br>'
    fig.add_trace(go.Scatter(
        x=x, y=y, name=label_i,
        mode='lines', fill='toself', showlegend=False,
        hovertemplate=hoverinfo, hoveron='points+fills'))

plotly.io.show(fig)
```



```
It's bad I know but that's because we used the raw image instead of the pre-procesed image. This
is just something interesting to learn.
```