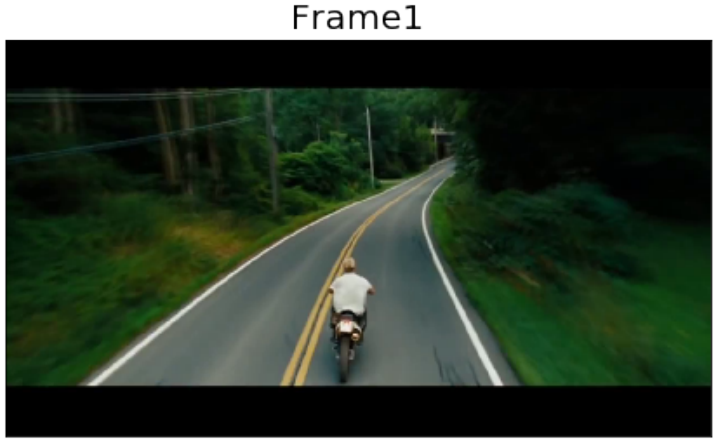
Problem 1

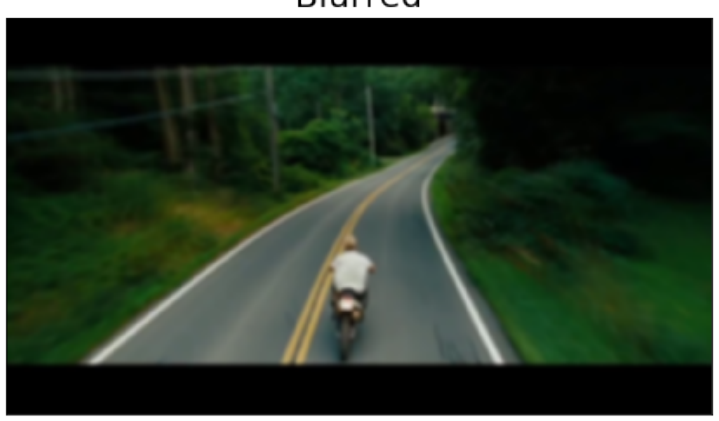
In [27]:

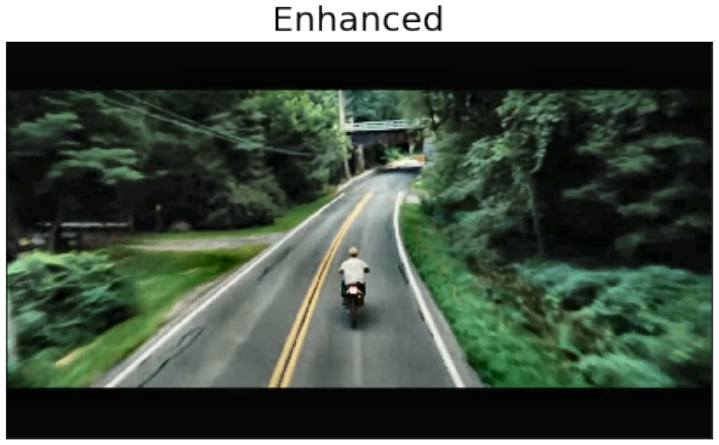
```
import cv2
import pylab
import imageio
from PIL import Image, ImageEnhance
from matplotlib import pyplot as plt
filename = 'movie.mp4'
vid = imageio.get_reader(filename,  'mp4')

image1 = vid.get_data(10)
blur = cv2.GaussianBlur(image1,(15,15),cv2.BORDER_DEFAULT)


image2 = vid.get_data(100)
clahe = cv2.createCLAHE(clipLimit=3., tileGridSize=(8,8))

lab = cv2.cvtColor(image2, cv2.COLOR_BGR2LAB)
l, a, b = cv2.split(lab)

l2 = clahe.apply(l)
```

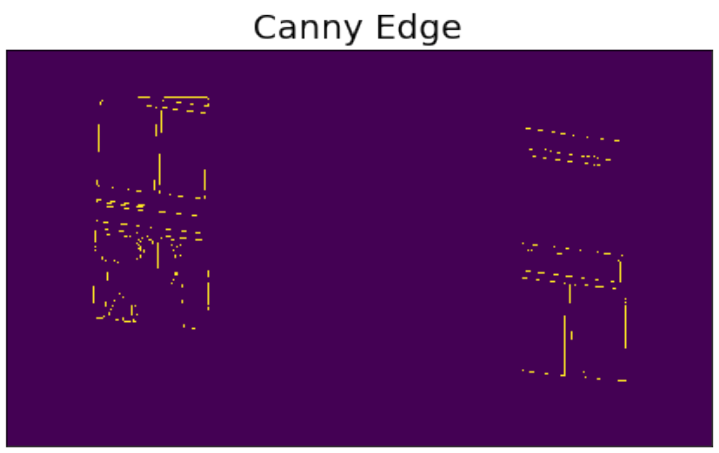
```
lab = cv2.merge((l2,a,b))
enhanced = cv2.cvtColor(lab, cv2.COLOR_LAB2BGR)

image3=vid.get_data(287)
edges = cv2.Canny(image3,100,200)

fig = plt.figure(figsize=(16,16))
plt.subplot(321)
plt.imshow(image1)
plt.title("Frame1",fontsize=20), plt.xticks([]), plt.yticks([])
plt.subplot(322)
plt.imshow(blur)
plt.title("Blurred",fontsize=20), plt.xticks([]), plt.yticks([])

plt.subplot(323)
plt.imshow(image2)
plt.title("Frame2",fontsize=20), plt.xticks([]), plt.yticks([])
plt.subplot(324)
plt.imshow(enhanced)
plt.title("Enhanced",fontsize=20), plt.xticks([]), plt.yticks([])

plt.subplot(325)
plt.imshow(image3)
plt.title("Frame3",fontsize=20), plt.xticks([]), plt.yticks([])
plt.subplot(326)
plt.imshow(edges)
```

```
plt.title( 'Canny Edge' ,fontsize=20), plt.xticks([]), plt.yticks([])
plt.show()
```









In [ ]:

```
```

Problem 2

In [28]:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
from skimage import io
```

In [29]:

```
one=io.imread('1.gif')

three=io.imread('3.gif')
```

In [30]:

```
one = np.float32(one)
dst1 = cv2.cornerHarris(one,2,3,0.1)
dst1 = cv2.dilate(dst1,None)
```

In [31]:

```
two=io.imread('2.gif')
two = np.float32(two)
dst = cv2.cornerHarris(two,2,3,0.1)
dst = cv2.dilate(dst,None)
```

In [32]:

```
oneDots = cv2.cvtColor(one,cv2.COLOR_GRAY2RGB)
oneDots[dst1>0.0001*dst1.max()]=[0,0,255]
```

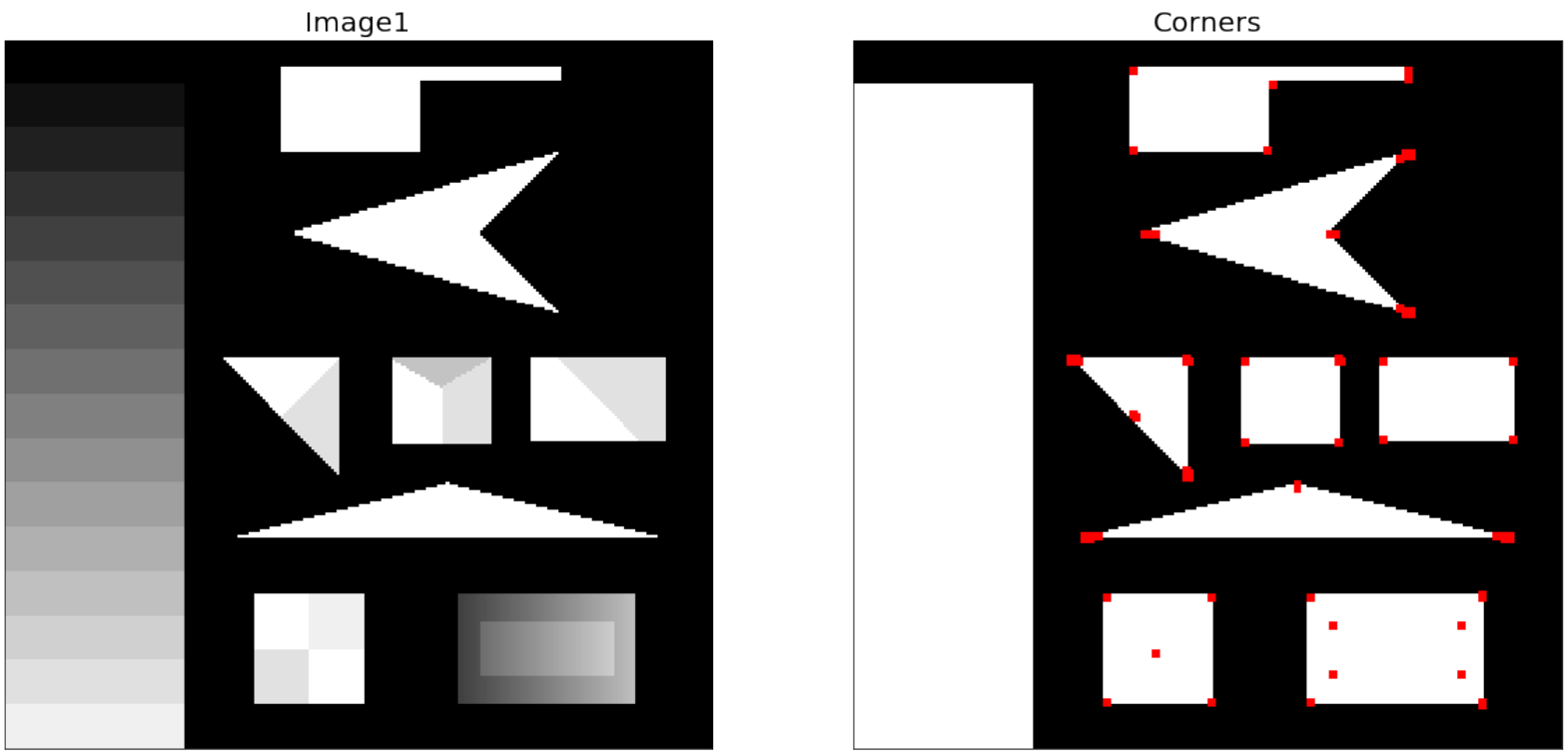
In [33]:

```
twoDots = cv2.cvtColor(two,cv2.COLOR_GRAY2RGB)
twoDots[dst>0.01*dst.max()]=[0,0,255]
```

In [34]:

```
fig = plt.figure(figsize=(20,20))
plt.subplot(121)
plt.imshow(one, cmap='gray')
plt.title("Image1",fontsize=20), plt.xticks([]), plt.yticks([])
plt.subplot(122)
plt.imshow(cv2.cvtColor(oneDots, cv2.COLOR_BGR2RGB))
plt.title("Corners",fontsize=20), plt.xticks([]), plt.yticks([])


plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1
] for floats or [0..255] for integers).
```



In [35]:

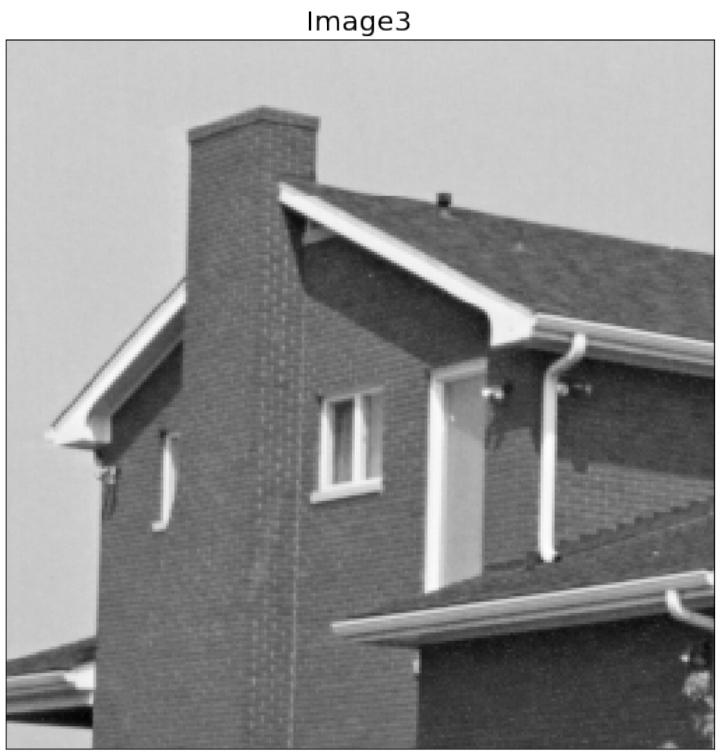
```
image=three.copy()
three = np.float32(three)
dst3 = cv2.cornerHarris(three,2,3,0.1)
dst3 = cv2.dilate(dst3,None)
```

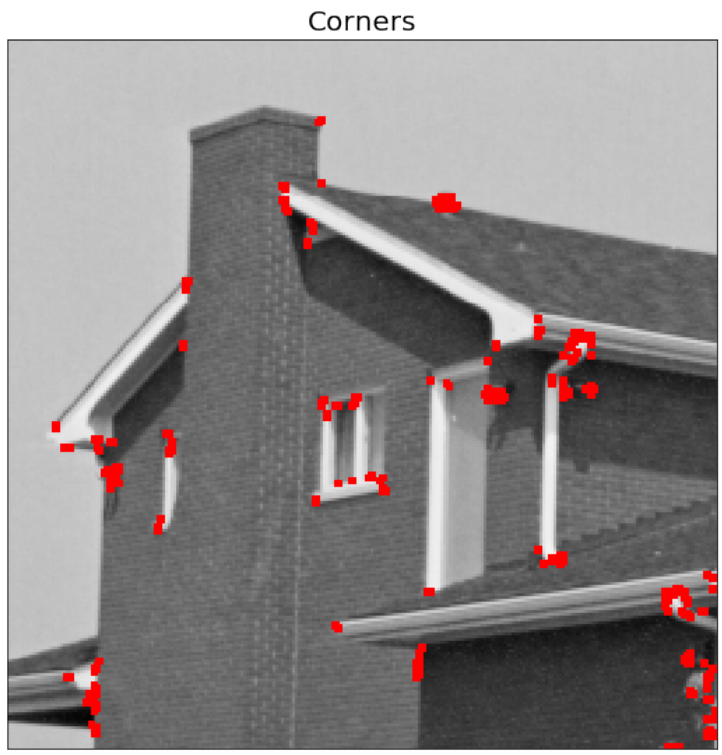
In [36]:

```
threeDots = cv2.cvtColor(image,cv2.COLOR_GRAY2RGB)
threeDots[dst3>0.01*dst3.max()]=[0,0,255]
```

In [37]:

```
fig = plt.figure(figsize=(20,20))
plt.subplot(121)
plt.imshow(image, cmap='gray')
plt.title("Image3",fontsize=20), plt.xticks([]), plt.yticks([])
plt.subplot(122)
plt.imshow(cv2.cvtColor(threeDots, cv2.COLOR_BGR2RGB))
plt.title("Corners",fontsize=20), plt.xticks([]), plt.yticks([])
plt.show()
```





In [38]:

```
two=io.imread('2.gif')
```

In [39]:

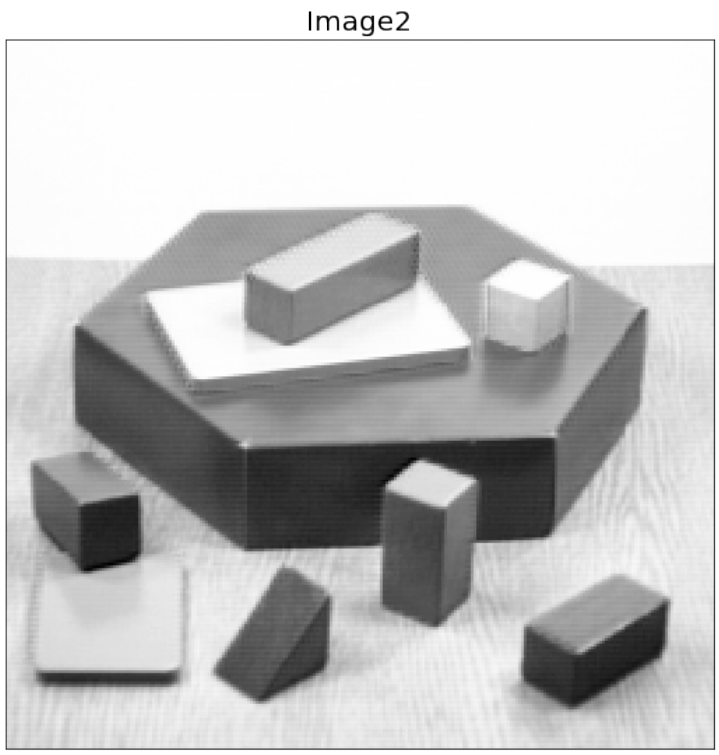
```
img=two.copy()
two = np.float32(two)
dst2 = cv2.cornerHarris(two,2,3,0.01)
dst2 = cv2.dilate(dst2,None)
```

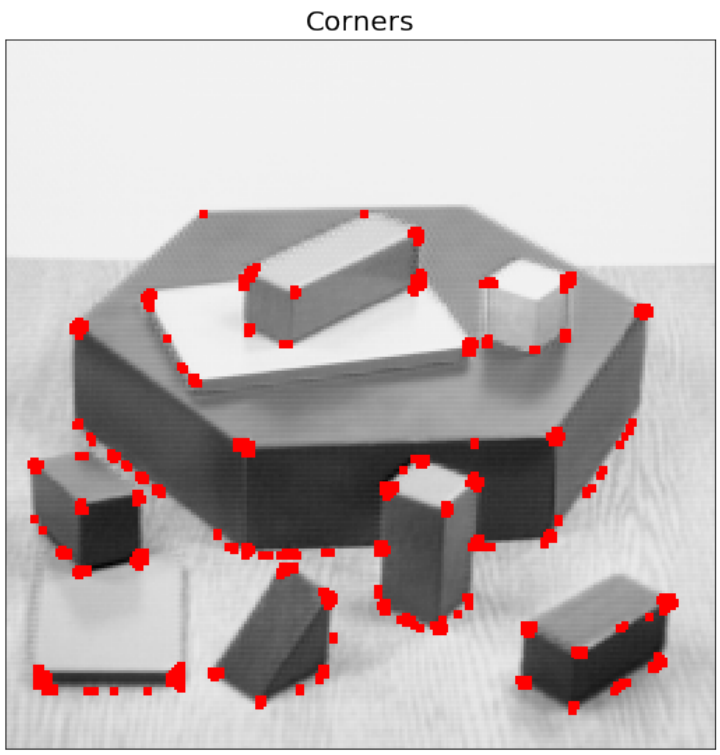
In [40]:

```
twoDots = cv2.cvtColor(img,cv2.COLOR_GRAY2RGB)
twoDots[dst2>0.01*dst2.max()]=[0,0,255]
```

In [41]:

```
fig = plt.figure(figsize=(20,20))
plt.subplot(121)
plt.imshow(img, cmap='gray')
plt.title("Image2",fontsize=20), plt.xticks([]), plt.yticks([])
plt.subplot(122)
plt.imshow(cv2.cvtColor(twoDots, cv2.COLOR_BGR2RGB))
plt.title("Corners",fontsize=20), plt.xticks([]), plt.yticks([])
plt.show()
```





Problem 3

In [42]:

```
import cv2
from pylab import *
import numpy as np
import matplotlib.pyplot as plt

flower = cv2.imread("sunflowers.jpg",0)
from scipy import ndimage
from scipy.ndimage import filters
from scipy import spatial
k = 1.414
sigma = 1.0

flower = flower/255.0
def LoG(sigma):
    #window size
    n = np.ceil(sigma*6)
    y,x = np.ogrid[-n//2:n//2+1,-n//2:n//2+1]
    y_filter = np.exp(-(y*y/(2.*sigma*sigma)))
    x_filter = np.exp(-(x*x/(2.*sigma*sigma)))
    final_filter = (-(2*sigma**2) + (x*x + y*y) ) *  (x_filter*y_filter) * (1/(2*np.
    return final_filter
```

In [43]:

```
def LoG_convolve(img):
    log_images = []
    for i in range(0,9):
        y = np.power(k,i)
        sigma_1 = sigma*y
        filter_log = LoG(sigma_1)
        image = cv2.filter2D(img,-1,filter_log)
        image = np.pad(image,((1,1),(1,1)),'constant')
        image = np.square(image)
        log_images.append(image)
    log_image_np = np.array([i for i in log_images])
    return log_image_np
log_image_flower = LoG_convolve(flower)
```

In [44]:

```
print(log_image_flower.shape)
```

```
(9, 359, 330)
```

In [45]:

```
def detect_blob(log_image_np):
    co_ordinates = []
    (h,w) = flower.shape
    for i in range(1,h):
        for j in range(1,w):
            slice_img = log_image_np[:,i-1:i+2,j-1:j+2]
            result = np.amax(slice_img)
            if result >= 0.03:
                z,x,y = np.unravel_index(slice_img.argmax(),slice_img.shape)
                co_ordinates.append((i+x-1,j+y-1,k**z*sigma))
    return co_ordinates


Fco_ordinates = list(set(detect_blob(log_image_flower)))
```

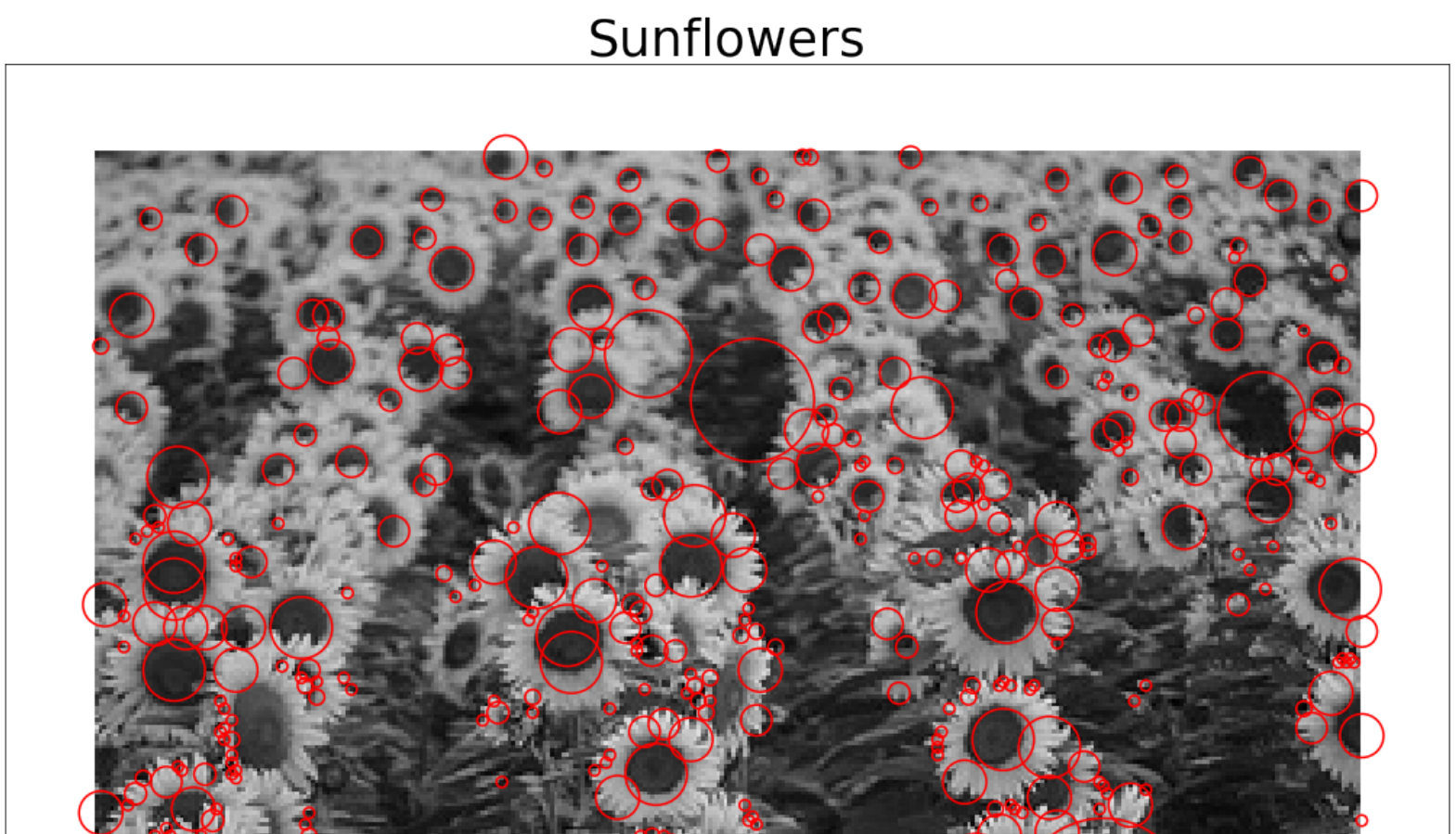
In [46]:

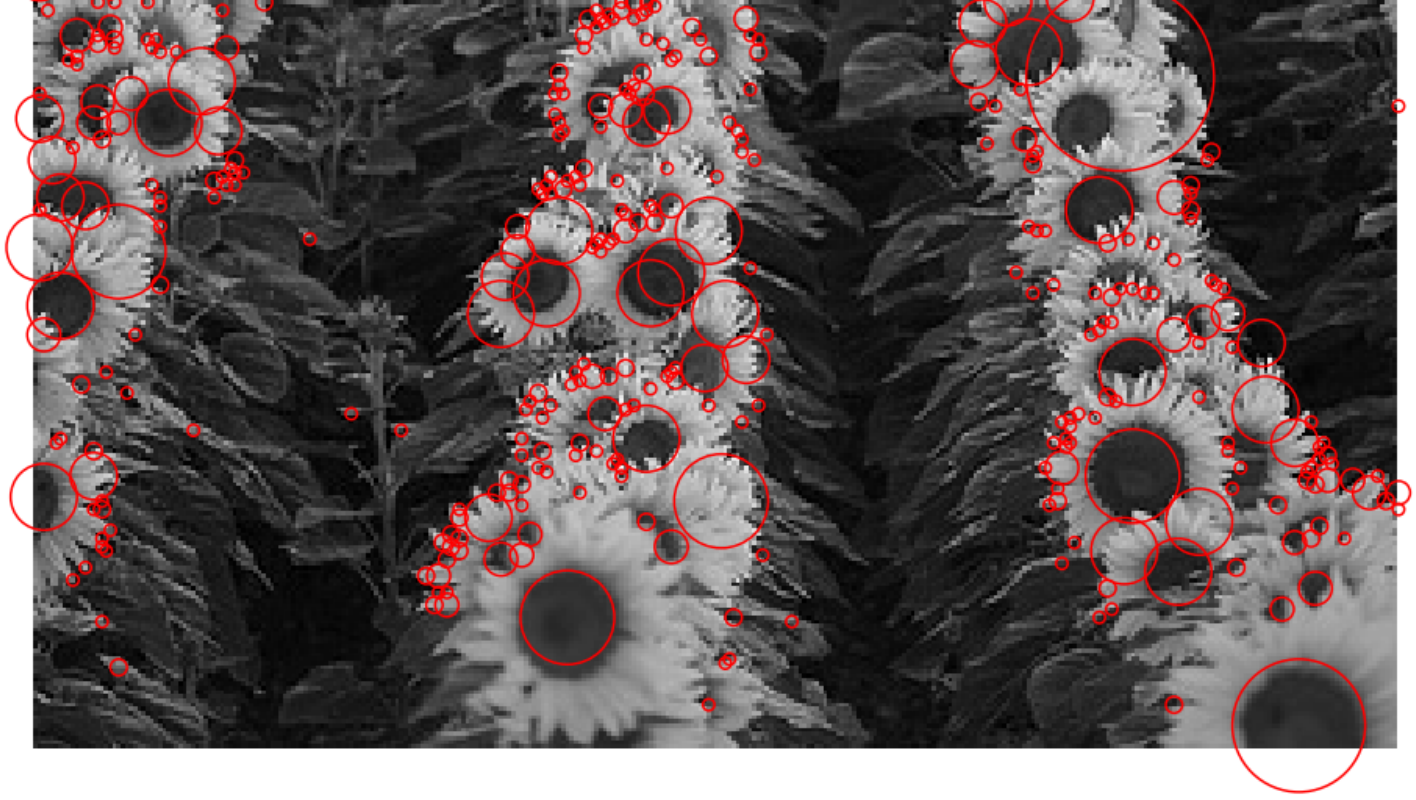
```
def blob_overlap(blob1, blob2):
    n_dim = len(blob1) - 1
    root_ndim = sqrt(n_dim)

    r1 = blob1[-1] * root_ndim
    r2 = blob2[-1] * root_ndim

    d = sqrt(np.sum((blob1[:-1] - blob2[:-1])**2))


    if d > r1 + r2:
        return 0

    elif d <= abs(r1 - r2):
        return 1
    else:

        ratio1 = (d ** 2 + r1 ** 2 - r2 ** 2) / (2 * d * r1)
        ratio1 = np.clip(ratio1, -1, 1)
        acos1 = math.acos(ratio1)
        ratio2 = (d ** 2 + r2 ** 2 - r1 ** 2) / (2 * d * r2)
        ratio2 = np.clip(ratio2, -1, 1)
        acos2 = math.acos(ratio2)
    a = -d + r2 + r1
    b = d - r2 + r1
    c = d + r2 - r1
    d = d + r2 + r1
    area = (r1 ** 2 * acos1 + r2 ** 2 * acos2 -0.5 * sqrt(abs(a * b * c * d)))
    return area/(math.pi * (min(r1, r2) ** 2))

def redundancy(blobs_array, overlap):
```

```
    sigma = blobs_array[:, -1].max()
    distance = 2 * sigma * sqrt(blobs_array.shape[1] - 1)
    tree = spatial.cKDTree(blobs_array[:, :-1])
    pairs = np.array(list(tree.query_pairs(distance)))
    if len(pairs) == 0:
        return blobs_array
    else:
        for (i, j) in pairs:
            blob1, blob2 = blobs_array[i], blobs_array[j]
            if blob_overlap(blob1, blob2) > overlap:
                if blob1[-1] > blob2[-1]:
                    blob2[-1] = 0
                else:
                    blob1[-1] = 0
    return np.array([b for b in blobs_array if b[-1] > 0])

Fco_ordinates = np.array(Fco_ordinates)
Fco_ordinates = redundancy(Fco_ordinates,0.5)
```

In [47]:

```
fig, ax = plt.subplots(figsize=(20,20))
nh,nw = flower.shape
count = 0
ax.imshow(flower, cmap='gray',interpolation='nearest')
for blob in Fco_ordinates:
    y,x,r = blob
    c = plt.Circle((x, y), r*1.414, color='red', linewidth=1.5, fill=False)
    ax.add_patch(c)
ax.plot()
plt.title("Sunflowers",fontsize=35), plt.xticks([]), plt.yticks([])
plt.show()
```





In [ ]:

```
butterfly = cv2.imread("butterfly.jpg",0)
butterfly = butterfly/255.0

log_image_butterfly = LoG_convolve(butterfly)
def detect_blob(log_image_np):
    co_ordinates = []
    (h,w) = butterfly.shape
    for i in range(1,h):
        for j in range(1,w):
            slice_img = log_image_np[:,i-1:i+2,j-1:j+2] #9*3*3 slice
            result = np.amax(slice_img)
            if result >= 0.03:
                z,x,y = np.unravel_index(slice_img.argmax(),slice_img.shape)
                co_ordinates.append((i+x-1,j+y-1,k**z*sigma))
    return co_ordinates

Bco_ordinates = list(set(detect_blob(log_image_butterfly)))
Bco_ordinates = np.array(Bco_ordinates)
Bco_ordinates = redundancy(Bco_ordinates,0.5)



fig, ax = plt.subplots(figsize=(20,20))
nh,nw = butterfly.shape
count = 0
ax.imshow(butterfly, cmap='gray',interpolation='nearest')
for blob in Bco_ordinates:
    y,x,r = blob
    c = plt.Circle((x, y), r*1.414, color='red', linewidth=1.5, fill=False)
    ax.add_patch(c)
ax.plot()
plt.title("Butterfly",fontsize=35), plt.xticks([]), plt.yticks([])
plt.show()
```

In [ ]:

```
einstein = cv2.imread("einstein.jpg",0)
einstein = einstein/255.0
log_image_einstein = LoG_convolve(einstein)


def detect_blob(log_image_np):
    co_ordinates = []
    (h,w) = einstein.shape
    for i in range(1,h):
        for j in range(1,w):
            slice_img = log_image_np[:,i-1:i+2,j-1:j+2]
            result = np.amax(slice_img)
            if result >= 0.03:
                z,x,y = np.unravel_index(slice_img.argmax(),slice_img.shape)
                co_ordinates.append((i+x-1,j+y-1,k**z*sigma))
    return co_ordinates


Eco_ordinates = list(set(detect_blob(log_image_einstein)))
Eco_ordinates = np.array(Eco_ordinates)
Eco_ordinates = redundancy(Eco_ordinates,0.5)
```

```
fig, ax = plt.subplots(figsize=(20,20))
nh,nw = einstein.shape
count = 0
ax.imshow(einstein, cmap='gray',interpolation='nearest')
for blob in Eco_ordinates:
    y,x,r = blob
    c = plt.Circle((x, y), r*1.414, color='red', linewidth=1.5, fill=False)
    ax.add_patch(c)
ax.plot()
plt.title("Einstein",fontsize=35), plt.xticks([]), plt.yticks([])
plt.show()
```

In [ ]:

```
fishes = cv2.imread("fishes.jpg",0)
fishes = fishes/255.0
log_image_fishes = LoG_convolve(fishes)

def detect_blob(log_image_np):
    co_ordinates = []
    (h,w) = fishes.shape
    for i in range(1,h):
        for j in range(1,w):
            slice_img = log_image_np[:,i-1:i+2,j-1:j+2]
            result = np.amax(slice_img)
            if result >= 0.03:
                z,x,y = np.unravel_index(slice_img.argmax(),slice_img.shape)
                co_ordinates.append((i+x-1,j+y-1,k**z*sigma))
    return co_ordinates



fishco_ordinates = list(set(detect_blob(log_image_fishes)))
fishco_ordinates = np.array(fishco_ordinates)
fishco_ordinates = redundancy(fishco_ordinates,0.5)
```

```
fig, ax = plt.subplots(figsize=(20,20))
nh,nw = fishes.shape
count = 0
ax.imshow(fishes, cmap='gray',interpolation='nearest')
for blob in fishco_ordinates:
    y,x,r = blob
    c = plt.Circle((x, y), r*1.414, color='red', linewidth=1.5, fill=False)
    ax.add_patch(c)
ax.plot()
plt.title("Fishes",fontsize=35), plt.xticks([]), plt.yticks([])
plt.show()
```

In [ ]:

In [ ]:

In [ ]:

```

```

In [ ]:

```

```