04_HoughTransform_final

November 20, 2019

1 Assignment 4: Hough Transform

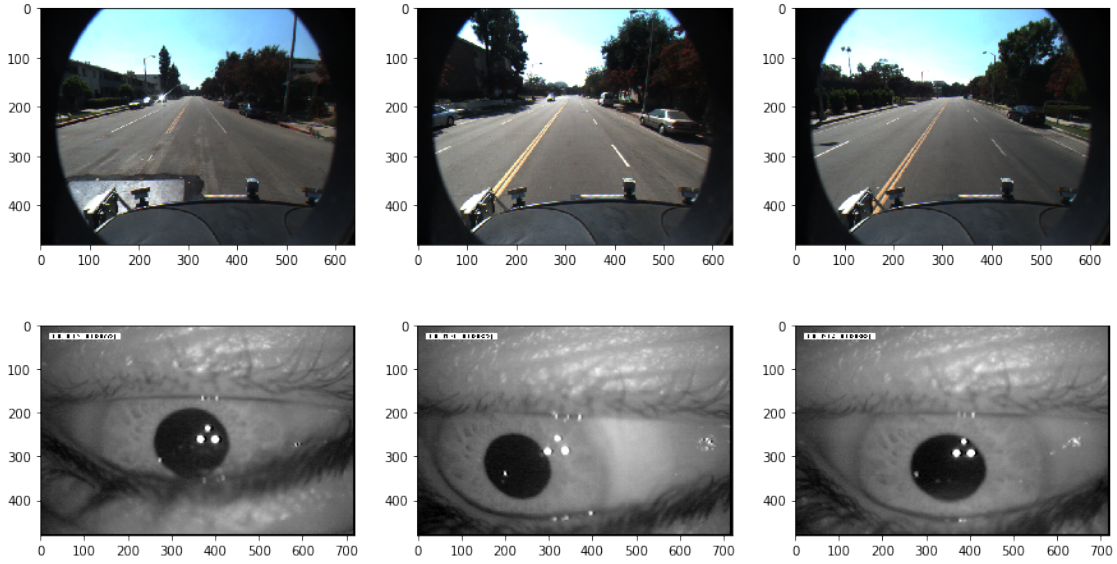
1.1 Ex. 4.1 Detect lanes and eyes

- there are two datasets available: "images/eye_tracking" and "images/lane_detection" - decide for one of them
- implement the classical Hough Transform for lines (for lane detection) **OR** circles (eye tracking) as shown in the lecture
- use a Canny edge detector to produce edge images for the sequence of images

In [24]:
```
%matplotlib inline
import matplotlib.pyplot as plt
from skimage import io, data, feature, color
import numpy as np

lane1 = io.imread('images/lane_detection/f00000.png')
lane2 = io.imread('images/lane_detection/f00050.png')
lane3 = io.imread('images/lane_detection/f00090.png')

eye1 = io.imread('images/eye_tracking/0000.jpeg')
eye2 = io.imread('images/eye_tracking/0050.jpeg')
eye3 = io.imread('images/eye_tracking/0090.jpeg')

fig = plt.figure(figsize=(15, 8))
ax11 = plt.subplot(2, 3, 1)
ax12 = plt.subplot(2, 3, 2)
ax13 = plt.subplot(2, 3, 3)
ax21 = plt.subplot(2, 3, 4)
ax22 = plt.subplot(2, 3, 5)
ax23 = plt.subplot(2, 3, 6)


ax11.imshow(lane1)
ax12.imshow(lane2)
ax13.imshow(lane3)

ax21.imshow(eye1)
```

```
ax22.imshow(eye2)
ax23.imshow(eye3)
```

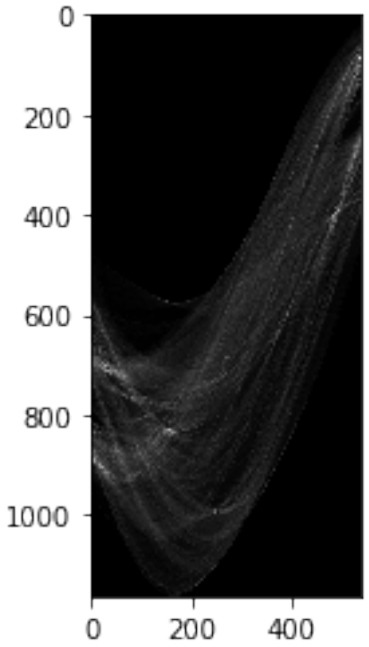
Out[24]: <matplotlib.image.AxesImage at 0x11a1cfc10>



In [25]:
```
from skimage.feature import canny
from skimage.color import rgb2gray


NUM_DISCRETE_ANGLES = 3 * 180
ANGLES = np.linspace(0, np.pi, num=NUM_DISCRETE_ANGLES, endpoint=False)


def get_accumulator(img, *, angles=ANGLES, canny_kwargs={}):
    gray_img = rgb2gray(img)
    edge_img = canny(gray_img, **canny_kwargs).astype(np.int)

    rr, cc = np.where(edge_img == True)

    # Precompute 'cos' and 'sin' for all discrete angles.
    angle_matrix = np.array([np.cos(angles), np.sin(angles)])
    # zip operation
    indices = np.array([rr, cc]).T
    # print(indices.shape, 'Œ', angle_matrix.shape, '=', (indices @ angle_matrix).sha

    # Matrix of all values for 'r' as ints so they can be used as indices for 'A'.
    R = np.round(indices @ angle_matrix).astype(np.int)

    # 'r' can be negative (e.g. (x=1, y=1), a=179ř => r = cos(179) + sin(179) < 0)
```

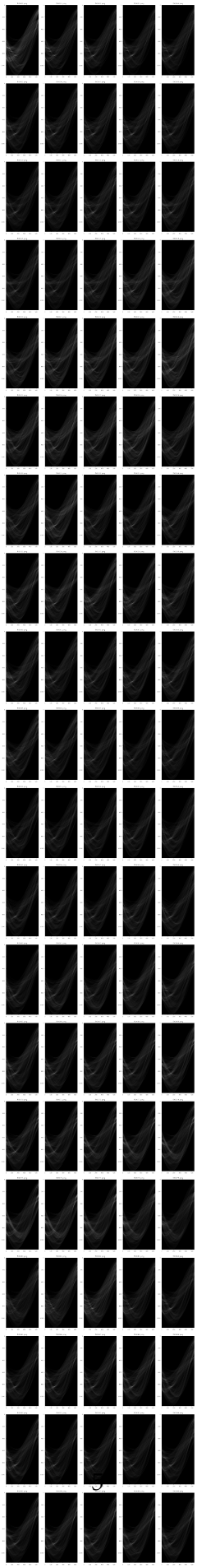
```
            # Since we always have all angles from 0 - 180 degrees and most likely at least 1
            # the above scenario will most certainly happen. Thus we can assume that 'min_r'
            # We also set the size of 'A' (for 'r') dynamically instead using 2*sqrt(width**2
            # in order to reduce the accumulator size (=> performance).
            min_r, max_r = np.min(R), np.max(R)
            offset_r = -min_r
            # NOTE: +1 for the zero.
            A = np.zeros((max_r + offset_r + 1, len(angles)))

            for angle_index in range(0, R.shape[1]):
                # Count occurences of distinct values of 'r' for each column (== discrete ang
                r_s, counts = np.unique(R[:, angle_index], return_counts=True, axis=0)
                a_s = np.repeat(angle_index, r_s.shape[0])
                A[r_s + offset_r, a_s] = counts
            return A, offset_r


        acc, offset = get_accumulator(lane1, canny_kwargs=dict(sigma=1.7))
        plt.imshow(acc, cmap='gray')
```

Out[25]: <matplotlib.image.AxesImage at 0x1243e2510>



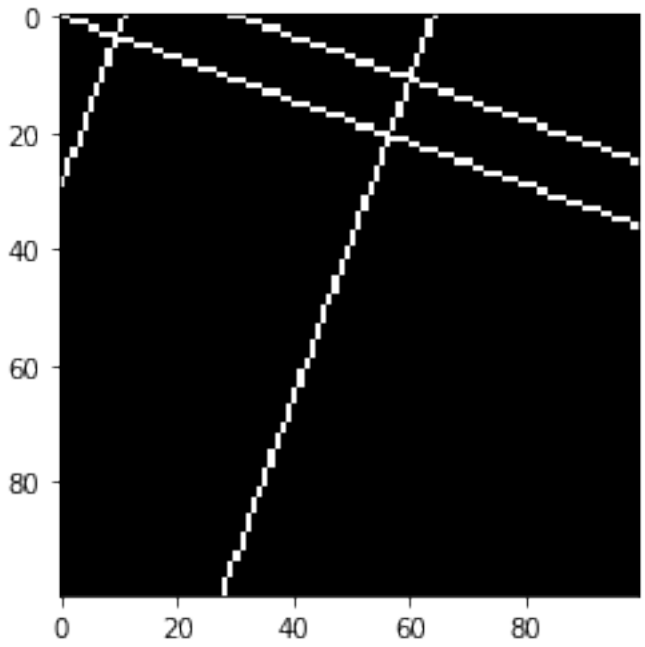
In [27]:
```
from math import ceil
from multiprocessing import Pool
```

```
def _get_acc(frame):
    return get_accumulator(frame, canny_kwargs=dict(sigma=1.7))


frames = io.imread_collection("images/lane_detection/*")
pool = Pool()
accs_with_offset = pool.map(_get_acc, frames)

N = len(accs_with_offset)
NCOLS = 5
NROWS = ceil(N / NCOLS)
fig, axes = plt.subplots(
    nrows=NROWS,
    ncols=NCOLS,
)
f = 3.6
fig.set_figwidth(f*NCOLS)
fig.set_figheight(f*NROWS*2)

plt.subplots_adjust(wspace=.01)
fig.tight_layout()
axes = np.asarray(axes).ravel()
for i, acc_with_offset in enumerate(accs_with_offset):
    acc, offset = acc_with_offset
    axes[i].imshow(acc, cmap='gray')
    axes[i].set_title(f'f000{i:02}.png')
```



In [28]:
```
# dieser Code wurde als Musterlösung von Sebastian Oltmanns zur Verfügung gestellt un

%matplotlib inline
from skimage.draw import line
from skimage import io
import math
import numpy as np

def draw_line_hessian_normal(image, a, r, color=1):
    dimy, dimx = image.shape[:2]

    # r = x*cos(a) + y*sin(a)
    rad = math.radians(a)
    cos = math.cos(rad)
    # Prevent division by zero.
    sin = math.sin(rad) or 1e-4

    # compute start and end point of line
    x0 = 0
    y0 = round((r - x0*cos) / sin)
    x1 = dimx - 1
    y1 = round((r - x1*cos) / sin)

    # print(x0, y0, x1, y1)

    liney, linex = line(y0, x0, y1, x1)
    min_y, max_y = 0, dimy - 1
    # Clip indices to shape of given image.
    ii = np.where((liney >= min_y) & (liney <= max_y))
    # print(linex[ii][0], liney[ii][0], linex[ii][-1], liney[ii][-1])

    ret = np.copy(image)
    ret[liney[ii], linex[ii]] = color
    return ret


image = np.zeros((100,100))
image = draw_line_hessian_normal(image, 20, 60)
image = draw_line_hessian_normal(image, 20, 10)
image = draw_line_hessian_normal(image, 110, 0)
image = draw_line_hessian_normal(image, -70, 10)
plt.imshow(image, cmap='gray')
```

Out[28]:
```
<matplotlib.image.AxesImage at 0x137ce45d0>
```



1.2 OPTION 1: line detection for lane detection

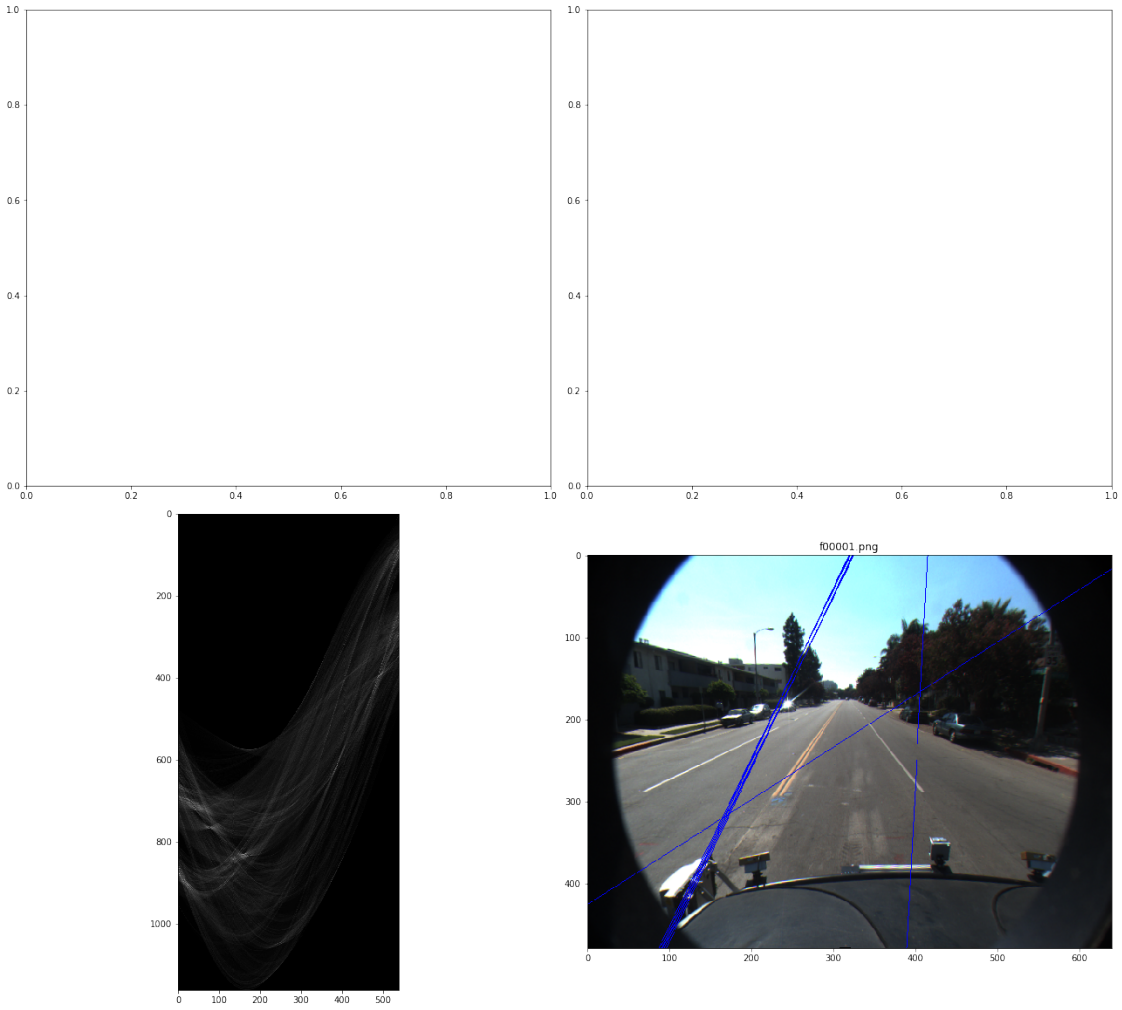
- use your implementation of the Hough Transform to find the 10 strongest lines in the image
- display your result set (draw those lines on the image) (**RESULT**)
- can you improve the performance by limiting the space of solutions? implement and draw lines again! (**BONUS**)

In [29]:
```
from math import degrees


def top_n_indexes(arr, n):
    """See https://gist.github.com/tomerfiliba/3698403"""
    idx = np.argpartition(arr, arr.size - n, axis=None)[-n:]
    width = arr.shape[1]
    return [divmod(i, width) for i in idx]


# def draw_line_hessian_normal(image, a, r, color=1):
#     dimy, dimx = image.shape[:2]

# #     a += 45

#     # r = x*cosa + y*sina
#     rad = math.radians(a)
#     cos = math.cos(rad)
#     # Prevent division by zero.
```

```
#     sin = math.sin(rad) or 1e-4

#     # compute start and end point of line
#     x0 = 0
#     y0 = round((r - x0*cos) / sin)
#     x1 = dimx - 1
#     y1 = round((r - x1*cos) / sin)

#     print(x0, y0, x1, y1)

# #     dx, dy = 140, -150
#     dx, dy = 0, 0
#     liney, linex = line(y0+dy, x0+dx, y1+dy, x1+dx)
# #     min_y, max_y = 0, dimy - 1
#     # Clip indices to shape of given image.
#     ii = np.where((liney >= 0) & (liney <= dimy - 1) & (linex >= 0) & (linex <= dim
#     print(linex[ii][0], liney[ii][0], linex[ii][-1], liney[ii][-1])

#     ret = np.copy(image)
#     ret[liney[ii], linex[ii]] = color
#     return ret



_accs_with_offset = accs_with_offset[0:2]
N = len(_accs_with_offset) * 2
NCOLS = 2
NROWS = ceil(N / NCOLS)
fig, axes = plt.subplots(
    nrows=NROWS,
    ncols=NCOLS,
)
f = 18 / NCOLS
fig.set_figwidth(f*NCOLS)
fig.set_figheight(f*NROWS*0.9)
plt.subplots_adjust(wspace = .01)
fig.tight_layout()
axes = np.asarray(axes).ravel()

# frames_with_lines = []
line_color = np.array([0, 0, 255])
for i, acc_with_offset in enumerate(_accs_with_offset):
    if i == 0:
        continue

    acc, offset = acc_with_offset
    top_indices = top_n_indexes(acc, n=8)
    print(top_indices, end='\n\n')
```

```
            img = frames[i]
            for index in top_indices:
                r_with_offset, angle_index = index
                r = int(r_with_offset - offset)
                a = degrees(ANGLES[angle_index])
                print(f'r={r}, ={a}')
                img = draw_line_hessian_normal(img, a, r, line_color)
            axes[2*i].imshow(acc, cmap='gray')
            axes[2*i + 1].imshow(img)
            axes[2*i + 1].set_title(f'f000{i:02}.png')
        #     frames_with_lines.append(img)

        # io.imshow_collection(frames_with_lines)

        None
```

```
[(890, 9), (764, 78), (768, 77), (835, 172), (764, 77), (764, 76), (768, 76), (768, 78)]

r=414, =3.0000000000000004
r=288, =26.0
r=292, =25.666666666666664
r=359, =57.333333333333336
r=288, =25.666666666666664
r=288, =25.333333333333332
r=292, =25.333333333333332
r=292, =26.0
```



1.3 OPTION 2: circle detection for eye detection

- use your implementation of the Hough Transform to find the 10 strongest circles in the image
- display your result set (draw those circles on the image) (**RESULT**)
- can you improve the performance by limiting the space of solutions? implement and draw circles again! (**BONUS**)

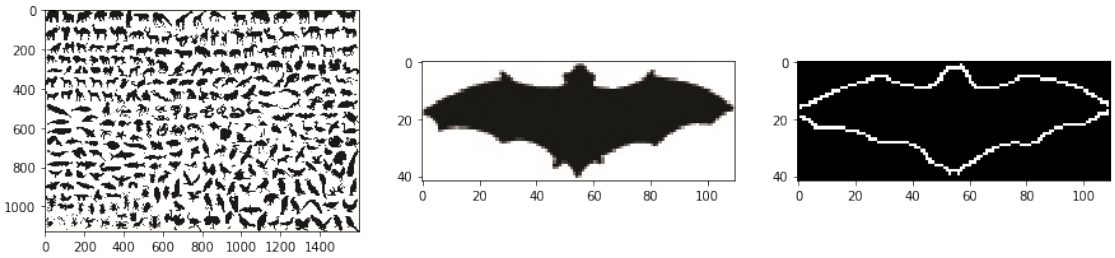
```
In [30]: None
```

2 Ex. 4.2 Generalized Hough Transform

- implement the Generalized Hough Transform as described in the lecture for localizing a given template
- find the given template (see below) and mark its location in the image "animals.png" (**RESULT**)

In [31]:
```
%matplotlib inline
import matplotlib.pyplot as plt
from skimage import io, data, feature, color
import numpy as np

animals = io.imread('images/animals.png')

xmin = 1271
xmax = 1381
ymin = 519
ymax = 561
# xmin = 825
# xmax = xmin + 75
# ymin = 561
# ymax = ymin + 57

bat = animals[ymin:ymax, xmin:xmax]
bat_edge = feature.canny(color.rgb2gray(bat), 2)

fig = plt.figure(figsize=(15, 10))
ax1 = plt.subplot(1, 3, 1)
ax2 = plt.subplot(1, 3, 2)
ax3 = plt.subplot(1, 3, 3)

ax1.imshow(animals)
ax2.imshow(bat)
ax3.imshow(bat_edge, cmap='gray')
```

Out[31]: <matplotlib.image.AxesImage at 0x1368d8250>

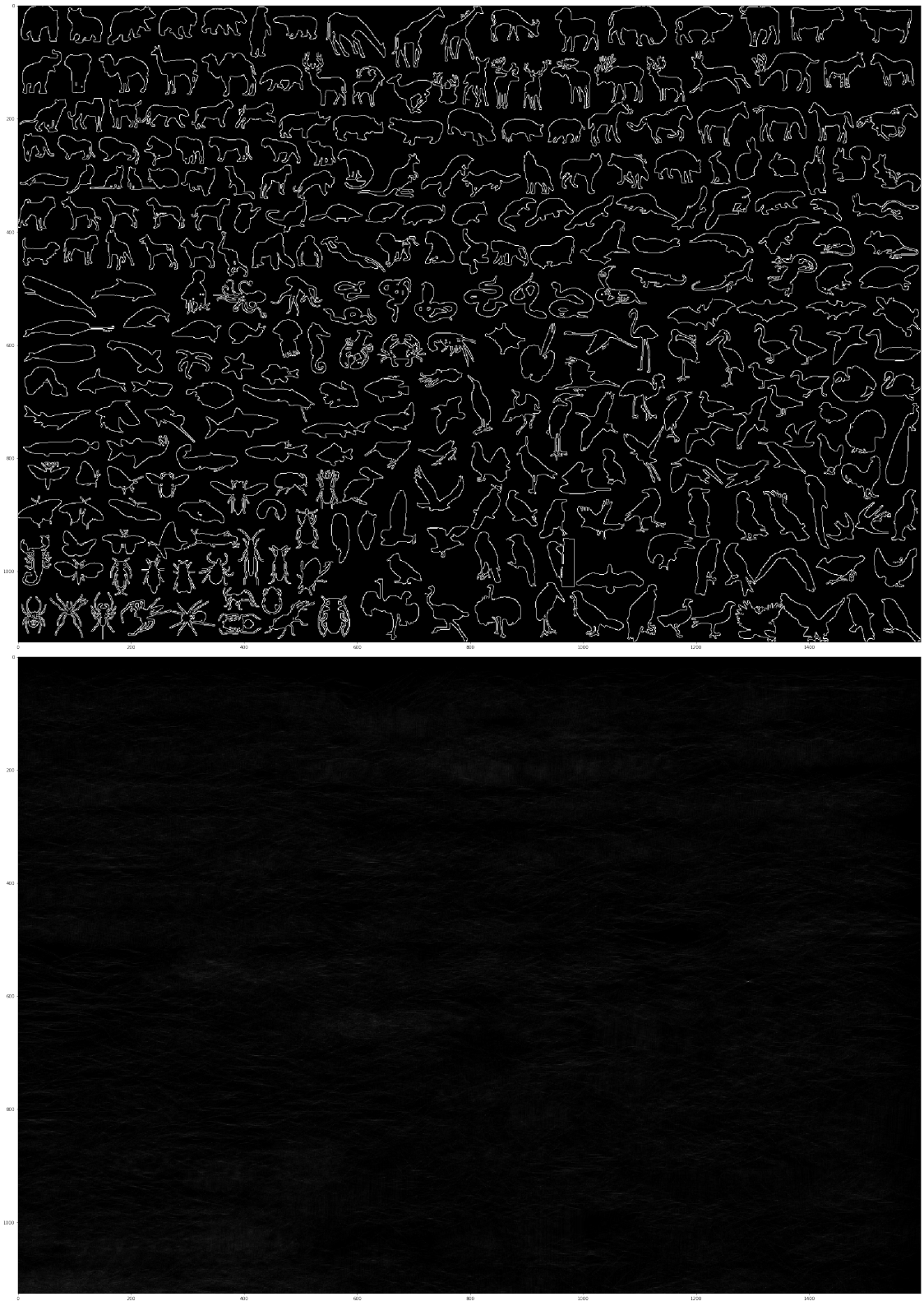


In [32]:
```
from collections import defaultdict
from pprint import pprint

from skimage.filters import sobel_h, sobel_v


# Get center of the template.
```

```
shape_x, shape_y = bat_edge.shape
center = np.array([shape_x // 2, shape_y // 2])

G_x = sobel_v(bat_edge)
G_y = sobel_h(bat_edge)

# Build an r-table with 20ř theta steps.
# (see 02_convolution_canny.ipynb for details)
NUMBER_BINS = 19
bins = np.linspace(-180, 180, num=NUMBER_BINS)
angles = np.degrees(np.arctan2(G_y, G_x))
indices = np.digitize(angles, bins)
theta = np.zeros(angles.shape)
theta[:] = bins[(indices[:] - 1) % (NUMBER_BINS - 1)]
print(theta)


r_table = defaultdict(list)
bat_edge_pixels = np.where(bat_edge == True)
for row, col in zip(*bat_edge_pixels):
    angle = theta[row, col]
    p = np.array([col, row])
    v = center - p
    r_table[angle] += [v]
# pprint(r_table)


# Locate the object.
animals_edge = feature.canny(color.rgb2gray(animals), 0.01, low_threshold=0, high_thre
G_x = sobel_v(animals_edge)
G_y = sobel_h(animals_edge)
angles = np.degrees(np.arctan2(G_y, G_x))
indices = np.digitize(angles, bins)
theta = np.zeros(angles.shape)
theta[:] = bins[(indices[:] - 1) % (NUMBER_BINS - 1)]
animals_edge_pixels = np.where(animals_edge == True)
acc = np.zeros(animals_edge.shape)
for row, col in zip(*animals_edge_pixels):
    angle = theta[row, col]
    p = np.array([col, row])
    for v_i in r_table[angle]:
        x, y = p + v_i
        t = p + v_i
        if 0 <= t[0] < animals.shape[1] and 0 <= t[1] < animals.shape[0]:
            acc[y, x] += 1
print(acc)
```

```
        fig, axes = plt.subplots(
            nrows=2,
            ncols=1,
        )
        fig.set_figwidth(40)
        fig.set_figheight(40)
        fig.tight_layout()
        axes[0].imshow(animals_edge, cmap='gray')
        axes[1].imshow(
            acc,
            cmap='gray',
        #     vmin=0,
        #     vmax=np.max(acc),
        )
        fig.savefig("temp.png")
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [5. 6. 5. ... 0. 0. 0.]
 [5. 6. 5. ... 0. 0. 0.]
 [5. 4. 7. ... 0. 0. 0.]]
```

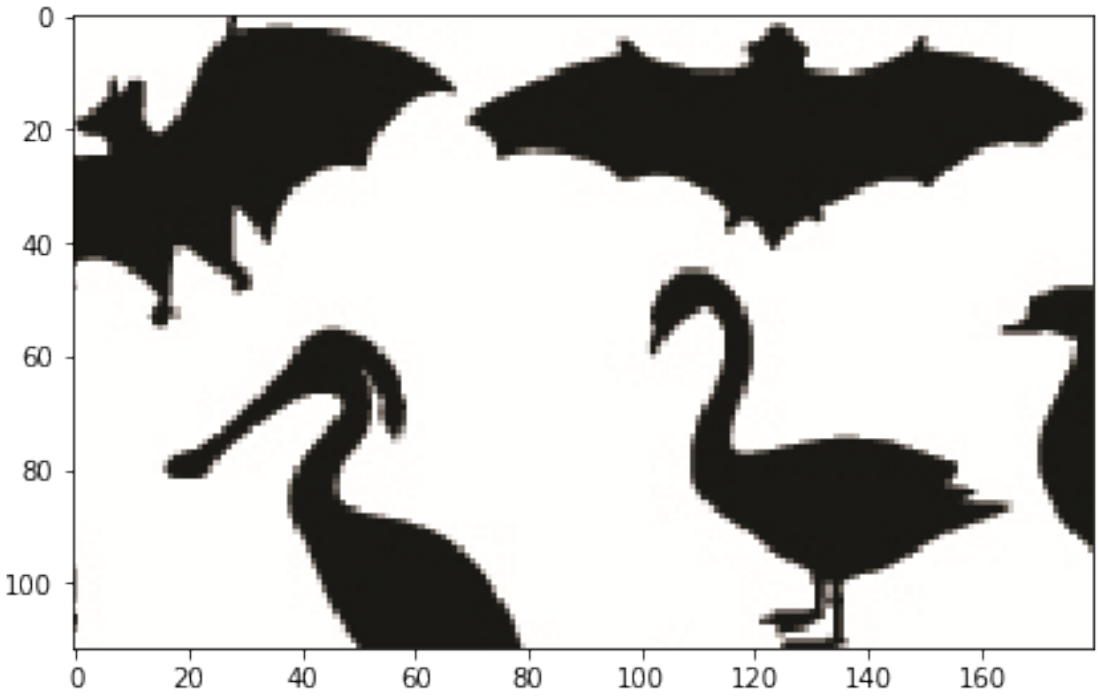


```
In [33]: def top_n_indexes(arr, n):
```

```
        """See https://gist.github.com/tomerfiliba/3698403"""
        idx = np.argpartition(arr, arr.size - n, axis=None)[-n:]
        width = arr.shape[1]
        return [divmod(i, width) for i in idx]


    top_indices = top_n_indexes(acc, n=1)
    best_matched_center = top_indices[0]
    cy, cx = best_matched_center
    print('center:', cy, cx)
    bat_height, bat_width = bat.shape[:2]
    padding = 35

    extracted_piece = animals[
        max(cy - bat_height//2 - padding, 0):min(cy + bat_height//2 + padding, animals.sha
        max(cx - bat_width//2 - padding, 0):min(cx + bat_width//2 + padding, animals.shape
    ]
    io.imshow(extracted_piece)
```

```
center: 574 1292
```

Out[33]: <matplotlib.image.AxesImage at 0x13229da50>



2.1 BONUS

- now implement an extended version of the GHT that find rotated and scaled variants of the template.
- find Italy (see “italy.jpg”) and the map of Europe (“europe_map_political.gif”)
- note that you can binarize your italy template by using a simple color lookup
- draw the location of italy on the map and print its scale and orientation (**BONUS**)