In [1]:
```
import math
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import uniform
from matplotlib.image import imread
```

Homework 2 (4)

(S) Introducing noise

In step (J), you have the images $S_L(x,y)$ and $S_R(x,y)$. Add noise $N_L(x,y)$ and $N_R(x,y)$ to them, such that

$$S_L(x,y) \to S_L(x,y) + N_L(x,y)$$

$$S_R(x,y) \to S_R(x,y) + N_R(x,y)$$

At each pixel location (x,y), the noise $N_L(x,y)$ and $N_R(x,y)$ are independent of each other, and is a random number uniformly distributed in the range between $-N_{max}$ and N_{max}, i.e., $-N_{max} \le N_L(x,y) \le N_{max}$ and $-N_{max} \le N_R(x,y) \le N_{max}$. Noise in different pixels should be independent random numbers in this range. Choose $N_{max} = \sqrt{(R^S_{11} + R^S_{22})/2}$ (or you can choose a larger or smaller N). Plot out the noisy images $S_L(x,y)$ and $S_R(x,y)$. With these noisy images, repeat steps (B)-(R), and observe what happens and reflect on the result. You can play with the value of N_{max} and see different effects by different N_{max} values, and try to reflect why.

In [2]:
```
def noise(nmax = 1, n = 1):
    return uniform.rvs(loc = -nmax, scale = nmax, size = n)
```

In [3]:
```
def noisy_image(original, nmax = 1):
    noise_data = np.reshape(noise(nmax, n = np.size(original)), newshape = original.shape)
    return original + noise_data
```

In [4]:
```
def covariance(x, y):
    _x = x - np.mean(x)
    _y = y - np.mean(y)
    N = np.size(x)

    return np.array([
        [np.sum(_x * _x) / N, np.sum(_x * _y) / N],
        [np.sum(_y * _x) / N, np.sum(_y * _y) / N]
    ])
```

In [5]:
```
def correlation_coefficient(x, y):
    _x = x - np.mean(x)
    _y = y - np.mean(y)

    return np.sum(_x * _y) / np.sqrt(np.sum(_x * _x) * np.sum(_y * _y))
```

In [6]:
```
def correlation(x, y):
    return np.array([
        [correlation_coefficient(x, x), correlation_coefficient(x, y)],
        [correlation_coefficient(y, x), correlation_coefficient(y, y)]
    ])
```

In [7]:
```
left_original = imread("resources/left.png")
right_original = imread("resources/right.png")
```

In [8]:
```
r = correlation(left_original, right_original)
```

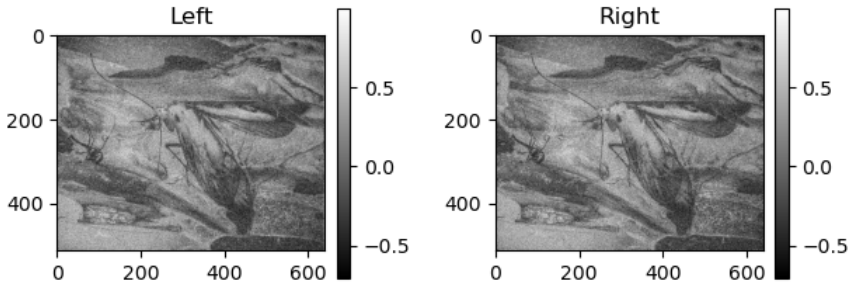
In [9]:
```
nmax = math.sqrt(r[0][0] + r[1][1]) / 2

print(f"Using nmax = {nmax}")

left_original = noisy_image(left_original, nmax = nmax)
right_original = noisy_image(right_original, nmax = nmax)
```

```
Using nmax = 0.7071067811865476
```

In [10]:
```
def plot_left_and_right(left, right, apply_lim = False, title_left = "Left", title_right = "Right"):
    max_lim = np.max(np.abs([left, right]))

    plt.subplot(1, 2, 1)
    if apply_lim:
        plt.imshow(left, cmap = "gray", vmin = -max_lim, vmax = max_lim)
    else:
        plt.imshow(left, cmap = "gray")
    plt.colorbar(fraction = 0.046, pad = 0.04)
```

```
    if title_left != "":
        plt.title(title_left)

    plt.subplot(1, 2, 2)
    if apply_lim:
        plt.imshow(right, cmap = "gray", vmin = -max_lim, vmax = max_lim)
    else:
        plt.imshow(right, cmap = "gray")
    plt.colorbar(fraction = 0.046, pad = 0.04)

    if title_right != "":
        plt.title(title_right)

    plt.tight_layout()
    plt.show()
```

In [11]:
```
plot_left_and_right(left_original, right_original)
```



(B) Normalizing the images

Please normalize each image as follows. For $S_i(x, y)$, with $i = L$ or $i = R$, find S_i^{min} and S_i^{max} as the mininum and maximum of $S_i(x, y)$ across all pixel locations (x, y). Then, for $\hat{S} = 255$, do

$$ S_i(x, y) \rightarrow \hat{S} \frac{S_i(x, y) - S_i^{min}}{S_i^{max} - S_i^{min}} $$

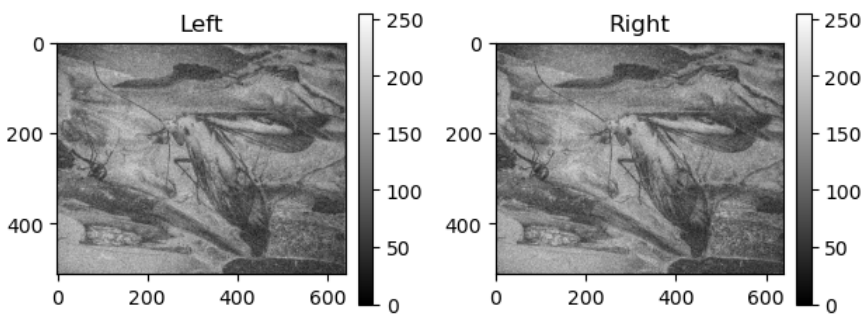
Now $0 \leq S_i(x, y) \leq \hat{S}$. Round each $S_i(x, y)$ into an integer value so that $S_i(x, y)$ is an integer between 0 and $\hat{S}$.

In [12]:
```
def normalize(image, s_hat = 255):
    s_min = np.min(image)
    s_max = np.max(image)

    image = s_hat * (image - s_min) / (s_max - s_min)
    image = np.round(image).astype(np.uint8)

    return image
```

In [13]:
```
left = normalize(left_original)
right = normalize(right_original)

print(f"Before normalization left: min = {np.min(left_original)}, max = {np.max(left_original)}")
print(f"After normalization left: min = {np.min(left)}, max = {np.max(left)}")

print(f"Before normalization right: min = {np.min(right_original)}, max = {np.max(right_original)}")
print(f"After normalization right: min = {np.min(right)}, max = {np.max(right)}")
```

```
Before normalization left: min = -0.7055875859343761, max = 0.9990245389659125
After normalization left: min = 0, max = 255
Before normalization right: min = -0.7047442708259074, max = 0.9985928893377534
After normalization right: min = 0, max = 255
```

In [14]:
```
plot_left_and_right(left, right)
```

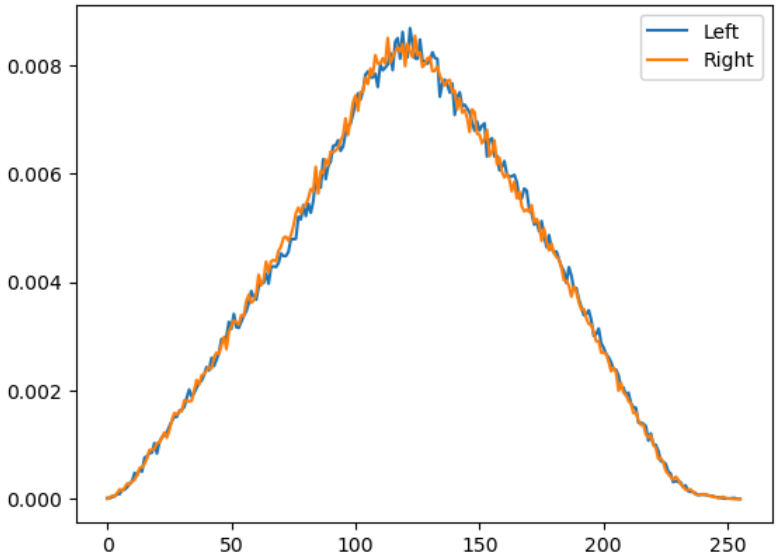


(C) Signal probability

For each $S_i(x, y)$, with $i = L$ or $i = R$, calculate the probability $P(S)$ for $S_i(x, y) = S$ across all (x, y). This means, for each $S = 0, 1, 2, \ldots, \hat{S}$, let $n(S)$ be the number of pixels (x, y) for which $S_i(x, y) = S$, and let N be the total number of pixels in S_i, then $P(S) = \frac{n(S)}{N}$. Plot out $P(S)$ vs S for each image S_i.

In [15]:
```
def probability(x, min_val = 0, max_val = 255):
    bins = np.linspace(min_val, max_val + 1, num = max_val + 2)
    return [bins[:-1], np.histogram(x, bins = bins)[0] / np.size(x)]
```

In [16]:
```
s, p = probability(left)
plt.plot(s, p, label = "Left")

s, p = probability(right)
plt.plot(s, p, label = "Right")

plt.legend()
plt.show()
```



(D) Signal entropy

For each $S_i(x, y)$, with $i = L$ or $i = R$, calculate the pixel entropy

$$H(S_i) = -\sum_{S_i} P(S_i)\ log_2\ P(S_i)$$

Please note that, if for some values S you have $P(S) = 0$, in such a case $P(S)\ log_2\ P(S) = 0$. Your computer will complain if you try to calculate $log_2\ P(S)$ when $P(S) = 0$. So omit these S values with zero $P(S)$ when doing the sum above.

Write out what $H(S)$ is for each image S_i.

In [17]:
```
def entropy(probs):
    return -np.sum([p * np.log2(p) if p > 0 else 0 for p in probs])
```

In [18]:
```
_, p = probability(left)
print(f"Entropy for i = L: {round(entropy(p), 2)}")

_, p = probability(right)
print(f"Entropy for i = R: {round(entropy(p), 2)}")
```

```
Entropy for i = L: 7.56
Entropy for i = R: 7.56
```

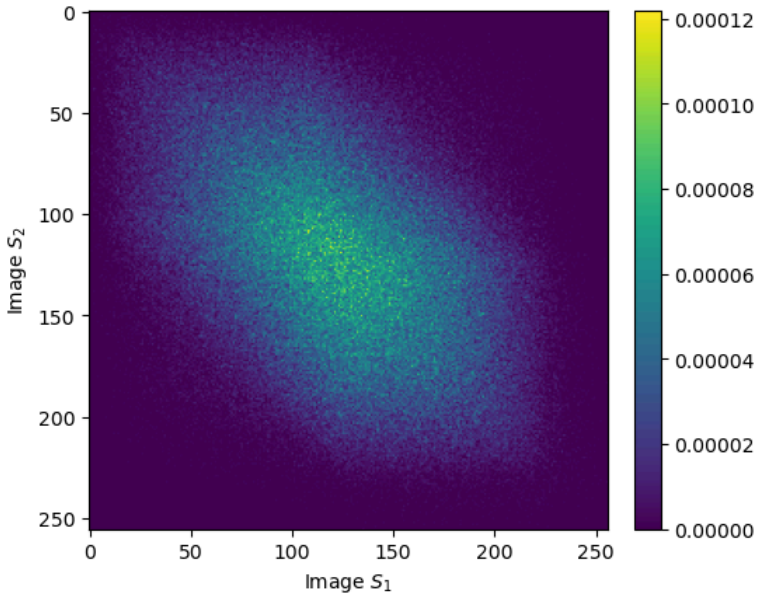
(E) Joint probability

Calculate the joint probability $P(S_1, S_2)$ of $S_L(x, y) = S_1$ and $S_R(x, y) = S_2$ for $S_1 = 0, 1, 2, \ldots, \hat{S}$ and $S_2 = 0, 1, 2, \ldots, \hat{S}$. This means, for each possible pair of values (S_1, S_2), go across all pixels (x, y) to find $n(S_1, S_2)$ as the number of pixels satisfying $S_L(x, y) = S_1$ and $S_R(x, y) = S_2$. Then $P(S_1, S_2) = \frac{n(S_1, S_2)}{N}$. Plot out $P(S_1, S_2)$ (which is the joint probability distribution function) versus S_1 and S_2.

In [19]:
```
def joint_probability(x, y, min_val = 0, max_val = 255):
    counts, _, _ = np.histogram2d(x.flatten(), y.flatten(), bins = max_val + 1)

    return counts / np.size(x)
```

In [20]:
```
probs = joint_probability(left, right)
```

In [21]:
```
plt.imshow(probs)
plt.xlabel(r'Image $S_1$')
plt.ylabel(r'Image $S_2$')
plt.colorbar(fraction = 0.046, pad = 0.04)
plt.show()
```



(F) Joint entropy

Calculate joint entropy as

$$H(S_1, S_2) = -\sum_{S_1, S_2} P(S_1, S_2)\ log_2\ P(S_1, S_2)$$

Write out the value for H(S_1, S_2).

In [22]:
```
print(f"Joint entropy: {round(entropy(probs.flatten()), 2)}")
```

```
Joint entropy: 14.85
```

(G) Mutual information

Calculate mutual information between corresponding pixels in the two images as

$$I(S_1, S_2) = \sum_{S_1, S_2} P(S_1, S_2)\ log_2\ \frac{P(S_1, S_2)}{P(S_1)P(S_2)} = H(S_1) + H(S_2) - H(S_1, S_2)$$

Write out the value for $I(S_1, S_2)$.

In [23]:
```
def mutual_information(x, y, min_val = 0, max_val = 255):
    _, p = probability(x, min_val = min_val, max_val = max_val)
    entropy_x = entropy(p)

    _, p = probability(y, min_val = min_val, max_val = max_val)
    entropy_y = entropy(p)

    p = joint_probability(x, y, min_val = min_val, max_val = max_val)
    entropy_xy = entropy(p.flatten())

    return entropy_x + entropy_y - entropy_xy
```

In [24]:
```
print(f"Mutual information: {round(mutual_information(left, right), 2)}")
```

```
Mutual information: 0.27
```

(H) Redundancy

Calculate the redundancy between the left and right eye images as

$$\text{Redundancy} = \frac{H(S_1) + H(S_2)}{H(S_1, S_2)} - 1$$

In [25]:
```
def redundancy(x, y, min_val = 0, max_val = 255):
    _, p = probability(x, min_val = min_val, max_val = max_val)
    entropy_x = entropy(p)

    _, p = probability(y, min_val = min_val, max_val = max_val)
    entropy_y = entropy(p)

    p = joint_probability(x, y, min_val = min_val, max_val = max_val)
    entropy_xy = entropy(p.flatten())

    return ((entropy_x + entropy_y) / entropy_xy) - 1
```

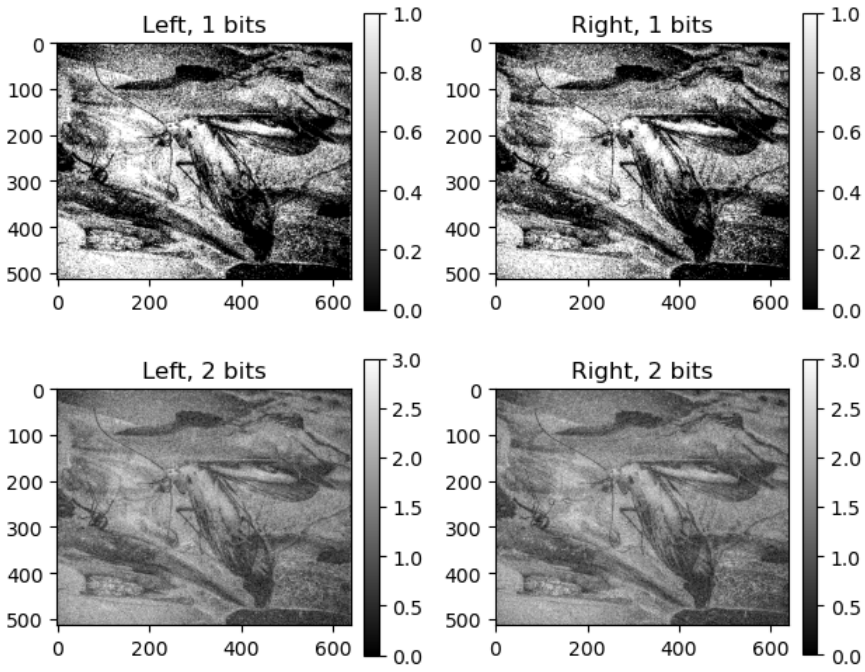
In [26]:
```
print(f"Redundancy: {round(redundancy(left, right), 2)}")
```

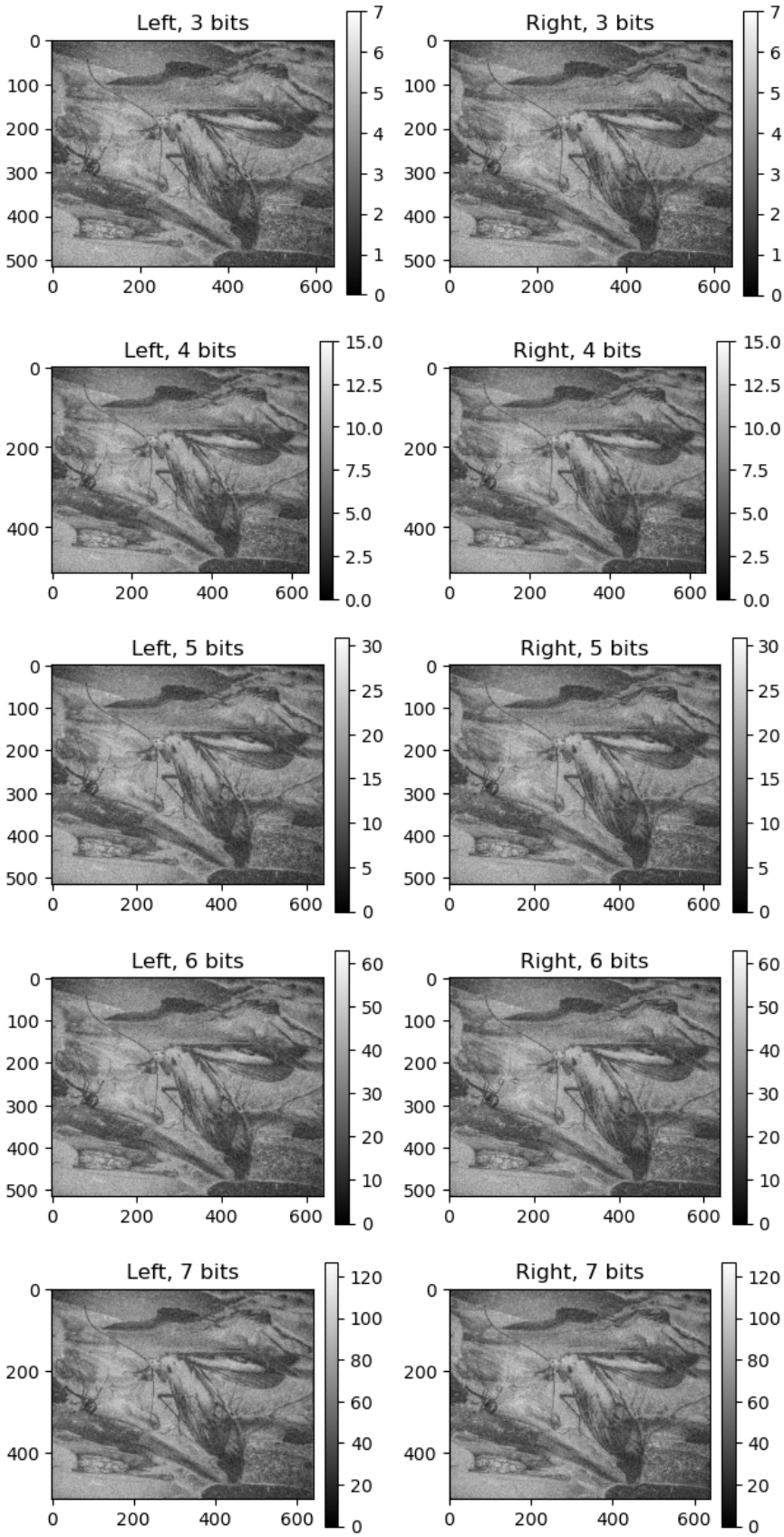
```
Redundancy: 0.02
```

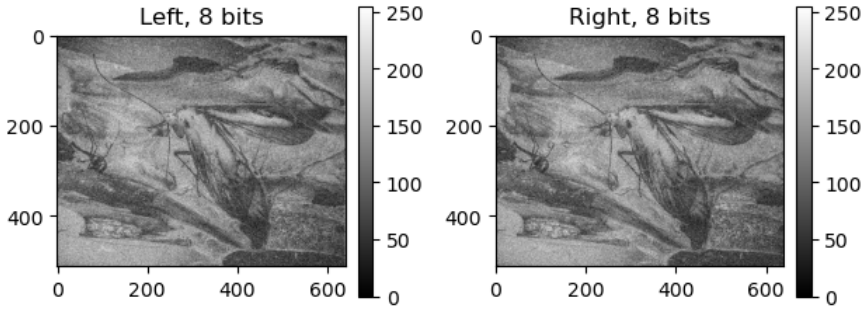
(I) Using n bits to present each image pixel

So far, you have done (B)-(H) when the highest pixel value is $\hat{S} = 255$. Now, repeat (B)-(H) for $\hat{S} = 127, 63, 31, 15, 7, 3, 1$. In other words, you can take $\hat{S} = 2^n - 1$ for $n = 1, 2, 3, \ldots, 8$ (so that $\hat{S} = 255$ when $n = 8$), so that you use n bits to present each image pixel. Plot $H(S_i)$, $H(S_1, S_2)$, $I(S_1, S_2)$ and Redundancy as functions of n. Also, please plot out the two images for each n value, and see if they make sense.

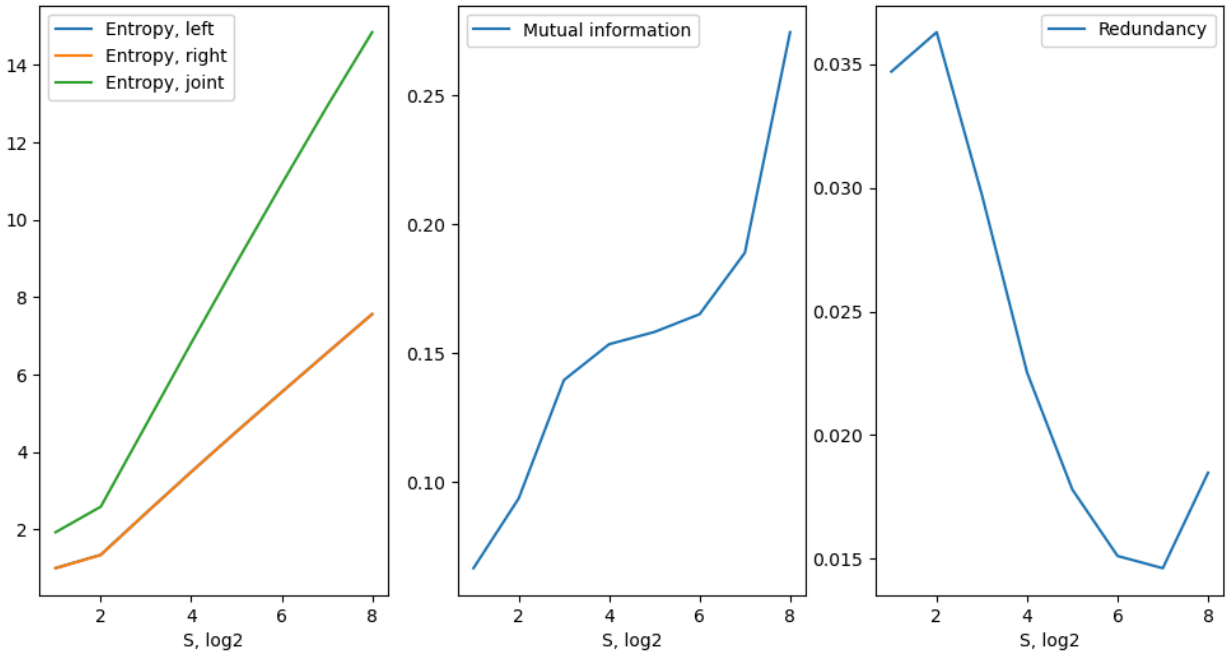
In [27]:
```
h_l = []
h_r = []
h_rl = []
i_rl = []
red = []
n_bits = []

for n_bit in range(1, 9):
    s_hat = round(math.pow(2, n_bit) - 1)
    n_bits.append(n_bit)

    _left = normalize(left_original, s_hat = s_hat)
    _right = normalize(right_original, s_hat = s_hat)

    plot_left_and_right(_left, _right, title_left = f"Left, {n_bit} bits", title_right = f"Right, {n_bit} bits"

    _, p = probability(_left, max_val = s_hat)
    h_l.append(entropy(p))

    _, p = probability(_right, max_val = s_hat)
    h_r.append(entropy(p))

    p = joint_probability(_left, _right, max_val = s_hat)
    h_rl.append(entropy(p.flatten()))

    i_rl.append(mutual_information(_left, _right))

    red.append(redundancy(_left, _right, max_val = s_hat))
```




Left, 3 bits
Right, 3 bits
Left, 4 bits
Right, 4 bits
Left, 5 bits
Right, 5 bits
Left, 6 bits
Right, 6 bits
Left, 7 bits
Right, 7 bits



In [28]:
```
plt.figure(figsize = (12, 6))

plt.subplot(1, 3, 1)
plt.plot(n_bits, h_l, label = "Entropy, left")
plt.plot(n_bits, h_r, label = "Entropy, right")
plt.plot(n_bits, h_rl, label = "Entropy, joint")
plt.legend()
plt.xlabel("S, log2")

plt.subplot(1, 3, 2)
plt.plot(n_bits, i_rl, label = "Mutual information")
plt.legend()
plt.xlabel("S, log2")

plt.subplot(1, 3, 3)
plt.plot(n_bits, red, label = "Redundancy")
plt.legend()
plt.xlabel("S, log2")
plt.show()
```



(J) Correlation

Let us go back to take $\hat{S} = 255$, so that each $S(x, y)$ pixel is represented by 8 bits. For each image $S(x, y)$, let

$$\bar{S}_i = \sum_{x,y} \frac{S_i(x, y)}{N}$$

be the average value of $S(x, y)$ across all the image pixels. Then shift the pixel value

$$S_i(x, y) \to S_i(x, y) - \bar{S}_i$$

so that each image should now have a zero mean value.

Now, the correlation between $S_i(x, y)$ and $S_j(x, y)$ across pixels is

$$
\begin{aligned}
R^S_{ij} &= \langle S_i S_j \rangle \\
&= \frac{\sum_{x,y} S_i(x,y) S_j(x,y)}{\sum_{x,y} 1} \\
&= \frac{\sum_{x,y} S_i(x,y) S_j(x,y)}{N}
\end{aligned}
$$

So you can get a 2×2 matrix R^S with elements R^S_{ij} for $i = 1, 2$ and $j = 1, 2$.

$$
R^S = \begin{pmatrix} R^S_{11} & R^S_{12} \\ R^S_{21} & R^S_{22} \end{pmatrix}
$$

The diagonal element, R^S_{11} and R^S_{22} of this matrix are the variance of pixel values in each monocular image, and the off-diagonal values are the covariance between the two monocular images. Please write out this matrix value.

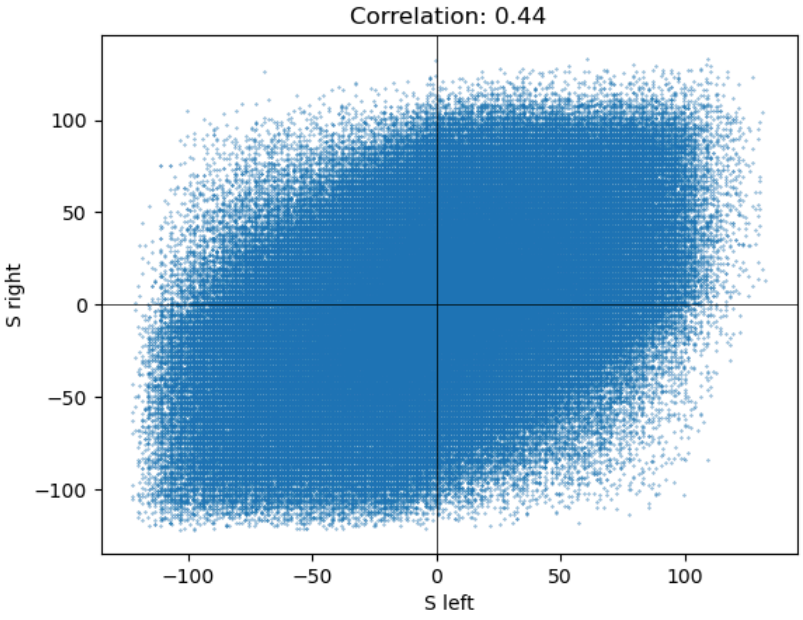
In [29]:
```
r = covariance(left, right)

print("Covariance:")
print(np.round(r, 2))

print("Correlation:")
print(np.round(correlation(left.flatten(), right.flatten()), 2))
```

```
Covariance:
[[2165.65  947.39]
 [ 947.39 2154.8 ]]
Correlation:
[[1.   0.44]
 [0.44 1.  ]]
```

(K) Scatter plot

Give a scatter plot of $S_L(x,y)$ versus $S_R(x,y)$. This means, start with a plot with horizontal and vertical axes, plot a point at location $(S_L(x,y), S_R(x,y))$, with the value of $S_L(x,y)$ and $S_R(x,y)$ on the horizontal and vertical axes respectively, for each pixel (x,y) in images $S_L(x,y)$ and $S_R(x,y)$. Compare your plot with one in Figure 1, and see if they look similar.

In [30]:
```
shifted_left = (left - np.mean(left)).flatten()
shifted_right = (right - np.mean(right)).flatten()

corr = correlation_coefficient(shifted_left, shifted_right)

plt.scatter(shifted_left, shifted_right, s = 0.1)
plt.axhline(y = 0, color = "black", linewidth = 0.5)
plt.axvline(x = 0, color = "black", linewidth = 0.5)
plt.xlabel("S left")
plt.ylabel("S right")
plt.title(f"Correlation: {round(corr, 2)}")
plt.show()
```

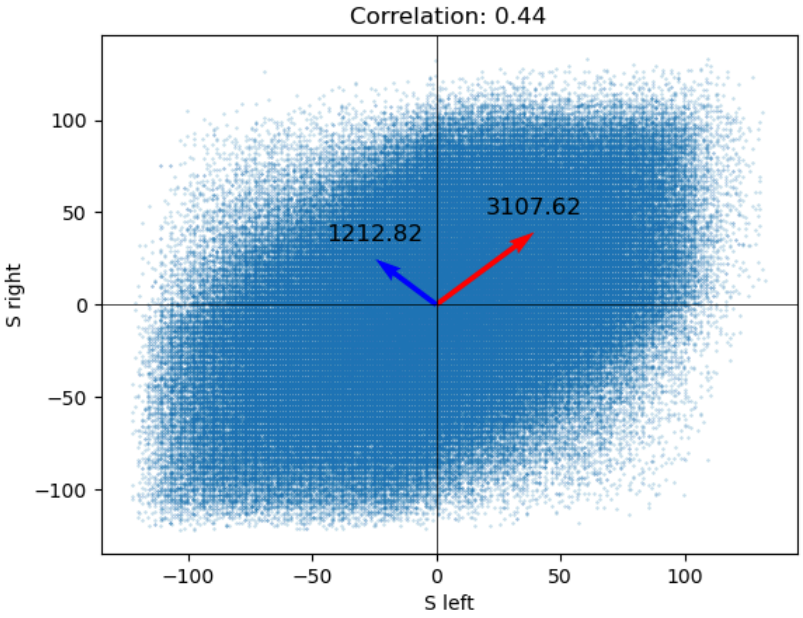


(L) Eigenvalues and eigenvectors

Calculate the eigenvalues and eigenvectors of the 2×2 matrix R^S. Plot each eigenvector as a vector in the scatter plot you obtained above in (K), and observe how each eigenvector is related to the character of this scatter plot of data, and observe how each eigenvalue is related to the variance of the data projected onto each eigenvector.

In [31]:
```
eigenvalues, eigenvectors = np.linalg.eig(r)
```

In [32]:
```
plt.scatter(shifted_left, shifted_right, s = 0.1, alpha = 0.5)
plt.axhline(y = 0, color = "black", linewidth = 0.5)
plt.axvline(x = 0, color = "black", linewidth = 0.5)
plt.xlabel("S left")
plt.ylabel("S right")
plt.title(f"Correlation: {round(corr, 2)}")

for i in range(len(eigenvalues)):
    eigen_x = np.sqrt(eigenvalues)[i] * eigenvectors[0, i]
    eigen_y = np.sqrt(eigenvalues)[i] * eigenvectors[1, i]

    plt.quiver(0, 0, eigen_x, eigen_y,
               angles='xy', scale_units='xy', scale=1, color=['r', 'b'][i])
    plt.annotate(text = round(eigenvalues[i], 2), xy = (eigen_x - 20, eigen_y + 10), size = 12)

plt.show()
```



(M) Image decorrelation

Define

$$S_+(x,y) = \frac{1}{\sqrt{2}}(S_L(x,y) + S_R(x,y))$$

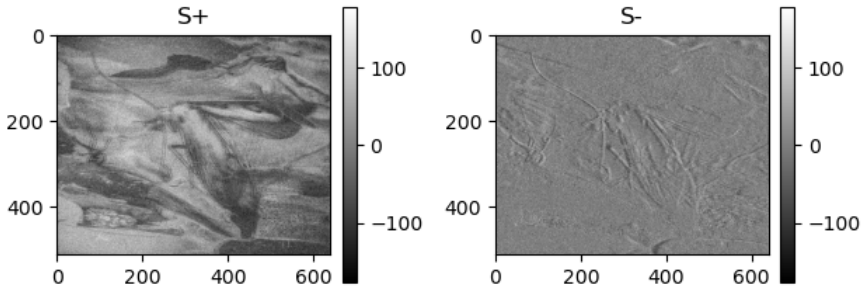
$$S_-(x,y) = \frac{1}{\sqrt{2}}(-S_L(x,y) + S_R(x,y))$$

Now $S_+(x,y)$ and $S_-(x,y)$ are two new images. Plot them out. Their pixel values are the projections of the original data $(S_L(x,y), S_R(x,y))$ onto the eigenvectors, or they are the principal components of the data.

In [33]:
```
left = normalize(left_original)
left = left - np.mean(left)
right = normalize(right_original)
right = right - np.mean(right)
```

In [34]:
```
s_plus = 1 / math.sqrt(2) * (left + right)
s_minus = 1 / math.sqrt(2) * (left - right)
```

In [35]:
```
plot_left_and_right(s_plus, s_minus, apply_lim = True, title_left = "S+", title_right = "S-")
```



(N) Correlation of de-correlated images

Calculate the 2×2 correlation matrix with elements

$$R^S_{ij} \equiv \langle S_i S_j \rangle$$

with $i = +$ or $-$ and $j = +$ or $-$. Write out the correlation matrix

$$R^S = \begin{pmatrix} R^S_{++} & R^S_{+-} \\ R^S_{-+} & R^S_{--} \end{pmatrix}$$

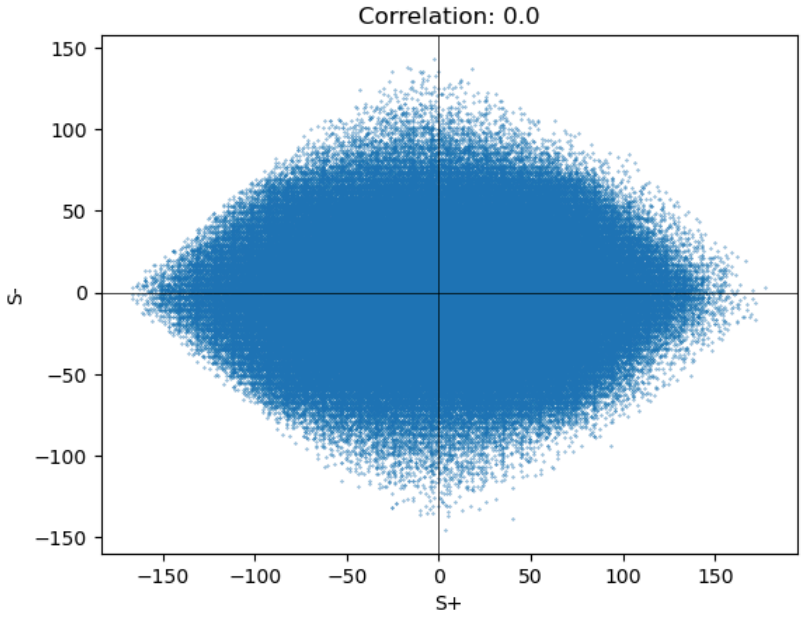
and verify that the off diagonal elements $R^S_{+-} = R^S_{-+} \approx 0$ in comparison to the diagonal elements. Reflect on what this means. Do a scatter plot of data points $(S_+(x,y), S_-(x,y))$ using all pixels (x,y) and observe how the matrix R^S reflects the character of the data in the scatter plot. Observe the variance of $S_+(x,y)$ versus the variance of $S_-(x,y)$. Which one has a larger variance?

In [36]:
```
print("Covariance: ")
print(covariance(s_plus.flatten(), s_minus.flatten()))

print("Correlation: ")
print(correlation(s_plus.flatten(), s_minus.flatten()))
```

```
Covariance:
[[3107.60713373    5.42638652]
 [   5.42638652 1212.83690231]]
Correlation:
[[1.         0.00279509]
 [0.00279509 1.        ]]
```

The diagonal elements are the covariance between the two principal components of the data. The elements R^S_{++} and R^S_{--} reflect the variance. As expected, the S_+ has a larger variance compared to S_-.

In [37]:
```
shifted_s_plus = (s_plus - np.mean(s_plus)).flatten()
shifted_s_minus = (s_minus - np.mean(s_minus)).flatten()

corr = correlation_coefficient(shifted_s_plus, shifted_s_minus)

plt.scatter(shifted_s_plus, shifted_s_minus, s = 0.1)
plt.axhline(y = 0, color = "black", linewidth = 0.5)
plt.axvline(x = 0, color = "black", linewidth = 0.5)
plt.xlabel("S+")
plt.ylabel("S-")
plt.title(f"Correlation: {round(corr, 2)}")
plt.show()
```



(O) Gain control

Give gains g_+ and g_- to $S_+(x,y)$ and $S_-(x,y)$, respectively, to create the gain controlled images

$$O_+(x,y) = g_+ S_+(x,y)$$

$$O_-(x,y) = g_- S_-(x,y)$$

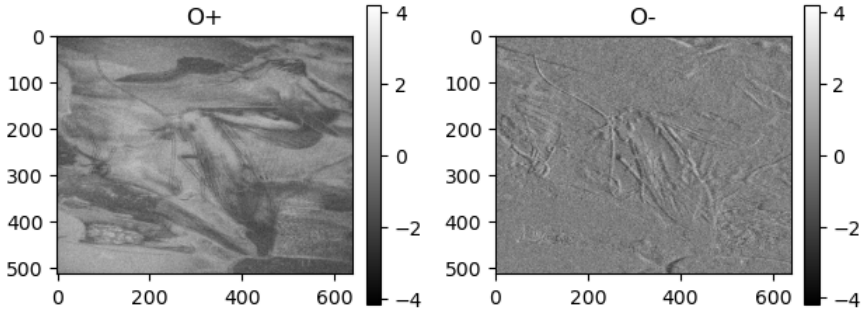
Let the gain values be

$$g_+ = \frac{1}{\sqrt{R^S_{++}}},\ g_- = \frac{1}{\sqrt{R^S_{--}}}$$

Plot the two images $O_+(x,y)$ and $O_-(x,y)$.

In [38]:
```
r = covariance(s_plus.flatten(), s_minus.flatten())

g_plus = 1 / math.sqrt(r[0][0])
g_minus = 1 / math.sqrt(r[1][1])

o_plus = g_plus * s_plus
o_minus = g_minus * s_minus
```

In [39]:
```
plot_left_and_right(o_plus, o_minus, apply_lim = True, title_left = "O+", title_right = "O-")
```



(P) Correlation of gain controlled images

Do a scatter plot of data $(O_+(x,y), O_-(x,y))$, and calculate the correlation matrix R^O with matrix element

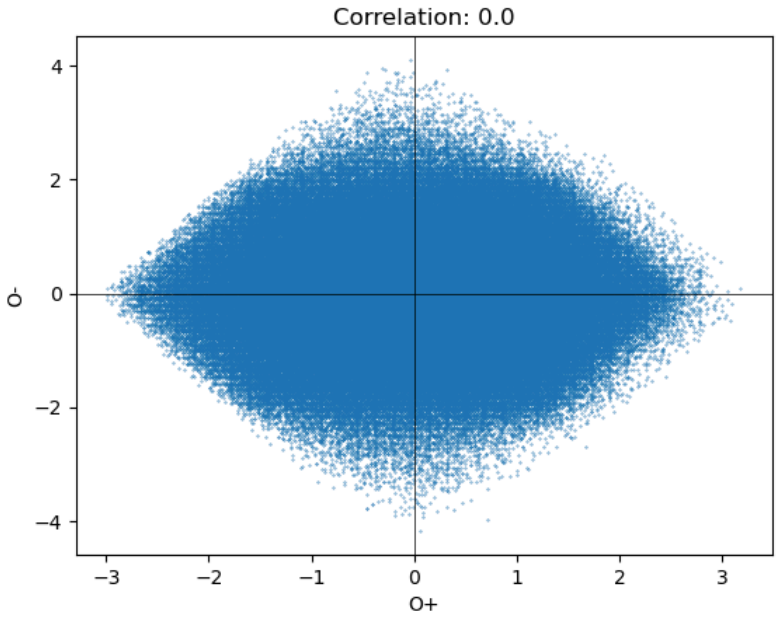
$$R^O_{ij} \equiv \langle O_i O_j \rangle$$

with $i = +$ or $-$ and $j = +$ or $-$. Write out the correlation matrix

$$R^O = \begin{pmatrix} R^O_{++} & R^O_{+-} \\ R^O_{-+} & R^O_{--} \end{pmatrix}$$

You should see that $O_+(x,y)$ and $O_-(x,y)$ are not correlated with each other, but have roughly the same variance. This is because we used the gains g_+ and g_- which are for whitening.

In [40]:
```
shifted_o_plus = (o_plus - np.mean(o_plus)).flatten()
shifted_o_minus = (o_minus - np.mean(o_minus)).flatten()

corr = correlation_coefficient(shifted_o_plus, shifted_o_minus)

plt.scatter(shifted_o_plus, shifted_o_minus, s = 0.1)
plt.axhline(y = 0, color = "black", linewidth = 0.5)
plt.axvline(x = 0, color = "black", linewidth = 0.5)
plt.xlabel("O+")
plt.ylabel("O-")
plt.title(f"Correlation: {round(corr, 2)}")
plt.show()
```



In [41]:
```
r = correlation(o_plus, o_minus)
np.round(r, 2)
```

Out[41]:
```
array([[1., 0.],
       [0., 1.]])
```

(Q) Output images

Construct two new images $O_1(x,y)$ and $O_2(x,y)$ from $O_\pm(x,y)$ as follows

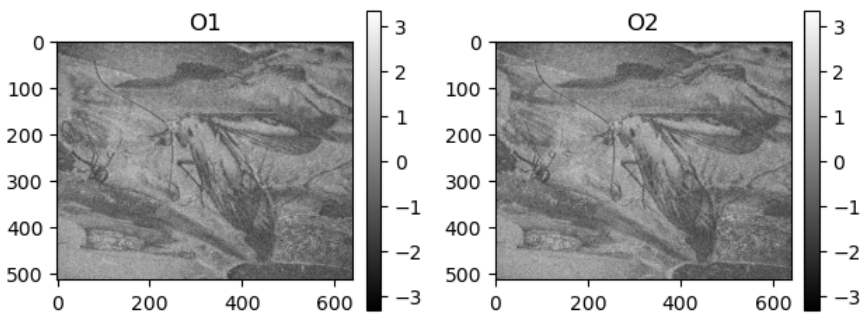
$$O_1(x,y) = \frac{1}{\sqrt{2}}(O_+(x,y) + O_-(x,y))$$

$$O_2(x,y) = \frac{1}{\sqrt{2}}(O_+(x,y) - O_-(x,y))$$

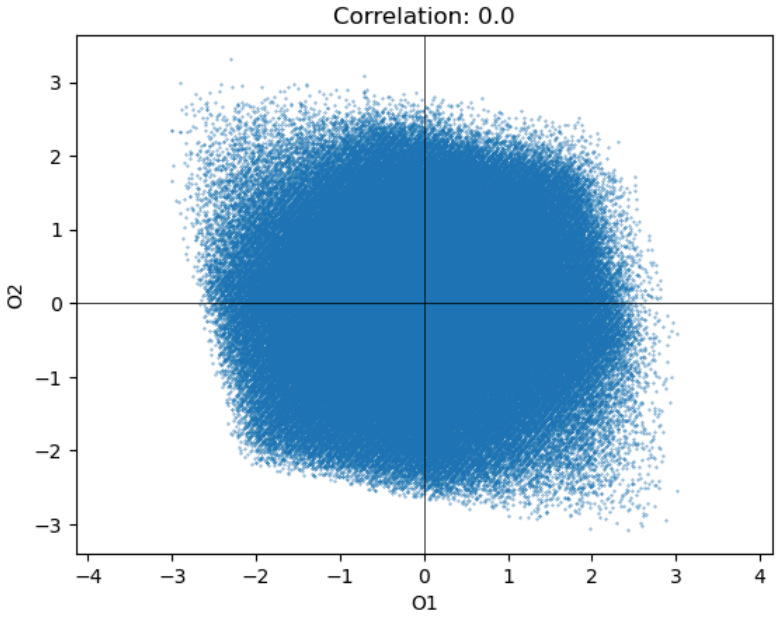
and plot them out. Also, do a scatter plot of data $(O_1(x,y), O_2(x,y))$. You should see that $O_1(x,y)$ and $O_2(x,y)$ are not correlated with each other, but have roughly the same variance.

In [42]:
```
o1 = 1 / math.sqrt(2) * (o_plus + o_minus)
o2 = 1 / math.sqrt(2) * (o_plus - o_minus)
```

In [43]:
```
plot_left_and_right(o1, o2, apply_lim = True, title_left = "O1", title_right = "O2")
```



In [44]:
```
shifted_o1 = o1 - np.mean(o1)
shifted_o2 = o2 - np.mean(o2)

corr = np.corrcoef(shifted_o1.flatten(), shifted_o2.flatten())[1][0]
lim_max = np.max([np.max(np.abs(shifted_o1)), np.max(np.abs(shifted_o2))]) * 1.25

plt.scatter(shifted_o1, shifted_o2, s = 0.1)
plt.axhline(y = 0, color = "black", linewidth = 0.5)
plt.axvline(x = 0, color = "black", linewidth = 0.5)
plt.xlabel("O1")
plt.ylabel("O2")
plt.title(f"Correlation: {round(corr, 2)}")
plt.xlim(-lim_max, lim_max)
plt.show()
```



(R) Output correlation

Construct the 2×2 correlation matrix with elements

$$R^O_{ij} \equiv \langle O_i O_j \rangle$$

with $i = 1$ or 2 and $j = 1$ or 2, for the matrix

$$R^O = \begin{pmatrix} R^O_{11} & R^O_{12} \\ R^O_{21} & R^O_{22} \end{pmatrix}$$

Relate this matrix with the scatter plot in (Q).

In [45]:
```
r = correlation(o1, o2)
np.round(r, 2)
```

Out[45]:
```
array([[1., 0.],
       [0., 1.]])
```

The elements $R^O_{21} = R^O_{12} = 0$ show that the outputs are not correlated with each other.

In [45]: