Robust Linear Regression

Question 2

Implement AM-RR (alternating minimization for robust regresion) on the same dataset as Q1. Form a matrix X whose columns are the first 70 bird images. Form a vector y that is the 71st image. Then try to fit

$$y \approx Xw$$

using the Robust Linear Regression problem.

Illustrate the following two plots. The first plot should show the image that was not covered by set S (S is an output of AM-RR). In other words, y(S^c), where S^c is the complement of S with respect to the set of all pixels. Pixels in S should be set to white (255, when the grey scale image is from 0 to 255) in this image.

The second plot should show the image indexed by S, in other words, X(S,:)w. Fill in the entries not indexed by S to white.

Marks: 33.

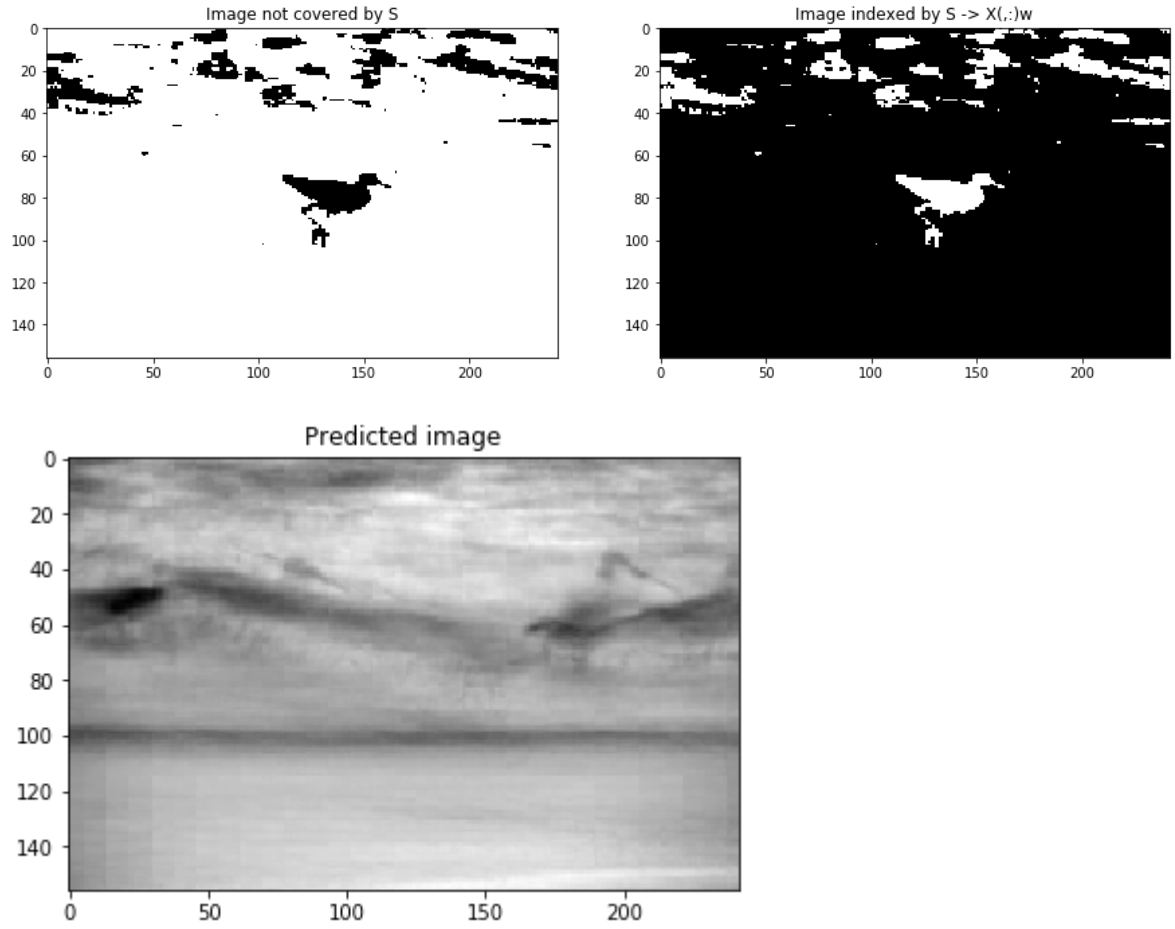
In [62]:
```
class AMRR():
    def __init__(self, y, w1):
        self.n = y.shape[0]
        self.k = initParams['k'] # 11000
        self.alpha = 0.1 / self.n
        self.y = y
        self.w1 = w1
        self.S1 = np.arange(1, self.n - self.k)

    def __str__(self):
        return 'Alternating Minimization for Robust Regression'

    def objFunc(self, X, w, S):
        y = self.y
        return 0.5 * np.linalg.norm(y[S] - X[S] @ w) ** 2

    def gradW(self, X, w, S):
        y = self.y
        sigma = X[S] @ w - y[S]
        return np.sum((sigma * X[S].T).T, axis = 0)

    def step4(self, X, w):
        n = self.n
        k = self.k
        y = self.y
        return np.argsort((y - X @ w) ** 2)[: n - k + 1]

    def amrr(self, X, dataW, dataS):
        n = self.n
        max_iters = initParams['max_iter']
        alpha = self.alpha
        y = self.y
        w = self.w1
        S = self.S1

        for i in range(max_iters):
            if (i % 100 == 0): print('Iteration', i + 1, 'of', max_it
ers)
            gradW = self.gradW(X, w, S)
            w -= alpha * gradW
            S = self.step4(X, w)
            dataW.append(w)
            dataS.append(S)
        return w, S
```

In [63]:
```
dataW = []
dataS = []
X = A[:, : 70]
y = A[:, 70]
w1 = np.zeros(X.shape[1])
amrr = AMRR(y, w1)
start = time.time()
w, S = amrr.amrr(X, dataW, dataS)
print('Done', initParams['max_iter'], 'iterations of', amrr.__str__
(), 'in', time.time() - start, 's')
```

```
Iteration 1 of 1000
Iteration 101 of 1000
Iteration 201 of 1000
Iteration 301 of 1000
Iteration 401 of 1000
Iteration 501 of 1000
Iteration 601 of 1000
Iteration 701 of 1000
Iteration 801 of 1000
Iteration 901 of 1000
Done 1000 iterations of Alternating Minimization for Robust Regressio
n in 8.626481533050537 s
```

In [67]:
```
y = X @ w
ySc = np.copy(y) # Not Covered
yS = np.copy(y)  # Covered

for i in range(y.shape[0]):
    if i not in S:
        ySc[i] = 1

for i in S:
    yS[i] = 1

ySc = np.reshape(ySc, shape)
yS = np.reshape(yS, shape)

print(amrr.__str__())
f, axarr = plt.subplots(1, 2, figsize = (15, 20))

axarr[0].imshow(ySc, cmap = 'gray')
axarr[0].title.set_text('Image not covered by S')

axarr[1].imshow(yS, cmap = 'gray')
axarr[1].title.set_text('Image indexed by S -> X(,:)w')

plt.show()

plt.imshow(np.reshape(y, shape), cmap = 'gray')
plt.title('Predicted image')
plt.show()
```

```
Alternating Minimization for Robust Regression
```



Nonnegative matrix factorization

Consider the nonnegative matrix factorization problem. For this problem, we showed that the modified multiplicative updates algorithm satisfies the nonnegativity constraints at each iteration. Consider now the nonnegative sparse coding problem:

$$\begin{aligned} \text{minimize}_{W,H} \; & F(W,H) := \frac{1}{2}\|WH - X\|_F^2 + \lambda \|H\|_1 \\ \text{subj. to } & W_{ij} \geq 0 \; \forall i,j \\ & H_{ij} \geq 0 \; \forall i,j, \end{aligned}$$

where

$$\|H\|_1 = \sum_{i,j} |H_{ij}|.$$

and $\lambda \geq 0$ is a parameter that controls the effect of the l1-norm. This problem is called sparse coding because the l1-norm forces a lot of weights in matrix H to become zero.