



► Pre-course  
Materials

► Topic 1: Course  
Overview

▼ Topic 2:  
Lossless  
Source Coding:  
Hamming  
Codes

2.1 Source Coding

Week 1 Quiz due Nov  
02, 2015 at 15:30 UTC

2.2 Sequence of  
Yes/No Questions

Week 1 Quiz due Nov  
02, 2015 at 15:30 UTC

2.3 Entropy of a Bit

Week 1 Quiz due Nov  
02, 2015 at 15:30 UTC

2.4 Entropy of a  
Discrete Random  
Variable

Week 1 Quiz due Nov  
02, 2015 at 15:30 UTC

2.5 Average Code  
Length

Week 1 Quiz due Nov  
02, 2015 at 15:30 UTC

2.6 Huffman Code

Week 1 Quiz due Nov  
02, 2015 at 15:30 UTC

2.7 Lab 1 - Source  
Coding

Lab due Nov 02, 2015  
at 15:30 UTC

► MATLAB  
download and

## LAB 1 - TASK 2 (3 points possible)

In this task, you will implemente of the function runlength\_encode that you used in Task 1. .

```
18 for bit_count = 1:length(img_1D),
19     if img_1D(bit_count) == pixel_value,
20         run_len = run_len + 1;
21     else
22         pixel_value = img_1D(bit_count);
23         runs = [runs run_len];
24         run_len = 1;
25     end
26     % % % % Revise the following code to split runs longer than
27     if run_len > 255,
28         runs = runs;
29     end
30 end
31 runs = [runs run_len];
32
33 %%%%%%%%%%
```

Unanswered

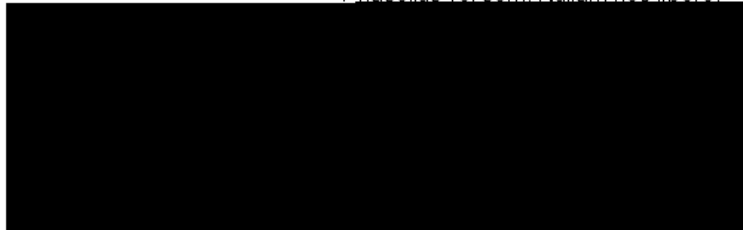
Figure 1

Vestibulum nisi nisi, ultricies et, volutpat sit amet, tincidunt ac, diam. Nam vel dolor. Praesent ante neque, tincidunt eu, adipiscing eget, blandit ac, lacus. Nulla facilisi. In commodo semper mi. Aliquam erat volutpat. Aenean consectetur arcu a arcu. Proin aliquet odio ut nunc. Phasellus vel sem. Nullam nec libero.

Figure 2

## Recreated image

tetuer adipiscing elit. Etiam aliquet. Lorem ipsum dolor sit amet, consec  
dapibus, lorem id interdum interdum, libero erat consequat risus. Proin  
et vehicula eros lacus non nibh. Fusce suscipit. ipsum in portitor  
hicula feugiat nisl tellus eu ante. tempor, odio purus tempor libero, ve  
erat lectus. Duis quis quam eu elit pellentesque Maecenas euismod place  
stas varius. Etiam non pede a arcu euismod tempor. Etiam tincidunt ege  
or diam, condimentum id, nunc. Fusce auctor semper tortor. Morbi dol  
am varius. volutpat a, sagittis a, sem. Praesent ac pede ac nisl aliqu  
nterdum, ligula eros posuere nisl. Vivamus lacinia, magna ut bibendum i  
in dui vel enim. Maecenas vitae pede. Praesent at eleifend sapie  
et, Maecenas justo nisi, ullamcorper id, congue ac, convallis ege  
gilla. Nulla quis purus. purus. Fusce vel augue ac velit faucibus frin  
d. Nullam in leo. Sed aliquet nisi sit amet sed urna cursus euism  
msan libero. Morbi eros augue. laoreet lectus. Phasellus blandit accu  
da quis, purus. Morbi et elit eget elit ut, blandit non, malesua  
llum gravida sem vel urna. Fusce lacinia. consectetuer pretium. N  
venenatis felis. Quisque tortor lorem, portifor non, consequat eu,  
nisi, ultricies et, volutpat sit amet, tincidunt ac, diam. Vestibulum nisl r  
ite neque. tincidunt eu. adipiscing eget. Nam vel dolor. Praesent ar  
; lacus. Nulla facilisi. In commodo semper mi. Aliquam erat blandit ac  
an consectetuer arcu a arcu. Proin aliquet odio ut nunc. volutpat. Aene  
Phasellus vel sem. Nullam nec impero.



```
[Warning: dec2binArray: Invalid Input. Input values should be <= 255]
```

```
size_raw_data =
```

```
    250000
```

```
size_run_length =
```

```
    296152
```

[Warning: runlength\_decode: Invalid Run Length]

Run Code

*You have used 0 of 10 submissions*

## INSTRUCTIONS

The initial MATLAB code in the above window contains a faulty implementation of the function **runlength\_encode** that you used in Task 1. In this task, you will need to correct this function.

If you click the Run Code button, the initial code will generate two figures. Figure 1 displays the original image. Figure 2 shows the recreated image from the run-length code. You will observe that the recreated image does not match the original image. This is because there is an error in the implementation. If you correct it, then the two images will match.

In the following, we describe the function of the code in more detail to help you find and correct the error.

The first part of the code reads and displays the image, as in Task 1.

The next part, starting from the comment run-length encode, contains the code implementing **runlength\_encode**. It starts by converting the 2D image into a 1D array, containing the binary pixel values listed out row by row.

The **for** loop then scans through this 1D array, counting the number of consecutive black or white pixels, assuming that the first run contains white pixels. It does this by counting the number of white pixels until it encounters a black pixel. It then appends the run length in the vector **runs**, and then starts counting black pixels until it reaches a white pixel, and so on. The vector **runs** contains the run lengths as decimal integers. Note that there is no loss

of generality in assuming the first run contains white pixels, since if the first pixel is black, the code will just return a length of 0 for the first run of white pixels.

Once the **for** loop has finished scanning through the 1D array of pixel values, it passes the decimal vector **runs** to the function **dec2binArray**, which represents each decimal run length as an 8 bit binary number. The **reshape** command simply lists the binary numbers out as a single bit stream.

The problem in the code stems from the fact that the largest integer we can represent with 8 bits is 255, but the length of the runs of white or black pixel may exceed 255. Thus, some elements of the array **runs**, which is passed to **dec2binArray**, may exceed 255.

When **dec2binArray** encounters an element greater than 255, it generates a warning message and simply encodes the number as 255 ([1 1 1 1 1 1 1 1]). Thus, when reconstructing the black and white image from the bit stream contained in the vector **run\_length\_code**, the function **runlength\_decode** thinks the run is shorter than it actually is. This is why there are so many black pixels at the bottom of the image.

We can handle by splitting runs with length longer than 255 to separate runs of at most 255 pixels, separated by runs of opposite color pixels with length 0. For example, a run of 520 pixels would be encoded by run lengths

[255 0 255 0 10]

One way to do this is to modify the code below the line

```
% % % % Revise the following code to split runs longer than 255 bits% % %  
%
```

However, there are other ways. For example, instead of modifying the code within the **for** loop, you might write another for loop that scans through the final set of run lengths created by the unmodified **for** loop looking for runs longer than 255 and then handling them in some way.

However, if you decide to do it, makes sure that your code generates the correct value for the decimal vector **runs** that is passed to the **dec2binArray** function.



© edX Inc. All rights reserved except where noted. EdX, Open edX and the edX and Open EdX logos are registered trademarks or trademarks of edX Inc.

