University of Pennsylvania

EAS 205: Spring 2010

MATLAB PROJECT: Steganography

Instructor: Donna Dietz

Steganography is the hiding of information in such a way that onlookers do not suspect information is being hidden. In contrast, cryptography is explicitly hiding information known to be present and even typically publishing the encrypted message and the entire method apart from the key! Steganography can consist of altering a photograph or sound file or other such item in such a way that casual observance of the item will not even arouse suspicion. So many documents, pictures, sound files and such are posted over the internet or sent over email, that any one of them could easily contain a hidden message. This is equivalent to hiding a $100 bill in your garden for safe-keeping. Someone who wants to truly safeguard data would be wise to encrypt their message even prior to using steganography. However, this would make for a horrendously long project, so we will just poke around a tiny bit with two types of steganography and leave cryptography alone.

Part I

The first type of steganography we will play with is LSB Steganography. LSB stands for "Least Significant Bits". The idea is that if an image is altered only slightly, the eye will not pick up the differences, and thus the message can pass through unsuspected. For this part of the assignment, you will download the zipped steganography package at the Mathworks website which was written by M. Manikandan. (Search under his name plus "LBS Steganography MATLAB" and you should find it quickly.) Play with it a little bit. Send a message to someone (or to yourself). Then, decrypt the file (found in Blackboard) *LSB_DecryptMe.bmp* which contains the remainder of the instructions for this part of the assignment. (You will be asked to encrypt something famous and give it a specific name.) The code is a little glitchy, so before decrypting the directions, rename the file *secret.bmp* and remove the existing *output.txt*. Double-check that your encrypted file can be decrypted by sending it to yourself on another computer, or by sending it to a classmate for decryption.

Part II

Now, you realize that the great part about downloading a ready-made package is that you don't have to do any of the work! However, this author has not given you the main file that you would really like to play with, the original source code! You can reverse engineer this a little bit, by encrypting a message and then comparing the two files. You would find that the pixels on the left side of the bitmap are altered first, and this alteration propogates across the page, in some sort of fashion, roughly left to right, in strips of some sort.

Of course, there are several drawbacks to using someone else's software if you are actually trying to *hide* messages. Can you think of any? (Yeah, of course you

can!) And since the original code is not available, you can't just change it a little bit. So, for the second part of this assignment, we will implement a simplistic "home-made" scheme which would be easily broken if anyone were to attempt to break it. It's naive enough that it's even visually detectable sometimes. I'm calling it "Dietz-Simplistic-Homebrew (DSH) Steganography"

DSH Steganography - "Encrypting"

Step One: Use *imread* to read in your picture file. You will get a matrix of size $P \times Q \times 3$ of $uint8$. You can see your image with the *image* command. Do yourself a favor. Save the file using *imwrite* and write to PNG format. Even with this precaution, there could be a little distortion when you open your encrypted file, but this will cut way down on the possibility. You do not want to use JPG format, because the compression process will erase any work you do when you go to save your new picture.

Step Two: Split the image into three matrices of equal size ($P \times Q$) , call them A, B, and C.

Step Three: Look at matrices A and B. For some threshold (*thresh*), the logical matrix: $T = (A < thresh \quad \& \quad B < thresh)$ contains roughly 5% 1s. Search for the largest *thresh* value (which is an integer, of course) that still gives less than 5% of the entries of T to be 1s. The idea is to use this T matrix to determine which entries in C we will alter in order to hide our message. You can use $spy(T)$ to help you see this matrix.

Step Four: Read in a file containing a short message in a text file. (You may use $textscan(fid,'\%s','delimiter','\backslash n')$ to do this after properly opening the file.)

Step Five: Convert the message to a numerical vector using the *abs* command, which turns strings into vectors of integers. (The *char* command reverses this process.) The values range between 65 and 122 for captial and lowercase letters. Spaces become 32.

Step Six: We would like to make spaces less obvious, so we will replace them with a value between 92 and 97 (I will use 95), as those values are used for punctuation but are in the same range as the letters. You now cover a range of 65 to 122 of important data. Toss out or otherwise get rid of anything out of this range.

Step Seven: T gives you a mask, telling you where in C you will hide data. Of the four values between 0 and 255 which are equivalent modulo 64 to the data element, pick the one closest to the value in the original picture. Use this value to hide your data element. When you retrieve it, just find its modulo 64 equivalent and add 64 back in. The rationale here is to use the mask to locate what will tend to be darker parts of the picture. From my own looking at the colors charts online, I noticed that, for example, dark red and dark blue are more similar than light red and light blue. So, I'm presuming that if the Red and Green matrixes of the picture are dark, the Blue one probably is as well. That could be incorrect. So, giving ourselves 4

choices for how to write the data is one more precaution for hiding the errant pixels. We could force ourselves to use only 32 characters and then we'd have 8 choices for how to write the data, for example.

Step Eight: Place the message, one number at a time, into a matrix which is zero everywhere T is zero and contains a number everywhere T is 1. Do this in MATLAB's standard order, column-by-column. Note that to convert T from a logical matrix to $uint8$, simply $uint8(T)$. Then, T and $\tilde{T}$ can interact with $uint8$ matrices.

Step Nine: Create a new $P \times Q \times 3$ matrix using $size$, then put A and B into it. C will be the same as before except for locations where T is 1. Those spots will contain the message!

Step Ten: Write the message to a text file.

Create two functions:
function [ ] = DSH_StegEncrypt(inputTextFileName, inputPictureName, outputPictureName)

function [ ] = DSH_StegDecrypt(inputTextFileName, inputPictureName, outputPictureName)

Again, decrypt the file *DSH_DecryptMe.png* which contains the remainder of the instructions for this assignment. Please feel free to send secret messages to your classmates as practice.