Practice Quiz: Steganography

Programming Exercise Guide

Before attempting this quiz you should write the following JavaScript program to complete the Steganography assignment to be able to hide one image inside another image by hiding information inside half of each pixel.

Your task is to do the following:

1. Select two images for which you want to hide one image in another image. They most likely will be different sizes, and in the next step we will crop both of them to be the same size.
2. In order to hide one image inside of another image, the images need to be the exact same size. Write the function `crop(image, width, height)` that has three parameters: `image` is a complete image, and `width` and `height` are numbers. This function creates and returns a new image from the parameter image, with the new image having the width and height of the specified width and height parameters, thus cropping the picture by ignoring the pixels on the right side or bottom side that are beyond the specified width or height of the picture.

For example, here is Owen's image, which is width 240 and height 360, on the left, and his image on the right after running the `crop` function with width 200 and height 300. The rightmost and bottommost pixels from the original image are not in the cropped image. The new cropped image is 200 width and 300 height.


3. Test the crop function you wrote by
    a. Inputting two images of different sizes.
    b. Determining the size of the cropped images
    c. Calling the function `crop` to create new images of the same size.
    d. Test the `crop` function by printing the values of the width and heights of the two images before and after cropping them. You will be able to tell if the images are the same size, and if the images are dramatically different sizes, you should also see the crop quite clearly.
4. Make room to hide an image by
    a. Type in the `chop2Hide` and `pixchange` functions from the steganography lesson. How well did you understand those functions? See if you can write them without looking at the code!
    b. Then call the function `chop2Hide` with the image that will be hiding another image, saving the resulting image in a variable called `start`.
    c. How do we know it worked correctly?
        i. Pick one pixel from your original image, say at location x=50 and y=60, and print out its red, green, and blue values before applying `chop2Hide`, and then print out the same pixel location in the new image returned from the call to `chop2Hide`. Perform a calculation to see if it is the correct value.

   ii. You can use either method to print and see the values of a few pixels to convince yourself that `chop2Hide` is working. There are too many pixels to print out all the values.

5. Prepare the image to hide by shifting the left most half of the bits to the right and replacing the leftmost half of the bits to 0s.
   a. Type in the `shift` function from the lesson. How well did you understand it? Can you write it without looking at the code from the lesson?
   b. Call the `shift` function on the image you want to hide to prepare it for hiding.
   c. You can test the function in the same way you tested the `chop2Hide` function—by printing the RGB values of a few pixels before and after calling `shift`.
   d. Write the function `combine` to create a new image that is the chopped image combined with the shifted image to create an image with another image hidden inside of it. The steps you should take are:
      i. Write a function called `newpv(p,q)` with the integer parameters `p` and `q` to return the sum of `p` and `q`. This function should print an error message if the sum is greater than 255. With the way we have written the code, the sum should never be greater than 255, but this will let us know if we did something wrong.
      ii. Write the function `combine(image1, image2)`, which returns a new image that is the combination of the two images. That is, for each pixel in one image, find the pixel in the same position in the other image and add the red values together, the blue values together, and the green values together to form a pixel for the new image.
      iii. Test the resulting image by sampling a pixel that is in `image1`, `image2` and the new image and check their RGB values to see if they added correctly.