Team Name: Team Steganosaurus
Youtube Link : https://youtu.be/R7Lg8eB35UM

Section: A01, A10

MEMBERS:

1. Daniel Chorn(A01)
2. Gabriel Ketron (A10)
3. Isabelle Crisostomo(A10)

Peer Evaluation:

Daniel - Developed code structure and debugged.
Isabelle - Developed the GUI.
Gabe - Designed embedding methods and extraction methods.

Individual Contributions:
Gabe: The four coding methods we implemented follow the same overarching process: altering pixel values of the individual layers of the host image as little as possible while still imparting recallable information. The first method takes finds the ten's place in the pixel and makes it odd or even, subtracting by 10 for the pixel value, unless it is too close to zero where it is added by ten.

The second method changes the ten's place value to be a multiple of three or not when the input pixel is black or white. This requires the pixel values to be changed by 20 at a time, and by 40 depending on the limitations of being too close to 0 or 255. This change brings the pixel tens place to its closest multiple of three, while still being odd or even for the first method.

The third method finds the one's place in the pixel and makes it odd or even, subtracting by 1 for the pixel value.

The last method finds the one's place value, then adds or subtracts by 2 to find the closest pixel value that is both a multiple of three and also even or odd as determined by the third method.

Once all the images of each file are embedded, the grayscale color channels are layered on top of each other, and displayed to the viewer. The result is a picture that is almost unchanged from the original, yet has ten images hidden inside.

The recovery process is very simple compared to the embedding process. To recover the first method, a matrix of the pixel's tens' place, and using the function "mod" to find the remainder of the tens place divided by 2, we created a 400x400x1 logical matrix. We use the Expand function to return the final presented image. To recover the second method, the mod function finds the remainder of the tens place divided by 3. The final recovered image is expanded and presented in the GUI. To recover the third method, the code simply checks if the whole pixel value is odd or even. The fourth method checks if the very last pixel value is divisible wholly by 3.

Daniel: Hi I'm Daniel Chorn and I just wanted to cover the basics of our steganography coding. The technique is relayed using a nested for loop and if statements. First the coding resolves the host image into its three RBG layers, from there we set [row, column] variables to equal the size of each layer. From there, the nested for functions will include from i:row and from j:column, with the if functions testing the previously mentioned parameters of the hidden image, its greyscale value and whether it is black or white. Based on these logical matrices, the nested for loops then test pixel by pixel to properly adjust the grayscale host image's pixels. Finally each color channel is combined to the final image.

Isabelle: Hey it's Isabelle. My job was managing the GUI, specifically the creation of the callback functions in the buttons that will do things such as display images, activate the embedding and recovery techniques, and save the image or restart the program. The initial images as well as the extraction of embedded images will both load as a separate window. In terms of coding, my job relied on writing the program to flatten the images to be embedded into their binary black and white images. This was what allowed for the methodology we provided to be adjusted depending on the loops we decided to use.

Now let's run through the program together. We have some instructions we can follow. The first step is to click on the load host image, and that will be the dog photo. Now let's load all ten images to be embedded, and these will be already flattened. Next, we'll click on this button so that the embedding techniques will be called, so the ten images will be hidden in the dog image. This takes a while so we have some music for you to listen to while you wait.

Perfect, now let's save that image and name it. Okay then we'll load that same image that we just saved and then we press the recovery button, so that the images we've hidden can come into view. They're not all perfect, but that's okay. And to clear all of the boxes and start again we have a restart button. We also have an easter egg. If you press of this gif you'll receive a message from our team. Finally when you close the program, you will get another message with a sound. Let's go to the code view and take a closer look at the code behind the buttons.

The graphics used are taken in mostly by imread and displayed by imshow, into the proper axes. In order to get those graphics, i used uigetfile to retrieve files from my computer with all the images. One function that is also important is the expand function, which makes a logical matrix have three color channels, then multiplies the zeros and ones as numbers by 255. I used imwrite and uiputfile to save the image after it was embedded, and to restart i used CLA. The sound effects came from these audio functions. That’s all from Isabelle!