# Step 1: Loading/Reading an Image into MATLAB

Firstly, we load an image into MATLAB using the *imread* function. We specifically choose a PNG or BMP image with the dimensions , where *M* and *N* are divisible by 8. We choose a PNG or BMP image as a JPEG image, is already a compressed image via the DCT image compression technique.

# Step 2: Convert the Image to Grayscale

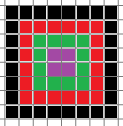
We convert the loaded image to grayscale using the function *rgb2gray*. The reason we do this is because an image, colour or not imported into MATLAB creates a 3-dimensional array, where the first two elements of the array is the dimensions of the image and the third dimension is the colour map. Converting the image to a grayscale one, creates a 2-dimensional array, which is the dimensions of the uploaded image.

# Step 3: Divide the Image into smaller blocks

We now divide the image into smaller blocks, creating the domain pool. Once the image is divided into smaller blocks, we index each block from the top left starting from 1 and increment each index by one. The reason for creating such an index is so that the receiver will be able to determine which smaller block in the domain pool contains holes so that the receiver can fill these holes.

# Step 4: Create holes in the smaller blocks

With the domain pool created, we create holes by starting at the center square within the square. We start off with the center square and calculate the average value of these 4 pixels. If the average value is within values of each pixel, then a hole can be created with those 4 pixels. We then check a larger square pixel in the same block where the center square is now . Once again, we calculate the average of the surrounding pixels. As before, if the average compared to each pixel is within a specific range, a larger hole can be created. This can continue until we reach a center square size of , which we will define as the largest hole that can be made.



# Step 5: If a hole is created, record which index has the hole

Now that we have created a hole, we can index the block and this can be transmitted with the image itself. So that the receiver will now which blocks in the domain pool will have a hole.

# Step 6: Compress the image using a known technique

For example, if the chosen compression algorithm is the DCT compression technique, we perform the DCT on each block in the domain pool. Since DCT is done on blocks, this was the reason we chose an image where the dimensions are divisible by 8. Now the image is compressed it can be transmitted through the channel.

# Step 7: Encode and Transfer Image

We now encode the image as a binary encoded message, and when we transmit it through the channel, we can introduce errors randomly which can be easily detected and corrected. The data that will be sent through the channel will be a bit stream and the errors introduced will be based off of a byte with a probability of where the byte can then be randomized. Alternatively, each bit can be random selected to be prone to an error and the bit value flipped.

# Step 8: Decode and Error Check

The first thing to look at in this step is error correction. Identification of the location of the errors is required. We first correct them and then decode the image. We convert the image from a bit stream into its 2 dimensional matrix.

# Step 9: Inverse DCT

Since DCT compression was used, we now perform the inverse DCT to obtain the compressed image and its respective values in the 2-dimensional matrix.

# Step 10: Fill the holes in the image

Using the array created to log which index in the domain pool has a hole, we then fill the holes by calculating the average value of the pixels and filling in the subsequent inner pixels with this average value.