Homework 01

Deepak Sharma
CSCI 731 Advance Computer Vision

June 18, 2017

Part B, Video Display

I was able to display all sample videos successfully with bellow OS.

```
Distributor ID: Ubuntu
Description:  Ubuntu 14.04.5 LTS
Release: 14.04
Codename:  trusty
```

Table 1: Video name and display status

Video Name	Display Status
CAST_TREE_FALLING.mp4	Success
diving_video_far__board_477E38120CAF.MOV	Success
diving_video_far__board_53E7B76AF195.MOV	Success
D_Kinsman_Boomerang.mp4	Success
IMG_1744.m4v	Success
VID_20170304_140105.3gp	Success
D_Kinsman_Boomerang.mp4	Success
Video_Surveilance__Police_Officer_attacked_by_ferocious_rodent.mp4	Success

Part C, Steganography

We were previously emailed below image for analysis. One of the bit planes, on one of the color channels, contains a secret message. Find the secret message, and save that bit plane to a file. I suggest writing a program that tries all possible colors, and all bit-planes. Then include the secret message in your write-up.There are three ways to scan through an image, pixel by pixel. They are discussed in the OpenCV Tutorial provided. (Caution the tutorial uses variable names that are too short for proper documentation.) Which technique did you use? How did you isolate each bit plane? What was your secret message or secret image.

Figure 1: Input Image

Solution:
The secret message was **"THE SATELLITE ORBITS THE PLANET"**, message was visible in the third bit plan and second bit plan of the green channel, as shown in figure 2 and 3.

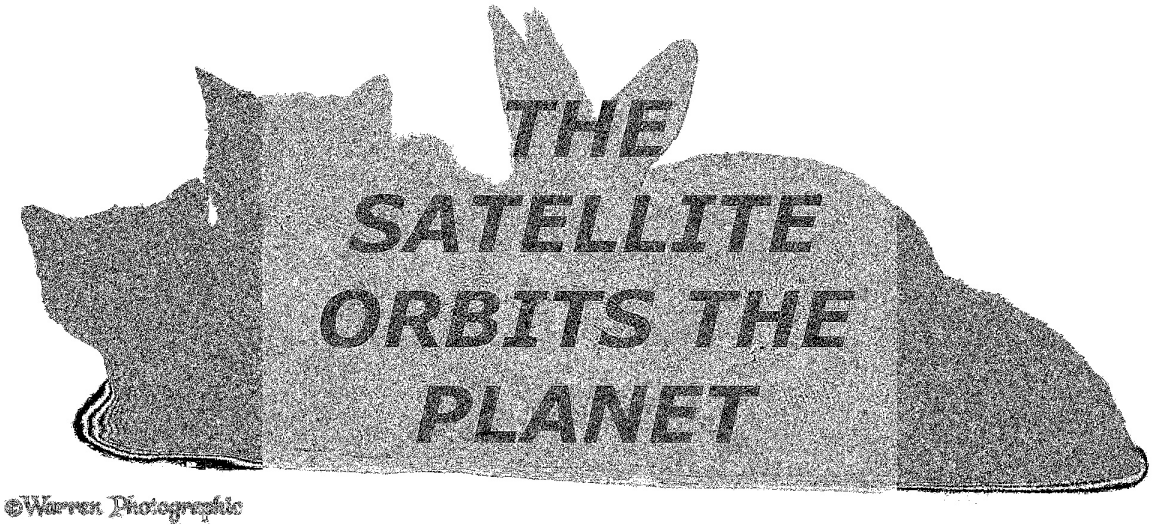

Figure 2: Secret Image, third bit plan of green channel

For keeping my program simple, I decided to go with method "On-the-fly address calculation with reference returning". This method is somewhat similar to the method we have learned in MATLAB. I kept it further simple by first creating three one channel empty(initialized with 0 value at all spatial location) images for each color channel:

Figure 3: Secret Image, second bit plan of the green channel

```
for (int channelCounter=0; channelCounter<nChannels; channelCounter++){
    channels.push_back(Mat(nRows, nCols, CV_8UC1, Scalar(0)));}
```

Then assigned pixel intensity to these empty one channel images by accessing one by one each pixel location in the input image.

```
    for (int rowCounter=0; rowCounter<nRows; rowCounter++)
        {
        for(int colCounter=0; colCounter<nCols; colCounter++)
            {
            // Acessing intensity of a pixel at specified location
            Vec3b val = image.at<Vec3b >(rowCounter, colCounter);
            for (int channelCounter=0; channelCounter<nChannels;\
                    channelCounter++)
                {
                // Assigning intensity values to corresponding channel image
                channels[channelCounter].at<uchar>(rowCounter, colCounter)=\
                                                    val[channelCounter];
                }
            }
        }
```

Though this approach is not recommended, but I used it because it gave me chance to play with the image the way I wanted to play. There is no doubt other methods are more efficient.
In order to access bit value from particular location of a byte, P number of right shift operations were performed on the byte value which was followed by a logical AND operation with mask value(1) [8]. Here P is the position of the required bit [5].

```
unsigned char getBit(unsigned char byte, int position)
{
       // Performing right shift and logical and operation with mask
       // for accessing required bit
       bool bitBool = (byte >> position) & 0x1;
       // Converting bool type varibale to unsinged char tyep
       unsigned char bit = bitBool?1:0;
       return bit;
}
```

Now for each bit location[0-7], collected bit values corresponding to all spatial locations of the input color channel, which produced a binary image.

```
vector<Mat> getBitPlans(vector<Mat> &channels){
    int nChannels = channels.size();       //Acessing number of channels
    if (nChannels < 0)
        // Throwing invalide input error if no channel image received
        throw invalid_argument("received no channel");
    int nBitPlans = DEFAULT_N_BIT_PLANS*nChannels; //For number of bit plans
    int unsignedCharLimit = 255;               // For storing highest possible
        pixel value
    int nRows = channels[0].rows;              // Accessing number of row in an
        input channel
    int nCols = channels[0].cols;              // Accessing number of col in an
        input channel
    int channelCounter = 0;                    // Store channel couter
    unsigned char bitVal;
    vector<Mat> bitPlans;                      // For storing bit plans
    for(int bitPlanCounter=0; bitPlanCounter < nBitPlans; bitPlanCounter++){
        //Intialzing bit plans matrixies with 0 value
        bitPlans.push_back(Mat(nRows, nCols, CV_8UC1, Scalar(0)));}

    // Iterating over pixel locations for accessing intensity of channels
    for (int rowCounter=0; rowCounter<nRows; rowCounter++){
        for(int colCounter=0; colCounter<nCols; colCounter++){
            for(int bitPlanCounter=0; bitPlanCounter < nBitPlans;\
                    bitPlanCounter++){
                channelCounter = bitPlanCounter/DEFAULT_N_BIT_PLANS;
                // Acessing intensity of channel at specific location
                Scalar val = channels[channelCounter].at<uchar>(rowCounter, \
                        colCounter);
                // Acessing bit value at specific bit location from the
                // intensity value corresponding to specific spatial location.
                bitVal = getBit(val[0], bitPlanCounter - \
                        DEFAULT_N_BIT_PLANS*channelCounter);
                // Assigning the bit value to bit plan image
                bitPlans[bitPlanCounter].at<uchar>(rowCounter, colCounter) =\
```

```
                unsignedCharLimit*bitVal;}
        }
    }
    return bitPlans;}
```

Source code credit

1. Image and video load examples [1, 4].
2. Exception Handling [6, 9].
3. Codding style and guidelines [2].
4. Image channel extraction [3].
5. Assertion of a file path [7].
6. Bit value extraction [8, 5].

References

[1] Gary Bradski Adrian Kaehler. *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library 1st*. O'Reilly Media, Inc., 2 edition, 2016. ISBN 1491937998 978149193799.

[2] Errin Fulp. C++ coding guidelines, 2006. URL `http://csweb.cs.wfu.edu/~fulp/CSC112/codeStyle.html`.

[3] Haris. how do i separate the channels of an rgb image and save each one, using the 2.4.9 version of opencv?, 2015.

[4] *The OpenCV Reference Manual*. Itseez, 2.4.9.0 edition, April 2014.

[5] Stewbond. Most efficient way to read a bit from a byte?, 2013. URL `http://www.cplusplus.com/forum/general/97378/`.

[6] Nsanders Terry Li, John Dibling. How to throw a c++ exception, 2012. URL `https://stackoverflow.com/questions/8480640/how-to-throw-a-c-exception`.

[7] Pherric Oxide Vincent, Pevik. Fastest way to check if a file exist using standard c++/c++11/c?, 2012.

[8] 0612 TV w/ NERDfirst. Bit manipulation 04: Bit masking, "2006". URL `https://www.youtube.com/watch?v=lUzQtTLCglk`.

[9] Jonathan Wakely. Throw exception if the file does not exist in constructor and try/catch it when creating an object in main(), if good - start using the object, 2016.