Algorithm Used

I used Least Significant Bit steganography algorithm to create this program.

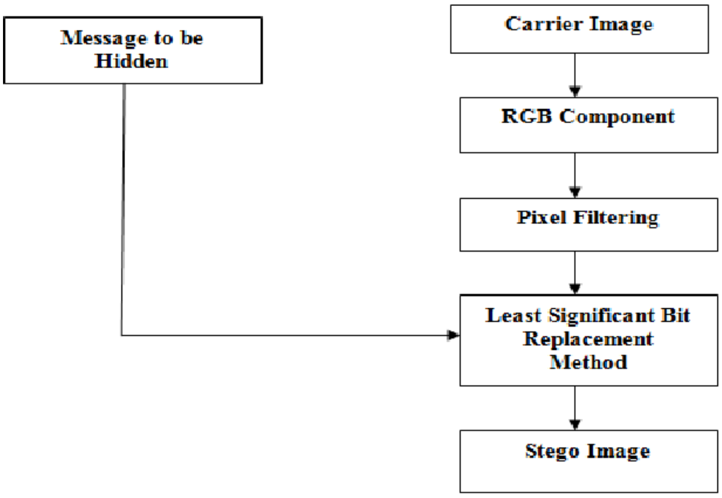


Figure 1. Least Significant Bit steganography algorithm

Digital image structure

To understand this technique, a few reminders of some digital imaging basics might be useful.

- A digital image is composed of X rows by Y columns.
- The point of coordinates [a,b] with 0⩽a<X and 0⩽b<Y, is called a pixel. The pixel represents the smallest addressable element of a picture.
- Each pixel is associated with a color, usually decomposed in three primary colors: Red, Green, Blue. A pixel can then be specified as pixel (Red, Green, Blue), that’s what we call the RGB model.
- Red, Green and Blue intensities can vary from 0 to 255.
- WHITE = (255,255,255) and BLACK = (0,0,0).
- A pixel takes 3 bytes of memory, 1 for each primary component (hence the maximum value of 255).
- A byte consists of 8 bits, representing a binary number (example: 1010 0101).
- The highest value a byte can take is 1111 1111, which is equal to 255 in decimal.

As its name suggests, the Least Significant Bit technique is based on hiding information in the least significant bit of each byte of the picture. There are multiple variants of LSB but, in this article, we will set the focus on the most common one.

Let’s take the following representation of a byte, where the weight is annotated below each bit:

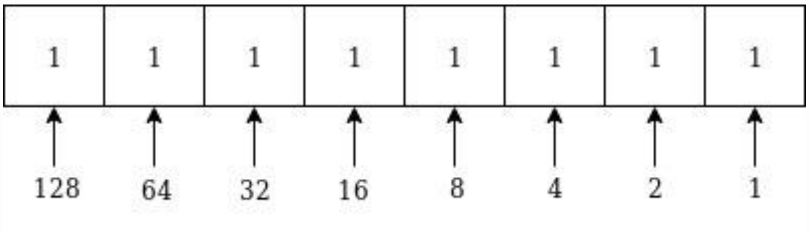


Figure 2. Representation of a byte

The first bit on the left is the “heaviest” one since it’s the one that has the biggest influence on the value of the byte. Its weight is 128.

Now look at the bit on the very right. Its weight is 1 and it has a very minor impact on the value of the byte. In a way, this bit is the **least significant bit** of this byte.

Why do we modify this very specific bit?

Well, simply because it’s the least significant one. The following diagram illustrates the color difference when the least significant bit of the red channel is modified.

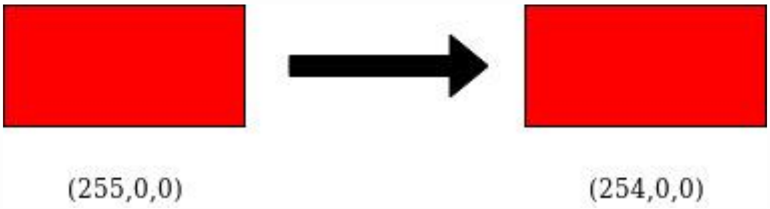


Figure 3. Red channel modified with one bit

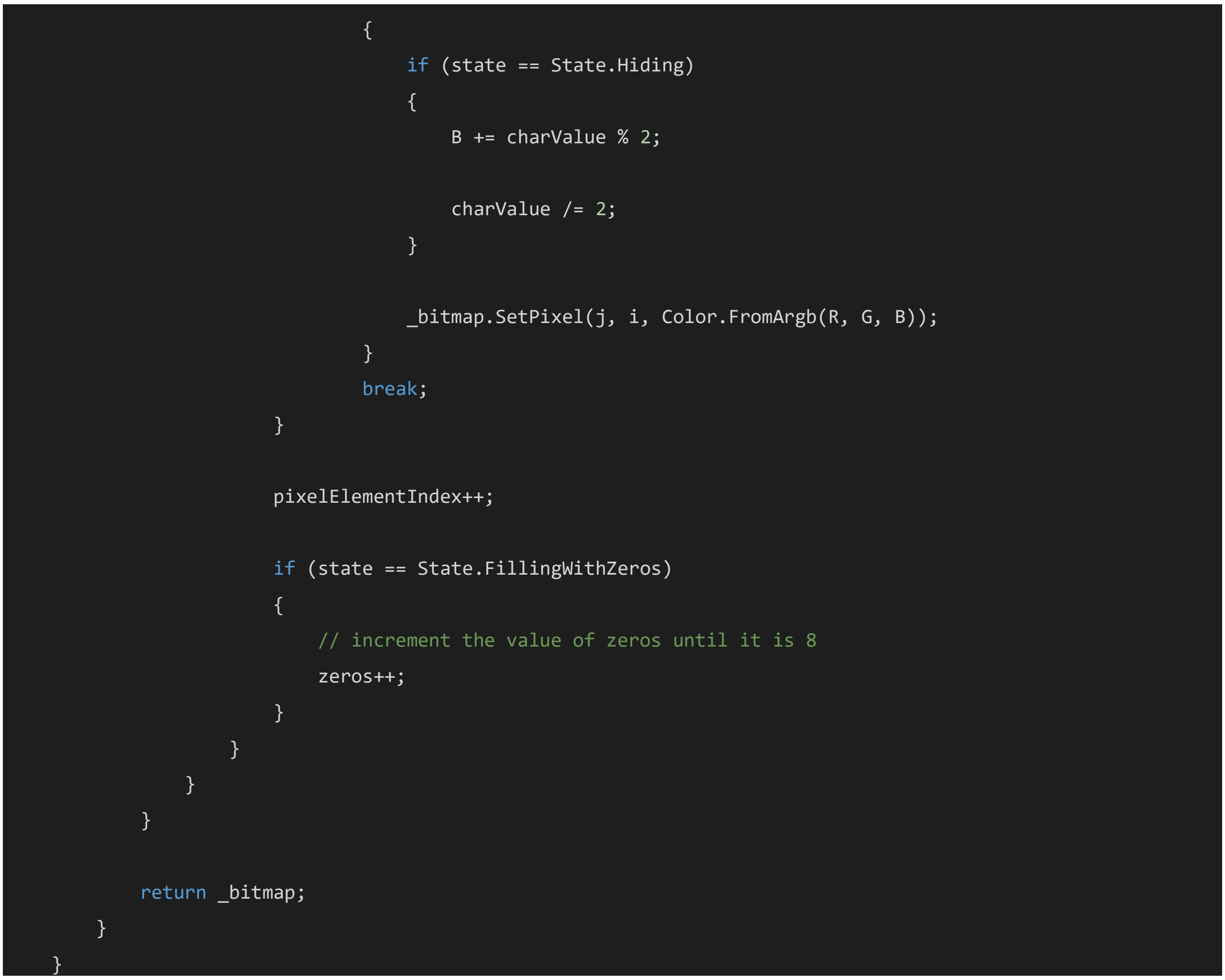
There is no noticeable difference! That way, we can modify **3 bits per pixel** without it is noticeable.

Source Code

The detailed algorithm is described below. This method embeds given set of characters into the image.

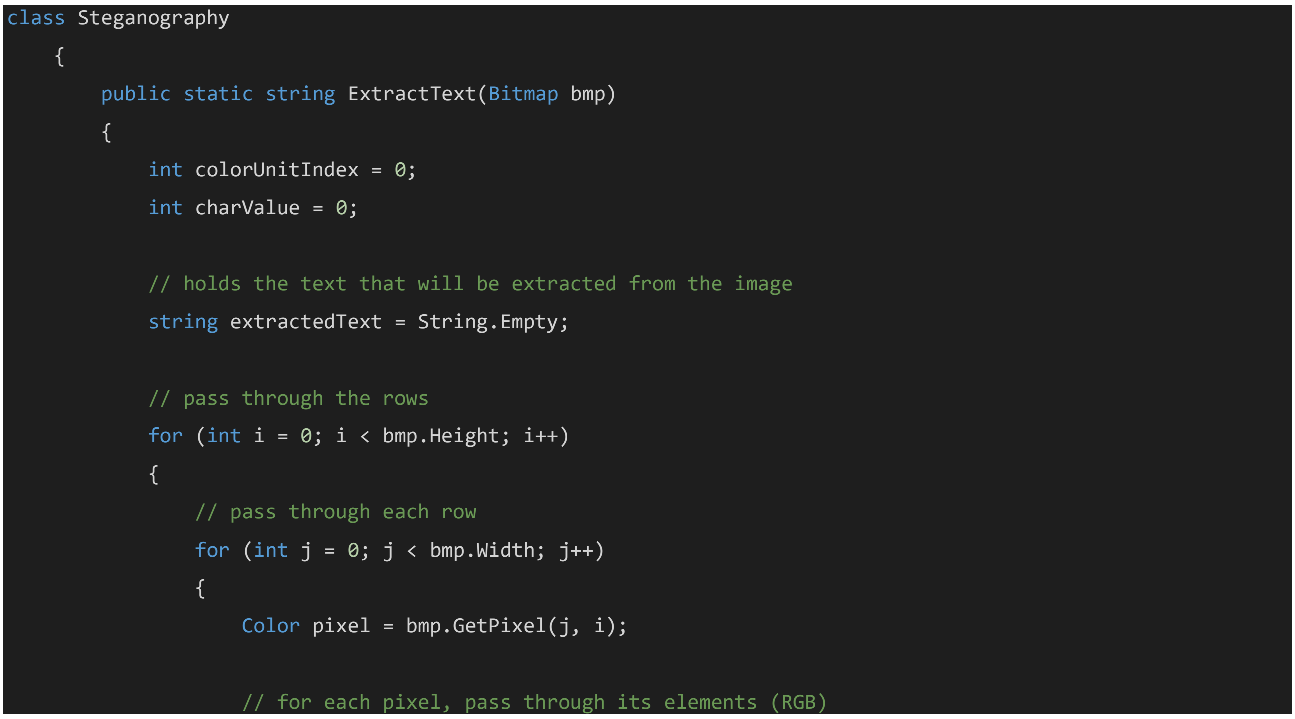
```
class Steganography
    {
        public enum State
        {
            Hiding,
            FillingWithZeros
        };
```

```
public static Bitmap EmbedText(string text, Bitmap _bitmap)
{
    // initially, we'll be hiding characters in the image
    State state = State.Hiding;

    // holds the index of the character that is being hidden
    int charIndex = 0;

    // holds the value of the character converted to integer
    int charValue = 0;

    // holds the index of the color element (R or G or B) that is currently being processed
    long pixelElementIndex = 0;

    // holds the number of trailing zeros that have been added when finishing the process
    int zeros = 0;

    // hold pixel elements
    int R = 0, G = 0, B = 0;

    // pass through the rows
    for (int i = 0; i < _bitmap.Height; i++)
    {
        // pass through each row
        for (int j = 0; j < _bitmap.Width; j++)
        {
            // holds the pixel that is currently being processed
            Color pixel = _bitmap.GetPixel(j, i);

            // now, clear the least significant bit (LSB) from each pixel element
            R = pixel.R - pixel.R % 2;
            G = pixel.G - pixel.G % 2;
            B = pixel.B - pixel.B % 2;

            // for each pixel, pass through its elements (RGB)
            for (int n = 0; n < 3; n++)
            {
                // check if new 8 bits has been processed
                if (pixelElementIndex % 8 == 0)
                {
                    // check if the whole process has finished
                    // we can say that it's finished when 8 zeros are added
                    if (state == State.FillingWithZeros && zeros == 8)
                    {
                        // apply the last pixel on the image
                        // even if only a part of its elements have been affected
                        if ((pixelElementIndex - 1) % 3 < 2)
                        {
                            _bitmap.SetPixel(j, i, Color.FromArgb(R, G, B));
                        }

                        // return the bitmap with the text hidden in
                        return _bitmap;
                    }

                    // check if all characters has been hidden
                    if (charIndex >= text.Length)
                    {
                        // start adding zeros to mark the end of the text
                        state = State.FillingWithZeros;
                    }
                    else
                    {
                        // move to the next character and process again
                        charValue = text[charIndex++];
                    }
                }

                // check which pixel element has the turn to hide a bit in its LSB
                switch (pixelElementIndex % 3)
                {
                    case 0:
                        {
                            if (state == State.Hiding)
                            {
                                // the rightmost bit in the character will be (charValue % 2)
                                // to put this value instead of the LSB of the pixel element
                                // just add it to it
                                // recall that the LSB of the pixel element had been cleared
                                // before this operation
                                R += charValue % 2;

                                // removes the added rightmost bit of the character
                                // such that next time we can reach the next one
                                charValue /= 2;
                            }
                        }
                        break;
                    case 1:
                        {
                            if (state == State.Hiding)
                            {
                                G += charValue % 2;

                                charValue /= 2;
                            }
                        }
                        break;
                    case 2:
```



```
                                {
                                    if (state == State.Hiding)
                                    {
                                        B += charValue % 2;

                                        charValue /= 2;
                                    }

                                    _bitmap.SetPixel(j, i, Color.FromArgb(R, G, B));
                                }
                                break;
                        }

                        pixelElementIndex++;

                        if (state == State.FillingWithZeros)
                        {
                            // increment the value of zeros until it is 8
                            zeros++;
                        }
                    }
                }
            }

            return _bitmap;
        }
    }
```

This method extracts the hidden message from the image.



```
class Steganography
    {
        public static string ExtractText(Bitmap bmp)
        {
            int colorUnitIndex = 0;
            int charValue = 0;

            // holds the text that will be extracted from the image
            string extractedText = String.Empty;

            // pass through the rows
            for (int i = 0; i < bmp.Height; i++)
            {
                // pass through each row
                for (int j = 0; j < bmp.Width; j++)
                {
                    Color pixel = bmp.GetPixel(j, i);

                    // for each pixel, pass through its elements (RGB)
                    for (int n = 0; n < 3; n++)
                    {
                        switch (colorUnitIndex % 3)
                        {
                            case 0:
                                {
                                    // get the LSB from the pixel element (will be pixel.R % 2)
                                    // then add one bit to the right of the current character
                                    // this can be done by (charValue = charValue * 2)
                                    // replace the added bit (which value is by default 0) with
                                    // the LSB of the pixel element, simply by addition
                                    charValue = charValue * 2 + pixel.R % 2;
                                }
                                break;
                            case 1:
                                {
                                    charValue = charValue * 2 + pixel.G % 2;
                                }
                                break;
                            case 2:
                                {
                                    charValue = charValue * 2 + pixel.B % 2;
                                }
                                break;
                        }

                        colorUnitIndex++;

                        // if 8 bits has been added, then add the current character to the result text
                        if (colorUnitIndex % 8 == 0)
                        {
                            // reverse? of course, since each time the process happens on the right (for
simplicity)
                            charValue = reverseBits(charValue);

                            // can only be 0 if it is the stop character (the 8 zeros)
                            if (charValue == 0)
                            {
                                return extractedText;
                            }

                            // convert the character value from int to char
                            char c = (char)charValue;

                            // add the current character to the result text
                            extractedText += c.ToString();
                        }
                    }
                }
            }
            return extractedText;
        }

        public static int reverseBits(int n)
        {
            int result = 0;

            for (int i = 0; i < 8; i++)
            {
                result = result * 2 + n % 2;

                n /= 2;
            }

            return result;
        }
    }
```

Assume that the image was 200 pixels width by 200 pixels height, then we'll have 200 x 200 x 3 = 120000 LSBs. And as each character can be represented by 8 bits, then that image can hide 120000 / 8 = 15000 characters.

If the length of text is larger than that, we have to advise user that the text length is large.

```
private void btnEmbed_Click(object sender, EventArgs e)
{
    pnlSidePanel.Top = btnEmbed.Top;
    pnlSidePanel.Height = btnEmbed.Height;

    if (txtPath.Text.Equals(""))
    {
        MessageBox.Show(@"Image cannot be empty! Browse for an image and retry.", "Warning", 
MessageBoxButtons.OK, MessageBoxIcon.Warning);
        return;
    }

    _bitmap = (Bitmap)pbSelectedImage.Image;

    string text = txtTextToEmbed.Text;

    if (text.Equals(""))
    {
        MessageBox.Show(@"The text you want to hide can't be empty!", "Warning", MessageBoxButtons.OK, 
MessageBoxIcon.Warning);
        return;
    }
    _possibleChars = (_bitmap.Height * _bitmap.Width * 3) / 8;
    if (_possibleChars < text.Length)
    {
        MessageBox.Show(@"Text is too lengthy for the selected image. Insufficient space", "Warning",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
        return;
    }

    _bitmap = Steganography.EmbedText(text, _bitmap);

    MessageBox.Show(@"Your text was hidden in the image successfully! Don't forget to save your new image.",
"Done",MessageBoxButtons.OK,MessageBoxIcon.Information);
}
```

I used following paragraph as text to be hidden.

Video provides a powerful way to help you prove your point. When you click Online Video, you can paste in the embed code for the video you want to add. You can also type a keyword to search online for the video that best fits your document. To make your document look professionally produced, Word provides header, footer, cover page, and text box designs that complement each other. For example, you can add a matching cover page, header, and sidebar. Click Insert and then choose the elements you want from the different galleries.

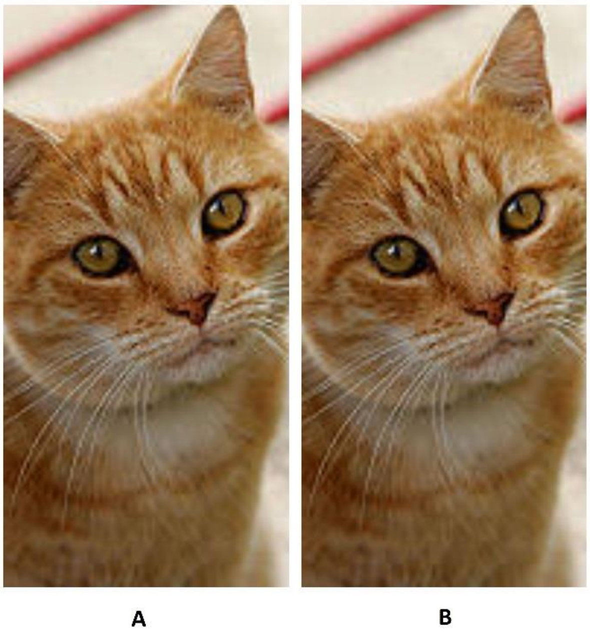


Figure 4. A: Original image B: Text embedded image.

As you can see, the difference is unnoticeable. Shown below is the code map for the executable used to extract text and insert text.

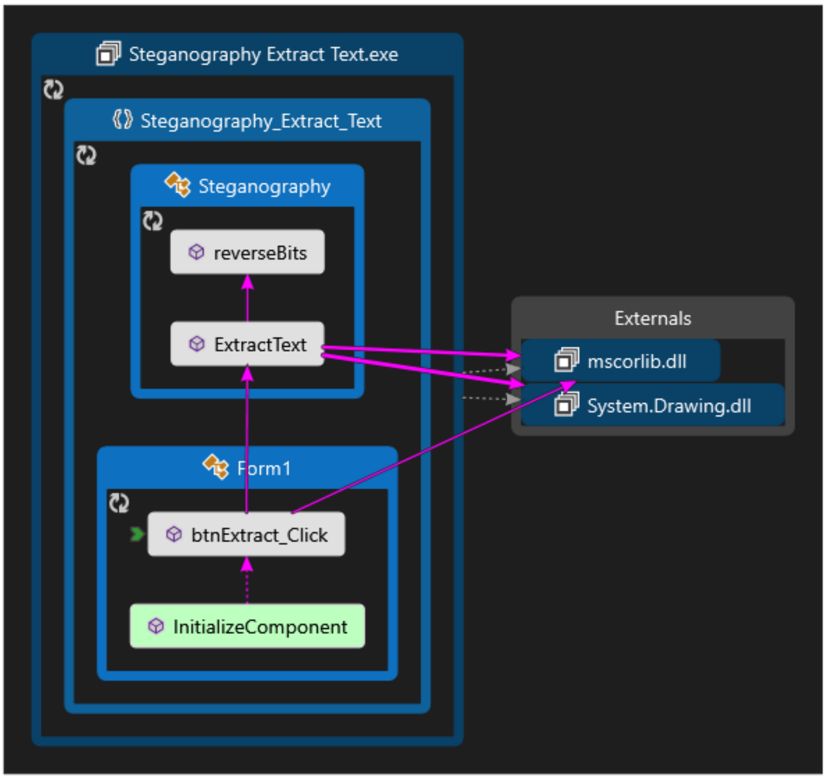


Figure 5. Extract text code map

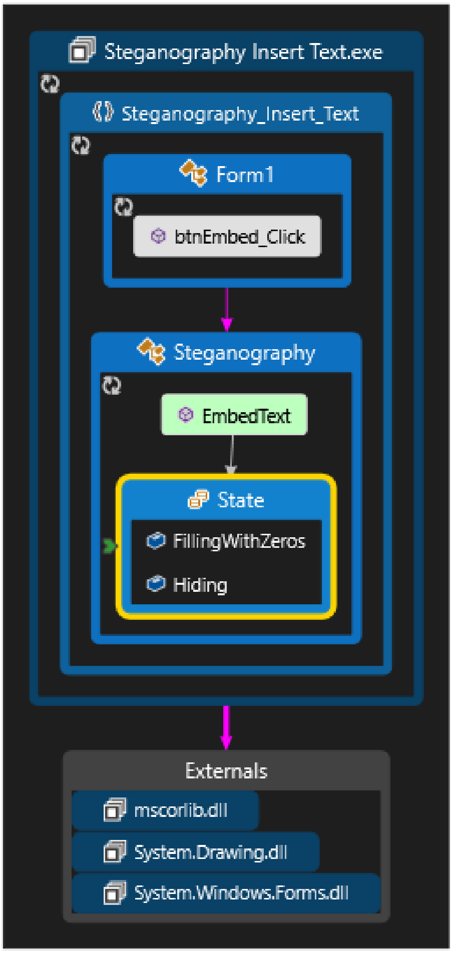
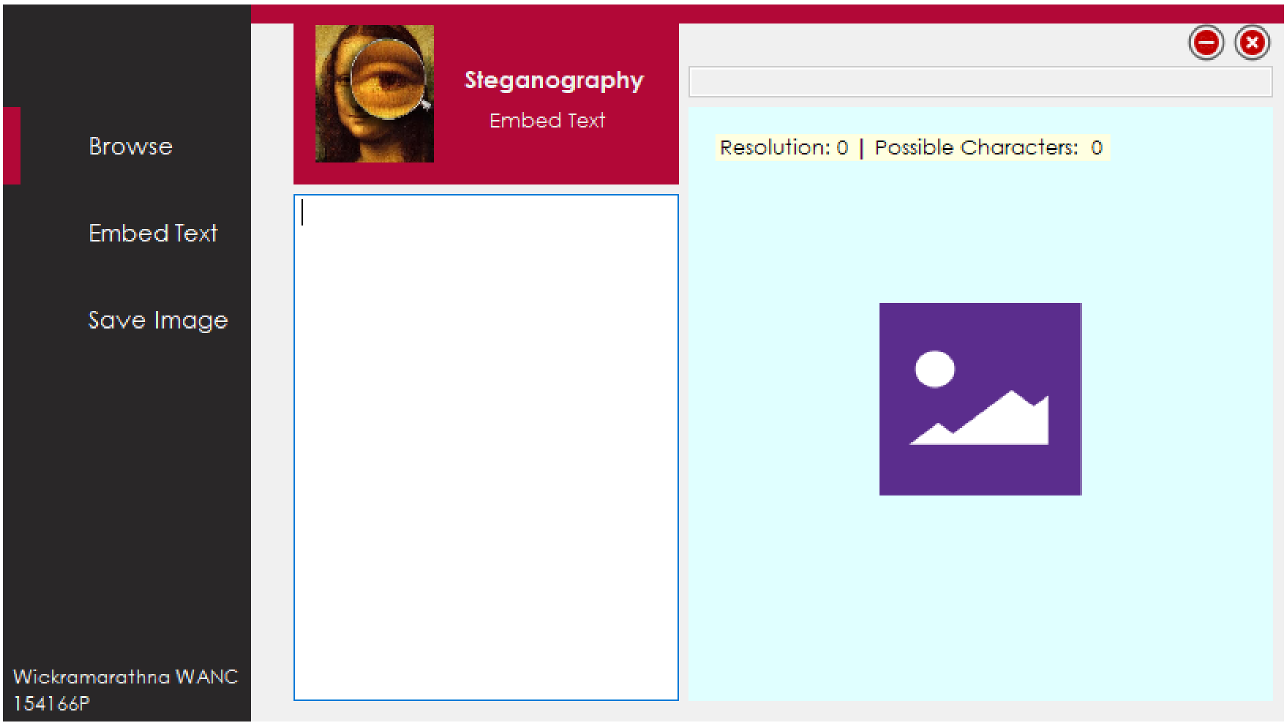


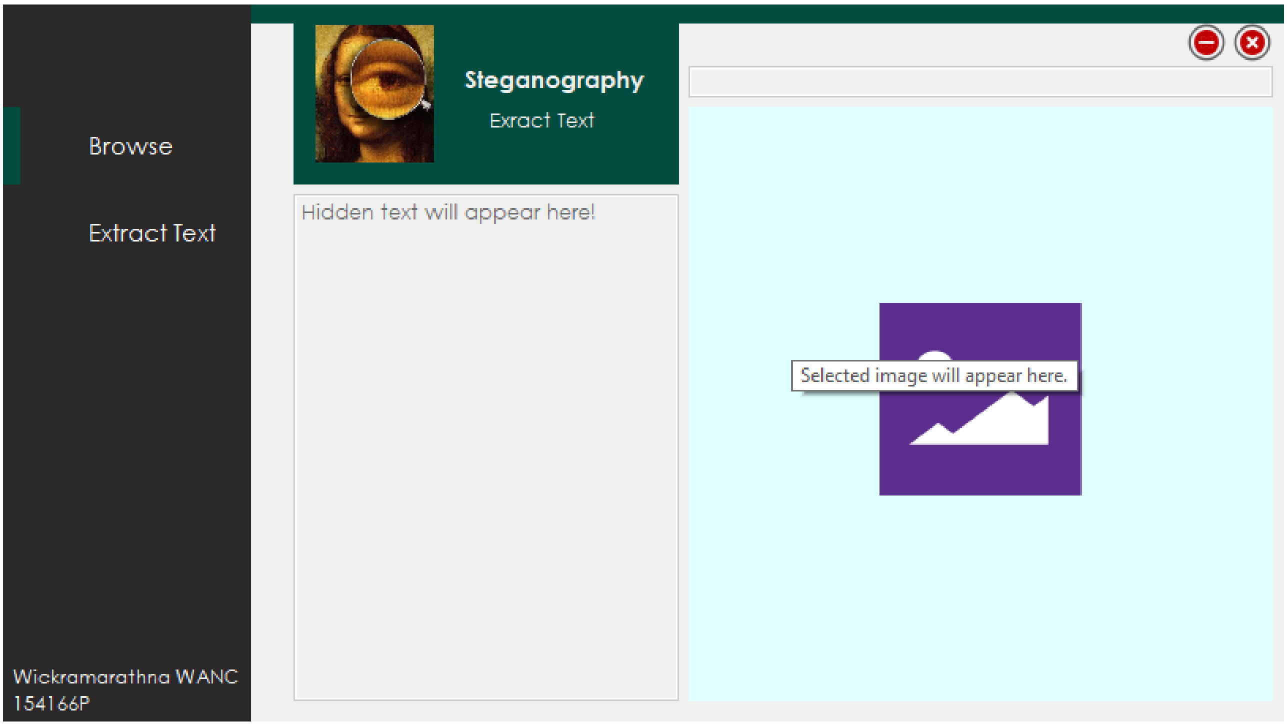
Figure 6. Insert text code map

User Manual

Following window will appear for the “Embed Text” program.

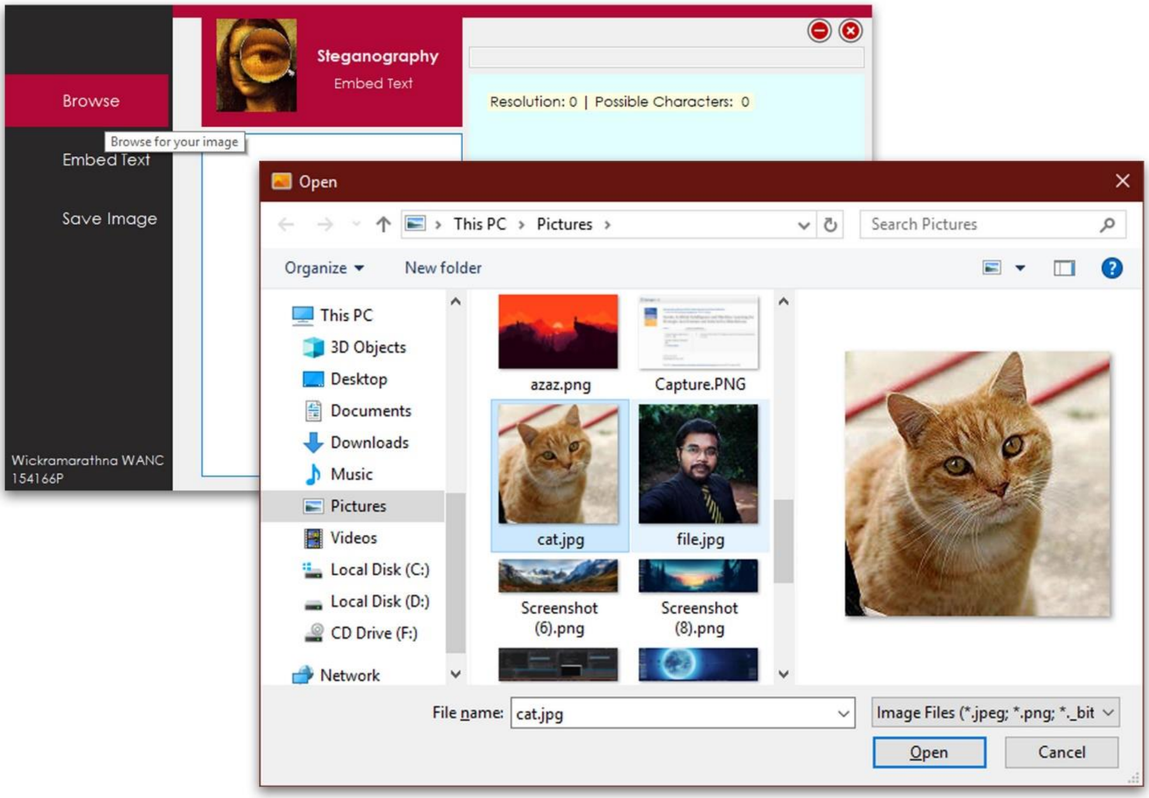


This widow will appear for “Extract Text” Program.



First, lets see how to embed text in an image.

Click on the “Browse” button and select an image.

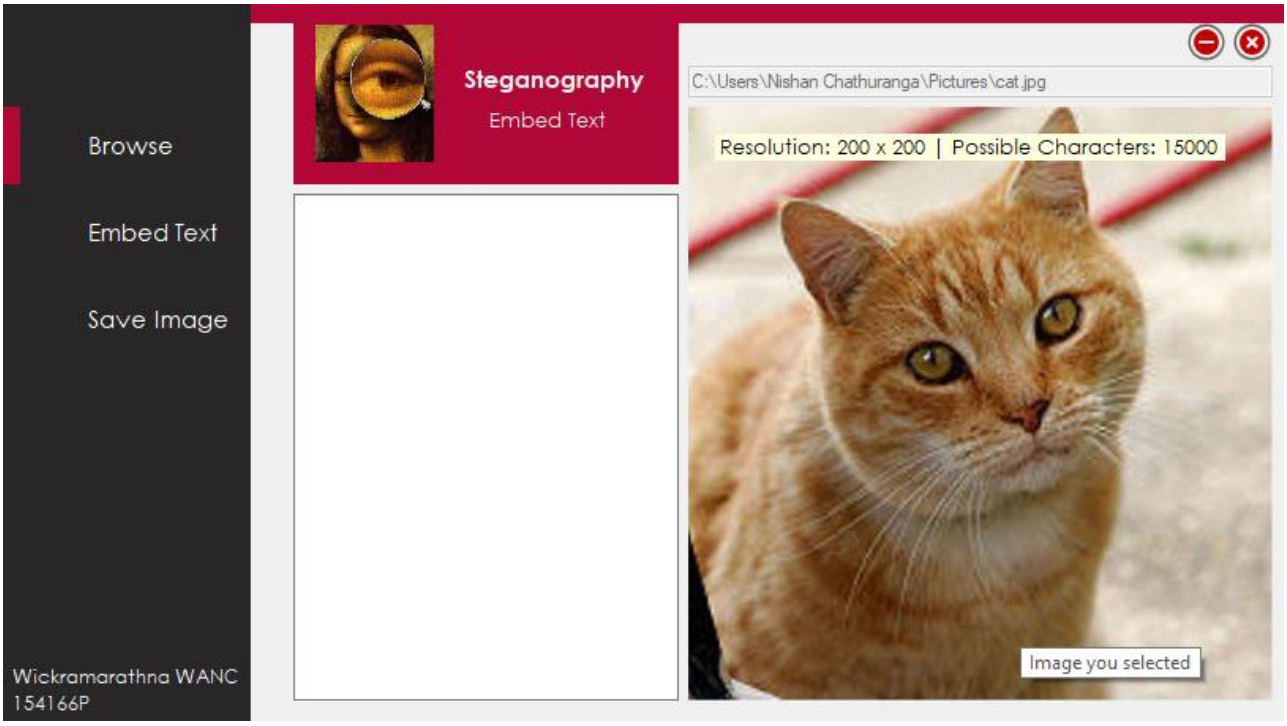


Resolution: 0 | Possible Characters: 0

File name: cat.jpg

Image Files (*.jpeg; *.png; *._bit

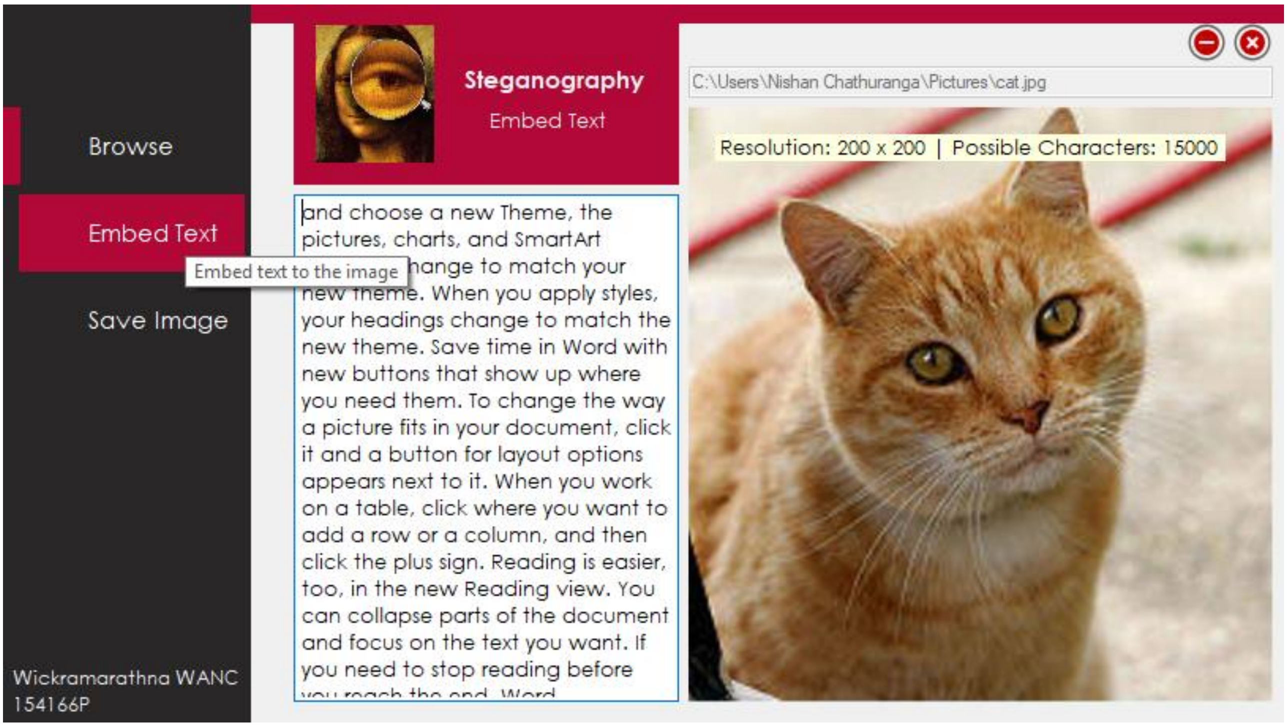
It will appear in the program, with respective resolution and number of characters allowed.



C:\Users\Nishan Chathuranga\Pictures\cat.jpg

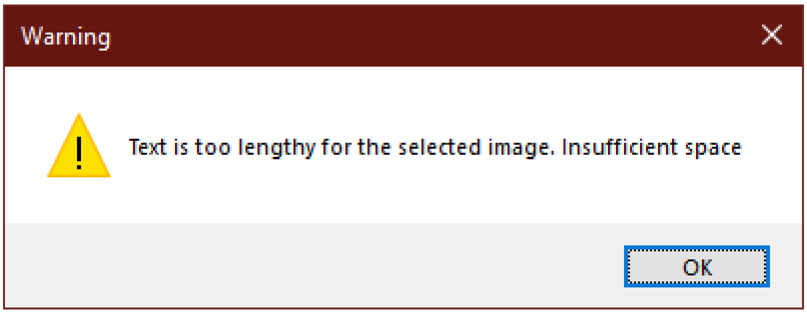
Resolution: 200 x 200 | Possible Characters: 15000

Next, paste the text that you want to be embedded in the image into the txt box and click "Embed".



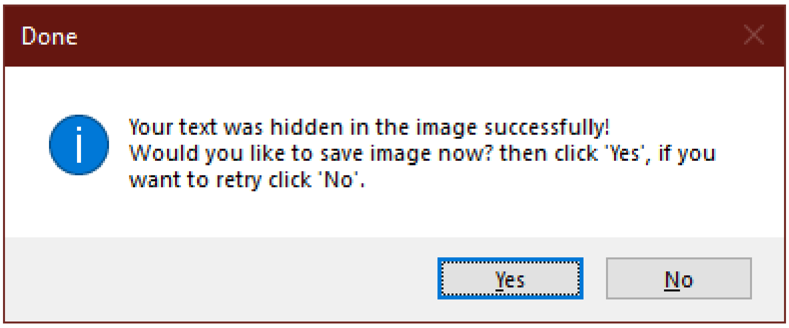
and choose a new Theme, the pictures, charts, and SmartArt hange to match your new theme. When you apply styles, your headings change to match the new theme. Save time in Word with new buttons that show up where you need them. To change the way a picture fits in your document, click it and a button for layout options appears next to it. When you work on a table, click where you want to add a row or a column, and then click the plus sign. Reading is easier, too, in the new Reading view. You can collapse parts of the document and focus on the text you want. If you need to stop reading before

If the character limit exceeds the possible character limit, it will give you following error message.



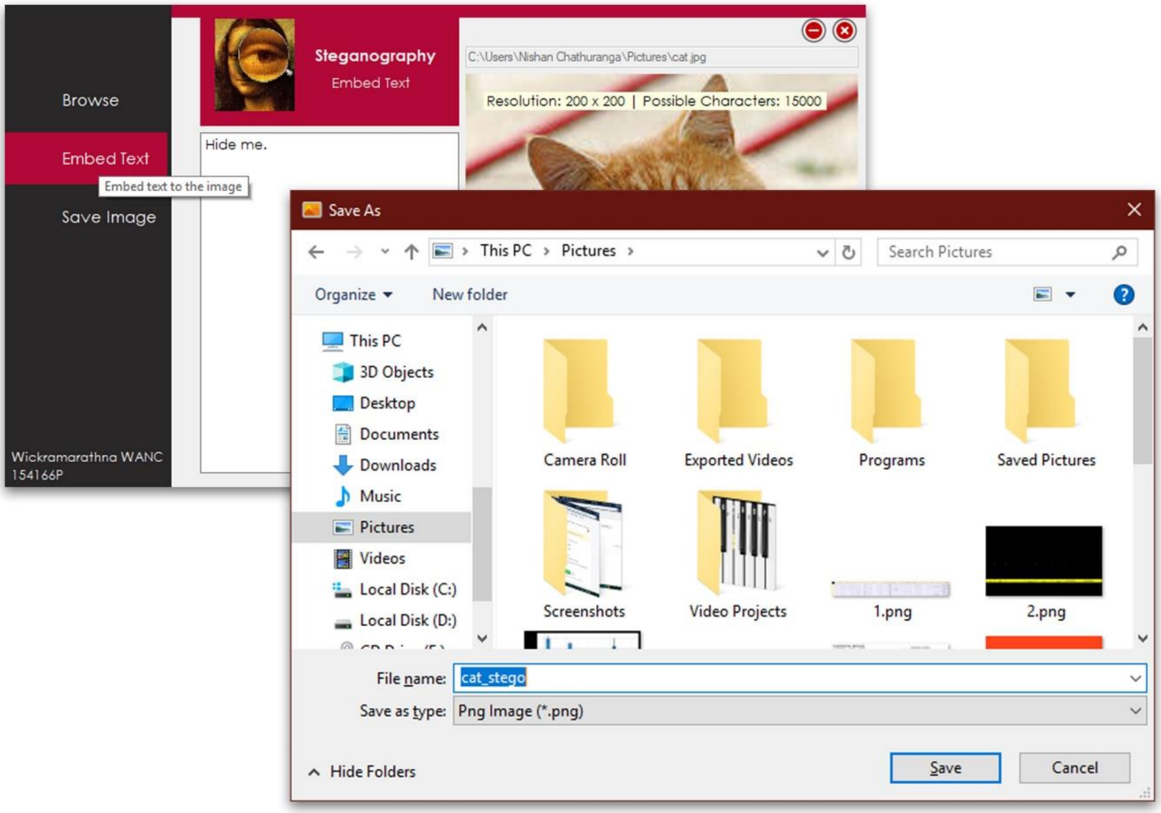
Text is too lengthy for the selected image. Insufficient space

If there's no error as such, you will get the following message.



Your text was hidden in the image successfully!
Would you like to save image now? then click 'Yes', if you want to retry click 'No'.

You can click "Yes" to save it, or you can click no and change the text again and click "Embed Text". You can also use the "Save Image" button to save image.



C:\Users\Nishan Chathuranga\Pictures\cat.jpg

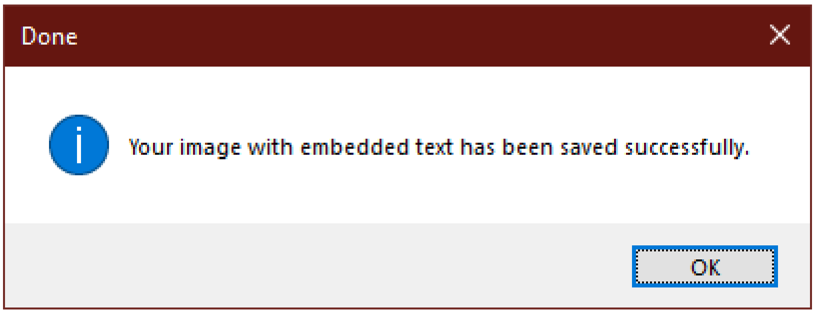
Resolution: 200 x 200 | Possible Characters: 15000

Hide me.

File name: cat_stego

Save as type: Png Image (*.png)

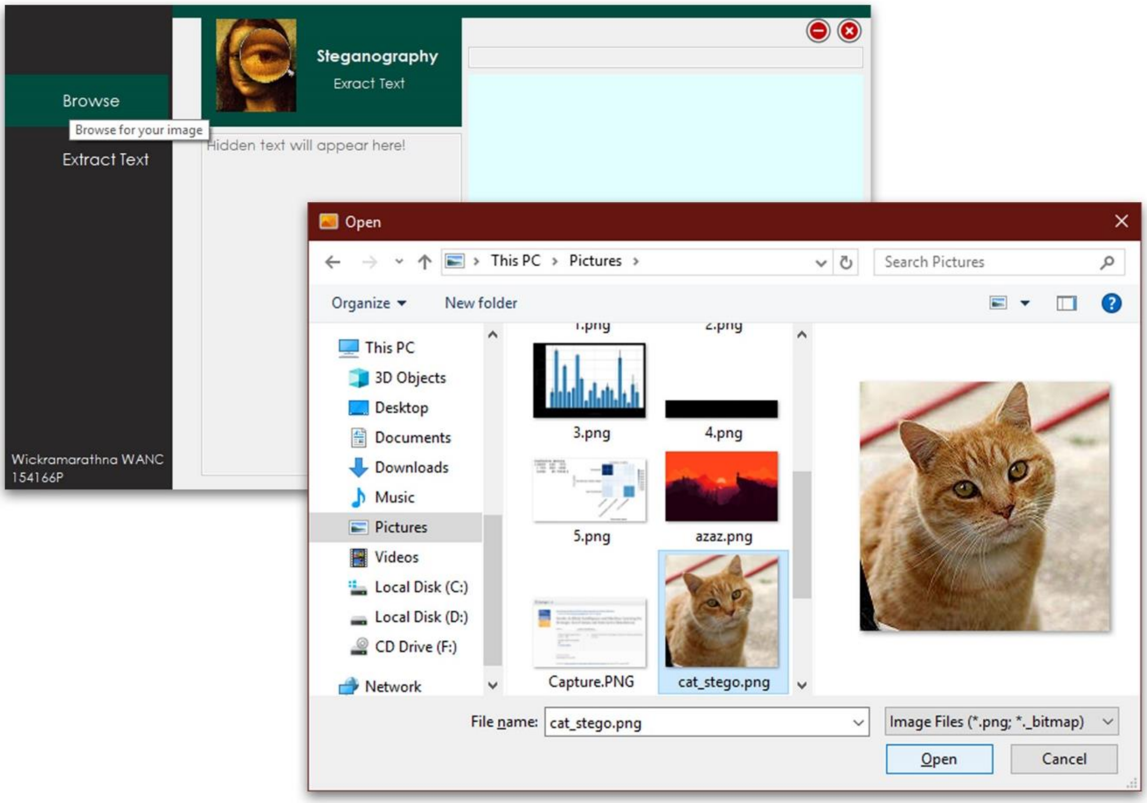
Give a file name for the new image and click "Save". If the save operation is successful this message will appear.



Your image with embedded text has been saved successfully.

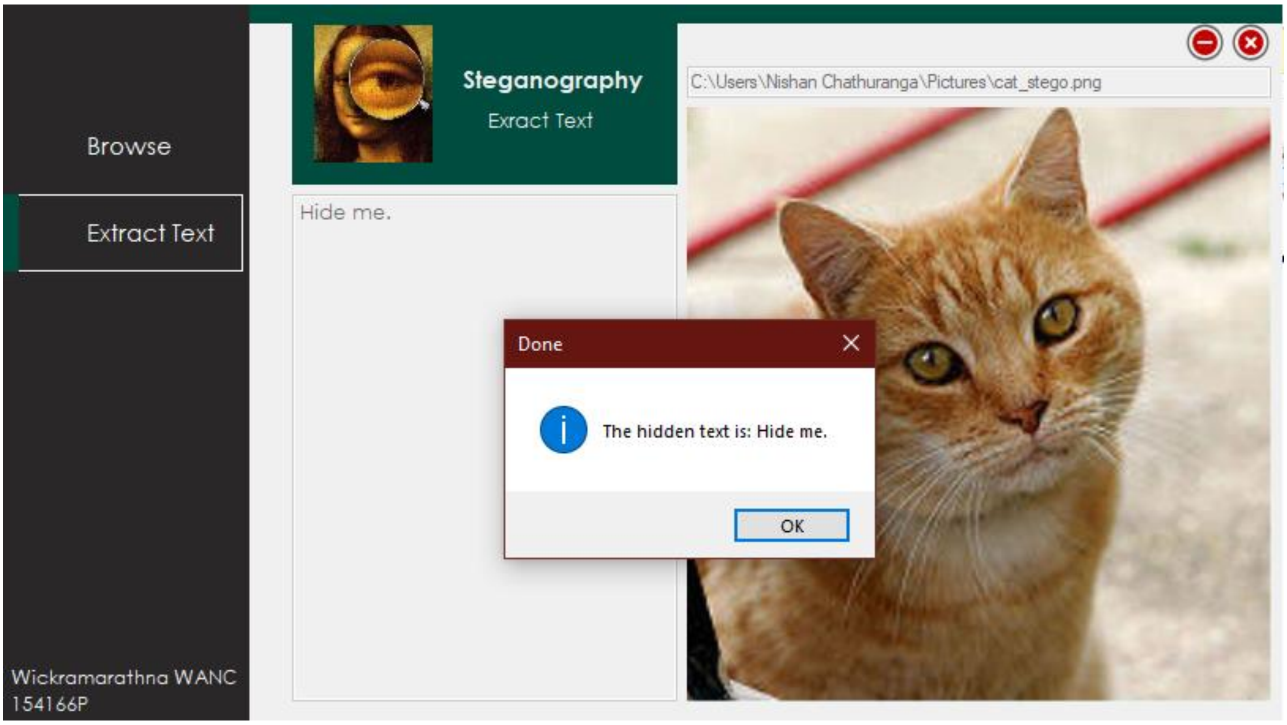
Now let’s see how the extract text program works.

Click on “Browse” button and select the image you want to extract text from.



File name: cat_stego.png

I have selected the image I saved earlier with a hidden text. After the image has been selected, click on “Extract Text” button to uncover hidden text.

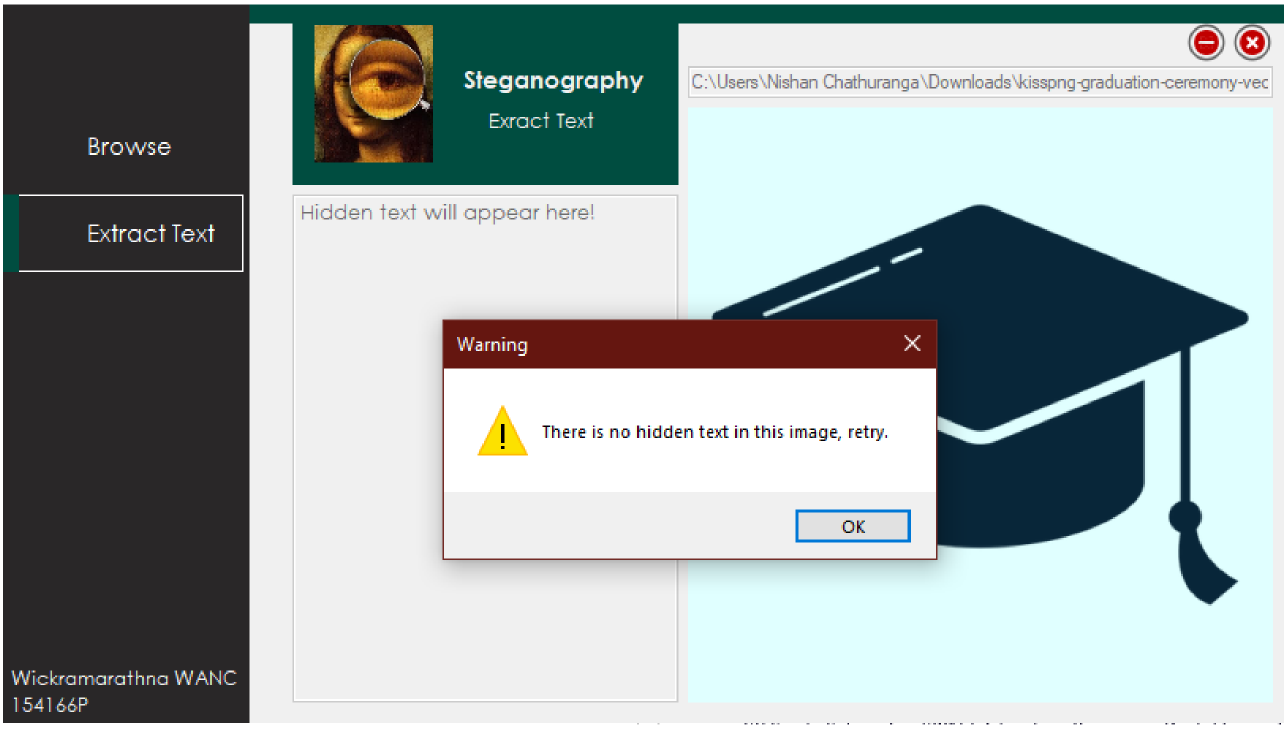


C:\Users\Nishan Chathuranga\Pictures\cat_stego.png

Hide me.

The hidden text is: Hide me.

If the image contains no hidden text, this message will appear.



Browse
Extract Text
Wickramarathna WANC
154166P
Steganography
Exract Text
C:\Users\Nishan Chathuranga\Downloads\kisspng-graduation-ceremony-vec
Hidden text will appear here!
Warning
OK

There is no hidden text in this image, retry.