Final Report

Steganography

Josh Ward

James Wilcox

24/03/17

Contents

1. Introduction………………………………………….**1-2**
   1. Abstract…………………………………………1
   2. Motivation………………………………………1
   3. Aims and Objectives……………………………1-2
   4. Scope and Limitations…………………………..2
2. Program Overview…………………………………..**3-4**
   1. Class hierarchy………………………………….3-4
      1. ‘Stego’……………………………………..3
      2. ‘Encode’…………………………………...4
      3. ‘Decode’…………………………………...4
   2. Function Summary……………………………...4-9
      1. ‘ImageAnalyser’…………………………...4-5
      2. ‘ArrayConverter’…………………………..5
      3. ‘ImageEncoder’……………………………6-8
      4. ‘ImageDecoder’……………………………8-9
   3. Additional Features……………………………..9-11
      1. Graphical User Interface (GUI)……………9-11
   4. Software Testing………………………………..11-13
      1. Methodology………………………………11-12
      2. Results……………………………………..12-13
3. Conclusion and Evaluation………………………….**14**
   1. Evaluation of Aims and Objectives…………….14
   2. Further Work……………………………………14
4. References...…………………………………………**15**
5. Appendix…………………………………………….**16-19**
6. Introduction
   1. Abstract

Within this project, a steganography program will be created that will be able to hide an image within another image. The main method of steganography that has been investigated within this report is the Least Significant Bit (LSB) method which involves changing the last two bits of the binary sequence of the RGB values of a cover image to match the RGB values of the image that is being hidden [1]. Other methods have also been discussed for performing steganography and the appropriateness of different file types have been investigated.

* 1. Motivation

Steganography has long been used as a method of information transfer when discretion is a primary concern. Most currently used methods of steganography are based around the concealment of digital data, whether it is hidden within an image, an audio file, or a video, however, the concept dates back thousands of years [4]. An example that is often given for the use of steganography dates to the time of the Roman Empire where messages where written on the scalp of messengers and the messenger’s hair would the hide the message.

Modern methods of steganography pose large challenges for the monitoring of illicit activities as properly executed steganography will be innocuous, reducing the chances that someone would suspect that there is anything hidden at all [3]. Steganography can also be combined with cryptography to protect the data hidden within if it is discovered.

Developments are being made continuously on both sides of the battle: Advances in steganalysis software have enabled more accurate predictions of the likelihood that a given image contains data hidden within it; conversely, more sophisticated steganography methods are being developed that make detecting the use of steganography much more difficult.

* 1. Aims and Objectives

The main objectives of this project were as follows:

* Research possible steganography techniques and discuss the relative merits and challenges of each.
* Investigate the appropriateness of different file types for this application with respect to complexity and level of compression.
* Write a program to hide a user-selected image within a cover image.
* Create a Graphical User Interface for the program.
* Implement encryption of the hidden data.
* Write a program to detect the presence of hidden data within an image.
  1. Scope and Limitations

Scope:

Upon conducting research, it was found that the most appropriate file type for this application is the bmp file type. The jpeg file type is unsuitable for this application due to its aggressive compression of files. The use of jpeg would likely cause the data hidden within the file to become corrupted or unrecoverable [1]. Other file types that could be used for this application include tiff and png files, these file types also contain an alpha value that dictated the opacity of the pixel but this was found to cause no problems. Upon conducting testing, it was discovered that any raster file type that uses light or no compression is compatible with the program.

The steganography technique that will be demonstrated within this report is the Least Significant Bit (LSB) method of steganography. This has been chosen due to its relative simplicity compared to other methods [2]. The technique works by taking a cover image and hiding another Image within the least significant bits of the pixels of the cover image, this in turn creates a new stenographic image which contains both images however only one is visible to an observer. However, this method is susceptible to a ‘visual attack’ where upon a close examination an observer can notice the noise in the image, making it exploitable.

Limitations:

Another method which could be used for image steganography is discrete cosine transform steganography. This method works by taking a .jpeg image and separating the image into parts of differing importance, this separates the image into high, middle, and low frequency. The hidden image is then hidden within these frequencies; this allows the image to be hidden and is more effective than LSB steganography as it is not exploited using a ‘visual attack’ [2]. However, due to time constraints placed upon this project, this will not be investigated at this stage.

A common way of counteracting the use of steganography is using steganalysis software. These software programs commonly work by performing ‘statistical attacks’ on images that are suspected of containing hidden data [7]. However, most of these programs are only capable of estimating the probability that data is hidden within an image and cannot retrieve the data unless the steganography method is known. The creation of a steganalysis program could be investigated in the future given more time.

Further error handling could be carried out within the program to ensure that the program will not run when a wrong input is selected such as the user not selecting a bitmap image.

1. Program Overview
   1. Class Hierarchy

Figure 1 below shows the class hierarchy used within the steganography program.

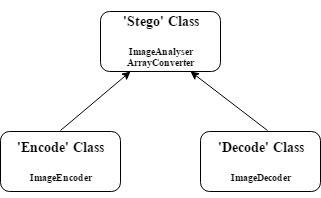


Figure 1. Class hierarchy diagram

The child classes (‘Encode’ and ‘Decode’) inherit both the ‘ImageAnalyser’ and ‘ArrayConverter’ functions from the parent (‘Stego’) class. The use of a class structure within the program improves the efficiency of the program by enabling the functions to be used multiple times without redefining them every time they are required. It also creates a much more structured program in which the relationships between functions is much clearer.

* + 1. ‘Stego’

The main function of the ‘Stego’ class is to facilitate reuse of the ‘ImageAnalyser’ and the ‘ArrayConverter’ functions for both the ‘Encode’ and ‘Decode’ classes.

* + 1. ‘Encode’

Figure 2 below shows the beginning code for the ‘Encode’ class.

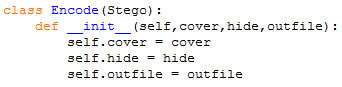


Figure 2. ‘Encode’ class definition

The ‘Encode’ class controls the encoding of the image to hide within the cover image. The cover image, image to hide, and the file to which the stego object will be written are passed into this class. This class contains the ‘ImageEncoder’ function which performs the encoding. After the encoding process is complete, the stego object which has been created is saved to the file defined in the outfile variable.

* + 1. ‘Decode’

Figure 3 below shows the beginning code of the ‘Decode’ class.

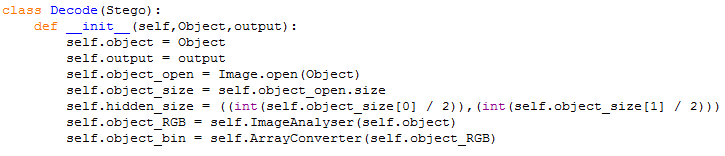


Figure 3. ‘Decode’ class definition

The ‘Decode’ class is a container for the ‘ImageDecoder’ function. The stego object is passed into this class and from this, the size of the hidden image is calculated by dividing both dimensions of the stego object by two. After the hidden image is retrieved it is saved to the file defined in the output variable.

* 1. Function Summary
     1. ‘ImageAnalyser’

Figure 4 below shows the script for the ‘ImageAnalyser’ function.

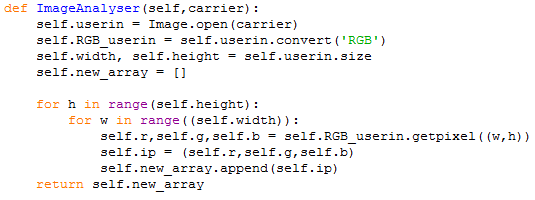


Figure 4. ‘ImageAnalyser’ function script

The process of the ‘ImageAnalyser’ function is as follows:

* The ‘ImageAnalyser’ function opens the target image and converts it to RGB format.
* Two variables (‘self.width’ and ‘self.height’) are then created to store the width and height of the target image.
* A new, empty array (‘self.new\_array’) is created.
* An iteration for each value in ‘self.width’ is set up within a loop of length ‘self.height’ and each R, G, and B value of the target image is read, stored in a list, and the list is then appended to the array ‘self.new\_array’.
* The function then returns the array ‘self.new\_array’.
  + 1. ‘ArrayConverter’

Figure 5 below shows the python script for the ‘ArrayConverter’ function.

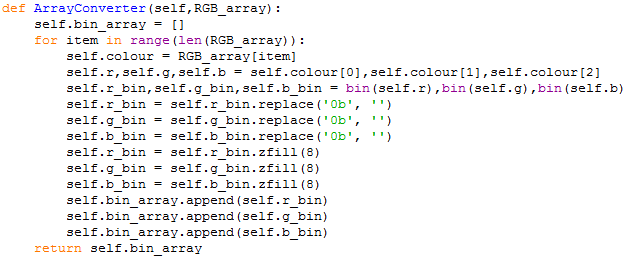
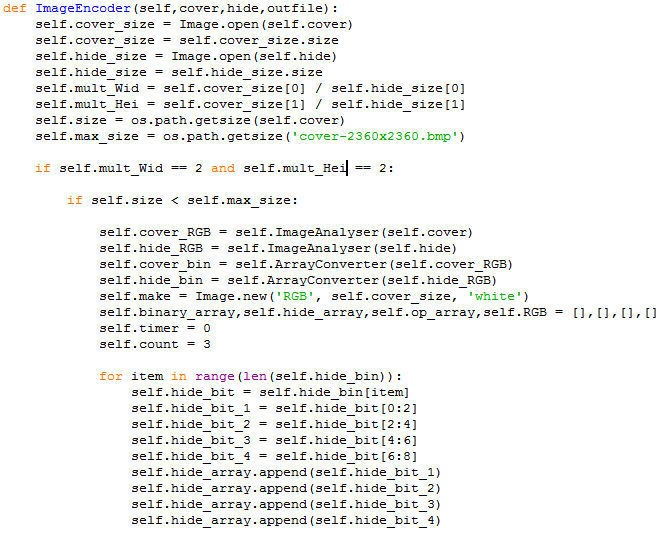


Figure 5. ‘ArrayConverter’ function script

The ‘ArrayConverter’ function carries out the following process:

* The function takes in the array returned by an instance of the ‘ImageAnalyser’ function.
* A new, empty array (‘self.bin\_array’) is created.
* An iteration is set up for the length of the input array.
* The values of each RGB sequence are separated and converted into binary.
* The two signing bits (‘0b’) are removed from the start of the binary values.
* ‘zfill’ is used to ensure that all binary values are an 8-bit sequence.
* The binary values are appended to the array ‘self.bin\_array’.
* The function returns the array ‘self.bin\_array’.
  + 1. ‘ImageEncoder’

Figure 6 below shows the python script for the ‘ImageEncoder’ function.



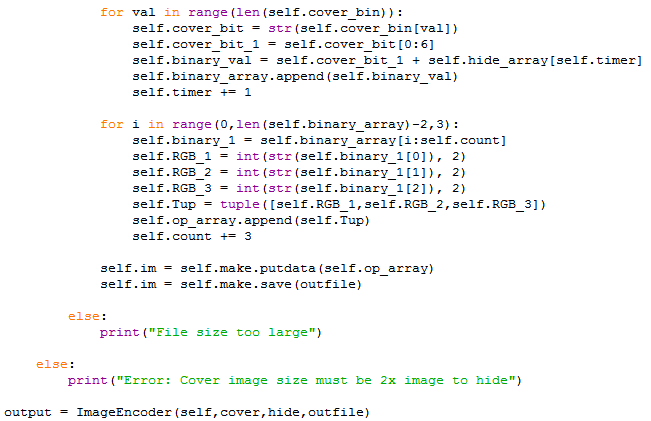


Figure 6. ‘ImageEncoder’ function script

Figure 7 below shows the flowchart that the ‘ImageEncoder’ function follows. [5]

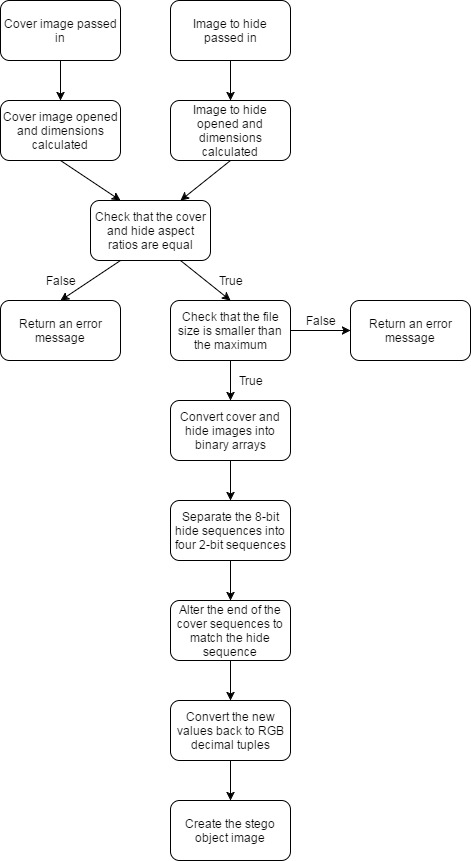


Figure 7. ‘ImageEncoder’ flowchart

The process of the ‘ImageEncoder’ function is as follows:

* The image to hide, cover image, and outfile are passed into the function.
* The cover image and image to hide are opened and their dimensions are stored in a variable.
* The relationship between the dimensions of the cover image and image to hide are calculated.
* The file size of the cover image is found and the maximum file size that was found from the software testing was found.
* The aspect ratio of the cover image and the image to hide are compared to ensure that they are the same.
* The file size of the cover image is checked to ensure that it is lower than the maximum.
* If either of the two above conditions are not true, the rest of the script is not run and an error message is displayed in the python shell.
* If the conditions are true, the following process will occur.
* Instances of the ‘ImageAnalyser’ function for the cover image and the image to hide were created.
* Instances of the ‘ArrayConverter’ function were created for the arrays returned from the instances of the ‘ImageAnalyser’ function.
* A new, blank image is created.
* Four new, empty arrays are created.
* Two timing variables are created.
* For each item in the array created from the instance of the ‘ArrayConverter’ function for the image to hide, the 8-bit binary sequences are separated into four 2-bit sequences and stored in an array, self.hide\_array.
* For each item in the array created from the instance of the ‘ArrayConverter’ function for the cover image, the first six bits are selected and one of the 2-bit sequences from ‘self.hide\_array’ are added to the end. This new 8-bit sequence is then added to the array, ‘self.binary\_array’.
* Three values of ‘self.binary\_array’ at a time are converted to decimal, casted to a tuple, and stored in an array, ‘self.op\_array’.
* The data from ‘self.op\_array’ is added to the blank image created previously.
* The image with the added data is then saved.
* An instance is created of the ‘ImageEncoder’ function.
  + 1. ‘ImageDecoder’

Figure 8 below shows the python script for the ‘ImageDecoder’ function.

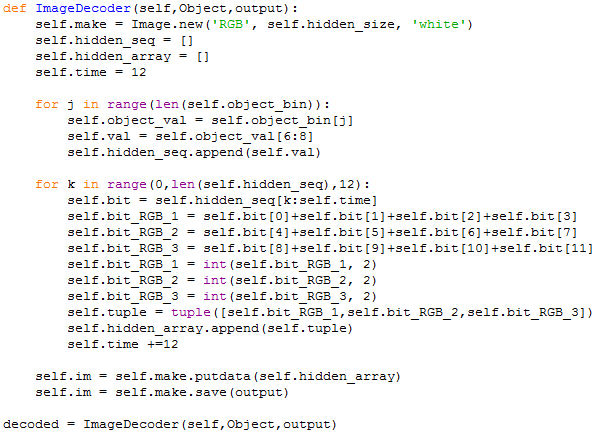
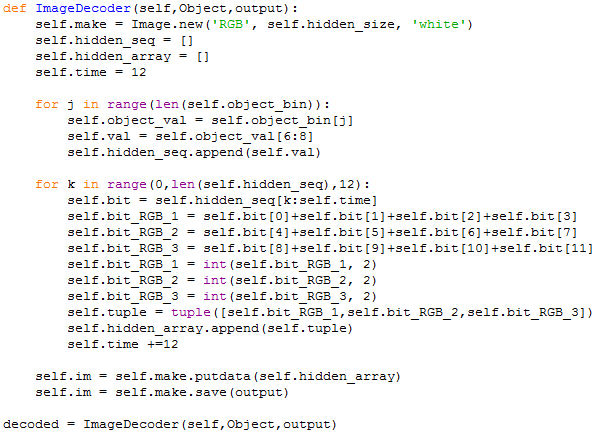
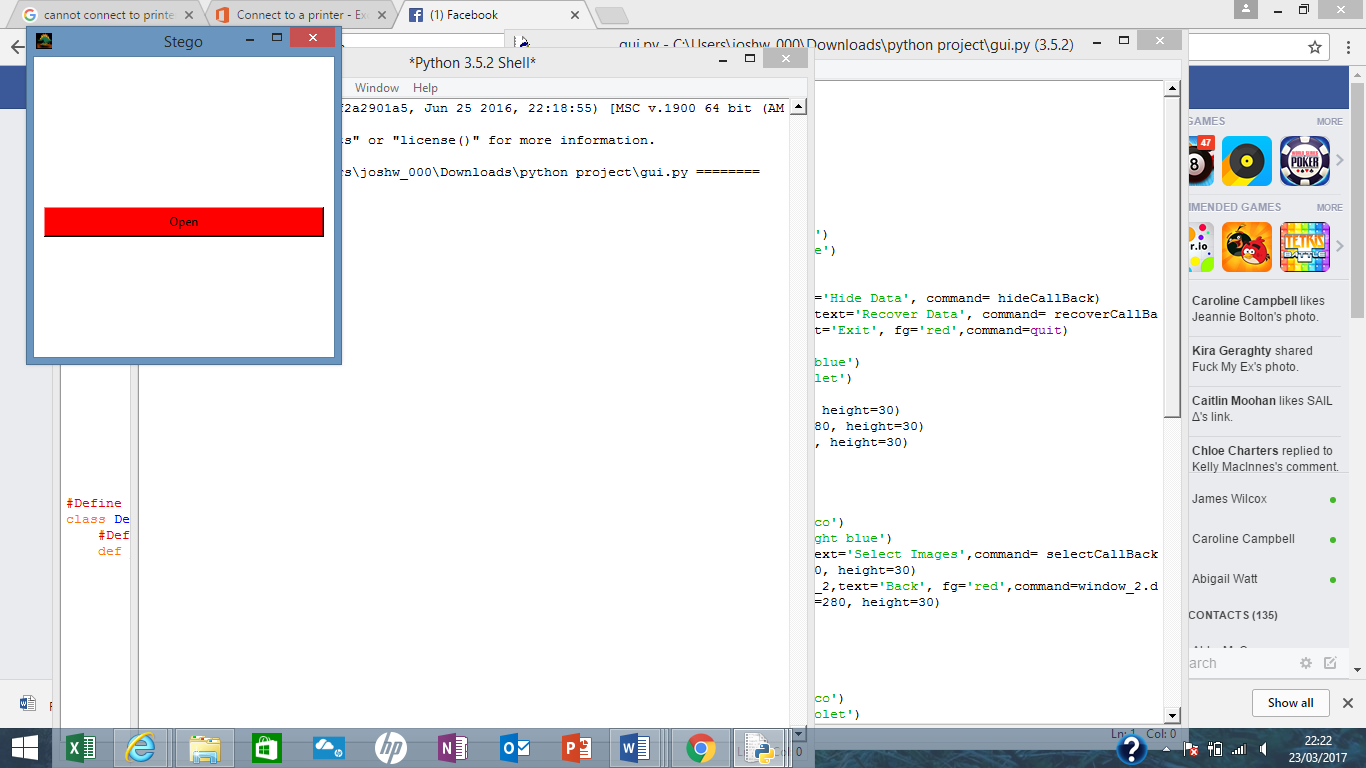


Figure 8. ‘ImageDecoder’ function script

The ‘ImageDecoder’ function follows the process detailed below:

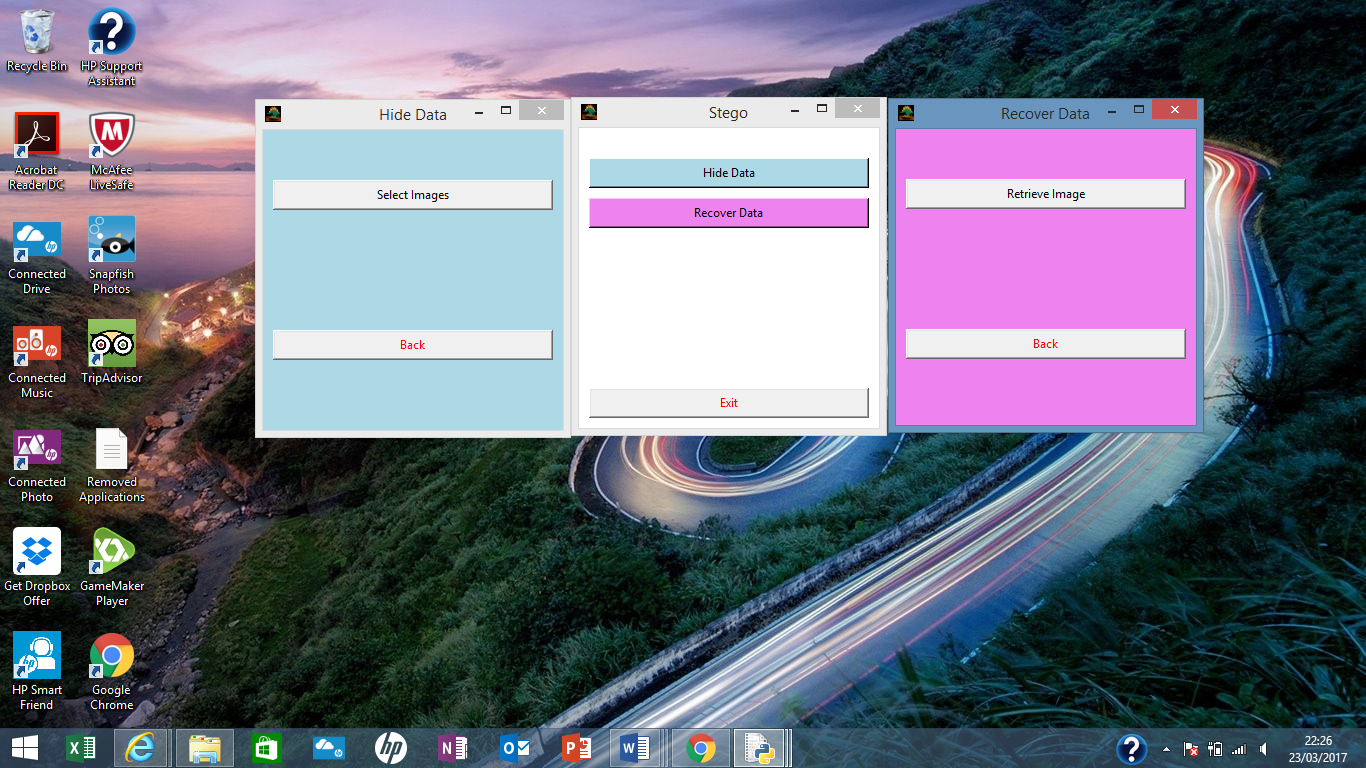
* The stego object and the output file are passed into the function.
* A new, blank image is created.
* Two empty arrays are created.
* A timing variable is set up.
* For each value in the array created from the instance of the ‘ArrayConverter’ function for the stego object, the last two bits of the 8-bit binary sequences are selected and added to an array, ‘self.hidden\_seq’.
* Four of the 2-bit sequences from ‘self.hidden\_seq’ are added together to get the values of the hidden data. These are grouped in groups of three to make up each pixel’s values.
* The three 8-bit hidden data sequences are cast to decimal and then casted as a list to a tuple. This tuple is then added to an array, ‘self.hidden\_array’.
* The data from ‘self.hidden\_array’ is then added to the blank image created previously.
* The image with the new added data is then saved.
* An instance is then created of the ‘ImageDecoder’ function.
  1. Additional Features
     1. Graphical User Interface (GUI)

The Graphical user interface or GUI is used in combination with the overall steganography, this gives a user-friendly platform to conduct the steganography. The GUI is made using python’s Tkinter module as well as filedialog which is imported from the Tkinter module, these allow for the creation of the GUI, widgets and windows as well as the browse functions respectively. The GUI is accessed first using a simple open button however in further work could be changed to incorporate a password system in order to make the system even more secure.



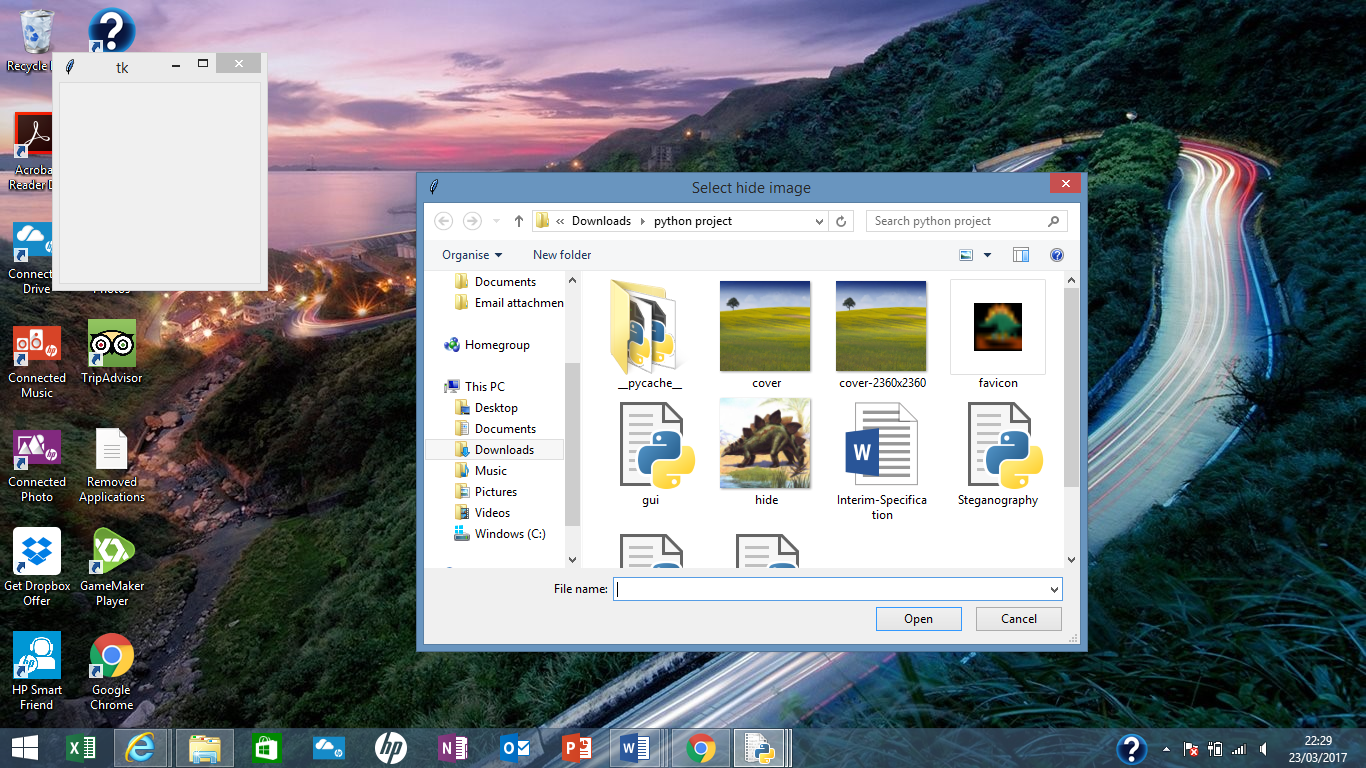
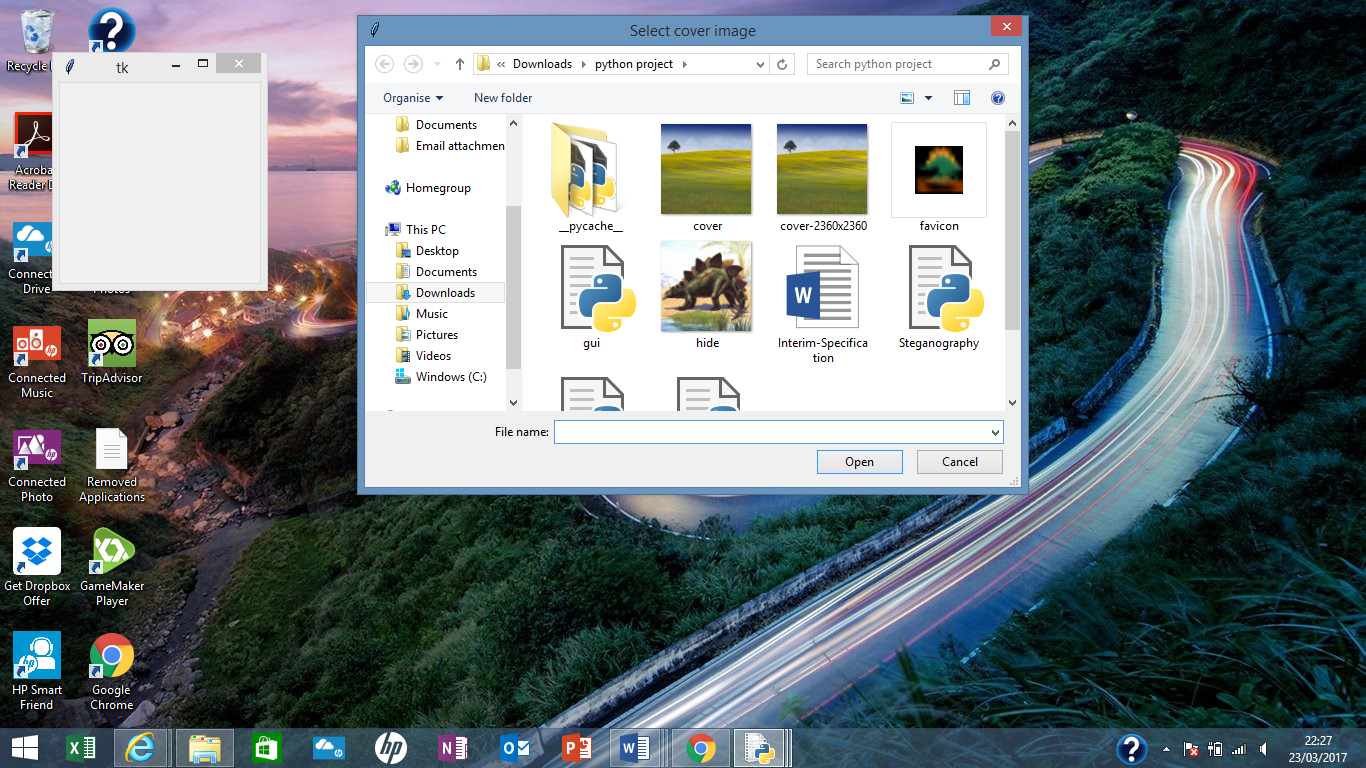
GUI open window

After the window is accessed the menu section of the GUI appears which allows the users to select the Hide Data and Retrieve data functions respectively, further work in this area could involve disabling the retrieve button until the steganographic image is processed and made using the hide data button.

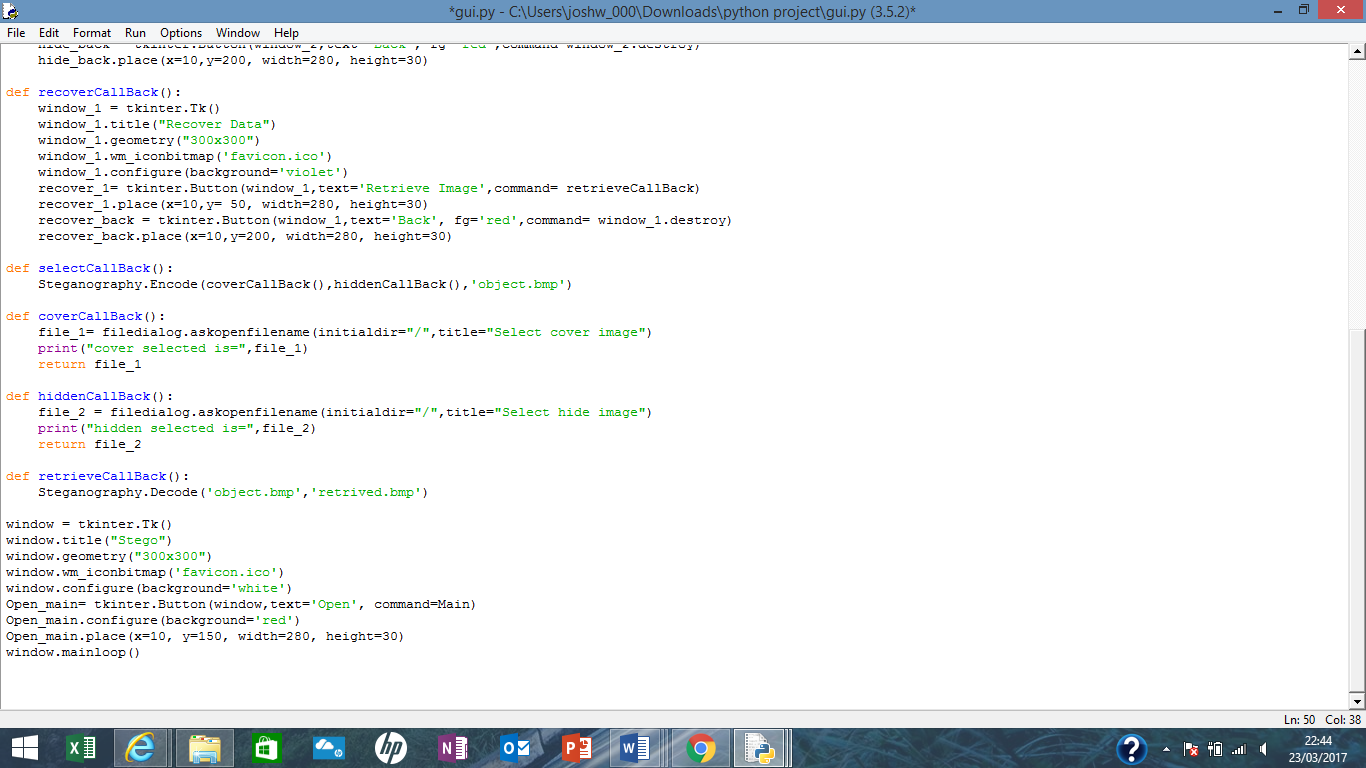


GUI Menu with selected windows

Once accessed, the Hide data allows the user to choose a cover image as well as an image they are wishing to hide within it using a browse function using the filedialog function imported from Tkinter, this is shown below:



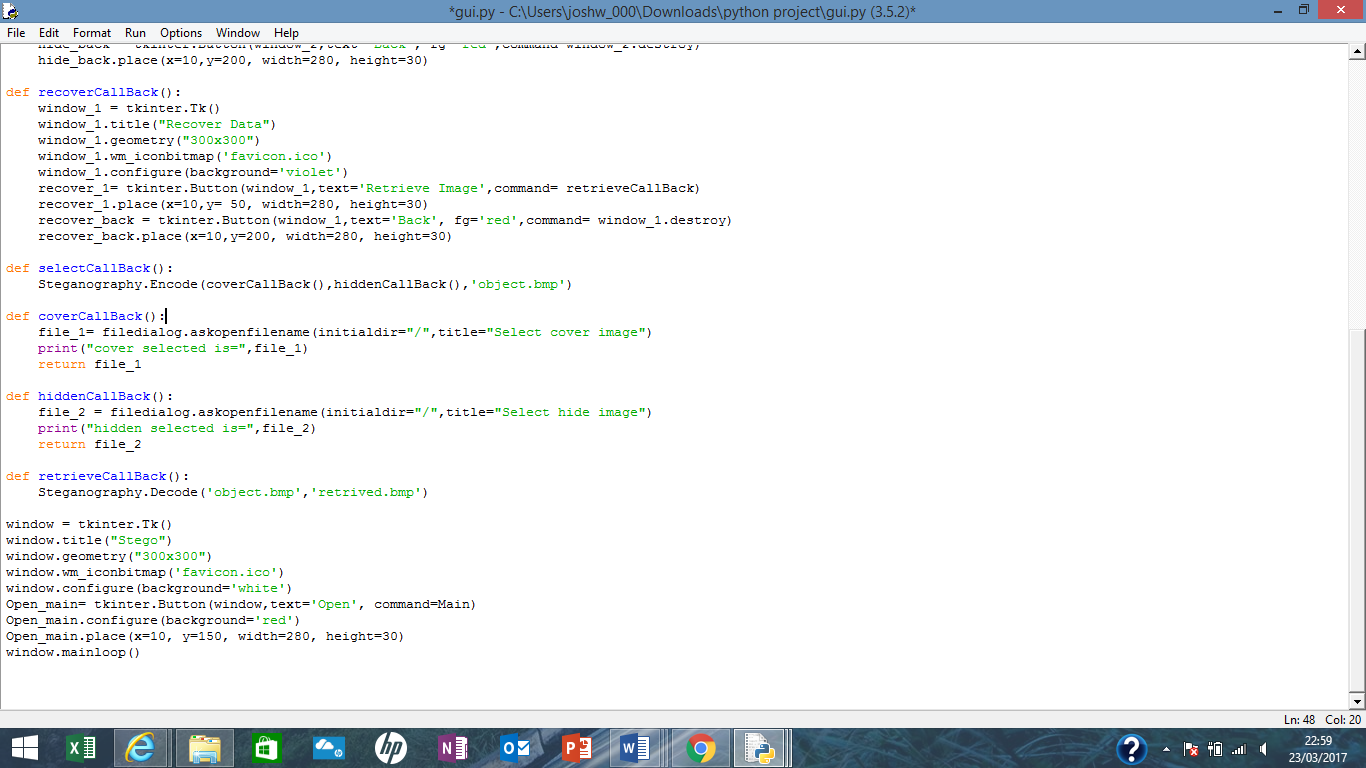
GUI browse functions

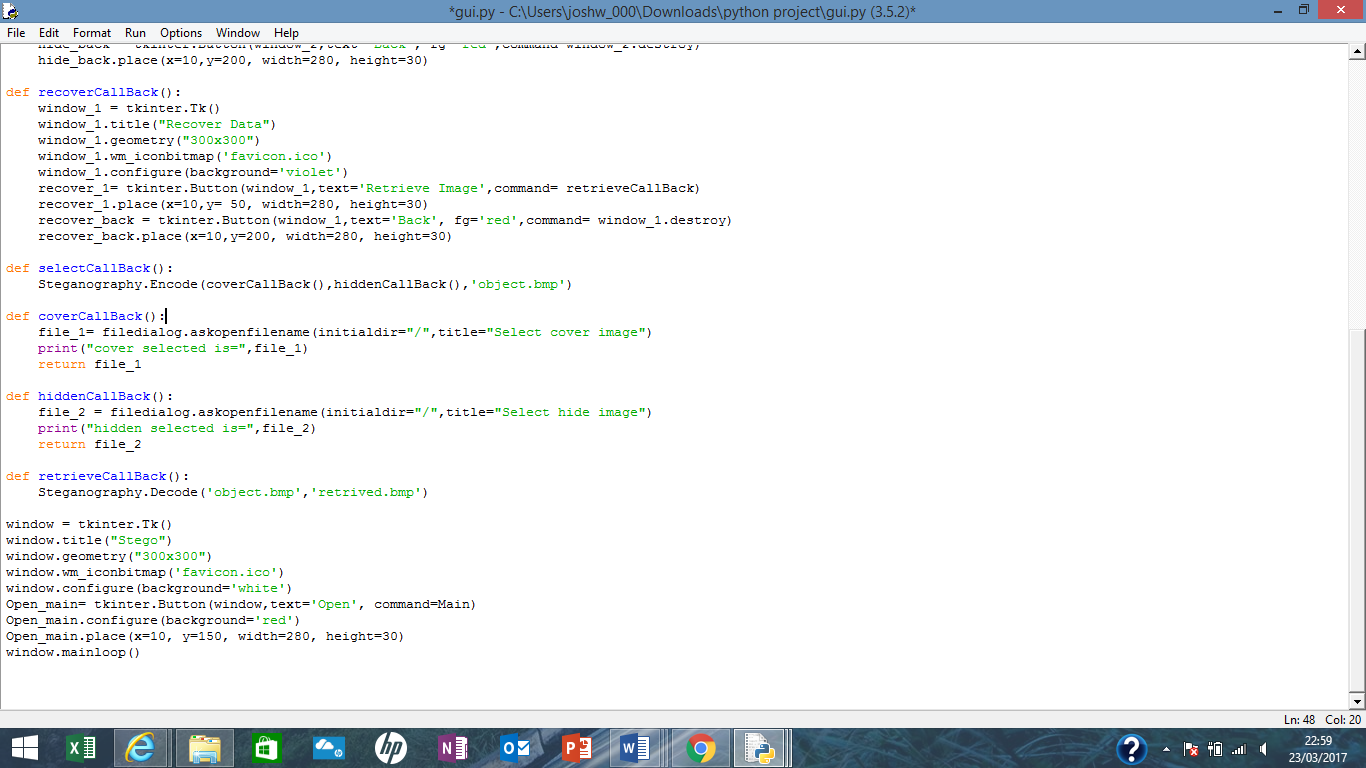


Code written for browse function

Once the images are selected the encode class of the steganography module will activate taking a few seconds to run before outputting an output image named ‘object’, further work in this section could be to have a progress bar which shows the user when the encode class has completed and the outfile is made. After this the user, can press the retrieve button which access’s the decode class of the steganography module and outputs an image named ‘retrieved’, this is the decoded image taken from the steganographic image.

The GUI works by importing the Steganography module and then calling the encode class from the hide data section of the interface, where the retrieve data function calls upon the decode class from the steganography module as can be seen below.





* 1. Software Testing
     1. Methodology

The primary focus of the software testing conducted for this project was to test the effect of increasing the file size of the cover image and image to hide on the time taken to encode and decode the data hidden within the cover image. Bitmap images of the cover image and image to hide were created for each size using paint to resize the original cover image and image to hide.

To get an accurate representation of the time taken to carry out the program, the ‘time’ library was used. Figure 9 below shows the code used at the beginning of the script to store the current time in a variable.

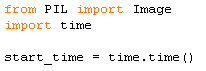


Figure 9. Timing initialisation

At the end of the program, the current time was then compared with the time recorded at the beginning of the script. This gives the time that was taken to run the program in seconds. The hash mark was used to isolate either the ‘Encode’ function or the ‘Decode’ function. Figure 10 below shows the code used the find the time taken to run the program.

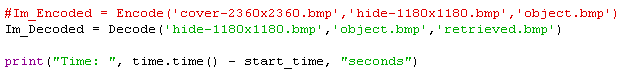


Figure 10. Print timing result

For this test, variables were controlled by ensuring that the script used for testing contained only code that was essential for the program to run properly [6]. The only programs running on the test computer during the testing stage were the Steganography python script, the python shell, and Excel (for recording the results). The specifications of the test computer were as follows:

* Intel Core i5-6200U (2.3GHz), NVIDIA GeForce 940m (2GB), 8 GB DDR3 L, 1 TB HDD, Windows 10 running Python 3.6 (64-bit).

In addition to the file size test, tests were also run to test the compatibility of the program with different file types (jpeg, png, tiff, bmp) and different aspect ratios (1:1, 4:3, 16:9).

* + 1. Results

The tests were conducted as described in section 2.4.1. above. Table 1 below shows the results obtained for the time taken to run the program for different file sizes.

Table 1. Results of timing test



The results obtained in Table 1 above were used to produce the following graph in Figure 11 below. This graph shows the file sizes which gave valid results against the time that was taken to run the program.

Figure 11. Plot of valid results against time taken

The main findings from the software tests conducted were as follows:

* The cover image and image to hide must be of the same aspect ratio. However, from testing, a particular aspect ratio is not required as all of the aspect ratios tested (1:1, 4:3, and 16:9) functioned as expected.
* The dimensions of the cover image must be twice those of the image to hide, i.e. the size of the cover image must be quadruple that of the image to hide.
* The maximum file size of the cover image that was successfully run was 15.9 MB, therefore, if statements have since been added to the script to ensure that this file size is not exceeded to ensure that the program runs correctly.
* From tests with a variety of file types (jpeg, tiff, bmp, and png), it was found that all except the jpeg file types worked as expected due to their much lower compression than jpeg. Figure 12 below shows the image to hide in jpeg before encoding (left) and after the attempted retrieval (right).



Figure 12. Attempted jpeg program original (left) and retrieved (right)

1. Conclusion and Evaluation
   1. Evaluation of Aims and Objectives

The first two objectives of researching possible steganography techniques and researching the suitability of different file types were covered extensively within the scope and limitations section of this report (section 1.4).

The third objective of writing a program to hide a user-selected image within a cover image was also completed very successfully as demonstrated in sections 2.1 and 2.2 of this report.

The fourth objective of creating a Graphical User Interface (GUI) for the program was also completed to a high standard as shown in section 2.3.1 previously. This aspect did not form part of the original program specification; however, it was agreed that the inclusion of a GUI would improve the user experience.

The fifth and sixth objectives of implementing encryption of the hidden data and creating a program to detect the presence of hidden data within an image have not been achieved due to constraints on both time and programming experience.

* 1. Further Work

If further work was to be carried out, the primary focus would be on using a more robust method of steganography to greater protect the data that is being hidden. Discrete cosine transformation, as discussed previously, would be a good candidate to achieve this.

Further error handling could also be implemented to increase the robustness of the program by ensuring that the user inputs compatible files. This would include ensuring that if text files or jpeg files were input, an error would be returned.

Extra features that could be implemented are objectives five and six of this experiment that were not completed. Encryption would add an extra layer of protection for the hidden data in the event of discovery as the key would be required to unencrypt the information. A steganalysis program would also be beneficial as it would allow the effectiveness of the steganography methods to be tested.

1. References

[1] **Kessler, G**. “Steganography: Hiding Data Within Data”. ‘www.garykessler.net/library/steganography.html’. (accessed on 2/2/17).

[2] **Various Authors**. “Least Significant Bit algorithm for image steganography”. ‘ijact.org/volume3issue4/IJ0340004.pdf’. (accessed on 3/2/17).

[3] **Miller, A**. “Least Significant Bit Embedding: Implementation and Detection”. ‘www.aaronmiller.in/thesis/’. (accessed on 2/2/17).

[4] **Wikipedia**. “Steganography”. ‘en.wikipedia.org/wiki/Steganography’.

(accessed on 28/1/17).

[5] **Wikipedia**. “Flowchart”. ‘en.wikipedia.org/wiki/flowchart’. (accessed on 29/1/17).

[6] **Wikipedia**. “Software Testing”. ‘en.wikipedia.org/wiki/Software testing’.

(accessed on 2/2/17).

[7] **Gupta, Ankit and Garg, Rahul**. “Detecting LSB Steganography in Images”. ‘http://rahuldotgarg.appspot.com/data/steg.pdf’. (accessed on 16/3/17).

1. Appendix

Appendix 1 below shows the full script of the steganography program.

from PIL import Image

import time

import os

class Stego():

def ImageAnalyser(self,carrier):

self.userin = Image.open(carrier)

self.RGB\_userin = self.userin.convert('RGB')

self.width, self.height = self.userin.size

self.new\_array = []

for h in range(self.height):

for w in range((self.width)):

self.r,self.g,self.b = self.RGB\_userin.getpixel((w,h))

self.ip = (self.r,self.g,self.b)

self.new\_array.append(self.ip)

return self.new\_array

def ArrayConverter(self,RGB\_array):

self.bin\_array = []

for item in range(len(RGB\_array)):

self.colour = RGB\_array[item]

self.r,self.g,self.b = self.colour[0],self.colour[1],self.colour[2]

self.r\_bin,self.g\_bin,self.b\_bin = bin(self.r),bin(self.g),bin(self.b)

self.r\_bin = self.r\_bin.replace('0b', '')

self.g\_bin = self.g\_bin.replace('0b', '')

self.b\_bin = self.b\_bin.replace('0b', '')

self.r\_bin = self.r\_bin.zfill(8)

self.g\_bin = self.g\_bin.zfill(8)

self.b\_bin = self.b\_bin.zfill(8)

self.bin\_array.append(self.r\_bin)

self.bin\_array.append(self.g\_bin)

self.bin\_array.append(self.b\_bin)

return self.bin\_array

class Encode(Stego):

def \_\_init\_\_(self,cover,hide,outfile):

self.cover = cover

self.hide = hide

self.outfile = outfile

def ImageEncoder(self,cover,hide,outfile):

self.cover\_size = Image.open(self.cover)

self.cover\_size = self.cover\_size.size

self.hide\_size = Image.open(self.hide)

self.hide\_size = self.hide\_size.size

self.mult\_Wid = self.cover\_size[0] / self.hide\_size[0]

self.mult\_Hei = self.cover\_size[1] / self.hide\_size[1]

self.size = os.path.getsize(self.cover)

self.max\_size = os.path.getsize('cover-2360x2360.bmp')

if self.mult\_Wid == 2 and self.mult\_Hei == 2:

if self.size < self.max\_size:

self.cover\_RGB = self.ImageAnalyser(self.cover)

self.hide\_RGB = self.ImageAnalyser(self.hide)

self.cover\_bin = self.ArrayConverter(self.cover\_RGB)

self.hide\_bin = self.ArrayConverter(self.hide\_RGB)

self.make = Image.new('RGB', self.cover\_size, 'white')

self.binary\_array,self.hide\_array,self.op\_array,self.RGB = [],[],[],[]

self.timer = 0

self.count = 3

for item in range(len(self.hide\_bin)):

self.hide\_bit = self.hide\_bin[item]

self.hide\_bit\_1 = self.hide\_bit[0:2]

self.hide\_bit\_2 = self.hide\_bit[2:4]

self.hide\_bit\_3 = self.hide\_bit[4:6]

self.hide\_bit\_4 = self.hide\_bit[6:8]

self.hide\_array.append(self.hide\_bit\_1)

self.hide\_array.append(self.hide\_bit\_2)

self.hide\_array.append(self.hide\_bit\_3)

self.hide\_array.append(self.hide\_bit\_4)

for val in range(len(self.cover\_bin)):

self.cover\_bit = str(self.cover\_bin[val])

self.cover\_bit\_1 = self.cover\_bit[0:6]

self.binary\_val = self.cover\_bit\_1 + self.hide\_array[self.timer]

self.binary\_array.append(self.binary\_val)

self.timer += 1

for i in range(0,len(self.binary\_array)-2,3):

self.binary\_1 = self.binary\_array[i:self.count]

self.RGB\_1 = int(str(self.binary\_1[0]), 2)

self.RGB\_2 = int(str(self.binary\_1[1]), 2)

self.RGB\_3 = int(str(self.binary\_1[2]), 2)

self.Tup = tuple([self.RGB\_1,self.RGB\_2,self.RGB\_3])

self.op\_array.append(self.Tup)

self.count += 3

self.im = self.make.putdata(self.op\_array)

self.im = self.make.save(outfile)

else:

print("File size too large")

else:

print("Error: Cover image size must be 2x image to hide")

output = ImageEncoder(self,cover,hide,outfile)

class Decode(Stego):

def \_\_init\_\_(self,Object,output):

self.object = Object

self.output = output

self.object\_open = Image.open(Object)

self.object\_size = self.object\_open.size

self.hidden\_size = ((int(self.object\_size[0] / 2)),(int(self.object\_size[1] / 2)))

self.object\_RGB = self.ImageAnalyser(self.object)

self.object\_bin = self.ArrayConverter(self.object\_RGB)

def ImageDecoder(self,Object,output):

self.make = Image.new('RGB', self.hidden\_size, 'white')

self.hidden\_seq = []

self.hidden\_array = []

self.time = 12

for j in range(len(self.object\_bin)):

self.object\_val = self.object\_bin[j]

self.val = self.object\_val[6:8]

self.hidden\_seq.append(self.val)

for k in range(0,len(self.hidden\_seq),12):

self.bit = self.hidden\_seq[k:self.time]

self.bit\_RGB\_1 = self.bit[0]+self.bit[1]+self.bit[2]+self.bit[3]

self.bit\_RGB\_2 = self.bit[4]+self.bit[5]+self.bit[6]+self.bit[7]

self.bit\_RGB\_3 = self.bit[8]+self.bit[9]+self.bit[10]+self.bit[11]

self.bit\_RGB\_1 = int(self.bit\_RGB\_1, 2)

self.bit\_RGB\_2 = int(self.bit\_RGB\_2, 2)

self.bit\_RGB\_3 = int(self.bit\_RGB\_3, 2)

self.tuple = tuple([self.bit\_RGB\_1,self.bit\_RGB\_2,self.bit\_RGB\_3])

self.hidden\_array.append(self.tuple)

self.time +=12

self.im = self.make.putdata(self.hidden\_array)

self.im = self.make.save(output)

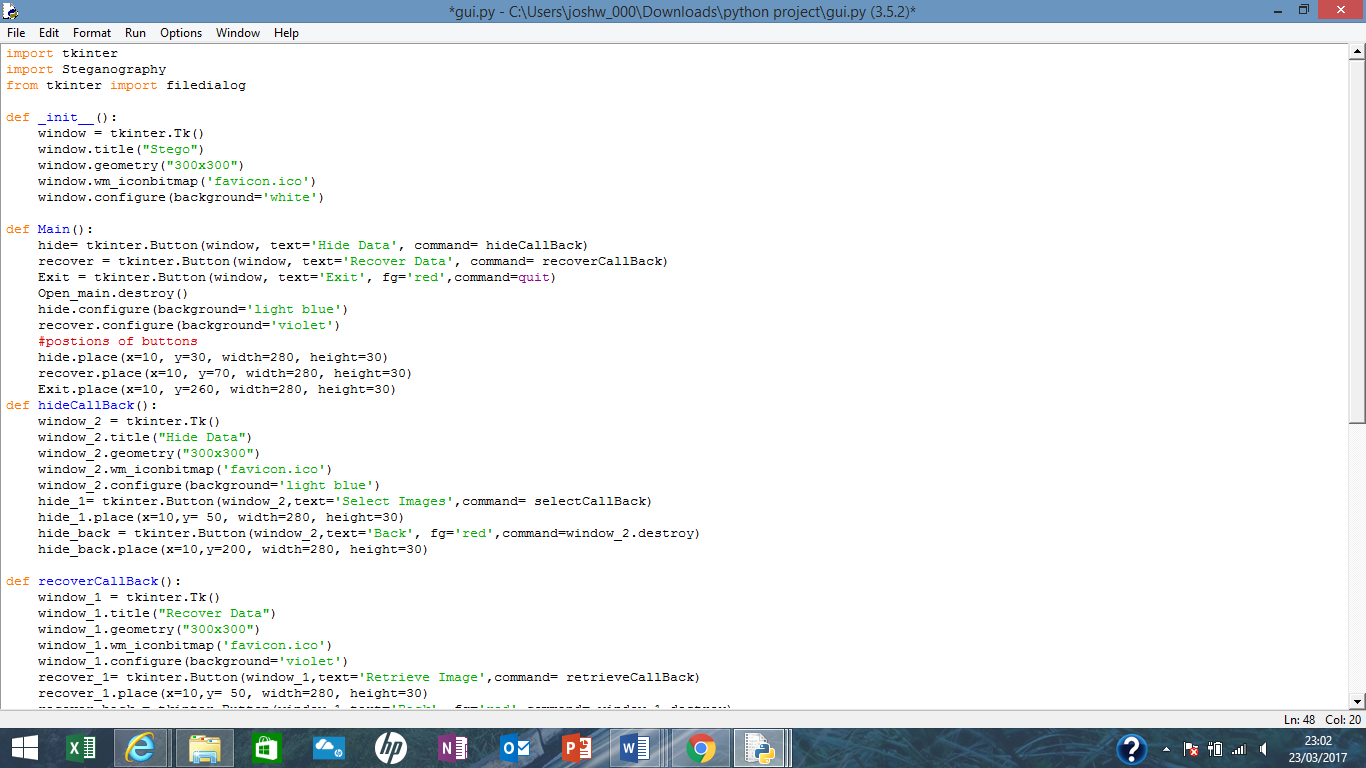
decoded = ImageDecoder(self,Object,output)

Im\_Encoded = Encode('cover.bmp','hide.bmp','object.bmp')

Im\_Decoded = Decode('object.bmp','retrieved.bmp')

Appendix 1. Full steganography program script

Appendix 2 below shows the full script for the GUI section of the program.



Appendix 2. GUI script