**Steganography**

**Serta Melakukan Kompresi dan Dekompresi Pada Gambar**

**Dikerjakan Oleh**

**Arya Anugerah (1203210033)  
Raffel Firmansyah Cahyono (1203210121)  
Muhammad Ezra Firdaus (1203210151)**

**A logo with a red book in the middle

Description automatically generated**

**Program Studi Teknik Informatika**

Penjelasan Code

import base64

import cv2

import time

import imutils

import os

import numpy as np

import matplotlib

from matplotlib.animation import FuncAnimation

import matplotlib.pyplot as plt

from IPython.display import display

from IPython.display import HTML

%matplotlib inline

* *import base64*: Modul untuk melakukan *encoding* dan *decoding* data menggunakan *Base64*.
* *import cv2:* Modul *OpenCV* untuk pemrosesan gambar dan video.
* *import time*: Modul untuk manipulasi waktu.
* *import imutils*: Modul untuk operasi dasar pada gambar seperti resize, rotate, dan lainnya.
* *import os*: Modul untuk berinteraksi dengan sistem operasi.
* *import numpy as np*: *Numpy* adalah *library* untuk operasi matematika pada array dan matriks.
* *import matplotlib*: Modul untuk visualisasi data, termasuk plot dan animasi.
* *from matplotlib.animation import FuncAnimation*: Kelas dari modul *Matplotlib* untuk membuat animasi.
* *import matplotlib.pyplot as plt*: Sub-modul Matplotlib untuk membuat plot.
* *from IPython.display import display, HTML*: Modul IPython untuk menampilkan *output* dalam *Jupyter Notebook.*

# define configuration params

figureSize = (12,10) # image size for display

displayWidth = (1024+128) # max width of image for display

maxDebugBytes = 10 # max num of bytes to view in debug mode

eotInd = 2730 # '101010101010' - 12 bit End-Of-Text indicator

maxCharRange = 127 # max range of characters as per ASCII table

debugMode = False # flag to view debug prints

visFps = 30 # frames per second for the captured visualization video

visStack = 0 # direction (0-horizontal/1-vertical) for visualization stack

Kode ini berisi definisi beberapa parameter konfigurasi yang akan digunakan dalam proses untuk menampilkan atau memvisualisasikan gambar

*class* Steganography(object):

    @staticmethod

*def* \_\_int\_to\_bin(*rgb*):

        """Convert an integer tuple to a binary (string) tuple.

        :param rgb: An integer tuple (e.g. (220, 110, 96))

        :return: A string tuple (e.g. ("00101010", "11101011", "00010110"))

        """

        r, g, b = *rgb*

        return ('{0*:08b*}'.format(r),

                '{0*:08b*}'.format(g),

                '{0*:08b*}'.format(b))

    @staticmethod

*def* \_\_bin\_to\_int(*rgb*):

        """Convert a binary (string) tuple to an integer tuple.

        :param rgb: A string tuple (e.g. ("00101010", "11101011", "00010110"))

        :return: Return an int tuple (e.g. (220, 110, 96))

        """

        r, g, b = *rgb*

        return (int(r, 2),

                int(g, 2),

                int(b, 2))

    @staticmethod

*def* \_\_hide\_rgb(*rgb1*, *rgb2*):

        """hide two RGB tuples.

        :param rgb1: A string tuple (e.g. ("00101010", "11101011", "00010110"))

        :param rgb2: Another string tuple

        (e.g. ("00101010", "11101011", "00010110"))

        :return: An integer tuple with the two RGB values hidden.

        """

        r1, g1, b1 = *rgb1*

        r2, g2, b2 = *rgb2*

        rgb = (r1[:4] + r2[:4],

               g1[:4] + g2[:4],

               b1[:4] + b2[:4])

        return rgb

    @staticmethod

*def* insert(*imfile*, *txtfile*, *outputfile*, *debug*=False):

        """Hide a text within an image.

        :param imfile: Input Image file

        :param txtfile: Input Text file

        :param outputfile: Output Image file

        :param debug: Flag for debug prints

        :return: True or False depending on Success or Failure of operation

        """

        # check is image path is valid

        if not os.path.exists(*imfile*):

          print("!! Unable to locate image file:",*imfile*)

          return False

        # check is text path is valid

        if not os.path.exists(*txtfile*):

          print("!! Unable to locate text file:",*txtfile*)

          return False

        # Load the image

        image = cv2.imread(*imfile*,cv2.IMREAD\_UNCHANGED)

        image = cv2.cvtColor(image,cv2.COLOR\_BGR2RGB)

        # Clone the original image to create a copy that will contain hidden content

        new\_image = image.copy()

        # check if image contains 3 channels, else quit

        im\_shape = image.shape

        if len(im\_shape) != 3:

          print("!! Unable to insert text as image does not have 3 channels !!")

          return False

        elif im\_shape[2] != 3:

          print("!! Unable to insert text as image does not have 3 channels !!")

          return False

        # Open the text file and read the content

        fp = open(*txtfile*, 'rb')

        text\_bin = fp.read()

        fp.close()

        print("Length of text: {} chars".format(len(text\_bin)))

        # check if image size is sufficient to encode full text, else quit

        required\_bytes = len(text\_bin) \* 3

        if required\_bytes > im\_shape[0] \* im\_shape[1] \* 3:

          print("!! Unable to insert full text as image does not have sufficient size !!")

          return False

        # initialize variables

        idx = 0

        break\_flag = False

        for i in range(im\_shape[0]):

            for j in range(im\_shape[1]):

                rgb1 = Steganography.\_\_int\_to\_bin(image[i,j])

                if idx == len(text\_bin):

                  # if no more text left to insert, add the end of text signature

                  bin\_val = "{0*:012b*}".format(eotInd)

                  break\_flag = True

                  if *debug*:

                    print("Hit end of text at idx:",idx)

                    print("No of bytes used:", idx\*3)

                else:

                  if text\_bin[idx] > maxCharRange:

                    # if character is out of max range, insert a 'dot' character in 12-bit binary format

                    bin\_val = "{0*:012b*}".format(ord('.'))

                  else:

                    # convert character to 12-bit binary format

                    bin\_val = "{0*:012b*}".format(text\_bin[idx])

                # split into 3 channels of 4 bits each

                rgb2 = (bin\_val[:4],bin\_val[4:8],bin\_val[8:12])

                # hide the two pixels and convert it to a integer tuple

                rgb = Steganography.\_\_hide\_rgb(rgb1, rgb2)

                new\_image[i,j] = Steganography.\_\_bin\_to\_int(rgb)

                if *debug*:

                  if idx < maxDebugBytes:

                    # visualize first maxDebugBytes

                    # print(text\_bin[idx])

                    # print(chr(text\_bin[idx]))

                    print("bin\_val[{}]:{}".format(idx,bin\_val))

                idx += 1

                if break\_flag == True:

                  break

            if break\_flag == True:

              break

        plt.figure(*figsize*=figureSize)

        plt.imshow(imutils.resize(image,*width*=displayWidth))

        plt.title('Original Image')

        plt.xticks([]), plt.yticks([])  # Hides the graph ticks and x / y axis

        plt.show()

        plt.figure(*figsize*=figureSize)

        plt.imshow(imutils.resize(new\_image,*width*=displayWidth))

        plt.title('Image with Hidden Text')

        plt.xticks([]), plt.yticks([])  # Hides the graph ticks and x / y axis

        plt.show()

        new\_image = cv2.cvtColor(new\_image, cv2.COLOR\_RGB2BGR)

        if os.path.exists(*outputfile*):

            os.remove(*outputfile*)

        cv2.imwrite(*outputfile*,new\_image)

        return True

    @staticmethod

*def* extract(*imfile*, *txtfile*, *visualize*, *visfile*, *debug*=False):

        """Extract hidden text from an image.

        :param imfile: input image file containing hidden text

        :param txtfile: output text file containing extracted text

        :param visualize: flag to enable/disable visualization

        :param visfile: output file to store visualization video

        :param debug: Flag for debug prints

        :return: True or False depending on Success or Failure of operation.

        """

        # check is image path is valid

        if not os.path.exists(*imfile*):

          print("!! Unable to locate image file:",*imfile*)

          return False

        # Load the image

        image = cv2.imread(*imfile*,cv2.IMREAD\_UNCHANGED)

        image = cv2.cvtColor(image,cv2.COLOR\_BGR2RGB)

        # Store the image size

        im\_shape = image.shape

        # check if image contains 3 channels, else quit

        if len(im\_shape) != 3:

          print("!! Unable to extract text as image does not have 3 channels !!")

          return False

        elif im\_shape[2] != 3:

          print("!! Unable to extract text as image does not have 3 channels !!")

          return False

        # initialize variables

        text\_bin = []

        idx = 0

        break\_flag = False

        # create a canvas for displaying extracted text

        text\_canvas =  np.zeros\_like(image)

        # create a white canvas for displaying image

        white\_canvas =  255 \* np.ones\_like(image)

        if *visualize*.lower() == 'true':

            vis\_width = displayWidth

            vis\_height = int(0.5\*displayWidth\*im\_shape[0]/im\_shape[1])

            visframes = []

            capture\_vis = True

        else:

            capture\_vis = False

        print("Begin processing image file... this may take a few minutes")

        start\_time = time.time()

        for i in range(im\_shape[0]):

            for j in range(im\_shape[1]):

                # Get the RGB (as a string tuple) from the current pixel

                r, g, b = Steganography.\_\_int\_to\_bin(image[i,j])

                # Extract the last 4 bits (corresponding to the hidden text)

                bin\_val = r[4:] + g[4:] + b[4:]

                if *debug*:

                # view first maxDebugBytes bytes

                  if idx < maxDebugBytes:

                    print("bin\_val[{}]:{}".format(idx,bin\_val))

                # Convert it to an integer

                int\_val = int(bin\_val,2)

                if int\_val == eotInd:

                  # Check if End-Of-Text indicator is found

                  if *debug*:

                    print("Found end of text signature at index:", idx)

                  break\_flag = True

                elif int\_val <= maxCharRange:

                  # only take characters within max range and append to output list

                  text\_bin.append(int\_val)

                  if capture\_vis == True:

                    # visualize the extraction process

                    try:

                      xpos = (j\*30) % im\_shape[1]

                      ypos = i+30 + (j\*30//im\_shape[1]) \* 30

                      cv2.putText(text\_canvas, chr(text\_bin[idx]), (xpos,ypos),cv2.FONT\_HERSHEY\_SIMPLEX,1.0,(0, 255, 0),4)

                      # create a canvas for displaying original image while extraction with some blurring effect

                      img\_canvas = image.copy()

                      img\_canvas = cv2.addWeighted(img\_canvas, 0.30, white\_canvas, 0.70, 0)

                      # show a small moving box to indicate pixel processing

                      rectx = (j\*30) % im\_shape[1]

                      recty = (i + (j\*30//im\_shape[1]) \* 30) % im\_shape[0]

                      rectw = 30

                      recth = 30

                      cv2.rectangle(img\_canvas, (rectx, recty), (rectx+rectw, recty+recth), (0,0,255), 5)

                      if idx%20 < 10:

                        # Add a blinking text effect

                        cv2.putText(img\_canvas, 'Scanning Image', (int(0.6\* im\_shape[1]//2),im\_shape[0]//2),cv2.FONT\_HERSHEY\_SIMPLEX,3.0,(255, 0, 0), 10)

                      if visStack == 0:

                        # stack\_img = np.hstack((img\_canvas,text\_canvas))

                        stack\_img = np.hstack((imutils.resize(img\_canvas,*width*=displayWidth//2),imutils.resize(text\_canvas,*width*=displayWidth//2)))

                      else:

                        # stack\_img = np.vstack((img\_canvas,text\_canvas))

                        stack\_img = np.vstack((imutils.resize(img\_canvas,*width*=displayWidth//2),imutils.resize(text\_canvas,*width*=displayWidth//2)))

                      stack\_img = cv2.cvtColor(stack\_img, cv2.COLOR\_RGB2BGR)

                      visframes.append(stack\_img)

                      if ypos >= im\_shape[0]:

                        capture\_vis = False

                    except Exception as err:

                        print("!! Visualization error:", err)

*visualize* = 'false'

                        capture\_vis = False

                  idx += 1

                if break\_flag == True:

                  break

            if break\_flag == True:

              break

        end\_time = time.time()

        print("Finished processing image file... time taken: {*:0.02f*}s".format(end\_time-start\_time))

        if *visualize*.lower() == 'true':

            print('Begin saving visualization file... this may take a few minutes')

            start\_time = time.time()

            delay = 1000 / visFps

            num\_frames = len(visframes)

            fig = plt.figure(*figsize*=(vis\_width/100, vis\_height/100), *tight\_layout*=True)

            im = plt.imshow(visframes[0])

            plt.axis('off')

            plt.grid(True)

*def* update\_vis(*i*):

              """Updates the visualization"""

              im.set\_data(visframes[*i*])

            anim = FuncAnimation(

                    fig,

                    update\_vis,

*frames*=np.arange(1, num\_frames),

*interval*=delay,

*blit*=False)

            if os.path.exists(*visfile*):

                os.remove(*visfile*)

            if *visfile*[-3:] == 'gif':

              anim.save(*visfile*, *dpi*=100, *fps*=visFps)

            elif *visfile*[-3:] == 'gif':

              anim.save(*visfile*, *writer*='imagemagick', *fps*=visFps, *dpi*=100)

            plt.close()

            end\_time = time.time()

            print("Finished saving visualization file... time taken: {*:0.02f*}s".format(end\_time-start\_time))

        if *debug*:

          # view first maxDebugBytes

          print("view first {} bytes".format(maxDebugBytes))

          print(text\_bin[:maxDebugBytes])

          print(bytes(text\_bin[:maxDebugBytes]))

          print("maxval:",max(text\_bin))

          print("minval:",min(text\_bin))

        plt.figure(*figsize*=figureSize)

        plt.imshow(imutils.resize(image,*width*=displayWidth))

        plt.title('Image with Hidden Text')

        plt.xticks([]), plt.yticks([])  # Hides the graph ticks and x / y axis

        plt.show()

        try:

          plt.figure(*figsize*=figureSize)

          plt.imshow(imutils.resize(text\_canvas,*width*=displayWidth))

          plt.title('Extracted Text')

          plt.xticks([]), plt.yticks([])  # Hides the graph ticks and x / y axis

          plt.show()

          text\_data = bytes(text\_bin).decode()

          print("Extracted Text:")

          print("===================================")

          print(text\_data)

          print("===================================")

        except Exception as err:

          print("!! Decode Error:",err)

          return False

        if os.path.exists(*txtfile*):

            os.remove(*txtfile*)

        fp = open(*txtfile*,'wb')

        fp.write(bytes(text\_bin))

        fp.close()

        return True

kode ini berfungsi sebagai steganografi, yaitu metode yang dilakukan untuk menyembunyikan teks dalam gambar tanpa merusak visual gambar

kode tersebut memiliki 2 metode dalam melakukan steganografi

1. *insert* : Menyisipkan teks dari sebuah file teks ke dalam gambar tanpa mengubah tampilan visualnya secara signifikan.

Proses : melakukan konversi teks ke biner – menyembunyikan setiap bit teks ke dalam setiap nilai pixel dalam gambar

1. *extract* : Mengekstrak teks yang disembunyikan dari gambar tanpa merusak atau mengubah gambar tersebut.

Proses : mengekstrak setiap bit teks di dalam gambar – lalu mengkonversi setiap nilai bit itu menjadi teks

*def* insert\_text(*img\_file*, *text\_file*, *output\_file*):

    start\_time = time.time()

    ret\_val = Steganography.insert(*img\_file*,*text\_file*,*output\_file*,debugMode)

    end\_time = time.time()

    if not ret\_val:

      print("!! Insert Failure !!")

    else:

      print("Total time taken for Insert operation: {*:0.02f*}s".format(end\_time-start\_time))

kode ini berfungsi untuk menyisipkan teks dari sebuah file teks ke dalam sebuah gambar tanpa mengubah tampilan visual gambar

# sample inputs with local files, uncomment to run

img\_file = 'MomoiMidori.jpg' # input image file

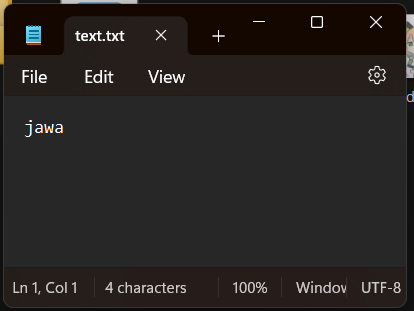
text\_file = 'text.txt' # input text file

output\_file = 'output.png' # output image file containing hidden text

kode ini berfungsi untuk menyimpan nama gambar dan teks yang akan di gunkan serta melakukan *output* untuk mencetak hasil gambar

insert\_text(img\_file,text\_file,output\_file)

kode ini bertujuan untuk menyisipkan teks

isi text pada file text.txt dan gambar  


A screenshot of a computer

Description automatically generated

hasil tampilan :   
A screenshot of a cartoon

Description automatically generated

import zlib

*def* compress\_image(*input\_file*, *output\_file*):

    with open(*input\_file*, 'rb') as f\_in:

        img\_data = f\_in.read()

        compressed\_data = zlib.compress(img\_data, *level*=zlib.Z\_BEST\_COMPRESSION)

    with open(*output\_file*, 'wb') as f\_out:

        f\_out.write(compressed\_data)

*def* decompress\_image(*input\_file*, *output\_file*):

    with open(*input\_file*, 'rb') as f\_in:

        compressed\_data = f\_in.read()

        img\_data = zlib.decompress(compressed\_data)

    with open(*output\_file*, 'wb') as f\_out:

        f\_out.write(img\_data)

# Contoh penggunaan

compressed\_file = 'compressed\_output.png'

compress\_image('output.png', compressed\_file)

decompressed\_file = 'decompressed\_output.png'

decompress\_image(compressed\_file, decompressed\_file)

kode ini bertujuan untuk melakukan kompresi dan dekompresi ukuran gambar

sebagai contoh gambar yang akan di kompresi Bernama *“output.png”* setelah di kompresi akan berubah nama menjadi *“compressed\_output.png”.*

lalu gambar yang di telah di kompresi akan di dekompresi ulang dan file tersebut berubah nama menjadi *“decompressed\_output.png”.*

*def* extract\_text(*imfile*, *txtfile*, *visualize*, *visfile*):

    start\_time = time.time()

    ret\_val = Steganography.extract(*imfile*,*txtfile*,*visualize*,*visfile*,debugMode)

    end\_time = time.time()

    if not ret\_val:

      print("!! Extraction Failure !!")

    else:

      print("Total time taken for extraction: {*:0.02f*}s".format(end\_time-start\_time))

kode ini bertujuan untuk mengeluarkan teks yang disisipkan pada gambar (*extract text*) melalui metode steganografi.

Jika gambar gagal di ekstrak maka akan muncul pesan *"!! Extraction Failure !!"*, jika gambar berhasil di ekstrak makan akan muncul pesan "Total time taken for extraction: (sesuai waktu ekstraksi)”

new\_img\_file = 'decompressed\_output.png' # input image file for text extraction

output\_txt\_file = 'text\_output.txt' # output file to store extracted text

visualize = 'true'  # flag to enable/disable visualization

visfile = 'visualize.gif' # file to store visualization video

kode ini menyimpan file dengan nama baru yang setelah teks telah di ekstrak dari gambar dengan nama “*decompressed\_output.png”* dan *“text\_output.png”.*

import matplotlib

print(matplotlib.\_\_version\_\_)

extract\_text(new\_img\_file,output\_txt\_file,visualize,visfile)

kode tersebut di gunakan untuk mengekstrak teks yang disisipkan di dalam gambar.

Hasilnya :  
A screenshot of a cartoon

Description automatically generated

Hasil output seluruh file :

gambar ukuran file asli  
A screenshot of a cartoon

Description automatically generated

ukuran file setelah disisipkan text ke dalam gambar  
A screenshot of a cartoon

Description automatically generated

ukuran gambar setelah mengalami kompresi  
A screenshot of a computer

Description automatically generated

ukuran gambar Kembali semula saat mengalami dekompresi  
A screenshot of a phone

Description automatically generated

untuk ukuran file text setelah dan sesudah mengalamin kompresi dan dekompresi saat disisipkan ke dalam gambar tidak mengalami perubahan  
Screens screenshot of a computer

Description automatically generated