



Digital Image Processing-1

(UE20EC317)

Project

Group: 12

S L Nithyananda Rao - PES1UG20EC357

Vaishnav Mohan - PES1UG20EC216

Srinidhi B S - PES1UG20EC201

LSB based video steganography with added feature of object detection

Introduction

We have chosen the military field as our subject matter, and our project will focus on that aspect. Video steganography can be used to send secret messages to soldiers over long distances without interference from enemies, whereas object detection is a viable tool that can be used for segmenting data and identifying possible enemy aircrafts using accurately trained datasets and could be of significant assistance to the military.

Object Detection

Theory

Identifying objects in an image is a common task for the human brain, but not so simple for a machine. Object detection is a computer vision task that identifies and localises objects in photos, and several algorithms have emerged in recent years to address the problem. YOLO is one of the most popular real-time object detection algorithms to date (You Only Look Once)

Algorithm

YOLO is a unique convolutional neural network (CNN) that recognises objects in real-time with high accuracy. This method processes the entire image using a single neural network, then divides it into sections and predicts bounding boxes and probabilities for each component. These bounding boxes are weighted by the estimated probability. The approach "just looks once" at the image in the sense that it provides predictions after only one forward propagation run through the neural network. It then provides detected items after non-max suppression (which ensures that the object detection algorithm only identifies each object once).

Dataset

<https://drive.google.com/file/d/1B-bj2IUv-QEKoArRre-sRILzFFyfxJeT/view?usp=sharing>

Code

```
# -*- coding: utf-8 -*-
```

```
"""YOLOv5 Tutorial
```

Automatically generated by Colaboratory.

Original file is located at

<https://colab.research.google.com/github/ultralytics/yolov5/blob/master/tutorial.ipynb>

<div align="center">

<im
g src="https://colab.research.google.com/assets/colab-badge.svg" alt="Open In Colab">

This YOLOv5 🚀 notebook by Ultralytics presents simple train, validate and predict examples

to help start your AI adventure.
See GitHub for community support or contact us for professional support.

</div>

Setup

Clone GitHub [repository](https://github.com/ultralytics/yolov5), install [dependencies](https://github.com/ultralytics/yolov5/blob/master/requirements.txt) and check PyTorch and GPU.

```
"""
```

```
# Commented out IPython magic to ensure Python compatibility.
```

```
!git clone https://github.com/ultralytics/yolov5 # clone
```

```
# %cd yolov5
```

```
# %pip install -qr requirements.txt # install
```

```
import torch
```

```
import utils
```

```
display = utils.notebook_init() # checks
```

```
!unzip -q ../train_data.zip -d ../
```

```
"""# 1. Detect
```

`detect.py` runs YOLOv5 inference on a variety of sources, downloading models automatically from the [latest YOLOv5 release](https://github.com/ultralytics/yolov5/releases), and saving results to `runs/detect`. Example inference sources are:

```
```shell
```

```
python detect.py --source 0 # webcam
```

img.jpg # image

```
vid.mp4 # video
```

screen # screenshot

path/ # directory

```
'path/*.jpg' # glob
```

'https://youtu.be/Zgi9g1ksQHc' # YouTube

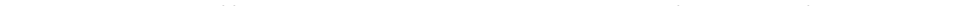
```
'rtsp://example.com/media.mp4' # RTSP, RTMP, HTTP stream
```

...

|||||

```
!python detect.py --weights /content/yolov5/runs/train/exp2/weights/last.pt --img 640 --conf 0.30 --source ../topgunmav_og.mp4
```

```
display.Image(filename='runs/detect/exp/zidane.jpg', width=600)
```

[illegible]

commit	message	author
26833433	add test for the new feature	david
127574988	fix a bug in the new feature	david
6a558aa1	add a new feature	david
d268-44b9-bf6b-62d4c605cc72	add a new feature	david

## # 2. Validate

Validate a model's accuracy on the [COCO](https://cocodataset.org/#home) dataset's `val` or `test` splits. Models are downloaded automatically from the [latest YOLOv5 release](https://github.com/ultralytics/yolov5/releases). To show results by class use the `--verbose` flag.

|||||

```
Download COCO val
```

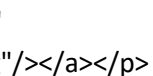
```
torch.hub.download_url_to_file('https://ultralytics.com/assets/coco2017val.zip', 'tmp.zip') #
download (780M - 5000 images)
```

```
!unzip -q tmp.zip -d ../datasets && rm tmp.zip # unzip
```

```
Validate YOLOv5s on COCO val
```

```
!python val.py --weights yolov5s.pt --data coco.yaml --img 640 --half
```

```
"""# 3. Train
```

<https://roboflow.com/?ref=ultralytics>

Close the active learning loop by sampling images from your inference conditions with the `roboflow` pip package

Train a YOLOv5s model on the [COCO128](https://www.kaggle.com/ultralytics/coco128) dataset with `--data coco128.yaml`, starting from pretrained `--weights yolov5s.pt`, or from randomly initialized `--weights "" --cfg yolov5s.yaml`.

- **Pretrained [Models](https://github.com/ultralytics/yolov5/tree/master/models)** are downloaded

automatically from the [latest YOLOv5 release](https://github.com/ultralytics/yolov5/releases)

- **[Datasets](https://github.com/ultralytics/yolov5/tree/master/data)** available for autodownload include: [COCO](https://github.com/ultralytics/yolov5/blob/master/data/coco.yaml), [COCO128](https://github.com/ultralytics/yolov5/blob/master/data/coco128.yaml), [VOC](https://github.com/ultralytics/yolov5/blob/master/data/VOC.yaml), [Argoverse](https://github.com/ultralytics/yolov5/blob/master/data/Argoverse.yaml), [VisDrone](https://github.com/ultralytics/yolov5/blob/master/data/VisDrone.yaml), [GlobalWheat](https://github.com/ultralytics/yolov5/blob/master/data/GlobalWheat2020.yaml), [xView](https://github.com/ultralytics/yolov5/blob/master/data/xView.yaml), [Objects365](https://github.com/ultralytics/yolov5/blob/master/data/Objects365.yaml), [SKU-110K](https://github.com/ultralytics/yolov5/blob/master/data/SKU-110K.yaml).

- **Training Results** are saved to `runs/train/` with incrementing run directories, i.e. `runs/train/exp2`, `runs/train/exp3` etc.

A **Mosaic Dataloader** is used for training which combines 4 images into 1 mosaic.

**## Train on Custom Data with Roboflow** 🌟 NEW

[Roboflow](https://roboflow.com/?ref=ultralytics) enables you to easily **organize, label, and prepare** a high quality dataset with your own custom data. Roboflow also makes it easy to establish an active learning pipeline, collaborate with your team on dataset improvement, and integrate directly into your model building workflow with the `roboflow` pip package.

- Custom Training Example: [https://blog.roboflow.com/how-to-train-yolov5-on-a-custom-dataset/](https://blog.roboflow.com/how-to-train-yolov5-on-a-custom-dataset/?ref=ultralytics)

- Custom Training Notebook: [![Open In Colab](https://colab.research.google.com/assets/colab-badge.svg)](https://colab.research.google.com/github/roboflow-ai/yolov5-custom-training-tutorial/blob/main/yolov5-custom-training.ipynb)

<br>

<p align=""><a href="https://roboflow.com/?ref=ultralytics"></a></p>Label images lightning fast (including with model-assisted labeling)

""""

# Commented out IPython magic to ensure Python compatibility.

#@title Select YOLOv5 🚀 logger {run: 'auto'}

logger = 'TensorBoard' #@param ['TensorBoard', 'Comet', 'ClearML']

if logger == 'TensorBoard':

# %load\_ext tensorboard

# %tensorboard --logdir runs/train

elif logger == 'Comet':

```
%pip install -q comet_ml

import comet_ml; comet_ml.init()

elif logger == 'ClearML':

%pip install -q clearml

import clearml; clearml.browser_login()

Train YOLOv5s on COCO128 for 3 epochs

!python train.py --img 640 --batch 16 --epochs 70 --data custom_data.yaml --weights yolov5s.pt --
cache
```

```
"""# 4. Visualize
```

**## Comet Logging and Visualization 🌟 NEW**

[Comet](<https://bit.ly/yolov5-readme-comet>) is now fully integrated with YOLOv5. Track and visualize model metrics in real time, save your hyperparameters, datasets, and model checkpoints, and visualize your model predictions with [Comet Custom Panels](<https://bit.ly/yolov5-colab-comet-panels>)! Comet makes sure you never lose track of your work and makes it easy to share results and collaborate across teams of all sizes!

Getting started is easy:

```
``shell
```

```
pip install comet_ml # 1. install
```

```
export COMET_API_KEY=<Your API Key> # 2. paste API key
```

```
python train.py --img 640 --epochs 3 --data coco128.yaml --weights yolov5s.pt # 3. train
```

```
```
```

To learn more about all of the supported Comet features for this integration, check out the [Comet Tutorial](<https://github.com/ultralytics/yolov5/tree/master/utils/loggers/comet>). If you'd like to learn more about Comet, head over to our [documentation](<https://bit.ly/yolov5-colab-comet-docs>). Get started by trying out the Comet Colab Notebook:

[![Open In Colab]](https://colab.research.google.com/assets/colab-badge.svg)](https://colab.research.google.com/drive/1RGOWOQyxIDlo5Km8GogJpIEJlg_5lyYO?usp=s haring)

ClearML Logging and Automation 🌟 NEW

[ClearML](https://cutt.ly/yolov5-notebook-clearml) is completely integrated into YOLOv5 to track your experimentation, manage dataset versions and even remotely execute training runs. To enable ClearML (check cells above):

- `pip install clearml`

- run `clearml-init` to connect to a ClearML server (**deploy your own [open-source server](https://github.com/allegroai/clearml-server)**), or use our [free hosted server](https://cutt.ly/yolov5-notebook-clearml))

You'll get all the great expected features from an experiment manager: live updates, model upload, experiment comparison etc. but ClearML also tracks uncommitted changes and installed packages for example. Thanks to that ClearML Tasks (which is what we call experiments) are also reproducible on different machines! With only 1 extra line, we can schedule a YOLOv5 training task on a queue to be executed by any number of ClearML Agents (workers).

You can use ClearML Data to version your dataset and then pass it to YOLOv5 simply using its unique ID. This will help you keep track of your data without adding extra hassle. Explore the [ClearML Tutorial](https://github.com/ultralytics/yolov5/tree/master/utils/loggers/clearml) for details!

Local Logging

Training results are automatically logged with [Tensorboard](https://www.tensorflow.org/tensorboard) and [CSV](https://github.com/ultralytics/yolov5/pull/4148) loggers to `runs/train`, with a new experiment directory created for each new training as `runs/train/exp2`, `runs/train/exp3`, etc.

This directory contains train and val statistics, mosaics, labels, predictions and augmented mosaics, as well as metrics and charts including precision-recall (PR) curves and confusion matrices.

Environments

YOLOv5 may be run in any of the following up-to-date verified environments (with all dependencies including

[CUDA](https://developer.nvidia.com/cuda)/[CUDNN](https://developer.nvidia.com/cudnn), [Python](https://www.python.org/) and [PyTorch](https://pytorch.org/) preinstalled):

- **Notebooks** with free GPU:
- **Google Cloud** Deep Learning VM. See [GCP Quickstart Guide](https://github.com/ultralytics/yolov5/wiki/GCP-Quickstart)
- **Amazon** Deep Learning AMI. See [AWS Quickstart Guide](https://github.com/ultralytics/yolov5/wiki/AWS-Quickstart)
- **Docker Image**. See [Docker Quickstart Guide](https://github.com/ultralytics/yolov5/wiki/Docker-Quickstart)

Status

![YOLOv5 CI](https://github.com/ultralytics/yolov5/actions/workflows/ci-testing.yml/badge.svg)

If this badge is green, all [YOLOv5 GitHub Actions](https://github.com/ultralytics/yolov5/actions) Continuous Integration (CI) tests are currently passing. CI tests verify correct operation of YOLOv5 training ([train.py](https://github.com/ultralytics/yolov5/blob/master/train.py)), testing ([val.py](https://github.com/ultralytics/yolov5/blob/master/val.py)), inference ([detect.py](https://github.com/ultralytics/yolov5/blob/master/detect.py)) and export ([export.py](https://github.com/ultralytics/yolov5/blob/master/export.py)) on macOS, Windows, and Ubuntu every 24 hours and on every commit.

Appendix

Additional content below.

.....

YOLOv5 PyTorch HUB Inference (DetectionModels only)

```
import torch
```

```
model = torch.hub.load('ultralytics/yolov5', 'yolov5s') # yolov5n - yolov5x6 or custom
```

```
im = 'https://ultralytics.com/images/zidane.jpg' # file, Path, PIL.Image, OpenCV, nparray, list
```

```
results = model(im) # inference
```

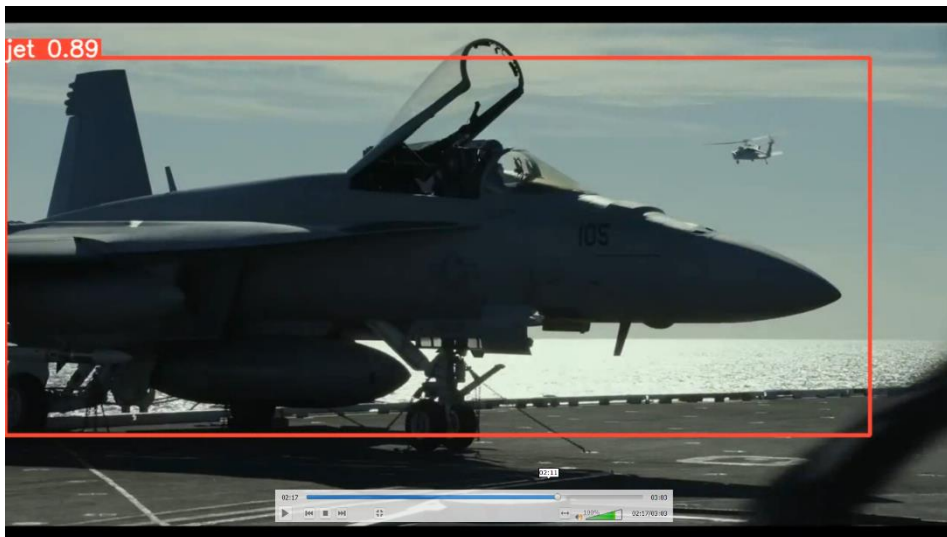
```
results.print() # or .show(), .save(), .crop(), .pandas(), etc.
```

Output

1. Fighter pilot detection



2. Jet detection



3. Combined detection with discretion



LSB-Based Steganography

The term "steganography" describes the process of hiding data or a file inside of a digital image, video, or audio file. A person will not be able to tell that there is any hidden information if they view the object that contains the information. As a result, the person won't attempt to decode the data. Text, image, and audio/video steganography are the three main categories of steganography. An extremely effective algorithm called LSB is employed to embed a text message in a cover file by making variations to the Least significant bit of certain pixels.

Video Splitting

In this project, we will be performing image steganography. The audio-visual input is split into two parts, an audio(.wav) file and a folder containing all the frames in the video extracted at 30 frames per second. Each frame is stored as an image.

Encryption

The steganographic encryption process involves converting the image into binary format followed by replication. The copy of the image is encoded with a message, for a given range of frames, with each iteration extracting 3 bits for modification at a time. The LSB encryption type can be chosen from a range of 2 to 8 bits. The modified frames are saved as new frames to be combined with audio to produce an encrypted video.

Decryption

Since the holistic theme of the project is aimed at military and security functionality, the steganography process here is aimed at message transmission with decryption key details being accessible to the sending and receiving parties only. The Decryption process involves stripping the frame images in the given range to 3 channels namely, red, green, and blue.

Detection and Decoding

Steganography detection system uses the average values of these three channels. The average of blue channel is the closest to the total average (experimentally found out). Therefore, if it is a value between 0.5 and 0.52, the image is likely to contain steganographical information.

When steganography has been successfully detected, the binary pattern encoded in the pixels is converted to characters and appended to a data variable containing the message.

Methodologies/Functions used

Encoding and decoding is performed by the Codec function in python while the Audio-visual handling is done using ffmpeg tools integrated into python 3.7 and above.

Code

analysis.py

```
#import matplotlib.pyplot as plt
```

```
import numpy
```

```
from PIL import Image
```

```
# Global Variables
```

```
detection = 0 # Detection Flag
```

```
steg_frames = [] # To Store Frames with Steganography Present
```

```
# Image Steganography Detection Function
```

```

def analyse(file):

    block_size = 100

    img = Image.open(file)

    (width, height) = img.size # Get Image Resolution

    print(">> Frame Resolution: %d x %d Pixels" % (width, height))

    converted_data = img.convert("RGBA").getdata() # Obtain Red Green Blue Alpha Channel
Information

    lsb_r = []      # Red Channel LSB Holder

    lsb_g = []      # Green Channel LSB Holder

    lsb_b = []      # Blue Channel LSB Holder

    for height_for in range(height):

        for width_for in range(width):

            (r, g, b, a) = converted_data.getpixel((width_for, height_for)) # Obtain Pixel
Data from Each Channel

            lsb_r.append(r & 1)

            lsb_g.append(g & 1)

            lsb_b.append(b & 1)

    lsb_r_avg = [] # Average Red Channel LSB Holder

    lsb_g_avg = [] # Average Green Channel LSB Holder

    lsb_b_avg = [] # Average Blue Channel LSB Holder

    for i in range(0, len(lsb_r), block_size): # Calculating Averages

        lsb_r_avg.append(numpy.mean(lsb_r[i:i + block_size]))

        lsb_g_avg.append(numpy.mean(lsb_g[i:i + block_size]))

        lsb_b_avg.append(numpy.mean(lsb_b[i:i + block_size]))

    global detection

    for i in range(0, 3): # To Check If LSB Steganography Exists

```

if lsb_b_avg[i] >= 0.50 and lsb_b_avg[i] < 0.52: # Range of 0.5 to 0.52 chosen due to the results obtained from the test cases that shows increased accuracy // Blue Channel showed consistency in the test cases, hence it was chosen

detection = 1

if detection == 1: # Output

print(">> Steganography Detected!\n")

else:

print(">> No Steganography Detected!\n")

<<<<< For Testing/Documentation >>>>>

#numBlocks = len(lsb_r_avg)

#blocks = [i for i in range(0, numBlocks)]

#plt.axis([0, len(lsb_r_avg), 0, 1]) # Graph Creation

#plt.ylabel("Average LSB Per Block")

#plt.xlabel("Block Number")

#plt.plot(blocks, lsb_r_avg, "ro") # Plot Values for Red

#plt.plot(blocks, lsb_g_avg, "go") # Plot Values for Green

#plt.plot(blocks, lsb_b_avg, "bo") # Plot Values for Blue

#plt.show()

Multi-Frame Iterations Function

def iterations(num_frames, file_type, file_location):

global steg_frames, detection

current = 0 # Initial Frame

while current != num_frames:

try:

print("\nAnalysing Frame %d..." % current)

analyse(str(file_location) + "\\\" + str(current) + "." + file_type)


```

        if detection == 1:

            steg_frames.append(current)

            detection = 0

            current += 1

    except KeyboardInterrupt:

        print("\nUser canceled analysis, exiting...")

        break

    results()

# Final Output Function

def results():

    if not steg_frames:

        print(">> No Steganography Detected in the Frames Provided!")

    else:

        print(">> Steganography Detected in Frame:")

        print(">> " + str(steg_frames)[1:-1])

# Runtime

try:

    file_location = input("Directory of Frame(s): ")

except KeyboardInterrupt:

    print("\nUser canceled analysis, exiting...")

    quit()

while True:

    try:

        num_frames = int(input("Number of Frame(s): "))

        break

    except ValueError:

```

```

        print("\nInteger Expected, Please Try Again...")

    except KeyboardInterrupt:

        print("\nUser canceled analysis, exiting...")

        quit()

try:

    iterations(num_frames, "png", file_location)

except FileNotFoundError:

    print("\nFrames not found, are you on the right folder? Are the frames in numerical order starting from 0?\n Exiting....")

except KeyboardInterrupt:

    quit()

```

aviGui.py

```

import tkinter as tk

import sys

import os

import cv2


from PIL import Image

from moviepy.editor import *


def wrapper():

    """Wrapper function to call subfunction"""

    vf, base_filename = get_video_filename_base()

    option = var.get()

    if option == 1:

        video_object = VideoFileClip(vf)

        get_video(base_filename, video_object)

    elif option == 2:

        video_object = VideoFileClip(vf)

```

```

        get_audio(base_filename, video_object)

elif option == 3:

    video_file = video_filename.get()

    audio_file = audio_filename.get()

    og_file = og_filename.get()

    combine_audio_video(video_file, audio_file, og_file)

else:

    video_object = VideoFileClip(vf)

    get_frames(video_object, base_filename)


def get_video_filename_base():

    """Returns filename and base filename"""

    vf = video_filename.get()

    return vf, os.path.splitext(os.path.basename(vf))[0]


def get_audio_filename_base():

    """Returns audio filename"""

    return audio_filename.get()


def get_video(base_filename, video_object):

    """Returns the video track only of a video clip"""

    video_object.write_videofile(filename=f'output\\{base_filename}_video_only.mp4', audio=False)


def get_audio(base_filename, video_object):

    """Returns the audio track only of a video clip"""

    video_object.audio.write_audiofile(filename=f'output\\{base_filename}_audio.wav')

    video_object.audio.write_audiofile(filename=f'output\\{base_filename}_audio.mp3')

```

```

def combine_audio_video(video_path, audio_path, og_path):

    """Combines an audio and a video object together"""

    capture = cv2.VideoCapture(og_path) # Stores OG Video into a Capture Window

    fps = capture.get(cv2.CAP_PROP_FPS) # Extracts FPS of OG Video


    video_path_real = video_path + "\\%d.png" # To Get All Frames in Folder


    os.system("ffmpeg-4.3.1-2020-10-01-full_build\\bin\\ffmpeg -framerate %s -i \"%s\" -codec copy
output\\combined_video_only.mkv" % (str(int(fps)), video_path_real)) # Combining the Frames into
a Video

    os.system("ffmpeg-4.3.1-2020-10-01-full_build\\bin\\ffmpeg -i output\\combined_video_only.mkv
-i \"%s\" -codec copy output\\combined_video_audio.mkv" % audio_path) # Combining the Frames
and Audio into a Video


    print("Combining Complete!")


def get_frames(video_object, base_filename):

    """Returns all frames in the video object"""

    directory = "output\\" + base_filename + '_frames\\'

    if not os.path.isdir(directory):

        os.makedirs(directory)

    for index, frame in enumerate(video_object.iter_frames()):

        img = Image.fromarray(frame, 'RGB')

        img.save(f'{directory}{index}.png')


def sel():

    """Helper function to handle radio button selection"""

    if var.get() == 3:

        audio_filename_widget.config(state=tk.NORMAL)

        og_filename_widget.config(state=tk.NORMAL)

```

else:

audio_filename_widget.config(state=tk.DISABLED)

og_filename_widget.config(state=tk.DISABLED)

window_main = tk.Tk()

window_main.title('AVI-Extractor')

window_main.geometry("400x200")

window_main.grid_columnconfigure((0,1), weight=1)

video_filename = tk.StringVar()

video_label = tk.Label(window_main, text="Video File/Frame Path: ")

video_label.grid(row=1, column=0)

video_filename_widget = tk.Entry(window_main, textvariable=video_filename)

video_filename_widget.grid(row=1, column=1)

audio_filename = tk.StringVar()

audio_label = tk.Label(window_main, text="Audio File: ")

audio_label.grid(row=2, column=0)

audio_filename_widget = tk.Entry(window_main, textvariable=audio_filename, state=tk.DISABLED)

audio_filename_widget.grid(row=2, column=1)

og_filename = tk.StringVar()

og_label = tk.Label(window_main, text="Original Video File: ")

```
og_label.grid(row=3, column=0)
```

```
og_filename_widget = tk.Entry(window_main, textvariable=og_filename, state=tk.DISABLED)
```

```
og_filename_widget.grid(row=3, column=1)
```

```
var = tk.IntVar()
```

```
R1 = tk.Radiobutton(window_main, text="Extract Video Without Audio", variable=var, value=1,  
command=sel)
```

```
R1.grid(row=4, columnspan=2, sticky=tk.W)
```

```
R2 = tk.Radiobutton(window_main, text="Extract Audio Without Video", variable=var, value=2,  
command=sel)
```

```
R2.grid(row=5, columnspan=2, sticky=tk.W)
```

```
R3 = tk.Radiobutton(window_main, text="Combine Audio and Video", variable=var, value=3,  
command=sel)
```

```
R3.grid(row=6, columnspan=2, sticky=tk.W)
```

```
R4 = tk.Radiobutton(window_main, text="Get Frames of Video", variable=var, value=4,  
command=sel)
```

```
R4.grid(row=7, columnspan=2, sticky=tk.W)
```

```
run = tk.Button(window_main, text="Run", command=wrapper)
```

```
run.grid(row=7, columnspan=2, sticky=tk.N)
```

```
window_main.mainloop()
```

Decoder.py

```
import re
```

```
from PIL import Image
```

```
# Global Variable
```

```
global frame_location
```

```
# Decode the data in the image
```

```
def decode(number):
```

```
    data = "
```

```
    numbering = str(number)
```

```
    decoder_numbering = frame_location + "\\\" + numbering + ".png"
```

```
    image = Image.open(decoder_numbering, 'r')
```

```
    imagedata = iter(image.getdata())
```

```
    while (True):
```

```
        pixels = [value for value in imagedata.__next__()[0:3] + imagedata.__next__()[0:3] +  
imagedata.__next__()[0:3]]
```

```
        # string of binary data
```

```
        binstr = "
```

```
        for i in pixels[0:8]:
```

```
            if (i % 2 == 0):
```

```
                binstr += '0'
```

```
            else:
```

```
                binstr += '1'
```

```
        if re.match("[ -~]", chr(int(binstr,2))) is not None: # only decode printable data
```

```
            data += chr(int(binstr, 2))
```

```
        if (pixels[-1] % 2 != 0):
```

```
            return data
```

```
# Runtime
```

```
print("Please Enter Start and End Frame where Data is Hidden At")
```

```
frame_start = int(input("Start Frame: "))
```

```

frame_end = int(input("End Frame: "))

frame_location = input("Frames Location: ")

print("Extracting Data...")

decodedtextfile = open('output\decoded_frame.txt', 'a')

decodedtextfile.write('Decoded Text:\n')

for convnum in range(frame_start, frame_end + 1):

    try:

        decodedtextfile.write(decode(convnum))

        print("Data found in Frame %d" % convnum)

    except StopIteration:

        print("No data found in Frame %d" % convnum)

decodedtextfile.close()

print("\nExtraction Complete!")

```

Encoder.py

```

import math

from PIL import Image

# Convert encoding data into 8-bit binary ASCII

def generateData(data):

    newdata = []

    for i in data: # list of binary codes of given data

        newdata.append(format(ord(i), '08b'))

    return newdata

# Pixels modified according to encoding data in generateData

def modifyPixel(pixel, data):

    datalist = generateData(data)

    lengthofdata = len(datalist)

```



```

imagedata = iter(pixel)

for i in range(lengthofdata):

    # Extracts 3 pixels at a time

    pixel = [value for value in imagedata.__next__():3] + imagedata.__next__():3] +
imagedata.__next__():3]

    # Pixel value should be made odd for 1 and even for 0

    for j in range(0, 8):

        if (datalist[i][j] == '0' and pixel[j] % 2 != 0):

            pixel[j] -= 1

        elif (datalist[i][j] == '1' and pixel[j] % 2 == 0):

            if(pixel[j] != 0):

                pixel[j] -= 1

            else:

                pixel[j] += 1

        # Eighth pixel of every set tells whether to stop or read further. 0 means keep reading; 1 means
        the message is over.

        if (i == lengthofdata - 1):

            if (pixel[-1] % 2 == 0):

                if(pixel[-1] != 0):

                    pixel[-1] -= 1

                else:

                    pixel[-1] += 1

            else:

                if (pixel[-1] % 2 != 0):

                    pixel[-1] -= 1

        pixel = tuple(pixel)

        yield pixel[0:3]

        yield pixel[3:6]

        yield pixel[6:9]

```

```

def encoder(newimage, data):

    w = newimage.size[0]

    (x, y) = (0, 0)


    for pixel in modifyPixel(newimage.getdata(), data):

        # Putting modified pixels in the new image

        newimage.putpixel((x, y), pixel)

        if (x == w - 1):

            x = 0

            y += 1

        else:

            x += 1


# Improved Encoding Function

# Instead of performing Steganography on all the frames, the function will now instead perform
Steganography on selected range of frames

def encode(start, end, filename, frame_loc):

    total_frame = end - start + 1

    try:

        with open(filename) as fileinput: # Store Data to be Encoded

            filedata = fileinput.read()

    except FileNotFoundError:

        print("\nFile to hide not found! Exiting...")

        quit()

    datapoints = math.ceil(len(filedata) / total_frame) # Data Distribution per Frame

    counter = start

    print("Performing Steganography...")

```

```

for convnum in range(0, len(filedata), datapoints):

    numbering = frame_loc + "\\\" + str(counter) + ".png"

    encodetext = filedata[convnum:convnum+datapoints] # Copy Distributed Data into Variable

    try:

        image = Image.open(numbering, 'r') # Parameter has to be r, otherwise ValueError will occur
        (https://pillow.readthedocs.io/en/stable/reference/Image.html)

    except FileNotFoundError:

        print("\n%d.png not found! Exiting..." % counter)

        quit()

    newimage = image.copy() # New Variable to Store Hiddend Data

    encoder(newimage, encodetext) # Steganography

    new_img_name = numbering # Frame Number

    newimage.save(new_img_name, str(new_img_name.split(".")[1].upper())) # Save as New Frame

    counter += 1

print("Complete!\n")

# Runtime

while True:

    try:

        print("Please Enter Start and End Frame where Data will be Hidden At")

        frame_start = int(input("Start Frame: "))

        frame_end = int(input("End Frame: "))

        if frame_start < frame_end:

            break

        else:

            print("\nStarting Frame must be larger than ending Frame! Please try again...")

    except ValueError:

        print("\nInteger expected! Please try again...")

frame_location = input("Frames Location: ")

```

```
filename = input("File to Hide (inc. extension): ")  
encode(frame_start, frame_end, filename, frame_location)
```

frames to video.py

```
import cv2
```

```
import numpy as np
```

```
import os
```

```
from os.path import isfile, join
```

```
def convert_frames_to_video(pathIn,pathOut,fps):
```

```
    frame_array = []
```

```
    files = [f for f in os.listdir(pathIn) if isfile(join(pathIn, f))]
```

```
    #for sorting the file names properly
```

```
    files.sort(key = lambda x: int(x[5:-4]))
```

```
    for i in range(len(files)):
```

```
        filename=pathIn + files[i]
```

```
        #reading each files
```

```
        img = cv2.imread(filename)
```

```
        height, width, layers = img.shape
```

```
        size = (width,height)
```

```
        print(filename)
```

```
        #inserting the frames into an image array
```

```
        frame_array.append(img)
```

```
    out = cv2.VideoWriter(pathOut,cv2.VideoWriter_fourcc(*'DIVX'), fps, size)
```

```
for i in range(len(frame_array)):
```

```
    # writing to a image array
```

```
    out.write(frame_array[i])
```

```
out.release()
```

```
def main():
```

```
    pathIn= 'C:\\Users\\Nithya\\Desktop\\dip_project\\output\\topgun_dip_frames'
```

```
    pathOut = 'C:\\Users\\Nithya\\Desktop\\dip_project\\output\\video.avi'
```

```
    fps = 25.0
```

```
    convert_frames_to_video(pathIn, pathOut, fps)
```

```
if __name__=="__main__":
```

```
    main()
```

RunStartHere.py

```
import os
```

```
# Checks if output Directory Exists, otherwise Create It
```

```
if not os.path.exists('output'):
```

```
    os.makedirs('output')
```

```
# Menu
```

```
print("\n1: Video Splitter and Combiner")
```

```
print("2: Hide Data in Audio")
```

```
print("3: Recover Data in Audio")
```

```
print("4: Hide Data in Frames")
```

```
print("5: Recover Data in Frames")
```

```
print("6: Steganography Detection in Images")
```

```

# User Selection

try:

    start_step = int(input("\nSelect the Program to Run: "))

    if start_step == 1:

        print("Starting Program...\n")

        os.system("python aviGUI.py")

    if start_step == 2:

        print("Starting Program...\n")

        print("==== Hide Data in Audio ====")

        file_text = input("Text File to Hide: ")

        file_audioOld = input("Original Audio File (inc. extension): ")

        bits_int = str(input("LSB Bits: "))

        line_string = ("python wav-steg.py -h -d \"" + file_text + "\" -s \"" + file_audioOld + "\" -o\noutput\\steg_audio.wav" + " -n " + bits_int)

        os.system(line_string)

    elif start_step == 3:

        print("Starting Program...\n")

        print("==== Recover Data in Audio ====")

        audio_file = input("File to Recover From (inc. extension): ")

        no_of_lsb_bits = str(input("LSB Bits: "))

        no_of_bytes = str(input("Number of Bytes: "))

        line_string = ("python wav-steg.py -r -s \"" + audio_file + "\" -o output\\decoded_audio.txt" + " -n\n" + no_of_lsb_bits + " -b " + no_of_bytes)

        os.system(line_string)

    elif start_step == 4:

```

```

    print("Starting Program...\n")

    print("=== Hide Data in Frames ===")

    os.system("python Encoder.py")


elif start_step == 5:

    print("Starting Program...\n")

    print("=== Recover Data in Frames ===")

    os.system("python Decoder.py")


elif start_step == 6:

    print("Starting Program...\n")

    print("=== Steganography Image Detection ===")

    os.system("python analysis.py")


else:

    print("\nInvalid Input! Exiting...\n")

    quit()


except ValueError:

    print("Non-Integer Input Entered. Exiting...\n")

except KeyboardInterrupt:

    print("\nUser canceled, exiting...")

    quit()


wave_steg.py

import getopt, os, sys, math, struct, wave


def print_usage():

    print("Error Occured")

```

```

#print("\nUsage options:\n",
# "-h, --hide    If present, the script runs to hide data\n",
# "-r, --recover If present, the script runs to recover data\n",
# "-s, --sound   What follows is the name of carrier wav file\n",
# "-d, --data    What follows is the file name having data to hide\n",
# "-o, --output  Output filename of choice\n",
# "-n, --nlsb   Number of LSBs to use\n",
# "-b, --bytes   Number of bytes to recover\n"
#" --help      Display help\n")

```

```
def prepare(sound_path):
```

```
    global sound, params, n_frames, n_samples, fmt, mask, smallest_byte
```

```
    sound = wave.open(sound_path, "r")
```

```
    params = sound.getparams()
```

```
    num_channels = sound.getnchannels()
```

```
    sample_width = sound.getsampwidth()
```

```
    n_frames = sound.getnframes()
```

```
    n_samples = n_frames * num_channels
```

```
    if (sample_width == 1): # samples are unsigned 8-bit integers
```

```
        fmt = "{}B".format(n_samples)
```

```
        # Used to set the least significant num_lsb bits of an integer to zero
```

```
        mask = (1 << 8) - (1 << num_lsb)
```

```
        # The least possible value for a sample in the sound file is actually
```

```
        # zero, but we don't skip any samples for 8 bit depth wav files.
```

```
        smallest_byte = -(1 << 8)
```

```
    elif (sample_width == 2): # samples are signed 16-bit integers
```



```

    fmt = "{}h".format(n_samples)

    # Used to set the least significant num_lsb bits of an integer to zero
    mask = (1 << 15) - (1 << num_lsb)

    # The least possible value for a sample in the sound file
    smallest_byte = -(1 << 15)

else:

    # Python's wave module doesn't support higher sample widths
    raise ValueError("File has an unsupported bit-depth")

def hide_data(sound_path, file_path, output_path, num_lsb):

    global sound, params, n_frames, n_samples, fmt, mask, smallest_byte
    prepare(sound_path)

    # We can hide up to num_lsb bits in each sample of the sound file
    max_bytes_to_hide = (n_samples * num_lsb) // 8
    filesize = os.stat(file_path).st_size

    if (filesize > max_bytes_to_hide):

        required_LSBs = math.ceil(filesize * 8 / n_samples)

        raise ValueError("Input file too large to hide, "
                        "requires {} LSBs, using {}".format(required_LSBs, num_lsb))

    print("Using {} B out of {} B".format(filesize, max_bytes_to_hide))

    # Put all the samples from the sound file into a list
    raw_data = list(struct.unpack(fmt, sound.readframes(n_frames)))
    sound.close()

```

```

input_data = memoryview(open(file_path, "rb").read())

# The number of bits we've processed from the input file
data_index = 0
sound_index = 0

# values will hold the altered sound data
values = []
buffer = 0
buffer_length = 0
done = False

while(not done):
    while (buffer_length < num_lsb and data_index // 8 < len(input_data)):
        # If we don't have enough data in the buffer, add the
        # rest of the next byte from the file to it.
        buffer += (input_data[data_index // 8] >> (data_index % 8)
                    ) << buffer_length
        bits_added = 8 - (data_index % 8)
        buffer_length += bits_added
        data_index += bits_added

    # Retrieve the next num_lsb bits from the buffer for use later
    current_data = buffer % (1 << num_lsb)
    buffer >>= num_lsb
    buffer_length -= num_lsb

```

```

while (sound_index < len(raw_data) and
      raw_data[sound_index] == smallest_byte):
    # If the next sample from the sound file is the smallest possible
    # value, we skip it. Changing the LSB of such a value could cause
    # an overflow and drastically change the sample in the output.
    values.append(struct.pack(fmt[-1], raw_data[sound_index]))
    sound_index += 1

if (sound_index < len(raw_data)):
    current_sample = raw_data[sound_index]
    sound_index += 1

    sign = 1
    if (current_sample < 0):
        # We alter the LSBs of the absolute value of the sample to
        # avoid problems with two's complement. This also avoids
        # changing a sample to the smallest possible value, which we
        # would skip when attempting to recover data.
        current_sample = -current_sample
        sign = -1

    # Bitwise AND with mask turns the num_lsb least significant bits
    # of current_sample to zero. Bitwise OR with current_data replaces
    # these least significant bits with the next num_lsb bits of data.
    altered_sample = sign * ((current_sample & mask) | current_data)

    values.append(struct.pack(fmt[-1], altered_sample))

```

```

if (data_index // 8 >= len(input_data) and buffer_length <= 0):

    done = True

while(sound_index < len(raw_data)):

    # At this point, there's no more data to hide. So we append the rest of
    # the samples from the original sound file.

    values.append(struct.pack(fmt[-1], raw_data[sound_index]))

    sound_index += 1

sound_steg = wave.open(output_path, "w")

sound_steg.setparams(params)

sound_steg.writeframes(b"".join(values))

sound_steg.close()

print("Data hidden over { } audio file".format(output_path))

def recover_data(sound_path, output_path, num_lsb, bytes_to_recover):

    # Recover data from the file at sound_path to the file at output_path

    global sound, n_frames, n_samples, fmt, smallest_byte

    prepare(sound_path)

    # Put all the samples from the sound file into a list

    raw_data = list(struct.unpack(fmt, sound.readframes(n_frames)))

    # Used to extract the least significant num_lsb bits of an integer

    mask = (1 << num_lsb) - 1

    output_file = open(output_path, "wb+")

    data = bytearray()

    sound_index = 0

```

```

buffer = 0

buffer_length = 0

sound.close()

while (bytes_to_recover > 0):

    next_sample = raw_data[sound_index]

    if (next_sample != smallest_byte):

        # Since we skipped samples with the minimum possible value when
        # hiding data, we do the same here.

        buffer += (abs(next_sample) & mask) << buffer_length

        buffer_length += num_lsb

    sound_index += 1

while (buffer_length >= 8 and bytes_to_recover > 0):

    # If we have more than a byte in the buffer, add it to data

    # and decrement the number of bytes left to recover.

    current_data = buffer % (1 << 8)

    buffer >>= 8

    buffer_length -= 8

    data += struct.pack('1B', current_data)

    bytes_to_recover -= 1

output_file.write(bytes(data))

output_file.close()

print("Data recovered to { } text file".format(output_path))

try:

```

```
    opts, args = getopt.getopt(sys.argv[1:], 'hrs:d:o:n:b:',
                                ['hide', 'recover', 'sound=', 'data=',
                                'output=', 'nlsb=', 'bytes=', 'help'])

except getopt.GetoptError:

    print_usage()

    sys.exit(1)


hiding_data = False

recovering_data = False


for opt, arg in opts:

    if opt in ("-h", "--hide"):

        hiding_data = True

    elif opt in ("-r", "--recover"):

        recovering_data = True

    elif opt in ("-s", "--sound"):

        sound_path = arg

    elif opt in ("-d", "--data"):

        file_path = arg

    elif opt in ("-o", "--output"):

        output_path = arg

    elif opt in ("-n", "--nlsb"):

        num_lsb = int(arg)

    elif opt in ("-b", "--bytes"):

        bytes_to_recover = int(arg)

    elif opt in ("--help"):

        print_usage()

        sys.exit(1)
```

else:

```
print("Invalid argument {}".format(opt))
```

try:

```
if (hiding_data):
```

```
    hide_data(sound_path, file_path, output_path, num_lsb)
```

```
if (recovering_data):
```

```
    recover_data(sound_path, output_path, num_lsb, bytes_to_recover)
```

except Exception as e:

```
print("Ran into an error during execution. Check input and try again.\n")
```

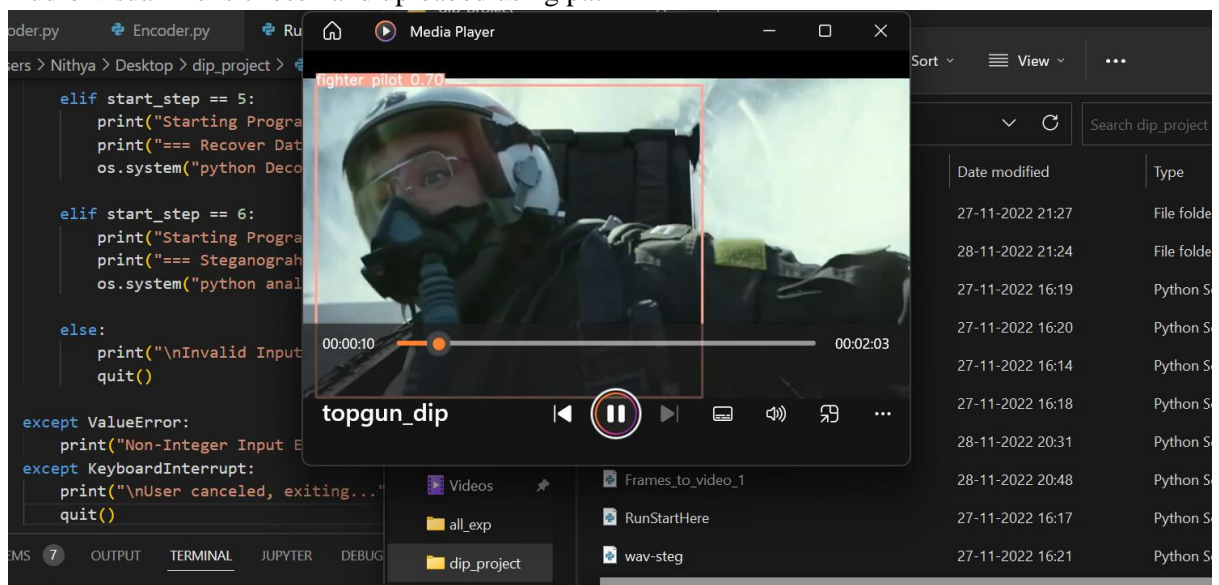
```
print(e)
```

```
print_usage()
```

```
sys.exit(1)
```

Execution

1. Audio-visual file is chosen and uploaded using path



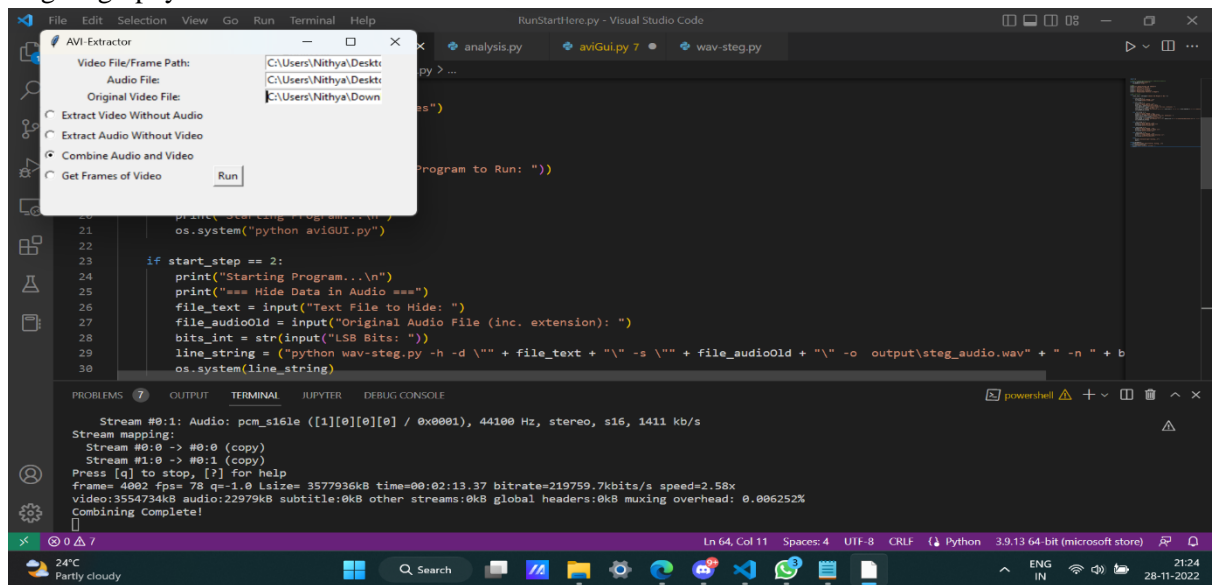
Program menu

```
PS C:\Users\Nithya\Desktop\dip_project> python RunStartHere.py
```

- 1: Video Splitter and Combiner
- 2: Hide Data in Audio
- 3: Recover Data in Audio
- 4: Hide Data in Frames
- 5: Recover Data in Frames
- 6: Steganography Detection in Images

```
Select the Program to Run: 2
Starting Program...
```

2. Audio and video are separated and an audio and frames folder are created.
3. The original input file along with the two newly extracted files are uploaded for steganography

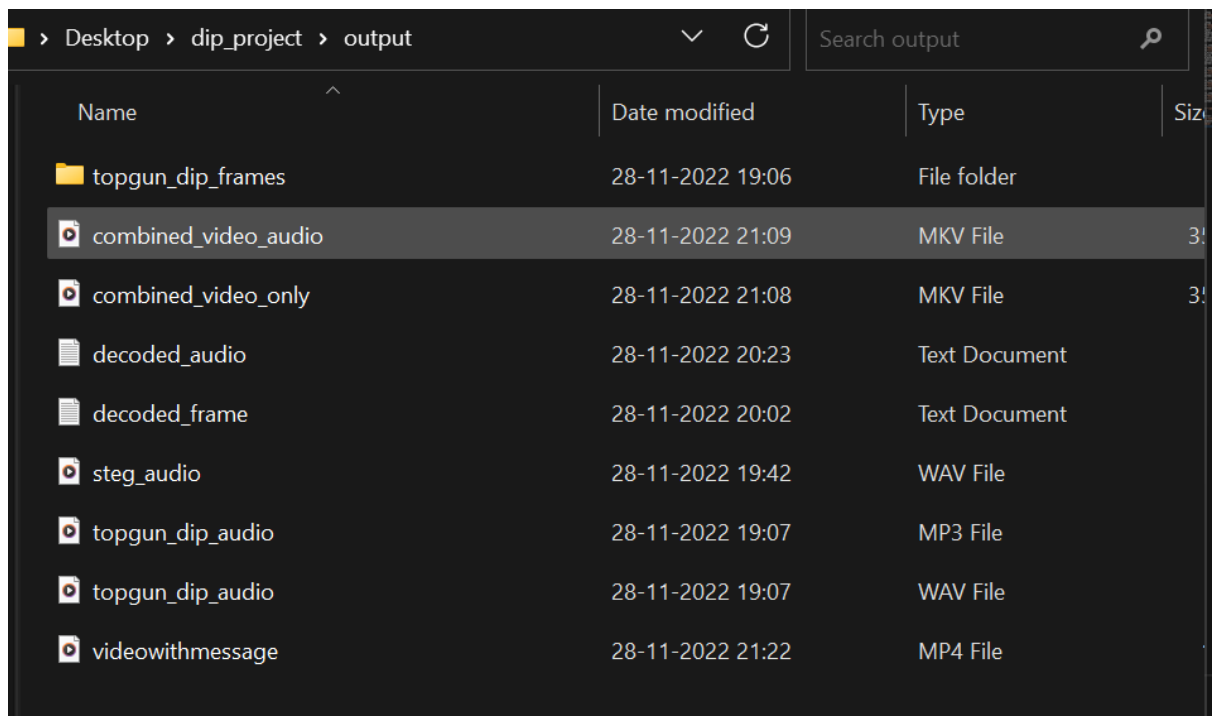


4. Data is Encrypted by providing desired specifications

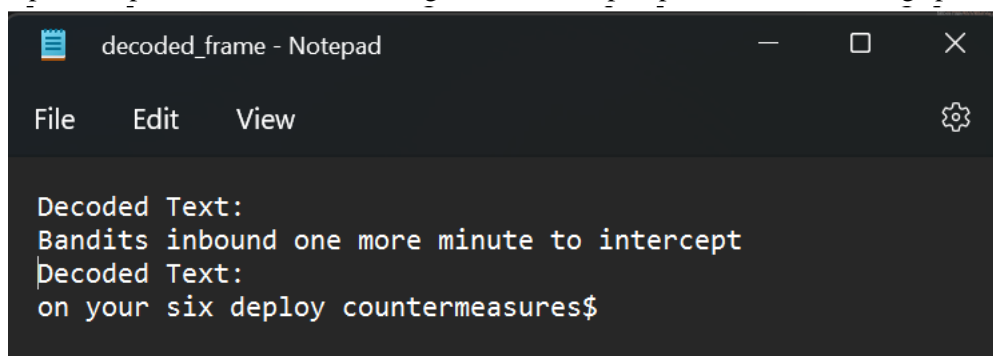
```
Select the Program to Run: 4
Starting Program...

=== Hide Data in Frames ===
Please Enter Start and End Frame where Data will be Hidden At
Start Frame: 0
End Frame: 58
Frames Location: C:\Users\Nithya\Desktop\dip_project\output\topgun_dip_frames
File to Hide (inc. extension): C:\Users\Nithya\Desktop\dip_message1.txt
Performing Steganography...
Complete!
```

5. An encrypted video is received in the form of a .mkv file



6. Upon reception, the decoded message is received upon provision of encoding specifications

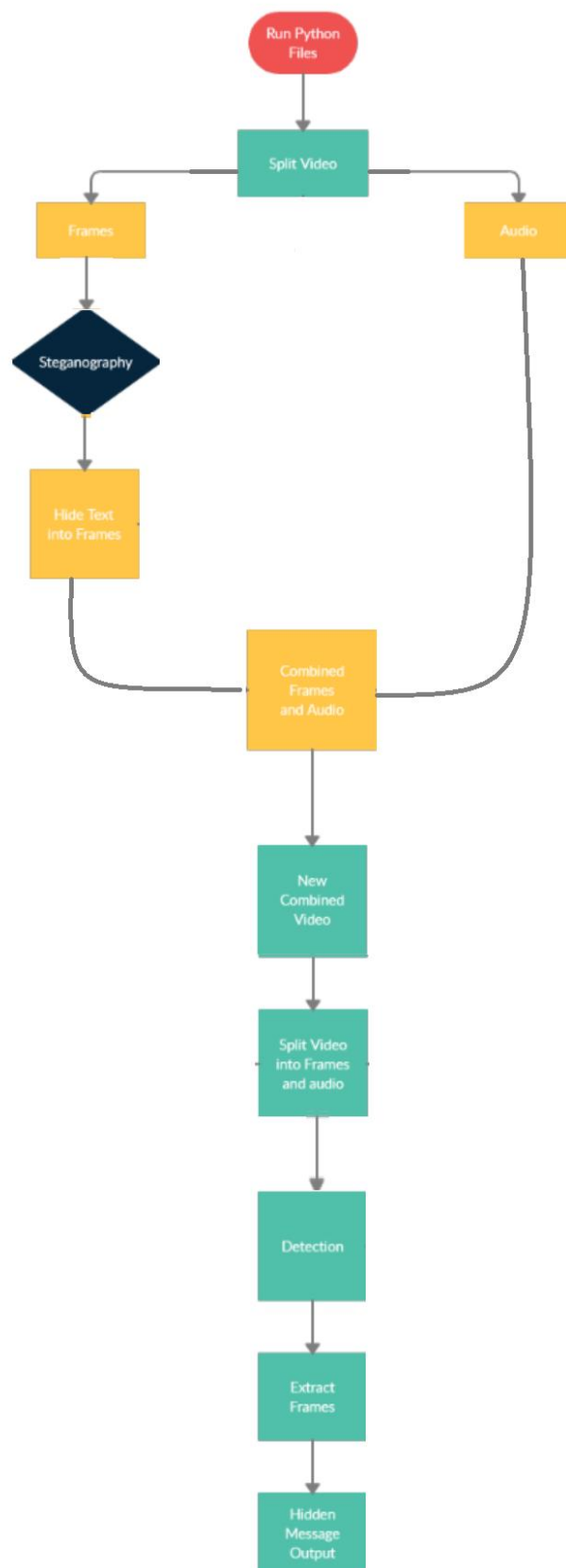


Data frame steganography detection

```
Select the Program to Run: 5
Starting Program...

=== Recover Data in Frames ===
Please Enter Start and End Frame where Data is Hidden At
Start Frame: 0
End Frame: 58
Frames Location: C:\Users\Nithya\Desktop\dip_project\output\topgun_dip_frames
Extracting Data...
Data found in Frame 0
Data found in Frame 1
Data found in Frame 2
Data found in Frame 3
Data found in Frame 4
Data found in Frame 5
Data found in Frame 6
Data found in Frame 7
Data found in Frame 8
Data found in Frame 9
Data found in Frame 10
Data found in Frame 11
Data found in Frame 12
Data found in Frame 13
Data found in Frame 14
Data found in Frame 15
```

Block diagram



Observations:

As we can observe this could be of great help to military sector in detecting enemy missiles and decrypting secret enemy messages as well as warning fellow soldiers by sending secrets using steganography.