Odd Semester (2023)

BINUS UNIVERSITY

BINUS INTERNATIONAL

Final Project Cover Letter

(Individual Work)

Student Information

Surname: Wang **Given Name:** Cindy Reginia **Student ID:** 2702334815

Course Code : COMP6047001 **Course Name:** Algorithm and Programming

Class : L1AC **Lecturer:** Jude Joseph Lamug Martinez MCS

Type of Assignment: Final Project Report

Submission Pattern

Due Date **:** 13 January 2023 **Submission Date:** 13 January 2023

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

Binus International seriously regards all forms of plagiarism, cheating, and collusion as academic offences which may result in severe penalties, including loss/drop of marks, course/class discontinuity, and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept, and consent to Binus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student: Cindy Reginia Wang

A. Background

A requirement for passing this algorithm and programming course is to create a program using Python. This program should include all of the things we learned during this course to test our knowledge on what we learned in class, making sure we have a firm understanding of basic python skills. We are also instructed to go beyond and explore new concepts and techniques that we have not yet learned in class. For this reason, I decided to work with cv2 and tkinter to make my algorithm and programming final project, camera filter app.

The reason why I decided to make a camera filter application was primarily due to the fact that it seemed like a good way for me to learn new techniques and work with trained models. Another reason being that it sounded like a challenging yet fun project to take on since I get to be creative in making various filters of characters I enjoy. This camera filter app is built for entertainment purposes, therefore I hope the users have fun trying out the various filters I created

B. Index

A. Background.....3
B. Index.....3
C. Project Specification.....4
1. Introduction.....4
2. Modules/Library/Framework.....4
3. Important Folder/Files.....5
D. Solution Design.....6
1. Sketchboard.....6
2. Use Case Diagram.....7
3. Activity Diagram.....8
4. Class Diagram.....9
E. Essential Algorithms.....9
F. Evaluation and Reflection.....26
1. Evaluation.....26
2. Reflection.....27
G. Working Program Evidence.....28
1. Demo Video.....28
2. Screenshots.....28

H. References..... 28

C. Project Specification

1. Introduction

At first, I didn't really have an idea as to what I wanted to make for this final project. I considered lots of topics like match calculators, cosy games and necessities trackers, however in the end I decided to try my hand in making a camera filter app. I asked around, researched and found that they had pre-trained models that can track people's face using OpenCV (though mostly they used OpenCV for face detection, object detection, etc and not silly face filters). With this technology, I hoped to make a goofy, lighthearted application that can showcase the skills I learned during this course and also the new technologies that I learned during the making of this project.

After running the program, the main camera of the laptop will be turned on and showcased in the tkinter window. The user will be greeted with a live video feed of their camera and two rows of buttons. The first row shows a "take photo" button that can be used to take photos. The second row shows the options for the various filters, "Anonymous Filter", "Clown Filter", "Ayam Filter", "Luffy Filter" and "Gojo Filter" respectively. Clicking on any of the filters will apply the selected filter onto the users face. Interacting with any of the buttons will also display a message on the third row, for example taking a picture will trigger the snapshot counter message and applying a filter will trigger the filter applied message. Clicking on an already applied filter will take off the filter and there will be a message indicating that the filter is turned off. The Users can access the images taken during the session inside the file where the program is run.

2. Modules/Library/Framework

- ➔ **OpenCV - Library**

OpenCV is an open source computer vision library used for computer vision tasks like accessing video from the camera, capturing images, applying filters and detecting faces. Specifically, I used the XML file (Extensible Markup Language files) "**haarcascade_frontalface_default.xml**" for help in detecting faces. Other modules I used include VideoCapture, cvtColor, CascadeClassifier, rectangle, circle, resize, bitwise_and and imread. These are going to be further explained in part E, essential algorithms.

- **Tkinter - Library**

  Tkinter is a library used to make the GUI (graphical user interface) elements of this application such as the main window, video feed canvas, buttons (taking photos and applying filters) and labels for status updates.

- **Numpy - Library**

  Numpy is a library used to perform numerical computations and array manipulations. I used this particularly to compute array operations with "**numpy.array**".

- **Python Imaging Library (PIL) - Library**

  PIL is used for handling images. "**Image**" module is used for opening and manipulating images, while "**ImageTk**" is used to convert images to a format compatible with Tkinter for displaying on the canvas.

3. **Important Folder/Files**

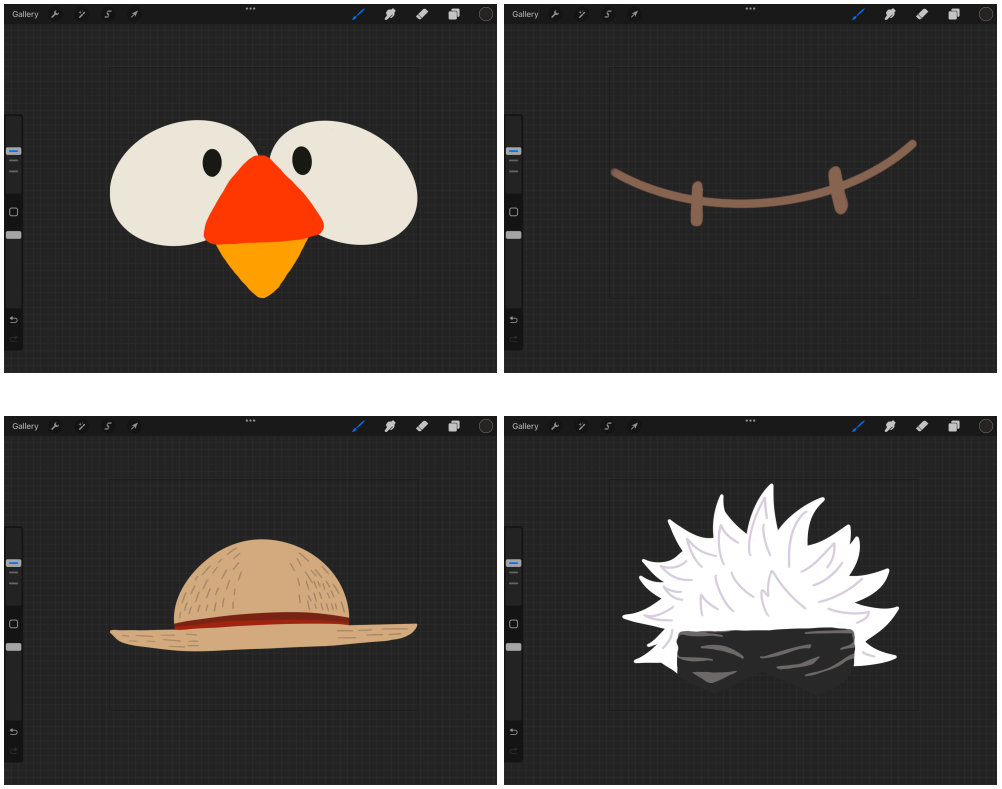
- cam.py - contains the python code to run the camera filter app
- camera.ipynb - alternative version of Cam.py, if users find it easier to run in Jupyter Notebook

- ➔ ayameye.PNG, ayamnose.png, gojo.PNG, luffyhat.PNG, luffyscar.PNG - images used for filters
- ➔ snapshot_{self.photo_counter}.png - filename for the pictures saved to the file

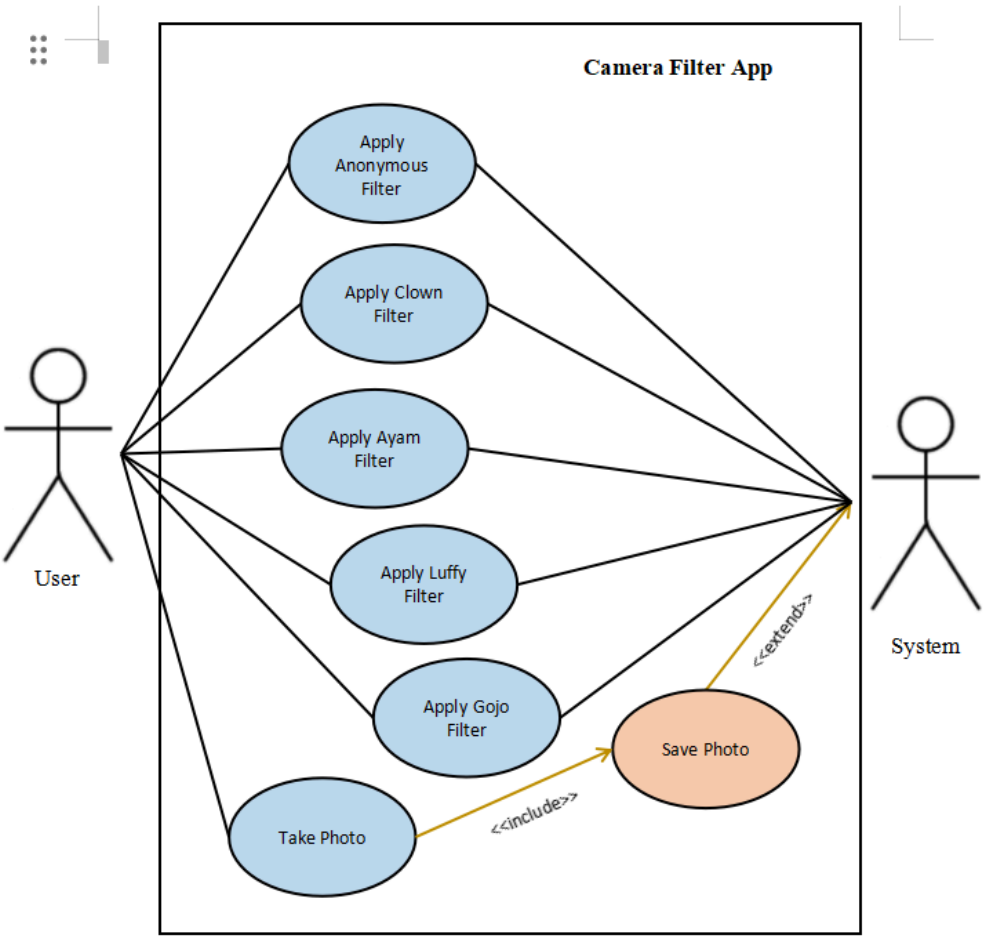
D. Solution Design

1. Sketchboard

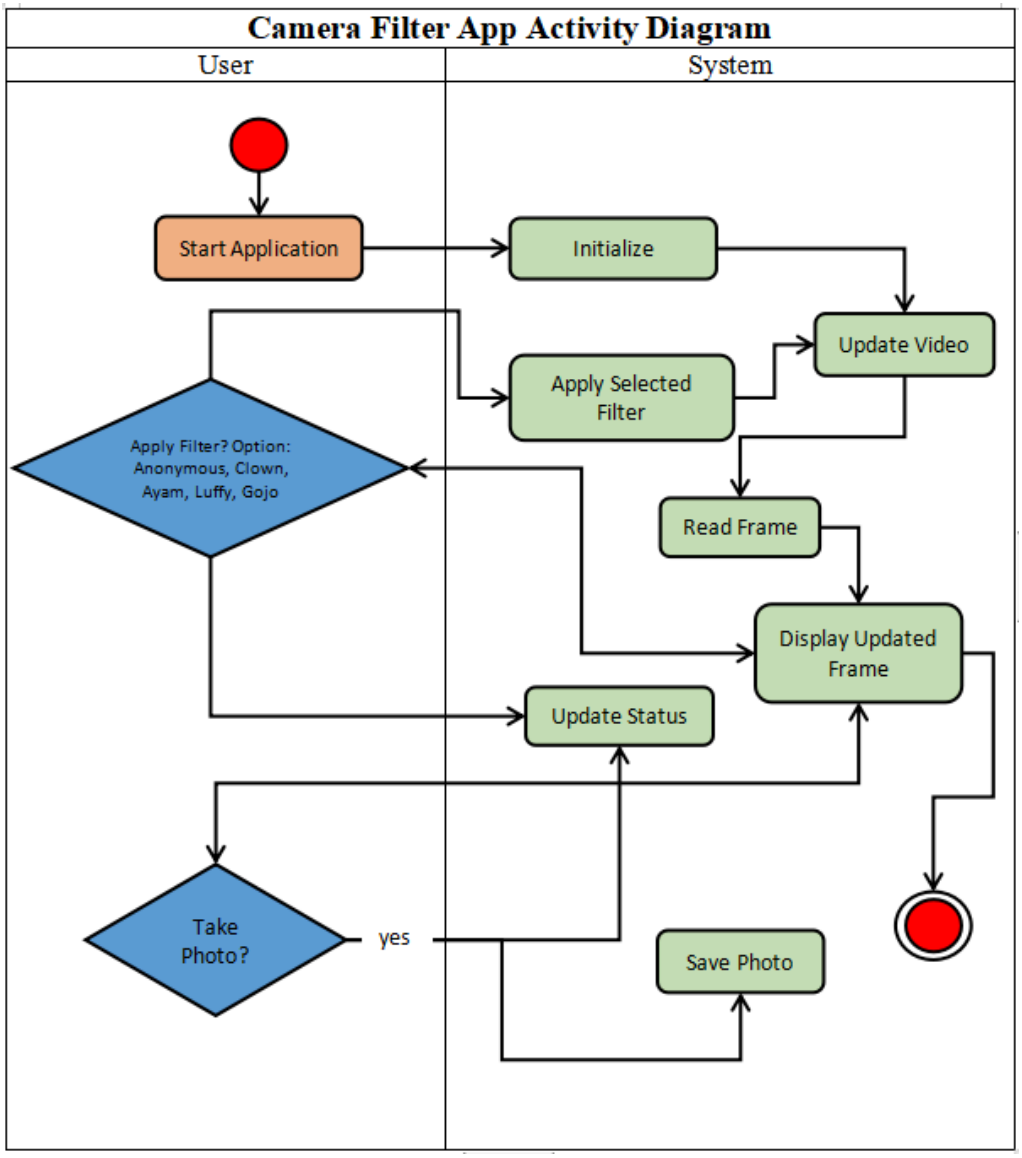
These are screenshots of the filters I drew using procreate.



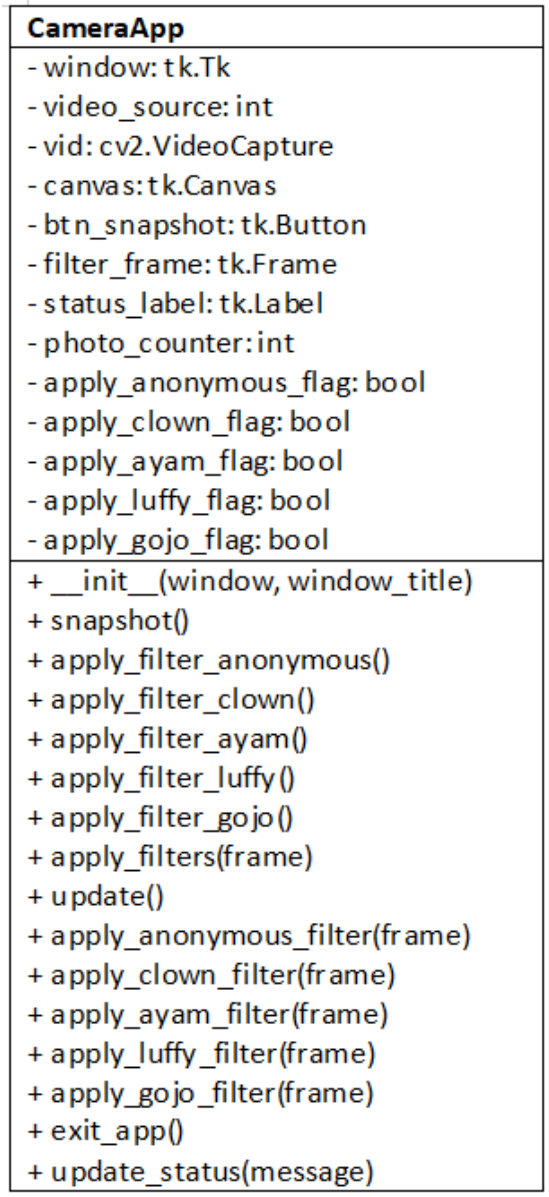
2. Use Case Diagram



3. Activity Diagram



4. Class Diagram



E. Essential Algorithms

➔ Libraries

```
#libraries

import tkinter as tk

import cv2

import numpy as np

from PIL import Image, ImageTk
```

Used pip to install opencv, numpy and Pillow, tkinter came pre-installed.

➔ **CameraApp Class**

```
class CameraApp:
  def __init__(self, window, window_title):
    self.window = window
    self.window.title(window_title)
```

Creates the class "**CameraApp**", which will be the class that runs the whole code. "**__init__**" is the method that acts as the class constructor. "**self.window**" stores the tkinter window object passed during instantiation, setting its title to the provided "**window_title**"

```
    # 0 is main camera
    self.video_source = 0
    self.vid = cv2.VideoCapture(self.video_source) # capture vid using cv2
```

"**self.vid**" uses OpenCV's VideoCapture (**cv2.VideoCapture**) to initialise a video capture object associated with the main camera (video_source = 0).

```
    # make canvas using tk, showcase live cam
    self.canvas = tk.Canvas(window, width=self.vid.get(cv2.CAP_PROP_FRAME_WIDTH),
height=self.vid.get(cv2.CAP_PROP_FRAME_HEIGHT))
    self.canvas.pack() # pack canvas to window
```

Create a canvas where the dimensions are based on the video feed frame width and height (information provided in pixels), then pack it to the window so that the live video stream can be displayed. "**pack()**" method is used to display the canvas widget within its parent window. "**self.canvas.pack()**", lets Tkinter adjust the size of the Canvas widget to fit its content and then display it within the main window (**self.window**).

```
# make button to take pics
self.btn_snapshot = tk.Button(window, text="Take Photo", width=20, command=self.snapshot)
self.btn_snapshot.pack(pady=10) # pack button to window

# Create a frame to contain the filter buttons
self.filter_frame = tk.Frame(window)
self.filter_frame.pack() # pack to window

# Pack filter buttons in a single row
filter_buttons = [
    ("Anonymous Filter", self.apply_filter_anonymous),
    ("Clown Filter", self.apply_filter_clown),
    ("Ayam Filter", self.apply_filter_ayam),
    ("Luffy Filter", self.apply_filter_luffy),
    ("Gojo Filter", self.apply_filter_gojo)
]

# make and pack filter button using for loop with corresponding commands
for text, command in filter_buttons:
    btn = tk.Button(self.filter_frame, text=text, width=20, command=command)
    btn.pack(side=tk.LEFT, padx=5, pady=5)

# make status label
self.status_label = tk.Label(window, text="", font=("Arial", 12))
self.status_label.pack(pady=5) # pack
```

First, it initialises a button "**self.btn_snapshot**" labelled with (Take Photo) that will trigger the "**self.snapshot**" method when clicked. Second, the frame "**self.filter_frame**" is created to store the row of filter buttons. These filter buttons are defined as a list of tuples containing text labels and corresponding command functions. A for loop creates buttons for each filter using the defined text labels and command functions. These buttons are packed into the "**self.filter_frame**" using the

layout (**side=tk.LEFT, padx=5, pady=5**), “side=tk.LEFT” means the widget will be packed from left to right within the container, “padx=5” and “pady=5” add 5 pixel padding around each button. Lastly, “**self.status_label**” is created, which is first set to an empty string then packed below the filter buttons with the y padding of 5 pixels.

```
# initilialise flags for photo counter + application of filters
self.photo_counter = 0
self.apply_anonymous_flag = False
self.apply_clown_flag = False
self.apply_ayam_flag = False
self.apply_luffy_flag = False
self.apply_gojo_flag = False

#start update loop to continuously refresh video feed
self.update()

self.window.protocol("WM_DELETE_WINDOW", self.exit_app) # close window
self.window.mainloop() # start mainloop of window
```

This part initialises flags to help control the functionalities, this is the base function when the application starts up. It sets “**photo_counter**” to 0 so to keep track of how many photos are going to be taken in this session. Flags are created (apply_anonymous_flag, apply_clown_flag, apply_ayam_flag, apply_luffy_flag, apply_gojo_flag) to serve as indicators, to make it easier to indicate which filter is turned on later on. The function “**self.update()**” initiates a loop responsible for continuously refreshing the video feed, ensuring that the displayed content remains up-to-date. This constant refresh provides a real-time view from the camera while allowing for filter application or taking snapshots. The lines “**self.window.protocol("WM_DELETE_WINDOW", self.exit_app)**” and “**self.window.mainloop()**” handle the window's close behaviour. The first line calls

the exit_app method when the user tries to exit, and the second line starts the main loop of the window, allowing the application to run and respond to user interactions.

```
def snapshot(self): # this method takes pic from video feed
    ret, frame = self.vid.read() # capture frame from video feed
    if ret: # if sucessful
        frame = self.apply_filters(frame) # apply the filter to the frame
        self.photo_counter += 1 # increase photo counter whenever photo taken
        filename = f"snapshot_{self.photo_counter}.png" # create file name (based on photo counter)
        cv2.imwrite(filename, frame) # save captured frame as image using filename
        self.update_status(f"Snapshot {self.photo_counter} taken!") # update status label
```

The snapshot method is responsible for capturing the image from the live feed. First it reads the frame from the video feed with "**self.vid.read()**". If ret is True, it applies the active filter to the frame using the "**apply_filters**" method, it also increments the "**photo_counter**" to keep track of how many photos are taken. The snapshot is then saved with "snapshot_{self.photo_counter}.png" in which "self.photo_conter" is the current photo count. This ensures that each snapshot is saved with a different filename. *Keep in mind that if you have the same file, it writes over it and saves over the image so the file you had previously will be deleted.*

```
def apply_filter_anonymous(self):
    self.apply_anonymous_flag = not self.apply_anonymous_flag # flips the state true to false, vice
versa
    if self.apply_anonymous_flag: # if click on, turn off other filters
        self.apply_ayam_flag = False
        self.apply_clown_flag = False
        self.apply_luffy_flag = False
        self.apply_gojo_flag = False
        self.update_status("Anonymous Filter applied.") # update status
    else: # if click off
```

```
        self.update_status("Anonymous Filter turned off.") # update status


def apply_filter_clown(self):
    self.apply_clown_flag = not self.apply_clown_flag
    if self.apply_clown_flag:
        self.apply_ayam_flag = False
        self.apply_anonymous_flag = False
        self.apply_luffy_flag = False
        self.apply_gojo_flag = False
        self.update_status("Clown Filter applied.")
    else:
        self.update_status("Clown Filter turned off.")


def apply_filter_ayam(self):
    self.apply_ayam_flag = not self.apply_ayam_flag
    if self.apply_ayam_flag:
        self.apply_anonymous_flag = False
        self.apply_clown_flag = False
        self.apply_luffy_flag = False
        self.apply_gojo_flag = False
        self.update_status("Ayam Filter applied.")
    else:
        self.update_status("Ayam Filter turned off.")


def apply_filter_luffy(self):
    self.apply_luffy_flag = not self.apply_luffy_flag
    if self.apply_luffy_flag:
        self.apply_anonymous_flag = False
        self.apply_clown_flag = False
        self.apply_ayam_flag = False
        self.apply_gojo_flag = False
        self.update_status("Luffy Filter applied.")
    else:
        self.update_status("Luffy Filter turned off.")
```

```
def apply_filter_gojo(self):
    self.apply_gojo_flag = not self.apply_gojo_flag
    if self.apply_gojo_flag:
        self.apply_anonymous_flag = False
        self.apply_clown_flag = False
        self.apply_ayam_flag = False
        self.apply_luffy_flag = False
        self.update_status("Gojo Filter applied.")
    else:
        self.update_status("Gojo Filter turned off.")
```

These methods toggle the state of different filters and ensure only one filter is active at a time. The filter buttons packed to the window in "**__innit__**" have these methods applied to them if you click on the buttons. It will reverse the current flag of the filter and if the flag is reversed to True, it will make sure to turn off all the other filters (set all the other filter flags to False). It also updates the "**update_status**" accordingly when the filter is applied or when the filter is turned off.

```
def apply_filters(self, frame): # apply filter to frame
    if self.apply_anonymous_flag: # if active
        frame = self.apply_anonymous_filter(frame) # apply
    elif self.apply_clown_flag:
        frame = self.apply_clown_filter(frame)
    elif self.apply_ayam_flag:
        frame = self.apply_ayam_filter(frame)
    elif self.apply_luffy_flag:
        frame = self.apply_luffy_filter(frame)
    elif self.apply_gojo_flag:
        frame = self.apply_gojo_filter(frame)
    return frame
```

This method checks if the flags are True, if they are True, it applies that filter's method to the frame. Then it returns the frame after applying the filters.

```
def update(self): #updates video feed
    ret, frame = self.vid.read() #read frame
    if ret: #if sucessful
        frame = self.apply_filters(frame) #apply filter to frame
        #convert frame to image compatible w tk (BGR to RGB)
        self.photo = ImageTk.PhotoImage(image=Image.fromarray(cv2.cvtColor(frame,
cv2.COLOR_BGR2RGB)))
        self.canvas.create_image(0, 0, image=self.photo, anchor=tk.NW) #display image
        self.canvas.image = self.photo  # Keep a reference to the image to prevent garbage collection
    self.window.after(10, self.update) #updates every 10 miliseconds
```

Continuously refreshes the video feed. It reads a frame from the video source, applies the selected filter to that frame using the "**apply_filters**" method, converts the frame into an image compatible with tkinter (converting BGR to RGB format), creates an image on the canvas using that frame, and updates the canvas to display the new image. This process repeats every 10 milliseconds (self.window.after(10, self.update)) to maintain the video feed in near real-time.

```
def apply_anonymous_filter(self, frame):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) #convert to grayscale for detection

    #load face detection Haar Cascade
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

    #detect the faces in the grayscale frame
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5)
```

```
#covers eye region with black rectangle
for (x, y, w, h) in faces:
    eye_region_y = y + int(h * 1 / 4)
    eye_region_h = int(h * 1 / 3)
    cv2.rectangle(frame, (x, eye_region_y), (x + w, eye_region_y + eye_region_h), (0, 0, 0), -1)

#convert to grayscale then to color BGR format
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
frame = cv2.cvtColor(gray, cv2.COLOR_GRAY2BGR)
return frame # return w anonymous filter applied
```

The method "**apply_anonymous_filter**" takes a frame as an input and returns the modified frame. First this method converts the frame into grayscale using "**cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)**" Then it loads the pre-trained Haar Cascade classifier used for frontal face detection using "**cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')**". It then detects faces in the grayscale frame using detectMultiScale (this allows for more than one face to be detected at a time). "scaleFactor" specifies how much the image size is reduced at each image scale. 1.3 means the image is reduced by 30% at each scale. Smaller values increase detection time but improve the accuracy of smaller details while for "minNeighbors", it defines how many neighbours each candidate rectangle should have to retain it. This parameter affects the quality of the detected faces. Higher values result in fewer detections but with higher quality. Lower values can result in more detections but with lower quality. Then for each detected face a black rectangle is drawn onto the eye region of the face.

- **for (x, y, w, h) in faces**: This loop iterates over each detected face, retrieving its coordinates and size (width and height).

- **eye_region_y = y + int(h * 1 / 4)**: It calculates the vertical position of the eye region within the face. y represents the top-left y-coordinate of the detected face. The int(h * 1 / 4) part indicates that the top of the eye region starts a quarter down from the top of the face (h represents the height of the face).
- **eye_region_h = int(h * 1 / 3)**: This calculates the height of the eye region. It's set to one-third of the total height of the detected face. This is used to determine the height of the rectangle that covers the eyes.
- **cv2.rectangle(frame, (x, eye_region_y), (x + w, eye_region_y + eye_region_h), (0, 0, 0), -1)**: This draws a black rectangle over the detected eye region. The "rectangle()" function from OpenCV takes the top-left and bottom-right coordinates (**x, eye_region_y**) and (**x + w, eye_region_y + eye_region_h**) respectively, as well as the colour (0, 0, 0) (black in BGR colour format), and -1 specifies that the rectangle should be filled entirely.

Afterwards, the frame is converted from BGR to Gray colour to have a black and white filter over the image, then converted again to BGR to save the image.

```
  def apply_clown_filter(self, frame):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5)

    for (x, y, w, h) in faces:
      # calc coordinates of nose in detected face rectangle
      nose_x = x + w // 2 #width divide by 2 (top left)
      nose_y = y + 3 * h // 5 #height div by 3 (top left)
      cv2.circle(frame, (nose_x, nose_y), 30, (0, 0, 255), -1)  # red circle for nose

      eye_y = y + h // 3 #height div by 3
```

```
        # Blue triangles for eyes
        cv2.drawContours(frame, [
            np.array([
                (x + w // 4, eye_y), #left
                (x + w // 4 + 30, eye_y - 30), #top
                (x + w // 4 + 50, eye_y) #right
            ])
        ], 0, (255, 0, 0), -1)
        cv2.drawContours(frame, [
            np.array([
                (x + 3 * w // 4, eye_y),
                (x + 3 * w // 4 - 30, eye_y - 30),
                (x + 3 * w // 4 - 50, eye_y)
            ])
        ], 0, (255, 0, 0), -1)

        eye_bottom_y = y + h // 2 - 10

        # Inverted blue triangles for eye bottoms
        cv2.drawContours(frame, [
            np.array([
                (x + w // 4, eye_bottom_y),
                (x + w // 4 + 30, eye_bottom_y + 30),
                (x + w // 4 + 50, eye_bottom_y)
            ])
        ], 0, (255, 0, 0), -1)
        cv2.drawContours(frame, [
            np.array([
                (x + 3 * w // 4, eye_bottom_y),
                (x + 3 * w // 4 - 30, eye_bottom_y + 30),
                (x + 3 * w // 4 - 50, eye_bottom_y)
            ])
        ], 0, (255, 0, 0), -1)
```

```
    return frame
```

The clown filter method draws a red circle onto the nose of the user.

- **nose_x**: The x-coordinate of the clown nose is set to the horizontal center of the detected face (x + w // 2), which is the middle of the face.
- **nose_y**: The y-coordinate of the clown nose is set slightly below the vertical centre (y + 3 * h // 5) to place it closer to the bottom of the detected face rectangle. (three-fifths of the height)
- **cv2.circle(frame, (nose_x, nose_y), 30, (0, 0, 255), -1)**: (nose_x, nose_y) are the centre coordinates of the circle, 30 is the radius of the circle, 30 pixels, (0, 0, 255) is the colour red and -1 is filled in.

The clown filter method draws two triangles at the top of the eyes.

- **eye_y**: The y-coordinate for the eyes is set to a position approximately one-third (y + h // 3) from the top of the detected face rectangle.
- **For the left eye**: It draws a triangle using three points: left, top, and right points calculated with offsets (w // 4, w // 4 + 30, w // 4 + 50).
- **For the right eye**: Another triangle is drawn using points calculated from the right side of the face rectangle (3 * w // 4, 3 * w // 4 - 30, 3 * w // 4 - 50).

The clown filter method draws two inverted triangles at the bottom of the eyes.

- **eye_bottom_y**: The y-coordinate for the bottom part of the eye is calculated as approximately halfway down the detected face rectangle minus an offset (y + h // 2 - 10).
- **For the left eye**: An inverted triangle is drawn using points calculated from the left side of the face rectangle (w // 4, w // 4 + 30, w // 4 + 50).
- **For the right eye**: Another inverted triangle is drawn using points calculated from the right side of the face rectangle (3 * w // 4, 3 * w // 4 - 30, 3 * w // 4 - 50).

```
    def apply_ayam_filter(self, frame):
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
        faces = face_cascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5)

        eye_img = cv2.imread('ayameye.PNG', -1)  # Load the eye image with alpha channel
        nose_img = cv2.imread('ayamnose.PNG', -1)  # Load the nose image with alpha channel

        for (x, y, w, h) in faces:
            face_roi = frame[y:y+h, x:x+w] # area of frame that encompasses the detected face

            # Resize the eye image to match the size of the detected face region
            eye_resized = cv2.resize(eye_img, (w, h))

            # Create a mask using the alpha channel of the eye image
            eye_mask = eye_resized[:, :, 3]  # Alpha channel of the eye image
            eye_mask = cv2.cvtColor(eye_mask, cv2.COLOR_GRAY2BGR)  # Convert to 3-channel mask

            # Use the mask to blend the eye image and the frame
            eye_overlay = eye_resized[:, :, 0:3]  # Extract RGB channels
            eye_background = cv2.bitwise_and(face_roi, 255 - eye_mask)  # Invert mask and extract region
from the frame
            face_roi[:] = cv2.add(eye_overlay, eye_background)  # Blend the image onto the frame

            # Place the resized nose image over the nose area same cuz manual positioning
            nose_resized = cv2.resize(nose_img, (w, h))

            # Create a mask using the alpha channel of the nose image
            nose_mask = nose_resized[:, :, 3]  # Alpha channel of the nose image
            nose_mask = cv2.cvtColor(nose_mask, cv2.COLOR_GRAY2BGR)  # Convert to 3-channel
mask

            # Use the mask to blend the nose image and the frame
```

```
        nose_overlay = nose_resized[:, :, 0:3]  # Extract RGB channels
        nose_background = cv2.bitwise_and(face_roi, 255 - nose_mask)  # Invert mask and extract
region from the frame
        face_roi[:] = cv2.add(nose_overlay, nose_background)  # Blend the image onto the frame

    return frame
```

First loads eye and nose images with alpha channel, png format that allows transparency. Images here are not positioned, this uses manual positioning. The images are just resized to fit the width and height of the detected face. The alpha channel of the eye image, stored in "**eye_resized[:, :, 3]**", is extracted to create a grayscale mask. To ensure compatibility with the frame, the mask is then converted to a 3-channel (RGB) format using "**cv2.cvtColor**". Subsequently, the eye overlay is separated into its RGB channels "**eye_resized[:, :, 0:3]**". The mask is then inverted "**255 - eye_mask**" and used to extract the corresponding region from the face in the frame "**cv2.bitwise_and(face_roi, 255 - eye_mask)**". This inverted mask essentially represents the background of the eye region in the face. Finally, the eye overlay and the extracted background are blended together using the **cv2.add** function, resulting in the final appearance of the eye on the face. The blended image is then assigned back to the face_roi, effectively replacing the original face region with the modified version that includes the overlaid eye. The same process is repeated for the nose.

```
    def apply_luffy_filter(self, frame):
        # Load hat image with transparent background
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
        faces = face_cascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5)
        hat_img = cv2.imread('luffyhat.PNG', -1)  # read hat image
        hat_img = cv2.resize(hat_img, (300, 300))  # Adjust the size
```

```
    scar_img = cv2.imread('luffyscar.PNG', -1)  # read scar image
    scar_img = cv2.resize(scar_img, (50, 50))  # Adjust size as needed



    for (x, y, w, h) in faces:
        # Adjust the hat position and size based on face detection
        hat_width = int(2 * w)  # hat twice the face width
        hat_height = int(1.25 * h) # hat 1.25 times the face height

        # Resize the hat image based on the calculated width and height
        resized_hat = cv2.resize(hat_img, (hat_width, hat_height))

        # Calculate adjusted x-coordinate to ensure the hat remains within the frame
        hat_x = max(0, x - int((hat_width - w) / 2))

        # Overlay the hat onto the frame
        for i in range(hat_height):
            for j in range(hat_width):
                if hat_x + j < frame.shape[1]:  # Ensure x-coordinate is within frame width
                    if resized_hat[i, j][3] != 0:  # Check alpha channel for transparency
                        frame[y + i - int(0.7 * h), hat_x + j] = resized_hat[i, j, 0:3]

        # Add the scar under the right eye
        scar_x = x + int(w * 0.6)  # x-coordinate for scar placement
        scar_y = y + int(h * 0.4)  # y-coordinate for scar placement

        # Overlay the scar onto the frame
        for i in range(scar_img.shape[0]):
            for j in range(scar_img.shape[1]):
                if scar_y + i < frame.shape[0] and scar_x + j < frame.shape[1]:
                    if scar_img[i, j][3] != 0:  # Check alpha channel for transparency
                        frame[scar_y + i, scar_x + j] = scar_img[i, j, 0:3]

    return frame
```

The hat is resized to be twice the face width and 1.25 times the face height. The hat's x coordinate is then changed to make sure it fits into the frame. The adjusted x-coordinate is determined by taking the maximum value between 0 and the original x-coordinate (x) minus half of the difference between the hat width and the width of the detected face (w). This calculation centres the hat horizontally over the face, adjusting it to fit within the frame if necessary. Following the adjustment, a nested loop iterates over the pixels of the resized hat image "**resized_hat**". For each pixel, it checks whether the corresponding x-coordinate, considering the adjustment, is within the width of the frame. If so, it further checks the alpha channel "**resized_hat[i, j][3]**" of the pixel to determine its transparency. If the pixel is not transparent, meaning it contributes to the hat image, the colour channels of the hat pixel are overlaid onto the corresponding position in the frame. This process is repeated for each pixel in the hat, effectively blending the hat onto the detected face at the adjusted position. The term **y + i - int(0.7 * h)** determines the vertical position for overlaying the hat, adjusting it to appear slightly above the top of the detected face. The scar is added under the right eye.

```
def apply_gojo_filter(self, frame):
    gojo_img = cv2.imread('gojo.png', -1)  # Load the Gojo image with alpha channel

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5)

    gojo_img = cv2.resize(gojo_img, (300, 300))  # Adjust size

    for (x, y, w, h) in faces:
```

```
        # Adjust the gojo position and size based on face detection
        gojo_width = int(2 * w)  #hat twice as wide
        gojo_height = int(1.5 * h) # hat 1.5 times taller

        # Resize the hat image based on the calculated width and height
        resized_gojo = cv2.resize(gojo_img, (gojo_width, gojo_height))

        # Calculate adjusted x-coordinate to ensure the hat remains within the frame
        gojo_x = max(0, x - int((gojo_width - w) / 2))

        # Overlay gojo onto the frame
        for i in range(gojo_height):
            for j in range(gojo_width):
                if gojo_x + j < frame.shape[1]:  # Ensure x-coordinate is within frame width
                    if resized_gojo[i, j][3] != 0:  # Check alpha channel for transparency
                        frame[y + i - int(0.7 * h), gojo_x + j] = resized_gojo[i, j, 0:3]

    return frame
```

The gojo filter uses the same mechanism as Luffy's hat, however it's height is 1,5 the detected face height.

```
def exit_app(self): # for closing
    self.vid.release()
    self.window.destroy()
```

This method is used for exiting the app, it stops the video capture process "**self.vid.release()**" and then destroys the tkinter window "**self.window.destroy()**".

```
def update_status(self, message): # update status label
    self.status_label.config(text=message)
```

This method is responsible for updating the text displayed on the status label within the tkinter window. It takes a message parameter and uses the "**config**" method of the "**status_label**" object to set the "text" attribute to the provided message. This method dynamically changes the status message displayed in the GUI based on the input it receives.

```
if __name__ == "__main__":
    root = tk.Tk()
    app = CameraApp(root, "Camera Filter App")
```

This code block is commonplace python code used to ensure that the following code only runs when the script is executed directly, not when it's imported as a module into another script. When the script is executed directly, it creates a Tkinter window (root) and initialises an instance of the CameraApp class, passing the root window and the title "Camera Filter App". This line effectively starts the application by instantiating the CameraApp class with the necessary arguments.

F. Evaluation and Reflection

1. Evaluation

Overall, I am quite happy with the result of my camera filter app. However there are of course things that can definitely be improved upon. For example, the Luffy filter and Gojo filter are both laggy and the loading time of the application is quite slow. However these are quite minor things that don't really affect the app. When I become more knowledgeable on how to improve code efficiency, I would like to revisit this code to see if I can make any adjustments to make it run smoother.

2. Reflection

Working on this project has been both stressful and fun. I had a hard time trying to find resources that would teach me how to use cv2 for making filters. But I didn't want to give up because I thought that a camera filter app was a unique challenge to take on. Though working with cv2 as someone who had no prior experience in coding before is absolutely frustrating, maybe because I had planned for an even more complete camera filter app. My first thought was to make an anonymous filter, where you could censor out someone's face and voice. So I had planned on just focusing on one filter and perfecting it. However I came to realise that CV2 could not record audio, so i had to use other libraries like PyAudio to record the video then use Pydub to censor out the voice and change its pitch, but for some reason Pydub was not working on my laptop. In addition to that since the video and audio is recorded separately, I had to use another library to edit the videos together. It would've been simple enough but since the cv2 camera capture takes ages to load, the audio would get recorded first then the video would start, so I played around with adjusting the time and adding a delay to the audio capture. That worked but for some reason the video was taken in slow motion (most likely due to lag) so I tried speeding the video up, but then if you try to take longer videos it would first be in sync then slowly get out of sync. Not to mention the black and white filter would work when I took a picture but it would mysteriously be gone when I recorded the video. Sometimes the video would turn me blue and make me look like a smurf. I was not having fun working on the code so I took a break from the code and rethought about my purpose in making this app. I wanted this project to be fun. So I decided to add in aspects that I liked. I liked drawing, so I thought why not make anime character filters. Things started looking up and I found the whole process fun again. I fixed the smurf situation thankfully and started working on adding more filters, which is why Iended up with 5 filters. I am proud of what I ended up making. Youtube and StackOverflow helped

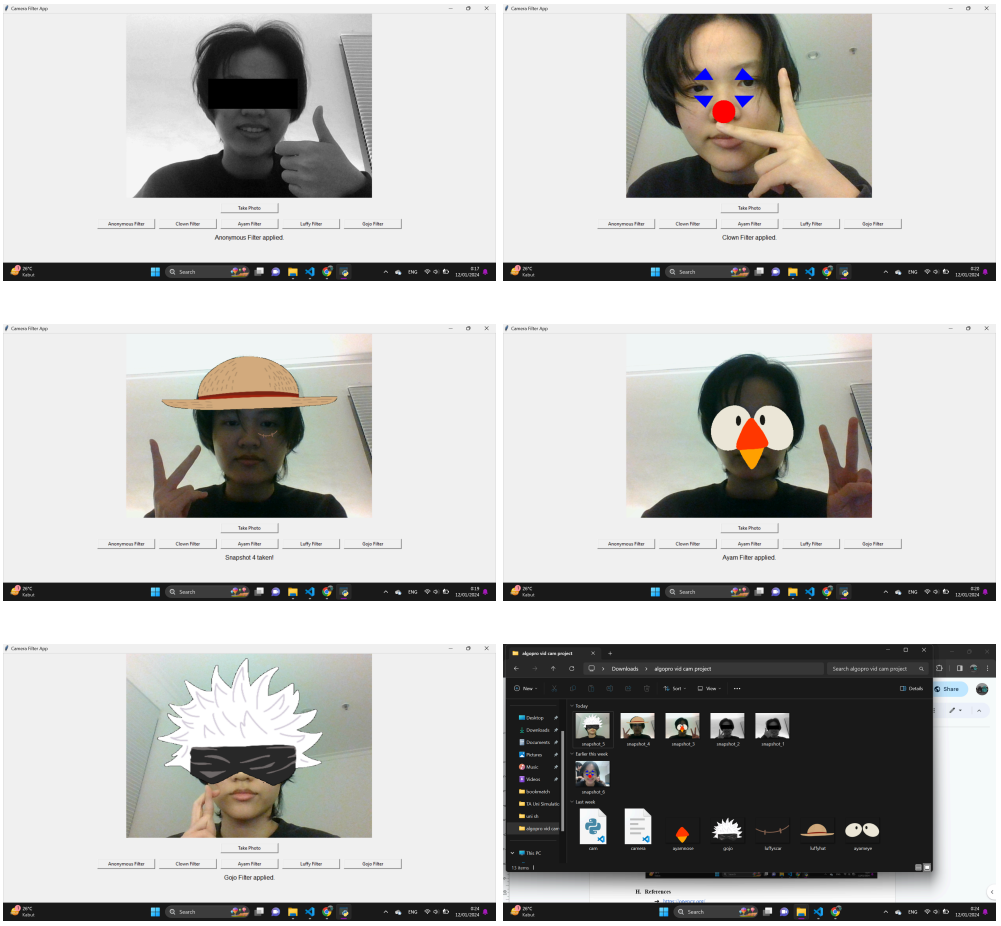
tremendously in helping me fix some code. Overall, I am grateful for the experience I gained by making this project, I hope that I can continue doing projects that are fun.

G. Working Program Evidence

1. Demo Video

   https://drive.google.com/file/d/1TOpGzDNs_rj34g6Fb4xaEyD5x4sbrU22/view?usp=sharing

2. Screenshots



H. References

➔ https://opencv.org/

- https://medium.com/@arpit23sh/adding-sunglasses-to-your-photo-using-opencv-and-python-b99c769e6f3f
- https://stackoverflow.com/questions/29262793/opencv-with-python-trying-to-overlay-an-image-over-the-eyes-on-a-webcam-feed
- https://youtu.be/HZ3mhgbjtRU?si=ih2PGMka0qNWUNC0
- https://forum.opencv.org/t/blue-tint-in-image-how-to-adjust-blue-in-opencv/1574
- https://stackoverflow.com/questions/42406338/why-cv2-imwrite-changes-the-color-of-pics
- https://github.com/jremmons/pyfakewebcam/issues/2
- https://youtube.com/playlist?list=PLS1QulWo1RIa7D1O6skqDQ-JZ1GGHKK-K&si=bc54kzmdzX5kDhuQ