```python
from picamera2 import Picamera2
import cv2
import mediapipe as mp
import pygame
import numpy as np
import os
import sys
import RPi.GPIO as GPIO
import time
import math
import pygame,pigame
from pygame.locals import *
from cartoonize import caart

# ----------------- flag set -----------------
RUNNING = True
in_start_menu = True

# ----------------- time set -----------------
start_time = time.time()
run_time = 1000
fps_start_time = time.time()
fps = 0

# ----------------- GPIO set -----------------
GPIO.setmode(GPIO.BCM)
GPIO.setup(27, GPIO.IN, pull_up_down=GPIO.PUD_UP)

def GPIO27_callback(channel):
    global RUNNING
    print("Quit....")
    RUNNING = False

GPIO.add_event_detect(27, GPIO.FALLING, callback=GPIO27_callback, bouncetime=200)

# Set SDL environment to ensure PiTFT display
os.putenv('SDL_VIDEODRIVER', 'fbcon')  # Frame cache
os.putenv('SDL_FBDEV', '/dev/fb0')
os.putenv('SDL_MOUSEDRV', 'dummy')
os.putenv('SDL_MOUSEDEV', '/dev/null')
os.putenv('DISPLAY', '')

# ----------------- pygame set -----------------
# Initialize pygame
pygame.init()
pitft = pigame.PiTft()
screen_size = (320, 240)
screen = pygame.display.set_mode(screen_size)
pygame.display.set_caption('Magic Menu')
screen.fill((0,0,0))
pygame.display.update()


# ----------------- display set -----------------
# pygame.mouse.set_visible(False)
# Colors
ORANGE = (255, 165, 0)
YELLOW = (255, 255, 0)
BLACK = (0, 0, 0)
RED = (255, 0, 0)
WHITE = (255, 255, 255)
# Define colors for the grid
BLUE = (0, 116, 217)      # #0074D9
NAVY = (0, 31, 63)        # #001F3F
GOLD = (255, 195, 0)      # #FFC300
GRAY = (179, 179, 179)    # #B3B3B3
```

```python
BEIGE = (253, 245, 230)   # #FDF5E6
FOREST = (34, 139, 34)    # #228B22

font = pygame.font.Font(None, 30)
font_mid = pygame.font.Font(None, 45)
font_big = pygame.font.Font(None, 60)
options = ['glass', 'hat', 'all', 'sketch', 'cartoon', 'skeleton']
clock = pygame.time.Clock()

# ----------------- camera set -----------------
# Initialize PiCamera2
picam2 = Picamera2()
picam2.configure(picam2.create_preview_configuration(main={"size": (320, 240)}))
picam2.start()

# Initialize MediaPipe
mp_drawing = mp.solutions.drawing_utils
mp_face_mesh = mp.solutions.face_mesh
mp_hands = mp.solutions.hands
# mp_holistic = mp.solutions.holistic

# ----------------- frame set -----------------
frame_count = 0
frame_interval = 2
result = None

fps_frame_count = 0

# ----------------- mode flags -----------------
mode1 = False  # Glasses filter mode
mode2 = False  # Hat filter mode
mode3 = False  # All filters mode (glasses + hat + cigarette)
mode4 = False  # Sketch filter mode
mode5 = False  # Cartoon filter mode
mode6 = False  # Skeleton/wireframe mode

# Global variables for countdown
countdown_active = False
countdown_start_time = 0
countdown_duration = 3  # 3 seconds countdown
new_mode = None

# read glasses image
glasses_img = cv2.imread('./assets/glasses.png', cv2.IMREAD_UNCHANGED)  # Include alpha channel
if glasses_img is None:
    print("Error: Glasses image not found. Please check the path.")
    exit()

hat_img = cv2.imread('./assets/hat.png', cv2.IMREAD_UNCHANGED)  # Include alpha channel
if glasses_img is None:
    print("Error: Hat image not found. Please check the path.")
    exit()

cigarette_img = cv2.imread('./assets/cigarette.png', cv2.IMREAD_UNCHANGED)  # Include alpha channel
if cigarette_img is None:
    print("Error: Cigarette image not found. Please check the path.")
    exit()

# Add near the image loading section at the beginning of the file
button_img = cv2.imread('./assets/button.png', cv2.IMREAD_UNCHANGED)  # Read button image
if button_img is None:
    print("Error: Button image not found. Please check the path.")
    exit()

# Display start menu with project title and options
def display_start_menu():
```

```
    # Fill top half with orange and bottom half with yellow
    screen.fill(ORANGE, rect=(0, 0, 320, 100))
    screen.fill(YELLOW, rect=(0, 100, 320, 140))

    # Draw circles at (160, 100)
    pygame.draw.circle(screen, YELLOW, (160, 80), 80)  # Larger yellow circle
    pygame.draw.circle(screen, ORANGE, (160, 80), 70)  # Smaller orange circle

    # Display title in the middle of the top half
    title_surface_b = font_big.render('Magic', True, WHITE)
    title_rect_b = title_surface_b.get_rect(center=(170, 65))
    screen.blit(title_surface_b, title_rect_b)

    title_surface = font_big.render('Magic', True, BLACK)
    title_rect = title_surface.get_rect(center=(160, 75))
    screen.blit(title_surface, title_rect)

    # Calculate dimensions for each grid cell
    cell_width = 320 // 3
    cell_height = 140 // 2  # (240-100) / 2 = 70

    for i in range(6):
        row = i // 3
        col = i % 3
        x = col * cell_width
        y = 100 + row * cell_height
        # screen.fill(colors[i], rect=(x, y, cell_width, cell_height))

        # Calculate the size and position of the button image
        button_size = (int(cell_width * 0.9), int(cell_height * 0.9))
        button_x = x + (cell_width - button_size[0]) // 2
        button_y = y + (cell_height - button_size[1]) // 2

        # Resize the button image
        resized_button = cv2.resize(button_img, button_size)
        # rotate the image 90 degrees clockwise
        rotated_button = cv2.rotate(resized_button, cv2.ROTATE_90_CLOCKWISE)

        # Correctly handle the transparent channel
        if rotated_button.shape[2] == 4:  # Check if there is an alpha channel
            # Separate BGR and alpha channels
            bgr = rotated_button[:, :, :3]
            alpha = rotated_button[:, :, 3]

            # Convert to RGB
            rgb = cv2.cvtColor(bgr, cv2.COLOR_BGR2RGB)

            # Create surface with alpha
            button_surface = pygame.Surface(rgb.shape[:-1], pygame.SRCALPHA)

            # Convert surface to a compatible format
            button_surface = button_surface.convert_alpha()  # Use convert_alpha() for surfaces with alpha

            pygame.surfarray.pixels3d(button_surface)[:] = rgb
            pygame.surfarray.pixels_alpha(button_surface)[:] = alpha

        screen.blit(button_surface, (button_x, button_y))

    # Display options
    for i, option in enumerate(options):
        row = i // 3
        col = i % 3
        x_position = col * cell_width + cell_width// 2
        y_position = 100 + row * cell_height + cell_height// 2

        option_surface = font.render(f'{option}', True, BEIGE)
```

```
        option_rect = option_surface.get_rect(center=(x_position, y_position))
        screen.blit(option_surface, option_rect)

    pygame.display.update()

# Function to check which option is selected
def check_option_selection(x, y):
    global mode1, mode2, mode3, mode4, mode5, mode6
    # Calculate dimensions for each grid cell
    cell_width = 320 // 3
    cell_height = 140 // 2  # (240-100) / 2 = 70

    for i, option in enumerate(options):
        row = i // 3
        col = i % 3
        x_position = col * cell_width + cell_width// 2
        y_position = 100 + row * cell_height + cell_height// 2

        # Create a rectangle around the button (40 pixels padding)
        option_rect = pygame.Rect(x_position - 40, y_position - 15, 80, 40)

        if option_rect.collidepoint(x, y):
            print(option)
            # Reset all modes
            mode1 = mode2 = mode3 = mode4 = mode5 = mode6 = False
            # Set the selected mode
            if option == 'glass':
                mode1 = True
            elif option == 'hat':
                mode2 = True
            elif option == 'all':
                mode3 = True
            elif option == 'sketch':
                mode4 = True
            elif option == 'cartoon':
                mode5 = True
            elif option == 'skeleton':
                mode6 = True
            return option
    return None


def sketch_image(img):
    # Convert the image to grayscale
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Apply median blur
    gray = cv2.medianBlur(gray, 5)
    # Use Canny edge detection
    edges = cv2.Canny(gray, 50, 150)  # Adjust thresholds to enhance edges
    # Use Laplacian operator to enhance edges
    laplacian = cv2.Laplacian(gray, cv2.CV_8U, ksize=5)
    edges = cv2.bitwise_or(edges, laplacian)

    # Downsample the image
    color = cv2.bilateralFilter(img, 9, 250, 250)
    # Combine edges with color image
    edges_colored = cv2.cvtColor(edges, cv2.COLOR_GRAY2BGR)
    cartoon = cv2.bitwise_and(color, edges_colored)
    return edges_colored

# overlay image
def overlay_image(bg_image, overlay_image, x, y, overlay_size):
    overlay = cv2.resize(overlay_image, overlay_size)
    h, w, _ = overlay.shape
    alpha_overlay = overlay[:, :, 3] / 255.0
    alpha_bg = 1.0 - alpha_overlay
```

```
265
266     # Ensure overlay position is within background image bounds
267     if y < 0 or x < 0 or y + h > bg_image.shape[0] or x + w > bg_image.shape[1]:
268         print("Overlay position is out of bounds, skipping overlay.")
269         return
270
271     for c in range(0, 3):
272         bg_image[y:y+h, x:x+w, c] = (alpha_overlay * overlay[:, :, c] +
273                                      alpha_bg * bg_image[y:y+h, x:x+w, c])
274
275
276 # with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
277 with mp_face_mesh.FaceMesh(min_detection_confidence=0.5, min_tracking_confidence=0.5) as face_mesh, \
278      mp_hands.Hands(min_detection_confidence=0.5, min_tracking_confidence=0.5) as hands:
279     while RUNNING:
280         if time.time() - start_time > run_time:
281             print("Time's up! Exiting...")
282             RUNNING = False
283         pitft.update()
284         for event in pygame.event.get():
285             if event.type is MOUSEBUTTONDOWN:
286                 if in_start_menu:
287                     x, y = pygame.mouse.get_pos()
288                     selected_option = check_option_selection(x, y)
289                     if selected_option is not None:
290                         in_start_menu = False
291                 else:
292                     # Handle other events during real-time detection if necessary
293                     pass
294             if event.type == pygame.KEYDOWN and event.key == pygame.K_q:
295                 RUNNING = False
296
297         if in_start_menu:
298             display_start_menu()
299         else:
300             # time.sleep(0.05)
301             frame = picam2.capture_array()
302             if frame is None:
303                 print("Unable to capture frame, please check the camera.")
304                 break
305
306             if frame_count%frame_interval == 0:
307                 # Convert frame to RGB and process with MediaPipe
308                 image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
309                 image.flags.writeable = False
310                 # results = holistic.process(image)
311                 face_results = face_mesh.process(image)
312                 hand_results = hands.process(image)
313                 image.flags.writeable = True
314                 image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
315                 frame_count = 0
316             else:
317                 image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
318                 image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
319             # Comment out the following line to see if no frames are extracted. Currently, two frames are extracted.
320             # frame_count += 1
321
322             # Draw hand landmarks
323             display_hand = False
324             gesture = "Unknown Gesture"
325             if hand_results.multi_hand_landmarks:
326                 for hand_landmarks in hand_results.multi_hand_landmarks:
327                     # mp_drawing.draw_landmarks(
328                     #     image, hand_landmarks, mp_hands.HAND_CONNECTIONS,
329                     #     mp_drawing.DrawingSpec(color=(80, 22, 10), thickness=2, circle_radius=4),
330                     #     mp_drawing.DrawingSpec(color=(80, 44, 121), thickness=2, circle_radius=2))
```

```
                    # Detect gestures
                    thumb_tip = hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP]
                    index_tip = hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP]
                    middle_tip = hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_TIP]
                    ring_tip = hand_landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_TIP]
                    pinky_tip = hand_landmarks.landmark[mp_hands.HandLandmark.PINKY_TIP]
                    index_mcp = hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_MCP]

                    # Gesture 1: Only index finger up
                    if (index_tip.y < index_mcp.y and
                        middle_tip.y > index_mcp.y and
                        ring_tip.y > index_mcp.y and
                        pinky_tip.y > index_mcp.y and
                        thumb_tip.x > index_mcp.x):
                        gesture = "Gesture 1"
                    # Gesture 2: Index and middle fingers up
                    elif (index_tip.y < index_mcp.y and
                          middle_tip.y < index_mcp.y and
                          ring_tip.y > index_mcp.y and
                          pinky_tip.y > index_mcp.y and
                          thumb_tip.x > index_mcp.x):
                        gesture = "Gesture 2"
                    # Gesture 3: Index, middle, and ring fingers up
                    elif (index_tip.y < index_mcp.y and
                          middle_tip.y < index_mcp.y and
                          ring_tip.y < index_mcp.y and
                          pinky_tip.y > index_mcp.y and
                          thumb_tip.x > index_mcp.x):
                        gesture = "Gesture 3"
                    # Gesture 4: All fingers except thumb up
                    elif (index_tip.y < index_mcp.y and
                          middle_tip.y < index_mcp.y and
                          ring_tip.y < index_mcp.y and
                          pinky_tip.y < index_mcp.y and
                          thumb_tip.x > index_mcp.x):
                        gesture = "Gesture 4"
                    # Gesture 5: All fingers up
                    elif (thumb_tip.x < index_mcp.x and
                          index_tip.y < index_mcp.y and
                          middle_tip.y < index_mcp.y and
                          ring_tip.y < index_mcp.y and
                          pinky_tip.y < index_mcp.y):
                        gesture = "Gesture 5"
                    # Gesture 6: Only thumb and pinky up (phone gesture)
                    elif (thumb_tip.x < index_mcp.x and
                          pinky_tip.y < index_mcp.y and
                          index_tip.y > index_mcp.y and
                          middle_tip.y > index_mcp.y and
                          ring_tip.y > index_mcp.y):
                        gesture = "Gesture 6"
                    else:
                        gesture = "Unknown Gesture"

                    # print(f"Detected Gesture: {gesture}")
                    display_hand = True
                    # cv2.putText(image, gesture, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

                    # Reset countdown if gesture doesn't match new_mode
                    if gesture != "Unknown Gesture" and \
                    (new_mode == "Mode 1" and gesture != "Gesture 1" or \
                    new_mode == "Mode 2" and gesture != "Gesture 2" or \
                    new_mode == "Mode 3" and gesture != "Gesture 3" or \
                    new_mode == "Mode 4" and gesture != "Gesture 4" or \
                    new_mode == "Mode 5" and gesture != "Gesture 5" or \
                    new_mode == "Mode 6" and gesture != "Gesture 6"):
                        print("Reset countdown")
```

```
                    countdown_active = False
                    new_mode = None

                # Check and switch modes based on detected gesture
                if gesture == "Gesture 1" and not mode1:
                    new_mode = "Mode 1"
                elif gesture == "Gesture 2" and not mode2:
                    new_mode = "Mode 2"
                elif gesture == "Gesture 3" and not mode3:
                    new_mode = "Mode 3"
                elif gesture == "Gesture 4" and not mode4:
                    new_mode = "Mode 4"
                elif gesture == "Gesture 5" and not mode5:
                    new_mode = "Mode 5"
                elif gesture == "Gesture 6" and not mode6:
                    new_mode = "Mode 6"

                if new_mode and not countdown_active:
                    countdown_active = True
                    countdown_start_time = time.time()


            # Inside the main loop, after rendering the gesture and FPS
            if mode1:
                if face_results.multi_face_landmarks:
                    for face_landmarks in face_results.multi_face_landmarks:
                        h, w, _ = image.shape
                        left_eye_landmark = face_landmarks.landmark[33]
                        right_eye_landmark = face_landmarks.landmark[263]

                        left_eye_x, left_eye_y = int(left_eye_landmark.x * w), int(left_eye_landmark.y * h)
                        right_eye_x, right_eye_y = int(right_eye_landmark.x * w), int(right_eye_landmark.y * h)

                        # Calculate midpoint, distance and angle for glasses placement
                        mid_x = (left_eye_x + right_eye_x) // 2
                        mid_y = (left_eye_y + right_eye_y) // 2
                        distance = math.sqrt((right_eye_x - left_eye_x)**2 + (right_eye_y - left_eye_y)**2)
                        angle = math.degrees(math.atan2((right_eye_y - left_eye_y), (right_eye_x - left_eye_x)))
                        # print(angle)

                        # Scale glasses (glasses width = 1.2x eye distance)
                        scale_factor = 1.2
                        new_width = int(distance * scale_factor)
                        aspect_ratio = glasses_img.shape[0] / glasses_img.shape[1]
                        new_height = int(new_width * aspect_ratio)

                        # Resize glasses first
                        resized_glasses = cv2.resize(glasses_img, (new_width, new_height))

                        # Calculate canvas size needed after rotation
                        angle_rad = math.radians(-angle)  # Note the negative sign here
                        cos_a = abs(math.cos(angle_rad))
                        sin_a = abs(math.sin(angle_rad))
                        new_w = int(new_width * cos_a + new_height * sin_a)
                        new_h = int(new_width * sin_a + new_height * cos_a)

                        # Create larger canvas to accommodate rotated image
                        rot_mat = cv2.getRotationMatrix2D((new_width//2, new_height//2), -angle, 1.0)
                        rot_mat[0, 2] += (new_w - new_width)//2
                        rot_mat[1, 2] += (new_h - new_height)//2
                        rotated_glasses = cv2.warpAffine(resized_glasses, rot_mat, (new_w, new_h))

                        # Adjust placement position
                        offset_y = int(new_height * 0.3)  # Reduce offset to make glasses closer to eyes
                        top_left_x = mid_x - new_w//2
                        top_left_y = mid_y - new_h//2 - offset_y
```

```
                    overlay_image(image, rotated_glasses, top_left_x, top_left_y, (new_w, new_h))
            elif mode2:
                if face_results.multi_face_landmarks:
                    for face_landmarks in face_results.multi_face_landmarks:
                        h, w, _ = image.shape
                        # Use forehead keypoint from face mesh
                        forehead = face_landmarks.landmark[10]  # Top of the head keypoint
                        chin = face_landmarks.landmark[152]  # Chin keypoint
                        left_cheek = face_landmarks.landmark[234]  # Left cheek keypoint
                        right_cheek = face_landmarks.landmark[454]  # Right cheek keypoint

                        # Calculate face width and height
                        face_width = int(abs(right_cheek.x - left_cheek.x) * w)
                        face_height = int(abs(chin.y - forehead.y) * h)

                        # Calculate head tilt angle and invert
                        delta_x = right_cheek.x - left_cheek.x
                        delta_y = right_cheek.y - left_cheek.y
                        angle = -np.arctan2(delta_y, delta_x) * (180.0 / np.pi)

                        # Determine hat position and size
                        scale_factor = 1.5  # Adjust this factor to change hat size
                        x_hat = int(forehead.x * w) - int(face_width * scale_factor) // 2
                        y_hat = int(forehead.y * h) - int(face_height * scale_factor) // 2 + int(0.2 * face_height)  # Move hat down

                        hat_width = int(face_width * scale_factor)
                        hat_height = int(hat_width * hat_img.shape[0] / hat_img.shape[1])

                        if y_hat >= 0 and x_hat >= 0 and (y_hat + hat_height) <= h and (x_hat + hat_width) <= w:
                            # Calculate rotation center
                            center = (hat_img.shape[1] // 2, hat_img.shape[0] // 2)
                            # Calculate rotation matrix
                            rotated_hat = cv2.getRotationMatrix2D(center, angle, 1.0)
                            # Calculate rotated bounding box
                            cos = np.abs(rotated_hat[0, 0])
                            sin = np.abs(rotated_hat[0, 1])
                            new_w = int((hat_img.shape[0] * sin) + (hat_img.shape[1] * cos))
                            new_h = int((hat_img.shape[0] * cos) + (hat_img.shape[1] * sin))
                            # Adjust translation part of rotation matrix
                            rotated_hat[0, 2] += (new_w / 2) - center[0]
                            rotated_hat[1, 2] += (new_h / 2) - center[1]
                            # Rotate image
                            rotated_hat_img = cv2.warpAffine(hat_img, rotated_hat, (new_w, new_h), flags=cv2.INTER_LINEAR, borderMode=cv2.BORDER_CONSTANT, borderValue=(0, 0, 0, 0))
                            overlay_image(image, rotated_hat_img, x_hat, y_hat, (hat_width, hat_height))
                        else:
                            print("Hat position is out of bounds, skipping overlay.")
            elif mode3:
                if face_results.multi_face_landmarks:
                    for face_landmarks in face_results.multi_face_landmarks:
                        # glasses
                        h, w, _ = image.shape
                        left_eye_landmark = face_landmarks.landmark[33]
                        right_eye_landmark = face_landmarks.landmark[263]

                        left_eye_x, left_eye_y = int(left_eye_landmark.x * w), int(left_eye_landmark.y * h)
                        right_eye_x, right_eye_y = int(right_eye_landmark.x * w), int(right_eye_landmark.y * h)

                        # Calculate midpoint, distance and angle for glasses placement
                        mid_x = (left_eye_x + right_eye_x) // 2
                        mid_y = (left_eye_y + right_eye_y) // 2
                        distance = math.sqrt((right_eye_x - left_eye_x)**2 + (right_eye_y - left_eye_y)**2)
                        angle = math.degrees(math.atan2((right_eye_y - left_eye_y), (right_eye_x - left_eye_x)))
                        # print(angle)

                        # Scale glasses (glasses width = 1.2x eye distance)
```

```
                        scale_factor = 1.2
                        new_width = int(distance * scale_factor)
                        aspect_ratio = glasses_img.shape[0] / glasses_img.shape[1]
                        new_height = int(new_width * aspect_ratio)

                        # Resize glasses first
                        resized_glasses = cv2.resize(glasses_img, (new_width, new_height))

                        # Calculate canvas size needed after rotation
                        angle_rad = math.radians(-angle)  # Note the negative sign here
                        cos_a = abs(math.cos(angle_rad))
                        sin_a = abs(math.sin(angle_rad))
                        new_w = int(new_width * cos_a + new_height * sin_a)
                        new_h = int(new_width * sin_a + new_height * cos_a)

                        # Create larger canvas to accommodate rotated image
                        rot_mat = cv2.getRotationMatrix2D((new_width//2, new_height//2), -angle, 1.0)
                        rot_mat[0, 2] += (new_w - new_width)//2
                        rot_mat[1, 2] += (new_h - new_height)//2
                        rotated_glasses = cv2.warpAffine(resized_glasses, rot_mat, (new_w, new_h))

                        # Adjust placement position
                        offset_y = int(new_height * 0.3)  # Reduce offset to make glasses closer to eyes
                        top_left_x = mid_x - new_w//2
                        top_left_y = mid_y - new_h//2 - offset_y

                        overlay_image(image, rotated_glasses, top_left_x, top_left_y, (new_w, new_h))

                        # hat

                        forehead = face_landmarks.landmark[10]  # Top of the head keypoint
                        chin = face_landmarks.landmark[152]  # Chin keypoint
                        left_cheek = face_landmarks.landmark[234]  # Left cheek keypoint
                        right_cheek = face_landmarks.landmark[454]  # Right cheek keypoint
                        # Calculate face width and height
                        face_width = int(abs(right_cheek.x - left_cheek.x) * w)
                        face_height = int(abs(chin.y - forehead.y) * h)

                        # Calculate head tilt angle and invert
                        delta_x = right_cheek.x - left_cheek.x
                        delta_y = right_cheek.y - left_cheek.y
                        angle = -np.arctan2(delta_y, delta_x) * (180.0 / np.pi)

                        # Determine hat position and size
                        scale_factor = 1.5  # Adjust this factor to change hat size
                        x_hat = int(forehead.x * w) - int(face_width * scale_factor) // 2
                        y_hat = int(forehead.y * h) - int(face_height * scale_factor) // 2 + int(0.2 * face_height)  # Move hat down

                        hat_width = int(face_width * scale_factor)
                        hat_height = int(hat_width * hat_img.shape[0] / hat_img.shape[1])

                        if y_hat >= 0 and x_hat >= 0 and (y_hat + hat_height) <= h and (x_hat + hat_width) <= w:
                            # Calculate rotation center
                            center = (hat_img.shape[1] // 2, hat_img.shape[0] // 2)
                            # Calculate rotation matrix
                            rotated_hat = cv2.getRotationMatrix2D(center, angle, 1.0)
                            # Calculate rotated bounding box
                            cos = np.abs(rotated_hat[0, 0])
                            sin = np.abs(rotated_hat[0, 1])
                            new_w = int((hat_img.shape[0] * sin) + (hat_img.shape[1] * cos))
                            new_h = int((hat_img.shape[0] * cos) + (hat_img.shape[1] * sin))
                            # Adjust translation part of rotation matrix
                            rotated_hat[0, 2] += (new_w / 2) - center[0]
                            rotated_hat[1, 2] += (new_h / 2) - center[1]
                            # Rotate image
                            rotated_hat_img = cv2.warpAffine(hat_img, rotated_hat, (new_w, new_h), flags=cv2.INTER_LINEAR, borderMode=cv2.BORDER_CONSTANT, borderValue=(0, 0, 0, 0))
```

```
                        overlay_image(image, rotated_hat_img, x_hat, y_hat, (hat_width, hat_height))
                    else:
                        # print("Hat position is out of bounds, skipping overlay.")
                        pass

                    # cigarette
                    # Use mouth keypoints from face mesh
                    mouth_left = face_landmarks.landmark[61]  # Left corner of the mouth
                    mouth_right = face_landmarks.landmark[291]  # Right corner of the mouth
                    mouth_center = face_landmarks.landmark[13]  # Center of the mouth

                    # Calculate mouth width and height with a scaling factor
                    scale_factor = 2.5  # Increase this factor to make the image larger
                    mouth_width = int(abs(mouth_right.x - mouth_left.x) * w * scale_factor)
                    mouth_height = int(mouth_width * cigarette_img.shape[0] / cigarette_img.shape[1])

                    # Determine cigarette position
                    x_cigarette = int(mouth_center.x * w) - mouth_width
                    y_cigarette = int(mouth_center.y * h) - mouth_height // 2 - 10

                    # Horizontal flip cigarette image to correct orientation
                    flipped_cigarette = cv2.flip(cigarette_img, 1)

                    # Ensure cigarette position is within image bounds
                    if y_cigarette >= 0 and x_cigarette >= 0 and (y_cigarette + mouth_height) <= h and (x_cigarette + mouth_width) <= w:
                        overlay_image(image, flipped_cigarette, x_cigarette, y_cigarette, (mouth_width, mouth_height))
                    else:
                        # print("Cigarette position is out of bounds, skipping overlay.")
                        pass
            elif mode4:
                # Apply sketch effect
                cartoon_frame = sketch_image(image)
                image = cartoon_frame
            elif mode5:
                # Apply the cartoon effect using the caart function
                cartoon_frame = caart(image)
                # Display the cartoonized image
                image = cartoon_frame
            elif mode6:
                if face_results.multi_face_landmarks:
                    for face_landmarks in face_results.multi_face_landmarks:
                        mp_drawing.draw_landmarks(
                            image, face_landmarks, mp_face_mesh.FACEMESH_TESSELATION,
                            mp_drawing.DrawingSpec(color=(80, 110, 10), thickness=1, circle_radius=1),
                            mp_drawing.DrawingSpec(color=(80, 256, 121), thickness=1, circle_radius=1))
                if hand_results.multi_hand_landmarks:
                    for hand_landmarks in hand_results.multi_hand_landmarks:
                        mp_drawing.draw_landmarks(
                            image, hand_landmarks, mp_hands.HAND_CONNECTIONS,
                            mp_drawing.DrawingSpec(color=(80, 22, 10), thickness=2, circle_radius=4),
                            mp_drawing.DrawingSpec(color=(80, 44, 121), thickness=2, circle_radius=2))

            # Convert image to Pygame surface and render
            # image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            image = np.rot90(image)
            surface = pygame.surfarray.make_surface(image)
            screen.blit(surface, (0, 0))

            # must behind the surface rendering
            if mode1:
                mode_text = font.render("Mode 1", True, RED)
                mode_rect = mode_text.get_rect(bottomleft=(0, 240))
                screen.blit(mode_text, mode_rect)
            elif mode2:
                mode_text = font.render("Mode 2", True, RED)
                mode_rect = mode_text.get_rect(bottomleft=(0, 240))
```

```
661                 screen.blit(mode_text, mode_rect)
662             elif mode3:
663                 mode_text = font.render("Mode 3", True, RED)
664                 mode_rect = mode_text.get_rect(bottomleft=(0, 240))
665                 screen.blit(mode_text, mode_rect)
666             elif mode4:
667                 mode_text = font.render("Mode 4", True, RED)
668                 mode_rect = mode_text.get_rect(bottomleft=(0, 240))
669                 screen.blit(mode_text, mode_rect)
670             elif mode5:
671                 mode_text = font.render("Mode 5", True, RED)
672                 mode_rect = mode_text.get_rect(bottomleft=(0, 240))
673                 screen.blit(mode_text, mode_rect)
674             elif mode6:
675                 mode_text = font.render("Mode 6", True, RED)
676                 mode_rect = mode_text.get_rect(bottomleft=(0, 240))
677                 screen.blit(mode_text, mode_rect)
678 
679             # Render gesture type to screen
680             if display_hand:
681                 gesture_text = font.render(f"{gesture}", True, RED)  # Red text
682                 gesture_rect = gesture_text.get_rect(topleft=(0, 0))
683                 screen.blit(gesture_text, gesture_rect) # Position at the top left
684 
685             # Render FPS to screen
686             fps_text = font.render(f"FPS: {fps:.2f}", True, RED)  # Red text
687             fps_rect = fps_text.get_rect(topright=(320, 0))
688             screen.blit(fps_text, fps_rect) # Position at the top right
689 
690             # Handle countdown
691             if countdown_active:
692                 current_time = time.time()
693                 elapsed_time = current_time - countdown_start_time
694                 remaining_time = countdown_duration - int(elapsed_time)
695 
696                 # Display countdown
697                 if remaining_time > 0:
698                     countdown_text = font_mid.render(f"Will change to {new_mode}", True, RED)
699                     countdown_rect = countdown_text.get_rect(center=(160, 40))
700                     screen.blit(countdown_text, countdown_rect)
701 
702                     number_text = font_big.render(f"{str(remaining_time)}.....", True, RED)
703                     number_rect = number_text.get_rect(topright=(320, 60))
704                     screen.blit(number_text, number_rect)
705                 else:
706                     # Apply the new mode after countdown
707                     if new_mode == "Mode 1":
708                         mode1 = True
709                         mode2 = mode3 = mode4 = mode5 = mode6 = False
710                     elif new_mode == "Mode 2":
711                         mode2 = True
712                         mode1 = mode3 = mode4 = mode5 = mode6 = False
713                     elif new_mode == "Mode 3":
714                         mode3 = True
715                         mode1 = mode2 = mode4 = mode5 = mode6 = False
716                     elif new_mode == "Mode 4":
717                         mode4 = True
718                         mode1 = mode2 = mode3 = mode5 = mode6 = False
719                     elif new_mode == "Mode 5":
720                         mode5 = True
721                         mode1 = mode2 = mode3 = mode4 = mode6 = False
722                     elif new_mode == "Mode 6":
723                         mode6 = True
724                         mode1 = mode2 = mode3 = mode4 = mode5 = False
725 
726                     # reset new_mode and countdown_active
```

```
                    new_mode = None
                    countdown_active = False  # Reset countdown


            pygame.display.update()

            # Increment frame count
            fps_frame_count += 1

            # Calculate and print FPS every second
            elapsed_time = time.time() - fps_start_time
            if elapsed_time >= 1.0:
                fps = fps_frame_count / elapsed_time
                # print(f"FPS: {fps:.2f}")
                fps_frame_count = 0
                fps_start_time = time.time()

        clock.tick(30)

GPIO.cleanup()
pygame.quit()
sys.exit(0)


import cv2
import scipy
from scipy import stats
import numpy as np
from collections import defaultdict


def update_c(C,hist):
    """
    Updates cluster centers using mean values of assigned pixels
    Args:
        C: Current cluster centers
        hist: Image histogram
    Returns:
        Updated cluster centers and groupings
    """
    max_iterations = 1
    for _ in range(max_iterations):
        groups=defaultdict(list)
        non_zero_indices = np.nonzero(hist)[0]

        d = np.abs(C[:, np.newaxis] - non_zero_indices)
        index = np.argmin(d, axis=0)
        for i, indice in enumerate(non_zero_indices):
            groups[index[i]].append(indice)

        new_C=np.array(C)
        for i,indice in groups.items():
            if(np.sum(hist[indice])==0):
                continue
            new_C[i]=int(np.sum(indice*hist[indice])/np.sum(hist[indice]))

        if(np.sum(new_C-C)==0):
            break
        C=new_C

    return C,groups

# Calculates K Means clustering
def K_histogram(hist):
    """
    Performs adaptive K-means clustering on image histogram
```

```
    - Starts with single cluster center at 128
    - Splits clusters that fail normality test and meet minimum size
    Args:
        hist: Image histogram
    Returns:
        Final cluster centers
    """
    alpha=0.001
    N=80
    C=np.array([128])

    while True:
        C,groups=update_c(C,hist)

        new_C=set()
        for i,indice in groups.items():
            if(len(indice)<N):
                new_C.add(C[i])
                continue

            z, pval=stats.normaltest(hist[indice])
            if(pval<alpha):
                left=0 if i==0 else C[i-1]
                right=len(hist)-1 if i ==len(C)-1 else C[i+1]
                delta=right-left
                if(delta >=3):
                    c1=(C[i]+left)/2
                    c2=(C[i]+right)/2
                    new_C.add(c1)
                    new_C.add(c2)
                else:
                    new_C.add(C[i])
            else:
                new_C.add(C[i])
        if(len(new_C)==len(C)):
            break
        else:
            C=np.array(sorted(new_C))
    return C

# The main controlling function
def caart(img):
    """
    Main cartoonization function that:
    1. Applies bilateral filtering to reduce noise while preserving edges
    2. Detects edges using Canny
    3. Converts to HSV color space
    4. Performs color quantization using adaptive clustering
    5. Draws detected edges and applies final processing
    Args:
        img: Input RGB image
    Returns:
        Cartoonized version of input image
    """
    # Apply bilateral filter to reduce noise while preserving edges
    kernel=np.ones((2,2), np.uint8)
    output=np.array(img)
    x,y,c=output.shape
    for i in range(c):
        output[:,:,i]=cv2.bilateralFilter(output[:,:,i],5,150,150)

    # Detect edges and convert to HSV for better color processing
    edge=cv2.Canny(output, 100, 200)
    output=cv2.cvtColor(output,cv2.COLOR_RGB2HSV)

    # Calculate histograms for each HSV channel
```

```
    hists = []

    hist,_=np.histogram(output[:,:,0],bins =np.arange(180+1))
    hists.append(hist)
    hist,_=np.histogram(output[:,:,1],bins =np.arange(256+1))
    hists.append(hist)
    hist,_=np.histogram(output[:,:,2],bins =np.arange(256+1))
    hists.append(hist)


    C=[]
    for h in hists:
        C.append(K_histogram(h))
    #print("centroids: {0}".format(C))

    output=output.reshape((-1,c))
    for i in range(c):
        channel=output[:,i]
        index=np.argmin(np.abs(channel[:, np.newaxis] - C[i]), axis=1)
        output[:,i]=C[i][index]
    output=output.reshape((x,y,c))
    output=cv2.cvtColor(output, cv2.COLOR_HSV2RGB)

    contours,_=cv2.findContours(edge,cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
    cv2.drawContours(output,contours,-1,0,thickness=1)
    #cartoon = cv2.bitwise_and(output, output, mask=contours)
    for i in range(3):
        output[:,:,i]=cv2.erode(output[:,:,i], kernel, iterations=1)
    #Laplacian = cv2.Laplacian(output,cv2.CV_8U, ksize=11)
    #output=output-Laplacian
    return output

#output=caart(cv2.imread("original.jpg"))
#cv2.imwrite("cartoon.jpg", output)
```