Project Title: Real-Time Dual-Mode Object Tracking and Pan-Tilt Control System

--- Introduction ---

This project implements a robust computer vision system designed to track objects in real time and control a 2-Degree-of-Freedom (2-DOF) Pan-Tilt servo platform. The system supports two primary tracking methods: color-based segmentation and feature-based detection. It utilizes a Proportional (P) control loop with position integration to ensure smooth, stable, and rate-limited movement of the physical platform via serial communication (PySerial).

##REPOSITORY AND VIDEO LINKS

--- Git Repository ---

URL: (https://github.com/ArmaanMeh/Vision_Pilot_OpenCV.git)

--- Demo / Tutorial Video ---

URL: [https://youtu.be/G3oHDGoSqUs?si=LIlOn546Ncgafgh-]

DEPENDENCIES AND IMPORTS

This section lists all required Python libraries and explains their role in the object tracking and control pipeline.

--- Required Imports ---

```
import cv2
import numpy as np
import serial
import time
import os

# --- Dependency Table ---
```

```
| Library | Import Name | Purpose in Project |
| :--- | :--- | :--- |
| **OpenCV** | `cv2` | Handles all image processing tasks, including camera
capture, color space conversion, contour finding, and frame visualization. |
| **NumPy** | `numpy` | Essential for efficient mathematical operations on arrays.
Used for color ranges and filter kernels. |
| **PySerial** | `serial` | Manages the physical communication link (e.g., COM8)
between the Python script and the controller hardware. |
| **Time** | `time` | Used for timing operations and implementing rate-limiting. |
| **OS** | `os` | Used for operating system interface tasks, such as handling file
paths for loading Haar Cascade XML classifiers. |


# DETECTION SETTINGS: COLOR-BASED & FEATURE-BASED TRACKING
# --- HSV Color Thresholds (Red Object) ---
```

LOWER_RED1 = np.array([0, 80, 85]) UPPER_RED1 = np.array([10, 255, 255]) LOWER_RED2 = np.array([170, 80, 85]) UPPER_RED2 = np.array([180, 255, 255])

--- Contour Size Constraints ---

```
MIN_AREA = 1500
MAX_AREA = 60000

# --- Haar Cascade Configuration (XML File Paths) ---
```

FACE_CASCADE_PATH = "Haar_Cascades XML/haarcascade_frontalface_default.xml" EYE_CASCADE_PATH = "Haar_Cascades XML/haarcascade_eye.xml" SMILE_CASCADE_PATH = "Haar_Cascades XML/haarcascade_smile.xml"

SERIAL INTERFACE AND CONTROL LOGIC FUNCTIONS

--- Function: init_serial() ---

```
def init_serial():
    """Attempts to connect to the serial port without crashing."""
    global ser
    try:
        if ser is None or not ser.is_open:
            ser = serial.Serial(SERIAL_PORT, BAUD_RATE, timeout=0.1)
            print(f"Serial connected to {SERIAL_PORT}")
            time.sleep(2)
    except serial.SerialException:
        ser = None
```

```
# --- Function: send_data_smoothed(d_pan, d_tilt) ---
```

def send_data_smoothed(d_pan, d_tilt): """ Acts as the Integrator (I-term). Accumulates velocity commands into an absolute position, clamps it, and sends the final position (-1.0 to 1.0) to the Arduino. """ global ser, last_send_time, global_pan_position, global_tilt_position

```
current_time = time.time()

if current_time - last_send_time < SEND_INTERVAL:
    return

global_pan_position += d_pan
global_tilt_position += d_tilt

global_pan_position = max(-1.0, min(1.0, global_pan_position))
global_tilt_position = max(-1.0, min(1.0, global_tilt_position))

if ser is None:
    init_serial()
    return

msg = f"{global_pan_position:.4f},{global_tilt_position:.4f}\n"
print(f"Sending POSITION: {msg.strip()} (Maps to Pan: {global_pan_position:.2f},
Tilt: {global_tilt_position:.2f})")

try:
    ser.write(msg.encode('utf-8'))
    last_send_time = current_time
except (serial.SerialException, OSError):
    ser = None
```

MAIN SETUP AND INITIALIZATION

--- Camera and Resolution Setup ---

```
cam = cv2.VideoCapture(CAMERA_ID)
cam.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
cam.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)

# --- Morphological Kernels ---
```

kernel_open = np.ones((7, 7)) kernel_close = np.ones((11, 11))

--- Haar Cascade Loading and Error Check ---

```
try:
    face_cascade = cv2.CascadeClassifier(FACE_CASCADE_PATH)
    eye_cascade = cv2.CascadeClassifier(EYE_CASCADE_PATH)
    smile_cascade = cv2.CascadeClassifier(SMILE_CASCADE_PATH)

    if face_cascade.empty() or eye_cascade.empty() or smile_cascade.empty():
        raise FileNotFoundError
except FileNotFoundError:
    print("\nOne or more Haar Cascade XML files not found.")
    print("Ensure the 'Haar_Cascades XML' folder is correct relative to the
script.")
    exit()

# --- Window and Console Output Setup ---
```

WINDOW_NAME = "Tracker Control" cv2.namedWindow(WINDOW_NAME) print(f"\nSending {SEND_RATE_HZ} msgs/sec.") print(f" Dead-Band: {DEAD_BAND_PIXELS} pixels | Smoothing: {SMOOTHING_FACTOR}") print("--- Keyboard Toggles ---") print("Press 'r' to toggle Red Ball Tracking (Servo Control).") print("Press 'f' to toggle Face Tracking (Servo Control).") print("Press 'e' for Eyes, 's' for Smile (Visual Only - Requires face detected). Press 'q' to quit.")

MAIN TRACKING LOOP: EXECUTION CONTEXT

--- Execution Block Start ---

MAIN LOOP

```
try:
    while True:
        ret, img = cam.read()
        if not ret:
            print("Camera lost.")
            break

        h, w, _ = img.shape
        center_x, center_y = w // 2, h // 2

        # Reset per-frame control inputs
        delta_pan = 0.0
        delta_tilt = 0.0
        display_color = (255, 0, 0)
```

```
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

        tracked_center_x = None
        tracked_center_y = None
        face_rect = None

# RED BALL TRACKING LOGIC (toggle_red_ball)
```

if toggle_red_ball: # Pre-processing for Red Ball blurred = cv2.GaussianBlur(img, (11, 11), 0) imgHSV = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)

```
# Red Mask Generation and Cleanup
mask1 = cv2.inRange(imgHSV, LOWER_RED1, UPPER_RED1)
mask2 = cv2.inRange(imgHSV, LOWER_RED2, UPPER_RED2)
mask = mask1 + mask2
mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel_open)
mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel_close)

# Find Contours and find largest valid Ball
contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
largest_contour = None
max_area = 0

for cnt in contours:
    area = cv2.contourArea(cnt)
    if MIN_AREA < area < MAX_AREA:
        x_bb, y_bb, cw_bb, ch_bb = cv2.boundingRect(cnt)
        aspect = float(cw_bb) / ch_bb
        if 0.5 < aspect < 1.5:
            if area > max_area:
                max_area = area
                largest_contour = cnt
                cv2.drawContours(img, [cnt], -1, (0, 255, 0), 2)

# Calculate Control Signals if ball is found
if largest_contour is not None:
    M = cv2.moments(largest_contour)
    if M["m00"] != 0:
        cx = int(M["m10"] / M["m00"])
        cy = int(M["m01"] / M["m00"])

    tracked_center_x, tracked_center_y = cx, cy

    # Visuals for Ball
    x_bb, y_bb, cw_bb, ch_bb = cv2.boundingRect(largest_contour)
    cv2.rectangle(img, (x_bb, y_bb), (x_bb+cw_bb, y_bb+ch_bb), (0, 255, 0), 2)
    cv2.circle(img, (cx, cy), 5, (0, 0, 255), -1)
    cv2.line(img, (center_x, center_y), (cx, cy), (0, 255, 255), 2)
```

FACE TRACKING LOGIC (toggle_face)

```
elif toggle_face:
    display_color = (0, 0, 255)

    # Detect faces in the grayscale image.
    faces = face_cascade.detectMultiScale(gray, 1.1, 5, minSize=(100, 100))

    largest_face = None
    max_face_area = 0

    # Find the largest face to track
    for rect in faces:
        x, y, w, h = rect
        area = w * h
        if area > max_face_area:
            max_face_area = area
            largest_face = rect

    if largest_face is not None:
        x, y, w, h = largest_face
        face_rect = largest_face

        # Calculate center of the tracked face
        cx = x + w // 2
        cy = y + h // 2

        # Store center for control
        tracked_center_x, tracked_center_y = cx, cy

        # Draw Face rectangle and text (Control Active)
        cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)
        cv2.putText(img, "Tracking Face", (x, y - 10), cv2.FONT_HERSHEY_COMPLEX,
0.8, (255, 0, 0), 2)
        cv2.circle(img, (cx, cy), 5, (255, 255, 0), -1)

# P-CONTROL CALCULATION AND EXECUTION
```

if tracked_center_x is not None:

```
# --- CALCULATE ERROR ---
error_x = tracked_center_x - center_x
error_y = tracked_center_y - center_y

pan_direction = -1
tilt_direction = -1

# --- Apply Dead-Band and P-Gain ---
if abs(error_x) > DEAD_BAND_PIXELS:
```

```
    delta_pan = error_x * P_GAIN * pan_direction

if abs(error_y) > DEAD_BAND_PIXELS:
    delta_tilt = error_y * P_GAIN * tilt_direction

# --- Clamp Max Speed ---
delta_pan = max(-MAX_SPEED_CAP, min(MAX_SPEED_CAP, delta_pan))
delta_tilt = max(-MAX_SPEED_CAP, min(MAX_SPEED_CAP, delta_tilt))
```

SUB-DETECTION AND VISUAL OVERLAYS (Eyes/Smile)

If Face Tracking wasn't active, we need to detect a face for Eyes/Smile visuals

```
if face_rect is None and (toggle_eyes or toggle_smile):
    faces = face_cascade.detectMultiScale(gray, 1.1, 5, minSize=(100, 100))
    if len(faces) > 0:
        # Find largest face for sub-detection drawing
        max_area = 0
        for rect in faces:
            x, y, w, h = rect
            area = w * h
            if area > max_area:
                max_area = area
                face_rect = rect

                # Draw a simple outline if Face Control is OFF
                if not toggle_face:
                    cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 255), 1)

if face_rect is not None:
    x, y, w, h = face_rect
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]

    # Eye Detection (Visual Only)
    if toggle_eyes:
        eyes = eye_cascade.detectMultiScale(roi_gray, 1.1, 5)
        for (ex, ey, ew, eh) in eyes:
            cv2.rectangle(roi_color, (ex, ey), (ex+ew, ey+eh), (0, 255, 255), 2)
            cv2.putText(roi_color, "Eyes", (ex, ey-5), cv2.FONT_HERSHEY_COMPLEX,
0.5, (0, 255, 255), 1)

    # Smile Detection (Visual Only)
    if toggle_smile:
        smile = smile_cascade.detectMultiScale(roi_gray,
                                               scaleFactor=1.7,
                                               minNeighbors=22,
```

```
                                                minSize=(25, 25))
        for (sx, sy, sw, sh) in smile:
            cv2.rectangle(roi_color, (sx, sy), (sx+sw, sy+sh), (255, 0, 255), 2)
            cv2.putText(roi_color, "Smile", (sx, sy + sh + 15),
cv2.FONT_HERSHEY_COMPLEX, 0.5, (255, 0, 255), 1)


# ================================================================================
# SERIAL EXECUTION AND UI UPDATES
# ================================================================================
```

send_data_smoothed(delta_pan, delta_tilt)

cv2.putText(img, f"Pan POS: {global_pan_position:.2f}", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2) cv2.putText(img, f"Tilt POS: {global_tilt_position:.2f}", (10, 60), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)

Dead-band zone

cv2.rectangle(img, (center_x - DEAD_BAND_PIXELS, center_y - DEAD_BAND_PIXELS), (center_x + DEAD_BAND_PIXELS, center_y + DEAD_BAND_PIXELS), (0, 255, 255), 1)

Serial Connection Status

```
status = "CONN" if ser is not None else "DISC"
col = (0, 255, 0) if ser is not None else (0, 0, 255)
cv2.putText(img, status, (10, h-20), cv2.FONT_HERSHEY_SIMPLEX, 0.7, col, 2)

# Toggle Status Indicators
```

y_pos = h - 20 cv2.putText(img, f"R: {'ON' if toggle_red_ball else 'OFF'}", (w - 200, y_pos), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0) if toggle_red_ball else (0, 0, 255), 2) is_face_active_control = toggle_face and not toggle_red_ball cv2.putText(img, f"F: {'ON' if toggle_face else 'OFF'}", (w - 150, y_pos), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0) if toggle_face else (0, 0, 255), 2) cv2.putText(img, f"E: {'ON' if toggle_eyes else 'OFF'}", (w - 100, y_pos), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0) if toggle_eyes else (0, 0, 255), 2) cv2.putText(img, f"S: {'ON' if toggle_smile else 'OFF'}", (w - 50, y_pos), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0) if toggle_smile else (0, 0, 255), 2)

KEYBOARD INPUT HANDLING AND CLEANUP

--- KEYBOARD INPUT HANDLING ---

```
key = cv2.waitKey(10) & 0xFF
```

```
    if key == ord('q'):
        break

    elif key == ord('r'):
        toggle_red_ball = not toggle_red_ball
        if toggle_red_ball:
            toggle_face = False
            print("Red Ball TRACKING (Control) is now ON (Face Control OFF).")
        else:
            print("Red Ball TRACKING (Control) is now OFF.")

    elif key == ord('f'):
        toggle_face = not toggle_face
        if toggle_face:
            toggle_red_ball = False
            print(f"Face TRACKING (Control) is now ON (Red Ball OFF).")
        else:
            print(f"Face TRACKING (Control) is now OFF.")

    elif key == ord('e'):
        toggle_eyes = not toggle_eyes
        print(f"Eye detection is now {'ON' if toggle_eyes else 'OFF'}")

    elif key == ord('s'):
        toggle_smile = not toggle_smile
        print(f"Smile detection is now {'ON' if toggle_smile else 'OFF'}")

    cv2.imshow(WINDOW_NAME, img)

    # --- CLEANUP ---
```

except KeyboardInterrupt: pass finally: cam.release() if ser is not None: ser.close() cv2.destroyAllWindows()