#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Thu Dec 19 17:59:40 2024

@author: rowanbrown

"""

def detect\_rugby\_ball(frame):

"""

Detect rugby ball using color and shape characteristics

"""

# Convert to HSV color space

hsv = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)

# Define white/off-white color ranges for rugby ball

# Multiple ranges to handle different lighting conditions

white\_ranges = [

# Bright white

(np.array([0, 0, 200]), np.array([180, 30, 255])),

# Off-white/cream

(np.array([0, 0, 160]), np.array([180, 40, 220])),

# Shadowed white

(np.array([0, 0, 140]), np.array([180, 50, 200]))

]

# Combined mask for all white ranges

combined\_mask = np.zeros(hsv.shape[:2], dtype=np.uint8)

for lower, upper in white\_ranges:

mask = cv2.inRange(hsv, lower, upper)

combined\_mask = cv2.bitwise\_or(combined\_mask, mask)

# Noise reduction

kernel = np.ones((5,5), np.uint8)

combined\_mask = cv2.morphologyEx(combined\_mask, cv2.MORPH\_OPEN, kernel)

combined\_mask = cv2.morphologyEx(combined\_mask, cv2.MORPH\_CLOSE, kernel)

# Find contours

contours, \_ = cv2.findContours(combined\_mask, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

ball\_candidates = []

for contour in contours:

area = cv2.contourArea(contour)

# Filter by area (adjust these values based on your video)

if 100 < area < 2000:

# Fit ellipse to contour

if len(contour) >= 5: # Need at least 5 points to fit ellipse

ellipse = cv2.fitEllipse(contour)

(x, y), (MA, ma), angle = ellipse

# Calculate aspect ratio

aspect\_ratio = max(MA, ma) / min(MA, ma)

# Rugby balls typically have aspect ratio around 2:1

if 1.5 < aspect\_ratio < 2.5:

ball\_candidates.append({

'center': (int(x), int(y)),

'ellipse': ellipse,

'area': area,

'aspect\_ratio': aspect\_ratio

})

# Return best candidate based on aspect ratio closest to 2:1

if ball\_candidates:

best\_candidate = min(ball\_candidates, key=lambda x: abs(x['aspect\_ratio'] - 2.0))

return best\_candidate

return None

def identify\_ball\_carrier(ball\_position, keypoints, frame):

"""

Identify ball carrier based on proximity to detected ball

"""

if ball\_position is None or len(keypoints) < 2:

return None

# Extract player positions (using torso center)

player\_positions = []

for kp in keypoints:

pose = kp.xy[0].cpu().numpy()

# Calculate torso center using shoulders and hips

torso\_center = np.mean([

pose[5:7], # shoulders

pose[11:13] # hips

], axis=0)

player\_positions.append({

'center': torso\_center.mean(axis=0),

'pose': pose

})

# Find closest player to ball

distances = [np.linalg.norm(ball\_position['center'] - p['center']) for p in player\_positions]

closest\_idx = np.argmin(distances)

return player\_positions[closest\_idx]

def draw\_ball\_detection(frame, ball\_candidate, ball\_carrier=None):

"""

Visualize ball detection and carrier identification

"""

if ball\_candidate is not None:

# Draw ellipse around ball

cv2.ellipse(frame, ball\_candidate['ellipse'], (0, 255, 255), 2)

# Draw ball center

cv2.circle(frame, ball\_candidate['center'], 4, (0, 255, 255), -1)

# Add ball annotation

cv2.putText(frame,

f"Ball (AR: {ball\_candidate['aspect\_ratio']:.2f})",

(ball\_candidate['center'][0] + 10, ball\_candidate['center'][1]),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 255), 2)

if ball\_carrier is not None:

# Draw line from ball to carrier

cv2.line(frame,

ball\_candidate['center'],

tuple(map(int, ball\_carrier['center'])),

(0, 255, 0), 2)

# Highlight ball carrier

cv2.circle(frame,

tuple(map(int, ball\_carrier['center'])),

15, (0, 255, 0), 2)

cv2.putText(frame,

"Ball Carrier",

tuple(map(int, ball\_carrier['center'] + np.array([20, 0]))),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 255, 0), 2)

return frame

# Update the main processing loop to include ball detection:

def process\_frame(frame, keypoints):

"""

Process a single frame for ball and player detection

"""

# Detect ball

ball = detect\_rugby\_ball(frame)

# Identify ball carrier if ball is detected

ball\_carrier = None

if ball is not None:

ball\_carrier = identify\_ball\_carrier(ball, keypoints, frame)

# Draw detections

frame = draw\_ball\_detection(frame, ball, ball\_carrier)

return frame, ball\_carrier