Playing Chrome Dinosaur Game With Computer Vision._Hand Posture

This code is a hand gesture recognition system that uses OpenCV and PyAutoGUI to simulate keyboard inputs to control a game. Below is a step-by-step explanation:

1. Import Required Libraries

```
import numpy as np
import cv2
import math
import pyautogui
```

NumPy: Used for numerical operations.
OpenCV (cv2): Used for image processing and computer vision tasks.
Math: Used for mathematical computations.
PyAutoGUI: Used to simulate keyboard key presses.

2. Initialize Camera

```
capture = cv2.VideoCapture(0)
```

Opens the default camera (0 is the default camera index).
Captures video from the webcam.

3. Start Video Capture and Process Each Frame

```
while capture.isOpened():
    ret, frame = capture.read()
```

capture.read(): Reads frames from the webcam.
ret: Boolean value to check if the frame was captured successfully.
frame: Stores the current frame.

4. Define Region of Interest (ROI)

```
cv2.rectangle(frame, (100, 100), (300, 300), (0, 255, 0), 0)
crop_image = frame[100:300, 100:300]
```

A green rectangle is drawn on the screen to define the hand detection area.
The part of the frame inside this rectangle is extracted as crop_image.

5. Apply Preprocessing

```
blur = cv2.GaussianBlur(crop_image, (3, 3), 0)
hsv = cv2.cvtColor(blur, cv2.COLOR_BGR2HSV)
mask2 = cv2.inRange(hsv, np.array([2, 0, 0]), np.array([20, 255, 255]))
```

Gaussian Blur reduces noise in the image.
Color Space Conversion (BGR to HSV) helps in better segmentation of skin color.
Thresholding (cv2.inRange) creates a binary mask where hand regions are white (255) and others are black (0).

6. Apply Morphological Transformations

```
kernel = np.ones((5, 5))
dilation = cv2.dilate(mask2, kernel, iterations=1)
erosion = cv2.erode(dilation, kernel, iterations=1)
filtered = cv2.GaussianBlur(erosion, (3, 3), 0)
ret, thresh = cv2.threshold(filtered, 127, 255, 0)
```

Dilation & Erosion remove noise and small unwanted regions.

Thresholding converts the image to binary format.

7. Find Contours

```
contours, hierachy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

Finds the edges (contours) of objects in the binary image.
Contours are the boundaries of detected hand shapes.

8. Identify Hand Contour & Convex Hull

```
contour = max(contours, key=lambda x: cv2.contourArea(x))
x, y, w, h = cv2.boundingRect(contour)
cv2.rectangle(crop_image, (x, y), (x + w, y + h), (0, 0, 255), 0)
hull = cv2.convexHull(contour)
cv2.drawContours(drawing, [contour], -1, (0, 255, 0), 0)
cv2.drawContours(drawing, [hull], -1, (0, 0, 255), 0)
```

Finds the largest contour (hand) and draws a bounding rectangle.
Computes the convex hull, which forms an outer boundary around the hand.

9. Detect Convexity Defects (Finger Gaps)

```
hull = cv2.convexHull(contour, returnPoints=False)
defects = cv2.convexityDefects(contour, hull)
```

Identifies concave defects (spaces between fingers).

10. Count Fingers

```
count_defects = 0
for i in range(defects.shape[0]):
    s, e, f, d = defects[i, 0]
    start = tuple(contour[s][0])
    end = tuple(contour[e][0])
    far = tuple(contour[f][0])

    a = math.sqrt((end[0] - start[0]) ** 2 + (end[1] - start[1]) ** 2)
    b = math.sqrt((far[0] - start[0]) ** 2 + (far[1] - start[1]) ** 2)
    c = math.sqrt((end[0] - far[0]) ** 2 + (end[1] - far[1]) ** 2)
    angle = (math.acos((b ** 2 + c ** 2 - a ** 2) / (2 * b * c)) * 180) / 3.14

    if angle <= 90:
        count_defects += 1
        cv2.circle(crop_image, far, 1, [0, 0, 255], -1)

    cv2.line(crop_image, start, end, [0, 255, 0], 2)
```

Uses cosine rule to detect angles between fingers.
Counts how many gaps (defects) exist between fingers.

11. Perform Actions Based on Finger Count

```
if count_defects >= 4:
    pyautogui.press('space')
    cv2.putText(frame, "JUMP", (115, 80), cv2.FONT_HERSHEY_SIMPLEX, 2, 2, 2)
```

If 4 or more fingers are detected, the program simulates a SPACE key press (useful for jumping in games).

12. (Optional) Simulate Racing Game Controls

```
if count_defects == 1:
    pyautogui.press('w')
if count_defects == 2:
    pyautogui.press('s')
if count_defects == 3:
    pyautogui.press('aw')
if count_defects == 4:
    pyautogui.press('dw')
if count_defects == 5:
    pyautogui.press('s')
```

Detects different numbers of fingers and maps them to WASD keys (for racing games).

13. Display Output

```
cv2.imshow("Gesture", frame)
if cv2.waitKey(1) == ord('q'):
    break
```

Displays the video feed with gestures detected.
Press 'q' to exit the program.