Human pose recognition

Routine Experiment Effect

In this section, we will learn how to use K230 to realize the function of human key point detection.

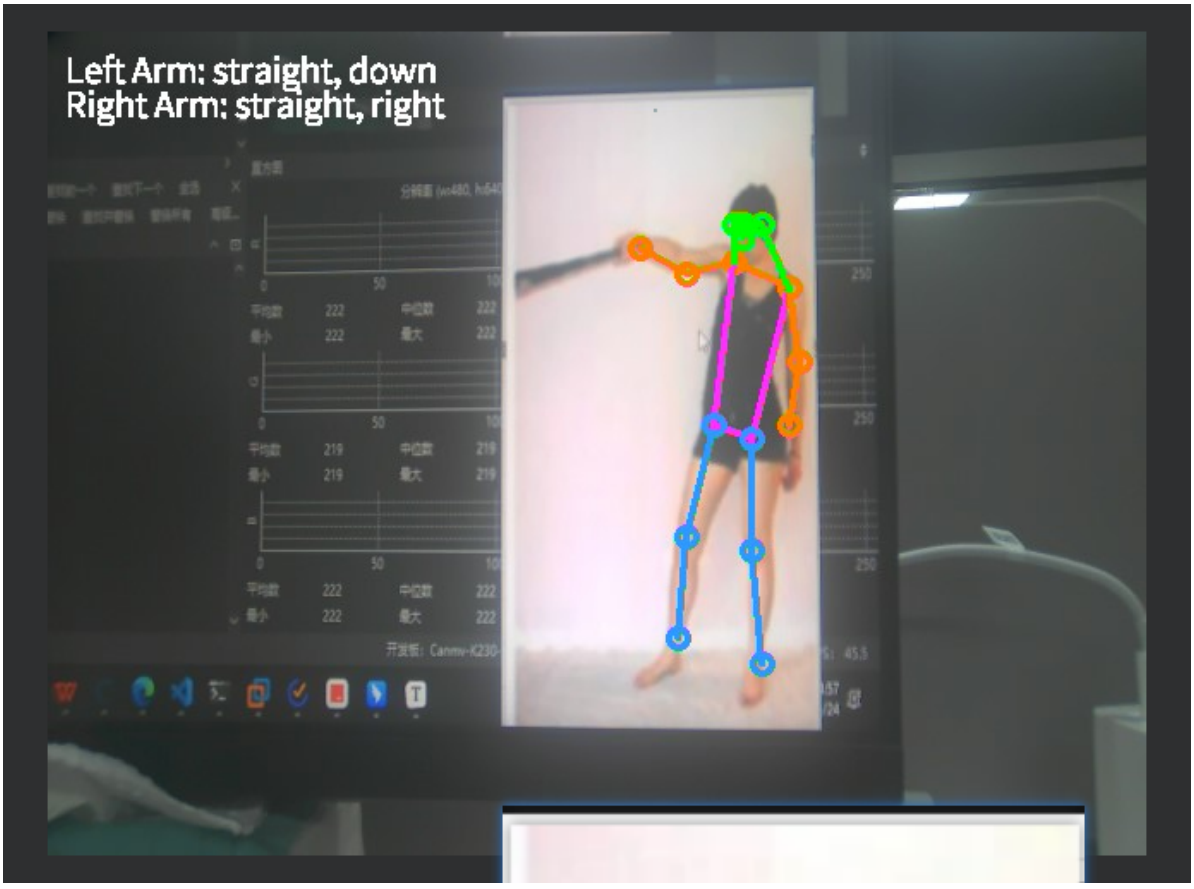
The example code is in [Source code/08.Body/03.person_gesture_cls.py]

After connecting to the IDE, run the sample code in this section. Use K230 to aim at a picture with a human body. You can see the key points of the human body marked on the screen.

By comparing the relative positions of these (arm) key points, the state of the arm is judged:

> States include straight, bent, left, and right.

The judgment result will be displayed in the upper left corner of the screen



Code Explanation

We mainly analyze the ideas of judging posture

This code implements the human posture analysis function, mainly analyzing the state (straight/bent) and orientation of the arm. Let me explain it in detail in modules:

1. analyze_pose Function

```
def analyze_pose(self, kpses):
```

This is the main analysis function, handling posture analysis of multiple people:

- The input parameter kpses contains the key point data of multiple people in the format of [number of people, number of key points, 3]
- Each key point contains three values: (x, y, confidence)
- Analyze the left and right arms separately for each detected person
- Call _analyze_arm to process the left arm (key points 5, 7, and 9) and the right arm (key points 6, 8, and 10) respectively.

1. _analyze_arm function

```
def _analyze_arm(self, shoulder, elbow, wrist):
```

This is the core analysis function, which analyzes a single arm by following these steps:

A. Validity Check:

- Check if the confidence of the three key points (shoulder, elbow, wrist) is > 0
- If invalid, return "unknown" status

B. Angle calculation:

- Calculate the shoulder-to-elbow and elbow-to-wrist vectors
- Calculate the angle between two vectors using the vector dot product
- Get the angle value through arccos

C. Status judgment:

- Angle <= 60 degrees is considered "straight"
- Angle > 60 degrees is considered "bent"

D. Direction judgment:

- Compare the x and y components of the shoulder to elbow vector
- If the horizontal component is large, judge left/right
- If the vertical component is large, it determines up/down

1. draw_result function

```
def draw_result(self, pl, res, pose_results):
```

Here is the visualization function:

- Draw bone key point connection lines
- Display pose analysis results above each detected person
- Includes coordinate transformations to ensure correct display

Key technologies:

1. Vector computing: vector operations using numpy
2. Angle calculation: use dot product and arccos to calculate the angle

3. Posture determination: based on angle threshold and vector projection
4. Coordinate transformation: coordinate mapping considering display size

Core code

```
    def analyze_pose(self, kpses):
        """
        分析人体姿势，判断手臂的伸直/弯曲状态及朝向
        Analyze human pose, determine arm straight/bent state and direction

        Args:
            kpses: 关键点数据，格式为 [num_persons, num_keypoints, 3]，每个关键点包含 (x, y, confidence)
                   (Keypoint data in format [num_persons, num_keypoints, 3], each keypoint contains (x, y, confidence))
        Returns:
            pose_results: 包含每个人的姿势分析结果的列表 (List containing pose analysis results for each person)
        """
        pose_results = []
        for i in range(len(kpses)):  # 遍历每个检测到的人 (Iterate through each detected person)
            person_kps = kpses[i]
            person_pose = {}

            # 左臂分析（关键点 5: 左肩, 7: 左肘, 9: 左手腕）
            # Left arm analysis (keypoint 5: left shoulder, 7: left elbow, 9: left wrist)
            left_arm_result = self._analyze_arm(
                shoulder=person_kps[5],  # 关键点索引从1开始，数组从0开始 (Keypoint index starts from 1, array starts from 0)
                elbow=person_kps[7],
                wrist=person_kps[9]
            )
            person_pose["left_arm"] = left_arm_result

            # 右臂分析（关键点 6: 右肩, 8: 右肘, 10: 右手腕）
            # Right arm analysis (keypoint 6: right shoulder, 8: right elbow, 10: right wrist)
            right_arm_result = self._analyze_arm(
                shoulder=person_kps[6],
                elbow=person_kps[8],
                wrist=person_kps[10]
            )
            person_pose["right_arm"] = right_arm_result

            pose_results.append(person_pose)

        return pose_results

    def _analyze_arm(self, shoulder, elbow, wrist):
        """
        分析单条手臂的姿势
        Analyze the pose of a single arm
```

```
        Args:
            shoulder: 肩关节坐标 (x, y, confidence) (Shoulder joint coordinates
(x, y, confidence))
            elbow: 肘关节坐标 (x, y, confidence) (Elbow joint coordinates (x, y,
confidence))
            wrist: 手腕坐标 (x, y, confidence) (Wrist coordinates (x, y,
confidence))
        Returns:
            result: 包含伸直/弯曲状态和朝向的字典 (Dictionary containing
straight/bent state and direction)
        """
        result = {"state": "unknown", "direction": "unknown"}

        # 检查关键点是否有效（置信度 > 0）(Check if keypoints are valid (confidence >
0))
        if shoulder[2] <= 0 or elbow[2] <= 0 or wrist[2] <= 0:
            return result

        # 计算肩-肘向量和肘-腕向量 (Calculate shoulder-elbow vector and elbow-wrist
vector)
        vec_shoulder_to_elbow = np.array([elbow[0] - shoulder[0], elbow[1] -
shoulder[1]])
        vec_elbow_to_wrist = np.array([wrist[0] - elbow[0], wrist[1] -
elbow[1]])

        # 计算向量模长 (Calculate vector magnitudes)
        norm_se = np.linalg.norm(vec_shoulder_to_elbow)
        norm_ew = np.linalg.norm(vec_elbow_to_wrist)

        # 避免除零 (Avoid division by zero)
        if norm_se == 0 or norm_ew == 0:
            return result

        # 计算夹角（角度）(Calculate angle (in degrees))
        cos_theta = np.dot(vec_shoulder_to_elbow, vec_elbow_to_wrist) / (norm_se
* norm_ew)
        cos_theta = min(1.0, max(-1.0, cos_theta))  # 限制cos值范围 (Restrict cos
value range)
        angle = math.degrees(math.acos(cos_theta))

        # 判断手臂伸直或弯曲 (Determine if arm is straight or bent)
        if angle <= 60:
            result["state"] = "straight"
        elif angle > 60:
            result["state"] = "bent"

        # 判断手臂朝向 (Determine arm direction)
        dx = elbow[0] - shoulder[0]
        dy = elbow[1] - shoulder[1]
        if abs(dx) > abs(dy):
            # 水平方向为主 (Primarily horizontal direction)
            result["direction"] = "left" if dx > 0 else "right"
        else:
            # 垂直方向为主 (Primarily vertical direction)
            result["direction"] = "down" if dy > 0 else "up"

        return result
```

```
    def draw_result(self, pl, res, pose_results=None):
        """
        绘制结果，包括骨骼、关键点和姿势信息
        Draw results including skeleton, keypoints and pose information

        Args:
            pl: PipeLine实例 (PipeLine instance)
            res: 检测结果 (Detection results)
            pose_results: 姿势分析结果 (Pose analysis results)
        """
        with ScopedTiming("display_draw", self.debug_mode > 0):
            if res[0]:  # 如果检测到人 (If persons are detected)
                pl.osd_img.clear()
                kpses = res[1]
                for i in range(len(res[0])):
                    # 绘制关键点和骨骼（原有逻辑）(Draw keypoints and skeleton
(original logic))
                    for k in range(17+2):
                        if k < 17:
                            kps_x, kps_y, kps_s = round(kpses[i][k][0]),
round(kpses[i][k][1]), kpses[i][k][2]
                            kps_x1 = int(float(kps_x) * self.display_size[0] //
self.rgb888p_size[0])
                            kps_y1 = int(float(kps_y) * self.display_size[1] //
self.rgb888p_size[1])
                            if kps_s > 0:
                                pl.osd_img.draw_circle(kps_x1, kps_y1, 5,
self.KPS_COLORS[k], 4)

                        ske = self.SKELETON[k]
                        pos1_x, pos1_y = round(kpses[i][ske[0]-1][0]),
round(kpses[i][ske[0]-1][1])
                        pos1_x_ = int(float(pos1_x) * self.display_size[0] //
self.rgb888p_size[0])
                        pos1_y_ = int(float(pos1_y) * self.display_size[1] //
self.rgb888p_size[1])

                        pos2_x, pos2_y = round(kpses[i][(ske[1]-1)][0]),
round(kpses[i][(ske[1]-1)][1])
                        pos2_x_ = int(float(pos2_x) * self.display_size[0] //
self.rgb888p_size[0])
                        pos2_y_ = int(float(pos2_y) * self.display_size[1] //
self.rgb888p_size[1])

                        pos1_s, pos2_s = kpses[i][(ske[0]-1)][2], kpses[i]
[(ske[1]-1)][2]
                        if pos1_s > 0.0 and pos2_s > 0.0:
                            pl.osd_img.draw_line(pos1_x_, pos1_y_, pos2_x_,
pos2_y_, self.LIMB_COLORS[k], 4)

                    # 绘制姿势信息 (Draw pose information)
                    if pose_results and i < len(pose_results):
                        pose = pose_results[i]
                        # 显示左臂信息 (Display left arm information)
                        left_arm_text = f"Left Arm: {pose['left_arm']['state']},
{pose['left_arm']['direction']}"
                        pl.osd_img.draw_string_advanced(
```

```
                            10, 10 + i * 60, 20, left_arm_text, (255, 255, 255,
255)
                        )
                        # 显示右臂信息 (Display right arm information)
                        right_arm_text = f"Right Arm: {pose['right_arm']
['state']}, {pose['right_arm']['direction']}"
                        pl.osd_img.draw_string_advanced(
                            10, 30 + i * 60, 20, right_arm_text, (255, 255, 255,
255)
                        )
                gc.collect()
            else:
                pl.osd_img.clear()
```