Posture detection

1、synopsis

MediaPipe is a data stream processing machine learning application development framework developed and open-source by Google. It is a graph based data processing pipeline that enables the construction of various forms of data sources, such as video, frequency, sensor data, and any time series data. MediaPipe is cross platform and can run on embedded platforms (such as Raspberry Pi), mobile devices (iOS and Android), workstations, and servers, while supporting mobile GPU acceleration. MediaPipe provides cross platform, customizable ML solutions for real-time and streaming media. The core framework of MediaPipe is implemented in C++and provides support for languages such as Java and Objective C. The main concepts of MediaPipe include Packets, Streams, Calculators, Graphs, and Subgraphs.

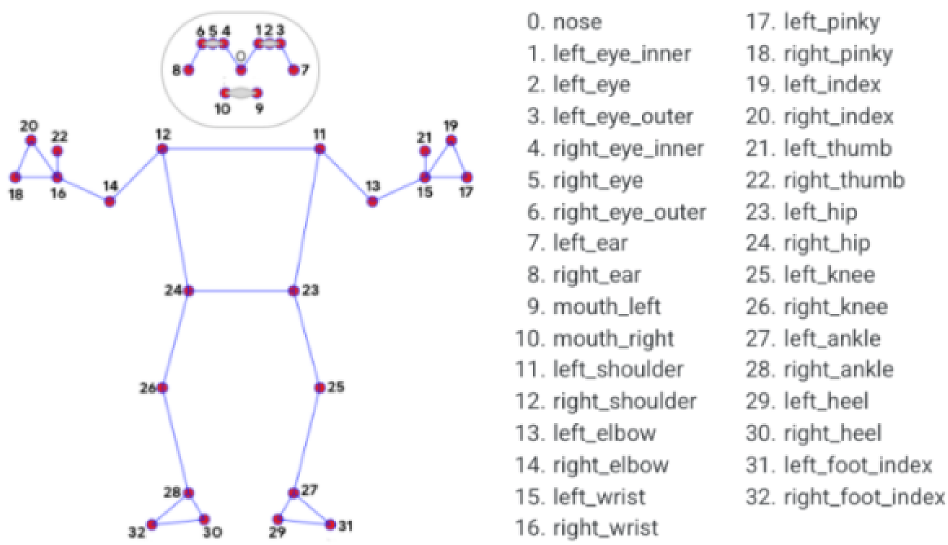
The characteristics of MediaPipe：

- End to end acceleration: Built in fast ML inference and processing that can accelerate even on regular hardware。
- Build once, deploy anytime, anywhere: Unified solution suitable for Android, iOS, desktop/cloud, web, and IoT。
- Instant solution: A cutting-edge ML solution that showcases all features of the framework。
- Free and open source: Framework and solution under Apache 2.0, fully scalable and customizable.

2、MediaPipe Pose

MediaPipe Pose, an ML solution for high-fidelity body pose tracking, uses BlazePose research to infer 33 3D coordinates and full-body background segmentation masks from RGB video frames, which also powers the ML Kit pose detection API.

The Landmark model in MediaPipe postures predicts the position of 33 postural coordinates (see figure below).



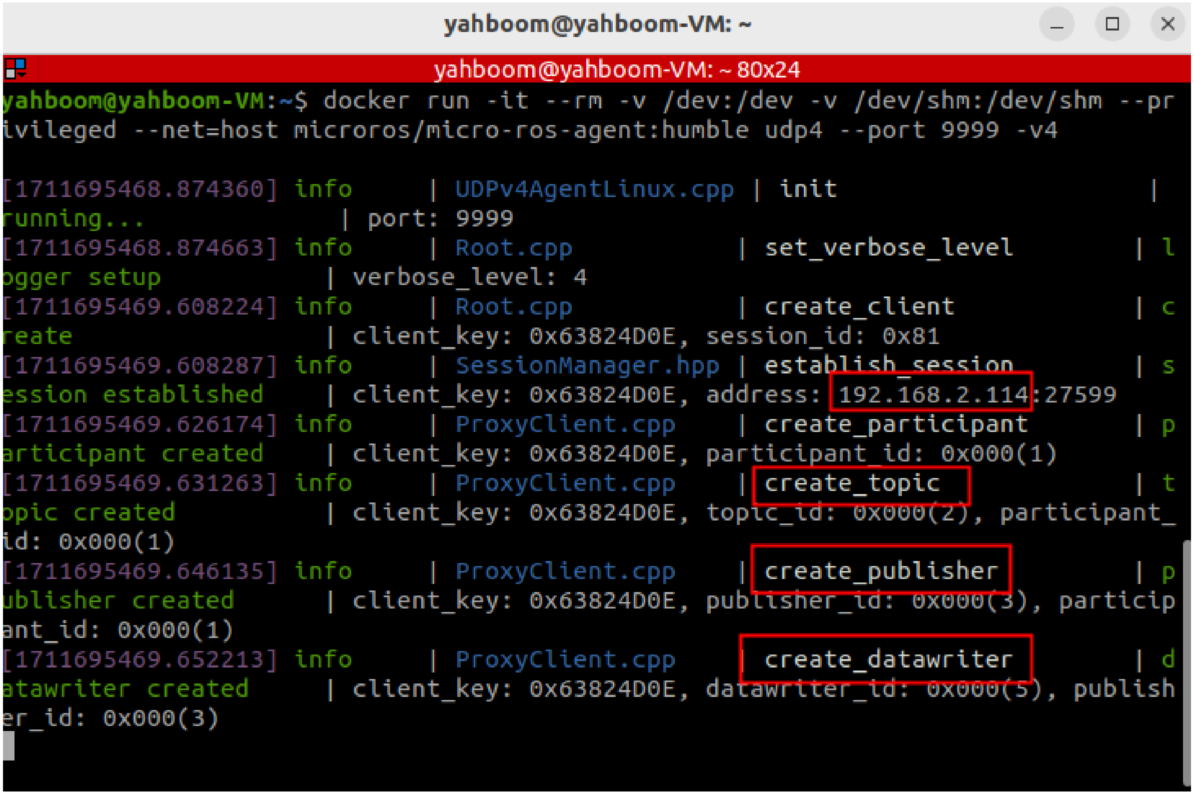
3、Attitude detection

3.1、activate

1. First, set the proxy IP for the ROS-wifi image transmission module. For specific steps, please refer to the tutorial **1. Basic Usage of ROS-Wifi Image Transmission Module on Micros Car.** This tutorial will not cover this process in detail.
2. The Linux system is connected to the ROS-wifi image transfer module, and the Docker is started. Enter the following command to establish a connection with the ROS-wifi image transfer module.

```
#Use the provided system for direct input
sh start_Camera_computer.sh
```

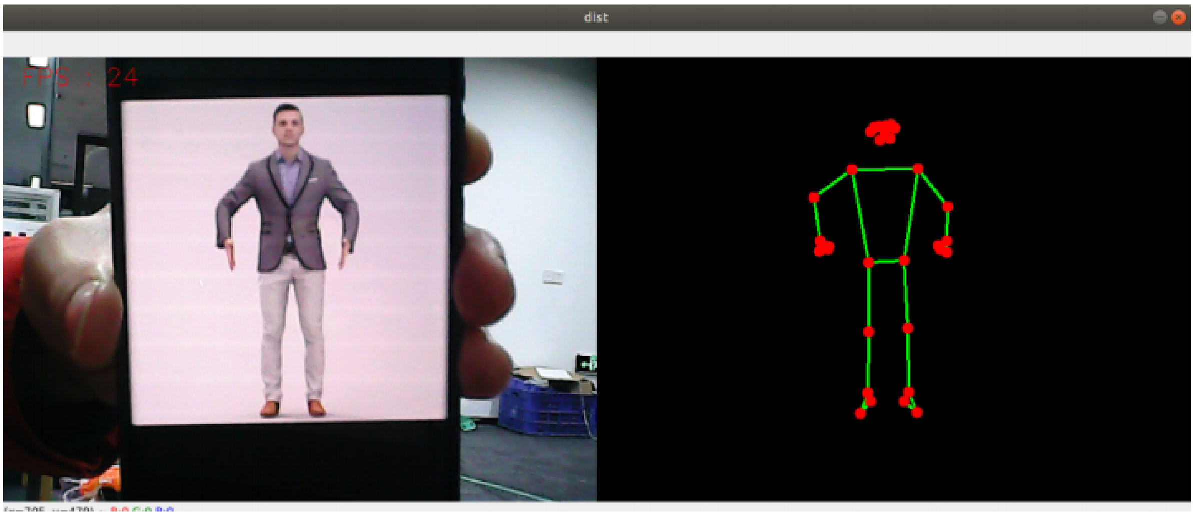
```
#Systems that are not data:
docker run -it --rm -v /dev:/dev -v /dev/shm:/dev/shm --privileged --net=host
microros/micro-ros-agent:humble udp4 --port 9999 -v4
```



If the preceding information is displayed, the proxy connection is successful

3. Open a new terminal and execute the following command

```
ros2 run yahboom_esp32_mediapipe 02_PoseDetector
```



4. If the camera picture is upside down, see **3. Camera picture correction (must-read)** tutorial, this tutorial is no longer explained

3.2、Code parsing

The location of the function source code is located,

```
~/yahboomcar_ws/src/yahboom_esp32_mediapipe/yahboom_esp32_mediapipe/02_PoseDetector.py
```

```
class PoseDetector(Node):
    def __init__(self, name,mode=False, smooth=True, detectionCon=0.5,
trackCon=0.5):
        super().__init__(name)
        self.mpPose = mp.solutions.pose
        self.mpDraw = mp.solutions.drawing_utils
        self.pose = self.mpPose.Pose(
            static_image_mode=mode,
            smooth_landmarks=smooth,
            min_detection_confidence=detectionCon,
            min_tracking_confidence=trackCon )
        self.pub_point =
self.create_publisher(PointArray,'/mediapipe/points',1000)
        self.lmDrawSpec = mp.solutions.drawing_utils.DrawingSpec(color=(0, 0,
255), thickness=-1, circle_radius=6)
        self.drawSpec = mp.solutions.drawing_utils.DrawingSpec(color=(0, 255,
0), thickness=2, circle_radius=2)

    def pubPosePoint(self, frame, draw=True):
        pointArray = PointArray()
        img = np.zeros(frame.shape, np.uint8)
        img_RGB = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
        self.results = self.pose.process(img_RGB)
        if self.results.pose_landmarks:
            if draw: self.mpDraw.draw_landmarks(frame,
self.results.pose_landmarks, self.mpPose.POSE_CONNECTIONS, self.lmDrawSpec,
self.drawSpec)
            self.mpDraw.draw_landmarks(img, self.results.pose_landmarks,
self.mpPose.POSE_CONNECTIONS, self.lmDrawSpec, self.drawSpec)
            for id, lm in enumerate(self.results.pose_landmarks.landmark):
```

```
                point = Point()
                point.x, point.y, point.z = lm.x, lm.y, lm.z
                pointArray.points.append(point)
        self.pub_point.publish(pointArray)
        return frame, img

    def frame_combine(slef,frame, src):
        if len(frame.shape) == 3:
            frameH, frameW = frame.shape[:2]
            srcH, srcW = src.shape[:2]
            dst = np.zeros((max(frameH, srcH), frameW + srcW, 3), np.uint8)
            dst[:, :frameW] = frame[:, :]
            dst[:, frameW:] = src[:, :]
        else:
            src = cv.cvtColor(src, cv.COLOR_BGR2GRAY)
            frameH, frameW = frame.shape[:2]
            imgH, imgW = src.shape[:2]
            dst = np.zeros((frameH, frameW + imgW), np.uint8)
            dst[:, :frameW] = frame[:, :]
            dst[:, frameW:] = src[:, :]
        return dst

class MY_Picture(Node):
    def __init__(self, name):
        super().__init__(name)
        self.bridge = CvBridge()
        self.sub_img = self.create_subscription(
            CompressedImage, '/espRos/esp32camera', self.handleTopic, 1) #Get
the image from the esp32

        self.pose_detector = PoseDetector('pose_detector')

    def handleTopic(self, msg):
        start = time.time()

        frame = self.bridge.compressed_imgmsg_to_cv2(msg)
        frame = cv.resize(frame, (640, 480))
        cv.waitKey(10)
        frame, img = self.pose_detector.pubPosePoint(frame,draw=False)

        end = time.time()
        fps = 1 / (end - start)
        text = "FPS : " + str(int(fps))
        cv.putText(frame, text, (20, 30), cv.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0,
255), 1)

        dist = self.pose_detector.frame_combine(frame, img)
        cv.imshow('dist', dist)
        # print(frame)

        cv.waitKey(10)

def main():
    print("start it")
    rclpy.init()
```

```
    esp_img = MY_Picture("My_Picture")
    try:
            rclpy.spin(esp_img)
    except KeyboardInterrupt:
        pass
    finally:
        esp_img.destroy_node()
        rclpy.shutdown()
```

The main process of the program: subscribe to the image from esp32, through MediaPipe to do the relevant recognition, and then through opencv to display the processed image.