Hand 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：

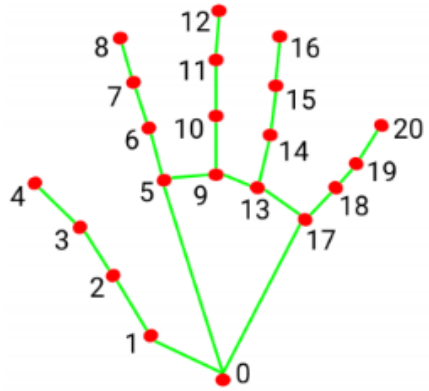
- End to end acceleration: Built in fast ML inference and processing that can accelerate even on regular hardware。
- Build once, deploy anytime, anywhere: Unified solution suitable for Android, iOS, desktop/cloud, web, and IoT。
- Instant solution: A cutting-edge ML solution that showcases all features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

2、MediaPipe Hands

MediaPipe Hands is a high-fidelity hand and finger tracking solution. It utilizes machine learning (ML) to infer the 3D coordinates of 21 hand joints from a single frame.

After detecting the entire hand in the image, precise keypoint localization of the 21 3D hand joint coordinates within the detected hand region is performed through regression using a hand landmark model. This involves direct coordinate prediction. The model learns a consistent internal hand pose representation and even exhibits robustness to partially visible hands and self-occlusion.

In order to obtain real-world ground truth data, approximately 30,000 real-world images were manually annotated with 21 3D hand coordinates as shown below (obtaining Z values from the depth map of each image if corresponding coordinates have Z values). To better cover possible hand poses and provide additional supervision on the nature of hand articulation, high-quality synthetic hand models in various backgrounds were also created and mapped to their respective 3D coordinates.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

3、Hand detection

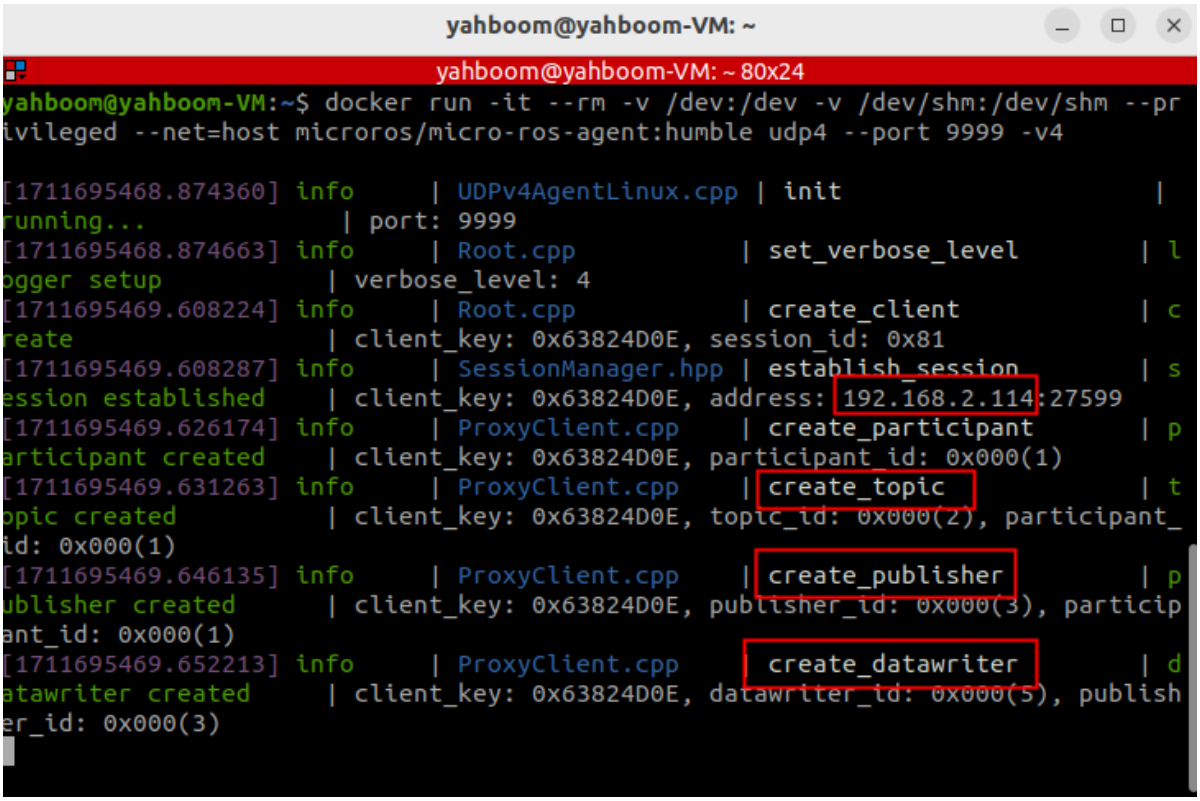
3.1、start

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
```

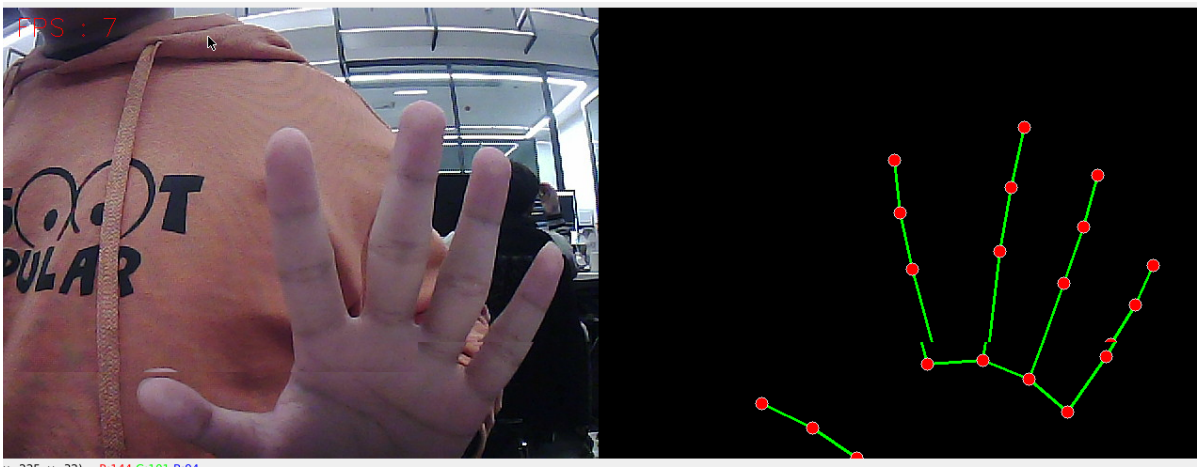
```
#Systems that are not data:
docker run -it --rm -v /dev:/dev -v /dev/shm:/dev/shm --privileged --net=host
microros/micro-ros-agent:humble udp4 --port 9999 -v4
```



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 01_HandDetector
```



4. Notes

- "If the camera angle is not centered, the Linux system needs to connect to the car agent. Please refer to the specific steps for development preparation before proceeding with tutorial **0. which provides essential information for a quick start (must-read)**. This tutorial will not provide further elaboration."
- After connecting the car's agent, enter the following command

```
ros2 run yahboom_esp32_mediapipe control_servo
```

After the steering engine is in the center, press "ctrl+C" to end the program.

5. 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/01_HandDetect
or.py
```

```
class HandDetector(Node):
    def __init__(self, name, mode=False, maxHands=2, detectorCon=0.5,
trackCon=0.5):
        super().__init__(name)
        self.mpHand = mp.solutions.hands
        self.mpDraw = mp.solutions.drawing_utils
        self.hands = self.mpHand.Hands(
            static_image_mode=mode,
            max_num_hands=maxHands,
            min_detection_confidence=detectorCon,
            min_tracking_confidence=trackCon)
        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)
        # create a publisher
        self.pub_point = self.create_publisher(
            PointArray, '/mediapipe/points', 1000)

    def pubHandsPoint(self, frame, draw=True):
        pointArray = PointArray()
        img = np.zeros(frame.shape, np.uint8)
        img_RGB = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
        self.results = self.hands.process(img_RGB)
        if self.results.multi_hand_landmarks:
            for i in range(len(self.results.multi_hand_landmarks)):
                if draw:
                    self.mpDraw.draw_landmarks(
                        frame, self.results.multi_hand_landmarks[i],
self.mpHand.HAND_CONNECTIONS, self.lmDrawSpec, self.drawSpec)
                self.mpDraw.draw_landmarks(
                    img, self.results.multi_hand_landmarks[i],
self.mpHand.HAND_CONNECTIONS, self.lmDrawSpec, self.drawSpec)
                for id, lm in
enumerate(self.results.multi_hand_landmarks[i].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) #获取
esp32传来的图像

        self.hand_detector = HandDetector('hand_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.hand_detector.pubHandsPoint(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.hand_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.