7. 3D object recognition

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、Three-dimensional object recognition

3D object recognition: Recognisable objects have：['Shoe', 'Chair', 'Cup', 'Camera'],There are four categories; Click [f key] to switch the recognized object.

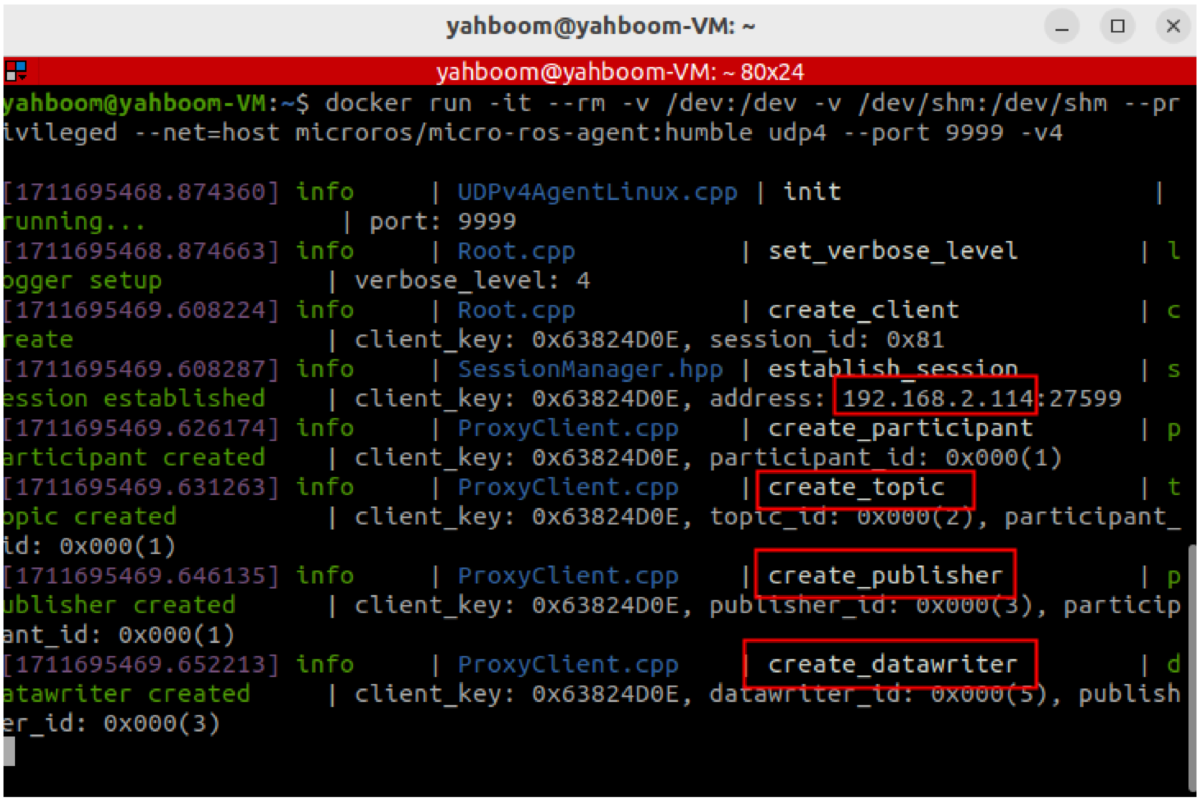
Note: The f key can switch objects only when the Raspberry PI is connected to the Internet, because the pre-training model needs to be connected to the Internet.

2.1、activate

1. First set the proxy IP for the ROS-wifi image transfer module, for specific steps, please see the basic use of **1. The use of ROS-wifi image transfer module in micros car** tutorial, this tutorial will not be elaborated.
2. Linux system connects to ROS-wifi image transfer module, start docket, enter the following command to connect 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
```



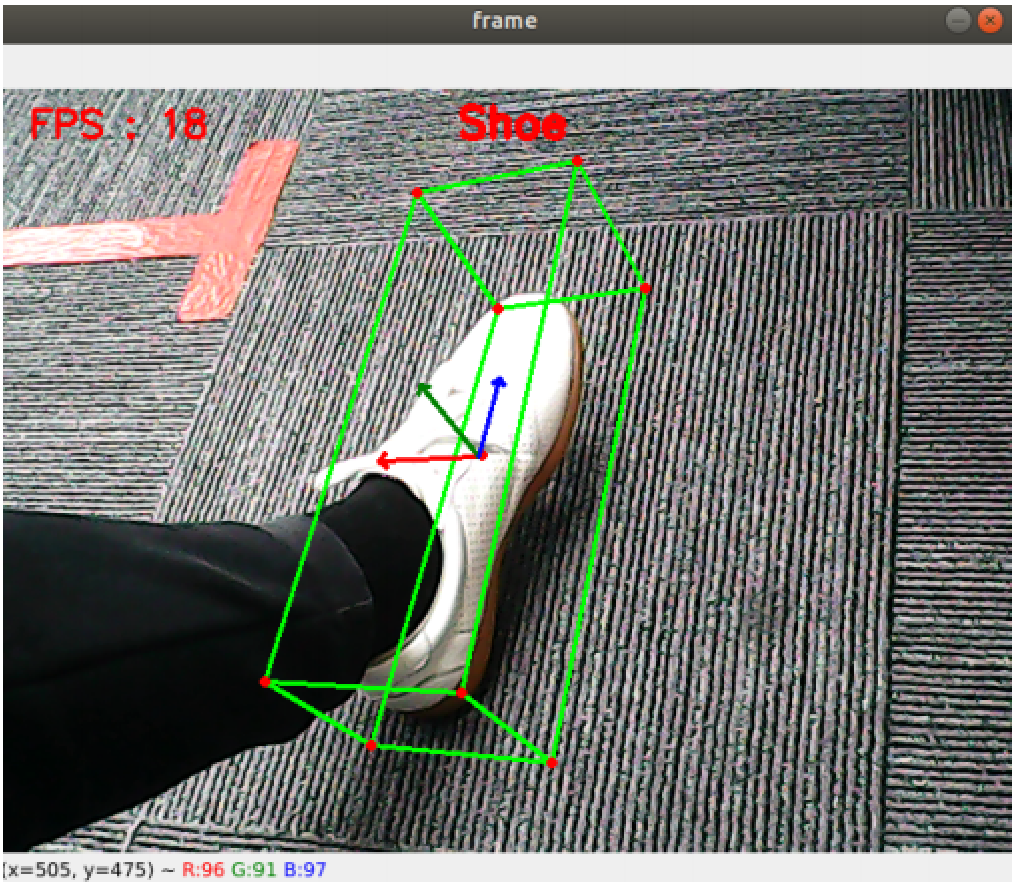
```
yahboom@yahboom-VM:~$ docker run -it --rm -v /dev:/dev -v /dev/shm:/dev/shm --pr
ivileged --net=host microros/micro-ros-agent:humble udp4 --port 9999 -v4

[1711695468.874360] info     | UDPv4AgentLinux.cpp | init                     |
running...              | port: 9999
[1711695468.874663] info     | Root.cpp            | set_verbose_level        | l
ogger setup            | verbose_level: 4
[1711695469.608224] info     | Root.cpp            | create_client            | c
reate                  | client_key: 0x63824D0E, session_id: 0x81
[1711695469.608287] info     | SessionManager.hpp | establish_session         | s
ession established     | client_key: 0x63824D0E, address: 192.168.2.114:27599
[1711695469.626174] info     | ProxyClient.cpp     | create_participant       | p
articipant created     | client_key: 0x63824D0E, participant_id: 0x000(1)
[1711695469.631263] info     | ProxyClient.cpp     | create_topic             | t
opic created           | client_key: 0x63824D0E, topic_id: 0x000(2), participant_
id: 0x000(1)
[1711695469.646135] info     | ProxyClient.cpp     | create_publisher         | p
ublisher created       | client_key: 0x63824D0E, publisher_id: 0x000(3), particip
ant_id: 0x000(1)
[1711695469.652213] info     | ProxyClient.cpp     | create_datawriter        | d
atawriter created      | client_key: 0x63824D0E, datawriter_id: 0x000(5), publish
er_id: 0x000(3)
```

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

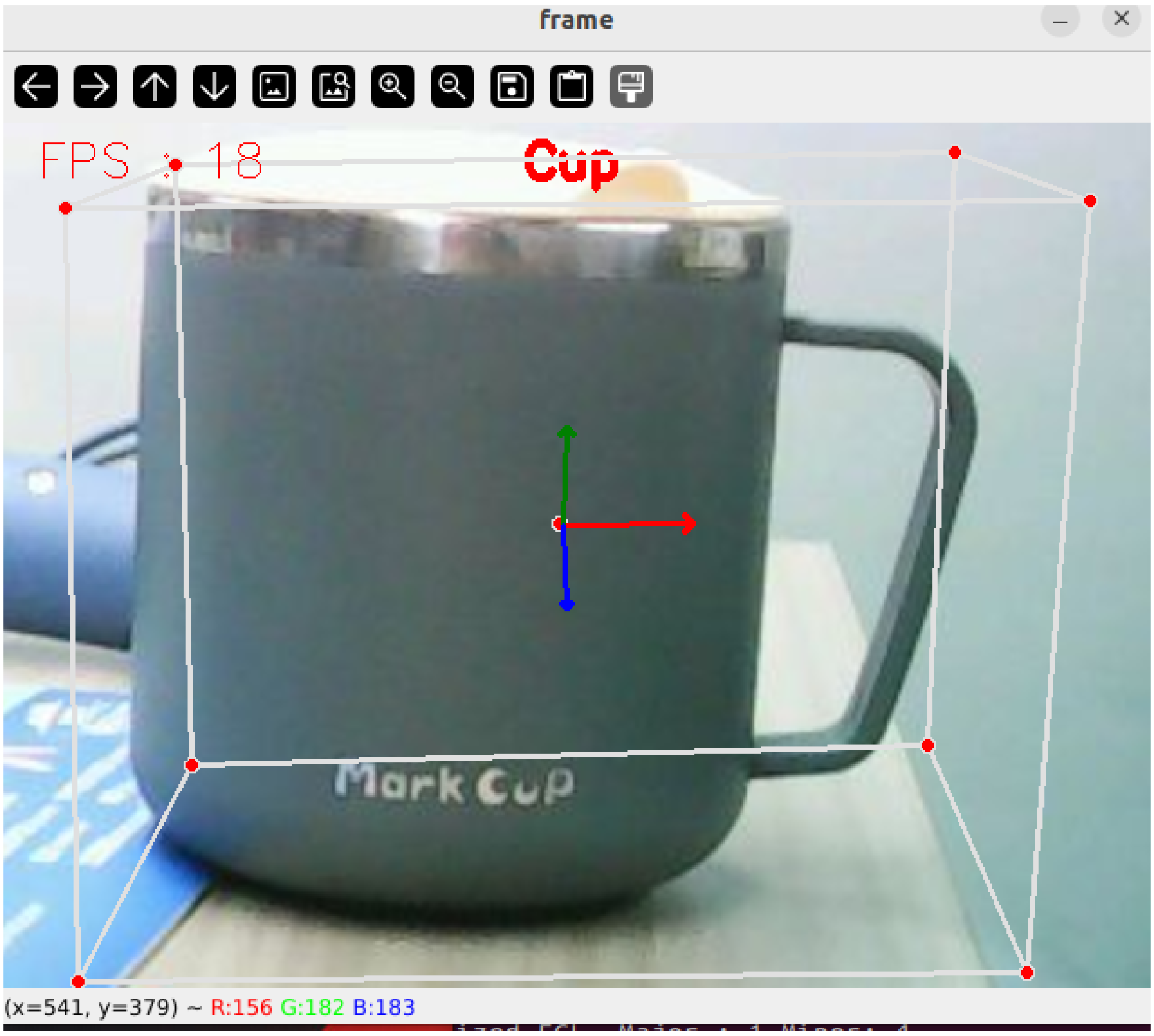
3. Open a new terminal and execute the following command

```
ros2 run yahboom_esp32_mediapipe 08_Objectron
```

Shoe



Cup



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

2.2、Code parsing

The location of the function source code is located,

```
~/yahboomcar_ws/src/yahboom_esp32_mediapipe/yahboom_esp32_mediapipe/08_Objectron.
py
```

```
#!/usr/bin/env python3
class Objectron:
    def __init__(self, staticMode=False, maxObjects=5, minDetectionCon=0.5,
minTrackingCon=0.99):
        self.staticMode=staticMode
        self.maxObjects=maxObjects
        self.minDetectionCon=minDetectionCon
        self.minTrackingCon=minTrackingCon
        self.index=3
        self.modelNames = ['Shoe', 'Chair', 'Cup', 'Camera']
        self.mpObjectron = mp.solutions.objectron
        self.mpDraw = mp.solutions.drawing_utils
        self.mpobjectron = self.mpObjectron.Objectron(
            self.staticMode, self.maxObjects, self.minDetectionCon,
self.minTrackingCon, self.modelNames[self.index])

    def findObjectron(self, frame):
        cv.putText(frame, self.modelNames[self.index], (int(frame.shape[1] / 2) -
30, 30),
                   cv.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 3)
        img_RGB = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
        results = self.mpobjectron.process(img_RGB)
        if results.detected_objects:
            for id, detection in enumerate(results.detected_objects):
                self.mpDraw.draw_landmarks(frame, detection.landmarks_2d,
self.mpObjectron.BOX_CONNECTIONS)
                self.mpDraw.draw_axis(frame, detection.rotation,
detection.translation)
        return frame

    def configUP(self):
        self.index += 1
        if self.index>=4:self.index=0
        self.mpobjectron = self.mpObjectron.Objectron(
            self.staticMode, self.maxObjects, self.minDetectionCon,
self.minTrackingCon, self.modelNames[self.index])


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)

        self.objectron = Objectron()

    def handleTopic(self, msg):
        start = time.time()
        frame = self.bridge.compressed_imgmsg_to_cv2(msg)
        frame = cv.resize(frame, (640, 480))

        action = cv.waitKey(1) & 0xFF
        if action == ord('f') or action == ord('F') :  self.objectron.configUP()
        frame = self.objectron.findObjectron(frame)

        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)
        cv.imshow('frame', frame)


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.