Hand detection

1. Content Description

This course implements color image acquisition and hand joint detection using the MediaPipe framework.

This section requires entering commands in the terminal. The terminal you open depends on your motherboard type. This lesson uses the Raspberry Pi 5 as an example. For Raspberry Pi and Jetson-Nano boards, you need to open a terminal on the host computer and enter the command to enter the Docker container. Once inside the Docker container, enter the commands mentioned in this section in the terminal. For instructions on entering the Docker container from the host computer, refer to this product tutorial **[Configuration and Operation Guide]--[Enter the Docker (Jetson Nano and Raspberry Pi 5 users, see here)]**.

Simply open the terminal on the Orin motherboard and enter the commands mentioned in this section.

2. Program startup

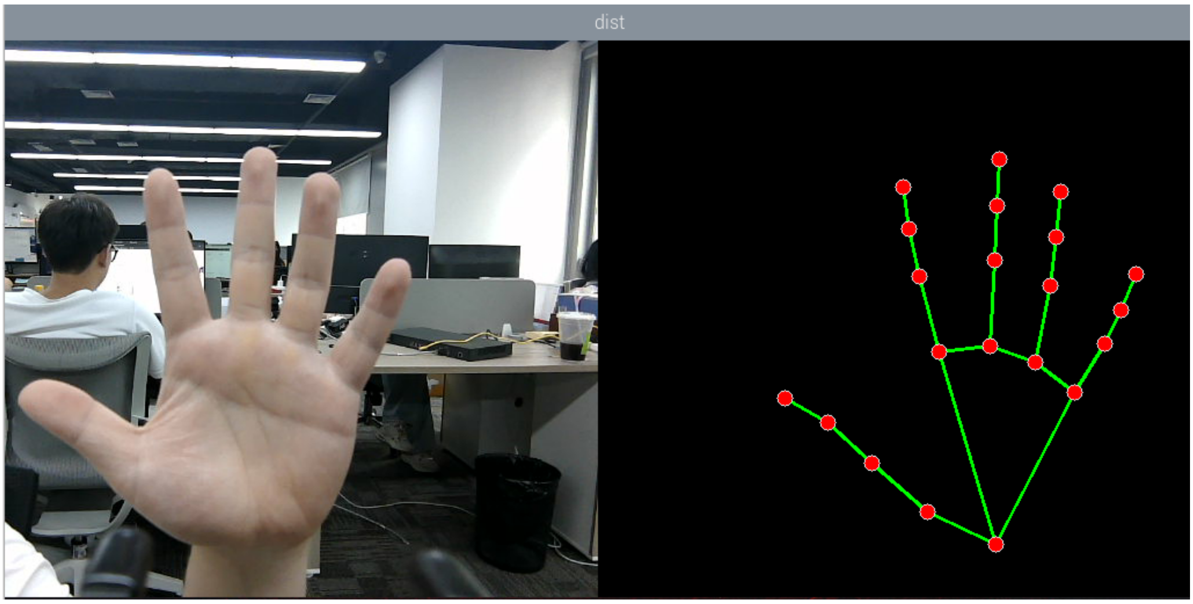
First, in the terminal, enter the following command to start the camera,

```
ros2 launch orbbec_camera dabai_dcw2.launch.py
```

After successfully starting the camera, open another terminal and enter the following command to start the hand detection program.

```
ros2 run yahboomcar_mediapipe 01_HandDetector
```

After the program is run, the following figure will be shown. The hand joint points detected will be displayed on the right side of the image.



3. Core code analysis

Program code path:

- Raspberry Pi 5 and Jetson-Nano board

  The program code is in the running docker. The path in docker is `/root/yahboomcar_ws/src/yahboomcar_mediapipe/yahboomcar_mediapipe/01_HandDetector.py`

- Orin Motherboard

  The program code path is `/home/jetson/yahboomcar_ws/src/yahboomcar_mediapipe/yahboomcar_mediapipe/01_HandDetector.py`

Import the library files used,

```
import rclpy
from rclpy.node import Node
import mediapipe as mp
import cv2 as cv
import numpy as np
import time
import os
from cv_bridge import CvBridge
from sensor_msgs.msg import Image
from arm_msgs.msg import ArmJoints
import cv2
```

Initialize data and define publishers and subscribers,

```
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.rgb_bridge = CvBridge()
    #Define the topic for controlling 6 servos and publish the hand detection
posture
    self.TargetAngle_pub = self.create_publisher(ArmJoints, "arm6_joints", 10)
    self.init_joints = [90, 150, 10, 20, 90, 90]
    self.pubSix_Arm(self.init_joints)
    #Define subscribers for the color image topic
    self.sub_rgb =
self.create_subscription(Image,"/camera/color/image_raw",self.get_RGBImageCallBa
ck,100)
```

Color image callback function,

```
def get_RGBImageCallBack(self,msg):
    #Use CvBridge to convert color image message data into image data
    rgb_image = self.rgb_bridge.imgmsg_to_cv2(msg, "bgr8")
    #Put the obtained image into the defined pubHandsPoint function, draw=False
means not to draw the joint points on the original color image
    frame, img = self.pubHandsPoint(rgb_image, draw=False)
    dist = self.frame_combine(frame, img)
    key = cv2.waitKey(1)
    cv.imshow('dist', dist)
```

pubHandsPoint function,

```
def pubHandsPoint(self, frame, draw=True):
    #Create a new image based on the incoming image size. The image data type is
uint8
    img = np.zeros(frame.shape, np.uint8)
    #Convert the color space of the incoming image from BGR to RGB to facilitate
subsequent image processing
    img_RGB = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
    #Call the process function in the mediapipe library for image processing.
During init, the self.hands object is created and initialized.
    self.results = self.hands.process(img_RGB)
    #Judge whether self.results.multi_hand_landmarks exists, that is, whether the
palm is recognized
    if self.results.multi_hand_landmarks:
        #Traverse the palm list and get the information of each point
        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)
            #Connect each joint point on the blank image created previously
            self.mpDraw.draw_landmarks(img,
self.results.multi_hand_landmarks[i], self.mpHand.HAND_CONNECTIONS,
self.lmDrawSpec, self.drawSpec)
    return frame, img
```

frame_combine merge image function,

```
def frame_combine(slef,frame, src):
    #Judge whether the image is a 3-channel, that is, RGB image
    if len(frame.shape) == 3:
        #Get the size of the two images and stitch them together
        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
```