Medipipe gesture control robotic arm action group

1. Content Description

This function captures color images and uses the mediapipe framework to recognize gestures, then uses these gestures to move the robotic arm to a predefined set of actions.

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
```

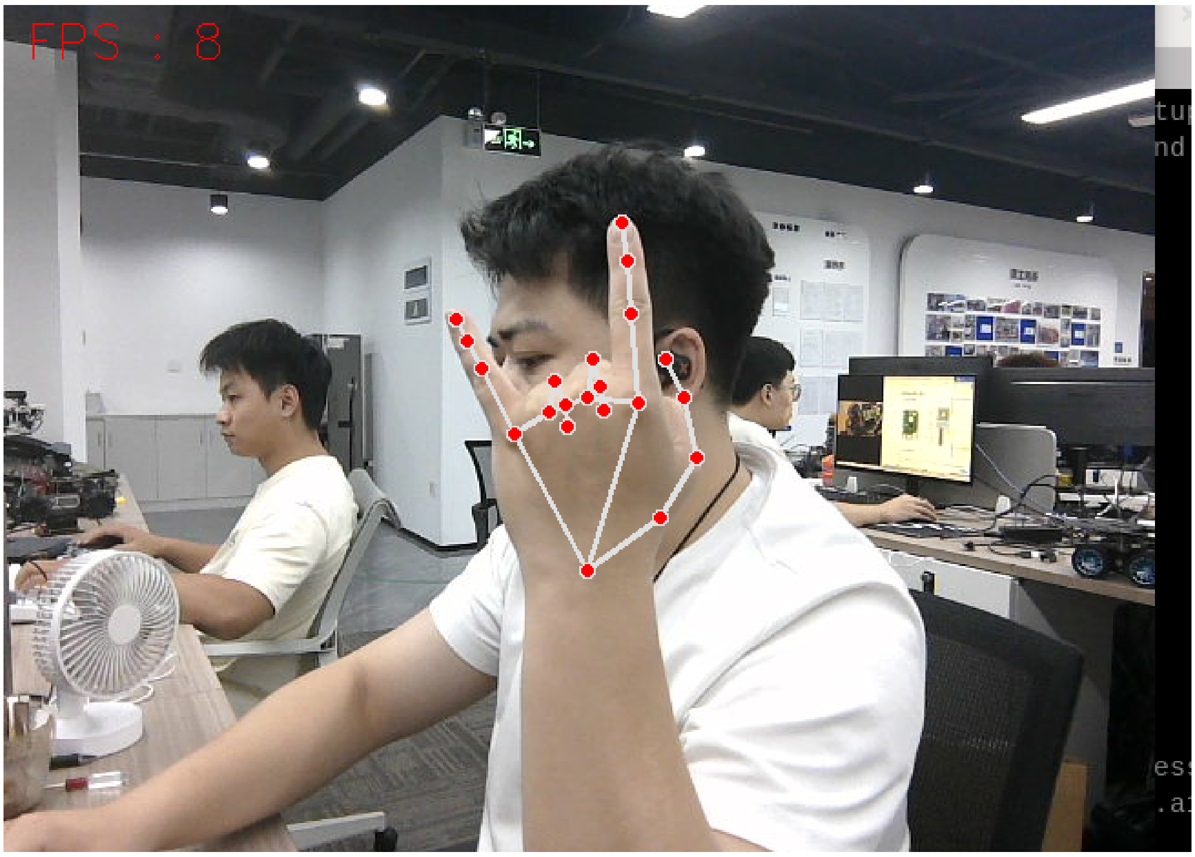
After successfully starting the camera, open another terminal and enter the following command in the terminal to start the gesture control program of the robotic arm.

```
ros2 run M3Pro_demo Gesture_Moving
```

After the program runs, the robotic arm will move to recognize gestures. There are several gestures to recognize:

- Yes gesture: The robotic arm will "dance" according to the set movements and then return to the initial posture.
- OK gesture: The robotic arm will shake its head left and right, then return to its initial position.
- Disdain gesture (clenched fist, thumbs below the line): The robotic arm will kneel down and then return to its original position.
- One-finger gesture (recognizing only one finger extended): The robotic arm nods and then returns to its initial posture.
- Rock gesture (thumb, middle finger, and ring finger are bent, index finger and pinky finger are straight): the robotic arm stretches upward and shakes left and right, then returns to the initial posture.
- Five fingers (all 5 fingers extended): The robotic arm claps and then returns to its initial position.

For example, the following is the Rock gesture.



After we make a gesture, press the space bar, and the robotic arm will perform a set of movements based on the recognized gesture. You need to press the space bar every time recognition is performed.

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/M3Pro_demo/M3Pro_demo/Gesture_Moving.py`

- Orin Motherboard

The program code path is `/home/jetson/yahboomcar_ws/src/M3Pro_demo/M3Pro_demo/Gesture_Moving.py`

Import the necessary library files,

```
import cv2
import os
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import cv2 as cv
from arm_msgs.msg import ArmJoints
from arm_msgs.msg import ArmJoint
import time
#Import the custom library, which contains the mediapipe related library
from M3Pro_demo.media_library import *
from rclpy.node import Node
import rclpy
import threading
```

The program initializes and creates publishers and subscribers,

```
def __init__(self, name):
    super().__init__(name)
    self.init_joints = [90, 150, 12, 20, 90, 0]
    self.rgb_bridge = CvBridge()
    #Call the media_library library to create an object of the HandDetector
class
    self.hand_detector = HandDetector()
    self.move_flag = True
    self.pr_time = time.time()
    self.pTime = self.cTime = 0
    self.event = threading.Event()
    self.event.set()
    self.arm_status = False
    self.move = False
    #Define the topic publisher for controlling a single servo, and publish the
topic for controlling the angle of a single servo
    self.pub_SingleTargetAngle = self.create_publisher(ArmJoint, "arm_joint",
10)
    #Define the topic publisher for controlling 6 servos and publish the topic
for controlling the angles of 6 servos
    self.TargetAngle_pub = self.create_publisher(ArmJoints, "arm6_joints", 10)
    #Define subscribers for the color image topic
    self.sub_rgb =
self.create_subscription(Image,"/camera/color/image_raw",self.get_RGBImageCallBa
ck,100)
    time.sleep(2)
    self.pubSix_Arm(self.init_joints)
```

Color image callback function,

```
def get_RGBImageCallBack(self,color_msg):
    #Use CvBridge to convert color image message data into image data
    rgb_image = self.rgb_bridge.imgmsg_to_cv2(color_msg, "bgr8")
    # Pass the obtained image into the process function for gesture detection
    self.process(rgb_image)
```

process function,

```
def process(self, frame):
    #Call the object method to perform palm detection and return the detected
image as well as the lmList list and bbox list
    frame, lmList, bbox = self.hand_detector.findHands(frame)
    key = cv2.waitKey(10)
    if key==32:
        self.move = True
    #Judge whether the palm is detected and whether the space bar is pressed. If
so, start the thread to execute the gesture.
    if len(lmList) != 0 and self.move == True:
        gesture = threading.Thread(target=self.Arm_Moving_threading, args=
(lmList,bbox))
        gesture.start()
    #Calculate frame rate display
    self.cTime = time.time()
    fps = 1 / (self.cTime - self.pTime)
    self.pTime = self.cTime
    text = "FPS : " + str(int(fps))
    cv.putText(frame, text, (20, 30), cv.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255),
1)
    if cv.waitKey(1) & 0xFF == ord('q'):
        cv.destroyAllWindows()
    cv.imshow('frame', frame)
```

Arm_Moving_threadin gesture recognition thread program,

```
def Arm_Moving_threading(self, lmList,bbox):
    if self.event.is_set():
        self.event.clear()
        #Call the finger extension detection function to check which fingers are
extended
        fingers = self.hand_detector.fingersUp(lmList)
        self.hand_detector.draw = False
        #Call the gesture function to get the current gesture
        gesture = self.hand_detector.get_gesture(lmList)
        #For the current gesture and finger extension, execute the corresponding
custom robotic arm action group function
        if gesture == "Yes":
            self.arm_status = False
            self.dance()
            time.sleep(0.5)
            self.init_pose()
            time.sleep(1.0)
            self.arm_status = True

        elif gesture == "OK":
            self.arm_status = False
            for i in range(3):
                time.sleep(0.1)
                self.pubSix_Arm([80, 135, 0, 15, 90, 180])
                time.sleep(0.5)
                self.pubSix_Arm([110, 135, 0, 15, 90, 90])
                time.sleep(0.5)
            self.init_pose()
            sleep(1.0)
            self.arm_status = True
        elif gesture == "Thumb_down":
            self.arm_status = False
            self.pubSix_Arm([90, 0, 180, 0, 90, 180])
            time.sleep(0.5)
            self.pubSix_Arm([90, 0, 180, 0, 90, 90])
            time.sleep(1.5)
            self.init_pose()
            time.sleep(1.0)
            self.arm_status = True
        elif sum(fingers) == 1:
            self.arm_status = False
            self.arm_nod()
            self.init_pose()
            time.sleep(1.0)
            self.arm_status = True
        elif fingers[1] == fingers[4] == 1 and sum(fingers) == 2:
            self.arm_status = False
            self.shake()
            self.init_pose()
            time.sleep(1.0)
            self.arm_status = True
        elif sum(fingers) == 5:
            self.arm_status = False
            self.arm_applaud()
            self.init_pose()
            time.sleep(1.0)
            self.arm_status = True
        self.move  = False
        self.event.set()
```