8. Brush

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、8. Brush

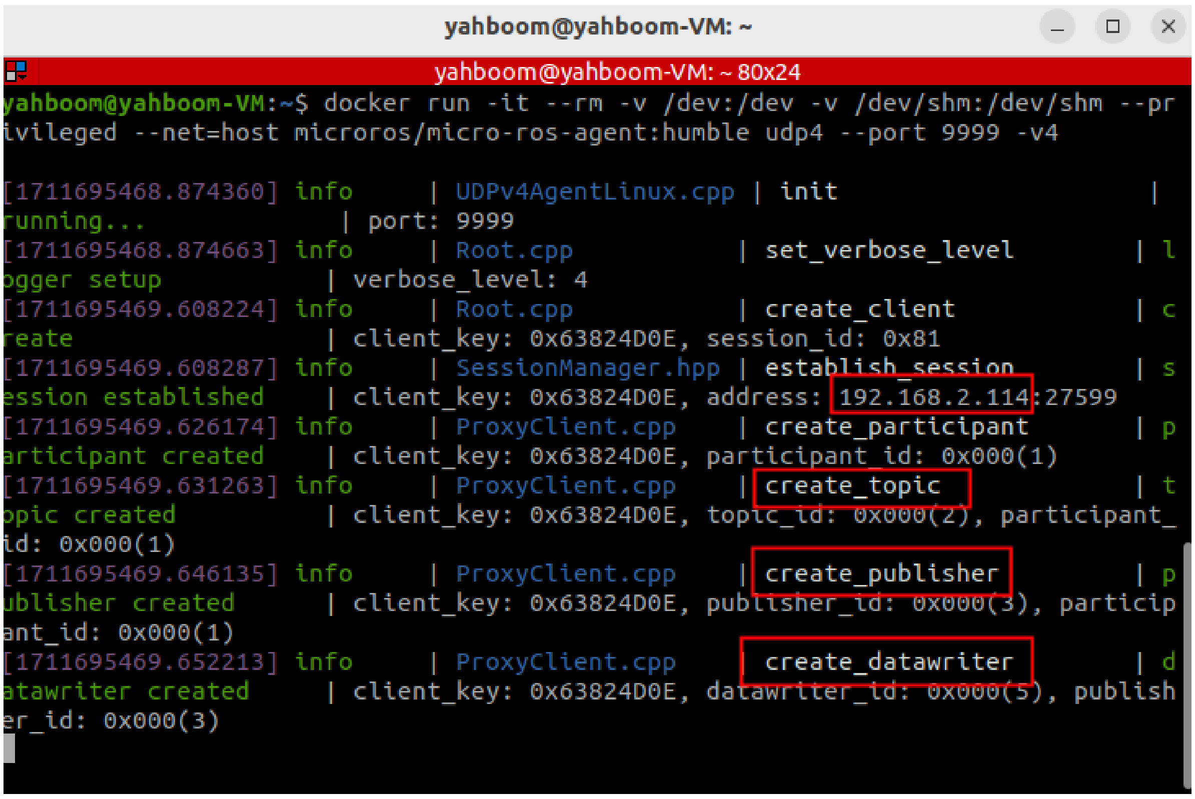
When the index finger and middle finger of the right hand are combined, the state is selected. At the same time, the color selection box pops up. When the two fingertips move to the corresponding color position, select the color (black is the eraser); The index finger and middle finger start as a painting state and can be drawn arbitrarily on the drawing board.

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
```



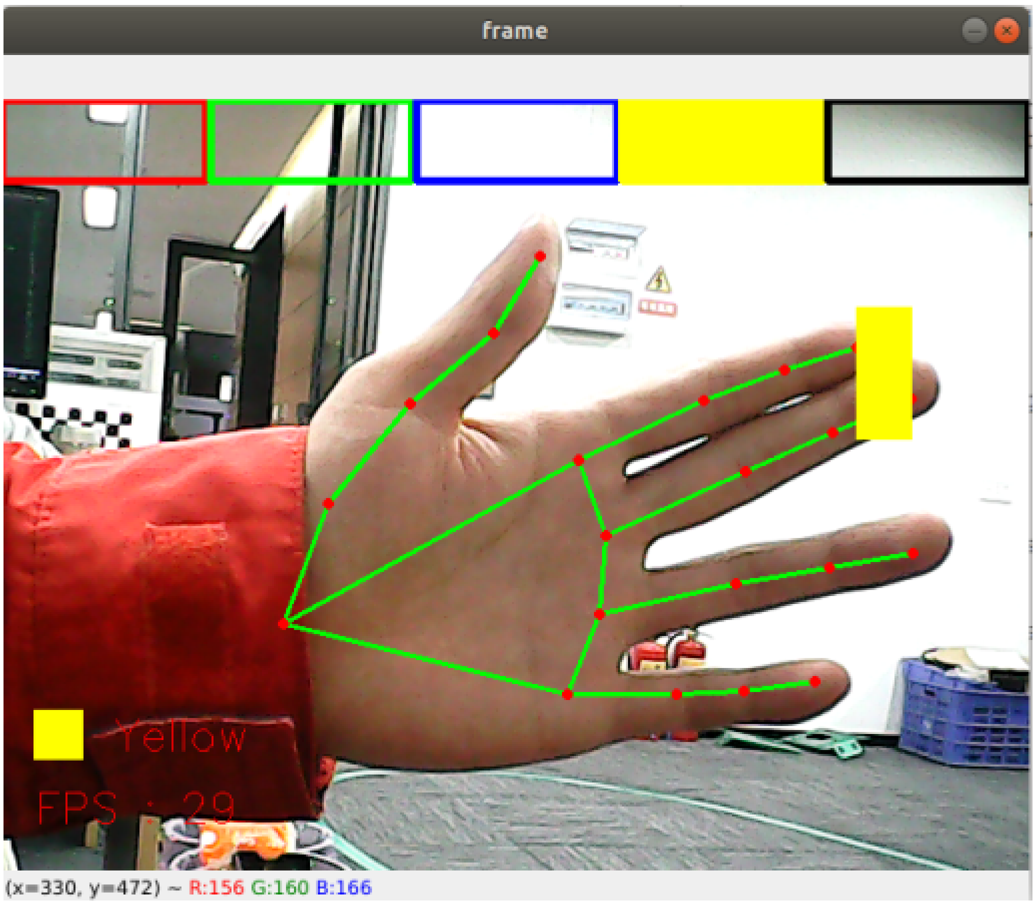
```
yahboom@yahboom-VM:~$ docker run -it --rm -v /dev:/dev -v /dev/shm:/dev/shm --pr
ivileged --net=host microros/micro-ros-agent:humble udp4 --port 9999 -v4

[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 09_VirtualPaint
```



4. 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

```
~/yahboomcar_ws/src/yahboom_esp32_mediapipe/yahboom_esp32_mediapipe/09_VirtualPa
int.py
```

```
xp = yp = pTime = boxx = 0
tipIds = [4, 8, 12, 16, 20]
imgCanvas = np.zeros((480, 640, 3), np.uint8)
brushThickness = 5
eraserThickness = 100
top_height = 50
Color = "Red"
ColorList = {
    'Red': (0, 0, 255),
    'Green': (0, 255, 0),
    'Blue': (255, 0, 0),
    'Yellow': (0, 255, 255),
    'Black': (0, 0, 0),
}

class handDetector:
    def __init__(self, mode=False, maxHands=2, detectorCon=0.5, trackCon=0.5):
        self.tipIds = [4, 8, 12, 16, 20]
        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=15)
        self.drawSpec = mp.solutions.drawing_utils.DrawingSpec(color=(0, 255,
0), thickness=10, circle_radius=10)

    def findHands(self, frame, draw=True):
        self.lmList = []
        img_RGB = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
        self.results = self.hands.process(img_RGB)
        if self.results.multi_hand_landmarks:
            for handLms in self.results.multi_hand_landmarks:
                if draw: self.mpDraw.draw_landmarks(frame, handLms,
self.mpHand.HAND_CONNECTIONS, self.lmDrawSpec, self.drawSpec)
                else: self.mpDraw.draw_landmarks(frame, handLms,
self.mpHand.HAND_CONNECTIONS)
            for id, lm in
enumerate(self.results.multi_hand_landmarks[0].landmark):
                h, w, c = frame.shape
                cx, cy = int(lm.x * w), int(lm.y * h)
                # print(id, cx, cy)
                self.lmList.append([id, cx, cy])
        return frame, self.lmList

    def fingersUp(self):
        fingers=[]
        # Thumb
        if (self.calc_angle(self.tipIds[0],
                            self.tipIds[0] - 1,
                            self.tipIds[0] - 2) > 150.0) and (
                self.calc_angle(
                    self.tipIds[0] - 1,
                    self.tipIds[0] - 2,
                    self.tipIds[0] - 3) > 150.0): fingers.append(1)
        else:
            fingers.append(0)
        # 4 finger
        for id in range(1, 5):
            if self.lmList[self.tipIds[id]][2] < self.lmList[self.tipIds[id] -
2][2]:
                fingers.append(1)
            else:
                fingers.append(0)
        return fingers



    def get_dist(self, point1, point2):
        x1, y1 = point1
        x2, y2 = point2
        return abs(math.sqrt(math.pow(abs(y1 - y2), 2) + math.pow(abs(x1 - x2),
2)))

    def calc_angle(self, pt1, pt2, pt3):
        point1 = self.lmList[pt1][1], self.lmList[pt1][2]
        point2 = self.lmList[pt2][1], self.lmList[pt2][2]
        point3 = self.lmList[pt3][1], self.lmList[pt3][2]
        a = self.get_dist(point1, point2)
        b = self.get_dist(point2, point3)
        c = self.get_dist(point1, point3)
        try:
            radian = math.acos((math.pow(a, 2) + math.pow(b, 2) - math.pow(c,
2)) / (2 * a * b))
            angle = radian / math.pi * 180
        except:
            angle = 0
        return abs(angle)


class MY_Picture(Node):
    def __init__(self, name,ColorROS,ColorListROS):
        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(detectorCon=0.85)
        self.Color = ColorROS
        self.ColorList = ColorListROS
        self.boxx = 0
        self.xp = 0
        self.yp = 0
        self.top_height = 50

    def handleTopic(self, msg):
        start = time.time()
        frame = self.bridge.compressed_imgmsg_to_cv2(msg)
        frame = cv.resize(frame, (640, 480))
        h, w, c = frame.shape
        cv.waitKey(10)

        frame,lmList  = self.hand_detector.findHands(frame, draw=False)
        if len(lmList) != 0:
            # print(lmList)
            # tip of index and middle fingers
            x1, y1 = lmList[8][1:]
            x2, y2 = lmList[12][1:]

            fingers = self.hand_detector.fingersUp()
            if fingers[1] and fingers[2]:
                # print("Seclection mode")
                if y1 < self.top_height:
                    if 0 < x1 < int(w / 5) - 1:
                        self.boxx = 0
                        self.Color = "Red"
                    if int(w / 5) < x1 < int(w * 2 / 5) - 1:
                        self.boxx = int(w / 5)
                        self.Color = "Green"
                    elif int(w * 2 / 5) < x1 < int(w * 3 / 5) - 1:
                        self.boxx = int(w * 2 / 5)
                        self.Color = "Blue"
                    elif int(w * 3 / 5) < x1 < int(w * 4 / 5) - 1:
                        self.boxx = int(w * 3 / 5)
                        self.Color = "Yellow"
                    elif int(w * 4 / 5) < x1 < w - 1:
                        self.boxx = int(w * 4 / 5)
                        self.Color = "Black"
                cv.rectangle(frame, (x1, y1 - 25), (x2, y2 + 25),
self.ColorList[self.Color], cv.FILLED)
                cv.rectangle(frame, (self.boxx, 0), (self.boxx + int(w / 5),
self.top_height), self.ColorList[self.Color], cv.FILLED)
                cv.rectangle(frame, (0, 0), (int(w / 5) - 1, self.top_height),
self.ColorList['Red'], 3)
                cv.rectangle(frame, (int(w / 5) + 2, 0), (int(w * 2 / 5) - 1,
self.top_height), self.ColorList['Green'], 3)
                cv.rectangle(frame, (int(w * 2 / 5) + 2, 0), (int(w * 3 / 5) -
1, self.top_height), self.ColorList['Blue'], 3)
                cv.rectangle(frame, (int(w * 3 / 5) + 2, 0), (int(w * 4 / 5) -
1, self.top_height), self.ColorList['Yellow'], 3)
                cv.rectangle(frame, (int(w * 4 / 5) + 2, 0), (w - 1,
self.top_height), self.ColorList['Black'], 3)
            if fingers[1] and fingers[2] == False and math.hypot(x2 - x1, y2 -
y1) > 50:
                # print("Drawing mode")
                if self.xp == self.yp == 0: self.xp, self.yp = x1, y1
                if self.Color == 'Black':
                    cv.line(frame, (self.xp, self.yp), (x1, y1),
self.ColorList[self.Color], eraserThickness)
                    cv.line(imgCanvas, (self.xp, self.yp), (x1, y1),
self.ColorList[self.Color], eraserThickness)
                else:
                    cv.line(frame, (self.xp, self.yp), (x1, y1),
self.ColorList[self.Color], brushThickness)
                    cv.line(imgCanvas, (self.xp, self.yp), (x1,
y1),self.ColorList[self.Color], brushThickness)
                cv.circle(frame, (x1, y1), 15, self.ColorList[self.Color],
cv.FILLED)
                self.xp, self.yp = x1, y1
            else: self.xp = self.yp = 0
        imgGray = cv.cvtColor(imgCanvas, cv.COLOR_BGR2GRAY)
        _, imgInv = cv.threshold(imgGray, 50, 255, cv.THRESH_BINARY_INV)
        imgInv = cv.cvtColor(imgInv, cv.COLOR_GRAY2BGR)
        frame = cv.bitwise_and(frame, imgInv)
        frame = cv.bitwise_or(frame, imgCanvas)
        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.rectangle(frame, (20, h - 100), (50, h - 70),
self.ColorList[self.Color], cv.FILLED)
        cv.putText(frame, self.Color, (70, h - 75), cv.FONT_HERSHEY_SIMPLEX,
0.9, (0, 0, 255), 1)
        cv.imshow('frame', frame)




def main():
    global Color,ColorList
    print("start it")
    rclpy.init()
    esp_img = MY_Picture("My_Picture",Color,ColorList)
    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.