3. Overall detection

3.1. Introduction

MediaPipe is an open-source data stream processing machine learning application development framework developed by Google. It is a graph-based data processing pipeline used to build data sources in various forms, such as video, audio, sensor data, and any time series data. MediaPipe is cross-platform and can run on embedded platforms (Raspberry Pi, etc.), mobile devices (iOS and Android), workstations and servers, and supports 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.

Features of MediaPipe:

- End-to-end acceleration: built-in fast ML inference and processing can be accelerated even on ordinary hardware.
- Build once, deploy anywhere: unified solution for Android, iOS, desktop/cloud, web, and IoT.
- Ready-to-use solution: cutting-edge ML solution that demonstrates the full capabilities of the framework.
- Free and open source: framework and solution under Apache2.0, fully extensible and customizable.

3.2, MediaPipe Hands

Refer to the content of hand detection [1.2] in the first section, which will not be repeated here.

3.3, MediaPipe Pose

Refer to the content of hand detection [2.2] in the first section, which will not be repeated here.

3.4, overall detection

Combining the content of the previous two sections, this section's routine implements the function of detecting both palms and human bodies.
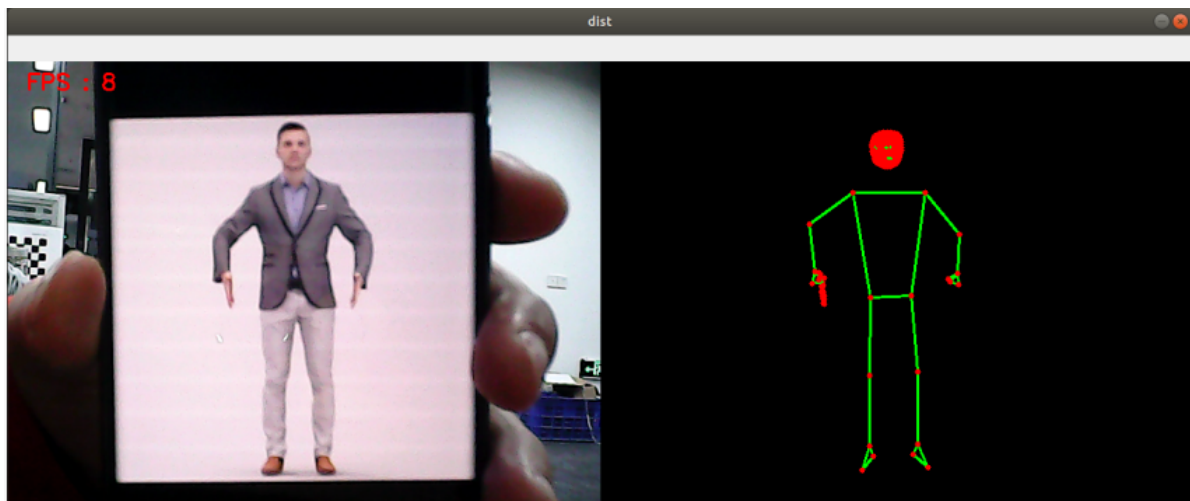
3.4.1, Start

Start the camera

```
roslaunch ascamera hp60c.launch
```

Terminal input,

```
roslaunch yahboomcar_mediapipe 03_Holistic.launch
```

3.4.2, Source code

Source code location:

/home/yahboom/ascam_ws_ws/src/yahboomcar_mediapipe/scripts/03_Holistic.py

```python
#!/usr/bin/env python3
# encoding: utf-8
import time
import rospy
import cv2 as cv
import numpy as np
import mediapipe as mp
from geometry_msgs.msg import Point
from yahboomcar_msgs.msg import PointArray
from sensor_msgs.msg import Image
from cv_bridge import CvBridge


class Holistic:
    def __init__(self, staticMode=False, landmarks=True, detectionCon=0.5,
trackingCon=0.5):
        self.mpHolistic = mp.solutions.holistic
        self.mpFaceMesh = mp.solutions.face_mesh
        self.mpHands = mp.solutions.hands
        self.mpPose = mp.solutions.pose
        self.mpDraw = mp.solutions.drawing_utils
        self.mpholistic = self.mpHolistic.Holistic(
            static_image_mode=staticMode,
            smooth_landmarks=landmarks,
            min_detection_confidence=detectionCon,
            min_tracking_confidence=trackingCon)
        self.pub_point = rospy.Publisher('/mediapipe/points', PointArray,
queue_size=1000)
        self.lmDrawSpec = mp.solutions.drawing_utils.DrawingSpec(color=(0, 0,
255), thickness=-1, circle_radius=3)
        self.drawSpec = mp.solutions.drawing_utils.DrawingSpec(color=(0, 255,
0), thickness=2, circle_radius=2)
        self.bridge = CvBridge()
        self.sub_image = rospy.Subscriber('/ascamera_hp60c/rgb0/image', Image,
self.image_callback, queue_size=1)
        self.pTime = 0
```

```python
    def findHolistic(self, frame, draw=True):
        pointArray = PointArray()
        img = np.zeros(frame.shape, np.uint8)
        img_RGB = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
        self.results = self.mpholistic.process(img_RGB)
        if self.results.face_landmarks:
            if draw:

                self.mpDraw.draw_landmarks(frame, self.results.face_landmarks,
self.mpFaceMesh.FACEMESH_CONTOURS, self.lmDrawSpec, self.drawSpec)
            self.mpDraw.draw_landmarks(img, self.results.face_landmarks,
self.mpFaceMesh.FACEMESH_CONTOURS, self.lmDrawSpec, self.drawSpec)
            for id, lm in enumerate(self.results.face_landmarks.landmark):
                point = Point()
                point.x, point.y, point.z = lm.x, lm.y, lm.z
                pointArray.points.append(point)
        if self.results.pose_landmarks:
            if draw:
                self.mpDraw.draw_landmarks(frame, self.results.pose_landmarks,
self.mpPose.POSE_CONNECTIONS, self.lmDrawSpec, self.drawSpec)
            self.mpDraw.draw_landmarks(img, self.results.pose_landmarks,
self.mpPose.POSE_CONNECTIONS, self.lmDrawSpec, self.drawSpec)
            for id, lm in enumerate(self.results.pose_landmarks.landmark):
                point = Point()
                point.x, point.y, point.z = lm.x, lm.y, lm.z
                pointArray.points.append(point)
        if self.results.left_hand_landmarks:
            if draw:
                self.mpDraw.draw_landmarks(frame,
self.results.left_hand_landmarks, self.mpHands.HAND_CONNECTIONS,
self.lmDrawSpec, self.drawSpec)
            self.mpDraw.draw_landmarks(img, self.results.left_hand_landmarks,
self.mpHands.HAND_CONNECTIONS, self.lmDrawSpec, self.drawSpec)
            for id, lm in enumerate(self.results.left_hand_landmarks.landmark):
                point = Point()
                point.x, point.y, point.z = lm.x, lm.y, lm.z
                pointArray.points.append(point)
        if self.results.right_hand_landmarks:
            if draw:
                self.mpDraw.draw_landmarks(frame,
self.results.right_hand_landmarks, self.mpHands.HAND_CONNECTIONS,
self.lmDrawSpec, self.drawSpec)
            self.mpDraw.draw_landmarks(img, self.results.right_hand_landmarks,
self.mpHands.HAND_CONNECTIONS, self.lmDrawSpec, self.drawSpec)
            for id, lm in enumerate(self.results.right_hand_landmarks.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(self, frame, src):
        if len(frame.shape) == 3:
            frameH, frameW = frame.shape[:2]
```

```python
            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

    def image_callback(self, msg):
        try:
            # Convert ROS image messages to OpenCV images
            frame = self.bridge.imgmsg_to_cv2(msg, desired_encoding="bgr8")

            # Process images and publish keypoints
            frame, img = self.findHolistic(frame, draw=True)

            # Calculate and display frame rate
            cTime = time.time()
            fps = 1 / (cTime - self.pTime)
            self.pTime = cTime
            text = "FPS : " + str(int(fps))
            cv.putText(frame, text, (20, 30), cv.FONT_HERSHEY_SIMPLEX, 0.8, (0,
0, 255), 2)

            # Merge and display images
            dist = self.frame_combine(frame, img)
            cv.imshow('dist', dist)

            # Check for 'q' key pressed to quit
            if cv.waitKey(1) & 0xFF == ord('q'):
                rospy.signal_shutdown("User requested shutdown")
        except Exception as e:
            rospy.logerr(f"Could not process image: {e}")


if __name__ == '__main__':
    rospy.init_node('Holistic', anonymous=True)
    holistic = Holistic()
    rospy.spin()
    cv.destroyAllWindows()
```