Fingertip trajectory recognition

1. Introduction

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 used to build and utilize various forms of data sources, such as video, audio, 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, with support for 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.

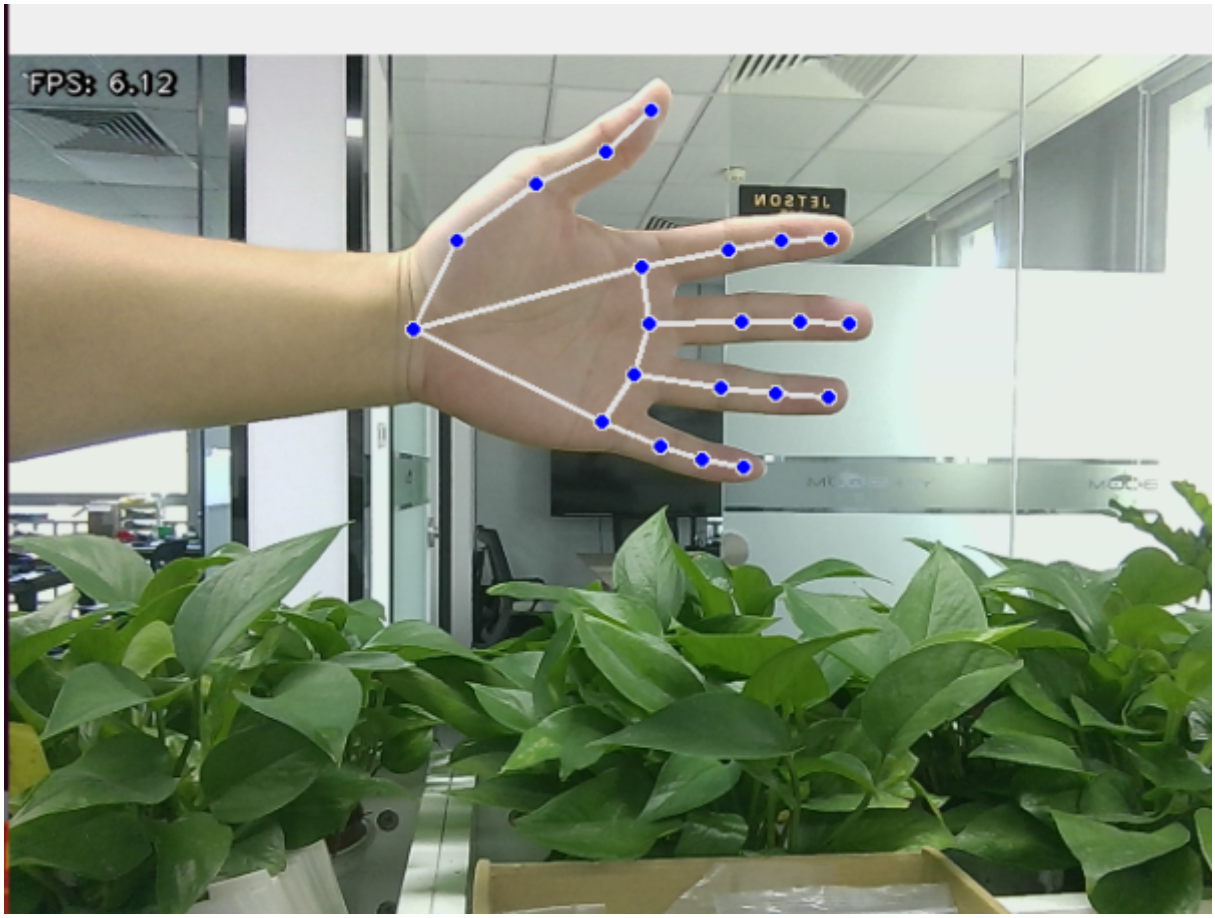
Features of MediaPipe:

-End to end acceleration: Built in fast ML inference and processing can accelerate even on regular hardware.
-Build once, deploy anytime, anywhere: A unified solution suitable for Android, iOS, desktop/cloud, web, and IoT.
-Ready to use solution: a cutting-edge ML solution that showcases all the functionalities of the framework.
-Free and open-source: frameworks and solutions under Apache 2.0, fully extensible and customizable.

2. Start

2.1 Program description

After the program starts, the camera captures an image and places the palm on the camera screen. The image will draw the joints on the palm.



2.2 Start program

- Input following command to start the program

```
roslaunch jetcobot_advance finger_trajectory.launch
```

2.3 About code

Code path：~/jetcobot_ws/src/jetcobot_advance/scripts/finger_trajectory.py

```
#!/usr/bin/env python3
# coding: utf8
import os
import enum
import cv2
import time
import numpy as np
import mediapipe as mp
import rospy
import queue
from pymycobot.mycobot import MyCobot
from sensor_msgs.msg import Image
import fps
from vutils import distance, vector_2d_angle, get_area_max_contour
import gc
```

```
def get_hand_landmarks(img, landmarks):
    """
    将landmarks从medipipe的归一化输出转为像素坐标
    :param img: 像素坐标对应的图片
    :param landmarks: 归一化的关键点
    :return:
    """
    h, w, _ = img.shape
    landmarks = [(lm.x * w, lm.y * h) for lm in landmarks]
    return np.array(landmarks)


def hand_angle(landmarks):
    """
    计算各个手指的弯曲角度
    :param landmarks: 手部关键点
    :return: 各个手指的角度
    """
    angle_list = []
    # thumb 大拇指
    angle_ = vector_2d_angle(landmarks[3] - landmarks[4], landmarks[0] -
landmarks[2])
    angle_list.append(angle_)
    # index 食指
    angle_ = vector_2d_angle(landmarks[0] - landmarks[6], landmarks[7] -
landmarks[8])
    angle_list.append(angle_)
    # middle 中指
    angle_ = vector_2d_angle(landmarks[0] - landmarks[10], landmarks[11] -
landmarks[12])
    angle_list.append(angle_)
    # ring 无名指
    angle_ = vector_2d_angle(landmarks[0] - landmarks[14], landmarks[15] -
landmarks[16])
    angle_list.append(angle_)
    # pink 小拇指
    angle_ = vector_2d_angle(landmarks[0] - landmarks[18], landmarks[19] -
landmarks[20])
    angle_list.append(angle_)
    angle_list = [abs(a) for a in angle_list]
    return angle_list


def h_gesture(angle_list):
    """
    通过二维特征确定手指所摆出的手势
    :param angle_list: 各个手指弯曲的角度
    :return : 手势名称字符串
    """
    thr_angle, thr_angle_thumb, thr_angle_s = 65.0, 53.0, 49.0
```

```
    if (angle_list[0] < thr_angle_s) and (angle_list[1] < thr_angle_s) and
(angle_list[2] < thr_angle_s) and (
            angle_list[3] < thr_angle_s) and (angle_list[4] < thr_angle_s):
        gesture_str = "five"
    elif (angle_list[0] > 5) and (angle_list[1] < thr_angle_s) and (angle_list[2] >
thr_angle) and (
            angle_list[3] > thr_angle) and (angle_list[4] > thr_angle):
        gesture_str = "one"
    else:
        gesture_str = "none"
    return gesture_str


class State(enum.Enum):
    NULL = 0
    TRACKING = 1
    RUNNING = 2


def draw_points(img, points, tickness=4, color=(255, 0, 0)):
    """
    将记录的点连线画在画面上
    """
    points = np.array(points).astype(dtype=np.int32)
    if len(points) > 2:
        for i, p in enumerate(points):
            if i + 1 >= len(points):
                break
            cv2.line(img, tuple(p), tuple(points[i + 1]), color, tickness)

def get_track_img(points):
    """
    用记录的点生成一张黑底白线的轨迹图
    """
    points = np.array(points).astype(dtype=np.int32)
    x_min, y_min = np.min(points, axis=0).tolist()
    x_max, y_max = np.max(points, axis=0).tolist()
    track_img = np.full([y_max - y_min + 100, x_max - x_min + 100, 1], 0,
dtype=np.uint8)
    points = points - [x_min, y_min]
    points = points + [50, 50]
    draw_points(track_img, points, 1, (255, 255, 255))
    return track_img


class FingerTrajectoryNode:
    def __init__(self):
        rospy.init_node('finger_trajectory')
        self.drawing = mp.solutions.drawing_utils
        self.timer = time.time()

        self.hand_detector = mp.solutions.hands.Hands(
```

```
            static_image_mode=False,
            max_num_hands=1,
            min_tracking_confidence=0.05,
            min_detection_confidence=0.6
        )


        self.fps = fps.FPS()  # fps计算器
        self.state = State.NULL
        self.points = []
        self.start_count = 0
        self.no_finger_timestamp = time.time()

        self.mc = MyCobot(str(os.getenv('MY_SERIAL')), 1000000)
        self.mc.send_angles([0, 0, -90, 90, 0, -45], 50)

        self.gc_stamp = time.time()
        self.image_queue = queue.Queue(maxsize=1)
        source_image_topic = rospy.get_param('~source_image_topic',
'/camera/color/image_raw')
        rospy.loginfo("source_image_topic = {}".format(source_image_topic))
        self.image_sub = rospy.Subscriber(source_image_topic, Image,
self.image_callback, queue_size=1)


    def image_callback(self, ros_image: Image):
        try:
            self.image_queue.put_nowait(ros_image)
        except Exception as e:
            pass

    def image_proc(self):
    #    rospy.loginfo('Received an image! ')
        ros_image = self.image_queue.get(block=True)
        rgb_image = np.ndarray(shape=(ros_image.height, ros_image.width, 3),
dtype=np.uint8, buffer=ros_image.data)
        rgb_image = cv2.flip(rgb_image, 1) # 水平翻转
        result_image = np.copy(rgb_image)
        result_call = None
        if self.timer <= time.time() and self.state == State.RUNNING:
            self.state = State.NULL
        try:
            results = self.hand_detector.process(rgb_image) if self.state !=
State.RUNNING else None
            if results is not None and results.multi_hand_landmarks:
                gesture = "none"
                index_finger_tip = [0, 0]
                self.no_finger_timestamp = time.time()  # 记下当期时间，以便超时处理
                for hand_landmarks in results.multi_hand_landmarks:
                    self.drawing.draw_landmarks(
                        result_image,
                        hand_landmarks,
```

```
                        mp.solutions.hands.HAND_CONNECTIONS)
                    landmarks = get_hand_landmarks(rgb_image,
hand_landmarks.landmark)
                    angle_list = (hand_angle(landmarks))
                    gesture = (h_gesture(angle_list))
                    index_finger_tip = landmarks[8].tolist()

                if self.state == State.NULL:
                    if gesture == "one":  # 检测到单独伸出食指，其他手指握拳
                        self.start_count += 1
                        if self.start_count > 20:
                            self.state = State.TRACKING
                            self.points = []
                    else:
                        self.start_count = 0

                elif self.state == State.TRACKING:
                    if gesture == "five": # 伸开五指结束画图
                        self.state = State.NULL

                        # 生成黑白轨迹图
                        track_img = get_track_img(self.points)
                        contours = cv2.findContours(track_img, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)[-2]
                        contour = get_area_max_contour(contours, 300)
                        contour = contour[0]
                        # 按轨迹图识别所画图形
                        # cv2.fillPoly在图像上绘制并填充多边形
                        track_img = cv2.fillPoly(track_img, [contour,], (255, 255,
255))
                        for _ in range(3):
                            # 腐蚀函数
                            track_img = cv2.erode(track_img,
cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5)))
                            # 膨胀函数
                            track_img = cv2.dilate(track_img,
cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5)))
                        contours = cv2.findContours(track_img, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)[-2]
                        contour = get_area_max_contour(contours, 300)
                        contour = contour[0]
                        h, w = track_img.shape[:2]

                        track_img = np.full([h, w, 3], 0, dtype=np.uint8)
                        track_img = cv2.drawContours(track_img, [contour, ], -1, (0,
255, 0), 2)
                        # 对图像轮廓点进行多边形拟合
                        approx = cv2.approxPolyDP(contour, 0.026 *
cv2.arcLength(contour, True), True)
                        track_img = cv2.drawContours(track_img, [approx, ], -1, (0,
0, 255), 2)
```

```
                        print(len(approx))
                        # 根据轮廓包络的顶点数确定图形
                        if len(approx) == 3:
                            cv2.putText(track_img, 'Triangle', (10,
40),cv2.FONT_HERSHEY_SIMPLEX, 1.2, (255, 255, 0), 2)
                        if len(approx) == 4 or len(approx) == 5:
                            cv2.putText(track_img, 'Square', (10,
40),cv2.FONT_HERSHEY_SIMPLEX, 1.2, (255, 255, 0), 2)
                        if 5 < len(approx) < 10:
                            cv2.putText(track_img, 'Circle', (10,
40),cv2.FONT_HERSHEY_SIMPLEX, 1.2, (255, 255, 0), 2)
                        if len(approx) == 10:
                            cv2.putText(track_img, 'Star', (10,
40),cv2.FONT_HERSHEY_SIMPLEX, 1.2, (255, 255, 0), 2)

                        cv2.imshow('track', track_img)

                    else:
                        if len(self.points) > 0:
                            if distance(self.points[-1], index_finger_tip) > 5:
                                self.points.append(index_finger_tip)
                            else:
                                self.points.append(index_finger_tip)

                        draw_points(result_image, self.points)
                else:
                    pass
            else:
                if self.state == State.TRACKING:
                    if time.time() - self.no_finger_timestamp > 2:
                        self.state = State.NULL
                        self.points = []

        except BaseException as e:
            rospy.logerr("e = {}".format(e))


        self.fps.update()
        self.fps.show_fps(result_image)
        result_image = cv2.cvtColor(result_image, cv2.COLOR_RGB2BGR)
        cv2.imshow('image', result_image)
        key = cv2.waitKey(1)

        if key == ord(' '): # 按空格清空已经记录的轨迹
            self.points = []
        if time.time() > self.gc_stamp:
            self.gc_stamp = time.time() + 1
            gc.collect()

if __name__ == "__main__":
    finger_track_node = FingerTrajectoryNode()
    while not rospy.is_shutdown():
```

```
        try:
            finger_track_node.image_proc()
        except Exception as e:
            rospy.logerr(str(e))
```