13.Mediapipe palm/arm targeting

Note: There are relevant running codes on Raspberry Pi and Jetson nano, but due to differences in motherboard performance, they may not run so smoothly. The supporting virtual machines also have running environments and programs installed.If the experience on the motherboard is not good, you can remove the camera and plug it into the virtual machine. Connect the camera device to the virtual machine to run the image processing program on the virtual machine and run the driver on the motherboard.The premise is that distributed communication needs to be set up between the motherboard and the virtual machine. Please refer to the previous tutorial for setting up.

13.1. 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 for building and using multiple 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 (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 Packet, Stream, Calculator, Graph and Subgraph.

Features of MediaPipe:

- End-to-end acceleration: Built-in fast ML inference and processing accelerates even on commodity hardware.
- Build once, deploy anywhere: Unified solution for Android, iOS, desktop/cloud, web and IoT.
- Ready-to-use solutions: cutting-edge ML solutions that showcase the full capabilities of the framework.
- Free and open source: frameworks and solutions under Apache2.0, fully extensible and customizable.

13.2. Start up

13.2.1. Program description

After the program is run, the camera captures the image, the program detects the palm, and prints the center coordinates of the palm to the terminal.

13.2.2. Program startup

Terminal input,

```
roscore
rosrun arm_mediapipe Find_Hand.py
```

13.2.3. Source code

Source code location：/home/yahboom/dofbot_ws/src/arm_mediapipe/scripts/Find_Hand.py

jetson-nano source code
location：/home/jetson/dofbot_ws/src/arm_mediapipe/scripts/Find_Hand.py

Raspberry Pi source code
location：/home/dofbot/dofbot_ws/src/arm_mediapipe/scripts/Find_Hand.py

```
#!/usr/bin/env python3
# encoding: utf-8
import threading
import numpy as np
from time import sleep, time
from media_library import *

class HandCtrlArm:
    def __init__(self):
        self.hand_detector = HandDetector()
        self.arm_status = True
        self.locking = True
        self.init = True
        self.pTime = 0
        self.add_lock = self.remove_lock = 0
        self.Joy_active = True
        self.event = threading.Event()
        self.event.set()
        self.Joy_active = True


    def process(self, frame):
        frame = cv.flip(frame, 1)
        if self.Joy_active:
            frame, lmList, bbox = self.hand_detector.findHands(frame)
            if len(lmList) != 0 and self.Joy_active:
                threading.Thread(target=self.find_hand_threading, args=(lmList,
bbox)).start()
        self.cTime = 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)
        return frame

    def find_hand_threading(self, lmList, bbox):
        fingers = self.hand_detector.fingersUp(lmList)
        self.hand_detector.draw = True
        angle = self.hand_detector.ThumbTOforefinger(lmList)
        value = np.interp(angle, [0, 70], [185, 20])
        indexX = (bbox[0] + bbox[2]) / 2
        indexY = (bbox[1] + bbox[3]) / 2
        print("indexX: ",indexX)
        print("indexY: ",indexY)
```

```python
if __name__ == '__main__':
    capture = cv.VideoCapture(0)
    capture.set(6, cv.VideoWriter.fourcc('M', 'J', 'P', 'G'))
    capture.set(cv.CAP_PROP_FRAME_WIDTH, 640)
    capture.set(cv.CAP_PROP_FRAME_HEIGHT, 480)
    print("capture get FPS : ", capture.get(cv.CAP_PROP_FPS))
    ctrl_arm = HandCtrlArm()
    while capture.isOpened():
        ret, frame = capture.read()
        action = cv.waitKey(1) & 0xFF
        frame = ctrl_arm.process(frame)
        if action == ord('q'):
            #ctrl_arm.media_ros.cancel()
            break
        cv.imshow('frame', frame)
    capture.release()
    cv.destroyAllWindows()
```