Mediapipe Gesture Recognition Robotic Arm Stacking

Note: There are related running codes on Raspberry Pi and Jetson nano, but due to the difference in motherboard performance, they may not run so smoothly. The supporting virtual machine also has the operating environment and programs installed. If the experience on the motherboard is not good, you can Remove the camera and insert 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 between the motherboard and the virtual machine needs to be set up. Refer to the previous tutorial to set it up.

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.

2. Start

2.1. Preparations before startup

Just like the color stacking of the robotic arm, fix the robotic arm on the map, and then place the four blocks in the four corner boxes. At this point, the preparations are complete.

2.2. Stacking program description

There are four stacking methods, and the corresponding gestures are:

- Gesture 1 (raised index finger): The robotic arm will stack four blocks on the map to form a vertical 1 (vertical)
- Gesture 2 (Yes gesture, index and middle fingers raised): The robotic arm will stack four blocks into a single row (single_row)

- Gesture 3 (thumb and little finger bent, other fingers raised): The robotic arm will stack four blocks into a biserial
- Gesture 4 (thumb bent, other fingers raised): The robotic arm will stack four blocks on the map into a T-shape (T_type)

2.3. Program startup

The motherboard runs:

```
roscore
rosrun arm_mediapipe arm_driver.py
rosrun arm_mediapipe HandCtrl.py
```

2.3. Source code

Raspberry Pi source code location:/root/dofbot_ws/src/arm_mediapipe/scripts/HandCtrl.py

```
#!/usr/bin/env python3
# encoding: utf-8
import threading
import cv2 as cv
import numpy as np
from media_library import *
from time import sleep, time

class PoseCtrlArm:
    def __init__(self):
        self.car_status = True
        self.stop_status = 0
        self.locking = False
        self.pose_detector = Holistic()
        self.hand_detector = HandDetector()
        self.pTime = self.index = 0
        self.media_ros = Media_ROS()
        self.Joy_active = True
        self.event = threading.Event()
        self.event.set()
        self.time_sleep = 1

    def process(self, frame):
        frame = cv.flip(frame, 1)
        if self.Joy_active:
            frame, lmList, _ = self.hand_detector.findHands(frame)
            if len(lmList) != 0:
                threading.Thread(target=self.hand_threading, args=
(lmList,)).start()
            else:self.media_ros.pub_vel(0, 0)

        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)
```

```
        self.media_ros.pub_imgMsg(frame)
        return frame

    def take_yellow(self):
        #take yellow
        self.media_ros.pub_arm([65, 22, 64, 56, 270, 60])
        sleep(self.time_sleep)
        self.media_ros.pub_arm([65, 22, 64, 56, 270, 135])
        sleep(self.time_sleep)

    def take_red(self):
        #take yellow
        self.media_ros.pub_arm([115, 22, 64, 56, 270, 60])
        sleep(self.time_sleep)
        self.media_ros.pub_arm([115, 22, 64, 56, 270, 135])
        sleep(self.time_sleep)

    def take_green(self):
        #take yellow
        self.media_ros.pub_arm([136, 66, 20, 29, 270, 60])
        sleep(self.time_sleep)
        self.media_ros.pub_arm([136, 66, 20, 29, 270, 135])
        sleep(self.time_sleep)

    def take_blue(self):
        #take yellow
        self.media_ros.pub_arm([44, 66, 20, 28, 270, 60])
        sleep(self.time_sleep)
        self.media_ros.pub_arm([44, 66, 20, 28, 270, 135])
        sleep(self.time_sleep)

    def Top(self):
        #Top
        self.media_ros.pub_arm([90, 80, 50, 50, 270, 60])
        sleep(self.time_sleep)

    def vertical(self):
        self.take_yellow()
        self.media_ros.pub_arm([91, 66, 19, 2, 90, 135])
        sleep(self.time_sleep)
        self.media_ros.pub_arm([91, 66, 19, 2, 90, 60])
        sleep(self.time_sleep)

        self.take_red()
        self.media_ros.pub_arm([90, 57, 32, 22, 90, 135])
        sleep(self.time_sleep)
        self.media_ros.pub_arm([90, 57, 32, 22, 90, 60])
        sleep(self.time_sleep)

        self.take_green()
        self.media_ros.pub_arm([89, 42, 43, 42, 90, 135])
        sleep(self.time_sleep)
        self.media_ros.pub_arm([89, 42, 43, 42, 90, 60])
        sleep(self.time_sleep)

        self.take_blue()
```

```
        self.media_ros.pub_arm([88, 16, 72, 54, 90, 135])
        sleep(self.time_sleep)
        self.media_ros.pub_arm([88, 16, 72, 54, 90, 60])
        sleep(self.time_sleep)
        self.Top()

    def single_row(self):
        self.take_yellow()
        self.media_ros.pub_arm([90, 48, 35, 30, 270, 135])
        sleep(self.time_sleep)
        self.media_ros.pub_arm([90, 48, 35, 30, 270, 60])
        sleep(self.time_sleep)

        self.take_red()
        self.media_ros.pub_arm([90, 65, 25, 36, 270, 135])
        sleep(self.time_sleep)
        self.media_ros.pub_arm([90, 65, 25, 36, 270, 60])
        sleep(self.time_sleep)

        self.take_green()
        self.media_ros.pub_arm([90, 65, 44, 17, 270, 135])
        sleep(self.time_sleep)
        self.media_ros.pub_arm([90, 65, 44, 17, 270, 60])
        sleep(self.time_sleep)

        self.take_blue()
        self.media_ros.pub_arm([90, 76, 40, 17, 270, 135])
        sleep(self.time_sleep)
        self.media_ros.pub_arm([90, 76, 40, 17, 270, 60])
        sleep(self.time_sleep)
        self.Top()

    def biserial(self):
        self.take_yellow()
        self.media_ros.pub_arm([74, 18, 94, 4, 85, 135])
        sleep(self.time_sleep)
        self.media_ros.pub_arm([74, 18, 94, 4, 85, 60])
        sleep(self.time_sleep)

        self.take_red()
        self.media_ros.pub_arm([73, 55, 43, 27, 85, 135])
        sleep(self.time_sleep)
        self.media_ros.pub_arm([73, 55, 43, 27, 85, 60])
        sleep(self.time_sleep)

        self.take_green()
        self.media_ros.pub_arm([104, 45, 49, 26, 107, 135])
        sleep(self.time_sleep)
        self.media_ros.pub_arm([104, 45, 49, 26, 107, 60])
        sleep(self.time_sleep)

        self.take_blue()
        self.media_ros.pub_arm([103, 49, 55, 20, 107, 135])
        sleep(self.time_sleep)
        self.media_ros.pub_arm([103, 49, 55, 20, 107, 60])
        sleep(self.time_sleep)
```

```
        self.Top()

    def T_type(self):
        self.take_yellow()
        self.media_ros.pub_arm([90, 37, 75, 1, 90, 135])
        sleep(self.time_sleep)
        self.media_ros.pub_arm([90, 37, 75, 1, 90, 60])
        sleep(self.time_sleep)

        self.take_red()
        self.media_ros.pub_arm([64, 27, 84, 6, 88, 135])
        sleep(self.time_sleep)
        self.media_ros.pub_arm([64, 27, 84, 6, 88, 60])
        sleep(self.time_sleep)

        self.take_green()
        self.media_ros.pub_arm([113, 22, 83, 16, 88, 135])
        sleep(self.time_sleep)
        self.media_ros.pub_arm([113, 22, 83, 16, 88, 60])
        sleep(self.time_sleep)

        self.take_blue()
        self.media_ros.pub_arm([88, 4, 93, 50, 88, 135])
        sleep(self.time_sleep)
        self.media_ros.pub_arm([88, 4, 93, 50, 88, 60])
        sleep(self.time_sleep)
        self.Top()

    def hand_threading(self, lmList):
        if self.event.is_set():
            self.event.clear()
            self.stop_status = 0
            self.index = 0
            fingers = self.hand_detector.fingersUp(lmList)
            # print("fingers: ", fingers)
            if sum(fingers) == 1 and fingers[1] == 1: #食指1
                print("vertical")   #竖直
                self.vertical()
            if sum(fingers) == 2 and fingers[1] == 1 and fingers[2] == 1: #食指中
指2
                print("single_row") #单列
                self.vertical()
            if sum(fingers) == 3 and fingers[0] == 0 and fingers[4] == 0: #食指中
指无名指3
                print("biserial")   #双列
                self.biserial()

            if sum(fingers) == 4 and fingers[0] == 0: #拇指弯曲4
                print("T_type") #T型
                self.T_type()
            self.event.set()

if __name__ == '__main__':
    rospy.init_node('PoseCtrlArm_node', anonymous=True)
    pose_ctrl_arm = PoseCtrlArm()
    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))
    while capture.isOpened():
        ret, frame = capture.read()
        frame = pose_ctrl_arm.process(frame)
        if cv.waitKey(1) & 0xFF == ord('q'): break
        cv.imshow('frame', frame)
    capture.release()
    cv.destroyAllWindows()
```