

Mediapipe palm arm target positioning

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 (such as Raspberry Pi), 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 showcases the full capabilities of the framework.
- Free and open source: framework and solution under Apache 2.0, fully extensible and customizable.

2. Start

2.1 Program description

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

2.2 Start program

- Input following command to start the program

```
roscore
roslaunch jetcobot_mediapipe Find_Hand.py
```

2.3 About code

Code path: ~/jetcobot_ws/src/jetcobot_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)

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()
```