Palm Targeting

Orin board users can directly open the terminal and input the tutorial commands to run directly. Jetson-Nano board users need to enter the docker container first, then input the tutorial commands in the docker to start the program.

1. Introduction

MediaPipe is a data stream processing machine learning application development framework developed and open-sourced by Google. It is a graph-based data processing pipeline used to build applications that use 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 (Jetson nano, 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.

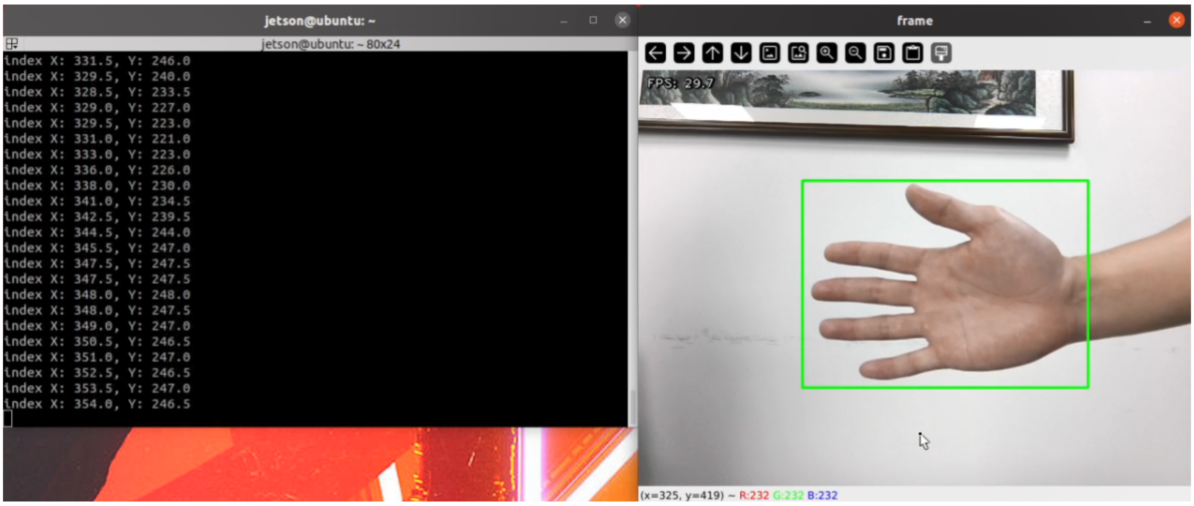
Features of MediaPipe:

- End-to-end acceleration: Built-in fast ML inference and processing accelerates even on ordinary 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: Framework and solutions under Apache2.0, fully scalable and customizable.

2. Launch

2.1. Program Description

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



```
index X: 331.5, Y: 246.0
index X: 329.5, Y: 240.0
index X: 328.5, Y: 233.5
index X: 329.0, Y: 227.0
index X: 329.5, Y: 223.0
index X: 331.0, Y: 221.0
index X: 333.0, Y: 223.0
index X: 336.0, Y: 226.0
index X: 338.0, Y: 230.0
index X: 341.0, Y: 234.5
index X: 342.5, Y: 239.5
index X: 344.5, Y: 244.0
index X: 345.5, Y: 247.0
index X: 347.5, Y: 247.5
index X: 347.5, Y: 247.5
index X: 348.0, Y: 248.0
index X: 348.0, Y: 247.5
index X: 349.0, Y: 247.0
index X: 350.5, Y: 246.5
index X: 351.0, Y: 247.0
index X: 352.5, Y: 246.5
index X: 353.5, Y: 247.0
index X: 354.0, Y: 246.5
```

2.2. Program Launch

- Enter the following command to start the program

```
ros2 run dofbot_pro_mediapipe 13_FindHand
```

Press the q key in the image or press Ctrl+c in the terminal to exit the program.

2.3. Source Code

Code path:

```
# Jetson-Nano users need to enter the docker container to view
~/dofbot_pro_ws/src/dofbot_pro_mediapipe/dofbot_pro_mediapipe/13_FindHand.py
```

```python
#!/usr/bin/env python3
# encoding: utf-8
import threading
import numpy as np
import time
import os
import sys
import cv2 as cv
from dofbot_utils.robot_controller import Robot_Controller
from dofbot_utils.fps import FPS
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import Image
from std_msgs.msg import Bool
from geometry_msgs.msg import Twist
from cv_bridge import CvBridge

sys.path.append('/home/jetson/dofbot_pro_ws/src/dofbot_pro_mediapipe/dofbot_pro_mediapipe')
from media_library import *

class HandCtrlArmNode(Node):
    def __init__(self):
        super().__init__('hand_ctrl_arm_node')

        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.event = threading.Event()
        self.event.set()

        self.robot = Robot_Controller()
        self.robot.move_init_pose()

        self.bridge = CvBridge()
        self.publisher_ = self.create_publisher(Image, 'processed_image', 10)
        self.capture = cv.VideoCapture(0, cv.CAP_V4L2)
        self.capture.set(cv.CAP_PROP_FRAME_WIDTH, 640)
        self.capture.set(cv.CAP_PROP_FRAME_HEIGHT, 480)
        self.get_logger().info(f"Camera FPS:
{self.capture.get(cv.CAP_PROP_FPS)}")

        self.timer = self.create_timer(0.1, self.timer_callback)

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

    def find_hand_threading(self, lmList, bbox):
        fingers = self.hand_detector.fingersUp(lmList)
        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("index X: %.1f, Y: %.1f" % (indexX, indexY))

    def timer_callback(self):
        ret, frame = self.capture.read()
        if ret:
            frame = self.process(frame)
            processed_image_msg = self.bridge.cv2_to_imgmsg(frame, "bgr8")
            self.publisher_.publish(processed_image_msg)
            cv.imshow('frame', frame)
            if cv.waitKey(1) & 0xFF == ord('q'):
                self.capture.release()
                cv.destroyAllWindows()
                rclpy.shutdown()

def main(args=None):
    rclpy.init(args=args)
    hand_ctrl_arm_node = HandCtrlArmNode()
    rclpy.spin(hand_ctrl_arm_node)
    hand_ctrl_arm_node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```