

Palm Targeting

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 Jetson nano), 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.

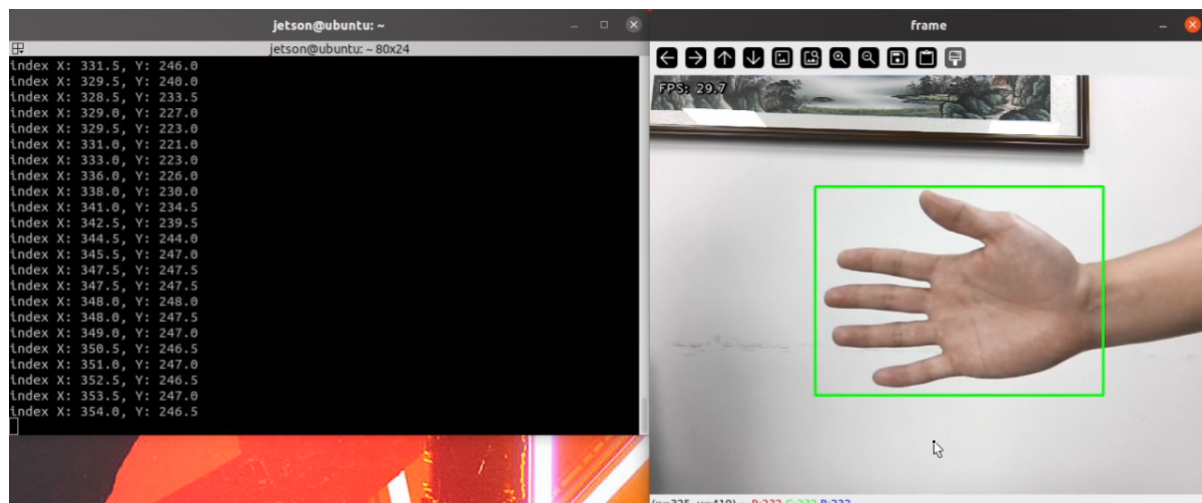
Features of MediaPipe:

- 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 demonstrates the full capabilities of the framework.
- Free and open source: framework and solution under Apache2.0, fully extensible and customizable.

2. Startup

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, Program Startup

- 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:

```
~/dofbot_pro_ws/src/dofbot_pro_mediapipe/dofbot_pro_mediapipe/13_FindHand.py
```

```
#!/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()

```