10. Finger control

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

10.2, Finger control

Click the [f key] to switch the recognition effect, and the distance between the thumb and index finger (open/closed) can control the effect of the image.

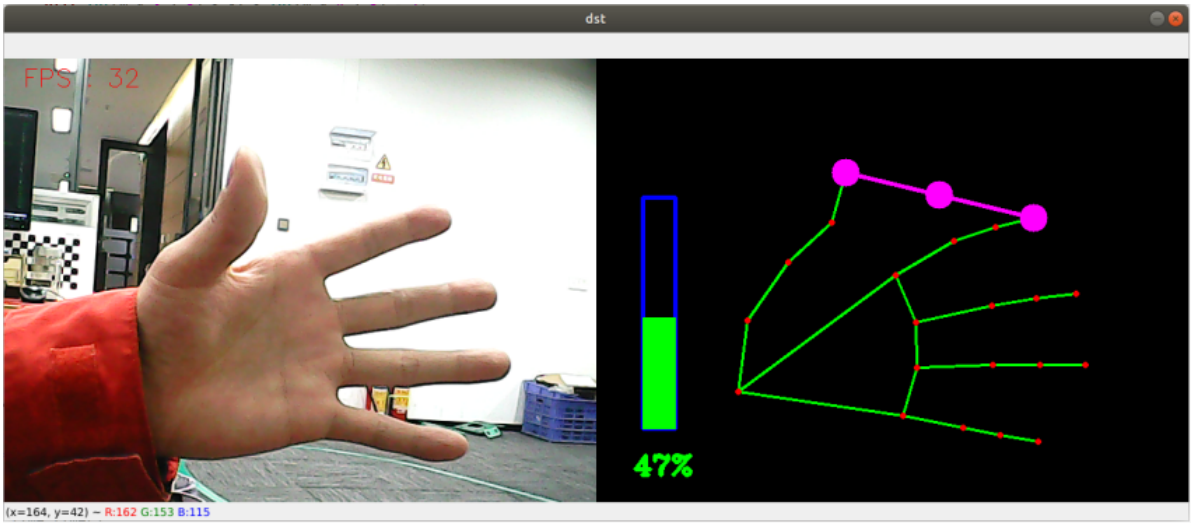
10.2.1, Start

Start the camera

```
roslaunch ascamera hp60c.launch
```

Terminal input,

```
roslaunch yahboomcar_mediapipe 10_HandCtrl.launch
```



10.2.2, Source code

Source code location:
/home/yahboom/ascam_ws/src/yahboomcar_mediapipe/scripts/10_HandCtrl.py

```
#!/usr/bin/env python3
# encoding: utf-8
import math
import time
import cv2 as cv
import numpy as np
import mediapipe as mp
import rospy
from sensor_msgs.msg import Image
from cv_bridge import CvBridge


pTime = cTime = volPer = value = index = 0
effect = ["color", "thresh", "blur", "hue", "enhance"]
volBar = 400


mpHands = mp.solutions.hands
mpDraw = mp.solutions.drawing_utils
hands = mpHands.Hands(static_image_mode=False, max_num_hands=2,
min_detection_confidence=0.5, min_tracking_confidence=0.5)
lmDrawSpec = mp.solutions.drawing_utils.DrawingSpec(color=(0, 0, 255),
thickness=-1, circle_radius=15)
drawSpec = mp.solutions.drawing_utils.DrawingSpec(color=(0, 255, 0),
thickness=10, circle_radius=10)

def get_dist(point1, point2):
    x1, y1 = point1
    x2, y2 = point2
    return abs(math.sqrt(math.pow(abs(y1 - y2), 2) + math.pow(abs(x1 - x2), 2)))

def calc_angle(pt1, pt2, pt3, lmList):
    point1 = lmList[pt1][1], lmList[pt1][2]
    point2 = lmList[pt2][1], lmList[pt2][2]
    point3 = lmList[pt3][1], lmList[pt3][2]
```

```
    a = get_dist(point1, point2)
    b = get_dist(point2, point3)
    c = get_dist(point1, point3)
    try:
        radian = math.acos((math.pow(a, 2) + math.pow(b, 2) - math.pow(c, 2)) /
(2 * a * b))
        angle = radian / math.pi * 180
    except:
        angle = 0
    return abs(angle)

def find_hands(frame):
    img = np.zeros(frame.shape, np.uint8)
    img_RGB = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
    results = hands.process(img_RGB)
    if results.multi_hand_landmarks:
        for handLms in results.multi_hand_landmarks:
            mpDraw.draw_landmarks(img, handLms, mpHands.HAND_CONNECTIONS)
    return img, results

def find_position(frame, results, draw=True):
    lmList = []
    if results.multi_hand_landmarks:
        for id, lm in enumerate(results.multi_hand_landmarks[0].landmark):
            h, w, c = frame.shape
            cx, cy = int(lm.x * w), int(lm.y * h)
            lmList.append([id, cx, cy])
            if draw:
                cv.circle(frame, (cx, cy), 15, (0, 0, 255), cv.FILLED)
    return lmList

def frame_combine(frame, src):
    if len(frame.shape) == 3:
        frameH, frameW = frame.shape[:2]
        srcH, srcW = src.shape[:2]
        dst = np.zeros((max(frameH, srcH), frameW + srcW, 3), np.uint8)
        dst[:, :frameW] = frame[:, :]
        dst[:, frameW:] = src[:, :]
    else:
        src = cv.cvtColor(src, cv.COLOR_BGR2GRAY)
        frameH, frameW = frame.shape[:2]
        imgH, imgW = src.shape[:2]
        dst = np.zeros((frameH, frameW + imgW), np.uint8)
        dst[:, :frameW] = frame[:, :]
        dst[:, frameW:] = src[:, :]
    return dst

def apply_effect(frame, effect, value):
    global volBar, volPer, index

    if effect == "thresh":
        gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
        frame = cv.threshold(gray, value, 255, cv.THRESH_BINARY)[1]
    elif effect == "blur":
```

```
        frame = cv.GaussianBlur(frame, (21, 21), np.interp(value, [0, 255], [0,
11]))
    elif effect == "hue":
        frame = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
        frame[:, :, 0] += int(value)
        frame = cv.cvtColor(frame, cv.COLOR_HSV2BGR)
    elif effect == "enhance":
        enh_val = value / 40
        clahe = cv.createCLAHE(clipLimit=enh_val, tileGridSize=(8, 8))
        lab = cv.cvtColor(frame, cv.COLOR_BGR2LAB)
        lab[:, :, 0] = clahe.apply(lab[:, :, 0])
        frame = cv.cvtColor(lab, cv.COLOR_LAB2BGR)

    return frame

def image_callback(msg):
    global pTime, volBar, volPer, value, index

    try:
        frame = bridge.imgmsg_to_cv2(msg, "bgr8")
    except Exception as e:
        rospy.logerr("Error converting image: %s", str(e))
        return

    img, results = find_hands(frame)
    lmList = find_position(frame, results, draw=False)

    if len(lmList) != 0:
        angle = calc_angle(4, 0, 8, lmList)
        x1, y1 = lmList[4][1], lmList[4][2]
        x2, y2 = lmList[8][1], lmList[8][2]
        cx, cy = (x1 + x2) // 2, (y1 + y2) // 2
        cv.circle(img, (x1, y1), 15, (255, 0, 255), cv.FILLED)
        cv.circle(img, (x2, y2), 15, (255, 0, 255), cv.FILLED)
        cv.line(img, (x1, y1), (x2, y2), (255, 0, 255), 3)
        cv.circle(img, (cx, cy), 15, (255, 0, 255), cv.FILLED)
        if angle <= 10:
            cv.circle(img, (cx, cy), 15, (0, 255, 0), cv.FILLED)
        volBar = np.interp(angle, [0, 70], [400, 150])
        volPer = np.interp(angle, [0, 70], [0, 100])
        value = np.interp(angle, [0, 70], [0, 255])

    frame = apply_effect(frame, effect[index], value)

    cTime = time.time()
    fps = 1 / (cTime - pTime)
    pTime = cTime
    text = "FPS : " + str(int(fps))

    cv.rectangle(img, (50, 150), (85, 400), (255, 0, 0), 3)
    cv.rectangle(img, (50, int(volBar)), (85, 400), (0, 255, 0), cv.FILLED)
    cv.putText(img, f'{int(volPer)}%', (40, 450), cv.FONT_HERSHEY_COMPLEX, 1,
(0, 255, 0), 3)
    cv.putText(frame, text, (20, 30), cv.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255),
1)
```

```
    dst = frame_combine(frame, img)
    cv.imshow('dst', dst)

    action = cv.waitKey(1) & 0xFF
    if action == ord('q'):
        rospy.signal_shutdown("User quit")

    if action == ord('f'):
        index += 1
        if index >= len(effect): index = 0


if __name__ == '__main__':
    rospy.init_node('hand_ctrl_node')
    bridge = CvBridge()
    image_sub = rospy.Subscriber('/ascamera_hp60c/rgb0/image', Image,
image_callback)
    rospy.spin()
```