

2. Attitude detection

2.1. 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.

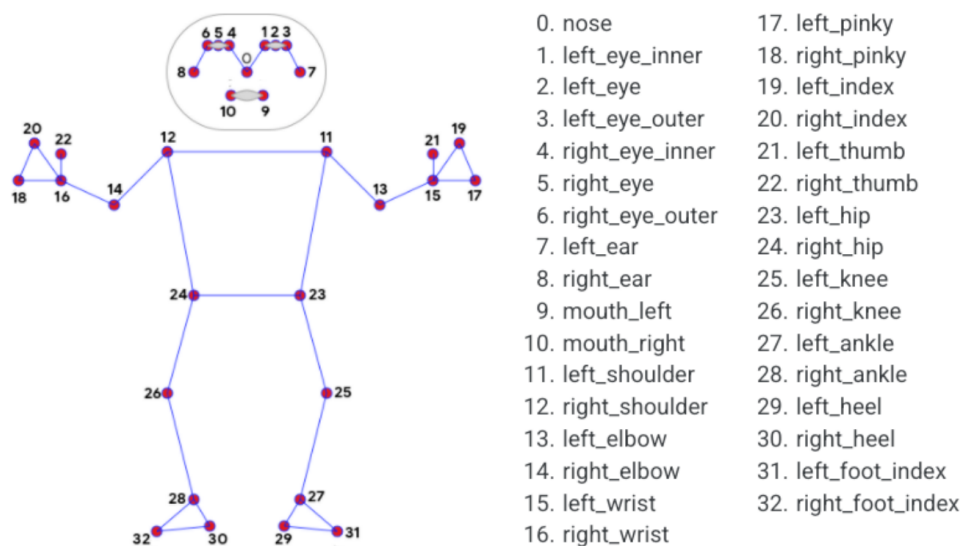
MediaPipe Features:

- 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.

2.2, MediaPipe Pose

MediaPipe Pose is an ML solution for high-fidelity body pose tracking, using BlazePose research, 33 3D coordinates and full background segmentation masks are inferred from RGB video frames. This research also provides motivation for the ML Kit pose detection API.

The landmark model in MediaPipe poses predicts the positions of 33 pose coordinates (see figure below).



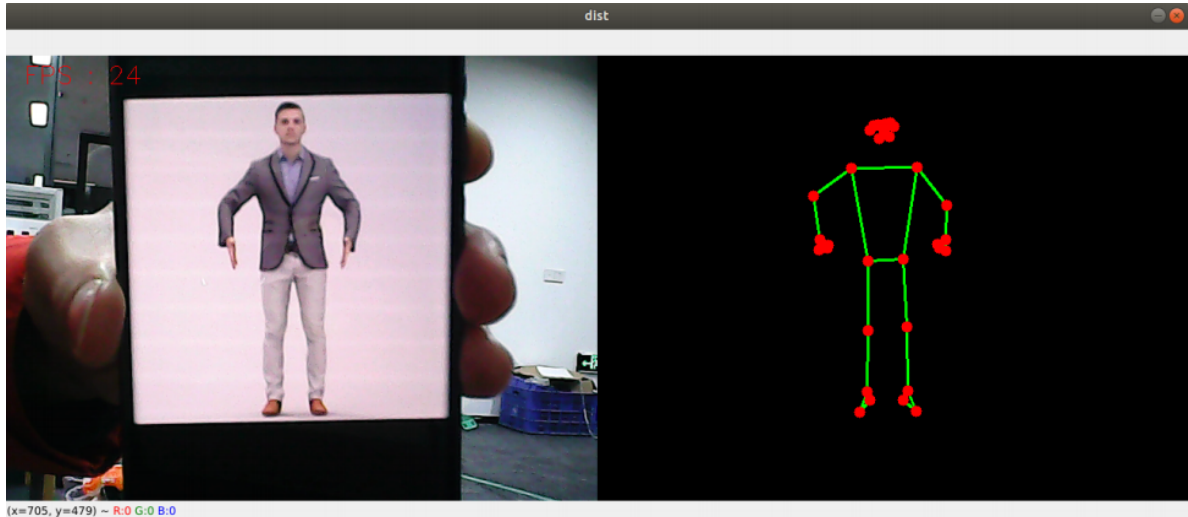
2.3. Attitude detection

2.3.1.Startup

Note: Before running this case, please make sure that the [/dev/video0] device has been successfully mounted into the docker container, otherwise the camera will not be opened.

Enter the docker container and execute:

```
ros2 run yahboomcar_mediapipe 02_PoseDetector
```



To view the point cloud data and node topic communication diagram, please refer to the [Hand Detection] case course.

2.3.2. Source code

Source code location:

```
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_mediapipe/yahboomcar_mediapipe/02_PoseDetector.py
```

```
#!/usr/bin/env python3
# encoding: utf-8

#import ros lib
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Point
import mediapipe as mp
#import define msg
from yahboomcar_msgs.msg import PointArray
#import common lib
import cv2 as cv
import numpy as np
import time
print("import done")
```

```

class PoseDetector(Node):
    def __init__(self, name, mode=False, smooth=True, detectionCon=0.5,
trackCon=0.5):
        super().__init__(name)
        self.mpPose = mp.solutions.pose
        self.mpDraw = mp.solutions.drawing_utils
        self.pose = self.mpPose.Pose(
            static_image_mode=mode,
            smooth_landmarks=smooth,
            min_detection_confidence=detectionCon,
            min_tracking_confidence=trackCon )
        self.pub_point =
self.create_publisher(PointArray, '/mediapipe/points', 1000)
        self.lmDrawSpec = mp.solutions.drawing_utils.DrawingSpec(color=(0, 0,
255), thickness=-1, circle_radius=6)
        self.drawSpec = mp.solutions.drawing_utils.DrawingSpec(color=(0, 255,
0), thickness=2, circle_radius=2)

    def pubPosePoint(self, frame, draw=True):
        pointArray = PointArray()
        img = np.zeros(frame.shape, np.uint8)
        img_RGB = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
        self.results = self.pose.process(img_RGB)
        if self.results.pose_landmarks:
            if draw: self.mpDraw.draw_landmarks(frame,
self.results.pose_landmarks, self.mpPose.POSE_CONNECTIONS, self.lmDrawSpec,
self.drawSpec)
            self.mpDraw.draw_landmarks(img, self.results.pose_landmarks,
self.mpPose.POSE_CONNECTIONS, self.lmDrawSpec, self.drawSpec)
            for id, lm in enumerate(self.results.pose_landmarks.landmark):
                point = Point()
                point.x, point.y, point.z = lm.x, lm.y, lm.z
                pointArray.points.append(point)
        self.pub_point.publish(pointArray)
        return frame, img

    def frame_combine(self, 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 main():
    print("start it")

```

```

rcipy.init()
pose_detector = PoseDetector('pose_detector')
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))
pTime = cTime = 0
index = 3
while capture.isOpened():
    ret, frame = capture.read()
    # frame = cv.flip(frame, 1)
    frame, img = pose_detector.pubPosePoint(frame, draw=False)
    if cv.waitKey(1) & 0xFF == ord('q'): break
    cTime = time.time()
    fps = 1 / (cTime - pTime)
    pTime = cTime
    text = "FPS : " + str(int(fps))
    cv.putText(frame, text, (20, 30), cv.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0,
255), 1)
    dist = pose_detector.frame_combine(frame, img)
    cv.imshow('dist', dist)
    # cv.imshow('frame', frame)
    # cv.imshow('img', img)
capture.release()
cv.destroyAllWindows()

```