

# Overall detection

---

## 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.MediaPipe Hands

Refer to the 【hand detection】 course.

## 3.MediaPipe Pose

Refer to the 【Posture detection】 course.

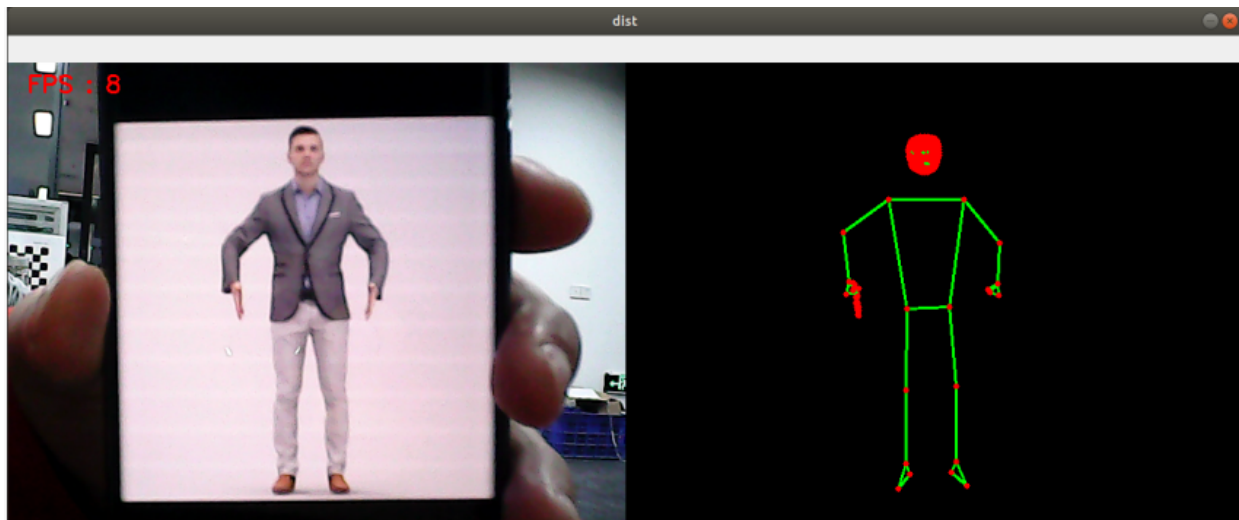
## 4. Overall detection

Combining the content of the previous course, this course can detect both palms and human bodies.

### 4.1 Start

- Input following command to start the program

```
roscore  
roslaunch jetcobot_mediapipe 03_Holistic.py
```



## 4.2 Code

Code path: ~/jetcobot\_ws/src/jetcobot\_mediapipe/scripts/03\_Holistic.py

```
#!/usr/bin/env python3
# encoding: utf-8
import time
import rospy
import cv2 as cv
import numpy as np
import mediapipe as mp
from geometry_msgs.msg import Point
from yahboomcar_msgs.msg import PointArray

class Holistic:
    def __init__(self, staticMode=False, landmarks=True, detectionCon=0.5,
trackingCon=0.5):
        self.mpHolistic = mp.solutions.holistic
        self.mpFaceMesh = mp.solutions.face_mesh
        self.mpHands = mp.solutions.hands
        self.mpPose = mp.solutions.pose
        self.mpDraw = mp.solutions.drawing_utils
        self.mpholistic = self.mpHolistic.Holistic(
            static_image_mode=staticMode,
            smooth_landmarks=landmarks,
            min_detection_confidence=detectionCon,
            min_tracking_confidence=trackingCon)
        self.pub_point = rospy.Publisher('/mediapipe/points', PointArray,
queue_size=1000)
        self.lmDrawSpec = mp.solutions.drawing_utils.DrawingSpec(color=(0, 0, 255),
thickness=-1, circle_radius=3)
        self.drawSpec = mp.solutions.drawing_utils.DrawingSpec(color=(0, 255, 0),
thickness=2, circle_radius=2)

    def findHolistic(self, frame, draw=True):
```

```

pointArray = PointArray()
img = np.zeros(frame.shape, np.uint8)
img_RGB = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
self.results = self.mpholistic.process(img_RGB)
if self.results.face_landmarks:
    if draw: self.mpDraw.draw_landmarks(frame, self.results.face_landmarks,
self.mpFaceMesh.FACEMESH_CONTOURS, self.lmDrawSpec, self.drawSpec)
    self.mpDraw.draw_landmarks(img, self.results.face_landmarks,
self.mpFaceMesh.FACEMESH_CONTOURS, self.lmDrawSpec, self.drawSpec)
    for id, lm in enumerate(self.results.face_landmarks.landmark):
        point = Point()
        point.x, point.y, point.z = lm.x, lm.y, lm.z
        pointArray.points.append(point)
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)
if self.results.left_hand_landmarks:
    if draw: self.mpDraw.draw_landmarks(frame,
self.results.left_hand_landmarks, self.mphHands.HAND_CONNECTIONS, self.lmDrawSpec,
self.drawSpec)
    self.mpDraw.draw_landmarks(img, self.results.left_hand_landmarks,
self.mphHands.HAND_CONNECTIONS, self.lmDrawSpec, self.drawSpec)
    for id, lm in enumerate(self.results.left_hand_landmarks.landmark):
        point = Point()
        point.x, point.y, point.z = lm.x, lm.y, lm.z
        pointArray.points.append(point)
if self.results.right_hand_landmarks:
    if draw: self.mpDraw.draw_landmarks(frame,
self.results.right_hand_landmarks, self.mphHands.HAND_CONNECTIONS, self.lmDrawSpec,
self.drawSpec)
    self.mpDraw.draw_landmarks(img, self.results.right_hand_landmarks,
self.mphHands.HAND_CONNECTIONS, self.lmDrawSpec, self.drawSpec)
    for id, lm in enumerate(self.results.right_hand_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

if __name__ == '__main__':
    rospy.init_node('Holistic', anonymous=True)
    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
    holistic = Holistic()
    while capture.isOpened():
        ret, frame = capture.read()
        # frame = cv.flip(frame, 1)
        frame, img = holistic.findHolistic(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, 0.8, (0, 0, 255),
2)

        dist = holistic.frame_combine(frame, img)
        cv.imshow('dist', dist)
        # cv.imshow('frame', frame)
        # cv.imshow('img', img)
    capture.release()
    cv.destroyAllWindows()

```