

4. Face Detection

4.1. Introduction

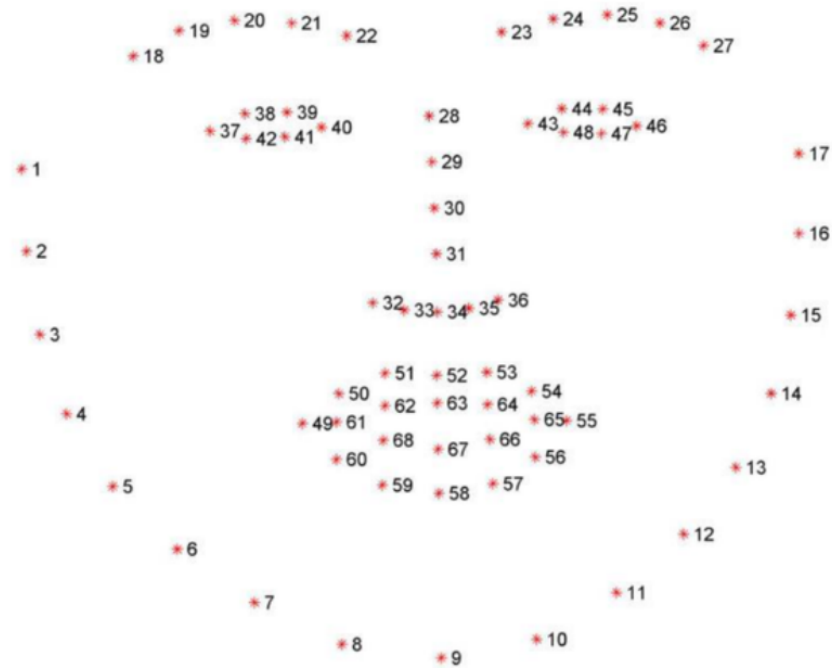
MediaPipe is an open-source data stream processing machine learning application development framework developed by Google. It is a graph-based data processing pipeline used to build data sources in various forms, such as video, audio, sensor data, and any time series data. MediaPipe is cross-platform and can run on embedded platforms (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 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 commodity hardware.
- Build once, deploy anywhere anytime: 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.

4.2, Dlib

DLIB is a modern C++ toolkit containing machine learning algorithms and tools for creating complex software in C++ to solve real-world problems. It is widely used by industry and academia in fields such as robotics, embedded devices, mobile phones and large high-performance computing environments. The dlib library uses 68 points to mark important parts of the face, such as 18-22 points to mark the right eyebrow, and 51-68 points to mark the mouth. Use the `get_frontal_face_detector` module of the dlib library to detect the face, and use the `shape_predictor_68_face_landmarks.dat` feature data to predict the facial feature values.



4.3, face detection

4.3.1, startup

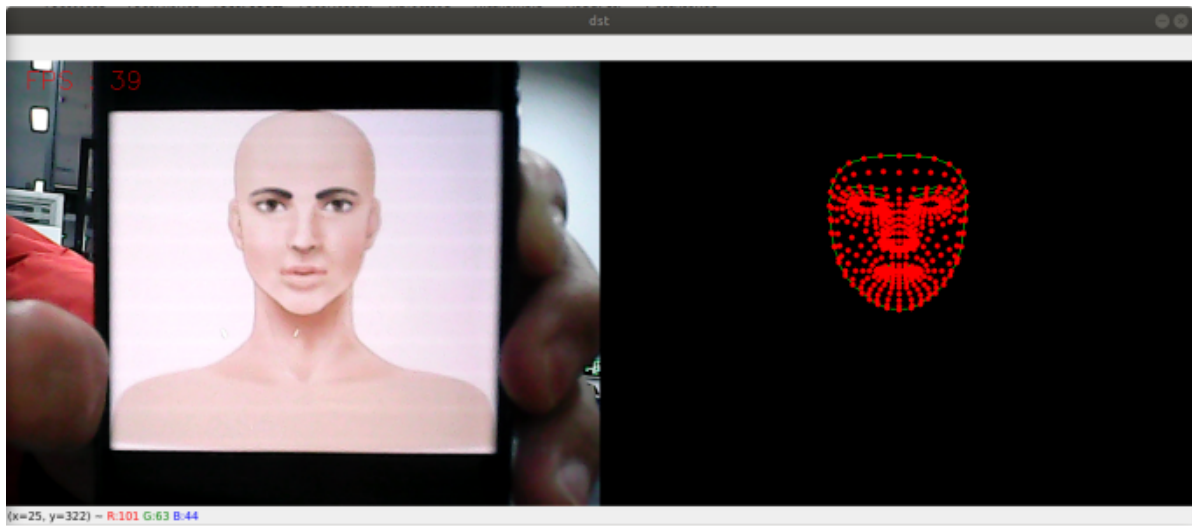
Terminal input,

Please do this step before running the program

```
sudo supervisorctl stop ChassisServer #Turn off the self-starting chassis
service
#Indoor version NAVROBO-astra_pro2 camera executes this command
/home/yahboom/YBAMR-COBOT-EDU-00001/start/OBColorViewer #Release color stream
video100
#[info][722318][Pipeline.cpp:251] start streams done!
#[info][722318][Pipeline.cpp:234] Pipeline start done! #[warning][722318]
[Pipeline.cpp:327] wait for frame timeout, you can try to increase the wait time!
current timeout=100
```

```
roslaunch yahboom_navrobo_mediapipe cloud_viewer.launch # Point cloud viewing:
support 01~04
```

```
roslaunch yahboom_navrobo_mediapipe 04_Face Mesh.launch # Face detection
```



4.3.2. Source code

Source code location: /home/yahboom/YBAMR-COBOT-EDU-00001/src/yahboom_navrobo_mediapipe/scripts/04_FaceMesh.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 FaceMesh:
    def __init__(self, staticMode=False, maxFaces=2, minDetectionCon=0.5,
minTrackingCon=0.5):
        self.mpDraw = mp.solutions.drawing_utils
        self.mpFaceMesh = mp.solutions.face_mesh
        self.faceMesh = self.mpFaceMesh.FaceMesh(
            static_image_mode=staticMode,
            max_num_faces=maxFaces,
            min_detection_confidence=minDetectionCon,
            min_tracking_confidence=minTrackingCon )
        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 = self.mpDraw.DrawingSpec(color=(0, 255, 0), thickness=1,
circle_radius=1)

    def pubFaceMeshPoint(self, frame, draw=True):
        pointArray = PointArray()
        img = np.zeros(frame.shape, np.uint8)
        imgRGB = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
        self.results = self.faceMesh.process(imgRGB)
        if self.results.multi_face_landmarks:
            for i in range(len(self.results.multi_face_landmarks)):
```

```

        if draw: self.mpDraw.draw_landmarks(frame,
self.results.multi_face_landmarks[i], self.mpFaceMesh.FACEMESH_CONTOURS,
self.lmDrawSpec, self.drawSpec)
        self.mpDraw.draw_landmarks(img,
self.results.multi_face_landmarks[i], self.mpFaceMesh.FACEMESH_CONTOURS,
self.lmDrawSpec, self.drawSpec)
        for id, lm in
enumerate(self.results.multi_face_landmarks[i].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('FaceMesh', 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, 0
    face_mesh = FaceMesh(maxFaces=2)
    while capture.isOpened():
        ret, frame = capture.read()
        # frame = cv.flip(frame, 1)
        frame, img = face_mesh.pubFaceMeshPoint(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)
        dst = face_mesh.frame_combine(frame, img)
        cv.imshow('dst', dst)
        # cv.imshow('frame', frame)
        # cv.imshow('img', img)
    capture.release()
    cv.destroyAllWindows()

```

