Face Mesh

4.1. Introduction

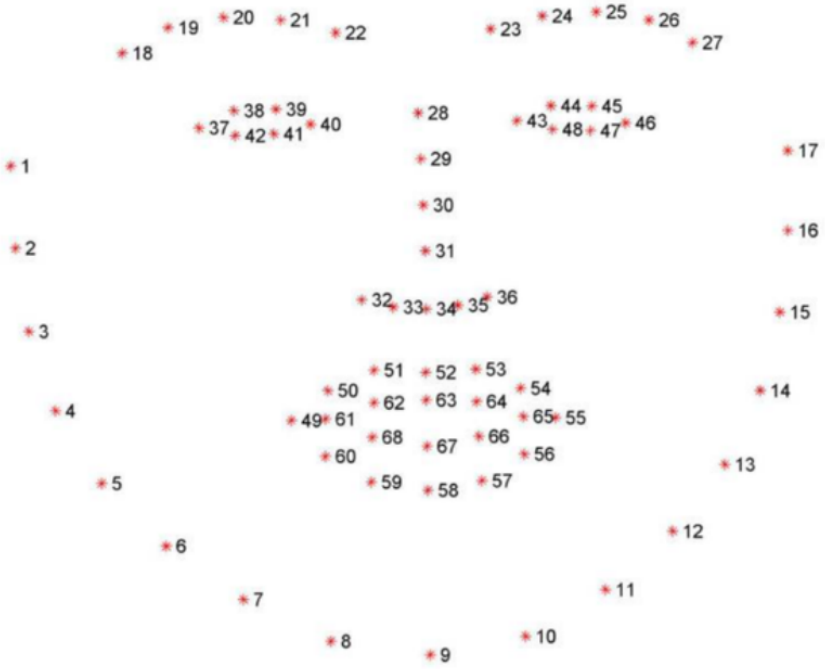
MediaPipe is an open-source data stream processing machine learning application development framework developed by Google. It is a graph-based data processing pipeline used to build data sources in various forms, such as video, audio, sensor data, and any time series data. MediaPipe is cross-platform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

4.2, Dlib

DLIB is a modern C++ toolkit that contains 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 mark the right eyebrow, 51-68 marks the mouth. Use the get_frontal_face_detector module of the dlib library to detect faces, and use the shape_predictor_68_face_landmarks.dat feature data to predict facial feature values.



4.3, Face mesh

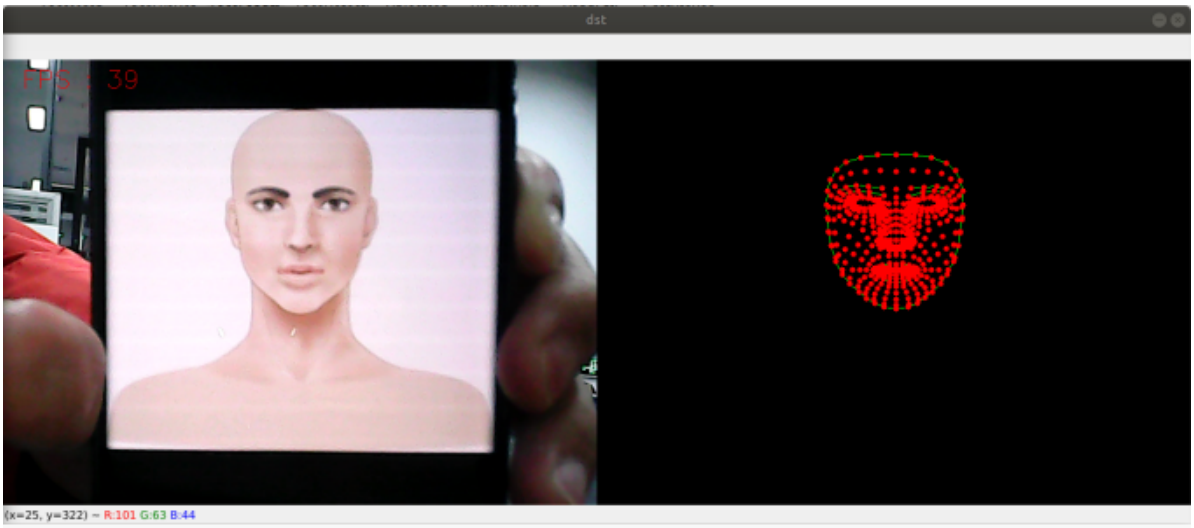
4.3.1, Startup

Start the camera

```
ros2 launch ascamera hp60c.launch.py
```

Open a new terminal and enter,

```
ros2 run yahboomcar_mediapipe 04_FaceMesh
```



4.3.2, Source code

Source code location:
~/ascam_ros2_ws/src/yahboomcar_mediapipe/yahboomcar_mediapipe/04_FaceMesh.py

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

import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Point
from sensor_msgs.msg import Image
import mediapipe as mp
from yahboomcar_msgs.msg import PointArray
import cv2 as cv
import numpy as np
import time
from cv_bridge import CvBridge

print("import done")


class FaceMesh(Node):
    def __init__(self, name, staticMode=False, maxFaces=2, minDetectionCon=0.5,
minTrackingCon=0.5):
        super().__init__(name)
        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
        )
        # Publisher for detected points
        self.pub_point = self.create_publisher(PointArray, '/mediapipe/points',
1000)
        # Initialize CvBridge
        self.bridge = CvBridge()
        # Create a subscriber to the image topic
        self.create_subscription(Image,
'/ascamera_hp60c/camera_publisher/rgb0/image', self.image_callback, 10)

        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)

        self.pTime = 0
        self.exit_flag = False  # Flag for exit condition

    def image_callback(self, msg):
        # Convert the ROS Image message to OpenCV format
        frame = self.bridge.imgmsg_to_cv2(msg, desired_encoding='bgr8')
```

```
        # Process the frame for face mesh detection
        frame, img = self.pubFaceMeshPoint(frame, draw=False)

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

        # Display FPS on frame
        cv.putText(frame, text, (20, 30), cv.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0,
255), 1)

        # Combine the original frame and the face mesh result
        dst = self.frame_combine(frame, img)

        # Show the combined image
        cv.imshow('FaceMesh Detection', dst)

        # Exit the program if 'q' is pressed
        if cv.waitKey(1) & 0xFF == ord('q'):
            self.exit_flag = True

    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)

        # Publish the detected points
        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 run(self):
        # Custom loop for handling ROS 2 callback and OpenCV events
        while rclpy.ok() and not self.exit_flag:
            rclpy.spin_once(self)  # Process one callback
            if self.exit_flag:
                break
        cv.destroyAllWindows()


def main():
    print("start it")
    rclpy.init()
    face_mesh = FaceMesh('face_mesh')

    try:
        face_mesh.run()
    except KeyboardInterrupt:
        pass
    finally:
        face_mesh.destroy_node()
        rclpy.shutdown()


if __name__ == '__main__':
    main()
```