5. Face recognition

5.1. 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.

5.2, Face recognition

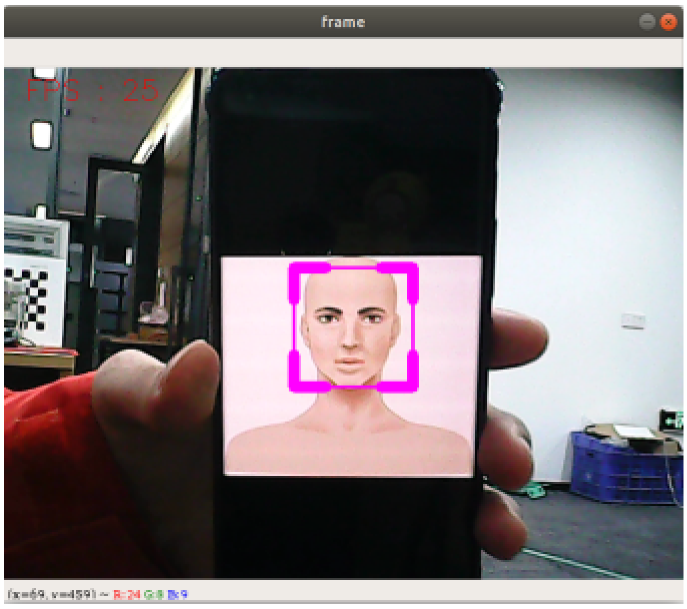
5.2.1, Start

Start the camera

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

Terminal input,

```
ros2 run yahboomcar_mediapipe 05_FaceEyeDetection
```



5.2.2, Source code

Source code location: ~/ascam_ros2_ws/src/yahboomcar_mediapipe/yahboomcar_mediapipe/05_FaceEyeDetection.py

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

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

print("import done")

class FaceEyeDetection(Node):
    def __init__(self, name):
        super().__init__(name)
        self.bridge = CvBridge()
        self.eyeDetect =
cv.CascadeClassifier("/home/yahboom/ascam_ros2_ws/src/yahboomcar_mediapipe/yahbo
omcar_mediapipe/file/haarcascade_eye.xml")
        self.faceDetect =
cv.CascadeClassifier("/home/yahboom/ascam_ros2_ws/src/yahboomcar_mediapipe/yahbo
omcar_mediapipe/file/haarcascade_frontalface_default.xml")
        self.pub_rgb = self.create_publisher(Image, "/FaceEyeDetection/image",
500)
        self.pTime = 0
        self.content_index = 0
        self.content = ["face", "eye", "face_eye"]
        self.subscription = self.create_subscription(
            Image,
            '/ascamera_hp60c/camera_publisher/rgb0/image',
            self.image_callback,
            10  # QoS profile
        )
        self.subscription  # prevent unused variable warning

    def image_callback(self, msg):
        # Convert ROS 2 image messages to OpenCV format
        frame = self.bridge.imgmsg_to_cv2(msg, desired_encoding='bgr8')

        # Select processing method based on current content index
        if self.content[self.content_index] == "face":
            frame = self.face(frame)
        elif self.content[self.content_index] == "eye":
            frame = self.eye(frame)
        else:
            frame = self.eye(self.face(frame))


        cTime = time.time()
        if self.pTime != 0:
            fps = 1 / (cTime - self.pTime)
            text = "FPS : " + str(int(fps))
            cv.putText(frame, text, (20, 30), cv.FONT_HERSHEY_SIMPLEX, 0.9, (0,
0, 255), 1)
        self.pTime = cTime
        cv.imshow('frame', frame)

        #Detecting key events
        key = cv.waitKey(1) & 0xFF
        if key == ord('f') or key == ord('F'):
            self.content_index += 1
            if self.content_index >= len(self.content):
                self.content_index = 0
        elif key == ord('q') or key == ord('Q'):
            self.get_logger().info('Q pressed, exiting...')
            self.cancel()
            rclpy.shutdown()


    # Publish the processed image
        self.pub_img(frame)

    def face(self, frame):
        gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
        faces = self.faceDetect.detectMultiScale(gray, 1.3)
        for face in faces:
            frame = self.faceDraw(frame, face)
        return frame

    def eye(self, frame):
        gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
        eyes = self.eyeDetect.detectMultiScale(gray, 1.3)
        for eye in eyes:
            cv.circle(frame, (int(eye[0] + eye[2] / 2), int(eye[1] + eye[3] /
2)), int(eye[3] / 2), (0, 0, 255), 2)
        return frame

    def faceDraw(self, frame, bbox, l=30, t=10):
        x, y, w, h = bbox
        x1, y1 = x + w, y + h
        cv.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 255), 2)
       # Draw the boundary line
        cv.line(frame, (x, y), (x + l, y), (255, 0, 255), t)
        cv.line(frame, (x, y), (x, y + l), (255, 0, 255), t)
        cv.line(frame, (x1, y), (x1 - l, y), (255, 0, 255), t)
        cv.line(frame, (x1, y), (x1, y + l), (255, 0, 255), t)
        cv.line(frame, (x, y1), (x + l, y1), (255, 0, 255), t)
        cv.line(frame, (x, y1), (x, y1 - l), (255, 0, 255), t)
        cv.line(frame, (x1, y1), (x1 - l, y1), (255, 0, 255), t)
        cv.line(frame, (x1, y1), (x1, y1 - l), (255, 0, 255), t)
        return frame

    def pub_img(self, frame):
        self.pub_rgb.publish(self.bridge.cv2_to_imgmsg(frame, "bgr8"))

    def cancel(self):
        self.get_logger().info('Publisher canceled')
        face_eye_detection.destroy_node()
        rclpy.shutdown()
        cv.destroyAllWindows()

def main(args=None):
    rclpy.init(args=args)
    face_eye_detection = FaceEyeDetection('face_eye_detection')

    try:
        rclpy.spin(face_eye_detection)
    except KeyboardInterrupt:
        pass
    finally:
        face_eye_detection.destroy_node()
        rclpy.shutdown()
        cv.destroyAllWindows()

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