5.Face recognition

Note: There are related running codes on Raspberry Pi and Jetson nano, but due to differences in motherboard performance, the running may not be as smooth.The supporting virtual machine also has the operating environment and programs installed. If the experience on the motherboard is not good, you can remove the camera, plug it into the virtual machine, and connect the camera device to the virtual machine to run the corresponding program on the virtual machine.

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 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.

Features of MediaPipe:

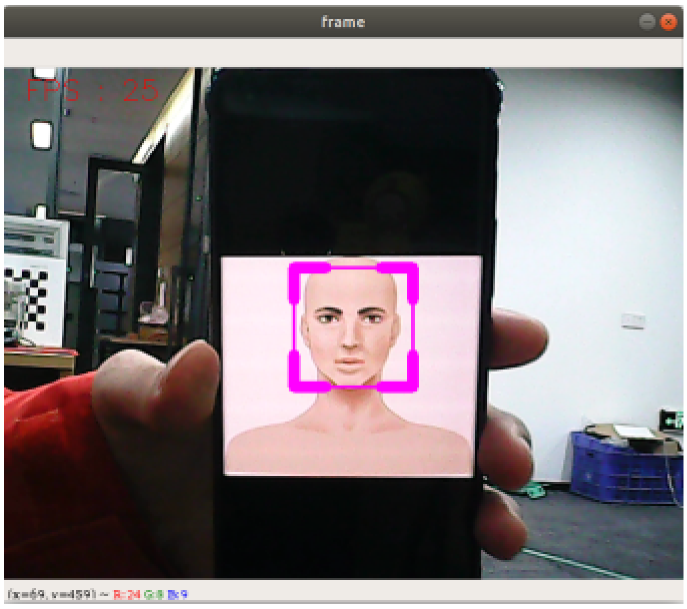
- End-to-end acceleration: Built-in fast ML inference and processing accelerates even on commodity hardware.
- Build once, deploy anywhere: Unified solution for Android, iOS, desktop/cloud, web and IoT.
- Ready-to-use solutions: cutting-edge ML solutions that showcase the full capabilities of the framework.
- Free and open source: frameworks and solutions under Apache2.0, fully extensible and customizable.

5.2. Face recognition

5.2.1.Startup

Terminal input,

```
roscore
rosrun dofbot_mediapipe 05_FaceEyeDetection.py
```



5.2.2.Source code

Source code location: /home/jetson/dofbot_ws/src/dofbot_mediapipe/scripts/05_FaceEyeDetection.py

```
#!/usr/bin/env python3
# encoding: utf-8
import time
import rospy
import rospkg
import cv2 as cv
from cv_bridge import CvBridge
from sensor_msgs.msg import CompressedImage, Image


class FaceEyeDetection:
    def __init__(self):
        self.bridge = CvBridge()

        rospy.init_node("FaceEyeDetection", anonymous=False)
        self.eyeDetect = cv.CascadeClassifier(
            rospkg.RosPack().get_path("dofbot_mediapipe") +
"/scripts/file/haarcascade_eye.xml")
        self.faceDetect = cv.CascadeClassifier(
            rospkg.RosPack().get_path("dofbot_mediapipe") +
"/scripts/file/haarcascade_frontalface_default.xml")
        self.pub_rgb = rospy.Publisher("/FaceEyeDetection/image", Image,
queue_size=1)
        rospy.on_shutdown(self.cancel)
    def cancel(self):
        self.pub_rgb.unregister()

    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)
        # Top left x,y
        cv.line(frame, (x, y), (x + l, y), (255, 0, 255), t)
        cv.line(frame, (x, y), (x, y + l), (255, 0, 255), t)
        # Top right x1,y
        cv.line(frame, (x1, y), (x1 - l, y), (255, 0, 255), t)
        cv.line(frame, (x1, y), (x1, y + l), (255, 0, 255), t)
        # Bottom left x1,y1
        cv.line(frame, (x, y1), (x + l, y1), (255, 0, 255), t)
        cv.line(frame, (x, y1), (x, y1 - l), (255, 0, 255), t)
        # Bottom right x1,y1
        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"))


if __name__ == '__main__':
    capture = cv.VideoCapture(0)
    capture.set(6, cv.VideoWriter_fourcc(*'XVID'))
    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, content_index = 0, 0, 0
    dat_file = "./file/shape_predictor_68_face_landmarks.dat"
    face_eye_detection = FaceEyeDetection()
    content = ["face", "eye", "face_eye"]
    while capture.isOpened():
        ret, frame = capture.read()
        # frame = cv.flip(frame, 1)
        action = cv.waitKey(1) & 0xFF
        if action == ord("f") or action == ord("F"):
            content_index += 1
            if content_index >= len(content): content_index = 0
        if content[content_index] == "face": frame =
face_eye_detection.face(frame)
        elif content[content_index] == "eye": frame =
face_eye_detection.eye(frame)
        else: frame = face_eye_detection.eye(face_eye_detection.face(frame))
        if action == ord('q') or action == 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)
        cv.imshow('frame', frame)
        face_eye_detection.pub_img(frame)
    capture.release()
    cv.destroyAllWindows()
```