7. Face Detection

7.1. 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 supports 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: 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.

7.2, Face Detection

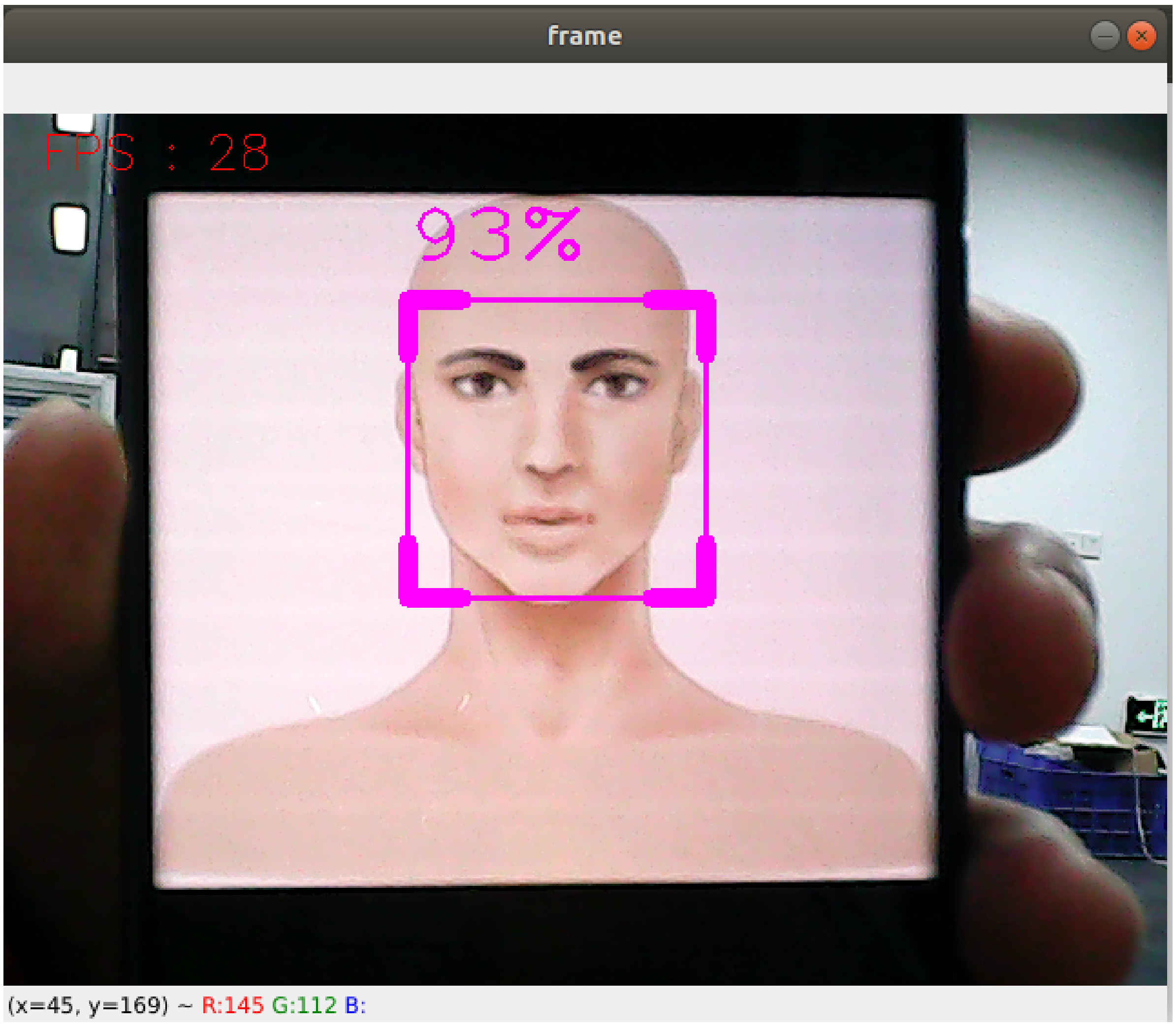
7.2.1, Start

Start the camera

```
roslaunch ascamera hp60c.launch
```

Terminal input,

```
roslaunch yahboomcar_mediapipe 07_FaceDetection.launch
```



7.2.2, Source code

Source code location: /home/yahboom/ascam_ws/src/yahboomcar_mediapipe/scripts/07_FaceDetection.py

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

class FaceDetector:
    def __init__(self, minDetectionCon=0.5):
        self.mpFaceDetection = mp.solutions.face_detection
        self.mpDraw = mp.solutions.drawing_utils
        self.facedetection =
self.mpFaceDetection.FaceDetection(min_detection_confidence=minDetectionCon)

    def findFaces(self, frame):
        img_RGB = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
        self.results = self.facedetection.process(img_RGB)
        bboxs = []
        if self.results.detections:
            for id, detection in enumerate(self.results.detections):
                bboxC = detection.location_data.relative_bounding_box
                ih, iw, ic = frame.shape
                bbox = int(bboxC.xmin * iw), int(bboxC.ymin * ih), \
                       int(bboxC.width * iw), int(bboxC.height * ih)
                bboxs.append([id, bbox, detection.score])
                frame = self.fancyDraw(frame, bbox)
                cv.putText(frame, f'{int(detection.score[0] * 100)}%',
                           (bbox[0], bbox[1] - 20), cv.FONT_HERSHEY_PLAIN,
                           3, (255, 0, 255), 2)
        return frame, bboxs

    def fancyDraw(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

class FaceDetectionNode:
    def __init__(self):
        self.bridge = CvBridge()
        rospy.init_node("FaceDetection", anonymous=False)
        self.face_detector = FaceDetector(0.75)
        self.pub_rgb = rospy.Publisher("/FaceDetection/image", Image,
queue_size=1)
        self.sub_image = rospy.Subscriber('/ascamera_hp60c/rgb0/image', Image,
self.image_callback, queue_size=1)
        self.pTime = 0

    def image_callback(self, msg):
        try:
            # Convert the ROS Image message to an OpenCV image
            frame = self.bridge.imgmsg_to_cv2(msg, desired_encoding="bgr8")

            # Process the image
            frame, _ = self.face_detector.findFaces(frame)

            # Calculate and display FPS
            cTime = time.time()
            fps = 1 / (cTime - self.pTime)
            self.pTime = cTime
            text = f"FPS: {int(fps)}"
            cv.putText(frame, text, (20, 30), cv.FONT_HERSHEY_SIMPLEX, 0.9, (0,
0, 255), 1)
```

```
            # Display the image
            cv.imshow('frame', frame)

            # Publish the processed image
            self.pub_rgb.publish(self.bridge.cv2_to_imgmsg(frame, "bgr8"))

            # Check for key press to quit
            if cv.waitKey(1) & 0xFF == ord('q'):
                rospy.signal_shutdown("User requested shutdown")
        except Exception as e:
            rospy.logerr(f"Could not process image: {e}")


if __name__ == '__main__':
    node = FaceDetectionNode()
    rospy.spin()
    cv.destroyAllWindows()
```