Face Detection

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 Jetson nano), 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:

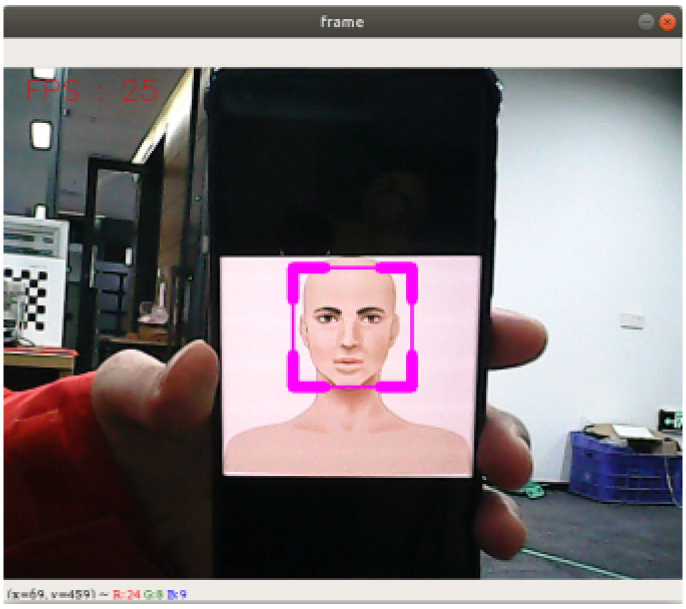
- End-to-end acceleration: built-in fast ML inference and processing can be accelerated even on commodity hardware.
- Build once, deploy anywhere: unified solution for Android, iOS, desktop/cloud, web and IoT.
- Ready-to-use solution: cutting-edge ML solution that demonstrates the full capabilities of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

2. Face detection

2.1. Start

- Enter the following command to start the program

```
ros2 run dofbot_pro_mediapipe 05_FaceDetection
```



2.2. Source code

Source code location:
~/dofbot_pro_ws/src/dofbot_pro_mediapipe/dofbot_pro_mediapipe/05_FaceDetection.py

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

import mediapipe as mp
import cv2 as cv
import time
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import Image
from cv_bridge import CvBridge

class FaceDetector(Node):
    def __init__(self, minDetectionCon=0.5):
        super().__init__('face_detector')
        self.publisher_ = self.create_publisher(Image, 'face_detected', 10)
        self.timer = self.create_timer(0.1, self.timer_callback)
        self.bridge = CvBridge()
        self.mpFaceDetection = mp.solutions.face_detection
        self.mpDraw = mp.solutions.drawing_utils
        self.facedetection = 
self.mpFaceDetection.FaceDetection(min_detection_confidence=minDetectionCon)
        self.capture = cv.VideoCapture(0, cv.CAP_V4L2)
        self.capture.set(6, cv.VideoWriter.fourcc('M', 'J', 'P', 'G'))
        self.capture.set(cv.CAP_PROP_FRAME_WIDTH, 640)
        self.capture.set(cv.CAP_PROP_FRAME_HEIGHT, 480)
        self.pTime = 0

    def timer_callback(self):
        ret, frame = self.capture.read()
        if not ret:
            self.get_logger().error('Failed to capture frame')
            return
        frame, _ = self.findFaces(frame)

        # Calculate FPS
        cTime = time.time()
        fps = 1 / (cTime - self.pTime)
        self.pTime = cTime

        # Display FPS on frame
        cv.putText(frame, f'FPS: {int(fps)}', (20, 50), cv.FONT_HERSHEY_SIMPLEX, 
1, (0, 255, 0), 2)

        msg = self.bridge.cv2_to_imgmsg(frame, "bgr8")
        self.publisher_.publish(msg)

        # Show the frame with face detection
        cv.imshow('Face Detection', frame)
        if cv.waitKey(1) & 0xFF == ord('q'):
            self.capture.release()
```

```python
            cv.destroyAllWindows()
            rclpy.shutdown()

    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

def main(args=None):
    rclpy.init(args=args)
    face_detector = FaceDetector()
    rclpy.spin(face_detector)
    face_detector.destroy_node()
    rclpy.shutdown()

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