Face Effects

Orin board users can directly open the terminal and input the tutorial commands to run directly. Jetson-Nano board users need to enter the docker container first, then input the tutorial commands in the docker to start the program.

1. Introduction

MediaPipe is a data stream processing machine learning application development framework developed and open-sourced by Google. It is a graph-based data processing pipeline used to build applications that use various 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 (Jetson nano, 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:

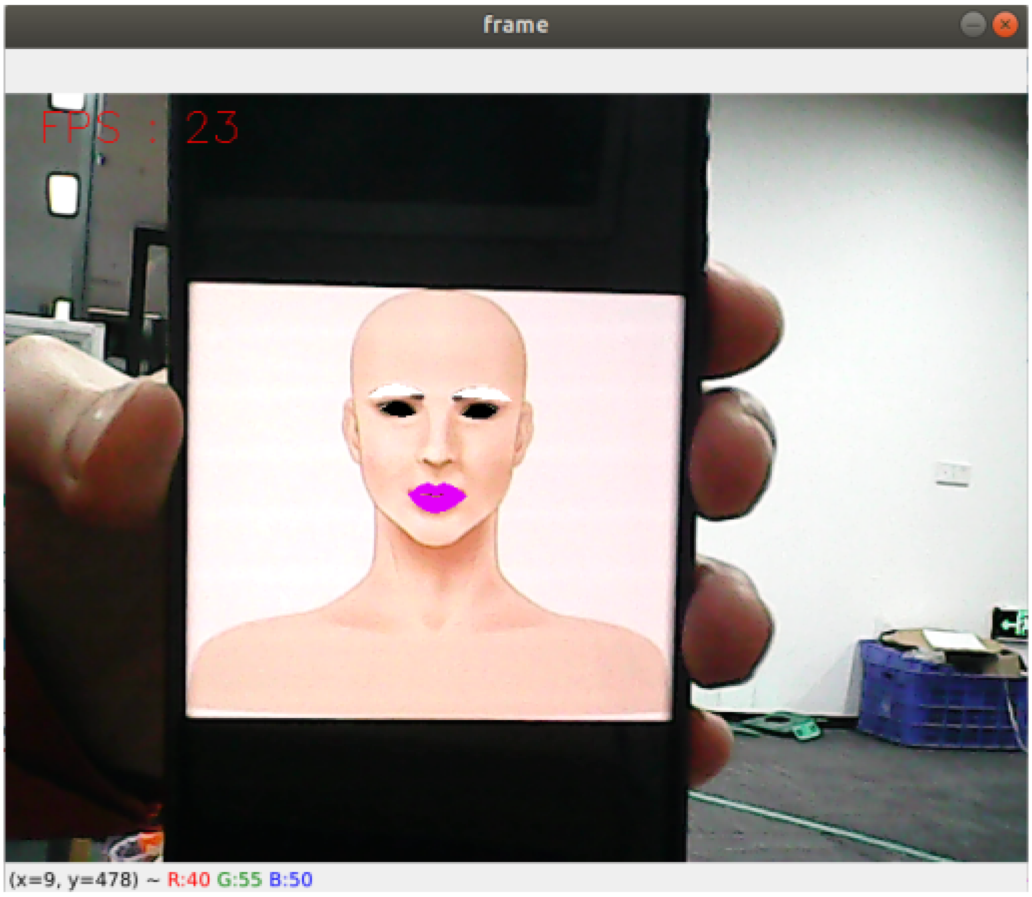
- End-to-end acceleration: Built-in fast ML inference and processing accelerates even on ordinary hardware.
- Build once, deploy anywhere: Unified solution for Android, iOS, desktop/cloud, web and IoT.
- Ready-to-use solutions: Cutting-edge ML solutions that showcase the full capabilities of the framework.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

2. Face Effects

2.1. Launch

- Enter the following command to start the program

```
ros2 run dofbot_pro_mediapipe 06_FaceLandmarks
```



2.2. Source Code

Source code location:

```
# Jetson-Nano users need to enter the docker container to view
~/dofbot_pro_ws/src/dofbot_pro_mediapipe/dofbot_pro_mediapipe/06_FaceLandmarks.p
y
```

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

import time
import dlib
import os
import cv2 as cv
import numpy as np
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import Image
from cv_bridge import CvBridge

class FaceLandmarks(Node):
    def __init__(self, dat_file):
        super().__init__('face_landmarks')
        self.publisher_ = self.create_publisher(Image,
'face_landmarks_detected', 10)
        self.timer = self.create_timer(0.1, self.timer_callback)
        self.bridge = CvBridge()
        self.hog_face_detector = dlib.get_frontal_face_detector()
```

```python
        self.dlib_facelandmark = dlib.shape_predictor(dat_file)
        self.capture = cv.VideoCapture(0, cv.CAP_V4L2)
        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.get_face(frame, draw=False)
        frame = self.prettify_face(frame, eye=True, lips=True, eyebrow=True,
draw=True)

        # 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, 30), cv.FONT_HERSHEY_SIMPLEX,
0.9, (0, 0, 255), 1)

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

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

    def get_face(self, frame, draw=True):
        gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
        self.faces = self.hog_face_detector(gray)
        for face in self.faces:
            self.face_landmarks = self.dlib_facelandmark(gray, face)
            if draw:
                for n in range(68):
                    x = self.face_landmarks.part(n).x
                    y = self.face_landmarks.part(n).y
                    cv.circle(frame, (x, y), 2, (0, 255, 255), 2)
                    cv.putText(frame, str(n), (x, y), cv.FONT_HERSHEY_SIMPLEX,
0.6, (0, 255, 255), 2)
        return frame

    def get_lmList(self, frame, p1, p2, draw=True):
        lmList = []
        if len(self.faces) != 0:
            for n in range(p1, p2):
                x = self.face_landmarks.part(n).x
                y = self.face_landmarks.part(n).y
                lmList.append([x, y])
                if draw:
                    next_point = n + 1
                    if n == p2 - 1: next_point = p1
                    x2 = self.face_landmarks.part(next_point).x
                    y2 = self.face_landmarks.part(next_point).y
                    cv.line(frame, (x, y), (x2, y2), (0, 255, 0), 1)
        return lmList

    def get_lipList(self, frame, lipIndexlist, draw=True):
        lmList = []
        if len(self.faces) != 0:
            for n in range(len(lipIndexlist)):
                x = self.face_landmarks.part(lipIndexlist[n]).x
                y = self.face_landmarks.part(lipIndexlist[n]).y
                lmList.append([x, y])
                if draw:
                    next_point = n + 1
                    if n == len(lipIndexlist) - 1: next_point = 0
                    x2 = self.face_landmarks.part(lipIndexlist[next_point]).x
                    y2 = self.face_landmarks.part(lipIndexlist[next_point]).y
                    cv.line(frame, (x, y), (x2, y2), (0, 255, 0), 1)
        return lmList

    def prettify_face(self, frame, eye=True, lips=True, eyebrow=True,
draw=True):
        if eye:
            leftEye = self.get_lmList(frame, 36, 42)
            rightEye = self.get_lmList(frame, 42, 48)
            if draw:
                if len(leftEye) != 0: frame = cv.fillConvexPoly(frame,
np.array(leftEye, dtype=np.int32), (0, 0, 0))
                if len(rightEye) != 0: frame = cv.fillConvexPoly(frame,
np.array(rightEye, dtype=np.int32), (0, 0, 0))
        if lips:
            lipIndexlistA = [51, 52, 53, 54, 64, 63, 62]
            lipIndexlistB = [48, 49, 50, 51, 62, 61, 60]
            lipsUpA = self.get_lipList(frame, lipIndexlistA, draw=True)
            lipsUpB = self.get_lipList(frame, lipIndexlistB, draw=True)
            lipIndexlistA = [57, 58, 59, 48, 67, 66]
            lipIndexlistB = [54, 55, 56, 57, 66, 65, 64]
            lipsDownA = self.get_lipList(frame, lipIndexlistA, draw=True)
            lipsDownB = self.get_lipList(frame, lipIndexlistB, draw=True)
            if draw:
                if len(lipsUpA) != 0: frame = cv.fillConvexPoly(frame,
np.array(lipsUpA, dtype=np.int32), (249, 0, 226))
                if len(lipsUpB) != 0: frame = cv.fillConvexPoly(frame,
np.array(lipsUpB, dtype=np.int32), (249, 0, 226))
                if len(lipsDownA) != 0: frame = cv.fillConvexPoly(frame,
np.array(lipsDownA, dtype=np.int32), (249, 0, 226))
                if len(lipsDownB) != 0: frame = cv.fillConvexPoly(frame,
np.array(lipsDownB, dtype=np.int32), (249, 0, 226))
        if eyebrow:
            lefteyebrow = self.get_lmList(frame, 17, 22)
            righteyebrow = self.get_lmList(frame, 22, 27)
            if draw:
                if len(lefteyebrow) != 0: frame = cv.fillConvexPoly(frame,
np.array(lefteyebrow, dtype=np.int32), (255, 255, 255))
                if len(righteyebrow) != 0: frame = cv.fillConvexPoly(frame,
np.array(righteyebrow, dtype=np.int32), (255, 255, 255))
        return frame
```

```
def main(args=None):
    rclpy.init(args=args)
    current_dir_path = os.path.dirname(__file__)
    dat_file = os.path.join(current_dir_path,
"/home/jetson/dofbot_pro_ws/src/dofbot_pro_mediapipe/dofbot_pro_mediapipe/file/s
hape_predictor_68_face_landmarks.dat")
    face_landmarks = FaceLandmarks(dat_file)
    rclpy.spin(face_landmarks)
    face_landmarks.destroy_node()
    rclpy.shutdown()

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