3D Object Recognition

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.

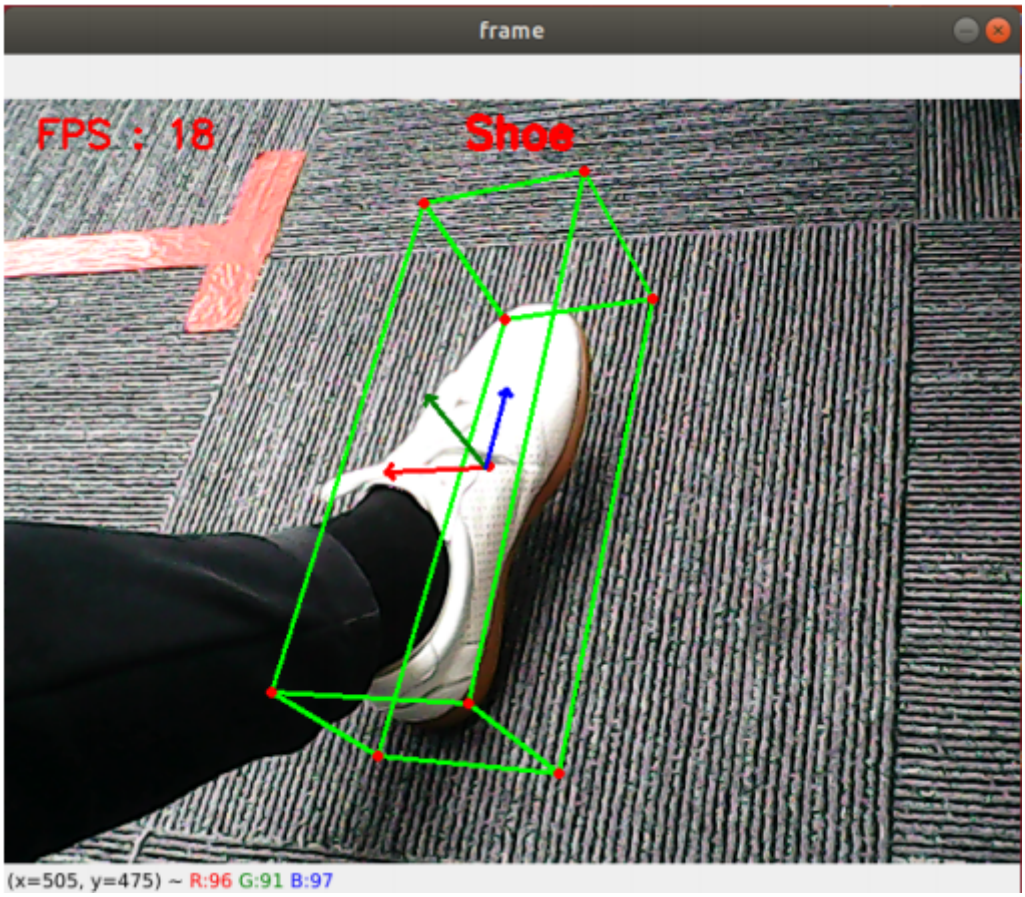
2. 3D Object Recognition

3D object recognition: Shoes.

2.1. Launch

- Enter the following command to start the program

```
ros2 run dofbot_pro_mediapipe 07_Objectron
```



After starting the program, place shoes in the camera view, and you can see the image recognize the shoes and frame them.

Press q to exit the program.

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/07_Objectron.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 Objectron(Node):
    def __init__(self, staticMode=False, maxObjects=5, minDetectionCon=0.5,
minTrackingCon=0.99):
        super().__init__('objectron')
        self.publisher_ = self.create_publisher(Image, 'objectron_detected', 10)
        self.timer = self.create_timer(0.1, self.timer_callback)
        self.bridge = CvBridge()
```

```
        self.staticMode = staticMode
        self.maxObjects = maxObjects
        self.minDetectionCon = minDetectionCon
        self.minTrackingCon = minTrackingCon
        self.index = 0
        self.modelNames = ['Shoe', 'Chair', 'Cup', 'Camera']
        self.mpObjectron = mp.solutions.objectron
        self.mpDraw = mp.solutions.drawing_utils
        self.mpobjectron = self.mpObjectron.Objectron(
            self.staticMode, self.maxObjects, self.minDetectionCon,
self.minTrackingCon, self.modelNames[self.index])
        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

        # Check for key press to switch model or quit
        action = cv.waitKey(1) & 0xFF
        if action == ord('q'):
            self.capture.release()
            cv.destroyAllWindows()
            rclpy.shutdown()
            return
        if action == ord('f') or action == ord('F'):
            self.configUP()

        frame = self.findObjectron(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, 30), cv.FONT_HERSHEY_SIMPLEX,
0.8, (0, 0, 255), 2)

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

        # Show the frame with object detection
        cv.imshow('Objectron', frame)

    def findObjectron(self, frame):
        cv.putText(frame, self.modelNames[self.index], (int(frame.shape[1] / 2)
- 30, 30),
                   cv.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 3)
        img_RGB = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
        results = self.mpobjectron.process(img_RGB)
        if results.detected_objects:
            for id, detection in enumerate(results.detected_objects):
```

```
                self.mpDraw.draw_landmarks(frame, detection.landmarks_2d,
self.mpObjectron.BOX_CONNECTIONS)
                self.mpDraw.draw_axis(frame, detection.rotation,
detection.translation)
        return frame

    def configUP(self):
        self.index += 1
        if self.index >= len(self.modelNames):
            self.index = 0
        self.mpobjectron = self.mpObjectron.Objectron(
            self.staticMode, self.maxObjects, self.minDetectionCon,
self.minTrackingCon, self.modelNames[self.index])

def main(args=None):
    rclpy.init(args=args)
    objectron = Objectron()
    rclpy.spin(objectron)
    objectron.destroy_node()
    rclpy.shutdown()

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