8. 3D Object Recognition

8.1. 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 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 ordinary 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.

8.2, 3D object recognition

3D object recognition: Recognizable objects are: ['Shoe', 'Chair', 'Cup', 'Camera'], a total of 4 categories; click [f key] to switch the recognized object.

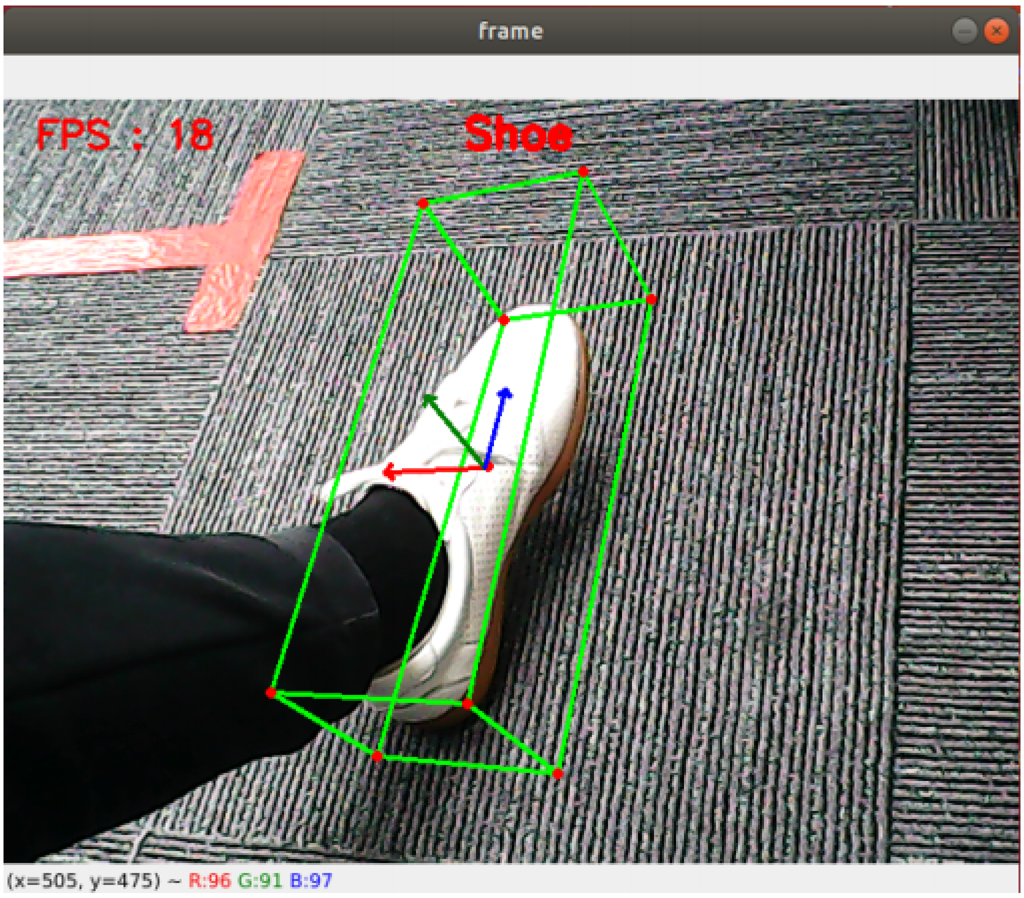
8.2.1, Start

Start the camera

```
roslaunch ascamera hp60c.launch
```

Terminal input,

```
roslaunch yahboomcar_mediapipe 08_Objectron.launch
```



8.2.2, Source code

Source code location:
/home/yahboom/ascam_ws/src/yahboomcar_mediapipe/scripts/08_Objectron.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 Objectron:
    def __init__(self, staticMode=False, maxObjects=5, minDetectionCon=0.5,
minTrackingCon=0.99):
        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])

    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>=4:self.index=0
        self.mpobjectron = self.mpObjectron.Objectron(
            self.staticMode, self.maxObjects, self.minDetectionCon,
self.minTrackingCon, self.modelNames[self.index])

class ObjectronNode:
    def __init__(self):
        #Initialize ROS node
        rospy.init_node('objectron_node', anonymous=True)

        # Create a CvBridge instance
        self.bridge = CvBridge()

        # Initialize the Objectron class
        self.objectron = Objectron()

        # Initialize FPS counter
        self.pTime = 0

        # Subscribe to the topic /ascamera_hp60c/rgb0/image
        self.image_sub = rospy.Subscriber('/ascamera_hp60c/rgb0/image', Image,
self.image_callback)

        # Posting processed images
        self.pub_image = rospy.Publisher('/objectron_processed_image', Image,
queue_size=1)

    def image_callback(self, data):
        try:
            # Convert ROS image messages to OpenCV format
            frame = self.bridge.imgmsg_to_cv2(data, "bgr8")
        except Exception as e:
            rospy.logerr(f"Error converting image: {e}")
            return

        # Checking for key events
        action = cv.waitKey(1) & 0xFF
        if action == ord('q'):
            rospy.signal_shutdown("User requested shutdown")
        elif action == ord('f') or action == ord('F'):
            self.objectron.configUP()

        # Processing images
        frame = self.objectron.findObjectron(frame)

        # Calculating FPS
        cTime = time.time()
        fps = 1 / (cTime - self.pTime)
        self.pTime = cTime
        text = "FPS : " + str(int(fps))
        cv.putText(frame, text, (20, 30), cv.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0,
255), 2)

        # Display results
        cv.imshow('frame', frame)

        # Posting processed images
        self.pub_image.publish(self.bridge.cv2_to_imgmsg(frame, "bgr8"))

    def shutdown_hook(self):
        rospy.loginfo("Shutting down objectron node.")
        cv.destroyAllWindows()


if __name__ == '__main__':
    try:
        node = ObjectronNode()
        rospy.on_shutdown(node.shutdown_hook)
        rospy.spin()
    except rospy.ROSInterruptException:
        pass
```