第5章 环境跟踪

5.1 简　　介

增强现实需要对周围环境进行准确理解，其中依靠的最重要的传感器就是摄像机。基于摄像机采集的图像进行分析和理解是计算机视觉的主要任务。计算机视觉的常见任务包括图像分类、目标定位、目标检测、图像分割等。

物体识别包括两个问题，一是图像分类，二是目标定位。图像分类任务是指从图像中提取特征并依据特征来为图像中的目标进行分类。图像分类的结果就是把图像中的某个区域划分为某一个类别的事物。目标定位任务是在识别出图像中的对象类别后进一步确定该对象在图像中的位置，位置会被一个矩形的包围盒框选出来。因此目标定位的结果不仅包括对象的类别信息，还有位置信息。目标检测相对目标定位而言，更适用于多目标的场景，其检测结果为场景内多个目标的类别和位置信息。

图像分割是将图像细分为多个具有相似性质且不相交的区域，是对图像中的每一个像素加标签的过程，即像素级的分割。图像分割任务主要有语义分割和实例分割两种。语义分割是为图像中的每个像素都赋予一个统一的类别标签，比如在一张有多辆汽车的图像上，语义分割的结果可以为图像中所有属于汽车的像素点标识同一色彩，但汽车个体之间是无差别的，也就是说，语义分割只识别类别而不判别个体，而实例分割可以实现对同一类别不同个体间的判别。

5.1.1 物体识别

物体识别是计算机视觉中的一项基础任务。在物体识别中，既需要考虑分类问题，也需要解决定位问题，目标是实现对图像中可变数量的对象的分类和定位。这里

的“可变”指不同的图像中可识别的对象数量可能不同。定位的结果是目标对象的边界框。物体识别的方法可以分为两大类：一类是基于模型的方法，另一类是基于上下文识别的方法。

物体识别一般要经过以下几个步骤。

1. 图像预处理

图像预处理是对图像数据进行简化的过程，在这个过程中会消除无关信息，以便于对有效信息进行提取。良好的预处理也有助于特征抽取、图像识别、定位、分割等任务的效果提升。常规操作一般有数字化、几何变换、归一化等。受采集图像的设备和应用场景的影响，需要采取不同的预处理运算来处理图像，这几乎是所有计算机视觉算法都需要的。预处理通常包括五种运算：编码、阈值或滤波运算、模式改善、正规化以及离散模式运算。

2. 特征提取

特征提取是指通过计算机提取图像信息，并确定每个图像的点所属的图像特征。常用的图像特征包括颜色、纹理、形状以及空间关系等。特征提取的结果是把图像上的点分为不同的集合，集合在图像上可以表现为孤立的点、连续的曲线或区域。不同形式的特征的计算复杂性和可重复性也大不相同。特征的好坏很大程度上影响着泛化性能。

3. 特征选择

进行特征提取后，可能得到许多特征，这时候需要从原始的众多特征中选取出最有效的特征组合以降低数据集维度，从而达到提高学习算法性能的目的。任何能够在选出来的部分特征上正常工作的模型在原特征上也都是可以正常工作的。反过来，特征选择是有可能导致一些有用的特征丢失的，但相比于节省下的巨大计算开销，有时候这样的特征丢失是值得的。

4. 建模

在物体识别中，每一类物体都是有相同点的，在给定特征集合后，从中提取相同点、分辨不同点是模型要解决的关键问题。因此可以说模型是整个识别系统的成败所在。模型主要建模的对象是特征与特征之间的空间结构关系；模型的选择主要有两个标准，一是模型的假设是否适用于我们的问题，二是根据模型所需要的计算复杂度，选择可以承受的方案。

5. 匹配

在得到模型之后，就可以用该模型对新的图像进行识别了，在识别出图像中对象的类别的同时，尽可能给出边界。

6. 定位

在识别出对象类别之后，需要对目标进行定位。部分模型本身就具有定位的能力（如描述生成模型、基于部分的模型），因为特征的空间分布就是模型处理的对象。

目标检测的常用框架有两种：第一种是 two-stage 方法，它将兴趣区域检测和分类分开进行，代表工作有 Fast RCNN、Faster RCNN；另一种是 one-stage 方法，它只用一个网络同时进行兴趣区域检测和分类，代表工作有 YOLO、SSD。

1) Faster RCNN

Ross B. Girshick 在 2016 年提出了新的 Faster RCNN（如图 5.1 所示），该算法在 ILSVRV 和 COCO 竞赛中获得多项第一。Faster RCNN 的卓越成果可以说是逐渐积累的结果。从 R-CNN 到 Fast-RCNN，再到 Faster-RCNN，乃至现在的 Mask-RCNN，Mask-RCNN 在实例分割领域已经取得了卓越的成果。

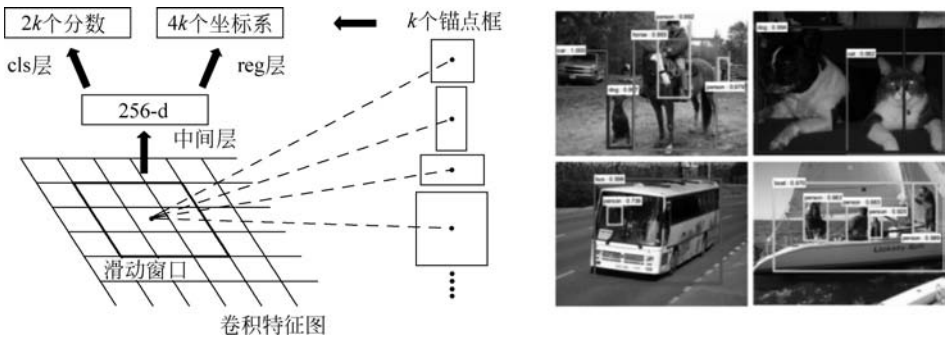

图 5.1　Faster RCNN 结构图

如图 5.1 所示，可以将 Faster RCNN 分为四个部分。首先是卷积层，卷积层包含 conv、pooling 以及 relu 三种层，负责提取特征图，特征图被共享用于后续 RPN 层和全连接层；然后是区域候选层（Region Proposal Networks，RPN），RPN 网络用于生成区域候选框，相对于传统的生成候选框的方式耗时很少，并且可以轻易地结合到 Fast RCNN 中，可以说，Faster RCNN 就是 RPN 与 Fast RCNN 结合的产物；第三部分是 RoI Pooling，该层综合之前的特征图和候选框信息，提取出候选框特征图，用于后续的全连接层进行类别的判定；最后进行分类，根据候选框特征图判定类别，并进行边

界框回归以获取最终的准确边界框。

2）SSD

这是一种使用单个深度神经网络进行目标检测的方法。SSD（如图 5.2 所示）将边界框的输出空间离散化为一组默认框，并且采用了不同尺寸和比例的默认框。在测试的时候，网络对默认框中的对象判定类别并打分，同时对框进行调整以更好地适配对象形状。此外，该网络结合了来自不同分辨率的多个特征图的预测以自然地处理各种尺寸的对象。

相对于其他需要候选框的方法而言，SSD 更简单，因为 SSD 中完全消除了候选框生成和后续像素与特征的重采样阶段，并将所有计算封装在单个网络之中。所以 SSD 更容易训练，也更容易被集成到系统中。在 PASCAL VOC、MS COCO 以及 ILSVRC 数据集上的实验结果证明，SSD 具有与使用候选框的方法可比的准确性，但是速度更快，而且 SSD 为训练和推理提供了一个统一的框架。与其他 one-stage 方法相比，SSD 有着更高的精度。

5.1.2 图像分割

图像分割是把图像分成若干个特定的、具有独特性质的区域并提取出感兴趣目标的技术和过程。物体分割实现的是像素级的分割，结果会为每一个像素赋予分类标签。物体分割可以分为语义分割和实例分割。其中，语义分割是为图像中的每个像素赋予一个统一的类别标签，比如在一张有多辆汽车的图像中，语义分割的结果可以为图像中所有属于汽车的像素点标识同一色彩，但汽车个体之间是无差别的，也就是说，语义分割只识别类别而不判别个体。而实例分割实现的是对同一类别不同个体间的判别。

典型的图像分割技术大概可划分为基于图论的方法、基于像素聚类的方法以及基于语义的方法。近年来，基于深度学习的方法取得了较好的效果，典型代表为 MASK RCNN。

1. 基于图论的分割方法

基于图论的分割方法充分利用图论的理论和方法，它将图像映射为带权无向图，把像素视作节点，这样一来，图像分割就被转换成了图的顶点划分问题。图像映射成带权无向图之后，每个像素点就是图的节点，相邻像素之间存在着边。每条边有自己

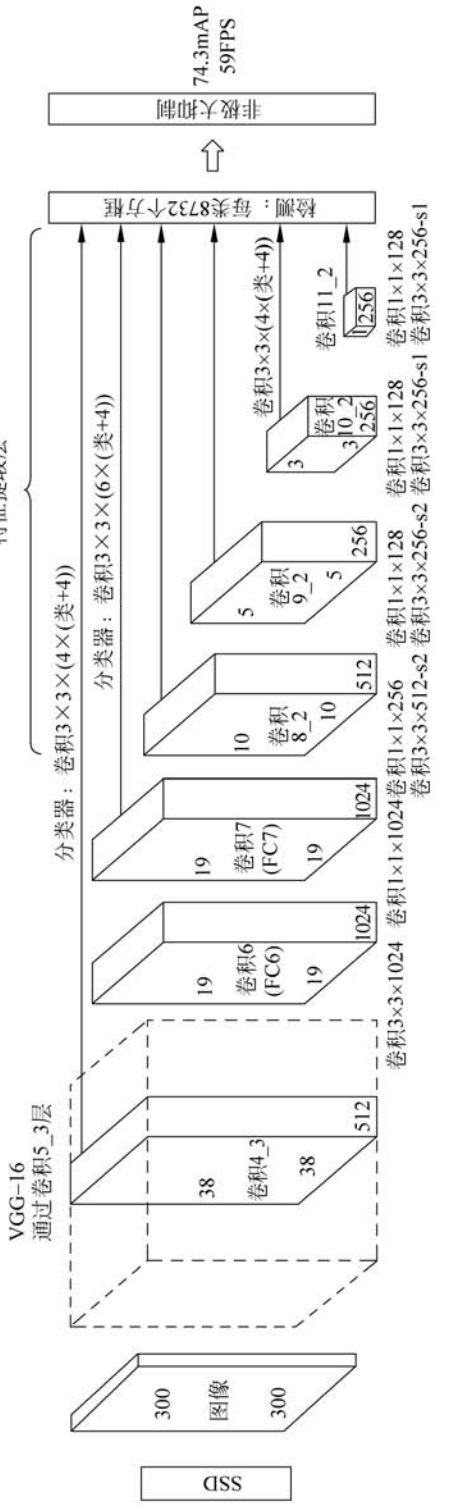
SSD
300
图像
300
VGG-16
通过卷积5_3层
38
卷积4_3
38
512
卷积3×3×1024
19
卷积6
(FC6)
19
1024
卷积1×1×1024
19
卷积7
(FC7)
19
1024
卷积1×1×256
卷积3×3×512-s2
10
卷积
8_2
10
512
卷积1×1×128
卷积3×3×256-s2
5
卷积
9_2
5
256
卷积1×1×128
卷积3×3×256-s1
3
卷积
10_2
3
256
卷积3×3×(4×(类+4))
卷积11_2
256
卷积1×1×128
卷积3×3×256-s1
卷积1×1×128
卷积3×3×256-s1
特征提取层
分类器：卷积3×3×(4×(类+4))
分类器：卷积3×3×(6×(类+4))
检测：每类8732个方框
非极大抑制
74.3mAP
59FPS

图 5.2　SSD 结构图

的权重，该权重可以表示相邻节点在颜色、灰度或纹理等特征上的相似度。然后可以利用最小剪切准则来得到图像的最佳分割。分割后的每个区域内部依据权重的类型都具有在某一特征方面的最大相似。代表方法有 NormalizedCut，GraphCut 和 GrabCut 等。

2. 基于像素聚类的方法

通过机器学习中的聚类方法也可以进行物体分割，其步骤如下。

(1) 初始化一个较为粗糙的聚类。

(2) 将在颜色、亮度、纹理等方面具有相似特征的像素点通过迭代的方式聚类到一个超像素，直至收敛，最终得到的就是分割的结果。

基于像素聚类的代表方法有 K-Means、Meanshift、谱聚类、SLIC 等。

3. 基于语义的方法

当物体的结构较为复杂，内部差异比较大的时候，基于聚类的方法只能利用像素点的颜色、亮度、纹理等较低层次的内容信息，这无法产生好的分割效果。因此，需要结合图像高层次的内容来帮助分割，这种方式称为语义分割。它有效地解决了传统图像分割方法中语义信息缺失的问题。

Mask RCNN 可以实现高效准确的实例分割。Mask RCNN(如图 5.3 所示)以 Faster RCNN 为原型，增加了一个分支用于分割任务。它更像是 FCN 和 Faster RCNN 的集成。Faster RCNN 会提取出图像中每一个对象的边界框，而 FCN 可以生成一类对象的掩膜，当 FCN 的对象只有一个的时候，就可以生成每一个对象的独立的掩膜了。

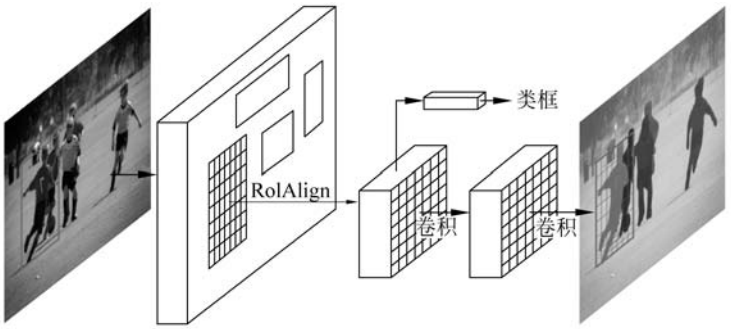

图 5.3　Mask RCNN 框架图

图 5.3 展示了 Mask RCNN 的系统框架。可以看出 Mask RCNN 的构建较为简单，只是在感兴趣区域对齐层(Region of Interest Align，ROIAlign)之后添加卷积层，进行掩模预测的任务。其中，Mask RCNN 一个非常重要的改进就是感兴趣区域对齐层。在之前 Faster RCNN 中存在的一个显著问题是：特征图与原始图像是不对准的，进而会影响检测的精度。所以 Mask R-CNN 提出了使用感兴趣区域对齐层的方法来取代感兴趣区域池化层。感兴趣区域对齐层可以保留大致的空间位置，最终使得检测精度大为提高。

5.2 HUAWEI AR Engine 中的环境跟踪

HUAWEI AR Engine 可跟踪设备周围的光照、平面、图像、物体、环境表面等环境信息，辅助应用实现虚拟物体以场景化的方式逼真地融入现实物理世界。目前，环境跟踪主要包括以下能力：光照估计、平面语义、图像跟踪、环境 Mesh。

1. 光照估计

可跟踪设备周围的光照信息，支持估计环境光的强度。HUAWEI AR Engine 可跟踪设备周围的光线信息，如平均光照强度等。光照估计能力可让虚拟物理融入真实的光照环境中，看起来更加逼真。

2. 平面语义

可检测水平和垂直平面(例如地面或墙面)。HUAWEI AR Engine 可识别到水平和垂直平面(地面或墙面)上的成群特征点，并可识别到平面的边界，应用可使用这些平面来放置需要的虚拟物体。目前可以识别墙面、地面、座位、桌子、天花板、门、窗户、床。

3. 图像跟踪

可识别和跟踪 2D 图像的位置和姿态。HUAWEI AR Engine 提供图像识别与跟踪的能力，检测场景中是否存在用户提供的图像，识别之后输出图像的位置与姿态。通过图像识别与跟踪功能，可实现基于现实世界场景中图像(海报或封面等)的增强现实。用户可提供一组参考图像，当这些图像出现在终端设备的相机视野范围内时，HUAWEI AR Engine 可实时跟踪图像，丰富场景理解及交互体验。

4. 环境 Mesh

可实时计算并输出当前画面中的环境 Mesh 数据，可用于处理虚实遮挡等应用场景。HUAWEI AR Engine 提供实时输出环境 Mesh 能力，输出内容包括终端设备在空间中的位姿，当前相机视角下的三维网格。目前拥有后置深度摄像头的机型支持此能力，且支持的扫描环境为静态场景。通过环境 Mesh 能力，可将虚拟物体放置在任意可重建的曲面上，而不再受限于水平面和垂直面；同时可利用重建的环境 Mesh 实现虚实遮挡和碰撞检测，使得虚拟角色能够准确地知道当前所在的周围三维空间情况，帮助用户实现更好的沉浸式 AR 体验。

5.3 环境跟踪关键 API

5.3.1 关键类

环境跟踪关键类见表 5.1。

表 5.1 环境跟踪关键类总览表

类/接口名称	描　　述
ARSession	用于管理 HUAWEI AR Engine 的整个运行状态，是 HUAWEI AR Engine 运行的基础
ARTrackable(接口)	被 HUAWEI AR Engine 识别的真实物体信息描述接口类
ARCamera	提供当前相机的信息，该类在引擎内部存在一个持久化对象，每次调用 ARSession.update()时，该对象的值均会被更新
ARCameraConfig	用于查询物理相机的相关配置
ARCameraIntrinsics	用于查询物理相机的离线内参(AR Engine Server 2.10 后可用)
ARConfigBase	所有 ARXXXTrackingConfig 的基类，包含各类设置项的枚举和公有设置项的接口
ARAugmentedImage	用于 2D 图像识别和跟踪时返回被跟踪的图像信息，派生自 ARTrackableBase
ARAugmentedImageDatabase	存储图片数据集，与 2D 图片识别配合使用，采集并保存到图片数据库中的图片越多，可识别的图片也越多
ARFrame	该类是 HUAWEI AR Engine 系统的快照，包含 trackable 对象、点云、深度图等信息，仅由 ARSession.update()创建
ARImage	实现 Android 里面的 Image 类，用于描述 ARFrame 获取到的预览图等 Image 信息，通过此类可以获取图片格式、宽高等信息

续表

类/接口名称	描　　述
ARLightEstimate	用于表示 HUAWEI AR Engine 对当前环境光照的估计
ARPlane	保存 HUAWEI AR Engine 识别的真实世界中的平面信息，派生自 ARTrackableBase。两个或者多个平面法向量朝向一致，位置靠近时会自动合并到一个父平面中，合并之后每个子平面的 getSubsumedBy()将返回父平面。被合并的子平面将继续被跟踪和返回
ARSceneMesh	用于环境 Mesh 跟踪时返回跟踪结果的环境识别信息，结果包括 Mesh 顶点坐标、三角形下标等。当前支持 2.5m 以内生成 Mesh，在相机移动的情况下，支持 Mesh 动态刷新

5.3.2　ARSession

ARSession 类用于管理 HUAWEI AR Engine 的整个运行状态，是 HUAWEI AR Engine 运行的基础，启动 ARSession 前，使用 ARSession.configure()设置相关配置，通过 ARSession 可以实现：

(1) 控制 HUAWEI AR Engine 的启动、暂停、结束等行为。

(2) 更新并获取 HUAWEI AR Engine 内部数据，如锚点、平面、可跟踪对象等。

具体见表 5.2。

表 5.2　ARSession 类的 API

方法名	类　　型	描　　述
configure(ARConfigBase config)	void	配置 session。由于设备能力差异，在调用该函数后，config 中的启用项 EnableItem 会被 service 尝试修改成设备支持的选项。应用需要在调用此接口后使用 config.getEnableItem()检查对应的设置项以便获取当前 session 的启用项
pause()	void	暂停 HUAWEI AR Engine，停止相机预览流，不清除平面和锚点数据，释放相机(否则其他应用无法使用相机服务)，不会断开与服务端的连接。调用后需要使用 resume()恢复
resume()	void	开始运行 ARSession，或者在调用 pause()以后恢复 ARSession 的运行状态

续表

方法名	类　　型	描　　述
stop()	void	停止 HUAWEI AR Engine，停止相机预览流，清除平面和锚点数据，并释放相机，终止本次会话。调用后，如果要再次启动，需要新建 ARSession
createAnchor（ARPose pose）	ARAnchor	使用 pose 创建一个持续跟踪的锚点
getAllAnchors()	Collection < ARAnchor >	获取所有锚点，包括 TrackingState 为 PAUSED，TRACKING 和 STOPPED。应用处理时需要仅绘制 TRACKING 状态的锚点，删除 STOPPED 状态的锚点
update()	ARFrame	更新 HUAWEI AR Engine 的计算结果，应用应在需要获取最新的数据时调用此接口，如相机发生移动以后，使用此接口可以获取新的锚点坐标、平面坐标、相机获取的图像帧等。如果 ARConfigBase.UpdateMode 为 BLOCKING，那么该函数会阻塞至有新的帧可用

5.3.3　ARConfigBase

所有 ARXXXTrackingConfig 的基类，包含各类设置项的枚举和公有设置项的接口。这些子类控制了 HUAWEI AR Engine 可使用的功能，具体见表 5.3。

表 5.3　ARConfigBase 类的 API

方法名/子类名	类　　型	描　　述
ARWorldTrackingConfig（子类）	ARWorldTrackingConfig	用于运动跟踪时配置 session，只支持环境识别。只能使用后置相机，默认对焦到无穷远
ARBodyTrackingConfig（子类）	ARBodyTrackingConfig	用于人体骨骼跟踪时配置 session，只支持人体识别。默认使用后置相机，默认自动对焦
ARFaceTrackingConfig（子类）	ARFaceTrackingConfig	用于人脸跟踪时配置 session，只支持人脸识别，目前外部输入与内部输入预览流两种方式仅支持前置

续表

方法名/子类名	类　　型	描　　述
ARHandTrackingConfig(子类)	ARHandTrackingConfig	用于手势跟踪时配置 session，只支持手势识别。默认使用后置相机，自动对焦模式
ARWorldBodyTrackingConfig(子类)	ARWorldBodyTrackingConfig	用于运动跟踪和人体骨骼跟踪同时运行时配置 session，支持环境识别与人体识别。只能使用后置相机，默认对焦无穷远
getLightingMode()	LightingMode	获取当前的光照估计模式

5.3.4 ARFrame

ARFrame 类是 HUAWEI AR Engine 系统的快照，包含 trackable 对象、点云、深度图等信息，仅由 ARSession.update()创建。具体见表 5.4。

表 5.4　ARFrame 类的 API

方法名	类　　型	描　　述
acquireCameraImage()	Image	在 camera 状态为 tracking 状态下，获取当前帧对应的图像，返回图像格式为 AIMAGE_FORMAT_YUV_420_888，只能在下一次 ARSession.update()前使用
acquireSceneMesh()	ARSceneMesh	返回当前帧对应的环境 Mesh

5.4 示 例 程 序

华为提供的示例代码已经基本包含所需的环境跟踪，可以通过示例代码熟悉相应的 API。打开 WorldActivity.java：

```
public class WorldActivity extends Activity {
    private static final String TAG = WorldActivity.class.getSimpleName();
    private static final int MOTIONEVENT_QUEUE_CAPACITY = 2;
    private static final int OPENGLES_VERSION = 2;
    private ARSession mArSession;
```

```
    private GLSurfaceView mSurfaceView;
    private WorldRenderManager mWorldRenderManager;
    private GestureDetector mGestureDetector;
    private DisplayRotationManager mDisplayRotationManager;
    private ArrayBlockingQueue< GestureEvent > mQueuedSingleTaps = new
ArrayBlockingQueue <>(MOTIONEVENT_QUEUE_CAPACITY);
    private String message = null;
    private boolean isRemindInstall = false;

    @Override
    protected void onCreate(Bundle savedInstanceState) {...}
    private void initGestureDetector() {...}
    private void onGestureEvent(GestureEvent e) {...}
    @Override
    protected void onResume() {...}
    private boolean arEngineAbilityCheck() {...}
    private void setMessageWhenError(Exception catchException){...}
    private void stopArSession(Exception exception) {...}
    @Override
    protected void onPause() {...}
    @Override
    protected void onDestroy() {...}
    @Override
    public void onWindowFocusChanged(boolean isHasFocus) {...}
}
```

属性方面，mArSession 用于管理 AR Engine 的整个生命周期，mWorldRenderManager 用于调用 rendering/WorldRenderManager. Java。

方法方面，onCreate(Bundle savedInstanceState)方法为应用初始化时执行，具体如下。

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.world_java_activity_main);     //关联 UI 界面
    mSurfaceView = findViewById(R.id.surfaceview);
    mDisplayRotationManager = new DisplayRotationManager(this);
                                    //设备旋转管理器，演示程序使用它来适应设备旋转
    initGestureDetector();
    mSurfaceView.setPreserveEGLContextOnPause(true);
    mSurfaceView.setEGLContextClientVersion(OPENGLES_VERSION);
```

```
        mSurfaceView.setEGLConfigChooser(8, 8, 8, 8, 16, 0);
        mWorldRenderManager = new WorldRenderManager(this, this);
                                                    //场景渲染管理以及 AREngine 运行时逻辑处理
        mWorldRenderManager.setDisplayRotationManage(mDisplayRotationManager);
        mWorldRenderManager.setQueuedSingleTaps(mQueuedSingleTaps);
        mSurfaceView.setRenderer(mWorldRenderManager);
        mSurfaceView.setRenderMode(GLSurfaceView.RENDERMODE_CONTINUOUSLY);
        PermissionManager.checkPermission(this); //获取摄像头权限
    }
```

initGestureDetector()、onGestureEvent(GestureEvent e)为手势动作绑定事件，用于用户交互部分。onPause()用于暂停 AR Engine，暂停后可通过 onResume()恢复；onDestroy()用于停止 AR Engine，停止后无法使用 onResume()恢复，只能重新初始化整个应用。方法 onResume()让 AR Engine 进入运行状态：

```
    @Override
    protected void onResume() {
        Log.d(TAG, "onResume");
        super.onResume();
        Exception exception = null;
        message = null;
        if (mArSession == null) {
            try {
                if (!arEngineAbilityCheck()) {
                    finish();
                    return;
                }
                mArSession = new ARSession(this);
                ARWorldTrackingConfig config = new ARWorldTrackingConfig(mArSession);
                config.setFocusMode(ARConfigBase.FocusMode.AUTO_FOCUS);
                config.setSemanticMode(ARWorldTrackingConfig.SEMANTIC_PLANE);
                mArSession.configure(config);
                mWorldRenderManager.setArSession(mArSession);
            } catch (Exception capturedException) {
                exception = capturedException;
                setMessageWhenError(capturedException);
            }
            if (message != null) {
                stopArSession(exception);
                return;
            }
```

```
        }
        try {
            mArSession.resume();
        } catch (ARCameraNotAvailableException e) {
            Toast.makeText(this, "Camera open failed, please restart the app", Toast.
    LENGTH_LONG).show();
            mArSession = null;
            return;
        }
        mDisplayRotationManager.registerDisplayListener();
        mSurfaceView.onResume();
    }
```

其中，ARWorldTrackingConfig config = new ARWorldTrackingConfig(mArSession)设置为应用场景只支持环境识别。而在游戏 Demo 中不仅需要支持环境，还需要支持人体姿态识别，故需改为：

```
ARWorldBodyTrackingConfig config = new ARWorldBodyTrackingConfig(mArSession);
```

注意，需要导入 ARWorldBodyTrackingConfig 包才能调用该类：

```
import com.huawei.hiar.ARWorldBodyTrackingConfig;
```

如此，该应用在此场景下既可以支持环境识别，也可以支持人体姿态识别。读者可将初始页面改回原来的 ChooseActivity.java 页面（具体修改方法见 4.3.3 节），尝试运行不同 ARConfig 下的 Demo（如图 5.4 所示）来体验不同 ARConfig 下的功能。

图 5.4　不同 ARConfig 下的 Demo 演示

目前仅环境识别与人体姿态识别可同时支持，ARFace 与 ARHand 只能单独支持，且 ARXXXConfig 类不可修改，读者需选择合适的 ARConfig。在以后，HUAWEI AR Engine 会将所有功能全部一体化，更加方便开发者进行第三方应用的开发，功能也将更加齐全。

打开 rendering/WorldRenderManager.java：

```
public class WorldRenderManager implements GLSurfaceView.Renderer {
    private static final String TAG = WorldRenderManager.class.getSimpleName();
    private static final int PROJ_MATRIX_OFFSET = 0;
    private static final float PROJ_MATRIX_NEAR = 0.1f;
    private static final float PROJ_MATRIX_FAR = 100.0f;
    private static final float MATRIX_SCALE_SX = -1.0f;
    private static final float MATRIX_SCALE_SY = -1.0f;
    private static final float[] BLUE_COLORS = new float[] {66.0f, 133.0f, 244.0f,
255.0f};
    private static final float[] GREEN_COLORS = new float[] {66.0f, 133.0f, 244.0f,
255.0f};
    private ARSession mSession;          //由 WorldAcivity.java 传入
    private Activity mActivity;          //由 WorldAcivity.java 传入
    private Context mContext;            //由 WorldAcivity.java 传入
    private TextView mTextView;          //UI,显示帧数
    private TextView mSearchingTextView; //UI,显示提示
    private int frames = 0;
    private long lastInterval;
    private float fps;                   //帧数

    private TextureDisplay mTextureDisplay = new TextureDisplay();
    private TextDisplay mTextDisplay = new TextDisplay();
    private LabelDisplay mLabelDisplay = new LabelDisplay();
    private ObjectDisplay mObjectDisplay = new ObjectDisplay();   //虚拟模型类
    private DisplayRotationManager mDisplayRotationManager;   //管理虚拟模型的旋转
    private ArrayBlockingQueue<GestureEvent> mQueuedSingleTaps;
    private ArrayList<VirtualObject> mVirtualObjects = new ArrayList<>();
                                                          //虚拟模型类 List
    private VirtualObject mSelectedObj = null;                    //选中虚拟模型

    //赋值 mContext 与 mActivity,并绑定 UI 组件
    public WorldRenderManager(Activity activity, Context context) {... }
    public void setArSession(ARSession arSession) {...}          //赋值 mSession
    public void setQueuedSingleTaps(ArrayBlockingQueue<GestureEvent>
queuedSingleTaps) {... }
```

```
    public void setDisplayRotationManage(DisplayRotationManager displayRotationManager) {... }
                                                                    //模型旋转
    @Override
    public void onSurfaceCreated(GL10 gl, EGLConfig config) {... }  //UI 刷新
    //UI 设置
    private void showWorldTypeTextView(final String text, final float positionX, final
float positionY) {...}
    @Override
    public void onSurfaceChanged(GL10 unused, int width, int height) {... }     //UI 更新
    @Override
    public void onDrawFrame(GL10 unused) {... }                                  //绘制帧
    //绘制虚拟模型
    private void drawAllObjects(float[ ] projectionMatrix, float[ ] viewMatrix, float
lightPixelIntensity) {...}
    private ArrayList < Bitmap > getPlaneBitmaps() {... }        //更新环境标签
    private Bitmap getPlaneBitmap(int id) {...}
    private void updateMessageData(StringBuilder sb) {...}      //更新 UI 信息
    private float doFpsCalculate() {... }                        //计算 fps
    private void hideLoadingMessage() {...}
    private void handleGestureEvent(ARFrame arFrame, ARCamera arCamera, float[]
projectionMatrix, float[ ] viewMatrix) {...}
    private void doWhenEventTypeDoubleTap(float[ ] viewMatrix, float[ ]
projectionMatrix, GestureEvent event) {... }
    private void doWhenEventTypeSingleTap(ARHitResult hitResult) {... }
    //操作屏幕
    private ARHitResult hitTest4Result(ARFrame frame, ARCamera camera, MotionEvent
event) {... }
    private static float calculateDistanceToPlane(ARPose planePose, ARPose
cameraPose) {...}
}
```

属性方面，mTextView 用于 UI 显示帧数 fps，mSearchingTextView 用于显示提示信息。mSession、mActivity、mContext 均由 WorldActivity. java 通过 WorldRenderManager (Activity activity, Context context) 和 setArSession(ARSession arSession) 方法赋值。mObjectDisplay 为展示的虚拟三维模型类，mVirtualObjects 为场景中三维模型的集合。mSelectedObj 为选中三维模型类，用于交互三维模型。mTextureDisplay、mTextDisplay、mLabelDisplay 用于场景识别后为场景中各物体打上标签。mDisplayRotationManager 用于管理三维虚拟模型的旋转。

方法方面，setDisplayRotationManage(DisplayRotationManager displayRotationManager)

用于管理虚拟模型在运行时的旋转，若开发的第三方应用需要在场景内对虚拟模型进行旋转，可调用该类。onSurfaceCreated(GL10 gl, EGLConfig config)方法用于 UI 的刷新，而 onSurfaceChanged(GL10 unused, int width, int height)则根据目前为横屏状态还是竖屏状态更新 UI。onDrawFrame(GL10 unused)用于每帧识别环境并调用其他方法来绘制单帧，可在此方法中处理开发者的逻辑。ARHitResult hitTest4Result(ARFrame frame, ARCamera camera, MotionEvent event)是用于处理用户操作屏幕的方法，包含点击和拖动。其余方法均为绘制 UI 信息的方法。由 getPlaneBitmaps()可知：

```
private ArrayList<Bitmap> getPlaneBitmaps() {
    ArrayList<Bitmap> bitmaps = new ArrayList<>();
    bitmaps.add(getPlaneBitmap(R.id.plane_other));
    bitmaps.add(getPlaneBitmap(R.id.plane_wall));
    bitmaps.add(getPlaneBitmap(R.id.plane_floor));
    bitmaps.add(getPlaneBitmap(R.id.plane_seat));
    bitmaps.add(getPlaneBitmap(R.id.plane_table));
    bitmaps.add(getPlaneBitmap(R.id.plane_ceiling));
    bitmaps.add(getPlaneBitmap(R.id.plane_door));
    bitmaps.add(getPlaneBitmap(R.id.plane_window));
    bitmaps.add(getPlaneBitmap(R.id.plane_bed));
    return bitmaps;
}
```

目前，HUAWEI AR Engine 对真实世界的理解，支持识别墙体、地面、桌子、座位、天花板、门、窗户、床，如图 5.5 所示。开发者可依据标签位置开发相应的功能，例如，只能在桌子上放杯子类型的虚拟模型，只能在天花板上放置吊灯类型的模型。

图 5.5　HUAWEI AR Engine 的物体识别功能

小　　结

本章介绍了增强现实下的环境跟踪概念及部分与环境跟踪相关的关键技术，包括图像分类、目标定位、物体识别等。然后介绍了 HUAWEI AR Engine 中的环境跟踪技术（光照估计、平面语义、图像跟踪、环境 Mesh），并介绍了 HUAWEI AR Engine 提供的有关环境跟踪的关键 API（ARSession、ARConfigBase、ARFrame）。最后梳理了示例代码的结构及功能，展示了不同 ARConfig 下的 Demo 功能以及识别平面的功能，并集成了 ARBody 的识别功能。本书的示例 Demo 是基于 HUAWEI AR Engine 官方提供的示例代码所开发的，读者在开发前需熟悉示例代码。

习　　题

1. 阅读环境跟踪技术相关文献，并做一个综述。

2. 体验不同 ARConfig 下 Demo 功能以及平面识别的功能。

3. 理解平面检测的实现代码，并将平面替换成虚拟三维平面（虚拟墙体替换真实环境的墙体）。