IOS端openVisionSdk集成说明

一、Xcode配置工程

1. 获取相关资源压缩包（由阿里云相关人员提供下载链接）后，解压压缩包，可看到如下资源文件framework包及支持相关能力的bin、license文件。如下图：



1. 解压openvison\_sdk\_ios.zip,得到OpenVision.framework。项目下新建Frameworks，放入解压得到的OpenVision.framework，在OPV.bundle文件夹下放入bin、license文件（需要集成什么能力导入对应的bin、license文件即可）。如下图：

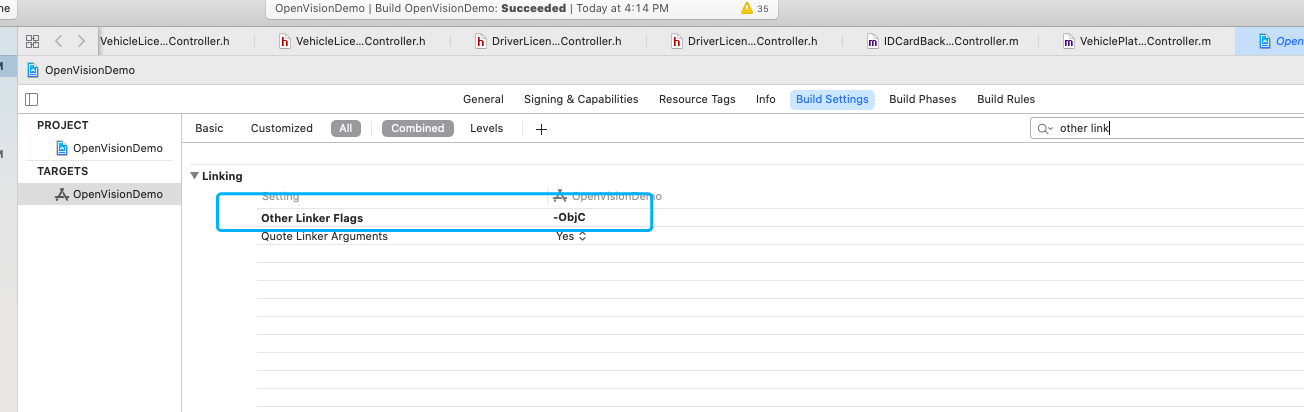


3、需要集成一些系统的库，项目设置target -> 选项卡Build Phases ->Linked Binary With Libraries如下图：



1. 需要配置相机的权限，项目下的Info.plist文件并在target 中添加如下flag，如下图：





二、功能实现：

1、通过相机预览实现银行卡、身份证（正、反面）、驾驶证（正、反面）、行驶证（正、反面）、车牌的信息识别。（注：相关功能的实现依赖于framework库中是否包含该功能及项目中是否有对应的bin和license文件）。集成过程如下：

在调用证件扫描功能的类中引入下面的头文件：

#**import** <OpenVision/OpenVerison.h>

声明相机类对象、算法类对象、扫描位置框，遵循OPVCameraDelegate 代理，卡类识别除了Engine声明和加载文件license、bin文件不同，其他都相同（可封装基类统一处理，详见Demo）：

1. @interface ViewController () <OPVCameraDelegate>
2. //声明相机类对象
3. @property (nonatomic, strong) OPVCamera    \*opvCamera;
4. //声明算法类对象（银行卡）
5. @property (nonatomic, strong) OPVBankCardEngine \*ocrEngine;
6. //添加扫描位置区域，可根据需求自行设置位置、大小，一般设置为身份证大小比例。
7. @property (nonatomic, strong) UIImageView \*imageView;
8. @end

视图添加相机View，设置扫描区域框：

1. //添加相机View
2. CGRect frame = self.view.bounds;
3. OPVCamera \*opvCamera = [[OPVCamera alloc] initWithCameraFrame:frame cameraHandler:^(OPVCameraRunningStatus status, NSError \*error) {
4. NSLog(@"相机启动状态码%ld",status);
5. }];
6. [self.view addSubview:opvCamera.cameraView];
7. opvCamera.delegate = self;
8. \_opvCamera = opvCamera;
9. //扫描区域框设置
10. CGRect rect = self.view.bounds;
11. //位置大小可根据需求自行设置
12. CGRect scanFrame = CGRectMake(0, rect.size.height\*0.15, rect.size.width, rect.size.height\*0.3516);
13. //资源文件为一个蓝色的框，可自行设置，也可查看Demo文件设置
14. NSString \*bundlePath = [[[NSBundle mainBundle] bundlePath] stringByAppendingPathComponent:@"OPV.bundle"];
15. NSString \* scanImagePath = [bundlePath stringByAppendingPathComponent:@"scan"];
16. UIImage \*scanImage = [UIImage imageNamed:scanImagePath];
17. \_imageView = [[UIImageView alloc] initWithFrame:scanFrame];
18. [self.view addSubview:\_imageView];
19. \_imageView.image = scanImage;
20. \_imageView.backgroundColor = [UIColor clearColor];

加载算法相关license、bin文件，配置算法相关参数，初始化算法。

1. - (**void**)crateXMediaEngine {
2. //create xmedia 相关文件加载
3. NSString \*bundlePath = [[[NSBundle mainBundle] bundlePath] stringByAppendingPathComponent:@"OPV.bundle"];
4. OPVConfig \*config = [[OPVConfig alloc] init];
5. config.license =  [bundlePath stringByAppendingPathComponent:@"bankcard.license"];
6. config.model = [bundlePath stringByAppendingPathComponent:@"bankcard.bin"];
7. //kOPVProcessFullPictureOutput：算法处理结果中是否输出处理成功当前帧图片 0不输出 1输出
8. //kOPVProcessTimeInterval：50ms处理一次算法
9. config.options = @{kOPVProcessFullPictureOutput:@(1),kOPVProcessTimeInterval:@(0.05)};
11. NSError \*error;
12. \_ocrEngine = [[OPVBankCardEngine alloc] initWithConfig:config error:&error];
13. **if** (!error) {
14. //扫描区域设置，位置即为上面设置扫描框的位置。不设置的话扫描区域为整个相机View，为提高扫描准确度
15. //建议设置
16. [\_opvCamera attachEngine:\_ocrEngine options:@{kOPVOptionsROI:@[@(0),@(0.15),@(1.0),@(0.3516)]}];
17. } **else** {
18. NSLog(@"ocrEngine init failed!");
19. }
20. }

其它几个证件识别初始化及文件加载说明，替换一下即可：

身份证：

1. //身份证正面：
2. @property (nonatomic, strong) OPVIDCardEngine  \*opvOCREngine;
3. OPVConfig \*config = [[OPVConfig alloc] init];
4. config.license =  [bundlePath stringByAppendingPathComponent:@"IDCard/idcard\_front.license"];
5. config.model = [bundlePath stringByAppendingPathComponent:@"IDCard/idcard\_front.bin"];
6. //身份证反面：
7. @property (nonatomic, strong) OPVIDCardEngine  \*opvOCREngine;
8. OPVConfig \*config = [[OPVConfig alloc] init];
9. config.license =  [bundlePath stringByAppendingPathComponent:@"IDCard/idcard\_back.license"];
10. config.model = [bundlePath stringByAppendingPathComponent:@"IDCard/idcard\_back.bin"];

车牌：

1. //车牌：
2. @property (nonatomic, strong) OPVPlateNumberEngine  \*opvOCREngine;
3. OPVConfig \*config = [[OPVConfig alloc] init];
4. config.license =  [bundlePath stringByAppendingPathComponent:@"VehiclePlate/vehicleplate.license"];
5. config.model = [bundlePath stringByAppendingPathComponent:@"VehiclePlate/vehicleplate.bin"];

行驶证：

1. //行驶证正面：
2. @property (nonatomic, strong) OPVVehicleLicenseEngine  \*opvOCREngine;
3. OPVConfig \*config = [[OPVConfig alloc] init];
4. config.license =  [bundlePath stringByAppendingPathComponent:@"VehicleLicense/vehiclelicense\_front.license"];
5. config.model = [bundlePath stringByAppendingPathComponent:@"VehicleLicense/vehiclelicense\_front.bin"];
6. //行驶证背面：
7. @property (nonatomic, strong) OPVVehicleLicenseEngine  \*opvOCREngine;
8. OPVConfig \*config = [[OPVConfig alloc] init];
9. config.license =  [bundlePath stringByAppendingPathComponent:@"VehicleLicense/vehiclelicense\_back.license"];
10. config.model = [bundlePath stringByAppendingPathComponent:@"VehicleLicense/vehiclelicense\_back.bin"];

驾驶证：

1. //驾驶证正面：
2. @property (nonatomic, strong) OPVDrivingLicenseEngine  \*opvOCREngine;
3. OPVConfig \*config = [[OPVConfig alloc] init];
4. config.license =  [bundlePath stringByAppendingPathComponent:@"DriverLicense/driverlicense\_front.license"];
5. config.model = [bundlePath stringByAppendingPathComponent:@"DriverLicense/driverlicense\_front.bin"];
6. //驾驶证背面：
7. @property (nonatomic, strong) OPVDrivingLicenseEngine  \*opvOCREngine;
8. OPVConfig \*config = [[OPVConfig alloc] init];
9. config.license =  [bundlePath stringByAppendingPathComponent:@"DriverLicense/driverlicense\_back.license"];
10. config.model = [bundlePath stringByAppendingPathComponent:@"DriverLicense/driverlicense\_back.bin"];

调用算法：

1. [self crateXMediaEngine];

算法结果回调：

1. - (**void**)OPVResultWithEngine:(OPVBaseEngine \*)engine result:(OPVResult \*)result {
2. **if** (!result.error && result.cvResult.count > 0) {
3. NSString \*resultText = @"";
4. **for** (**int** i = 0; i < result.cvResult.count; i++) {
5. **if** (result.cvResult[i].key.length > 0) {
6. resultText = [resultText stringByAppendingString:[NSString stringWithFormat:@"%@: ",result.cvResult[i].key]];
7. } **else** {
8. resultText = [resultText stringByAppendingString:@"label:"];
9. }
10. resultText = [resultText stringByAppendingString:result.cvResult[i].label];
11. resultText = [resultText stringByAppendingFormat:@"\n"];
12. resultText = [resultText stringByAppendingString:@"conf:"];
13. resultText = [resultText stringByAppendingString:[NSString stringWithFormat:@"%f ",result.cvResult[i].conf]];
14. }
15. NSLog(@"%@",resultText);
16. \_\_weak typeof(self) wself = self;
17. //通知主线程刷新
18. dispatch\_async(dispatch\_get\_main\_queue(), ^{
19. \_\_strong typeof(self) sSelf = wself;
20. //result.image、resultText为结果返回，可自定义控件接收展示
21. sSelf.roiImageResult.image = result.image;
22. sSelf.recResult.text = resultText;
23. });
24. }
25. }

生命周期管理：

1. - (**void**)viewWillDisappear:(**BOOL**)animated {
2. **if** (self.opvCamera) {
3. //关闭相机，关闭后想再次打开调startCamera
4. [self.opvCamera stopCamera];
5. //销毁整个算法引擎
6. [self.opvCamera removeEngine:\_ocrEngine];
7. }
8. }

2、相关功能实现，实现人像抠图的效果。集成过程如下：

在调用证件扫描功能的类中引入这个头文件：

#**import** <OpenVision/OpenVerison.h>

声明相机类对象、算法类对象，遵循OPVCameraDelegate 代理：

1. @interface ViewController () <OPVCameraDelegate>
2. //声明相机类对象
3. @property (nonatomic, strong) OPVCamera    \*opvCamera;
4. //声明算法类对象
5. @property (nonatomic, strong) OPVSegmentationEngine \*segEngine;
6. @end

视图添加相机View：

1. //添加相机View
2. CGRect frame = self.view.bounds;
3. OPVCamera \*opvCamera = [[OPVCamera alloc] initWithCameraFrame:frame cameraHandler:^(OPVCameraRunningStatus status, NSError \*error) {
4. NSLog(@"相机启动状态码%ld",status);
5. }];
6. [self.view addSubview:opvCamera.cameraView];
7. opvCamera.delegate = self;
8. \_opvCamera = opvCamera;

加载算法相关license、bin文件，配置算法相关参数，初始化算法：

1. - (**void**)createSegmentationEngine {
2. //create xmedia 加载相关文件
3. NSString \*bundlePath = [[[NSBundle mainBundle] bundlePath] stringByAppendingPathComponent:@"OPV.bundle"];
4. OPVConfig \*config = [[OPVConfig alloc] init];
5. config.license =  [bundlePath stringByAppendingPathComponent:@"segmentation.license"];
6. config.model = [bundlePath stringByAppendingPathComponent:@"segmentation.bin"];
7. config.options = @{kOPVProcessTimeInterval:@(0.07)};//70ms处理一次算法
9. \_segEngine = [[OPVSegmentationEngine alloc] initWithConfig:config error:nil];
10. //设置可识别区域，不传的话识别整个相机View
11. [\_opvCamera attachEngine:\_segEngine options:@{kOPVOptionsROI:@[@(0),@(0),@(1.0),@(1.0)]}];
12. }

调用算法：

1. [self createSegmentationEngine];

算法结果回调：

1. - (**void**)OPVResultWithEngine:(OPVBaseEngine \*)engine result:(OPVResult \*)result {
2. **if** (!result.error) {
3. result = (OPVSegmentationResult \*)result;
4. unsigned **char** \*data = ((OPVSegmentationResult \*)result).data;
5. **int** pixFormat = ((OPVSegmentationResult \*)result).format;
6. **int** width = ((OPVSegmentationResult \*)result).width;
7. **int** height = ((OPVSegmentationResult \*)result).height;
8. UIImage \*rstImg = [OPVUtils convertToImageWithPixelData:data format:(OPVPixelFomat)pixFormat width:width height:height];
9. \_\_weak typeof(self) wself = self;
10. //通知主线程刷新
11. dispatch\_async(dispatch\_get\_main\_queue(), ^{
12. \_\_strong typeof(self) sSelf = wself;
13. //rstImg为结果图片，可自定义控件接收展示
14. sSelf.imageView.image = rstImg;
15. });
16. }
17. }

相机前后摄像头切换：

1. [self.opvCamera switchCamera:nil];

生命周期管理：

1. - (**void**)viewWillDisappear:(**BOOL**)animated {
2. **if** (self.opvCamera) {
3. //关闭相机，关闭后想再次打开调startCamera
4. [self.opvCamera stopCamera];
5. //销毁整个算法引擎
6. [self.opvCamera removeEngine:\_segEngine];
7. }
8. }
9. 相关功能实现,人体姿态，获取人体描边的位置点。集成过程如下：

在调用证件扫描功能的类中引入这个头文件：

#**import** <OpenVision/OpenVerison.h>

声明相机类对象、算法类对象，遵循OPVCameraDelegate 代理：

1. @interface ViewController () <OPVCameraDelegate>
2. //声明相机类对象
3. @property (nonatomic, strong) OPVCamera    \*opvCamera;
4. //声明算法类对象
5. @property (nonatomic, strong) OPVHumanPoseEngine \*poseEngine;
6. @end

视图添加相机View：

1. //添加相机View
2. CGRect frame = self.view.bounds;
3. OPVCamera \*opvCamera = [[OPVCamera alloc] initWithCameraFrame:frame cameraHandler:^(OPVCameraRunningStatus status, NSError \*error) {
4. NSLog(@"相机启动状态码%ld",status);
5. }];
6. [self.view addSubview:opvCamera.cameraView];
7. opvCamera.delegate = self;
8. \_opvCamera = opvCamera;

加载算法相关license、bin文件，配置算法相关参数，初始化算法：

1. - (**void**)crateHumanposeEngine {
2. //create xmedia 加载相关文件
3. NSString \*bundlePath = [[[NSBundle mainBundle] bundlePath] stringByAppendingPathComponent:@"OPV.bundle"];
4. OPVConfig \*config = [[OPVConfig alloc] init];
5. config.license =  [bundlePath stringByAppendingPathComponent:@"humanpose/humanpose.license"];
6. config.model = [bundlePath stringByAppendingPathComponent:@"humanpose/humanpose.bin"];
7. config.options = @{kOPVProcessTimeInterval:@(0.07)};//算法70ms处理一次
9. \_poseEngine = [[OPVHumanPoseEngine alloc] initWithConfig:config error:nil];
10. [self.opvCamera attachEngine:\_poseEngine options:@{}];
11. }

调用算法：

1. [self crateHumanposeEngine];

算法结果回调：

1. - (**void**)OPVResultWithEngine:(OPVBaseEngine \*)engine result:(OPVResult \*)result {
2. **if** (!result.error) {
3. OPVHumanPoseResult \*poseResult = (OPVHumanPoseResult \*)result;
4. //poseResult.humanPoseResult 为坐标点数据数组，多个人像的话会有多个数组，每个数组有
5. //14个人体点，可根据需求自行画出各点，并连线（可参考Demo）进行其他操作
6. **if** ([poseResult.humanPoseResult count]==0) {
7. \_\_weak typeof(self) wself = self;
8. dispatch\_async(dispatch\_get\_main\_queue(), ^{
9. //无数据返回
10. [wself.humanPoseLineView setPosePointArray:nil];
11. });
12. **return**;
13. }
15. NSMutableArray \*pointArrays = [[NSMutableArray alloc] init];
16. **for** (**int** i = 0; i < poseResult.humanPoseResult.count; i++) {
17. NSArray \*items = poseResult.humanPoseResult[i];
19. NSMutableArray \*tempPointArr = [[NSMutableArray alloc] init];
20. **for** (**int** j = 0; j < items.count; j ++) {
21. CVResult \*item = items[j];
22. CGPoint point = CGPointMake(([item.pos[0] floatValue]) \* poseResult.pixelWidth, ([item.pos[1] floatValue]) \* poseResult.pixelHeight);
23. **if** (i==0 && j==0) {
24. NSLog(@"human head pos:[%f,%f]",point.x,point.y);
25. }
26. // frame -> view 坐标转换 \_viewWidth \_viewHeight为视图宽高
27. **if** (poseResult.pixelWidth !=0 && poseResult.pixelHeight != 0) {
28. CGPoint newPoint = CGPointMake(point.x \* \_viewWidth / poseResult.pixelWidth, point.y \* \_viewHeight / poseResult.pixelHeight);
29. NSValue \*pointValue = [NSValue valueWithCGPoint:newPoint];
30. [tempPointArr addObject:pointValue];
31. } **else** {
32. NSValue \*pointValue = [NSValue valueWithCGPoint:point];
33. [tempPointArr addObject:pointValue];
34. }
35. }
36. //pointArrays为所有转换完的点数据，pointArrays[i]为每组数据，每组有固定14个点
37. [pointArrays addObject:tempPointArr];
38. }
39. }
40. }

生命周期管理：

1. - (**void**)viewWillDisappear:(**BOOL**)animated {
2. **if** (self.opvCamera) {
3. //关闭相机
4. [self.opvCamera stopCamera];
5. //销毁整个算法引擎
6. [self.opvCamera removeEngine:\_poseEngine];
7. }
8. }

人体骨架连线规则，每组固定返回14个点，排序即为下标，按照固定点连线即可（每个坐标点都有参数conf， conf大于0 再画这个点和对应连线）如图：



三、其他：

1、证件检测必须为所选证件项进行检测，包括正、反面的区别，否则会检测没反应或检测失败。

2、封装好的相机OPVCamera相关操作请参考OPVCamera.h文件。

3、报错 You must rebuild it with bitcode enabled (Xcode setting ENABLE\_BITCODE) 解决方法。



。

4.Vin识别功能实现,人体姿态，获取人体描边的位置点。集成过程如下：

在调用Vin扫描功能的类中引入这个头文件：

#**import** <OpenVision/OpenVerison.h>

声明相机类对象、算法类对象，遵循OPVCameraDelegate 代理：

1. @interface ViewController () <OPVCameraDelegate>
2. //声明相机类对象
3. @property (nonatomic, strong) OPVCamera    \*opvCamera;
4. //声明算法类对象
5. @property (nonatomic, strong) OPVVinCodeEngine \*vinEngine;
6. @end

视图添加相机view

OPVCamera \*opvCamera = [[OPVCamera alloc] initWithCameraFrame:frame cameraHandler:^(OPVCameraRunningStatus status, NSError \*error) {

        NSLog(@"相机启动状态码%ld",status);

    }];

    [self.view addSubview:opvCamera.cameraView];

    opvCamera.delegate = self;

    \_opvCamera = opvCamera;

加载vin模块能力

1. - (**void**)crateVinSpotEngine {
2. //create xmedia 加载相关文件
3. NSString \*bundlePath = [[[NSBundle mainBundle] bundlePath] stringByAppendingPathComponent:@"OPV.bundle"];
4. OPVConfig \*config = [[OPVConfig alloc] init];
5. config.license =  [bundlePath stringByAppendingPathComponent:@"VinCodeSpot/vincode.license"];
6. config.model = [bundlePath stringByAppendingPathComponent:@"VinCodeSpot/vincode.bin"];
7. config.options = @{kOPVProcessTimeInterval:@(0.05)};//算法50ms处理一次
9. \_vinEngine = [[OPVVinCodeEngine alloc] initWithConfig:config error:nil];
10. [self.opvCamera attachEngine:\_poseEngine options:@{}];
11. }

算法结果回调：

1. - (**void**)OPVResultWithEngine:(OPVBaseEngine \*)engine result:(OPVResult \*)result {
2. **if** (!result.error) {
3. result = (OPVSegmentationResult \*)result;
4. unsigned **char** \*data = ((OPVSegmentationResult \*)result).data;
5. **int** pixFormat = ((OPVSegmentationResult \*)result).format;
6. **int** width = ((OPVSegmentationResult \*)result).width;
7. **int** height = ((OPVSegmentationResult \*)result).height;
8. UIImage \*rstImg = [OPVUtils convertToImageWithPixelData:data format:(OPVPixelFomat)pixFormat width:width height:height];
9. \_\_weak typeof(self) wself = self;
10. //通知主线程刷新
11. dispatch\_async(dispatch\_get\_main\_queue(), ^{
12. \_\_strong typeof(self) sSelf = wself;
13. //rstImg为结果图片，可自定义控件接收展示
14. sSelf.imageView.image = rstImg;
15. });
16. }
17. }