Computer Science and Application 计算机科学与应用, 2020, 10(11), 2131-2140
Published Online November 2020 in Hans. http://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2020.1011224

Hans汉斯

基于ORB-SLAM的智能手机实时同步定位建图与半稠密三位重构实现

卢鹏飞[1*]，吕剑清[2]，王先兵[2#]，何　涛[3]，吴中鼎[4]，柴婉秋[4]

[1]深圳市腾讯计算机有限公司，广东　深圳
[2]武汉大学国家网络安全学院，湖北　武汉
[3]中国科学院计算技术研究所，北京
[4]贵阳铝镁设计研究院有限公司，贵州　贵阳
Email: *541331815@qq.com

收稿日期：2020年11月5日；录用日期：2020年11月20日；发布日期：2020年11月27日

摘　要

实时定位与环境的三维重构已经成为越来越活跃的研究课题。随着增强现实技术的快速发展，移动设备了解自身位置与周围环境的需求越来越重要。增强现实想要实现追踪、物体识别，实现虚拟数据与现实场景相结合，就必须得到精确而可靠的位姿估计和周围详细的三维场景。相对于其他诸如谷歌眼镜等专用于AR的设备，智能手机作为当前应用范围最广的设备，在智能手机上实现实时定位与制图就变得非常重要。本次研究针对这种情况，基于当前最先进的单目视觉SLAM算法之一ORB-SLAM上实现了智能手机平台上的实时定位与三维重构，并对ORB-SLAM算法进行针对于Android上的优化，使应用可以在近年的智能手机硬件上进行实时定位与三维重构的效果，并且在ORB-SLAM的基础上实现了半稠密的三维重建，可以获得周围环境的轮廓信息而不是地图上稀疏的散点。应用获得到的结果可以用于物体识别与导航以及更有意义并且更加丰富的AR应用。

关键词

ORB-SLAM，半稠密，同时定位与制图，AR，三维重构

The Alization of Real-Time Semi-Dense 3D Reconstruction and Simultaneous Localization and Mapping of Smart Phone Based on Orb-Slam

Pengfei Lu[1*], Jianqing Lv[2], Xianbing Wang[2#], Tao He[3], Zhongding Wu[4], Wanqiu Chai[4]

[*]第一作者。
[#]通讯作者。

文章引用: 卢鹏飞, 吕剑清, 王先兵, 何涛, 吴中鼎, 柴婉秋. 基于 ORB-SLAM 的智能手机实时同步定位建图与半稠密三位重构实现[J]. 计算机科学与应用, 2020, 10(11): 2131-2140. DOI: 10.12677/csa.2020.1011224

[1]Shenzhen City Tencent Computer System Co. Ltd., Shenzhen Guangdong
[2]School of Cyber Science and Engineering, Wuhan University, Wuhan Hubei
[3]Institute of Computing Technology, Chinese Academy of Sciences, Beijing
[4]Guiyang Aluminum Magnesium Design and Research Institute Co., Ltd., Guiyang Guizhou
Email: *541331815@qq.com

Received: Nov. 5th, 2020; accepted: Nov. 20th, 2020; published: Nov. 27th, 2020

Abstract

Real-time location and three-dimensional reconstruction of the environment have become an increasingly active research topic. Augmented reality wants to implement tracking, object recognition, data and virtual reality scenarios combining, they must be aware of accurate and reliable pose estimation and around detailed 3D scenes. Compared to others such as Google for AR glasses and other special equipments, smart phone, as the current most widely used equipment, real-time locating and mapping on smartphones become very important. In this case, based on orb-slam, one of the most advanced monocular vision slam algorithms, the real-time location and 3D reconstruction on the smartphone platform are realized. The orb-slam algorithm is optimized for Android so that applications can be real-time location and the effect of three-dimensional reconstruction, using the smartphone hardware in recent years. And it implements a semi-dense three-dimensional reconstruction on the basis of ORB-SLAM, the contour information may be obtained in surroundings instead of sparse points on the map. The results of the applications got can be used for object recognition and navigation as well as more meaningful and richer AR applications.

Keywords

ORB-SLAM, Semi-Dense, Simultaneous Localization and Mapping, Augment Reality, Three-Dimensional Reconstruction

1. 引言

当前智能手机作为应用最广的装置之一，在硬件上已经允许提供 AR (Augment Reality)必须的三维场景。但是 AR 应用要想实现跟踪，物体检测的功能，就必须获得世界的详细的三维场景，而且由于显示场景总是在不停地发生变化，要求算法能够快速地进行场景的更改和修正。所以获取周围环境信息与当前位置是智能手机上实现增强现实应用的必要条件。原本一直活跃在机器人领域的 SLAM 算法进入 AR 的视野。

同步定位建图(Simultaneous Localization and Mapping，简称 SLAM)最早由 Randall Smith 和 Peter Cheeseman 等人于 1988 年提出[1]，最早被用于机器人定位问题。SLAM 可以描述为未知环境中估计移动传感器的位置，与此同时进行周围三维环境的重建问题。

2014 年提出的 ORB-SLAM 算法(Oriented FAST and Rotated BRIEF SLAM，简称 ORB-SLAM) [2]在 PTAM 的基础上加入了闭环检测[3]，闭环[4]，以及在大规模时的误差调整方法[5]，以此构成一个新的

SLAM 系统。ORB-SLAM 系统克服了 PTAM 的主要不足，但是由于 ORB-SLAM 仍然基于特征点，所以依然只能得到稀疏的地图点。

跟 ORB 同时提出的直接 SLAM 算法 LSD-SLAM (Large-Scale Direct Monocular SLAM，简称 LSD-SLAM) [6]，实现了一个实时的半稠密 SLAM 算法。这种方法使用直接图像对齐结合基于滤波的半稠密深度图估计。同样使用直接方法进行追踪的还有 2010 年提出的 DTAM (Dense Tracking and Mapping，简称 DTAM) [7]。与 LSD-SLAM 不同的是 DTAM 是一个生成稠密地图的 SLAM 算法。

在 LSD-SLAM 提出后不久，ORB-SLAM 的作者对 LSD-SLAM 半稠密原理[8]进行修改，得到了一种适用于宽基线帧之间的半稠密重建方法[9]。由于它的半稠密方法是在宽基线帧之间的，它的理论上的误差必定小于窄基线帧之间的计算。并且通过新的融合方式和对异常点严格的检测标准，产生了令人难以置信的结果。

对于现今的 SLAM 算法来说，可以根据它所生成的地图密度分为稀疏，半稠密和稠密三种。其中稀疏地图 SLAM 主要用于定位，基本丧失了外界环境的形状信息，优点是速度很快，修正和使用的时候速度都很快，当前很多基于特征的方法都是使用的这种制图。稠密建图会将图像中的所有区域的形状及纹理都展示出来，但是由于点的数量过于庞大，需要在当前最高性能的 GPU 上工作。而半稠密建图综合了稀疏建图速度快和稠密制图具备形状和轮廓信息的特点，在产生具备大部分形状信息和高纹理区域地图的同时摆脱对 GPU 并行计算的依赖，使用 CPU 并行计算生成地图。显然半稠密的地图可以得到更多的环境信息，用途更多。

目前的 AR 设备主要是智能手机和其他专有设备，智能手机相对于专用设备来说更加普及，更加廉价。但是目前在智能手机上的 SLAM 算法能够达到实时性要求的并不多。本实验的目的是实现智能手机上的实时定位与三维建模功能。智能手机的实时定位功能基于最新提出的 ORB-SLAM 算法。ORB-SLAM 目前作为最新的 SLAM 方案之一，还没有将其移植到手机上的先例。但是 ORB-SLAM 公布出的开源算法中并没有之后加入的半稠密的功能，只能得到稀疏的点云。稀疏的地图只在追踪相机的位置的时候有用，不能提供给 AR 应用提供场景的有效信息。

主要工作是完成了智能手机上基于 ORB-SLAM 算法的实时定位功能，并且优化 ORB-SLAM 在智能手机上的运行效率。之后根据论文[9]中半稠密的原理实现了基于 ORB-SLAM 满足性能要求的半稠密三维重构。

主要内容可以分为以下三点：

(1) 将 ORB 算法移植到 Android 手机上，优化算法在 Android 上的实现效果。

(2) 在 Android 平台上重建 ORB-SLAM 的输入输出部分，改进在 Android 上的运行效率。

(3) 根据论文[9]独立实现满足性能需要的基于 ORB-SLAM 算法的半稠密建模功能。

2. 算法设计

目的是在 Android 实现满足实时性要求的半稠密 SLAM 算法。总体任务可以划分成两个部分：1) 选择合适的 SLAM 算法，SLAM 算法是整个项目的核心模块，通过摄像机输入连续的图像在确定摄像机位置的同时建立周围环境的半稠密的三维模型；2) SLAM 算法向 Android 平台的移植，这里包含在 Android 上获取 SLAM 算法需要的参数，获取 Android 摄像机的产生的图像作为输入，以及 Android 的应用层与算法所在的 Native 的交互。

2.1. SLAM 算法的选择

最新的 SLAM 算法有很多种，像 LSD-SLAM、DPPTAM [10]、DTAM 和 ORB-SLAM，繁多的 SLAM

系统各自有着各自的优点和缺陷。DTAM 使用的直接法，但是由于直接生成稠密地图的原因，需要的运算量非常大，需要 GPU 参与并行运算，移植到 Android 程序上比较困难。DPPTAM 与 LSD-SLAM 的原理一致，都是直接使用图片像素进行的相机位姿的修正，最后生成半稠密的地图，包含了场景中高梯度的区域。一般高梯度的区域就包含了场景中主要的形状信息。但是对于直接法 SLAM 来说，由于其涉及到每个像素的实际位置，需要的相机参数信息非常严格。由于手机平台的限制，在手机上只能算出相机内参的理想值，并且得不到畸变参数。畸变参数只有通过标定获取，对于用户来说，进行相机标定是不现实的。排除了直接法 SLAM，就只能使用基于特征点的 SLAM 应用来减小相机畸变参数的不准确所造成的影响。

所以选择了最新的基于特征点的 SLAM 算法 ORB-SLAM 算法，在它的基础上实现半稠密算法，获取包含场景三维形状的地图。

2.2. 基于 ORB-SLAM 的半稠密

ORB-SLAM 的半稠密算法是一个适用于宽基线的半稠密算法。它只使用关键帧进行半稠密算法的计算，对于任意两个关键帧之间，在极线上搜索同名点，逐像素计算高梯度区域内点的逆深度，然后使用卡方检验滤去异常值，使用方差的倒数计算逆深度的加权平均值。最后将改点投影到相邻的关键帧上，寻找投影点附近的兼容点，只有找到兼容点的邻近关键帧的数量大于某个阈值时，才会保留改点。

基于 ORB-SLAM 半稠密的实现是按照 ORB-SLAM 半稠密论文 Probabilistic Semi-Dense Mapping from Highly Accurate Feature-Based Monocular SLAM [11]中的原理进行的，为了加快程序的运行速度，对中间有些细微的地方做了修改。以下是方法的大致步骤：

(1) 对于每一个关键帧，进行逐像素的计算高梯度区域的逆深度。每一个像素计算在 N 个邻近关键帧上的极线，沿极线搜索改点的同名点，根据同名点和逆深度计算出一个逆深度的假定值。这样对于一个关键帧上每个像素就得到了 N 个对应于邻近关键帧的假定值。

(2) 每一个逆深度假定值都符合应当是一个高斯分布。在不考虑相机位置计算误差的情况下，这个高斯分布只受图像噪声、视差和匹配错误的影响。

(3) 由于在关键帧上的极线很宽，在图像上的搜索区域就会很大。为了处理相似像素点和梯度等值线区域造成的错误点，我们从 N 个假定值中得到其中一个最大数量的相互兼容点的子集，对这个子集中的逆深度进行融合。每一个在逆深度图上的逆深度都应该符合一个高斯分布。

(4) 由[11]中提到的正规化步骤，将上几步得到的点与周围点进行平滑调整。将一个像素点的逆深度设置为周围点的加权平均值。

(5) 在每个关键帧的深度都已经计算出来后，对于每一个点将其投影到他的邻近关键帧上。比较投影点与周围邻近点的一致性，邻近关键帧中一致点数量较少的时候，认为该点是异常点并删除该点。对于存在较多数量关键帧的邻近点数量，将深度设置为优化后的参数。

3. Android 上的移植和优化

3.1. ORB-SLAM 在 Android 上的优化

ORB-SLAM 在电脑上运行速度非常快，一般在每帧 50 毫秒左右，但是在手机上的运行结果并不理想，在手机上不加任何修改或优化的运行时间是每帧 1 秒左右，完全不能够满足实时运行的需要。所以在 Android 上的 SLAM 需要进行特别的修改和优化。并且优化应当在不影响算法本身的运行结果情况下进行。

3.1.1. 相机输入调整

由于算法本身是一个对图片进行处理的过程可以将相机输入的图片进行缩放后再运算，缩放图片肯定可以提升算法的运行效率。在获得相机输入的时候，将图片按比例缩放到宽为 400 像素的固定值。

3.1.2. 修改 ORB-SLAM 的参数需求

ORB-SLAM 是一个基于特征点的 SLAM 算法，需要一定的特征点才能够进行实时定位。这里可以优化的地方有两个方面，一个是特征点提取时的总数量要求，另外一个是当前关键帧没有丢失的特征点数量要求。我将特征点提取的总数量设置为原先的一半，原先每张图片要求提取 1000 个特征点，我将它设置为 500，大大加快了特征匹配的时间。另一方面将追踪失败的特征点匹配数量要求改为原先的 0.8。经过优化后算法明显的加快，并且由于初始化的要求降低，所以初始化更加容易，这里进行的参数调整将每帧的时间减少 0.5 秒。

3.1.3. 多线程优化

在算法中运行中，可以对算法中耗时较长的地方进行多线程的优化来提高算法的运行速度。具体方法是运行时加入算法时间的输出，着重优化算法耗时较长的部分。由于算法的主要耗时集中在循环内部，所以可以使用 OpenMP 并行加快算法的运行。主要进行多线程优化的部分集中在两个地方。

一是算法初始化寻找特征点匹配是通过 RANSAC (随机抽样一致性算法)方法计算照片之间的本征矩阵时的运算，这里简单描述一下计算步骤：1) 在当前已经匹配点的集合中，随机选取 8 对同名点计算这八个点之间的本征矩阵，计算出本征矩阵后，将匹配的其余点代入计算实际点与本征矩阵计算出的极线的距离，以距离作为误差值，获得一个对当前计算出的本征矩阵的总体误差；2) 进行 200 次随机选取，选取总体误差最小的本征矩阵作为图像的本征矩阵。这个部分只在算法初始化的时候用到，能够将每帧初始化的时间提升 0.1 秒。

二是在特征点提取的部分可以进行多线程的优化。这里简单描述下特征点提取的流程，首先将图像分成若干个网格，在每一网格中提取特征点，分区域进行特征树的构建，主要是为了能够在每个区域存在特征点，返回后计算每个特征的特征描述符，这里能够进行多线程优化的地方有两个：一是在网格中提取特征点，另外一个是在计算每个特征的特征描述符上，其他步骤不适合进行多线程的优化。在这里进行的多线程优化可以将每帧的时间减少 0.3 秒左右。

3.2. Android 的输入板块

SLAM 算法在 Android 上的输入主要有三个部分，一个是配置文件的读取，二是相机标定参数的获取，三是相机的输入。

3.2.1. 配置文件的输入

配置文件包括两个部分，一个是应用本身的配置使用，应用本身的设置使用 SharedPreferences 保存。另外的 SLAM 算法的配置和词典文件放到 Asserts 中，在第一次运行的时候将 Asserts 中的内容拷贝到 Android 应用的文件目录 files 文件夹下。这里的词典文件被用于 SLAM 的重定位和闭环检测的地方。由于词典文件是一个 143 M 大小的庞然大物，手机上读取词典文件一般要花费两分钟的时间。又因为词典可以在程序运行的时候自动从场景中提取，所以这里将词典文件设置为空词典，由之后的运行进行提取。

3.2.2. 相机标定参数的获取

根据 Computer Vision: Algorithms and Applications ([12])中提到的相机参数的计算，假设图像的宽高为 W，H，图像 x，y 方向上的视场角分别为 θ_x，θ_y，可以得到相机的标定参数为：

$$c_x = \frac{W}{2}, c_y = \frac{H}{2}, f_x = \frac{c_x}{\tan\frac{\theta_x}{2}}, f_y = \frac{c_y}{\tan\frac{\theta_y}{2}} \tag{1}$$

这样就可以使用图像宽高与相机 x，y 方向上的视场角获取到相机的标定参数。

3.2.3. 相机输入

相机输入使用 OpenCV 的依赖库实现，通过继承 OpenCV 的 JavaCameraView 的类实现。在相机初始化的时候设置摄像机的聚焦模式为定焦，设置帧率固定到[5, 15]的范围。它们的关系如图 1 所示。

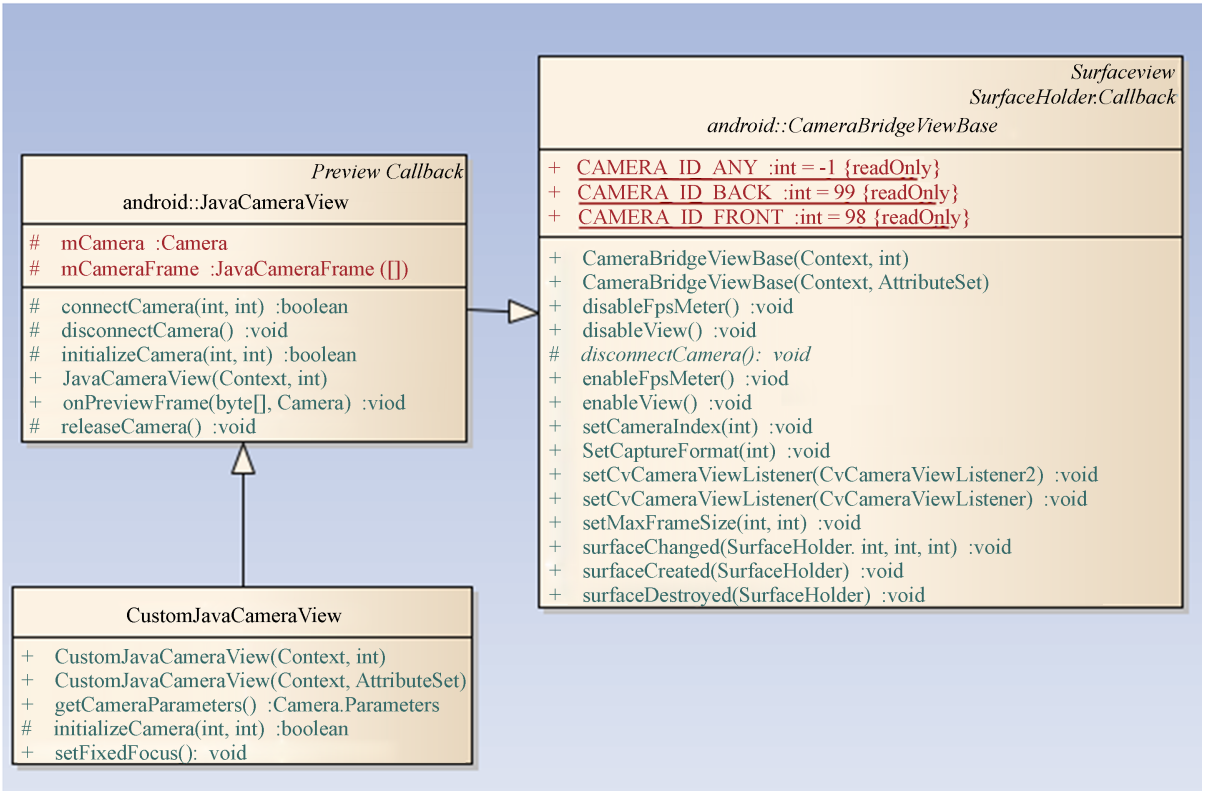

Figure 1. Inheritance structure of camera view
图 1. 相机 View 的继承结构

其中在 CameraBridgeBase 是一个抽象类，其中只完成了照相机的绘制屏幕工作，通过抽象方法调用子类的相机输入绘制当前相机图像。JavaCameraView 中继承 CameraBridgeBase 并且实现了自身的的相机输入，里面保持一个工作线程去一直不停读取相机输入保存到自身队列中，并且在绘制的时候调用接口的监听函数 onCameraFrame 将照片传送到注册的接口中。

在图 2 中，我们通过 SlamConnector 实现照片接受接口，在接受到照片时将图片传到原生层的算法部分，做为算法的输入，并且 SlamConnector 还负责算法的初始化和结束，还负责在项目当前的声明周期发生变化的时候，传送消息到 Native 层。其中 OrbSlamActivity 是当前算法的主要 Activity，在其中处理算法的输入输出。

3.3. Android 的输出部分

本项目的输出部分可以划分为两个模块，两个模块共同完成输出。第一个模块位于 Java 上，通过

SlamGLSurfaceView 对 GLSurfaceView 的继承，并且实现自己的 GLRender，在 GLRender 里的绘图函数 onDrawFrame 中调用 C++部分完成输入。第二个模块使用 C++代码完成。由于 Java 部分只是用来进行消息传递和一个绘制入口的作用，所以主要的输出在原生代码中。

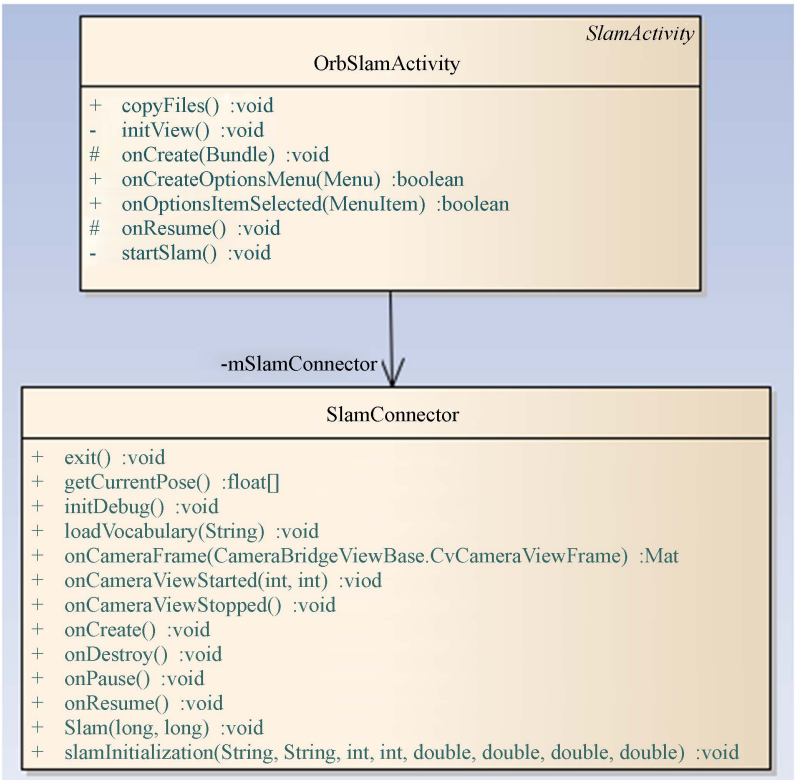

Figure 2. Acceptance of camera input
图 2. 相机输入的接受

4. 实验评估

4.1. 半稠密的实现效果

在电脑上测试了两个数据集与 ORB-SLAM 的半稠密部分的论文中提到的效果进行对比。数据集都是从网址 http://vision.in.tum.de/data/datasets/rgbd-dataset/download 中获得。第一个数据集 A 是一个接近平面的数据集，fr3/nostructure texture near with loop.，是在一个桌子上贴了几张报表。照相机绕着桌子上的几张报表转了一圈，场景时长是 56.49 秒，帧率是 30 帧每秒。第二个数据集 B 是一个桌面立体场景，名称是 fr2/desk，照相机绕着工作台转了一圈。场景时长为 99.36 秒，帧率是 30 帧每秒。

4.2. ORB-SLAM 在 Android 上的优化效果

经过 SLAM 算法在 Android 上的参数调整和对一些模块进行并行计算的修改后，在 Android 上的实现效果有了很大的提升。

如表 1 所示，在经过参数调整后，优化后的时间与先前的时间进行比较，可以看出经过参数调整后在 Android 上的运行效率有了明显的提升，而且定位精度的误差在可接受的范围内。

由表 2 可以看出在经过多线程优化后，算法在提取特征点所花费的时间有了显著的加快，并且关于多线程的优化没有更改算法的定位与制图结果，只加快了算法的运行时间。

Table 1. Running time comparison of SLAM algorithm before and after parameter adjustment
表 1. SLAM 算法在参数调整前后运行时间对比

ORB-SLAM 花费时间	调整前	调整后
提取特征点花费时间(ms)	432	420
追踪花费时间(ms)	580	41
局部制图花费时间(ms)	342	50
每帧花费时间(ms)	1134	513

Table 2. Running time comparison of SLAM algorithm before and after multithreading optimization
表 2. SLAM 算法在多线程优化前后运行时间对比

ORB-SLAM 花费时间	优化前	优化后
提取特征点花费时间(ms)	420	130
追踪花费时间(ms)	41	45
局部制图花费时间(ms)	50	52
每帧花费时间(ms)	513	175

4.3. 在 Android 上的输出效果

在 Android 上输出良好，图 3 与图 4 分别是在初始化未完成与完成后进行追踪时的运行的截图。上面的输出分别代表当前关键帧的数量，地图点的数量，与当前帧上匹配点的数量。

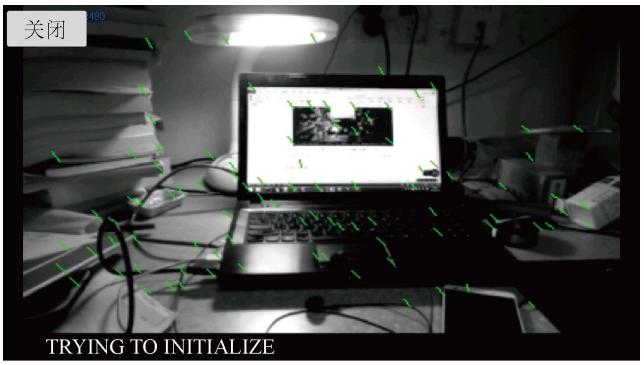

Figure 3. Running effect on Android (during initialization)
图 3. Android 上的运行效果(初始化时)

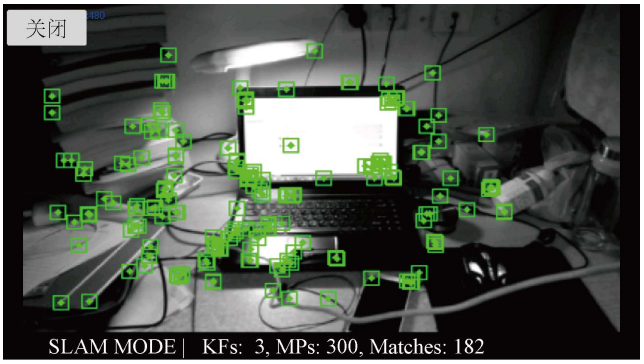

Figure 4. Output on Android (duringtracking)
图 4. Android 上的输出效果(追踪时)

图 5 是算法在 Android 手机上的运行时结果。由图 5 和图 6 来看算法在手机上运行能够达到所要的效果。但是在 Android 上运行时的速度很慢，每一帧在运行时会花费 4 秒左右的时间，还需要在手机上的进一步优化实现。

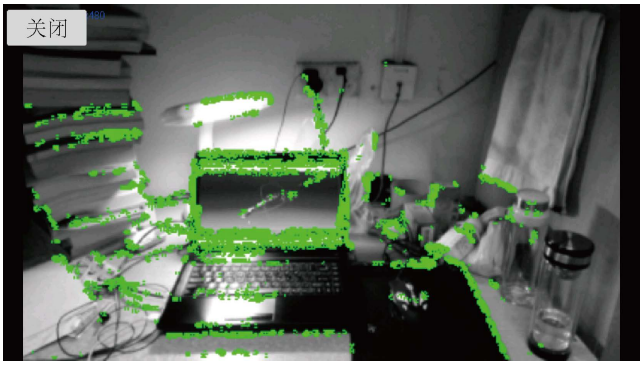

Figure 5. Output on Android (duringSemi-dense)
图 5. Android 上的输出效果(做半稠密时)

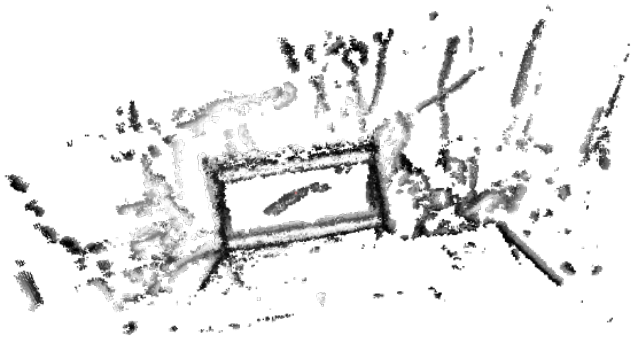

Figure 6. Semi-dense output on Android (when viewed on a computer)
图 6. Android 上半稠密的输出效果(在电脑上查看时)

5. 结束语

总体上来说，对于 ORB-SLAM 移植到 Android 上的优化是非常成功的，成功在智能手机上达到了实时运行需要的效率，在 Android 上的输入输出部分非常完善地实现了所需的效果但是基于 ORB-SLAM 半稠密的实现结果并不理想，在结果的精确与花费时间上均落后于 ORB 半稠密论文[11]中提到的效果。所以可以认为半稠密部分的算法还有很大的提升空间，还需要进一步的优化。

基金项目

黔科合重大专项字[2016]3012。

参考文献

[1] Smith, R., Self, M. and Cheeseman, P. (1988) Estimating Uncertain Spatial Relationships in Robotics. *Journal of Cardiovascular Electrophysiology*, **5**, 435-461.

[2] Mur-Artal, R., Montiel, J.M.M. and Tardos, J.D. (2015) ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *Computer Science*, **31**, 1147-1163. https://doi.org/10.1109/TRO.2015.2463671

[3] Galvez-López, D. and Tardos, J.D. (2012) Bags of Binary Words for Fast Place Recognition in Image Sequences.

IEEE Transactions on Robotics, **28**, 1188-1197. https://doi.org/10.1109/TRO.2012.2197158

[4] Strasdat, H., Montiel, J.M.M. and Davison, A.J. (2010) Scale Drift-Aware Large Scale Monocular SLAM. *Robotics*: *Science and Systems VI*, Universidad de Zaragoza, Zaragoza, Spain, 27-30 June 2010. https://doi.org/10.15607/RSS.2010.VI.010

[5] Strasdat, H., Davison, A.J., Montiel, J.M.M., *et al*. (2011) Double Window Optimisation for Constant Time Visual SLAM. *IEEE International Conference on Computer Vision*, Barcelona, 6-13 November 2011, 2352-2359. https://doi.org/10.1109/ICCV.2011.6126517

[6] Engel, J., Schöps, T. and Cremers, D. (2014) LSD-SLAM: Large-Scale Direct Monocular SLAM. *European Conference on Computer Vision*, **8690**, 834-849. https://doi.org/10.1007/978-3-319-10605-2_54

[7] Concha, A. and Civera, J. (2015) DPPTAM: Dense Piecewise Planar Tracking and Mapping from a Monocular Sequence. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Hamburg, 28 September-2 October 2015, 5686-5693. https://doi.org/10.1109/IROS.2015.7354184

[8] Engel, J., Sturm, J. and Cremers, D. (2013) Semi-Dense Visual Odometry for a Monocular Camera. *IEEE International Conference on Computer Vision*, Sydney, 1-8 December 2013, 1449-1456. https://doi.org/10.1109/ICCV.2013.183

[9] Mur-Artal, R. and Tardos, J.D. (2015) Probabilistic Semi-Dense Mapping from Highly Accurate Feature-Based Monocular SLAM. *Robotics*: *Science and Systems*, Rome, 13-17 July 2015. https://doi.org/10.15607/RSS.2015.XI.041

[10] Newcombe, R.A., Lovegrove, S.J. and Davison, A.J. (2010) DTAM: Dense Tracking and Mapping in Real-Time. *International Conference on Computer Vision*, Barcelona, 6-13 November 2011, 2320-2327. https://doi.org/10.1109/ICCV.2011.6126513

[11] Davison, A.J., Reid, I.D., Molton, N.D., *et al.* (2007) MonoSLAM: Real-Time Single Camera SLAM. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, **29**, 1052-1067. https://doi.org/10.1109/TPAMI.2007.1049

[12] Szeliski, R. (2015) Computer Vision: Algorithms and Applications. *Journal of Polymer Science Polymer Chemistry Edition*, **21**, 2601-2605.