文章编号：2096-1618(2019)02-0118-05

图优化 SLAM 的嵌入式处理技术

吴林峰， 王录涛
(成都信息工程大学计算机学院，四川 成都 610225)

摘要：基于图优化的同时定位与地图构建(simultaneous localization and mapping，SLAM)，可以让移动机器人在未知环境下构建周围的地图同时确定机器人在地图中的位置。复杂环境下其精确度和计算效率均优于基于滤波的SLAM，现已成为主流的方法。图优化SLAM用图来表示和解决SLAM问题，对计算资源要求很高，这极大限制了其在嵌入式系统上实时应用。对此提出了一种基于NVIDIA TX2的图优化SLAM处理技术，根据其架构特点优化SLAM算法，加速处理，实现图优化的实时处理。实验证明，通过高性能嵌入式处理硬件与算法优化的有机结合，可有效提升图优化SLAM在嵌入式系统中的处理性能。

关 键 词：同时定位与地图构建；图优化；GPU并行计算；嵌入式系统
中图分类号：TP391 **文献标志码**：A
doi：10.16836/j.cnki.jcuit.2019.02.003

0 引言

在自动导航机器人中，机器人自主定位是最困难的挑战之一。同时定位与地图构建(simultaneous localization and mapping，SLAM)技术能在未知的环境下增量构建机器人移动路径的地图同时估计机器人在地图中的位置[1]，为实现机器人完全自主导航提供有效解决路径，现已成为机器人领域的研究热点，其构建的地图由外部传感器(例如相机)所检测到的路标(landmark)这种特殊的元素组成。

基于图优化的SLAM结合图的思想，把SLAM问题用图表示[2-3]。图中顶点由机器人的位姿和路标来建模，边用连接顶点之间的非线性约束来建模。基于图优化的SLAM是一种平滑的方法，基于滤波的SLAM不考虑历史数据，随着机器人移动范围增大累积传感器的测量误差，最终生成不一致的地图。基于图优化的SLAM则考虑历史数据，对全局进行优化，来消除累积误差，因此是一种平滑的方法。除了估计地图还要估计机器人移动轨迹。由于平滑特性，基于图优化的SLAM在大型环境下需要大量的计算资源，因而限制了这种方法在嵌入式系统上实时应用。

文献[4]使用了光束平差法将PTAM(parallel tracking and mapping)移植到iPhone上，实现了用手持摄像头在很小的环境下进行定位。文献[5]设计了一个嵌入式视觉SLAM系统，运行在携带摄像头的扫地机器人上。

过去几年里，研究者们做出了很多工作来解决基于图优化SLAM计算巨大的问题。其中的一些工作利用了图结构的稀疏性来克服计算量的问题，文献[6]提出了一种稀疏位姿调整法，其实验取得了良好的收敛速度和精确度。文献[7]构建了一种最大加权生成树来表示稀疏图，利用其特殊结构和所设计的最优逼近算法，来构建一张最佳稀疏图用于图优化SLAM中，在数据集上有着良好的收敛速度。此外还有一些工作则着力于解决图的复杂度和存储空间的问题[8]，文献[9]提出了一种局部建图的算法，在大规模的SLAM下依然能保持良好的实时性。文献[10]使用深度摄像头(RGB-D)实现了稠密的视觉SLAM建图，然而测试这些算法都在桌面级的高功耗平台上进行，很少有工作针对嵌入式平台来解决基于图优化的SLAM问题。对此，使用两个互补的方法来克服计算复杂度问题，一是通过优化的方法来减小复杂度，二是在新的嵌入式系统上利用GPU并行计算加速处理。虽然已有一些工作使用GPU并行加速计算，如文献[11-12]在GPU上分别实现了光束平差法和ICP(iterative closest point)算法，但这些工作都是在桌面级的GPU上进行的。文中使用的嵌入式系统拥有特殊架构，其CPU和GPU有一块物理共享的内存，利用好这个特性可以减少不必要的数据传送时间，从而提高性能。

1 基于图优化的SLAM

图优化使用图来表示和解决SLAM问题，具有较高的精度和良好的实时性，是解决SLAM的主流方法之一。图是由顶点和边所组成的图形且用来描述事物之间的某些关系。引入到SLAM问题中，将机器人的位姿

收稿日期：2018-06-28
基金项目：四川省科技厅资助项目(2018GZ0184)

和路标表示为顶点,边表示连接顶点之间的非线性约束,通过不断向旧图中添加新的顶点和边构成新图结构,图优化的目标就是通过调整各顶点的位姿来最大可能的满足边之间的约束。基于图优化的 SLAM 从处理流程上可分为两块任务,前端和后端。前端对传感器捕获到的数据进行处理来构建图;后端又称为图优化,使用前端构建的图来估计机器人的轨迹和地图。

1.1　前端处理

在前端中,传感器将捕获的数据进行处理后用来构建和更新图结构。移动机器人携带传感器(相机、雷达等)在未知环境下移动并捕获图像数据,通过对这些数据进行特征提取、匹配进而通过视觉里程计[13]的方法估计移动机器人当前的位姿和离图像中所有路标的距离。其中位姿定义为某个时刻移动机器人的位置和转角(x,y,θ),路标定义为二维的位置(x,y)。随着机器人的移动,视觉里程计会持续地估计移动机器人当前的位姿和离图像中路标的距离。因此把新估计的位姿作为新的顶点加入图中并且和前一个顶点生成运动边,把对路标新的测量作为观测边加入到图中,如图 1 所示。

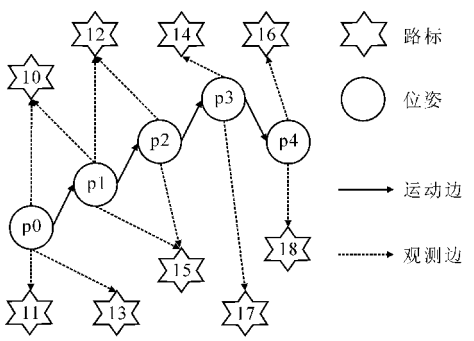

图 1　构建示意图

1.2　后端处理

在后端,一旦有新的节点和边加入图中便更新图结构,然后对新图进行优化。而对图进行优化,其实质是寻找最大可能满足边之间约束的图结构。在图优化之前主流的方法是采用概率来估计机器人最佳位姿,例如经典的 EKF[14]算法。然而随着场景的增大,这些方法都无法克服精度不高、计算量巨大这些问题[15]。

基于图优化的 SLAM 假定移动机器人所携带的传感器在测量时会受到高斯噪声的影响,现定义误差向量 $\boldsymbol{e}_{ij}$,它表示传感器对图中某条边连接的顶点 i、j 之间真实测量值 Z_{ij} 与计算预测值 $\hat{Z}_{ij}$ 之差。接着定义信息矩阵 $\boldsymbol{\Omega}_{ij}$,其每个元素值表示对误差项 $\boldsymbol{e}_{ij}$ 的权重,值越大表示跟真实值的误差越大,因而优化时给予较小权重。连接顶点 i、j 之间的约束 F 为:$F_{ij}=\boldsymbol{e}_{ij}^{\mathrm{T}}\boldsymbol{\Omega}_{ij}\boldsymbol{e}_{ij}$、$\boldsymbol{e}_{ij}=Z_{ij}-\hat{Z}_{ij}$。而图优化目的是寻找一个图结构 y^* 可以最小化代价函数 F:$F(y)=\sum_{ij}F_{ij}$。为获得 y^* 需要求解以下非线性最小二乘问题:

$$y^*=\underset{y}{\operatorname{argmin}}\,F(y) \tag{1}$$

对式(1)的求解,在 SLAM 中通常采用高斯牛顿或者列文伯格-马夸尔特等非线性优化的算法,从设定的初始值开始迭代寻找下降方向 Δ_y 来寻求目标函数的最优解。最终,最小化代价函数 F 将转化为线性方程组:

$$\boldsymbol{H}\Delta_y=-\boldsymbol{b} \tag{2}$$

式中 $\boldsymbol{b}$ 为信息向量,$\boldsymbol{H}$ 为信息矩阵。最后 y^* 可由求解出的 Δ_y 和初始值 y_0 计算求得:

$$y^*=y_0+\Delta_y \tag{3}$$

高斯牛顿或者列文伯格-马夸尔特算法迭代求解线性方程组式(2),然后更新并计算式(3)直到收敛。

然而随着场景的增大,系统所需要优化的位姿和路标也会越来越多。因此信息矩阵 $\boldsymbol{H}$ 的维度急剧增大,这使式(2)的规模变大,如直接对信息矩阵 $\boldsymbol{H}$ 求逆来解式(2)则复杂度为 $O(n^3)$,系统无法承受这样巨大的运算量。

2　GPU 并行优化

近些年来对 SLAM 研究的一个关键进展是发现了信息矩阵 $\boldsymbol{H}$ 的重要性质——稀疏性[16](机器人在某个位姿下只能测量到附近有限的路标并与它们产生约束),同时还发现这种稀疏性可以自然地用图优化来表示,这就使求解线性方程组变得可行。将稀疏矩阵 $\boldsymbol{H}$ 分块,并由式(4)表示待求解的线性方程组。

$$\begin{pmatrix}\boldsymbol{H}_{pp} & \boldsymbol{H}_{p1}\\ \boldsymbol{H}_{pl}^{\mathrm{T}} & \boldsymbol{H}_{11}\end{pmatrix}\begin{pmatrix}\Delta y_p\\ \Delta y_1\end{pmatrix}=\begin{pmatrix}-b_p\\ -b_1\end{pmatrix} \tag{4}$$

为减小图的规模,基于图优化的 SLAM 使用舒尔补从矩阵 $\boldsymbol{H}$ 中消去冗余的路标信息[17-18],其结果由式(5)表示,在求出 Δy_p 后可由式(6)对 Δy_1 进行求解:

$$(\boldsymbol{H}_{pp}-\boldsymbol{H}_{p1}\boldsymbol{H}_{11}^{-1}\boldsymbol{H}_{p1}^{\mathrm{T}})\Delta y_p=-b_p+\boldsymbol{H}_{p1}\boldsymbol{H}_{11}^{-1}b_1 \tag{5}$$

$$\boldsymbol{H}_{11}\Delta y_1=-b_1-\boldsymbol{H}_{p1}^{\mathrm{T}}\Delta y_p \tag{6}$$

新的线性方程组(5)、(6)求解速度更快,尤其是当图中路标数相较于移动机器人位姿数更多时可以减少很多计算量。

2.1　并行优化分析与设计

舒尔补主要进行稀疏矩阵乘法的操作,要想获得最好的性能,必须最大限度的利用线性方程组稀疏性的优势。稀疏矩阵相乘已有很多有效的实现方法[19],而近些年才提出在图优化 SLAM 中有效实现的方法[20]。它

利用了图优化 SLAM 中线性方程组的特殊结构,从而有效地进行舒尔补运算。然而在大部分方法中仍然包含着冗余的计算,它们没有提供一些先验信息来记录稀疏矩阵 $\boldsymbol{H}$ 的非零元素,因此每次都会使用矩阵 $\boldsymbol{H}$ 中所有的元素进行计算。随着机器人移动范围的扩大,前端所构建的图也会不断增大,矩阵 $\boldsymbol{H}$ 中元素同样也会增加,这显然产生了很多不必要的计算。

由上述的分析,提出一种结合矩阵 $\boldsymbol{H}$ 的稀疏性和 GPU 并行计算优化的方法进行加速,从而提高 SLAM 系统的运行效率,优化后的算法框架如图 2 所示。

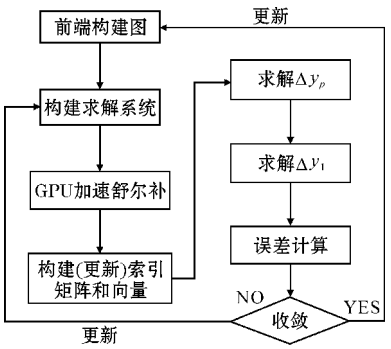

图 2 优化算法框架

在整个计算开始前先构建索引矩阵 $\boldsymbol{H}_{pp}^{*}$ 和 $\boldsymbol{H}_{p1}^{*}$ 用来指示矩阵 $\boldsymbol{H}$ 中的非零块,如果图中某个顶点或者某条边不存在,则将矩阵 $\boldsymbol{H}_{pp}^{*}$(或者 $\boldsymbol{H}_{p1}^{*}$)中对应索引用空值表示。矩阵 $\boldsymbol{H}_{pp}^{*}$ 表示位姿和运动边的非零块,$\boldsymbol{H}_{p1}^{*}$ 表示观测边的非零块。此外定义 $\boldsymbol{L}_p=(\boldsymbol{H}_{pp}-\boldsymbol{H}_{p1}\boldsymbol{H}_{11}^{-1}\boldsymbol{H}_{p1}^{\mathrm{T}})$,$R_p=-b_p+\boldsymbol{H}_{p1}\boldsymbol{H}_{11}^{-1}b_1$。首先计算 $\boldsymbol{H}_{11}^{-1}$,由于 $\boldsymbol{H}_{11}$ 是一个只包含路标变量的对角块矩阵且每个块互相独立,大小和路标变量的维度一样,因此可以高效地在 GPU 上对每块对角矩阵求逆。接下来计算矩阵 $\boldsymbol{L}_p$ 和向量 $\boldsymbol{R}_p$,同样它们之间互相独立,因此仍然可以在 GPU 上并行计算来提高速度。原本计算 $\boldsymbol{L}_p$ 需要消耗最多的时间,在引入索引矩阵 $\boldsymbol{H}_{pp}^{*}$ 和 $\boldsymbol{H}_{p1}^{*}$ 后只需要计算非零元素即可,这减少了很多不必要的计算。经过以上优化加速舒尔补这一过程后,在 GPU 上使用 Cholesky 分解法加速计算 Δy_p,求出 Δy_p 后按式(6)在 GPU 上做并行矩阵乘法运算来求解 Δy_1。

2.2 并行优化实现

算法在 TX2 上实现,它是 NVIDIA 在 2017 推出的一款性能强劲的多核移动 SOC,主要面向智能机器人、无人机、无人驾驶、智能摄像机和便携医疗设备等智能终端设备。其 CPU 一共拥有 6 个核心,包括 4 个 Cortex-A57、2 个定制的丹佛(Denver)核心。此外它还是异构的系统,其中集成了一块拥有 256 个 CUDA 核心、Pascal 架构的 GPU,其主要参数如表 1 所示。

表 1 TX2 主要性能参数

参数名称	参数指标
CPU	ARM Cortex-A57 (quad-core) 2 GHz + NVIDIA Denver2(dual-core) 2 GHz
GPU	256-core Pascal 1300 MHz
Memory	8GB 128-bit LPDDR4 1866 MHz
TDP	7.5 W

与传统构架相比,TX2 是 CPU-GPU 异构,并且集成了一块共享内存,这使 ARM-CPU 和 Pascal-GPU 可以从物理上共享这块内存。因此在 TX2 上 CPU 和 GPU 共享数据时不再需要把数据在 CPU 和 GPU 之间来回拷贝(zero copy)。有了 zero copy,在 TX2 上使用 CUDA(compute unified device architecture)进行 GPU 并行计算能显著提高性能。但是 zero copy 直接从共享内存中读取数据,不再从 GPU 缓存中读取数据,在反复读写同一块数据时会导致性能下降[21],因此仅把与图结构相关的数据(顶点和边的数量等等)保存在共享内存上以供 CPU、GPU 灵活地更新这些数据,而不用从 CPU 和 GPU 上反复来回拷贝,这减少了大量不必要的读写操作。

在 CPU 上主要执行前端的任务,移动机器人通过传感器将捕获的数据进行处理随后用来构建和更新图结构。同时还需为 CPU 申请一块私有内存空间,以供前端处理所用。后端的任务需要消耗大量计算资源且这些计算任务独立性较高,这便于 GPU 并行计算,因此 GPU 上主要执行后端任务。同样需要给 GPU 申请一块私有内存空间以供后端计算所用,例如存储待求解线性方程组等。

GPU 的显存有 2 种类型:全局内存和共享内存。前者容量大访问速度却慢,后者容量很小但是速度极快。因此,在 GPU 上计算有 2 种方法:一是数据的存取和计算均在全局内存上,二是使用共享内存计算,每次计算若干块数据然后保存结果至全局内存中,再将下一批数据放入共享内存中计算。然而共享内存为了获得高宽带,被设计为若干块并以 32 单位对齐。因此想要使用共享内存来达到最好的加速性能,这就需要机器人每个位姿的邻域内至少有 32 个路标与之关联从而产生约束,但实际上这个邻域相对较大且不能确保是机器人的移动路径。因此采用第一种计算方法,使用 5 个 CUDA 核函数来分别计算 $\boldsymbol{H}_{11}^{-1}$、L_p、R_p、Δy_p 和 Δy_1。

3 实验结果与分析

为了与文中优化后方法做对比,实验使用了

G2O[22](general graph optimization),一个开源的C++通用图优化框架。G2O 与文中方法类似,同样用图来表示并求解非线性最小二乘问题,在基于图优化的 SLAM 后端中广泛被使用。实验使用了 5 个常用的 2D 数据集,在每个数据集上分别执行 G2O、仅用 CPU 以及 GPU 加速这 3 种方法。每种方法均迭代 30 次并测量其平均执行时间,所有的实验均在 TX2 上运行,结果如表 2 所示。

表 2　平均执行时间比较　　ms

数据集	G2O	仅用 CPU	GPU 加速
Manhattan	68.21	51.73	24.63
Freiburg 079	12.45	9.83	4.66
Freiburg H	26.18	20.35	9.83
Victoria Park	925.37	659.38	295.17
CityTrees 10k	971.40	683.16	316.49

由实验结果可知,仅用 CPU 实现文中优化方法时比 G2O 平均快 25%。这是因为优化后的方法在首次计算前会构建索引矩阵和向量来记录矩阵 $\boldsymbol{H}$ 中非零块,在后续计算时便可利用这些信息只计算非零块,这减少了计算时间,而 G2O 则计算了全部子块。当然这得益于 TX2 上 CPU 和 GPU 有物理上共享内存的特殊架构,如果是普通的架构,在更新索引矩阵和向量时会消耗大量时间用于 CPU 和 GPU 之间频繁地来回传送数据。通过 G2O 和仅用 CPU 之间对比,这证明了在 TX2 上设置索引矩阵和向量减少了计算时间。使用 GPU 加速后比仅用 CPU 处理快了一倍,相比 G2O 快了 3 倍。

应当指出,尽管文中方法在速度上明显要快于 G2O,但是定位精度和 G2O 比较则非常接近,图 3 是文中方法估计的路径和真实路径之间的对比,图 4 是 G2O 估计的路径和真实路径之间的对比,图 5 是 G2O 和文中方法之间的位移误差对比,实验均在同一数据集下。可以看出,误差并没有随着位移距离的增加而累积变大,而是不断减小,这显示出了图优化平滑的性质。

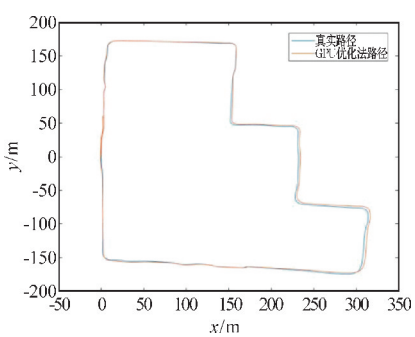

图 3　GPU 优化法路径误差对比

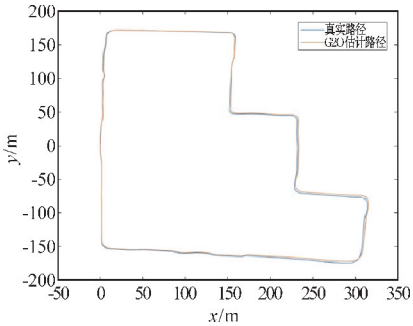

图 4　G2O 路径误差对比

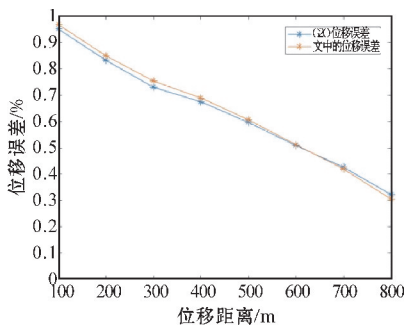

图 5　位移误差对比

基于图优化的 SLAM 在复杂环境下有着高精确度和计算效率,已成为 SLAM 中主流的方法。其缺点在于对计算资源需求高,因此限制了这种方法在嵌入式系统上实时的应用,目前大量的工作都在桌面级高功耗平台上进行。近年来嵌入式系统的性能一直在提升,甚至出现了集成 GPU 的异构系统。针对此问题提出了一种优化算法,首先利用 SLAM 中图结构的稀疏性,消除了对零元素块的不必要计算,通过实验验证,计算速度得到明显提升。其次在嵌入式异构系统(TX2)上使用 GPU 并行优化来加速计算,速度比未优化方法快 3 倍,实验证明在嵌入式系统上使用 GPU 能有效提高 SLAM 的速度。为在嵌入式异构系统上设计基于图优化的 SLAM 系统提供了一种优化思路。

参考文献:

[1] Durrant-Whyte H, Bailey T. Simultaneous Localization and Mapping: Part I[J]. IEEE Robotics & Amp Amp Automation Magazine, 2017, 13(2): 99–110.

[2] Grisetti G, Kummerle R, Stachniss C, et al. A Tutorial on Graph-Based SLAM[J]. IEEE Intelligent Transportation Systems Magazine, 2011, 2(4): 31–43.

[3] Carlevaris-Bianco N, Eustice R M. Generic factor-based node marginalization and edge sparsification for pose-graph SLAM[C]. IEEE International Conference on Robotics and Automation. IEEE, 2013: 5748–5755.

[4] Klein G, Murray D. Parallel Tracking and Mapping for Small AR Workspaces[C]. IEEE and ACM International Symposium on Mixed and Augmented

Reality. IEEE Computer Society, 2007: 1-10.
[5] Lee S. Embedded Visual SLAM: Applications for Low-Cost Consumer Robots[J]. IEEE Robotics & Automation Magazine, 2013, 20(4): 83-95.
[6] Konolige K, Grisetti G, Kümmerle R, et al. Efficient Sparse Pose Adjustment for 2D mapping[C]. Ieee/rsj International Conference on Intelligent Robots and Systems. IEEE, 2010: 22-29.
[7] Khosoussi K, Sukhatme G S, Huang S, et al. Designing Sparse Reliable Pose-Graph SLAM: A Graph-Theoretic Approach[J]. 2016.
[8] Kretzschmar H, Stachniss C. Information-theoretic compression of pose graphs for laser-based SLAM [M]. Sage Publications, Inc. 2012.
[9] Zhao L, Huang S, Dissanayake G. Linear SLAM: A linear solution to the feature-based and pose graph SLAM based on submap joining[C]. Ieee/rsj International Conference on Intelligent Robots and Systems. IEEE, 2013: 24-30.
[10] Whelan T, Leutenegger S, Moreno R S, et al. ElasticFusion: Dense SLAM Without A Pose Graph[C]. Robotics: Science and Systems. 2015.
[11] Wu C, Agarwal S, Curless B, et al. Multicore bundle adjustment[J]. 2016: 3057-3064.
[12] Rodriguez-Losada D, Segundo P S, Hernando M, et al. GPU-Mapping: Robotic Map Building with Graphical Multiprocessors[J]. IEEE Robotics & Automation Magazine, 2013, 20(2): 40-51.
[13] Fraundorfer F, Scaramuzza D. Visual Odometry: Part II: Matching, Robustness, Optimization, and Applications[J]. IEEE Robotics & Automation Magazine, 2012, 19(2): 78-90.
[14] Paz L M, Jensfelt P, Tardos J D, et al. EKF SLAM updates in O(n) with Divide and Conquer SLAM [C]. IEEE International Conference on Robotics and Automation. IEEE, 2007: 1657-1663.
[15] Huang S, Dissanayake G. Convergence and Consistency Analysis for Extended Kalman Filter Based SLAM[J]. IEEE Transactions on Robotics, 2007, 23(5): 1036-1049.
[16] Polok L, Ila V, Solony M, et al. Incremental Block Cholesky Factorization for Nonlinear Least Squares in Robotics[C]. Robotics: Science and Systems. 2013.
[17] Sibley G, Matthies L, Sukhatme G. Sliding window filter with application to planetary landing[J]. Journal of Field Robotics, 2010, 27(5): 587-608.
[18] Leutenegger S, Furgale P, Rabaud V, et al. Keyframe-Based Visual-Inertial SLAM using Nonlinear Optimization[C]. Robotics: Science and Systems. 2013: 789-795.
[19] Bell N. Implementing sparse matrix-vector multiplication on throughput-oriented processors[C]. Conference on High PERFORMANCE Computing Networking, Storage and Analysis. ACM, 2009: 18.
[20] Dine A, Elouardi A, Vincke B, et al. Graph-based SLAM embedded implementation on low-cost architectures: A practical approach[C]. IEEE International Conference on Robotics and Automation. IEEE, 2015: 4612-4619.
[21] Otterness N, Yang M, Rust S, et al. An Evaluation of the NVIDIA TX1 for Supporting Real-Time Computer-Vision Workloads[C]. Real-Time and Embedded Technology and Applications Symposium. IEEE, 2017: 353-364.
[22] Kümmerle R, Grisetti G, Strasdat H, et al. G2O: A general framework for graph optimization[C]. IEEE International Conference on Robotics and Automation. IEEE, 2011: 3607-3613.

Graph-based SLAM Embedded Processing Technology

WU Linfeng, WANG Lutao
(College of Computer Science, Chengdu University of Information Technology, Chengdu 610225, China)

Abstract: Graph-based simultaneous localization and mapping allows the mobile robot to construct the surrounding map in an unknown environment while determining the robot's position in the map. The accuracy and computational efficiency of Graph-based SLAM are better than those of the filter-based SLAM in complex environment, which has become the mainstream method. Graph-based SLAM uses the graph to represent and solve the SLAM problem, which requires high computational resources and greatly limits its application in real-time on embedded systems. This paper proposes an optimized SLAM processing technology based on NVIDIA TX2 and optimizes the SLAM algorithm according to its architecture characteristics to speed up processing and achieve real-time processing of graph optimization. The experiments have proved that the organic combination of high-performance embedded processing hardware and algorithm optimization can effectively improve the processing performance of Graph-based SLAM in embedded systems.

Keywords: simultaneous localization and mapping; graph optimization; GPU parallel computing; embedded systems