

目录	1
----	---

目录

1 背景介绍	3
2 文献调研	4
2.1 手动标定方法	4
2.2 自动标定方法	4
2.2.1 依赖标志物的标定方法	4
2.2.2 无标志物的标定方法	5
3 基于深度互信息的标定算法	6
3.1 互信息	6
3.2 使用深度互信息进行标定	6
3.3 优化方法	9
3.3.1 梯度方法	9
3.3.2 网格搜索方法	11
4 实验	12
4.1 损失函数在最优值附近的性态	12
4.2 初始值对算法的影响	13
4.3 损失函数对平移参数的敏感度	14
5 总结	15
6 附录一：工程实现	17
6.1 硬件环境	17
6.1.1 网络连接	17
6.1.2 机械连接	18
6.2 软件环境	20
6.2.1 软件框架	20
6.2.2 传感器驱动	22

目录	2
----	---

6.2.3 传感器之间的同步	22
6.2.4 传感器配置	22
6.3 使用 Ceres Solver 求解问题	24
6.3.1 数值方法	24
6.3.2 解析方法	24
6.4 网格搜索方法	25

1 背景介绍

传感器是机器人系统的重要组成部分。近年来，机器人领域取得了很大的发展，越来越复杂的环境需求不断推动传感器技术的发展。如今的机器人系统的传感器子系统往往由多种传感器组合而成。特别是在无人驾驶中，汽车需要有多种传感器来同时对环境进行感知，从而得到上层决策系统所需要的信息。无人驾驶中常见的传感器包括激光雷达、毫米波雷达、相机、卫星定位系统、惯导系统等。而相机和激光雷达是其中最为重要的两种。这两种传感器优势互补：相机的数据是稠密的，但是却难以得到准确的深度信息；激光雷达的数据是稀疏的，但是可以提供非常准确的距离信息。为了融合这两种传感器的信息，得到更加丰富、准确的环境信息，需要求得两种传感器坐标系之间的变换—即外参，外参的求解过程称为外参标定。

本文实验中使用的激光雷达是 Velodyne 公司生产的 VLP-16 激光雷达。这是一款具有十六对激光收发器的激光雷达，每次测量时发射 16 条成固定角度的激光光线，得到每条光线上最近的障碍物距离。因为此种雷达具有多条扫描线，可以实现三维空间的扫描，被称之为 3D 激光雷达。与之相对的是 2D 激光雷达，这类激光雷达只有一条扫描线，扫描范围是一个平面。本文主要研究的是 3D 激光雷达与相机之间的外参数求解方法。

目前，解决此问题的最成熟的方法是基于特殊标志物的标定算法，代表文献有 [4]。此类方法需要借助特殊标志物如棋盘格等，操作不便。与此类方法相对，还有一类不依赖特殊标志物的标定算法，代表文献有 [9]。本文提出了一种全新的、完全自动的、无标志物的标定方法，此方法基于图像深度与激光雷达深度互信息，结果相比同种方法更稳定准确。

本文2节将回顾现有的标定方法。3节对核心算法进行了描述。4节对所述算法进行了实验分析，证明了其有效性。5节给出了一个简要的总结。具体的工程实现细节在附录一中给出。

2 文献调研

2.1 手动标定方法

与相机之间的外参标定不同，相机与激光雷达之间的标定涉及到两种不同的数据模态。相机之间的外参标定中常常需要依赖匹配点，即同一个空间点在两个相机中的像素位置。这种匹配关系常常通过梯度方法 [5]、特征描述法 [13] 等方法得到。在激光雷达与相机的标定问题中，两种数据模态有很大差别，无法使用传统方法找到匹配关系。所以在此问题研究之初，大多算法都依赖于人工标记的匹配关系 [14][16][6]。在得到了多组匹配关系之后，外参数解就转变成了一个常见的优化问题。

2.2 自动标定方法

2.2.1 依赖标志物的标定方法

手动标定方法虽然在一定程度上解决了外参标定问题，但是其操作繁琐，标定结果受到很大的人为干扰。在相机之间的外参标定问题中，人们常常使用棋盘格等特殊标志物来帮助寻找匹配关系。棋盘格方法是经典的相机内参标定方法 [15]，并被广泛地用于多目相机的外参标定中。在相机与激光雷达的标定问题中，基于特殊标志物的自动标定算法也得到了研究。[4] 中介绍的算法使用多个棋盘

格实现了相机与 3D 激光雷达之间的外参标定：首先对多个相机组成的阵列进行标定，标定过程中得到棋盘格格点的坐标；然后通过法向量寻找点云中的平面来找到棋盘；最后使用迭代方法优化外参。此方法更加繁琐，需要在一个空房间的四周墙壁上放置多个棋盘格，而且相机系统必须由多个相机组成，有一定局限性。

2.2.2 无标志物的标定方法

为了解决上述算法的问题，我们需要一种全自动的、不依赖特殊标志物的标定算法。此问题也得到了广泛的研究。[\[4\]](#) 中描述的方法首先将标定问题看作一个尺度不确定的手眼标定问题，通过传统算法求解手眼标定问题得到初始解；然后通过 2D 与 3D 空间中的直线匹配误差将问题转化为优化问题，求得精度更高的结果。本文所述的方法受到 [\[9\]](#) 的启发，此文献中描述的方法以灰度图像和激光雷达测得的反射强度之间的互信息（下文称之为强度互信息）作为误差标准，将标定问题转化为一个标准的优化问题。由于不依赖特征匹配，所以此方法在准确度和稳定性上都有提高。但是此方法对光照情况敏感，容易受到阴影的影响，在实际使用中常常得不到很好的效果。本文所述的方法借鉴了互信息作为误差标准的想法，但是不再以灰度和反射度之间的互信息作为误差标准，而是使用视觉 SLAM (simultaneous localization and mapping) 算法输出的半稠密深度图与激光雷达测得的深度之间的互信息（下文称之为深度互信息）作为误差标准。相比于强度互信息，深度互信息方法更加稳定，更加适用于室外强光与阴影共存的场景。

3 基于深度互信息的标定算法

3.1 互信息

随机变量 X, Y 之间的相关性可以用互信息来度量。互信息有多种不同的定义，本文中使用的互信息定义为：

$$MI(X, Y) = H(X) + H(Y) - H(X, Y) \quad (1)$$

其中 $H(X)$ 表示随机变量 X 的熵， $H(X, Y)$ 表示 X, Y 联合分布的熵。当 X, Y 是离散随机变量时，熵定义如下：

$$H(X) = \sum_{x \in X} p_X(x) \log \frac{1}{p_X(x)}$$

$$H(Y) = \sum_{y \in Y} p_Y(y) \log \frac{1}{p_Y(y)}$$

$$H(X, Y) = \sum_{x \in X} \sum_{y \in Y} p_{XY}(x, y) \log \frac{1}{p_{XY}(x, y)}$$

熵表示的是一个概率分布的不确定性，于是可以知道 X, Y 之间的互信息即 X, Y 联合分布的不确定性相比于两个随机变量边缘分布不确定性之和的减少量。

3.2 使用深度互信息进行标定

相机输出的图像经过视觉 SLAM (V-SLAM) 算法的处理之后可以产生一个半稠密的深度图，即在图像上纹理较为丰富的区域，每个像素点都会有一个深度的估计值。视觉 SLAM 算法输出的半稠密深度图见图1。

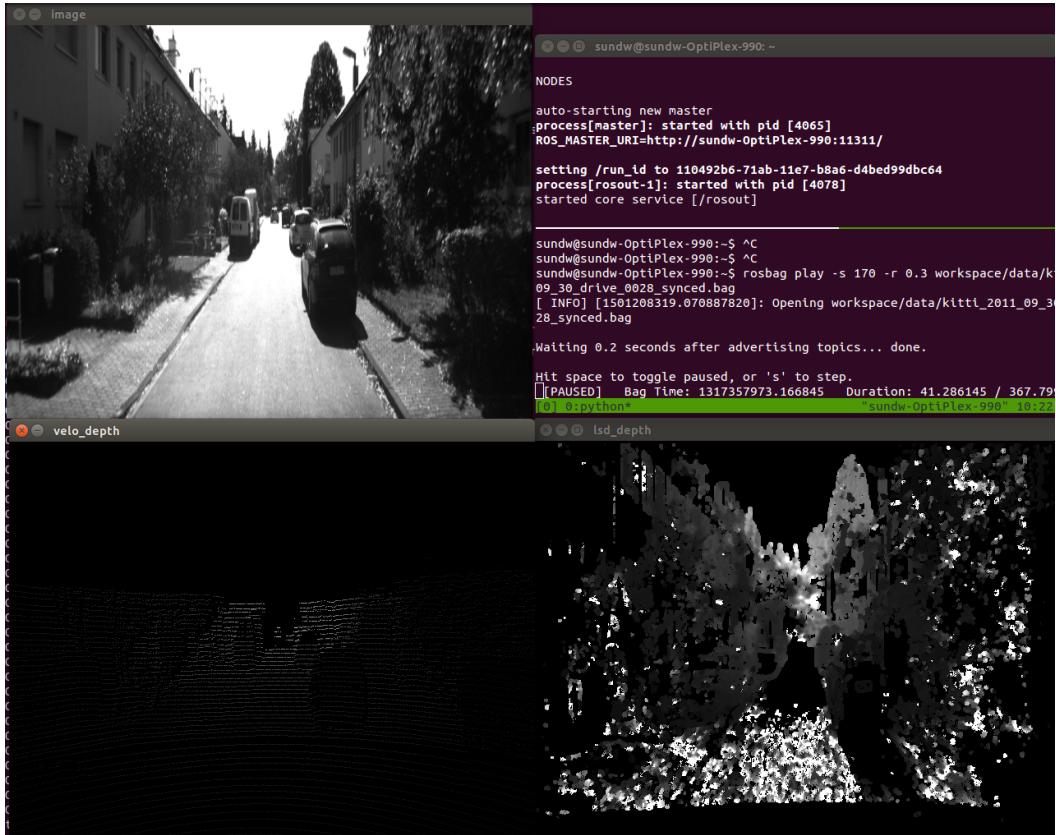


图 1: 不同模态的信息: 每张图片左上角为相机灰度图, 左下角为激光雷达投影到图像平面后得到的稀疏深度图, 右下角为 V-SLAM 算法得到的半稠密深度图。实验数据来自于 KITTI 数据集 [3], 使用的 V-SLAM 程序为 LSD-SLAM[2]

激光雷达输出的数据为一个点云，其中每一个点的坐标都是在激光雷达坐标系中测量得到的。假设从激光雷达坐标系到相机坐标系的变换矩阵为 $T = [R | t]$ ，其中 $R \in SO(3)$, $t \in R^3$ 。 T 是一个六自由度的矩阵，亦即本文算法要求解的外参。此变换可以有多种不同的表示，直观起见，本文选用平移 + 欧拉角的表示方法。欧拉角选取的是 $sxyz$ 格式¹。于是有

$$\Theta = (t_x, t_y, t_z, r_x, r_y, r_z)^T \quad (2)$$

其中 t_x, t_y, t_z 表示位移， r_x, r_y, r_z 表示欧拉角， Θ 表示这一组参数。

本文中使用的相机模型为经典的针孔模型，且假设图像畸变已经得到矫正。于是从激光雷达坐标系到图像坐标系的转换为：

$$P_c^i = d * K * T * P_l^i \quad (3)$$

其中 K 是相机内参矩阵， P_l^i 为点在激光雷达坐标系中的齐次坐标，角标 i 表示点序号， P_c^i 是三维点投影到图像平面的齐次坐标， d 为深度。

使用式(3)将激光雷达点云映射到图像平面中，滤除那些投影后的像不在图像中的点和像位置没有深度（因为 SLAM 输出的深度图是稀疏的，部分像素处没有深度）的点，将过滤之后的点表示为集合 $P_l = \{P_l^i, i = 1, 2, 3, \dots, N\}$ 。集合 P_l 中的每一个点经过式(3)变换之后都落在图像中，且对应一个深度估计值，同时也对应一个深度测量值（即式(3)中的 d ）。将这些深度信息看作是随机向量 $X = (X_1, X_2)^T$ 的一组采样，其中 X_1 表示 SLAM 估计深度， X_2 表示激光雷

¹https://en.wikipedia.org/wiki/Euler_angles

达测量得到的深度。由上面的描述可知 $I(X) = MI(X_1, X_2)$ 是外参的函数，即 $I(X | \Theta)$ 。在下面的叙述中我们假设在外参优化过程中集合 P_l 保持不变。这个假设并不符合实际，但是在每一步优化中，参数变化量极小，这个假设近似正确。在具体实现中我们使用核函数密度估计法 [12][10] 估计 X 的边缘分布和联合分布。通过最大化 $I(X | \Theta)$ 来实现两种传感器的配准，从而得到外参，即

$$\Theta^* = \operatorname{argmax}_{\Theta} I(X | \Theta)$$

在实际实验中，我们使用多帧数据求解参数，同时考虑到互信息的非负特性和现有最小二乘优化框架的特性，实验中实际求解的是下面的问题

$$\Theta^* = \operatorname{argmin}_{\Theta} \sum_{i=1}^m \frac{1}{(I(X^i | \Theta))^2} \quad (4)$$

其中 m 表示所用的帧数

3.3 优化方法

为了求解式(4)所表示的优化问题，我们尝试了几种不同的优化方法，包括梯度方法和网格搜索方法。

3.3.1 梯度方法

基于梯度的方法是求解优化问题最常用的方法。下面首先推导互信息对参数 Θ 的梯度。

首先介绍互信息的微分法则，见定理3.1，此定理的证明过程见 [1]。

定理 3.1

$$I(X + \Delta) - I(X) = E \{ \Delta^T \beta_X(X) \} + o(\Delta)$$

其中 Δ 是一个微小扰动随机变量, $\beta_X(X)$ 是得分函数差

其中 $\beta_X(X)$ 是对随机变量分布的一种度量, 表示边缘得分函数与联合得分函数的差, 具体定义见 [1]。从定理3.1可以得到直接推论3.1.1, 此推论表示的是互信息的链式求导法则。

推论 3.1.1

$$\begin{aligned} & I[X(\xi + \Delta)] - I[X(\xi)] \\ &= I[X + \frac{\partial X}{\partial \xi^T} \Delta] - I[X(\xi)] \\ &= E \left\{ \left(\frac{\partial X}{\partial \xi^T} \Delta \right)^T \beta_X(X) \right\} + o(\cdot) \\ &= E \left\{ \Delta^T \frac{\partial X^T}{\partial \xi} \beta_X(X) \right\} + o(\cdot) \end{aligned}$$

在我们的问题中, 互信息对参数的梯度可使用上述推论求得。将 SLAM 算法输出的深度图表示为 $D(\cdot, \cdot)$, $D(u, v)$ 表示对应坐标处的深度估计。为了方便叙述, 我们使用下面定义的变换函数 w

$$(u, v, d)^T = w(x, y, z)$$

其中 x, y, z 表示雷达坐标系中的点坐标, u, v 表示图像坐标系中的坐标, d 表示

点投影到图像坐标系后的深度。那么对于第 i 个点，有

$$X_1^i = D(u^i, v^i), X_2^i = d^i$$

于是有

$$\frac{\partial I(X)}{\partial \Theta} = E \left\{ \frac{\partial X^T}{\partial \Theta} \beta_X(X) \right\}$$

$$\frac{\partial X^T}{\partial \Theta} = \left[\frac{\partial x_1}{\partial \Theta}, \frac{\partial x_2}{\partial \Theta} \right]$$

$$\frac{\partial X_1}{\partial \Theta} = \left[\frac{\partial u}{\partial \Theta}, \frac{\partial v}{\partial \Theta} \right] \cdot \left[\frac{\partial D}{\partial u}, \frac{\partial D}{\partial v} \right]^T$$

$$\frac{\partial X_2}{\partial \Theta} = \frac{\partial z}{\partial \Theta}$$

其中 $\frac{\partial u}{\partial \Theta}, \frac{\partial v}{\partial \Theta}, \frac{\partial z}{\partial \Theta}$ 的求解是非常简单的， $\frac{\partial D}{\partial u}, \frac{\partial D}{\partial v}$ 是深度图的梯度，具体求解细节参照附录一。

3.3.2 网格搜索方法

实验中发现，由于深度图存在较大的噪声以及图像梯度存在局部性，目标函数并不具有凸性。使用基于梯度的方法进行优化时，会有很大的概率无法达到全局最优解。为了解决梯度局部性带来的问题，我们实验了一种网格搜索算法。即在六维参数空间中以固定步长划分网格，计算每个格点上的参数对应的目标函数值，取最小的目标函数值对应的参数作为最优解。但是此方法的问题在于计算量太大，在六维的参数空间中每个维度的网格数为 N ，则最终需要计算目标函数的次数为 $O(N^6)$ 。而 N 的大小决定了结果的精度，实验中需要的 N 往往在 1000 量级。这意味着需要计算目标函数的次数为 10^{18} 量级，这是不可接受的。

为了减少计算量，我们借鉴了二分查找的思想，逐步提高结果精度。首先在一个较大步长的网格上进行搜索，确定最优格点之后再以此格点为中心，在步长减半的网格上进行搜索，如此迭代。迭代次数视要求的精度而定。此算法的复杂度为 $O((\log N)^6)$ 。

此方法在很大程度上避免了陷入局部最优中，实验中也发现此方法基本可以保证找到最优解，实验结果见4节。

4 实验

4.1 损失函数在最优点附近的性态

为了对优化问题有具体的了解，我们测试了损失函数在最优点附近的性态，见图2。从图中可以看到，对于要优化的参数来说，损失函数在小范围内具有单谷特性。虽然图2只展示了一个参数变化时的情况，我们有理由相信当六个参数同时变化时损失函数基本具有单谷特性。同时可以看出，由于噪声的存在，曲线并不平滑，这导致了导数波动较大，具有明显的局部性，不利于基于梯度的优化方法。在实验中我们也发现，由于目标函数非凸，基于梯度的方法往往不能求得全局最优解，这也促使我们改用网格搜索方法求解此问题。图2说明了损失函数在参数等于真值的时候取得最小值，这说明式(4)所示问题的最优解是参数的真实值。求解外参等价于求解式(4)所示问题。

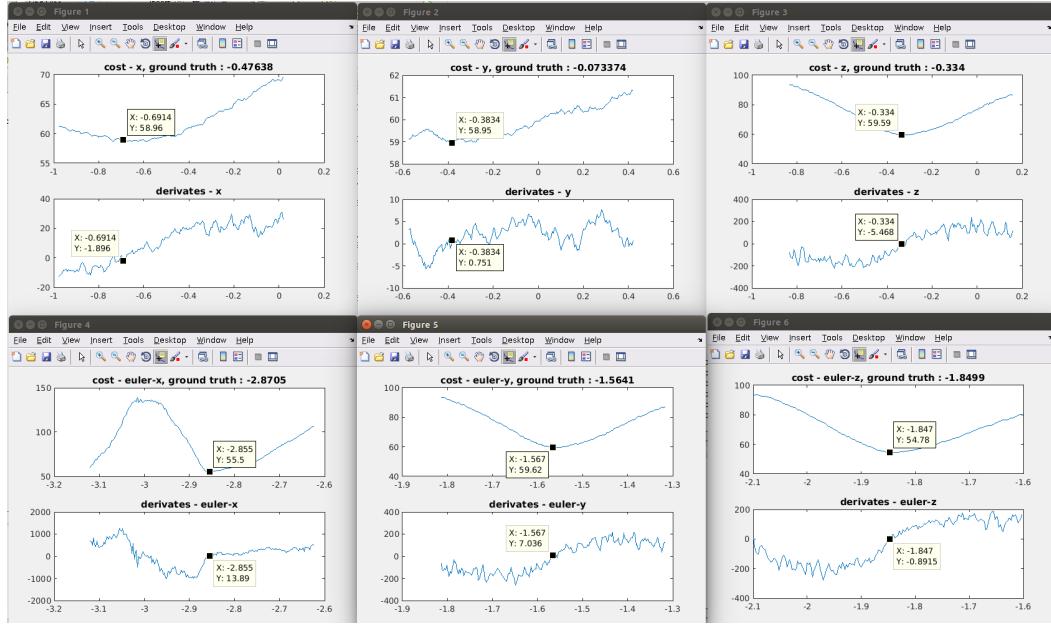


图 2: 损失函数在最优值附近的性态, 图中展示了当某个参数在真值附近变化时, 损失函数的变化以及导数 (使用人工选定的步长, 通过数值方法求得) 的变化。每条曲线中只有被测试的平移参数变化, 其他五个参数均固定在真值处。注: 此组曲线在 KITTI 数据集 [3] 上测得

4.2 初始值对算法的影响

参数的初始设定值会影响算法最终的结果, 实验中发现当初始值偏离真实值太远时, 算法将不能得到正确结果。为了探究此算法能在何种初始值配置下工作, 我们使用不同的初始旋转参数, 进行了如下实验。首先是单个参数的优化实验, 其他五个参数全部固定在真实值处, 不参与优化, 结果如表1。下列所有实验的参数真实值都是 $\Theta = (0.0, -0.0583, -0.015, 1.57, -1.34, 0.0)$

表 1: 单个参数的优化实验 (表示为 “-” 的参数固定在真实值处不参与优化)

/	t_x	t_y	t_z	r_x	r_y	r_z
initial	-	-	-	1.37	-	-
result	-	-	-	1.6225	-	-
initial	-	-	-	-	-1.55	-
result	-	-	-	-	-1.35	-
initial	-	-	-	-	-	0.2
result	-	-	-	-	-	-0.0175

多个参数的优化实验如表2。

表 2: 多个参数的优化实验

/	t_x	t_y	t_z	r_x	r_y	r_z
initial	-	-	-	1.37	-1.55	-
result	-	-	-	1.6125	-1.3525	-
initial	-	-	-	1.37	-1.55	0.2
result	-	-	-	1.6	-1.36	-0.035
initial	-	-	-	1.47	-1.45	0.2
result	-	-	-	1.605	-1.36	-0.035

表 3: 所有参数的优化实验

/	t_x	t_y	t_z	r_x	r_y	r_z
initial	0	0	0	1.37	-1.55	0.2
result	-0.34375	0.0104167	0.385417	1.60958	-1.28958	-0.0395833

所有参数都参与优化的实验结果见表3。实验表明，算法对旋转参数敏感，优化精度较高。当旋转参数与真实值相差 12° 时，仍能正常工作。加入平移参数之后，由于参数之间有一定的耦合，最终结果变差，但是仍有较高精度。

在三个旋转参数参与优化的实验中，初始参数和最终结果对应的配准结果如图3所示。

4.3 损失函数对平移参数的敏感度

在实验中我们发现，算法对于旋转参数的敏感度非常高，结果较为准确。但是算法对平移参数的敏感度较差，求解误差较大。所以在这一节中我们通过实验探究了损失函数对于平移参数的敏感度，发现了问题的原因。图4展示了在平移参数上加小扰动之后，配准图像的变化。从图中可知，平移参数 x 对应的是激光雷达水平扫描方向，在此方向上点云很稠密，所以损失函数对此参数的敏感度最高，见图5。平移参数 y 对应的是激光雷达竖直方向，由于激光雷达只有

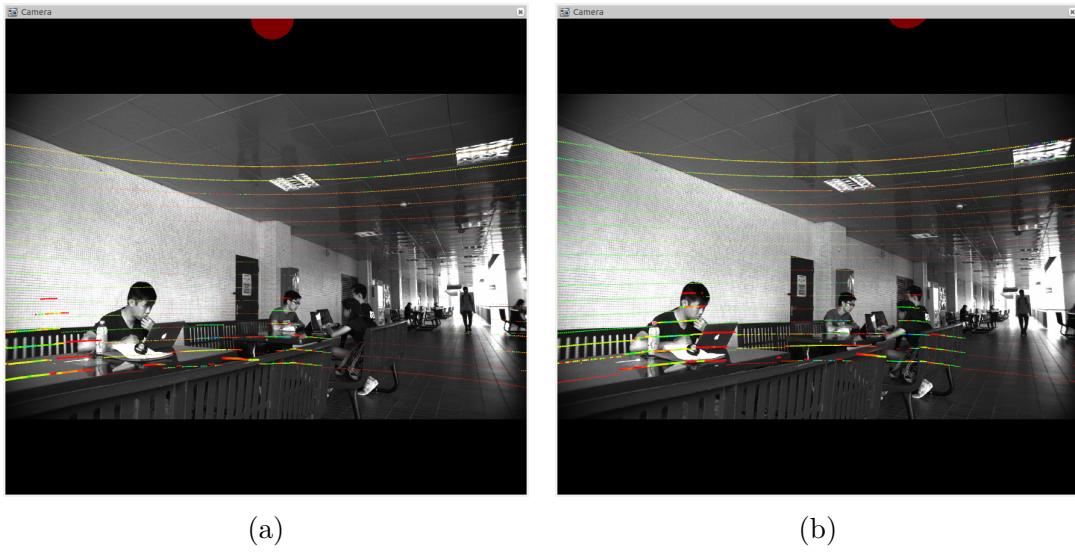


图 3: (a) 初始参数对应的配准结果, 此时参数为 $(-, -, -, 1.37, -1.55, 0.2)$, (b) 算法给出的参数对应的配准结果, 此时参数为 $(-, -, -, 1.6, -1.36, -0.035)$ 。参数真实值为 $(-, -, -, 1.57, -1.34, 0.0)$

16 条激光光线, 所以此方向上的点云较为稀疏, 损失函数的敏感度也较差。平移参数 z 对应的是垂直于图像平面的方向, 由于实验配置中激光雷达和相机距离非常近 (见附录一), 所以此方向上移动激光雷达时, 对配准图像的影响很小, 从而导致损失函数对平移参数 z 呈现奇怪的变化趋势。

5 总结

我们提出了一种全新的基于互信息的标定方法, 此方法具有很好的稳定性, 适用于室内外环境。此方法完全自动, 不需要特殊的标志物即可工作。此方法可以使用手眼标定算法的结果作为初始值, 优化得到更加准确的值。本文使用实验结果说明了算法的有效性。

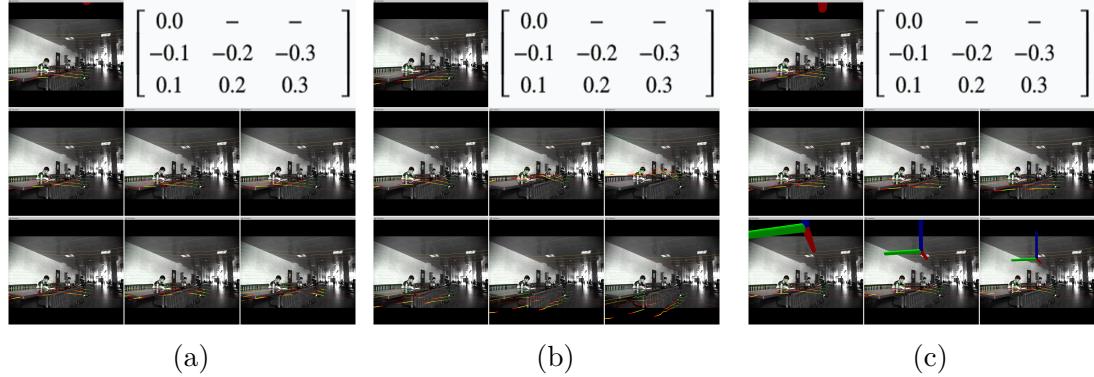


图 4: 图像配准对于平移参数的敏感性, 图中可视化了当某个参数变化时得到的配准图像: (a) 参数 x 变化时的配准图像, (b) 参数 y 变化时的配准图像, (c) 参数 z 变化时的配准图像。右上角的数字表示对应位置处图像所使用的参数与真值的差, 每张图中只有被测试的平移参数变化, 其他五个参数均固定在真值处。
注: 此图在我们自己采集的数据集上测得, 实验配置见附录一

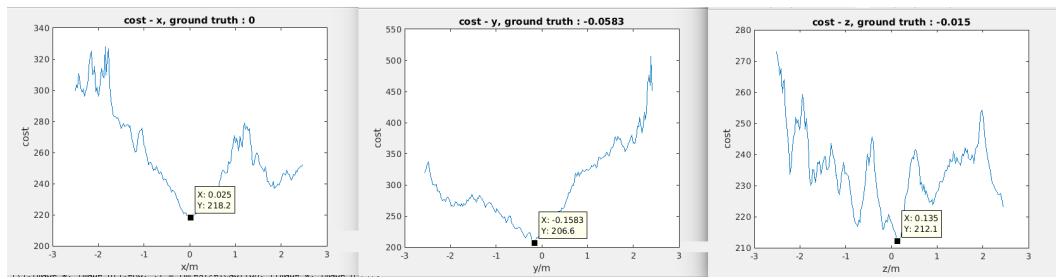


图 5: 损失函数对平移参数的敏感性, 图中展示了当某个平移参数在真值附近变化时, 损失函数的变化。每条曲线中只有被测试的平移参数变化, 其他五个参数均固定在真值处。
注: 此组曲线在我们自己采集的数据集上测得, 实验配置见附录一

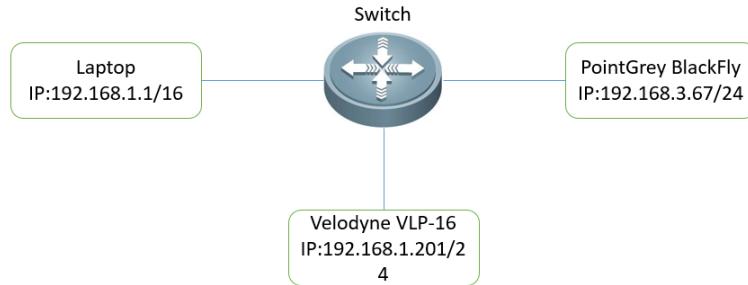


图 6: 传感器设备与数据采集电脑的网络连接

6 附录一：工程实现

6.1 硬件环境

6.1.1 网络连接

实验中为了验证算法的有效性，我们在 KITTI 数据集上进行了测试。除此之外，我们使用一台 VLP-16²激光雷达和一台 PointGrey BlackFly³工业相机采集了新的数据，并对此系统进行了标定。此系统中的两种传感器都使用以太网进行通信，我们使用图6所示的网络拓扑结构将它们与数据采集电脑连接在一起。

²<http://velodynelidar.com/vlp-16.html>

³<https://www.ptgrey.com/blackfly-usb3-vision-cameras>



图 7: 激光雷达与相机之间的机械连接

6.1.2 机械连接

两种传感器中都有标准螺纹孔，为了方便得到参数真实值，我们直接将两种传感器连接在了一起，连接方法如图7所示。两种传感器的坐标系关系如图8。

从图7中可知，两种传感器的下表面完全重合，且相机光轴通过激光雷达下表面中心，所以可以确定部分外参如下

$$t_x = 0, r_x = -\frac{\pi}{2}, r_z = 0$$

为了得到 t_y, t_z ，我们使用直尺进行测量，两种传感器的机械参数如图9和图10所示，从图中参数和测量结果可以计算得到 $t_y = -0.0583, t_z = -0.015$ 。



图 8: 激光雷达与相机坐标系之间的关系

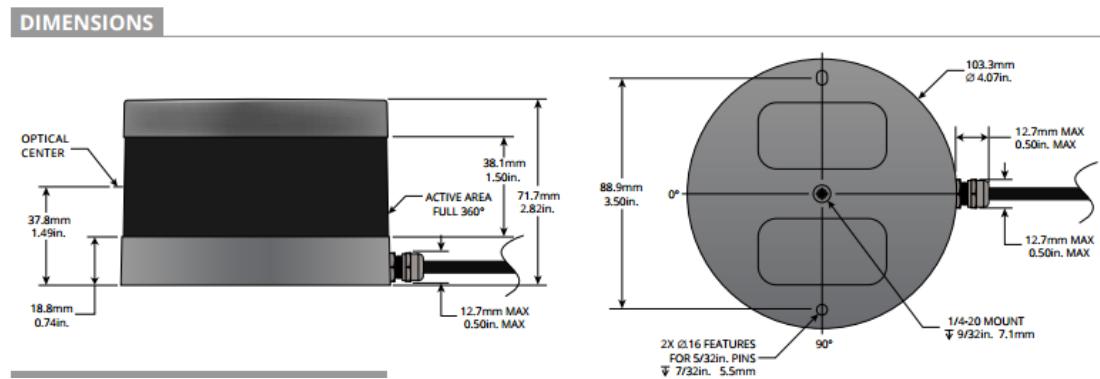


图 9: 激光雷达机械参数

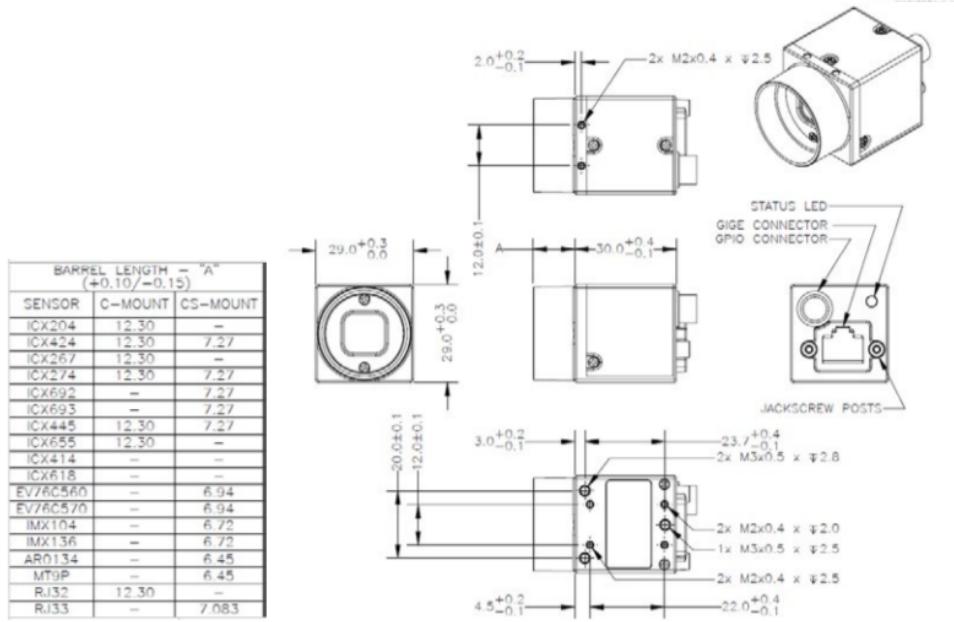


图 10: 相机机械参数

为了得到 r_y , 我们开发了一个工具, 进行手动搜索, 人工确定最佳配准位置, 最终得到 $r_y = -1.34$ 。

6.2 软件环境

6.2.1 软件框架

本文中的实验软件环境是 ROS(Robert Operating System)[11], 选用 ROS 的原因有二: 本文使用的 V-SLAM 算法框架为 LSD-SLAM[2], 此框架使用 ROS; ROS 提供了丰富的进程间通讯方法, 非常适合复杂机器人系统、多传感器系统的构建。

ROS 中使用 node (节点) 表示一个运行中的进程, 一般一个 node 对应一个简单的任务, 能够与其他 node 交互信息。本文数据采集实验中使用的 ROS 系统构成如图11, 标定过程中使用的系统构成如图12。

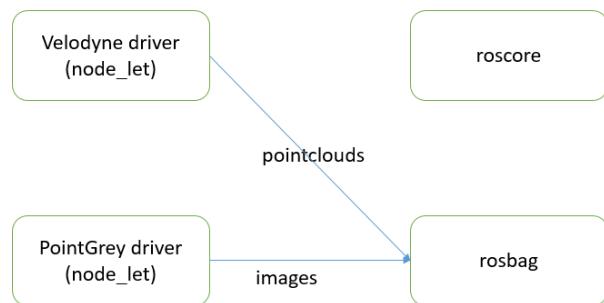


图 11: 数据采集时的节点关系

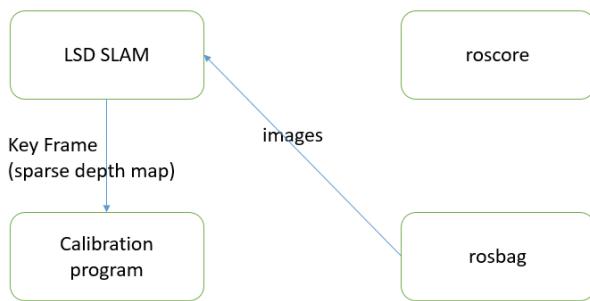


图 12: 标定时的节点关系

6.2.2 传感器驱动

我们使用的是 VLP-16 激光雷达的 ROS 驱动⁴和 PointGrey 的 ROS 驱动⁵。同时使用 PointGrey 的配置软件 FlyCapture⁶对相机进行网络配置。使用 ROS 的 rqt_configure⁷程序对相机的曝光参数进行配置。

6.2.3 传感器之间的同步

由于实验中假设激光雷达和相机的数据来自同一时刻，所以需要对传感器进行同步。两种传感器都支持硬件触发，保证数据同时性的最好方法是使用外部的触发信号对两种传感器进行同时触发，但是受到实验条件的限制，我们并没有采用这种方法。实验中使用的相机最高帧率为 $60Hz$ ，激光雷达标准帧率为 $10Hz$ ，但是受到实验中使用的数据采集电脑的硬盘写入速率限制，我们只能将相机设置为 $15Hz$ ，激光雷达设置为 $10Hz$ 。为了尽可能减少同步误差，我们找到与相机关键帧时间戳最接近的激光雷达数据作为配对数据，此方法理论上存在的最大误差为 $50ms$ 。

6.2.4 传感器配置

我们使用 ROS 的 rqt_configure 程序对相机的曝光参数进行配置。为了适用于实验环境，减小运动畸变，相机被设置为快门优先的曝光方式，快门时间设置为 $\frac{1}{1000}s$ 。同时为了具有更大的景深，我们将相机光圈设置为最小，对焦位置设置为无限远处。实验中的拍摄效果如图13。

⁴<http://wiki.ros.org/velodyne>

⁵http://wiki.ros.org/pointgrey_camera_driver

⁶<https://www.ptgrey.com/support/downloads>

⁷http://wiki.ros.org/rqt_reconfigure

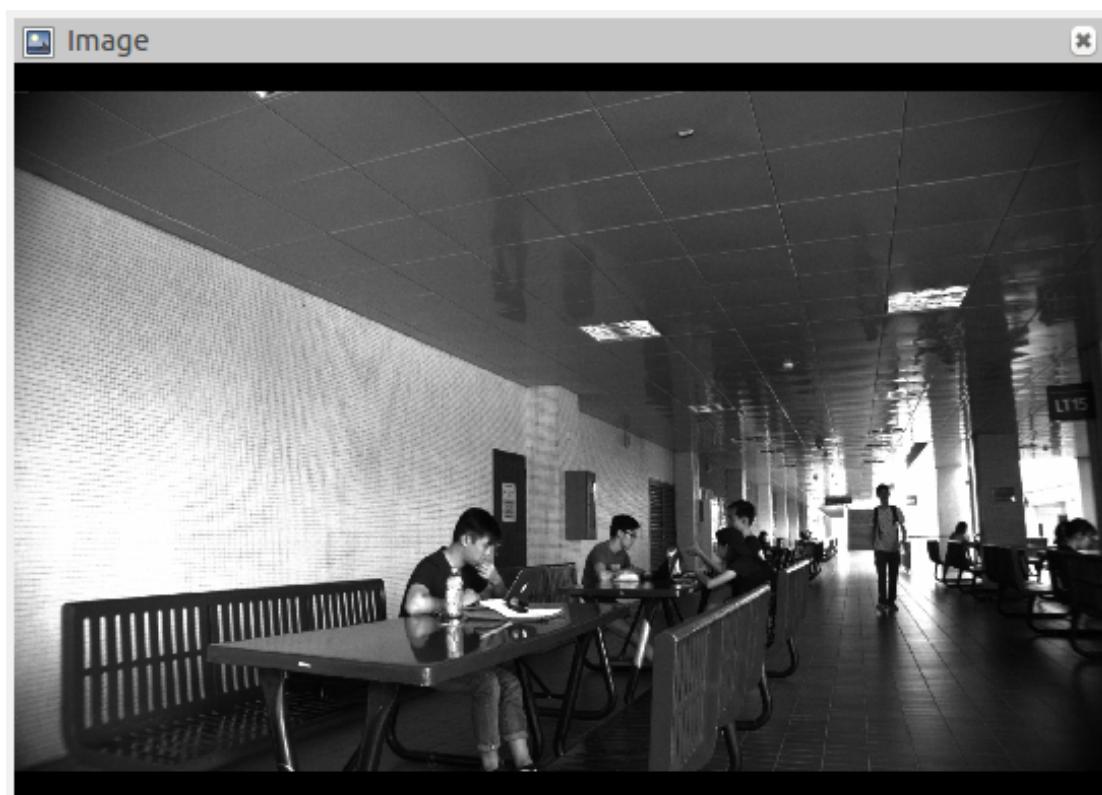


图 13: 相机拍摄样张

6.3 使用 Ceres Solver 求解问题

实验中我们首先尝试了使用 Ceres Solver⁸对问题进行求解。Ceres Solver 是一个非常著名的最小二乘问题优化框架，适合式(4)中的形式。我们使用此框架默认的优化算法，即 Levenberg-Marquardt[7][8] 优化方法，此方法是一个基于梯度的优化方法。为了求得梯度我们使用了两种不同的方法，即数值方法和解析方法。

6.3.1 数值方法

使用数值方法求解梯度适用于目标函数特别复杂的情况，在我们的问题中，我们首先尝试了这个简单的方法。此方法的问题在于求解梯度所使用的步长不易确定：本文中在求解 $D(u, v)$ 时使用最近邻插值，如果步长太小，会导致 u, v 变化很小，以至于 $D(u, v)$ 不会发生变化，导致求得的梯度不正确。

Ceres Solver 的数值求解方法中提供了一种自适应的求解方法，此方法会自动确定步长，在一定程度上解决了上述问题。但是实验中发现，使用数值方法的求解算法大部分时间耗费在 Jacobian 矩阵的求解上，求解时间在几个小时左右，完全不具备实用价值。

6.3.2 解析方法

为了解决数值方法耗时长的问题，我们自己求得了 Jacobian 矩阵的解析解。具体求解见3.3.1节。其中 $\frac{\partial u}{\partial \Theta}, \frac{\partial v}{\partial \Theta}, \frac{\partial z}{\partial \Theta}$ 的求解虽然是平凡的，但是极其复杂，人工完成其推导是困难的。我们使用符号计算框架 SymPy⁹求解这些导数，并生成

⁸<http://ceres-solver.org/>

⁹<http://www.sympy.org/en/index.html>

```

void Jacobian_P_T(const double *P_L, const double *T, double *out_4693495860126994303) {
    double sinT3 = sin(T[3]); double cosT3 = cos(T[3]);
    double sinT4 = sin(T[4]); double cosT4 = cos(T[4]);
    double sinT5 = sin(T[5]); double cosT5 = cos(T[5]);
    out_4693495860126994303[0] = 1.0;
    out_4693495860126994303[1] = 0.0;
    out_4693495860126994303[2] = ((-316.473266602*cosT3*cosT5)*P_L[1] - sinT4*cosT4*P_L[0] + cosT3*cosT4*P_L[2] + P_L[3]*T[2]);
    out_4693495860126994303[3] = (-316.473266602*cosT3*cosT5)*P_L[0] + (374.672943115*cosT4*cosT5) + 316.473266602*sinT3*cosT4 - 374.672943115*sinT3*sinT5 + 374.672943115*sinT4*cosT3*cosT5 + 316.473266602*cosT3*cosT4)*P_L[2];
    out_4693495860126994303[4] = 0.0;
    out_4693495860126994303[5] = (-316.473266602*cosT3*cosT5)*P_L[1] - sinT4*cosT4*P_L[0] + cosT3*cosT4*P_L[2] + P_L[3]*T[2];
    out_4693495860126994303[6] = 0.0;
    out_4693495860126994303[7] = ((-316.473266602*cosT3*cosT5)*P_L[0] + (374.672943115*cosT4*cosT5) + 316.473266602*sinT3*cosT4 - 374.672943115*sinT3*sinT5 + 374.672943115*sinT4*cosT3*cosT5 + 316.473266602*cosT3*cosT4)*P_L[2];
    out_4693495860126994303[8] = (-316.473266602*cosT3*cosT5)*P_L[1] - sinT4*cosT4*P_L[0] + cosT3*cosT4*P_L[2] + P_L[3]*T[2];
    out_4693495860126994303[9] = ((-316.473266602*cosT3*cosT5)*P_L[0] + (374.672943115*cosT4*cosT5) + 316.473266602*sinT3*cosT4 - 374.672943115*sinT3*sinT5 + 374.672943115*sinT4*cosT3*cosT5 + 316.473266602*cosT3*cosT4)*P_L[2];
    out_4693495860126994303[10] = 0.0;
    out_4693495860126994303[11] = ((-316.473266602*cosT3*cosT5)*P_L[1] - sinT4*cosT4*P_L[0] + cosT3*cosT4*P_L[2] + P_L[3]*T[2]);
    out_4693495860126994303[12] = 0.0;
    out_4693495860126994303[13] = 0.0;
    out_4693495860126994303[14] = P_L[3];
    out_4693495860126994303[15] = 0.0;
    out_4693495860126994303[16] = 0.0;
    out_4693495860126994303[17] = 0.0;
}

```

图 14: 变换函数求导使用的 C 语言代码, 使用 sympy 自动生成

C 语言代码, 如图14。

最终的求导代码如图15, 实验中发现此方法相比于数值方法极大地减少了求解时间, 求解时间在一分钟以内。

6.4 网格搜索方法

由于深度图的梯度存在局部性, 使用图像金字塔方法虽然可以解决此问题, 但会带来很大的工作量, 我们选择了一种较为简单的方法, 即网格搜索算法。为了减少搜索时间, 我们采用了分层搜索的方法, 将算法复杂度降低到了对数复杂度。

此算法的具体实现如图16, 图中的搜索步长、搜索范围以及搜索层数都是依据经验确定的, 在实验中取得了良好的效果。

```

// 5. calculate the jacobian matrix
// 5.1 sampleGradients 6xN (N = P_L_Matched.cols())
MatrixXd sampleGradients(6, P_L_Matched.cols());
for(int i=0; i<P_L_Matched.cols(); i++)
{
    // "gradient" of this sample
    Matrix<double, 6, 1> gradient;
    // a sample point in PointCloud
    Matrix<double, 4, 1> P_L = P_L_Matched.col(i);
    Matrix<double, 3, 1> P_C = P_C_Matched.col(i);
    P_C(0, 0) /= P_C(2, 0);
    P_C(1, 0) /= P_C(2, 0);
    double u = P_C(0,0), v = P_C(1,0);

    // beta_x
    Matrix<double, 2, 1> X;
    X << sampleBuffer(0, i), sampleBuffer(1, i);
    Matrix<double, 2, 1> beta_x = probability.getBeta_x(X);
    // std::cout<< beta_x.transpose() << std::endl;

    // Jacobian_X_xi
    Matrix<double, 6, 2> Jacobian_X_xi;
    Matrix<double, 6, 3> Jacobian_uvz_xi = getJacobian_uvz_xi(P_L, xi_cam_velo); // 6x3
    Matrix<double, 2, 1> Jacobian_D_uv; Jacobian_D_uv << (*_depth_gradient_x)(int(v), int(u)), (*_depth_gradient_y)(int(v), int(u));
    Jacobian_X_xi.col(0) = Jacobian_uvz_xi.leftCols(2) * Jacobian_D_uv;
    Jacobian_X_xi.col(1) = Jacobian_uvz_xi.col(2);

    // gradient
    gradient = Jacobian_X_xi * beta_x;

    sampleGradients.col(i) = gradient;
}
// 5.2 J = E(sampleGradients)
Matrix<double, 6, 1> J = sampleGradients.rowwise().mean();
for(int i=0; i<6; i++){
    jacobian[i] = J(i, 0);
}

```

图 15: 最终求导完整代码

```

void GridSearch(const double *initial, const double *range, const int *num_steps, double *result, std::vector<MICostFunction *> &costV)
{
    double minCost = 1e10;
    double param[6] = {0.0};
    double * param1 = {param};
    double residuals[1];
    double total_cost = 0.0;

    int total_steps = 1;
    int dimension_steps[6] = {0};
    for(int i=5;i>0;i--){
        dimension_steps[i] = total_steps;
        total_steps *= num_steps[i];
    }
    // std::cout<<"dimension steps = ["; for ( auto a:dimension_steps ) std::cout<<a<<" "; std::cout<<"]";<<std::endl;
    std::cout<<"total steps = "<<total_steps<<"; will cost about total_steps*costV.size()*1e-3 = "<<total_steps<<"*costV.size()<<"*1e-3<<"*s"<<std::endl;
    for(int i=0;i<total_steps;i++){
        int res_steps = i;
        int current_indexs[6];
        for(int p=0;p<6;p++){
            current_indexs[p] = res_steps / dimension_steps[p];
            res_steps = res_steps % dimension_steps[p];
            param[p] = initial[p] + range[p]/2 + double(current_indexs[p])/std::max(1, num_steps[p]-1)*range[p];
        }
        // std::cout<<"current_indexs = ["; for ( auto a:current_indexs ) std::cout<<a<<" "; std::cout<<"]";<<std::endl;
        total_cost = 0.0;
        for(int j=0;j<costV.size();j++){
            costV[j].Evaluate(param, residuals, nullptr);
            total_cost += residuals[0];
        }
        if(total_cost < minCost)
        {
            minCost = total_cost;
            for(int p=0;p<6;p++){
                result[p] = param[p];
            }
        }
    }
    std::cout<<"final cost: "<<minCost<<std::endl;
}

```

图 16: 网格搜索算法的代码

Reference

- [1] Massoud Babaie-Zadeh, Christian Jutten, and Kambiz Nayebi. Differential of the Mutual Information. *IEEE Signal Processing Letters*, 11(1):48–51, 2004.
- [2] Jakob Engel, Thomas Schöps, and Daniel Cremers. LSD-SLAM: Large-Scale Direct monocular SLAM. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8690 LNCS, pages 834–849, 2014.
- [3] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The KITTI dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.
- [4] Andreas Geiger, Frank Moosmann, Omer Car, and Bernhard Schuster. Automatic camera and range sensor calibration using a single shot. *IEEE Conference on Robotics and Automation (ICRA)*, pages 3936–3943, 2012.
- [5] C. Harris and M. Stephens. A Combined Corner and Edge Detector. In *Proceedings of the Alvey Vision Conference 1988*, pages 23.1–23.6, 1988.
- [6] Lili Huang and Matthew Barth. A novel multi-planar LIDAR and computer vision calibration procedure using 2D patterns for automated navigation. In *IEEE Intelligent Vehicles Symposium, Proceedings*, pages 117–122, 2009.
- [7] K Levenberg and K Levenberg. A Method for the Solution of Certain Problems in Least Squares. In *Quart. Appl. Math.*, volume 2, pages 164—168, 1944.

- [8] Donald W. Marquardt. An Algorithm for Least-Squares Estimation of Nonlinear Parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.
- [9] Gaurav Pandey, James R McBride, Silvio Savarese, and Ryan M Eustice. Automatic Targetless Extrinsic Calibration of a 3D Lidar and Camera by Maximizing Mutual Information. *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, pages 2053–2059, 2012.
- [10] Emanuel Parzen. On Estimation of a Probability Density Function and Mode. *The Annals of Mathematical Statistics*, 33(3):1065–1076, 1962.
- [11] Morgan Quigley, Brian Gerkey, and William D Smart. *Programming Robots with ROS: a practical introduction to the Robot Operating System.* ” O'Reilly Media, Inc.”, 2015.
- [12] Murray Rosenblatt. Remarks on Some Nonparametric Estimates of a Density Function. *The Annals of Mathematical Statistics*, 27(3):832–837, 1956.
- [13] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2564–2571, 2011.
- [14] Qilong Zhang and Robert Pless. Extrinsic Calibration of a Camera and Laser Range Finder (improves camera calibration) . *Iros*, 3:2301–2306, 2004.
- [15] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.

- [16] Huijing Zhao, Yuzhong Chen, and Ryosuke Shibasaki. An efficient extrinsic calibration of a multiple laser scanners and cameras ' sensor system on a mobile platform. *Intelligent Vehicles Symposium*, pages 422–427, 2007.