SVO 详细解读

Yuanlizheng

前言

接上一篇文章《VR\AR 空间定位中的地图点滤波》。

SVO（Semi-Direct Monocular Visual Odometry）是苏黎世大学 Scaramuzza 教授的实验室，在 2014 年发表的一种视觉里程计算法，它的名称是半直接法视觉里程计，通俗点说，就是结合了特征点法和直接法的视觉里程计。目前该算法已经在 github 上面开源（https://github.com/uzh-rpg/rpg_svo）。贺一家在它的开源版本上面进行改进，形成了 SVO_Edgelet（https://github.com/HeYijia/svo_edgelet）。相比原版，SVO_Edgelet 增加了一些功能，比如结合本质矩阵和单应矩阵来初始化，把边缘特征点加入跟踪等，对 SVO 的鲁棒性有较大的改善。

虽然 SVO 已经有论文[1]了，但是论文里面只是讲了最核心的算法理论，可以用论文来了解其算法思想。但是具体的实现方法和技巧，都隐藏在源代码里。要想彻底掌握它，并且在具体实践中灵活运用它，还是必须要阅读源代码才行。

所以我利用了好几个周末的时间，通读了这 2 万多行的源代码，力求从代码中反推出所有的具体实现、算法、公式、技巧、作者的意图。

源码之下，了无秘密。

在把这 2 万多行的代码全部搞懂之后，我把代码里面的具体实现方法全都一五一十地还原出来，研究其优缺点、适用情况，探讨其可以改进的地方，总结成本文，与各位分享。

以下视频为我们小组最近对 SVO 进行改进后，它的实际运行效果。

本文对应的程序为 SVO_Edgelet。

本文目标读者：对 SVO 有一定了解的 SLAM 算法工程师。

流程图

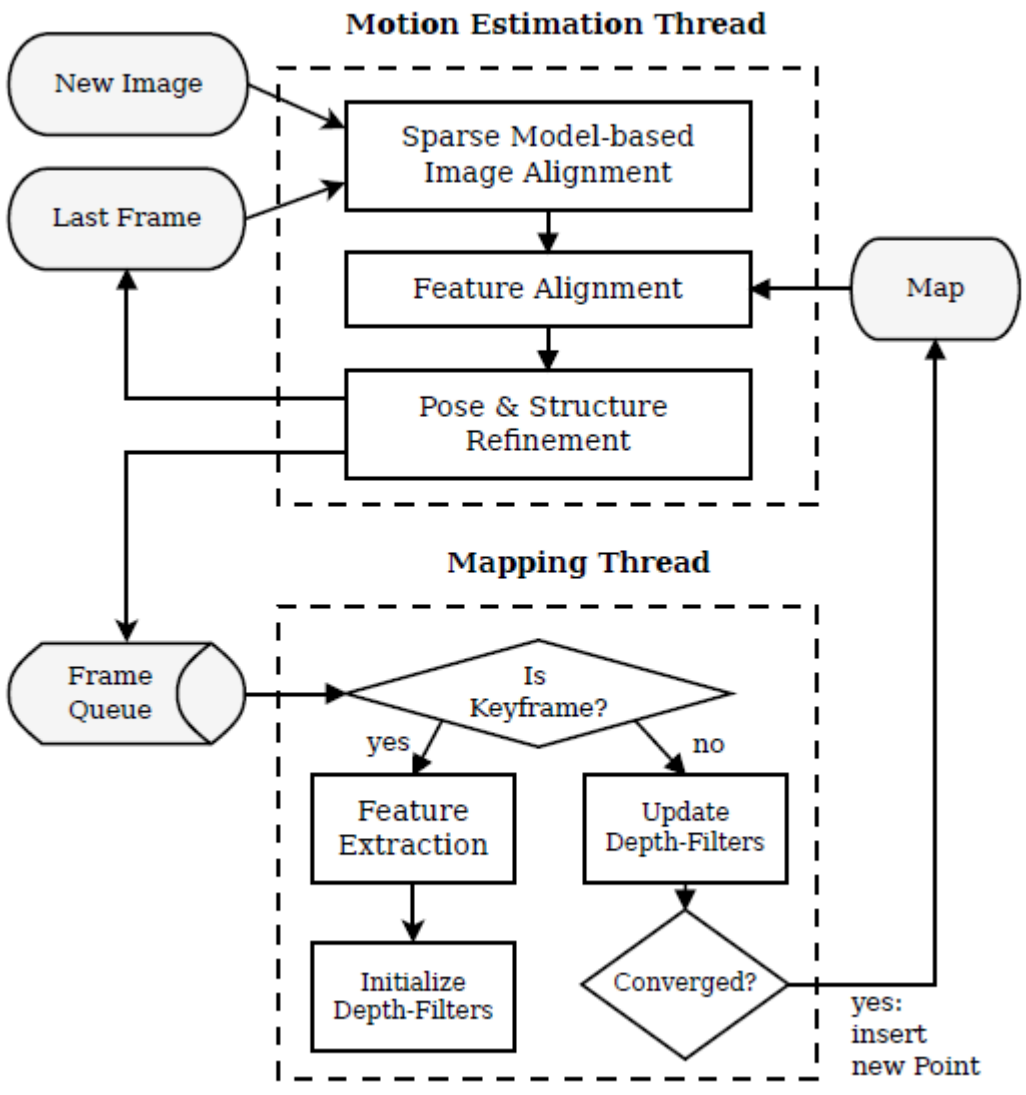


Fig. 1: Tracking and mapping pipeline

1.跟踪

其实，SVO 的跟踪部分的本质是跟 ORBSLAM 一样的，TrackWithMotionModel 和 TrackLocalMap，只是匹配的方法从特征点法改成了灰度值匹配法。

但是，随后，与 ORBSLAM 有不同的地方，SVO 在优化出相机位姿之后，还有可选项，可以再优化地图点，还可以再把地图点和相机位姿一起优化。

1.1 初始化

图像刚进来的时候，就获取它的金字塔图像，5 层，比例为 2。

然后处理第一张图像，processFirstFrame()。先检测 FAST 特征点和边缘特征。如果图像中间的特征点数量超过 50 个，就把这张图像作为第一个关键帧。

然后处理第一张之后的连续图像，processSecondFrame()，用于跟第一张进行三角初始化。从第一张图像开始，就用光流法持续跟踪特征点，把特征像素点转换成在相机坐标系下的深度归一化的点，并进行畸变校正,再让模变成 1,映射到单位球面上面。

如果匹配点的数量大于阈值，并且视差的中位数大于阈值。如果视差的方差大的话，选择计算 E 矩阵，如果视差的方差小的话，选择计算 H 矩阵。如果计算完 H 或 E 后，还有足够的内点，就认为这帧是合适的用来三角化的帧。根据 H 或 E 恢复出来的位姿和地图点，进行尺度变换，把深度的中值调为 1。

然后把这一帧，作为关键帧，送入到深度滤波器中。（就是送到的深度滤波器的 updateSeedsLoop()线程中。深度滤波器来给种子点在极线上搜索匹配点，更新种子点，种子点收敛出新的候选地图点。如果是关键帧的话，就初始化出新的种子点，在这帧图像里面每层的每个 25x25 大小的网格里，取一个最大的 fast 点。在第 0 层图像上，找出 canny 边缘点。）

之后就是正常的跟踪 processFrame()。

1.2 基于稀疏点亮度的位姿预估

把上一帧的位姿作为当前帧的初始位姿。

把上一帧作为参考帧。

先创建 n 行 16 列的矩阵 ref_patch_cache_，n 表示参考帧上面的特征点的个数，16 代表要取的图块的像素数量。

再创建 6 行 n*16 列的矩阵 jacobian_cache_。代表图块上的每个像素点误差对相机位姿的雅克比。

要优化的是参考帧相对于当前帧的位姿。

把参考帧上的所有图块结合地图点，往当前帧图像的金字塔图像上投影。在当前帧的金字塔图像上，从最高层开始，一层层往低层算。每次继承前一次的优化结果。如果前一次的误差相比前前次没有减小的话，就继承前前次的优化后的位姿。每层的优化，迭代 30 次。

要优化的残差是，参考帧 I_{k-1} 上的特征点的图块与投影到当前帧 I_k 上的位置上的图块的亮度残差。投影位置是，参考帧中的特征点延伸到三维空间中到与对应的地图点深度一样的位置，然后投影到当前帧。这是 SVO 的一个创新点，直接由图像上的特征点延伸出来，而不是地图点（因为地图点与特征点之间也存在投影误差），这样子就保证了要投影的图块的准确性。延伸出来的空间点肯定也与特征点以及光心在一条直线上。这样子的针孔模型很漂亮。

SVO 的另外一个创新点是，以当前帧上的投影点的像素值为基准，通过优化调整参考帧投影过来的像素点的位置，以此来优化这两者像素值残差。这样子，投影过来的图块 *patch* 上的像素值关于像素点位置的雅克比，就可以提前计算并且固定了。（而以前普通的方法是，以参考帧投影过去的像素值为基准，通过优化投影点的位置，来优化这两者的残差。）

残差用公式表示为。

$$\sigma I = I_k(u) - patch\left(u - \left(\pi\left(T_{k,k-1}p_{k-1}\right) - l\right)\right)$$

其中，l 表示半个图块的大小。

把扰动加在 $T_{k,k-1}$ 上，也就是扰动参考帧 $k-1$ 相对于第 k 帧的位姿。因为一般用左乘扰动，所以这里也用左乘扰动。

$$\sigma I = I_k(u) - patch\left(u - \left(\pi\left(T(\xi)T_{k,k-1}p_{k-1}\right) - l\right)\right)$$

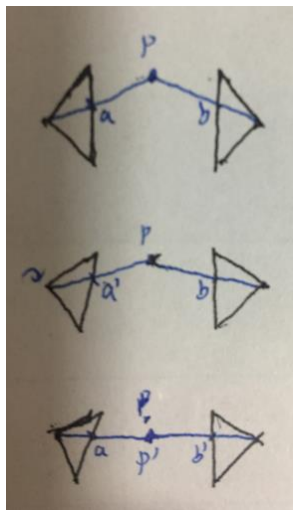
所以，残差关于扰动的雅克比为。

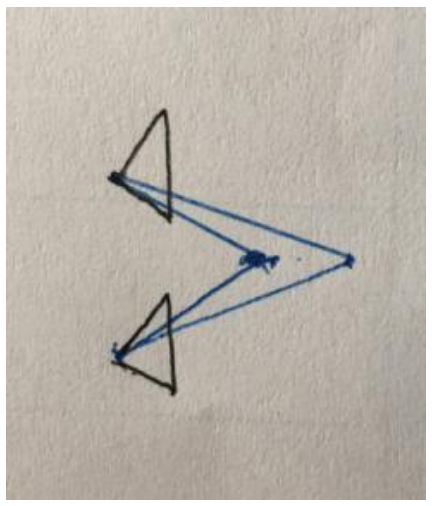
$$J = \frac{\partial \sigma I}{\partial \xi} = \frac{I_k(u) - patch\left(u - \left(\pi\left(T(\xi)T_{k,k-1}p_{k-1}\right) - l\right)\right)}{\partial \xi}$$

级联求导，但是，这样子的话，每次迭代后，雅克比 J 都会发生改变。一般情况下的优化，都会遇到这样的问题。

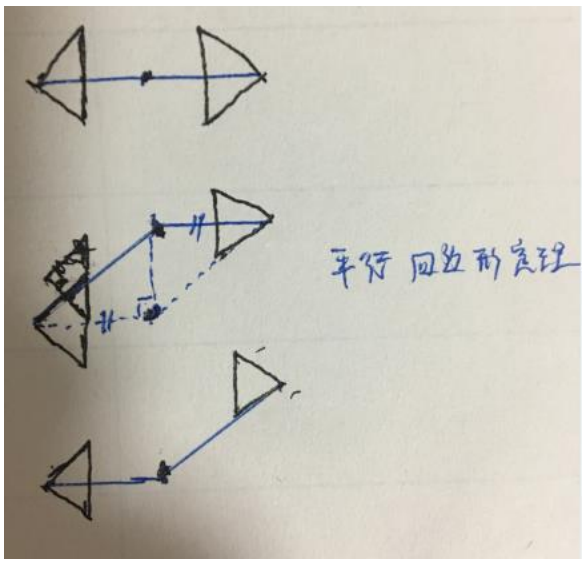
所以，采用近似的思想。首先，认为空间点 p 固定不动，只调整参考帧 $k-1$ 的位姿，所以这个扰动不影响在当前帧上的投影点的位置，只会影响图块 *patch* 的内容。然后，参考帧在新的位姿上重新生成新的空间点，再迭代下去。虽然只是近似优化，但每次迭代的方向都是对的，假设步长也差不多，所以最终也可以优化成功。

只是对公式优化的近似，抽象成如下的模型。

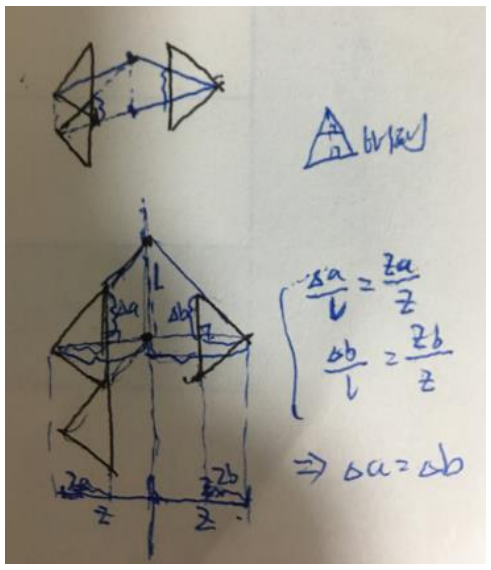


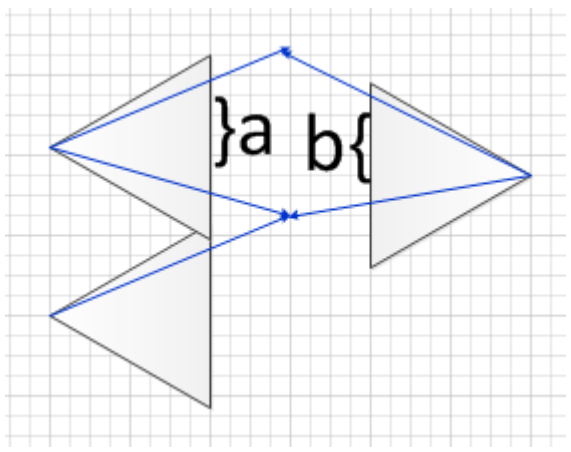


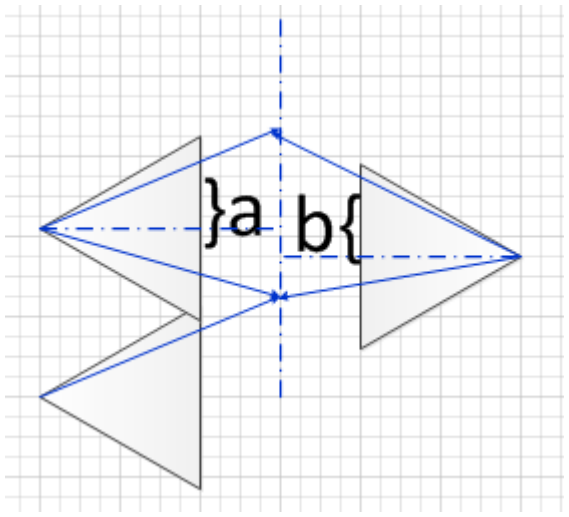
如果两个相机是互相正对着的，并且地图点在两个相机光心连线的中心的话，那肯定是符合的，跟公式优化的效果是相同的。可以通过平行四边形定理证明。



两相机的法线相同；或两相机的法线相反，地图点到两相机光心的距离相同。那这样子调整，也是跟优化的效果相同的。可以通过相似三角形比例定理来证明，作一个辅助平面出来，地图点在这个平面上，这个平面过两个相机光心连线的中点。可以用相似三角形定理证明 a 等于 b。

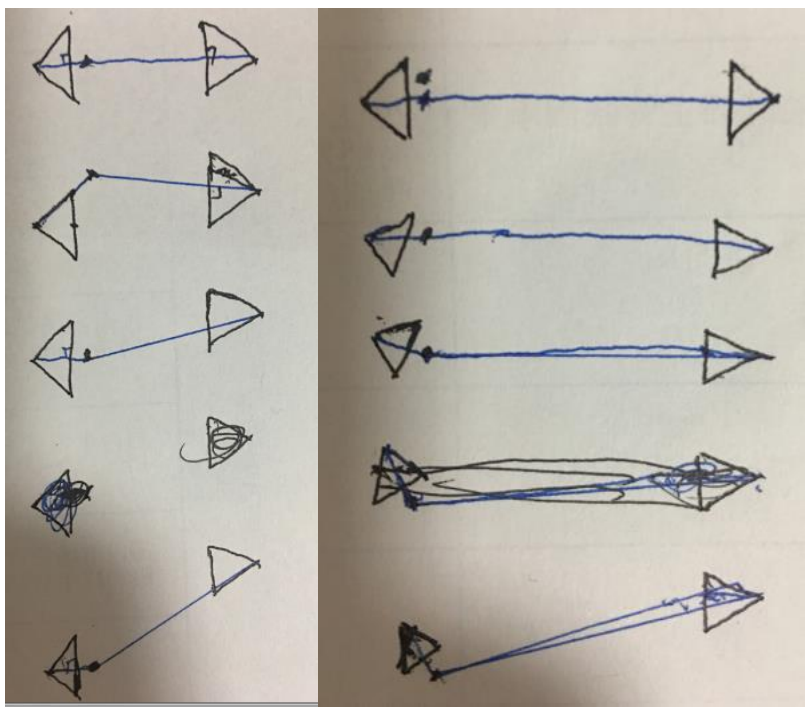






但是，这种模型，只有在这种情况下（两相机的法线相同；或两相机的法线相反，地图点到两相机光心的距离相同），才跟优化相同。

当不满足这种情况的时候，就会出现问题。比如，当地图点距离两个相机的差别很大的时候。按照公式优化，正确的优化结果应该是第 4 行，但是，按照这种近似的方法优化，优化的结果是第 3 行。所以，就与理想优化情况差别蛮大了。



而在 SVO 中是这样的情况，两个相机的法线方向相差不大，并且地图点离两个相机的光心都远大于光心距离，可以近似成地图点到两光心的距离是相等的。所以，在 SVO 中可以使用这个近似方法。

当前的残差是这样的。

$$\delta I(u_i) = I_k\left(\pi\left(\hat{T}_{k,k-1}p_{k-1}\right)\right) - I_{k-1}\left(\pi\left(T_{k-1,k-1}p_{k-1}\right)\right)$$

假设，给参考帧的相机位姿加个扰动ψ，变到$(k-1)'$的位姿，即

$$T_{k,(k-1)'} = T_{k,k-1}T_{k-1,(k-1)'} = T_{k,k-1}T(\psi)$$

则残差与扰动的关系可以表达如下。

$$\delta I(\psi, u_i) = I_k\left(\pi\left(\hat{T}_{k,k-1}p_{k-1}\right)\right) - I_{k-1}\left(\pi\left(T_{(k-1)',k-1}p_{k-1}\right)\right) = I_k\left(\pi\left(\hat{T}_{k,k-1}p_{k-1}\right)\right) - I_{k-1}\left(\pi\left(T(\psi)^{-1}\,p_{k-1}\right)\right)$$

因为其中有个逆矩阵，这样子在求导时会不方便，所以，使用$T(\xi)$来代替，在算出$T(\xi)$之后再逆过去。至于为什么在这种情况下，在优化的时候可以用另一中形式的变量取代掉，在算出来后再变换回去，参考《优化过程的中间误差的传递》。

$$T(\xi)=T(\psi)^{-1}$$

所以，原式就可以转换为。

$$\delta I(\xi,u_i)=I_k\left(\pi\left(\hat{T}_{k,k-1}p_{k-1}\right)\right)-I_{k-1}\left(\pi\left(T_{(k-1)',k-1}p_{k-1}\right)\right)=I_k\left(\pi\left(\hat{T}_{k,k-1}p_{k-1}\right)\right)-I_{k-1}\left(\pi\left(T(\xi)\,p_{k-1}\right)\right)$$

然后，就可以计算雅克比了。

$$J=\frac{\partial\delta I(\xi,u_i)}{\partial\xi}=-\frac{\partial I_{k-1}(a)}{\partial a}\Big|_{a=u_i}\cdot\frac{\partial\pi(b)}{\partial b}\Big|_{b=p_i}\cdot\frac{\partial T(\xi)}{\partial\xi}\Big|_{\xi=0}\cdot p_{k-1}=-\frac{\partial I}{\partial u}\cdot\frac{\partial u}{\partial b}\cdot\frac{\partial b}{\partial\xi}$$

对于上一帧的每一个特征点，都进行这样的计算，在自己本来的层数上，取那个特征点左上角的 4x4 图块。如果特征点映射回原来的层数时，坐标不是整数，就进行插值，其实，本来提取特征点的时候，在这一层特征点坐标就应该是整数。把图块往这一帧的图像上的对应的层数投影，然后计算雅克比和残差。计算残差时，因为投影的位置并不刚好是整数的像素，所以会在投影点附近插值，获取与投影图块对应的图块。

最后，得到一个巨大的雅克比矩阵，以及残差矩阵。但是为了节省存储空间，提前就转换成了 H 矩阵。$H=J^TJ$ 。

用高斯牛顿法算出扰动ξ。

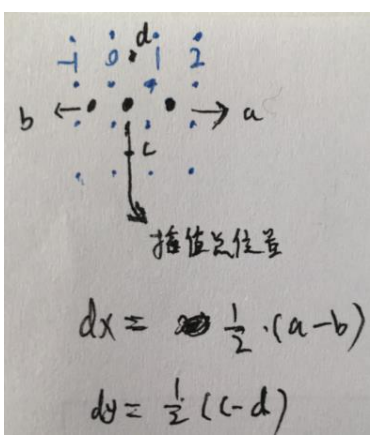
$$J^TJ\xi=-J^T\delta I$$
$$H\xi=-J^T\delta I$$
$$\Rightarrow\xi=-H^{-1}J^T\delta I$$

然后，得到$T(\xi)$，逆矩阵得到$T(\psi)$，再更新出$T_{k,(k-1)'}$。然后，在新的位置上，再从像素点坐标，投影出新的点 p_{k-1}。

每一层迭代 30 次。因为这种 inverse-compositional 方法，用这种近似的思想，雅克比就可以不用再重新计算了。（因为重新投影出新的 p 点的位置，这个过程没有在残差公式里面表现出来。）这样子逐层下去，重复之前的步骤。

对于每一个图块的每一个像素，它的雅克比计算如下。

$\frac{\partial I}{\partial u}$，是这个像素插值点在图像上的梯度，就是水平右边的像素点减去水平左边的像素点，竖直下边的像素点减去竖直上边的像素点。如下图所示。



$$\frac{\partial I}{\partial u}=\begin{bmatrix}dx & dy\end{bmatrix}$$

$\frac{\partial u}{\partial b}$，算的是投影雅克比。

$$u=\pi(b)=K\begin{bmatrix}b_x/b_z\\ b_y/b_z\\ b_z/b_z\end{bmatrix}=\begin{bmatrix}f_x & 0 & u_0\\ 0 & f_y & v_0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}b_x/b_z\\ b_y/b_z\\ 1\end{bmatrix}=\begin{bmatrix}f_xb_x/b_z+u_0\\ f_yb_y/b_z+v_0\\ 1\end{bmatrix}\Rightarrow$$

$$\frac{\partial u}{\partial b}=\begin{bmatrix}f_x/b_z & 0 & -f_xb_x/b_z^2\\ 0 & f_y/b_z & -f_yb_y/b_z^2\end{bmatrix}$$

$$\frac{\partial b}{\partial\xi}=\frac{\partial T(\xi)\,p}{\partial\xi}=\frac{\partial\left(\delta\phi^{\wedge}p+\delta\rho\right)}{\partial\begin{bmatrix}\delta\rho & \delta\phi\end{bmatrix}}=\begin{bmatrix}I & -p^{\wedge}\end{bmatrix}=\begin{bmatrix}1&0&0&0&p_z&-p_y\\0&1&0&-p_z&0&p_x\\0&0&1&p_y&-p_x&0\end{bmatrix}$$

为了方便计算，虽然$b=T(\xi)\,p$，但因为$T(\xi)$是一个很小的扰动，所以可以认为$b\approx T(\xi)\,p$，$\frac{\partial u}{\partial b}=\frac{\partial u}{\partial\left(T(\xi)\,p\right)}\approx\frac{\partial u}{\partial p}$。所以，后两项就可以相乘，统一用 p 来表示了。可以认为 $f=f_x=f_y$。

$$\frac{\partial u}{\partial b}\frac{\partial b}{\partial \xi}=\begin{bmatrix} f_x/p_z & 0 & -f_x p_x/p_z^2 \\ 0 & f_y/p_z & -f_y p_y/p_z^2 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 & 0 & p_z & -p_y \\ 0 & 1 & 0 & -p_z & 0 & p_x \\ 0 & 0 & 1 & p_y & -p_x & 0 \end{bmatrix}$$

$$=\begin{bmatrix} f_x/p_z & 0 & -f_x p_x/p_z^2 & -f_x p_x p_y/p_z^2 & f_x+f_x p_x^2/p_z^2 & -f_x p_y/p_z \\ 0 & f_y/p_z & -f_y p_y/p_z^2 & -f_y-f_y p_y^2/p_z^2 & f_y p_x p_y/p_z^2 & f_y p_x/p_z \end{bmatrix}$$

$$=\begin{bmatrix} 1/p_z & 0 & -p_x/p_z^2 & -p_x p_y/p_z^2 & 1+p_x^2/p_z^2 & -p_y/p_z \\ 0 & 1/p_z & -p_y/p_z^2 & -1-p_y^2/p_z^2 & p_x p_y/p_z^2 & p_x/p_z \end{bmatrix} f$$

对于每一个特征点，根据像素位置算出 $\frac{\partial I}{\partial u}$，再根据反投影出来的空间点位置 p 算出上式的左边项，根据特征点所在的层数算出 f。然后，相乘，就得到了一行雅克比矩阵。再根据 $H=J^TJ$， $J^TJ\xi=-J^T\delta I$，加到 H 矩阵和残差矩阵上。

最后，在优化出 ξ 后，应该是这样更新，$T_{k,(k-1)'}=T_{k,k-1}T_{k-1,(k-1)'}=T_{k,k-1}T(\psi)=T_{k,k-1}T(\xi)^{-1}$。

但是，在程序里面，直接就是，$T_{k,(k-1)'}=T_{k,k-1}T(-\xi)$。可能是为了加快计算，认为 $T(\xi)^{-1}\approx T(-\xi)$。在 sparse_align.cpp 的 307 行，T_curnew_from_ref = T_curold_from_ref * SE3::exp(-x_); 这个地方，为什么不是 T_curnew_from_ref = T_curold_from_ref * (SE3::exp(x_)).inverse(); 需要以后研究一下。

这样子，就可以得到当前帧的位姿。

$$T_{k,w}=T_{k,k-1}T_{k-1,w}$$

1.3 基于图块的特征点匹配

因为当前帧有了 1.1 的预估的位姿。对于关键帧链表里面的那些关键帧，把它们图像上的分散的 5 点往当前帧上投影，看是否能投影成功，如果能投影成功，就认为共视。再把所有的共视关键帧，按照与当前帧的距离远近来排序。然后，按照关键帧距离从近到远的顺序，依次把这些关键帧上面的特征点对应的地图点都往当前帧上面投影，同一个地图点只被投影一次。如果地图点在当前帧上的投影位置，能取到 8x8 的图块，就把这个地图点存入到当前帧投影位置的网格中。

再把候选地图点都往当前帧上投影，如果在当前帧上的投影位置，能取到 8x8 的图块，就把这个候选地图点存入到当前帧投影位置的网格中。如果一个候选点有 10 帧投影不成功，就把这个候选点删除掉。

然后，对于每一个网格，把其中对应的地图点，按照地图点的质量进行排序（TYPE_GOOD> TYPE_UNKNOWN> TYPE_CANDIDATE> TYPE_DELETED）。如果是 TYPE_DELETED，则在网格中把它删除掉。

遍历网格中的每个地图点，找到这个地图点被观察到的所有的关键帧。获取那些关键帧光心与这个地图点连线，与，地图点与当前帧光心连线，的夹角。选出夹角最小的那个关键帧作为参考帧，以及对应的特征点。（注意，这里的这种选夹角的情况，是只适合无人机那样的视角一直朝下的情况的，应该改成 ORBSLAM 那样，还要再把视角转换到对应的相机坐标系下，再筛选一遍）。这个对应的特征点，必须要在它自己的对应的层数上，能获取 10x10 的图块。

然后，计算仿射矩阵。首先，获取地图点在参考帧上的与光心连线的模。然后它的对应的特征点，在它对应的层数上，取右边的第 5 个像素位置和下边的第 5 个像素位置，再映射到第 0 层。再转换到单位球上，再映射到三维空间中，直到与地图点的模一样的长度。把对应的特征点也映射到三维空间中，直到与地图点的模一样的长度。然后，再把这 3 个点映射到当前帧的（有畸变的）图像上。根据它们与中心投影点的位置变换，算出了仿射矩阵 A_cur_ref。A_cur_ref.col(0) = (px_du - px_cur)/halfpatch_size; A_cur_ref.col(1) = (px_dv - px_cur)/halfpatch_size;。仿射矩阵 A，就是把参考帧上的图块在它自己对应的层数上，转换到当前帧的第 0 层上。（这种把比例变换转换成矩阵表示的方法，很好）。

然后，计算在当前帧的目标搜索层数。通过计算仿射矩阵 A_cur_ref 的行列式，其实就是面积放大率。如果面积放大率超过 3，就往上一层，面积放大率变为原来的四分之一。知道面积放大率不再大于 3，或者到最高层。就得到了目标要搜索的层数。

然后，计算仿射矩阵的逆仿射矩阵 A_ref_cur。然后，这样子，如果以投影点为中心（5,5），取 10x10 的图块，则图块上每个像素点的（相对中心点的）位置，都可以通过逆仿射矩阵，得到对应的参考帧上的对应层数图像上的（相对中心点的）像素位置。进行像素插值。就是，把参考帧上的特征点附近取一些像素点过来，可以组成，映射到当前帧上的对应层数的投影点位置的附近，这些映射到的位置刚好组成 10x10 的图块。

然后，从映射过来的 10x10 的图块中取出 8x8 的图块，作为参考图块。对这个图块的位置进行优化调整，使得它与目标位置的图块最匹配。残差表达式为。

$$\delta I=I_{cur}(u')-I_r(u)-m$$

其中，δI 表示这个像素点对应的残差，u' 表示在这个像素点对应的当前图像上的对应图块的位置，u 表示这个像素点在参考图块上的位置，m 表示两个图块的均值差。

在这里，SVO 有两个创新点。

第一个创新的地方是。因为一般情况下，是基于自己图块不变，通过优化 u' 使得投影位置的图块跟自己最接近。而 SVO 是投影位置的图块不变，通过优化 u 使得自己图块与投影位置的图块最接近。这样的话，就可以避免重复计算投影位置图块像素关于位置的雅克比了。因为自己图块是固定的，所以雅克比是固定的，所以只需要计算一次。其实，这个创新点与 1.2 中的反向创新点一样，都是用近似优化的方法来。因为，如果是一般的方法的话，计算目标投影位置的图块的雅克比，是知道自己参考图块重新移动后，会遇到怎样的目标图块。而，这个反向的方法，并不知道重新移动后会遇到怎样的图块，只知道移动后，对当前的目标图块可以匹配得更好。也是一种迭代，近似优化的方法，但速度可以块很多，避免了重复计算雅克比。

第二个创新的地方是。一般情况下，两图块的均值差 m，都是直接把两个图块的均值相减的。但是，这样子的话，可能容易受某些极端噪声的影响。所以，SVO 中，直接把 m 也作为优化变量了。

于是，雅克比可以计算如下。

$$J=\frac{\partial\delta I}{\partial\begin{bmatrix}u & m\end{bmatrix}}=\frac{\partial\left(I_{cur}(u')-I_r(u)-m\right)}{\partial\begin{bmatrix}u & m\end{bmatrix}}=-\frac{\partial\left(I_r(u)+m\right)}{\partial\begin{bmatrix}u & m\end{bmatrix}}=-\begin{bmatrix}\frac{\partial I_r(u)}{\partial u} & 1\end{bmatrix}$$

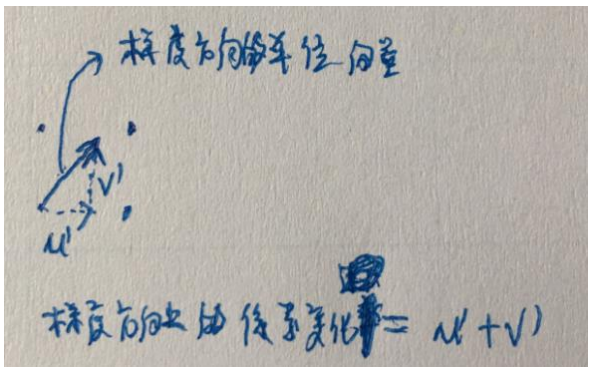
其中，$\frac{\partial I_r(u)}{\partial u}$就是参考图块上的雅克比。这个点的像素值关于位置（横坐标，纵坐标）的雅克比，其实就是这个点的右左像素值相减和下上像素值相减，得到的梯度。

对这个图块上的所有的像素点都进行这样的操作。然后用高斯牛顿法进行迭代。最多迭代 10 次，如果某次调整位置的模小于 0.03，就认为收敛了，退出迭代。得到最佳匹配点的位置。认为匹配成功。

$$J\begin{bmatrix}u\\m\end{bmatrix}=\delta I\Rightarrow$$

$$J^TJ\begin{bmatrix}u\\m\end{bmatrix}=J^T\delta I$$

而，如果是对于那些边缘上的点。则只在梯度方向上进行调整，只调整这 1 个维度，即梯度方向上的长度。右左下上像素的变化，映射到梯度方向上，得到在梯度方向上的像素变化。最后优化完后，再从这个维度上映射出横纵坐标。最后，也得到最佳匹配点的位置。



上面的优化，必须在 1.2 估算出的位姿较准确的情况下，才能使用这样的方法，在预测的投影点位置用像素梯度来优化出最佳匹配点位置。

如果是一个 TYPE_UNKNOWN 类型的地图点，它找匹配失败的次数大于 15 次，就把它变为 delete 类型的点。如果是一个 TYPE_CANDIDATE 类型的点，它匹配失败的次数大于 30 次，就把它变为 delete 类型的点。

如果匹配成功的话，就在当前图像上，新生成一个特征点（包括坐标和层数），特征点指向那个地图点。如果对应的参考帧上的特征点是边缘点的话，则新的特征点的类型也设为边缘点，把梯度也仿射过来，归一化后，作为这个新特征点的梯度。

每个网格里，只要有一个地图点匹配成，就跳出这个网格的遍历循环。如果有 180 个网格匹配成功了，直接跳出所有网格的循环。循环结束后，如果成功匹配的网格的数量小于 30 个，就认为当前帧的匹配失败。

1.4 进一步优化位姿

然后，对于 1.3 中的，当前帧上的所有的新的特征点u'，如果它指向的是地图点 p 的话，通过优化当前帧的相机位姿$T_{k,w}$，使得地图点的在对应的层数上的预测投影位置和最佳匹配位置的残差δ最小。注意，是在对应层数上的残差。

$$\delta=u'-\pi\left(T(\xi)T_{k,w}p\right)$$

雅克比为J。

$$J=\frac{\partial\delta}{\partial\xi}=\frac{\partial\left(u'-\pi\left(T(\xi)T_{k,w}p_w\right)\right)}{\partial\xi}=-\frac{\partial\left(\pi\left(T(\xi)T_{k,w}p_w\right)\right)}{\partial\xi}=-\frac{\partial\left(\pi(b)\right)}{\partial\xi}=-\frac{\partial\left(\pi(b)\right)}{\partial b}\frac{\partial b}{\partial\xi}$$

$$=-\begin{bmatrix}1/p_z & 0 & -p_x/p_z^2 & -p_xp_y/p_z^2 & 1+p_x^2/p_z^2 & -p_y/p_z\\ 0 & 1/p_z & -p_y/p_z^2 & -1-p_y^2/p_z^2 & p_xp_y/p_z^2 & p_x/p_z\end{bmatrix}f$$

上式结果中的$p=p_k=T_{k,w}p_w$。

程序里为了计算方便，优化公式$J\xi=\delta$的左右两边都约掉第 0 层的f，所以就只剩下 1.0 / (1<<(*it)->level)了，右边也只需要算到单位平面再乘以 1.0 / (1<<(*it)->level)就可以了。

如果是边缘点的话，则把重投影误差映射到梯度方向上。

使用了核函数 TukeyWeightFunction，根据误差的模来调整误差的权重。Tukey's hard re-descending function，http://en.wikipedia.org/wiki/Redescending_M-estimator。

用高斯牛顿方来优化。

$$J\xi=\delta\Rightarrow\xi=\left(J^TJ\right)^{-1}J^T\delta$$

然后，程序里，通过误差平方和的值是否变大，来判断这次优化是否有效。（但是，这个误差平方和是在优化之前的，程序里可能写错了，应该再优化之后再算误差平方和。）

总共优化迭代 10 次，如果某次优化量约等于 0，则跳出优化循环。

优化结束后，接下来，要算这个算出来的位姿的协方差，即增加的扰动ξ的协方差，就是对应的高斯分布里面的那个协方

差。这里，可以通过高斯分布，转换出位姿的协方差。因为，参考卡尔曼滤波的状态转移方程，协方差，也是会随着状态转移矩阵而改变的。假设，在对应的层数上，测量值的协方差都为 1 个像素，即测量值满足方差为 1 的高斯分布。即 $P_\delta = 1 \cdot I$（如果是其它方差的话，改成 $x \cdot I$，同样代入下面的公式即可），要求 P_ξ。

根据 $J\xi = \delta \Rightarrow \xi = \left(J^T J\right)^{-1} J^T \delta$，得出，

$$P_\xi = \left(\left(J^T J\right)^{-1} J^T\right) P_\delta \left(\left(J^T J\right)^{-1} J^T\right)^T = \left(J^T J\right)^{-1} J^T J \left(J^T J\right)^{-T} = \left(J^T J\right)^{-T} = \left(J^T J\right)^{-1}$$

最后，如果有些点的的重投影误差，映射到第 0 层上，模大于 2 个像素的话，则把这个特征点指向地图点的指针赋值为空。如果最后剩下的匹配成功点的数量大于 20 个，就认为优化成功。

1.5 优化地图点

就是 optimizeStructure。在程序里，用 nth_element 找出前 20 个，最近一次优化帧的 id，离当前帧 id 较远的，地图点。

针对每个地图点，优化地图点的三维位置，使得它在它被观察到的每个关键帧上的重投影误差最小。每个地图点优化 5 次。如果这次优化的误差平方和小于上次优化的误差平方和，就接受这次的优化结果。（注意，这里的平方和也是在优化之前算的，其实应该在优化之后算）。如果是边缘点的话，则把重投影误差映射到梯度方向上，成为梯度方向上的模，就是与梯度方向进行点积。相应的，雅克比也左乘对应的梯度方向。相当于是，优化重投影误差在梯度方向上的映射。

对于普通点，把扰动 σ 加在三维坐标 p 上，重投影误差为，

$$\delta = u' - \pi\left(T_{k,w}\left(p+\sigma\right)\right)$$

雅克比为 J，

$$J = \frac{\partial \delta}{\partial \sigma} = \frac{\partial\left(u' - \pi\left(T_{k,w}\left(p+\sigma\right)\right)\right)}{\partial \sigma} = -\frac{\partial\left(\pi\left(T_{k,w}\left(p+\sigma\right)\right)\right)}{\partial \sigma} = -\frac{\partial\left(\pi(b)\right)}{\partial \sigma} = -\frac{\partial \pi(b)}{\partial b}\frac{\partial b}{\partial \sigma}$$

$$= -f\begin{bmatrix} 1/p_z & 0 & -p_x/p_z^2 \\ 0 & 1/p_z & -p_y/p_z^2 \end{bmatrix} \frac{\partial\left(T_{k,w}p + T_{k,w}\sigma\right)}{\partial \sigma}$$

$$= -f\begin{bmatrix} 1/p_z & 0 & -p_x/p_z^2 \\ 0 & 1/p_z & -p_y/p_z^2 \end{bmatrix} \frac{\partial\left(R_{k,w}\sigma + t_{k,w}\right)}{\partial \sigma}$$

$$= -f\begin{bmatrix} 1/p_z & 0 & -p_x/p_z^2 \\ 0 & 1/p_z & -p_y/p_z^2 \end{bmatrix} R_{k,w}$$

所以，根据 $J\sigma = \delta$，用高斯牛顿法来进行计算。在程序里，为了计算方便，$J\sigma = \delta$ 的左右两边，都约去了对应层数的 f，所以，右边就只需要算到深度为 1 的平面上的残差就可以。

如果是边缘点的话，则上式的左右两边都要乘以梯度的转置。$grad^T J\sigma = grad^T \delta$。也用高斯牛顿法来算。

1.6 BA

SVO 里面有个选项，可以开启使用 g2o 的 BA 功能。

如果开启使用这个功能的话，则在一开始的两张图像初始化之后，两张图像以及初始化出来的地图点，会用 BA 来优化。用的是 g2o 里面的模板。

另外，会在 1.5 优化完地图点后，对窗口里的所有的关键帧和地图点，或者全局关键帧和地图点，进行优化。用的是 g2o 里面的模板。

1.7 对畸变图像处理的启发

SVO 的跟踪都是在畸变的鱼眼图像上跟踪的，没有对图像进行校正，这样子可以尽可能地保留图像的原始信息。

又因为在 1.2 中的逆向图块雅克比的方法，除了可以加快计算外，还避免了对畸变参数的雅克比计算。因为如果用正向图像雅克比的话，在计算雅克比的时候，必须要把畸变参数也考虑进来。

而在 1.3 中，图块匹配就是用畸变的图块取匹配的，保证了准确性。为了避免对畸变参数的雅克比计算，在匹配完成后，把投影点位置和匹配点位置都从畸变的图像上，转换到了单位平面上。以后在畸变图像上，计算重投影误差，就用这样的方法。

2.创建地图点

特征点提取的方法，放在了地图线程里。因为与 ORBSLAM 不同的是，它跟踪的时候，不需要找特征点再匹配，而是直接根据图块亮度差匹配的。

而如果是 vins 的话，也可以参考这个方法，把特征点提取放到地图线程里，连续帧之间的特征点用光流匹配。但光流要求帧与帧之间不能差别太大。

而在 SVO 中，后端的特征点是只在关键帧上提取的，用 FAST 加金字塔。而上一个关键帧的特征点在这一个关键帧上找匹配

点的方法，是用极线搜索，寻找亮度差最小的点。最后再用 depthfilter 深度滤波器把这个地图点准确地滤出来。

选取 30 个地图点，如果这 30 个地图点在当前帧和最近一个关键帧的视差的中位数大于 40，或者与窗口中的关键帧的距离大于一定阈值，就认为需要一个新的关键帧。然后把当前帧设置为关键帧，对当前帧进行操作。

2.1 初始化种子

当关键帧过来的时候，对关键帧进行处理。在当前图像上，划分出 25 像素*25 像素的网格。

首先，当前帧上的这些已经有的特征点，占据住网格。

在当前帧的 5 层金字塔上，每层头提取 fast 点，首先用 3x3 范围的非极大值抑制。然后，对剩下的点，全部都计算 shiTomasi 分数，有点像 Harris 角点里面的那个分数。再全部映射到第 0 层的网格上，每个网格只保留分数最大的，且大于阈值的那个点。

找边缘点的话，都只在第 0 层上面找。同样也是画网格，然后再每个网格中找 canny 线，然后对于网格中的在 canny 线上的点，计算它的梯度的模，保留模梯度最大的那个点，作为边缘点。梯度方向是二维的，就是这个点的右左下上梯度。程序里用了 cv::Scharr 结合 cv::magnitude 来快速算出所有点的横纵方向的梯度。

然后，对于所有的新的特征点，初始化成种子点。用高斯分布表示逆深度。均值为最近的那个点的深度的倒数。深度范围为当前帧的最近的深度的倒数，即 1.0/depth_min。高斯分布的标准差为 1/6*1.0/depth_min。

2.2 更新种子，深度滤波器

如果新来一个关键帧，或者是当前的普通的帧，或者之前的关键帧，用于更新种子点。对于每个种子点，通过正负 1 倍标准差，确定逆深度的搜索范围。这些参数都是对应种子点在它自己被初始化的那一帧。

然后把深度射线上的最短和最长的深度，映射到当前帧的单位深度平面上，其实就得到的在单位平面上的极线线段。然后，再把逆深度的均值对应的深度，映射到当前帧，就是跟 1.3 中的同样的方法，得到图块仿射矩阵，和最佳搜索层数。

（对于边缘点，如果把梯度仿射过来后，梯度的方向与极线方向的夹角大于 45 度，就认为沿着极线找，图块像素也不会变化很大，就不搜索了，直接返回 false。）

把极线线段投影到对应的层数上，如果两个端点间的像素距离小于 2 个像素，就直接进行优化位置。用的是 1.3 中的找图块匹配的方法，把对应的图块映射过来。找到最佳匹配位置后，进行三角定位。三角定位的方法参考《视觉 SLAM 十四讲》的三角定位，矩阵分块计算。

$$
\begin{aligned}
&s_{cur}x_{cur} = s_{ref}R_{c\_ref}x_{ref} + t_{cur\_ref} \\
&s_r R_{c,r}x_r - s_c x_c = -t_{c,r} \\
&\begin{bmatrix} R_{c,r}x_r & -x_c \end{bmatrix}\begin{bmatrix} s_r \\ s_c \end{bmatrix} = -t_{c,r} \\
&\begin{bmatrix} s_r \\ s_c \end{bmatrix} = -\left(\begin{bmatrix} R_{c,r}x_r & -x_c \end{bmatrix}^T \begin{bmatrix} R_{c,r}x_r & -x_c \end{bmatrix}\right)^{-1} \begin{bmatrix} R_{c,r}x_r & -x_c \end{bmatrix}^T t_{c,r}
\end{aligned}
$$

如果两个端点间像素距离大于 2 个像素，就在极线上进行搜索。首先，确定总步长数，以两端点间的距离除以 0.7，得到总步长数 n_steps。然后，把单位深度平面上的极线线段分 n_steps 段，从一个端点开始往另外一个端点走，每走一步，就把位置投影（包括畸变）到对应层数的图像上，坐标取整后，获取图块。（这里可以改进，不应该对坐标进行取整，而应该改成插值）。然后，计算投影过来的图块与投影位置图块的相似度，相似度的计算公式如下，其中有消除均值的影响。

$$
s = \sum_i \sum_j \left(\left(A(i,j) - B(i,j) \right)^2 \right) - \frac{1}{i*j} * \left(\sum_i \sum_j A(i,j) - \sum_i \sum_j B(i,j) \right)^2
$$

如果分数小于阈值，就认为两个图块是相似的。在当前位置，再进行优化位置，用的是 1.3 中的找图块匹配然后优化位置的方法。然后再进行三角定位。

接下来，计算这个三角定位出来的深度值的协方差。用的是《视觉 SLAM 十四讲》的深度滤波。假设，在图像上的测量协方差为 1 个像素，则这个协方差的传递到深度上的过程如下。这个传递的，都还是标准差 σ，而不是 σ^2。

$$
\begin{aligned}
&\delta\beta = 2\arctan\left(\frac{0.5}{f}\right) \\
&a = p - t \\
&\alpha = \arccos\left(\frac{p \cdot t}{\|p\|\|t\|}\right) \\
&\beta = \arccos\left(\frac{a \cdot (-t)}{\|a\|\|t\|}\right) \\
&\beta' = \beta + \delta\beta \\
&\gamma = \pi - \alpha - \beta' \\
&\|p'\| = \frac{\|t\|\sin\beta'}{\sin\gamma} \\
&\sigma_{obs} = \|p'\| - \|p\|
\end{aligned}
$$

再把这个协方差传递到逆深度上。假设这时候三角定位出来的深度值为 z，则在逆深度上的标准差 δ_{inv} 为，

$$\delta_{inv} = \frac{1}{2}\left(\frac{1}{z-\sigma_{obs}} - \frac{1}{z+\sigma_{obs}}\right)$$

所以，这个测量出来的深度，满足的分布为 $N\left(\frac{1}{z}, \delta_{inv}^2\right)$。然后，就是更新种子点的深度分布了，参考《VR\AR 空间定位中的地图点滤波》。但是在 DepthFilter.cpp 的 486-490 行对系数进行平均了。这里与 depthfilter 的论文里推导的不一样。可能这里程序写错了，应该改成和论文里面一样。

如果种子点的方差，小于深度范围/200 的时候，就认为收敛了，它就不再是种子点，而是 candidate 点。candidate 点被成功观察到 1 次，就变成 UNKNOW 点。UNKNOW 被成功观察到 10 次，就变成 GOOD 点。如果多次应该观察而没有被观察到，就变成 DELETE 点。

3.重定位

SVO 中重定位，实现很简单，就是在跟丢之后，仍然假设当前帧的位姿和前一帧一样，往这个位姿上投地图点，用第 1 部分中的方法去优化计算，如果优化成功，就重定位回来，如果优化不成功，就继续下一帧。所以，在跟丢后，只能再回到跟丢时的位置，才能重定位回来。

这样子实现重定位的方法很简单，可重定位的效果就很差了。这地方可以进行改进。

4.总结

SVO 的定位很好，抖动很小。尤其在重复纹理的环境中，表现得比基于特征点法的 ORBSLAM2 要出色。

将来可以在上面增加更鲁棒的重定位，回环闭环，全局地图的功能，来满足更多的实际应用场景，比如室内机器人、无人机、无人车。

参考文献

[1] Forster C, Pizzoli M, Scaramuzza D. SVO: Fast semi-direct monocular visual odometry[C]// IEEE International Conference on Robotics and Automation. IEEE, 2014:15-22.

[2] 高翔.视觉 SLAM 十四讲[M].北京:电子工业出版社,2017:325-327.

致谢

感谢小组同事 ryangu, shonxiao, zonghaochen 的支持与帮助。

yuanlizheng

2018 年 3 月