SLAM 中的几何与学习方法

Geometric and Learning Approaches within SLAM systems

盐粒

updated 2020 年 5 月 7 日

目录

1 Hello SLAM

1.1 SLAM 研究方向的变迁

机器通过与环境发生一定的交互来完成相关的任务。为了将劳动者从繁杂、危险的环境中解放出来，机器被越来越广泛的应用在各种现实场景中，从扫地机器人到无人驾驶汽车。通过与环境发生一定的交互，机器可以自主的在较为复杂的场景中工作，因此它们也需要像人类一样的感知系统。视觉感知是感知系统的一部分，通过视觉传感器来完成机器的 6 自由度的运动估计和环境模型重建就是 SLAM（Simultaneous Localization and Mapping）需要完成的主要工作。

1.1.1 不断进化的框架

SLAM 框架通过二三十年的发展，逐渐成熟起来。中间跨时代的作品都获得了巨大的论文引用量。

对与基于特征法的 SLAM 系统，PTAM 算是一个萌芽，它发表与 2007 年，指出了多线程，关键帧的发展道路。这篇文章中是 Tracking and Mapping 在两个线程中分开运行。其中 Mapping 部分是只处理关键帧的，并使用了局部 BA。The map 由两帧图像密集初始化，5 点法计算 pose。新的特征点是使用极线搜索来实现匹配的。比起这篇文章更以前的方法，它可以实时维护几千个点的地图。同样是基于点特征的系统，ORB-SLAM2 [10] 是一个灿烂的花朵。这个系统是由 Raúl Mur-Artal 博士等人提出的一个完备的基于特征点的纯视觉实时 SLAM 系统，适用于单目、双目和 RGB-D 相机，发表于 2017 年。作者在 2015 年第一次将单目版本（ORB-SLAM [9]）发表在 IEEE Transactions on Robotics（T-RO）期刊上，这是一个基于纯特征点的单目实时 SLAM 框架。总体来说，该 ORB-SLAM 的两篇文章是基于特征的 SLAM 系统的大成之作，系统本身非常的完整，包含了稀疏地图，回环和优化环节。ORBSLAM2 将传感器从单目扩展到双目和 RGB-D，成为了迄今为止影响力最大的特征法 SLAM 系统之一，为 SLAM 在工业界和学术界的发展作出巨大贡献。

1.1.2 多几何特征与多传感器

为了应对低纹理场景中特征点能力的限制，线面特征被应用在系统中。

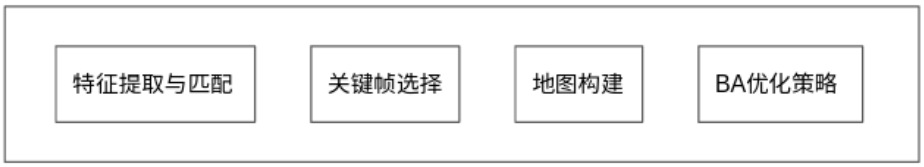


图 1: Caption

1.1.3 几何与学习的融合

1.2 为什么要写这本书

近年来，深度学习在更高层次的视觉任务中取得瞩目的成绩，如：物体识别，语义分割等，这些课题曾是传统视觉无法或很难解决的任务。正是由于深度学习拓展了我们对高层次视觉任务的想象，越来越多的 SLAM 开始在他们的框架中融合学习的方法，来改进位姿估计的准确程度和环境重建的效果。但是深度学习是一个非常宽广的领域，和 SLAM 相关的课题只是它的一个分支。本书将会挑选和聚焦和 SLAM 相关的深度学习话题，我们希望能通过这本书来介绍 SLAM 系统中使用的几何和深度学习的方法，帮助读者掌握最新的进展。从整体上理解基于几何的 SLAM 系统的技术细节，同时还能了解深度学习相关课题对传统几何 SLAM 的影响。

本书首先分析单帧图像入手：单帧图像的几何特征提取，单帧图像的深度学习任务分析。通过相机模型和刚体变换的规则，我们把图像帧联系起来，构建帧与帧之间的关系。然后按照 SLAM 中的每一个重要环节来详细介绍相关的知识。几何与学习特征的处理、优化问题的构建与求解，以及环境地图的创建与维护。在每个环节中，我们都深入的介绍相关的理论与工程细节，力求本书为读者带来快速的提高。几何与学习就像是 SLAM 的一双翅膀，未来的 SLAM 将提供更加的鲁棒、更加丰富的信息。最后，我们一起探讨 SLAM 的发展方向，畅想这个领域未来蓝图。

2 几何视觉基础

2.1 几何特征提取

2.1.1 点特征提取与匹配

2.1.2 线特征提取与匹配

2.1.3 面特征提取与匹配

3 深度学习视觉基础

基于相机的 RGB 影像可以提供环境中的结构信息和纹理信息，他们被广泛应用在计算机视觉和摄影测量的众多问题中。机器人和无人驾驶汽车利用相机传感器采集图像序列，在未知环境中的姿态估计和环境重建工作。

随着传感器技术的发展和计算能力的提高，价格便宜的 RGB-D 相机已经在很多场景中已经取得了很好的精度。其中 depth map 可以直接提供稠密的 3D 点云，他们对于场景理解，尤其是近距离场景，起到了重要的作用。但是这种传感器仍旧有很多限制，包括场景的大小、深度图空洞、边缘精度较差等。与此同时，深度学习迅速崛起，不仅在图片分类、物体检测、语义分割等任务上取得了突出的效果，而且在密集重建、场景补全、姿态估等傳統的 SLAM 问题上爆出巨大能力。这种数据驱动的方法，虽然在很大的部分仍然未得到充分解释，但是他为计算机视觉带来的巨大想象空间是不容忽视的。

不管从传感器的价格，还是应用场景（高温、高压、极寒、较远距离）的通用性来说，单目相机永远保持着他独特的优势。当我们无法使用主动光获取深度图时，能否利用这种学习的方法来获得深度图呢？实际上，国内外的研究学者已经在这个问题上取得了很不错的效果，本文就是要对众多的理论、方法进行梳理，分析其中主要方法的逻辑和优缺点，以便后来的读者可以更简单的理解问题、贡献智慧。

3.1 点线面的预测

3.2 物体检测与分割

3.3 深度图与法向量预测

3.3.1 监督学习

FCN [8] 是一篇无法避开的文章，他的 google scholar 引用已经达到 9600 左右。这是一个划时代的工作，他的出现为深度学习进入像素领域的操作实现了思想解放，大量的基于像素的语义分割的工作随之而来。回头来看 FCN 提出来的反卷积（deconvolution）操作是对特征图的上采样，实验效果也别后来的方法 [CRF][] 刷新，但是他对领域的贡献，已经反应在引用量上了。回到深度预测的话题，David et al. [2] 早在 2014 就提出使用深度学习做这项工作，他使用了非常基础的网络结构，如图一所示。框架分

为两个部分：coarse net 和 fine net，前一个网络通过使用较大卷积核子和全链接层试图获得更加全局的结构（structure）信息, 第二个网络完全使用卷积操作，并在把 coarseNet 的结果也作为一种 feature map 输入。这两个网络也是分开训练的。在这个框架基础上，David et al. 提出了更加复杂的框架 [3], 来同时处理 depth prediction, Surface normals(SN) and semantic segmentation. CoarseNet 和 FineNet 经过少许变化之后，输入 Scale3 以获得更细致的结果。

另外，论文 [1] 在 [2] 深度估计的 Loss 基础上进行修改，直接使用深度之差（predicted and GT）代替对数化的差，同时考虑差在水平、竖直图像上的梯度。

作者表示，这种设计不仅可以使预测图像接近 GT，同时还有助于保持局部结构的相似。

除此之外，[1] 同样给出了 SN 和 semantic labels 的 LOSS 表示：对于 SN，作者计算每个像素处的点积（predicted and GT），并对整体图像求平均。对于 Semantic 部分，作者使用 softmax 分类器进行逐项素预测，不同 channels 之间使用 cross-entropy 给出最后预测。数据在监督学习中起着重要的作用,David 的工作是在 NYU-Depth 上训练的。Wang et al.[8] 提出了一种结构来进行 normal estimation. 不同的是, 他们考虑了环境特点（man-made, Manhattan World）和中间表述（房间布局等），并提出一种全局和局部结合的网络，并使用融合策略统一他们。[16] 提出了一种在 Physically-Based Rendering 的渲染方法，并利用 SUNCG [13] 产生很多 synthetic 数据, 作者使用这些数据去训练网络，并通过 NYU 数据做 finetuning 之后，效果更好。这种渲染方法被说成是以后实时渲染的趋势，我不太了解，还是回到数据集和模型。这篇文章的框架是比较简单，使用 VGG16 提取特征，然后是全卷积网络进行 normal 估计，Loss 则是沿用了 David Eigen 的设计, 训练方法是 RMSprop。同时这个工作也涉及到目标边界检测, 并取得不错的效果。接着这个工作，作者发表了为 RGB-D 数据进行深度补全（deep completion）的论文 [15]。论文首先是利用之前工作 [13]，通过单帧图像获得 SN 和物体边界（occlusion boundaries），然后通过一个优化函数，可以训练 SN，OB 与深度之间的关系。作者指出，对整体图像进行训练比仅训练洞处的像素效果更好。

同样使用 FCN 结构的还有，Iro [6] 的论文，今天还在 chair 遇见她。这是她和 Christian 完成的工作。Christian 已经去 VGG 做 PostDoc 了，接

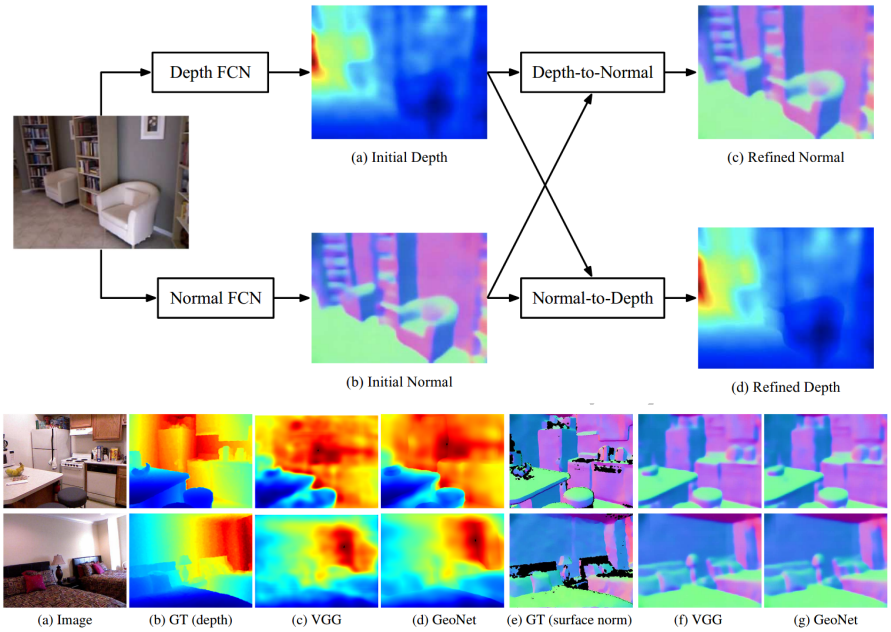


图 2: Geo-Net [11] 的网络框架与训练结果。

触比较少，但是很喜欢他的安静与真诚。他们的工作是使用 ResNet-50 来提取特征，并且获得了很好的效果。后来，Keisuke 和 Iro 连接了 Iro-Net and LSD-SLAM 一起做了 CNN-SLAM [14], 虽然网络可以很好的估计深度图，但是计算速度却达不到 real-time，因此 CNN-SLAM 只对 keyframe 做深度估计。

上面这些工作确实是比较久远了，网络结构没有那么好，显得比较笨重。我们来看一下比较轻量级的网络：mobileNet V2 [12].mobileNetV2 提供了一个语义分割的框架。论文指出在与其他网略表现相近的情况下，mobileNet V2 的速度和超参的数量有很大的优势。而原因主要是，该网络将标准卷积分解成一个深度卷积和一个点卷积（1*1)。对于实时性要求的 SLAM 领域，这个网络结构具有较大的意义。论文 [12] 指出，联合 DeepLabv3 和 MNetV2 feature map 可以较 MNetV1 减少参数，并保持相似表现。这篇文章没有估计深度图，但是网络结构可以有很好的借鉴意义。Geo-Net [11] 和 Deep ordinal [4] 是朋友推荐的两篇，第一篇是港中文 Prof. Jiaya Jia 的团队，第二篇是悉尼大学的 Prof. Dacheng Tao。GeoNet 考虑了 depth map 与 normals 之间的紧密关系，建立两个 steam 的网络结构。

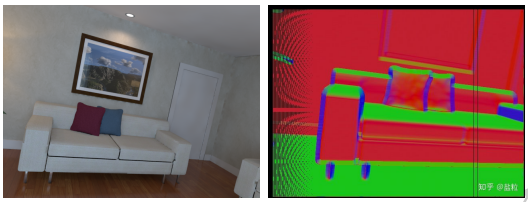


图 3: 制作数据集

3.3.2 动手训练一个法向量预测网络

在 David Eigen 的 depth prediction,surface normals predition 的论文网页中，给出了 NYU depth 的 normal 训练集，场景也比较丰富，不过数据量太少了。

制作数据集 这里是我们自己做的训练集合，使用 ICL 的深度图来制作自己的法向量数据集。

```
void Frame::EstimateSurfaceNormalGradient(const
    cv::Mat &imDepth, const double &timeStamp,cv::
    Mat &K)
{
    //ofstream surfaceNormal;
    //surfaceNormal.open("surfaceN.txt", ios::out
        | ios::trunc );
    //PointCloud::Ptr tmp( new PointCloud() );
    vector<SurfaceNormal> surfaceNormalVector;

    int cellsize=10;
    cv::Mat mPointCLoud=cv::Mat::zeros(imDepth.
        rows,imDepth.cols,CV_32FC3);//x,y,z

    for ( int m=0; m<imDepth.rows; m++ )
    {
        for ( int n=0; n<imDepth.cols; n++ )
        {
```

```
                float d = imDepth.ptr<float>(m)[n];
    if(d<0.001){ continue;}                mPointCLoud.at
                <cv::Vec3f>(m,n)[2]=d;

                mPointCLoud.at<cv::Vec3f>(m,n)[0]= (n
                    - K.at<float>(0,2))*d / K.at<
                    float>(0,0);

                mPointCLoud.at<cv::Vec3f>(m,n)[1]= (m
                    - K.at<float>(1,2))*d / K.at<
                    float>(1,0);
            }
        }

        //读取中心点 和周围的四个点
        //创建Mat
        cv::Mat mTangeMask=cv::Mat::zeros(imDepth.
            rows,imDepth.cols,CV_8U);
        cv::Mat tangeMaskIntegralTemp;cv:: Mat
            mUTangeMapIntegralTemp;Mat
            mVTangeMapIntegralTemp;

        cv::Mat tangeMaskIntegral;cv:: Mat
            mUTangeMapIntegral;Mat mVTangeMapIntegral;

        cv::Mat mUTangeMap=cv::Mat::zeros(imDepth.
            rows,imDepth.cols,CV_32FC3);
        cv::Mat mVTangeMap=cv::Mat::zeros(imDepth.
            rows,imDepth.cols,CV_32FC3);
        for ( int m=1; m<imDepth.rows-1; m++ )
        {
            for ( int n=1; n<imDepth.cols-1; n++ )
            {
```

```
            //根据深度，获取有效点
    }     }
        //求intergral
        integral(mTangeMask,tangeMaskIntegralTemp);
        integral(mUTangeMap,mUTangeMapIntegralTemp,
            CV_32FC3);

        integral(mVTangeMap,mVTangeMapIntegralTemp,
            CV_32FC3);
        //去掉积分图的padding

        tangeMaskIntegralTemp(cv::Range(1,imDepth.
            rows-3), cv::Range(1,imDepth.cols-3)).
            copyTo(tangeMaskIntegral);
        mUTangeMapIntegralTemp(cv::Range(1,imDepth.
            rows-3), cv::Range(1,imDepth.cols-3)).
            copyTo(mUTangeMapIntegral);
        mVTangeMapIntegralTemp(cv::Range(1,imDepth.
            rows-3), cv::Range(1,imDepth.cols-3)).
            copyTo(mVTangeMapIntegral);

        int cellCount=0;
        for (int v=cellsize+1; v<imDepth.rows-3;v++
            )
        {
        //处理每个cell
        }
        vSurfaceNormal=surfaceNormalVector;
    }
```

完成一个神经网络 其实我的工作不是一个语义分割的任务，由于目前正在准备 IROS2019，如果后面能接受，我在详细的介绍。训练集和测试集是相同 size 的图片，一个是 RGB，一个是 mask。

训练集是 RGB 图像和 mask, 把图像尺寸压缩成 160× 160 大小。

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456,
        0.406], std=[0.229, 0.224, 0.225])])

class BagDataset(Dataset):

    def __init__(self, transform=None):
        self.transform = transform

    def __len__(self):
        return len(os.listdir('bag_data'))

    def __getitem__(self, idx):
        img_name = os.listdir('bag_data')[idx]
        imgA = cv2.imread('bag_data/'+img_name)
        imgA = cv2.resize(imgA, (160, 160))
        imgB = cv2.imread('bag_data_msk/'+
            img_name, 0)
        imgB = cv2.resize(imgB, (160, 160))
        imgB = imgB/255
        imgB = imgB.astype('uint8')
        imgB = onehot(imgB, 2)
        imgB = imgB.transpose(2,0,1)
        imgB = torch.FloatTensor(imgB)
        #print(imgB.shape)
        if self.transform:
            imgA = self.transform(imgA)

        return imgA, imgB

bag = BagDataset(transform)
```

```
train_size = int(0.9 * len(bag))test_size = len(
bag) - train_size
train_dataset, test_dataset = random_split(bag, [
    train_size, test_size])

train_dataloader = DataLoader(train_dataset,
    batch_size=4, shuffle=True, num_workers=4)
test_dataloader = DataLoader(test_dataset,
    batch_size=4, shuffle=True, num_workers=4)
```

在开始网络结构之前，还是要回顾一下 feature map 的尺寸是如何变化的，我们需要保持两段 size 的一致。

$$Out_{height} = \frac{Input_{height} - kernel + 2 \times padiing}{stride} + 1 \tag{3.1}$$

我们还是使用预训练好的 VGG16 做为卷积层，这里要输出五个 size 的 feature map, 用来反卷积回来。

```
class VGG16(nn.Module):
    def __init__(self):
        super(VGG16, self).__init__()
        self.conv1_1 = nn.Conv2d(3, 64, 3,
            padding=100)
        self.relu1_1 = nn.ReLU(inplace=True)
        self.conv1_2 = nn.Conv2d(64, 64, 3,
            padding=1)
        self.relu1_2 = nn.ReLU(inplace=True)
        self.pool1 = nn.MaxPool2d(2, stride=2,
            ceil_mode=True) # 1/2

class FCNs(nn.Module):

    def __init__(self, pretrained_net, n_class):
        super().__init__()
```

```
        self.n_class = n_class
    self.pretrained_net = pretrained_net        self.
        relu    = nn.ReLU(inplace=True)
        self.deconv1 = nn.ConvTranspose2d(512,
            512, kernel_size=3, stride=2, padding
            =1, dilation=1, output_padding=1)
        self.bn1     = nn.BatchNorm2d(512)
        self.deconv2 = nn.ConvTranspose2d(512,
            256, kernel_size=3, stride=2, padding
            =1, dilation=1, output_padding=1)
        self.bn2     = nn.BatchNorm2d(256)
        self.deconv3 = nn.ConvTranspose2d(256,
            128, kernel_size=3, stride=2, padding
            =1, dilation=1, output_padding=1)
        self.bn3     = nn.BatchNorm2d(128)
        self.deconv4 = nn.ConvTranspose2d(128,
            64, kernel_size=3, stride=2, padding
            =1, dilation=1, output_padding=1)
        self.bn4     = nn.BatchNorm2d(64)
        self.deconv5 = nn.ConvTranspose2d(64, 32,
             kernel_size=3, stride=2, padding=1,
            dilation=1, output_padding=1)
        self.bn5     = nn.BatchNorm2d(32)
        self.classifier = nn.Conv2d(32, n_class,
            kernel_size=1)
        # classifier is 1x1 conv, to reduce
            channels from 32 to n_class

    def forward(self, x):
        output = self.pretrained_net(x)
        x5 = output['x5']
        x4 = output['x4']
        x3 = output['x3']
```

```
            x2 = output['x2']
        x1 = output['x1']
            score = self.bn1(self.relu(self.deconv1(
                x5)))
            score = score + x4
            score = self.bn2(self.relu(self.deconv2(
                score)))
            score = score + x3
            score = self.bn3(self.relu(self.deconv3(
                score)))
            score = score + x2
            score = self.bn4(self.relu(self.deconv4(
                score)))
            score = score + x1
            score = self.bn5(self.relu(self.deconv5(
                score)))
            score = self.classifier(score)
            return score
```

```
def train(epo_num=50, show_vgg_params=False):

    device = torch.device('cuda' if torch.cuda.
        is_available() else 'cpu')

    vgg_model = VGGNet(requires_grad=True,
        show_params=show_vgg_params)
    fcn_model = FCNs(pretrained_net=vgg_model,
        n_class=2)
    fcn_model = fcn_model.to(device)
    criterion = nn.BCELoss().to(device)
    optimizer = optim.SGD(fcn_model.parameters(),
        lr=1e-2, momentum=0.7)
```

```
    all_train_iter_loss = []
all_test_iter_loss = []
    # start timing
    prev_time = datetime.now()
    for epo in range(epo_num):

        train_loss = 0
        fcn_model.train()
        for index, (bag, bag_msk) in enumerate(
            train_dataloader):

            bag = bag.to(device)
            bag_msk = bag_msk.to(device)

            optimizer.zero_grad()
            output = fcn_model(bag)
            output = torch.sigmoid(output) #
                output.shape is torch.Size([4, 2,
                160, 160])
            loss = criterion(output, bag_msk)
            loss.backward()
            iter_loss = loss.item()
            all_train_iter_loss.append(iter_loss)
            train_loss += iter_loss
            optimizer.step()

            output_np = output.cpu().detach().
                numpy().copy() # output_np.shape =
                 (4, 2, 160, 160)
            output_np = np.argmin(output_np, axis
                =1)
            bag_msk_np = bag_msk.cpu().detach().
                numpy().copy() # bag_msk_np.shape
```

```
                = (4, 2, 160, 160)
bag_msk_np = np.argmin(bag_msk_np, axis=1)
            if np.mod(index, 15) == 0:
                print('epoch {}, {}/{},train loss
                    is {}'.format(epo, index, len
                    (train_dataloader), iter_loss)
                    )
                                # plt.subplot(1, 2, 1)
            # plt.imshow(np.squeeze(bag_msk_np[0,
                ...]), 'gray')
            # plt.subplot(1, 2, 2)
            # plt.imshow(np.squeeze(output_np[0,
                ...]), 'gray')
# plt.pause(0.5)
```

3.3.3 无监督学习

monodepth 和 monodepthv2。

4 相机模型

5 位姿估计的模型

6 地图创建

7 非线性优化

前面的章节介绍了特征的提取匹配、相机姿态求解和全局地图的构建，但是他们获得的相机姿态和路标点存在误差，并且误差会随着时间的增加逐渐积累。因此本章介绍光束法平差模型对上述信息进行优化，以获得准确的相机姿态求解和全局地图，平差模型解算结果会直接影响机器人、无人机所提供服务的质量。首先，我们描述基于点线面特征的光束法平差模型，然后介绍光束法平差模型常用的解算方法。

7.1 光束法平差模型分析

光束法平差（bundle adjustment，BA）是三维重建和视觉定位中非常关键的一步，在获得相机内参数、外参、影像特征点和空间 3D 物点坐标初始值之后，可以使用 BA 对包括相机的内参和外参在内的上述数据进行整体的优化，消除误差并获得精确的视觉信息。

光束法平差被认为是摄影测量和计算机视觉中的黄金定律，而数学本质上来说，光束法平差目标方程模型就是非线性最小二乘模型，并且是关于相机姿态 (ξ) 与三维路标信息 ($Landmark$) 的最小二乘优化问题。相机姿态包含平移矩阵 R 和旋转矩阵 t，而常用的三维路标信息有点 P、线 L 和面 Q。

7.1.1 点特征的误差方程

一组同名像点对应着一个空中的三维点 $P_j = (x, y, z)$，通过计算三维点在影像上的重投影与对应的 2D 特征之间的距离，我们可以获得点的冲投影误差方程，

$$r_{k,j}^{p} = p_k - \pi(R_{k,j}P_j + t_{k,j}) \tag{7.1}$$

公式中的 $\pi()$ 就是投影过程，f_x 与 f_y 则是相机的内参信息。在优化过程中，误差方程对于相机位姿的的 Jacobian 矩阵可以表示为，

$$\frac{\partial e_{k,j}^{p}}{\partial \xi} = \begin{bmatrix} \frac{xyf_x}{z^2} & -\frac{z^2+x^2}{z^2}f_x & \frac{yf_x}{z} & -\frac{f_x}{z} & 0 & \frac{xf_x}{z^2} \\ \frac{z^2+y^2}{z^2}f_y & -\frac{xyf_y}{z^2} & -\frac{xf_y}{z} & 0 & -\frac{f_y}{z} & \frac{yf_y}{z^2} \end{bmatrix} \tag{7.2}$$

7.1.2 线特征的误差方程

与特征点相比，直线可以被表示为两个端点 p_{start} 和 p_{end}。首先我们利用这两个端点计算直线的方程，通过单位化处理，可以显出直线长度带来的影响。

$$l = \frac{p_{start} \times p_{end}}{\|p_{start}\|\|p_{end}\|} = (a, b, c) \tag{7.3}$$

类似与点的重投影误差，我们来计算线特征的误差方程。在 2D 图像上获得直线方程之后，通过将空间 3D 直线的端点（P_{start} 和 P_{end}）重投影回来。对于每一个端点 $P_x, x \in [start, end]$, 误差方程可以被描述为:

$$e^l_{k,j} = l\pi(R_{k,j}P_x + t_{k,j}) \tag{7.4}$$

同理，上述误差方程带来的 Jacobian 矩阵为

$$\frac{\partial e^l_{k,j}}{\partial \xi} = \begin{bmatrix} -\frac{f_y^2 z^2 + axyf_x + by^2 f_y}{z^2}, & \frac{az^2 f_x + ax^2 f_x + bxyf_y}{z^2}, & -\frac{ayf_x - bxf_y}{z}, \\ \frac{af_x}{z}, & \frac{bf_y}{z}, & -\frac{axf_x + byf_y}{z^2} \end{bmatrix} \tag{7.5}$$

7.1.3 面特征的误差方程

平面信息可以在人造环境中随处可见，常用的平面表示方式是海森表示 $n_\pi P + d = 0$, 这种方式可以很直观的平面信息，但是存在过参的问题，因此我们使用 $q(\pi) = (\phi, \psi, d)$ 来描述平面，这里 ϕ 和 ψ 是平面法向量的 azimuth 和 elevation，d 表示点到平面的距离。

$$q(\pi) = (\phi = arctan(\frac{n_y}{n_x}), \psi = arcsin(n_z), d). \tag{7.6}$$

所以，我们可以描述两个匹配平面（π_k 和 π_x）之间的误差方程，

$$e^\pi_{k,\pi_x} = q(\pi_k) - q(T^{-T}_{cw}\pi_x) \tag{7.7}$$

这里 $T^{-T}_{cw} = (R_{cw}, t_{cw})$ 是从世界坐标系到相机坐标系下的转化矩阵，R_{cw} 用来旋转平面的法向量，t_{cw} 用来处理点到平面的距离。

同理，我们可以获得面误差方程(7.7)带来的 Jacobian 矩阵，

$$\frac{\partial e^\pi_{k,\pi_x}}{\partial \xi} = \begin{bmatrix} \frac{n_{cx}n_{cz}}{n_{cx}^2+n_{cy}^2} & \frac{n_{cy}n_{cz}}{n_{cx}^2+n_{cy}^2} & -1 & 0 & 0 & 0 \\ \frac{n_{cy}}{\sqrt{1-n_{cz}^2}} & \frac{n_{cx}}{\sqrt{1-n_{cz}^2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & n_{cx} & n_{cy} & n_{cz} \end{bmatrix} \tag{7.8}$$

7.1.4 构建光束法平差模型

$$
\begin{aligned}
t* = argmin \sum_{j}^{M} \rho_p \left({e_{k,j}^{p}}^T \Lambda_{p_{k,j}} e_{k,j}^{p} \right) \\
+ \rho_l \left({e_{k,P_x}^{l}}^T \Lambda_{p_{k,P_x}} e_{k,P_x}^{l} \right) + \rho_\pi \left({e_{k,\pi_x}^{\pi}}^T \Lambda_{k,\pi_x} e_{k,\pi_x}^{\pi} \right)
\end{aligned} \tag{7.9}
$$

在公式7.9中，Λ 是一个协方差矩阵，同一个传感器观场景进行观测的时候，我们认为误差之间是相互独立的的，故通常作为一个对角阵存在，在实际的工作中通常取单位阵。通过对目标方程的解算，我们最终可以获得优化的结果。

7.2 经典的光束法平差解算方法

在非线性优化问题的诸多解法中，牛顿法因为具备二阶收敛速度而得到广泛的应用，但是在迭代求解的过程中，牛顿法因计算复杂度过高、海森矩阵稠密等问题，无法应用在实际的光束法平差问题中。因此，我们要有光束法平差问题的专门解算方法。

在光束法平差模型解算放法中，高斯牛顿（Gauss-Newton，GN）方法通过忽略牛顿方法中产生的高阶导数很好的降低了算法的复杂度，作为一种具有快速收敛性的收敛方法，被广泛用来解算光束法平差中，获得精准的相机位姿信息和 3D 特征点信息。但是 GN 对初值的精度要求较高，在初值不好的情况下，将面临发散的危险。列温伯格-马夸尔特 (Leverberg-Marquart，LM) 方法通过增加阻尼项，保持了收敛的鲁棒性，但是这种方法使用置信区间来尝试阻尼的值，不能准确的获得下降方向，因此需要多次迭代才能完成收敛。这两种方法是最广泛使用的解算方法，除此之外还有共轭梯度法、拟牛顿等解算方法。

7.2.1 高斯牛顿方法

高斯牛顿方法从牛顿方法简化而来，因此在剖析高斯牛顿方法发散原因之前，本文首先介绍牛顿优化方法。通过牛顿法的介绍，可以更加容易理解高斯牛顿发散的数学机理。在数值优化领域，牛顿法因具有较快的收敛速度，被用来求解无约束优化问题。这个方法的主要思路是在迭代点处进行泰勒展开，使用展开式对目标方程进行近似。因此非线性问题就可以转化为线性问题，并使用求二次模型的极小点的方式来获取新的迭代点。通过多次迭

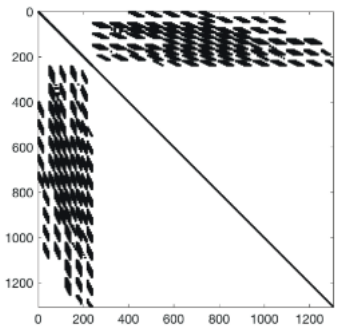


图 4: 海森矩阵的对称稀疏结构。

代，达到满足的精度为止。对公式7.9进行一阶泰勒展开, 这里我们只展开公式中特征点的部分

$$z(x) = f(x_{k+1}) + g_k(x - x_{k+1}) + \frac{(x - x_{k+1})^T G(x - x_{k+1})}{2} \tag{7.10}$$

这里 $G = \Delta^2 f(x_{k+1})$, $g_{k+1} = \Delta f(x_{k+1})$。对于高斯解法，通过对上述公式求导，并令导数为零，就可以获得极值点，

$$\Delta z(x) = \Delta f(f(x_{k+1})) + G(x - x_{k+1}) = 0 \tag{7.11}$$

如果 G^{-1} 是非奇异阵，那么 $x_{k+1} = x_k - G^{-1} g_{k+1}$，这样就可以实现迭代更新。在之前已经描述过了，牛顿法具有不低于二阶的收敛速度，但是该方法在遇到 BA 问题时，求解 Hessian 矩阵工作量巨大，同时不能保证目标函数的 Hessian 矩阵在每个迭代点处都能保持正定。为了利用牛顿法的优势，避免不足，出现了 Gauss-Newton 等 BA 解法。

正常情况下 $Hd = J(x)^T F(x)$,其中 H 为 Hessian 矩阵,$H = J(x)^T J(x) + \sum F_i(x)^2$。在 Gauss-Newton 方法中，忽略公式的第二项，使用 $J(x)^T J(x)$ 来代替 Hesse 矩阵，求解方程的下降方向。因此，可以看出缺少高阶导数信息的高斯牛顿算法在迭代求解过程中必须满足 Hessian 矩阵正定的条件，进而要求 Jacobian 矩阵必须是列满秩，不然算法失效。而 Jacobian 矩阵是相机与三维特征点之间的表示方式是随着输入参数的变化而变化的。因此对 Jacobian 矩阵的要求就变成了对观测初值的要求，这就是高斯牛顿方法对初值敏感的根本原因。

值得说明的是，虽然高斯牛顿法不是非常的鲁棒，但是中对称稀疏的海森矩阵形式（如图4），有利的推动了光束法平差模型的使用，因为这种稀疏

Algotithm 1: Guass-Newton

Input: A vector function f: R^m ->R^n n>m,
A measure vector $x \epsilon R^n$ 初值 P_0
Output: a vector P^+ =argmin$\|f(p)\|^2$ $f(p) = x - r(p)$
K =0 $A = J^T Jg = J^T f(p)$ $B_0 = A$

$$Ad_k^{GN} = -J_k^T f(x_k)$$
$$\partial = 1$$
$$x := x + d_k^{GN}$$

图 5: 高斯牛顿法的伪码算法过程

结构可以通过矩阵知识很快速的被计算出来，即使是面对数万维度大小的矩阵。

7.2.2 Leverberg-Marquart

有上面内容可知，对于 GN 方法，使用 $J^T J$ 矩阵近似 Hessian 矩阵很容易发生奇异，因此，光束法平差问题常常无法解算。为了克服 GN 方法的缺点，LM 方法被提出来，并作为应用最为广泛的方法求解 BA 模型。LM 算法可以表示为公式

$$(J^T J + \lambda I) d_k^{LM} = -J^T g_k \tag{7.12}$$

其中 I 矩阵表示单位阵,LM 算法的原理是在近似 Hessian 矩阵上的对角线上加上一个阻尼值，从而近似 Hessian 矩阵的奇异问题。通过观察公式7.12，LM 方法是一种结构化的方法，其中包含两个部分：$J^T J$ 与 λI，可以通过调整 λ 的大小，调节 $J^T J$ 与 λ 之间的比重。当 λ 特别大时，$J^T J$ 可以忽略，即公式7.12可以变成梯度下降方法公式：

$$d_k^{LM} = -\frac{1}{\lambda} J^T g_k \tag{7.13}$$

当 λ 特别小时候，λI 可以忽略，求解方法即变成 GN 方法。因此可以把 LM 方法看做是最速梯度下降法和高斯牛顿法的结合。使用置信区间来更新 λ，

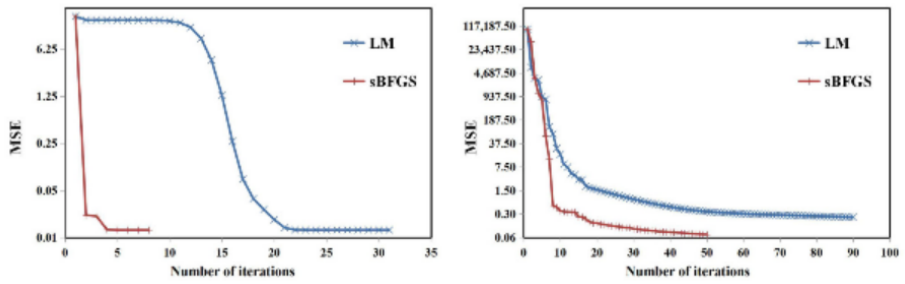


图 6: 两种方法收敛结果。其中红色稀疏结构化的 BFGS 方法的收敛结果，蓝色为 LM 法收敛结果。

如果当前次迭代可以得到比前一次迭代更小的优化结果，此时公式7.12就需要减小 λ 的大小（$\lambda = \lambda \cdot t,\ t = max(\rho, \frac{1}{3})$）；假如不满足条件时，需要增大 λ 值（$\lambda = \lambda \cdot v,\ v = 2v$）。

进入的优化环节的初值情况是无法控制的，与高斯牛顿的初值敏感特点一样，LM 方法对于初值较差的情况，LM 同样是被动的调整阻尼值，不能根据当前迭代的实际情况来完成阻尼的更新，因此 LM 方法常常在需要较多次的迭代才能完成。从图6的实验结果可以看出，LM 迭代方向并不是稳定的，会不会在迭代初期经历近似平原区域，还要看光束平差模型中优化的输入数据。

7.2.3 稀疏结构化 BFGS 方法

在上述篇幅分析了常见的解算方法：高斯牛顿方法和 LM 方法，清楚的把这两种方法的优势与缺陷，及其成因展现在读者眼中。针对上述问题，稀疏结构化 BFGS（sBFGS）[7] 通过探索海森矩阵中高阶信息的保留方式，提出一个保持更好下降方向的解算方法。

BFGS 正定保持特性 BFGS 是一种常用的拟牛顿方法，由于该方法经过秩 2 修正，因此在迭代求解过程中，对于输入的正定矩阵，在满足情况的条件下，能保持正定性。

$$A_{k+1} = A_k - \frac{A_k r_k r_k^T A_k}{r_k^T A_k r_k} + \frac{z_k z_k^T}{z_k^T r_k} \quad (z_k^T r_k > 0) \tag{7.14}$$

其中，$r_k = X_{k+1} - X_k$ 是两次迭代之间的残差向量。另外，z_k 是雅克比矩阵的差，$z_k = (J_{k+1}^T - J_k^T)||r_{k+1}||$。

结构化的 BFGS 方法 在上面的分析中我们明白高斯牛顿方法无法保持矩阵正定，现在我们为其加入一个正定矩阵，使得在任何时候都能够保持正定，这样可以解算任意情况的初值。如下式所示：

$$B_{k+1} = J_{k+1}^T J_{k+1} + A_{k+1} \tag{7.15}$$

在本方法中通过 BFGS 算法来更新，具体的证明过程可以参考优化书籍，不再赘述。

由于 BFGS 对于正定性质的保持能力，假设为正定矩阵，使用 BFGS 算法更新时，只要，并且那么可以保障的正定性，进而为正定矩阵，保证了下降方向始终是正确的。

$$A_{k+1} = \begin{cases} A_k - \dfrac{A_k s_k s_k^T A_k}{s_k^T A_k s_k} + \dfrac{z_k z_k^T}{z_k^T s_k} & if(z_k^T s_k > \upsilon) \\ A_k & otherwise \end{cases} \tag{7.16}$$

因此上面的式子中，使用 A_{k+1} 近似 Hessian 矩阵中的含有高阶导的部分，我们通过 BFGS 方法来更好的模拟 Hessian 矩阵。通过公式 7.16获得的 A_{k+1} 是一个密集矩阵，由于 BA 问题涉及到的矩阵的维数非常大，因此对于密集矩阵的求解将会非常困难。因此本文根据 Hessian 矩阵的低阶导数信息的稀疏化高阶导数矩阵，因此可以实现稀疏化的 A_{k+1}，记做 A_{k+1}^s。

$$B_k^s = \begin{cases} J_{k+1}^T J_{k+1} + A_{k+1}^s & if(z_k^T s_k > \upsilon) \\ J_{k+1}^T J_{k+1} + ||r_k||I & otherwise \end{cases} \tag{7.17}$$

如公式 7.18所示，可以通过上面的稀疏化的 B_k^s 矩阵，在求解下降方向以后，可以顺利的对光束法平差问题进行解算。与 LM 方法相比，本方法在迭代点远离真值时，依然能保持超线性收敛速度，随着迭代的进行，当前迭代点逐渐接近真值，本方法中的高阶导数矩阵的作用就比较微弱，本方法以高斯牛顿方法进行收敛，同样保持超线性收敛速度。

$$B_k^s d_k^{sBFGS} = -J^T g_k \tag{7.18}$$

其中 d_k^{sBFGS} 表示 sBFGS 方法下降方向。

8 优化中的位姿描述

8.1 四元素

8.2 李群与李代数

9 优化中的特征描述

9.1 点特征-逆深度

9.2 线特征-正交表示

10 经典的纯几何 SLAM

10.1 ORBSLAM (V1 and V2)

ORB-SLAM2 [10] 是由 Raúl Mur-Artal 博士等人提出的一个完备的基于特征点的纯视觉实时 SLAM 系统，适用于单目、双目和 RGB-D 相机，发表于 2017 年。作者在 2015 年第一次将单目版本（ORB-SLAM [9]）发表在 IEEE Transactions on Robotics（ToR）期刊上，这是一个基于纯特征点的单目实时 SLAM 框架。总体来说，该 ORB-SLAM 的两篇文章是基于特征的 SLAM 系统的大成之作，系统本身非常的完整，包含了稀疏地图，回环和优化环节。ORBSLAM2 将传感器从单目扩展到双目和 RGB-D，成为了迄今为止影响力最大的特征法 SLAM 系统之一，为 SLAM 在工业界和学术界的发展作出巨大贡献。

在特征匹配、跟踪和回环检测等所有环节全部采用统一的 ORB 描述子，通过地图复用，有效提高了系统的鲁棒性。该系统分为四个模块，分别是前端、局部建图和回环检测以及闭环，系统使用 3 个线程。。下面我们着重介绍 ORB-SLAM2 中重要的几个模块。

10.1.1 单目系统的初始化

系统有两个初始化模块，双目和 RGB-D 共用一个模块，单目有自己的一个初始化模块，在追踪开始后，根据输入图像的样式，进去相应的初始化程序。相比于双目和 RGB-D，单目不具有尺度，因此更加的复杂，这里我们介绍单目的初始化过程。首先是对连续的两帧（F_i 和 F_{i+1}）提取特征点，

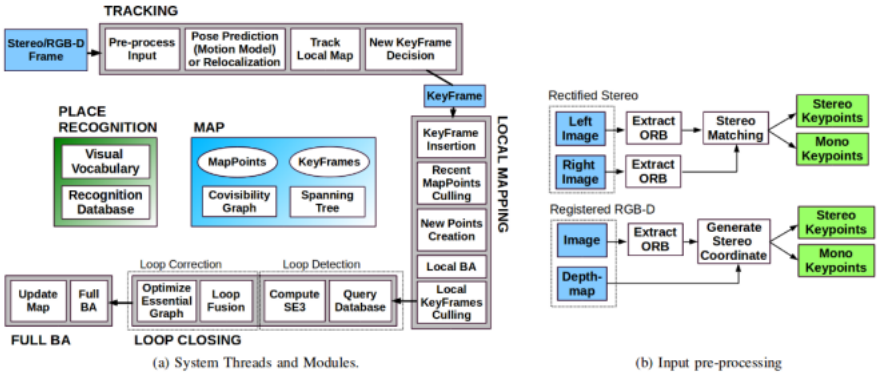


图 7: ORB-SLAM2 系统框图。[10]

如果他们满足特征匹配数量的约束。就进一步计算两帧之间的位姿。如果，不是连续的两帧那么就重新开始。

为了提升初始化的精度，作者同时估计两帧图像之间的单应矩阵（Homograph）和基础矩阵（Fundamental），单应矩阵可以用来处理平面之间/较小视差角的旋转与平移，基础矩阵则能处理更普遍的特征点。对于两帧图像之间的同名点，系统同时估计上述的两个矩阵 H_{cr} 和 $F(cr)$, 其中 c 和 r 分别表示当前帧和参考帧。对于同名点随机选择八个点，并建立很多组。然后通过公式10.1将这两个矩阵分别将一帧图像中同名点转换到另一帧图像上，来计算同名点在每帧图像上的像素距离。

$$S_M = \sum_i (\rho_M(d^2_{cr}(x^i_c, x^i_r, M)) + \rho_M(d^2_{rc}(x^i_c, x^i_r, M))) \tag{10.1}$$

这里 i 表示同名点的数量，M 为 H_{cr} 或 $F(cr)$ 矩阵中的一个。最后，利用公式10.2来比较两个误差的大小，并选择一个矩阵。

$$R_H = \frac{S_H}{S_H + S_F} \tag{10.2}$$

在 ORB-SLAM 中，如果 $R_H > 0.045$ 选择单应矩阵，否则选择基础矩阵来初始化两帧图像之间的位姿。在完成位姿之后，系统利用该信息和同名点信息进行路标点的三角化（Triangulation）工作，创建稀疏地图。

10.1.2 姿态估计与优化

对于相机姿态的估计是 SLAM 系统的一个重要任务,这个环节为 AR/VR, 机器人和无人驾驶等应用提供支持。在 ORB-SLAM2 中，利用相机基线来实现 RGB-D 向双目相机的转化。

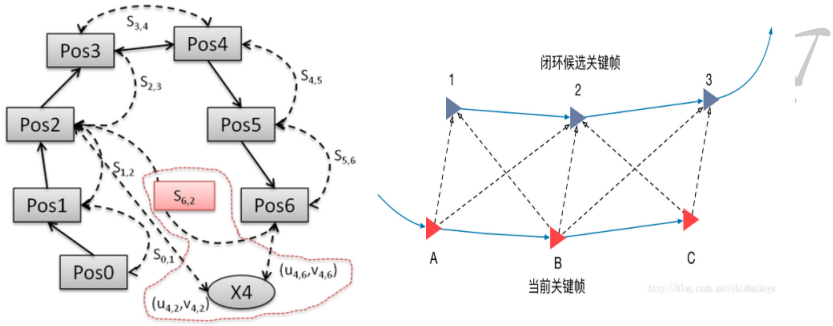
- 系统在 2D 特征提取中为了使特征分布更加均匀，将图像分块提取特征。对于双目和 RGB-D 相机，系统可以直接获得特征点的 3D 信息，而单目相机需要在后面进行三角化操作。
- 完成当前帧的特征提取之后，系统通过追踪参考帧或使用运动模型的方式获得当前帧的初始 6D 位姿。两个方法都是先给当前相机的位姿赋一个值，然后再用重投影误差去优化。然后使用局部地图再优化一遍，局部建图模块是利用前端输入的关键帧插入局部地图进行局部优化。
- 如果检测出来是关键帧，另一个线程基于共视图的优化。

ORB-SLAM 的这种相机位姿估计策略非常鲁棒，尤其是当检测的特征点充足时，可以实现高精度的位姿估计。

10.1.3 基于词袋的闭环检测

对于基于 Frame-to-Frame 和 Map-to-Frame 的 SLAM 系统，随着追踪时间的增长，系统也在不断积累着误差，这种现象被称之为系统漂移 (drift)。回环检测（Loop Closure）是一种消除漂移的有效的解决方案。闭环检测是在历史轨迹中查找是否存在与新关键帧具有相同观测的和约束的关键帧，筛选过程主要包括位置识别和几何验证。

在 ORBSLAM2 中，闭环检测线程通过词袋法 (DBoW2) 加速闭环匹配帧的筛选。这种方法的目的是用一种更加抽象、更加高效的特征集合来描述一幅图像。通过 DBoW2, 作者训练了一个较大的特征字典。对于每一帧图像中提取的特征，只要在字典树中逐层查找，最后都能找到与之对应的单词。最后可以使用一个分布或直方图来准确的描述这个图像。从图8 (a) 中可以看到 Pose2 和 Pose6 之间的共同观测。并通过共同观测计算两帧图像之间的旋转矩阵和平移矩阵，对于单目相机，还要通过 Sim3 优化尺度。然后在上一步求得的 Sim3 和对应同名点的基础上，纠正了当前帧以及所有有同视关系的关键帧的位姿，以及路标点。



(a) (b)

图 8: 闭环检测检测

10.2 LSD-SLAM

LSD-SLAM(large-scale direct monocular SLAM) [3] 是发表在 ECCV2014 的直接法的 SLAM。与 ORB-SLAM 等所用的特征法不同，直接法视觉里程计（VO）是直接利用图像像素点的灰度信息来构图与定位，克服了特征点提取方法的局限性，可以使用图像上的所有信息。该方法在特征点稀少的环境下仍能达到很高的定位精度与鲁棒性，而且提供了更多的环境几何信息，这在机器人和增强现实额应用中都是非常有意义的。

这篇文章的第一作者 Jakob Engel 对直接法的发展作出了很大的贡献。本篇文章提出的方法能够构建大尺度的，全局一致性的环境地图。除了能够基于直接图像配准得到高度准确的姿态估计外，还能够将三维环境地图实时重构为关键帧的姿态图和对应的半稠密的深度图。这些都是通过对大量像素点对之间的基线立体配准结果滤波后得到的。算法提出了计算尺度漂移的公式，即便是当图像序列的场景尺度变化较大时也能够适用。

10.2.1 整体流程

算法有三个主要组成部分，分别为图像跟踪、深度图估计和地图优化。如图 9所示。

- 图像跟踪：连续跟踪从相机获取到的新“图像帧”。也就是说用前一帧图像帧作为初始姿态，估算出当前参考关键帧和新图像帧之间的刚体变换 $\xi \in se(3)$。

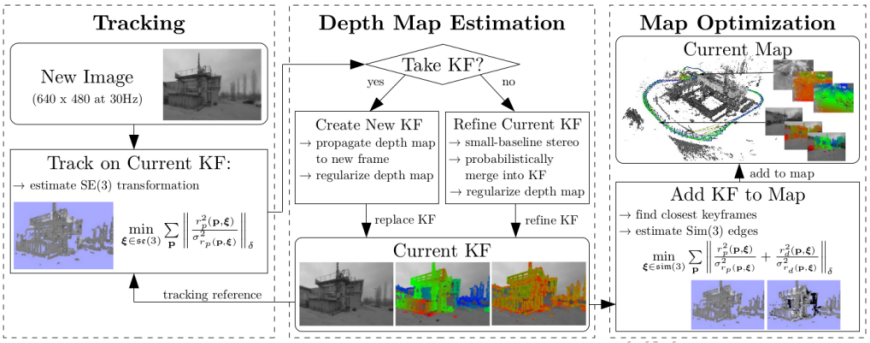


图 9: Pipeline of LSD-SLAM [3]

- 深度图估计：使用被跟踪的“图像帧”，要么对当前关键帧深度更新，要么替换当前关键帧。深度更新是基于像素小基线立体配准的滤波方式，同时耦合对深度地图的正则化。如果相机移动足够远，就初始化新的关键帧，并把现存相近的关键帧图像点投影到新建立的关键帧上。
- 地图优化模块：一旦关键帧被当前的图像替代，它的深度信息将不会再被进一步优化，而是通过地图优化模块插入到全局地图中。为了检测闭环和尺度漂移，采用尺度感知的直接图像配准方法来估计当前帧与现有邻近关键帧之间的相似性变换。

10.2.2 光度误差与深度误差

关键帧是 LSD-SLAM 系统中重要的媒介。启动 LSD-SLAM 系统时，只需要初始化首帧关键帧即可，而关键帧深度信息初步设定为一个方差很大的随机变量。在算法运行最开始的几秒钟，一旦摄像头运动了足够的平移量，LSD-SLAM 算法就会“锁定”到某个特定的深度配置，经过几个关键帧的传递之后，就会收敛到正确的深度配置。

$$E(\xi) = \sum_i (I_{ref}(p_i) - I(w(p_i, D_{ref}(p_i), \xi)))^2 \tag{10.3}$$

公式 10.3是当前帧 I 与参考帧 I_{ref} 之间的光度误差。其中 $w(p_i, D_{ref}(p_i), \xi)$ 是将这些像素重投影到参考帧 I_{ref}。一旦新图像帧被选择成为关键帧，把上一帧关键帧的兴趣点投影到新创建的关键帧上得到这一帧的兴趣点。之后，深度图被平均至逆深度为 1，就可以使用相似变换 sim(3) 来计算关键帧之间的边，因为相似变换 sim(3) 可以较好地结合关键帧之间的尺度缩放差异。

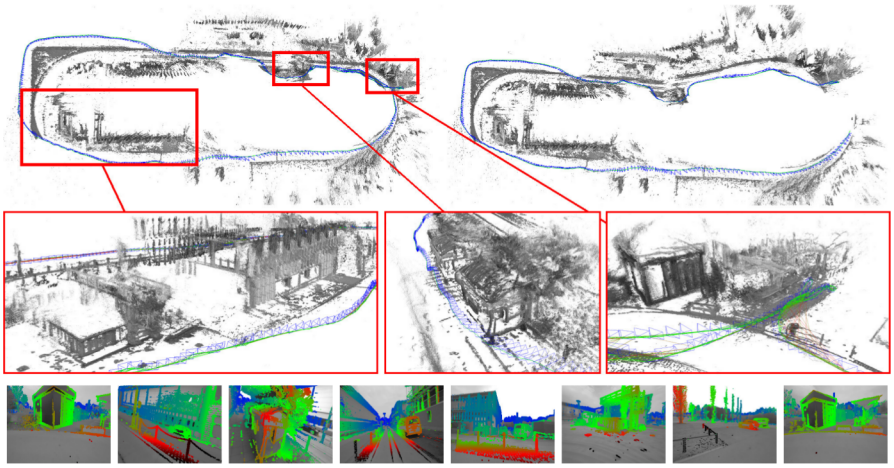


图 10: LSD-SLAM 中的闭环检测模块。左图是闭环检测的效果，右图是没有闭环检测。[3]

除了包含光度测量残差 r_p，该论文还引入深度残差（depth residual）r_d 来惩罚关键帧之间的逆深度偏差，能直接够估计出帧间的相似变换。LSD-SLAM 论文中使用的误差函数 $E(\xi_{ji})$，

$$E(\xi_{ji}) = \sum || \frac{r_p^2(p, \xi_{ji})}{\sigma^2_{r_p(p,\xi_{ji})}} + \frac{r_d^2(p, \xi_{ji})}{\sigma^2_{r_d(p,\xi_{ji})}} || \tag{10.4}$$

其中 r_d 是深度惨差，$\sigma_{r_p}^{-2}$ 和 $\sigma_{r_d}^{-2}$ 分别是光度残差和深度惨差的逆方差。

10.3 SVO

全称 fast Semi-direct monocular Visual Odometry（半直接视觉里程计），是苏黎世大学机器人感知组的克里斯蒂安. 弗斯特（Christian Forster 等人于 2014 年 ICRA 会议上发表的工作。2016 年扩展了多相机和 IMU 之后，写成期刊论文，称为 SVO 2.0。与基于特征法的 ORB-SLAM 和直接法的 LSD-SLAM 不同，SVO 提取了稀疏点，但是使用直接法来做图像之间的对齐，故称它为半直接法。高翔博士在 LDSO [5] 中同样使用特征点和光度，特征的主要作用是实现闭环检测，只不过现在半稠密的概念已经不如以往流行了。SVO 结合了直接法和特征点法，但是特征点只是从关键帧中提取，以保证系统的运行速度。SVO 算法分成两部分：位姿估计，深度估计。下文详细介绍这两个模块。

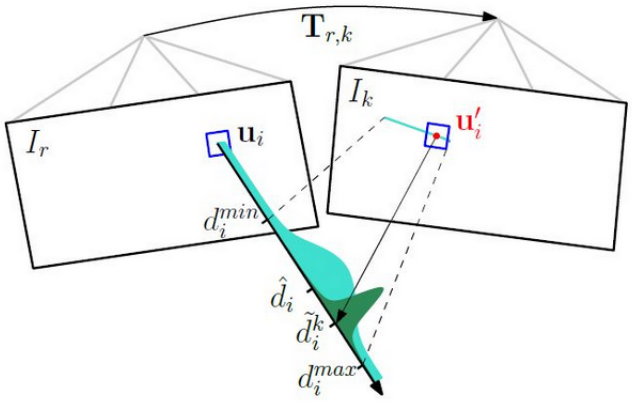


图 11: 深度的高斯-均匀分布。

10.3.1 位姿估计

对稀疏的特征块使用 direct method 配准，获取相机位姿；通过获取的位姿预测参考帧中的特征块在当前帧中的位置，由于深度估计的不准导致获取的位姿也存在偏差，从而使得预测的特征块位置不准。由于预测的特征块位置和真实位置很近，所以可以使用牛顿迭代法对这个特征块的预测位置进行优化。特征块的预测位置得到优化，说明之前使用直接法预测的有问题。利用这个优化后的特征块预测位置，再次使用直接法，对相机位姿 (pose) 以及特征点位置 (structure) 进行优化。而特征点值从关键帧中提取，文章使用重投影误差来做 pose refinement。

10.3.2 深度估计

处了 tracking 线程，svo 还有一个 mapping 线程。2d 特征的深度估计是在这个线程中完成的。SVO 中使用高斯-均匀分布对逆深度信息建模。如果当前帧被选为关键帧，就提取关键帧上的新特征点，这些特征点并未有对应的深度信息，因此需要获得计算深度。如果进来一个普通帧，就用普通帧的信息，更新所有种子点的概率分布。于是你从这条极线的一个端点走到另一个端点，把每处的图像块都和参考的去比较，就可以找到正确的匹配。如果某个种子点的深度分布已经收敛，就把它放到地图中，供追踪线程使用。

参考文献

[1] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE international conference on computer vision*, pages 2650–2658, 2015.

[2] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in neural information processing systems*, pages 2366–2374, 2014.

[3] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European conference on computer vision*, pages 834–849. Springer, 2014.

[4] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, and Dacheng Tao. Deep ordinal regression network for monocular depth estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2002–2011, 2018.

[5] Xiang Gao, Rui Wang, Nikolaus Demmel, and Daniel Cremers. Ldso: Direct sparse odometry with loop closure. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2198–2204. IEEE, 2018.

[6] Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. Deeper depth prediction with fully convolutional residual networks. In *2016 Fourth international conference on 3D vision (3DV)*, pages 239–248. IEEE, 2016.

[7] Yanyan Li, Shiyue Fan, Yanbiao Sun, Qiang Wang, and Shanlin Sun. Bundle adjustment method using sparse bfgs solution. *Remote Sensing Letters*, 9(8):789–798, 2018.

[8] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[9] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

[10] Raul Mur-Artal and Juan D Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[11] Xiaojuan Qi, Renjie Liao, Zhengzhe Liu, Raquel Urtasun, and Jiaya Jia. Geonet: Geometric neural network for joint depth and surface normal estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 283–291, 2018.

[12] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[13] Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1746–1754, 2017.

[14] Keisuke Tateno, Federico Tombari, Iro Laina, and Nassir Navab. Cnn-slam: Real-time dense monocular slam with learned depth prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6243–6252, 2017.

[15] Yinda Zhang and Thomas Funkhouser. Deep depth completion of a single rgb-d image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 175–185, 2018.

[16] Yinda Zhang, Shuran Song, Ersin Yumer, Manolis Savva, Joon-Young Lee, Hailin Jin, and Thomas Funkhouser. Physically-based rendering for indoor scene understanding using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5287–5295, 2017.