Stereo Camera System 大作业报告

2151105 崔屿杰　　2152945 任效民

1. 问题描述

本实验旨在通过双目立体视觉技术实现实时深度图生成，同时完成了稀疏与稠密点云的计算，工作流程如下图所示：

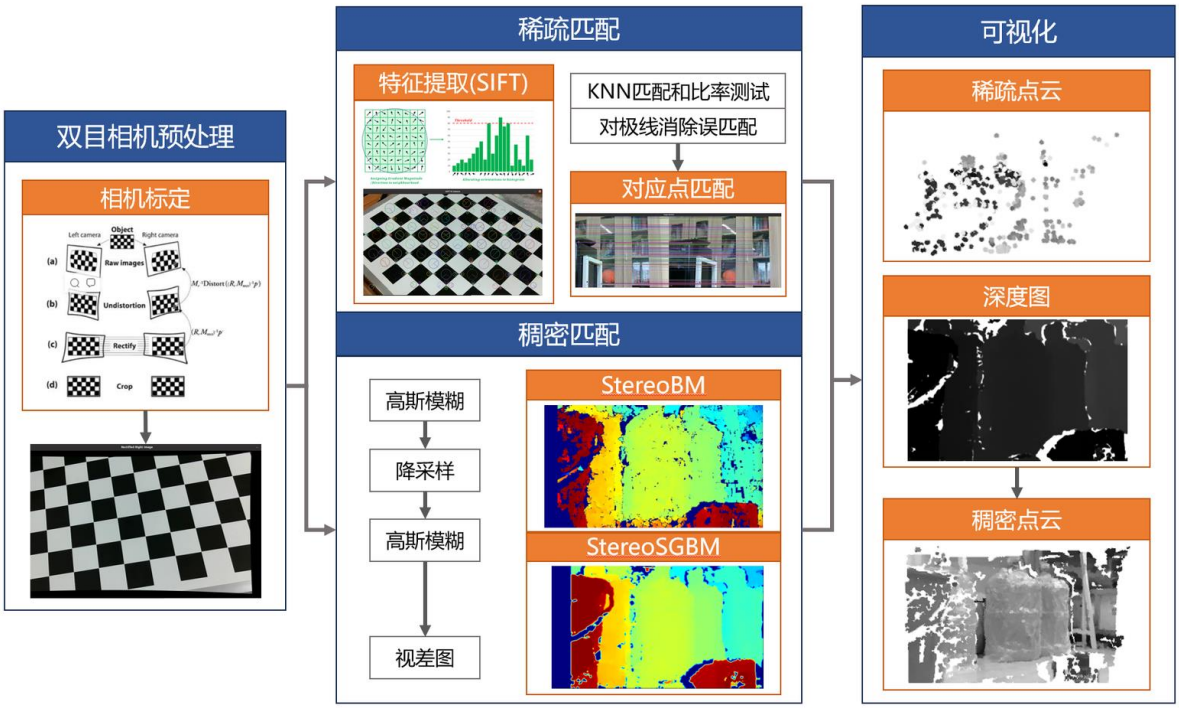


图 1：工作流程总览

首先是建立双目相机系统并进行校正，采用 Matalb 完成了相机的标定，将结果输出并使用 OpenCV 读取，利用标定参数对左右相机图像进行预处理，切除多余黑边，得到对齐的、去畸变的左右眼图像。

在特征点匹配过程中，实验使用了 SIFT 算法，然后通过 KNN 匹配和对极几何去掉错误点，提高了特征点匹配的鲁棒性和准确性。利用正确匹配的结果，完成了稀疏点云的计算及可视化。

实验采用 BM 和 SGBM 算法分别进行视差计算，通过对比两种算法在生成视差图方面的性能，分析了两种算法的优劣，然后采用了较好的方案进一步生成了深度图。通过每个点的三维坐标信息，使用 pangolin 库进行可视化，最终得到了图像的稠密点云。

本实验代码仓库：https://github.com/YukiaCUI/Stereo_Camera_HBVCAM.git

2. 图像校正

2.1. 相机标定(Stereo Calibration)

用相机拍摄一组图片（30 张），我们发现得到的图片是两张拼起来的形式（2560*720），通过

裁剪即可分别得到两张图片（1280*720），将 30 组图片保存后使用 matlab 的相机标定功能，先标定单个相机，再对两个相机进行立体标定，删去无效的图片和误差较大的图片后即可得到双目相机各方面的具体数据。

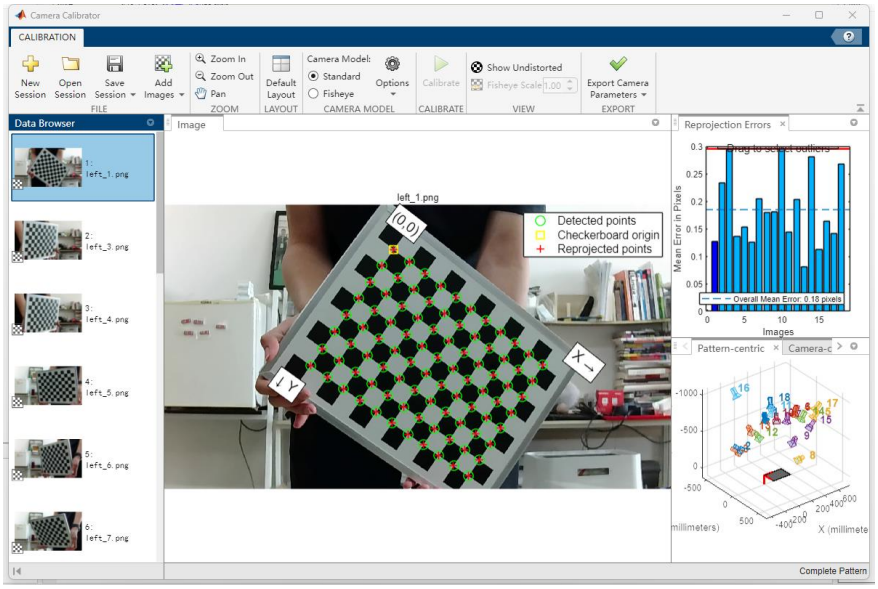


图 2：单目相机标定

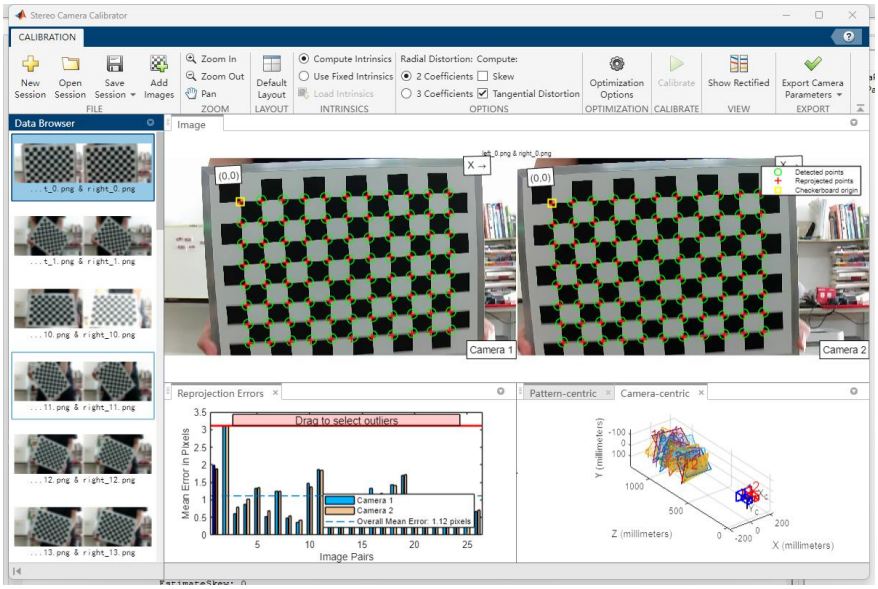


图 3：双目相机标定

为了数据读取方便，完成标定后我们写了一个脚本文件（save_param.m）将我们需要的相机信息（左右内参矩阵和畸变系数、左右的相对位姿）输出为一个 calib_param.yml 文件，这样就可以在 OpenCV 下直接读取这些数据了。

```
fileID = fopen('calib_param.yml', 'w');
fprintf(fileID, '%%YAML:1.0\n');
fprintf(fileID, 'cameraMatrixL: !!opencv-matrix\n');
fprintf(fileID, '   rows: 3\n   cols: 3\n   dt: d\n   data: [%f, %f, %f, %f, %f, %f, %f, %f,
```

代码 1：读取 matlab 中相机参数，保存为 yml 文件

```
// 从文件加载标定结果
FileStorage fs("../calibration/calib_param.yml", FileStorage::READ);
Mat cameraMatrixL, distCoeffsL, cameraMatrixR, distCoeffsR, R, T;
fs["cameraMatrixL"] >> cameraMatrixL;
fs["distCoeffsL"] >> distCoeffsL;
```

代码 2：opencv 从 yml 文件中读取相机标定参数

2.2. 图像校正（Rectification）

双目校正的作用就是要把消除畸变后的两幅图像严格地行对应，使得两幅图像的对极线恰好在同一水平线上，这样一幅图像上任意一点与其在另一幅图像上的对应点就必然具有相同的行号，只需在该行进行一维搜索即可匹配到对应点。

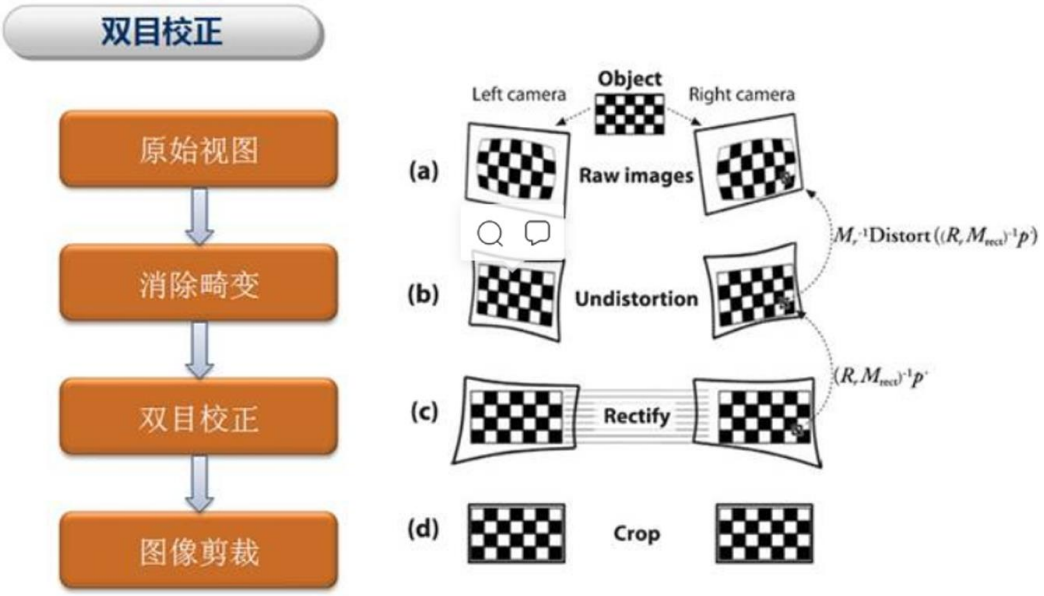


图 4：双目矫正流程

实现时，我们使用 cv::stereoRectify()函数根据标定结果计算校正变换矩阵 R1, R2，投影矩阵 P1, P2 和重投影矩阵 Q，然后使用 cv::initUndistortRectifyMap()及 cv::remap()生成校正后的图像。其中，有效像素的保留可以通过 cv::stereoRectify()的参数 alpha 来调整，例如 alpha 默认为-1，通常会保留所有有效像素，但可能会出现一些黑色边界；alpha 取 0 时裁剪到最小矩形以去除所有不需要的像素，从而最大化视差图的有效区域；alpha 取 1 时保留所有有效像素，视野最大化，但可能会包含更多的黑色边界。

```
// 立体校正
Mat R1, R2, P1, P2, Q;
stereoRectify(cameraMatrixL, distCoeffsL, cameraMatrixR, distCoeffsR,
              imageSize, R, T, R1, R2, P1, P2, Q,CALIB_ZERO_DISPARITY, 0);
```

代码 3：opencv 立体矫正

校正后的结果如下图所示，可以看到原图（左）的边缘具有桶形畸变的特征，线条出现了外扩的弧形，而校正后的图像（右）基本不存在这个问题，同时我们也可以通过改变 alpha 的值来更直观地看到有效像素和黑色边界。

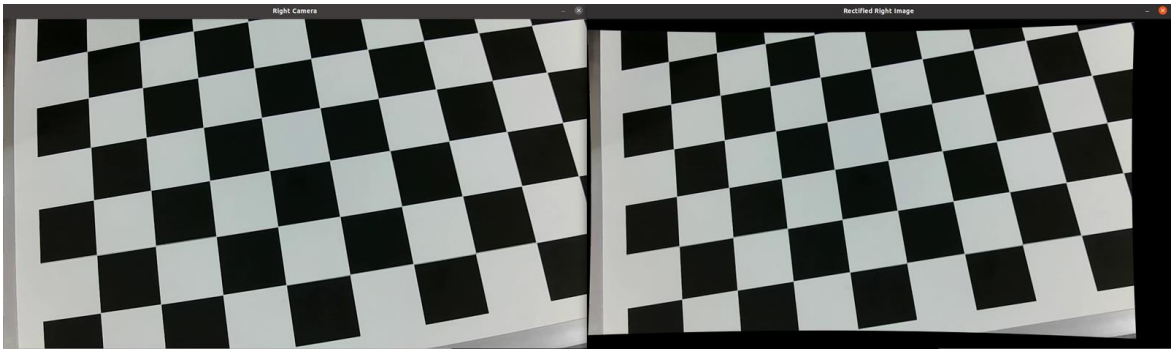


图 5：双目矫正效果

3. 特征点匹配（Feature Matching）

3.1. 特征点检测

我们采用 SIFT（Scale-Invariant Feature Transform，尺度不变特征变换）算法进行特征点的检测，这是一种用于检测和描述局部特征的算法，具有尺度和旋转不变性，使得其在图像识别、拼接、物体检测等领域中表现出色。主要包括尺度空间极值检测——关键点定位——方向分配——关键点描述符生成——特征点匹配等步骤。SIFT 算法的优点在于其对缩放、旋转以及光照变化具有很好的不变性，能在复杂场景中可靠地检测和描述图像特征。

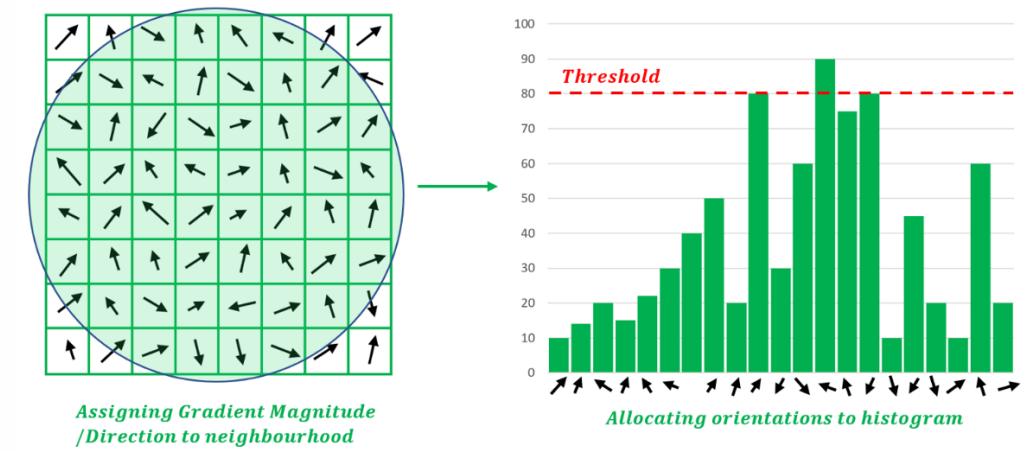


图 6：SIFT 原理图

通过 SIFT::detectAndCompute()函数和 drawKeypoints()函数，我们对校正后的图像完成了特征点的提取并在图像上画出：

```cpp
// SIFT 特征提取
Ptr<SIFT> sift = SIFT::create();
vector<KeyPoint> keypoints1, keypoints2;
Mat descriptors1, descriptors2;
Mat SIFT_left, SIFT_right;

sift->detectAndCompute(rectified_left, noArray(), keypoints1, descriptors1);
sift->detectAndCompute(rectified_right, noArray(), keypoints2, descriptors2);

drawKeypoints(rectified_left, keypoints1, SIFT_left, Scalar::all(-1), DrawMatchesFlags::DRAW_RICH_KEYPOINTS);
drawKeypoints(rectified_right, keypoints2, SIFT_right, Scalar::all(-1), DrawMatchesFlags::DRAW_RICH_KEYPOINTS);
```

代码 4：SIFT 特征提取代码

最终得到的图像如下：

图 7：SIFT 效果

3.2. 特征点匹配

完成检测后就需要对这些特征点进行匹配，然而两张图中的很大一部分特征点都是无法相互匹配的无效点，我们采用了 KNN 和对极线两种方式来解决这个问题。使用 KNN 匹配可以获取每个描述符的两个最近邻，并应用比率测试来过滤错误匹配，这可以有效地去除大部分错误匹配。

KNN 是通过测量不同特征值之间的距离进行分类。它的的思路是：如果一个样本在特征空间中的 k 个最相似(即特征空间中最邻近)的样本中的大多数属于某一个类别，则该样本也属于这个类别。KNN 算法中，所选择的邻居都是已经正确分类的对象。该方法在定类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。

当我们只用 SIFT 算法提取了左右图的特征点及其描述符，KNN 匹配算法可以返回每个描述符在目标描述符集合中的前 k 个最近邻描述符。在本实验中的 k 值选为 2，意味着每个描述符将找到两个最近的匹配。

找到两个最近的邻居后，通常会进行比率测试（如 Lowe's ratio test）来筛选出更可靠的匹配。比率测试的步骤如下：

1. 找到两个最近邻：对于每个描述符，找到两个距离最近的匹配项（在这里是 knn_matches 中的两个）。

2. 计算距离比率：计算两个最近邻的距离之比。
3. 比率测试：如果最近邻的距离与次近邻的距离之比小于设定的阈值（通常为 0.75），则认为这个匹配是可靠的。

```
// 特征点匹配
BFMatcher matcher(NORM_L2);
vector<vector<DMatch>> knn_matches;
matcher.knnMatch(descriptors1, descriptors2, knn_matches, 2);

// 应用比率测试来过滤匹配
const float ratio_thresh = 0.75f;
vector<DMatch> good_matches;
for (size_t i = 0; i < knn_matches.size(); i++) {
    if (knn_matches[i][0].distance < ratio_thresh * knn_matches[i][1].distance) {
        good_matches.push_back(knn_matches[i][0]);
    }
}
```

代码 5：KNN 匹配与比率测试过滤错误匹配

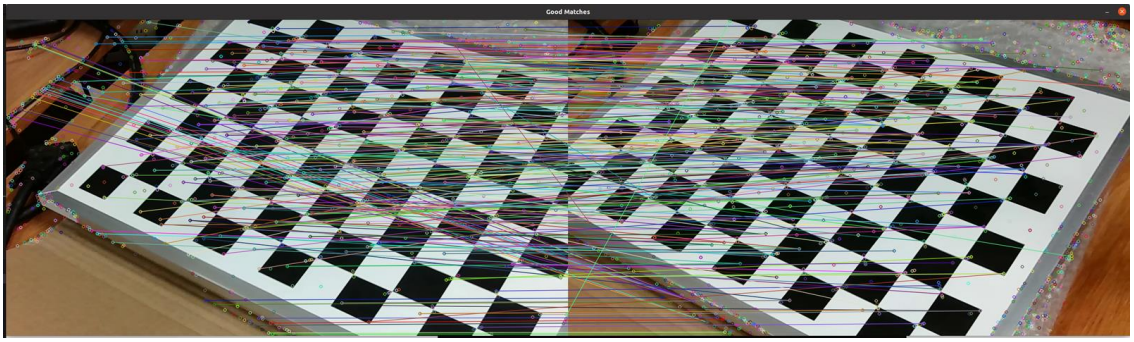


图 8：特征点匹配（过滤前）效果

第一次过滤后，图中仍存在大量杂乱的匹配点，此时使用两个匹配点的 y 坐标之间的距离来判断匹配是否有效的确是一种简单有效的方法。尤其是在经过立体校正后，理想情况下两个图像中的匹配点应该在相同的行上，y 坐标的差值应该接近于零。因此，可以设定一个阈值，如果匹配点的 y 坐标之差小于这个阈值，则认为是有效匹配。使用 cv::drawMatches()即可画出图像。

```
// 根据y坐标差值进行过滤
std::vector<cv::DMatch> y_filtered_matches;
const float y_threshold = 10.0; // y坐标差值阈值，根据需要调整
for (const auto& match : good_matches) {
    cv::Point2f pt1 = keypoints1[match.queryIdx].pt;
    cv::Point2f pt2 = keypoints2[match.trainIdx].pt;
    if (std::abs(pt1.y - pt2.y) < y_threshold) {
        y_filtered_matches.push_back(match);
    }
}
```

代码 6：根据 y 坐标差值过滤

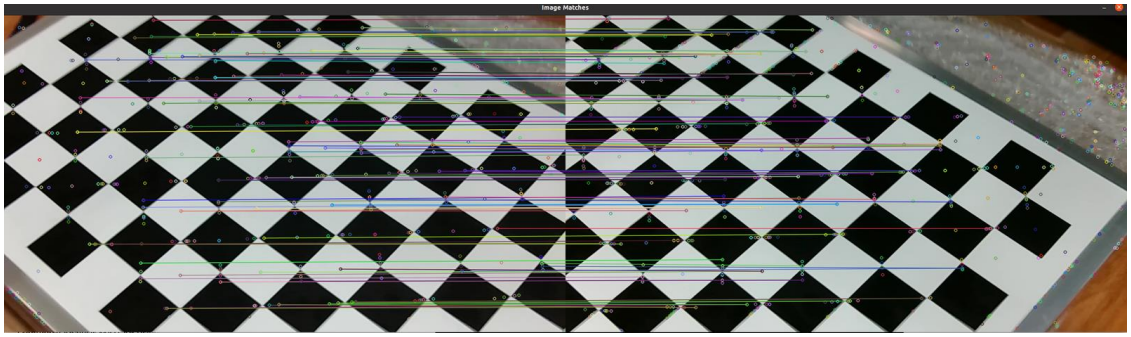


图 9：特征点匹配（过滤后）效果

可以看到，过滤之后能够正确匹配的特征点对保留了下来，杂乱的错误匹配点也基本得到了去除。这也为我们后面画出稀疏点云提供了精确、可靠的数据。

4. 视差图（Disparity Map）

4.1. BM（Block Matching）算法

BM 算法是一种基于块的立体匹配方法，通过在左图和右图中搜索相似的图像块来计算视差。首先在左图的每个像素位置上，选择一个预定义大小的窗口（如 9x9），在右图的对应扫描线上搜索与该窗口最相似的窗口。然后进行匹配代价计算，通常使用 SAD 或 SSD 来计算两个窗口的相似度。IL 和 IR 分别为左图和右图，(x,y)是左图像素位置，d 是视差，k 是窗口半径。

$$\mathrm{SAD}(x,y,d)=\sum_{i=-k}^{k}\sum_{j=-k}^{j}|I_L(x+i,y+j)-I_R(x+i-d,y+j)|$$

在计算完所有可能视差的代价后，选择代价最小的视差作为当前像素的视差值。最后再进行视差图的平滑处理、误匹配去除和视差图填充等。BM 算法的优点是简单易实现，但在处理纹理较少的区域和有噪声的图像时，匹配效果较差。

4.2. SGBM（Semi-Global Block Matching）算法

SGBM 算法结合了 BM 和全局匹配的优点，通过在多条路径上累积匹配代价，得到更准确的视差图包括代价计算——路径代价累积——代价聚合——视差选择等步骤。

代价计算：

与 BM 类似，SGBM 也使用 SAD 或其他方法计算匹配代价，但会在多个方向上累积匹配代价。

$$\mathrm{Cost}(x,y,d)=\sum_{p\in\mathcal{N}(x,y)}|I_L(p)-I_R(p-d)|$$

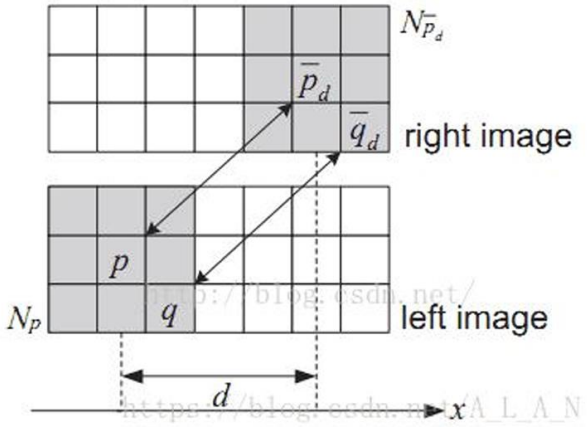


图 10：SGBM 原理图

其中，N(x,y)表示像素(x,y)周围的邻域。

路径代价累积：

在多个方向（如水平、垂直和对角线方向）上累积代价，形成全局代价。

$$\mathrm{L}_r(p,d) = C(p,d) + \min \begin{pmatrix} L_r(p-r,d) \\ L_r(p-r,d-1)+P_1 \\ L_r(p-r,d+1)+P_1 \\ \min_k L_r(p-r,k)+P_2 \end{pmatrix}$$

其中，Lr(p,d)表示路径代价，C(p,d)表示原始代价，P1 和 P2 分别是小惩罚和大惩罚，(p−r)表示前一个像素位置，k 表示视差范围内的所有可能值。

代价聚合：

对多个路径上的代价进行聚合，得到总代价。

$$\mathrm{S}(x,y,d) = \sum_r L_r(p,d)$$

视差选择与后处理：

与 BM 类似，选择总代价最小的视差作为当前像素的视差值。

唯一性检测：视差窗口范围内最低代价是次低代价的(1 + uniquenessRatio / 100)倍时，最低代价对应的视差值才是该像素点的视差，否则该像素点的视差为 0。其中 uniquenessRatio 是一个常数参数。

亚像素插值：

$$\mathrm{denom2} = \frac{\max(\mathrm{Sp[d-1]+Sp[d+1]-2*Sp[d]},1)}{16}$$

$$\mathrm{d} = \mathrm{d} + \frac{(\mathrm{Sp[d-1]-Sp[d+1]})+\mathrm{denom\,2})}{\mathrm{denom\,2*2}}$$

SGBM 算法在处理纹理较少区域和细节方面表现优于 BM 算法，因为其在多个方向上累积代价，

考虑了更多的全局信息，提高了视差图的准确性。

4.3. 算法对比

在实际操作时，我们分别写了 stereoBM()和 stereoSGBM()来计算和显示视差图，为了减少图像的噪点并减少计算量，我们对转成灰度图像进行了高斯模糊——降采样——高斯模糊的操作，然后再进行 BM/SGBM 运算，得到灰度视差图，使用 cv::applyColorMap()来得到彩色的视差图。

进行高斯模糊——降采样——高斯模糊的操作，有助于减少噪点、提高计算效率和增强视差图的质量。具体采用这样的步骤的原因：

1. 高斯模糊（第一次）：平滑原始图像，减少噪点和高频成分，确保在降采样过程中不会引入伪影。
2. 降采样：减少图像尺寸，降低计算量，提高处理速度。
3. 高斯模糊（第二次）：在降采样后的图像上再次进行平滑处理，确保在较低分辨率下的图像仍然平滑，并为接下来的立体匹配提供良好的基础。

cv::applyColorMap()是一种将灰度图像转换为伪彩色图像的方法，其原理是通过查找表（color map）将灰度值映射到对应的颜色，从而生成彩色图像。本实验程序中，我们选用的颜色查找表是 COLORMAP_JET 使用的是一种渐变色，它是从蓝色到红色的渐变色系，通过这种颜色映射，可以更直观地显示灰度图像中的不同强度值，有助于图像的分析和解释。

```cpp
void stereoBM(cv::Mat lpng,cv::Mat rpng,cv::Mat &disp)
{
    cv::GaussianBlur(lpng, lpng, cv::Size(5, 5), 1.5);
    cv::GaussianBlur(rpng, rpng, cv::Size(5, 5), 1.5);
    cv::pyrDown(lpng, lpng);
    cv::pyrDown(rpng, rpng);
    cv::GaussianBlur(lpng, lpng, cv::Size(5, 5), 1.5);
    cv::GaussianBlur(rpng, rpng, cv::Size(5, 5), 1.5);

    disp.create(lpng.rows,lpng.cols,CV_16S);
    cv::Mat disp1 = cv::Mat(lpng.rows,lpng.cols,CV_8UC1);
    cv::Size imgSize = lpng.size();
    cv::Rect roi1,roi2;
    cv::Ptr<cv::StereoBM> bm = cv::StereoBM::create(16,9);
    cv::Mat disp_color;

    int nmDisparities = ((imgSize.width / 8) + 15) & -16;//视差搜索范围

    bm->setPreFilterType(0);//预处理滤波器类型
    bm->setPreFilterSize(15);//预处理滤波器窗口大小
    bm->setPreFilterCap(31);//预处理滤波器截断值
    bm->setBlockSize(9);//SAD窗口大小
    bm->setMinDisparity(0);//最小视差
    bm->setNumDisparities(nmDisparities);//视差搜索范围
    bm->setTextureThreshold(10);//低纹理区域的判断阈值
    bm->setUniquenessRatio(5);//视差唯一性百分比
    bm->setSpeckleWindowSize(100);//检查视差连通区域变化度窗口大小
    bm->setSpeckleRange(32);//视差变化阈值
    bm->setROI1(roi1);
    bm->setROI2(roi2);
    bm->setDisp12MaxDiff(1);//左右视差图最大容许差异
    bm->compute(lpng,rpng,disp);
```

```cpp
void stereoSGBM(cv::Mat lpng,cv::Mat rpng,cv::Mat &disp_8u)
{
    cv::GaussianBlur(lpng, lpng, cv::Size(5, 5), 1.5);
    cv::GaussianBlur(rpng, rpng, cv::Size(5, 5), 1.5);
    cv::pyrDown(lpng, lpng);
    cv::pyrDown(rpng, rpng);
    cv::GaussianBlur(lpng, lpng, cv::Size(5, 5), 1.5);
    cv::GaussianBlur(rpng, rpng, cv::Size(5, 5), 1.5);

    cv::Mat disp = cv::Mat(lpng.rows,lpng.cols,CV_16S);
    disp_8u.create(lpng.rows,lpng.cols,CV_8UC1);
    cv::Size imgSize = lpng.size();
    cv::Ptr<cv::StereoSGBM> sgbm = cv::StereoSGBM::create();
    cv::Mat disp_color;

    int nmDisparities = ((imgSize.width / 8) + 15) & -16;//视差搜索范围
    int pngChannels = lpng.channels();//获取左视图通道数
    int winSize = 9;

    sgbm->setPreFilterCap(31);//预处理滤波器截断值
    sgbm->setBlockSize(winSize);//SAD窗口大小
    sgbm->setP1(16*pngChannels*winSize*winSize);//控制视差平滑度第一参数
    sgbm->setP2(32*pngChannels*winSize*winSize);//控制视差平滑度第二参数
    sgbm->setMinDisparity(0);//最小视差
    sgbm->setNumDisparities(nmDisparities);//视差搜索范围
    sgbm->setUniquenessRatio(10);//视差唯一性百分比
    sgbm->setSpeckleWindowSize(200);//检查视差连通区域变化度的窗口大小
    sgbm->setSpeckleRange(32);//视差变化阈值
    sgbm->setDisp12MaxDiff(1);//左右视差图最大容许差异
    sgbm->setMode(cv::StereoSGBM::MODE_SGBM);//采用全尺寸双通道动态编程算法
    sgbm->compute(lpng,rpng,disp);
```

代码 7：BM 算法（左）与 SGBM 算法（右）对比

两个算法最终得到的结果如下：

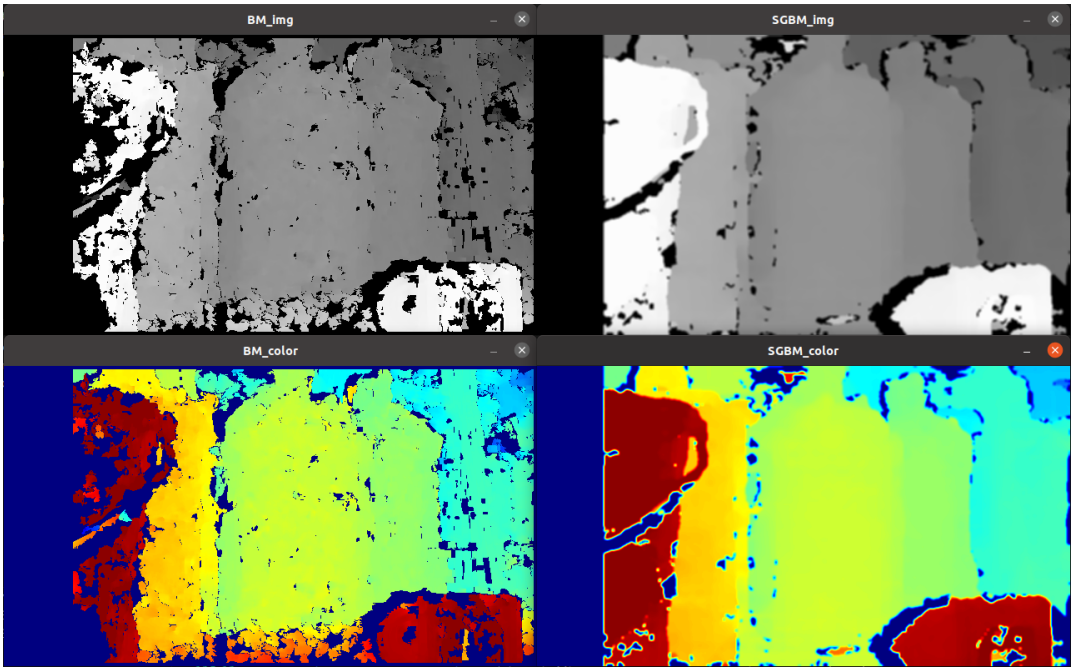


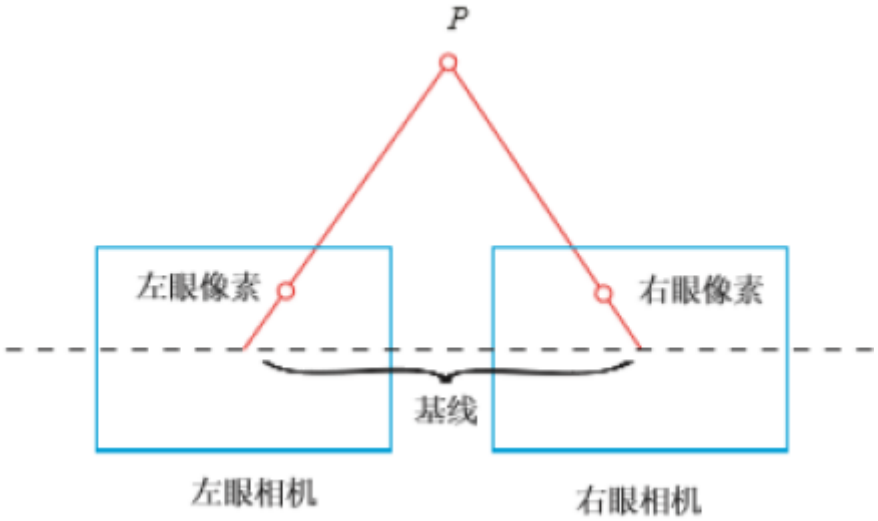
图 11：BM 算法效果（左）与 SGBM 算法效果（右）对比图

可以看到 SGBM 算法相比 BM 算法得到的视差图更准确、噪点更少。

5. 深度图（Depth Map）

5.1. 视差图转深度图原理

视差图转深度图的过程依赖于立体视觉原理，二者成反比关系。通过两个相机的成像关系和几何计算，将视差转换为深度。如下图原理图所示：



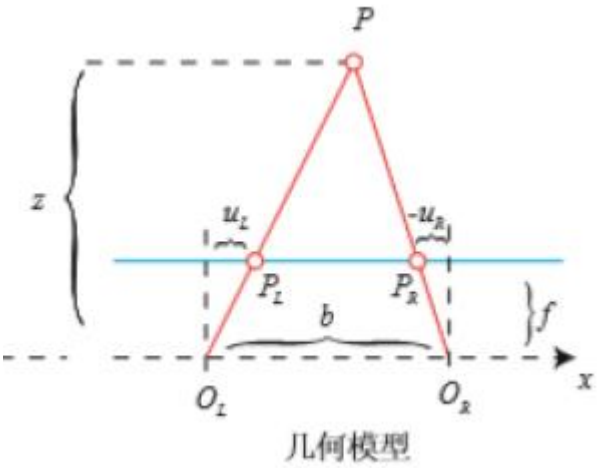


图 12：双目相机立体视觉原理图

对于一个双目模型，考虑空间有一目标点P，它在左右相机中各成一像，记作P_L和P_R，两相机的光心为O_L和O_R，相机基线b为两光心间距，即$b = O_R - O_L$。原理图中焦距在 x 方向为f_x，目标点在左右成像平面的坐标分别是u_l和u_r，因为我们此时的输入图像已经进行了矫正，所以我们认为左右目的同一个像素在一条直线上，因此视差$d = u_R - u_L$，只计算横坐标的差值即可。根据$\triangle PP_LP_R$和$\triangle PO_LO_R$的相似关系，可得到公式：

$$\frac{z - f_x}{z} = \frac{b - u_L + u_R}{b}$$

代入视差公式，可得视差转换为深度的公式为：

$$z = \frac{f_x b}{d}$$

其中，z 为深度，b 为视差。根据视差，我们可以估计一个像素与相机之间的距离。视差与距离成反比：视差越大，距离越近。同时，由于视差最小为一个像素，于是双目的深度存在一个理论上的最大值，由$f_x b$确定。不难看出，基线越长，双目能测到的最大距离就越远；反之，小型双目器件则只能测量很近的距离。相似地，我们人眼在看非常远的物体时（如很远的飞机），通常不能准确判断它的距离。

通过这种方法我们可以从视差计算出深度信息。f_x由单目相机标定时内参矩阵$K(0,0)$(两个内参矩阵相似，一般使用左相机的为基准)，而基线（baseline）b是由双目标定求到的：

$$b = \|t\|_2。$$

5.2. cv::reprojectImageTo3D() 实现转换

在 OpenCV 中，可以直接通过函数 cv::reprojectImageTo3D()来完成从视差图到三维深度图的转换。值得注意的是，该函数计算得到的图像是一个三通道 32 位浮点数的图像，但这三个通道并不是常用的 RGB 通道，而是每个像素对应的三维坐标 xyz。

在生成的三维坐标图像中，需要提取 z 通道以获得深度信息。使用 cv::split()函数将三通道图像拆分为三个单通道图像，分别表示 x、y 和 z 坐标。然后，从拆分出的通道中提取第三个通道，即 z 通道，对应于每个像素的深度值。提取后，需要对深度图中的异常值进行过滤。本实验中，是将所有小于 0 的深度值设为 0，将大于 4000 的深度值设为 4000。这一步的目的是去除无效的深度信息，使得深度图更加可靠和有意义。

最后，为了将深度图像归一化到一个单通道 8 位整数的图像上，可以使用 cv::normalize()函数。该函数将 32 位浮点数图像的深度值归一化到 0 到 255 的范围内，并转换为 8 位无符号整数图像。这一转换不仅有助于在显示和可视化深度图时更加直观，同时也为后续的图像处理操作提供了便利。

```
cv::Mat depth_map_3D;
cv::reprojectImageTo3D(disparity, depth_map_3D, Q, true); // 8UC1 -> 32FC3
std::vector<cv::Mat> channels(3);
cv::split(depth_map_3D, channels);
cv::Mat depth = channels[2]; // 32FC3 -> 32FC1
```

代码 8：cv::reprojectImageTo3D()实现转化，并改变深度图格式

最终得到归一化的深度图如下：

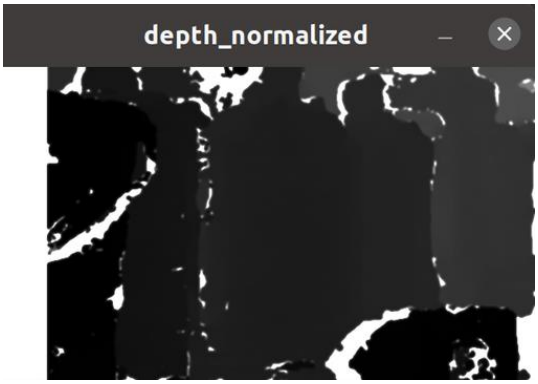


图 13：归一化的深度图效果

6. 3D 点云可视化

6.1. Pangolin 库

Pangolin 是一组轻量级和可移植的实用程序库，用于制作基于 3D、数字或视频的程序和算法的原型。它在计算机视觉领域被广泛使用，作为删除特定于平台的样板并使数据可视化变得容易的一种手段。我们使用该程序库来完成 3D 点云的生成和显示。

Pangolin 的总体精神是通过简单的界面和工厂，而不是窗口和视频，最大限度地减少样板文件并最大限度地提高可移植性和灵活性。它还提供了一套用于交互式调试的实用程序，例如 3D 操作、绘图仪、调整变量，以及用于 python 脚本和实时调整的下拉式类似 Quake 的控制台。

6.2. 稀疏点云（Sparse Pointcloud）

在“3.特征点匹配”中，我们使用 SIFT 提取到了特征点。作为图像的关键点，他们也是构成稀疏点云的点。

图 14：SIFT 特征点提取关键点效果图

我们需要将深度图中的像素转换为三维坐标：对于深度图中的每个像素，利用相机内参和深度值，可以将像素位置转换为相机坐标系下的三维坐标，如下图原理图所示。立体空间中的点具有四维参数，分别为 x，y，z 坐标以及点的颜色。

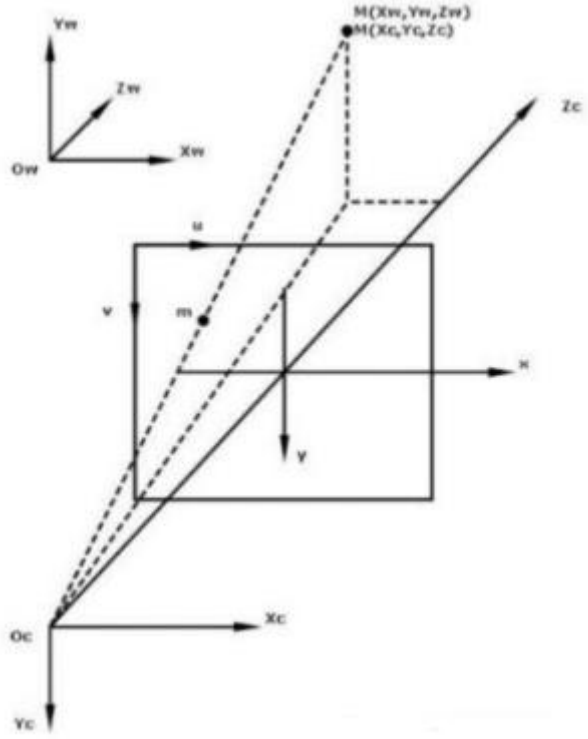


图 15：深度图像素转换为三维坐标原理图

点云的坐标计算公式是：

$$X = \frac{(u - c_x) \cdot Z}{f_x}$$

$$Y = \frac{(v - c_y) \cdot Z}{f_y}$$

$$Z = D(u, v)$$

- X, Y, Z 是三维坐标
- (u, v) 是深度图像中的像素坐标。
- f_x, f_y 是相机主距
- c_x, c_y 是相机主点坐标
- $D(u, v)$ 是深度图中像素 (u, v) 处的深度值。

$D(u, v)$的值我们在“5.1 视差图转深度图原理”中已经介绍，这里不进行过多赘述。

对于点信息第四维元素，我们可以根据深度图中的像素坐标，提取灰度图的相应灰度信息，转换成点的颜色信息。最后将四维变量点云信息传入使用 Pangolin 的点云可视化相关函数进行稀疏点云的生成。

关键代码如下：

```
// 点云数组
vector<Vector4d, Eigen::aligned_allocator<Vector4d>> pointcloud_sparse;
// 遍历像素点，利用disparity和相机内参计算点深度信息，将点存入数组中
for (auto match : rectify_matches)
{
    auto point_left = match.first;
    auto point_right = match.second;
    // 计算disparity
    double disparity = point_left.first - point_right.first;
    // 计算点深度信息
    double depth = (fx * baseline) / disparity;
    // 获取灰度信息
    double gray_left = rectified_left.at<uchar>((int)point_left.second, (int)point_left.first) / 255.0;
    // 将点存入数组中
    Vector4d point = Vector4d(0, 0, 0, gray_left); // 前三维为xyz,第四维为颜色
    double x = (point_left.first - cx) / fx;
    double y = (point_left.second - cy) / fy;
    point[0] = x * depth;
    point[1] = y * depth;
    point[2] = depth;
    pointcloud_sparse.push_back(point);
}

// 画出稀疏点云
showPointCloud(pointcloud_sparse);
```

代码 9：像素坐标转稀疏点云坐标

最终的稀疏点云效果：

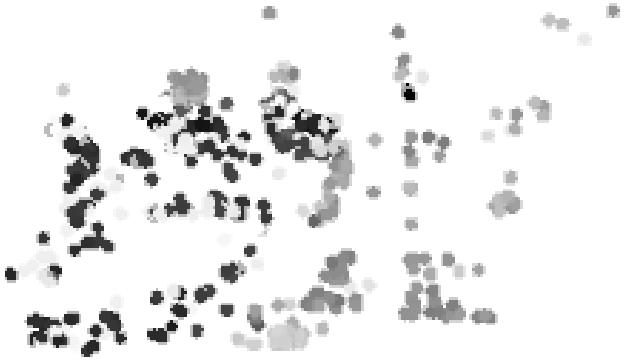


图 16：稀疏点云效果图

6.3. 稠密点云（Dense Pointcloud）

生成稀疏点云与稠密点云，在像素点转换成空间坐标点时的逻辑是一样的，区别在于左右成像平面中点对的选取。区别于稀疏点云生成时，我们选取了特征点检测 SIFT 算法提取的关键点，生成稠密点云时，我们使用了所有的像素点。

我们使用 SGBM 算法求出视差图 Disparity Map 之后，直接对该视差图进行求深度（见 5.1 视差图转深度图原理）以及成像平面像素点转空间坐标点（见 6.2 稀疏点云），获取稠密点云，传入使用 Pangolin 的点云可视化相关函数进行稠密点云的生成。

关键代码如下：

```
cv::Ptr<cv::StereoSGBM> sgbm = cv::StereoSGBM::create(
0, 96, 9, 8 * 9 * 9, 32 * 9 * 9, 1, 63, 10, 100, 32);
cv::Mat disparity_sgbm;
sgbm->compute(gray_left, gray_right, disparity_sgbm);
disparity_sgbm.convertTo(disparity, CV_32F, 1.0 / 16.0f);

vector<Vector4d, Eigen::aligned_allocator<Vector4d>> pointcloud_dense;
for (int v = 0; v < gray_left.rows; v++)
    for (int u = 0; u < gray_left.cols; u++) {
        if (disparity.at<float>(v, u) <= 0.0 || disparity.at<float>(v, u) >= 96.0) continue;

        Vector4d point(0, 0, 0, gray_left.at<uchar>(v, u) / 255.0); // 前三维为xyz,第四维为颜色

        // 根据双目模型计算 point 的位置
        double x = (u - cx) / fx;
        double y = (v - cy) / fy;
        double depth = fx * b / (disparity.at<float>(v, u));
        point[0] = x * depth;
        point[1] = y * depth;
        point[2] = depth;

        pointcloud_dense.push_back(point);
    }
```

代码 10：像素坐标转稠密点云坐标

最终的视差图、深度图、稠密点云的效果：

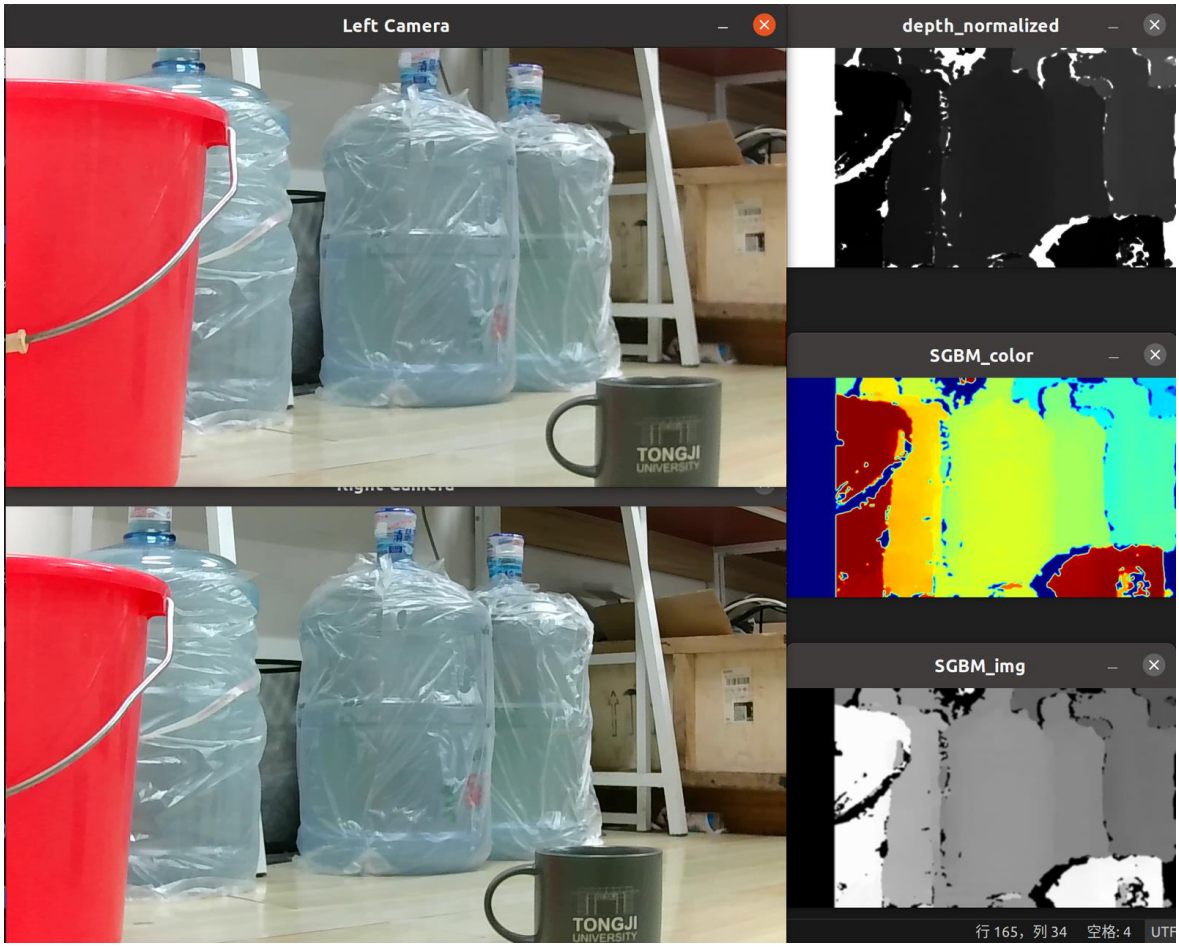


图 17：原图（左）、深度图（右上）与视差图（右下）效果

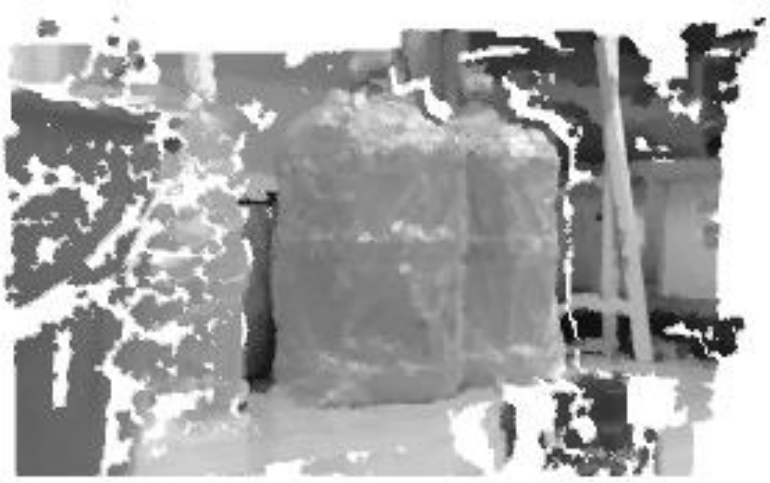


图 18：稠密点云效果

为了更好展示稠密点云的深度信息效果，我们还在 Pangolin 的点云可视化窗口中通过拖拽，展示了点云的 Z 坐标，即侧面，如下图所示：

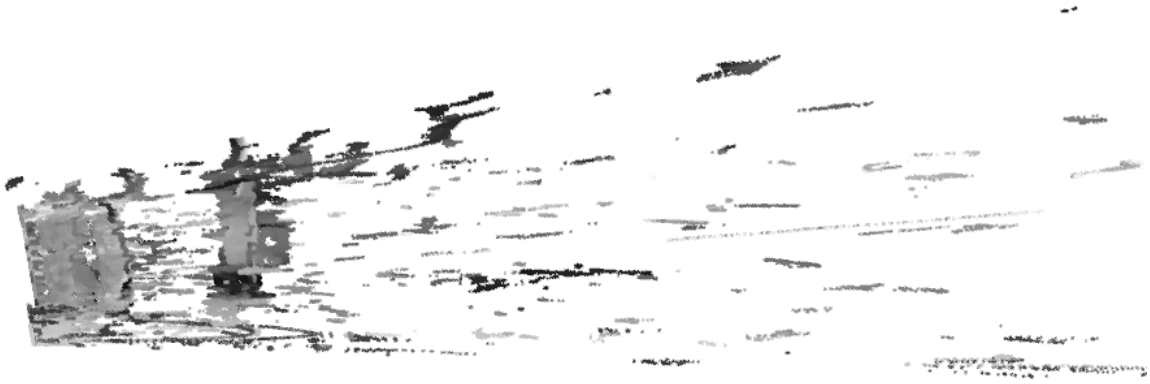


图 19：稠密点云效果（侧面）

可以看出，稠密点云图较好地还原了真实场景物体的相对位置，拟合出了点云的深度信息。

7. 分析与展望

7.1. 双目相机应用

随着自动化程度越来越高，机器视觉扮演着越来越重要的角色，双目相机系统在自动化和机器视觉领域相较于传统单目相机系统具有显著的优势，主要体现在深度感知能力、三维重建以及精确的环境理解方面。

深度感知能力

传统的单目相机只能捕捉到场景的二维信息，缺乏深度感知能力，无法准确判断物体的距离和空间位置。这在许多自动化应用中是一个重要的限制，例如在机器人导航和避障中，单目相机难以提供精确的距离信息。然而，双目相机通过两个摄像头捕捉到的图像对，可以利用视差原理计算每个像素点的深度信息，从而生成场景的三维深度图。这使得机器人能够准确地感知和理解其周围环境的三维结构，进行更加精确的路径规划和避障操作，大幅提升了自动化系统的可靠性和安全性。

三维重建

双目相机的另一个重要优势在于其强大的三维重建能力。通过捕捉同一场景的两个不同视角的图像，双目相机可以通过立体匹配算法生成点云数据，构建出场景的三维模型。这在许多工业和科研领域具有重要应用，例如在制造业中，三维重建技术可以用于产品的质量检测和逆向工程，帮助识别缺陷和进行精确的尺寸测量。在考古和文化遗产保护中，双目相机可以用于高精度的三维扫描和数字化保存，为研究和修复提供详尽的数据支持。

精确的环境理解

双目相机系统能够提供比单目相机更丰富和准确的环境信息，这对于自动化和机器视觉的许多应用至关重要。例如，在自动驾驶中，双目相机可以帮助车辆准确识别和定位道路标志、行人和其他车辆，并估算其距离和运动轨迹，提高了自动驾驶系统的感知能力和决策精度。在仓储物流中，双目相机可以用于机器人自主导航和货物识别，确保其能够安全、高效地执行物品搬运和分类任务。

此外，在智能监控系统中，双目相机可以提供更加详尽的场景分析和行为识别，提升安全监控的精确性和可靠性。

7.2. 单目相机的深度感知

虽然我们本次实验是使用双目相机构建立体成像系统，来感知物体深度信息，但是我们也不禁产生疑问，难道单目相机就不可以进行深度感知吗？

类比人眼，当我们遮住一只眼睛，只用单眼观察世界时，虽然不如双目，但是我们依然可以感觉到物体的远近。那是因为，一方面我们人时可以活动的，即便用一只眼睛， 我们通过移动，不同距离、角度的观察就可以了解物体的深度信息；另一方面，我们人是有经验的，通过近大远小，物体遮挡关系等信息，结合生活经验判断，也可以较好地判读物体深度。除此之外，我们只是遮住了一只眼睛，但是我们的耳朵、鼻子等是正常工作的，可以结合听觉、嗅觉、触觉等信息联合感知。

所以，为了完成单目相机的深度感知，我们有这些方案：使用运动的单目相机，结合深度学习方案，以及使用 Lidar 等联合感知方案。

7.2.1. 使用运动的单目相机

使用运动的单目相机，通过多帧图像来估计深度信息。这种方法称为结构从运动（Structure from Motion, SfM）。具体步骤如下：

1. 图像采集：移动单目相机，采集多帧图像。
2. 特征点检测与匹配：在连续帧图像中检测并匹配特征点。
3. 相机运动估计：利用匹配的特征点，估计相机的运动轨迹。
4. 深度估计：通过三角测量方法，根据相机运动和匹配的特征点，计算物体的深度信息。

7.2.2. 结合深度学习方案

使用深度学习网络模型，从单帧图像中直接预测深度图。这种方法称为单目深度估计（Monocular Depth Estimation）。具体步骤如下：

1. 数据准备：收集大量带有深度信息的图像数据集，用于训练模型。
2. 模型训练：选择合适的深度学习模型（如卷积神经网络），使用标注的深度图像数据进行训练。
3. 深度预测：训练好的模型可以将单帧图像输入，并输出对应的深度图。
4. 后处理：对深度图进行必要的后处理，提升深度估计的准确性。

7.2.3. 使用 Lidar 等联合感知方案

结合激光雷达（Lidar）和单目相机，利用多传感器融合技术来获取深度信息。具体步骤如下：

1. 传感器同步：确保 Lidar 和单目相机的数据采集同步。
2. 数据融合：将 Lidar 获取的点云数据与单目相机的图像数据对齐。
3. 深度补全：利用 Lidar 提供的稀疏深度信息，结合图像中的纹理特征，通过插值或深度学习方法补全深度图。
4. 优化处理：对融合后的深度图进行优化，提高深度感知的精度和鲁棒性。

通过这三种方法，单目相机也可以实现深度感知，满足不同场景下的需求。

8. 成员分工

学号	姓名	工作内容	占比
2151105	崔屿杰	相机标定，特征点匹配，稀疏点云，稠密点云，报告撰写	50%
2152945	任效民	图像矫正，特征点匹配，视差图，深度图，报告撰写、PPT 制作	50%