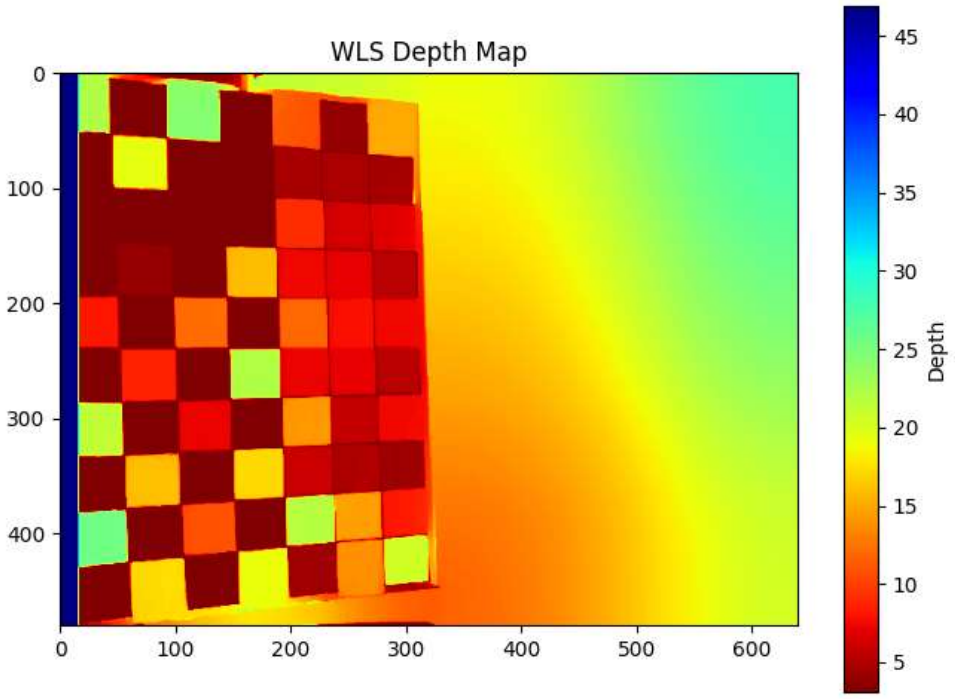
2023-EXP-SYSU_ISE-Implementation of binocular ranging algorithm based on openCV

本项目已经全部托管在了GitHub，诚邀您浏览并给予宝贵的意见

https://github.com/pbcn2/2023-EXP-SYSU_ISE-Implementation-of-binocular-ranging-algorithm-based-on-openCV

> This project uses MATLAB and python-based openCV to implement binocular ranging. Specifically, the camera is calibrated through stereo camera calibration in MATLAB, and then openCV is used for stereo correction, and then the disparity map is calculated on the corrected image to obtain Depth map. WLS filtering is used for optimization.

结果展示：



项目结构概览

本项目结构如下：

- `all_img` 文件夹中分别存放了left和right两组数据，用于在matlab中的stereo camera calibration APP进行双目相机标定
- `img_file` 文件夹中存放了两组数据：`left_017.png` `right_017.png` 两张图片是根据要求从all_img中挑选出来的用于双目测距的测试图片，最终的结果就是基于这两张图片实现的

  `left_rectified.png` `right_rectified.png` 两张图片是经过立体矫正算法纠正以后生成的，运行 `rectification.py` 以后会自动在这个文件夹下面生成这两个文件。

- `result`文件夹内有两个子文件夹，分别是`identification`和`rectification`

  `identification`文件夹中存放着MATLAB stereo camera calibration APP对图片集进行标定以后的UI界面，`stereoParameters.mat`文件中保存着APP生成的`stereoParameters`变量，里面有相机的各项参数

- `src`文件夹是存放代码的文件夹，内有以下几个文件：
  - `ExportStereoParamsOpenCV.m`是一个函数文件，将`stereoParameters.mat`文件转换成openCV风格的YAML格式文件，转换完成以后就储存为这个目录下面的`matlabStereoParam.yml`文件供`rectification.py`调用
  - `mat2openCV.m`运行用，调用ExportStereoParamsOpenCV函数
  - `matlabStereoParam.yml`一个openCV格式的ymal文件，储存两个相机的全部参数，包括内参、旋转矩阵、平移矩阵
  - `rectification.py`图像矫正程序，立体矫正
  - `compute_disparity_and_depth.py`直接利用相机参数和校正后的图像计算视差图和深度图
  - `wls_improve.py`在计算视差的过程中引入了wls滤波，也是用于计算视差图

实现原理详解

思路

本项目大致按照如下思路实现：

双目标定 --> 立体校正（含消除畸变） --> 立体匹配 --> 视差计算 --> 深度计算/3D座标计算

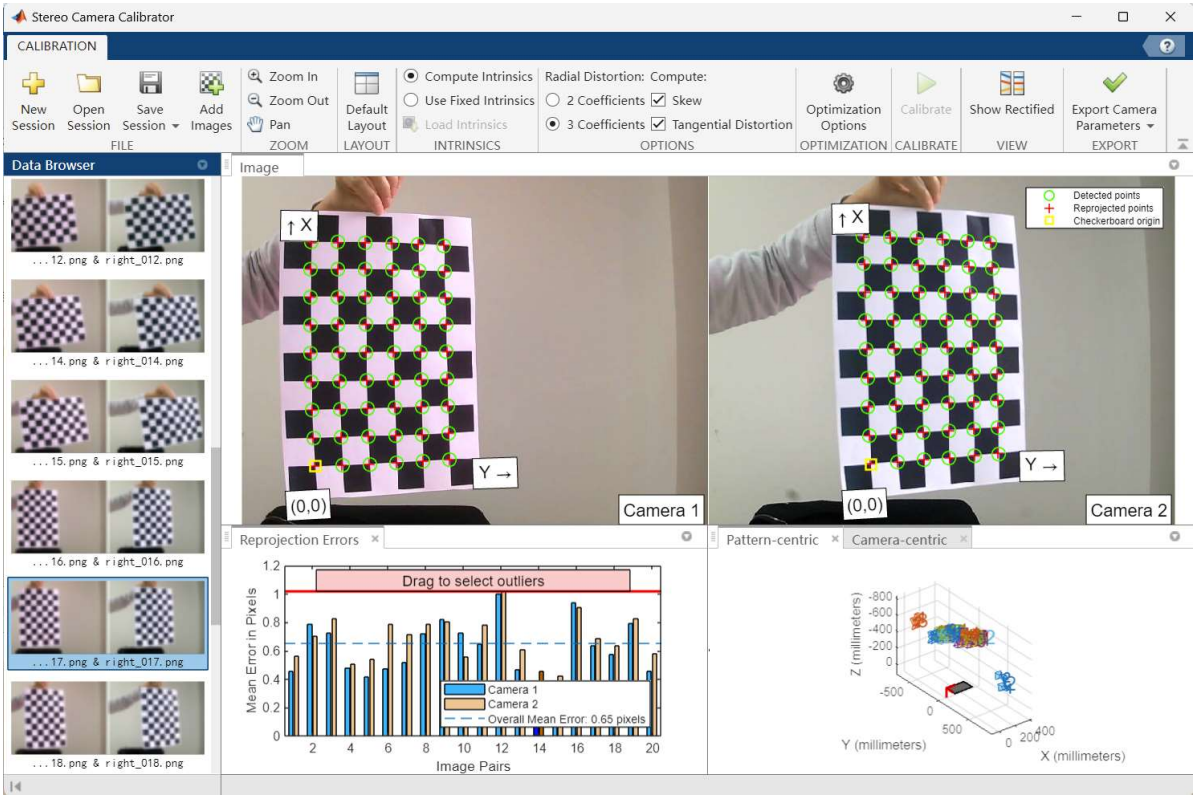
双目标定

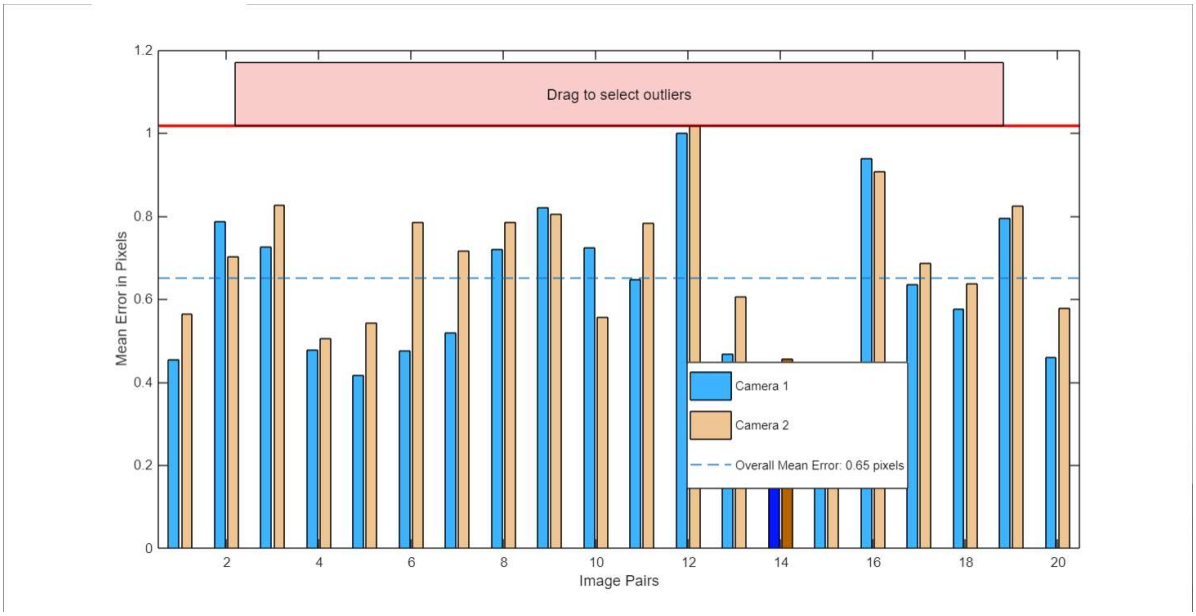
考虑到双目标定作为全部项目的基础，其生成的相机参数在项目后续的进展中会经常被调用，因此考虑使用一个成熟的解决方案。以尽可能减少误差，避免影响后续算法的效果。

考虑以后，选择了实验课上介绍过的MATLAB中的stereo camera calibration APP，这一解决方案有如下好处：

- UI交互方便易懂，课堂上有详细的介绍手册，因此操作的时候轻车熟路
- 删除+重新标定的方式可以直观的感受到误差的大小，便于控制误差
- 生成的变量是格式化的，避免了操作非格式化的文件容易出的矩阵格式错误

运行效果展示：


Stereo Camera Calibrator
CALIBRATION
New Session
Open Session
Save Session
Add Images
FILE
Zoom In
Zoom Out
Pan
ZOOM
Default Layout
LAYOUT
Compute Intrinsics
Use Fixed Intrinsics
Load Intrinsics
INTRINSICS
Radial Distortion: Compute:
2 Coefficients
3 Coefficients
Skew
Tangential Distortion
OPTIONS
Optimization Options
OPTIMIZATION
Calibrate
CALIBRATE
Show Rectified
VIEW
Export Camera Parameters
EXPORT
Data Browser
. . . 12. png & right_012. png
. . . 14. png & right_014. png
. . . 15. png & right_015. png
. . . 16. png & right_016. png
. . . 17. png & right_017. png
. . . 18. png & right_018. png
Image
Detected points
Reprojected points
Checkerboard origin
Camera 1
Camera 2
Reprojection Errors
Drag to select outliers
Mean Error in Pixels
Image Pairs
Camera 1
Camera 2
Overall Mean Error: 0.65 pixels
Pattern-centric
Camera-centric
Z (millimeters)
Y (millimeters)
X (millimeters)


Drag to select outliers
Mean Error in Pixels
Image Pairs
Camera 1
Camera 2
Overall Mean Error: 0.65 pixels

```
stereoParams =

  stereoParameters - 属性:

   Parameters of Two Cameras
        CameraParameters1: [1×1 cameraParameters]
        CameraParameters2: [1×1 cameraParameters]

   Inter-camera Geometry
              PoseCamera2: [1×1 rigidtform3d]
        FundamentalMatrix: [3×3 double]
          EssentialMatrix: [3×3 double]

   Accuracy of Estimation
    MeanReprojectionError: 0.6503

   Calibration Settings
              NumPatterns: 20
              WorldPoints: [54×2 double]
               WorldUnits: 'millimeters'
```

考虑到后期是利用基于python的openCV进行后续操作，因此需要把适用于MATLAB的mat文件转为openCV适用的格式。经过查阅官方手册，了解到openCV可以直接读取一种特定的YAML文件，因此想要将mat文件转换成YAML格式，但是在这一步遇到了很大阻力。

openCV需要如下参数：

- 左右相机的内参
- 左右相机的畸变系数
- 旋转矩阵和平移矩阵

但是新版本的MATLAB的stereo camera calibrationAPP生成的变量出现了一些变化，导致网上找的教程都不太使用，因此只能一点一点找。

最后总结得：

`stereoParams.CameraParameters1.IntrinsicMatrix`指令可以调出1号相机的内参矩阵

左右相机的畸变系数`[k1, k2, p1, p2, k3]`，`K1~K3`通过`stereoParams.CameraParameters1.RadialDistortion`调出，P1 P2为`stereoParams.CameraParameters1.TangentialDistortion`

最终形成转换代码如下：

`.\src\ExportStereoParamsOpenCV.m`

```
function ExportStereoParamsOpenCV(stereoParams, file)
% ExportStereoParams2OpenCV(cameraParams, file)
% 功能：将双目相机标定的参数保存为OpenCV可读取的xml文件
% 输入：
%   stereoParams：双目相机标定数据的对象
%   file：输出的xml文件名
% xml文件是由一层层的节点组成的。
% 首先创建父节点 fatherNode，
```

```
% 然后创建子节点 childNode=docNode.createElement(childNodeName)，
% 再将子节点添加到父节点 fatherNode.appendChild(childNode)

%创建xml文件对象，添加opencv存储标识
doc = com.mathworks.xml.XMLUtils.createDocument('opencv_storage');
docRootNode = doc.getDocumentElement; %获取根节点

M1 = (stereoParams.CameraParameters1.IntrinsicMatrix).'; % 右相机内参矩阵，需要装置为左乘的矩阵
RadialDistortion1 = stereoParams.CameraParameters1.RadialDistortion; % 右相机径向畸变参数向量1*3
TangentialDistortion1 =stereoParams.CameraParameters1.TangentialDistortion; % 右相机切向畸变向量1*2
D1 = zeros(1,5); % OpenCV的畸变参数矩阵最多可以有14个参数
D1(1:4) = [RadialDistortion1(1:2), TangentialDistortion1]; %构成opencv中的畸变系数向量[k1,k2,p1,p2,k3]
if (size(RadialDistortion1,2)>2 )
    D1(5) = RadialDistortion1(3);
end
M2 = (stereoParams.CameraParameters2.IntrinsicMatrix).'; % 右相机内参矩阵，需要装置为左乘的矩阵
RadialDistortion2 = stereoParams.CameraParameters2.RadialDistortion; % 右相机径向畸变参数向量1*3
TangentialDistortion2 =stereoParams.CameraParameters2.TangentialDistortion; % 右相机切向畸变向量1*2
D2 = zeros(1,5);
D2(1:4) = [RadialDistortion2(1:2), TangentialDistortion2]; %构成opencv中的畸变系数向量[k1,k2,p1,p2,k3]
if (size(RadialDistortion1,2)>2 )
    D2(5) = RadialDistortion2(3);
end

R = (stereoParams.RotationOfCamera2).';
T = (stereoParams.TranslationOfCamera2).';
sizeImage = stereoParams.CameraParameters1.ImageSize;

AddMat(sizeImage, 'size', doc, docRootNode);
AddMat(M1, 'M1', doc, docRootNode);
AddMat(D1, 'D1', doc, docRootNode);
AddMat(M2, 'M2', doc, docRootNode);
AddMat(D2, 'D2', doc, docRootNode);
AddMat(R, 'R', doc, docRootNode);
AddMat(T, 'T', doc, docRootNode);

xmlwrite(file, doc);

end

function AddMat(A, name, doc, docRootNode)

mat = doc.createElement(name); %创建mat节点

mat.setAttribute('type_id','opencv-matrix'); %设置mat节点属性

rows = doc.createElement('rows'); %创建行节点
```

```
rows.appendChild(doc.createTextNode(sprintf('%d',size(A,1)))); %创建文本节点，并作为行的子节点
mat.appendChild(rows); %将行节点作为mat子节点

cols = doc.createElement('cols');
cols.appendChild(doc.createTextNode(sprintf('%d',size(A,2))));
mat.appendChild(cols);

dt = doc.createElement('dt');
dt.appendChild(doc.createTextNode('d'));
mat.appendChild(dt);

data = doc.createElement('data');
for i=1:size(A,1)
    for j=1:size(A,2)
        data.appendChild(doc.createTextNode(sprintf('%.15f ',A(i,j))));
    end
    data.appendChild(doc.createTextNode(sprintf('\n')));
end
mat.appendChild(data);
docRootNode.appendChild(mat);

end
```

通过运行`./src/mat2openCV.m`执行函数

```
ExportStereoParamsOpenCV(stereoParams, "matlabStereoParam.yml")
```

立体校正

矫正过程是双目立体视觉中的一个关键步骤，可以将图像转换为对齐的形式，使得图像中的每一行对应的是真实世界中的同一个水平平面。

核心功能函数如下：

```
def stereo_image_rectification(left_image, right_image, stereo_params):
    # Extract the stereo parameters
    M1 = stereo_params['M1']
    D1 = stereo_params['D1']
    M2 = stereo_params['M2']
    D2 = stereo_params['D2']
    R = stereo_params['R']
    T = stereo_params['T']
    # size = tuple(map(int, stereo_params['size'][0]))
    size = [640,480]
    print("1")
    # Calculate rectification transform for both cameras
    R1, R2, P1, P2, Q, validPixROI1, validPixROI2 = cv2.stereoRectify(
        M1, D1, M2, D2, size, R, T, alpha=0
    )
```

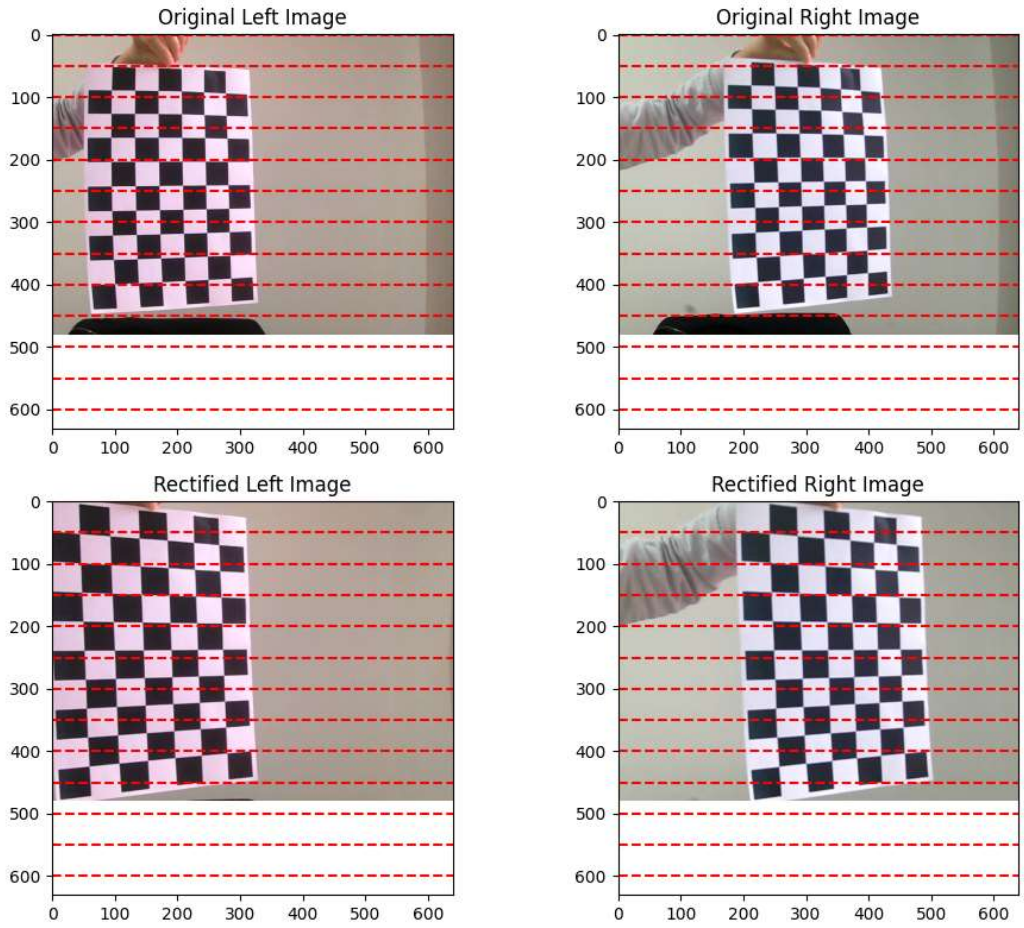
```
    print("2")
    # Compute the mapping for rectification
    map1x, map1y = cv2.initUndistortRectifyMap(M1, D1, R1, P1, size,
cv2.CV_32FC1)
    map2x, map2y = cv2.initUndistortRectifyMap(M2, D2, R2, P2, size,
cv2.CV_32FC1)
    print("3")
    # Rectify the images
    left_rectified = cv2.remap(left_image, map1x, map1y, cv2.INTER_LINEAR)
    right_rectified = cv2.remap(right_image, map2x, map2y, cv2.INTER_LINEAR)

    return left_rectified, right_rectified
```

以下是详细解释：

1. **提取双目参数**:
    - `M1` 和 `M2` 是左右相机的内部参数矩阵，描述了相机的内部特性，如焦距和主点坐标。
    - `D1` 和 `D2` 是左右相机的畸变系数，描述了相机镜头引起的畸变。
    - `R` 是左相机到右相机的旋转矩阵。
    - `T` 是左相机到右相机的平移向量。
    - `size` 是图像的大小。
2. **计算矫正变换**:
   `cv2.stereoRectify` 函数计算了两个摄像机的矫正变换。这些变换可以将摄像机的图像从原始的、可能倾斜的形式转换为对齐的形式。
    - `R1` 和 `R2` 是左右相机的矫正旋转矩阵。
    - `P1` 和 `P2` 是左右相机的新投影矩阵。
    - `Q` 是深度/视差重投影矩阵，可以用来将视差图转换为3D点云。
    - `validPixROI1` 和 `validPixROI2` 表示矫正后的图像中有效的像素区域。
3. **计算映射**:
   `cv2.initUndistortRectifyMap` 函数为每个相机计算了像素映射。这些映射描述了如何将原始图像的每个像素移动到矫正图像中的位置。
    - `map1x`, `map1y` 和 `map2x`, `map2y` 描述了如何将原始图像的像素重新映射到矫正图像中。
4. **应用映射进行矫正**:
   `cv2.remap` 函数使用上一步计算的映射来生成矫正后的图像。
    - `left_rectified` 和 `right_rectified` 是矫正后的左右图像。

这个函数最后返回矫正后的左右图像。经过矫正后，图像中的每一行应该是对齐的，这使得计算视差图变得更加简单和准确。

为`stereo_image_rectification`函数添加一些辅助代码以后就可以看到效果图：



立体匹配 & 视差计算 & 深度计算

匹配部分使用了`openCV`搭载的`StereoSGBM`算法进行分析

`emi-global matching`（缩写SGM）是一种用于计算双目视觉中disparity的半全局匹配算法。在OpenCV中的实现为`semi-global block matching`（SGBM）。

SGBM的思路是：

通过选取每个像素点的`disparity`，组成一个`disparity map`，设置一个和`disparity map`相关的全局能量函数，使这个能量函数最小化，以达到求解每个像素最优`disparity`的目的。

能量函数形式如下：

$$E(D)=\sum_{p}\left(C\left(p,D_p\right)+\sum_{q\in N_p}P_1I\left[\left|D_p-D_q\right|=1\right]+\sum_{q\in N_p}P_2I\left[\left|D_p-D_q\right|>1\right]\right)$$

`D`指`disparity map`。`E(D)`是该`disparity map`对应的能量函数。

`p`,`q`代表图像中的某个像素

`Np`指像素p的相邻像素点（一般认为8连通）

`C(p, Dp)`指当前像素点`disparity`为Dp时，该像素点的`cost`

`P1`是一个惩罚系数，它适用于像素p相邻像素中`dsparity`值与`p`的`dsparity`值相差1的那些像素。

`P2`是一个惩罚系数，它适用于像素`p`相邻像素中`dsparity`值与`p`的`dsparity`值相差大于1的那些像素。

函数返回1如果函数中的参数为真，否则返回0

利用上述函数在一个二维图像中寻找最优解是一个`NP-complete`问题，耗时过于巨大，因此该问题被近似分解为多个一维问题，即线性问题。而且每个一维问题都可以用动态规划来解决。因为1个像素有8个相邻像素，因此一般分解为8个一维问题。

考虑从左到右这一方向，如下图所示：

则每个像素的`disparity`只和其左边的像素相关，有如下公式：

$$L_r(p,d)=C(p,d)+\min\left(L_r(p-r,d),L_r(p-r,d-1)+P_1,L_r(p-r,d+1)+P_1,\min_i L_r(p-r,i)+P_2\right)-\min_k L_r(p-r,k)$$

`r`指某个指向当前像素p的方向，在此可以理解为像素`p`左边的相邻像素。
`Lr(p, d)`表示沿着当前方向（即从左向右），当目前像素`p`的`disparity`取值为d时，其最小`cost`值。

这个最小值是从4种可能的候选值中选取的最小值：

1.前一个像素（左相邻像素）`disparity`取值为d时，其最小的`cost`值。

2.前一个像素（左相邻像素）`disparity`取值为d-1时，其最小的`cost`值+惩罚系数P1。

3.前一个像素（左相邻像素）`disparity`取值为d+1时，其最小的`cost`值+惩罚系数P1。

4.前一个像素（左相邻像素）`disparity`取值为其他时，其最小的`cost`值+惩罚系数P2。

另外，当前像素p的`cost`值还需要减去前一个像素取不同`disparity`值时最小的`cost`。这是因为`Lr(p, d)`是会随着当前像素的右移不停增长的，为了防止数值溢出，所以要让它维持在一个较小的数值。

`C(p, d)`的计算很简单，由如下两个公式计算：

$$C(p,d)=\min\left(d(p,p-d,I_L,I_R),d(p-d,p,I_R,I_L)\right)$$

$$d(p,p-d,I_L,I_R)=\min_{p-d-0.5\le q\le p-d+0.5}|I_L(p)-I_L(q)|$$

即，当前像素`p`和移动`d`之后的像素`q`之间，经过半个像素插值后，寻找两个像素点灰度或者RGB差值的最小值，作为`C(p, d)`的值。

具体来说：设像素`p`的灰度/RGB值为`I(p)`，先从`I(p)`，`(I(p)+I(p-1))/2`,`(I(p)+I(p+1))/2`三个值中选择出和`I(q)`差值最小的,即

`d(p,p-d)`。然后再从`I(q)`，`(I(q)+I(q-1))/2`,`(I(q)+I(q+1))/2`三个值中选择出和`I(p)`差值最小的,即`d(p-d,p)`。最后从两个值中选取最小值，就是`C(p, d)`

上面是从一个方向（从左至右）计算出的像素在取值为某一`disparity`值时的最小`cost`值。但是一个像素有8个邻域，所以一共要从8个方向计算（左右，右左，上下，下上，左上右下，右下左上，右上左下，左下右上）这个`cost`值。

然后把八个方向上的`cost`值累加，选取累加`cost`值最小的`disparity`值作为该像素的最终`disparity`值。对于每个像素进行该操作后，就形成了整个图像的`disparity map`。公式表达如下：

$$S(p,d)=\sum_r L_r(p,d)$$

SGBM算法遍历每个像素，针对每个像素的操作和`disparity`的范围有关，故时间复杂度为：

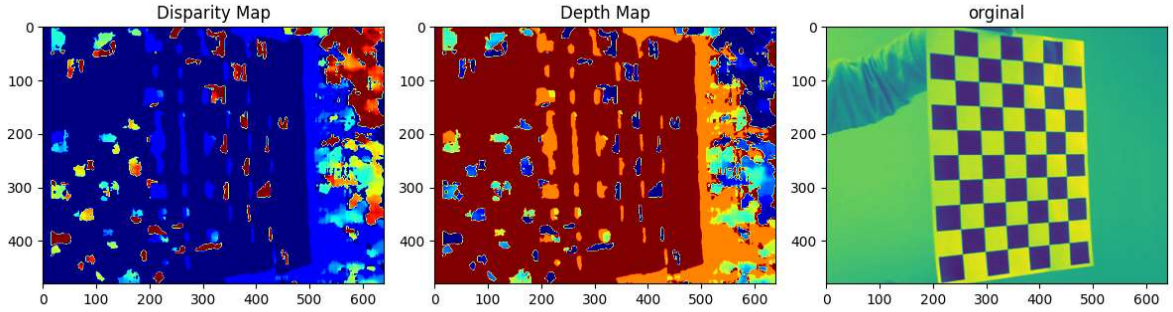
$$O(w\cdot h\cdot n)$$

在视差图的基础上，利用`depth = f * T / (disparity + 0.00001)`将视差图转换成深度图

关键代码如下，完整代码见`.\src\compute_disparity_and_depth.py`

```
def compute_disparity_and_depth(left_rectified, right_rectified):
    # Set up parameters for StereoSGBM algorithm
    window_size = 5
    min_disp = 8
    num_disp = 16 - min_disp
    stereo = cv2.StereoSGBM_create(minDisparity=min_disp,
                                   numDisparities=num_disp,
                                   blockSize=16,
                                   P1=8*3*window_size**2,
                                   P2=32*3*window_size**2,
                                   disp12MaxDiff=1,
                                   uniquenessRatio=10,
                                   speckleWindowSize=100,
                                   speckleRange=32
                                  )

    # Compute disparity image
    disparity = stereo.compute(left_rectified,
right_rectified).astype(np.float32) / 16.0

    # Assuming you have the focal length (f) in pixel and baseline (T) in meters
    # This is just an example value, you need to use your actual values
    f = 900  # example value in pixels
    T = 0.123  # example value in meters

    # Compute depth map
    depth = f * T / (disparity + 0.00001)  # adding a small value to avoid
division by zero

    # Return disparity and depth maps
    return disparity, depth
```

运行结果如下所示，从左到右依次是视差图、深度图、原图



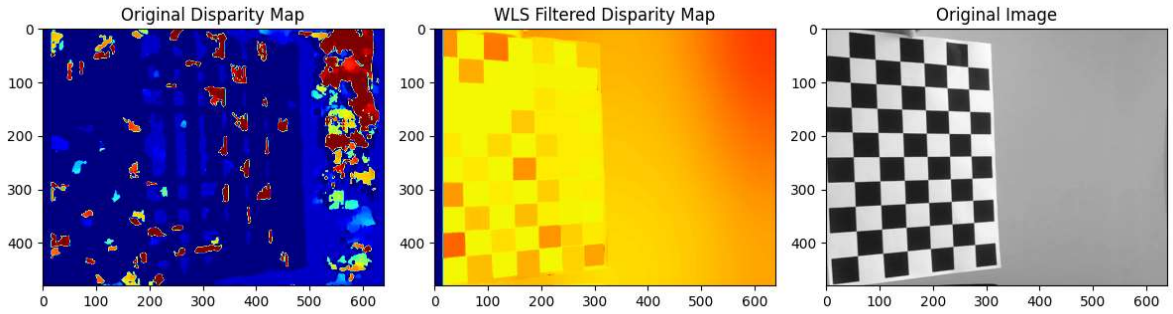
存在的问题：可以观察到，生成的视差图和深度图均有很多点计算有误，出现这一情况的原因是因为，部分区域几乎没有表面纹理，因此当SGBM算法进行匹配的时候，各个点都很相似，导致很多点匹配不准确。在小视差点如果误判成了大视差点，就会将原来的远点计算成近点。反过来也是同理。

优化方案——wls滤波器

针对存在的问题，我尝试了很多方法去解决，尝试预习资料提问中同学们提到过的限制匹配窗口的大小的方法，但是这个图片的特殊性在于相似点都集中在目标点很近且很密集，因此限制窗口的效果并不好。

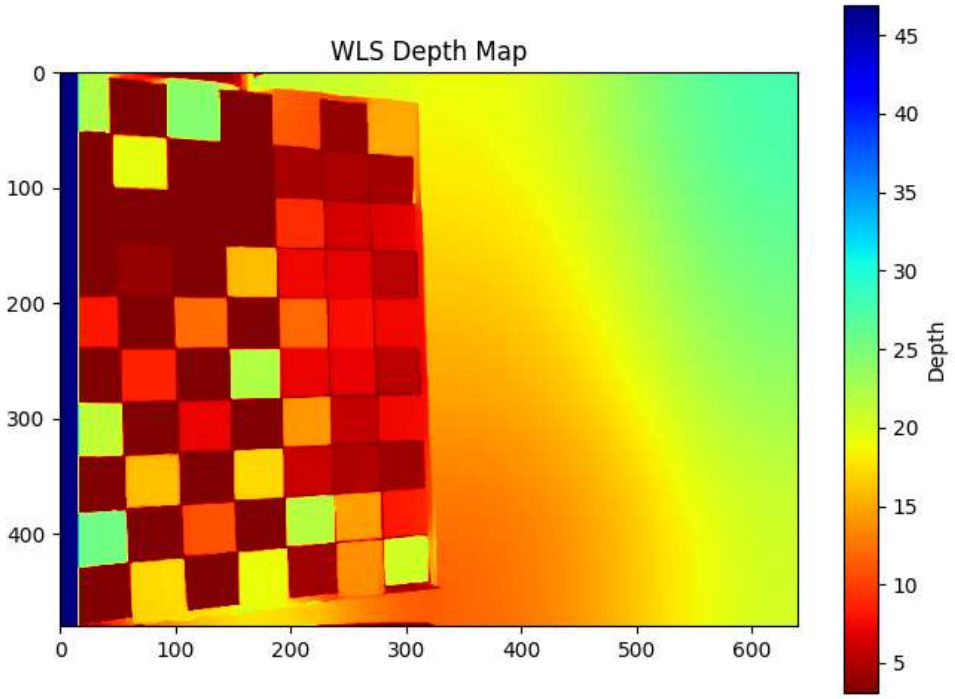
多次尝试以后选择使用滤波器，最后发现加权最小滤波器效果最好，加权最小二乘滤波器是一种保边滤波器，其目标是是滤波结果尽可能接近原图，同时在梯度较小区域尽可能平滑，而强梯度的边缘部分尽可能保持。

效果展示，从左到右依次是 不加滤波器的匹配结果，添加wls滤波器的结果，原图（灰度图）：



可以看到，使用WLS滤波后，视差图中的噪声和不一致性大大减少，同时保留了边缘和其他重要特征。

最终结果展示：



reference：

https://www.cnblogs.com/polly333/p/5130375.html

Edge-Preserving Decompositions for Multi-Scale Tone and Detail Manipulation

https://blog.csdn.net/u014230360/article/details/107639764

版权声明（Copyright Notice）

3. Indicate clearly in your project that the code is derived from this repository.

Thank you for your understanding and respect.

Contact me if you need more help: