IAVI Lab3 Report

第二组 焦笑然 温子奇 张雯琪

提交文件	说明
lab3.pdf	提交报告
/src/Calibration/	Task1 标定实现工程: camera_params/存放标定结果，img*/存放3组棋盘图像，calibrate.ipynb为运行代码
/src/Reconstruction/	Task2 计算深度图和点云：camera_params/存放相机参数，reconstruct_this/存放原始图像，reconstruct_result/存放四组深度图和点云文件，disparity.ipynb为运行代码
/src/Change_factor/	Task2 评估参数对点云数据的影响，存放3组参数改变后得到的深度图和点云文件
/src/correct_color/	Task3 色差校正

实验目的和要求

1. Perform stereo calibration with OpenCV sample
2. Compute a dense depth map / 3D point cloud from two calibrated input views via triangulation
    - Evaluate the impact of different parameters (e.g., patch size) over final quality
3. Resolve the color discrepancy between two views and produce the final colored 3D point cloud, along with the cameras, in a single .ply file
    - You may want to perform radiometric calibration first.

实验任务和分析

Task1 立体标定

- 基于OpenCV实现双目相机的内外参标定。
- 双目视觉可通过两摄像头成像所产生的视差信息来确定现实世界中物体所在准确位置，即三维坐标。双目视觉系统经过校准后，现实世界中任一点或者物体在左右视图中应位于同一水平线上，一旦极线对齐后，只需要在同一水平线上对左右视图中的同一物体进行搜索匹配，得到视差信息后根据三角关系便可得到该物体在三维空间中的位置。

Task2 计算深度图并评估patch size等参数对点云数据的影响

- 根据标定结果对原始图像校正，校正后的两张图像位于同一平面且互相平行。
- 校正后的两张图像根据极线约束进行像素点匹配，根据匹配结果计算每个像素的深度，从而获得深度图。
- 使用深度映射将像素重投影到三维空间中，生成点云文件。

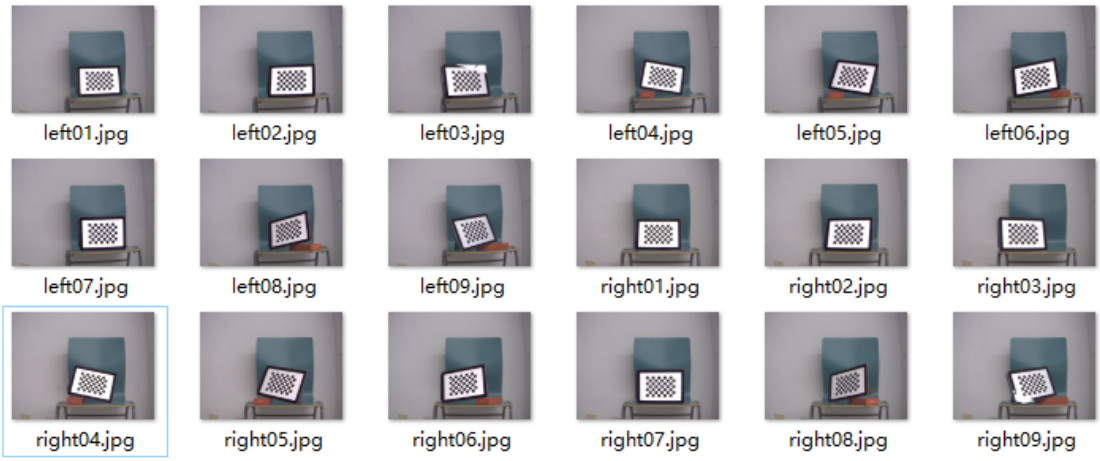
Task3 矫正色差

- 两个相机因为生产制造、参数设定等不一致，同一颜色的物体在照片中呈现的颜色会略有差异。为了更好的融合两张图片进行三维重建，也为了更好的视觉效果，需要我们矫正两张图片的色差。
- 可以额外拍照对两个相机进行颜色标定，也可以根据已经拍得的照片检测并匹配其特征点，拟合RGB向量之间的变换矩阵。

实验过程和结果

Task1 立体标定

- 首先对双目相机进行标定，得到两个相机的内外参数。
  双目立体标定的难点在于获取深度信息，我们本次一共拍摄了3组棋盘标定照片（见*calibration*中*img123*等3个文件夹），通过比较发现若棋盘同一位置在两个相机坐标系中的差值过大会造成较大误差，因此最后我们选择了*img3*中的9组棋盘照片进行标定,将得到的相机参数信息，摄像机矩阵（k）、畸变系数（dist）、旋转和平移矢量（rvecs和tvecs）存入numpy文件中
- 图片示例如下



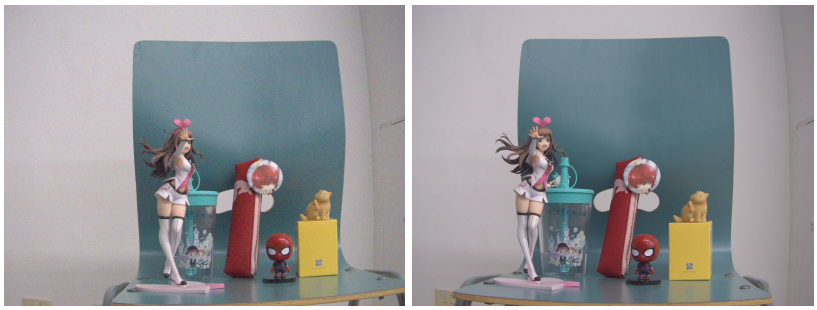
- 关键代码如下

```
#read images

calibration_paths = glob.glob('./img3/*')

#Iterate over images to find intrinsic matrix
for image_path in tqdm(calibration_paths):

  #Load image
  image = cv2.imread(image_path)
  gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  print("Image loaded, Analizying...")
  #find chessboard corners
  ret,corners = cv2.findChessboardCorners(gray_image, chessboard_size, None)

  if ret == True:
    print("Chessboard detected!")
    print(image_path)
    #define criteria for subpixel accuracy
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30,
0.001)
    #refine corner location (to subpixel accuracy) based on criteria.
    cv2.cornerSubPix(gray_image, corners, (5,5), (-1,-1), criteria)
    obj_points.append(objp)
```

```
    img_points.append(corners)

#Calibrate camera
ret, K, dist, rvecs, tvecs = cv2.calibrateCamera(obj_points,
img_points,gray_image.shape[::-1], None, None)

#Save parameters into numpy file
np.save("./camera_params/ret", ret)
np.save("./camera_params/K", K)
np.save("./camera_params/dist", dist)
np.save("./camera_params/rvecs", rvecs)
np.save("./camera_params/tvecs", tvecs)

#Get focal length in decimal form fx/fy
focal_length = 4449/1296

#Save focal length
np.save("./camera_params/FocalLength", focal_length)
```

Task2 计算深度图并评估patch size等参数对点云数据的影响

- 本次实验我们参考了一定资料后进行了以下几个步骤
  - 图片示例如下



图像畸变：消除重建所用图像中的镜头畸变。

- 考虑到在特征匹配算法中使用的图像大小会影响视差的正确计算，我们先对图像进行降采样处理，能提高图像处理速度且能在计算视差图时进行参数调整
  - 实现代码如下

```
#Get optimal camera matrix for better undistortion
new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(K,dist,(w,h),1,
(w,h))

#Undistort images
img_1_undistorted = cv2.undistort(img_1, K, dist, None,
new_camera_matrix)
img_2_undistorted = cv2.undistort(img_2, K, dist, None,
new_camera_matrix)

#Downsample each image 3 times (because they're too big)
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)
```

特征匹配：在两张图片之间查找相似的特征并构建深度图

- 立体匹配的典型技术是块匹配。块匹配的要点是在具有重叠可视区域的两个图像之间寻找强匹配点。Open CV中提供了两种块匹配的实现，立体块匹配侧重于高纹理图像（想象树的图片），半全局块匹配（SGBM）侧重于子像素级别匹配和具有更光滑纹理的图片（想象走廊的图片）。考虑到本次实验图片是在室内拍摄的，其中有很多光滑的纹理，我们采用了半全局块匹配。
- 实现代码如下

```
#Set disparity parameters
#Note: disparity range is tuned according to specific parameters obtained
through trial and error.
win_size = 5
min_disp = -1
max_disp = 63 #min_disp * 9
num_disp = max_disp - min_disp # Needs to be divisible by 16

#Create Block matching object.
stereo = cv2.StereoSGBM_create(minDisparity= min_disp,
  numDisparities = num_disp,
  blockSize = 3,
  uniquenessRatio = 5,
  speckleWindowSize = 5,
  speckleRange = 5,
  disp12MaxDiff = 2,
  P1 = 8*3*win_size**2,#8*3*win_size**2,
  P2 =32*3*win_size**2) #32*3*win_size**2)

#Compute disparity map
print ("\nComputing the disparity  map...")
disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)
```

将像素重投影到三维空间中,生成点云文件。

- 在得到视差图后，我们还需要得到图像中使用的颜色的数组。同时由于我们对图像进行了下采样，需要得到图像的高度和宽度。OpenCV的文档有一个转换矩阵，可以将深度和颜色重新投射到三维空间中
- 转换矩阵如下

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{f'} & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ \frac{z}{f'} \end{bmatrix} \tag{1}$$

- 实现代码如下

```
#Get new downsampled width and height
h,w = img_2_downsampled.shape[:2]

#Load focal length.
focal_length = np.load('./camera_params/FocalLength.npy')

#Perspective transformation matrix
#This transformation matrix is from the openCV documentation, didn't seem to
work for me.
Q = np.float32([[1,0,0,-w/2.0],
        [0,-1,0,h/2.0],
```

```
        [0,0,0,-focal_length],
        [0,0,1,0]])

#This transformation matrix is derived from Prof. Didier Stricker's power
point presentation on computer vision.
#Link : https://ags.cs.uni-kl.de/fileadmin/inf_ags/3dcv-ws14-
15/3DCV_lec01_camera.pdf
Q2 = np.float32([[1,0,0,0],
        [0,-1,0,0],
        [0,0,focal_length*0.05,0], #Focal length multiplication obtained
experimentally.
        [0,0,0,1]])

#Reproject points into 3D
points_3D = cv2.reprojectImageTo3D(disparity_map, Q2)
#Get color points
colors = cv2.cvtColor(img_1_downsampled, cv2.COLOR_BGR2RGB)

#Get rid of points with value 0 (i.e no depth)
mask_map = disparity_map > disparity_map.min()

#Mask colors and points.
output_points = points_3D[mask_map]
output_colors = colors[mask_map]

#Define name for output file
output_file = 'reconstructed.ply'

#Generate point cloud
print ("\n Creating the output file... \n")
create_output(output_points, output_colors, output_file)
```

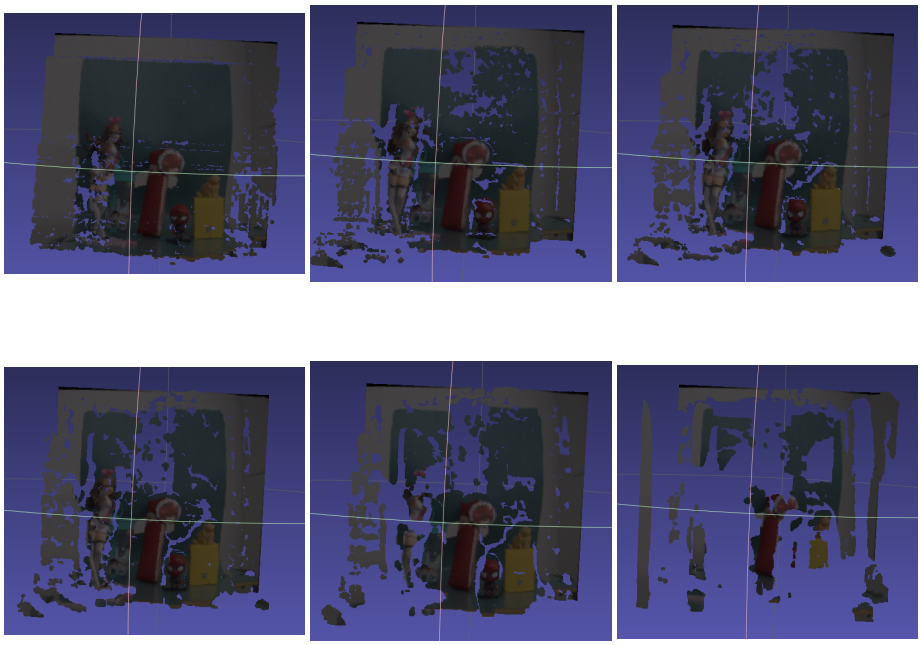
改变块匹配的参数对生成文件的影响

- SGBM半全局块匹配包括3个步骤

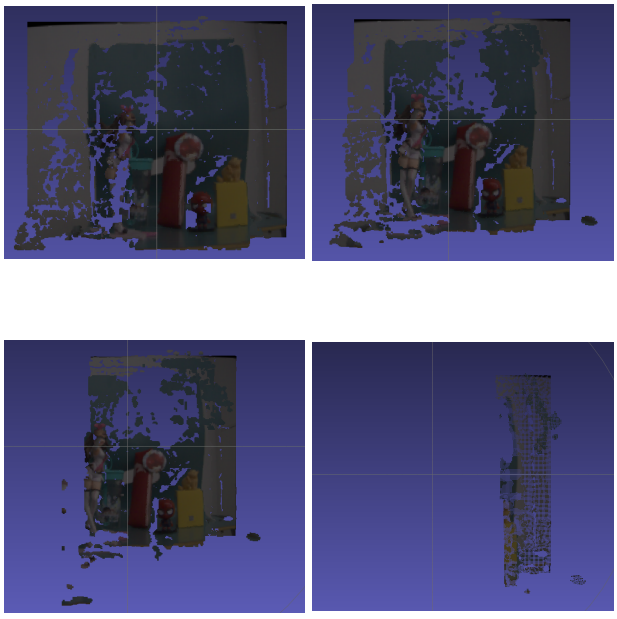
  1. 预过滤图像，用于归一化亮度和增强纹理
     通过在图像上运行一个窗口（修改窗口大小的参数win_size）来完成归一化亮度和增强纹理操作。
  2. 使用SAD窗口沿水平极线执行相应的搜索
     通过滑动SAD窗口来计算相关性，沿极线遍历找到匹配项。
  3. 后过滤图像，以消除不良的相关匹配
     对特征匹配之后可能存在假证样本（即错误匹配）进行处理。

```
#Create Block matching object.
stereo = cv2.StereoSGBM_create(
  minDisparity= min_disp, #最小可能的差异值
  numDisparities = num_disp, #最大差异减去最小差异
  blockSize = 3, #匹配的块大小
  uniquenessRatio = 5, #最佳（最小）计算成本函数值
  speckleWindowSize = 5, #平滑视差区域的最大尺寸
  speckleRange = 5, #每个连接组件内的最大视差变化
  disp12MaxDiff = 2, #左右视差检查中允许的最大差异
  P1 = 8*3*win_size**2, #控制视差平滑度
  P2 =32*3*win_size**2 #控制视差平滑度
)
```

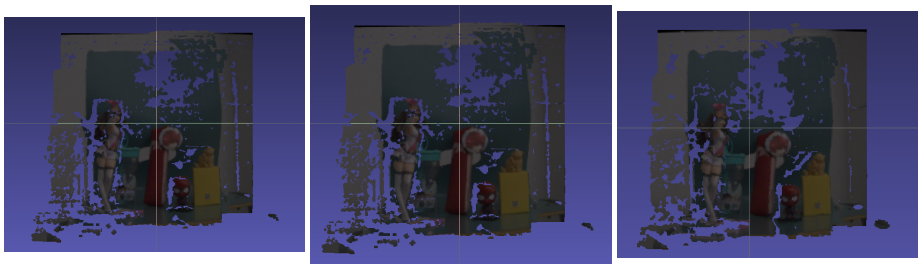
- blocksize 更改匹配的块大小
    - 将block_size的值依次改为1、3、5、9、15、21，得到的点云图像如下所示



    - 可以看出block_size的取值越小，生成的图像还原度越高
- numDisparities 更改最大差异减去最小差异的值
    - 将numDisparities的值依次改为32、64、128、256，得到的点云图像如下所示



    - 可以看出numDisparities的取值越小，生成的图像还原度越高
- disp12MaxDiff 更改左右视差检查中允许的最大差异
    - 将disp12MaxDiff的值依次改为0、±1，得到的点云图像如下所示



Task3 矫正色差

- 特征点检测
  - SIFT特征是图像的局部特征，对旋转、尺度缩放、亮度变化保持不变性，对视角变化、仿射变换、噪声也保持一定程度的稳定性。因此，我们采用SIFT算法来进行特征点检测和特征描述子的提取。

```
sift = cv2.xfeatures2d.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)
```

- 特征点匹配
  - 我们根据上一步提取出的各个角点的描述子的相似度来在两幅图中匹配特征点，最简单的算法是暴力搜索。为了提高效率，我们采用了flann算法，即快速最近邻搜索。 索引方式采用kd tree, 搜索方式采用knn.

```
FLANN_INDEX_KDTREE = 0#kd树
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks=50)   # or pass empty dictionary
flann = cv2.FlannBasedMatcher(index_params,search_params)
matches = flann.knnMatch(des1,des2,k=2)
```

- 优化变换矩阵

$$\begin{bmatrix} R_d \\ G_d \\ B_d \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} R_s \\ G_s \\ B_s \end{bmatrix} \tag{2}$$

其中，理想情况下变换矩阵应为

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

  - 从初值出发，在upper bound和lower bound之间的搜索空间内搜索一组参数，使得以该组参数为变换矩阵进行颜色变换后误差最小。
  - 误差函数核心代码如下：

```
r = P(1) * sr + P(2) * sg + P(3) * sb;
g = P(4) * sr + P(5) * sg + P(6) * sb;
b = P(7) * sr + P(8) * sg + P(9) * sb;

error = error + sqrt((r - dr)^2 + (g - dg)^2 + (b - db)^2);
```

- 非线性优化部分核心代码如下：

```
para0  = [  1;      0;      0;       0;       1;       0;       0;
0;      1 ];
lb     = [  0.98;   -0.02;   -0.02;    -0.02;     0.98;     -0.02;
 -0.02;    -0.02;   0.98 ];
ub     = [  1.02;      0.02;    0.02;     0.02;    1.02;      0.02;
0.02;    0.02;    1.02 ];


options = optimset('Display','iter-detailed','Algorithm','interior-
point','FunValCheck','on',...

 'TolFun',10^-6,'LargeScale','off','TolX',10^-6,'MaxFunEvals',10^6,...
   'MaxIter',10000);

[respara, reserror, exitflag, output] = fmincon(@errorfunc, para0, [],
[], [], [], lb, ub, [], options);
```

- 运行结果如下：

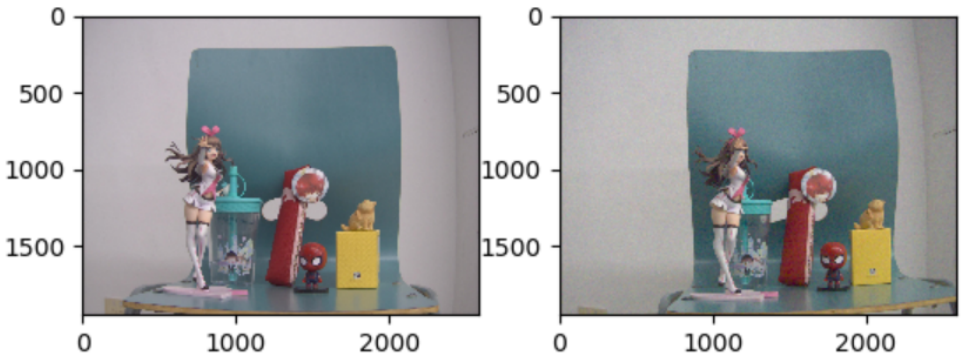
```
M = [0.98,0.002007,-0.02,-0.014843,0.98,-0.02,-0.019999,-0.01737,0.98]
```

```
13 —    lb     = [  0.98;   -0.02;   -0.02;    -0.02;     0.98;     -0.02;    -0.02;    -0.02;   0.98 ];
14 —    ub     = [  1.02;      0.02;    0.02;     0.02;    1.02;      0.02;     0.02;     0.02;    1.02 ];
15
16
17 —    options = optimset('Display','iter-detailed','Algorithm','interior-point','FunValCheck','on',...
18          'TolFun',10^-6,'LargeScale','off','TolX',10^-6,'MaxFunEvals',10^6,...
19        'MaxIter',10000);
20
```

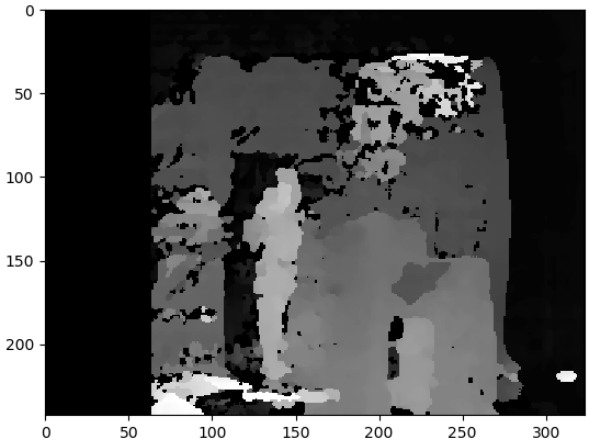
命令行窗口

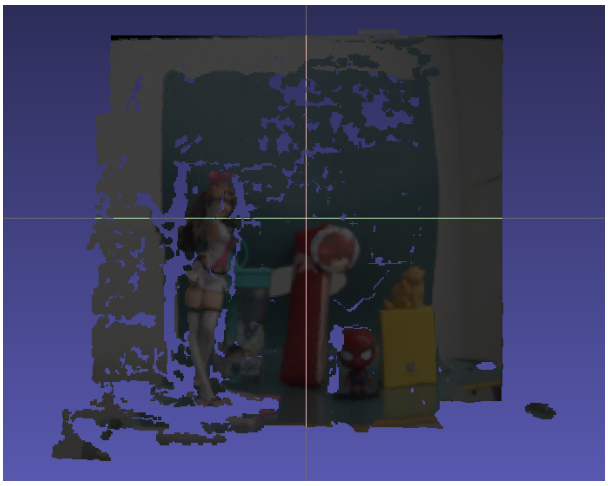
```
    13     156    6.543173e+03    0.000e+00    4.448e+00    2.018e-04
    14     166    6.543173e+03    0.000e+00    3.804e-01    1.001e-05
    15     176    6.543173e+03    0.000e+00    1.910e-03    5.943e-07

Optimization stopped because the relative changes in all elements of x are
less than options.StepTolerance = 1.000000e-06, and the relative maximum constraint
violation, 0.000000e+00, is less than options.ConstraintTolerance = 1.000000e-06.

0.98,
0.002007,
-0.02,
-0.014843,
0.98,
-0.02,
-0.019999,
```

- 最终效果



- 生成视差图及点云





参考资料

1. 基于OpenCV的三维重建
2. 由双目立体视差图重建三维点云
3. 使用iPhone相机和openCV来完成3D重建
4. OpenCV立体相机标定Stereo Calibration与校准检验Rectification详述
5. Stereo calibration using C++ and OpenCV
6. cv::StereoSGBM Class Reference