IAVI Assignment3 Report

第二组 焦笑然 温子奇 张雯琪

实验目的和要求

• Single Camera Calibration with OpenCV
• You may use your own camera if its focal length can be fixed
• Recommend using a chessboard on a Pad / monitor
• Goals

1. Finish intrinsic and extrinsic calibration
2. Discuss the impact of different factors (e.g., # of images, coverage of the chessboard, view angle, etc) over the final reprojection error
3. Output the estimated camera centers and the chessboard into a .ply file for 3D viewing in software like MeshLab
4. Project some interesting 3D points on the input images (i.e., augmented reality)
5. [Bonus 15%] Make a real-time demo. You may want to fix the intrinsic parameters and solve for the extrinsic ones only.

实验任务和分析

Task1 相机的内外参标定

- 相机标定即获取摄像机的内参和外参矩阵，建立起三维世界和二维图像平面的对应关系，涉及如下坐标系的变化：
  世界坐标系 --> 相机坐标系 --> 图像坐标系 --> 像素坐标系
- 内参矩阵

$$f(x) = \begin{bmatrix} fx & s & x0 \\ 0 & fy & y0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

  其中fx、fy为焦距，x0、y0为相对于成像平面的主点坐标，s为坐标轴倾斜参数

- 外参矩阵

$$W2C = rotateZ(rz) \times rotateY(ry) \times rotateX(rx) \times dtranslate(dx, dy, dz) \tag{2}$$

  其中

$$rotateX(rx) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(rx) & -sin(rx) & 0 \\ 0 & sin(rx) & cos(rx) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

$$rotateY(ry) = \begin{bmatrix} cos(ry) & 0 & sin(ry) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(ry) & 0 & cos(ry) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

$$rotateZ(rz) = \begin{bmatrix} cos(rz) & -sin(rz) & 0 & 0 \\ sin(rz) & cos(rz) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

$$dtranslate(x, y, z) = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

一般情况下，世界坐标系和相机坐标系不重合，这时，世界坐标系中的某一点P要投影到像面上时，先要将该点的坐标转换到相机坐标系下。外参矩阵由旋转和平移矩阵构成，旋转矩阵描述了世界坐标系相对于相机坐标系的方向，平移矩阵描述了相机坐标系下空间原点的位置

- 本实验采用光学标定法，即利用已知的几何信息(定长棋盘格)实现参数求解。这是一个非线性优化过程，具体实现方法如下：
  - 初始化世界坐标系，棋盘平面为z平面，棋盘左上角内格点坐标为(0,0,0)，确定所有8*11个棋盘格点的世界坐标系下坐标；
  - 对图像进行棋盘格点检测，记录格点的像素坐标；
  - 通过当前内外参，将棋盘格点的世界坐标投影到像素坐标系下，计算和图像上格点的误差；
  - 根据误差反向更新相机内外参。

Task2 探究重投影误差影响因子

- 重投影误差是像素坐标（观测到的投影位置）与3D点按照当前估计的位姿进行投影得到的位置相比较得到的误差。
- 通过改变相机位置、不同位置遮挡棋盘等方法，采集多组相机拍摄数据进行标定，探究对重投影误差的影响

Task3 输出估算出的相机位姿和棋盘

- 相机外参共有6个参数，其中三个平移分量，三个旋转分量。平移分量描述了相机坐标系下空间原点的位置，据此可得出相机位置；旋转分量描述了世界坐标系相对于相机坐标系的方向，平移之后，坐标系分别绕x\y\z轴旋转即可得到相机坐标系，据此可得出相机姿态。
- 棋盘格点的坐标可由以下两种方式得到：
  - 根据棋盘格点的边长和长宽格点数计算得出，这一步在标定程序中已经实现，可在输出的xml文件的grid_points下获取格点坐标。

VID5.xml | #out_camera_data.xml | in_VID5.xml | out_img.xml

```
    1.08167615e+03 1.69420886e+03 1.23885498e+03 1.68677271e+03
    1.39665015e+03 1.67817004e+03 1.54567297e+03 1.66902893e+03
    1.69094031e+03 1.65778491e+03 1.83313330e+03 1.64826514e+03</data></image_po
<grid_points>
  0. 0. 0. 11. 0. 0. 22. 0. 0. 33. 0. 0. 44. 0. 0. 55. 0. 0. 66. 0. 0.
  77. 0. 0. 88. 0. 0. 99. 0. 0. 110. 0. 0. 0. 11. 0. 11. 11. 0. 22. 11.
  0. 33. 11. 0. 44. 11. 0. 55. 11. 0. 66. 11. 0. 77. 11. 0. 88. 11. 0.
  99. 11. 0. 110. 11. 0. 0. 22. 0. 11. 22. 0. 22. 22. 0. 33. 22. 0. 44.
  22. 0. 55. 22. 0. 66. 22. 0. 77. 22. 0. 88. 22. 0. 99. 22. 0. 110. 22.
  0. 0. 33. 0. 11. 33. 0. 22. 33. 0. 33. 33. 0. 44. 33. 0. 55. 33. 0.
  66. 33. 0. 77. 33. 0. 88. 33. 0. 99. 33. 0. 110. 33. 0. 0. 44. 0. 11.
  44. 0. 22. 44. 0. 33. 44. 0. 44. 44. 0. 55. 44. 0. 66. 44. 0. 77. 44.
  0. 88. 44. 0. 99. 44. 0. 110. 44. 0. 0. 55. 0. 11. 55. 0. 22. 55. 0.
  33. 55. 0. 44. 55. 0. 55. 55. 0. 66. 55. 0. 77. 55. 0. 88. 55. 0. 99.
  55. 0. 110. 55. 0. 0. 66. 0. 11. 66. 0. 22. 66. 0. 33. 66. 0. 44. 66.
  0. 55. 66. 0. 66. 66. 0. 77. 66. 0. 88. 66. 0. 99. 66. 0. 110. 66. 0.
  0. 77. 0. 11. 77. 0. 22. 77. 0. 33. 77. 0. 44. 77. 0. 55. 77. 0. 66.
  77. 0. 77. 77. 0. 88. 77. 0. 99. 77. 0. 110. 77. 0.</grid_points>
</opencv_storage>
```

  - 根据标定出的内外参，对图像上检测出的格点进行逆投影，确定对应格点在世界坐标系下的x/y分量。可在标定程序输出的xml文件的image_points中获得格点的图像像素坐标。

```
<image_points type_id="opencv-matrix">
  <rows>7</rows>
  <cols>88</cols>
  <dt>"2f"</dt>
  <data>
    3.12163940e+02 5.31266235e+02 4.73000885e+02 5.23420898e+02
    6.43302551e+02 5.16351563e+02 8.15705627e+02 5.10749451e+02
    9.95348022e+02 5.06451965e+02 1.17394666e+03 5.03680695e+02
    1.35363940e+03 5.02134674e+02 1.53524780e+03 5.02050201e+02
    1.71058154e+03 5.03500214e+02 1.88515051e+03 5.06513763e+02
    2.05490332e+03 5.10582123e+02 2.99291290e+02 6.96564514e+02
    4.62722809e+02 6.90493774e+02 6.35060547e+02 6.85133667e+02
    8.10802856e+02 6.80969177e+02 9.92377136e+02 6.77466431e+02
    1.17366199e+03 6.75250854e+02 1.35605066e+03 6.73673096e+02
```

- 将想要观察的点输入到ply文件中，通过meshlab打开进行观察。

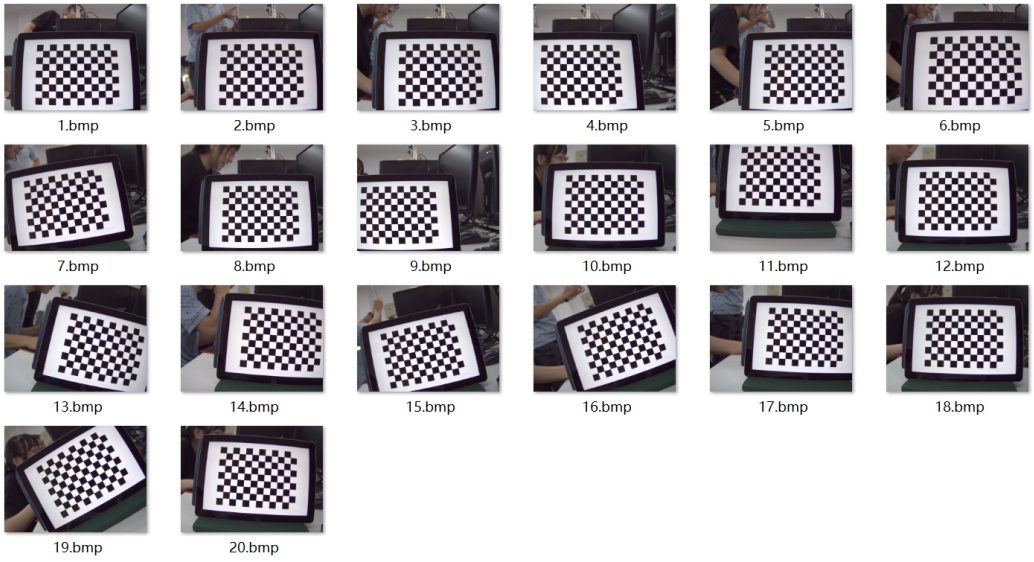
Task4 增强现实，在输入图像上投射部分3D点

- 在世界坐标系中选取一些有趣的点，根据内外参矩阵将其投影到图像像素坐标系，在对应图像中画出并观察。

实验过程和结果

Task1 相机的内外参标定

- 本实验我们通过移动相机位置拍摄20组棋牌图像数据来进行标定，具体实现工程详见文件夹***calib***
  - 相机拍摄数据如下



- 计算出相机内参矩阵信息如下

```
<camera_matrix type_id="opencv-matrix">
    <rows>3</rows>
    <cols>3</cols>
    <dt>d</dt>
    <data>
        7.1467922804866757e+03 0. 1296. 0. 7.1467922804866757e+03 972. 0. 0.
        1.</data></camera_matrix>
```

- 计算出相机外参矩阵信息如下

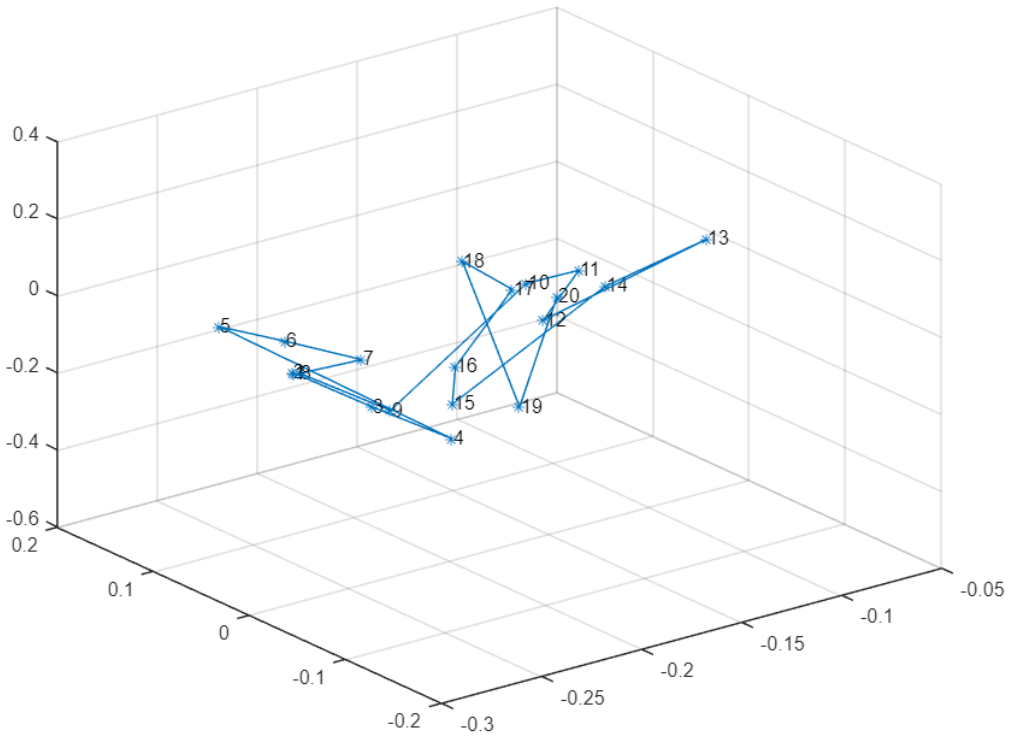
```
<extrinsic_parameters type_id="opencv-matrix">
<rows>20</rows> //20组数据
```

```
<cols>6</cols>  //每组数据对应的外参矩阵由6个参数构成，前3个为旋转参数，后3个为平移参数
<dt>d</dt>
<data>
    -2.7686971972156993e-01 1.9405004663472310e-03 -8.4554706730206580e-03
-5.2338386756889442e+01 -9.7682330954605039e+00 6.2416193821289517e+02
    -2.7654569508156329e-01 4.2875295235356315e-03 -7.4171337702818767e-03
-3.9643593561081474e+01 -9.9495838969902657e+00 6.1947453978282999e+02
    -2.7264574772749317e-01 -7.0405867757513993e-02 -1.6582122796150747e-02
-3.5699064412624026e+01 -9.9931387486599057e+00 6.1416772438552118e+02
    -2.6662590635330446e-01 -1.4157107575840350e-01 -2.6486774548942438e-02
-9.4252669508909307e+01 -9.2377930609847247e+00 6.2415658687313339e+02
    -2.7117502187125742e-01 9.1845991413645203e-02 3.9569366621246670e-03
-1.5838064520347556e+01 -1.0223424326410743e+01 6.4173271637976870e+02
    -2.5802232147734311e-01 5.0715850964367148e-02 -5.1975679682888538e-03
-2.9177890905322712e+01 -1.8763913950910784e+01 5.5506465239992292e+02
    -2.0819346067144390e-01 7.4438743774857305e-02 -1.5044331243680675e-01
-7.6290729963309730e+01 -1.8948923545894260e+01 6.4397912439549896e+02
    -2.7443072517858819e-01 1.1514138303242369e-03 -9.0536662714723777e-03
-3.9911907682338587e+01 -9.8460390411890852e+00 6.7954844084018498e+02
    -2.6606033476815238e-01 -7.6090315165273495e-02 -2.8455970895032358e-02
-1.0678536033934178e+02 -7.9931750641434869e+00 6.5493136069084085e+02
    -1.4432265893789861e-01 3.5810098637345855e-02 2.4516531210014955e-03
-5.4768921675455800e+01 -2.7414183229685761e+01 6.5194438014711443e+02
    -1.2333149982713895e-01 2.5176640981221864e-02 1.8713237215157569e-02
-5.5437130051947399e+01 -7.5510700547221205e+01 6.3811982310693020e+02
    -1.6477403996608403e-01 -2.4318633023725618e-02 4.3256505897300906e-03
-5.3624216108138320e+01 -3.3366567978012789e+01 6.5813384327787196e+02
    -1.1441521728153679e-01 -8.9529972182984099e-02 2.1830197188421185e-01
1.6819840377027502e+00 -2.9091287112790639e+01 6.3611684826251121e+02
    -1.4116458481142957e-01 -3.9820170338548386e-02 7.7095256236896517e-02
-1.3093956064183703e+00 -4.0892423165736126e+01 6.3005245719161826e+02
    -1.9209576665050135e-01 1.3603246821543749e-02 -2.1874800867856356e-01
-7.2380245665862290e+01 1.2111369061024289e+00 6.7794218474061427e+02
    -1.3564490866955356e-01 1.2817075287710472e-01 -3.3294473498393623e-01
-3.4335235324712897e+01 -1.3236250715449538e+01 6.6401352006733532e+02
    -1.6513046524582881e-01 7.3539788750613018e-03 4.8457880292578415e-02
-1.0250468191943078e+01 -4.2412244670614704e+01 6.7059684322773023e+02
    -1.5454578053147205e-01 8.1462853757251680e-02 2.1817523824356366e-02
-4.2040008812505668e+01 -3.9363171167777182e+01 6.6656783734113151e+02
    -9.2742315174486001e-02 1.5068920150385137e-01 -5.2286424816212518e-01
-5.3841135813384852e+01 -1.4943963580149108e+01 6.6215030558177932e+02
    -1.6040577331259179e-01 -2.9533387956331824e-02 6.5807955436433030e-02
-4.2645122721521837e+01 -4.3705128459946415e+01
6.5666098390267655e+02</data></extrinsic_parameters>
```

Task2 探究重投影误差影响因子

- 探究拍摄角度对重投影误差的影响

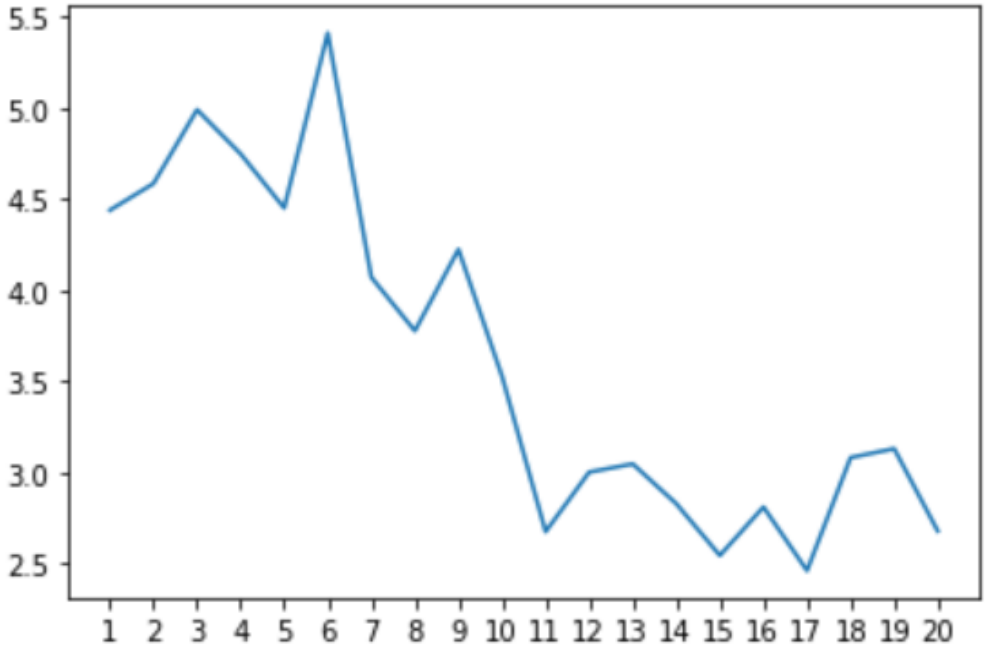
- 在Task1中我们选取了20组不同拍摄角度下的棋盘图像，将外参矩阵中的旋转参数绘制三维图像，如下图所示



- 计算出重投影误差如下：

```
<per_view_reprojection_errors type_id="opencv-matrix">
    <rows>20</rows>
    <cols>1</cols>
    <dt>f</dt>
    <data>
        4.43985748e+00 4.58611202e+00 4.99191093e+00 4.74948502e+00
        4.45187855e+00 5.41345787e+00 4.07121325e+00 3.77667904e+00
        4.22449923e+00 3.52721906e+00 2.67319655e+00 3.00046992e+00
        3.04542112e+00 2.82669473e+00 2.54249907e+00 2.80882645e+00
        2.45928144e+00 3.08040547e+00 3.12914181e+00
2.67694426e+00</data></per_view_reprojection_errors>
```

- 绘制重投影误差的二维图像



- 由图像数据对比可知，当拍摄偏移角度越大时，重投影误差越大

- 探究棋牌遮挡对重投影误差的影响
  - 考虑到棋盘遮挡对标定结果的影响，我们重新拍摄了多组存在遮挡的图像数据并与同一位置未被遮挡的图像进行对比，删去标定失败的图像后共计4组图像数据
    相机拍摄数据如下


1.bmp


2.bmp

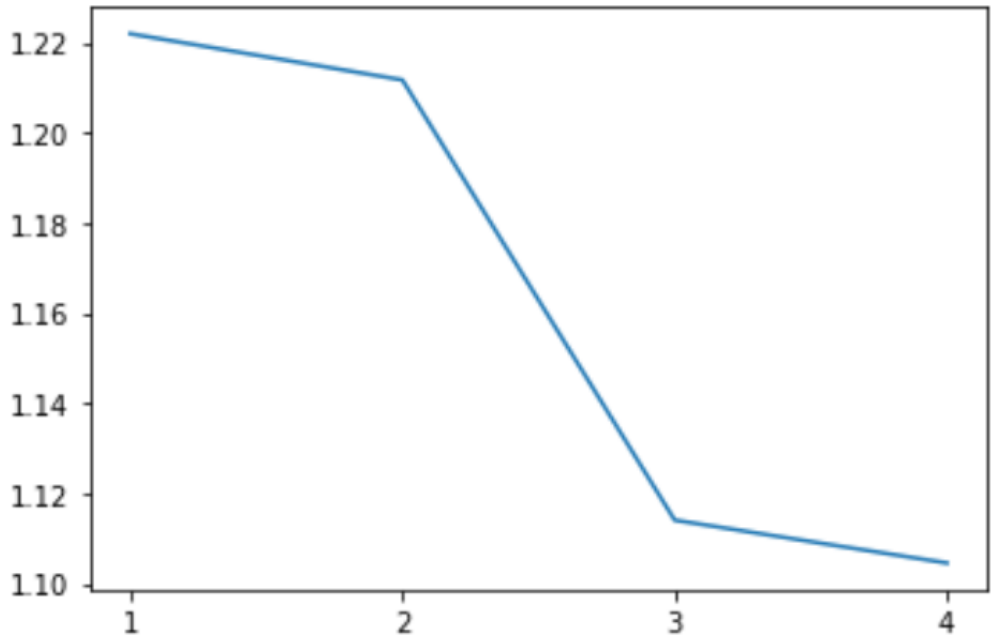

3.bmp


4.bmp

  - 计算出重投影误差如下

```
  <per_view_reprojection_errors type_id="opencv-matrix">
  <rows>4</rows>
  <cols>1</cols>
  <dt>f</dt>
  <data>
  1.22202301e+00 1.21175766e+00 1.11403012e+00 1.10451210e+00 </data>
</per_view_reprojection_errors>
```

- 绘制二维图像



  - 如果遮挡导致OpenCV无法正确识别所有格点，则此图无法作废，无法用于标定。考虑到多幅棋盘遮盖图像未能成功标定且实验存在一定误差，最后进行整理的数据有限，我们在结合实验数据和查阅资料的基础上发现：部分遮挡会影响棋盘格点的识别精度，势必引入更大的标定重投影误差。

- 探究图片数目对重投影误差的影响
  - 我们选取了不同数目的图像组进行标定，结合标定过程这一以重投影误差为目标函数、以内外参为参数的非线性优化过程，当组内图像数目较少时，容易过拟合，重投影平均误差较小；随着组内图像数目的增加，内外参的泛化能力提升，重投影平均误差增加，增加到一定程度后趋于稳定。

Task3 输出估算出的相机中心和棋盘

- 估算相机位姿
  - 工程project主要用于估计相机位姿及投影3D点，project.cpp文件负责正逆投影，xml.cpp负责解析xml文件。
  - 读入内外参

```
    string ex_name =
"D:\\Courses\\2020\\iavi\\lab2\\project\\extrinsic_parameters.xml";
    FileStorage e_fs(ex_name, FileStorage::READ);

    e_fs.open(ex_name, FileStorage::READ);

    Mat extrinsic_parameters = Mat(n_frames, 6, CV_32FC1,
Scalar::all(0));
    e_fs["extrinsic_parameters"] >> extrinsic_parameters;
    //cout << extrinsic_parameters;

    e_fs.release();
```

  - 投影部分
    - 平移、旋转矩阵的生成

```
MatrixXd IPM::rotateX(double r)
{
    MatrixXd R(4, 4);
    r = r * PI / 180.0;
    R << 1, 0, 0, 0, 0, cos(r), -sin(r), 0, 0, sin(r), cos(r), 0, 0,
0, 0, 1;
    return R;
}

MatrixXd IPM::rotateY(double r)
{
    MatrixXd R(4, 4);
    r = r * PI / 180.0;
    R << cos(r), 0, sin(r), 0, 0, 1, 0, 0, -sin(r), 0, cos(r), 0, 0,
0, 0, 1;
    return R;
}

MatrixXd IPM::dtranslate(double x, double y, double z)
{
    MatrixXd R(4, 4);
    R << 1,0,0,x,0,1,0,y,0,0,1,z, 0, 0, 0, 1;
    return R;
}
```

- 初始化内外参矩阵

```
void IPM::init(Mat extrinsic_para, Mat cameraMatrix)
{
    extrinsic_para_ = extrinsic_para;

    camera_matrix_ = MatrixXd(4, 4);
    MatrixXd CM(3, 3);
    // camera_matrix_ << cameraMatrix;
    cv2eigen(cameraMatrix, CM);
    camera_matrix_ << CM(0,0), 0, CM(0, 2), 0, 0, CM(1, 1), CM(1,
2), 0, 0, 0, 1, 0, 0, 0, 0, 1;
    cout << endl << "CM:" << camera_matrix_ << endl;

    auto RXw2p = extrinsic_para_.at<float>(0,0);
    auto RYw2p = extrinsic_para_.at<float>(0, 1);
    auto RZw2p = extrinsic_para_.at<float>(0, 2);

    auto Xw2p = extrinsic_para_.at<float>(0, 3);
    auto Yw2p = extrinsic_para_.at<float>(0, 4);
    auto Zw2p = extrinsic_para_.at<float>(0, 5);

    MatrixXd w2p(4, 4);
    w2p = rotateZ(RZw2p) * rotateY(RYw2p) * rotateX(RXw2p) *
dtranslate(Xw2p, Yw2p, Zw2p);
    cout << endl << "EM:" << w2p << endl;

    A_ = camera_matrix_ * w2p;
    inv_A_ = A_.inverse();
}
```

- 正投影，变换矩阵左乘世界坐标系下的其次坐标。

```
Point2d IPM::project(double x_real, double y_real, double z_real)
{
    MatrixXd real(4, 1);
    real << x_real, y_real, z_real, 1;

    MatrixXd pointImage = A_ * real;
    double u = pointImage(0) / pointImage(2);
    double v = pointImage(1) / pointImage(2);

    return Point2d(u, v);
}
```

- 逆投影，逆变换矩阵左乘图像坐标。

```
Point2d IPM::inverseProject(double u, double v, double z_real)
{
  double c1 = inv_A_(2, 0);
  double c2 = inv_A_(2, 1);
  double c3 = inv_A_(2, 2);
  double c4 = inv_A_(2, 3);

  double s = (z_real - c4) / (u * c1 + v * c2 + c3);

  MatrixXd foo(4, 1);
  foo << s * u, s * v, s, 1;

  MatrixXd res(4, 1);
  res = inv_A_ * foo;

  double x = res(0);
  double y = res(1);
  return Point2d(x, y);
}
```

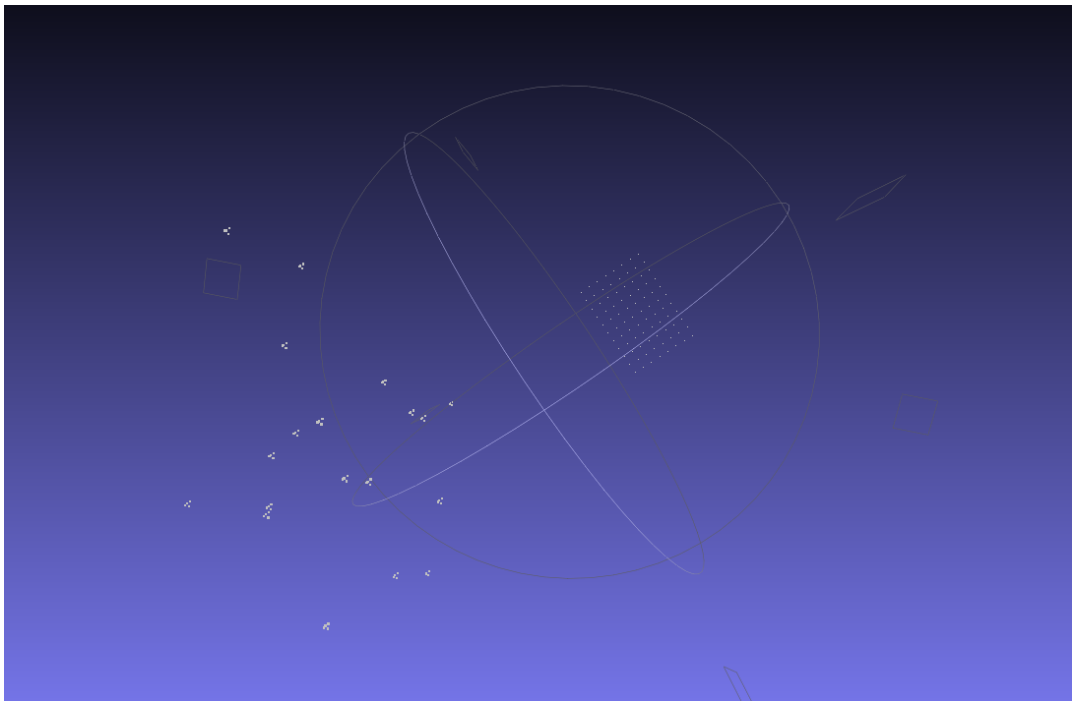
- 对每个checker，选取相机坐标系下的(0,0,0),(0,1,0),(1,0,0),(0,0,1)四个点来代表相机，根据外参对其进行平移旋转，计算出世界坐标。具体实现调用rotate_camera函数，输入相机外参，返回相机的姿态。

```
MatrixXd rotate_camera(double rx, double ry, double rz, double x, double
y, double z, double w)
{
  IPM ipm;
  MatrixXd R(4, 4);
  R = ipm.rotateZ(rz) * ipm.rotateY(ry) * ipm.rotateX(rx);
  MatrixXd point(4, 1);
  point << x, y, z, w;
  MatrixXd new_point = R * point;
  MatrixXd cnt(1, 3);
  cnt = new_point.block<3, 1>(0, 0).transpose();
  return cnt;
}
```

- ply文件

- 将上一步得到的代表每一个相机位姿的4个点坐标写入ply文件，用meshlab打开点云文件观察。
- ply文件包括头和体两部分，其中文件头规定了元素个数、属性信息，体部分则为具体的元素描述。详情见final_camera_chessboard.ply文件。

```
element vertex 168
property float x
property float y
property float z
```



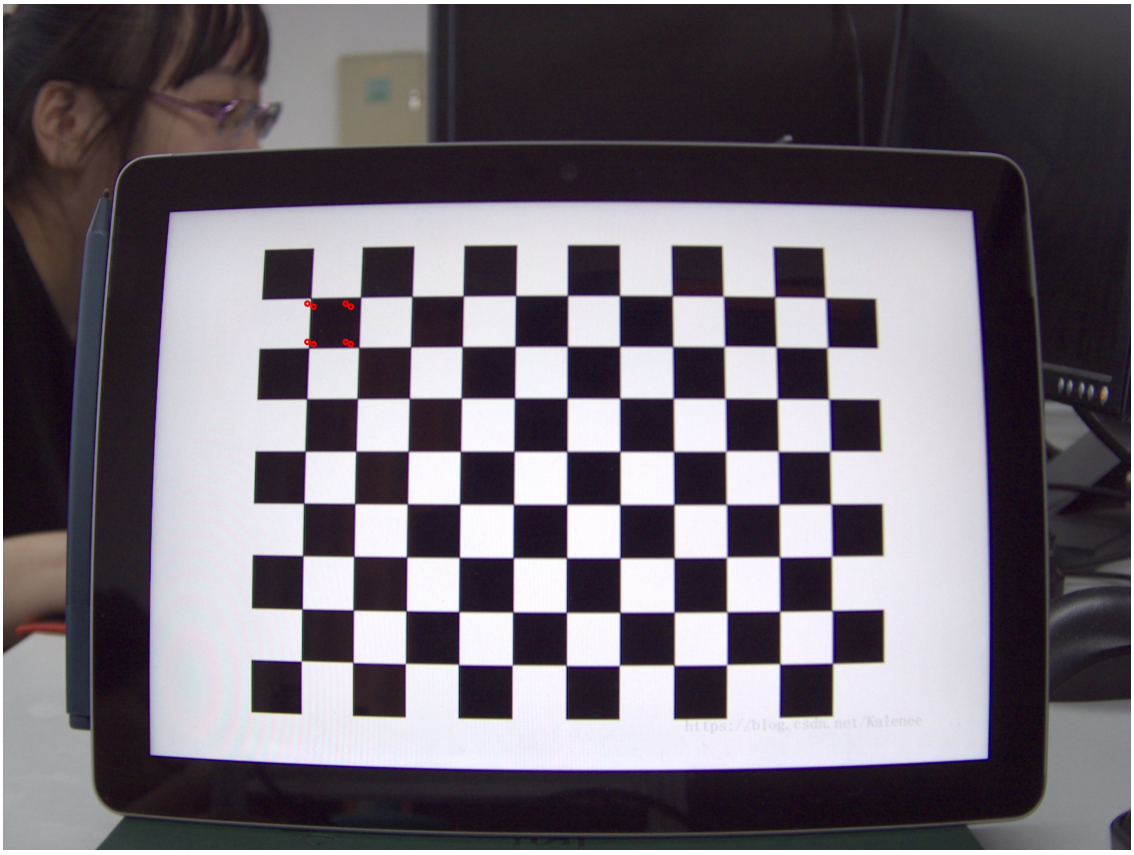
Task4 增强现实，在输入图像上投射部分3D点

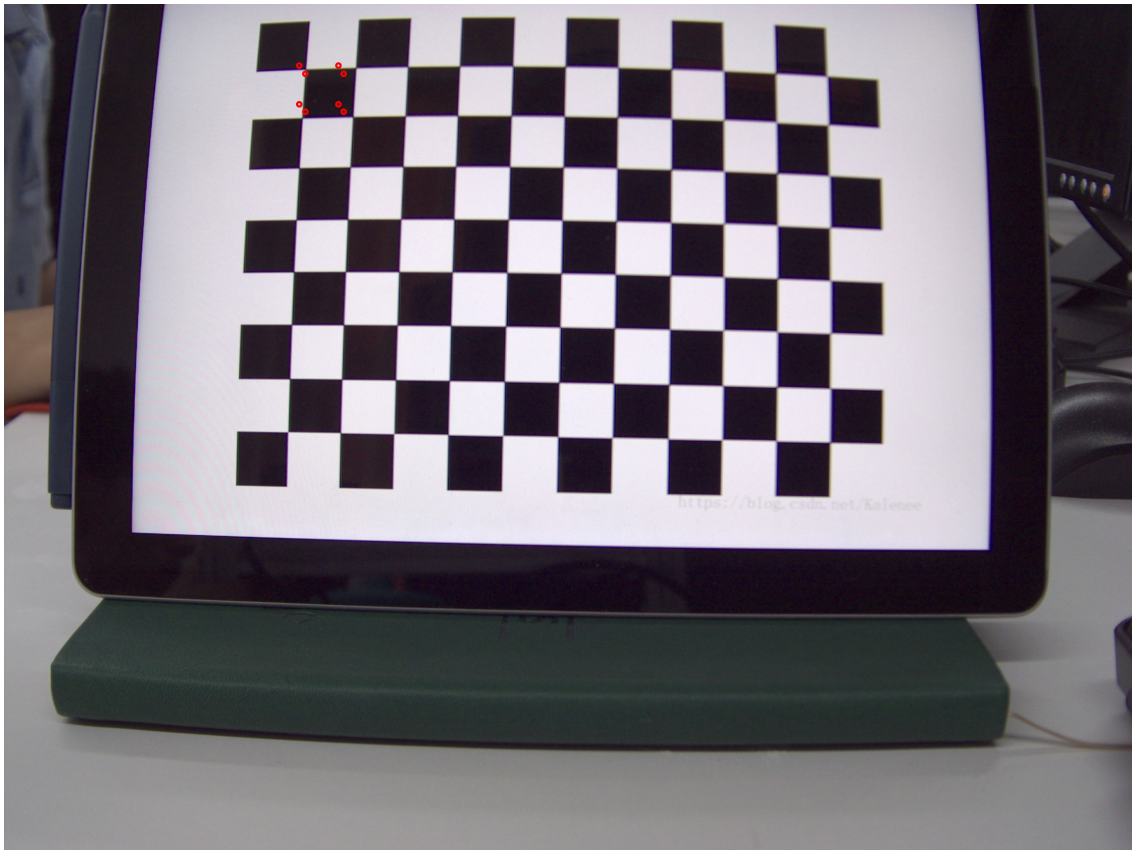
- 投影一个世界坐标系原点附近的立方体到每张图片，观察并保存。

```
    for (int i = 0; i < n_frames; i++)
    {
        vector<Point2d> v;
        IPM ipm;
        ipm.init(extrinsic_parameters.row(i), cameraMatrix);
        v.push_back(ipm.project(0, 0, 15));
        v.push_back(ipm.project(0, 8, 15));
        v.push_back(ipm.project(8, 0, 15));
        v.push_back(ipm.project(8, 8, 15));
        v.push_back(ipm.project(0, 0, 0));
        v.push_back(ipm.project(0, 8, 0));
        v.push_back(ipm.project(8, 0, 0));
        v.push_back(ipm.project(8, 8, 0));

        int flag = 1;
        for (vector<Point2d>::iterator iter = v.begin(); iter != v.end();
iter++)
        {
            std::cout << *iter << endl;
```

```
            if ((*iter).x > 2500 || (*iter).x < 0 || (*iter).y > 1900 ||
(*iter).y < 0)
            {
                flag = 0;
            }
        }
        cout << endl;

        if (!flag) continue;
        Mat src =
imread("D:\\Courses\\2020\\iavi\\lab2\\MyIAVI\\"+to_string(i+1)+".bmp");

        for (vector<Point2d>::iterator iter = v.begin(); iter != v.end();
iter++)
        {
            std::cout << *iter << endl;
            cv::Point point;//特征点，用以画在图像中
            point.x = (*iter).x;//特征点在图像中横坐标
            point.y = (*iter).y;//特征点在图像中纵坐标
            cv::circle(src, point, 5, cv::Scalar(0, 0, 255),3);
        }
        cv::namedWindow("src", CV_WINDOW_NORMAL);
        imshow("src", src);
        cv::imwrite("D:\\Courses\\2020\\iavi\\lab2\\MyIAV\\" + to_string(i +
1) + ".jpg", src);
        waitKey(0);
    }
```

立方体在棋盘左上角内格点附近，有一定的立体感，投影结果和预期相符。





参考资料

1. opencv--相机的内外参标定
2. 旋转矩阵