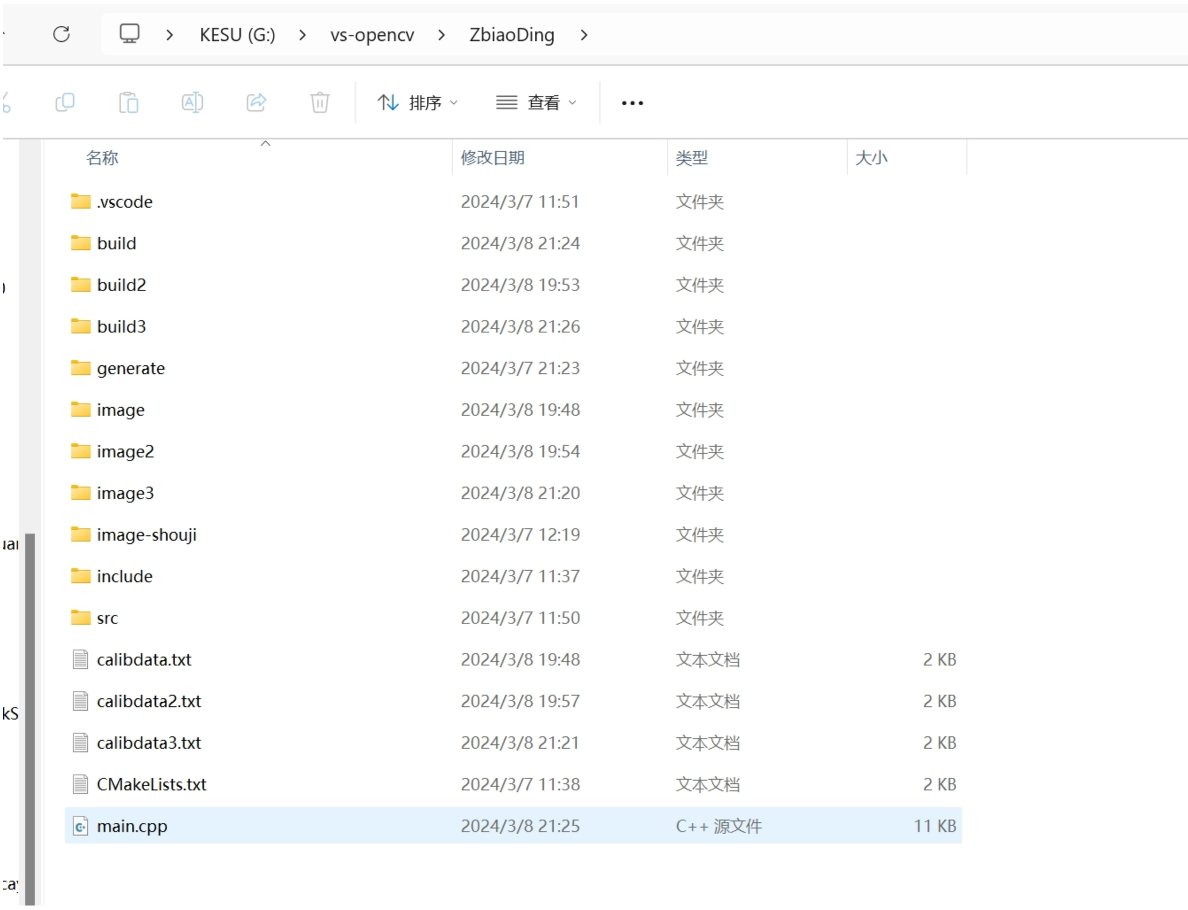
使用opencv标定工业相机

之前vscode配置过opencv库，因此此处不再赘述。

文件结构

我的文件架构如下：



image是相机01的图片

image2是相机02的图片位置

image3是相机03的图片位置

- 01--步兵
- 02--哨兵上云台
- 03--哨兵下云台

依次的编译文件夹是build build2 build3 标定完成的图片和相机参数在对应build文件夹中。

程序：（依赖opencv库）

```
#include <iostream>
#include <fstream>
#include <io.h>
#include "opencv2/opencv.hpp"
#include <opencv2/imgproc/types_c.h>
using namespace cv;
using namespace std;

void LoadImages_TXT(string filename, vector<string> &imageNames)
{
    // step 1: open file
    ifstream file(filename);

    if (!file)
    {
        cout << "Error! File " << filename << " does not exist!" << endl;
        return;
    }
    else
    {
        cout << "Open file " << filename << " successfully!" << endl;
    }
    // step 2: load data
    imageNames.clear();

    while (!file.eof())
    {
        string imageName;
        file >> imageName;

        if (file.eof())
            break;
        imageNames.push_back(imageName);
        cout << "Load Image, imageName: " << imageName << endl;
    }
    file.close();
    return;
}

int main()
{
```

```
    std::ofstream fout("caliberation_result.txt"); /* File to save calibration results */
    // Read each image, extract corners, and refine corners
    std::cout << "Start extracting corners..." << std::endl;
    int image_count = 0;                        /* Number of images */
    Size image_size;                            /* Size of images */
    Size board_size = Size(12, 8);              /*这是表示标定纸是12x8个角点8*/
    vector<Point2f> image_points_buf;           /* 这是一个二维向量，用来保存每幅图像上检测到的角点 */
    vector<vector<Point2f>> image_points_seq; /* 保存检测到的所有角点 */
    string filename;
    int count = -1; // Number of corners.

    // string dataFileName = "calibdata.txt"; /* Path to the image files used for calibration */
    // string dataFileName = "G:\\vs-opencv\\ZbiaoDing\\calibdata.txt";

    // string dataFileName = "G:\\vs-opencv\\ZbiaoDing\\calibdata2.txt";
    string dataFileName = "G:\\vs-opencv\\ZbiaoDing\\calibdata3.txt";
    vector<string> imageNames;
    LoadImages_TXT(dataFileName, imageNames); // 加载图片文件名

    for (int i = 0; i < imageNames.size(); i++)
    {
        image_count++;
        // For observation and verification   // 用于观察和验证
        cout << "image_count = " << image_count << endl;
        /* Verification output */ // 验证输出
        cout << "-->count = " << count << endl;

        string imageName = imageNames[i];
        Mat imageInput = imread(imageName);

        imshow("Camera Calibration", imageInput); // Display image
        waitKey(100);                             // Pause for 1s 每展示一张图片等待1s

        if (image_count == 1) // Get image width and height when reading the first image
        {
            image_size.width = imageInput.cols;
            image_size.height = imageInput.rows;
            cout << "image_size.width = " << image_size.width << endl;
            cout << "image_size.height = " << image_size.height << endl;
        }

        /* Extract corners */
        if (0 == findChessboardCorners(imageInput, board_size, image_points_buf))
        {
            cout << "Cannot find chessboard corners!" << endl;
            // exit(1) ;
        }
        else
        {
```

```
            Mat view_gray;
            cvtColor(imageInput, view_gray, CV_RGB2GRAY);
            /* Refine corners */
            find4QuadCornerSubpix(view_gray, image_points_buf, Size(5, 5)); //
Refine the detected corners

            image_points_seq.push_back(image_points_buf); // Save the refined
corners
            /* Display corners on the image */
            //
drawChessboardCorners(view_gray,board_size,image_points_buf,true); // Used to
mark corners in the image
            drawChessboardCorners(imageInput, board_size, image_points_buf,
true); // Used to mark corners in the image
            imshow("Camera Calibration", imageInput);
   // Display image
            imwrite("Calibration" + to_string(image_count) + ".png",
imageInput);  // Display image
            waitKey(100);
   // Pause for 0.1s
        }
    }

    int total = image_points_seq.size();
    cout << "total = " << total << endl;
    int CornerNum = board_size.width * board_size.height; // Total number of
corners on each image
    for (int ii = 0; ii < total; ii++)
    {
        if (0 == ii % CornerNum) // 24 is the number of corners per image. This
condition is used to output the image number for console viewing
        {
            int i = -1;
            i = ii / CornerNum;
            int j = i + 1;
            cout << "--> Data of image " << j << " --> : " << endl;
        }
        if (0 == ii % 3) // This condition is used to format the output for
console viewing (3 image information displayed in one line)
        {
            cout << endl;
        }
        else
        {
            cout.width(10);
        }
        // Output all corners
        cout << " -->" << image_points_seq[ii][0].x;
        cout << " -->" << image_points_seq[ii][0].y;
    }
    cout << "Corner extraction completed!" << endl;

    //
    // Step 2: 2D calibration
    //
```

```
    // Camera calibration
    cout << "Start calibration..." << endl;
    /* Chessboard 3D information */
    Size square_size = Size(5, 5); /* Size of each chessboard square obtained by
actual measurement */
    cout << "!!!!!!!!!!!!!!!!!!!!!!!" << endl;
    cout << square_size.height << endl;
    cout << square_size.width << endl;
    vector<vector<Point3f>> object_points; /* Store the 3D coordinates of
corners on different images */
    /* Internal and external parameters */
    Mat cameraMatrix = Mat(3, 3, CV_32FC1, Scalar::all(0)); /* Camera internal
parameter matrix */
    vector<int> point_counts;                               // Number of corners
in each image
    Mat distCoeffs = Mat(1, 5, CV_32FC1, Scalar::all(0));   /* 5 distortion
coefficients of the camera: k1, k2, p1, p2, k3 */
    vector<Mat> tvecsMat;                                   /* Translation
vectors of each image */
    vector<Mat> rvecsMat;                                   /* Rotation vectors
of each image */
    /* Initialize the 3D coordinates of corners on the calibration board */
    int i, j, t;
    for (t = 0; t < image_count; t++) // Number of images
    {
        vector<Point3f> tempPointSet;
        for (i = 0; i < board_size.height; i++)
        {
            for (j = 0; j < board_size.width; j++)
            {
                Point3f realPoint;
                /* Assume that the calibration board is placed on the z=0 plane
in the world coordinate system */
                realPoint.x = i * square_size.height;
                realPoint.y = j * square_size.width;
                realPoint.z = 0;
                tempPointSet.push_back(realPoint);
            }
        }
        object_points.push_back(tempPointSet);
    }
    /* Initialize the number of corners in each image, assuming that the entire
calibration board can be seen in each image */
    for (i = 0; i < image_count; i++)
    {
        point_counts.push_back(board_size.width * board_size.height);
    }
    /* Start calibration */
    calibrateCamera(object_points, image_points_seq, image_size, cameraMatrix,
distCoeffs, rvecsMat, tvecsMat, 0);
    // cout<<tvecsMat[0]<<endl;
    cout << "Calibration completed!" << endl;
    // Evaluate the calibration results
    cout << "Start evaluating calibration results..." << endl;
```

```
    double total_err = 0.0;        /* Total average error of all images */
    double err = 0.0;              /* Average error of each image */
    vector<Point2f> image_points2; /* Store the reprojected points */
    cout << "\tAverage error of each image:" << endl;
    fout << "Average error of each image:" << endl;
    for (i = 0; i < image_count; i++)
    {
        vector<Point3f> tempPointSet = object_points[i];
        /* Calculate the new 2D projection points by reprojecting the 3D points
in space using the obtained camera internal and external parameters */
        projectPoints(tempPointSet, rvecsMat[i], tvecsMat[i], cameraMatrix,
distCoeffs, image_points2);
        /* Calculate the error between the new projection points and the
original projection points */
        vector<Point2f> tempImagePoint = image_points_seq[i]; // Original 2D
points
        Mat tempImagePointMat = Mat(1, tempImagePoint.size(), CV_32FC2);
        Mat image_points2Mat = Mat(1, image_points2.size(), CV_32FC2); // 32-bit
floating-point, 2-channel
        for (int j = 0; j < tempImagePoint.size(); j++)                // j
corresponds to the number of 2D points
        {
            image_points2Mat.at<Vec2f>(0, j) = Vec2f(image_points2[j].x,
image_points2[j].y);
            tempImagePointMat.at<Vec2f>(0, j) = Vec2f(tempImagePoint[j].x,
tempImagePoint[j].y);
        }
        err = norm(image_points2Mat, tempImagePointMat, NORM_L2);
        total_err += err /= point_counts[i];
        std::cout << "Average error of image " << i + 1 << ": " << err << "
pixels" << endl;
        fout << "Average error of image " << i + 1 << ": " << err << " pixels"
<< endl;
    }
    std::cout << "Overall average error: " << total_err / image_count << "
pixels" << endl;
    fout << "Overall average error: " << total_err / image_count << " pixels" <<
endl
         << endl;
    std::cout << "Evaluation completed!" << endl;

    // Save calibration results
    std::cout << "Start saving calibration results..." << endl;
    Mat rotation_matrix = Mat(3, 3, CV_32FC1, Scalar::all(0)); /* Rotation
matrix of each image */
    fout << "Camera internal parameter matrix:" << endl;
    fout << cameraMatrix << endl
         << endl;
    fout << "Distortion coefficients:" << endl;
    fout << distCoeffs << endl;
    for (int i = 0; i < image_count; i++)
    {
        fout << "Rotation vector of image " << i + 1 << ":" << endl;
        fout << rvecsMat[i] << endl;
        /* Convert the rotation vector to the corresponding rotation matrix */
```

```
        Rodrigues(rvecsMat[i], rotation_matrix);
        fout << "Rotation matrix of image " << i + 1 << ":" << endl;
        fout << rotation_matrix << endl;
        fout << "Translation vector of image " << i + 1 << ":" << endl;
        fout << tvecsMat[i] << endl
             << endl;
    }
    std::cout << "Saving completed" << endl;
    fout << endl;
}
```

注意

注意1:

```
int main()
{
    std::ofstream fout("caliberation_result.txt"); /* File to save calibration results */
    // Read each image, extract corners, and refine corners
    std::cout << "Start extracting corners..." << std::endl;
    int image_count = 0;                    /* Number of images */
    Size image_size;                        /* Size of images */
    Size board_size = Size(12, 8);          /*这是表示标定纸是12×8个角点8*/
    vector<Point2f> image_points_buf;       /* 这是一个二维向量，用来保存每幅图像上检测到的角点 */
    vector<vector<Point2f>> image_points_seq; /* 保存检测到的所有角点 */
    string filename;
    int count = -1; // Number of corners.

    // string dataFileName = "calibdata.txt"; /* Path to the image files used for calibration */
```

把 board_size改成指定的大小 表示角点数

12*8个角点

根据你拍的标定纸的角点数确定

注意2:

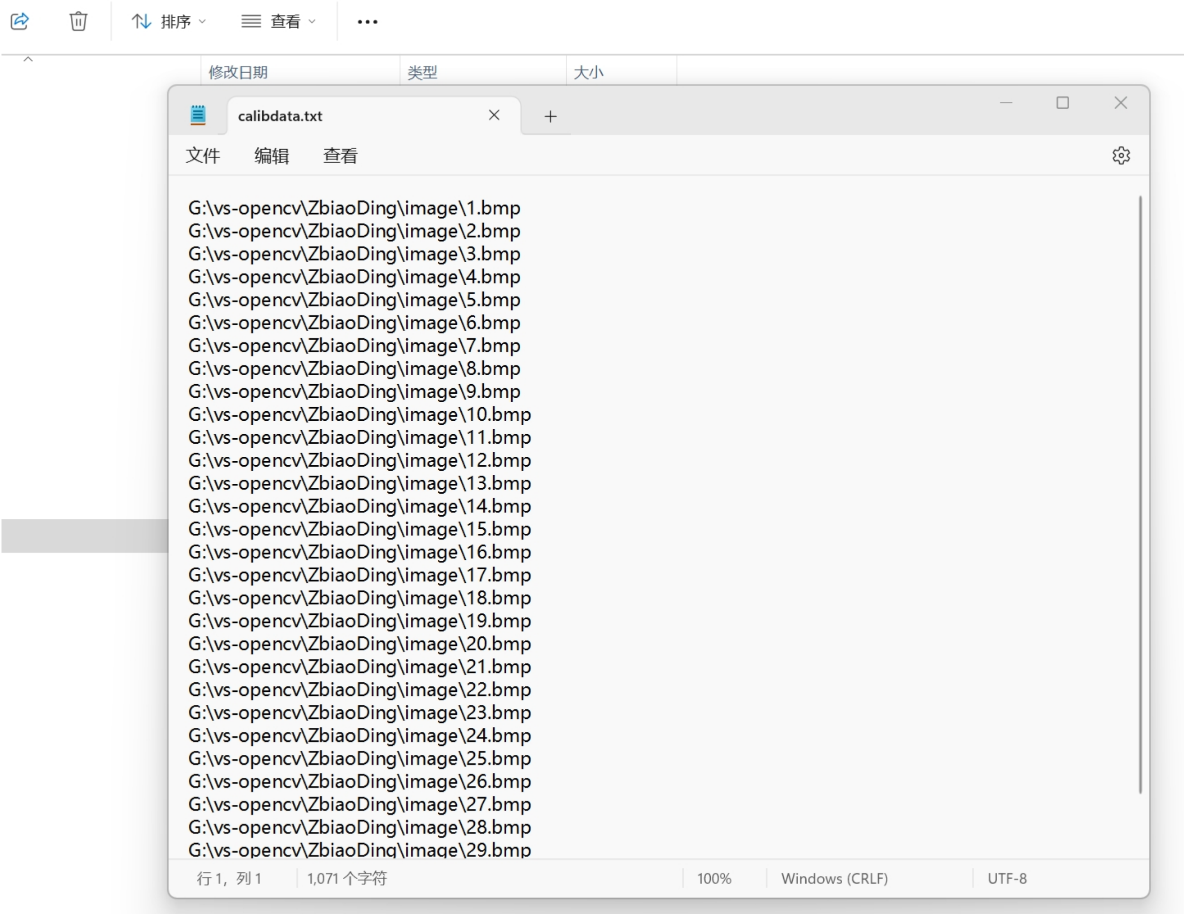
```
    string filename;
    int count = -1; // Number of corners.

    // string dataFileName = "calibdata.txt"; /* Path to the image files used for calibration */
    // string dataFileName = "G:\\vs-opencv\\ZbiaoDing\\calibdata.txt";

    // string dataFileName = "G:\\vs-opencv\\ZbiaoDing\\calibdata2.txt";
    string dataFileName = "G:\\vs-opencv\\ZbiaoDing\\calibdata3.txt";
    vector<string> imageNames;
    LoadImages_TXT(dataFileName, imageNames); // 加载图片文件名

    for (int i = 0; i < imageNames.size(); i++)
    {
```

注意把文本文件路径设置正确

设置绝对路径

注意3:

文本文件路径设置如下:



运行

进入指定的build文件夹

cmake

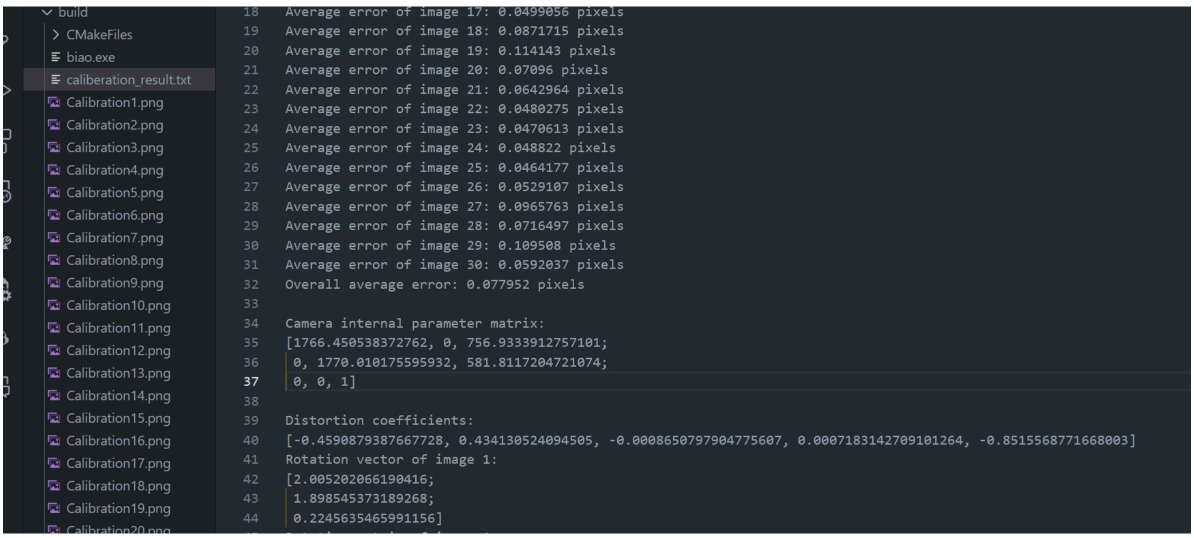
make

执行程序

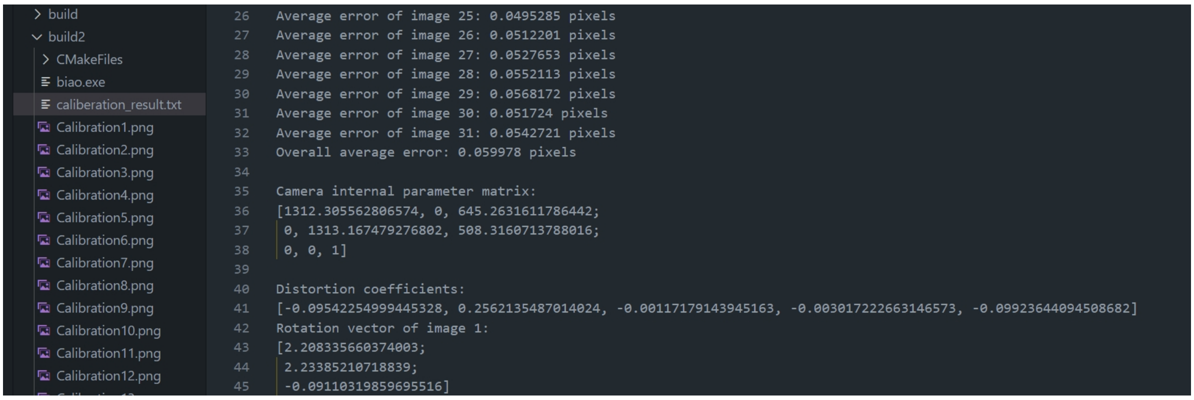
结果

下面是各个build之后的各个相机的参数结果：

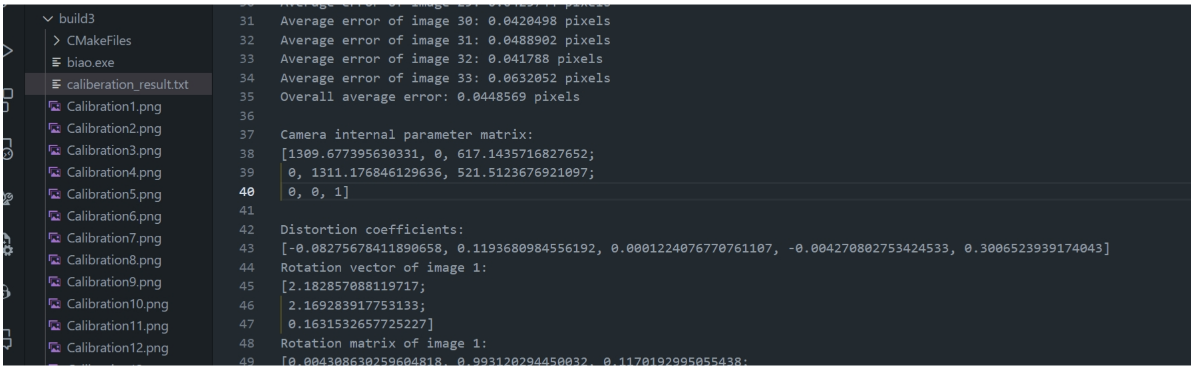
01



02


build
build2
CMakeFiles
biao.exe
caliberation_result.txt
Calibration1.png
Calibration2.png
Calibration3.png
Calibration4.png
Calibration5.png
Calibration6.png
Calibration7.png
Calibration8.png
Calibration9.png
Calibration10.png
Calibration11.png
Calibration12.png
Average error of image 25: 0.0495285 pixels
Average error of image 26: 0.0512201 pixels
Average error of image 27: 0.0527653 pixels
Average error of image 28: 0.0552113 pixels
Average error of image 29: 0.0568172 pixels
Average error of image 30: 0.051724 pixels
Average error of image 31: 0.0542721 pixels
Overall average error: 0.059978 pixels
Camera internal parameter matrix:
[1312.305562806574, 0, 645.2631611786442;
0, 1313.167479276802, 508.3160713788016;
0, 0, 1]
Distortion coefficients:
[-0.09542254999445328, 0.2562135487014024, -0.00117179143945163, -0.003017222663146573, -0.09923644094508682]
Rotation vector of image 1:
[2.208335660374003;
2.23385210718839;
-0.09110319859695516]

03


build3
CMakeFiles
biao.exe
caliberation_result.txt
Calibration1.png
Calibration2.png
Calibration3.png
Calibration4.png
Calibration5.png
Calibration6.png
Calibration7.png
Calibration8.png
Calibration9.png
Calibration10.png
Calibration11.png
Calibration12.png
Average error of image 30: 0.0420498 pixels
Average error of image 31: 0.0488902 pixels
Average error of image 32: 0.041788 pixels
Average error of image 33: 0.0632052 pixels
Overall average error: 0.0448569 pixels
Camera internal parameter matrix:
[1309.677395630331, 0, 617.1435716827652;
0, 1311.176846129636, 521.5123676921097;
0, 0, 1]
Distortion coefficients:
[-0.08275678411890658, 0.1193680984556192, 0.0001224076770761107, -0.004270802753424533, 0.3006523939174043]
Rotation vector of image 1:
[2.182857088119717;
2.169283917753133;
0.1631532657725227]
Rotation matrix of image 1: