//单目摄像机标定

void CimageProcessDlg::OnBnClickedButton23()

{

// TODO: 在此添加控件通知处理程序代码

ifstream fin("F://program//image//camer\_cab//imageDatalist\_right.txt"); /\* 标定所用图像文件的路径 \*/

ofstream fout("F://program//image//camer\_cab//caliberation\_result\_right.txt"); /\* 保存标定结果的文件 \*/

//读取每一幅图像，从中提取出角点，然后对角点进行亚像素精确化

\_cprintf("start to find corners………………\n");

int image\_count = 0; /\* 图像数量 \*/

Size image\_size; /\* 图像的尺寸 \*/

Size board\_size = Size(9, 6); /\* 标定板上每行、列的角点数 \*/

vector<Point2f> image\_points\_buf; /\* 缓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;//用于存储角点个数。

while (getline(fin, filename))

{

image\_count++;

// 用于观察检验输出

\_cprintf("image\_count = %d\n",image\_count);

Mat imageInput = imread(filename);

if (image\_count == 1) //读入第一张图片时获取图像宽高信息

{

image\_size.width = imageInput.cols;

image\_size.height = imageInput.rows;

\_cprintf("image\_size.width = %d\n",image\_size.width);

\_cprintf("image\_size.height = %d\n",image\_size.height);

}

/\* 提取角点 \*/

if (0 == findChessboardCorners(imageInput, board\_size, image\_points\_buf))

{

\_cprintf("can not find chessboard corners!\n"); //找不到角点

waitKey(0);

exit(1);

}

else

{

Mat view\_gray;

cvtColor(imageInput, view\_gray, CV\_RGB2GRAY);

/\* 亚像素精确化 \*/

find4QuadCornerSubpix(view\_gray, image\_points\_buf, Size(5, 5)); //对粗提取的角点进行精确化

//cornerSubPix(view\_gray,image\_points\_buf,Size(5,5),Size(-1,-1),TermCriteria(CV\_TERMCRIT\_EPS+CV\_TERMCRIT\_ITER,30,0.1));

image\_points\_seq.push\_back(image\_points\_buf); //保存亚像素角点

/\* 在图像上显示角点位置 \*/

drawChessboardCorners(view\_gray, board\_size, image\_points\_buf, false); //用于在图片中标记角点

imshow("Camera Calibration", view\_gray);//显示图片

waitKey(50);//暂停0.5S

}

}

int total = image\_points\_seq.size();

\_cprintf("total = %d\n",total);

int CornerNum = board\_size.width\*board\_size.height; //每张图片上总的角点数

for (int ii = 0; ii<total; ii++)

{

if (0 == ii%CornerNum)// 24 是每幅图片的角点个数。此判断语句是为了输出 图片号，便于控制台观看

{

int i = -1;

i = ii / CornerNum;

int j = i + 1;

\_cprintf("--> the %d image's data --> :",j);

}

if (0 == ii % 3) // 此判断语句，格式化输出，便于控制台查看

{

\_cprintf("\n");

}

else

{

cout.width(10);

}

//输出所有的角点

\_cprintf(" -->%f",image\_points\_seq[ii][0].x);

\_cprintf(" -->%f",image\_points\_seq[ii][0].y);

}

\_cprintf("find corner successfully！\n");

//以下是摄像机标定

\_cprintf("start calibration………………");

/\*棋盘三维信息\*/

Size square\_size = Size(10, 10); /\* 实际测量得到的标定板上每个棋盘格的大小 \*/

vector<vector<Point3f>> object\_points; /\* 保存标定板上角点的三维坐标 \*/

/\*内外参数\*/

Mat cameraMatrix = Mat(3, 3, CV\_32FC1, Scalar::all(0)); /\* 摄像机内参数矩阵 \*/

vector<int> point\_counts; // 每幅图像中角点的数量

Mat distCoeffs = Mat(1, 5, CV\_32FC1, Scalar::all(0)); /\* 摄像机的5个畸变系数：k1,k2,p1,p2,k3 \*/

vector<Mat> tvecsMat; /\* 每幅图像的旋转向量 \*/

vector<Mat> rvecsMat; /\* 每幅图像的平移向量 \*/

/\* 初始化标定板上角点的三维坐标 \*/

int i, j, t;

for (t = 0; t<image\_count; t++)

{

vector<Point3f> tempPointSet;

for (i = 0; i<board\_size.height; i++)

{

for (j = 0; j<board\_size.width; j++)

{

Point3f realPoint;

/\* 假设标定板放在世界坐标系中z=0的平面上 \*/

realPoint.x = i\*square\_size.width;

realPoint.y = j\*square\_size.height;

realPoint.z = 0;

tempPointSet.push\_back(realPoint);

}

}

object\_points.push\_back(tempPointSet);

}

/\* 初始化每幅图像中的角点数量，假定每幅图像中都可以看到完整的标定板 \*/

for (i = 0; i<image\_count; i++)

{

point\_counts.push\_back(board\_size.width\*board\_size.height);

}

/\* 开始标定 \*/

calibrateCamera(object\_points, image\_points\_seq, image\_size, cameraMatrix, distCoeffs, rvecsMat, tvecsMat, 0);

\_cprintf("calibrate successfully！\n");

//对标定结果进行评价

\_cprintf("start evaluate the calibration results………………\n");

double total\_err = 0.0; /\* 所有图像的平均误差的总和 \*/

double err = 0.0; /\* 每幅图像的平均误差 \*/

vector<Point2f> image\_points2; /\* 保存重新计算得到的投影点 \*/

\_cprintf("\t each image's calibration error：\n");

\_cprintf("each image's calibration error：\n");

for (i = 0; i<image\_count; i++)

{

vector<Point3f> tempPointSet = object\_points[i];

/\* 通过得到的摄像机内外参数，对空间的三维点进行重新投影计算，得到新的投影点 \*/

projectPoints(tempPointSet, rvecsMat[i], tvecsMat[i], cameraMatrix, distCoeffs, image\_points2);

/\* 计算新的投影点和旧的投影点之间的误差\*/

vector<Point2f> tempImagePoint = image\_points\_seq[i];

Mat tempImagePointMat = Mat(1, tempImagePoint.size(), CV\_32FC2);

Mat image\_points2Mat = Mat(1, image\_points2.size(), CV\_32FC2);

for (int j = 0; j < tempImagePoint.size(); j++)

{

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];

\_cprintf("the %d th image's average err: %f pixels\n", i + 1, err);

fout << "第" << i + 1 << "幅图像的平均误差：" << err << "像素" << endl;

}

\_cprintf(" the total average err：%f \n",total\_err / image\_count);

fout << "总体平均误差：" << total\_err / image\_count << "像素" << endl << endl;

\_cprintf("evaluate successfully！\n");

//保存定标结果

\_cprintf("start save calibration results………………\n");

Mat rotation\_matrix = Mat(3, 3, CV\_32FC1, Scalar::all(0)); /\* 保存每幅图像的旋转矩阵 \*/

fout << "相机内参数矩阵：" << endl;

fout << cameraMatrix << endl << endl;

fout << "畸变系数：\n";

fout << distCoeffs << endl << endl << endl;

for (int i = 0; i<image\_count; i++)

{

fout << "第" << i + 1 << "幅图像的旋转向量：" << endl;

fout << tvecsMat[i] << endl;

/\* 将旋转向量转换为相对应的旋转矩阵 \*/

Rodrigues(tvecsMat[i], rotation\_matrix);

fout << "第" << i + 1 << "幅图像的旋转矩阵：" << endl;

fout << rotation\_matrix << endl;

fout << "第" << i + 1 << "幅图像的平移向量：" << endl;

fout << rvecsMat[i] << endl << endl;

}

\_cprintf("save successfully\n");

fout << endl;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

显示定标结果

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Mat mapx = Mat(image\_size, CV\_32FC1);

Mat mapy = Mat(image\_size, CV\_32FC1);

Mat R = Mat::eye(3, 3, CV\_32F);

\_cprintf("save rectified image\n");

string imageFileName;

std::stringstream StrStm;

for (int i = 0; i != image\_count; i++)

{

\_cprintf("Frame # %d ...\n",i + 1);

initUndistortRectifyMap(cameraMatrix, distCoeffs, R, cameraMatrix, image\_size, CV\_32FC1, mapx, mapy);

StrStm.clear();

imageFileName.clear();

string filePath = "F://program//image//camer\_cab//chess";

StrStm << i + 1;

StrStm >> imageFileName;

filePath += imageFileName;

filePath += ".jpg";

Mat imageSource = imread("F://program//image//camer\_cab//left01.jpg");

Mat newimage = imageSource.clone();

//另一种不需要转换矩阵的方式

//undistort(imageSource,newimage,cameraMatrix,distCoeffs);

remap(imageSource, newimage, mapx, mapy, INTER\_LINEAR);

StrStm.clear();

filePath.clear();

filePath = "F://program//image//camer\_cab//chess";

StrStm << i + 1;

StrStm >> imageFileName;

filePath += imageFileName;

filePath += "\_d.jpg";

imwrite(filePath, newimage);

}

\_cprintf(" save successfully\n");

waitKey(0);

}

/\*计算标定板上模块的实际物理坐标\*/

void calRealPoint(vector<vector<Point3f>>& obj, int boardWidth, int boardHeight, int imgNumber, int squareSize)

{

vector<Point3f> imgpoint;

for (int rowIndex = 0; rowIndex < boardHeight; rowIndex++)

{

for (int colIndex = 0; colIndex < boardWidth; colIndex++)

{

imgpoint.push\_back(Point3f(rowIndex \* squareSize, colIndex \* squareSize, 0));

}

}

for (int imgIndex = 0; imgIndex < imgNumber; imgIndex++)

{

obj.push\_back(imgpoint);

}

}

Mat R, T, E, F;

Mat Rl, Rr, Pl, Pr, Q;

//映射表

Mat mapLx, mapLy, mapRx, mapRy;

Mat cameraMatrixL = (Mat\_<double>(3, 3) << 530.1397548683084, 0, 338.2680507680664,

0, 530.2291152852337, 232.4902023212199,

0, 0, 1);

//获得的畸变参数

Mat distCoeffL = (Mat\_<double>(5, 1) << -0.266294943795012, -0.0450330886310585, 0.0003024821418382528, -0.001243865371699451, 0.2973605735168139);

Mat cameraMatrixR = (Mat\_<double>(3, 3) << 530.1397548683084, 0, 338.2680507680664,

0, 530.2291152852337, 232.4902023212199,

0, 0, 1);

Mat distCoeffR = (Mat\_<double>(5, 1) << -0.266294943795012, -0.0450330886310585, 0.0003024821418382528, -0.001243865371699451, 0.2973605735168139);

void outputCameraParam(void)

{

/\*保存数据\*/

/\*输出数据\*/

FileStorage fs("intrisics.yml", FileStorage::WRITE);

if (fs.isOpened())

{

fs << "cameraMatrixL" << cameraMatrixL << "cameraDistcoeffL" << distCoeffL << "cameraMatrixR" << cameraMatrixR << "cameraDistcoeffR" << distCoeffR;

fs.release();

cout << "cameraMatrixL=:" << cameraMatrixL << endl << "cameraDistcoeffL=:" << distCoeffL << endl << "cameraMatrixR=:" << cameraMatrixR << endl << "cameraDistcoeffR=:" << distCoeffR << endl;

}

else

{

cout << "Error: can not save the intrinsics!!!!" << endl;

}

fs.open("extrinsics.yml", FileStorage::WRITE);

if (fs.isOpened())

{

fs << "R" << R << "T" << T << "Rl" << Rl << "Rr" << Rr << "Pl" << Pl << "Pr" << Pr << "Q" << Q;

cout << "R=" << R << endl << "T=" << T << endl << "Rl=" << Rl << endl << "Rr" << Rr << endl << "Pl" << Pl << endl << "Pr" << Pr << endl << "Q" << Q << endl;

fs.release();

}

else

{

cout << "Error: can not save the extrinsic parameters\n";

}

}

//双目摄像机标定

void CimageProcessDlg::OnBnClickedButton24()

{

// TODO: 在此添加控件通知处理程序代码

//摄像头的分辨率

const int imageWidth = 640;

const int imageHeight = 480;

//横向的角点数目

const int boardWidth = 9;

//纵向的角点数目

const int boardHeight = 6;

//总的角点数目

const int boardCorner = boardWidth \* boardHeight;

//相机标定时需要采用的图像帧数

const int frameNumber = 14;

//标定板黑白格子的大小 单位是mm

const int squareSize = 10;

//标定板的总内角点

const Size boardSize = Size(boardWidth, boardHeight);

Size imageSize = Size(imageWidth, imageHeight);

//R旋转矢量 T平移矢量 E本征矩阵 F基础矩阵

vector<Mat> rvecs; //R

vector<Mat> tvecs; //T

//左边摄像机所有照片角点的坐标集合

vector<vector<Point2f>> imagePointL;

//右边摄像机所有照片角点的坐标集合

vector<vector<Point2f>> imagePointR;

//各图像的角点的实际的物理坐标集合

vector<vector<Point3f>> objRealPoint;

//左边摄像机某一照片角点坐标集合

vector<Point2f> cornerL;

//右边摄像机某一照片角点坐标集合

vector<Point2f> cornerR;

Mat rgbImageL, grayImageL;

Mat rgbImageR, grayImageR;

Mat intrinsic;

Mat distortion\_coeff;

//校正旋转矩阵R，投影矩阵P，重投影矩阵Q

//映射表

Mat mapLx, mapLy, mapRx, mapRy;

Rect validROIL, validROIR;

//图像校正之后，会对图像进行裁剪，其中，validROI裁剪之后的区域

Mat img;

int goodFrameCount = 1;

while (goodFrameCount <= frameNumber)

{

char filename[100];

/\*读取左边的图像\*/

sprintf\_s(filename, "F://program//image//camer\_cab//left%02d.jpg", goodFrameCount);

rgbImageL = imread(filename, 1);

imshow("chessboardL", rgbImageL);

cvtColor(rgbImageL, grayImageL, CV\_BGR2GRAY);

/\*读取右边的图像\*/

sprintf\_s(filename, "F://program//image//camer\_cab//right%02d.jpg", goodFrameCount);

rgbImageR = imread(filename, 1);

cvtColor(rgbImageR, grayImageR, CV\_BGR2GRAY);

bool isFindL, isFindR;

isFindL = findChessboardCorners(rgbImageL, boardSize, cornerL);

isFindR = findChessboardCorners(rgbImageR, boardSize, cornerR);

if (isFindL == true && isFindR == true)

{

cornerSubPix(grayImageL, cornerL, Size(5, 5), Size(-1, 1), TermCriteria(CV\_TERMCRIT\_EPS | CV\_TERMCRIT\_ITER, 20, 0.1));

drawChessboardCorners(rgbImageL, boardSize, cornerL, isFindL);

imshow("chessboardL", rgbImageL);

imagePointL.push\_back(cornerL);

cornerSubPix(grayImageR, cornerR, Size(5, 5), Size(-1, -1), TermCriteria(CV\_TERMCRIT\_EPS | CV\_TERMCRIT\_ITER, 20, 0.1));

drawChessboardCorners(rgbImageR, boardSize, cornerR, isFindR);

imshow("chessboardR", rgbImageR);

imagePointR.push\_back(cornerR);

\_cprintf("the image %d is good\n",goodFrameCount);

goodFrameCount++;

}

else

{

\_cprintf("the image is bad please try again\n");

}

if (waitKey(10) == 'q')

{

break;

}

}

//计算实际的校正点的三维坐标，根据实际标定格子的大小来设置

calRealPoint(objRealPoint, boardWidth, boardHeight, frameNumber, squareSize);

\_cprintf("cal real successful\n");

//标定摄像头

double rms = stereoCalibrate(objRealPoint, imagePointL, imagePointR,

cameraMatrixL, distCoeffL,

cameraMatrixR, distCoeffR,

Size(imageWidth, imageHeight), R, T, E, F, CALIB\_USE\_INTRINSIC\_GUESS,

TermCriteria(TermCriteria::COUNT + TermCriteria::EPS, 100, 1e-5));

\_cprintf("Stereo Calibration done with RMS error = %f\n",rms);

stereoRectify(cameraMatrixL, distCoeffL, cameraMatrixR, distCoeffR, imageSize, R, T, Rl,

Rr, Pl, Pr, Q, CALIB\_ZERO\_DISPARITY, -1, imageSize, &validROIL, &validROIR);

//摄像机校正映射

initUndistortRectifyMap(cameraMatrixL, distCoeffL, Rl, Pl, imageSize, CV\_32FC1, mapLx, mapLy);

initUndistortRectifyMap(cameraMatrixR, distCoeffR, Rr, Pr, imageSize, CV\_32FC1, mapRx, mapRy);

Mat rectifyImageL, rectifyImageR;

cvtColor(grayImageL, rectifyImageL, CV\_GRAY2BGR);

cvtColor(grayImageR, rectifyImageR, CV\_GRAY2BGR);

imshow("RecitifyL Before", rectifyImageL);

imshow("RecitifyR Before", rectifyImageR);

//经过remap之后，左右相机的图像已经共面并且行对准了

Mat rectifyImageL2, rectifyImageR2;

remap(rectifyImageL, rectifyImageL2, mapLx, mapLy, INTER\_LINEAR);

remap(rectifyImageR, rectifyImageR2, mapRx, mapRy, INTER\_LINEAR);

imshow("rectifyImageL", rectifyImageL2);

imshow("rectifyImageR", rectifyImageR2);

outputCameraParam();

//显示校正结果

Mat canvas;

double sf;

int w, h;

sf = 600. / MAX(imageSize.width, imageSize.height);

w = cvRound(imageSize.width \* sf);

h = cvRound(imageSize.height \* sf);

canvas.create(h, w \* 2, CV\_8UC3);

//左图像画到画布上

Mat canvasPart = canvas(Rect(0, 0, w, h));

resize(rectifyImageL2, canvasPart, canvasPart.size(), 0, 0, INTER\_AREA);

Rect vroiL(cvRound(validROIL.x\*sf), cvRound(validROIL.y\*sf),

cvRound(validROIL.width\*sf), cvRound(validROIL.height\*sf));

rectangle(canvasPart, vroiL, Scalar(0, 0, 255), 3, 8);

\_cprintf("Painted ImageL\n");

//右图像画到画布上

canvasPart = canvas(Rect(w, 0, w, h));

resize(rectifyImageR2, canvasPart, canvasPart.size(), 0, 0, INTER\_LINEAR);

Rect vroiR(cvRound(validROIR.x\*sf), cvRound(validROIR.y\*sf),

cvRound(validROIR.width\*sf), cvRound(validROIR.height\*sf));

rectangle(canvasPart, vroiR, Scalar(0, 255, 0), 3, 8);

\_cprintf("Painted ImageR\n");

//画上对应的线条

for (int i = 0; i < canvas.rows; i += 16)

line(canvas, Point(0, i), Point(canvas.cols, i), Scalar(0, 255, 0), 1, 8);

imshow("rectified", canvas);

\_cprintf("wait key\n");

waitKey(0);

}

int getDisparityImage(cv::Mat& disparity, cv::Mat& disparityImage, bool isColor)

{

cv::Mat disp8u;

disp8u = disparity;

// 转换为伪彩色图像 或 灰度图像

if (isColor)

{

if (disparityImage.empty() || disparityImage.type() != CV\_8UC3 || disparityImage.size() != disparity.size())

{

disparityImage = cv::Mat::zeros(disparity.rows, disparity.cols, CV\_8UC3);

}

for (int y = 0; y<disparity.rows; y++)

{

for (int x = 0; x<disparity.cols; x++)

{

uchar val = disp8u.at<uchar>(y, x);

uchar r, g, b;

if (val == 0)

r = g = b = 0;

else

{

r = 255 - val;

g = val < 128 ? val \* 2 : (uchar)((255 - val) \* 2);

b = val;

}

disparityImage.at<cv::Vec3b>(y, x) = cv::Vec3b(b, g, r);

}

}

}

else

{

disp8u.copyTo(disparityImage);

}

return 1;

}

const int imageWidth = 640; //摄像头的分辨率

const int imageHeight = 480;

Size imageSize = Size(imageWidth, imageHeight);

Mat rgbImageL, grayImageL;

Mat rgbImageR, grayImageR;

Mat rectifyImageL, rectifyImageR;

Rect validROIL;//图像校正之后，会对图像进行裁剪，这里的validROI就是指裁剪之后的区域

Rect validROIR;

Mat xyz; //三维坐标

int blockSize = 0, uniquenessRatio = 0, numDisparities = 0;

Ptr<StereoBM> bm = StereoBM::create(16, 9);

Mat T\_new = (Mat\_<double>(3, 1) << -3.3269653179960471e+01, 3.7375231026230421e-01,-1.2058042444883227e-02);//T平移向量

//Mat rec = (Mat\_<double>(3, 1) << -0.00306, -0.03207, 0.00206);//rec旋转向量

Mat R\_new = (Mat\_<double>(3, 3) << 9.9998505024526163e-01, 3.5253250461816949e-03,

4.1798767087380161e-03, -3.4957471578341281e-03,

9.9996894942320580e-01, -7.0625732745616225e-03,

-4.2046447876106169e-03, 7.0478558986986593e-03,

9.9996632377767658e-01);//R 旋转矩阵

/\*\*\*\*\*立体匹配\*\*\*\*\*/

void stereo\_match(int, void\*)

{

bm->setBlockSize(2 \* blockSize + 5); //SAD窗口大小，5~21之间为宜

bm->setROI1(validROIL);

bm->setROI2(validROIR);

bm->setPreFilterCap(31);

bm->setMinDisparity(0); //最小视差，默认值为0, 可以是负值，int型

bm->setNumDisparities(numDisparities \* 16 + 16);//视差窗口，即最大视差值与最小视差值之差,窗口大小必须是16的整数倍，int型

bm->setTextureThreshold(10);

bm->setUniquenessRatio(uniquenessRatio);//uniquenessRatio主要可以防止误匹配

bm->setSpeckleWindowSize(100);

bm->setSpeckleRange(32);

bm->setDisp12MaxDiff(-1);

Mat disp, disp8, disparityImage;

bm->compute(rectifyImageL, rectifyImageR, disp);//输入图像必须为灰度图

disp.convertTo(disp8, CV\_8U, 255 / ((numDisparities \* 16 + 16)\*16.));//计算出的视差是CV\_16S格式

reprojectImageTo3D(disp, xyz, Q, true); //在实际求距离时，ReprojectTo3D出来的X / W, Y / W, Z / W都要乘以16(也就是W除以16)，才能得到正确的三维坐标信息。

xyz = xyz \* 16;

getDisparityImage(disp8, disparityImage, true);

imshow("disparity", disparityImage);

}

//立体匹配

void CimageProcessDlg::OnBnClickedButton25()

{

// TODO: 在此添加控件通知处理程序代码

//立体校正

stereoRectify(cameraMatrixL, distCoeffL, cameraMatrixR, distCoeffR, imageSize, R\_new, T\_new, Rl, Rr, Pl, Pr, Q, CALIB\_ZERO\_DISPARITY,

0, imageSize, &validROIL, &validROIR);

initUndistortRectifyMap(cameraMatrixL, distCoeffL, Rl, Pr, imageSize, CV\_32FC1, mapLx, mapLy);

initUndistortRectifyMap(cameraMatrixR, distCoeffR, Rr, Pr, imageSize, CV\_32FC1, mapRx, mapRy);

rgbImageL = imread("F://program//image//camer\_cab//left01.jpg", CV\_LOAD\_IMAGE\_COLOR);

cvtColor(rgbImageL, grayImageL, CV\_BGR2GRAY);

rgbImageR = imread("F://program//image//camer\_cab//right01.jpg", CV\_LOAD\_IMAGE\_COLOR);

cvtColor(rgbImageR, grayImageR, CV\_BGR2GRAY);

imshow("ImageL Before Rectify", grayImageL);

imshow("ImageR Before Rectify", grayImageR);

/\*

经过remap之后，左右相机的图像已经共面并且行对准了

\*/

remap(grayImageL, rectifyImageL, mapLx, mapLy, INTER\_LINEAR);

remap(grayImageR, rectifyImageR, mapRx, mapRy, INTER\_LINEAR);

/\*

把校正结果显示出来

\*/

Mat rgbRectifyImageL, rgbRectifyImageR;

cvtColor(rectifyImageL, rgbRectifyImageL, CV\_GRAY2BGR); //伪彩色图

cvtColor(rectifyImageR, rgbRectifyImageR, CV\_GRAY2BGR);

//单独显示

//rectangle(rgbRectifyImageL, validROIL, Scalar(0, 0, 255), 3, 8);

//rectangle(rgbRectifyImageR, validROIR, Scalar(0, 0, 255), 3, 8);

imshow("ImageL After Rectify", rgbRectifyImageL);

imshow("ImageR After Rectify", rgbRectifyImageR);

//显示在同一张图上

Mat canvas;

double sf;

int w, h;

sf = 600. / MAX(imageSize.width, imageSize.height);

w = cvRound(imageSize.width \* sf);

h = cvRound(imageSize.height \* sf);

canvas.create(h, w \* 2, CV\_8UC3); //注意通道

//左图像画到画布上

Mat canvasPart = canvas(Rect(w \* 0, 0, w, h)); //得到画布的一部分

resize(rgbRectifyImageL, canvasPart, canvasPart.size(), 0, 0, INTER\_AREA); //把图像缩放到跟canvasPart一样大小

Rect vroiL(cvRound(validROIL.x\*sf), cvRound(validROIL.y\*sf), //获得被截取的区域

cvRound(validROIL.width\*sf), cvRound(validROIL.height\*sf));

//rectangle(canvasPart, vroiL, Scalar(0, 0, 255), 3, 8); //画上一个矩形

cout << "Painted ImageL" << endl;

//右图像画到画布上

canvasPart = canvas(Rect(w, 0, w, h)); //获得画布的另一部分

resize(rgbRectifyImageR, canvasPart, canvasPart.size(), 0, 0, INTER\_LINEAR);

Rect vroiR(cvRound(validROIR.x \* sf), cvRound(validROIR.y\*sf),

cvRound(validROIR.width \* sf), cvRound(validROIR.height \* sf));

//rectangle(canvasPart, vroiR, Scalar(0, 0, 255), 3, 8);

cout << "Painted ImageR" << endl;

//画上对应的线条

for (int i = 0; i < canvas.rows; i += 16)

line(canvas, Point(0, i), Point(canvas.cols, i), Scalar(0, 255, 0), 1, 8);

imshow("rectified", canvas);

/\*

立体匹配

\*/

namedWindow("disparity", CV\_WINDOW\_AUTOSIZE);

// 创建SAD窗口 Trackbar

createTrackbar("BlockSize:\n", "disparity", &blockSize, 8, stereo\_match);

// 创建视差唯一性百分比窗口 Trackbar

createTrackbar("UniquenessRatio:\n", "disparity", &uniquenessRatio, 50, stereo\_match);

// 创建视差窗口 Trackbar

createTrackbar("NumDisparities:\n", "disparity", &numDisparities, 16, stereo\_match);

stereo\_match(0, 0);

waitKey(0);

}