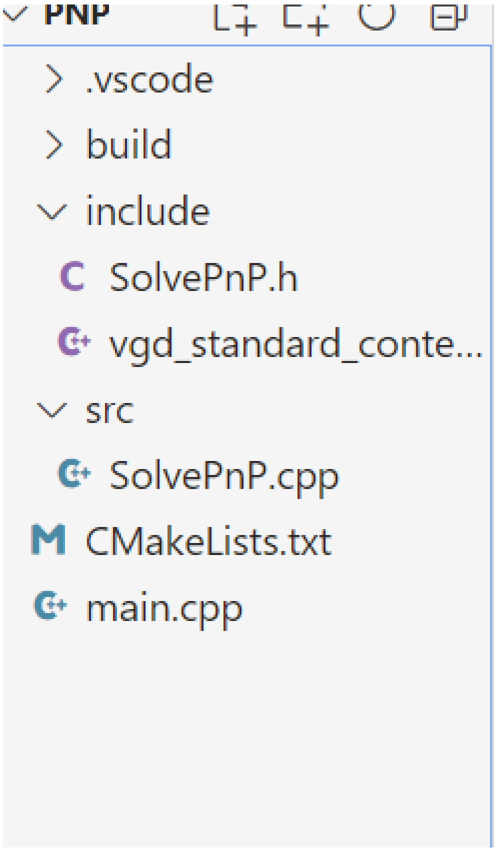
队内Pnp代码

1.文件目录



- **去除串口部分**
- **main中是自己自定义的测试函数，详见下文**
- **改写CMakeList（能跑就行）**

2.Pnp-test

首先先看了久远的培训时期的代码，pnp-test做一个简单的测试

```
int main(int argc, char **argv)
{
    Mat image = imread("G:/temp/vgd/vgd-test/pnp-test-before/test.jpg");

    // 2D 特征点像素坐标，这里是用PS找出，也可以用鼠标事件画出特征点
    // 用电脑自带的画图软件   光标指向对应点   得到像素特征点
    vector<Point2d> image_points; // 容器存储图像像素特征点
    image_points.push_back(Point2d(668, 647));
    image_points.push_back(Point2d(643, 878));
    image_points.push_back(Point2d(1282, 647));
    image_points.push_back(Point2d(1300, 870));

    // 画出四个特征点  这4个特征点就是装甲板灯条4个点  像素坐标系
    for (int i = 0; i < image_points.size(); i++)
    {
        circle(image, image_points[i], 3, Scalar(0, 0, 255), -1);
```

```
    }
    // imshow("image",image);
    // waitKey(0);
    // return 0;

    // 3D 特征点世界坐标，与像素坐标对应，单位是mm
    std::vector<Point3d> model_points;
    model_points.push_back(Point3d(-66.75f, -24.25f, 0));
    model_points.push_back(Point3d(+66.75f, -24.25f, 0));
    model_points.push_back(Point3d(-66.75f, +24.25f, 0));
    model_points.push_back(Point3d(+66.75f, +24.25f, 0));
    //   注意世界坐标和像素坐标要一一对应

    // 相机内参矩阵和畸变系数均由相机标定结果得出
    // 相机内参矩阵
    Mat camera_matrix = (Mat_<double>(3, 3) << 1.201371857055914e+03, 0,
7.494419594994199e+02,
                         0, 1.201435954410725e+03, 5.508546827593877e+02,
                         0, 0, 1);

    // 畸变系数
    Mat dist_coeffs = (Mat_<double>(5, 1) << -0.098380553375716,
0.006115203108383,
                       -4.766609631726518e-04, -0.001862163979558, 0);

    cout << "Camera Matrix: " << endl
         << camera_matrix << endl
         << endl;
    cout << "Distortion coefficient: " << endl
         << dist_coeffs << endl;

    // 旋转向量
    Mat rotation_vector;
    // 平移向量
    Mat translation_vector;

    // pnp求解
    // 传3D特征点世界坐标 图像像素特征点 相机内参矩阵 畸变系数 传了两个空矩阵  一个存旋转向量
一个存平移向量
    solvePnP(model_points, image_points, camera_matrix, dist_coeffs,
             rotation_vector, translation_vector, 0, SOLVEPNP_ITERATIVE);
    // 默认ITERATIVE方法，可尝试修改为EPNP（CV_EPNP）,P3P（CV_P3P）

    cout << "Rotation Vector " << endl
         << rotation_vector << endl
         << endl;
    cout << "Translation Vector" << endl
         << translation_vector << endl
         << endl;

    Mat Rvec;                                     // 接收旋转矩阵
    Mat_<float> Tvec;                             // 接收平移矩阵
    rotation_vector.convertTo(Rvec, CV_32F);      // 旋转向量转换格式
    translation_vector.convertTo(Tvec, CV_32F); // 平移向量转换格式  //表面上似乎只是
缩短了小数位
```

```
    cout << endl
         << "After convertion:\nRotation Vector " << endl
         << Rvec << endl
         << "Translation Vector " << endl
         << Tvec << endl;

    Mat_<float> rotMat(3, 3);
    // 旋转向量转成旋转矩阵
    Rodrigues(Rvec, rotMat); // 这个函数有两个作用  1.输入旋转向量，返回旋转矩阵 2.输入旋转矩阵返回旋转向量和雅可比矩阵
    cout << "rotMat" << endl
         << rotMat << endl
         << endl;
    // cout << rotationMatrixToEulerAngles(rotMat);

    float yawErr = atan(translation_vector.at<float>(0, 0) / translation_vector.at<float>(2, 0)) / CV_PI * 180;   // 转换为角度
    float pitchErr = atan(translation_vector.at<float>(1, 0) / translation_vector.at<float>(2, 0)) / CV_PI * 180; // 转换为角度
    float yaw = atan(Tvec.at<float>(0, 0) / Tvec.at<float>(2, 0)) / CV_PI * 180; // 转换为角度
    float pitch = atan(Tvec.at<float>(1, 0) / Tvec.at<float>(2, 0)) / CV_PI * 180;                            // 转换为角度
    cout << "yawErr:\t" << yawErr << endl;
    cout << "pitchErr:\t" << pitchErr << endl;
    cout << "yaw:\t" << yaw << endl;
    cout << "pitch:\t" << pitch << endl;
    Mat P_oc;
    P_oc = -rotMat.inv() * Tvec; //.inv()是对矩阵求逆  对象必须为方阵
                                 // 求解相机的世界坐标，得出p_oc的第三个元素即相机到物体的距离即深度信息，单位是mm
                                 // while (true)
    //{
    cout << "P_oc" << endl
         << P_oc << endl;
    // cout <<Tvec<<endl;
    //}
    // calAngle(camera_matrix,dist_coeffs,);
    imshow("Output", image);
    waitKey(0);
}
```

大致流程即是:

1. **选定所给图片的装甲板灯条的四个图像坐标点**
2. **匹配好世界坐标系下装甲板的四个三维特征点**
3. **设置好相机内参矩阵、畸变系数矩阵**
4. **pnp函数调用，拿到旋转向量、平移向量**
5. **平移向量做处理拿到两个角度（这步貌似有误，或者说角度算的不对）**
6. **旋转矩阵逆*平移向量得到新的向量，其第三个参数元素即深度（深度求的比较精确）**

3.SolvePnp

```
float SOLVEPNP::PNP(vgd_stl::armors &finalarmor, int flag)
{
    // 3D 特征点世界坐标，与像素坐标对应
    if (flag == 0) // 处理小装甲板 特征点
    {
        model_points.push_back(Point3d(-66.75f, -24.25f, 0));
        model_points.push_back(Point3d(+66.75f, -24.25f, 0));
        model_points.push_back(Point3d(-66.75f, +24.25f, 0));
        model_points.push_back(Point3d(+66.75f, +24.25f, 0));
    }
    if (flag == 1) // 处理大装甲板
    {
        model_points.push_back(Point3d(-114.0f, -24.25f, 0));
        model_points.push_back(Point3d(+114.0f, -24.25f, 0));
        model_points.push_back(Point3d(-114.0f, +24.25f, 0));
        model_points.push_back(Point3d(+114.0f, +24.25f, 0));
    }

    solvePnP(model_points, picture_points, camera_matrix, dist_coeffs,
             rotation_vector, translation_vector, 0, cv::SOLVEPNP_ITERATIVE);
    // 默认ITERATIVE方法，可尝试修改为EPNP（CV_EPNP）,P3P（CV_P3P）

    Mat Rvec;
    Mat Tvec;
    rotation_vector.convertTo(Rvec, CV_32F);    // 旋转向量转换格式
    translation_vector.convertTo(Tvec, CV_32F); // 平移向量转换格式

    // finalarmor.position[0]=translation_vector.at<double>(0) ;        //后边系数
可调，旋转向量并未转换成旋转矩阵
    // finalarmor.position[1]=translation_vector.at<double>(1) ;
    // finalarmor.position[2]=translation_vector.at<double>(2) ;
    // finalarmor.position[0]=Tvec.at<float>(0,0) ;
    // finalarmor.position[1]=Tvec.at<float>(1,0) ;
    // finalarmor.position[2]=Tvec.at<float>(2,0) ;
    // cout<<finalarmor.position[0]<<finalarmor.position[1]<<endl;

    // cout<<Tvec<<endl;

    // double current_yaw = std::atan2(finalarmor.position[0],
finalarmor.position[2])/CV_PI*180;
    // cout<<current_yaw<<endl;

    // double rm[9];
    // Mat rotMat(3,3,CV_64FC1,rm);
    Mat_<float> rotMat(3, 3);
    // Mat_<float> traMat(3, 3);
    Rodrigues(Rvec, rotMat); // 旋转向量转换为旋转矩阵
    // yaw=atan2(rotMat(2,1),rotMat(2,2))*57.2958;
    // pitch=atan2(-
rotMat(2,0),sqrt(rotMat(2,0)*rotMat(2,0)+rotMat(2,2)*rotMat(2,2)))*57.298;

    // 旋转向量转成旋转矩阵

    Mat P_oc;
    P_oc = -rotMat.inv() * Tvec;
```

```
    // 求解相机的世界坐标，得出p_oc的第三个元素即相机到物体的距离即深度信息，单位是mm

    // 迭代器版
    //  MatIterator_<float> it = P_oc.begin<float>();
    //  MatIterator_<float> it_end = P_oc.end<float>();
    //  for(int i = 1;it != it_end;it++,i++)
    //  {
    //      if(i == 3)
    //      {
    //          distance = (float)(*it);
    //          break;
    //      }

    // }
    // at版
    distance = Tvec.at<float>(2, 0);//矩阵的第三个元素就是距离
    distance /= 10;
    //  cout<<P_oc<<endl;
    // cout<<Tvec<<endl;
    // yaw=atan(Tvec.at<float>(0, 0)/Tvec.at<float>(2, 0))/CV_PI*180;
    // pitch=atan(Tvec.at<float>(1, 0)/Tvec.at<float>(2, 0))/CV_PI*180;
    // cout<<Tvec<<endl;
    // cout<<yaw<<endl;

    // cout<<yaw<<" "<<pitch<<" "<<endl;
    // cout << "P_oc " << endl << P_oc << endl;
    // cout << "distance " << abs(distance)<< endl;
    // cout<<flag<<endl;
    return abs(distance);
}
```

pnp部分和上面的test思路基本一致，但是角度值部分可以看到是放弃了，几种求角度的方法只有用平移向量的那个方法算比较准，其他的都存在问题

在实际计算时会传入当前图像帧中捕获的装甲板的图像坐标，作为图像坐标系下的重要点

```
void SOLVEPNP ::caculate(vgd_stl::armors &finalarmor)
{

    static float final_distance;
    float tmp;

    // 直接拿图像坐标 也就是二维图像坐标系的四点
    picture_points.push_back(finalarmor.corner[1]);
    picture_points.push_back(finalarmor.corner[4]);
    picture_points.push_back(finalarmor.corner[2]);
    picture_points.push_back(finalarmor.corner[3]);

    // 判断是大装甲板还是小装甲板  // 大装甲板
    if (finalarmor.number == 0)
    {
        xishu = (22.8 / finalarmor.boardw + 4.85 / finalarmor.boardh) / 2;
        // 世界坐标
        tmp = PNP(finalarmor, 1);
        // if(tmp > 10)
```

```
        final_distance = tmp; // 深度 距离
    }
    else // 小装甲板
    {
        xishu = (13.25 / finalarmor.boardw + 4.85 / finalarmor.boardh) / 2;
        tmp = PNP(finalarmor, 0);
        /// if(tmp > 10)
        final_distance = tmp;

        // cout << "final_distance  " << tmp<<endl;
    }
    //
calAngle(camera_matrix,dist_coeffs,finalarmor.center.x,finalarmor.center.y);

    double distance_to_midboard_x, distance_to_midboard_y;
    distance_to_midboard_x = xishu * (finalarmor.center.x - midx);
    distance_to_midboard_y = xishu * (finalarmor.center.y - midy);

    finalarmor.dtm = sqrt((finalarmor.center.x - midx) * (finalarmor.center.x -
midx) + (finalarmor.center.y - midy) * (finalarmor.center.y - midy));

    double angle_x = atan2(distance_to_midboard_x, final_distance);
    double angle_y = atan2(distance_to_midboard_y, final_distance) + 4.134;

    double final_angle_x = angle_x / P * 180;
    double final_angle_y = angle_y / P * 180;

    finalarmor.position[0] = distance_to_midboard_x / 100;
    finalarmor.position[1] = distance_to_midboard_y / 100;
    finalarmor.position[2] = final_distance / 100;
    //          cout << "final_distance  " << tmp<<endl;
    //          cout<<"yaw="<<final_angle_x<<endl;
    //          cout<<"pitch="<<final_angle_y<<endl<<endl;

    // #ifdef NX
    // if(tmp > 10)
    // uart.sSendData(final_angle_x, final_angle_y,final_distance,1);

    // #endif
}
```

角度部分看不懂