# 1. 项目概述

Keyhole-Detector 是一个基于计算机视觉和深度学习的高级自动化分析系统，旨在从输入的图像中精确提取锁孔的关键物理信息。系统能够实现端到端的处理流程，包括：

**目标定位:** 自动在复杂背景中检测并定位锁孔。

**状态识别:** 判断是否有钥匙插入。

**物理参数估算:** 计算相机与锁孔的距离，以及锁孔的精确旋转角度。

**归位判断:** 根据旋转角度，判断锁孔是否处于标准的垂直归位状态。

该项目采用模块化和配置驱动的工程化设计，确保了算法的高精度、高效率和高可维护性。

# 2. 项目架构与设计思想

本项目采用三层分离的架构设计，以实现关注点分离和高内聚、低耦合。

**配置层 (config.py):**

所有可调参数、路径、模型配置和算法阈值均在此定义。这使得非开发人员也能轻松调整系统行为，且代码逻辑与配置完全解耦。

**核心逻辑层 (main\_refactored.py):**

**LockAnalyzer 类:** 封装了所有核心算法和状态。采用面向对象设计，将重量级资源（如ONNX模型）作为类实例的属性，在初始化时加载一次，实现了高效的资源管理。

**私有方法封装:** 复杂的算法流程被拆分为一系列职责单一的私有方法 (\_detect\_objects, \_calculate\_distance, \_recognize\_angle)，提高了代码的可读性和可测试性。

**公共接口:** 提供一个简洁的 analyze\_image() 方法作为统一的入口，隐藏了内部复杂的实现细节。

**接口层 (process.py):**process.py为比赛接口适配脚本同时也是示例

**适配器模式:** 该文件扮演了适配器（Adapter）的角色，将内部复杂的 LockAnalyzer 对象适配到比赛或外部系统所要求的简单 process\_img(img\_path) 函数接口。

**单例模式实现:** 通过在模块加载时创建全局唯一的 GLOBAL\_ANALYZER 实例，确保了在多次调用 process\_img 时，模型和分析器状态得以复用，从而达到最佳性能。

# 3. 核心算法详解

系统的工作流程由三个核心算法模块串联而成：

## 3.1 目标检测算法

**模型:** 采用 YOLOv5 深度学习模型，该模型以其在速度和精度上的卓越平衡而著称。模型被导出为 ONNX (Open Neural Network Exchange) 格式，以实现跨平台的、高性能的推理。

**预处理:**

**Letterbox 缩放:** 为了在不扭曲图像长宽比的情况下，将任意尺寸的输入图像适配到模型要求的正方形输入（如 640x640），我们采用了 Letterbox 方法。该方法将图像等比例缩放，然后用灰色（值为114）填充空白区域。

**归一化:** 像素值从 [0, 255] 范围归一化到 [0.0, 1.0]。

**推理:** 使用 cv2.dnn.readNetFromONNX() 加载模型，并通过 net.forward() 执行推理。

**后处理:**

**阈值过滤:** 对模型输出的所有检测框，根据置信度 (YOLO\_CONF\_THRESHOLD) 进行初步过滤。

**非极大值抑制 (NMS):** 使用 cv2.dnn.NMSBoxes 解决同一物体被多次检测的问题，合并重叠度过高（由 YOLO\_IOU\_THRESHOLD 控制）的检测框，只保留置信度最高的那个。

**坐标逆转换:** 将在 Letterbox 图像上得到的检测框坐标，通过逆向的缩放和填充计算，精确地映射回原始图像的坐标系。

## 3.2 距离估算算法

**方法:** 采用基于单目视觉的几何估算法。其核心假设是：物体在图像中成像的大小与其到相机的距离成反比。

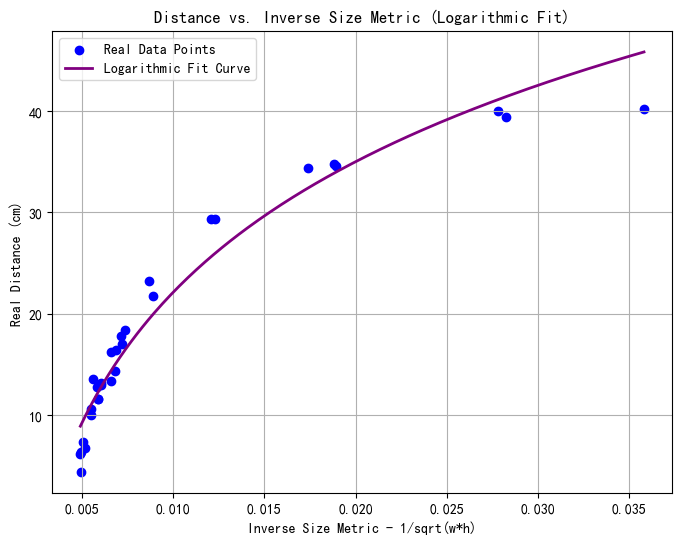
**公式:** 通过对大量样本（已知距离和对应的检测框像素尺寸）进行数据拟合，我们得到一个稳健的对数关系模型：

其中：

和 是检测框的像素宽度和高度。

近似代表了物体的“视觉直径”。

和 (即 DISTANCE\_PARAM\_A, DISTANCE\_PARAM\_B) 是通过回归分析得到的拟合常数，它们将像素尺寸与真实世界的距离（厘米）关联起来。



距离与（对数拟合）图

**优势:** 此方法无需相机标定，对相机的微小位移不敏感，且计算速度极快。

## 3.3 旋转角度识别算法

这是本系统最具创新性的部分，通过一套精密的图像处理流程实现高精度角度测量。

**兴趣区域裁剪 (ROI Cropping):**

利用目标检测得到的边界框，从原始高分辨率图像中精确裁剪出锁孔区域。

**图像预处理:**

**灰度化与二值化:** 将裁剪出的图像转换为灰度图，并使用一个固定的阈值进行反向二值化，使锁孔实体部分变为白色，背景为黑色。

**最大连通组件提取:** 为了消除可能由光照、反光或背景噪声引入的干扰白点，我们使用 cv2.connectedComponentsWithStats 找到并只保留图像中面积最大的白色区域，即锁孔主体。这确保了后续分析的纯净性。



处理后锁孔部分的二值图

**旋转投影分析 (Rotational Projection Analysis):**

核心思想: 物体的几何对称性会反映在它的投影数据上。我们通过模拟从0到179度旋转图像，并计算其在中心垂直线上的投影（白色像素总和），来寻找其对称轴。

**实现:**

计算锁孔二值图的质心作为旋转中心。

在一个循环中，迭代0到179度：

使用 cv2.getRotationMatrix2D 和 cv2.warpAffine 对图像进行旋转。

计算旋转后图像中心垂直线上的白色像素之和，作为该角度的投影值。

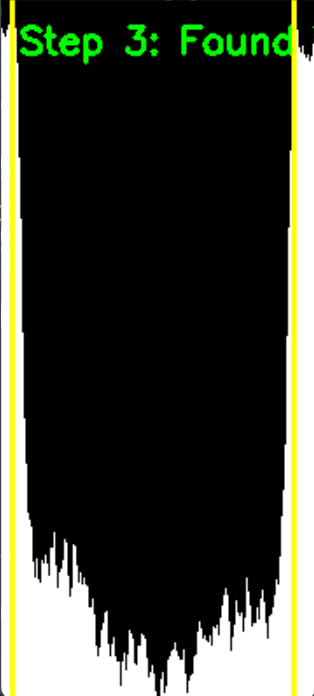
**结果:** 得到一个包含180个投影值的一维数组，其曲线形状蕴含了锁孔的对称信息。

**双峰检测与角平分线计算 (Dual-Peak Detection & Bisector Calculation):**

**物理洞察:** 锁孔的缺口部分所在的直线白色像素相对较多，会在投影曲线上形成两个主峰。这两个主峰所对应的角度，就是锁孔实体最厚部分的方向。

**峰值检测:** 使用自定义的 find\_two\_largest\_peaks\_np 函数，高效地从投影数据中找到两个最显著的局部峰值。

**对称轴确定:** 锁孔的水平对称轴正好是这两个峰值方向所形成的锐角的角平分线。通过一个考虑了0/180度环绕问题的算法，精确计算出这条角平分线的角度。



角度和直线经过像素直方图

**最终角度转换:**

将计算出的几何对称轴角度，转换为用户易于理解的、相对于垂直方向的旋转角度。

根据 ANGLE\_RESET\_THRESHOLD 判断锁孔是否处于归位状态。

# 4. 安装配置步骤

## 1. 设置本地开发环境

a. 克隆项目

|  |
| --- |
| git clone https://github.com/AdrienVon/Keyhole-Detector.git  cd Keyhole-Detector |

b. 创建并激活 Python 虚拟环境 (推荐)

Windows：

|  |
| --- |
| python -m venv venv  .\venv\Scripts\activate |

Macos/Linux

|  |
| --- |
| python3 -m venv venv  source venv/bin/activate |

c. 安装依赖

|  |
| --- |
| pip install -r requirements.txt |

## 2. 使用 process.py 脚本进行测试

您也可以提供一个文件夹的路径，脚本会自动处理该文件夹下的所有图片。

|  |
| --- |
| # 示例：  python src/process.py path/to/your/image\_folder/ |