用到的所有东西的型号、整个操作流程和系统流程、软件代码截图啥的你看下午有时间发一下不

硬件设计：

整体的硬件设计如图所示：



整个系统由三部分组成

1. 识别平台

识别平台使用大块的亚克力板和2020铝型材搭建而成，并使用一张蓝色的贴纸作为识别的背景，剔除杂色。2020铝型材将一个200w像素1080P的USB摄像头固定在了离识别平面40cm高的平面中心位置

1. 智能补光模块

智能补光模块由高亮度LED灯珠组成的原型灯板和LED控制器组合而成，可调整平台照明亮度，保障在低亮度，低清晰度环境下的正常尺寸识别

1. LM2596降压电路模块

LM2596降压电路模块，将7-12V的宽电压输入降低到稳定的5V电压输出，同时LM2596支持大电流输出，最大电流3A，保障智能补光模块的稳定工作

整个操作流程：

整个操作流程为，首先将usb摄像头插入电脑上，然后运行python脚本，该脚本会同时启动yolo识别和flask可视化，yolo用于识别当前摆放的待打包货物，flask将识别到的信息通过web可视化的方式在浏览器页面中显示



如图所示，左侧为usb摄像头输入的图像检测后的结果，右侧为待输出的打包盒尺寸设计图，以及打包盒动态装配示意图

同时，底栏可以查看检测到的对象参数。

首先，点击输出信息，可暂停输入图像检测，此时对象参数固定，可以通过点击底栏左侧的“查看检测对象参数”，得到一个上拉菜单

图形用户界面, 网站

AI 生成的内容可能不正确。

通过点击不同的对象，可以查看到该对象具体的参数，即长，宽，底面积

此时，点击右侧的选择打包对象，可以多选（也可以单选）对象进行打包，在完成多选后点击“生成打包“按钮，右侧刀版尺寸图和打包动态图都会更新



系统流程：

系统流程主要通过python脚本和html页面来进行讲解，首先是python脚本

1. 在启动python脚本后，首先我们会导入一些库，在代码中，这部分位于

1. from flask import Flask, render\_template, request, jsonify, Response

2. import cv2

3. import numpy as np

4. from ultralytics import YOLO

5. import threading

6. import queue

7. import time

8. from flask\_cors import CORS  # 导入 CORS (如果需要)

9. from collections import defaultdict

10. import os

11.

这其中，falsk用于在web浏览器页面上可视化出各种数据，cv2是opencv的库名，用于提供开源的计算机视觉库，numpy是用于数值计算的一个库，ultralytics是用于图像识别的一个知名开源库，threading库是用于线程管理， queue是用于队列的库，time是用于与时间相关的库，flask cores是对web可视化中一些功能提供支持，collections库提供了对常见数据类型的支持，比如counter，deque等，os库提供了python同操作系统之间的一些交互函数。

2. 在导入库后，我们需要对flask库做一些全局配置，代码位于

1. frame\_queue = queue.Queue(maxsize=30)

2. yolo\_error\_data = None

3. svg\_content\_data = None

4. video\_path\_data = None

5. yolo\_results\_data = []  # 初始化为空列表

6. app = Flask(\_\_name\_\_)

7. CORS(app)  # 允许跨域请求 (如果需要)

8.

首先我们指定了线程的最大容量为30，同时以None初始化了三个存放数据的变量，并将yolo识别结果用空列表初始化，app = Flask(\_\_name\_\_)创建了一个flask的应用程序实例，他是整个web程序的入口点，最后我们允许前后端的跨域请求。

1. 接着我们将相机标定后的结果硬编码在python脚本中，以便后面矫正摄像头时调用，代码位于

1. # 标定参数

2. # OptiCalib标定结果(200w摄像头)

3. fx = 1462.933715

4. fy = 1472.343506

5. cx = 969.398521

6. cy = 545.531149

7.

8. intrinsic\_matrix = np.array([[fx, 0, cx],

9.                              [0, fy, cy],

10.                              [0, 0, 1]], dtype=np.float32)

11.

12. # 替换为你的实际畸变系数 (k1, k2, p1, p2, k3)(200w摄像头)

13. k1 =  0.057745

14. k2 = -0.117250

15. p1 = -0.001262

16. p2 = 0.001061

17. k3 = 0.0

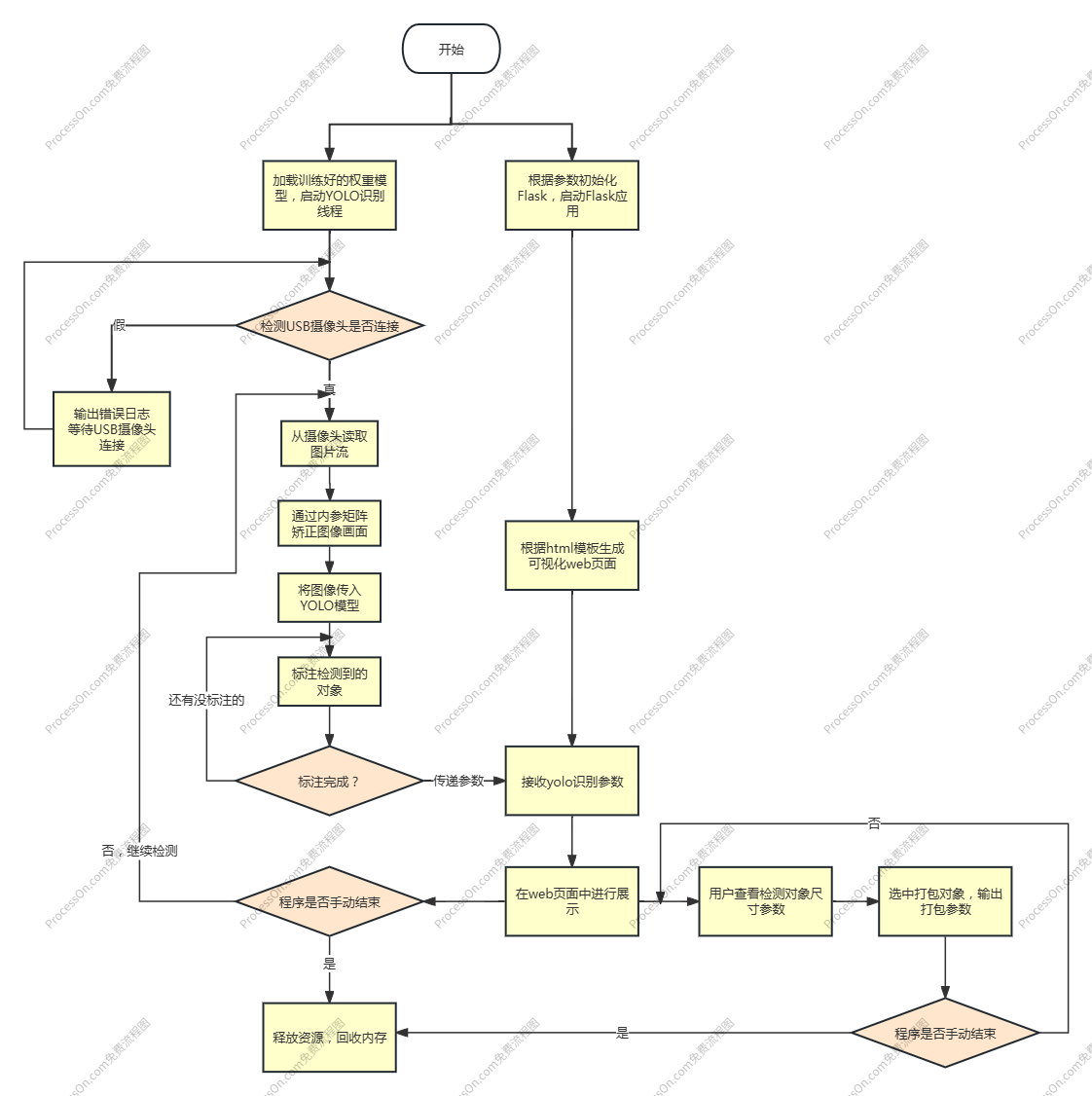
18.

19. distortion\_coefficients = np.array([k1, k2, p1, p2, k3], dtype=np.float32)

20.

这里我们定义了fx，fy，cx，cy和镜像畸变参数k1，k2，k3，切向畸变参数p1，p2

在导入库与初始化结束后，使用流程图来对程序框架进行描述



软件代码截图

后端yolo检测函数



在前端页面中使用Javascript实现将后端的yolo检测结果输出

