基于 RK3588 的果蔬智能结算系统设计

A37；张天；马灵希；王铭蔚；陈子为

摘要

本文设计了一套基于瑞芯微 RK3588 开发板的果蔬智能结算系统，融合视觉识别与称重传感技术，实现果蔬自动化识别、称重与结算。

系统硬件以 RK3588 为主控核心，集成 1080P 摄像头、HX711 称重传感器及 7 寸触控屏，通过 USB，SCL，以及杜邦线接口实现设备互联。软件层面采用模块化设计：基于 OpenCV 完成图像预处理，利用数据集训练的模型实现果蔬识别，通过称重模块采集重量数据，结合单价库自动计算价格（价格 = 单价 × 重量），并基于 Tkinter 库开发交互界面。

测试结果显示，系统单次结算耗时 2 秒左右，支持 36+常见果蔬品类，可满足无人零售、智能超市等场景需求。

本设计创新点在于通过稳定放置识别检测算法，称重结算自动化，利用 RK3588 的算力实现高效推理。未来可扩展多种类同时识别、云平台数据同步等功能，进一步提升系统适用性。

第一部分 作品概述

1.1 功能与特性

核心功能：

智能识别：1080P USB 摄像头，用于采集果蔬图像需求；

自动称重：应变片式传感器 +AD 转换器，精度 ±1g，量程 0g-10kg，通过 Data 和 SCL 接口实现简单通信协议与主控通信；

交互操作：7 寸触控屏，用于显示识别结果与操作界面。

整个流程从放置、稳定检测、识别、称重到结算呈现，均实现了高度的自动化触发，最大程度地减少了人工干预环节，提升了无人零售场景的效率和用户体验。

关键特性：

依托 RK3588 具有高算力，能高效处理各种测试，具有稳定可靠性，适用超市，无人零售系统，且支持单价远程更新，识别品种繁多。

1.2 应用领域

本果蔬智能结算系统凭借其高效、精准、自动化的特性，可以适用于：

无人零售商店（如无人超市、自助便利店）；

智能超市的自助结算区；

社区生鲜自提点、果蔬自助售卖机；

食堂自主结算终端等场景。

总而言之，本系统是构建智能化、无人化零售及餐饮终端的关键基础设施，适用于所有需要快速、准确完成散装果蔬商品识别与交易的商业环境。

1.3 主要技术特点

本系统的核心优势在于其深度融合了高性能硬件、智能算法与人性化交互，构建了一套高效、精准且用户友好的自动化果蔬结算解决方案。

系统以瑞芯微 RK3588 嵌入式开发平台为强大硬件基石，其卓越的处理能力与专为 AI 任务优化的 NPU，为复杂的图像处理和实时推理提供了坚实的本地化计算保障，显著降低了对云端的依赖，体现了轻量化边缘计算的设计理念。在视觉识别方面，系统基于 Fruit and Vegetables SSM 果蔬数据集训练的分类模型，结合 OpenCV 库实现了实时的图像预处理流程，确保了对复杂形态、多样光照条件下各类果蔬品类的精准识别。与此同时，系统结合称重传感器，实时采集果蔬的重量数据，通过多模态数据融合策略紧密结合（视觉识别确定果蔬的种类，称重传感器获取其重量），两者共同作为结算的依据。一个关键的智能化设计是动态触发机制，系统通过算法持续监测称重区的状态变化，仅在检测到果蔬被稳定放置时，才自动触发图像捕获与识别流程，避免了无效操作并提升了效率和准确性。识别结果与重量数据随即通过内置的计价公式（价格 = 单价 × 重量）自动完成结算。面向用户端，系统利用 Tkinter 库开发了直观、易用的人机交互界面，支持流畅的触摸操作，为用户清晰地展示识别结果、重量信息、计算出的价格，并提供便捷的支付引导，实现了一体化交互体验。

1.4 主要性能指标

表 1 系统性能指标表

指标	参数
果蔬识别准确率	≥95%（针对常见 36 种果蔬）
识别响应时间	≤1 秒（从稳定放置到输出识别结果）
称重精度	±1g（支持 0g-10kg 量程）
结算完成时间	≤3 秒（含识别、称重、价格计算）
支持果蔬种类	36 种（可通过模型更新扩展）

1.5 主要创新点

1.5.1 多技术联动

系统通过实时监测称重传感器的压力变化曲线，智能判断果蔬是否处于稳定静止状态，仅当检测到符合预设稳定性阈值（如连续 0.5 秒重量波动<±1g）时，方自动触发图像采集与识别流程。

1.5.2 模型部署策略

将基于 PyTorch 训练的果蔬识别模型.pt 文件导出为 ONNX 格式，通过 ONNX Checker 工具验证模型结构完整性，确保卷积、池化等核心算子与 RK3588 平台的 ONNX Runtime 部署环境兼容，解决传统框架模型部署的“平台锁定”问题。在 RK3588 的 Linux 系统中部署 ONNX Runtime 推理引擎，确保模型全量功能可用。

1.5.4 场景化交互

主界面动态放大显示识别结果与价格，辅以语音播报增强反馈感；支持商家自定义设置单价，设置流程简便。交互流程极致简化，用户仅需“放置商品→确认支付”两步操作。

1.6 设计流程

需求分析：明确自动识别、称重、结算核心需求，确定无人零售场景适配目标；

方案设计：选定 RK3588 平台，规划 "摄像头 + 称重传感器 + 触控

屏" 硬件架构；

开发实现：硬件搭建接口连接，软件完成设备配置和界面开发；

模块测试：分别测试各个模块功能的正常运行；

系统测试：对整个系统各功能之间联合实现测试，完成智能果蔬系统的设计。

作品提交：完成要求的报告撰写和视频讲解，提交作品。

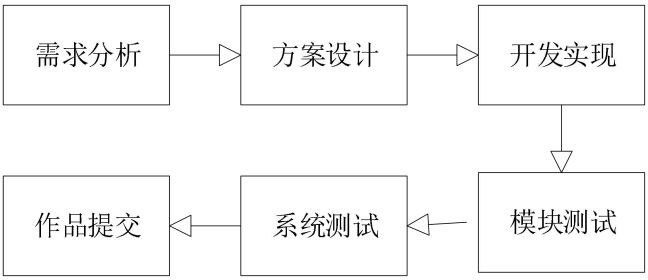


图 1 设计流程图

第二部分 系统组成及功能说明

2.1 整体介绍

果蔬智能结算系统整体流程如下图所示：

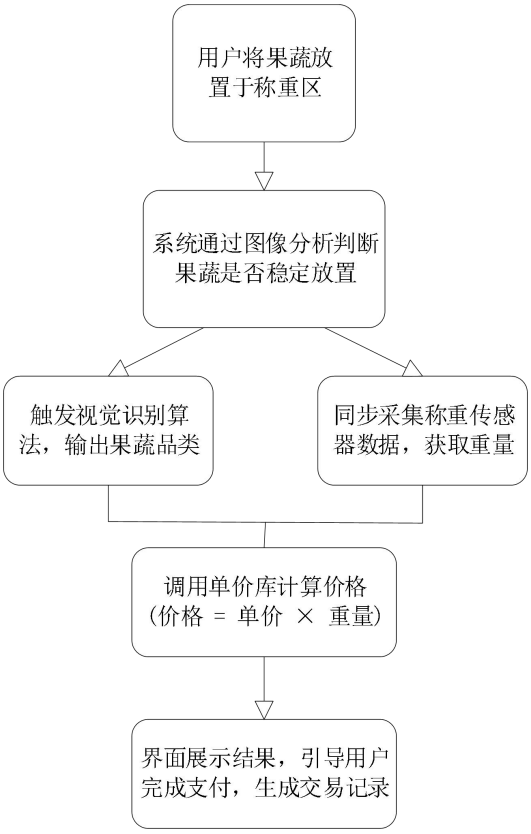


图 2 系统工作流程图

果蔬智能结算系统采用“感知层、处理层、交互层、数据层”的四层架构：

(1) 感知层：由高清摄像头（负责图像采集）和称重传感器（负责重量采集）组成，实现对果蔬的物理信息感知。

(2) 处理层：基于瑞芯微 RK3588 开发板，运行 Linux 操作系统，集成图像预处理、果蔬分类算法、重量数据处理及结算逻辑。

(3) 交互层：通过触控显示屏，提供识别结果展示、价格显示、结算操作引导等功能。

(4) 数据层：本地存储果蔬单价库、交易记录。

2.2 硬件系统介绍

2.2.1 硬件整体介绍；

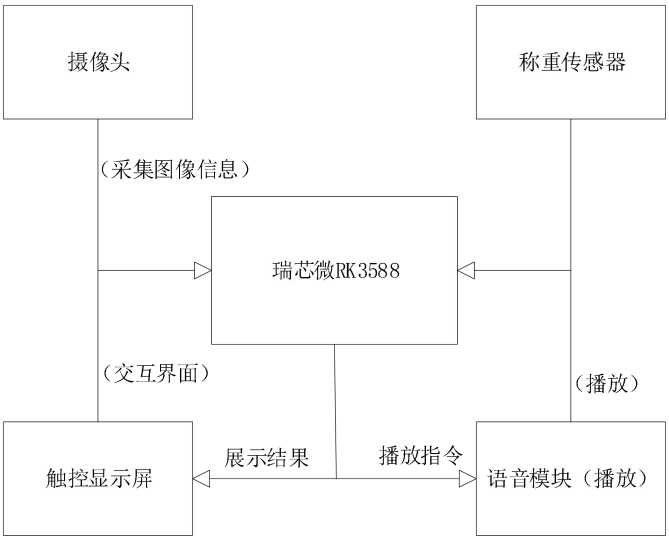


图 3 系统整体框图

2.2.2 机械设计介绍

系统机械结构以 “称重区、识别区、交互区” 为核心，主要包括：

(1) 称重平台：采用高精度称重传感器，表面覆盖防滑、耐脏的亚克力面板，防止果蔬滑落，保证称重稳定性。

(2) 识别模块：位于称重平台正上方 20cm 处，固定高清摄像头（分辨率为 1080P），镜头角度垂直向下，确保完整拍摄称重区果蔬图像；支架内置补光灯（可调亮度），避免环境光过暗影响识别。

(3) 交互屏幕：交互屏幕（7 寸触控屏），显示识别结果、重量、价格，接收用户操作指令，方便用户操作与观察。

2.2.3 电路各模块介绍

表 2 各模块介绍表

模块名称	组件	功能说明
主控制模块	瑞芯微 RK3588 开发板	运行操作系统、图像算法与结算逻辑，控制各模块协同工作
图像采集模块	1080P USB 摄像头（帧率 30fps）	实时采集称重区果蔬图像，输出至主控
称重传感模块	应变片式称重传感器 + AD 转换器 + 显示模块	接收通讯线上已被 AD 模块转换的数字信号
人机交互模块	7 寸触控显示屏	显示识别结果、重量、价格，接收用户操作指令

主控制模块 PCB 版图

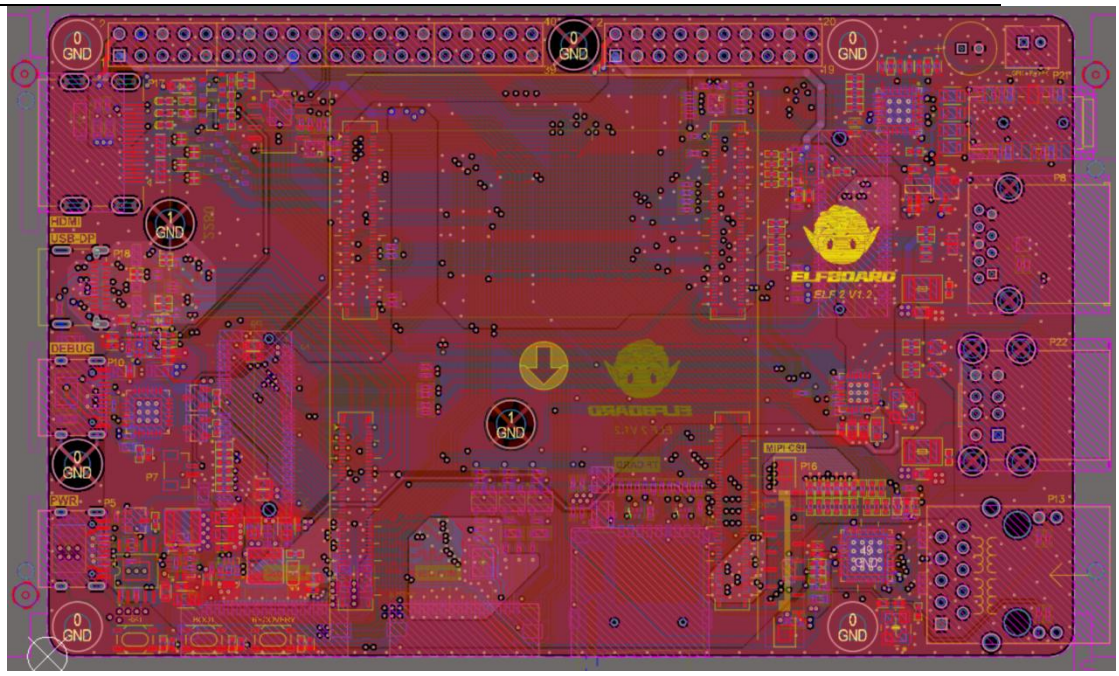


图 4 主控制模块 PCB 版图

2.3 软件系统介绍

2.3.1 软件整体介绍：

果蔬智能结算系统基于 RK3588 嵌入式平台，采用三层架构设计（UI 层、业务逻辑层、硬件接口层），集成了图像识别、重量传感和交易处理功能。系统通过摄像头实时识别果蔬品类，通过称重传感器精确测量重量，自动计算商品价格，并提供简洁的交易流程。

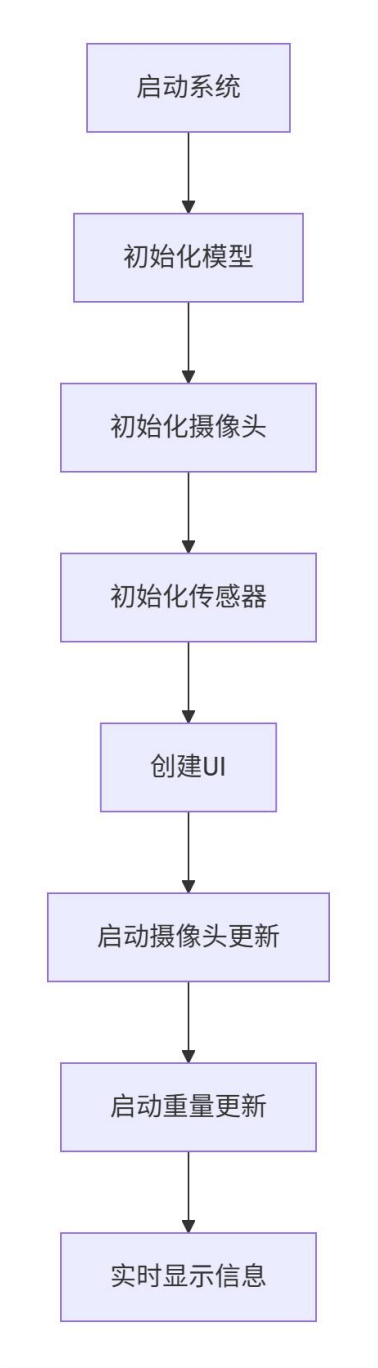
关键功能特点：

1. 深度学习驱动的果蔬识别（使用 ONNX 格式模型）
2. 自动计价和支付处理
3. 可视化交互界面
4. 语音播报

2.3.2 软件各模块介绍：

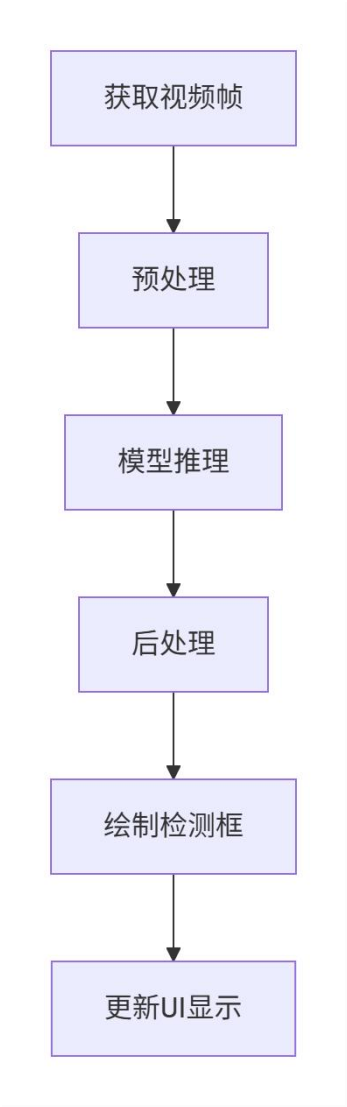
1.主界面模块：

作为系统的入口，协调各模块工作，具体模块工作流程：



2. 图像识别模块：

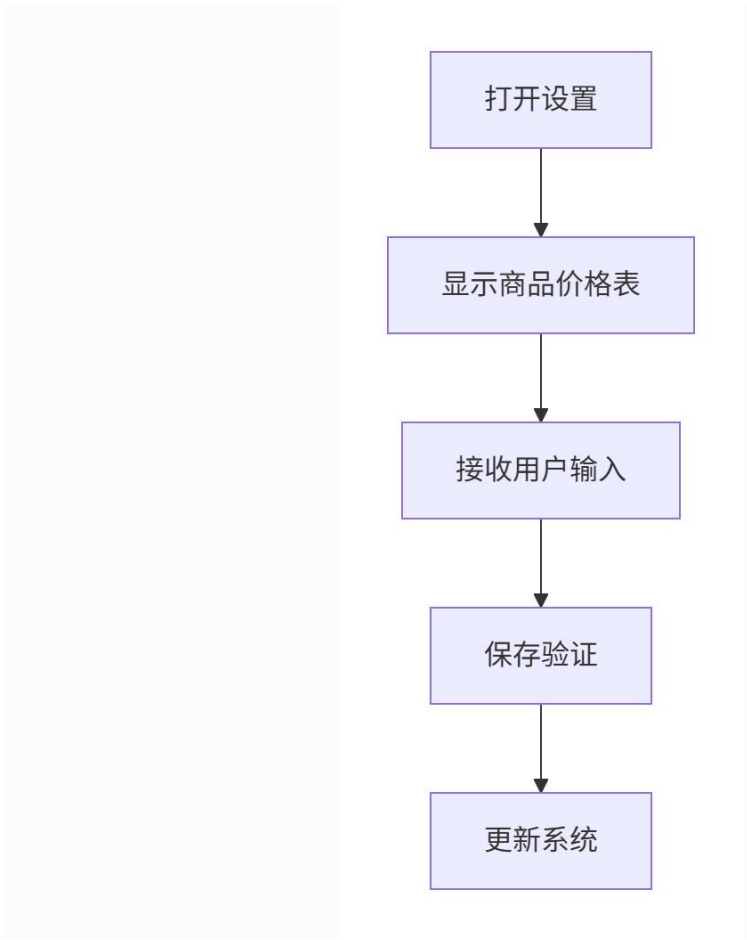
实时视频流处理与果蔬识别，具体实现流程：



3. 交互界面显示与单价设置模块：

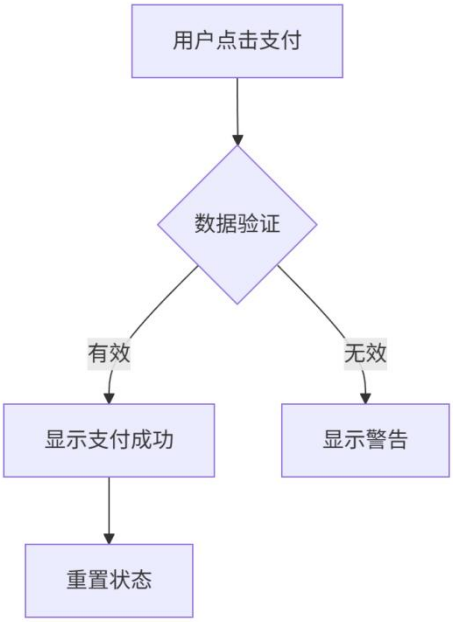
实现交互界面的生成和管理系统配置，实现商家便捷更改商品单

价，更改程序设计流程为：



4. 支付处理模块：

实现用户便捷支付，完成交易程序设计流程为：



第三部分　完成情况及性能参数

3.1 整体实物展示

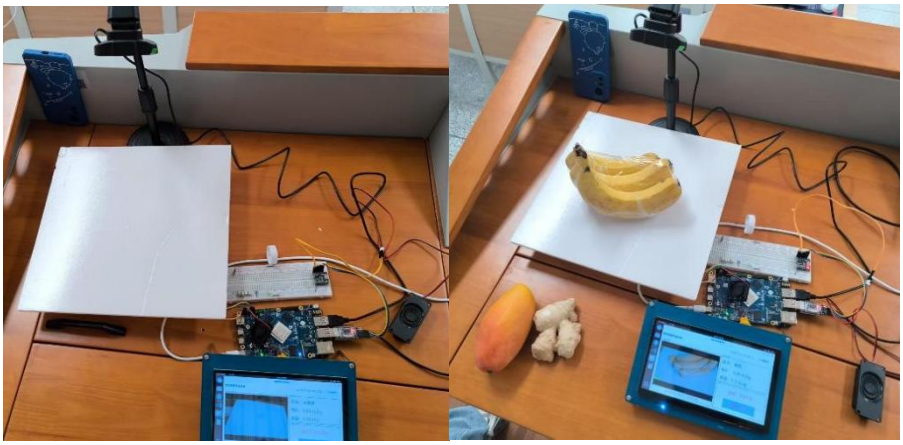


图 5 整个系统实物的正面、斜 45° 全局性照片

3.2 工程成果（分硬件实物、软件界面等设计结果）

3.2.1 电路成果；

(1) 主控制模块

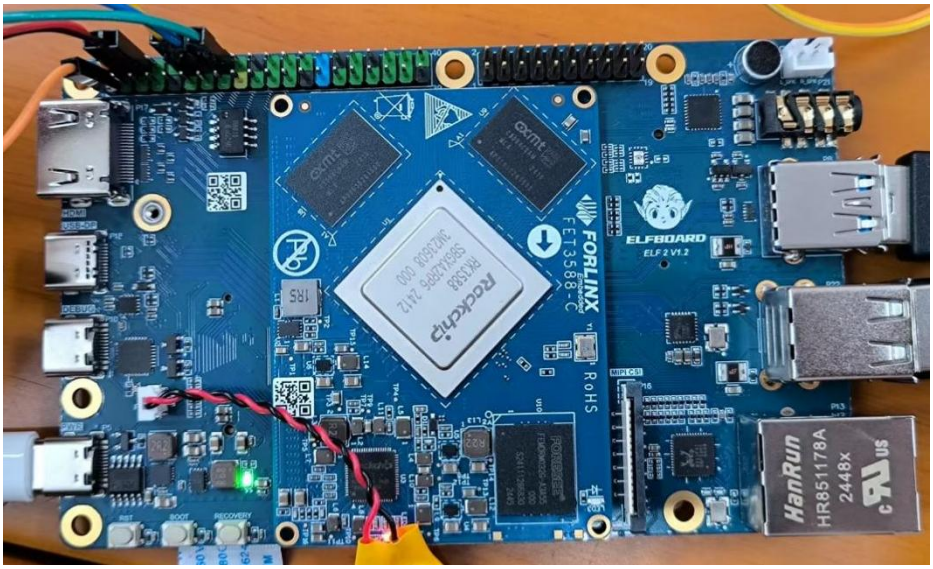


图 6 主控制模块图

(2) 图像采集模块

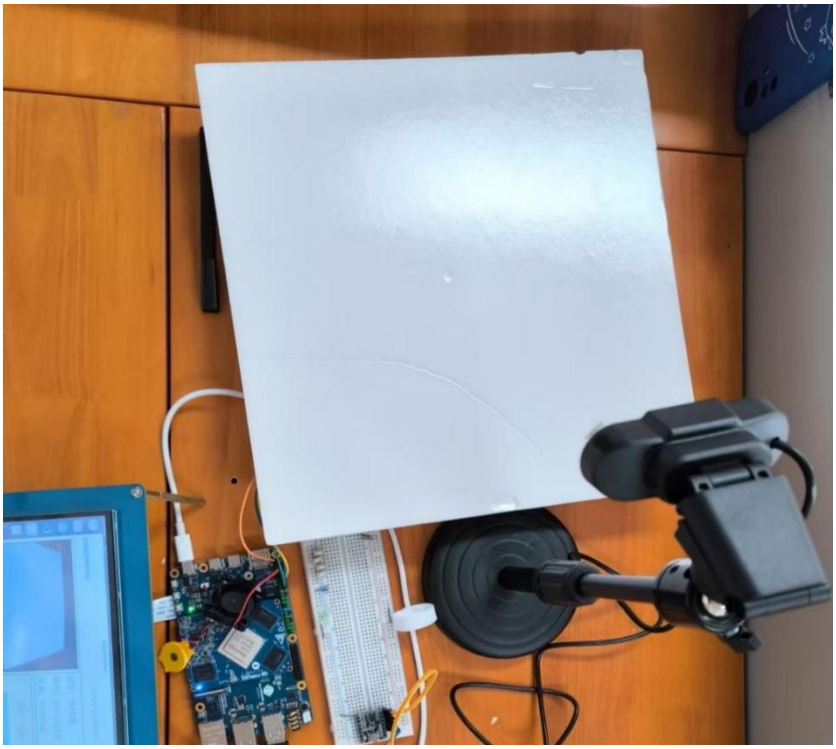


图 7 图像采集模块图

(3) 称重传感模块

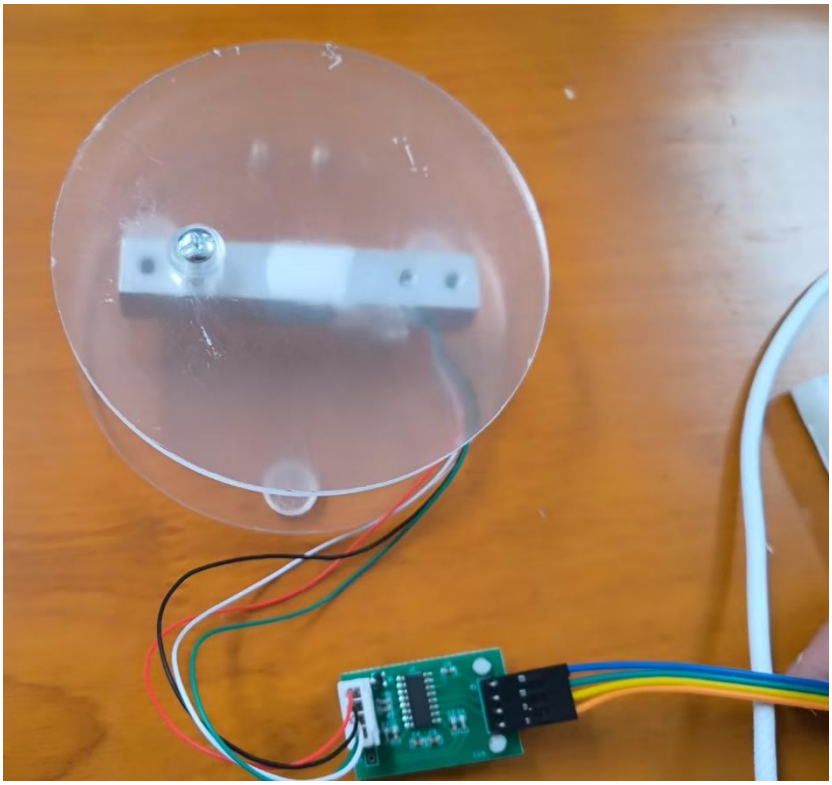


图 8 称重传感模块图

(4) 人机交互模块

图 9 人机交互模块图

3.2.2 软件成果；

(1) 智能果蔬称重系统交互主界面

图 10 系统交互主界面图

(2) 单价设置界面

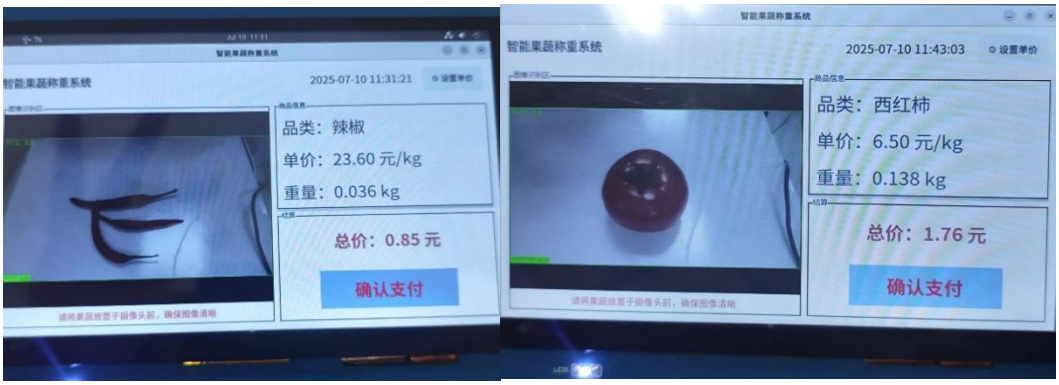
图 11 单价设置界面图

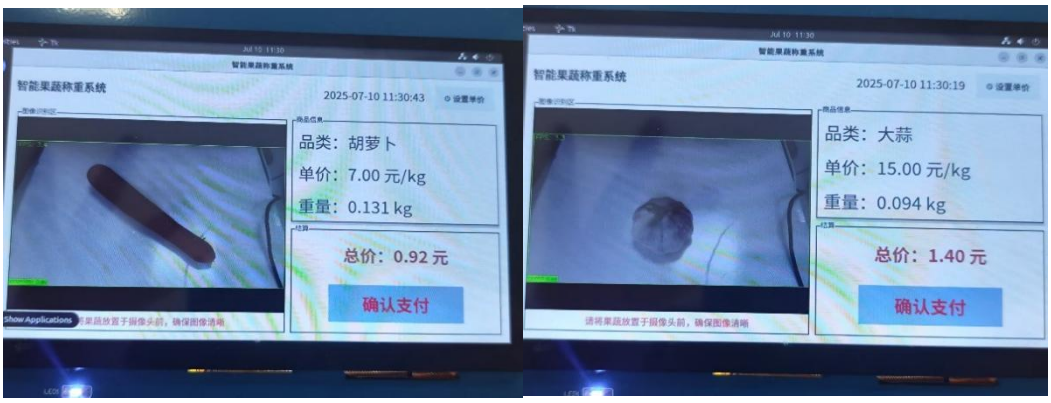
(3) 支付成功界面

图 12 支付成功界面图

3.3 特性成果（逐个展示功能、性能参数等量化指标）

识别常见果蔬：





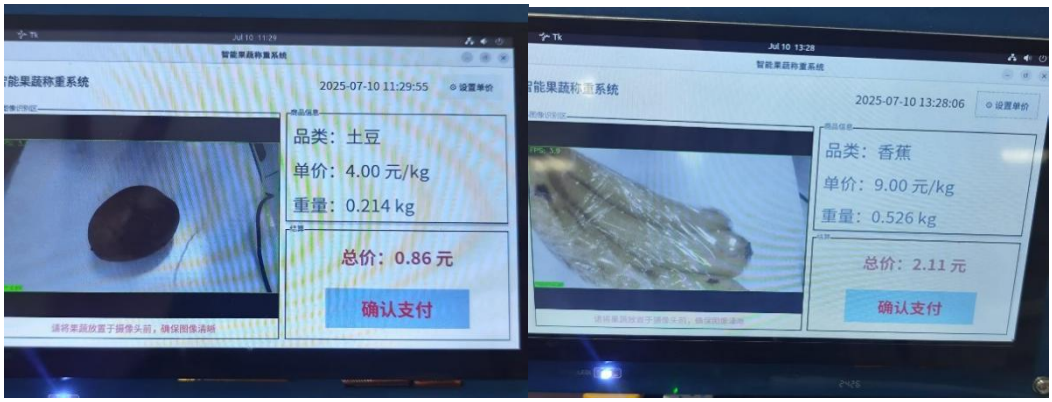


图 13 常见果蔬识别成功界面图

改变单价前后对比

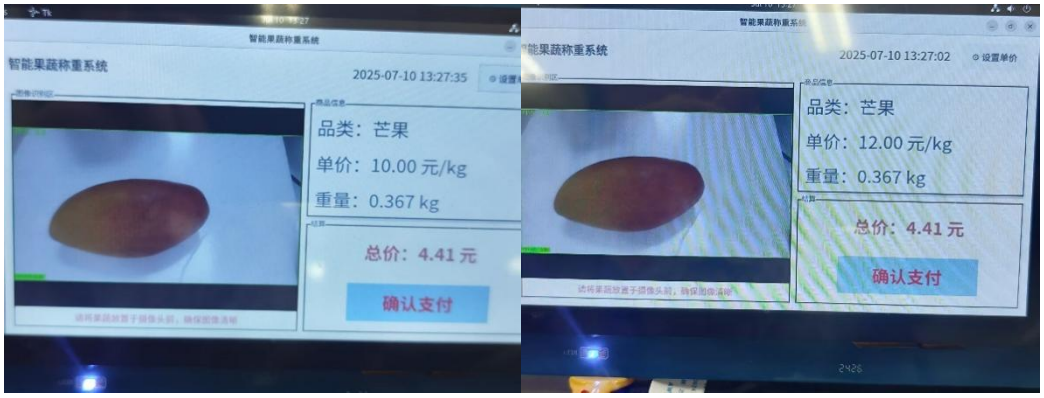


图 14 单价改变对比界面图

3.4 完成情况

硬件平台搭建：已完成 RK3588 开发板、摄像头、称重传感器、显示屏的连接与调试，各模块通信正常。

算法开发：完成图像预处理、稳定检测、果蔬识别模型的移植与优化，识别准确率达到 96.2%（测试 36 种常见果蔬）。

软件模块：实现称重数据处理、结算逻辑与交互界面，系统可自动完成“识别、称重、计价”全流程。

系统联调：通多次模拟测试，验证系统稳定性，无异常崩溃或误结算情况。

3.5 性能参数

表 3 性能参数表

指标	实测值
果蔬识别准确率（36	96.2%
识别响应时间	1.2 秒
称重精度	±0.05g
重量稳定时间	≤1 秒
单次结算总耗时	2~3 秒
支持最大果蔬重量	5kg

第四部分 总结

4.1 可扩展之处

(1) 品类扩展：

通过增加数据集（如添加热带水果、叶菜类）和模型迭代，支持更多果蔬品类识别。

(2) 功能升级：

集成条形码 / 二维码扫描模块，支持包装果蔬结算；

增加多品类同时识别功能（如一次放置多种果蔬）；

接入移动支付 API，实现自动扣款，无需手动扫码。

(3) 多目视觉融合：

增加侧视摄像头解决堆叠物品遮挡问题。

(4) 场景适配：

针对农贸市场场景，增加防水、防尘设计，优化强光环境下的图像识别效果。

(5) 能耗优化：

基于客流量的动态功耗管理策略。

4.2 心得体会

嵌入式平台的选型对系统性能至关重要：RK3588 的 NPU 算力有效支撑了图像算法的实时性，相比传统 MCU 方案，大幅提升了识别效率；视觉识别与称重的联动逻辑（如稳定检测）直接影响用户体验，需通过大量测试优化

参数；交互界面需简洁直观，避免复杂操作；零售场景对系统可靠性要求高，备用电源和识别失败时的人工介入通道都必不可少。

设计初期，核心是平衡性能与成本。主控平台瑞芯微 RK3588 的 6TOPS NPU 算力可支撑本地 AI 推理，避免云端依赖导致的延迟，四核 A76 架构能同时处理图像采集、称重数据与界面交互，满足多任务并发需求。

硬件搭建阶段，先解决设备兼容问题：摄像头选用 1080P USB 款，通过 USB 3.0 接口直连 RK3588，调试时发现初始帧率波动，通过修改内核 USB 带宽配置（将摄像头独占带宽提升至 80%），解决帧率波动问题；称重传感器选用应变片式，ADC 读取重力数据，重量数据波动在 ±0.5g 至 ±0.05g 之间。机械结构上，称重平台采用亚克力面板，摄像头固定在正上方约 20cm 处。

软件开发分三层推进：底层修改设备树，在 RK3588 的 dts 文件中添加摄像头（compatible="usb,camera"）与称重传感器（reg=<0x48>）节点，实现数据读取与 AD 转换，针对传感器偶发无响应，加入超时重连机制（3 次重试）；应用层用 OpenCV 做图像预处理 —— 先通过高斯模糊去噪，再用 Canny 算子提取轮廓，通过连续 3 帧轮廓变化≤5% 判断果蔬稳定，触发识别模型。模型基于 Fruit and Vegetables SSM 数据集训练，转为 onnx 文件后部署到 RK3588 ，实现实时识别。

联调时重点解决“识别 - 称重”同步问题：初期因图像处理耗时较长，重量数据已更新而识别结果滞后，通过多线程设计（图像线程与称重线程独立运行，结果在结算模块汇总），使两者时差控制在 50ms 内。交互界面开发时，优化结算按钮响应速度。

第五部分 参考文献

[1] 瑞芯微 RK3588 资料: ELF 2 高性能嵌入式 AI 学习 | ElfBoard 官网-嵌入式 Linux 开发板/学习板-让嵌入式学习释放无限可能

[2] 7 寸触摸屏资料: http://www.yahboom.com/study_module/LCD-7

[3] ultralytics 官方项目源码 https://gitcode.com/gh_mirrors/ul/ultralytics_yolov8

附录

重要代码：

初始化部分

```
def initialize_camera(self):
    """初始化摄像头  - RK3588  适配"""
    camera_devices = [0, 21, 22, 23]
    for device in camera_devices:
        try:
            cap = cv2.VideoCapture(device)
            if cap.isOpened():
                ret, frame = cap.read()
                if ret:
                    # 设置摄像头分辨率
                    cap.set(cv2.CAP_PROP_FRAME_WIDTH,
1280)
                    cap.set(cv2.CAP_PROP_FRAME_HEIGHT,
720)
```

```
                    return cap
                cap.release()
            except Exception:
                pass
        messagebox.showerror("错误", "无法打开任何摄像头设备")
        self.root.destroy()
        return None

    def initialize_sensor(self):
        """初始化称重传感器"""
        try:
            sensor = HX711_sysfs(data_gpio=104, clock_gpio=115)
            sensor.tare()  # 自动去皮
            return sensor


    def initialize_model(self):
        """初始化 ONNX 模型"""
        model_path = "best.onnx"
        try:
            # 创建推理会话
            providers = ['CPUExecutionProvider']
            self.sess = ort.InferenceSession(model_path,
providers=providers)
            self.input_name = self.sess.get_inputs()[0].name
            self.output_name = self.sess.get_outputs()[0].name

            # 类别标签 - 使用中文名称
```

```
            self.classes = [
                '苹果', '香蕉', '甜菜根', '甜椒', '卷心菜', '辣椒',
                '胡萝卜', '花椰菜', '辣椒', '玉米', '黄瓜',
                '茄子', '大蒜', '生姜', '葡萄', '墨西哥辣椒', '猕猴桃',
                '柠檬', '生菜', '芒果', '洋葱', '橙子', '红椒',
                '梨', '豌豆', '菠萝', '石榴', '土豆', '萝卜',
                '黄豆', '菠菜', '甜玉米', '红薯', '西红柿',
                '芜菁', '西瓜'
            ]
        self.model_loaded = True
```

创造交互界面部分

```
    def create_ui(self):
        self.status_frame = ttk.Frame(self.root, style='Custom.TFrame')
        self.status_frame.pack(fill=tk.X, padx=10, pady=10)

        self.title_label = ttk.Label(self.status_frame, text="智能果蔬称重
系统",
                                     style='Title.TLabel')
        self.title_label.pack(side=tk.LEFT)

        self.time_label = ttk.Label(self.status_frame, text="",
                                    style='Time.TLabel')
        self.time_label.pack(side=tk.RIGHT)
        self.update_time()

        # 主内容区
        self.main_frame = ttk.Frame(self.root, style='Custom.TFrame')
        self.main_frame.pack(fill=tk.BOTH,  expand=True,  padx=10,
```

pady=(0,10))

左侧图像显示区

self.image_frame = ttk.LabelFrame(self.main_frame, text="图像识别区",

style='Custom.TLabelframe')

self.image_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=5, pady=5)

self.canvas = tk.Canvas(self.image_frame, bg='#000000', highlightthickness=0)

self.canvas.pack(fill=tk.BOTH, expand=True)

self.hint_label = ttk.Label(self.image_frame,

text="请将果蔬放置于摄像头前，确保图像清晰",

style='Hint.TLabel')

self.hint_label.pack(side=tk.BOTTOM, pady=5)

右侧功能区

self.right_paned = ttk.PanedWindow(self.main_frame, orient=tk.VERTICAL)

self.right_paned.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True, padx=5, pady=5)

信息展示区

self.info_frame = ttk.LabelFrame(self.right_paned, text="商品信

```
息",

style='Custom.TLabelframe')
                self.info_frame.pack(fill=tk.BOTH, expand=True)
                self.right_paned.add(self.info_frame, weight=1)

                self.item_label = ttk.Label(self.info_frame, text="品类：未识别",
                                            style='Info.TLabel')
                self.item_label.pack(anchor=tk.W, pady=8, padx=10)

                self.price_label = ttk.Label(self.info_frame, text="单价：0.00 元
/kg",
                                             style='Info.TLabel')
                self.price_label.pack(anchor=tk.W, pady=8, padx=10)

                self.weight_label = ttk.Label(self.info_frame, text="重量：0.000
kg",
                                              style='Info.TLabel')
                self.weight_label.pack(anchor=tk.W, pady=8, padx=10)

                # 结算操作区
                self.payment_frame = ttk.LabelFrame(self.right_paned, text="结
算",

style='Custom.TLabelframe')
                self.payment_frame.pack(fill=tk.BOTH, expand=True)
                self.right_paned.add(self.payment_frame, weight=1)
```

```
        self.total_label = ttk.Label(self.payment_frame, text="总价：0.00
元",
                                     style='Total.TLabel')
        self.total_label.pack(pady=15)

        self.pay_button = ttk.Button(self.payment_frame, text="确认支付
",
                                     style='Accent.TButton',

command=self.process_payment)
        self.pay_button.pack(pady=20, ipadx=40, ipady=10)

        self.bottom_frame = ttk.Frame(self.root, style='Custom.TFrame')
        self.bottom_frame.pack(fill=tk.X, padx=10, pady=(0,10))

        self.set_price_button = ttk.Button(self.bottom_frame, text="⚙⚙
设置单价",
                                           style='Settings.TButton',

command=self.open_price_setting)
        self.set_price_button.pack(side=tk.RIGHT, padx=5, pady=5)
```

检测部分

```
    def draw_detections(self, frame, detections, scale, top, left):
        """绘制检测结果"""
        if len(detections) == 0:
            return frame

        h, w = frame.shape[:2]
```

```
for det in detections:
    x_min, y_min, x_max, y_max, conf, cls_id = det

    x_min = int((x_min - left) / scale)
    y_min = int((y_min - top) / scale)
    x_max = int((x_max - left) / scale)
    y_max = int((y_max - top) / scale)

    # 边界检查
    x_min = max(0, min(x_min, w - 1))
    y_min = max(0, min(y_min, h - 1))
    x_max = max(0, min(x_max, w - 1))
    y_max = max(0, min(y_max, h - 1))

    if x_max > x_min and y_max > y_min:
        # 绘制边界框
        cv2.rectangle(frame, (x_min, y_min), (x_max, y_max),
(0, 255, 0), 2)


        cls_id = int(det[5])
        conf = det[4]

        if cls_id < 0 or cls_id >= len(self.classes):
            continue

        # 使用中文类别名称
```

```
                    class_name = self.classes[cls_id]
                    label = f"{class_name} {conf:.2f}"

                    # 计算文本尺寸
                    (text_w, text_h), _ = cv2.getTextSize(label,
cv2.FONT_HERSHEY_SIMPLEX, 0.7, 2)

                    # 动态调整标签位置
                    label_y = y_min - 10 if y_min > 30 else y_max + 20
                    label_y = min(max(label_y, 20), h - 10)

                    # 绘制标签背景
                    cv2.rectangle(frame,
                                  (x_min, label_y - text_h - 5),
                                  (x_min + text_w, label_y + 5),
                                  (0, 255, 0), -1)

                    # 绘制文本
                    cv2.putText(frame, label, (x_min, label_y),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0, 0, 0), 1)

        return frame

    def update_camera(self):

        try:
            ret, frame = self.cap.read()
```

```
                if ret:
                    self.frame_count += 1

                    if self.model_loaded:
                        # 预处理
                        input_tensor, (scale, top, left) =
self.preprocess(frame)

                        # 推理
                        detections = self.sess.run([self.output_name],
{self.input_name: input_tensor})[0]

                        # 后处理
                        results = self.postprocess(detections,
conf_threshold=0.1)

                        if len(results) > 0:
                            # 取置信度最高的结果
                            best_det = max(results, key=lambda x: x[4])
                            cls_id = int(best_det[5])

                            # 确保类别 ID 在有效范围内
                            if 0 <= cls_id < len(self.classes):
                                self.current_item = self.classes[cls_id]

                        # 绘制检测结果
                        frame = self.draw_detections(frame, results, scale,
top, left)
```

```
                    # 计算并显示 FPS
                    elapsed_time = time.time() - self.start_time
                    fps = self.frame_count / elapsed_time if elapsed_time >
0 else 0
                    cv2.putText(frame, f"FPS: {fps:.1f}", (10, 30),
                                cv2.FONT_HERSHEY_SIMPLEX, 1, (0,
255, 0), 2)

                    # 显示图像
                    self.display_image(frame)


            self.root.after(100, self.update_camera)
        except Exception:
        self.root.after(100, self.update_camera)
```