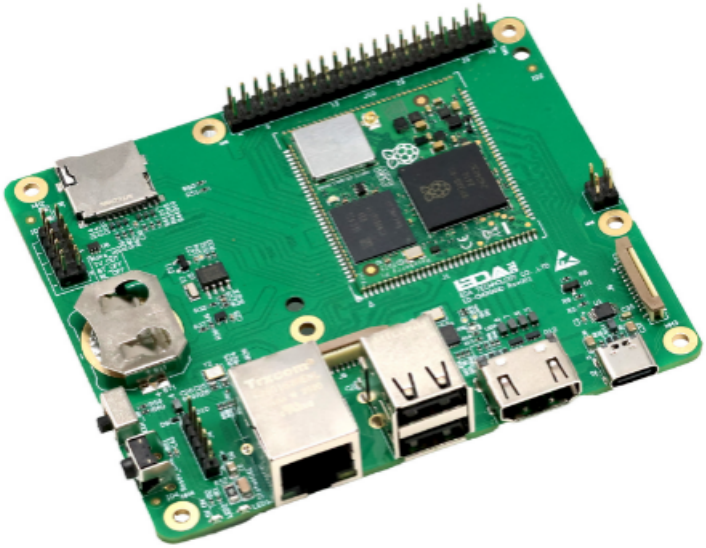
工业树莓派 CM0 NANO 单板计算机

——边缘 AI 与物联网应用案例



作者：无垠的广袤

时间：2026年06月30日

首发：微信公众号——树莓派开发者

目录

工业树莓派 CM0 NANO 单板计算机
——边缘 AI 与物联网应用案例
目录
【工业树莓派 CM0 NANO 单板计算机】介绍、镜像烧录、系统测试
介绍
外观
Top view
Bottom view
包装清单
特点
资源参数
系统框图
核心板
准备工作
软件安装
镜像下载
系统安装
系统测试
硬件连接
系统配置
SSH 登录
总结
【工业树莓派 CM0 NANO 单板计算机】本地部署 Home Assistant 智能家居平台

项目介绍
应用
硬件连接
环境搭建
Docker
加速
部署方案
Home Assistant
加速
镜像获取
启动容器
参数配置
总结
【工业树莓派 CM0 NANO 单板计算机】本地部署 EMQX
项目介绍
应用
硬件连接
Docker
网络加速
部署方案
EMQX
部署方案
启动容器
创建用户
通信测试
MQTT 转发
芯片温度上报
流程图
代码
总结
【工业树莓派 CM0 NANO 单板计算机】HACS 安装和米家集成
项目介绍
准备工作
硬件连接
操作系统
Docker 安装
Home Assistant
更换镜像源
拉取镜像
创建容器
HACS
下载
添加集成
Xiaomi Home
登录
添加 MIoT 设备
总结
【工业树莓派 CM0 NANO 单板计算机】小智语音聊天
项目介绍
硬件连接
环境搭建
程序运行
设备激活
效果演示
动态演示
语音唤醒
模型下载
下载模型包

        配置方案
    启用语音唤醒
    重启客户端
  总结
【工业树莓派 CM0 NANO 单板计算机】科学计算解决方案：常微分方程组的数值求解
  项目介绍
  准备工作
    硬件连接
    系统安装
    软件更新
  环境搭建
  Runge-Kutta 算法
  数值计算
    Lorenz 吸引子
    代码
    效果
  二能级系统
    Hamiltonian
    主方程
    代码
    效果
  布居振荡
    模型
    主方程
    代码
    效果
  总结
【工业树莓派 CM0 NANO 单板计算机】车牌识别
  项目介绍
  准备工作
    硬件连接
    OpenCV 安装
    字体安装
    Ultralytics 部署
  车牌识别
    模型
    流程图
    代码
    效果
  总结
【工业树莓派 CM0 NANO 单板计算机】人脸识别
  项目介绍
  准备工作
    硬件连接
    OpenCV 安装
  人脸识别
    模型
    训练图片
    文件目录
  流程图
  代码
  效果
  总结
【工业树莓派 CM0 NANO 单板计算机】基于舵机和人脸识别的智能门禁系统
  项目介绍
  准备工作
    硬件连接
    OpenCV 安装

        人脸识别
            模型获取
            训练图片
    舵机控制
        代码
        效果
    门禁系统
        文件目录
        流程图
        代码
            Flask 后端
            Web 前端
            白名单
        效果
    总结
【工业树莓派 CM0 NANO 单板计算机】YOLO26 部署方案
    项目介绍
    准备工作
        硬件连接
        OpenCV 安装
        Ultralytics 部署
    YOLO26
        模型获取
        流程图
    目标检测
        代码
        效果
    实例分割
        代码
        效果
    图像分类
        代码
        效果
    姿态估计
        代码
        效果
    旋转框检测
        代码
        效果
    总结
【工业树莓派 CM0 NANO 单板计算机】MLX90640 热成像仪
    项目介绍
    模块设计
        3D 渲染图
        原理图
        实物图
    硬件连接
        实物图
    环境搭建
    连接测试
    通信测试
        效果
    图像显示
        效果
    图像优化
        效果
    总结
【工业树莓派 CM0 NANO 单板计算机】步进电机的远程控制与LabVIEW数据采集

项目介绍
硬件连接
OLED
步进电机
环境搭建
MQTT Broker
OLED 驱动
代码
效果
步进电机驱动
代码
效果
远程控制
流程图
代码
效果
LabVIEW 上位机
前面板
程序面板
运行效果
数据采集
总结
视频介绍

【工业树莓派 CM0 NANO 单板计算机】介绍、镜像烧录、系统测试

本文介绍了树莓派 CM0 NANO 单板计算机的相关信息，包括资源分布、参数特点等，并演示了系统镜像烧录和 SSH 远程登录等流程。

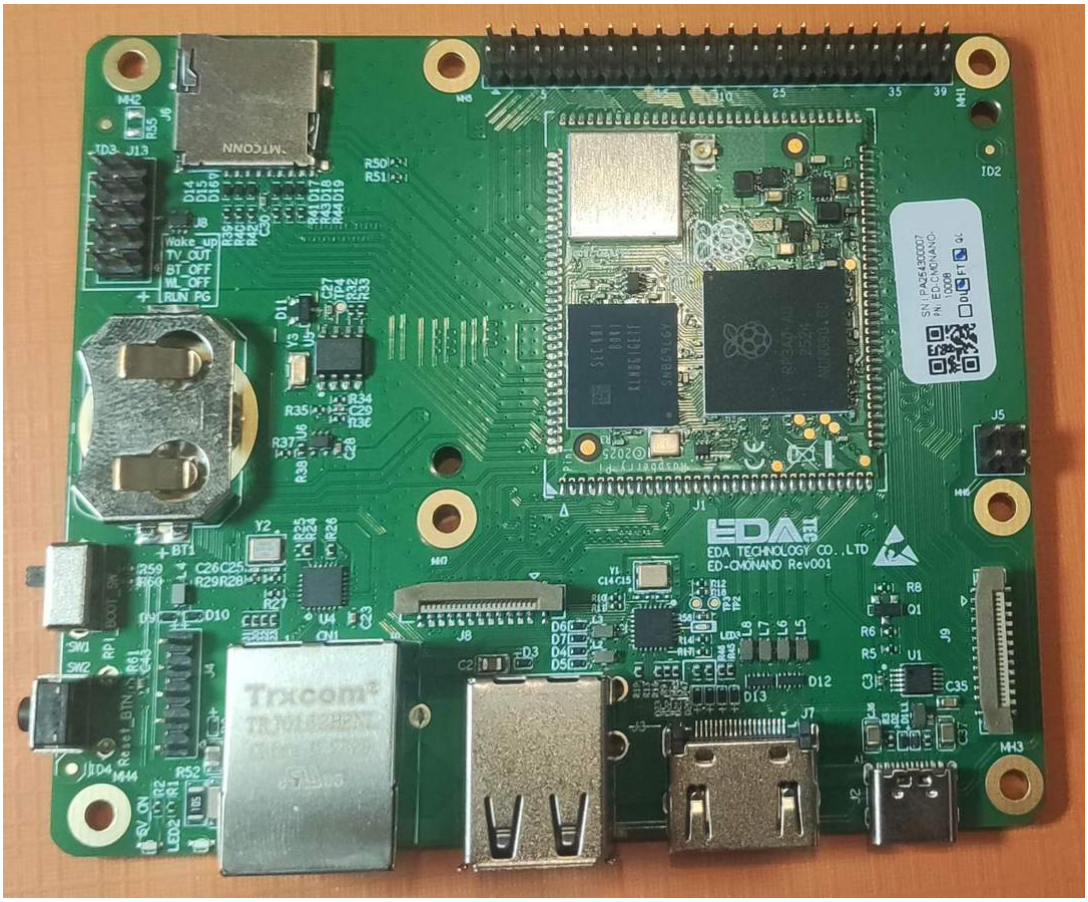
介绍

ED-CM0NANO 是一款基于 Raspberry Pi CM0 的单板计算机，默认为 512MB RAM，根据不同的应用场景和用户需求可选择不同规格的eMMC 的计算机系统。

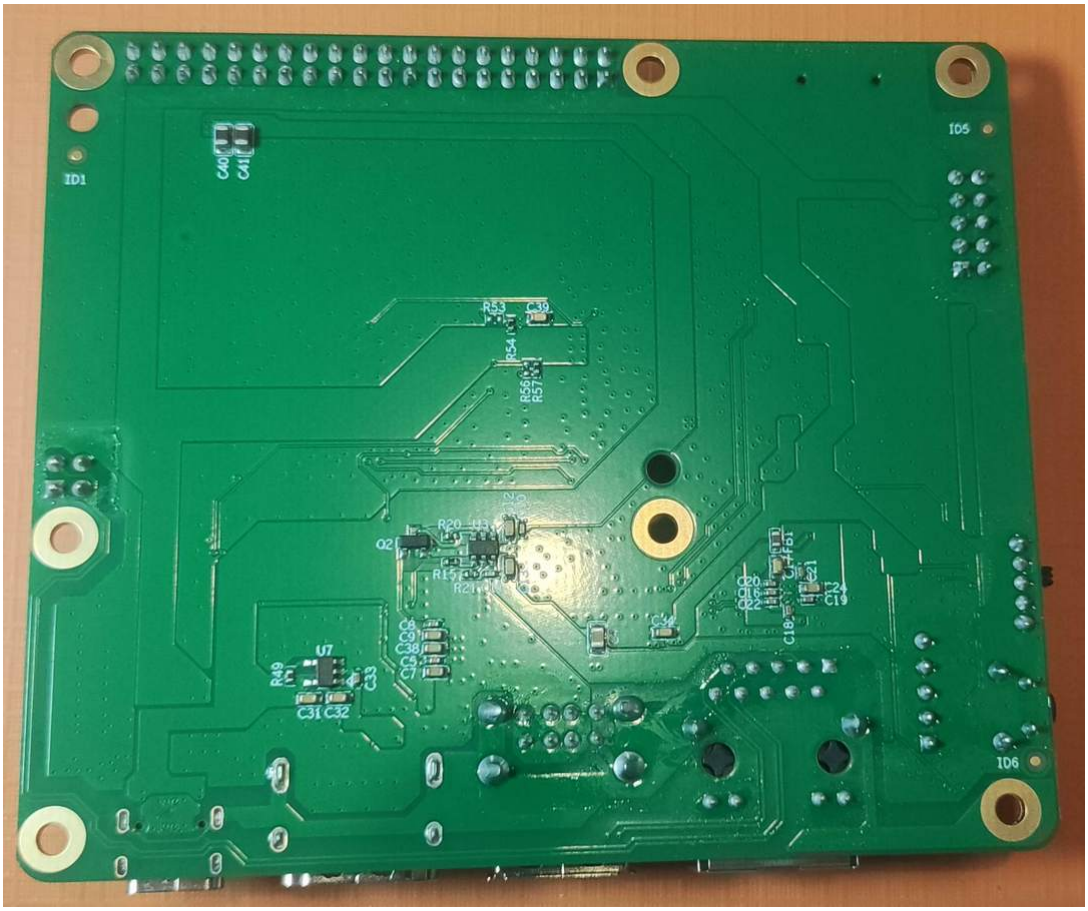
ED-CM0NANO 提供 HDMI、USB、MIPI DSI、MIPI CSI 和 Raspberry Pi 40-Pin 接口，支持选配 Wi-Fi（带外置天线）来接入网络，集成RTC 和 Watch Dog，主要用于工业控制和物联网领域。

外观

Top view



Bottom view



包装清单

- 1 x ED-CM0NANO主板
- [选配Wi-Fi/BT版本] 1 x FPC Wi-Fi/BT天线

特点

- 1GHz四核64位Arm Cortex-A53 处理器
- 512MB SDRAM，8GB/16GB的eMMC可选
- 1 x HDMI、2 x USB 2.0、1 x 100M网口
- 1 x 4-lane MIPI DSI、1 x 4-lane MIPI CSI
- 支持Raspberry Pi 40-Pin
- 支持选配2.4GHz Wi-Fi和蓝牙
- USB Type-C接口的5V电源输入
- 集成RTC和Watch Dog，可安装CR2032电池作为RTC备份电源

详见：ED-CM0NANO . ED-CM0NANO用户手册 .

资源参数

面板I/O	
RTC电池底座	1 x RTC电池底座，支持通过此接口安装RTC电池（CR2032）
拨码开关	1 x 拨码开关，支持正常运行模式和烧录模式的切换LD（拨码开关拨到靠近RTC电池底座）：烧录模式RUN（拨码开关拨到靠近复位按键）：正常运行模式（默认状态）
电源接口	1 x DC输入， USB Type-C连接器，支持5V输入
HDMI接口	1 x HDMI，Type-A接口，兼容HDMI 1.3a标准，分辨率支持1080p 30Hz
USB 2.0接口	2 x USB 2.0，双层Type-A接口，每一路最高支持480Mbps传输速率
100M以太网接口	1 x 以太网接口(10M/100M自适应)，RJ45端子，用于接入以太网
Micro SD卡槽	1 x Micro SD卡槽，支持安装Micro SD卡，用于启动系统 **注：Micro SD卡槽仅适用于CM0Lite**

指示灯和按键	
PWR	1 x 电源指示灯，红色，用于查看设备上电和断电的状态
ACT	1 x 系统状态指示灯，绿色，用于查看设备的工作状态
RESET	1 x 复位按键，隐藏式按键，按下按键可重新启动设备

扩展接口	
PoE HAT接口	1 x PoE HAT接口，2x2-Pin 2.54mm间距排针，支持扩展连接Raspberry Pi PoE HAT模块
IPEX-1接口	1 x IPEX-1接口，支持连接外置天线
Raspberry Pi 40-Pin接口	1 x Raspberry Pi 40-Pin接口，2x20-Pin 2.54mm间距排针，引脚定义包含GPIO2~GPIO27、2 x 3V3、2 x 5V、1 x ID_SD和1 x ID_SC，支持连接Raspberry Pi标准配件
4-lane CSI接口	1 x 4-lane CSI接口，22-Pin FPC连接器，支持扩展连接Raspberry Pi Camera
4-lane DSI接口	1 x 4-lane DSI接口，22-Pin FPC连接器，支持扩展连接Raspberry Pi Touch Display
USB 2.0接口	1 x USB 2.0接口，1x5-Pin 2.54mm间距排针，引脚定义为VBUS/USB_DM/USB_DP/GND/NC，支持扩展USB 2.0接口
10-Pin接口	1 x 10-Pin接口，2x5-Pin 2.54mm间距排针，引脚定义为4xGND/RUN_PG/WL_ON/BT_ON/TV_OUT/RUN_PG_BUF/GLOBAL_EN，预留功能短接GND（Pin 1）和RUN_PG（Pin 2）引脚：使ED-CM0NANO关机短接GND（Pin 3）和WL_ON（Pin 4）引脚：关闭Wi-Fi功能短接GND（Pin 5）和BT_ON（Pin 6）引脚：关闭蓝牙功能可在RUN_PG_BUF（Pin 9）和GLOBAL_EN（Pin 10）引脚之间增加按键，可用于将休眠的ED-CM0NANO唤醒

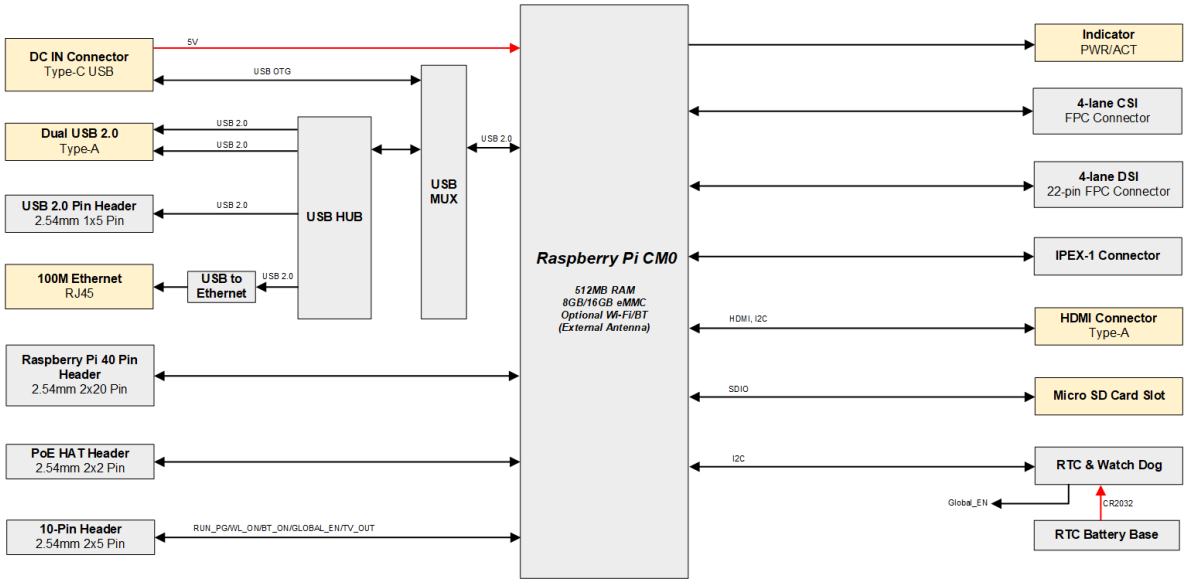
扩展功能	
RTC	内置RTC功能，且提供RTC底座，支持安装CR2032电池作为RTC备份电源
Watch Dog	内置Watch Dog功能，提高了系统的可靠性

电气参数	
输入电压	5V DC **注：推荐使用5V 3A的适配器**
最大功耗	10W

无线	
Wi-Fi/蓝牙(选配)	支持2.4GHz Wi-Fi和蓝牙，带FPC天线2.4GHz Wi-Fi：兼容IEEE 802.11 b/g/n蓝牙5.0：兼容频段2402MHz ~ 2480MHz

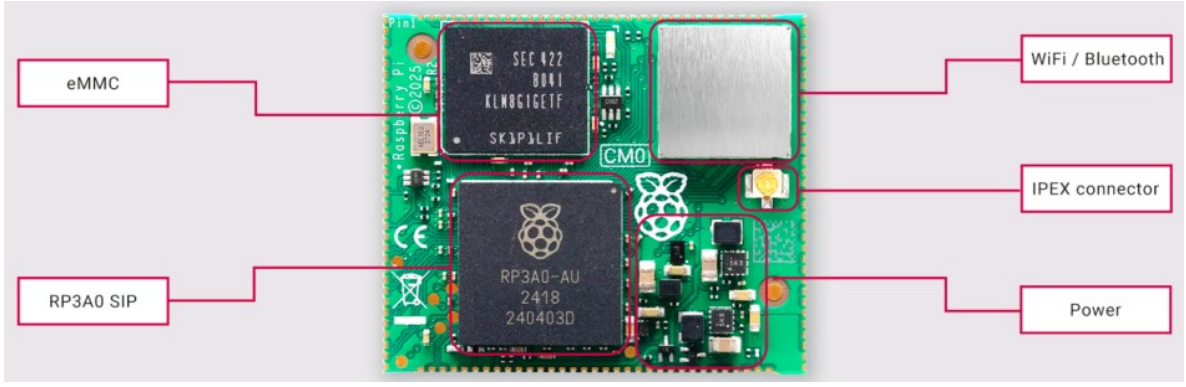
详见：ED-CM0NANO .

系统框图



核心板

核心板为树莓派 CM0



主要特性包括

- **高性能 SoC**
  Broadcom BCM2837 四核 Cortex-A53（Armv8）64 位处理器，主频 1 GHz。
- **紧凑模块设计**
  尺寸仅 39 mm × 33 mm × 2.8 mm。
- **视频支持**
  可全高清（1080p）30 fps 编解码：
  - 解码：H.264 或 MPEG-4（1080p30）
  - 编码：H.264（1080p30）
- **图形支持**
  支持 OpenGL ES 1.1 与 2.0 的 2D/3D 图形。
- **内存（RAM）**
  512 MB LPDDR2。
- **可选闪存**
  板载 eMMC 可选 8 GB 或 16 GB。
- **CM0Lite 额外 SDIO 接口**
  无板载 eMMC 时，可外接存储或扩展外设（仅 CM0Lite）。
- **可选无线连接**
  可选 Wi-Fi 与蓝牙，并带外接天线座。

- **GPIO**
  28 个引脚。
- **摄像头支持**
  4 通道 CSI 摄像头接口。
- **显示支持**
  - HDMI
  - 4 通道 DSI（Display Serial Interface）
  - DPI（Display Parallel Interface）
  - 复合视频输出
- **电源输入**
  单 5 V 供电。

详见：DateSheet - Raspberry Pi CM0 .

准备工作

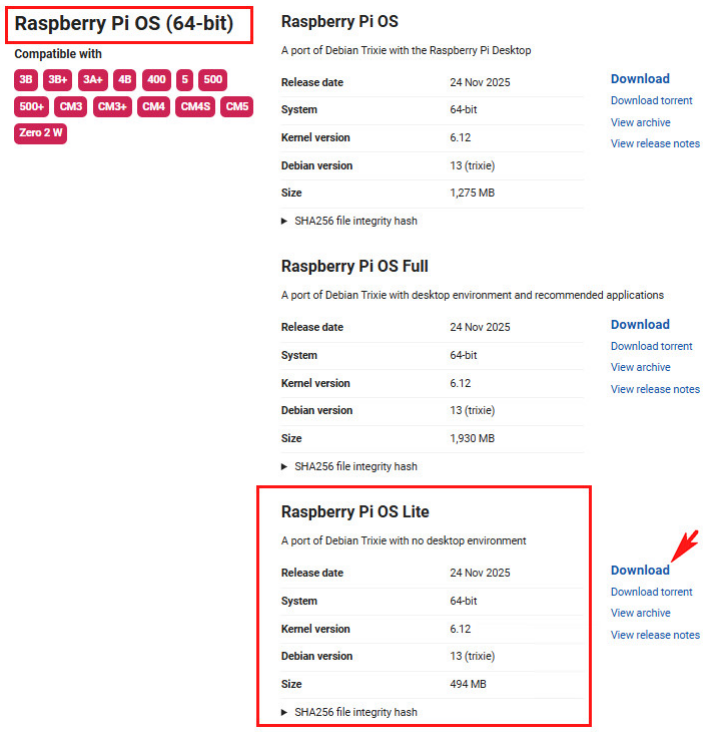
包括系统镜像下载、格式化软件、烧录软件的安装部署等。

软件安装

- 下载并安装 Rpiboot 软件，用以加载设备驱动；
- 下载并安装 SD Card Formatter ， 用以格式化 eMMC 磁盘；
- 下载并安装 Raspberry Pi Imager ， 用以烧录系统镜像文件；

镜像下载

选择 `Raspberry Pi OS(Lite) 64-bit-trixie (Debian 13)` 作为系统镜像；



下载地址：Raspberry Pi OS downloads – Raspberry Pi .

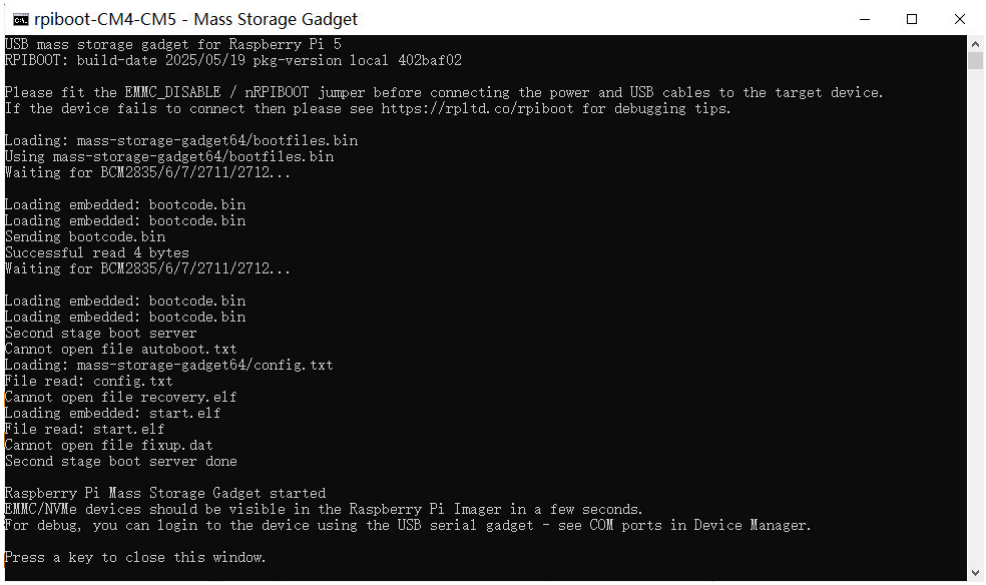
系统安装

CM0 NANO 出厂时，默认未安装操作系统，因此在使用之前需要下载和安装操作系统。

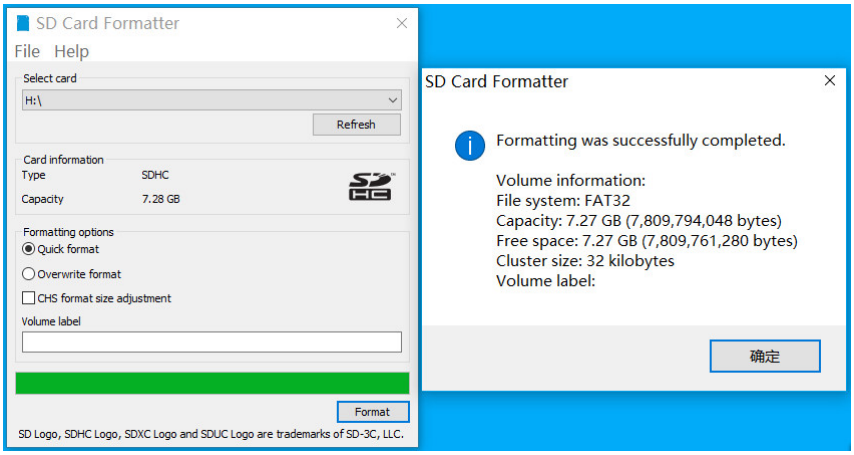
这里给出 eMMC 镜像烧录的具体操作。

- 板载拨码开关，支持正常运行模式和烧录模式的切换。
  - LD（拨码开关拨到靠近RTC电池底座）：烧录模式
  - RUN（拨码开关拨到靠近复位按键）：正常运行模式（默认状态）

- 连接 Type-C 接口和电脑；
- 打开已安装的 rpiboot 工具，自动进行盘符化；



- 待盘符化完成，运行 `SD Card Formatter` 软件，选择目标盘符，单击右下方 `Format` 进行格式化操作；



- 打开 Raspberry Pi Imager 软件，加载系统镜像文件，选择 eMMC 对应的存储设备盘符，开始写入镜像；

- 待写入和校验完成（耗时约 20 分钟），关闭 Raspberry Pi Imager；

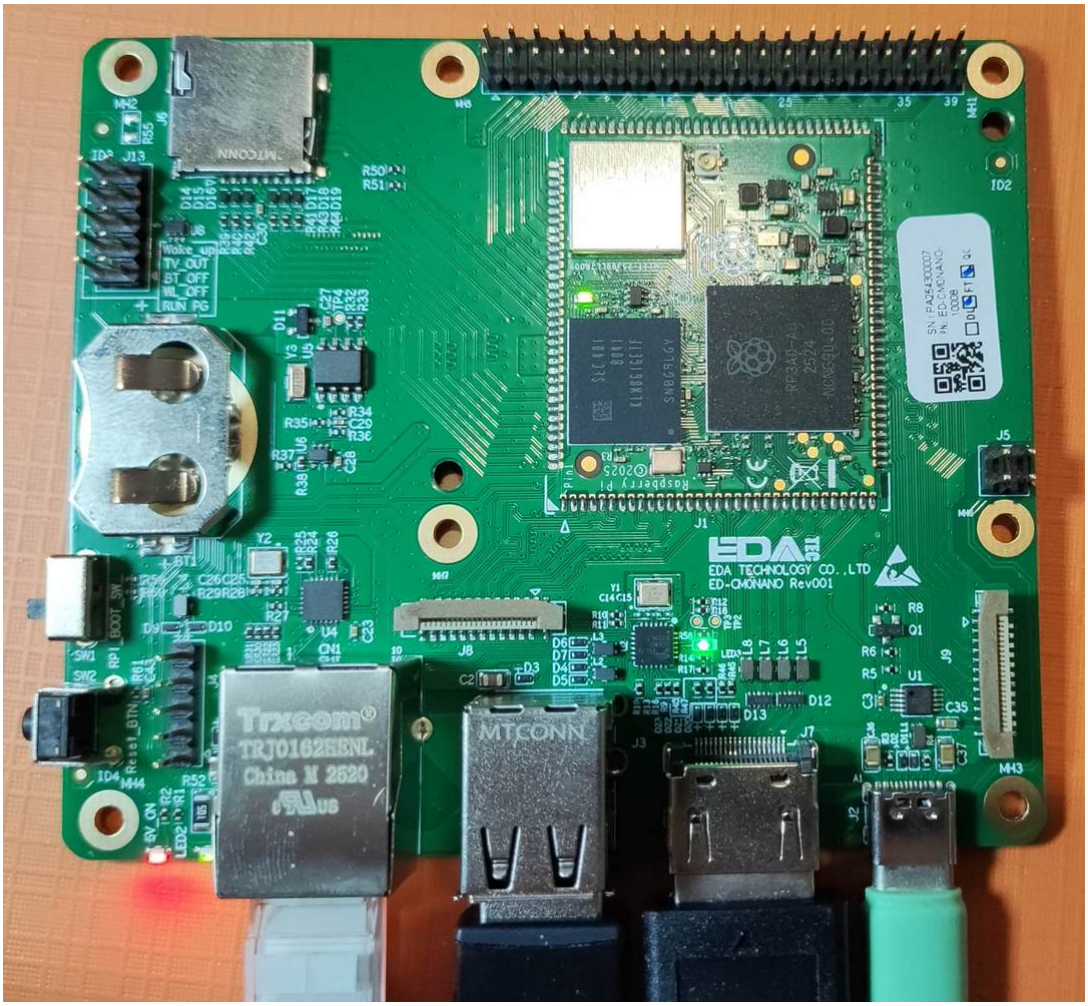
- 拨码开关拨至正常运行模式

系统测试

这里给出 Lite 命令行系统和桌面系统的测试流程。

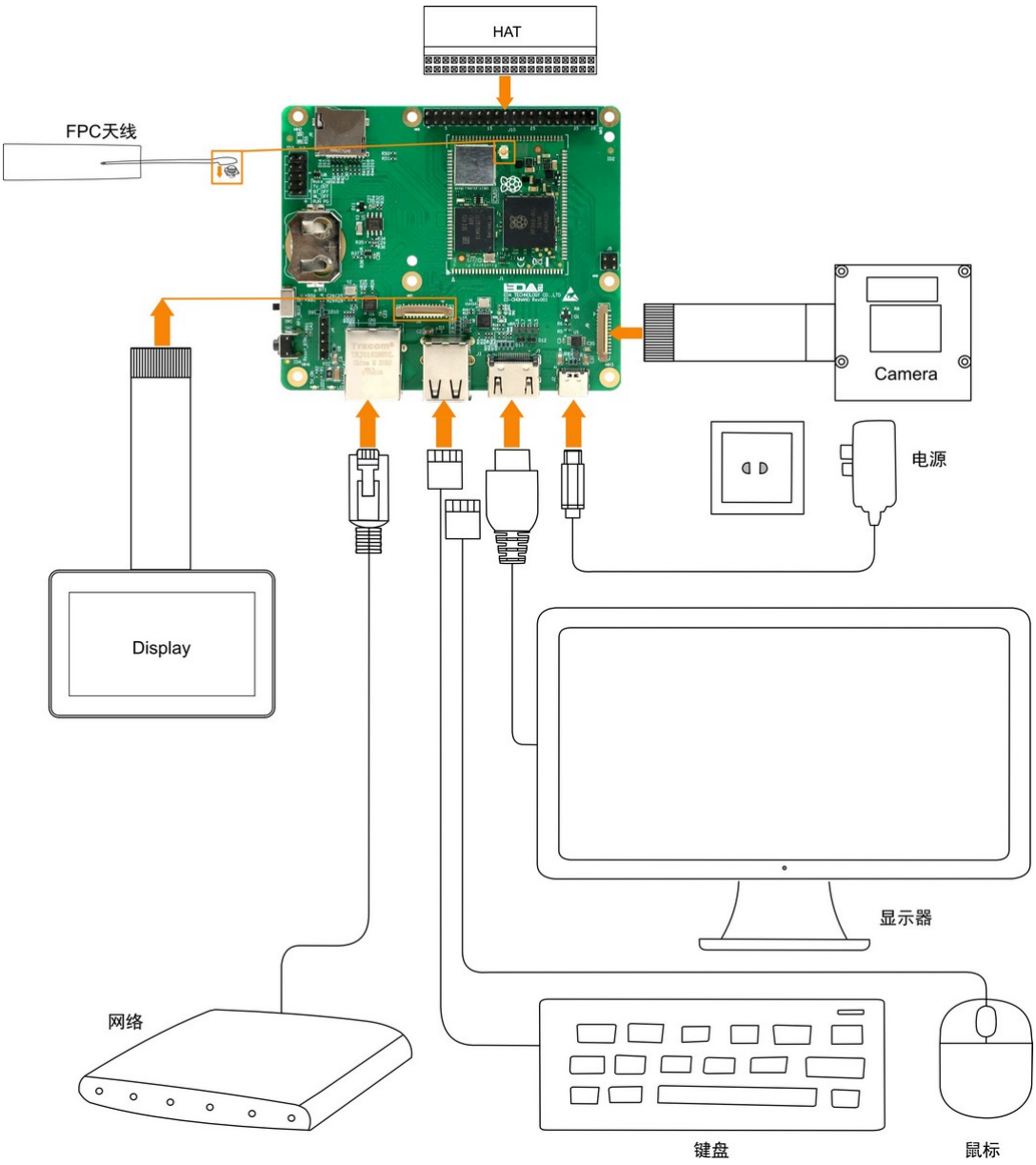
硬件连接

- 网线连接 LAN 接口、连接 HDMI 显示器、USB鼠标和键盘、5V 3A 电源适配器；



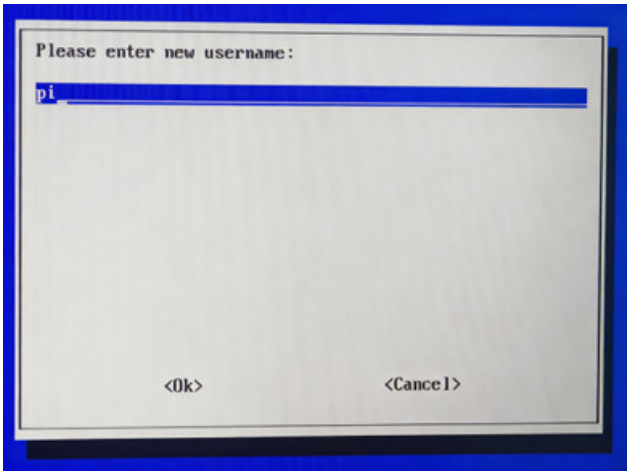
CM0 NANO 接入电源后，系统将开始启动。

- 红色 PWR 灯点亮，表示设备已正常供电；
- 绿灯 ACT 闪烁, 表示系统正常启动；

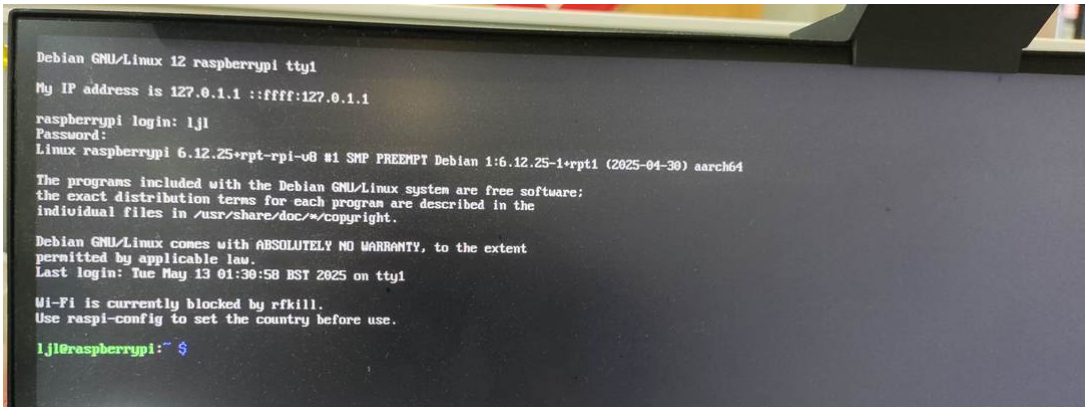


系统配置

- 首次进入系统，需根据提示，设置用户名和密码；

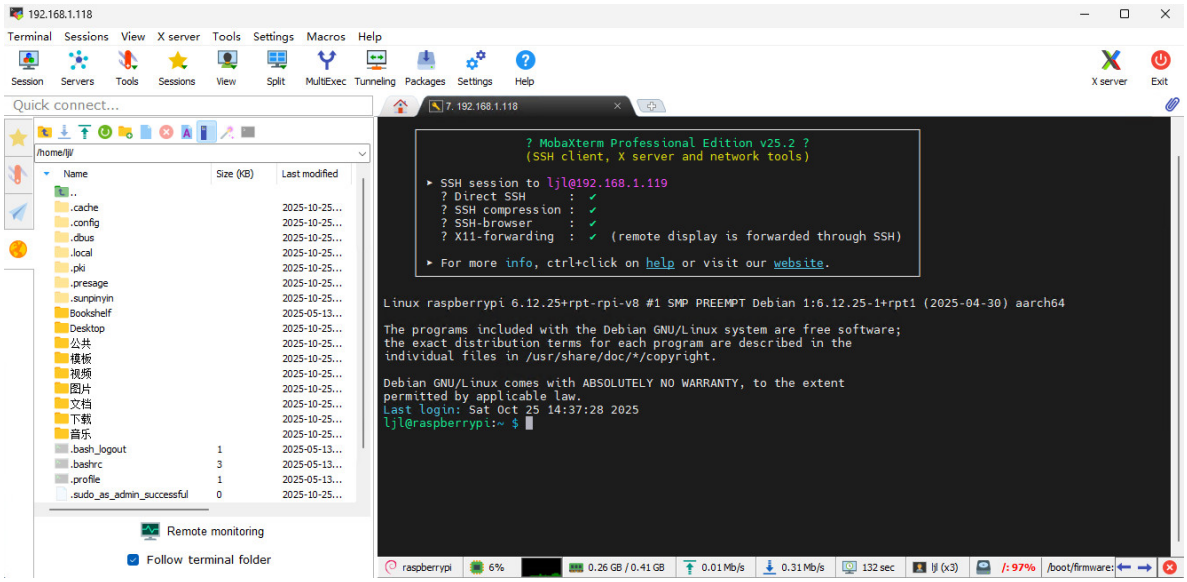


- 登录并进入系统



SSH 登录

- 进入 `首选项` 菜单并打开 `树莓派配置` 窗口；
- 单击 `接口` 选项卡，使能 `SSH`；
- 远程主机打开 MobaXterm 软件，选择 SSH 登录，输入开发板对应的 IP 地址，输入账户信息即可



总结

本文介绍了树莓派 CM0 NANO 单板计算机的相关信息，包括资源分布、参数特点、原理图等，并演示了系统镜像烧录和 SSH 远程登录等流程，为后续的深入研究做好铺垫，也为相关产品的开发设计和快速应用提供了参考。

发布链接：https://blog.csdn.net/qq_36654593/article/details/155359498

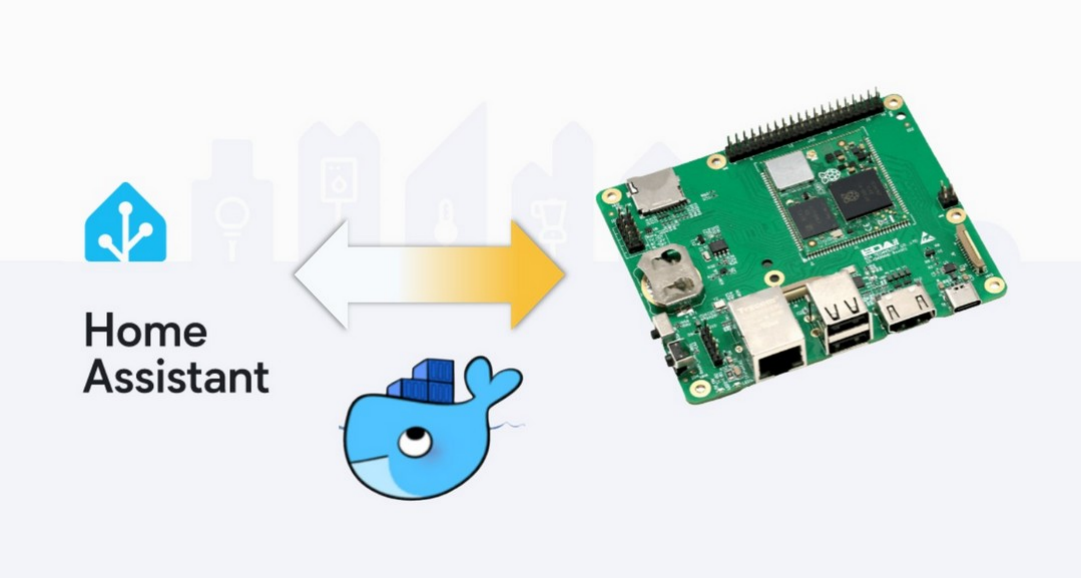
【工业树莓派 CM0 NANO 单板计算机】本地部署 Home Assistant 智能家居平台

本文介绍了树莓派 CM0 NANO 单板计算机通过 Docker 容器实现 Home Assistant 的本地部署，包括网络加速、Docker 安装、Home Assistant 镜像拉取、容器启动和参数配置等流程。

项目介绍

该项目通过 Docker 软件实现 Home Assistant 智能家居平台在树莓派 CM0 NANO 单板计算机的本地部署。

- Docker 安装：网络加速、安装脚本、相关指令和版本检测等；
- Home Assistant 部署：版本选取、更换镜像源、镜像获取、容器启动、参数配置等。

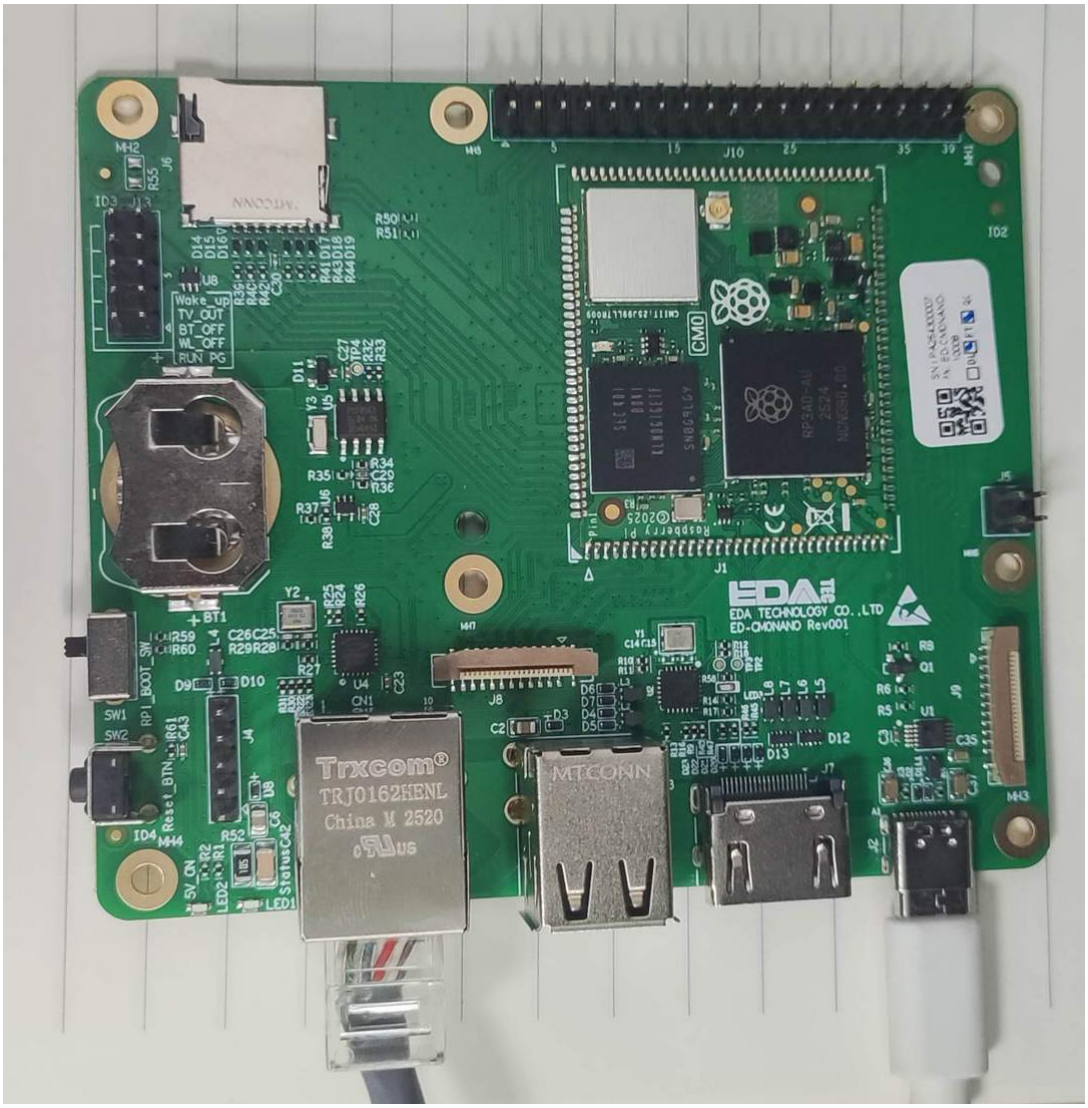


应用

- 智能家居终端控制与数据采集；
- 工业自动化数据采集与远程控制；
- 边缘 IIOT 工业物联网；
- 工业环境监测，历史数据转发与存储；
- 人员流动监控和危险行为检测；
- ……

硬件连接

- 若采用 SSH 远程登录方式，需连接 5V 3A 电源和网线；
- 若采用本地连接方式，需增加 HDMI 数据线连接显示器、USB 键盘及鼠标。



环境搭建

树莓派 CM0 NANO 需安装 RaspberryPi 官方操作系统；

详见：CM0NANO|EDATEC .

Docker

这里介绍树莓派 CM0 实现 Docker 部署的主要流程，包括网络加速及测试、脚本获取与执行、版本检测等。

加速

使用 Dev-Sidecar 软件实现网络加速。

- 执行如下执行，获取安装包并安装软件

```
wget "https://bgithub.xyz/docmirror/dev-sidecar/releases/download/v2.0.0.2/DevSidecar-2.0.0.2-linux-arm64.deb"
sudo dpkg -i DevSidecar-2.0.0.2-linux-arm64.deb
```

- 若报错或提示缺失文件，可通过补全缺失包的方法解决

```
sudo apt update
sudo apt -f install
sudo dpkg -l dev-sidecar
```

```
ljl@raspberrypi:~ $ sudo dpkg -l dev-sidecar
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name           Version      Architecture Description
+++-==============-============-============-=================================
ii  dev-sidecar    2.0.0        arm64
```

- 连接 docker.com 测试

```
ljl@raspberrypi:~ $ ping docker.com
PING docker.com (23.185.0.4) 56(84) bytes of data.
64 bytes from 23.185.0.4: icmp_seq=1 ttl=42 time=218 ms
64 bytes from 23.185.0.4: icmp_seq=2 ttl=42 time=219 ms
64 bytes from 23.185.0.4: icmp_seq=3 ttl=42 time=222 ms
```

部署方案

- 使用官方脚本安装，终端执行指令

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
```

```
  Version:          v2.1.5
  GitCommit:        fcd43222d6b07379a4be9786bda52438f0dd16a1
 runc:
  Version:          1.3.3
  GitCommit:        v1.3.3-0-gd842d771
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0

================================================================================

To run Docker as a non-privileged user, consider setting up the
Docker daemon in rootless mode for your user:

    dockerd-rootless-setuptool.sh install

Visit https://docs.docker.com/go/rootless/ to learn about rootless mode.


To run the Docker daemon as a fully privileged service, but granting non-root
users access, refer to https://docs.docker.com/go/daemon-access/

WARNING: Access to the remote API on a privileged Docker daemon is equivalent
         to root access on the host. Refer to the 'Docker daemon attack surface'
         documentation for details: https://docs.docker.com/go/attack-surface/

================================================================================
```

- 安装完成后，查询 Docker 版本号

```
docker -v
```

```
ljl@raspberrypi:~ $ docker -v
Docker version 29.0.4, build 3247a5a
```

详见：Install Docker Engine on Debian | dockerdocs .

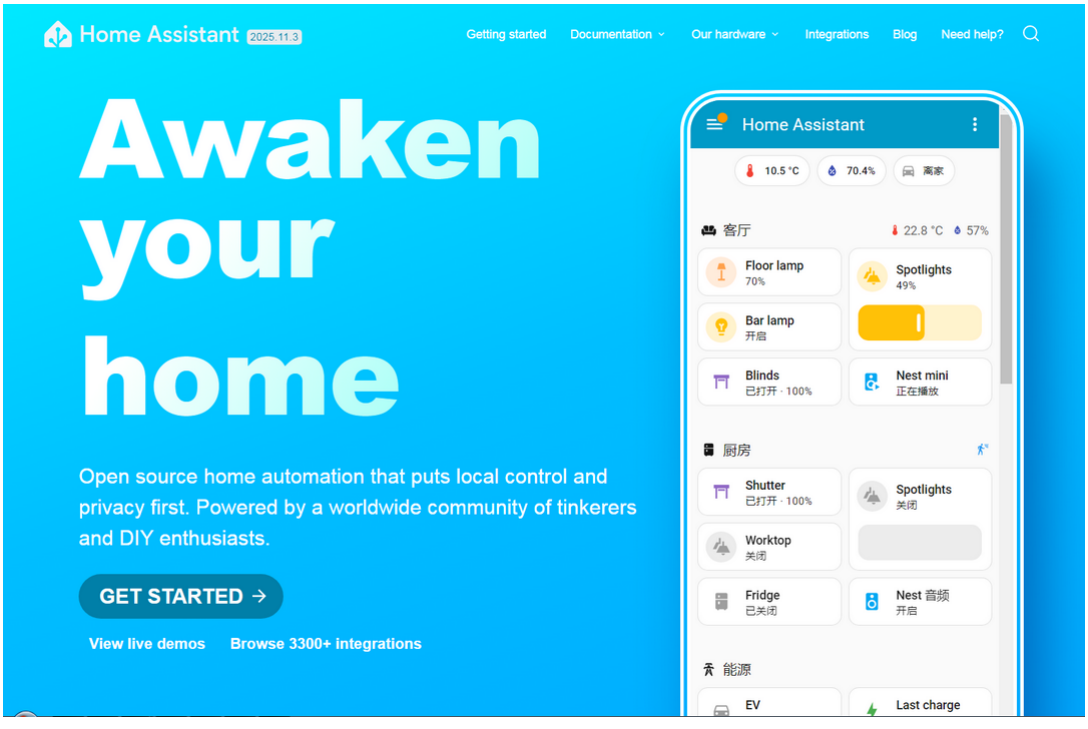
- 若报错，可更换国内软件源

```
# 添加 Docker 官方 GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL http://mirrors.aliyun.com/docker-ce/linux/debian/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# 添加仓库到 Apt 源:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] http://mirrors.aliyun.com/docker-ce/linux/debian \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker_aliyun.list > /dev/null
sudo apt-get update
```

详见：树莓派安装Docker .

Home Assistant

这里介绍树莓派 CM0 实现 Docker 部署 Home Assistant 的主要流程，包括更换镜像源、镜像选择和拉取、容器启动、个性化配置等。



加速

- 终端执行如下代码，添加镜像源，加速拉取镜像

```
sudo tee /etc/docker/daemon.json <<-'EOF'
{
```

```
    "registry-mirrors": [
        "https://docker.m.daocloud.io",
        "https://docker.imgdb.de",
        "https://docker-0.unsee.tech",
        "https://docker.hlmirror.com",
        "https://docker.1ms.run",
        "https://func.ink",
        "https://lispy.org",
        "https://docker.xiaogenban1993.com"
    ]
}
EOF
```

详见：Docker换源加速(更换镜像源)详细教程 .

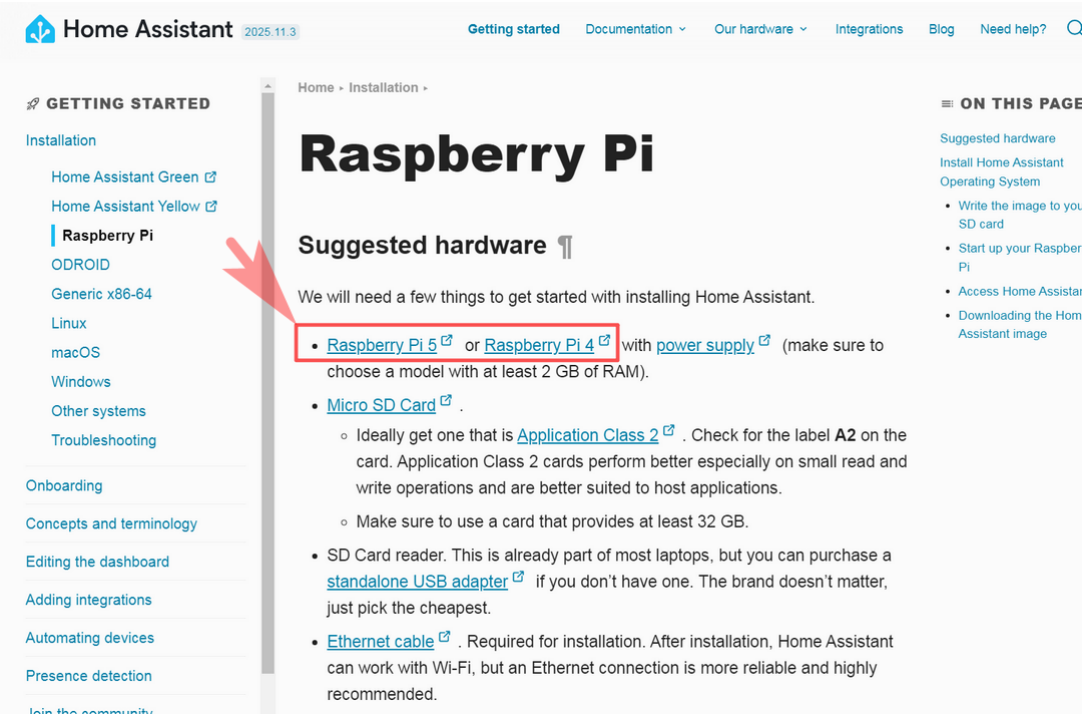
> 拉取 HA 镜像之前需要检查剩余存储空间，确保充足（大于 2.4G）
>
> ```
> ljl@raspberrypi:~ $ df -h
> Filesystem      Size  Used Avail Use% Mounted on
> udev             72M     0   72M   0% /dev
> tmpfs            84M  8.5M   75M  11% /run
> /dev/mmcblk0p2  6.7G  4.0G  2.4G  63% /
> tmpfs           209M     0  209M   0% /dev/shm
> tmpfs           5.0M  8.0K  5.0M   1% /run/lock
> tmpfs           1.0M     0  1.0M   0% /run/credentials/systemd-journald.service
> tmpfs           209M     0  209M   0% /tmp
> /dev/mmcblk0p1  510M   74M  437M  15% /boot/firmware
> tmpfs           1.0M     0  1.0M   0% /run/credentials/getty@tty1.service
> tmpfs           1.0M     0  1.0M   0% /run/credentials/serial-getty@ttyS0.service
> tmpfs            42M   20K   42M   1% /run/user/1000
> ```

- 重启 Docker，执行

```
sudo systemctl daemon-reload
sudo systemctl restart docker
```

镜像获取

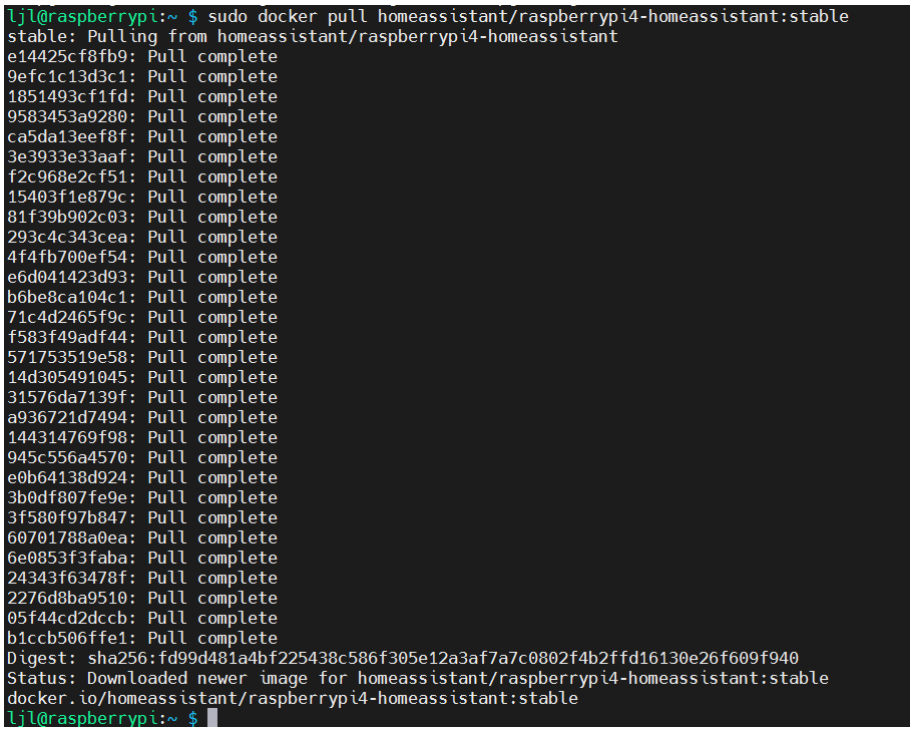
考虑到 Home Assistant OS 支持树莓派 5 和树莓派 4 等大运行内存需求，这里使用 Docker 安装轻量化的 `raspberrypi4-homeassistant` 稳定版。



详见：[Home Assistant installation | Raspberry Pi](#) .

- 拉取 HA，终端执行

```
sudo docker pull homeassistant/raspberrypi4-homeassistant:stable
```



- 拉取完成后自动解压。

启动容器

HA 镜像拉取完成后，终端执行如下代码，启动容器

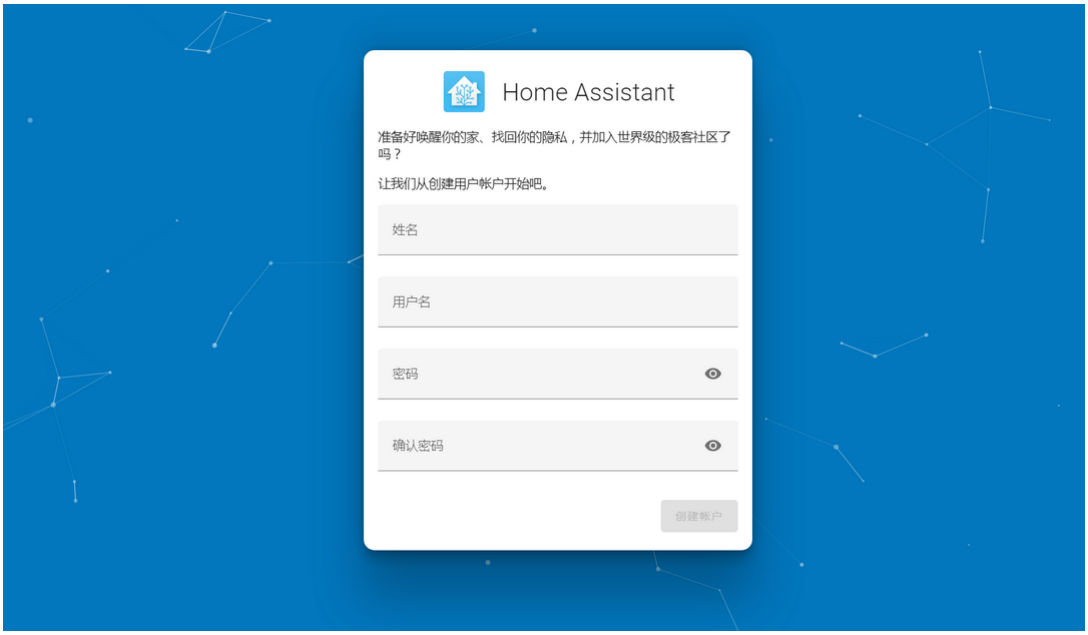
```
sudo docker run -d --name hass \
  --restart unless-stopped \
  --memory 350m --memory-swap 512m \
  -v /srv/homeassistant:/config \
  -v /etc/localtime:/etc/localtime:ro \
  --network host \
  homeassistant/raspberrypi4-homeassistant:stable
```

启动后进入初始化，需等待几分钟，之后执行

```
sudo docker logs -f hass
```

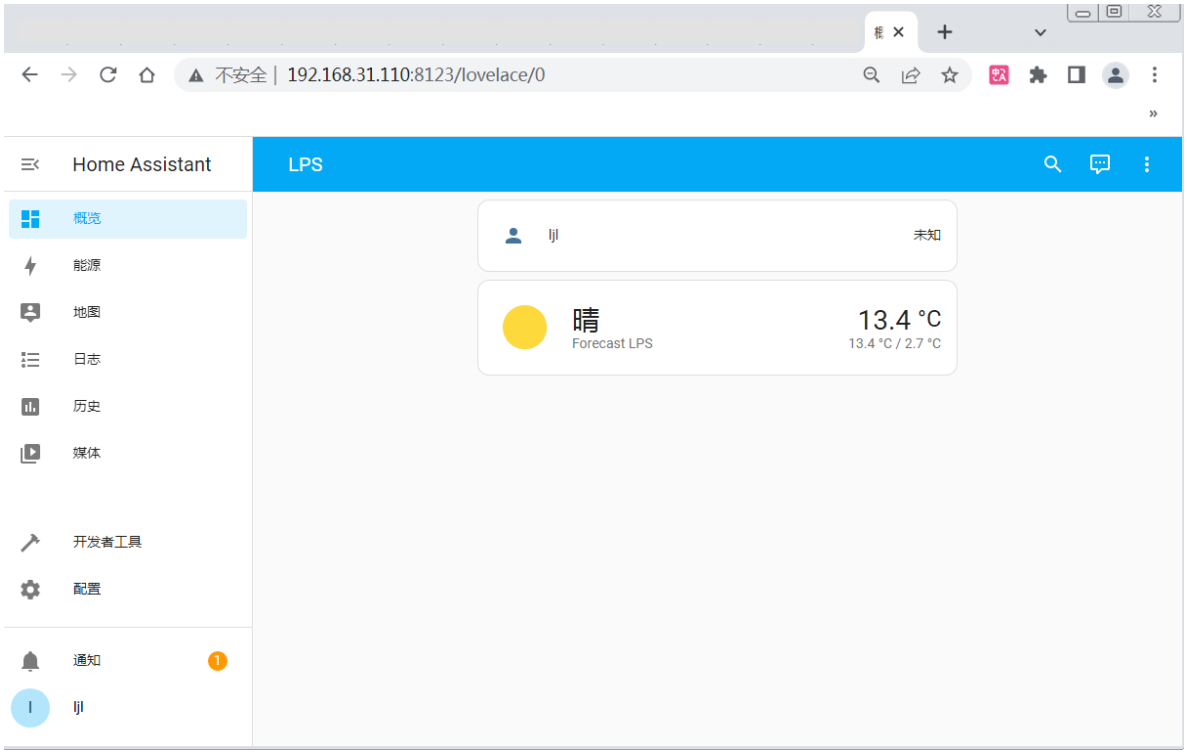
参数配置

- 浏览器输入网址 `http://<树莓派IP>:8123`，如 `192.168.1.102:8123`
- 进入 HA 主界面（首次打开需进行注册），输入用户名、密码等信息；

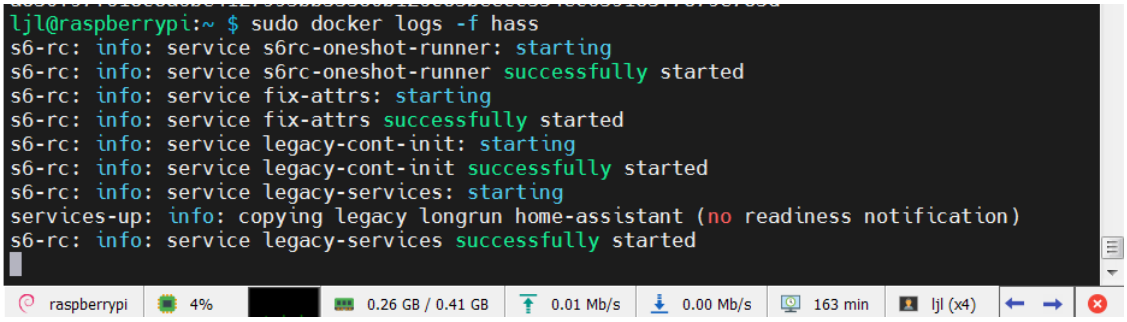


- 配置相关个性化参数，包括 HA 名称、地区、语言、时区、海拔、单位、货币等；

- 配置完成后进入 HA `概览` 标签页；



- 实测内存占用约 260 MB；



至此完成 Home Assistant 在树莓派 CM0 NANO 单板计算机的本地部署。

总结

本文介绍了树莓派 CM0 NANO 单板计算机通过 Docker 容器实现 Home Assistant 的本地部署，包括网络加速、Docker 安装、Home Assistant 镜像拉取、容器启动和参数配置等流程，为树莓派CM0等相关产品在工业物联网领域的快速开发和应用设计提供了参考。

发布链接：https://blog.csdn.net/qq_36654593/article/details/155321201

【工业树莓派 CM0 NANO 单板计算机】本地部署 EMQX

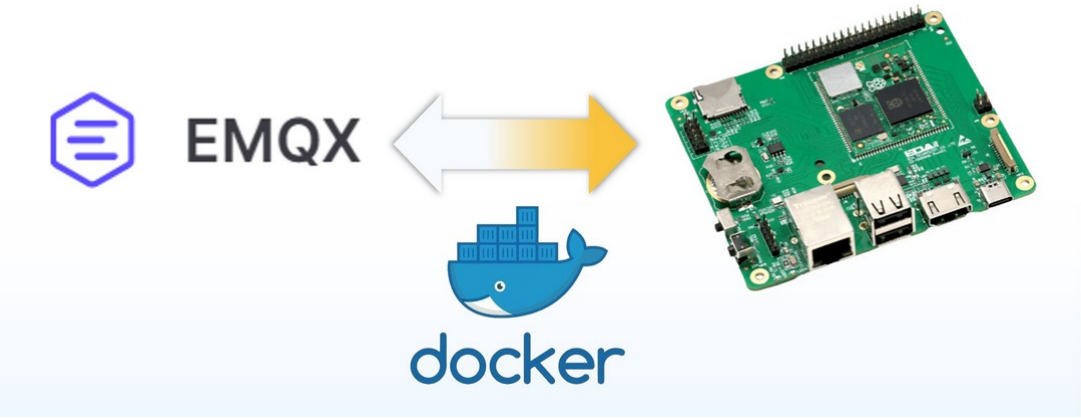
本文介绍了树莓派 CM0 NANO 单板计算机通过 Docker 容器实现 EMQX 的本地部署，包括 Docker 安装、EMQX 镜像拉取、容器启动、客户端配置、MQTT 测试等流程。

项目介绍

该项目通过 Docker 软件实现 EMQX 平台在树莓派 CM0 NANO 单板计算机的本地部署。

- Docker 安装：网络加速、安装脚本执行、软件源更换和版本检测等；

- EMQX 部署：更换镜像源、镜像获取、容器启动、创建用户以及参数配置等；
- 通信测试：MQTT 消息转发、芯片温度上报测试等流程。



应用

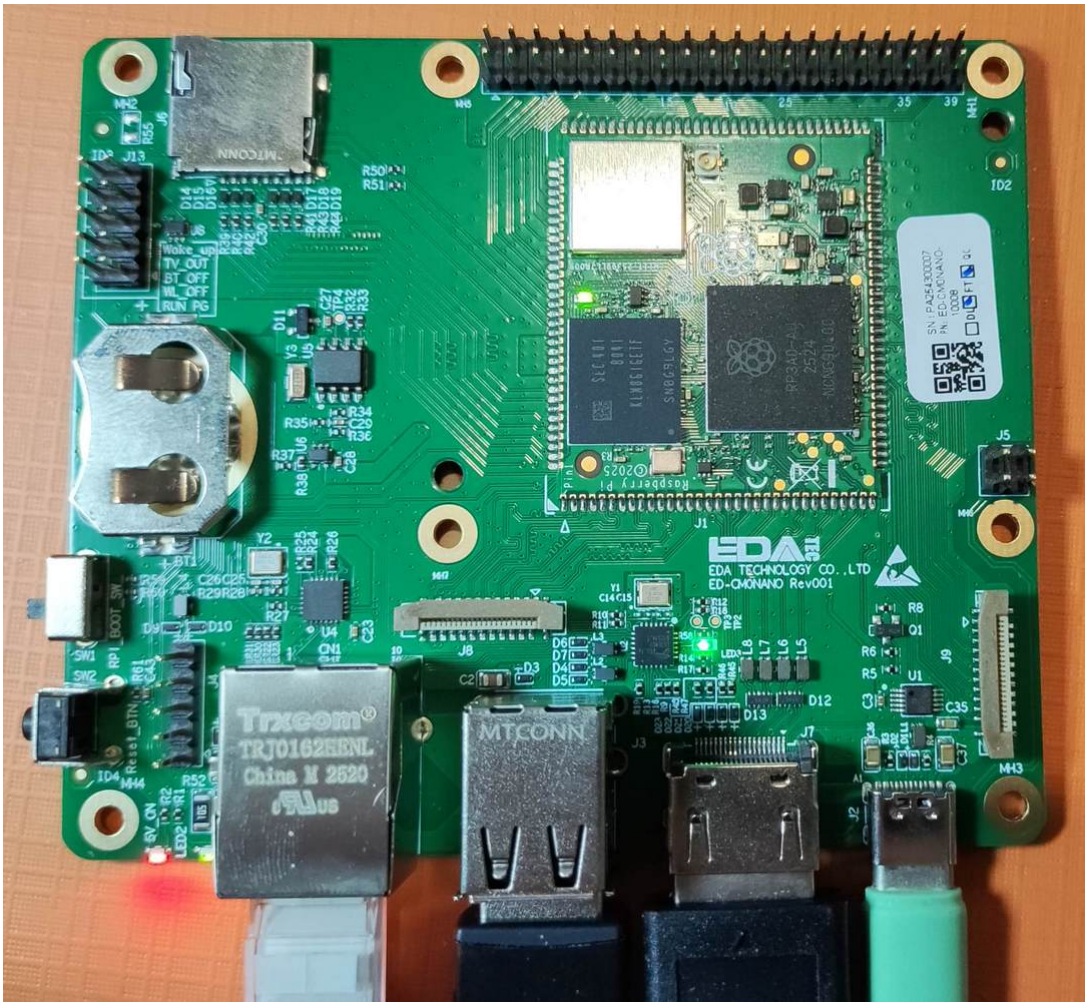
EMQX 作为高并发 MQTT 接入层，在工业场景主要承担 `设备-边缘-云端` 实时数据总线角色，应用场景包括且不限于

- 智能家居终端控制与数据采集；
- 工业自动化数据采集与远程控制；
- 边缘 IIOT 工业物联网；
- 工业环境监测，历史数据转发与存储；
- ……

配合规则引擎与流计算，实现设备互联、生产监控、能耗优化，逐渐成为智能制造与数字化转型的核心数据底座。

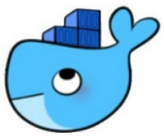
硬件连接

- 若采用 SSH 远程登录方式，需连接 5V 3A 电源和网线；
- 若采用本地连接方式，需增加 HDMI 数据线连接显示器、USB 键盘及鼠标。



Docker

这里介绍树莓派 CM0 NANO 实现 Docker 部署的主要流程，包括更换软件源、安装脚本获取与执行、版本检测等。



网络加速

使用 Dev-Sidecar 软件实现网络加速。

- 执行如下执行，获取安装包并安装软件

```
wget "https://bgithub.xyz/docmirror/dev-sidecar/releases/download/v2.0.0.2/DevSidecar-2.0.0.2-linux-arm64.deb"
sudo dpkg -i DevSidecar-2.0.0.2-linux-arm64.deb
```

- 若报错或提示缺失文件，可通过补全缺失包的方法解决

```
sudo apt update
sudo apt -f install
sudo dpkg -l dev-sidecar
```

- 执行 ping docker.com 测试网络连接；

部署方案

- 使用官方脚本安装，终端执行指令

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
```

- 安装完成后，查询 Docker 版本号

```
docker -v
```

详见：【工业树莓派 CM0 NANO 单板计算机】本地部署 Home Assistant 智能家居平台 .

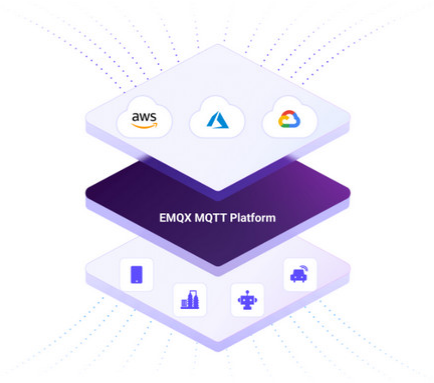
- 若报错，可更换国内软件源

```
# 添加 Docker 官方 GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL http://mirrors.aliyun.com/docker-ce/linux/debian/gpg -o
/etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# 添加仓库到 Apt 源:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-
by=/etc/apt/keyrings/docker.asc] http://mirrors.aliyun.com/docker-
ce/linux/debian \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker_aliyun.list > /dev/null
sudo apt-get update
```

详见：树莓派安装Docker .

EMQX

EMQX 是全球领先的开源分布式云原生 MQTT 物联网接入平台，支持单机千万级并发、毫秒级延迟，为万物互联提供高可靠、高可用的消息总线底座。



详见：EMQX：MQTT 与 AI 一体化平台 .

在前面完成 Docker 安装的基础上，这里进一步介绍了部署 EMQX 容器的主要流程，包括镜像源更换、镜像拉取、容器启动、个性化配置等。

部署方案

在添加软件镜像源的基础上，终端执行如下指令，拉取最新版 emqx 镜像

```
sudo docker pull emqx/emqx:latest
```

```
ljl@raspberrypi:~ $ sudo docker pull emqx/emqx:latest
latest: Pulling from emqx/emqx
e363695fcb93: Pull complete
8f668af5bd49: Pull complete
4ecfd1d06570: Pull complete
4f4fb700ef54: Pull complete
96af2f4f2f24: Pull complete
1747ed61d6eb: Pull complete
Digest: sha256:8426e9b5a1fd7717fa9aaf5bb0ec53e68b5318ecab8d494e4ef720aa7bbfbe7a
Status: Downloaded newer image for emqx/emqx:latest
docker.io/emqx/emqx:latest
ljl@raspberrypi:~ $
```

实测部署 EMQX 需要至少 400MB 存储空间。

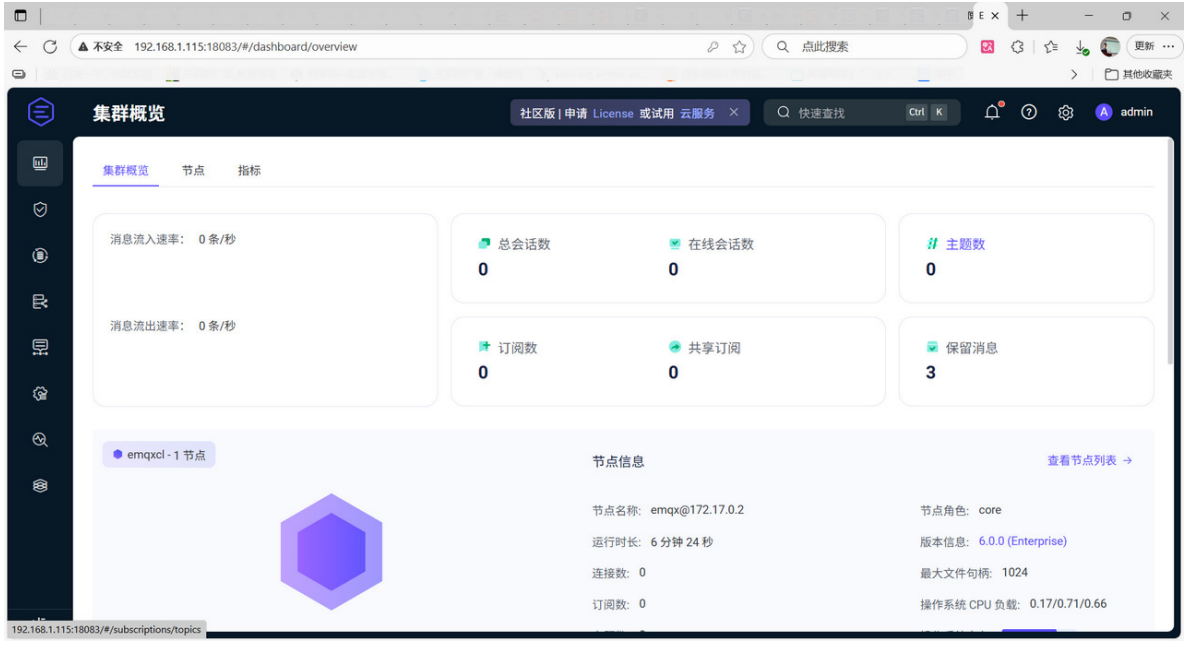
启动容器

终端执行如下指令，启动 EMQX 容器

```
docker run -d --name emqx -p 1883:1883 -p 8083:8083 -p 8084:8084 -p 8883:8883
-p 18083:18083 emqx/emqx:latest
```

```
ljl@raspberrypi:~ $ sudo docker run -d --name emqx -p 1883:1883 -p 8083:8083 -p 8084:8084 -p 8883:8883 -p 18083:18083
emqx/emqx:latest
b7768d9555d7beb3b3a7c86391e96d943e1d1daa941eb0539fcb24aed3f6dfd7
ljl@raspberrypi:~ $
```

- 浏览器输入网址 `http://<树莓派IP>:18083` ，如 `http://192.168.1.115:18083` ；
- 进入 EMQX 界面，初始账号 `admin` 密码 `public` ；
- 首次登录会提示更新密码并进入概览界面



- 运行过程占用内存约 310MB

```
ljl@raspberrypi:~ $ sudo docker run -d --name emqx -p 1883:1883 -p 8083:8083 -p 8084:8084 -p 8883:8883 -p 18083:18083
emqx/emqx:latest
b7768d9555d7beb3b3a7c86391e96d943e1d1daa941eb0539fcb24aed3f6dfd7
ljl@raspberrypi:~ $ docker logs emqx 2>&1 | grep "EMQX is running"
ljl@raspberrypi:~ $
raspberrypi 4% 0.31 GB / 0.41 GB 0.01 Mb/s 0.21 Mb/s 55 min ljl (x4) /: 71% /tmp: 0% /boot/firmware: 1
```

- 若要停止/启动/重启 EMQX，则执行指令

```
sudo docker stop emqx       # STOP
sudo docker start emqx      # RUN
sudo docker restart emqx    # Restart
```

创建用户

创建客户端

- 依次打开 访问控制 - 客户端认证 - 创建 - Password-Based - 内置数据库 - （默认配置）- 创建；

创建用户

- 用户管理 - 新建用户 - 自定义用户名和密码 - 保存.

至此，完成 EMQX 在树莓派 CM0 NANO 单板计算机的本地部署。

通信测试

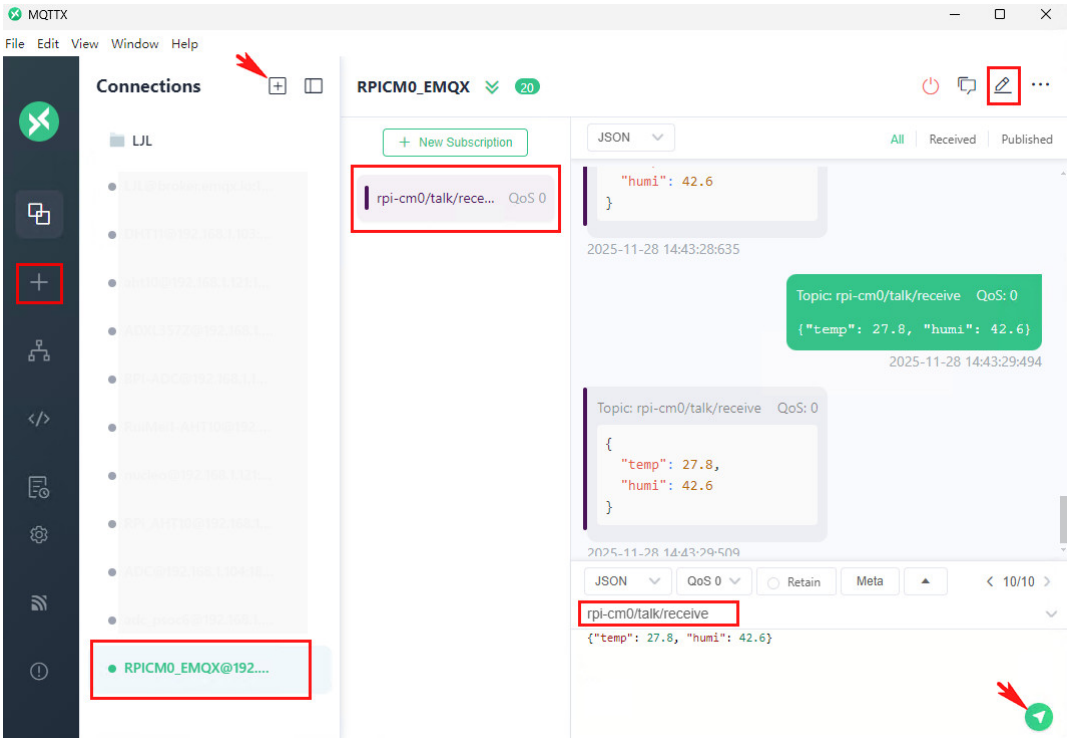
这里介绍了基于 EMQX 服务器和 MQTT 协议的通信测试，包括 MQTT 消息转发、芯片温度上报等。

MQTT 转发

通过 MQTTX 软件向 EMQX 服务器发送消息，并订阅相同主题，实现消息接收。

- 打开 MQTTX 软件，新建连接；
- 输入 EMQX 服务器 IP 地址，用户名和密码等信息，连接服务器；

- 添加订阅主题和发送主题，如 `rpi/cm0/demo`；
- 在发送框内输入任意 JSON 消息并点击发送，如 `{"temp": 27.8, "humi": 42.6}`，立即接收到相同消息；

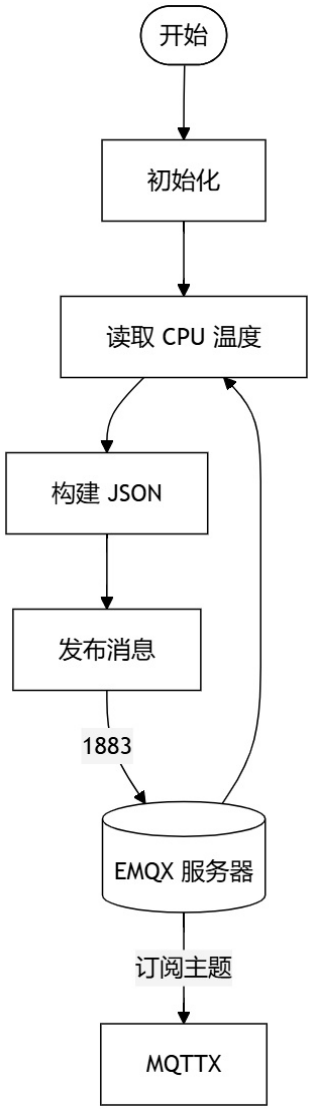


- 在 EMQX 服务器后台可以看到消息的流入流出、在线会话数量、主题等信息

芯片温度上报

在前面本地测试通过的基础上，将芯片温度上传至 EMQX 服务器并实现消息转发。

流程图



代码

终端执行 `touch mqtt_temp.py` 指令新建 python 文件，并 `nano mqtt_temp.py` 添加如下代码

```python
#!/usr/bin/env python3
import paho.mqtt.client as mqtt
import time, json, subprocess

broker   = "localhost"    # ip
port     = 1883           # port
topic    = "rpi/cm0/temp" # topic
username = "xxx"          # EMQX user name
password = "xxx"          # password

client = mqtt.Client(callback_api_version=mqtt.CallbackAPIVersion.VERSION2)
client.username_pw_set(username, password)   # user info
client.connect(broker, port, keepalive=60)

def get_cpu_temp():
    raw = subprocess.check_output(["vcgencmd", "measure_temp"]).decode()
    return float(raw.split("=")[1].split("'")[0])

while True:
    temp = get_cpu_temp()
```

```
    payload = json.dumps({"temp": temp, "unit": "℃", "ts": int(time.time())})
    client.publish(topic, payload, qos=0)
    print(payload)
    time.sleep(2)

```

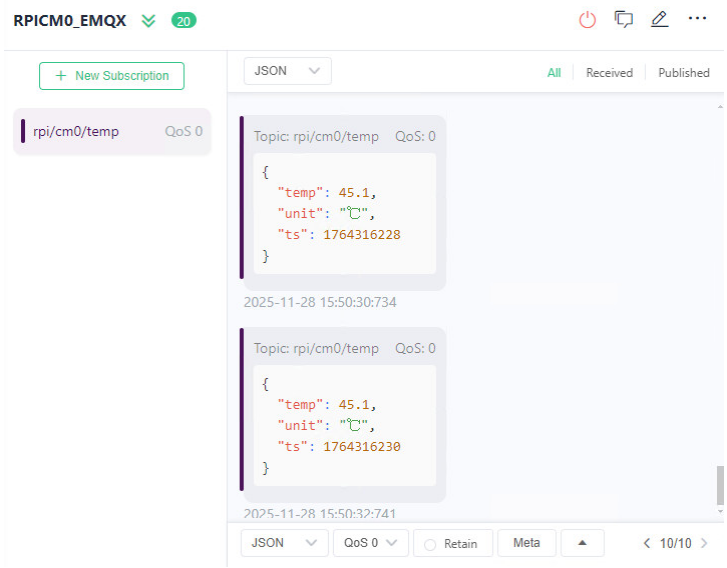
- 保存代码；

终端执行指令 `python mqtt_temp.py` 运行程序；

- 终端打印上报消息

```
ljl@raspberrypi:~/IoT $ python mqtt_temp.py
{"temp": 46.2, "unit": "\u2103", "ts": 1764316192}
{"temp": 45.6, "unit": "\u2103", "ts": 1764316194}
{"temp": 45.6, "unit": "\u2103", "ts": 1764316196}
{"temp": 45.1, "unit": "\u2103", "ts": 1764316198}
{"temp": 45.1, "unit": "\u2103", "ts": 1764316200}
{"temp": 45.1, "unit": "\u2103", "ts": 1764316202}
{"temp": 45.1, "unit": "\u2103", "ts": 1764316204}
{"temp": 45.1, "unit": "\u2103", "ts": 1764316206}
{"temp": 45.1, "unit": "\u2103", "ts": 1764316208}
```

- 打开 MQTTX 软件，接收到 EMQX 服务器转发的 CM0 芯片温度信息



总结

本文介绍了树莓派 CM0 NANO 单板计算机通过 Docker 容器实现 EMQX 的本地部署，包括 Docker 安装、EMQX 镜像拉取、容器启动、客户端配置、MQTT 测试等流程，为树莓派 CM0 等相关产品在工业物联网领域的快速开发和应用设计提供了参考。

发布链接：https://blog.csdn.net/qq_36654593/article/details/155360370

【工业树莓派 CM0 NANO 单板计算机】HACS 安装和米家集成

本文介绍了树莓派 CM0 NANO 单板计算机为本地部署的 Home Assistant 添加 HACS 集成并添加物联网终端设备，包括Home Assistant 部署、HACS 添加、Xiaomi Home 插件及设备导入、参数配置等流程。

项目介绍

该项目通过部署于 Docker 容器的 Home Assistant 智能家居平台，实现 HACS 商店安装，并在此基础上快速添加 Xiaomi Home 插件，实现物联网终端设备的物联网接入。

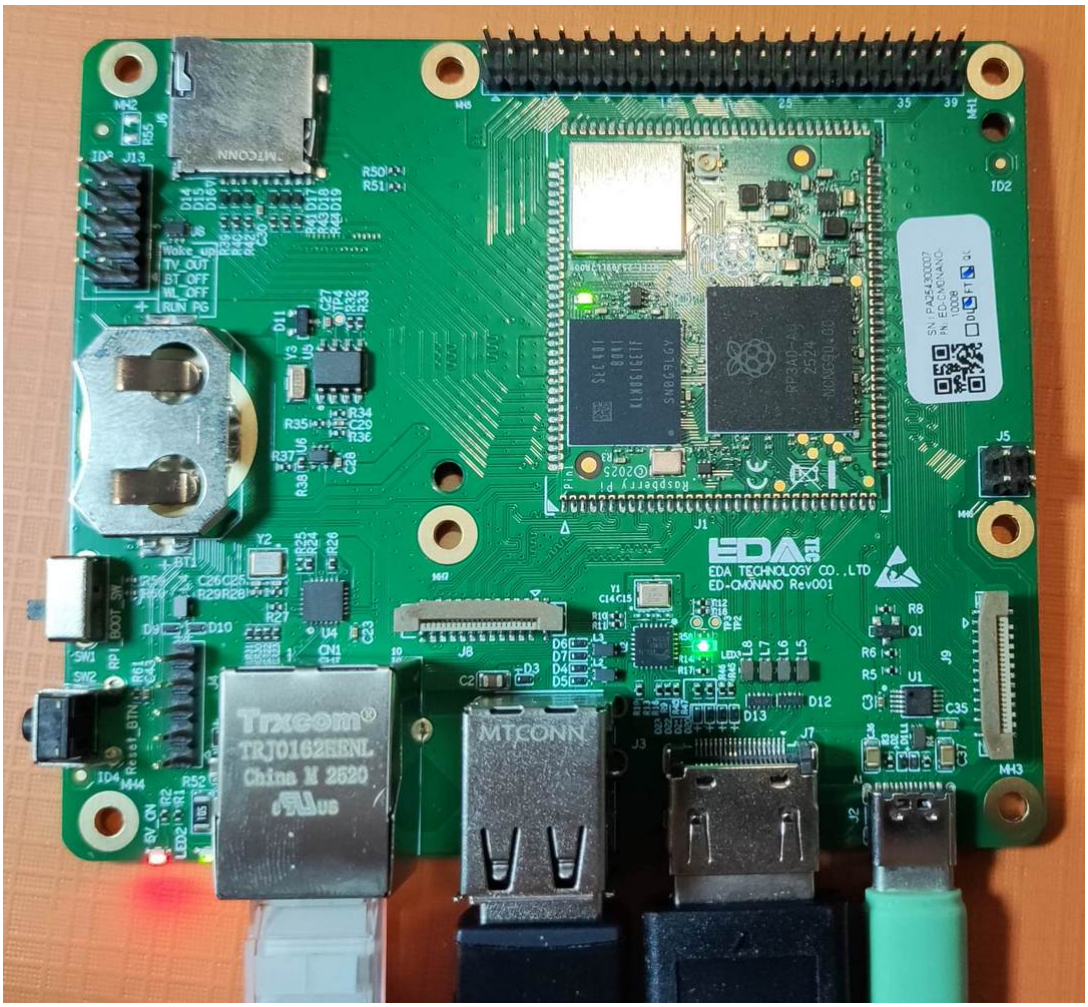
- Home Assistant 部署：版本选取、更换镜像源、镜像获取、容器启动、参数配置等。
- HACS 安装：资源下载、集成添加等；
- Xiaomi Home 插件安装：插件获取、账号登录、家庭设备接入等。

准备工作

包括硬件连接和 Docker 安装。

硬件连接

- 若采用 SSH 远程登录方式，需连接 5V 3A 电源和网线；
- 若采用本地连接方式，需增加 HDMI 数据线连接显示器、USB 键盘及鼠标。



操作系统

安装树莓派官方操作系统。

> 这里使用 Raspberry Pi OS(Lite) 64-bit-trixie (Debian 13) .

详见：[【工业树莓派CM0 NANO 单板计算机】介绍、镜像烧录、系统测试](#) .

Docker 安装

- 使用官方脚本安装，终端执行指令

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
```

详见：【工业树莓派 CM0 NANO 单板计算机】本地部署 Home Assistant 智能家居平台 .

Home Assistant

这里介绍了适合树莓派 CM0 硬件资源的 Home Assistant 的部署流程，包括镜像拉取、创建容器等。

更换镜像源

终端执行如下代码，添加镜像源，加速拉取镜像

```
sudo tee /etc/docker/daemon.json <<-'EOF'
{
    "registry-mirrors": [
        "https://docker.m.daocloud.io",
        "https://docker.imgdb.de",
        "https://docker-0.unsee.tech",
        "https://docker.hlmirror.com",
        "https://docker.1ms.run",
        "https://func.ink",
        "https://lispy.org",
        "https://docker.xiaogenban1993.com"
    ]
}
EOF
```

重启 Docker 执行指令

```
sudo systemctl daemon-reload
sudo systemctl restart docker
```

详见：【工业树莓派 CM0 NANO 单板计算机】本地部署 Home Assistant 智能家居平台 .

拉取镜像

考虑到板载存储空间与后续安装 Xiaomi Home 所需 HA 版本，这里选择体积较小且版本较新的 HA 镜像 2024.11.3；

终端执行指令

```
sudo docker pull homeassistant/home-assistant:2024.11.3
sudo docker tag homeassistant/home-assistant:2024.11.3 \
                homeassistant/home-assistant:2024.11.3
```

等待镜像拉取并解压完毕。

创建容器

终端执行指令

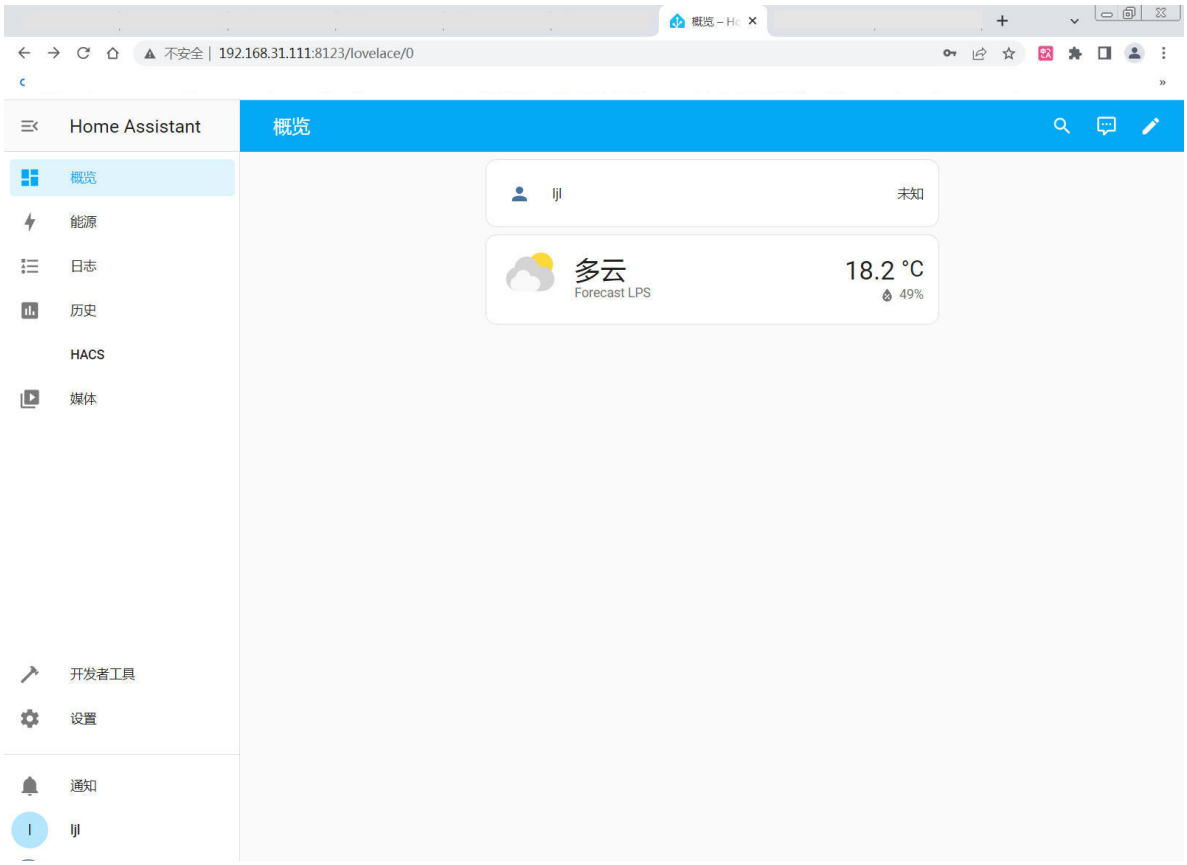
```
sudo docker run -d --name hass \
  --restart unless-stopped \
  --memory 250m --memory-swap 350m \
  -v /srv/homeassistant:/config \
  -v /etc/localtime:/etc/localtime:ro \
  --network host \
  --log-opt max-size=10m --log-opt max-file=2 \
  homeassistant/home-assistant:2024.11.3
```

验证指令

```
sudo docker logs -f hass | grep -m1 'Home Assistant Core'
```

输出日志。

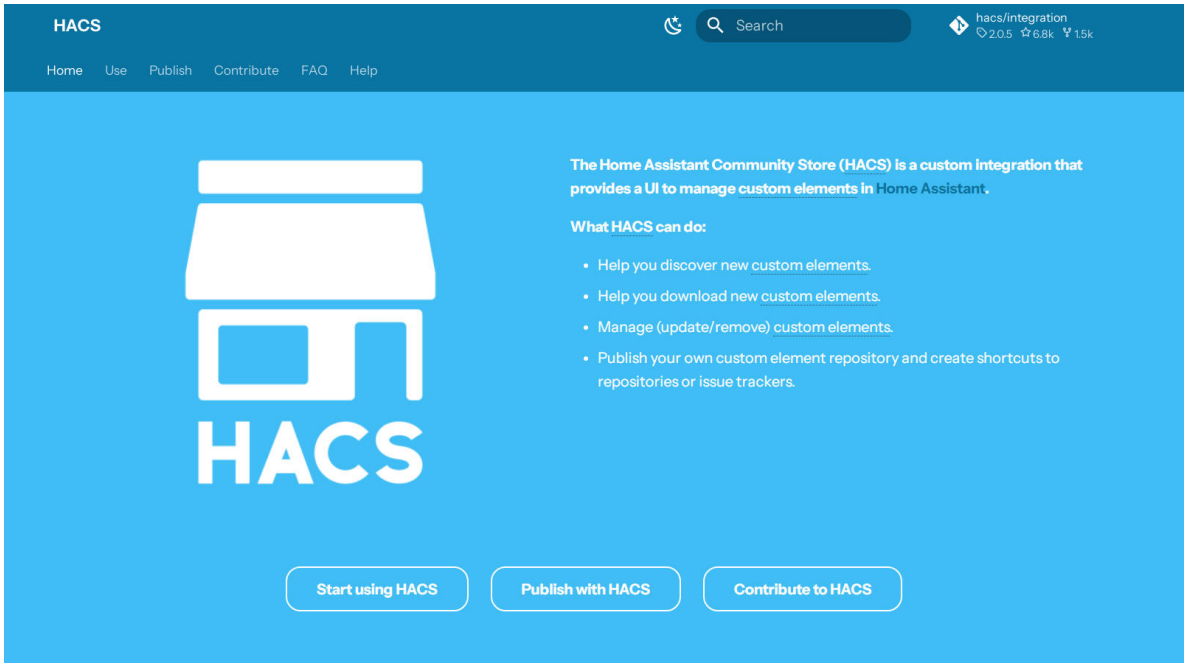
- 浏览器输入网址 `http://<树莓派IP>:8123`，如 `192.168.1.102:8123`
- 进入 HA 主界面（首次打开需进行注册），输入用户名、密码等信息；



详见：[【工业树莓派 CM0 NANO 单板计算机】本地部署 Home Assistant 智能家居平台](#) .

HACS

这里介绍了 Home Assistant 安装 HACS 集成的主要流程，包括资源下载、集成添加等。



下载

进入 HA 容器命令行模式

```
sudo docker exec -it hass bash
```

- 创建 HACS 容器目录，并下载 HACS 极速版

```
cd /config
mkdir -p custom_components
cd custom_components
wget -O- https://get.hacs.vip | bash -
```

- 执行 `exit` 指令退出；
- 重启 HA

```
sudo docker restart hass
```

添加集成

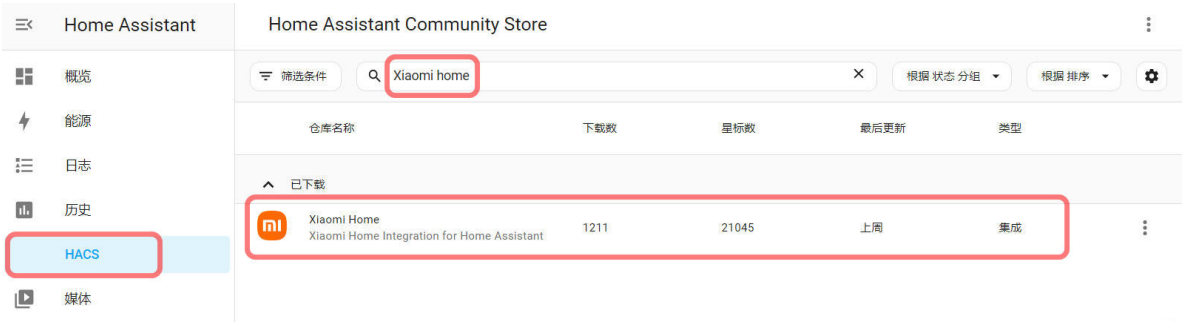
重启后，依次进入

- 设置 → 设备与服务 → 添加集成 → 搜索 `HACS` 并添加（使用公共 GitHub 授权即可）；
- 左侧标签页出现 HACS 标签；

Xiaomi Home

一键从 HACS 安装米家集成：

- HACS > 在搜索框中输入 **Xiaomi Home**；
- 点击 **Xiaomi Home**，进入集成详情页 > DOWNLOAD；



登录

- 设置 > 设备与服务 > 添加集成；
- 搜索 `Xiaomi Home` > 下一步；
- 点击此处进行登录 > 使用小米账号登录；

添加 MIoT 设备

- 登录成功后，会弹出会话框 `选择家庭与设备`；
- 选择需要添加的米家家庭，该家庭内的所有设备将导入 Home Assistant 。

总结

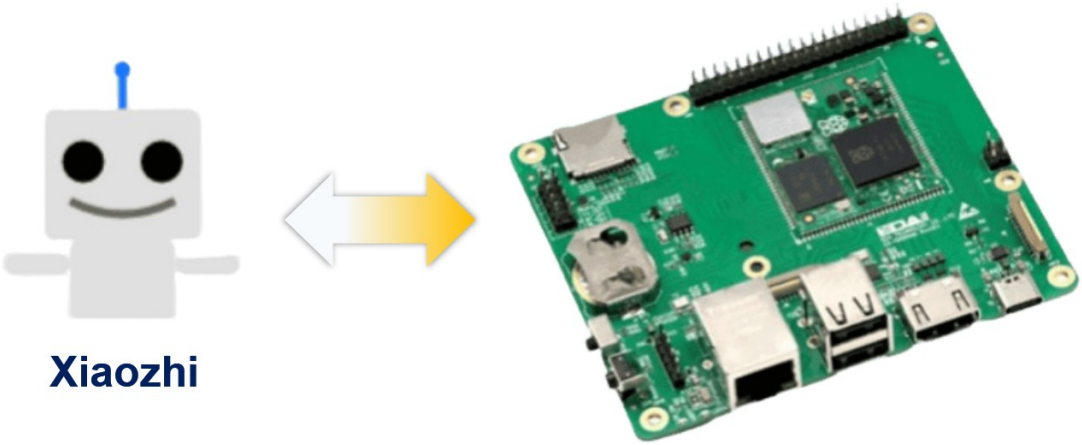
本文介绍了树莓派 CM0 实现 Home Assistant 智能家居平台的本地部署以及 Xiaomi Home 设备添加的项目流程，为相关产品在工业 IoT 领域的快速开发设计和应用提供了参考。

发布链接：https://blog.csdn.net/qq_36654593/article/details/155454522

【工业树莓派 CM0 NANO 单板计算机】小智语音聊天

本文介绍了树莓派 CM0 NANO 单板计算机通过本地部署 py-xiaozhi 小智实现 AI 智能体语音对话的项目设计。

项目介绍



该项目包括

- 环境搭建：拉取项目源码、安装所需软件包等；
- 程序运行：使用 CLI 模式运行程序；
- 设备激活：将验证码输入 xiaozhi.me 服务器账户，实现设备激活；
- 语音唤醒：下载模型、配置参数、使能语音唤醒，实现唤醒对话；
- 效果演示：动态演示对话效果。

详见：https://github.com/huangjunsen0406/py-xiaozhi

硬件连接

- 使用 WiFi 协议实现网络通信；
- 使用 USB 转 Audio 模块实现语音输入输出，即 AI 对话；
- 使用 Type-C 数据线实现设备供电；

环境搭建

开发板需安装树莓派官方最新操作系统，详见：https://www.raspberrypi.com/software/

拉取项目

```
git clone https://bgithub.xyz/huangjunsen0406/py-xiaozhi.git
cd py-xiaozhi
```

安装依赖

```
sudo apt install -y python3-pip
sudo pip3 install -r requirements.txt --break-system-packages
```

为了避免内存溢出，这里注释部分软件包

```
sed -i 's/^numpy==/#numpy==/' requirements.txt
sed -i 's/^cryptography==/#cryptography==/' requirements.txt
sed -i '/^PyQt5/d' requirements.txt
sed -i '/^pygame/d' requirements.txt
sed -i 's/sherpa-onnx==1.12.8/sherpa-onnx>=1.12.19/' requirements.txt
sed -i '/^rich==/d' requirements.txt
sed -i '/^python-dateutil==/d' requirements.txt
```

安装 Qt 相关软件包

```
sudo apt install -y python3-pyqt5 python3-pyqt5.qtwebengine
sudo apt install -y python3-pyqt5.qtquick qml-module-qtquick2 qml-module-qtquick-controls2
```

代码格式化

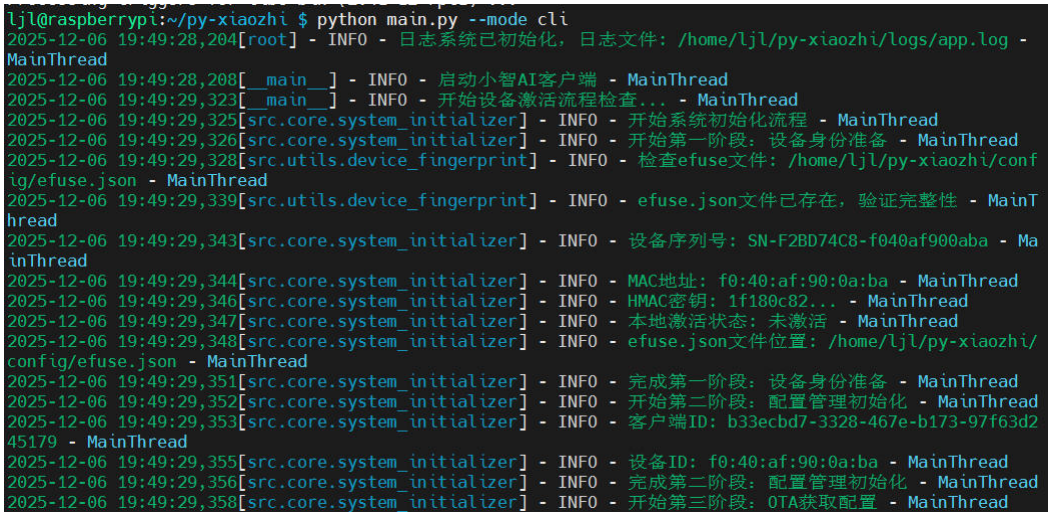
```
./format_code.sh
```

程序运行

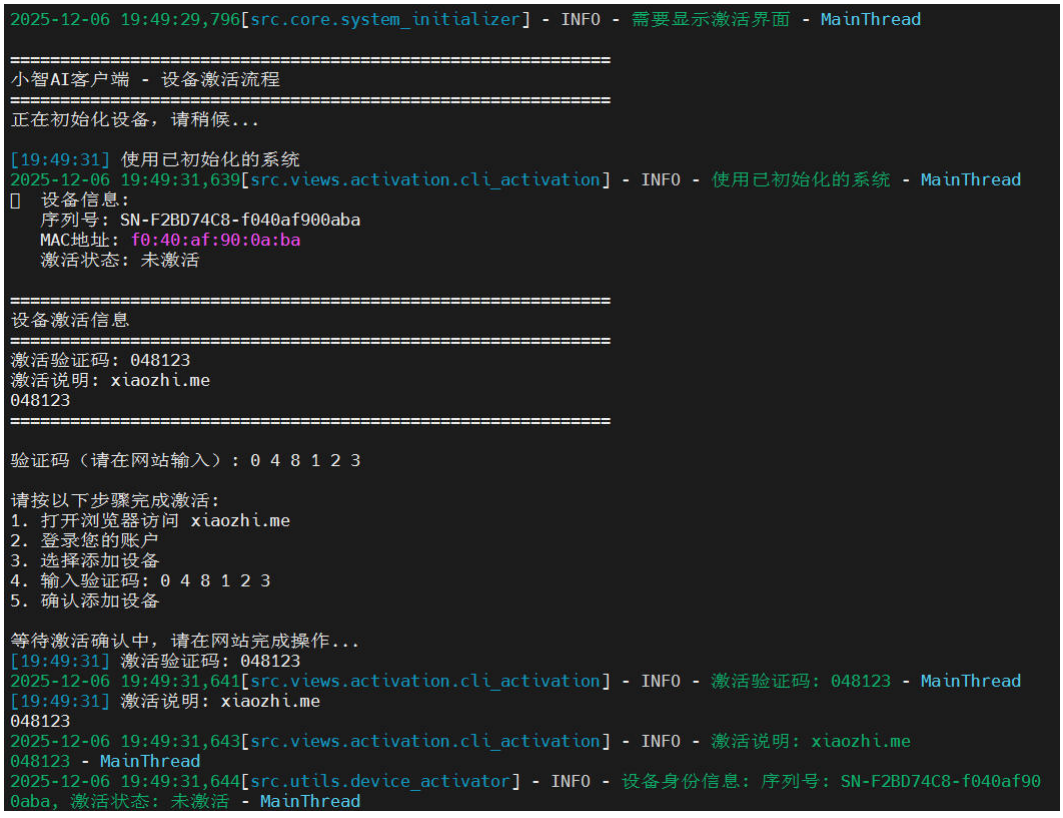
运行程序 - CLI模式

```
python main.py --mode cli
```

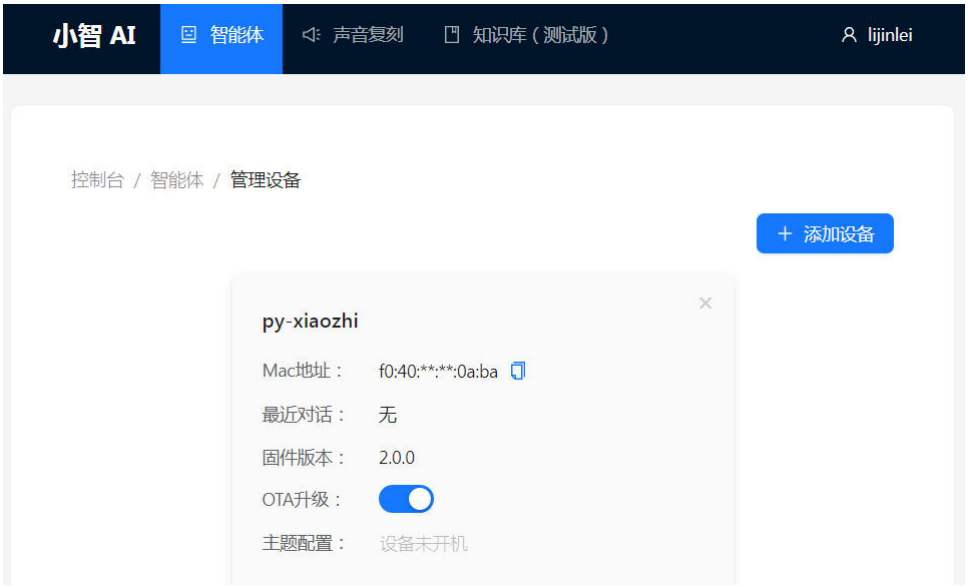
初始化



输入验证码



- 进入 xiaozhi.me 官网，登录账户，点击添加设备，输入验证码；



设备激活

验证码输入后，自动激活设备，终端打印激活日志

```
2025-12-06 19:51:06,281[src.utils.device_activator] - WARNING -
激活响应 (HTTP 200): - MainThread
2025-12-06 19:51:06,282[src.utils.device_activator] - WARNING - {
  "message": "Device activated",
  "device_id": 1305172
} - MainThread
2025-12-06 19:51:06,284[src.utils.device_activator] - INFO - 设备激活成功! - MainThread
[19:51:06]
设备激活成功!
2025-12-06 19:51:06,289[src.views.activation.cli_activation] - INFO -
设备激活成功!  - MainThread

============================================================
设备激活成功!
============================================================
设备已成功添加到您的账户
配置已自动更新
准备启动小智AI客户端...
============================================================
2025-12-06 19:51:06,291[__main__] - INFO - 激活流程完成，结果: True - MainThread
2025-12-06 19:51:06,293[src.application] - INFO - 启动Application，protocol=websocket - MainThread
2025-12-06 19:51:07,574[src.audio_codecs.audio_codec] - INFO - 首次运行，自动选择音频设备... - MainThre
ad
2025-12-06 19:51:07,580[src.audio_codecs.audio_codec] - INFO - 输入设备支持 128 声道，限制使用前 2 声道
 - MainThread
2025-12-06 19:51:07,582[src.audio_codecs.audio_codec] - INFO - 输出设备支持 128 声道，限制使用前 2 声道
 - MainThread
2025-12-06 19:51:07,583[src.audio_codecs.audio_codec] - INFO - 选择输入设备: default (48000Hz, 2ch) - M
ainThread
2025-12-06 19:51:07,585[src.audio_codecs.audio_codec] - INFO - 选择输出设备: default (48000Hz, 2ch) - M
ainThread
2025-12-06 19:51:07,609[src.audio_codecs.audio_codec] - INFO - Opus编解码器创建成功 - MainThread
2025-12-06 19:51:07,610[src.audio_codecs.audio_codec] - INFO - 输入声道下混: 2ch → 1ch - MainThread
2025-12-06 19:51:07,612[src.audio_codecs.audio_codec] - INFO - 输入重采样: 48000Hz → 16kHz - MainThread
2025-12-06 19:51:07,614[src.audio_codecs.audio_codec] - INFO - 输出重采样: 24000Hz → 48000Hz - MainThre
ad
2025-12-06 19:51:07,615[src.audio_codecs.audio_codec] - INFO - 输出声道上混: 1ch → 2ch - MainThread
2025-12-06 19:51:07,638[src.audio_codecs.audio_codec] - INFO - 音频流已启动 | 输入: 48000Hz 2ch | 输出:
 48000Hz 2ch - MainThread
2025-12-06 19:51:07,639[src.audio_codecs.audio_codec] - INFO - AudioCodec 初始化完成 - MainThread
```

自动进入对话界面

效果演示

终端 GUI 界面

动态演示

通过终端窗口可实现与小智的对话

执行快捷键 `Ctrl + C` 即可退出应用。

语音唤醒

py-xiaozhi 集成了基于 **Sherpa-ONNX** 的高精度语音唤醒功能，支持自定义唤醒词和实时检测。使用轻量级关键词检测模型，提供毫秒级响应速度。

模型下载

项目不包含模型文件，需要提前下载配置。

- 官方模型列表: https://csukuangfj.github.io/sherpa/onnx/kws/pretrained_models/index.html
- 推荐模型: `sherpa-onnx-kws-zipformer-wenetspeech-3.3M-2024-01-01`

下载模型包

使用 wget 工具获取模型文件并解压

```
cd ~/py-xiaozhi
wget https://bgithub.xyz/k2-fsa/sherpa-onnx/releases/download/kws-models/sherpa-onnx-kws-zipformer-wenetspeech-3.3M-2024-01-01.tar.bz2

tar xvf sherpa-onnx-kws-zipformer-wenetspeech-3.3M-2024-01-01.tar.bz2
```

配置方案

```
cd sherpa-onnx-kws-zipformer-wenetspeech-3.3M-2024-01-01

# 复制速度优先的 epoch-99 int8 三件套
cp encoder-epoch-99-avg-1-chunk-16-left-64.int8.onnx ../models/encoder.onnx
cp decoder-epoch-99-avg-1-chunk-16-left-64.onnx ../models/decoder.onnx
cp joiner-epoch-99-avg-1-chunk-16-left-64.int8.onnx ../models/joiner.onnx

# 复制配套文件
cp tokens.txt ../models/tokens.txt
```

配置完成后，`~/py-xiaozhi/models` 目录应包含

```
models/
├── encoder.onnx       # 编码器模型（重命名后）
├── decoder.onnx       # 解码器模型（重命名后）
├── joiner.onnx        # 连接器模型（重命名后）
├── tokens.txt         # 拼音Token映射表（228行版本）
├── keywords.txt       # 关键词配置文件（需创建）
└── keywords_raw.txt   # 原始关键词文件（可选）
```

启用语音唤醒

通过修改配置文件，实现 Xiaozhi 的语音唤醒。

打开 `~/py-xiaozhi/config/config.json` 检查并确认存在如下字段

```
{
  "WAKE_WORD_OPTIONS": {
    "USE_WAKE_WORD": true,
    "MODEL_PATH": "models",
    "NUM_THREADS": 4,
    "PROVIDER": "cpu",
    "MAX_ACTIVE_PATHS": 2,
    "KEYWORDS_SCORE": 1.8,
    "KEYWORDS_THRESHOLD": 0.2,
    "NUM_TRAILING_BLANKS": 1
  }
}
```

重启客户端

终端执行如下指令，重启客户端

```
python main.py --mode cli
```

日志出现：

```
KWS模型加载成功: encoder.onnx
```

即可语音唤醒 Xiaozhi 并对话。

详见：[语音唤醒功能 | PY-XIAOZHI](#) .

总结

本文介绍了树莓派 CM0 NANO 单板计算机通过本地部署 py-xiaozhi 小智实现 AI 智能体语音对话的项目设计，为相关产品在工业 AI 领域的快速开发设计和应用提供了参考。

发布链接：https://blog.csdn.net/qq_36654593/article/details/155647772

【工业树莓派 CM0 NANO 单板计算机】科学计算解决方案：常微分方程组的数值求解

本文介绍了工业树莓派 CM0 NANO 单板计算机结合 scipy 软件包和 Runge-Kutta 算法实现数值求解常微分方程组的项目设计。

项目介绍

工业树莓派 CM0 NANO 单板计算机结合 scipy 与 matplotlib 软件库实现常微分方程组的数值求解。

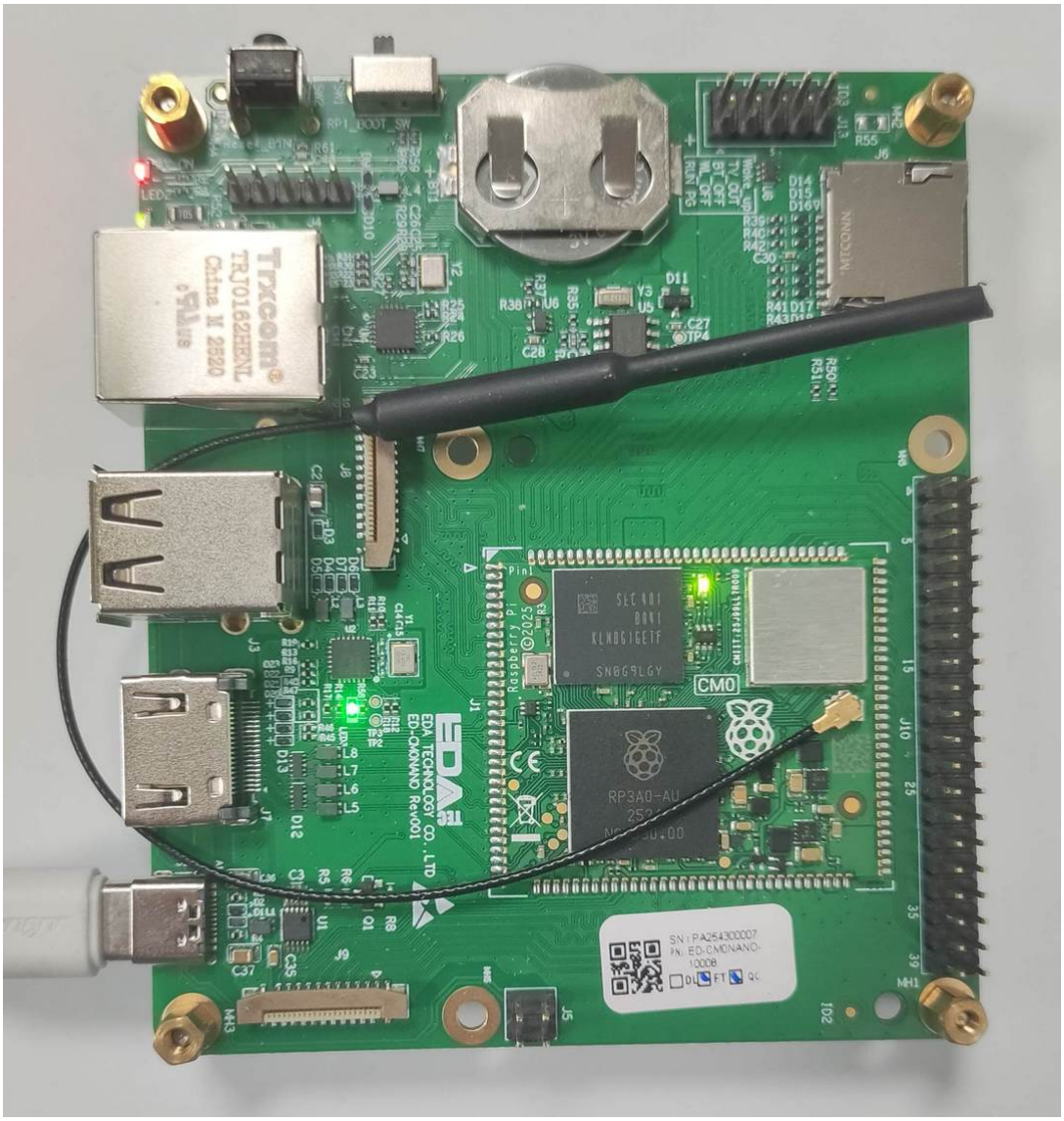
- 准备工作：硬件连接、系统安装、软件更新等；
- 环境搭建：科学计算所需软件库的安装、数值求解方程组测试等；
- 二能级系统：结合实际物理问题进行数值计算，包括流程图、代码、效果演示等；
- 布居振荡：考虑更为复杂的数值解应用场景，给出粒子 Rabi 振荡动力学及其参数依赖。

准备工作

包括硬件连接、系统安装、软件更新等。

硬件连接

这里使用 SSH 远程登录，仅需要 Type-C 供电和 WiFi 联网。



系统安装

开发板需安装树莓派官方最新操作系统，详见：https://www.raspberrypi.com/software/

软件更新

更新软件包

```
sudo apt update
sudo apt upgrade
```

环境搭建

安装 scipy 和 matplotlib 软件包，以便调用 RK45 算法和科学绘图；

```
sudo apt install python3-scipy
sudo apt install python3-matplotlib
```

Runge-Kutta 算法

为了获得更为精确的数值解，常用方案是采用四阶 Runge-Kutta 方法。

$$\begin{cases} y_{n+1} = y_n + h(\frac{1}{6}K_1 + \frac{2}{6}K_2 + \frac{2}{6}K_3 + \frac{1}{6}K_4) \\ K_1 = f(x_n, y_n) \\ K_2 = f(x_n + \frac{1}{2}h, y_n + h\frac{1}{2}K_1) \\ K_3 = f(x_n + \frac{1}{2}h, y_n + h\frac{1}{2}K_2) \\ K_4 = f(x_n + h, y_n + hK_3) \end{cases}$$

详见：Runge-Kutta method .

- 使用 Python 编程，通常采用 odeint 和 solve_ivp 函数实现；

> `solve_ivp` 是 Python 中 SciPy 库提供的一个函数，用于求解初值问题 (IVP, Initial Value Problem) 的常微分方程组 (ODEs)。
>
> 使用方法：
>
> ```
> scipy.integrate.solve_ivp(fun, t_span, y0, method='RK45', t_eval=None, vectorized=False, args=None, **options)
> ```

详见：solve_ivp | scipy .

数值计算

通过调用 solve_ivp 函数实现常微分方程组的数值求解。

Lorenz 吸引子

Lorenz attractor 是混沌理论中的经典模型，最早由美国气象学家爱德华·诺顿·洛伦兹提出。

该模型源于对大气对流方程的简化，旨在研究流体力学中的混沌现象。常用于描述 “蝴蝶效应” 。

Lorenz 吸引子具有三维、非线性和确定性特征，其相轨线呈现出复杂的双纽线形状，状态随时间以非重复模式演变。

其简化的方程形式为

$$\frac{dx}{dt} = \sigma(y - x)$$
$$\frac{dy}{dt} = x(\rho - z) - y$$
$$\frac{dz}{dt} = xy - \beta z$$

对于该常微分方程组，可使用 4 阶 Runge-Kutta 算法进行数值求解。

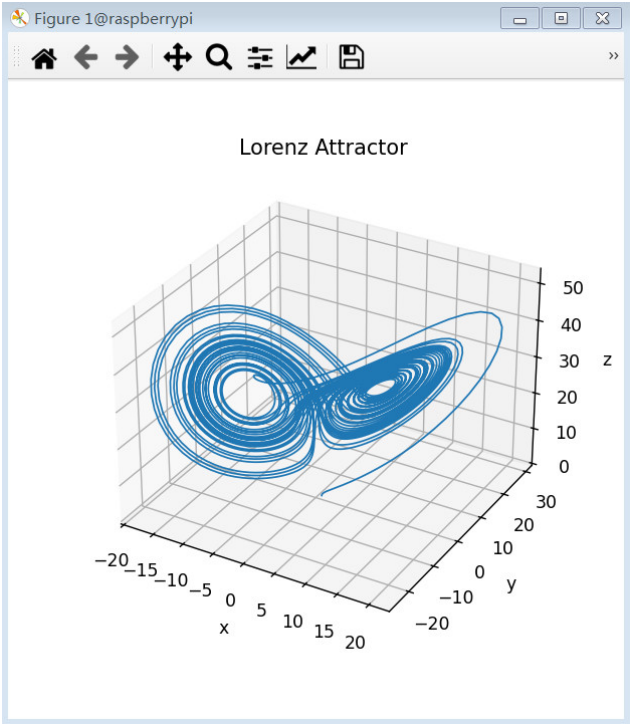
代码

终端执行 `touch ode_demo.py` 指令新建文件，并添加如下代码

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

# ----------- lorenz equations ------------
def lorenz(t, X, sigma, rho, beta):
    x, y, z = X
    return np.array([
        sigma * (y - x),          # dx/dt
        x * (rho - z) - y,        # dy/dt
        x * y - beta * z          # dz/dt
    ])

# ----------- parameters -------------
sigma = 0
beta  = 2
rho   = 0.0
x0    = [1.0, 0.0, 0]
t_span = (0, 0)                     # integral range
t_eval = np.arange(0, 0, 1)  # step 0.01

# -----------  numerical calculation -----------
sol = solve_ivp(lorenz, t_span, x0, args=(sigma, rho, beta),
                t_eval=t_eval, rtol=1e-4, atol=1e-4)

# ---------- plot -----------
fig = plt.figure(figsize=(6, 5))
ax = fig.add_subplot(111, projection='3d')
ax.plot(sol.y[0], sol.y[1], sol.y[2], lw=1.0)
plt.show()
```

效果

终端运行 `python lorenz_attractor.py` 弹窗显示数值结果



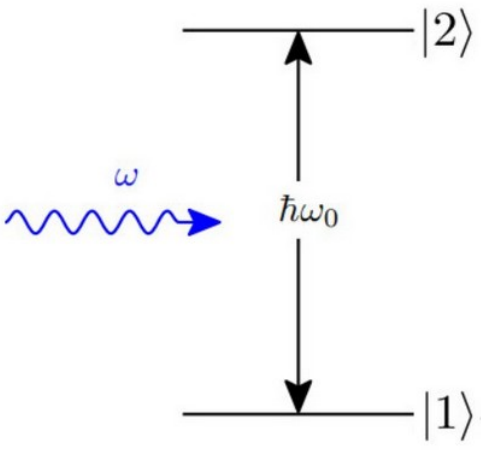
使用树莓派 CM0 完全可以胜任常微分方程的求解计算，并获得精确可靠的数值结果。

二能级系统

考虑更为复杂的实际科学计算。这里以量子光学中常见的二能级系统 Liouville 方程组数值求解为例，结合 Python 的 ODE 库函数实现。

Hamiltonian

考虑半经典条件下的光与二能级原子相互作用体系，示意图如下



该系统的 Hamiltonian 可表示为

$$\tilde{H}_I = -\hbar \begin{pmatrix} 0 & \Omega_s^* \\ \Omega_s & \Delta_s \end{pmatrix}$$

主方程

结合 Maxwell-Liouville 方程（亦称光学 Bloch 方程、 Liouville 方程或密度矩阵主方程），

$$\partial_t \rho = -i \left[\tilde{H}_I, \rho \right] / \hbar + \mathcal{D}(\rho)$$

给出 Hamiltonian 矩阵元对角元的运动方程

$$\dot{\rho}_{21} = -\left(\gamma_{21} - i\Delta_s \right) \rho_{21} + i\Omega_s$$

该方程组可采用数值或解析的方法求解。

详见：[电磁诱导透明机制下基于四波混频过程的光学参量放大动力学研究](#) .

代码

终端执行指令 `touch sci_two-level_system.py` 新建程序文件，添加如下代码

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

# ---------- parameters ----------
Delta   = np.arange
Omega   = 0
gamma21 = 0
t_span  = (0, 1)                    # integral range
R0      = np.array                   # initialize ρ=[ρ1, ρ2]
Nd      = len(Delta)                 # number of delta
# ---------- ODE ----------
def rk(t, Delta):
    r1, r2 = R
    dr1 = -(gamma21*r1)
    dr2 = -(1 - Delta*r1)
    return np.array([dr1, dr2])

# ---------- scan delta ----------
r21R = np.zeros(Nd)   # real χ'
r21I = np.zeros(Nd)   # image χ"

print("Start calculation...")
for m, d in enumerate(Delta):
    sol = solve_ivp(rk, t_span, R0,
                    method='RK45', atol=1e-6)
    # get terminal value
    r21R[m] = sol.y[1, -1]
    r21I[m] = sol.y[1, 0]
    # show progress
    if (m+1) % 10 == 0:
        print(f"Process: {m+1}/{Nd}")
print("Calculation complete!")

# ---------- Plot ----------
print("Start Drawing...")
plt.plot(Delta, r22R, label=r"$\chi'$", linewidth=4)
plt.legend()
plt.show()
```

```
40 print("Program Terminate.")
```

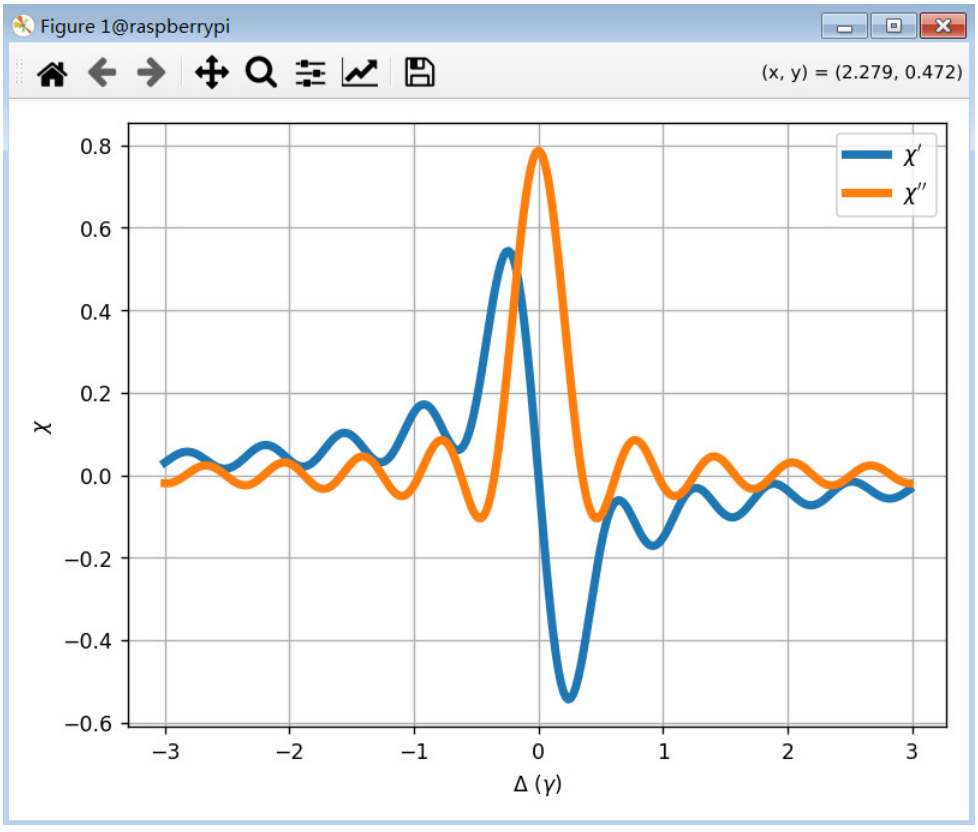
保存代码。

效果

终端运行 `python sci_two-level_system.py`，输出计算进度

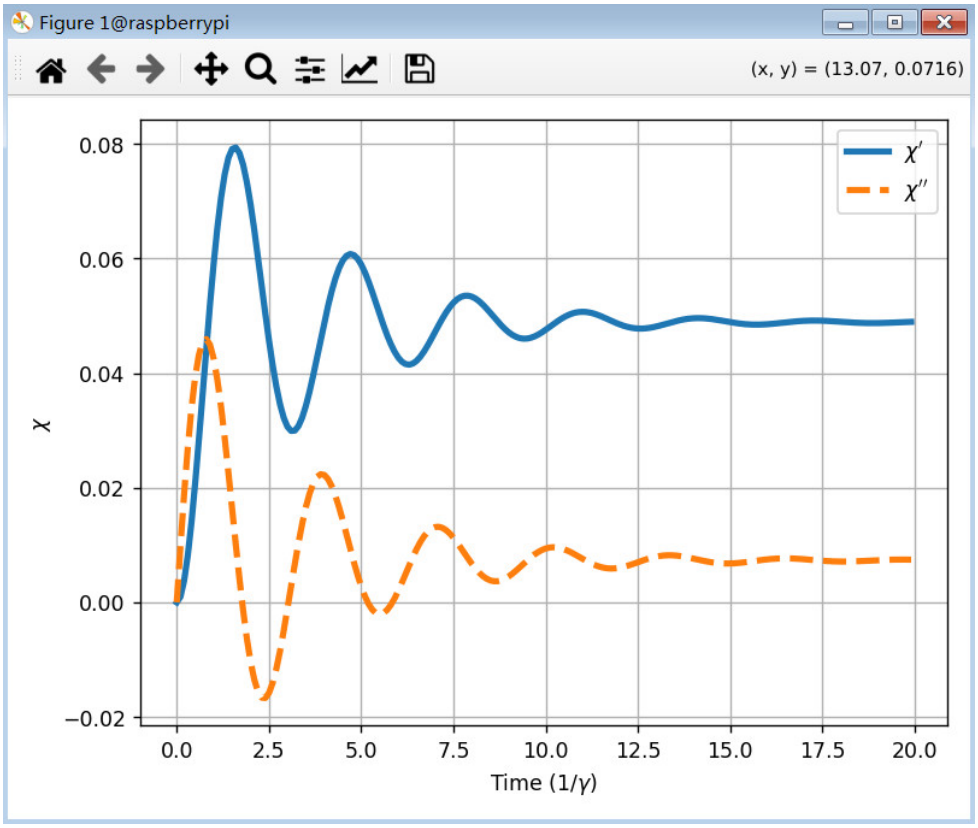
```
ljl@raspberrypi:~/SCI $ python3 twolevel_chi_delta.py
Start calculation...
Process: 10/120 (Delta = -2.6)
Process: 20/120 (Delta = -2.1)
Process: 30/120 (Delta = -1.6)
Process: 40/120 (Delta = -1.1)
Process: 50/120 (Delta = -0.6)
Process: 60/120 (Delta = -0.1)
Process: 70/120 (Delta = 0.4)
Process: 80/120 (Delta = 0.9)
Process: 90/120 (Delta = 1.4)
Process: 100/120 (Delta = 1.9)
Process: 110/120 (Delta = 2.4)
Process: 120/120 (Delta = 2.9)
Calculation complete!
Start Drawing...
Program Terminate.
```

弹窗显示稳态结果



表征介质的吸收（极化率虚部）和色散（极化率实部）特性。

相应的矩阵对角元随时间的演化曲线



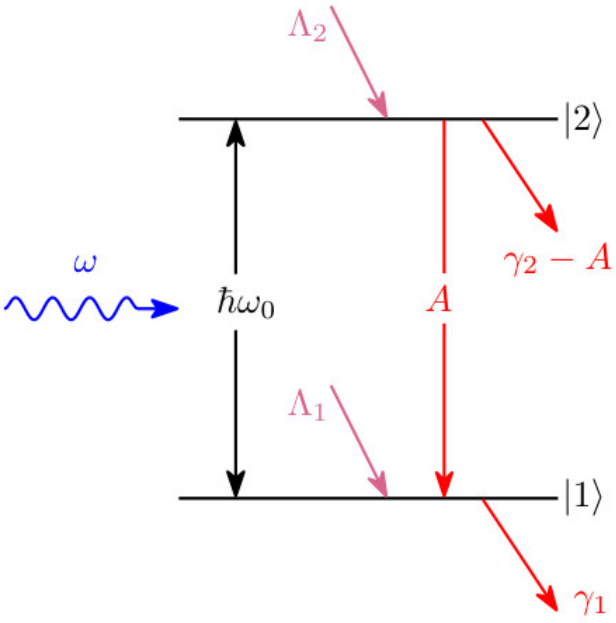
介质极化率随时间演化逐渐趋于稳定。

布居振荡

考虑粒子弛豫情况下的二能级系统 Liouville 方程组的数值求解，同样使用 Runge-Kutta 算法和 solve_ivp 函数实现。

模型

考虑包含耗散的二能级系统，示意图如下



主方程

结合二能级系统的 Hamiltonian 和密度矩阵的 Lindblad 主方程，可得

$$\dot{\sigma}_{21} = -\Gamma\sigma_{21} + i\delta\sigma_{21} - \frac{i}{2}\omega_1\left(\rho_{22} - \rho_{11}\right),$$

$$\dot{\rho}_{22} = \Lambda_2 - \gamma_2\rho_{22} - i\left(\omega_1^*\sigma_{21} - \sigma_{12}\omega_1\right),$$

$$\dot{\rho}_{11} = \Lambda_1 - \gamma_1\rho_{11} + A\rho_{22} + i\left(\omega_1^*\sigma_{21} - \sigma_{12}\omega_1\right).$$

使用 Runge-Kutta 算法，结合 solve_ivp 函数，给出全数值解决方案。

代码

终端执行指令 `touch twolevel_pop_t.py` 新建程序文件，添加如下代码

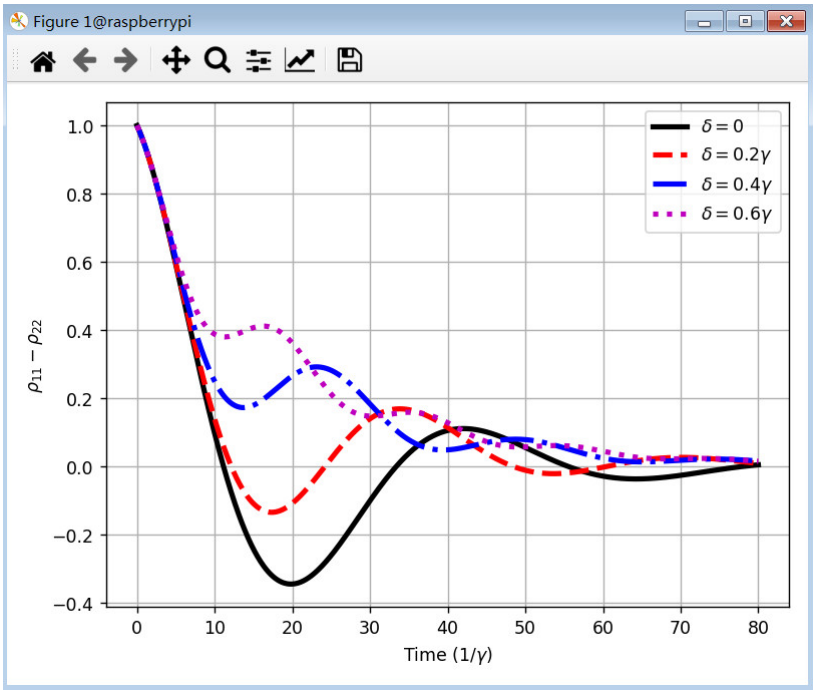
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

# ---------- Parameters ----------
omega1 = 0
Lambda1 = 0
Lambda2 = 0
gamma1 = 0
gamma2 = 0
F = 0
Gamma = (gamma1) / 2 + F
A = 0
Dn = np.array([0, 3])          # n = 0,1,2,3
t_eval = np.arange(0, 0.1)     # time range

# ---------- ODE defination ----------
def rk(t, R, delta, Gamma, gamma1, gamma2, A):
    r11, r22, re21, im21 = R
    dr11 = Lambda1 - A * r22 - im21
    dr22 = - r22 + 2 * im21
    return np.array([dr11, dim21])

# ---------- Scan N ----------
P = np.empty((len(t_eval), 4))   # Population ρ₁₁–ρ₂₂
for k, n in enumerate(Dn):
    delta = n * omega1
    R0 = np.array([1.0, 0])
    sol = solve_ivp(rk, [0, 80], R0, args=(Lambda2,
                                           A),
                    t_eval=t_eval, atol=1e-6)
    P[:, k] = sol.y[0, :]

print("Calculation complete!")

# ---------- Draw ----------
print("Start Drawing...")
plt.plot(t_eval, P[:, 0], '-k',  label=r'$\delta=0$', linewidth=3)
plt.show()
print("Program Terminate.")
```

保存代码。

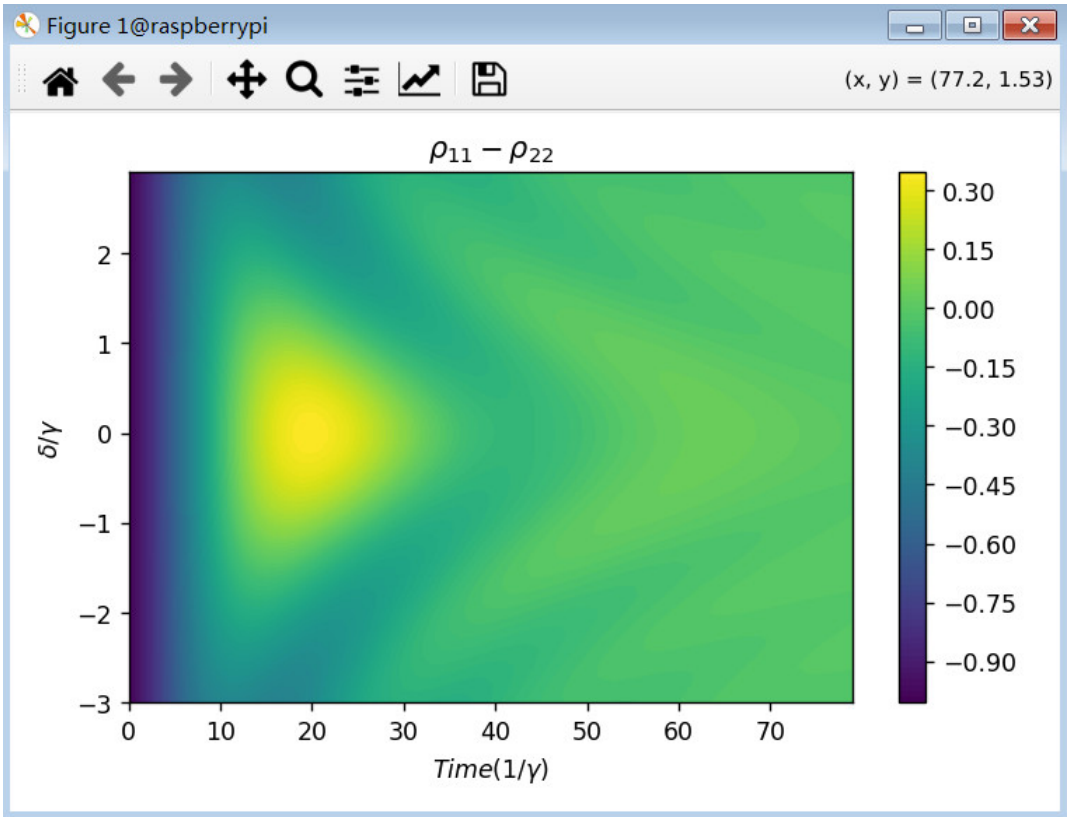
效果

终端运行 `python twolevel_pop_t.py`，弹窗显示绘图结果；

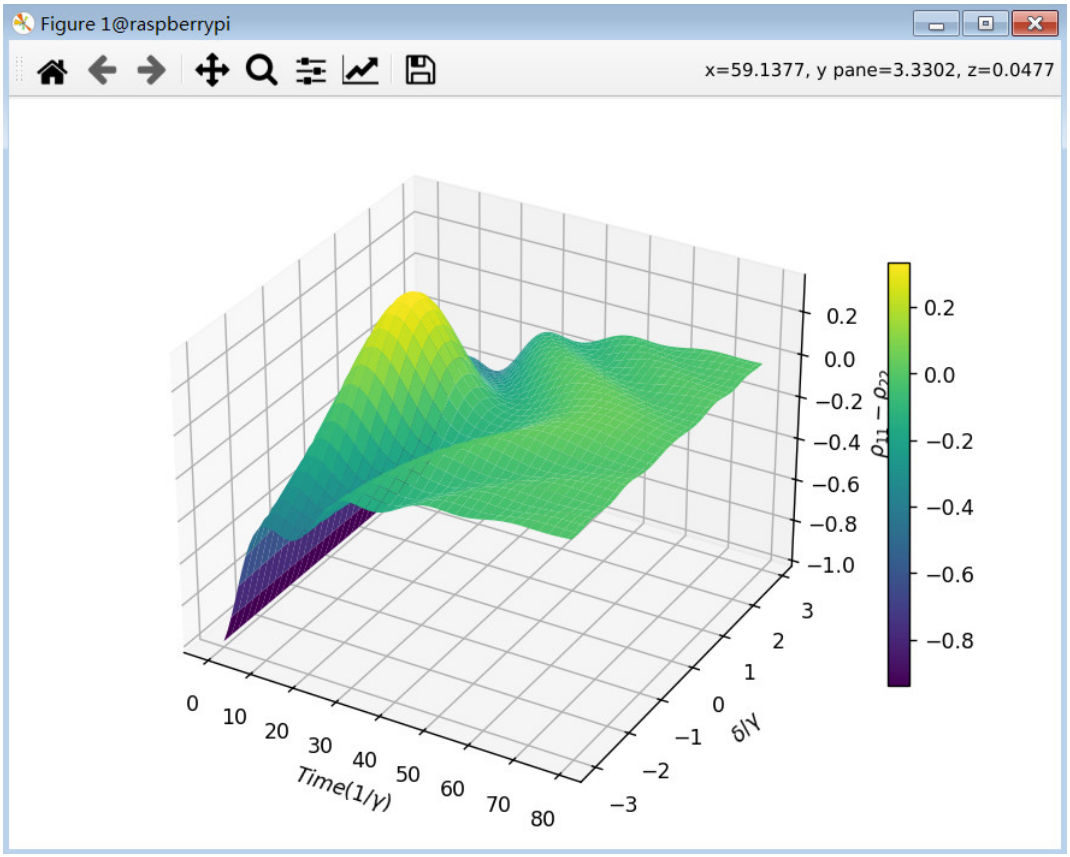
- 不同失谐（delta）条件下，粒子布居（ρ11-ρ22）随时间的演化如下



- 给出布居振荡随时间和失谐变化的伪彩图



- 给出对应的三维渲染结果



该结果表明粒子在基态和激发态之间的 Rabi 振荡随失谐的增加而减弱。

总结

本文介绍了工业树莓派 CM0 NANO 单板计算机结合 Python 软件包 scipy 实现常微分方程组的数值求解，给出了该设备在数值计算领域的应用解决方案，包括二能级系统的 Bloch 方程组的数值求解、方程、代码和数值模拟结果，为相关产品在工业、科研、科学计算领域的快速应用提供了参考。

发布链接：https://blog.csdn.net/qq_36654593/article/details/156080065

【工业树莓派 CM0 NANO 单板计算机】车牌识别

本文介绍了工业树莓派 CM0 NANO 单板计算机结合 LPRNet 算法和 Ultralytics 库实现物车牌识别的项目设计，包括环境部署、软件包安装、模型获取、关键代码以及板端推理等相关流程。

项目介绍

- 准备工作：OpenCV 安装、Ultralytics 软件包安装、预训练模型下载等；
- 车牌识别：采用 LPRNet 算法及 ONNX 模型实现车牌识别的板端推理；

为了快速实现图像分类，需完成 OpenCV 部署和 Ultralytics 软件包的安装等操作。

准备工作

包括硬件连接、虚拟环境创建、OpenCV 安装、Ultralytics 库部署等。

硬件连接

- 连接 WiFi 实现无线网络通信；
- 使用 Type-C 数据线实现设备供电；

OpenCV 安装

OpenCV 是一个开源的计算机视觉库，广泛应用于图像处理、视频分析和机器学习等领域。

为了避免影响系统 Python，采用虚拟环境的方案。

- 创建并激活虚拟环境

```
mkdir ~/cv && cd ~/cv     # 创建 cv 文件夹，便于管理
python3 -m venv venv      # 创建虚拟环境 venv
source venv/bin/activate # 激活虚拟环境 venv
```

- 安装 numpy 和 opencv

```
pip install -U pip numpy                              # 安装 numpy
pip install opencv-python opencv-contrib-python   # opencv 主模块及 contrib
```

- 验证安装

```
python3 -c "import
cv2,sys,numpy;print('OpenCV:',cv2.__version__,'NumPy:',numpy.__version__)"
```

- 输出版本号

详见：OpenCV .

字体安装

为了方便显示中文车牌，安装 CJK 字体

```
sudo apt install fonts-noto-cjk
fc-list | grep -i "Noto Sans CJK" | head -3
```

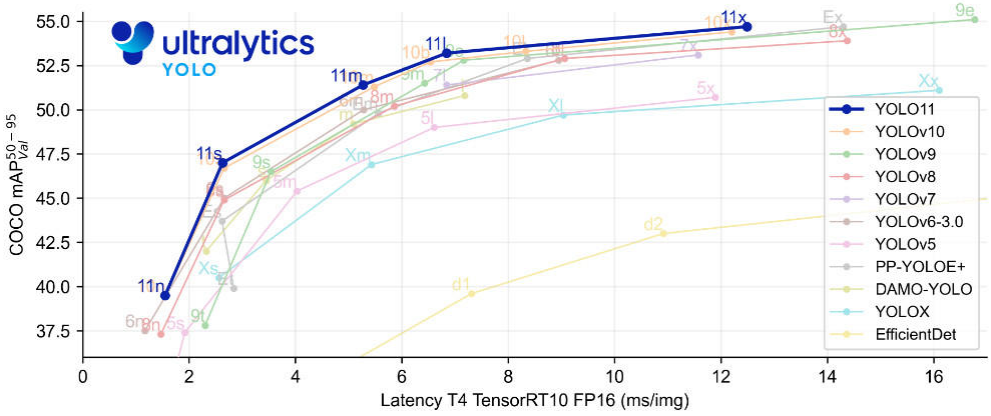
```
ljl@raspberrypi:~/AI_demo/LicensePlateRecognition $ fc-list | grep -i "Noto Sans CJK" | head -3
/usr/share/fonts/opentype/noto/NotoSansCJK-Regular.ttc: Noto Sans CJK JP:style=Regular
/usr/share/fonts/opentype/noto/NotoSansCJK-Regular.ttc: Noto Sans CJK HK:style=Regular
/usr/share/fonts/opentype/noto/NotoSansCJK-Regular.ttc: Noto Sans CJK KR:style=Regular
```

记录字体所在路径，如 `/usr/share/fonts/opentype/noto/NotoSansCJK-Regular.ttc` 以便调用。

Ultralytics 部署

Ultralytics 基于多年在计算机视觉和人工智能领域的基础研究，打造出尖端、先进的 YOLO 模型；具有 **速度快、精度高、操作简便** 等特点。

在目标检测、跟踪、实例分割、图像分类 和 姿态估计 等任务中表现出色。



- 安装 ultralytics 软件包

```
sudo apt install python3-dev python3-pip libopenblas-dev
sudo pip install python3-torch
sudo pip install ultralytics --break-system-packages
```

- 验证安装

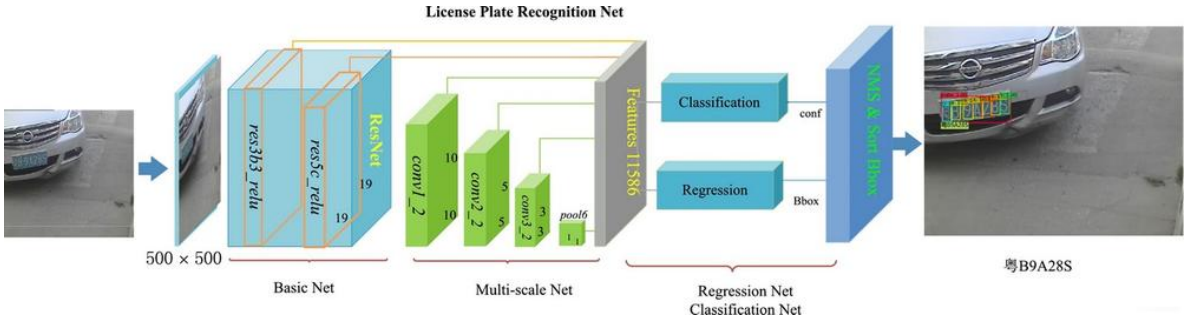
```
python3 -c "import ultralytics, sys, torch; print('ultralytics',
ultralytics.__version__, '| torch', torch.__version__, '| Python',
sys.version.split()[0])"
```

- 输出相应版本号

详见：ultralytics .

车牌识别

车牌识别网络（License Plate Recognition Network，LPRNet）是一种专为车牌识别设计的深度学习模型。



它采用端到端的训练方法，能够直接从原始图像中识别出车牌文本，无需进行传统的字符分割步骤。

这种设计使得 LPRNet 在处理车牌识别任务时更加高效和准确，特别是在面对复杂背景或不同国家的车牌样式时。

详见：LPRNet GitHub .

模型

下载所需模型文件；

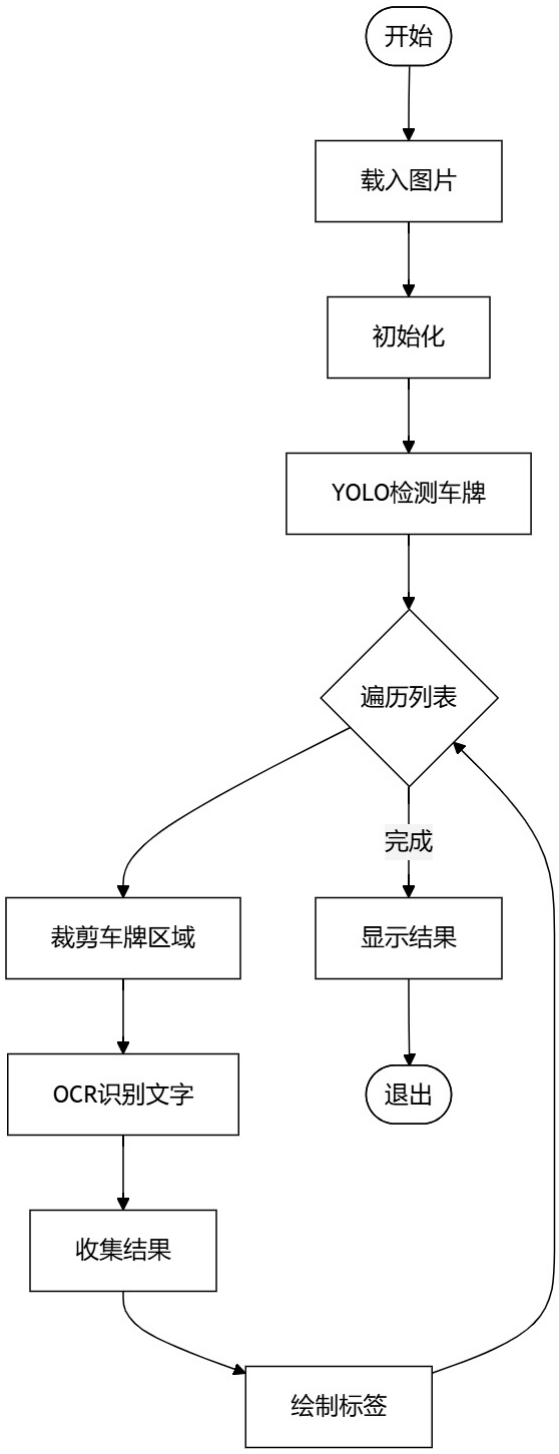
```
wget https://github.com/h030162/PlateRecognition/blob/main/ocr_rec.py
wget https://github.com/h030162/PlateRecognition/blob/main/license_models/dict.txt
wget https://github.com/h030162/PlateRecognition/blob/main/license_models/license_ocr.onnx
wget https://github.com/h030162/PlateRecognition/blob/main/license_models/y11n-pose_plate_best.onnx
```

将文件存放在对应路径

```
license_plate_recognition
   ├── img
   |   ├── yue.jpg
   ├── lpr_onnx.py
   ├── model
   |   ├── dict.txt
   |   ├── license_ocr.onnx
   |   └── y11n-pose_plate_best.onnx
   └── ocr_rec.py
```

参考：PlateRecognition | Github .

流程图



代码

终端执行 `touch lpr_onnx.py` 新建程序文件，并添加如下代码

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

```python
import numpy as np
import cv2
from ocr_rec import TextRecognizer, init_args
from PIL import Image, ImageDraw, ImageFont
from ultralytics import YOLO
import warnings
warnings.filterwarnings("ignore")

# ========== figure ==========
#IMG_FILE = "./img/yue.jpg"
args = init_args().parse_args()
IMG_FILE = args.image_path or './img/test.jpg'   # image path
# 使用方法：python lpr_onnx.py --image_path ./img/jing.jpg

# ========== class ==========
class PlateRecognizer:
    def __init__(self, det_model_path="./model/y11n-pose_plate_best.onnx"):
        self.model_det = YOLO(det_model_path)
        parser = init_args().parse_args()
        self.model_ocr = TextRecognizer(parser)

    def recognize(self, img):
        plate_objs = []
        plates = self.model_det(img, verbose=False)
        for plate, conf in zip(plates[0].boxes.xyxy, plates[0].boxes.conf):
            x1, y1, x2, y2 = map(int, plate.cpu())
            plate_img = img[y1:y2, x1:x2]
            try:
                rec_res, _ = self.model_ocr([plate_img])
            except Exception as E:
                print(E)
                continue
            if len(rec_res[0]) > 0:
                plate_objs.append({
                    'text': rec_res[0][0],
                    'score_text': rec_res[0][1],
                    'bbox': [x1, y1, x2, y2],
                    'score_bbox': conf.cpu().numpy().item()
                })
        return plate_objs

def DrawPlateNum(img, plate_num, x1, y1):
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_pil = Image.fromarray(img_rgb)
    draw = ImageDraw.Draw(img_pil)
    font = ImageFont.truetype("/usr/share/fonts/opentype/noto/NotoSansCJK-Regular.ttc", 40) # 系统字体

    # -------- 标签显示，填充背景 ----------------
    left, top, right, bottom = draw.textbbox((0, 0), plate_num, font=font)
    tw, th = right - left, bottom - top
    draw.rectangle([(x1, y1 - th - 8), (x1 + tw, y1)], fill=(0, 0, 255)) # BGR
    draw.text((x1, y1 - th - 16), plate_num, font=font, fill=(0, 255, 0))

    return cv2.cvtColor(np.array(img_pil, dtype=np.uint8), cv2.COLOR_RGB2BGR)

```

```python
# ========== 主程序 ==========
def main():
    img = cv2.imread(IMG_FILE)
    if img is None:
        print(f"未找到图片：{IMG_FILE}")
        cv2.waitKey(0)
        return

    recognizer = PlateRecognizer()
    plates = recognizer.recognize(img)

    for p in plates:
        x1, y1, x2, y2 = p['bbox']
        cv2.rectangle(img, (x1, y1), (x2, y2), (255, 0, 0), 2)
        img = DrawPlateNum(img, p['text'], x1, y1)
        print(f"车牌：{p['text']}  置信度：{p['score_text']:.4f}  框置信度：{p['score_bbox']:.4f}")

    cv2.imshow("LPR", img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

if __name__ == "__main__":
    main()
```

保存代码。

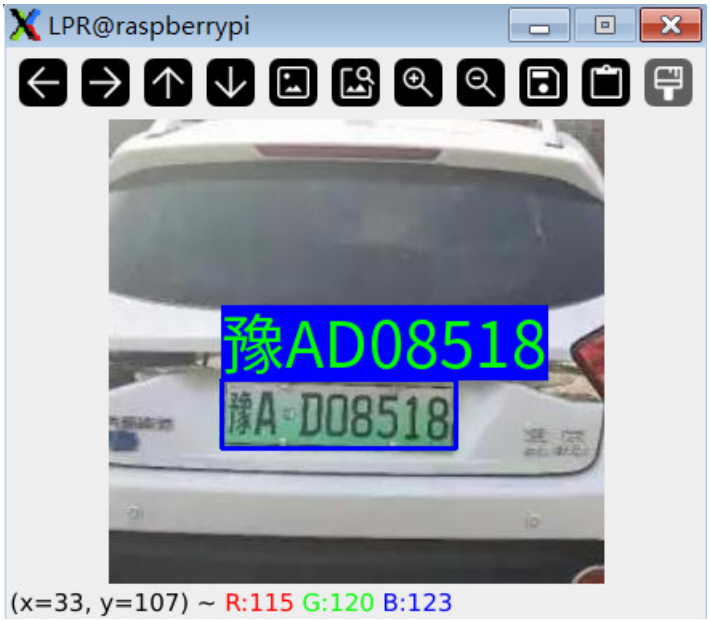
效果

- 终端执行 `python lpr_onnx.py --image_path ./img/yue.jpg` 指令，对目标车牌进行识别
- 终端打印识别到的车牌号、置信度等信息

```
ljl@raspberrypi:~/AI_demo/LicensePlateRecognition $ python lpr_onnx.py --image_path ./img/yu.jpg
2026-01-02 05:55:37.347607828 [W:onnxruntime:Default, device_discovery.cc:164 DiscoverDevicesForPlat
vice discovery failed: device_discovery.cc:89 ReadFileContents Failed to open file: "/sys/class/drm/
/vendor"
Loading ./model/y11n-pose_plate_best.onnx for ONNX Runtime inference...
Using ONNX Runtime 1.23.2 with CPUExecutionProvider
车牌：豫A510AT  置信度：1.0000  框置信度：0.8402
```

- 弹窗显示识别结果

- 更多测试效果



总结

本文介绍了工业树莓派 CM0 NANO 单板计算机结合 LPRNet 算法和 Ultralytics 库实现车牌识别的项目设计，包括环境搭建、预训练模型、工程代码和效果演示等，为相关产品在边缘 AI 领域的快速开发和应用设计提供了参考。

发布链接：https://blog.csdn.net/qq_36654593/article/details/156513147

【工业树莓派 CM0 NANO 单板计算机】人脸识别

本文介绍了工业树莓派 CM0 NANO 单板计算机结合 OpenCV 内置 YuNet 算法和 SFace 模型实现人脸识别的项目设计，包括环境部署、预训练模型获取、关键代码、板端推理、效果演示等流程。

项目介绍

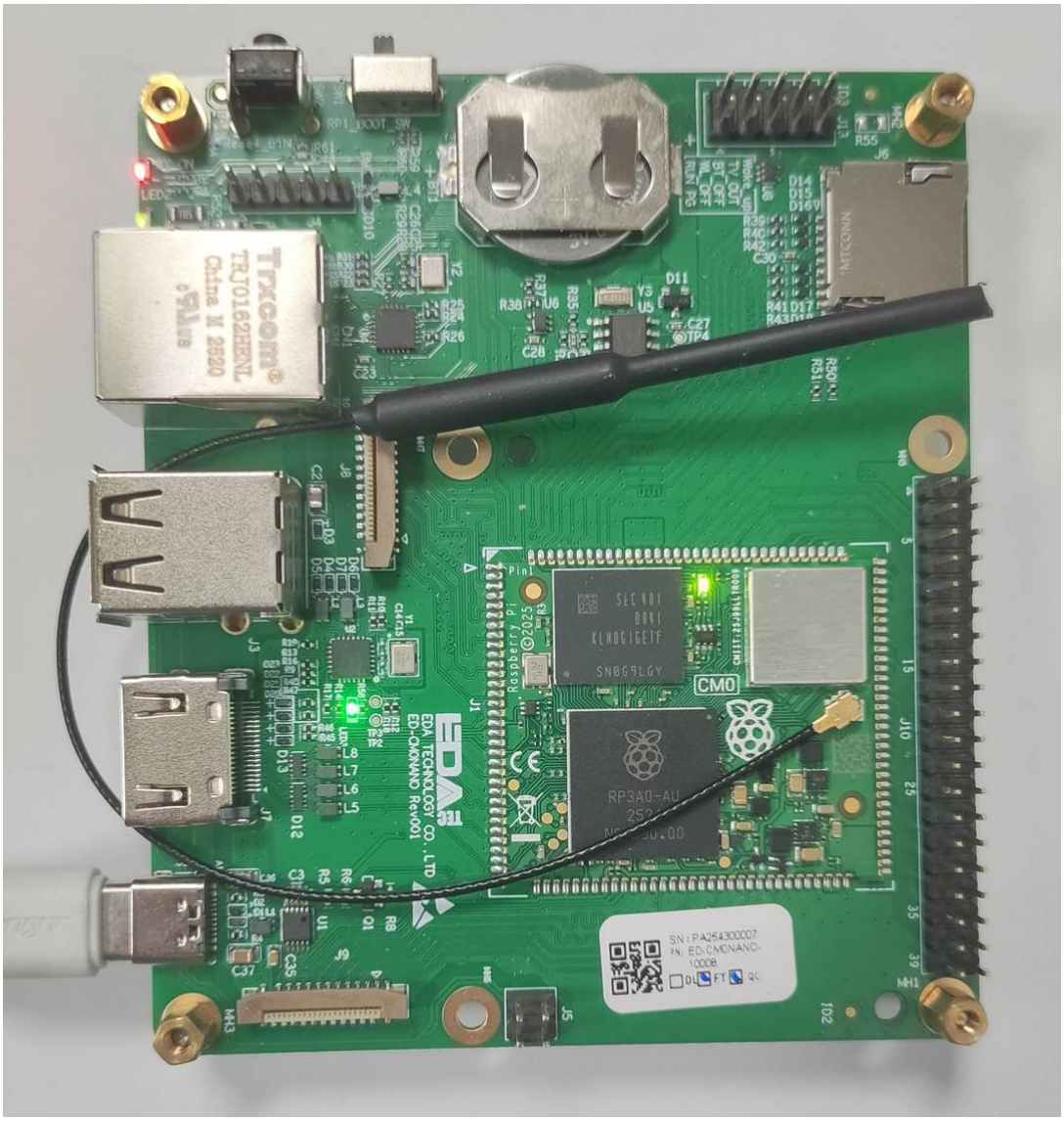
- 准备工作：硬件连接、OpenCV 安装、所需软件包和库安装等；
- 人脸识别：模型获取、训练图片、流程图、代码、人脸识别的板端推理等；

准备工作

包括硬件连接、虚拟环境创建、OpenCV 安装、软件包和库安装等。

硬件连接

- 连接 WiFi 实现无线网络通信；
- 使用 Type-C 数据线实现设备供电；



OpenCV 安装

OpenCV 是一个开源的计算机视觉库，广泛应用于图像处理、视频分析和机器学习等领域。

为了避免影响系统 Python，采用虚拟环境的方案。

- 创建并激活虚拟环境

```
mkdir ~/cv && cd ~/cv     # 创建 cv 文件夹，便于管理
python3 -m venv venv      # 创建虚拟环境 venv
source venv/bin/activate # 激活虚拟环境 venv
```

- 安装 numpy 和 opencv

```
pip install -U pip numpy                              # 安装 numpy
pip install opencv-python opencv-contrib-python   # opencv 主模块及 contrib
```

- 验证安装

```
python3 -c "import
cv2,sys,numpy;print('OpenCV:',cv2.__version__,'NumPy:',numpy.__version__)"
```

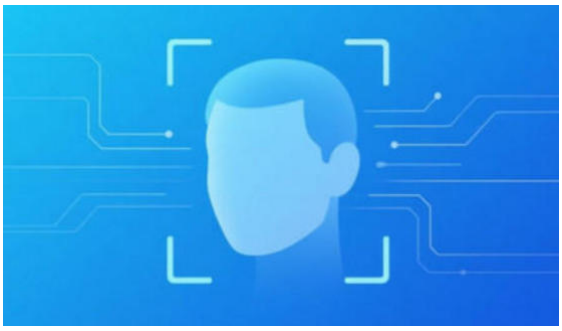
- 输出版本号

```
ljl@raspberrypi:~/AI_demo/FaceRecognition $ python3 -c "import cv2,sys,numpy;pri
nt('OpenCV:',cv2.__version__,'NumPy:',numpy.__version__)"
OpenCV: 4.12.0 NumPy: 2.2.4
```

详见：OpenCV .

人脸识别

OpenCV 作为计算机视觉领域的核心库，其 Python 接口提供了高效的人脸检测与识别能力。



OpenCV 注册并训练目标人脸，使用 YuNet 模型检测人脸，之后结合 sface 模型识别人脸。

详见：opencv_zoo/models/face_recognition_sface · GitHub .

模型

下载所需模型文件；

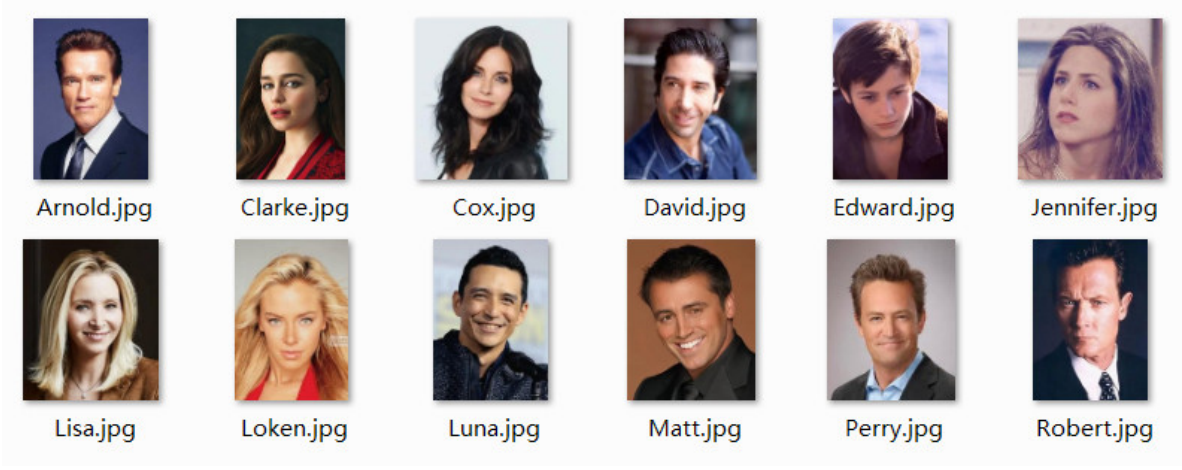
```
wget
https://github.com/opencv/opencv_zoo/blob/main/models/face_detection_yunet/fa
ce_detection_yunet_2023mar.onnx
wget
https://github.com/opencv/opencv_zoo/blob/main/models/face_recognition_sface/
face_recognition_sface_2021dec.onnx
```

将文件存放在 `./model` 路径

参考：SFace Recognition | Github .

训练图片

- 将目标人脸图片裁剪至合适大小；
- 文件名为对应的人名；
- 置于 `./face` 文件夹。

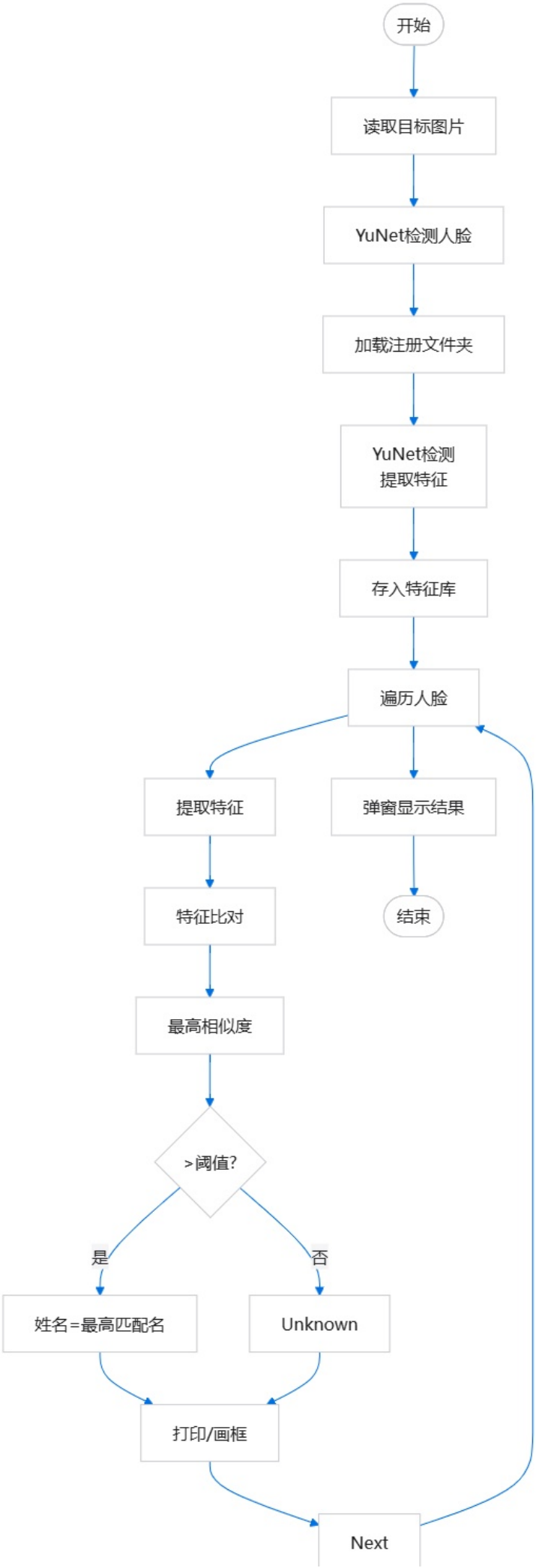


文件目录

```
~/faceRecognition $ tree
.
├── face
│   ├── Arnold.jpg
│   ├── Edward.jpg
│   ├── Linda.jpg
│   └── Robert.jpg
├── fr_onnx.py
├── img
│   ├── test.jpg
│   └── friends.jpg
└── model
    ├── face_detection_yunet_2023mar.onnx
    └── face_recognition_sface_2021dec.onnx
```

将目标识别图片置于 `./img` 文件夹。

流程图


开始
读取目标图片
YuNet检测人脸
加载注册文件夹
YuNet检测
提取特征
存入特征库
遍历人脸
提取特征
弹窗显示结果
特征比对
结束
最高相似度
>阈值?
是
否
姓名=最高匹配名
Unknown
打印/画框
Next

代码

终端执行 `touch fr_onnx.py` 新建程序文件，并添加如下代码

```python
#!/usr/bin/env python3
import cv2
import argparse
import os
import numpy as np
from pathlib import Path
import pickle

# ----------------- trainer file save -----------------------
CACHE = "./model/face_registry.pkl"          # 缓存文件位置

def load_registry():
    if os.path.exists(CACHE):
        with open(CACHE, "rb") as f:
            return pickle.load(f)
    return {}

def save_registry(reg):
    with open(CACHE, "wb") as f:
        pickle.dump(reg, f)

# ------------------ Face Detection ------------------
def detect_faces_yunet(image_path: str,
                       conf_threshold: float = 0.8,
                       model_path: str =
"./model/face_detection_yunet_2023mar.onnx") -> None:
    img = cv2.imread(image_path)
    if img is None:
        raise FileNotFoundError(image_path)
    h, w = img.shape[:2]

    # 初始化 YuNet
    detector = cv2.FaceDetectorYN_create(
        model=model_path,
        config="",
        input_size=(w, h),
        score_threshold=conf_threshold,
        nms_threshold=0.4,
        top_k=5000
    )
    detector.setInputSize((w, h))

    # detect 返回 (status, faces)  取第 1 个元素
    faces = detector.detect(img)[1]
    if faces is None:
        faces = []

    for face in faces:
        x, y, w_box, h_box, *_ = map(int, face[:4])
        score = face[-1]
        cv2.rectangle(img, (x, y), (x + w_box, y + h_box), (0, 255, 0), 2)
```

```
        label = f"{score:.2f}"
        label_size, _ = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX,
0.5, 1)
        cv2.rectangle(img, (x, y - label_size[1] - 4),
                      (x + label_size[0], y), (0, 255, 0), -1)
        cv2.putText(img, label, (x, y - 2), cv2.FONT_HERSHEY_SIMPLEX,
                    0.5, (0, 0, 0), 1, cv2.LINE_AA)

    cv2.namedWindow("YuNet Face Detection", cv2.WINDOW_NORMAL)
    cv2.imshow("YuNet Face Detection", img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

# -------------------- Face Recognition ------------------
def recognize_faces(img_path: str,
                    face_dir: str = "./face",
                    model_path: str =
"./model/face_detection_yunet_2023mar.onnx",
                    rec_model: str =
"./model/face_recognition_sface_2021dec.onnx") -> None:
    """
    1. 读取 img_path 并检测人脸
    2. 对 face_dir 下的每张注册照提取特征
    3. 将目标人脸与注册照逐一比对，取最高余弦相似度
    4. 弹窗画出框+姓名（或 Unknown）
    """
    img = cv2.imread(img_path)
    if img is None:
        raise FileNotFoundError(img_path)
    h, w = img.shape[:2]

    # 检测器
    detector = cv2.FaceDetectorYN_create(
        model=model_path, config="", input_size=(w, h),
        score_threshold=0.8, nms_threshold=0.4, top_k=5000)
    detector.setInputSize((w, h))
    faces = detector.detect(img)[1]
    if faces is None:
        print("未检测到人脸")
        return

    # 识别器
    recognizer = cv2.FaceRecognizerSF_create(rec_model, "")

    # 注册照特征库
    regist = {} if args.retrain else load_registry()  # name -> feature
    if args.retrain or not regist:          # 强制重训 或 缓存空
        print(">>> 重新提取注册照特征 <<<")
        regist = {}
        face_files = list(Path(face_dir).glob("*.*"))
        n_total = len(face_files)
        for idx, fp in enumerate(face_files, 1):
            name = fp.stem
            print(f"[Register] {idx:3d}/{n_total}  {name}", end="\r")
            reg_img = cv2.imread(str(fp))
            if reg_img is None:
                continue
            rh, rw = reg_img.shape[:2]
```

```
            detector.setInputSize((rw, rh))
            reg_faces = detector.detect(reg_img)[1]
            if reg_faces is not None:
                # 只取第一张脸
                aligned = recognizer.alignCrop(reg_img, reg_faces[0])
                feat = recognizer.feature(aligned)
                regist[name] = feat
        print()
        save_registry(regist)
    else:
        print(f">>> 已加载缓存  {len(regist)} 张注册照 <<<")
    detector.setInputSize((w, h))

    if not regist:
        print("注册库为空")
        return

    # 逐一比对
    for face in faces:
        aligned = recognizer.alignCrop(img, face)
        feat = recognizer.feature(aligned)
        best_score, best_name = -1, "Unknown"
        for name, reg_feat in regist.items():
            score = recognizer.match(feat, reg_feat,
cv2.FaceRecognizerSF_FR_COSINE)
            if score > best_score:
                best_score, best_name = score, name
        # 画框+名字
        x, y, w_box, h_box = map(int, face[:4])

        SIM_TH = 0.3                            # 可调，OpenCV 推荐 0.3~0.4
        if best_score < SIM_TH:
            best_name = "Unknown"

        print(f"[{best_name}]  score={best_score:.3f}  box=({x},{y},
{w_box},{h_box})")
        color = (0, 255, 0) if best_name != "Unknown" else (0, 0, 255)
        cv2.rectangle(img, (x, y), (x + w_box, y + h_box), color, 2)
        cv2.putText(img, f"{best_name}:{best_score:.2f}",
                    (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.6, color, 2)

    cv2.namedWindow("Face Recognition", cv2.WINDOW_NORMAL)
    cv2.imshow("Face Recognition", img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

# ---------- 命令行入口 ----------
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("-i", "--image", required=True, help="目标图片路径")
    parser.add_argument("-m", "--mode", choices=["detect", "recognize"],
                        default="recognize", help="detect：仅检测；
recognize：识别")
    parser.add_argument("--retrain", action="store_true",
                        help="重新扫描 face/ 并更新缓存")
    args = parser.parse_args()

    if args.mode == "detect":
```

```
161         detect_faces_yunet(args.image)
162     else:
163         recognize_faces(args.image)
164 
```

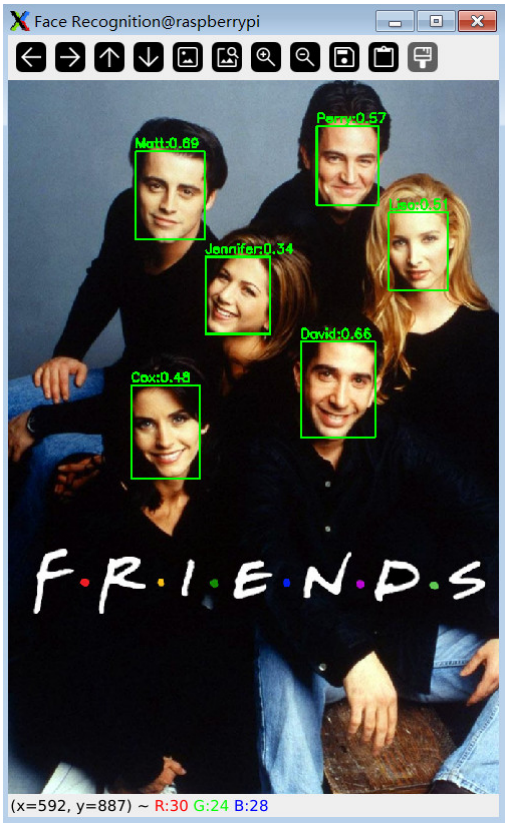
保存代码。

效果

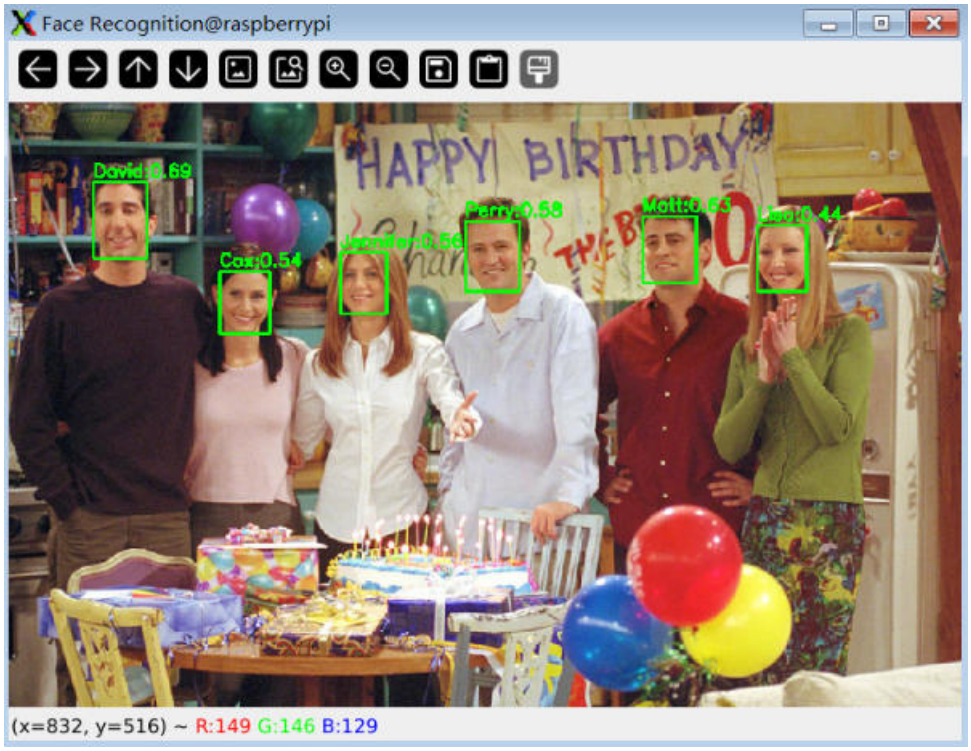
- 终端执行 `python fr_onnx.py -i ./img/test.jpg` 指令，对目标图片进行人脸识别；
- 终端打印识别到的人脸名称、置信度、坐标等信息；

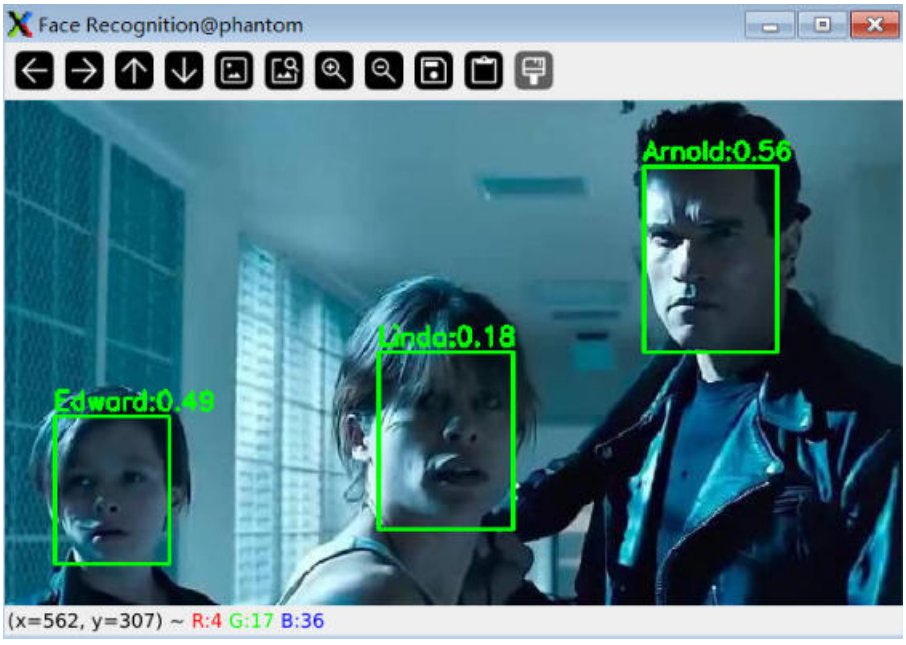
```
ljl@raspberrypi:~/AI_demo/FaceRecognition $ python fr_onnx.py -i ./img/friends.j
pg
[David]  score=0.658  box=(405,366,103,134)
[Matt]  score=0.690  box=(176,99,96,124)
[Cox]  score=0.483  box=(170,427,95,131)
[Perry]  score=0.569  box=(427,64,86,111)
[Lisa]  score=0.513  box=(527,185,82,109)
[Jennifer]  score=0.340  box=(273,247,90,108)
```

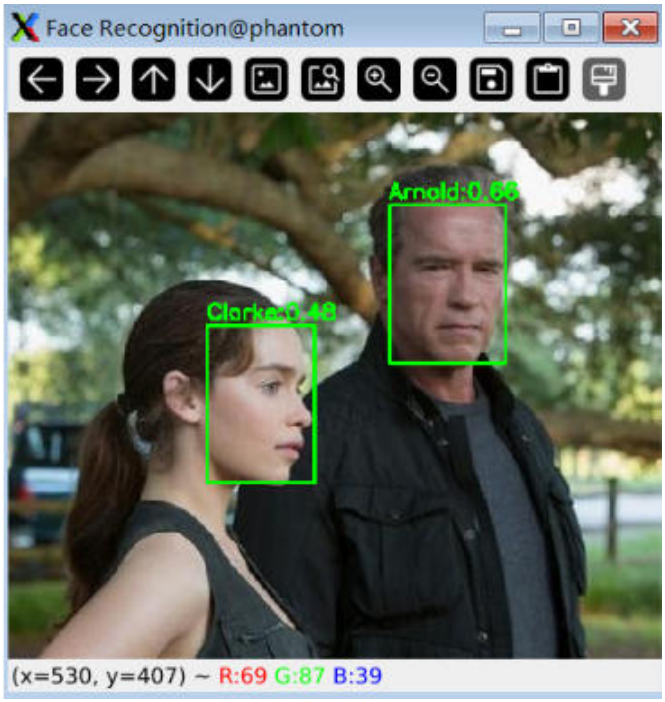
- 弹窗显示识别结果

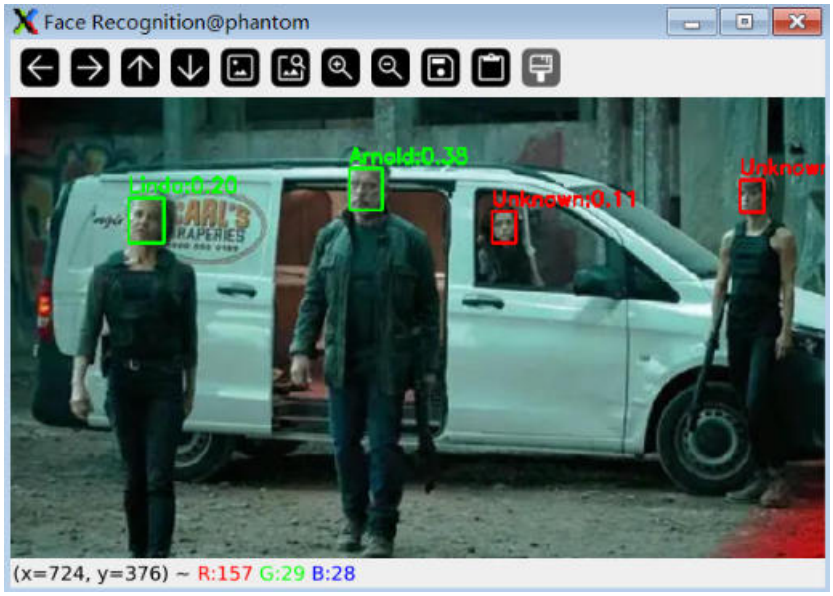


- 更多测试效果


Face Recognition@raspberrypi
HAPPY BIRTHDAY
David:0.69
Cox:0.54
Jennifer:0.66
Perry:0.58
Matt:0.63
Rachel:0.44
(x=832, y=516) ~ R:149 G:146 B:129


Face Recognition@phantom
Arnold:0.56
Linda:0.18
Edward:0.49
(x=562, y=307) ~ R:4 G:17 B:36





总结

本文介绍了工业树莓派 CM0 NANO 单板计算机结合 OpenCV 内置的 YuNet 算法和 SFace 模型实现人脸识别的项目设计，包括环境部署、预训练模型获取、关键代码、板端推理、效果演示等流程，为相关产品在边缘 AI 领域的快速开发和应用设计提供了参考。

发布链接：https://blog.csdn.net/qq_36654593/article/details/156575210

【工业树莓派 CM0 NANO 单板计算机】基于舵机和人脸识别的智能门禁系统

本文介绍了工业树莓派 CM0 NANO 单板计算机结合 OpenCV 人脸识别和 PWM 舵机控制实现智能门禁系统的项目设计，包括硬件连接、舵机控制、人脸识别、网页前端设计、网页服务器设计、流程图、代码和效果演示等流程。

项目介绍

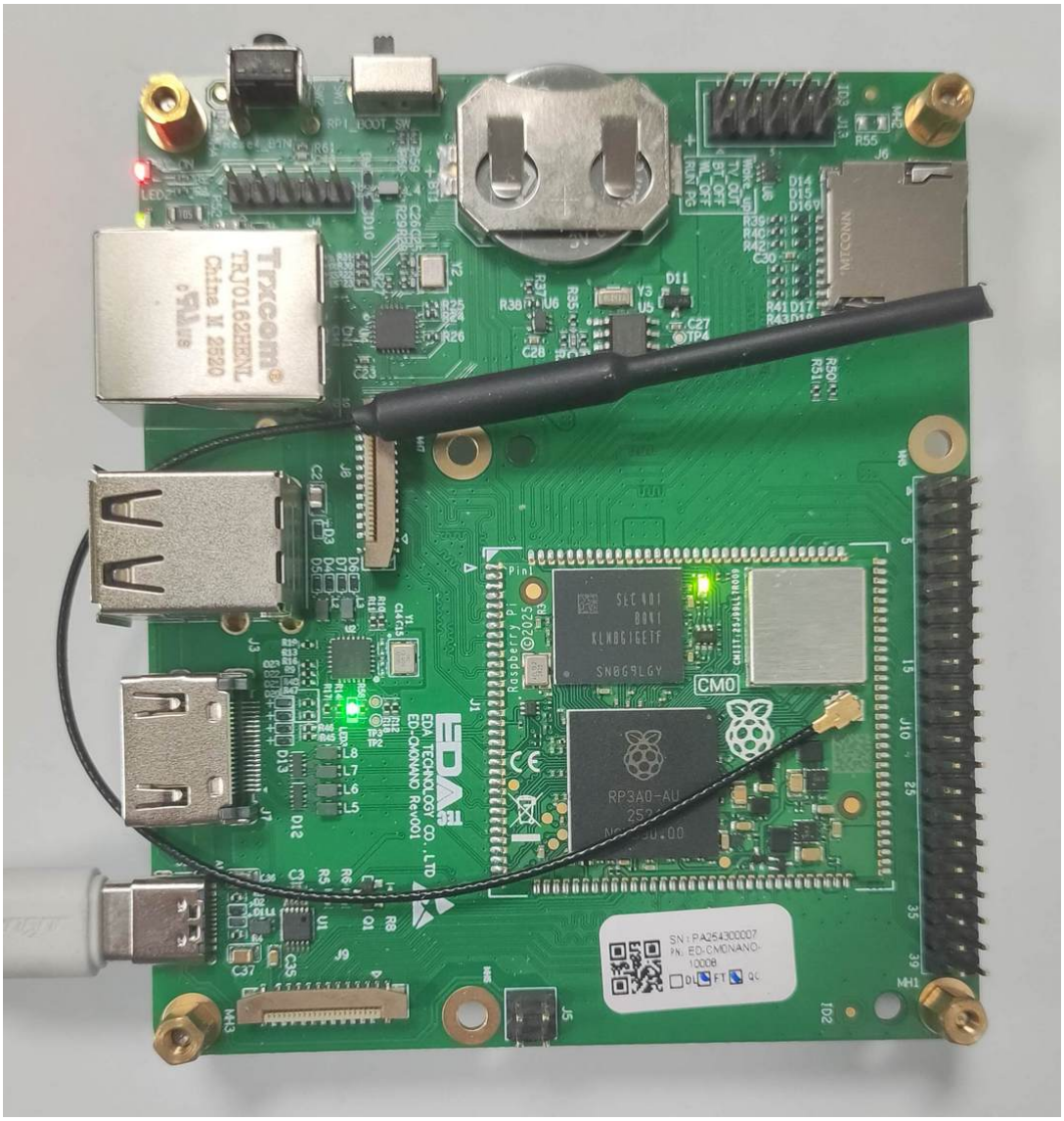
- 准备工作：硬件连接、OpenCV 安装、人脸识别模型、训练图像等；
- 舵机控制：PWM输出、转速和角度控制、代码、效果等；
- 门禁系统：文件目录、流程图、代码、效果等。

准备工作

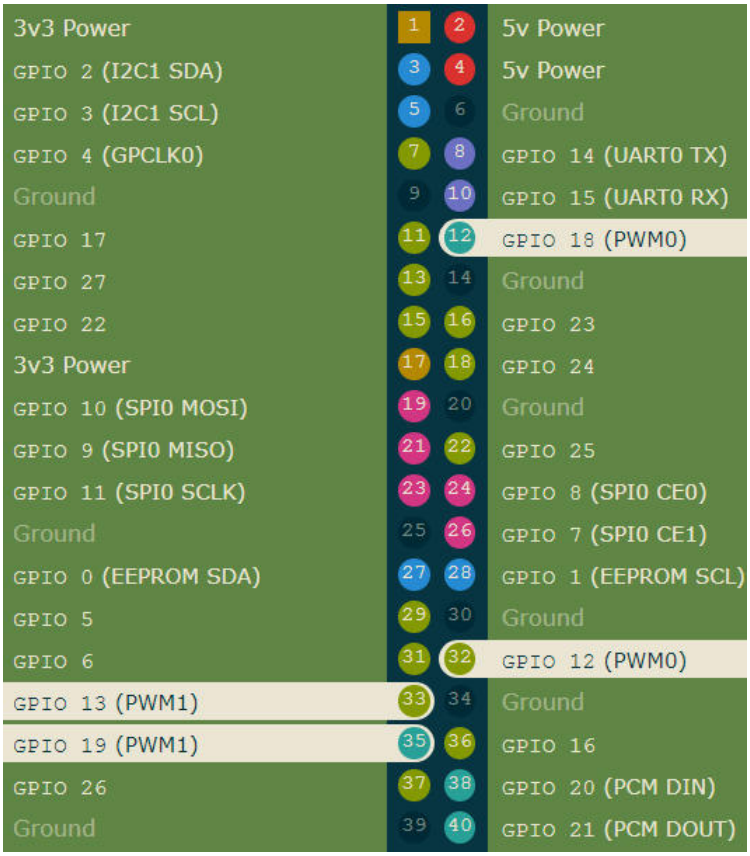
包括硬件连接、虚拟环境创建、OpenCV 安装、模型获取、图像训练等。

硬件连接

- 连接 WiFi 实现无线网络通信；
- 使用 Type-C 数据线实现设备供电；



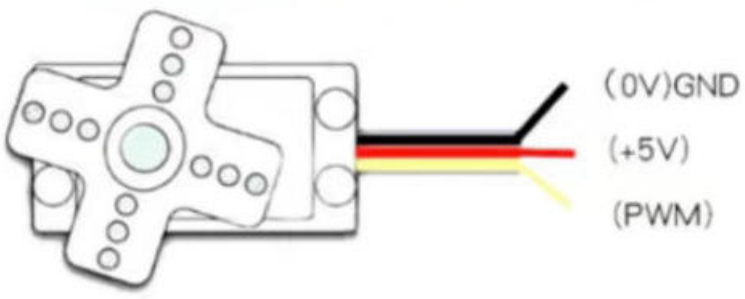
根据板载 40pin 引脚定义，驱动舵机使用支持 PWM 的物理引脚 12，对应 BCM 引脚编号 18；



详见：https://pinout.xyz/

将舵机的信号线与 GPIO18 连接，还有供电和接地

Raspberry Pi	SG90	Description
GPIO18 (Pin12)	S (Yellow)	Signal
5V (Pin2)	5V (Red)	Power supply
GND (Pin6)	GND (Brown)	Ground



OpenCV 安装

- 创建并激活虚拟环境

```
mkdir ~/cv && cd ~/cv     # 创建 cv 文件夹，便于管理
python3 -m venv venv      # 创建虚拟环境 venv
source venv/bin/activate # 激活虚拟环境 venv
```

- 安装 numpy 和 opencv

```
pip install -U pip numpy                              # 安装 numpy
pip install opencv-python opencv-contrib-python   # opencv 主模块及 contrib
```

- 验证安装

```
python3 -c "import
cv2,sys,numpy;print('OpenCV:',cv2.__version__,'NumPy:',numpy.__version__)"
```

详见：OpenCV .

人脸识别

OpenCV 注册并训练目标人脸，使用 YuNet 模型检测人脸，之后结合 sface 模型识别人脸。

详见：opencv_zoo/models/face_recognition_sface · GitHub .

模型获取

下载所需模型文件；

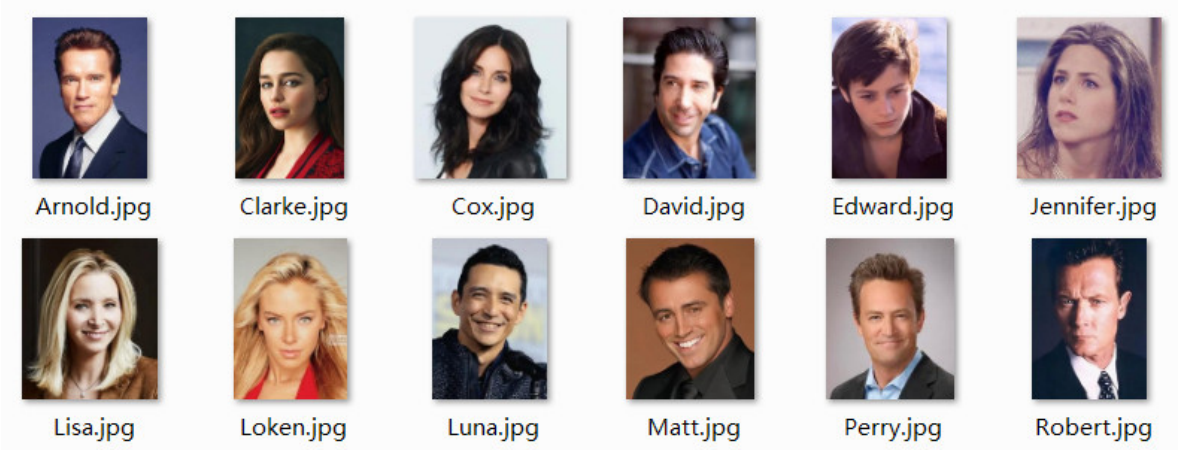
```
wget
https://github.com/opencv/opencv_zoo/blob/main/models/face_detection_yunet/fa
ce_detection_yunet_2023mar.onnx
wget
https://github.com/opencv/opencv_zoo/blob/main/models/face_recognition_sface/
face_recognition_sface_2021dec.onnx
```

将文件存放在 `./model` 路径

参考：SFace Recognition | Github .

训练图片

- 将目标人脸图片裁剪至合适大小；
- 文件名为对应的人名；
- 置于 `./face` 文件夹。



舵机控制

使用树莓派板载 40pin 引脚接口的 PWM 功能，实现 SG90 舵机驱动，并控制旋转速度和角度。

代码

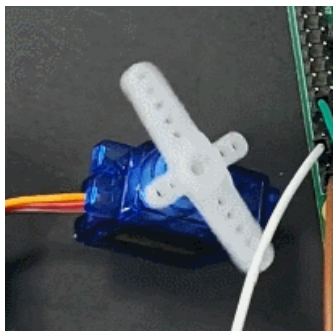
终端执行指令 `touch servo360.py` 新建程序文件并添加如下代码

```python
import sys, time
import RPi.GPIO as GPIO

GPIO_PIN  = 18
FREQ      = 50
CENTER    = 7.5
RANGE     = 2.5

# --------- Parameters ---------
SPEED_DPS = 480              # 实测：每秒 480 度
PWM_DEAD  = 0.05             # 停转
# ------------------------------

def duty(speed):
    return CENTER + max(-1, min(1, speed)) * RANGE

def rotate(target_deg, speed=1.0):
    """
    target_deg : 角度，负值反转
    speed      : 0~1，默认全速
    """
    if not target_deg:
        return
    direction = 1 if target_deg > 0 else -1
    run_speed = speed * direction
    run_time  = abs(target_deg) / (SPEED_DPS * speed)   # 时长

    pwm = GPIO.PWM(GPIO_PIN, FREQ)
    pwm.start(0)
    pwm.ChangeDutyCycle(duty(run_speed))
    time.sleep(run_time)
    pwm.ChangeDutyCycle(CENTER)   # 停
    time.sleep(PWM_DEAD)
    pwm.stop()

if __name__ == '__main__':
    if len(sys.argv) < 2:
        print("缺少角度"); sys.exit(1)
    deg = float(sys.argv[1])

    GPIO.setmode(GPIO.BCM)
    GPIO.setup(GPIO_PIN, GPIO.OUT)
    try:
        rotate(deg)
    finally:
        GPIO.cleanup()
```

保存代码。

效果

终端执行指令 `python servo360.py 90` 舵机逆时针转动 90 度。



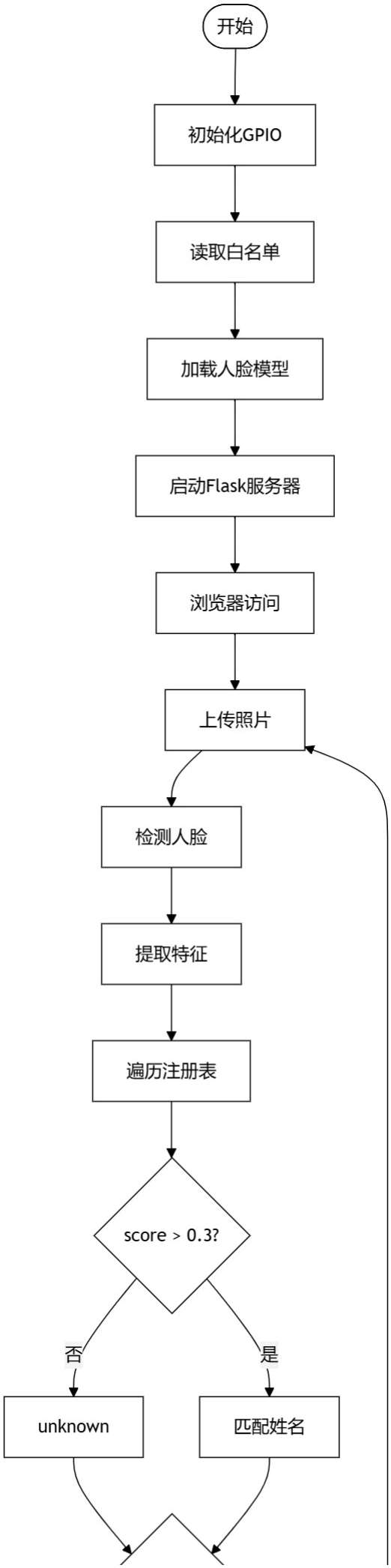
门禁系统

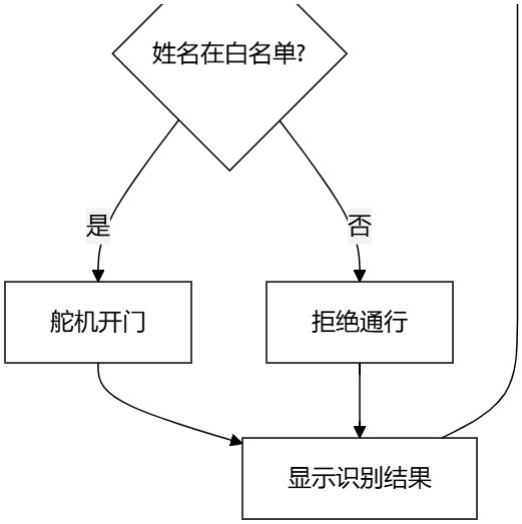
在人脸识别和舵机控制的基础上，实现门禁系统的项目设计，包括文件目录、流程图、代码、效果演示等。

文件目录

```
~/AI/FaceRecognition $ tree
.
├── access.names
├── app.py
├── face
│   ├── Arnold.jpg
│   ├── Clarke.jpg
│   ├── Perry.jpg
│   └── Robert.jpg
├── model
│   ├── face_detection_yunet_2023mar.onnx
│   ├── face_recognition_sface_2021dec.onnx
│   └── face_registry.pkl
├── static
│   └── result.jpg
└── templates
    └── index.html
```

流程图


开始
初始化GPIO
读取白名单
加载人脸模型
启动Flask服务器
浏览器访问
上传照片
检测人脸
提取特征
遍历注册表
score > 0.3?
否
是
unknown
匹配姓名



代码

包含三个代码文件，`./access.names` 为白名单，`./app.py` 为 flask 服务器后端，`` `./templates/index.html `` 为网页前端。

Flask 后端

终端执行 `touch app.py` 新建网页服务器后端程序文件，并添加如下代码

```
#!/usr/bin/env python3
import os, cv2, numpy as np, pickle, time
from pathlib import Path
from flask import Flask, request, jsonify, render_template, url_for
import RPi.GPIO as GPIO
import threading

PIN_SERVO = 18
FREQ      = 50
GPIO.setmode(GPIO.BCM)
GPIO.setup(PIN_SERVO, GPIO.OUT)
pwm = GPIO.PWM(PIN_SERVO, FREQ)
pwm.start(0)

# 读取白名单
ACCESS_LIST = set(line.strip() for line in open('access.names') if
line.strip())

# ---------- 人脸模型 ----------
detector   =
cv2.FaceDetectorYN_create("model/face_detection_yunet_2023mar.onnx", "",
(320, 320))
recognizer =
cv2.FaceRecognizerSF_create("model/face_recognition_sface_2021dec.onnx",
"")
registry   = pickle.loads(Path("model/face_registry.pkl").read_bytes()) if
Path("model/face_registry.pkl").exists() else {}

def rotate(angle, speed=480):
    duty = 2.5 if angle > 0 else 12.5
    pwm.ChangeDutyCycle(duty)
    time.sleep(abs(angle) / speed)
    pwm.ChangeDutyCycle(0)
```

```
28
29  def door_cycle():
30      rotate(90); time.sleep(3); rotate(-90)   # 门禁控制
31
32  # ---------- Flask ----------
33  app = Flask(__name__)
34
35  @app.route('/')
36  def index():
37      return render_template('index.html')
38
39  @app.route('/upload', methods=['POST'])
40  def upload():
41      file = request.files['image']
42      img  = cv2.imdecode(np.frombuffer(file.read(), np.uint8),
    cv2.IMREAD_COLOR)
43      h, w = img.shape[:2]
44      detector.setInputSize((w, h))
45      faces = detector.detect(img)[1]
46      name, score = "Unknown", 0.0
47      if faces is not None:
48          face = faces[0]
49          aligned = recognizer.alignCrop(img, face)
50          feat    = recognizer.feature(aligned)
51          for reg_name, reg_feat in registry.items():
52              s = recognizer.match(feat, reg_feat,
    cv2.FaceRecognizerSF_FR_COSINE)
53              if s > score:
54                  score, name = s, reg_name
55          if score < 0.3:          # 识别阈值
56              name = "Unknown"
57
58      # 门禁动作
59      if name != "Unknown" and name in ACCESS_LIST:
60          threading.Thread(target=door_cycle, daemon=True).start()
61          tip = f"{name} 请通行"
62      else:
63          tip = f"{name} 无权限，拒绝通行"
64
65      # 保存识别结果
66      if faces is not None:
67          x, y, w_box, h_box = map(int, face[:4])
68          cv2.rectangle(img, (x, y), (x + w_box, y + h_box), (0, 255, 0), 2)
69          cv2.putText(img, f"{name}:{score:.2f}", (x, y - 6),
70                      cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
71      out_path = "./static/result.jpg"
72      cv2.imwrite(out_path, img)
73
74      return jsonify(name=name, score=round(score, 3), tip=tip,
75                     result_url=url_for('static', filename='result.jpg'))
76
77  # ---------- 退出 ----------
78  import atexit
79  atexit.register(lambda: (pwm.stop(), GPIO.cleanup()))
80
81  if __name__ == '__main__':
82      app.run(host='0.0.0.0', port=5000, debug=False)
83
```

Web 前端

终端执行 `touch ./templates/index.html` 新建 HTML 前端网页程序，并添加如下代码

```
<!doctype html>
<html lang="zh">
<head>
<meta charset="utf-8">
<title>树莓派门禁</title>
<meta name="viewport" content="width=device-width,initial-scale=1">
<style>
:root{
  --accent:#00c853;
  --danger:#ff1744;
  --bg:#e3f2fd;
  --card:rgba(255,255,255,.75);
  --radius:16px;
  --trans:.35s cubic-bezier(.4,0,.2,1)
}
body{margin:0;height:100vh;display:flex;align-items:center;justify-content:center;background:var(--bg);font-family:system-ui,Arial}
#card{width:320px;padding:32px 24px;background:var(--card);backdrop-filter:blur(12px);border-radius:var(--radius);box-shadow:0 8px 32px rgba(0,0,0,.1);text-align:center;transition:var(--trans)}
h2{margin:0 0 20px;font-size:22px;font-weight:600;color:#0d47a1}
input[type=file]{display:none}
label{display:inline-block;padding:10px 20px;border:2px dashed var(--accent);border-radius:var(--radius);cursor:pointer;color:var(--accent);transition:var(--trans)}
label:hover{background:var(--accent);color:#fff}
button{margin-top:16px;padding:10px 0;width:100%;border:none;border-radius:var(--radius);background:var(--accent);color:#fff;font-size:16px;cursor:pointer;transition:var(--trans);box-shadow:0 2px 8px rgba(0,200,83,.3)}
button:active{transform:scale(.97)}
.status{margin-top:18px;font-size:17px;height:24px;opacity:0;transition:var(--trans)}
.status.show{opacity:1}
.status.ok{color:var(--accent)}
.status.no{color:var(--danger)}
img{width:100%;border-radius:var(--radius);margin-top:16px;box-shadow:0 4px 16px rgba(0,0,0,.08);display:none}
</style>
</head>
<body>
<div id="card">
  <h2>人脸识别门禁</h2>
  <input type="file" id="f" accept="image/*">
  <label for="f">选择照片</label>
  <button onclick="up()">上传识别</button>
  <div id="s" class="status"></div>
  <img id="i">
</div>

<script>
async function up(){
```

```
  const file=f.files[0];
  if(!file)return alert('请选择图片');
  s.className='status show';s.textContent='识别中…';
  const fd=new FormData();fd.append('image',file);
  const r=await(fetch('/upload',
{method:'POST',body:fd}).then(x=>x.json()));
  s.textContent=r.tip;
  s.classList.toggle('ok',!r.tip.includes('拒绝'));
  s.classList.toggle('no',r.tip.includes('拒绝'));
  i.src=r.result_url+'?t='+Date.now();i.style.display='block';
  setTimeout(()=>{s.textContent='已关门，等待识
别';i.style.display='none'},3000)
}
</script>
</body>
</html>
```

白名单

终端执行 `touch access.names` 新建白名单文件，并添加人名列表

```
Linda
Edward
Clarke
```

保存代码。

效果

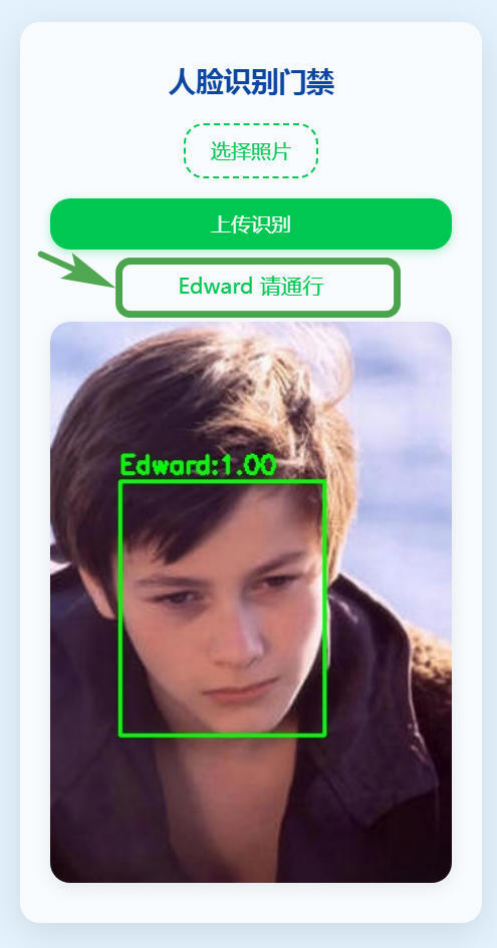
终端执行指令 `python app_DC.py` 运行程序；

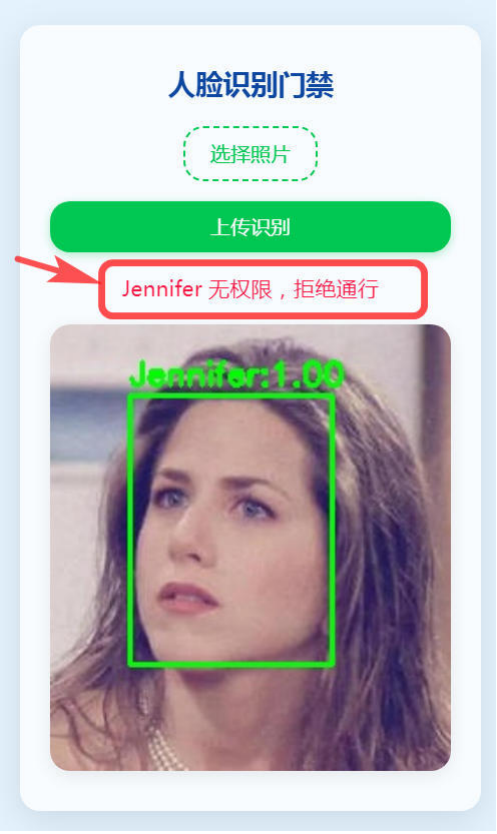
终端打印 Web 服务器网址，如 `http://192.168.31.117:5000/` ；

```
ljl@raspberrypi:~/AI/FaceRecognition $ python app_DC.py
 * Serving Flask app 'app_DC'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI
server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.31.117:5000
Press CTRL+C to quit
192.168.31.101 - - [07/Jan/2026 19:14:16] "POST /upload HTTP/1.1" 200 -
192.168.31.101 - - [07/Jan/2026 19:14:16] "GET /static/result.jpg?t=1767784459210 HTTP/1.1" 200 -
192.168.31.101 - - [07/Jan/2026 19:14:29] "POST /upload HTTP/1.1" 200 -
192.168.31.101 - - [07/Jan/2026 19:14:29] "GET /static/result.jpg?t=1767784472423 HTTP/1.1" 200 -
```

浏览器打开服务器前端网页；

- 点击 `选择文件` 按钮，加载目标识别人脸；
- 点击 `上传识别` 按钮，立即显示识别结果、是否允许通行；





- 同时舵机逆时针转动，控制门禁档杆移动，表示允许通过；
- 待三秒钟后，舵机顺时针旋转 90 度，表示门禁关闭；
- 网页前端显示门禁已关闭，回到 `等待识别` 状态。

动态效果

总结

本文介绍了工业树莓派 CM0 NANO 单板计算机结合 OpenCV 内置的 YuNet 算法和 SFace 模型实现人脸识别的项目设计，包括环境部署、预训练模型获取、关键代码、板端推理、效果演示等流程，为相关产品在边缘 AI 领域的快速开发和应用设计提供了参考。

发布链接：https://blog.csdn.net/qq_36654593/article/details/156761888

【工业树莓派 CM0 NANO 单板计算机】YOLO26 部署方案

本文介绍了工业树莓派 CM0 NANO 单板计算机结合 OpenCV 和 Ultralytics 库实现 YOLO26 板端部署，并实现目标识别、姿态估计、图像分割、图像分类、旋转框检测的项目设计，包括环境部署、模型获取、关键代码、效果演示等。

项目介绍

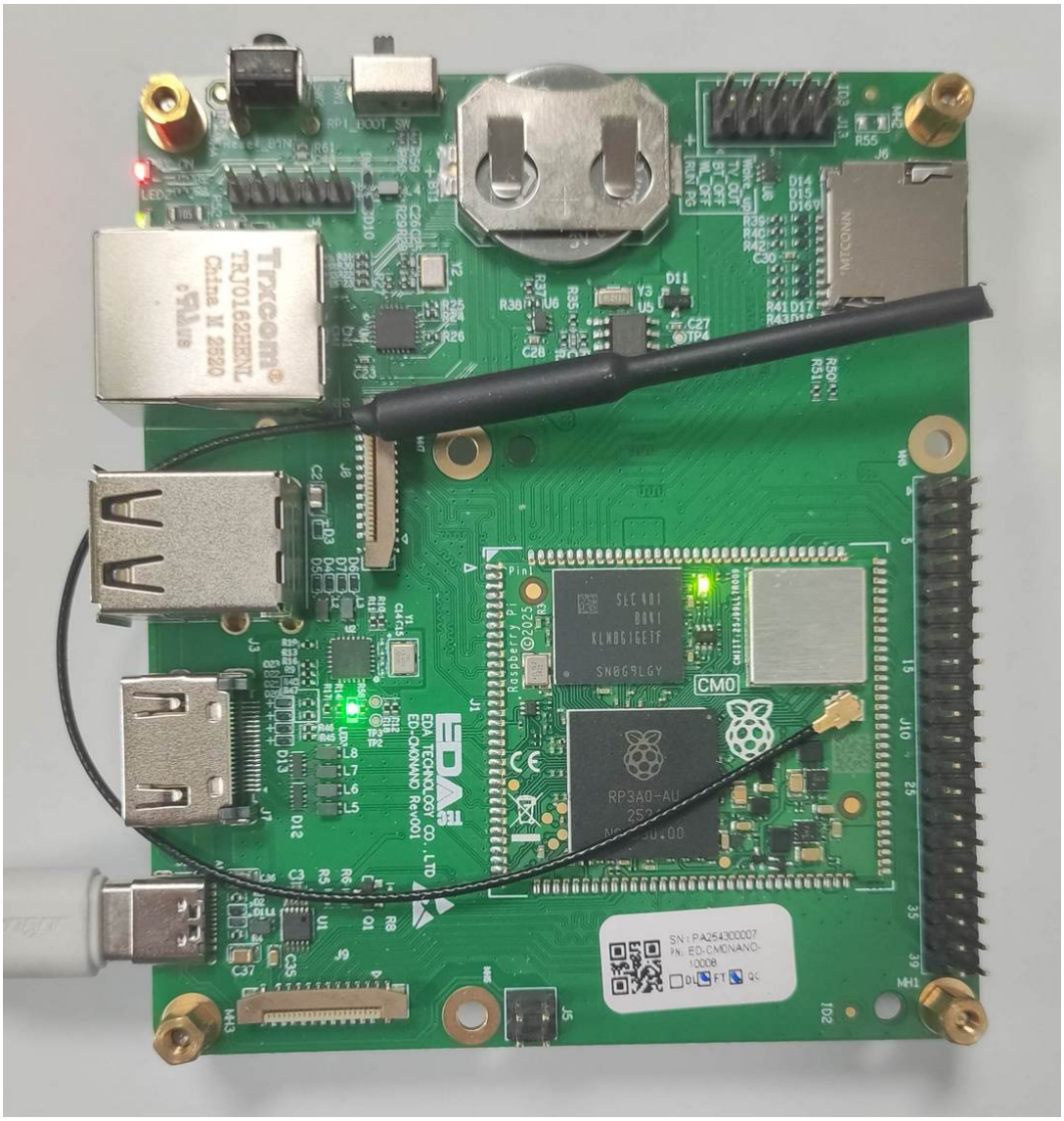
- 准备工作：OpenCV 安装、Ultralytics 软件包安装、YOLO26 预训练模型获取等；
- YOLO26：目标检测、实例分割、图像分类、姿态估计、旋转框检测等；

准备工作

包括硬件连接、OpenCV 安装、Ultralytics 库部署、YOLO26 模型下载等。

硬件连接

- 连接 WiFi 实现无线网络通信；
- 使用 Type-C 数据线实现设备供电；



OpenCV 安装

OpenCV 是一个开源的计算机视觉库，广泛应用于图像处理、视频分析和机器学习等领域。

为了避免影响系统 Python，采用虚拟环境的方案。

- 创建并激活虚拟环境

```
python3 -m venv venv      # 创建虚拟环境 venv
source venv/bin/activate # 激活虚拟环境 venv
```

- 安装 numpy 和 opencv

```
pip install opencv-python opencv-contrib-python
```

- 验证安装

```
python3 -c "import
cv2,sys,numpy;print('OpenCV:',cv2.__version__,'NumPy:',numpy.__version__)"
```

- 输出版本号

详见：OpenCV .

Ultralytics 部署

Ultralytics 基于多年在计算机视觉和人工智能领域的基础研究，打造出尖端、先进的 YOLO 模型；具有**速度快**、**精度高**、**操作简便** 等特点。

在目标检测、跟踪、实例分割、图像分类 和 姿态估计 等任务中表现出色。

- 安装 ultralytics 软件包（虚拟环境下）

```
pip install --no-cache-dir --no-deps torch # 省内存
pip install ultralytics
```

- 验证安装

```
python3 -c "import ultralytics, sys, torch; print('ultralytics',
ultralytics.__version__, '| torch', torch.__version__, '| Python',
sys.version.split()[0])"
```

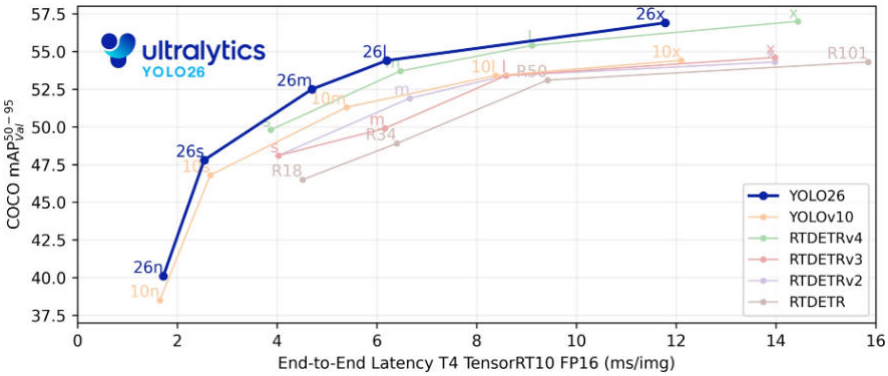
- 输出相应版本号

```
(venv) ljl@raspberrypi:~/AI_demo $ python3 -c "import ultralytics, sys, torch; p
rint('ultralytics', ultralytics.__version__, '| torch', torch.__version__, '| Py
thon', sys.version.split()[0])"
ultralytics 8.4.5 | torch 2.9.1+cpu | Python 3.13.5
(venv) ljl@raspberrypi:~/AI_demo $
```

详见：ultralytics .

YOLO26

Ultralytics YOLO26 是 YOLO 系列实时对象检测器的最新演进，从头开始专为**边缘和低功耗设备**而设计。它引入了简化的设计，消除了不必要的复杂性，同时集成了有针对性的创新，以实现更快、更轻、更易于访问的部署。



YOLO26 的架构遵循三个核心原则：

- **简洁性:** YOLO26是 **端到端** 模型，直接生成预测结果，无需非极大值抑制（NMS）。通过消除这一后处理步骤，推理变得更快、更轻量。
- **部署效率：** 端到端设计消除了管道的整个阶段，从而大大简化了集成，减少了延迟。
- **训练创新：** YOLO26 引入 **MuSGD 优化器**，带来了增强的稳定性和更快的收敛，将语言模型中的优化进展转移到计算机视觉领域。
- **任务特定优化：** YOLO26 针对专业任务引入了有针对性的改进，包括用于 Segmentation 的语义分割损失和多尺度原型模块，用于高精度姿势估计的残差对数似然估计 (RLE)，以及通过角度损失优化解码以解决旋转框检测中的边界问题。

这些创新共同提供了一个模型系列，该模型系列在小对象上实现了更高的精度，提供了无缝部署，并且在CPU 上的运行速度提高了 43%，使其更适合资源受限环境。

详见：YOLO26 | Ultralytics .

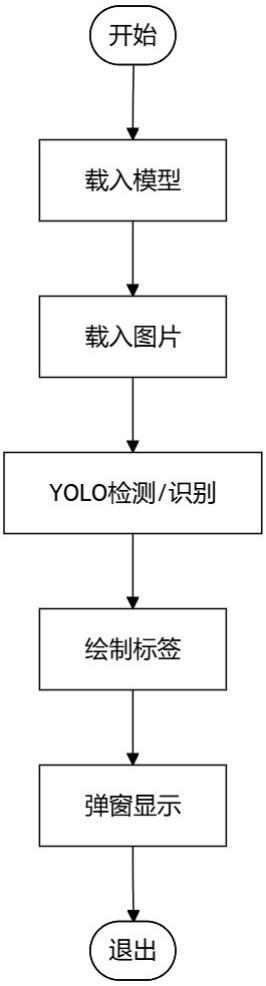
模型获取

下载所需模型文件；

```
wget https://github.com/ultralytics/assets/releases/download/v8.4.0/yolo26n.pt
wget https://github.com/ultralytics/assets/releases/download/v8.4.0/yolo26n-seg.pt
wget https://github.com/ultralytics/assets/releases/download/v8.4.0/yolo26n-cls.pt
wget https://github.com/ultralytics/assets/releases/download/v8.4.0/yolo26n-pose.pt
wget https://github.com/ultralytics/assets/releases/download/v8.4.0/yolo26n-obb.pt
```

将文件存放在 `./model` 路径。

流程图



目标检测

目标检测是一项涉及识别图像或视频流中目标的位置和类别的任务。

目标检测器的输出是一组边界框，这些边界框包围了图像中的目标，以及每个框的类别标签和置信度分数。

详见：Object Detection .

代码

终端执行 `touch or.py` 新建程序文件，并添加如下代码

```python
import cv2
from ultralytics import YOLO

model = YOLO('./model/yolo26n.pt')       # 加载模型
img = cv2.imread('./img/road.jpg')       # 加载图片

results = model(img, imgsz=640, conf=0.25)    # 返回 Boxes
annotated = results[0].plot()                 # 画框和标签

cv2.imshow("YOLO Detection", annotated)   # 弹窗显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

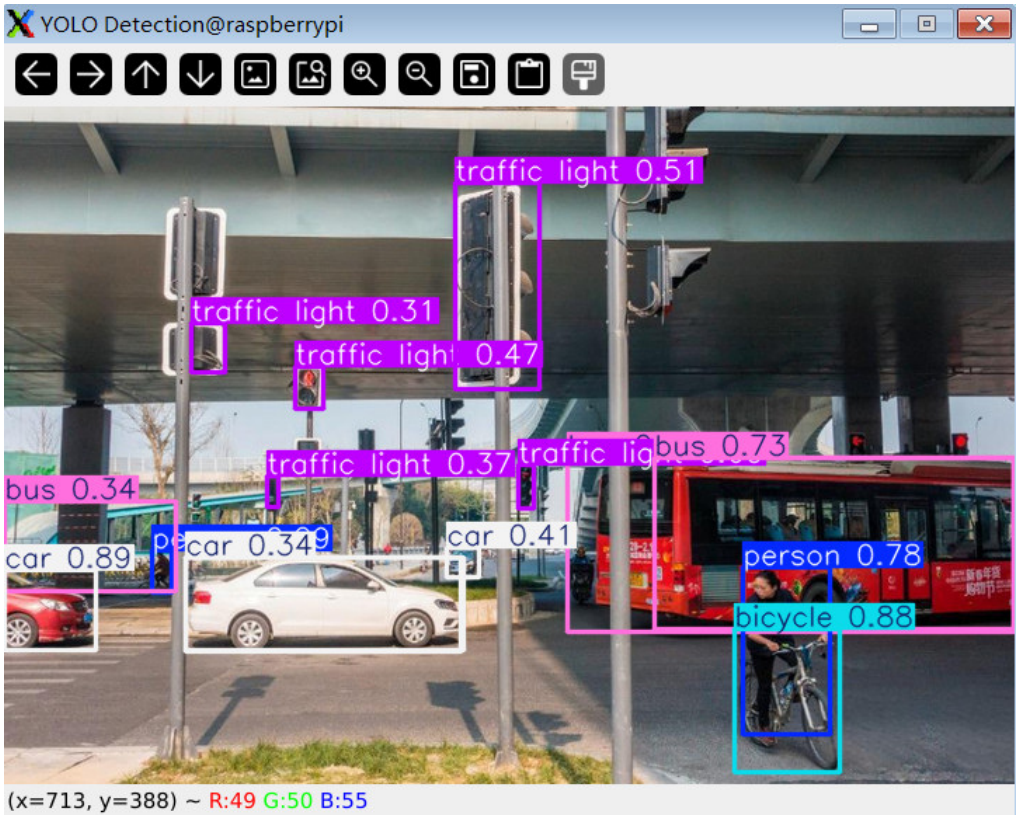
保存代码。

效果

- 终端执行 `python or.py` 指令，对图片进行物体识别
- 终端打印识别到的物体、置信度、耗时等信息

```
(venv) ljl@raspberrypi:~/AI_demo/YOLO26 $ python or.py

0: 448x640 2 persons, 1 bicycle, 4 cars, 3 buss, 5 traffic lights, 3922.8ms
Speed: 200.7ms preprocess, 3922.8ms inference, 36.1ms postprocess per image at s
hape (1, 3, 448, 640)
```

- 弹窗显示识别结果



实例分割

实例分割是识别图像中的各个对象，并将它们与图像的其余部分分割开来。

实例分割模型的输出是一组掩码或轮廓，它们勾勒出图像中每个对象，以及每个对象的类别标签和置信度分数。

详见：Instance Segmentation .

代码

终端执行 `touch seg.py` 新建程序文件，并添加如下代码

```python
import cv2
from ultralytics import YOLO

model = YOLO('./model/yolo26n-seg.pt')      # 分割模型
img = cv2.imread('./img/bicycle.jpg')


results = model(img, imgsz=640, conf=0.25)    # 推理分割
annotated = results[0].plot(boxes=False, labels=False)      # mask

cv2.namedWindow("YOLO26-Seg", cv2.WINDOW_NORMAL)
cv2.imshow("YOLO26-Seg", annotated)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

保存代码。

效果

- 终端执行 `python seg.py` 指令，对目标进行图像分割
- 终端打印识别到的物体、坐标、耗时等信息

```
(venv) ljl@raspberrypi:~/AI_demo/YOLO26 $ python seg.py

0: 640x640 1 person, 2 bicycles, 1 dog, 6951.1ms
Speed: 251.2ms preprocess, 6951.1ms inference, 198.0ms postprocess per image at
shape (1, 3, 640, 640)
```

- 弹窗显示图像分割结果

图像分类

图像分类是将整个图像分类到一组预定义的类别中。

图像分类器的输出是单个类别标签和一个置信度分数。

详见：Image Classification .

代码

终端执行 `touch cls.py` 新建程序文件，并添加如下代码

```python
import cv2
from ultralytics import YOLO

model = YOLO('./model/yolo26n-cls.pt')       # 加载模型
img = cv2.imread('./img/road.jpg')       # 加载图片

results = model(img, imgsz=640, conf=0.25)    # 返回 Boxes
annotated = results[0].plot()                 # 画框和标签

cv2.imshow("YOLO-Classify", annotated)    # 弹窗显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

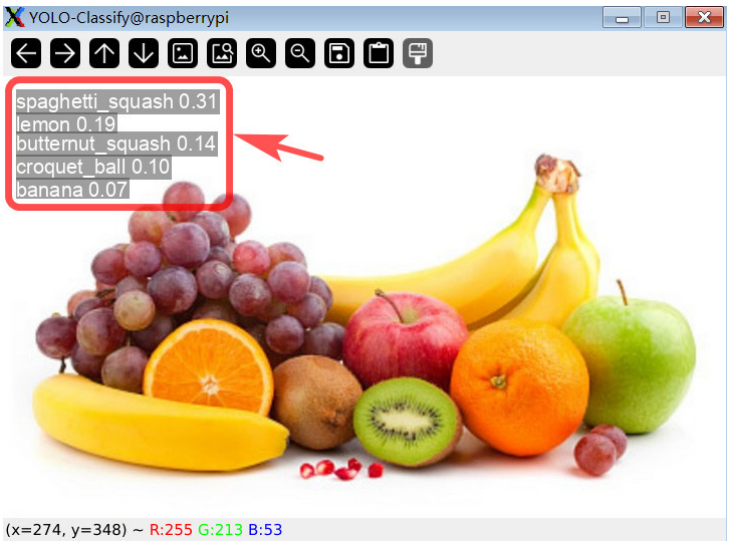
保存代码。

效果

- 终端执行 `python cls.py` 指令，对图片目标进行分类；
- 终端打印识别到的物体、置信度、耗时、坐标等信息

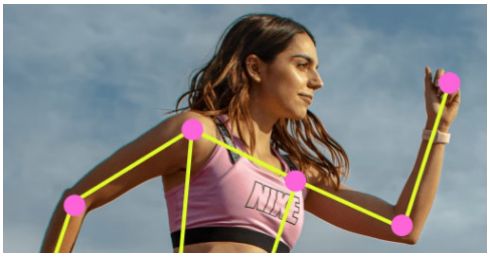
```
(venv) ljl@raspberrypi:~/AI_demo/YOLO26 $ python cls.py

0: 640x640 moving_van 0.19, minibus 0.10, pole 0.08, pay-phone 0.05, mobile_home
 0.03, 3034.3ms
Speed: 512.0ms preprocess, 3034.3ms inference, 11.8ms postprocess per image at s
hape (1, 3, 640, 640)
```

- 弹窗显示图像分类结果



姿态估计

姿势估计是一项涉及识别图像中特定点的位置的任务，这些点通常称为关键点。关键点可以代表对象的各个部分，例如关节、地标或其他独特特征。



姿势估计模型的输出是一组点，这些点代表图像中对象上的关键点，通常还包括每个点的置信度分数。

详见：Pose Estimation .

代码

终端执行 `touch pos.py` 新建程序文件，并添加如下代码

```python
import cv2
from ultralytics import YOLO

model = YOLO('./model/yolo26n-pose.pt')   # 关键点模型
img = cv2.imread('./img/dance.jpg')

results = model(img, imgsz=640, conf=0.25)   # 返回 Keypoints
# annotated = results[0].plot(kpt_radius=4, kpt_line=True)  # 画点+连线
annotated = results[0].plot(
    boxes=False,      # 无外框
    labels=False,     # 无标签
    conf=False,       # 无置信度
    kpt_radius=4,     # 点大小
    kpt_line=True     # 骨架连线
)

cv2.imshow("YOLO-Pose", annotated)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

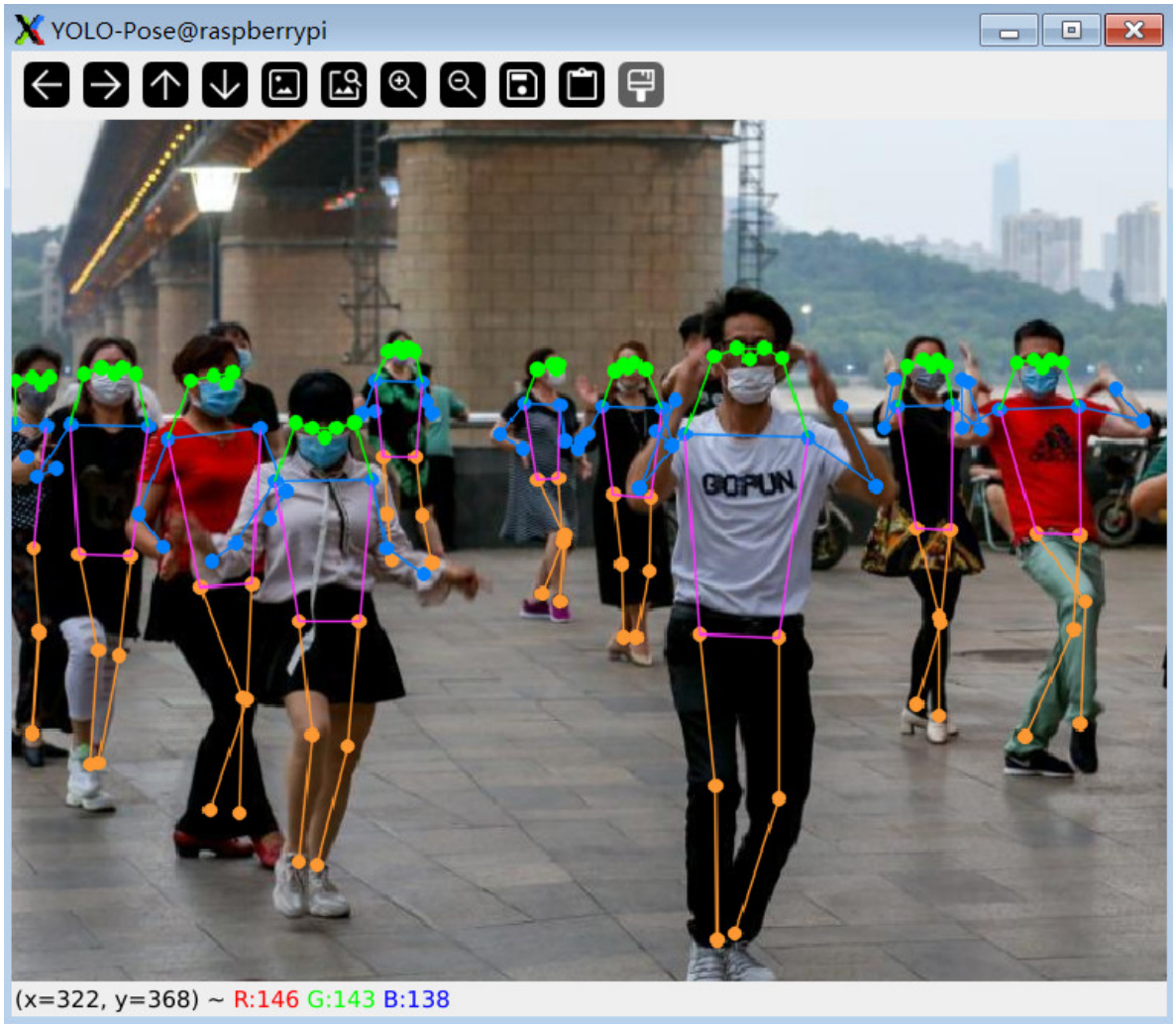
保存代码。

效果

- 终端执行 `python pos.py` 指令，对人体姿态进行关键点标注；
- 终端打印识别到的人体数量、推理速度、置信度、耗时等信息

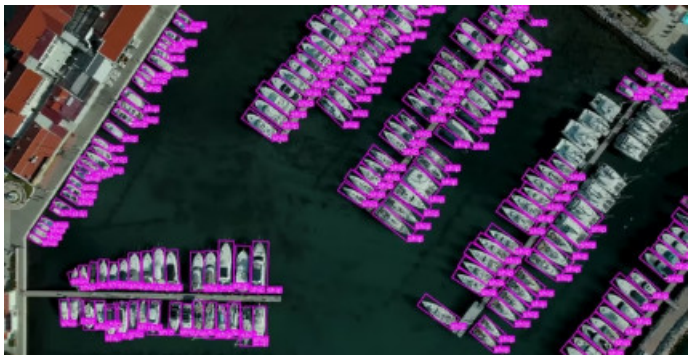
```
(venv) ljl@raspberrypi:~/AI_demo/YOLO26 $ python pos.py

0: 480x640 10 persons, 4683.3ms
Speed: 180.5ms preprocess, 4683.3ms inference, 31.1ms postprocess per image at s
hape (1, 3, 480, 640)
(venv) ljl@raspberrypi:~/AI_demo/YOLO26 $
```

- 弹窗显示姿态估计结果



旋转框检测

定向对象检测通过引入一个额外的角度来更准确地定位图像中的对象，从而比标准对象检测更进一步。



定向目标检测器的输出是一组旋转的边界框，这些边界框精确地包围了图像中的目标，以及每个框的类别标签和置信度分数。

详见：Oriented Bounding Boxes Object Detection .

代码

终端执行 `touch obb.py` 新建程序文件，并添加如下代码

```python
import cv2
from ultralytics import YOLO

model = YOLO('./model/yolo26n-obb.pt')       # 加载模型
img = cv2.imread('./img/parking.jpg')       # 加载图片

results = model(img, imgsz=640, conf=0.25)    # 返回 Boxes
annotated = results[0].plot(labels=False)     # 画框

cv2.imshow("YOLO-obb", annotated)    # 弹窗显示
cv2.waitKey(0)
cv2.destroyAllWindows()
```

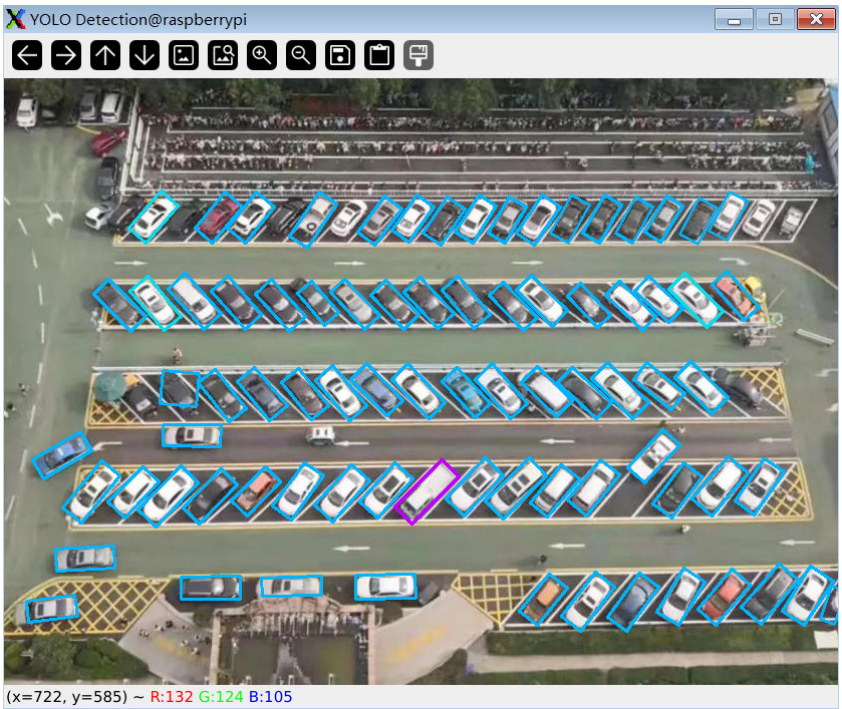
保存代码。

效果

- 终端执行 `python obb.py` 指令，对图片进行目标识别和框选；
- 终端打印识别到的车辆种类、数量、置信度、耗时等信息

```
(venv) ljl@raspberrypi:~/AI_demo/YOLO26 $ python obb.py

0: 480x640 4 ships, 1 large vehicle, 75 small vehicles, 4025.3ms
Speed: 298.5ms preprocess, 4025.3ms inference, 31.4ms postprocess per image at s
hape (1, 3, 480, 640)
```

- 弹窗显示旋转框标注结果



总结

本文介绍了工业树莓派 CM0 NANO 单板计算机结合 OpenCV 和 Ultralytics 库实现 YOLO26 板端部署，并实现目标识别、姿态估计、图像分割、图像分类、旋转框检测的项目设计，包括环境部署、模型获取、关键代码、效果演示等，为相关产品在边缘 AI 领域的快速开发和应用设计提供了参考。

发布链接：https://blog.csdn.net/qq_36654593/article/details/157097310

【工业树莓派 CM0 NANO 单板计算机】MLX90640 热成像仪

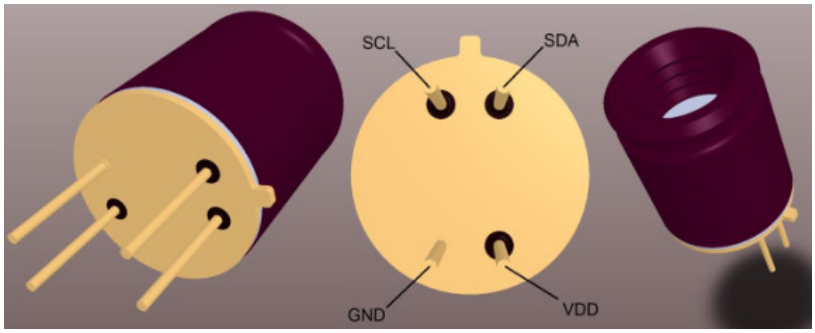
本文介绍了工业树莓派 CM0 NANO 单板计算机结合 MLX90640 热成像传感器模块，实现环境热成像显示的热成像仪项目设计，包括模块设计、硬件连接、环境搭建、通信测试、图像显示和优化等流程。

项目介绍

MLX90640 是一款高分辨率红外热成像阵列传感器，可与 Raspberry Pi 配合使用，实现实时温度可视化。

MLX90640 通过 Python 脚本读取数据并显示热图，调整刷新率和插值优化图像。

MLX90640 可广泛应用于人体检测、安防监控、工业设备温度监控、智能楼宇温控等场景。

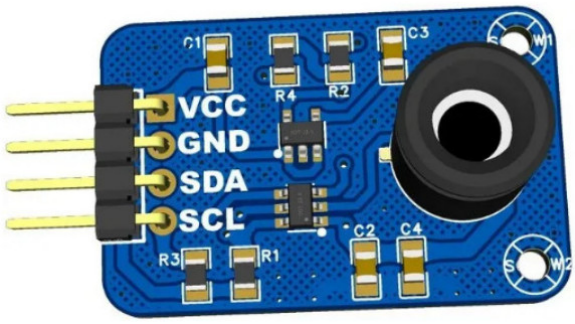


- 硬件连接：使用杜邦线将 MLX90640 模块连接至树莓派 CM0 NANO 的 IIC 接口；
- 环境搭建：创建虚拟环境，安装所需软件包；
- 通信测试：测试硬件连接、设备 IIC 地址、热成像数据打印；
- 图像显示：使用 python 编程和 matplotlib 库函数显示实时图像；
- 平滑优化：结合 matplotlib 库函数实现数据平滑处理以提升显示效果。

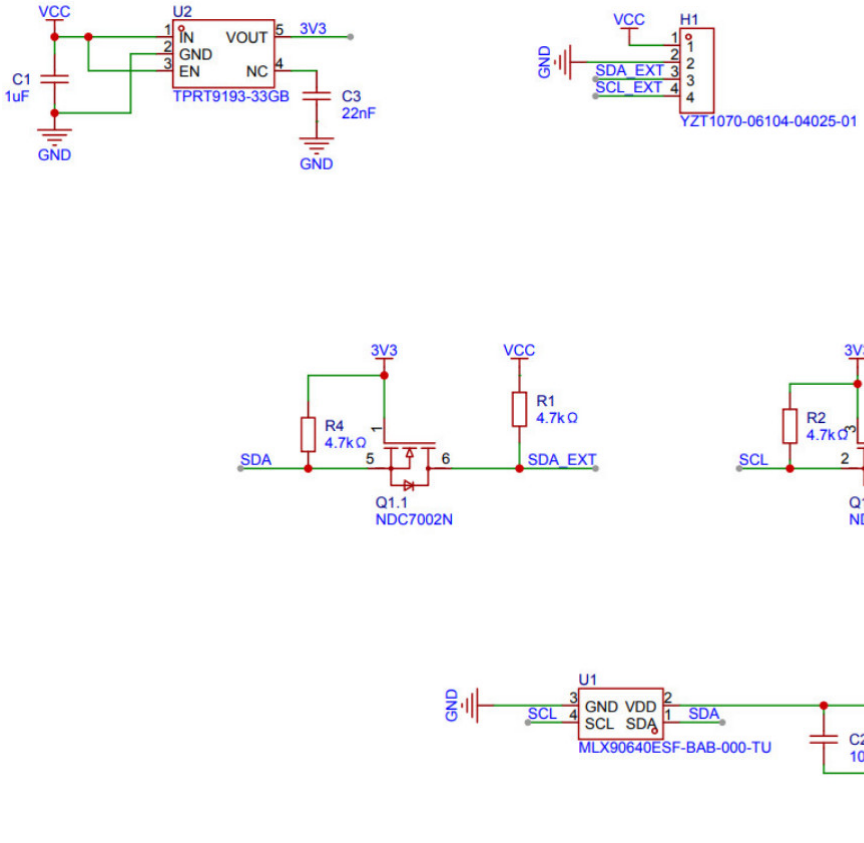
模块设计

为了便于设备连接，设计基于 MLX90640 传感器和 IIC 通信协议的 PCB 模块。

3D 渲染图



原理图



实物图

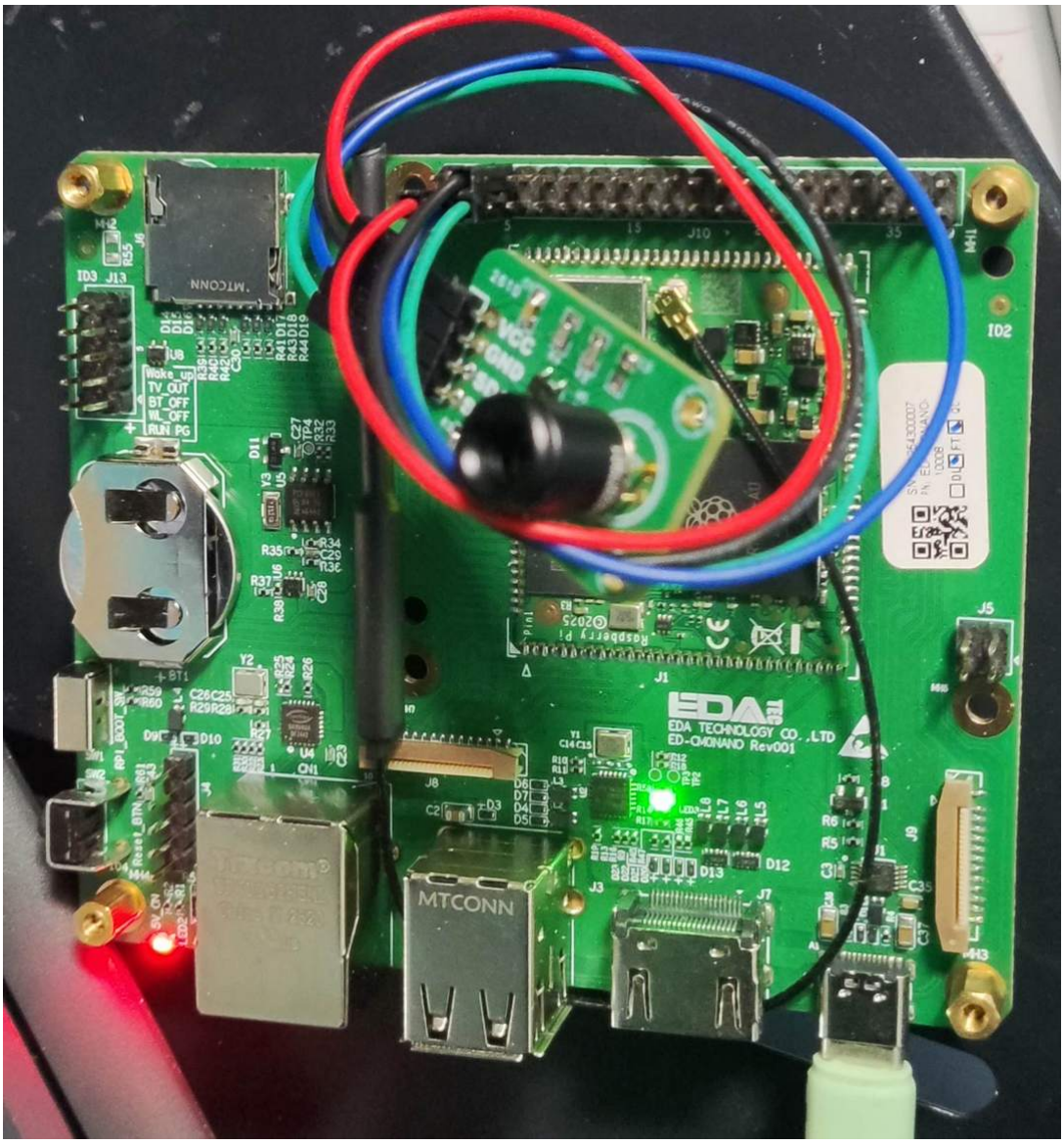
详见：[MLX90640热成像传感器 - 立创开源硬件平台](#) .

硬件连接

- 使用 Type-C 数据线实现设备供电；
- 连接 WiFi 实现网络通信和数据交换；
- 使用杜邦线连接 MLX90640 模块和 CM0 NANO 开发板，接线方式如下

MLX90640	RaspberryPi CM0	Note
SDA	SDA (Pin3)	Serial Data
SCL	SCL (Pin5)	Serial Clock
GND	GND	Ground
VIN	3V3	Power

实物图



环境搭建

创建并激活虚拟环境

```
python -m venv .venv
source .venv/bin/activate
```

安装所需软件包

安装所需软件包

```
pip install numpy
pip install matplotlib
pip install RPi.GPIO adafruit-blinka
pip install adafruit-circuitpython-mlx90640
```

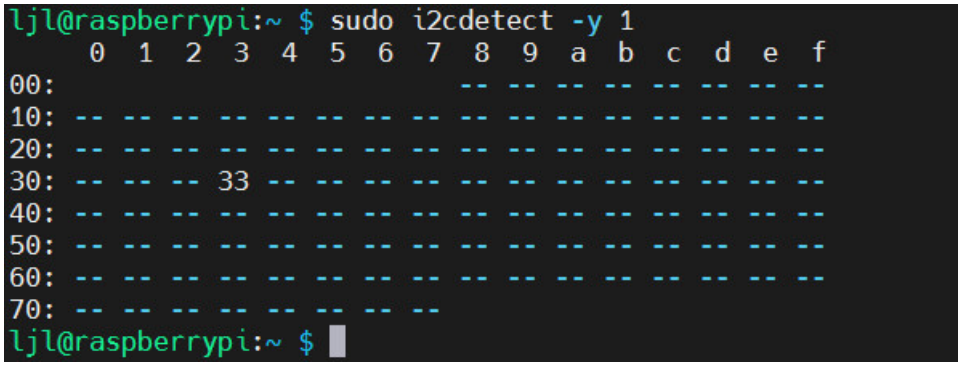
连接测试

终端执行指令

连接测试
终端执行指令

```
sudo i2cdetect -y 1
```

显示 iic 设备地址为 `0x33` 对应设备为 MLX90640



通信测试

终端执行 `mlx90640_print_temp.py` 新建文件，并添加如下代码

```
import time
import board
import busio
import numpy as np
import adafruit_mlx90640

def main():
    # Setup I2C connection
    i2c = busio.I2C(board.SCL, board.SDA, frequency=400000)
    mlx = adafruit_mlx90640.MLX90640(i2c)
    mlx.refresh_rate = adafruit_mlx90640.RefreshRate.REFRESH_2_HZ

    frame = np.zeros((24 * 32,))  # Initialize the array for all 768 temperature readings

    while True:
        try:
            mlx.getFrame(frame)  # Capture frame from MLX90640
            average_temp_c = np.mean(frame)
            average_temp_f = (average_temp_c * 9.0 / 5.0) + 32.0
            print(f"Average MLX90640 Temperature: {average_temp_c:.1f}C ({average_temp_f:.1f}F)")
            time.sleep(0.5)  # Adjust this value based on how frequently you want updates
```

```
22
23          except ValueError as e:
24              print(f"Failed to read temperature, retrying. Error: {str(e)}")
25              time.sleep(0.5)  # Wait a bit before retrying to avoid flooding
    with requests
26          except KeyboardInterrupt:
27              print("Exiting...")
28              break
29          except Exception as e:
30              print(f"An unexpected error occurred: {str(e)}")
31
32  if __name__ == "__main__":
33      main()
```

保存代码。

效果

终端执行指令 `python mlx90640_print.py` 运行代码；

- 终端打印传感器采集画面的平均温度；手掌靠近传感器，温度上升至体温。

```
(.venv) ljl@raspberrypi:~/mlx90640 $ python mlx90640_print_temp.py
/home/ljl/mlx90640/.venv/lib/python3.13/site-packages/adafruit_blinka/microcontrolle
:30: RuntimeWarning: I2C frequency is not settable in python, ignoring!
  warnings.warn(
Average MLX90640 Temperature: 25.5C (77.9F)
Average MLX90640 Temperature: 25.6C (78.2F)
Average MLX90640 Temperature: 25.7C (78.3F)
Average MLX90640 Temperature: 25.5C (77.9F)
Average MLX90640 Temperature: 25.3C (77.6F)
Average MLX90640 Temperature: 25.5C (77.8F)
Average MLX90640 Temperature: 25.6C (78.0F)
Average MLX90640 Temperature: 25.5C (77.9F)
Average MLX90640 Temperature: 25.2C (77.4F)
Average MLX90640 Temperature: 25.3C (77.5F)
Average MLX90640 Temperature: 25.6C (78.1F)
Average MLX90640 Temperature: 25.5C (77.8F)
Average MLX90640 Temperature: 25.5C (77.9F)
Average MLX90640 Temperature: 25.6C (78.1F)
```

若报错则执行下列指令，以定义开发板型号

```
1  export BLINKA_FORCEBOARD=RASPBERRY_PI_ZERO_2_W
2  export BLINKA_FORCECHIP=BCM2XXX
```

图像显示

终端执行 `touch mlx90640_plt.py` 新建文件，添加如下代码

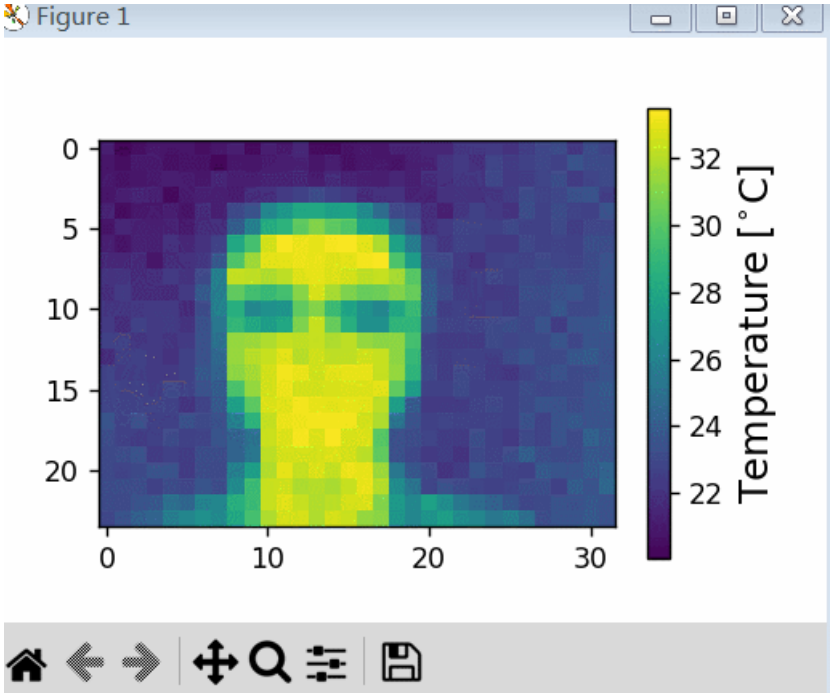
```
1  import time
2  import board
3  import busio
4  import numpy as np
5  import adafruit_mlx90640
6  import matplotlib.pyplot as plt
7
8  i2c = busio.I2C(board.SCL, board.SDA)
```

```
mlx = adafruit_mlx90640.MLX90640(i2c)
mlx.refresh_rate = adafruit_mlx90640.RefreshRate.REFRESH_4_HZ  # Set to a feasible refresh rate

plt.ion()
fig, ax = plt.subplots(figsize=(12, 7))
therm1 = ax.imshow(np.zeros((24, 32)), vmin=0, vmax=60)
cbar = fig.colorbar(therm1)
cbar.set_label(r'Temperature [$^{\circ}$C]', fontsize=14)

frame = np.zeros((24*32,))
t_array = []
max_retries = 5

while True:
    t1 = time.monotonic()
    retry_count = 0
    while retry_count < max_retries:
        try:
            mlx.getFrame(frame)
            data_array = np.reshape(frame, (24, 32))
            therm1.set_data(np.fliplr(data_array))
            therm1.set_clim(vmin=np.min(data_array), vmax=np.max(data_array))
            fig.canvas.draw()  # Redraw the figure to update the plot and colorbar
            fig.canvas.flush_events()
            plt.pause(0.001)
            t_array.append(time.monotonic() - t1)
            print('Sample Rate: {0:2.1f}fps'.format(len(t_array)/np.sum(t_array)))
            break
        except ValueError:
            retry_count += 1
        except RuntimeError as e:
            retry_count += 1
            if retry_count >= max_retries:
                print(f"Failed after {max_retries} retries with error: {e}")
                break
```

保存代码。

效果

- 终端执行指令 `python mlx90640_plt.py` 运行代码；
- 弹窗显示 matplotlib 伪彩图，调整摄像头位置，采集热成像动态画面；



图像优化

为了优化画面显示效果，使用 Matplotlib 内置函数对数据进行平滑处理。

终端执行 `touch mlx90640_plt_smooth.py` 新建文件，添加如下代码

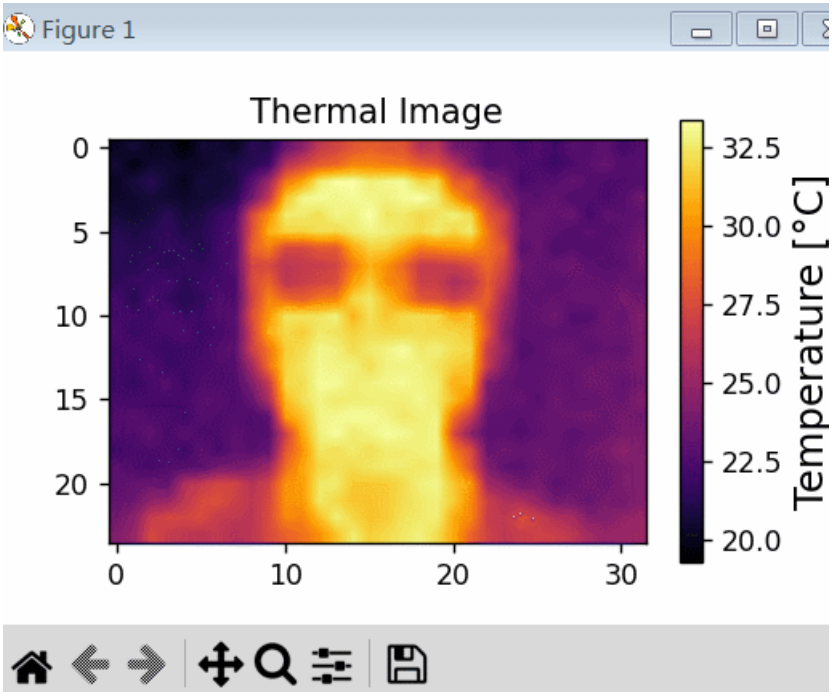
```
import time
import board
import busio
import numpy as np
import adafruit_mlx90640
import matplotlib.pyplot as plt

def initialize_sensor():
    i2c = busio.I2C(board.SCL, board.SDA)
    mlx = adafruit_mlx90640.MLX90640(i2c)
    mlx.refresh_rate = adafruit_mlx90640.RefreshRate.REFRESH_4_HZ
    return mlx

def setup_plot():
    plt.ion()
    fig, ax = plt.subplots(figsize=(12, 7))
    therm1 = ax.imshow(np.zeros((24, 32)), vmin=0, vmax=60, cmap='inferno',
interpolation='bilinear')
    cbar = fig.colorbar(therm1)
    cbar.set_label(r'Temperature [°C]', fontsize=14)
    plt.title('Thermal Image')
    return fig, ax, therm1

def update_display(fig, ax, therm1, data_array):
    therm1.set_data(np.fliplr(data_array))
    therm1.set_clim(vmin=np.min(data_array), vmax=np.max(data_array))
    ax.draw_artist(ax.patch)
    ax.draw_artist(therm1)
    fig.canvas.draw()
    fig.canvas.flush_events()

```

```
def main():
    mlx = initialize_sensor()
    fig, ax, therm1 = setup_plot()

    frame = np.zeros((24*32,))
    t_array = []
    max_retries = 5

    while True:
        t1 = time.monotonic()
        retry_count = 0
        while retry_count < max_retries:
            try:
                mlx.getFrame(frame)
                data_array = np.reshape(frame, (24, 32))
                update_display(fig, ax, therm1, data_array)
                plt.pause(0.001)
                t_array.append(time.monotonic() - t1)
                print('Sample Rate: {0:2.1f}fps'.format(len(t_array) /
np.sum(t_array)))
                break
            except ValueError:
                retry_count += 1
            except RuntimeError as e:
                retry_count += 1
                if retry_count >= max_retries:
                    print(f"Failed after {max_retries} retries with error:
{e}")
                    break

if __name__ == '__main__':
    main()

```

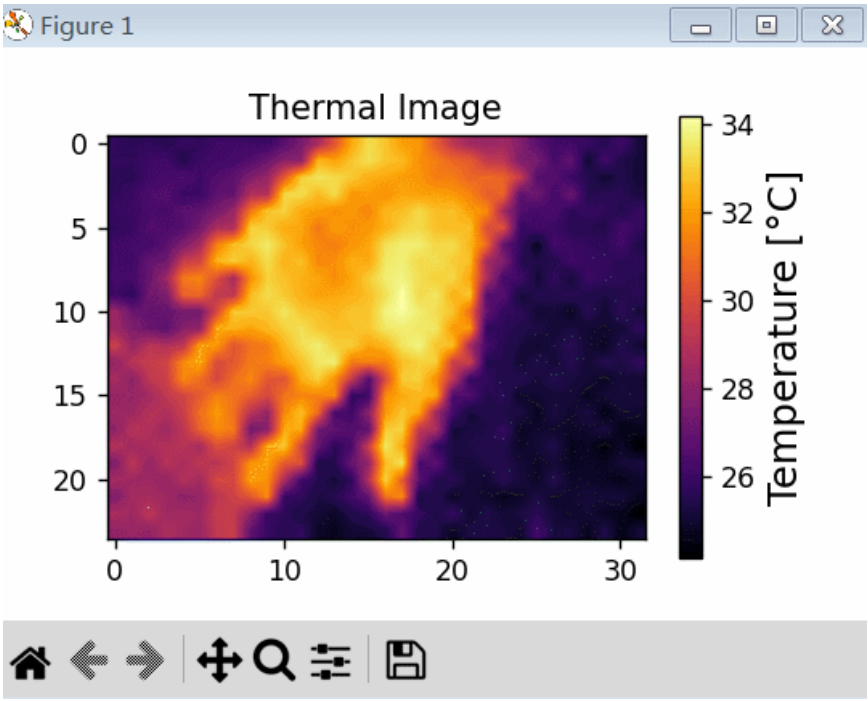
保存代码。

效果

终端执行指令 `python mlx90640_plt_smooth.py` 运行代码



画面清晰度相较于平滑处理前有较大提升。



总结

本文介绍了工业树莓派 CM0 NANO 单板计算机结合 MLX90640 热成像模块，实现环境红外热成像显示的热成像仪项目设计，包括模块设计、硬件连接、环境搭建、通信测试、图像显示和优化等流程，为相关产品的快速开发和应用设计提供了参考。

发布链接：https://blog.csdn.net/qq_36654593/article/details/158931762

【工业树莓派 CM0 NANO 单板计算机】步进电机的远程控制与LabVIEW数据采集

本文介绍了工业树莓派 CM0 NANO 单板计算机结合 TMC2209 模块实现 42 步进电机驱动，通过 MQTT 通信实现远程控制，进一步设计 LabVIEW 上位机，实现步进电机的自动化连续运行和数据采集，包括硬件连接、环境搭建、OLED 显示、流程图、工程代码、效果演示等。

项目介绍

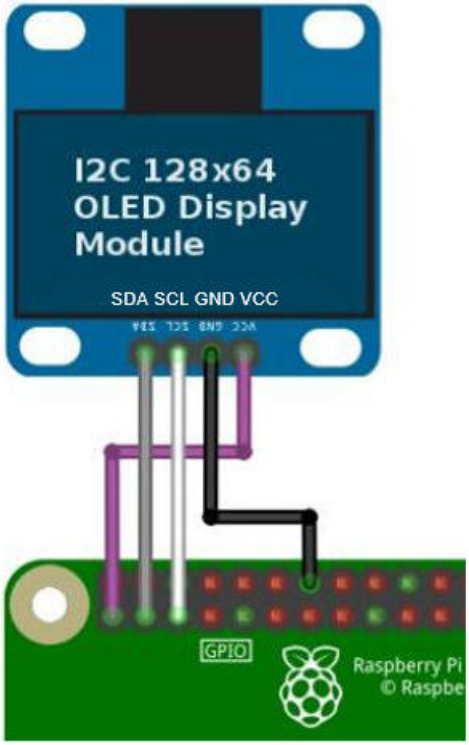
- 准备工作：硬件连接、环境搭建、python 库安装；
- OLED 驱动：获取 RTC 时间、OLED 显示、动图显示等；
- 步进电机驱动：通过 GPIO 控制 TMC2209 驱动步进电机，OLED 显示旋转状态信息；
- 远程控制：结合 MQTT 协议和云端服务器，实现远程控制步进电机；
- 数据采集：设计 LabVIEW 上位机，实现步进电机的连续自动化运行，并模拟采集存储数据。

硬件连接

- 使用 Type-C 数据线连接电源适配器；
- 使用杜邦线连接 CM0 NANO 开发板和 0.96 寸 OLED 模块；
- 使用杜邦线连接 CM0 NANO 和 TMC2209 步进电机驱动模块；
- 连接步进电机和 TMC2209 模块；
- 使用工业电源连接 TMC2209 为步进电机供电。

OLED

OLED 模块使用 SSD1306 驱动，将其与 IIC 总线 1 连接。



实物连接如下

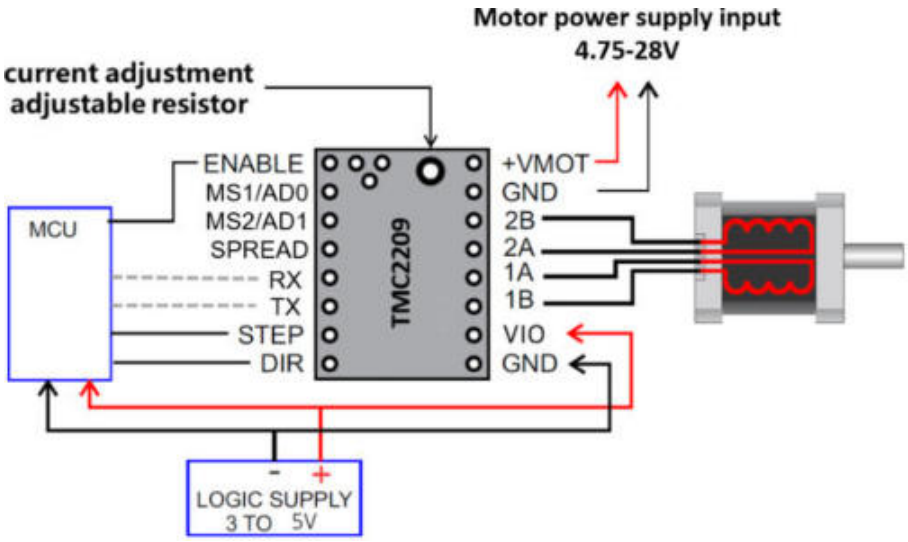
接线方式如下

OLED	CM0 Nano	Note
SCL	SCL	Serial Clock
SDA	SDA	Serial Data
GND	GND	Ground
VCC	3.3V	Power

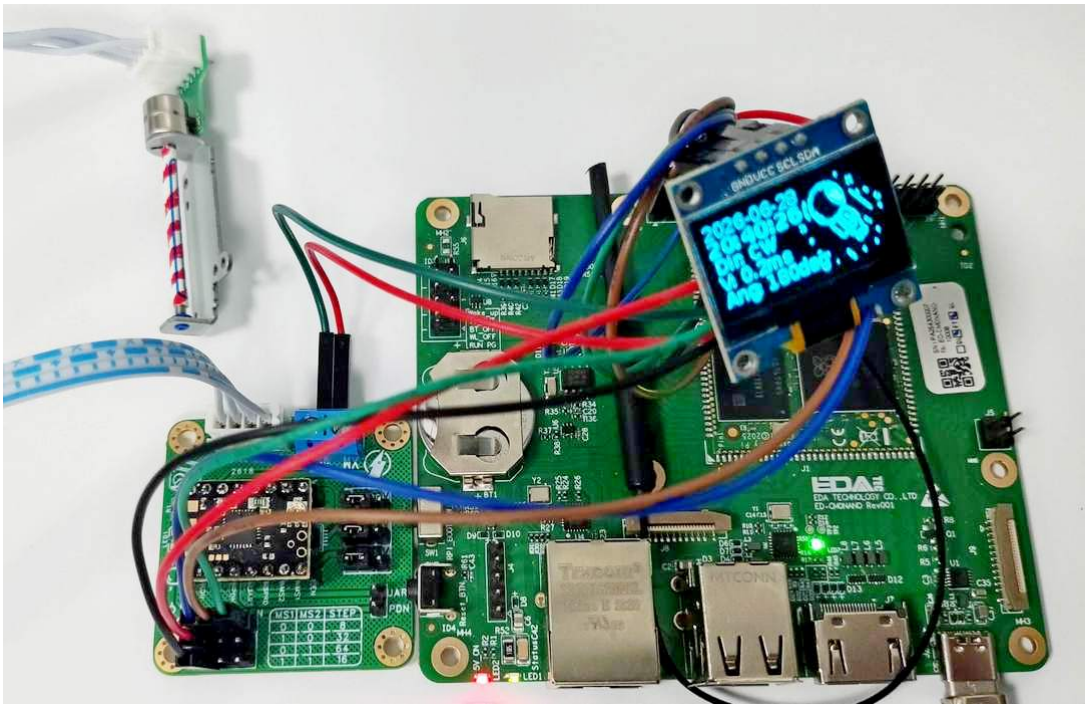
步进电机

步进电机采用 NIDEC 尼得科 8mm 丝杆步进电机，步距角 18 度，详见：[PM型步进电机（轴/螺杆型）| NIDEC（尼得科）](). .

示意图如下



驱动器使用 TMC2209，参数详见：[TMC2209SILENTSTEPSTICK](#) .



接线方式如下

TMC2209	CM0 Nano	Note
Dir	GPIO27	Dirction
STEP	GPIO17	Step
EN	GPIO22	Enable
GND	GND	Ground
VCC	3V3	Power

根据步进电机额定电压选择合适的电源，这里使用板载 5V 排针供电。

环境搭建

这里使用 `luma.oled` 库驱动 OLED 模块。

- 终端执行下列代码，安装 luma.oled 库

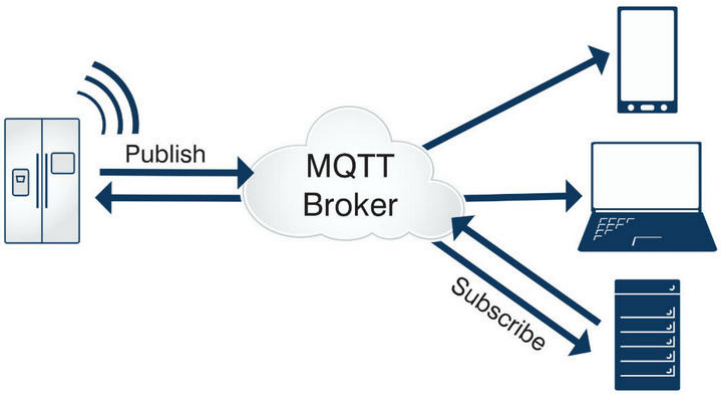
```
pip install luma.oled
```

详见：luma.oled · PyPI .

MQTT Broker

MQTT 是一种适用于物联网的轻量级协议，MQTT 服务器（MQTT Broker）是其核心组件。

MQTT Broker 是一种中介实体，帮助 MQTT 客户端进行通信。作为中央枢纽，MQTT Broker 能够高效管理设备与应用之间的信息流。具体来说，MQTT Broker 接收客户端发布的消息，根据主题对消息进行过滤，并分发给订阅者。



可使用 EMQX 搭建本地 MQTT 服务器，也可采用开发商提供的云端服务器。

这里使用 KZone喵星球 IoT Studio 作为订阅消息的中转平台。

OLED 驱动

制作一款桌面 RTC 时钟，OLED 屏幕显示数字时钟、日期和动画。

代码

终端执行指令 `touch oled_rtc.py` 新建文件，并添加如下代码

```
from luma.core.interface.serial import i2c
from luma.core.render import canvas
from luma.oled.device import ssd1306
from PIL import Image, ImageFont
import os
import time
import datetime
```

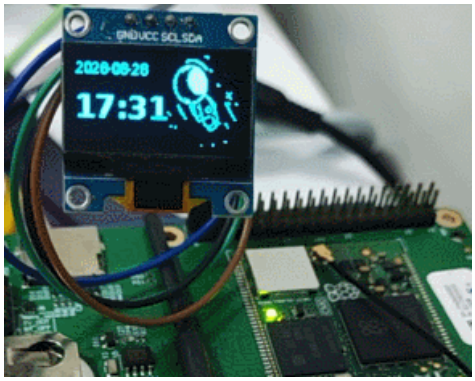
```
8
9  serial = i2c(port=1, address=0x3C)
10 device = ssd1306(serial)
11
12 # 字体
13 font_fg = ImageFont.truetype("/usr/share/fonts/truetype/dejavu/DejaVuSans-
   Bold.ttf", 22)
14 font_date =
   ImageFont.truetype("/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf", 20)
15
16 # 加载20帧
17 frame_list = []
18 for i in range(1, 21):
19     img_path = os.path.join("frame", f"{i}.jpg")
20     img = Image.open(img_path).convert("1")
21     frame_list.append(img)
22
23 frame_index = 0
24
25 try:
26     while True:
27         now = datetime.datetime.now()
28         date_str = now.strftime("%Y-%m-%d")
29         time_str = now.strftime("%H:%M")
30
31         with canvas(device) as draw:
32             # 动画
33             draw.bitmap((32, 0), frame_list[frame_index], fill=1)
34             # 时间
35             draw.text((2, 25), time_str, font=font_fg, fill=1)
36             # 日期
37             draw.text((2, 5), date_str, fill=1)
38
39         frame_index = (frame_index + 1) % 20
40         time.sleep(0.1)
41
42 except KeyboardInterrupt:
43     device.clear()
```

保存代码。

> 动画包含的帧画面可自定义，通过 GIF 或视频格式提取帧图片，按照顺序数字命名，保存至 frame 文件夹。

效果

- 终端执行指令 `python rtc_oled.py` 运行程序；
- OLED 显示日期、时间以及太空人动画；



步进电机驱动

使用树莓派 CM0 NANO 板载 40pin 引脚接口，控制 GPIO 输出电平，结合 TMC2209 实现 42 步进电机驱动，并显示旋转状态，包括转速、角度和方向。

代码

终端执行指令 `touch stepmotor_oled.py` 新建文件，并添加如下代码

```python
from luma.core.interface.serial import i2c
from luma.core.render import canvas
from luma.oled.device import ssd1306
from PIL import Image, ImageFont
import os
import time
import datetime
import threading
import RPi.GPIO as GPIO

# ---------------------- OLED 初始化 ----------------------
serial = i2c(port=1, address=0x3C)
device = ssd1306(serial)

# 字体
font_main = ImageFont.truetype("/usr/share/fonts/truetype/dejavu/DejaVuSans-Bold.ttf", 14)
font_small = ImageFont.truetype("/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf", 12)

# 加载帧动画
frame_list = []
for i in range(1, 21):
    img = Image.open(os.path.join("frame", f"{i}.jpg")).convert("1")
    frame_list.append(img)
frame_index = 0

# ---------------------- 运行状态变量 ----------------------
current_angle = 180
speed_ms = 0.2
direct_text = "CW"
running = True

# 独立屏幕刷新线程（动画持续运行，不受电机阻塞）
```

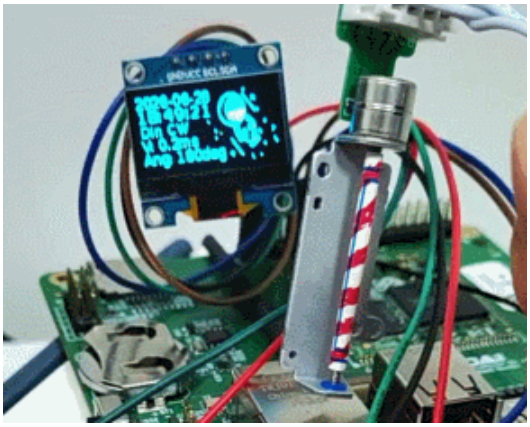
```python
def screen_task():
    global frame_index
    while running:
        now = datetime.datetime.now()
        date_str = now.strftime("%Y-%m-%d")
        time_str = now.strftime("%H:%M:%S")

        with canvas(device) as draw:
            # 太空人动画显示
            draw.bitmap((32, 0), frame_list[frame_index], fill=1)

            # 左侧时间信息
            draw.text((0, 0), date_str, font=font_small, fill=1)
            draw.text((0, 11), time_str, font=font_main, fill=1)

            # 左侧电机运行状态
            info_dir = f"Dir: {direct_text}"
            info_speed = f"V: {speed_ms}ms"
            info_angle = f"Ang: {abs(current_angle)}deg"
            draw.text((2, 24), info_dir, font=font_small, fill=1)
            draw.text((2, 36), info_speed, font=font_small, fill=1)
            draw.text((2, 48), info_angle, font=font_small, fill=1)

        frame_index = (frame_index + 1) % len(frame_list)
        time.sleep(0.1)

# ---------------------- 电机驱动函数 ----------------------
STEP = 17
DIR = 27
EN = 22
PULSE_PER_360DEG = 1600  # 8 细分
PULSE_PER_DEG = PULSE_PER_360DEG / 360

GPIO.setmode(GPIO.BCM)
GPIO.setup(STEP, GPIO.OUT)
GPIO.setup(DIR, GPIO.OUT)
GPIO.setup(EN, GPIO.OUT)
GPIO.output(EN, GPIO.LOW)

def run_angle(degree, speed):
    global direct_text
    pulses = int(abs(degree) * PULSE_PER_DEG)
    if degree > 0:
        GPIO.output(DIR, GPIO.HIGH)
        direct_text = "CW"
    else:
        GPIO.output(DIR, GPIO.LOW)
        direct_text = "CCW"

    delay_s = speed / 1000.0
    # 纯脉冲，无阻塞，转速稳定
    for _ in range(pulses):
        GPIO.output(STEP, GPIO.HIGH)
        time.sleep(delay_s)
        GPIO.output(STEP, GPIO.LOW)
        time.sleep(delay_s)

if __name__ == "__main__":
```

```
    # 启动后台屏幕线程
    t = threading.Thread(target=screen_task, daemon=True)
    t.start()

    try:
        while True:
            current_angle = 180
            run_angle(current_angle, speed_ms)
            time.sleep(1)

            current_angle = -180
            run_angle(current_angle, speed_ms)
            time.sleep(1)

    except KeyboardInterrupt:
        running = False
        time.sleep(0.2)
        device.clear()
        GPIO.output(EN, GPIO.HIGH)
        GPIO.cleanup()

```

保存代码。

效果

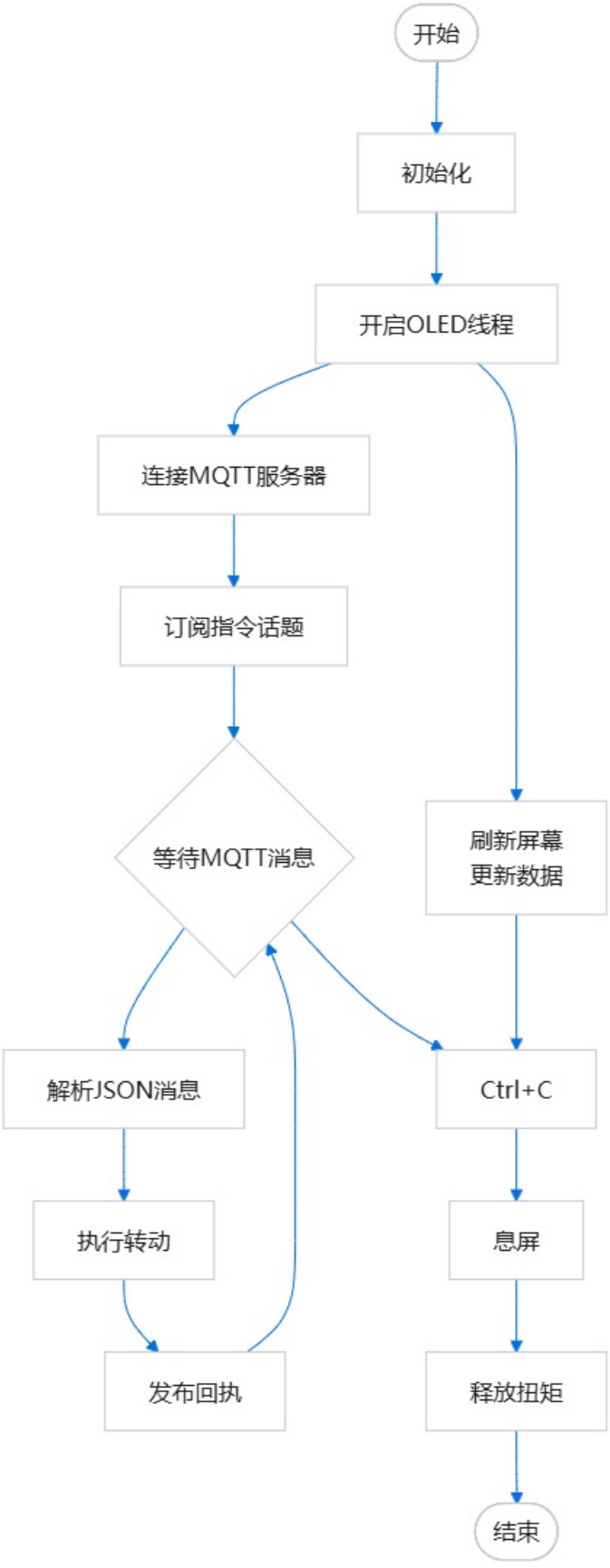
- 终端执行指令 `python stepmotor_oled.py` 运行程序；
- OLED 显示日期、时间、太空人动画、步进电机状态，包括方向、速度、角度等；
- 步进电机正反交替旋转；



远程控制

树莓派 CM0 NANO 连接网络，通过 MQTT 协议实现云端消息传输，进而控制终端步进电机旋转。

流程图


开始
初始化
开启OLED线程
连接MQTT服务器
订阅指令话题
等待MQTT消息
刷新屏幕
更新数据
解析JSON消息
Ctrl+C
执行转动
息屏
发布回执
释放扭矩
结束

代码

终端执行指令 `touch stepmotor_mqtt.py` 新建文件，并添加如下代码

```python
from luma.core.interface.serial import i2c
from luma.core.render import canvas
from luma.oled.device import ssd1306
from PIL import Image, ImageFont
import os
import time
import datetime
import threading
import json
import RPi.GPIO as GPIO
import paho.mqtt.client as mqtt

# ====================== 全局变量 ======================
current_angle = 0
speed_ms = 0.2
direct_text = "STOP"
running_flag = True
mqtt_client = None

# ====================== OLED 初始化 ======================
serial = i2c(port=1, address=0x3C)
device = ssd1306(serial)

font_main = ImageFont.truetype("/usr/share/fonts/truetype/dejavu/DejaVuSans-Bold.ttf", 14)
font_small = ImageFont.truetype("/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf", 12)

# 加载帧画面
frame_list = []
for i in range(1, 21):
    img = Image.open(os.path.join("frame", f"{i}.jpg")).convert("1")
    frame_list.append(img)
frame_index = 0

def screen_task():
    """OLED后台刷新线程"""
    global frame_index
    while running_flag:
        now = datetime.datetime.now()
        date_str = now.strftime("%Y-%m-%d")
        time_str = now.strftime("%H:%M:%S")

        with canvas(device) as draw:
            # 太空人动画
            draw.bitmap((32, 0), frame_list[frame_index], fill=1)
            # 日期时间
            draw.text((2, 0), date_str, font=font_small, fill=1)
            draw.text((2, 10), time_str, font=font_main, fill=1)
            # 运行状态
            draw.text((2, 24), f"Dir: {direct_text}", font=font_small, fill=1)
```

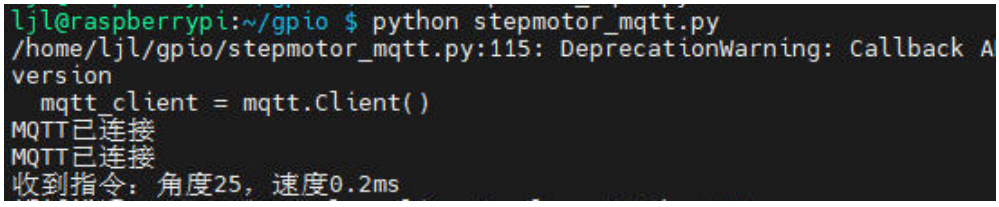
```
            draw.text((2, 36), f"V: {speed_ms}ms", font=font_small, fill=1)
            draw.text((2, 48), f"Ang: {current_angle}deg", font=font_small, fill=1)

        frame_index = (frame_index + 1) % len(frame_list)
        time.sleep(0.07)

# ====================== 电机配置 ======================
STEP = 17
DIR = 27
EN = 22
PULSE_PER_360DEG = 1600  # 8细分
PULSE_PER_DEG = PULSE_PER_360DEG / 360

GPIO.setmode(GPIO.BCM)
GPIO.setup(STEP, GPIO.OUT)
GPIO.setup(DIR, GPIO.OUT)
GPIO.setup(EN, GPIO.OUT)
GPIO.output(EN, GPIO.LOW)

def run_angle(degree, speed):
    global direct_text, current_angle
    pulses = int(abs(degree) * PULSE_PER_DEG)
    current_angle = degree
    if degree > 0:
        GPIO.output(DIR, GPIO.HIGH)
        direct_text = "CW"
    else:
        GPIO.output(DIR, GPIO.LOW)
        direct_text = "CCW"

    delay_s = speed / 1000.0
    for _ in range(pulses):
        GPIO.output(STEP, GPIO.HIGH)
        time.sleep(delay_s)
        GPIO.output(STEP, GPIO.LOW)
        time.sleep(delay_s)
    direct_text = "STOP"

# ====================== MQTT 回调 ======================
def on_connect(client, userdata, flags, rc):
    print("MQTT已连接")
    client.subscribe("ljl/stepmotor")

def on_message(client, userdata, msg):
    global speed_ms
    try:
        payload = json.loads(msg.payload.decode())
        angle = payload.get("angle", 0)
        speed_ms = payload.get("speed", 0.2)
        print(f"收到指令: 角度 {angle}, 速度 {speed_ms}ms")

        # 执行旋转
        run_angle(angle, speed_ms)
        # 旋转完成回复OK
        client.publish("ljl/stepmotor", '{"result":"OK"}')
    except Exception as e:
```

```
        print("解析错误:", e)

# ====================== 主程序 ======================
if __name__ == "__main__":
    # 启动OLED线程
    t_screen = threading.Thread(target=screen_task, daemon=True)
    t_screen.start()

    # MQTT客户端
    mqtt_client = mqtt.Client()
    mqtt_client.on_connect = on_connect
    mqtt_client.on_message = on_message
    mqtt_client.connect("iot.kittenbot.cn", 1883, 60)

    try:
        mqtt_client.loop_forever()
    except KeyboardInterrupt:
        running_flag = False
        time.sleep(0.2)
        device.clear()
        GPIO.output(EN, GPIO.HIGH)
        GPIO.cleanup()
```

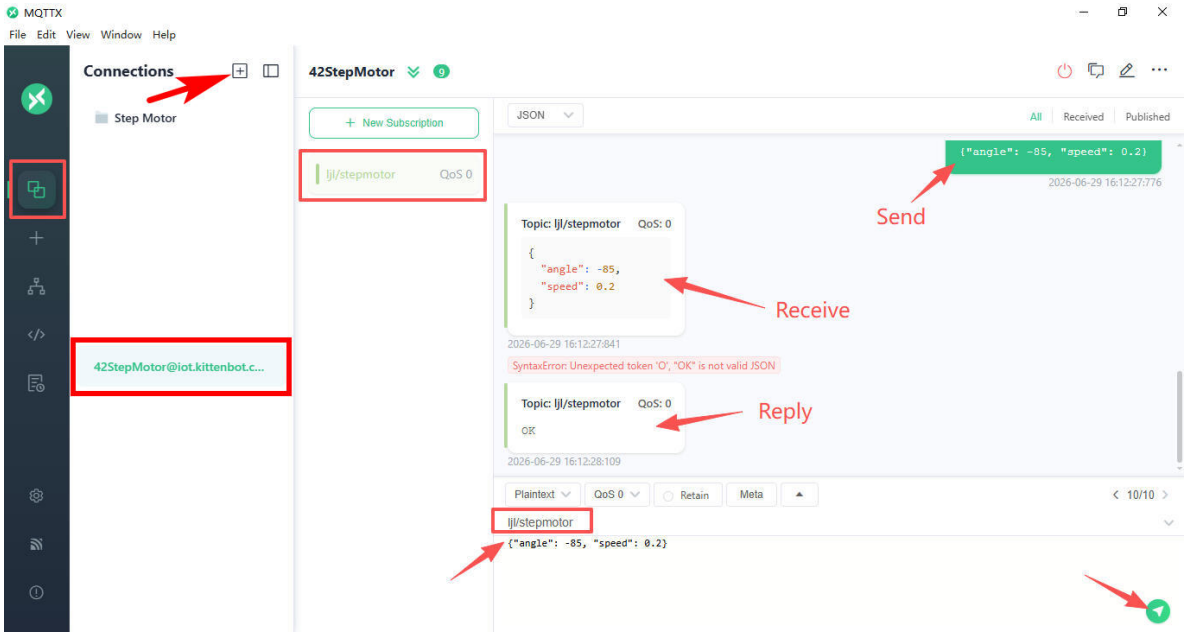
保存代码。

效果

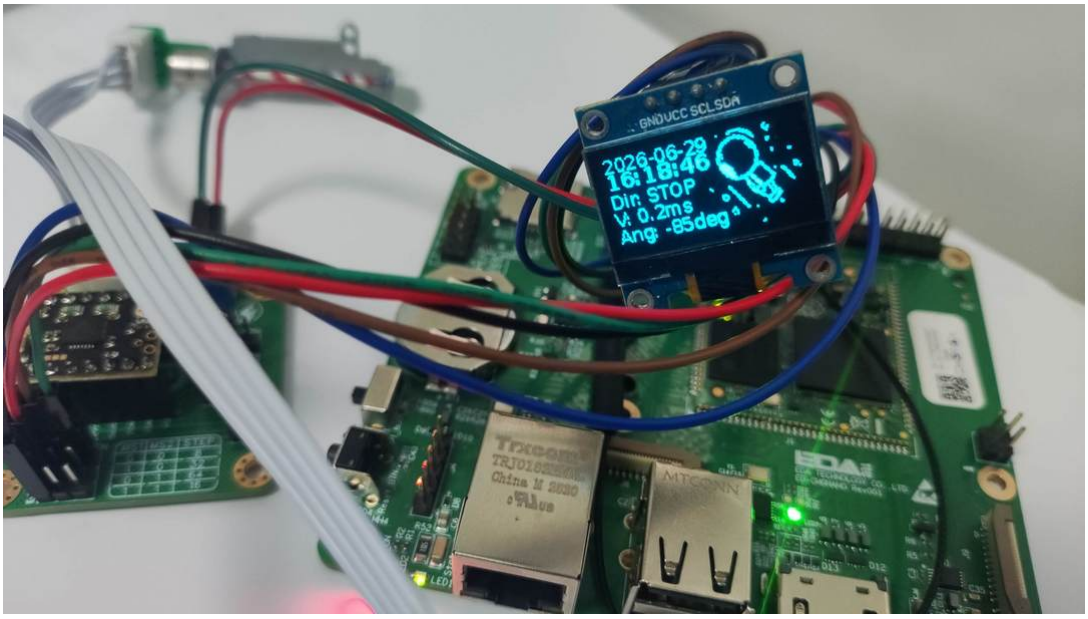
- 终端执行指令 `python stepmotor_mqtt.py` 运行程序；
- OLED 显示日期、时间、太空人动画、步进电机状态，包括方向、速度、角度等；
- 终端打印 MQTT 连接状态、指令接收、电机运转等信息；



- 运行 MQTTX 软件，新建连接，输入 MQTT 服务器地址 `iot.kittenbot.cn` ；
- 新建订阅话题 `ljl/stepmotor` ；
- 以 `ljl/stepmotor` 为 topic 发送 JSON 指令 `{"angle": -95, "speed": 0.2}` 旋转目标角度；

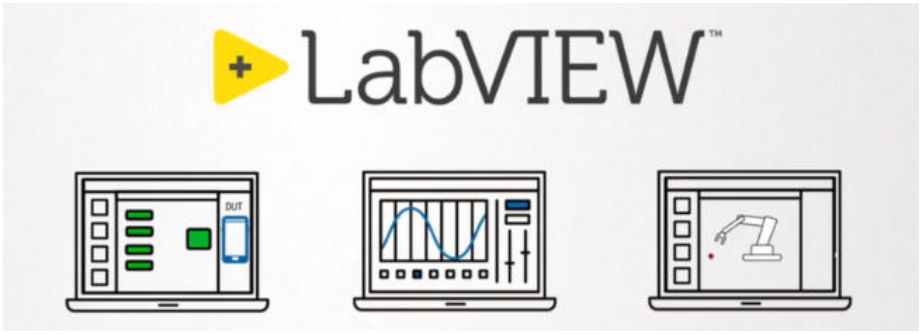


- 步进电机旋转，OLED 显示更新；



LabVIEW 上位机

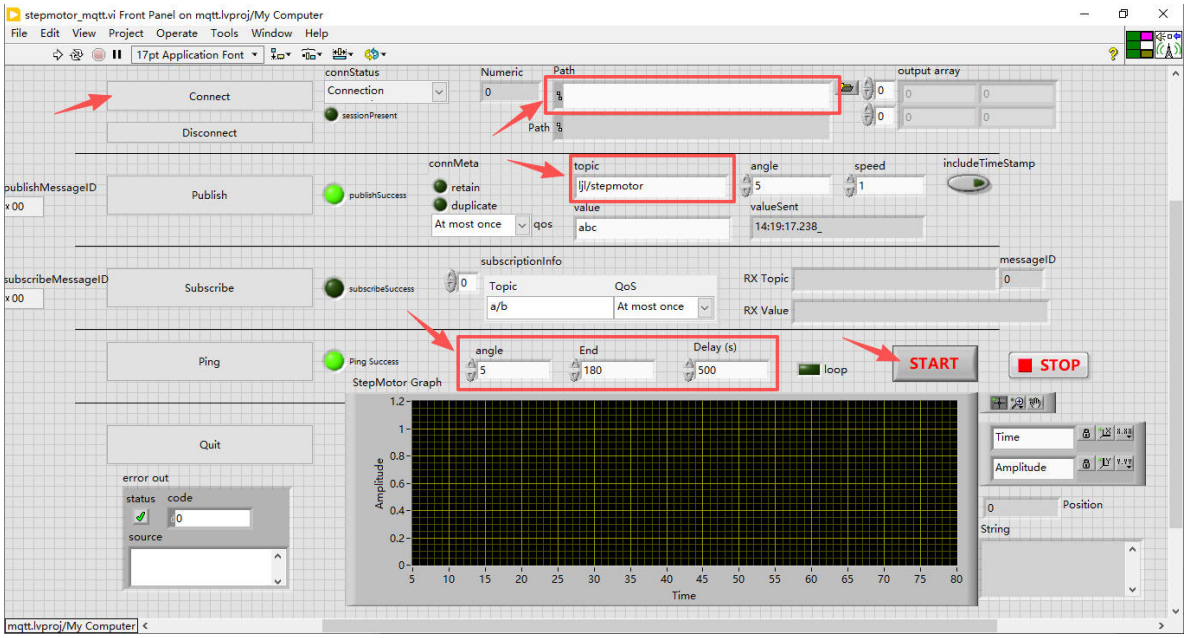
在工业和科研等实际应用场景，往往需要面对较长时间的步进电机控制和数据采集任务，手动发送消息的方案已无法满足实验需求。



使用 LabVIEW 快速搭建逻辑框架，通过 MQTT 协议定时发送消息，实现步进电机的远程自动化控制、仪器联合调用、数据采集与存储等复杂综合性项目的快速落地。

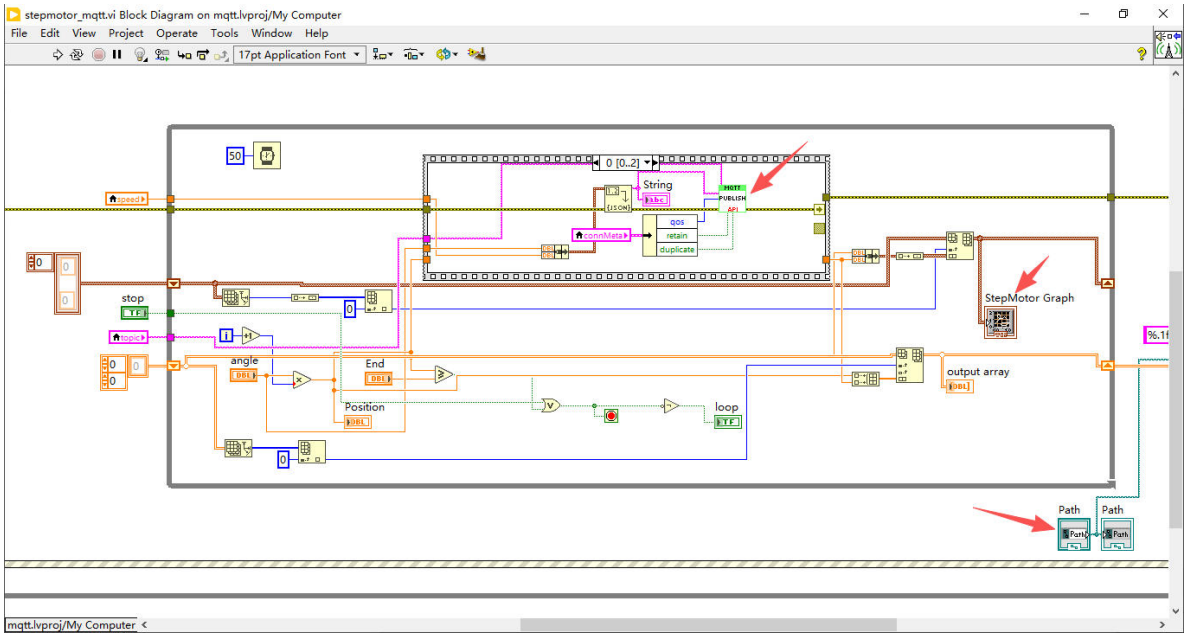
前面板

前面板设计包括 MQTT 配置、单步测试、连续运行测试、实时演化曲线、数据保存、程序控制等模块。



程序面板

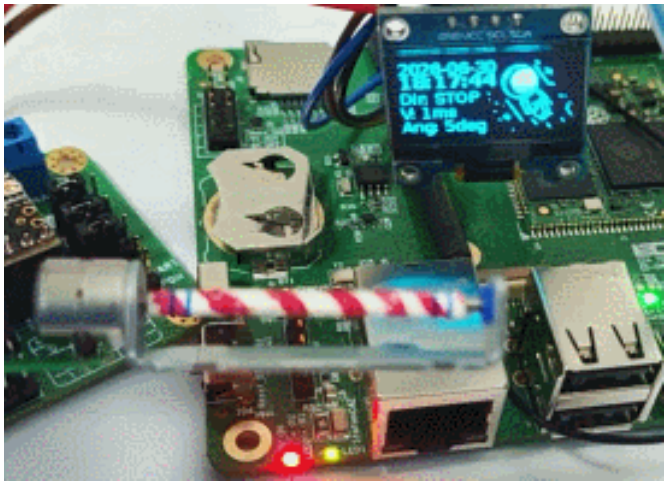
程序面板采用模块化设计，将 MQTT 消息封装，确保连续发送指令准确执行。



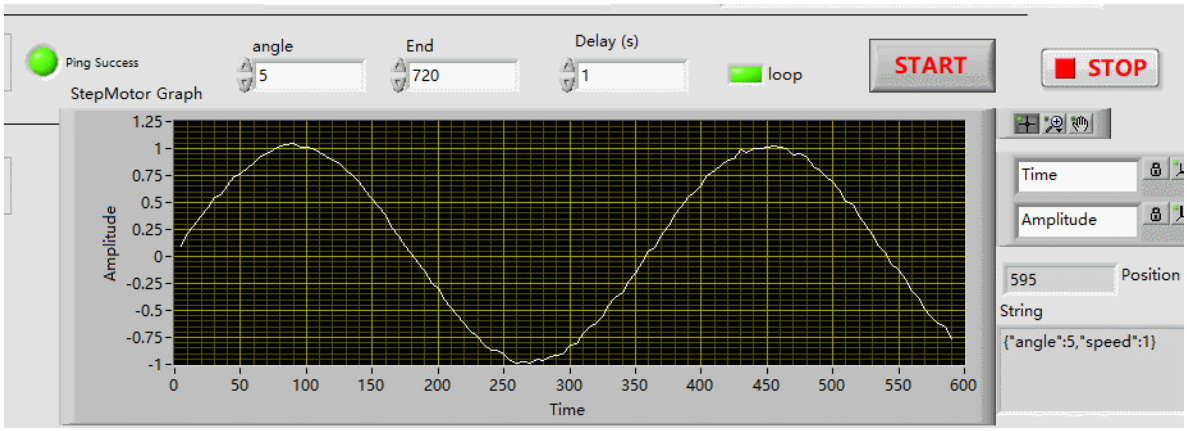
运行效果

- 配置 MQTT 服务器地址，运行程序；
- 设置主题、步长、目标角度、延时、存储路径等；
- 点击 START 按钮，开始运行步进电机并采集数据；

LabVIEW 程序运行后，步进电机按照设定的步长和延时连续旋转；



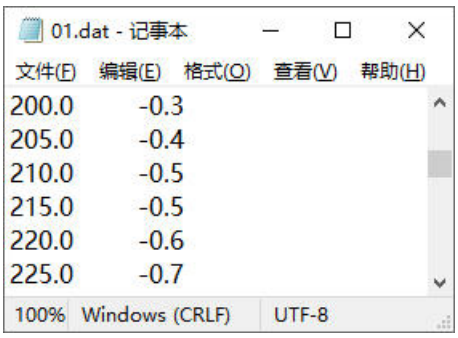
前面板显示采集数据的实时演化曲线；



数据采集

- 循环采集完成后，MQTT 停止发送消息，数据自动存储至目标路径；
- 根据前面板设置的存储路径，找到目标文件；

- 打开存储的数据文件，第一列为角度，第二列为模拟采集的数值；



总结

本文介绍了工业树莓派 CM0 NANO 单板计算机结合 TMC2209 模块实现 42 步进电机驱动，通过 MQTT 通信实现远程控制，进一步设计 LabVIEW 上位机，实现步进电机的自动化连续运行和数据采集，包括硬件连接、环境搭建、OLED 显示、流程图、工程代码、效果演示等，为相关产品在工业物联网领域的快速开发和应用设计提供了参考。

发布链接：https://blog.csdn.net/qq_36654593/article/details/162462722

视频介绍

投稿 B 站视频链接： https://www.bilibili.com/video/BV1TLTY6oEUp/

作者联系方式：lijinlei0907@163.com