

---

# 实验九 手势识别

---

**RT-AK 教育套件实验手册**

上海睿赛德电子科技有限公司 版权所有 @2021



**WWW.RT-THREAD.ORG**

**Tuesday 23<sup>rd</sup> November, 2021**

# 目录

目录	i
1 实验简介和目的	1
1.1 实验简介	1
1.2 实验目的	1
2 实验器材	2
3 实验步骤	3
3.1 AI 模型准备	3
3.2 模型部署	5
3.3 嵌入式 AI 模型应用	7
3.3.1 应对内存限制方法	7
3.3.2 代码流程	8
3.3.3 核心代码说明	9
4 编译烧录	11
4.1 编译	11
4.2 烧录	11
5 实验现象	12
6 附件	14
6.1 嵌入式 AI 开发 API 文档	14
6.2 LCD API 说明手册	15
6.3 Camera 使用	17
查找设备	17
初始化设备	17

打开和关闭设备 . . . . .	18
读写设备 . . . . .	19
数据收发回调 . . . . .	19

# 第 1 章

## 实验简介和目的

### 1.1 实验简介

感知手的形状和动作的能力是在各种技术领域和平台上改善用户体验的重要组成部分。

例如，它可以构成手语理解和手势控制的基础，并且还可以在增强现实中将数字内容和信息覆盖在物理世界之上。

该神经网络模型是一个回归类模型（即直接坐标预测），可以检测到照片内的 21 个 3D 手关节坐标，并进行关键点定位。

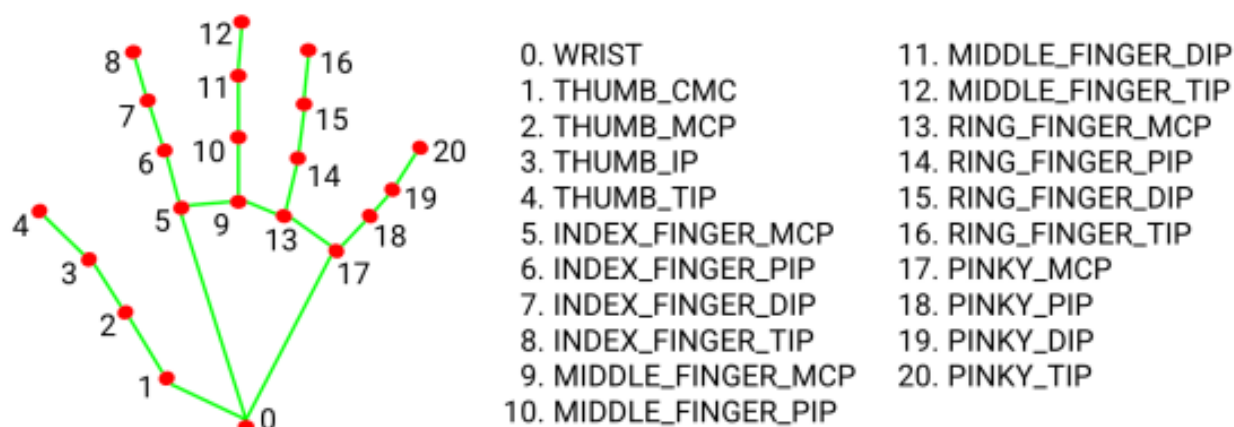


图 1.1: 21 个关键点

### 1.2 实验目的

1. 掌握 RT-AK 一行命令部署 AI 模型的使用
2. 完成嵌入式 AI 开发：输入一张照片，模型正常运行
3. 掌握 LCD 的使用：在 LCD 上有图片和模型的输出显示

## 第 2 章

# 实验器材

1. 上位机（电脑）
2. EgdeAI 实验板
3. LCD 驱动 IC 为 ILI9341

## 第 3 章

# 实验步骤

### 3.1 AI 模型准备

该节实验从训练好的模型开始，跳过训练部分。

模型是来源于 Google 的 MediaPipe 项目中的 Hands 项目。已经提前下载好，在附件的 Models 文件夹中。

- 项目介绍: <https://google.github.io/mediapipe/solutions/hands>
- 模型下载: [https://github.com/google/mediapipe/blob/master/mediapipe/modules/hand\\_landmark/hand\\_landmark.tflite](https://github.com/google/mediapipe/blob/master/mediapipe/modules/hand_landmark/hand_landmark.tflite)

模型信息:

- 输入: [height, width, channel] → 224x224x3, float32
- 三个输出

#### 1. MULTI\_HAND\_LANDMARKS, shape: [1, 63]

检测/跟踪手的集合，其中每只手表示为 21 个手部标志的列表，每个标志由  $x$ 、 $y$  和  $z$  组成。 $x$  和  $y$  分别  $[0.0, 1.0]$  由图像宽度和高度归一化。 $z$  表示以手腕深度为原点的地标深度，值越小，地标离相机越近。的幅度  $z$  用途大致相同的比例  $x$ 。该实验仅用到了  $x$ 、 $y$

#### 2. Score, shape: [1, 1]

预测的手的估计概率

#### 3. Label, shape: [1, 1]

检测左手还是右手，本实验没有用到

使用 Netron 查看网络模型结构: (篇幅有限，节选了输入和输出部分)

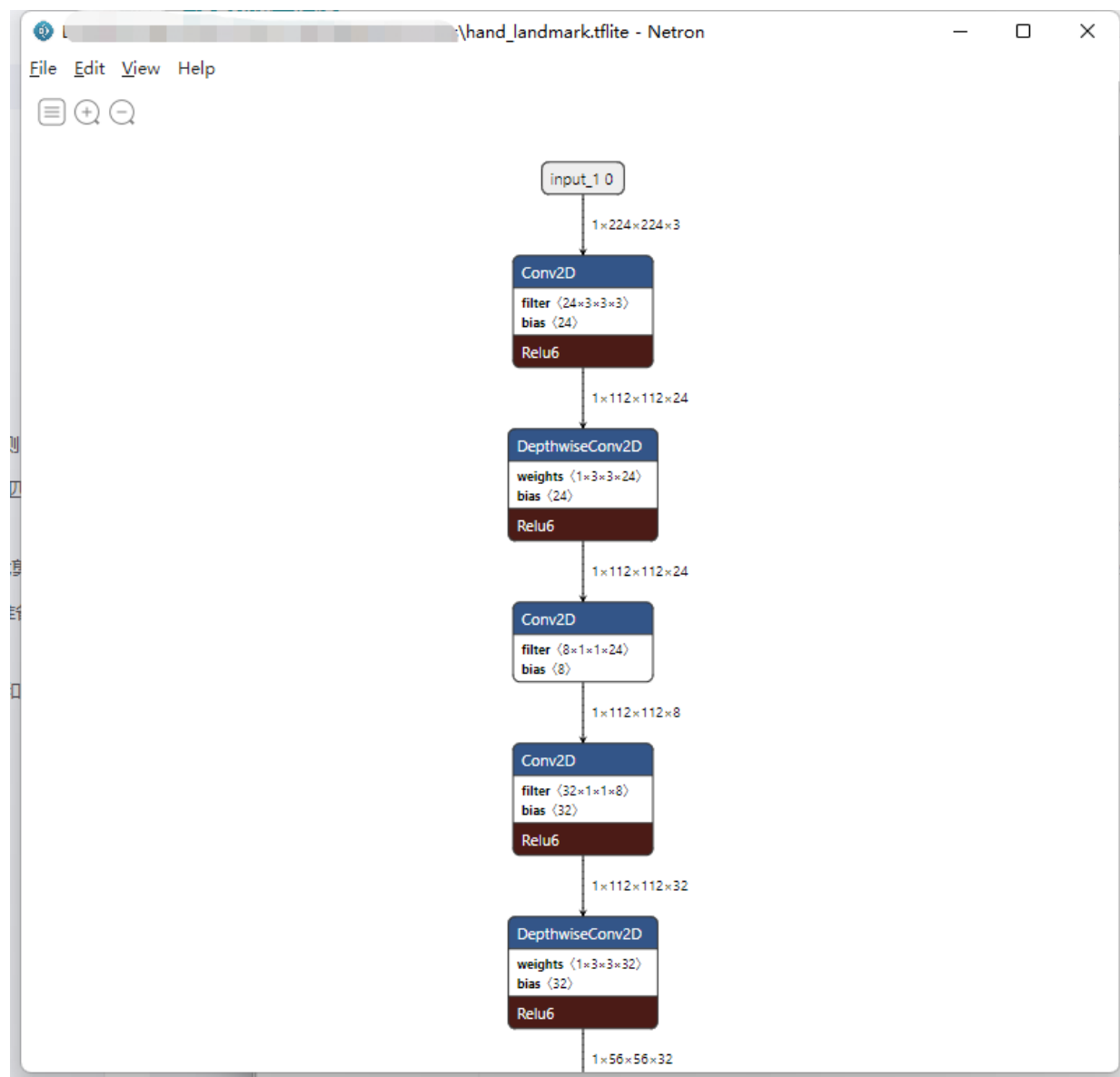


图 3.1: 神经网络模型

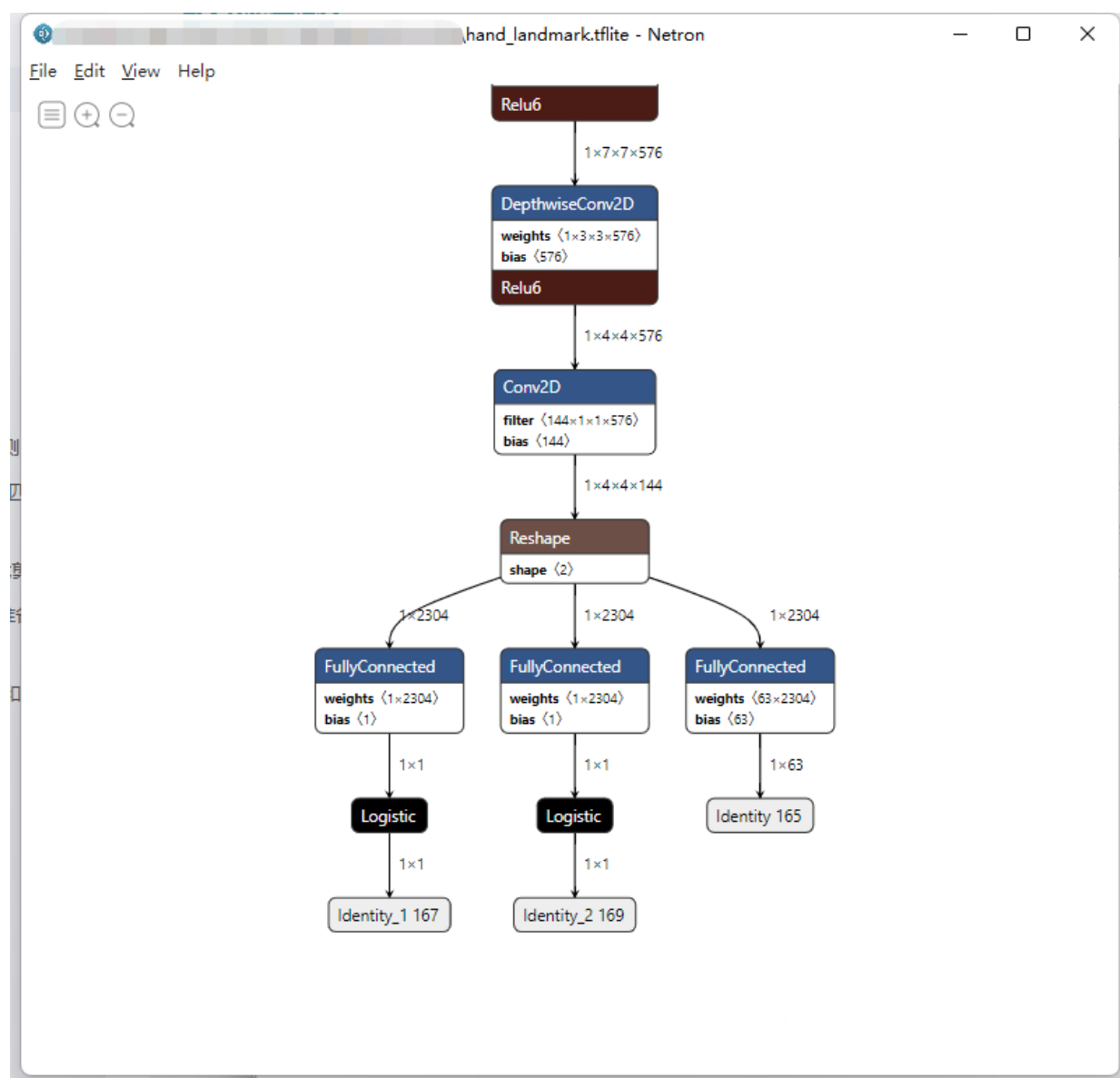


图 3.2: 神经网络模型

tflite 模型的推理代码位于附件的 `Utils/tflite_inference.py`,

运行:

```
python tflite_inference.py
```

## 3.2 模型部署

在 `RT-AK/rt_ai_tools` 路径下打开 Windows 终端



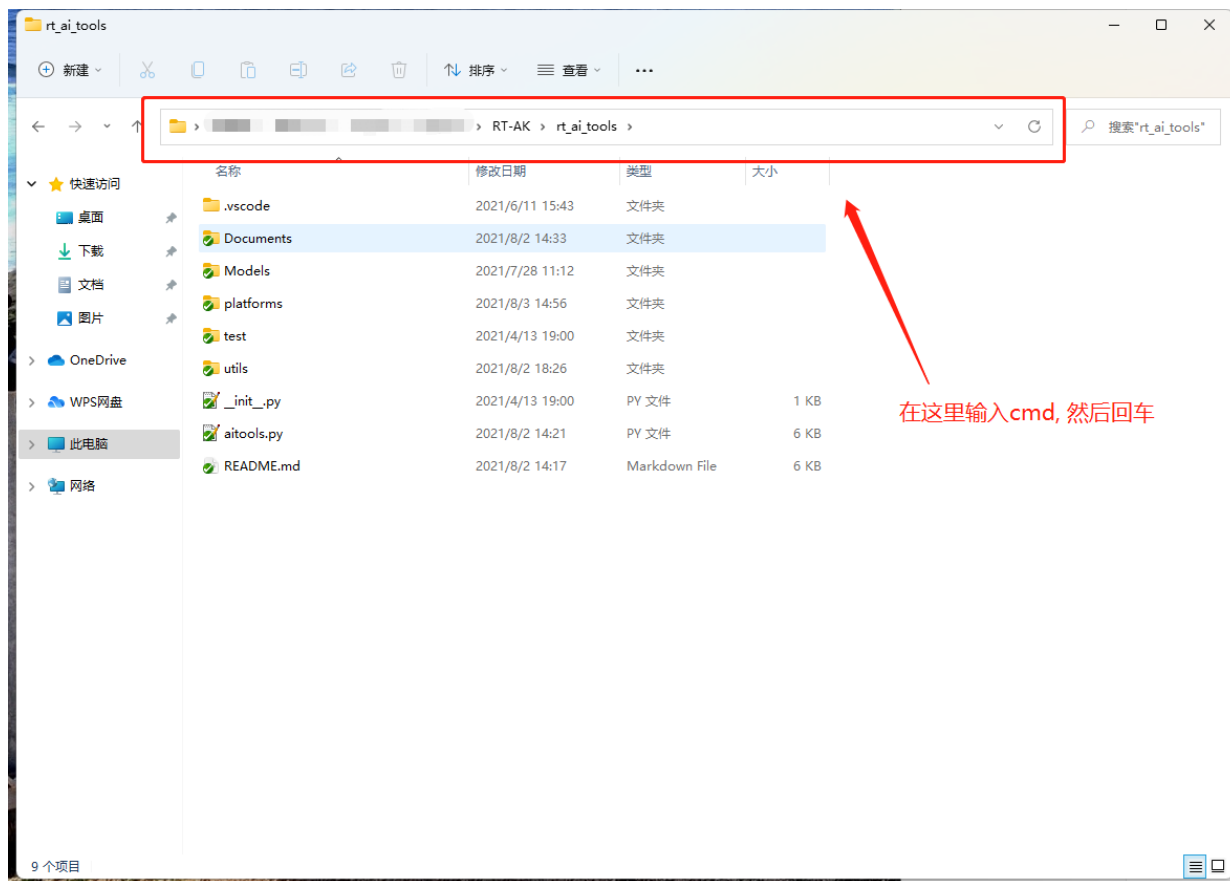


图 3.3: 打开终端

输入以下命令:

```
# 量化为 uint8, 使用 KPU 加速, 量化数据集为图片
$ python aitools.py --project=<your_project_path> --model=<your_model_path> --
  model_name=hand_landmark --platform=k210 --dataset=<your_val_dataset>
```

示例:

```
# 示例(量化模型, 图片数据集)
$ python aitools.py --project="D:\Project\K210_Demo\k210-test" --model="
  hand_landmark.tflite" --model_name=hand_landmark --platform=k210 --dataset="xxx\
  Datasets\Hands_quantize_datasets"
```

其中, `--project` 是你的目标工程路径, `--model` 是你的模型路径, `--model_name` 是转化的模型文件名, `--platform` 是指定插件支持的目标平台为 K210, `--dataset` 是模型量化所需要用到的数据集, 一般两百张左右。

更多详细的参数信息请看文档: `RT-AK\rt_ai_tools\platforms\plugin_k210\README.md`。

当部署成功之后, 目标工程文件会多出几个文件:

文件	描述
rt_ai_lib/	RT-AK Libs, 模型推理库

文件	描述
applications/hand_landmark_kmodel.c	kmodel 的十六进制储存
applications/rt_ai_hand_landmark_model.c	与目标平台相关的信息
applications/rt_ai_hand_landmark_model.h	模型相关信息

同时，在 `RT-AK\rt_ai_tools\platforms\plugin_k210` 路径下会生成两个文件

文件	描述
hand_landmark.kmodel	k210 所支持的模型格式
convert_report.txt	tflite 模型转成 kmodel 格式的缓存信息

如果不想生成上述两个文件，可以在模型部署的时候命令行参数末尾加上：`--clear`

注意：

1、RT-AK 部署成功后不会产生应用代码，比如模型推理代码，需要手工编写，详见“3. 嵌入式 AI 模型应用”

2、在应用开发过程中，请遵守 RT-Thread 的编程规范以及 API 使用标准

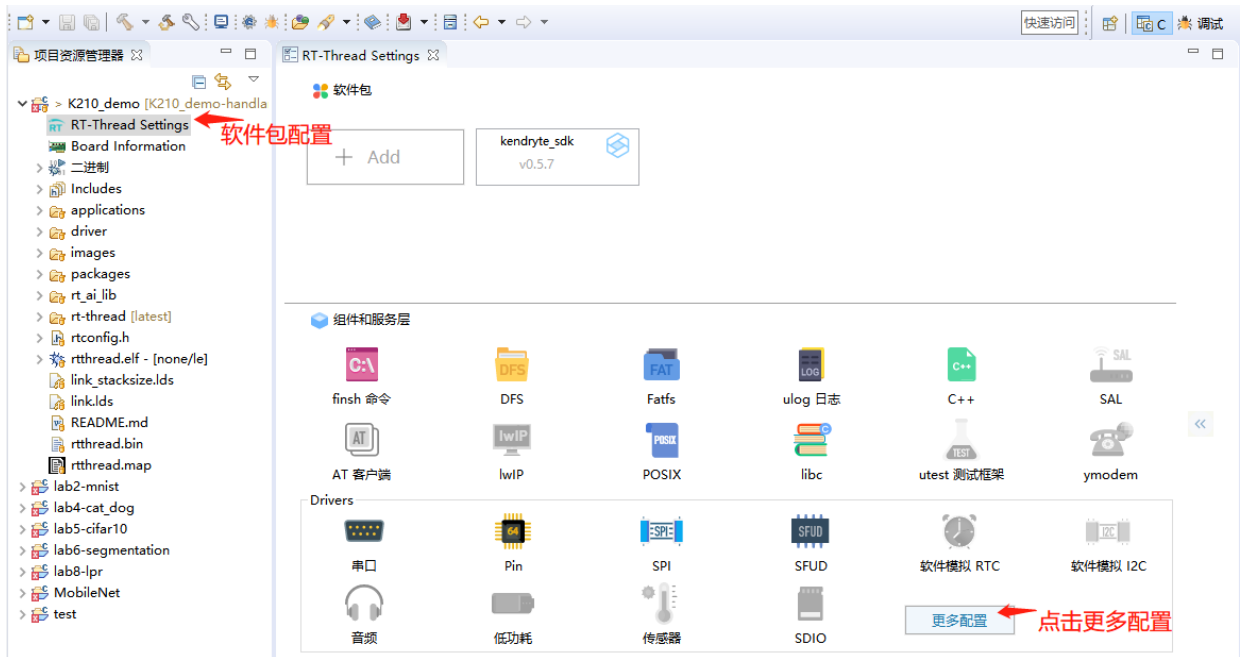
## 3.3 嵌入式 AI 模型应用

使用 RT-AK 将训练好的 tflite 模型成功部署到工程之后，我们就可以开始着手编写应用层代码来使用该模型。本节的所有代码详见附件中的 [Applications](#)

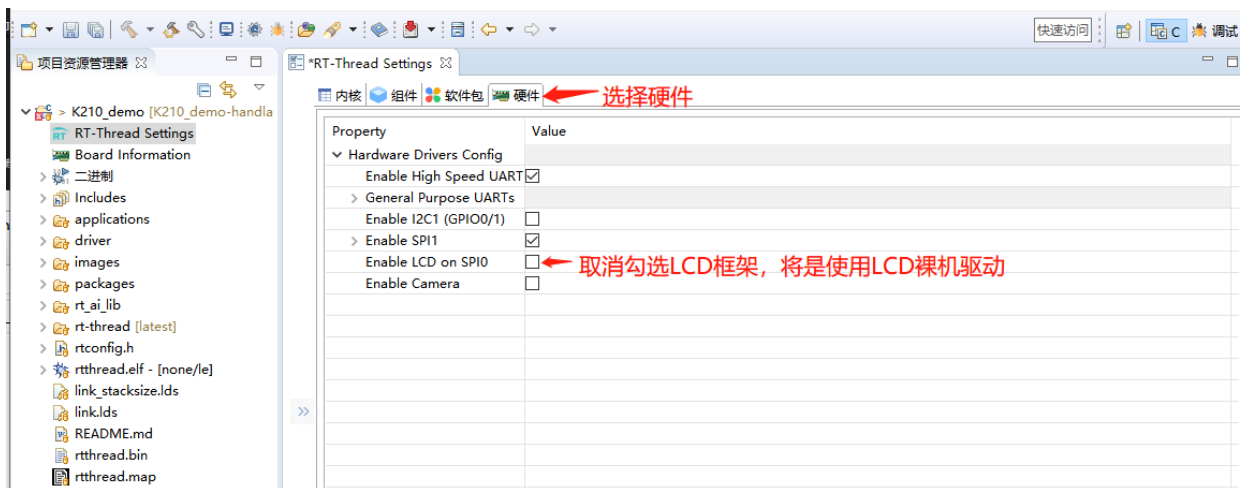
### 3.3.1 应对内存限制方法

按照之前每个实验的内容，我们发现配置本节实验将会出现实验板“卡死”的现象，这是由于我们的模型过大，导致了内存不够，最终程序无法实现。为了缓解内存不够造成的影响，在面向嵌入式开发过程中我们可以通过配置 `rtthread` 操作系统的组件，减少占用的内存，从而实现在内存受限的情况下运行较大规模的 AI 模型。具体的操作如下：

1) 首先点击 **RT-Thread settings**，进入软件包配置界面，并点击更多配置选项。



2) 选择硬件，我们看到了“Enable LCD on SPI0”，此时我们将在项目工程中屏蔽 RT-Thread 的 LCD 框架，从而释放部分内存，此时我们使用的就是 LCD 裸机驱动。释放的内存将用于 AI 模型的加载。



### 3.3.2 代码流程

系统内部初始化：

- 系统时钟初始化
- LCD 初始化

RT-AK Lib 模型加载并运行：

- 注册模型（代码自动注册，无需修改）
- 找到注册模型
- 初始化模型，挂载模型信息，准备运行环境
- 运行（推理）模型

- 获取输出结果

**Hand\_landmark** 业务逻辑层:

- 获取第一个输出, MULTI\_HAND\_LANDMARKS
  - 取出 (x, y) 的坐标
- 获取第二个输出, score
- 将坐标连线

### 3.3.3 核心代码说明

```
/* Set CPU and dvp clk */
sysctl_clock_enable(SYSCTL_CLOCK_AI);

// sysctl_clock_enable(SYSCTL_CLOCK_AI);
io_set_power();
io_mux_init();
lcd_init();
lcd_clear(BLACK);

...

mymodel = rt_ai_find(MY_MODEL_NAME);

if (rt_ai_init(mymodel, (rt_ai_buffer_t *)input_chw_data) != 0)
{ ... }

if(rt_ai_run(mymodel, ai_done, &g_ai_done_flag) != 0)
{ ... }

output = rt_ai_output(mymodel, 0); // 获取第一个输出
score = rt_ai_output(mymodel, 1); // 获取第二个输出
*score = (*score) * 100;
printf("prediction score: %.2f%%\r\n", *score);

// post-process output(21 points & coordinates)
float x[POINTS], y[POINTS]; // (x, y) coordinates
int index;
for(int i=0; i < MY_MODEL_OUT_1_SIZE; i++)
{
    index = i / 3;
    if (0 == i % 3)
        x[index] = output[i] / MODEL_INPUT_W * REAL_WIDTH;
    else if (1 == i % 3)
        y[index] = output[i] / MODEL_INPUT_H * REAL_HEIGHT;
}
```

```
// show result in LCD
...
lcd_draw_picture(0, 0, REAL_WIDTH, REAL_HEIGHT, Hand_10);
lcd_draw_string(4, 4, result, GREEN);
for(int i=0; i<POINTS; i++)
{
    lcd_draw_line(x[connections[i][0]], y[connections[i][0]],
                  x[connections[i][1]], y[connections[i][1]], RED);
}
for(int i=0; i<POINTS; i++)
    lcd_draw_point(x[i], y[i], BLACK);
rt_free(result);
rt_free(display_image.addr);
return 0;
```

补充模型输入输出说明：

- K210 指定的图片数据输入格式是 CHW, chanel: c, heigh:H, width:W;
- 模型的输入是 3\*224\*224, C=3, H=224, W=224, uint8,
- 模型的输出是 [1\*63, 1\*1, 1\*1], float32
- 模型的输入样本存放文件: my\_dataset.h

## 第 4 章

# 编译烧录

### 4.1 编译

参考 [lab2-env](#) 教程中 Studio 使用方法。新建->RT-Thread项目->基于开发板->K210-RT-DRACO 输入工程目录和工程名，新建基于开发板的模板工程。将实验代码复制到 **application** 文件中替换原文件代码。点击 [编译](#)。会在你的工程根目录下生成一个 **rtthread.bin** 文件，然后参考下面的 [烧录方法](#)。其中 **rtthread.bin** 需要烧写到设备中进行运行。

### 4.2 烧录

连接好串口，点击 Studio 中的下载图标进行下载，详细可参考[lab-env2](#)

或者

使用 K-Flash 工具进行烧写 bin 文件。

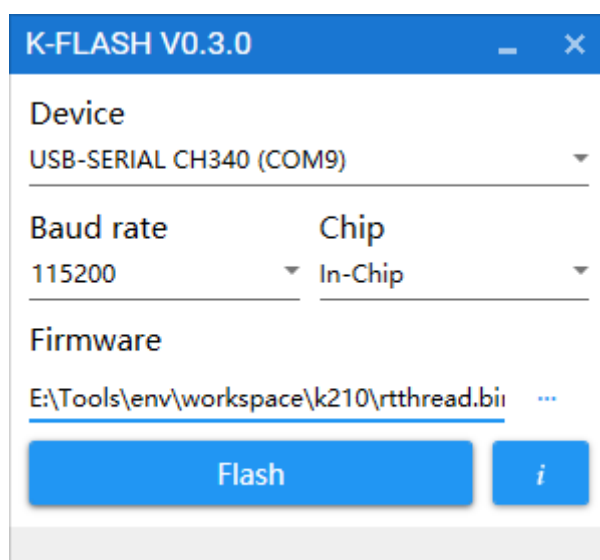
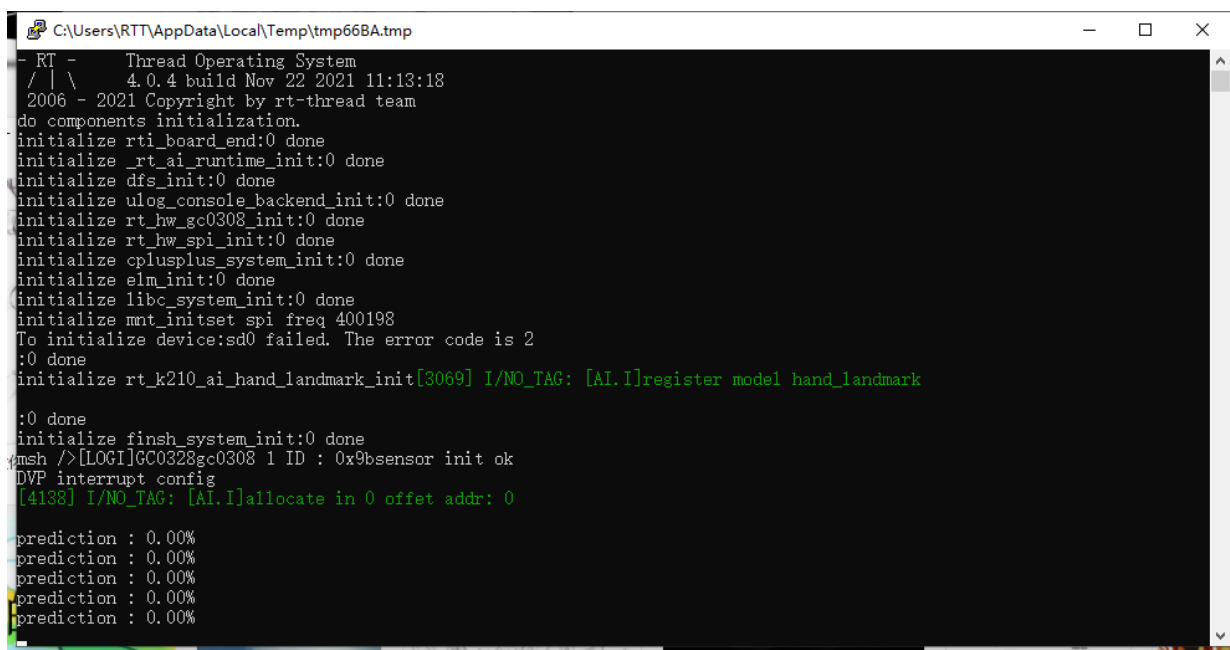


图 4.1: K-Flash

## 第 5 章

# 实验现象

如果编译 & 烧写无误，K-Flash 会自动打开 Windows 终端，自动连接实验板，系统启动后，我们的程序会自动运行，LCD 显示器先会显示图片 logo 和文字，一秒后显示预测结果。



```
- RT - Thread Operating System
/ | \ 4.0.4 build Nov 22 2021 11:13:18
2006 - 2021 Copyright by rt-thread team
do components initialization.
initialize rti_board_end:0 done
initialize rt_ai_runtime_init:0 done
initialize dfs_init:0 done
initialize ulog_console_backend_init:0 done
initialize rt_hw_gc0308_init:0 done
initialize rt_hw_spi_init:0 done
initialize cplusplus_system_init:0 done
initialize elm_init:0 done
initialize libc_system_init:0 done
initialize mnt_initset spi freq 400198
To initialize device:sd0 failed. The error code is 2
:0 done
initialize rt_k210_ai_hand_landmark_init[3069] I/NO_TAG: [AI.I]register model hand_landmark
:0 done
initialize finsh_system_init:0 done
tmsh />[LOGI]GC0328gc0308 1 ID : 0x9bsensor init ok
DVP interrupt config
[4138] I/NO_TAG: [AI.I]allocate in 0 offset addr: 0
prediction : 0.00%
prediction : 0.00%
prediction : 0.00%
prediction : 0.00%
prediction : 0.00%
```





# 第 6 章

## 附件

### 6.1 嵌入式 AI 开发 API 文档

```
rt_ai_t rt_ai_find(const char *name);
```

Paramaters	Description
name	注册的模型名
<b>Return</b>	—
rt_ai_t	已注册模型句柄
NULL	未发现模型

描述: 查找已注册模型

```
rt_err_t rt_ai_init(rt_ai_t ai, rt_aibuffer_t* work_buf);
```

Paramaters	Description
ai	rt_ai_t 句柄
work_buf	运行时计算所用内存
<b>Return</b>	—
0	初始化成功
非 0	初始化失败

描述: 初始化模型句柄, 挂载模型信息, 准备运行环境.

```
rt_err_t rt_ai_run(rt_ai_t ai, void (*callback)(void * arg), void *arg);
```

Paramaters	Description
ai	rt_ai_t 模型句柄
callback	运行完成回调函数
arg	运行完成回调函数参数
<b>Return</b>	—
0	成功
非 0	失败

**描述:** 模型推理计算

```
rt_aibuffer_t rt_ai_output(rt_ai_t aihandle,rt_uint32_t index);
```

Paramaters	Description
ai	rt_ai_t 模型句柄
index	结果索引
<b>Return</b>	—
NOT NULL	结果存放地址
NULL	获取结果失败

**描述:** 获取模型运行的结果, 结果获取后.

rt\_ai\_libs/readme.md 文件中有详细说明

## 6.2 LCD API 说明手册

```
/**
 * @fn void lcd_init(void);
 * @brief LCD初始化
 */
void lcd_init(void);
/**
 * @fn void lcd_clear(uint16_t color);
 * @brief 清屏
 * @param color 清屏时屏幕填充色
 */
void lcd_clear(uint16_t color);
/**
 * @fn void lcd_set_direction(lcd_dir_t dir);
 * @brief 设置LCD显示方向
 * @param dir 显示方向参数
 */
```

```

void lcd_set_direction(lcd_dir_t dir);
/**
 * @fn void lcd_set_area(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2);
 * @brief 设置LCD显示区域
 * @param x1 左上角横坐标
 * @param y1 左上角纵坐标
 * @param x2 右下角横坐标
 * @param y2 右下角纵坐标
 */
void lcd_set_area(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2);
/**
 * @fn void lcd_draw_point(uint16_t x, uint16_t y, uint16_t color);
 * @brief 画点
 * @param x 横坐标
 * @param y 纵坐标
 * @param color 颜色
 */
void lcd_draw_point(uint16_t x, uint16_t y, uint16_t color);

/**
 * @fn void lcd_draw_string(uint16_t x, uint16_t y, char *str, uint16_t color);
 * @brief 显示字符串
 * @param x 显示位置横坐标
 * @param y 显示位置纵坐标
 * @param str 字符串
 * @param color 字符串颜色
 */
void lcd_draw_string(uint16_t x, uint16_t y, char *str, uint16_t color);
/**
 * @fn void lcd_draw_picture(uint16_t x1, uint16_t y1, uint16_t width, uint16_t
    height, uint32_t *ptr);
 * @brief 显示图片
 * @param x1 左上角横坐标
 * @param y1 左上角纵坐标
 * @param width 图片宽
 * @param height 图片高
 * @param ptr 图片地址
 */
void lcd_draw_picture(uint16_t x1, uint16_t y1, uint16_t width, uint16_t height,
    uint32_t *ptr);
/**
 * @fn void lcd_draw_rectangle(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2,
    uint16_t width, uint16_t color);
 * @brief 画矩形
 * @param x1 左上角横坐标
 * @param y1 左上角纵坐标
 * @param x2 右下角横坐标
 * @param y2 右下角纵坐标
 * @param width 线条宽度(当前无此功能)

```

```
* @param color 线条颜色
*/
```

## 6.3 Camera 使用

Camera 已对接 RT-Thread 设备驱动框架，可通过 RT-Thread Device 标准接口进行调用。RT-Thread 设备框架文档：<https://www.rt-thread.org/document/site/#/rt-thread-version/rt-thread-standard/README>

下面将使用到的 API 进行简要说明，引用自 RTT 官方文档：

### 查找设备

应用程序根据设备名称获取设备句柄，进而可以操作设备。查找设备函数如下所示：

```
rt_device_t rt_device_find(const char* name);
```

参数	描述
name	设备名称
返回	——
设备句柄	查找到对应设备将返回相应的设备句柄
RT_NULL	没有找到相应的设备对象

### 初始化设备

获得设备句柄后，应用程序可使用如下函数对设备进行初始化操作：

```
rt_err_t rt_device_init(rt_device_t dev);
```

参数	描述
dev	设备句柄
返回	——
RT_EOK	设备初始化成功
错误码	设备初始化失败

[!NOTE] 注：当一个设备已经初始化成功后，调用这个接口将不再重复做初始化 0。

### 打开和关闭设备

通过设备句柄，应用程序可以打开和关闭设备，打开设备时，会检测设备是否已经初始化，没有初始化则会默认调用初始化接口初始化设备。通过如下函数打开设备：

```
rt_err_t rt_device_open(rt_device_t dev, rt_uint16_t oflags);
```

参数	描述
dev	设备句柄
oflags	设备打开模式标志
返回	——
RT_EOK	设备打开成功
-RT_EBUSY	如果设备注册时指定的参数中包括 RT_DEVICE_FLAG_STANDALONE 参数，此设备将不允许重复打开
其他错误码	设备打开失败

oflags 支持以下的参数：

```
#define RT_DEVICE_OFLAG_CLOSE 0x000 /* 设备已经关闭（内部使用）*/
#define RT_DEVICE_OFLAG_RDONLY 0x001 /* 以只读方式打开设备 */
#define RT_DEVICE_OFLAG_WRONLY 0x002 /* 以只写方式打开设备 */
#define RT_DEVICE_OFLAG_RDWR 0x003 /* 以读写方式打开设备 */
#define RT_DEVICE_OFLAG_OPEN 0x008 /* 设备已经打开（内部使用）*/
#define RT_DEVICE_FLAG_STREAM 0x040 /* 设备以流模式打开 */
#define RT_DEVICE_FLAG_INT_RX 0x100 /* 设备以中断接收模式打开 */
#define RT_DEVICE_FLAG_DMA_RX 0x200 /* 设备以 DMA 接收模式打开 */
#define RT_DEVICE_FLAG_INT_TX 0x400 /* 设备以中断发送模式打开 */
#define RT_DEVICE_FLAG_DMA_TX 0x800 /* 设备以 DMA 发送模式打开 */
```

应用程序打开设备完成读写等操作后，如果不需要再对设备进行操作则可以关闭设备，通过如下函数完成：

```
rt_err_t rt_device_close(rt_device_t dev);
```

参数	描述
dev	设备句柄
返回	——
RT_EOK	关闭设备成功
-RT_ERROR	设备已经完全关闭，不能重复关闭设备
其他错误码	关闭设备失败

### 读写设备

应用程序从设备中读取数据可以通过如下函数完成（摄像头设备中`pos`和`size`参数无作用）：

```
rt_size_t rt_device_read(rt_device_t dev, rt_off_t pos, void* buffer, rt_size_t size)
;
```

参数	描述
<code>dev</code>	设备句柄
<code>pos</code>	读取数据偏移量
<code>buffer</code>	内存缓冲区指针，读取的数据将会被保存在缓冲区中
<code>size</code>	读取数据的大小
返回	——
读到数据的实际大小	如果是字符设备，返回大小以字节为单位，如果是块设备，返回的大小以块为单位
0	需要读取当前线程的 <code>errno</code> 来判断错误状态

### 数据收发回调

当硬件设备收到数据时，可以通过如下函数回调另一个函数来设置数据接收指示，通知上层应用线程有数据到达：

```
rt_err_t rt_device_set_rx_indicate(rt_device_t dev, rt_err_t (*rx_ind)(rt_device_t dev, rt_size_t size));
```

参数	描述
<code>dev</code>	设备句柄
<code>rx_ind</code>	回调函数指针
返回	——
<code>RT_EOK</code>	设置成功