

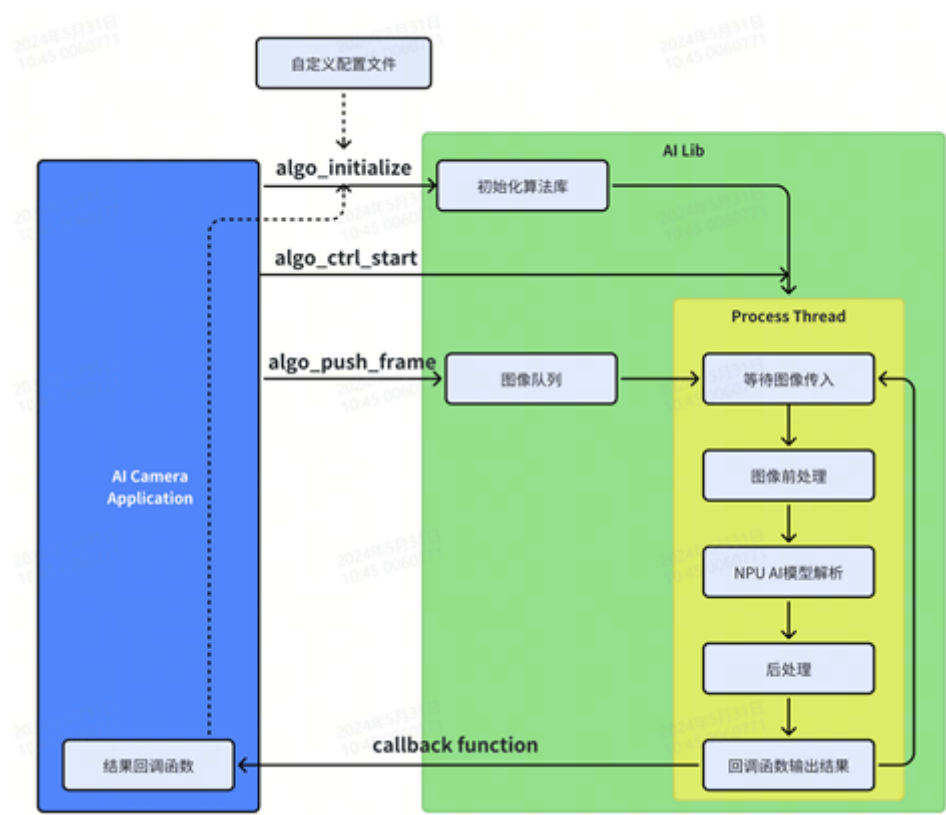
Morph AI Lib编程指南

1.AI Lib简介

AI Lib 是Morph AI相机核心的算法模块，是一个源码开放的通用NPU模型调用库。它负责从App侧获取所需图像，并接收控制指令。其内部封装了常用目标检测模型的调用流程，确保二次开发的便捷性。在算法计算完成后，通过回调函数，将计算结果经由App上报到上位机Scepter SDK，实现数据的高效传输。借助AI Lib，用户可以轻松快捷地部署自己的算法模型。

2.AI Lib流程与接口定义

流程图



由于源码开放，开发者可以按照需要，自由的修改库内部代码以实现特定功能，但是API接口函数声明不能修改，否则影响App与AI lib的调用流程。

接口定义

```
1 //初始化，资源传入（算法资源根目录（默认/userdata/algorithm_data），callback函数指针）
2 ALGO_RET_E algo_initialize(const ALGO_INIT_PARAM_T *p_init_param);
3 //callback函数声明
4 std::function<void (uint64_t timestamp, const char *p_proc_result, uint32_t result_len)>;
5     ALGO_PROCESS_CB_FUNC_T process_cb_func;
6
7 //开始运行
8 ALGO_RET_E algo_ctrl_start(void);
9
```

```

10 //停止运行
11 ALGO_RET_E algo_ctrl_stop(void);
12
13 //图像传入，根据设置的图像类型填入对应图像与sensor参数（图像，类型，分辨率，内外参等）
14 ALGO_RET_E algo_push_frame(const CAM_JOINT_FRAME_T *p_joint_frame);
15
16 //参数设置，端到端使用，由客户自由定义
17 ALGO_RET_E algo_set_param(uint32_t param_id, const char *p_in_param,
uint16_t param_len);
18
19 //参数读取，端到端使用，由客户自由定义
20 ALGO_RET_E algo_get_param(uint32_t param_id, const char **p_out_param,
uint16_t *p_param_len);

```

3.AI Lib代码说明

3.1 目录

Morph SDK的下载地址: <https://github.com/ScepterSW/MorphAISDK>

示例代码包含在Morph SDK中，路径为：MorphAISDK/2_deploy/algLib。

目录结构：

```

1  └─ 3rdparty
2  │   └─ jsoncpp
3  │   └─ opencv
4  └─ CMakeLists.txt
5  └─ Include
6  │   └─ alg_api.h
7  │   └─ alg_define.h
8  │   └─ alg_types.h
9  └─ Src
10     └─ alg_api.cpp
11     └─ alg_impl.cpp
12     └─ alg_impl.h
13     └─ image_process.cpp
14     └─ image_process.h
15     └─ log.cpp
16     └─ log.h
17     └─ Postprocess
18     │   └─ post_process.cpp
19     │   └─ post_process_custom.cpp
20     │   └─ post_process_custom.h
21     │   └─ post_process.h
22     │   └─ post_process_vzense_box.cpp
23     │   └─ post_process_vzense_box.h
24     │   └─ post_process_yolov5.cpp
25     │   └─ post_process_yolov5.h
26     └─ rknn_inference.cpp
27     └─ rknn_inference.h
28     └─ rknn_utils.cpp
29     └─ rknn_utils.h
30     └─ serialization.cpp
31     └─ serialization.h
32     └─ stoppable_thread.cpp
33     └─ stoppable_thread.h

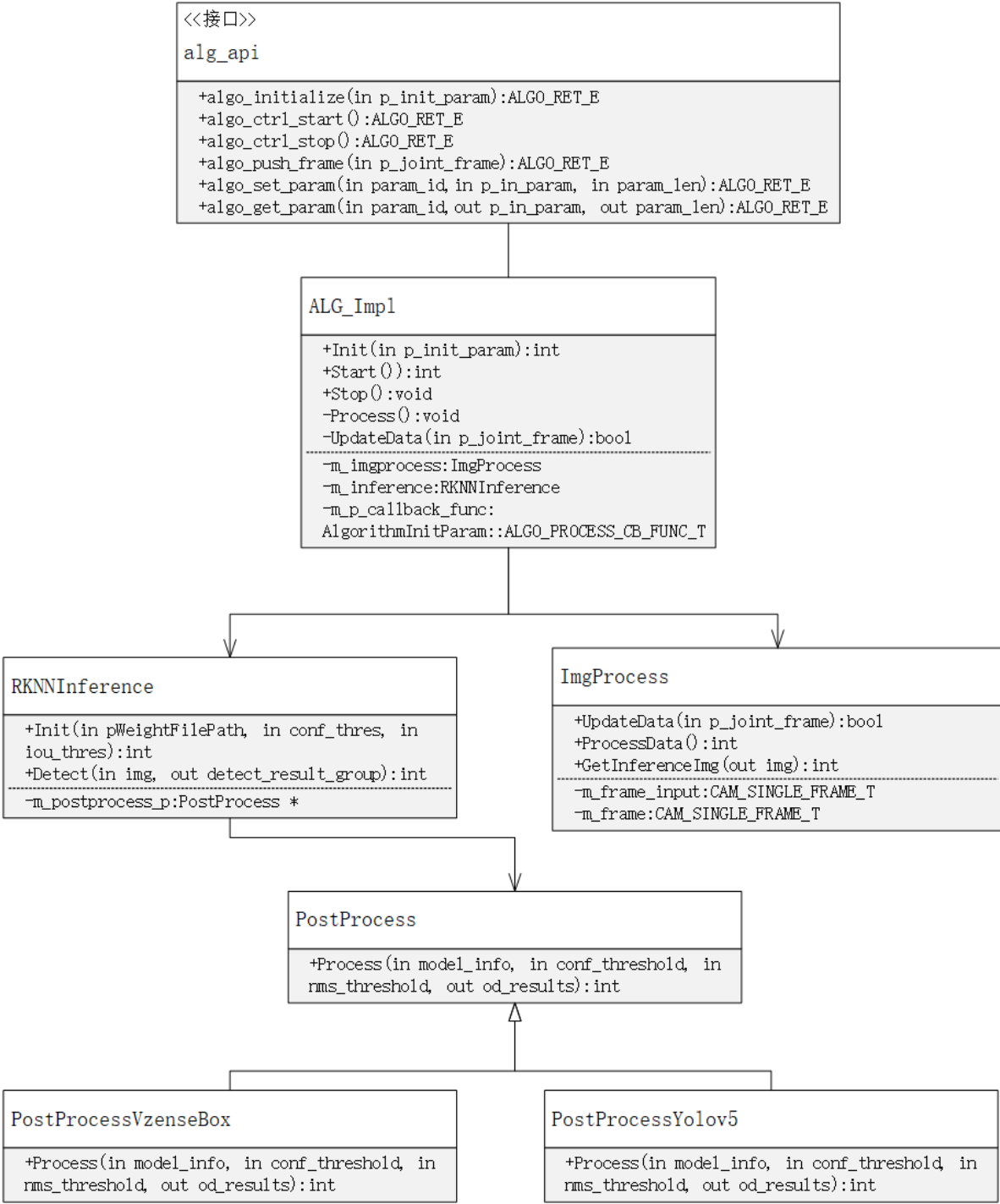
```

文件说明:

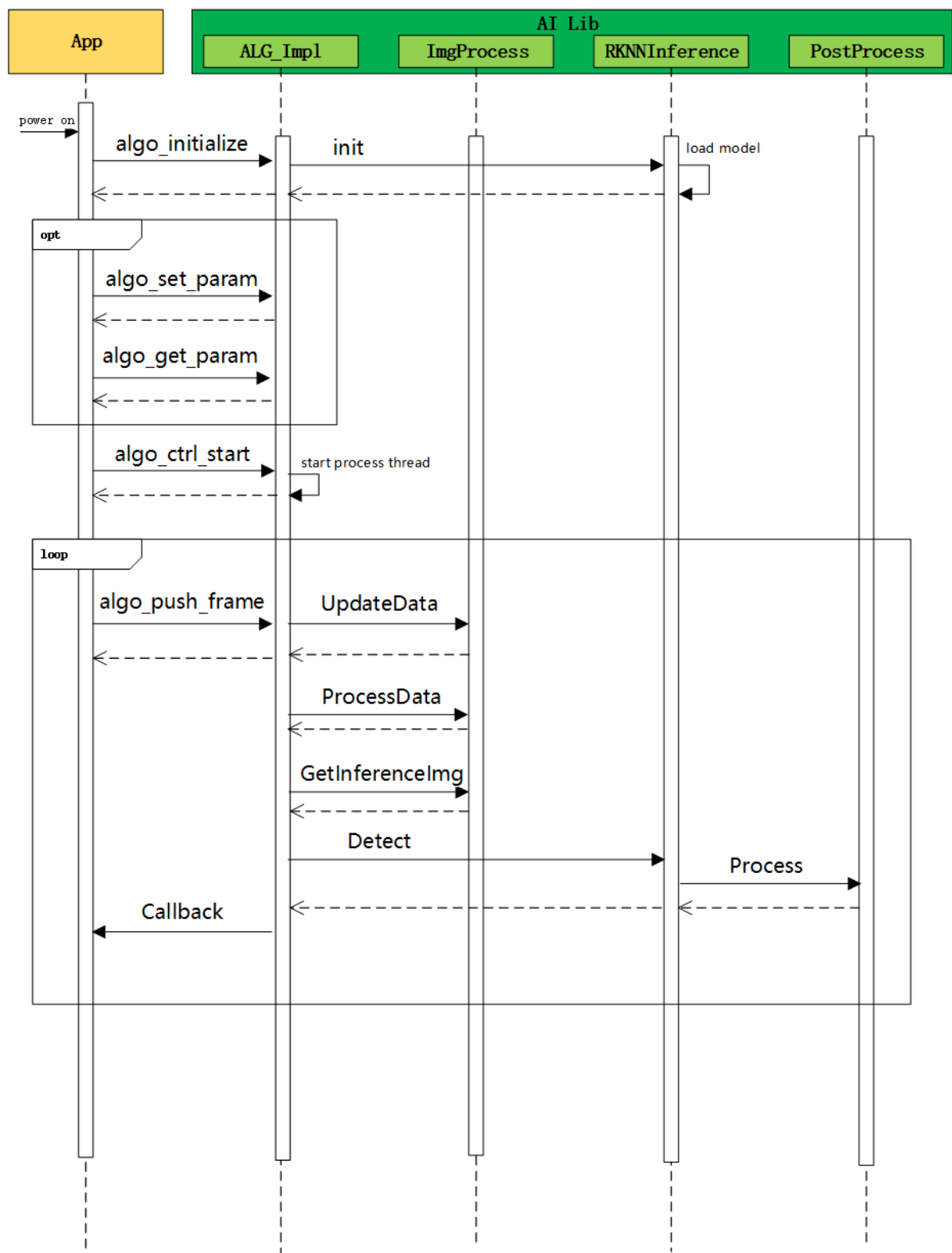
- 3rdparty: 依赖的第三方库
 - jsoncpp: 用于解析和生成 JSON 数据
 - opencv: 用于图像数据处理
- CMakeLists.txt: 构建algLib的配置文件
- Include
 - alg_api.h: 接口定义文件
 - alg_define.h: 定义接口中使用的enum, structure
 - alg_types.h: 定义lib内部使用的enum, structure
- Src
 - alg_api.cpp/h: 接口函数实现文件
 - alg_impl.cpp/h: 算法库管理文件
 - image_process.cpp/h: 图像前处理文件
 - log.cpp/h: 算法日志文件
 - Postprocess: 模型后处理文件, 解析模型检测结果
 - post_process.cpp/h: 模型后处理虚基类文件
 - post_process_custom.cpp/h: 用户自定义的模型后处理文件
 - post_process_vzense_box.cpp/h: vzense预制的相机包装盒模型的后处理文件
 - post_process_yolov5.cpp/h: yolov5模型的后处理文件
 - rknn_inference.cpp/h: rknn模型的模型预测处理文件
 - rknn_utils.cpp/h: rknn模型的模型预测处理工具文件
 - serialization.cpp/h: json格式的算法结果、算法参数的解析和生成的示例文件
 - stoppable_thread.cpp/h: 封装的线程工具文件

3.2 类图与时序图

类图



时序图



3.4 使用参考

示例代码是检测相机包装盒位置的例程，代码中使用彩色图做目标检测，输出预测的目标框，然后在对齐后的深度图上，计算目标框对应区域的点云均值。目标框的点云均值即是检测到的相机包装盒的位置。代码的处理过程大体如下：



当使用AI Lib实现检测其它类别物体的位置时，只需参考以下说明做简单修改，就能很方便的实现相应功能。

读写算法参数:

在算法接收图像开始处理前，需要先读写参数，确保算法执行效果。相关代码在文件alg_api.cpp中，具体函数如下：

```
1 // 写参数
2 ALGO_RET_E algo_set_param(uint32_t param_id, const char *p_in_param, uint16_t
  param_len)
3
4 // 读参数
5 ALGO_RET_E algo_get_param(uint32_t param_id, const char **p_out_param,
  uint16_t *p_param_len)
```

图像预处理:

示例中，App输入给算法的图像可能尺寸大小，像素格式等属性与模型要求的输入图像不符，因此算法在把图像输入给模型前，要按照模型要求对图像进行预处理，以满足模型需要。在实际工程中，根据自身需求进行更改即可。

相关代码在文件image_process.cpp中，具体函数如下：

```
1 int ImgProcess::ProcessData()
2 {
3     int result = ALGO_RET_GET_FRAME_TIMEOUT;
4     unique_lock<mutex> lk(m_mutex);
5     if (false == m_cv.wait_for(lk, std::chrono::milliseconds(1000), [this] {
6         return 0 != m_frame_input[0].frame_no; }))
7     {
8         return result;
9     }
10
11     m_frame_inference.frame_no = m_frame_input[0].frame_no;
12     m_frame_inference.timestamp = m_frame_input[0].timestamp;
13     memcpy(m_frame_inference.p_data + LABEL_BORDER * m_frame_inference.width
  * GetElemSize(m_frame_inference.pixel_format), m_frame_input[0].p_data,
  m_frame_input[0].data_len);
14     SwapFrame(m_frame_input[1], m_frame_Depth);
15
16     m_frame_input[0].frame_no = 0;
17
18     return 0;
19 }
```

模型预测:

模型预测是把符合要求的图像输入NPU模型，并取得模型结果。相关代码在文件rknn_inference.cpp中，具体函数如下：

```
1 int RKNNInference::Detect(const CameraSingleFrame &img,
  detect_result_group_t& detect_result_group)
2 {
3     memcpy(m_model_info.inputs[0].buf, img.p_data, img.data_len);
4     int ret = rknn_inputs_set(m_model_info.ctx, m_model_info.n_input,
  m_model_info.inputs);
5
6     //模型预测
7     ret = rknn_run(m_model_info.ctx, NULL);
8     if(ret < 0)
```

```

9      {
10         Log("result: %d", ret);
11         return ret;
12     }
13
14     //后处理
15     int m_postprocess_p->Process(&m_model_info, m_conf_thres, m_iou_thres,
16 &detect_result_group);
17
18     return ret;
19 }

```

模型的更换，请参考《Morph AI相机快速开始手册.pdf》文档。

模型预测后处理：

模型预测后处理是把RKNN模型的输出解析为可理解的格式（对于目标检测模型，其包含预测框位置，类别得分等），并进行NMS筛选。

相关代码在文件Postprocess/post_process_xxx.cpp中，在Postprocess/post_process_custom.cpp文件中，用户可以实现自定义的后处理，具体函数如下：

```

1  int PostProcessCustom::Process(MODEL_INFO *model_info, float conf_threshold,
2  float nms_threshold, detect_result_group_t *od_results)
3  {
4      Log("A custom implementation is required.");
5
6      return 0;
7  }

```

模型结果与深度图融合处理：

示例中，把模型的目标检测结果与深度图相结合，可以进一步计算检测目标的位置信息。相关代码在文件alg_impl.cpp中，具体函数如下：

```

1  void ALG_Impl::Run()
2  {
3      //图像预处理
4      int result = m_imgprocess.ProcessData();
5      if (ALGO_RET_OK == result)
6      {
7          CameraSingleFrame frame;
8          m_imgprocess.GetInferenceImg(frame);
9
10         //模型检测
11         detect_result_group_t detect_result_group = {0};
12         result = m_inference.Detect(frame, detect_result_group);
13         if(0 != result)
14         {
15             SetInfo(Log("Detect:%d is failed.", result));
16         }
17
18         //模型结果与深度图融合处理,计算位置信息
19         m_imgprocess.GetDepthImg(frame);
20         cv::Mat depth = cv::Mat(frame.height, frame.width, CV_16UC1,
21 frame.p_data);
22         for (int i = 0; i < detect_result_group.count; i++)

```

```

22     {
23         detect_result_t& det_result = detect_result_group.results[i];
24         cv::Rect detectBox = cv::Rect(det_result.box.left,
det_result.box.top, (det_result.box.right - det_result.box.left),
(det_result.box.bottom - det_result.box.top));
25         m_imgprocess.UpdatePointCloud(depth, detectBox,
det_result.centerPosInWorld);
26     }
27
28     //结果回调
29     CallBackFunc(frame.timestamp, detect_result_group);
30 }
31 else
32 {
33     SetInfo(result);
34 }
35 }

```

算法结果回调：

把算法执行结果，通过回调函数，传递给App。相关代码在文件alg_impl.cpp中，具体函数如下：

```

1  void ALG_Impl::CallBackFunc(uint64_t timestamp, const detect_result_group_t&
detect__result_group)
2  {
3      if (nullptr != m_p_callback_func)
4      {
5          uint8_t *pBuf = m_result_buf.get();
6          char *jsonData = (char *) (pBuf);
7          Serialization serialization;
8          uint32_t jsonLen = 0;
9          //使用Json格式化算法结果
10         serialization.GetResultJson(jsonData, jsonLen,
detect__result_group);
11
12         {
13             lock_guard<mutex> lk(m_mutex);
14             m_p_callback_func(timestamp, (const char *) pBuf, jsonLen);
15         }
16
17         Log("timestamp:%" PRIu64 " ", jsonLen:%d, pBuf:%s", timestamp,
jsonLen, pBuf);
18     }
19 }

```