ProcVision 算法快速入门（简化版）

目标：用最少步骤完成一个可交付的算法项目，并输出离线包。

1. 环境准备

- 创建并激活虚拟环境：
  - Windows：`python -m venv .venv`，`\.venv\Scripts\activate`
  - Linux/Mac：`python -m venv .venv`，`source .venv/bin/activate`
- 安装 SDK：`pip install procvision_algorithm_sdk`

2. 项目初始化（CLI）

- 最简命令：`procvision-cli init product_a_screw_check`
- 示例（指定目录与PID）：`procvision-cli init product_a_screw_check -d ./product_a_screw_check --pids A01,A02 -v 1.2.1`
- 命令与参数说明：
  - 命令格式：`procvision-cli init <name> [-d|--dir <dir>] [--pids <p1,p2>] [-v|--version <ver>] [-e|--desc <text>]`
  - `name`：算法名称；生成模块目录与入口类名（必填）
  - `-d|--dir`：目标目录；默认在当前目录下创建（可选）
  - `--pids`：支持的产品型号列表；逗号分隔，默认 `p001,p002`（可选）
  - `-v|--version`：算法版本；默认 `1.0.0`（可选）
  - `-e|--desc`：算法描述；写入 `manifest.json`（可选）
- 目录结构：

- product_a_screw_check/manifest.json
- product_a_screw_check/product_a_screw_check/__init__.py
- product_a_screw_check/product_a_screw_check/main.py

3. 编辑 manifest.json（不含 steps）

- 必填：name、version、entry_point、supported_pids
- 推荐：description
- 示例：

```
{
  "name": "product_a_screw_check",
  "version": "1.2.1",
  "entry_point": "product_a_screw_check.main:ProductAScrewCheckAlgorithm",
  "description": "A产品螺丝检测",
  "supported_pids": ["A01", "A02"]
}
```

- 约束：supported_pids 与 get_info() 一致。

4. 最小算法实现（step_index 从 1 开始，两步骤）

```
from typing import Any, Dict
from procvision_algorithm_sdk import BaseAlgorithm, Session,
read_image_from_shared_memory

class ProductAScrewCheckAlgorithm(BaseAlgorithm):
    def __init__(self) -> None:
        super().__init__()
        self._supported_pids = ["A01", "A02"]

    def get_info(self) -> Dict[str, Any]:
        return {
            "name": "product_a_screw_check",
            "version": "1.2.1",
            "description": "A产品螺丝检测",
            "supported_pids": self._supported_pids,
            "steps": [
                {"index": 1, "name": "定位主板", "params": []},
                {"index": 2, "name": "检测左上角螺丝", "params": []}
            ]
```

```
        }

    def pre_execute(self, step_index: int, pid: str, session: Session, user_params:
Dict[str, Any], shared_mem_id: str, image_meta: Dict[str, Any]) -> Dict[str, Any]:
        if pid not in self._supported_pids:
            return {"status": "ERROR", "message": f"不支持的产品型号: {pid}",
"error_code": "1001"}
        img = read_image_from_shared_memory(shared_mem_id, image_meta)
        if img is None:
            return {"status": "ERROR", "message": "图像数据为空", "error_code":
"1002"}
        if step_index == 1:
            return {
                "status": "OK",
                "message": "定位完成",
                "data": {
                    "calibration_rects": [
                        {"x": 100, "y": 120, "width": 200, "height": 250, "label":
"roi-1"}
                    ]
                }
            }
        return {"status": "OK", "message": "准备就绪"}

    def execute(self, step_index: int, pid: str, session: Session, user_params:
Dict[str, Any], shared_mem_id: str, image_meta: Dict[str, Any]) -> Dict[str, Any]:
        img = read_image_from_shared_memory(shared_mem_id, image_meta)
        if img is None:
            return {"status": "ERROR", "message": "图像数据为空", "error_code":
"1002"}
        if step_index == 2:
            return {"status": "OK", "data": {"result_status": "OK", "defect_rects":
[], "debug": {"latency_ms": 0.0}}}
        return {"status": "OK", "data": {"result_status": "OK", "defect_rects": [],
"debug": {"latency_ms": 0.0}}}
```

5. 校验与运行（CLI）

- 最简校验：`procvision-cli validate ./product_a_screw_check`
- 参数说明：
  - 命令格式：`procvision-cli validate [project] [--manifest <path>] [--zip <path>] [--json]`
  - `project`：算法项目根目录（位置参数）；默认 `.`（可选）
  - `--manifest`：指定 `manifest.json` 路径（可选）
  - `--zip`：离线包路径；检查包内结构（可选）
  - `--json`：以 JSON 输出校验结果（可选）
- 最简运行：`procvision-cli run ./product_a_screw_check --pid A01 --image ./test.jpg`

- 参数说明：
  - 命令格式：`procvision-cli run <project> --pid <pid> --image <path> [--params <json>] [--json]`
  - `project`：算法项目根目录（必填）
  - `--pid`：产品型号编码；必须在 `supported_pids` 列表内（必填）
  - `--image`：本地图片路径（JPEG/PNG）；用于写入共享内存（必填）
  - `--params`：用户参数（JSON 字符串）；本快速入门不使用（可选）
  - `--json`：以 JSON 输出运行结果（可选）

6. 依赖与离线打包

- 锁定依赖：`pip freeze > requirements.txt`
- 要点：请在目标 Python 版本（例如 3.10）的虚拟环境中执行 freeze，确保生成的版本在目标平台有可用的 wheel。
- 下载 wheels（示例为 Windows + Python 3.10）：

```
pip download -r requirements.txt -d ./product_a_screw_check/wheels \
  --platform win_amd64 --python-version 3.10 --implementation cp --abi cp310
```

- 最简打包：`procvision-cli package ./product_a_screw_check`
- 默认参数（可省略）：
  - `--output`：按 `name/version` 生成 `<name>-v<version>-offline.zip`
  - `--auto-freeze`：开启，缺失 `requirements.txt` 时自动生成
  - `--wheels-platform`：`win_amd64`
  - `--python-version`：`3.10`（或读取项目根 `.procvision_env.json`）
  - `--implementation`：`cp`
  - `--abi`：`cp310`（或读取项目根 `.procvision_env.json`）
  - 说明：初始化时会生成 `.procvision_env.json`，打包默认读取其中的目标环境配置。
- 参数说明：
  - 命令格式：`procvision-cli package <project> [--output <zip>] [--requirements <path>] [--auto-freeze] [--wheels-platform <p>] [--python-version <v>] [--implementation <impl>] [--abi <abi>] [--skip-download]`
  - `project`：算法项目根目录（必填）
  - `--output|-o`：输出 zip 路径；默认 `name-version-offline.zip`（可选）

- `--requirements|-r`：`requirements.txt` 路径；缺失时可配 `--auto-freeze`（可选）
- `--auto-freeze|-a`：自动生成 `requirements.txt`（`pip freeze`）（可选）
- `--wheels-platform|-w`：目标平台；默认 `win_amd64`（可选）
- `--python-version|-p`：目标 Python 版本；建议 `3.10`（可选）
- `--implementation|-i`：Python 实现；常用 `cp`（可选）
- `--abi|-b`：ABI；与 Python 版本匹配，如 `cp310`（可选）
- `--skip-download|-s`：跳过依赖下载，仅打包现有内容（可选）

7. 交付前核对（必查）

- `supported_pids` 与 `get_info()` 完全一致
- `step_index` 从 1 开始（`pre_execute/execute`）
- `pre_execute` 返回 `status: OK/ERROR`；含 `message`
- `execute` 返回 `status: OK/ERROR`；OK 时 `data.result_status: OK/NG`
- NG 时包含 `ng_reason` 与 `defect_rects≤20`
- 离线包结构：源码目录、`manifest.json`、`requirements.txt`、`wheels/`、可选 `assets/`