

orisys API 文档

本文件提供了 orisys 套件的中文文档

准备 Python 开发环境

推荐下载 Anaconda，并使用 Python 3.10

```
# 创建并激活虚拟环境
conda create -n orisys python=3.10
conda activate orisys
```

安装 CUDA Toolkit 和 cuDNN

在官方链接下载和安装：<https://developer.nvidia.com/cuda/toolkit> 或在 Anaconda 环境中安装（例子使用 cuda12.9）：

<https://anaconda.org/channels/nvidia/packages/cuda/overview>

```
conda install nvidia/label/cuda-12.9.0::cuda-toolkit nvidia::cudnn
```

SDK 安装方法

1. 根据系统（win / linux），平台（amd64, aarch64），python（如 p38、p39、p310）版本下载对应的.whl 文件
2. 运行 pip 命令安装：

```
pip install orisys-{version}-{python-version}-{python-version}-{os}_{machine}.whl
```

如

```
pip install orisys-0.4.2-cp310-cp310-win_amd64.whl
```

3. 安装 CuPy（根据您的 CUDA 版本选择）：

方法 1：使用 extras（推荐）

```
# 对于 CUDA 12.x
pip install orisys[cuda12]

# 对于 CUDA 13.x
pip install orisys[cuda13]
```

方法 2：手动安装

```
# 对于 CUDA 12.x
pip install cupy-cuda12x>=12.3.0
```

```
# 对于 CUDA 13.x
pip install cupy-cuda13x>=13.0.0
```

4. 测试导入:

```
import orisys
print(orisys.__version__)
```

Sensor 类别

描述

`orisys` 套件的核心类，用于从视频文件或实时摄像头中读取传感器图像，创建一个具体的传感器对象（实例）。并完成形变、光流及力信息的计算与管理。

输入参数

- **vid_src** (int | str, 必填): 传感器 ID、相机编号或视频路径。默认为 0。
- **config_name** (str, 可选): 传感器配置文件名称或 json 文件路径，用于加载对应的参数配置；例如 `ddjx01` 或 `./config/ddjx01.json`。
- **cal_path** (str, 可选, 已弃用): 该参数仅为兼容旧代码保留，当前版本中不再使用，传入后也会被忽略。
- **backend** (str, 可选): 光流场解耦计算后端，默认为 `auto`，可选 `cupy,opencv,cpu`。
- **verbose** (bool, 可选): 是否输出初始化及运行过程中的详细日志信息，默认为 `False`。

返回

- `Sensor` 实例，用于后续图像采集、形变计算和数据读取操作。### 示例

```
# Example 1: 用相机编号创建
from orisys import Sensor
sensor = Sensor(0)

# Example 2: 用视频文件创建
sensor = Sensor("/data/data.mp4")
```

2. `Sensor.get_img` 方法

描述

采集并预处理一帧图像并存到 `Sensor` 类别中。

返回

- `bool`: 是否成功读取图像。

状态更新

- 成功时更新 `sensor` 类里面的 `sensor.img`。处理后的 BGR 格式图像(uint8, 尺寸为 json 文件中 `resolution_stitch` 对应分辨率)。### 示例

```
sensor.get_img()
```

3. `Sensor.compute_deformation` 方法

描述

基于当前帧图像计算光流/形变向量场，并通过 HHD 分解将形变场分解为法向/切向分量，同时估计法向力、切向力与扭矩等标量输出。该方法会更新 `sensor` 的形变场与力相关状态，并返回当前帧的光流场。调用前置条件：需要先通过 `get_img()`成功更新图像。

输入参数

- **`check_motion`**(`bool`, 可选): 为 `True` 时先根据光流判断是否有运动，有运动才做 HHD 分解；为 `False` 时每帧都做 HHD。默认 `True`。
- **`threshold`**(`float`, 可选): **`check_motion`** 功能的阈值，默认为 6。
- **`tail_frames`** (`int`, 可选): 光流变化低于阈值后继续解算多少帧，确保传感器回复，默认为 10。

返回

- `bool`: 是否接触到物体，如果 **`check_motion`** 为 `False` 时返回 `None`。

示例

```
sensor.get_img()

is_touched = sensor.compute_deformation(check_motion=True, threshold=6)
# 步骤 2.3: 计算触碰点
if is_touched:
    sensor.compute_contact()
```

4. `Sensor.compute_contact` 方法

描述

基于已计算的法向形变场进行接触区域分析，输出接触质心与轮廓等信息，并生成接触深度图，默认配置在 json 文件中，可按需调节。

调用前置条件：需先执行 `compute_deformation()`。

返回

dict 为接触分析结果字典，至少包含以下键：

- **expansion_mask**: 扩张区域掩码 (`np.ndarray`) 。
- **contraction_mask**: 收缩区域掩码 (`np.ndarray`) 。
- **divergence_map**: 散度图 (`np.ndarray`) 。
- **centroid**: 质心坐标 (单点模式为 (x, y) ；多点模式为列表) 。
- **centroid_found**: 是否检测到接触区域 (`bool`) 。
- **contour**: 最大轮廓或轮廓列表 (可选字段，存在时返回) 。

状态更新

- **sensor.centroid**: 接触质心 (单点为 (x, y) ，多点为列表) 。
- **sensor.centroid_found**: 是否找到质心 (`bool`) 。

示例

```
sensor.compute_contact()
centroid, contour, depth = sensor.read_info(
    sensor.info.CENTROID,
    sensor.info.CONTOUR,
    sensor.info.DEPTH,
)
```

4. `Sensor.read_info` 方法

描述

按指定的信息类型 (`sensor.info` 枚举值) 批量读取传感器状态数据，并按传入顺序返回对应值的元组。该方法通常用于在完成形变计算后，读取光流场、分量场、力、帧率等信息。

输入参数

- **args:** 任意数量的 `Sensor.info` 枚举，用于指定需要获取的数据类型。支持的枚举值及对应数据如下：

支持的枚 `info keys`:

向量/图像类

- **VRAW:** 原始光流场, `np.ndarray`, `shape=(H, W, 2)`
- **VNORMAL:** 法向光流场, `np.ndarray`, `shape=(H, W, 2)`
- **VSHEAR:** 切向光流场, `np.ndarray`, `shape=(H, W, 2)`
- **DEPTH:** 深度场（可选），未启用/未生成时返回空数组 `np.array([])`。

标量类

- **FNORMAL:** 法向力, `float`。
- **FSHEARX:** X 切向力, `float`。
- **FSHEARY:** Y 切向力, `float`。
- **FPS:** 输出帧率, `float`。
- **CENTROID:** 触碰点的坐标, `(x, y)`。
- **CONTOUR:** 触碰区域轮廓, `np.ndarray`, `shape=(H, W)`, 未生成时返回 `np.empty`

时间戳

- **TIMESTAMP:** 时间戳字符串（HH:MM:SS），`str`。

注：H/W 由当前 json 文件中的 `resolution_stitch` 配置分辨率决定。

返回

Tuple

按传入的 `sensor.info` 参数顺序返回一个元组，
返回值数量与有效参数数量一致。

- 向量/图像类数据： `np.ndarray`
 - 若当前流程中尚未生成该数据，返回空数组
- 标量类数据： `float`
- 时间戳： `str`（TIMESTAMP，格式为 HH:MM:SS）

示例

```
# Example 1: 读取标量数据
fps, fn, fx, fy = sensor.read_info(
```

```
sensor.info.FPS,  
sensor.info.FNORMAL,  
sensor.info.FSHEARX,  
sensor.info.FSHEARY,  
)  
  
# Example 2: 读取单个向量场（注意解包方式）  
flow = sensor.read_info(sensor.info.VRAW)
```

5. `Sensor.disconnect` 方法

描述

释放传感器相关资源并停止后台异步线程。调用后将关闭视频/摄像头输入源，并销毁所有 OpenCV 窗口。

建议在程序退出前调用以确保资源被正确释放。

`orisys.finger` 手指 UV pipeline

`examples/finger_pipeline.py` 展示了一个更适合手指/单相机场景的高层 API：

1. 使用 `FingerRuntime.open()` 打开视频源、加载 Blender 导出的资产，并构建 UV 展开映射。
2. 用 `grab_and_process()` 或 `read_source() + process_frame()` 获取一帧展开后的触觉图像。
3. 通过 `runtime.tactile` (`TactileProcessor`) 调用 `compute_deformation()`、`compute_contact()`、`read_info()`。
4. 退出前调用 `runtime.close()` 释放视频源和内部 `Sensor` 资源。

相比直接使用 `Sensor`，`FingerRuntime` 已封装了视频读取、相机尺寸设置、UV 展开和有效区域掩码传递，更适合作为手指 2D/3D 示例与上层应用的入口。

1. `FingerRuntime.open` 方法

描述

创建一个可复用的手指 UV runtime。该方法会完成以下初始化：

- 解析 `export_dir` 下的 `mesh.npz`、`camera_intrinsics.npz`、`world_to_cam.npy`

- 根据配置和标定信息构建/加载 `unroll_map_uv.npz`
- 打开相机或视频文件
- 创建内部 `Sensor("external", ...)`
- 创建 `FrameUnwarper` 与 `TactileProcessor`

输入参数

- **export_dir** (str, 可选): 手指导出资产目录, 默认值来自 `orisys.finger.DEFAULT_FINGER_EXPORT_DIR`。目录下通常需要包含 `mesh.npz`、`camera_intrinsics.npz`、`world_to_cam.npy`。
- **config** (str, 可选): Orisys 配置文件路径, 默认 `./config/dd02-ov.json`。
- **map_cache** (str, 可选): UV 展开缓存文件路径, 默认值来自 `orisys.finger.DEFAULT_FINGER_MAP_CACHE`。
- **source_scale** (float, 可选): 输入相机分辨率缩放比例; 未提供时根据配置自动推断。
- **calib** (str | None, 可选): 外部相机标定文件路径; 未提供时优先使用导出资产中的内参。
- **video** (int | str, 可选): 相机编号、数字字符串或视频路径, 默认 `0`。
- **camera_id** (int | None, 可选): `video` 的别名; 若提供则优先使用。
- **flip_source_vertical** (bool, 可选): 是否在 UV 展开前垂直翻转输入源, 默认 `True`。
- **verbose** (bool, 可选): 是否输出初始化日志。
- **mirror** (bool, 可选): 是否对内部触觉图再做镜像, 默认 `False`。

返回

- `FingerRuntime` 实例。

示例

```
from orisys.finger import FingerRuntime

runtime = FingerRuntime.open(
    config="./config/dd02-ov.json",
    video=0,
    verbose=True,
)
```

2. FingerRuntime 常用属性

- **runtime.geometry**: `FingerGeometry`, 包含输入/输出分辨率、导出资产路径、相机内参、缓存路径等一次性初始化结果。
- **runtime.domain**: UV 展开域对象, 内部包含 `remap`、`valid mask` 以及可选 `surface` 数据。
- **runtime.tactile**: `TactileProcessor`, 用于执行形变、接触和数据读取。

- **runtime.capture**: OpenCV VideoCapture 对象。
- **runtime.valid_mask**: 当前 UV 域有效像素掩码。

3. FingerRuntime.read_source 方法

描述

仅读取一帧原始视频源，等价于对内部 `cv2.VideoCapture` 调用 `read()`。适合需要把采集和 UV 展开分别计时的场景。

返回

- (ok, frame):
 - **ok** (bool): 是否成功读取。
 - **frame** (np.ndarray | None): 原始 BGR 帧。

4. FingerRuntime.process_frame 方法

描述

对一帧原始输入执行源图预处理、UV 展开，并把展开后的结果送入内部 `TactileProcessor`。这是 `examples/finger_pipeline.py` 中 `profile` 模式使用的方法。

输入参数

- **frame** (np.ndarray, 必填): 原始输入帧。
- **color** (bool, 可选): 为 `False` 时走灰度展开流程；为 `True` 时保留 BGR 颜色展开。默认 `False`。

返回

- np.ndarray: 展开后的触觉图像。color=False 时通常为单通道灰度图；color=True 时为 BGR 图。

5. FingerRuntime.grab_and_process 方法

描述

先读取一帧视频源，再调用 `process_frame()` 完成 UV 展开并推入内部触觉处理器。非 `profile` 模式下，`examples/finger_pipeline.py` 使用该方法作为主循环入口。

输入参数

- **color** (bool, 可选): 是否保留颜色，默认 `False`。

返回

- (ok, unrolled):
 - **ok** (bool): 是否成功读取并处理。
 - **unrolled** (np.ndarray | None): 展开后的触觉图像。

6. FingerRuntime.get_img 方法

描述

`Sensor.get_img()` 风格的别名：内部执行一次采集和 UV 展开，并在成功时返回当前展开图，否则返回 `None`。

输入参数

- **color** (bool, 可选): 是否保留颜色，默认 `False`。

返回

- `np.ndarray` | `None`

7. FingerRuntime.warmup 方法

描述

先执行一次采集与 UV 展开，然后调用 `runtime.tactile.warmup()` 预热内部光流/HHD 相关缓存。`examples/finger_pipeline.py` 在进入主循环前会先执行一次 `warmup`。

输入参数

- **color** (bool, 可选): 是否保留颜色，默认 `False`。

返回

- (ok, unrolled):
 - **ok** (bool): 是否成功完成 `warmup` 前的采集。
 - **unrolled** (`np.ndarray` | `None`): `warmup` 时得到的展开图。

8. FingerRuntime.close 方法

描述

释放内部 `VideoCapture` 并调用 `runtime.tactile.disconnect()` 清理 `Sensor` 资源。程序退出前应调用。

9. TactileProcessor 类别

描述

`TactileProcessor` 是对内部 `Sensor("external", ...)` 的窄封装。它保留了 `compute_deformation()`、`compute_contact()`、`read_info()`、`reset()` 等常用接口，同时屏蔽了视频采集与 UV 展开细节。

常用属性

- **raw_sensor**: 底层 `Sensor` 对象。
- **info**: 与 `Sensor.info` 相同的枚举入口。
- **img / flow / vnormal / vshear / depth_map**: 当前帧处理结果。

- **mask_stats**: 有效 UV 区域统计信息；examples/finger_pipeline.py 会读取 mask_stats["coverage"] 输出当前掩码覆盖率。
- **valid_mask**: 当前有效区域掩码。

说明

除 push_frame() / warmup() 之外，TactileProcessor 的 compute_deformation()、compute_contact()、read_info()、reset() 调用方式与 Sensor 基本一致，因此可直接复用前文 Sensor 章节中的参数说明。

10. 最小手指 pipeline 示例

下面的流程与 examples/finger_pipeline.py 一致：

```
import time
import cv2
import orisys
from orisys.finger import FingerRuntime

runtime = FingerRuntime.open(
    config="./config/dd02-ov.json",
    video=0,
    verbose=True,
)

runtime.warmup(color=False)
tactile = runtime.tactile

try:
    while True:
        ok, unrolled = runtime.grab_and_process(color=False)
        if not ok:
            break

        tactile.compute_deformation(check_motion=True, threshold=0)
        tactile.compute_contact()

        fps, flow, vnormal, img, contour, centroid, depth_map, fnormal, fshearx, fsheary = tactile.read_info(
            tactile.info.FPS,
            tactile.info.VRAW,
            tactile.info.VNORMAL,
            tactile.info.IMG,
            tactile.info.CONTOUR,
            tactile.info.CENTROID,
            tactile.info.DEPTH,
            tactile.info.FNORMAL,
            tactile.info.FSHEARX,
```

```

        tactile.info.FSHEARY,
    )

    arrows = orisys.util.draw_arrows(
        unrolled,
        flow,
        threshold=0.4,
        grid_spacing=10,
        arrow_scale=1.0,
    )
    cv2.imshow("UV Unrolled", arrows)

    print(
        f"FPS={fps:.2f} normal={fnormal:.2f} "
        f"shear=({fshearx:.2f}, {fsheary:.2f}) "
        f"mask={tactile.mask_stats['coverage']:.2f}",
        end="\r",
    )

    key = cv2.waitKey(1) & 0xFF
    if key == ord("q"):
        break
    if key == ord("r"):
        tactile.reset()
finally:
    runtime.close()
    cv2.destroyAllWindows()

```

11. 关于 profile 模式

examples/finger_pipeline.py 中的 --profile / --profile-every 是示例脚本参数，不属于 orisys SDK 核心 API。本质上它只是把主循环拆成：

1. runtime.read_source()
2. runtime.process_frame(...)
3. tactile.compute_deformation(...)
4. tactile.compute_contact()
5. tactile.read_info(...)

并对各阶段做耗时统计。因此在业务代码中，也可以按同样方式自行插入计时逻辑。

可视化相关

1. `util.draw_arrows` 方法

输入参数

- **frame**: 背景图像
- **optical_flow**: 光流场
- **threshold**: 光流箭头显示阈值
- **grid_spacing**: 光流箭头间距 (pixel)
- **arrow_scale**: 箭头大小
- **margin**: 箭头最大长度/间距的比例
- **colormap**: 色图, 如 `cv2.COLORMAP_JET`
- **below_threshold_color**: RGB 格式的 uint8 颜色, 如(0, 0, 255)
- **edge_margin**: 不显示箭头边距 (pixel)

返回

`np.ndarray`: 箭头光流可视化图像

2. `util.draw_contact` 方法

输入参数

- **image**: 背景图像
- **contour**: 轮廓
- **centroid**: 触碰点
- **point_color**: 触碰点的颜色 (BGR)
- **region_color**: 轮廓的颜色 (BGR)

返回

`np.ndarray`: 箭头光流可视化图像

配置标定相关

1. list_config 方法

描述

列出内置传感器配置参数字典。

返回

- **list:** 内置传感器配置参数字典的名字列表。 *** ## 2. export_config 方法

描述

导出当前传感器实例使用的配置参数字典。

输入参数

- **config_name:** 需要导出的配置参数字典名字, str
- **path:** 导出的路径, 默认"./config/exported_config.json"

返回

- **dict:** 导出的配置参数字典。

保存

- 导出内置的 config_name 配置参数字典至 path
-
-

常见问题解答 (FAQ)

Q1:为什么必须先调用 `get_img()`, 再调用 `compute_deformation()`?

A: `compute_deformation()` 依赖最近一次获取并拼接的图像数据进行形变与力计算。在调用该方法前, 必须先通过 `get_img()` 更新当前帧图像, 否则无法进行有效计算。

Q2:为什么调用了 `read_info()`, 但读到的数据是空的?

A: `read_info()` 仅用于读取当前已计算的数据状态, 不会触发任何计算。部分字段依赖特定计算步骤生成, 例如: *VRAW / VNORMAL / VSHEAR 依赖 `compute_deformation()`; *DEPTH / CENTROID / CONTOUR 依赖 `compute_contact()`。

Q3:compute_contact()为一定要调用吗？什么时候需要？

A：当需要接触相关信息（如接触质心、轮廓或深度/散度图）时，必须调用 `compute_contact()`。若仅关注形变场或力信息（如光流、法向力、切向力），则无需调用该方法。

Q4:read_info()的返回值顺序是如何确定的？

A： `read_info()`按传入的 `sensor.info` 参数顺序返回一个元组。返回值顺序与参数顺序一一对应，与内部存储顺序无关。

Q5:如果没有标定文件，或拼接结果不正确，会发生什么？

A：若未指定或未找到标定文件，系统会在需要时创建新的拼接标定数据并保存到默认路径。拼接结果异常通常与标定数据、拼接配置或输入图像质量有关，建议重新生成拼接标定并检查配置参数。

Q6: 运行报错：“AttributeError: ‘Stitcher’ object has no attribute ‘pos_default’”

A: 请检查视频源的编号是否正确，以及是否能正常打开视频源。在 linux 使用 `v4l2-ctl -l -list-devices` 或 Windows 使用 `Get-PnpDevice -PresentOnly | Where-Object { $_.Class -match '^Camera' }` 列出摄像头设备。