

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-0.3.0-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 json 文件路径，用于加载对应的参数配置，默认为 ./config/ddjx01。 * **cal_path** (str, 可选): 复眼拼接校准文件路径，默认为 ./config/cal.npy。有多个传感器同时运行时建议传感器类创建时建议填写，避免校准文件覆写。在首次运行时程序会在找不到文件时自动创建。 * **backend** (str, 可选): 光流场解耦计算后端，默认为 auto，可选 cupy,opencl,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

- `np.ndarray`：处理后的 BGR 格式图像(uint8，尺寸为 json 文件中 `resolution_stitch` 对应分辨率)。
- `None`：当输入源读取失败（例如视频结束、摄像头断开）时返回。

状态更新

- 成功时更新 `sensor` 类里面的图像。 ### 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。### 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。

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

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 --list-devices` 或 Windows 使用 `Get-PnpDevice -PresentOnly | Where-Object { $_.Class -match '^Camera' }` 列出摄像头设备。