

接口说明

1 寻找传感器

在使用视触觉传感器所有 API 接口时，需要先初始化 VTSDDeviceFinder 类找到对应的传感器。

1.1 初始化 VTSDDeviceFinder

VTSDDeviceFinder()->VTSDDeviceFinder

功能：初始化 VTSDDeviceFinder 类

参数：

返回值：VTSDDeviceFinder 实例

1.2 返回当前所有传感器的配置参数

get_devices()->List[VTSDDeviceBaseConfig]

功能：返回当前所有传感器的配置参数

参数：

返回值：List[VTSDDeviceBaseConfig]

1.3 返回当前传感器的数量

count()->int

功能：返回当前传感器的数量

参数：

返回值：int

1.4 返回当前所有传感器对应的内部索引号

indexes() ->List[int]

功能：返回当前所有传感器对应的内部索引号

参数：

返回值：List[int]

1.5 返回当前所有传感器对应的 vendorID

get_vendorIDs() ->List[str]

功能：返回当前所有传感器对应的 vendorID

参数：

返回值：List[str]

1.6 返回当前所有传感器对应的 SN

`get_sns()` ->List[str]

功能：返回当前所有传感器对应的 SN

参数：

返回值：List[str]

1.7 返回指定序列号的传感器对应的配置参数

`get_device_by_sn(sn: str)` ->VTSDDeviceBaseConfig

功能：返回指定序列号的传感器对应的配置参数

参数：

sn：请根据传感器铭牌序列号或初始化 VTSDDeviceFinder 日志进行获取

返回值：VTSDDeviceBaseConfig

VTSDDeviceBaseConfig 用于描述设备的基础配置信息

字段：

name: str 设备名称

vendorID: str 设备 VID

SN: str 设备 PID

index:int 设备内部索引号

1.8 返回指定型号的传感器对应的配置参数

`get_devices_by_vendorID(vendor_id: str)` ->List[VTSDDeviceBaseConfig]

功能：返回指定型号的传感器对应的配置参数

参数：

vendor_id：传感器型号

返回值：List[VTSDDeviceBaseConfig]

1.1-1.8 示例如下：

```
from pyvitaidsdk import VTSDDeviceFinder

if __name__ == "__main__":

    finder = VTSDDeviceFinder()
    # 获取所有的序列号
    print('finder.get_sns()', finder.get_sns())

    # 获取所有的 Vendor ID
    print('finder.get_vendorIDs()', finder.get_vendorIDs())

    # 打印目前链接的所有传感器信息
    print('finder.get_devices()', finder.get_devices())

    # 打印目前链接的传感器数量
```

```

print('finder.count()', finder.count())

# 打印目前链接的传感器数量
print('finder.indexes()', finder.indexes())

# 打印指定型号的传感器信息
print('finder.get_devices_by_vendorID', finder.get_devices_by_vendorID("f225"))

# 打印指定序列号的传感器信息
print('finder.get_device_by_sn', finder.get_device_by_sn("0001"))

```

2 GF225 使用

2.1 GF225 对象初始化及释放

在使用 GF225 的所有 API 接口时，都需要先调用 GF225()方法初始化 GF225 对象，不再使用该对象时通过调用该对象的 release()方法进行释放。

```

GF225(config:VTSDeviceBaseConfig, model_path: str, device
:str) -> GF225

```

功能：初始化 GF225 参数：

config:从 VTSDeviceFinder 获取到的配置参数

model_path:深度恢复模型路径

device: 'cpu' or 'cuda'

返回值：GF225 实例

2.2 设置自动检测并进行透视变化边距参数

```

set_auto_warp_paddings(top:int,bottom:int,left:int,right:int,dsize:List[int]=DEF
AULT_WARP_DSIZE) -> None:

```

功能：设置自动检测并进行透视变化边距参数。

参数：

top (int): 顶部边距

bottom (int): 底部边距

left (int): 左侧边距

right (int): 右侧边距

dsize (List[int], optional):透视变换后输出的有效图像大小，默认[480, 480]

返回值：

2.3 设置手动设置透视变化参数

```

set_manual_warp_params(src: List[List[float]], scale: float, dsize:
List[int]=DEFAULT_WARP_DSIZE) -> None:

```

功能：设置手动设置透视变化参数

参数：

src (List[List[float]]): 原图像中希望进行透视变换的四个点.顺序依次为左上、右上、

右下、左下。如: [[240, 99], [434, 101], [417, 275], [249, 271]]。建议采用左上、右上、右下、左下四个 marker 点的中心点位置

scale (float): 缩放比例

dsiz (List[int], optional): 透视变换后输出的有效图像大小, 默认[480, 480]

返回值 :

2.4 获取一帧图像

read() -> Tuple[bool, np.ndarray, np.ndarray]

功能 : 获取一帧图像

参数 :

返回值 : Tuple[bool, np.ndarray, np.ndarray] # 是否读取成功, 原始图像, 变化后图像

2.5 对相机 flush

flush(nums: int) -> None

功能 : 对相机 flush 指定的帧数

参数 :

nums (int): 希望 flush 的帧数

返回值 :

2.1-2.5 示例如下 :

```
...
vtsd = VTSDDeviceFinder()

# 修改指定传感器 SN
config = vtsd.get_device_by_sn(vtsd.get_sns())[0]
vt = GF225(config=config)
vt.set_auto_warp_paddings(30, 40, 35, 30)
vt.flush(30)

while 1:
    ret, raw_frame, warped_frame = vt.read()
    if ret:
        cv2.imshow(f"raw_frame", raw_frame)
        cv2.imshow(f"warped_frame", warped_frame)

        key = cv2.waitKey(1) & 255
        if key == 27 or key == ord("q"):
            break

vt.release()

...
```

2.6 启动后端

`start_backend()` -> None

功能：启动后端,会启动一个后台线程持续获取图像.

参数：

返回值：

2.7 获取一帧原始图像

`get_raw_frame()` -> np.ndarray:

功能：获取一帧原始图像（与 `start_backend()` 配合使用，该接口返回最新的一帧原始图像）

参数：

返回值：np.ndarray

2.8 获取一帧变换后图像

`get_warped_frame()` -> np.ndarray:

功能：获取一帧变换后图像（与 `start_backend()` 配合使用，该接口返回最新的一帧变换后图像）

参数：

返回值：np.ndarray

2.9 停止后端

`stop_backend()` -> None

功能：停止后端

参数：

返回值：

2.10 释放对象

`release()` -> None

功能：释放对象

返回值：

2.6-2.10 示例如下：

```
...
vtsd = VTSDDeviceFinder()

# 修改指定传感器 SN
config = vtsd.get_device_by_sn(vtsd.get_sns()[0])
vt = GF225(config=config)

# 修改参数
vt.set_manual_warp_params([[258, 135], [389, 135], [383, 256], [264, 256]], 1.5,
dsz=[240, 240])
```

```

vt.start_backend()

while 1:
    cv2.imshow(f"get_raw_frame", vt.get_raw_frame())
    cv2.imshow("get_warped_frame", vt.get_warped_frame())
    key = cv2.waitKey(1) & 255
    if key == 27 or key == ord("q"):
        break

vt.release()
vt.stop_backend()

...

```

2.11 判断是否已经初始化 Marker

is_inited_marker() -> bool:

功能：是否已经初始化 Marker

参数：

返回值：bool

2.12 初始化 Marker

init_marker(image:np.ndarray) -> None:

功能：初始化 Marker

参数：

image：用来进行初始化的一帧的有效图像

返回值：

2.13 对当前图像进行 Marker 的追踪

tracking(image:np.ndarray) -> None:

功能：对当前图像进行 Marker 的追踪

参数：

image：用来进行跟踪的一帧的变化后图像

返回值：

2.14 返回初始帧按照行列排布的 mark 点坐标

get_origin_markers() -> np.ndarray:

功能：返回初始帧按照行列排布的 mark 点坐标

参数：

返回值：

2.15 返回当前帧按照行列排布的 mark 点坐标

get_markers() -> np.ndarray:

功能：返回当前帧按照行列排布的 mark 点坐标

参数：

返回值：

2.16 在图像上绘制 Marker 移动流

`draw_flow(frame:np.ndarray, flow) -> None:`

功能：在图像上绘制 Marker 移动流

参数：

frame:np.ndarray

flow:移动流

返回值：

2.11-2.16 示例如下：

```
...
ret, raw_frame, warped_frame = vt.read()

if not vt.is_inited_marker():
    vt.init_marker(warped_frame)
else:
    flow = vt.tracking(warped_frame)
    vt.draw_flow(warped_frame, flow)
    print(f"vts.get_origin_markers(): {vt.get_origin_markers().shape}")
    print(f"vts.get_markers(): {vt.get_markers().shape}")

cv2.imshow(f"tracking image", warped_frame)
...
```

2.17 获取当前标志位的三维向量

`get_3d_vector(frame:np.ndarray) -> np.ndarray:`

功能：获取当前标志位的三维向量

参数：

frame: 图像

返回值：shape 为 81*3 的 np 数组,依次为从左上角开始,先行后列的标志点坐标 (x , y , z)

2.17 示例如下：

```
...
vector = vts.get_3d_vector(frame)
...
```

2.18 进行标定

`calibrate(nums: int) -> None`

功能：标定，设置背景信息

参数：

nums (int): 希望标定的帧数

返回值：

2.19 进行再标定

re_calibrate(nums: int) -> None

功能：再标定，用于更新背景信息

参数：

nums (int): 希望标定的帧数

返回值：

2.20 判断是否已经标定

is_calibrate() -> bool

功能：判断是否已经标定

参数：

返回值：

2.21 启用滑动检测

enable_slip_detect() -> None

功能：启用滑动检测

参数：

返回值：

2.22 停用滑动检测

disable_slip_detect() -> None

功能：停用滑动检测

参数：

返回值：

2.23 查看滑动状态

slip_state() -> SlipState

功能：滑动状态

参数：

返回值：

UNKNOWN	= 0	# 未知
CONTACT	= 1	# 接触
INCIPIENT_SLIP	= 2	# 初始滑移
PARTIAL_SLIP	= 3	# 部分滑移
COMPLETE_SLIP	= 4	# 完全滑移
NO_OBJ	= 5	# 没有物体
STEADY_HOLD	= 6	# 静止保持

2.18-2.23 示例如下：


```

...
vt.start_backend()

calib_num = 50
slip_state = vt.slip_state()
vt.calibrate(calib_num) # 启动标定
while 1:
    frame = vt.get_warped_frame()
    if vt.is_calibrate():
        slip_state = vt.slip_state()
        frame_copy = frame.copy()
        cv2.imshow(f"frame", frame_copy)
        key = cv2.waitKey(1) & 0xFF
        if key == ord("q"):
            break
        elif key == ord("e"):
            # 按 e 开启滑动检测
            vt.enable_slip_detect()
        elif key == ord("d"):
            # 按 d 关闭滑动检测
            vt.disable_slip_detect()
        elif key == ord('r'):
            vt.re_calibrate(calib_num) # 重新标定

vt.stop_backend()
...

```

2.24 设置背景深度图

set_background_depth(image:np.ndarray) -> None

功能：设置背景深度图

参数：image 从传感器中读取到的变换后的图像

返回值：

2.25 清除背景深度图

clear_background_depth() -> None

功能：清除背景深度图

参数：

返回值：

2.26 判断背景深度图是否已设置

is_background_depth_init() -> bool

功能：判断背景深度图是否已设置

参数：

返回值：bool

2.27 进行 3d 重构

recon3d(image:np.ndarray) -> None

功能：基于传入的图像进行 3d 重构

参数：
返回值：None

2.28 获取深度图

get_depth_map() -> np.ndarray
功能：获取深度图,与 recon3d(image)配合使用
参数：
返回值：np.ndarray

2.29 获取背景深度图

get_background_depth_map() -> np.ndarray
功能：获取背景深度图,与 set_background_depth(image)配合使用
参数：
返回值：np.ndarray

2.30 获取深度图差

get_diff_depth_map() -> np.ndarray
功能：获取深度图差（当前深度图-背景深度图）,与 recon3d(image)配合使用
参数：
返回值：np.ndarray

2.24-2.30 示例如下：

```
...
vt.start_backend()
frame = vt.get_warped_frame()
vt.set_background_depth(frame)
while 1:

    frame = vt.get_warped_frame()
    cv2.imshow(f"get_warped_frame", frame)
    cv2.imshow(f"get_raw_frame", vt.get_raw_frame())
    if vt.is_background_depth_init():
        vt.recon3d(frame)
        background_depth_map = vt.get_background_depth_map()
        depth_map = vt.get_depth_map()
        diff_depth_map = vt.get_diff_depth_map()
        cv2.imshow(f"depth_map", depth_map)
        cv2.imshow(f"background_depth_map", background_depth_map)
        cv2.imshow(f"diff_depth_map", diff_depth_map)

    key = cv2.waitKey(1) & 255
    if key == 27 or key == ord("q"):
        break
    elif key == ord("e"):
        # 按 e 重新设置背景深度图
        vt.clear_background_depth()
        vt.set_background_depth(frame)
```

```
vt.stop_backend()  
vt.release()  
...
```